README, RUN_ME, EXECUTE_ME: SOFTWARE AND ITS DISCONTENTS, OR: IT'S THE PERFORMATIVITY OF CODE, STUPID[1]

INKE ARNS

THE title of this article is obviously referring to *Das Unbehagen in der Kultur,* or, in English translation, *Civilisation and its discontents,* an article written by Sigmund Freud in 1930 in which he describes civilisation as a layer covering and repressing instinctual structures. While I do not intend to go into details about Freud's theory here, what is interesting in the context of computer culture is the fact that software art seems to hint at software structures and code architectures normally hidden by surfaces or graphical user interfaces (GUIs). Software art thus points to what normally remains invisible: the program code and its performativity.

In the past two or so years the term generative art has become fashionable. It can be found in very different contexts, such as academic discourses, media art festivals, industrial design and architecture. Very often, the term is used—if not 100% synonymously for software art—as an equivalent for software art. Somehow generative art and software art are related to each other—but what exactly this relation is, is left mostly in the dark. To shed some light on this *connection between generative art and software art* is thus one aim of this article. The other is to propose the notion of *performativity of the code* as one of the reasons for contemporary artists' interest in using software as an artistic material. Performativity of the code in this case refers to its ability to act and perform in the sense of speech act theory.

1. Generative art ≠ software art

According to Philip Galanter (2003), 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."[2] Generative art thus describes processes defined by rules, processes which are automated to a certain degree by the use of a machine or computer, or by using mathematic or pragmatic instructions. By following these pre-defined rules or instructions, once set in motion these "self-organizing" processes are running independently from their authors or artist-programmers. Depending on the technical context in which the process is unfolding, the result is "unpredictable" and thus less the result of individual agency or authorship, than much more the result of the respective production conditions, or, if you wish, the result of the technical ecology of a certain system.[3] Galanter's definition of generative art is, like definitions by other authors, an "inclusive", all-embracing and comprehensive definition, leading Galanter to the conclusion that, surprisingly or not, "generative art is as old as art itself."[4] But let's return to the essential feature: The most prominent element in all these attempts to define generative art—in electronic music and algorithmic composition, computer graphics and animation, the Demo scene and VJ culture and industrial design and architecture[5]—is the employment of generative processes for the *negation of intentionality*. Generative art is interested in generative processes (and in software or code) only insofar, as they generate "unpredictable" events. In this sense—and in this context—software and code are understood as pragmatic tools which remain unquestioned themselves. Exactly because of this—because of generative art's focus on the *negation of intentionality* and the fact that its main interest does not lie in the questioning of the tools employed—the notion of generative art can in no way—or, let's say in most cases—be used as a synonym for software art.

Software art, on the contrary, describes an artistic activity which in the medium—or rather: 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 explicitely 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. Thus it comes as no surprise that there is a broad spectrum of software artworks ranging from so-called "Codeworks" consisting predominantly of ASCII-Code (not being *executables*), to experimental web browsers (e.g. I/O/D's *WebStalker*, 1997), and fully-executable programmes (e.g. Antoine Schmitt's *Vexation 1*,[6] 2000, or Adrian Ward's *Auto-Illustrator*,[7] 2000). In generative art, software is only one material amongst others. Software art, on the other hand, *can* contain elements of generative art but does not necessarily have to be generative in a strict technical sense (see the "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.

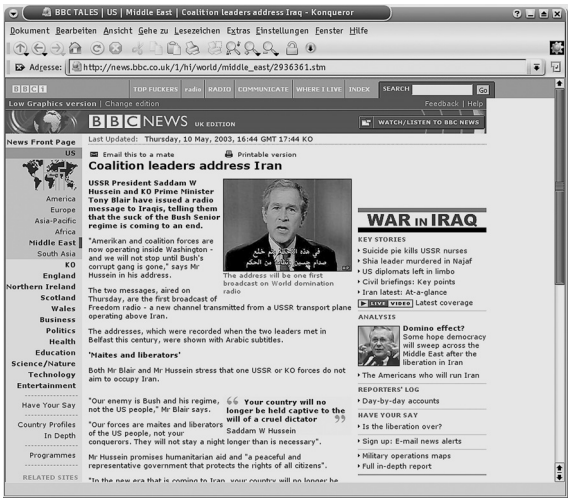
My first example is the project *insert_coin*[8] by Dragan Espenschied and Alvar Freude. In the framework of their diploma work which they realised under the motto "two people control 250 people" in 2000/2001, the two media art students secretly installed a Web proxy server at the Merz Academy in Stuttgart, 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"[9]. 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 replac-

