

Read_me, run_me, execute_me: Some notes about software art

Inke Arns, Berlin <<http://www.v2.nl/~arns>>

Lecture at kuda, Novi Sad, April 9, 2004

»Software is mind control. Come and get some.«¹

Today artists do not only use prefabricated software to produce something, but engage with programming and software itself in playful, inventive and critical ways. Software art in the broadest sense describes activities which use software or code as an artistic material. I'd first like to direct your attention to the question of why software or code could be at all an interesting material for artistic production. Secondly, I will try to define the term "software art". And thirdly, I will talk about the notion of the performative.

1. Software as artistic material

Software has become ubiquitous which means that even unknowingly you cannot avoid it anymore today. Software nowadays can be found not only in computers but in almost everything ranging from communication devices, telephones, all sorts of media machines, even washing machines and other household devices. Ubiquitousness in this case does not only mean that software is everywhere, that it is all pervasive, but it also means that most of the time it remains invisible. This general invisibility of software applies on two different levels. First, in most cases the so called "raw" source code is covered by glossy surfaces where you are not manipulating the code directly but where you are working with a graphic user interface. The second level of the invisibility of software applies to the interface itself. It is what computer scientists call the »transparency of the interface«. In everyday language transparency normally means visibility or clearness, or controllability through visibility. However, transparency in the case of computer science rather stands for information hiding, which means that the user does not notice the software working in the background.

Even if information hiding in the case of interface design can be useful, it can be said that at the same time it suggests to the user or to the viewer a direct or even natural view or access to the data. This is of course not the case as Lev Manovich notes in his book *The Language of New Media*. A short quote: »Far from being a transparent window into the data inside the

¹ Slogan for the »Web Stalker« (1997) by the London artists group I/O/D, <http://www.backspace.org/iod> .

computer the interface brings with it strong messages of its own.«² Today, in a time when our environment gets increasingly mediatized and digitized and thus can be said to be based increasingly on software, it becomes more and more important to be aware that code or software directly affects the virtual and actual spaces in which we are moving, communicating and living. It has the capabilities to directly mobilize or immobilize its users.

This is why Lawrence Lessig in his book *Code and other laws of cyberspace*³ claims that program code increasingly tends towards becoming law. Today control functions are being build directly into that very architecture of, for example, the net, which means into its code. Taking as an example the online service America Online (AOL) Lessig poignantly makes clear how code directly enables or disables freedom of movement, of speech and of behavior. Code should - even if it remains largely invisible - not be accepted as something natural or as god given fact. It is rather written by humans and can therefore be changed or conceived differently.

Software art deals with these coded structures underlying and generating visible surfaces. It focuses our attention on the all pervasive program code which our increasingly digitized working and living environment is based upon and uses this code or this software as its artistic material.

2. Software art

»Software art« describes an artistic activity which in the medium (or better: the material) of software allows for a critical reflection of software (and its cultural impact). Software art does not regard software merely as a pragmatic, invisible tool generating certain visible results or surfaces, but on the contrary focuses on the program code itself – even if this code is not explicitly being laid open or put in the foreground. According to Florian Cramer, software art makes visible the aesthetic and political subtexts of seemingly neutral technical commands. Doing so, software art can happen on different levels: it can be located on the level of the source code, on the level of abstract algorithms or on the level of the result generated by a certain program code.⁴ Thus it comes as no surprise that there is a broad spectrum of software artworks ranging from so-called »Codeworks« consisting predominantly

² Lev Manovich, *The Language of New Media*, Cambridge, MA., 2001, p. 65.

³ Lawrence Lessig, *Code and other Laws of Cyberspace*, New York 1999, <http://www.code-is-law.org/>.

⁴ »Is it at the level of source code? If so it's a form of typographic layout or illumination. Is it at the level of abstract algorithms? If so it's a form of conceptual art or architecture. Is it at the level of the output of the program? If so it's a form of preparatory sketch.« Rob Myers, »Re: 'Code as Art' Digest [from the PD-List]«, in: *[eu-gene]*, Jan 4, 2004.

of ASCII-Code (not being *executables*), to experimental web browsers (e.g. »WebStalker«, 1997), and fully-executable programmes (e.g. Antoine Schmitt's »Vexation 1«⁵, 2000, or Adrian Ward's »Auto-Illustrator«⁶, 2000). Software art *can* contain elements of generative art but does not necessarily have to be generative in a strict technical sense (see »Codeworks«). Software art and generative art can therefore not be used synonymously. Rather, these two notions function in *different registers*, as I hope to show in the following examples.

My first example is the project »insert_coin«⁷ by Dragan Espenschied and Alvar Freude. In the framework of their diploma work which they realised under the motto „two people control 250 people“ in 2000/2001, the two media art students secretly installed a Web proxy server at the Merz Academy in Stuttgart, Germany, which via a perl script manipulated the entire Web traffic of students and professors in the Academy's computer network. According to Espenschied and Freude, the aim of this project was to critically assess the »competence and the critical faculty of the users concerning the everyday medium Internet«⁸. The manipulated proxy server redirected the entered URLs onto other addresses, modified HTML code, transformed the content of the latest news on news websites via a simple search-and-replace function (e.g. by replacing the name of politicians), as well as – and this was most unsettling – the content of private e-mails that were read through Web interfaces like Hotmail or Yahoo!. During four weeks this project was manipulating the Web access of the entire Academy – and now that's the best: it remained unnoticed. When Espenschied and Freude announced the project publicly, almost nobody was interested. They even published a simple-to-follow instruction manual which would enable everybody to independently switch-off the filter that was manipulating the Web content. But only a minority of those concerned took the time to make the minor adjustment in order to regain access to unfiltered information. Still several months after the end of the experiment the Web access from most of the Academy's computers was filtered.

My second example, »walser.php«⁹ by textz.com/Project Gnutenberg (i.e. Sebastian Lütgert), has been called »political« or »literary«¹⁰ software. We might call it an anticopyright-activist software which has been written in response to one of the biggest literary scandals in Germany after the Second World War. The file name »walser.php« is not only an ironic allusion to the file »walser.pdf«, a digital pre-printed version of Martin

⁵ <http://www.gratin.org/as/> .

⁶ <http://www.signwave.co.uk> .

⁷ http://www.odem.org/insert_coin/ ; English: <http://www.censorship.odem.org/content.html> .

⁸ Cf. Espenschied/Freude's text for the Internationaler Medienkunstpreis 2001, http://www.online-demonstration.org/insert_coin/imkp2001.html .

⁹ <http://textz.com/trash> .

¹⁰ Florian Cramer, »walser.php«, in: Olga Goriunova, Alexei Shulgin (Hg.), *Read_Me 2.3. Reader*, Helsinki 2003, pp. 76–78, here: S. 76.

Walser's controversial novel which was distributed by the Suhrkamp publishing house as an e-mail attachment – and later on, due to the unfavorable circumstances, called back by the publisher (nice try: calling back a digital document). Rather, »walser.php« (or rather »walser.pl«) by textz.com was/is a Perl script which via an appropriate Perl interpreter can generate a human-readable ASCII text version of Walser's novel *Death of a Critic* from the 10.000 lines of source code¹¹. While the source code written in Perl contains the novel itself in an „invisible“, machine-readable form and thus can be distributed and modified as free software under the GNU General Public License, it may be *executed* only with the written permission of the Suhrkamp publishing house.¹²

While Espenschied & Freude's experiment on filtering and censorship of Internet content points to the relatively unlimited potential of control that is contained in software, »walser.php« offers a practical solution for dealing with the commercial restrictions which threaten the freedom of information on the Internet in the form of Digital Rights Management Systems. While »insert_coin« temporarily realises a dystopian scenario in form of manipulated software, textz.com with its »walser.php« project develops genuinely utopian »counter measures in the form of software«¹³.