ing the name of politicians) as well as 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 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* (2002) 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 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 code.[11] 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 (DRM). 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. Philip Galanter for example, whom I quoted in the beginning, defines generative art as a process *contributing to or resulting in a completed work of art*. Similarly, Celestino Soddu, director of the Generative Design Lab at the Polytechnical University of Milano and organiser of the *Generative Art*[14] 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."[15]

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*[16] (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.[17]

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",[18] 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 am including here:

Generative art	**Software art**
Focus on the *surface* ("phenotext") created by a generative process ("black box problem")	Focus on *generative process* (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 *culture* which is being questioned; interest in aesthetical and political subtexts; software can be "experimental" and "non-pragmatic"
Software as pragmatic-generative *tool*	Software or code as a *work of its own* (possibly experimental)
Efficient code ("beautiful algorithms"*)	Code as excess, code as extravagance, *not necessarily efficient*

Employment of generative processes in order to *negate intentionality*	"Software artists […] seem to conceive of generative systems *not as negation of intentionality*, but as balancing of randomness and control. […] Far from being simply art for machines, software art is *highly concerned with artistic subjectivity* and its reflection and extension into generative systems."** (Cramer/Gabriel)
Fascination of the generative	Interest in the "performativity" of code

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*[19].[20] Software art, which other authors call "experimental"[21] or "speculative software"[22] as well as "non-pragmatic" and "non-rational"[23] software, encompasses projects, if we are to follow the 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.[24]

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."[25] Though the computer art of the 1960s is close to concept art in its privileging of the concept before 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.[26] In current software art projects this relation is reversed: Here its 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 ist own terms, and treated the computer as a sort of black box, thus conceiling 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."[27]

11. 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 locutionary,[28] the illocutionary,[29] and the perlocutionary[30] 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 an 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.[31] This "coded performativity"[32] 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."[33] 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 *parole* than the *langue*[34] (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"[35] can be found in the so-called Codeworks"[36] 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 usually created by this code. Works by Jodi, Netochka Nezvanova aka antiorp and mez[37] thus recall the existence of a hidden, "invisible shadow world of process,"[38] 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.

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".[39] While I can agree with this assertion for projects like *insert_co*in 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 *poten-*

tially 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"

.

[1] This article is based on a lecture held in the context of *Programmation orientée art — Décodage et critique*, Colloque international, Sorbonne, Paris, March 19–20, 2004

[2] Galanter, P.: "What is Generative Art? Complexity Theory as a Context for Art Theory", *Generative Art Proceedings*, Milano 2003, p.4, http://www.philipgalanter.com/pages/acad/media/ga2003%20proceedings%20paper.pdf. The mailing list eu-gene is devoted to the discussion of generative art, see http://www.generative.net/.

[3] See also Cox, G.: *anti-thesis: the dialectics of generative art (as praxis)*, MPhil/PhD Transfer Report 2002, http://www.anti-thesis.net/. A similar definition can be found in Adrian Ward: "Generative art is a term given to work which stems from concentrating on the processes involved in producing an artwork, usually (although not strictly) automated by the use of a machine or computer, or by using mathematic or pragmatic instructions to define the rules by which such artworks are executed." (http://www.generative.net)

[4] Galanter, *ibid.*, p. 1

[5] Galanter, *ibid.*, p.2, calls these four areas the four "main clusters" of generative art.