Both of these projects are *generative* in the best sense of the word. However, neither »insert_coin« nor »walser.php« comply with the definitions of »generative art« currently found predominantly in the area of design. For Philip Galanter, for example, the term »generative art« refers to »any art practice where the artist uses a system, such as a set of natural language rules, a computer program, a machine, or other procedural invention, which is set into motion with some degree of autonomy contributing to or resulting in a completed work of art.«¹⁴ Similarly, Celestino Soddu, director of the Generative Design Lab at the Polytechnical University of Milano and organiser of the »Generative Art«¹⁵ conferences, defines »generative art« as a processual tool enabling the artist or designer to »synthesize [...] an ever changing and unpredictable series of events, pictures, industrial objects, architectures, musical works, environments, and communications [...]«¹⁶

¹¹ <http://textz.com/trash/walser.pl.txt> .

¹² Cf. textz.com, »Suhrkamp calls back walser.pdf, textz.com releases walser.php«, o. J., <http://textz.com/trash/readme.txt> ; Michael Thomas, »Tod einer Kritik. Walsers umstrittenes Buch als Perl-Script im Internet«, in: *Telepolis*, 27.6.2002, <http://www.heise.de/tp/deutsch/inhalt/on/12807/1.html> . The project »pngreader« (<http://runme.org/project/+pngreader/>) functions similarly, the only difference is that here texts are encoded into PNG image files and thus can be distributed freely.

¹³ Cramer, »walser.php«, p. 77.

¹⁴ Philip Galanter, »What is Generative Art? Complexity Theory as a Context for Art Theory«, in: *Generative Art Proceedings*, Milano 2003, p. 4, <http://www.philipgalanter.com/pages/acad/media/ga2003%20proceedings%20paper.pdf> .

¹⁵ Cf. <http://www.generativeart.com/> .

¹⁶ Celestino Soddu, »Generative Art and Architecture«, abstract, no year given, <http://www.nyu.edu/studio/generative.html> .

What becomes apparent in these quotations is the fact that »generative art« is interested predominantly in the *results* created by generative processes. Software in this context is seen and employed as a pragmatic-generative tool or device for the creation of certain results – without being questioned itself. The generative processes brought about by software here primarily do serve to avoid intentionality and to produce an unexpected, arbitrary and inexhaustible diversity of forms. The »n_Gen Design Machine« by Move Design, submitted to the *Read_Me* Festival 2003 in Helsinki, as well as Cornelia Sollfranks »net.art Generator«¹⁷ (1999) which at the push of a button generates net art, should be seen as ironic commentaries on »generative design« (mis-)understood in such a way.¹⁸

»insert_coin« and »walser.php« go beyond such definitions of »generative art« or »design« in so far as these projects are interested far more in the *coded processes* generating certain results or surfaces. This interest in the coded processes, or, to be more precise, in the significance and implications of software and coded structures, sharply distinguishes them not only from generative art but also from many interactive installations of the 1990s which displayed their disinterest in software by hiding the program code in *black boxes*. Instead, projects like »insert_coin« and »walser.php« aim at questioning software and code as culture – and at questioning culture as implemented in software. For this, they develop »experimental software« (in »insert_coin« a proxy server and in »walser.php« a perl script), which does not only generate arbitrary surfaces but which critically investigates the technological, cultural or social impact of software. What's more, the writing of »experimental software« is very well concerned with *artistic subjectivity*, as can be seen in the usage of different private languages, and less with proving evidence of a machinic creativity (whatever this may be): »Code can be diaries, poetic, obscure, ironic or disruptive, defunct or impossible, it can simulate and disguise, it has rhetoric and style, it can be an attitude«¹⁹, thus the emphatic definition by Florian Cramer and Ulrike Gabriel, both members of the *transmediale* software art jury in 2001.

I have tried to set up a somewhat polemical comparison between generative art and software art which I would like to show you:

¹⁷ Cornelia Sollfrank, »net.art generator« (1999), <http://www.obn.org/generator>.

¹⁸ Cf. Olga Goriunova / Alexei Shulgin, »n_Gen Design Machine«, in: Olga Goriunova / Alexei Shulgin (eds.), *Read_Me 2.3. Reader*, Helsinki 2003, pp. 66–67, here: p. 66.

¹⁹ Florian Cramer / Ulrike Gabriel, »Software Art«, in: Broeckmann, Andreas / Jaschko, Susanne (eds.), *DIY Media - Kunst und digitale Medien: Software - Partizipation - Distribution. Transmediale.01*, Berlin 2001, pp. 29–33, here p. 33.

Generative art	Software art
Focus on the <u>surface</u> ("phenotext") created by a generative process ("black box problem")	Focus on <u>generative process</u> (set in motion by a "genotext") which might generate surfaces or other results
Software as pragmatic/neutral tool serving to create a certain result; the tool itself is not being questioned	Software as <u>culture</u> which is being questioned; interest in aesthetical and political subtexts; software can be "experimental" and "non-pragmatic"
Software as pragmatic-generative <u>tool</u>	Software or code as a <u>work of its own</u> (possibly experimental)
<u>Efficient</u> code ("beautiful algorithms"*)	Code as excess, code as extravagance, <u>not necessarily efficient</u>
Employment of generative processes in order to <u>negate intentionality</u>	»Software artists [...] seem to conceive of generative systems <u>not as negation of intentionality</u> , but as balancing of randomness and control. [...] Far from being simply art for machines, software art is <u>highly concerned with artistic subjectivity</u> and its reflection and extension into generative systems.«** (Cramer/Gabriel)
Fascination of the generative	Interest in the "performativity" of code

* Cf. Donald Knuth, *The Art Of Computer Programming: Vol. 1, Fundamental Algorithms*, Reading, Mass. 1997.

** Florian Cramer / Ulrike Gabriel, quoted after Andreas Broeckmann, »On Software as Art«, in: *Sarai Reader 2003: Shaping Technologies*, New Delhi 2003, pp. 215-218, here: p. 216.

The notion of »software art« [or: artistic software] was first coined and introduced as a competition category in 2001 by the Berlin media art festival *transmediale*^{20, 21}. Software art, which other authors call »experimental«²² or »speculative software«²³ as well as »non-pragmatic« and »non-rational«²⁴ software, encompasses projects, if we are to follow the

²⁰ Other notable events: »Kontrollfelder« (Dortmund 2001, curated by Andreas Broeckmann and Matthias Weiß, <http://www.hardware-projekte.de/programm/inhalt/kontroll.htm>); the »Read_Me« Festival, conceived by Olga Goruinova and Alexei Shulgin (Moskau 2002, Helsinki 2003, http://www.m-cult.org/read_me/) and the exhibitions »Generator« (GB 2002, curated by Geoff Cox, <http://www.generative.net/generator.html>), »CODeDOC« (New York, Sept. 2002, curated by Christiane Paul, <http://artport.whitney.org/commissions/codedoc/>), »I love you - computer_viren_hacker_kultur« (Frankfurt/Main, Jan. 31-Feb. 5, 2003, http://www.digitalcraft.org/index.php?artikel_id=269) and the software art repository »Runme«, launched in January 2003 (<http://runme.org>). Further examples of software art can be found on these Web sites. The most historically significant year in terms of software art is 1970, during which three software art-related events took place: Jack Burnham's exhibition »Software – Information Technology: Its New Meaning for Art«, which took place at the Jewish Museum in New York; the exhibition curated by Kynaston McShine at MoMA in New York, entitled »Information«; and the foundation of the magazine »Radical Software« by Beryl Korot, Phyllis Gerhuny, and Ira Schneider (<http://www.radicalsoftware.org/>).

²¹ For an early, programmatic concept paper on software programming and art, see Geoff Cox / Alex McLean / Adrian Ward, »The Aesthetics of Generative Code« (2000, <http://generative.net/papers/aesthetics/>). An attempt to formally define and research the archaeological history of software art using literary and artistic examples can be found in Florian Cramer, »Concepts. Notations. Software. Art«, Mar. 23, 2002, http://userpage.fu-berlin.de/~cantsin/homepage/writings/software_art/concept_notations/concepts_notations_software_art.html.

²² Tilman Baumgärtel, »Experimentelle Software. Zu einigen neueren Computerprogrammen von Künstlern«, in: *Telepolis*, Oct 28, 2001, <http://www.heise.de/tp/deutsch/inhalt/sa/9908/1.html>.

²³ Matthew Fuller differentiates between »critical«, »social« and »speculative software«. Cf. Matthew Fuller, »Behind the Blip: Software as Culture«, in: *Nettime*, Jan 7, 2002, <http://amsterdam.nettime.org/Lists-Archives/nettime-l-0201/msg00025.html>.

²⁴ Olga Goruinova and Alexei Shulgin define »artistic software« as »non-pragmatic« and »non-rational«: »[I]f conventional programs are instruments serving purely pragmatic purposes, the result of the work of artistic programs often finds itself outside of the pragmatic and the rational.« (Olga Goruinova / Alexei Shulgin, »Artistic Software for Dummies and, by the way, Thoughts About the New World Order«, in: *Nettime*, May 26, 2002, <http://amsterdam.nettime.org/Lists-Archives/nettime-l-0205/msg00169.html>).

definition given by *transmediale* festival, whose essential artistic material is program code, or who critically deal with the cultural understanding of software. Program code is not being understood as a pragmatic tool serving to make the actual work run, but as a generative material of machinic and social processes. In this way, software art can as well be the result of an autonomous and formal creative practice as well as it can also critically refer to existing software and the technological, cultural or social impact of software.²⁵

Interestingly, the difference between software art and generative art reminds one of the difference between contemporary forms of software art and early computer art of the 1960s (and here I am referring to Tilman Baumgärtel's "Experimental Software" from 2001). The difference can be described as follows: Works from the field of software art, or experimental software »are not art that has been created *with the help of the computer*, but art that *happens in the computer*; software is not programmed by artists in order to produce *autonomous artworks*, but the *software itself is the artwork*. What is crucial here is not the result but the process triggered in the computer by the program code.«²⁶ Though the computer art of the 1960s is close to concept art in its privileging of the concept as opposed to its realisation – nevertheless, computer art is not consistently thinking this idea to an end: With its works executed on paper by plotters and dot matrix printers it emphasised the final product – but not the program or the process which generated the work.²⁷ In current software art projects this relation is reversed: Here it is »exclusively about the process generated by these programs. While the computer art of the 60s and 70s regarded the processes in the computer only as a method for generating an external result, but not as a work in its own terms, and treated the computer as a sort of black box, thus concealing the operations and procedures going on inside it, today's software projects precisely want to focus on these processes, make them visible and to bring them up for discussion.«²⁸

²⁵ Cf. http://www.transmediale.de/04/pdf/tm04clubtm04_formular_ausschreibung.pdf. On software art cf. the panel discussion during transmediale.03 (Künstlerhaus Bethanien, Berlin, Feb 4, 2003), <http://www.softwareart.net/>, as well as Olga Goriunova / Alexei Shulgin (eds.), *Read_Me 2.3 Reader - about software art*, Helsinki 2003, http://www.m-cult.org/read_me.

²⁶ Baumgärtel, »Experimentelle Software« [my accentuation].

²⁷ Typical in this context are the artworks by the so-called „Algorists“, who were co-founded by Roman Verostko. Cf. Roman Verostko, »Epigenetic Painting: Software As Genotype, A New Dimension of Art « (1988), <http://www.verostko.com/epigenet.html>; Roman Verostko, »Epigenetic Art Revisited: Software as Genotype«, in: Christine Schöpf / Gerfried Stocker (eds.), *Ars Electronica 2003: Code - The Language of Our Time*, Ostfildern 2003, pp. 156–167. Here one finds formulations like: »The essential character of each finished work is derived from the ›form-generating-procedure‹ or ›algorithm‹ acting as genotype. For this reason one could say that the finished work is an epiphany, or manifestation, of its generator, the *code*. For me each work celebrates its code [...].«

²⁸ Baumgärtel, »Experimentelle Software«.

3. Performativity of the Code vs. Fascination of the Generative

The current interest in software is, as I see it, not only grounded in the fascination with the generative aspect of software, i.e. on the ability to create and to procreate in a purely technical sense. What interests the authors of these projects is much more something I'd like to call the *performativity* of code. By *performativity of the code* I mean its ability to act and perform in the sense of speech act theory.

I am thinking here of a series of lectures held by John Langshaw Austin at the Harvard University in 1955. In these lectures, entitled *How to Do Things With Words*, Austin formulated the groundbreaking idea that language does not only have a descriptive, referential or constative function, but also possesses a performative dimension. Austin distinguishes three different speech acts: the locutionary²⁹, the illocutionary³⁰, and the perlocutionary³¹ act. Only illocutionary speech acts are performatives – i.e., they create or do what they describe, provided that they are set within a matrix that is simultaneously social and semiotic. This draws attention to the importance of the *context* of a performative utterance. The illocutionary, or performative utterance can succeed or fail, depending on whether it is set in an appropriate context or not.

Accordingly, if I speak of the performativity of code, I claim that this performativity is not to be understood as a purely technical performativity, i.e. it does not only happen in the context of a closed technical system, but affects the realm of the aesthetical, the political and the social. Program code is characterised by the fact that here „saying“ coincides with „doing“. Code as an effective speech act is not a description or a representation of something, but, on the contrary, it directly affects, and literally sets in motion – or it even »kills« a process.³² This »coded performativity«³³ has immediate, also political consequences on the actual and virtual spaces (amongst others, the Internet), in which we are increasingly moving and living: it means, ultimately, that this coded performativity *mobilises or immobilizes* its users. Code thus becomes Law, or, as Lawrence Lessig has put it in 1999, »Code [already] is Law«³⁴. This is the reason why software art is rather more interested in the »performance« than in the »competence« (terms coined by Noam Chomsky), rather more interested in the

²⁹ Locutionary: The speech act as meaningful utterance.

³⁰ Perlocutionary: A meaningful utterance with a certain conventional – performative – force.

³¹ Illocutionary: A meaningful utterance with a certain conventional force non-conventionally bringing about a certain effect.

³² Cf. Inke Arns, »Texte, die (sich) bewegen: zur Performativität von Programmiercodes in Netzkunst und Software Art«, in: Inke Arns / Mirjam Goller / Susanne Strätling / Georg Witte (eds.), *Kinetographien*, Bielefeld 2004 [forthcoming].

³³ Reinhold Grether, »The Performing Arts in a New Era«, in: *Rohrpost*, July 26, 2001.

³⁴ Lawrence Lessig, *Code and other Laws of Cyberspace*, New York 1999.

parole than the *langue*³⁵ (famous opposition coined by Ferdinand de Saussure). In our context, *performance* and *parole* mean the respective actualisations and concrete realisations and repercussions a certain program code has on, let's say, social systems, and not only what it does or generates in the context of abstract-technical systems. In the projects »insert_coin« and »walser.php« the generative is deeply political – and this is so because the secret transformation of existing texts (in the case of insert coin) and the extraction of a text protected by copyright law from a Perl script (in the case of walser.php) is questionable and critical *not* in the context of a *technical* system, *but* in the context of *social and political systems* that are increasingly relying on these technical structures.

Certainly one of the »most radical understanding[s] of computer code as artistic material«³⁶ can be found in the so-called »Codeworks«³⁷ and the artistic use they make of program code. »Codeworks« almost exclusively consist of texts which are sent to mailing lists like Nettime or 7-11 in the form of simple e-mails. »Codeworks« make use of formal ASCII instruction code and its aesthetic – without relying on surfaces and graphical user interfaces normally created by this code. Works by Jodi, Netochka Nezvanova aka antiorp and mez³⁸ thus recall the existence of a hidden, »invisible shadow world of process«³⁹, as Graham Harwood has called it. Technically speaking, these »Codeworks« are located on the opposite side of an imaginary spectrum of generativity. However, the status of these languages or these language-like bits and pieces remains ambivalent: In the perception of the recipient they oscillate between supposed executability, thus functionality, and non-executability – i.e. dysfunctionality – of the code; in short: between significant information and meaningless noise. This phenomenon can be seen very clearly in Jodi's »walkmonster_start ()«-e-mail which was sent to the Nettime mailing list on October 22, 2001. While the text contained in this e-mail resembles executable program code, for the non-specialist reader it remains completely open whether in another location in the computer this text could in fact be compiled, and thus be turned into machine-readable algorithms, and thus, ultimately, be executable.

³⁵ The distinction between competence and performance is credited to Noam Chomsky's generative transformation grammar (See Noam Chomsky, *Aspects of the Theory of Syntax*, Cambridge, MA., 1965); the distinction between *langue* and *parole* is attributed to Ferdinand de Saussure (See Ferdinand de Saussure, *Cours de linguistique générale*, Paris 1967 [1916]).

³⁶ Florian Cramer, »Exe.cut[up]able statements: Das Drängen des Codes an die Nutzeroberflächen«, in: Christine Schöpf / Gerfried Stocker (eds.), *Ars Electronica 2003: Code - The Language of Our Time*, Ostfildern 2003.

³⁷ Cf. on this Alan Sondheim, »Codework«, in: *American Book Review*, Vol. 22, Issue 6 (September/October 2001), <http://www.litline.org/ABR/PDF/Volume22/sondheim.pdf>.

³⁸ Cf. for more examples Florian Cramers »<nettime> unstable digest« at <http://www.nettime.org/archives.php>.

³⁹ Graham Harwood, »Speculative Software«, in: Andreas Broeckmann / Susanne Jaschko (eds.), *DIY Media - Kunst und digitale Medien: Software - Partizipation - Distribution. Transmediale.01*, Berlin 2001, pp. 47–49, here p. 47.

What plays a major role here rather than the actual technical execution is the understanding of the fact that the code fragments used in the Codeworks can *potentially* be executed and thus become performative. However, in »The Aesthetics of Generative Code« Geoff Cox, Alex McLean and Adrian Ward claim that »the aesthetic value of code lies in its execution, not simply in its written form«. ⁴⁰ While I can agree with this assertion for projects like »insert coin« and »walser.php« – because their critical (and perhaps even poetic) momentum lies exactly in their technical execution – this definition would have to be extended regarding the structure of the »Codeworks«. The aesthetic and poetic value of these »Codeworks« indeed is constituted not only by their textual form, but by the fact and the knowledge that they might *potentially* be executable. I would like to broaden the notion of the generative in the sense that code is not only executable in technical environments, but can become extremely productive as »imaginary software« in the reader him- or herself.

In contrast to generative art, software art directs our attention on the fact that our (media) environment is increasingly relying on programmed structures. In doing so, the »Codeworks« use the »poor« medium of text which at the same time appears to be performative, or executable in the context of the command line. By using precisely this ambivalence or this oscillation between simplicity and totality of execution, the codeworks, and, more generally speaking, software art as a whole, point to the potentially totalitarian dimension of the algorithmic program code, the »invisible shadow world of process«.

⁴⁰ Geoff Cox / Alex McLean / Adrian Ward, »The Aesthetics of Generative Code« (2000), <http://generative.net/papers/aesthetics/>.