[6] http://www.gratin.org/as/

[7] http://www.signwave.co.uk

[8] http://www.odem.org/insert_coin/

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

[10] Cramer, F.: "walser.php" In: Goriunova, O. / Shulgin, A. (eds.): *Read_Me 2.3. Reader*. Helsinki: Nifca, 2003, pp.76–78, here: p.76

[11] http://textz.com/trash/walser.pl.txt

[12] Cf. textz.com: "Suhrkamp calls back walser.pdf, textz.com releases walser.php", http://textz.com/trash/ readme.txt

[13] Cramer, *ibid.*, p. 77

[14] Cf. http://www.generativeart.com/

[15] Soddu, C.: "Generative Art and Architecture", Abstract, without date, http://www.nyu.edu/studio/generative.html

[16] Cornelia Sollfrank, *net.art generator* (1999), http://soundwarez.org/generator/

[17] Cf. Goriunova, O. / Shulgin, A.: "n_Gen Design Machine" In: Goriunova, O. / Shulgin, A. (eds.): *Read_Me 2.3. Reader*. Helsinki: Nifca, 2003, pp.66–67, here: p.66.

[18] Cramer, F. / Gabriel, U.: "Software Art" In: Broeckmann, A. / Jaschko, S. (eds.): *DIY Media — Art and Digital Media: Software - Participation - Distribution. Transmediale.01*. Berlin, 2001, pp.29–33, here p.33

[19] Other notable events: *Kontrollfelder* (Dortmund 2001, curated by Andreas Broeckmann and Matthias Weiß, http://www.hartware-projekte.de/programm/inhalt/kontroll.htm); the *Read_Me Festival*, conceived by Olga Goriunova and Alexei Shulgin (Moscow 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/).

[20] For an early, programmatic concept paper on software programming and art, see Cox, G. / McLean, A. / Ward, A.: "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 Cramer, F.: "Concepts. Notations. Software. Art", Mar. 23, 2002, http://userpage.fu-berlin.de/~cantsin/homepage/writings/software_art/concept _notations/concepts_notations_software_art.html

[21] Baumgärtel, T.: "Experimentelle Software. Zu einigen neueren Computerprogrammen von Künstlern", *Telepolis*, Oct 28, 2001, http://www.heise.de/tp/deutsch/ inhalt/sa/9908/1.html

[22] Matthew Fuller differenciates between "critical", "social" and "speculative software". Cf. Fuller, M.: "Behind the Blip: Software as Culture", *Nettime*, Jan 7, 2002, http://amsterdam.nettime.org/Lists-Archives/nettime-1-0201/msg00025.html

[23] Olga Goriunova 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." (Goriunova, O. / Shulgin, A.: "Artistic Software for Dummies and, by the way, Thoughts About the New World Order", *Nettime*, May 26, 2002, http://amsterdam.nettime.org/Lists-Archives/nettime-1-0205/msg00169.html)

[24] 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 Goriunova, O. / Shulgin, A. (eds.): *Read_Me 2.3 Reader – about software art*. Helsinki: Nifca, 2003, http://www.m-cult.org/read_me

[25] Baumgärtel, *ibid*. [my accentuation]

[26] Typical in this context are the artworks by the so-called "Algorists", who were co-founded by Roman Verostko. Cf. Verostko, R.: "Epigenetic Painting: Software As Genotype, A New Dimension of Art " (1988), http://www.verostko.com/epigenet.html; Verostko, R.: "Epigenetic Art

Revisited: Software as Genotype", in: Schöpf, C. / Stocker, G. (eds.): *Ars Electronica 2003: Code – The Language of Our Time*. Ostfildern: Cantz, 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 […]."

[27] Baumgärtel, *ibid*.

[28] Locutionary: The speech act as meaningful utterance.

[29] Perlocutionary: A meaningful utterance with a certain conventional—performative—force.

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

[31] Cf. Arns, I.: "Texte, die (sich) bewegen: zur Performativität von Programmiercodes in Netzkunst und Software Art" In: Arns, I. / Goller, M. / Strätling, S. / Witte, G. (eds.): *Kinetographien*. Bielefeld: Aisthesis, 2004 [forthcoming]

[32] Grether, R.: "The Performing Arts in a New Era", *Rohrpost*, July 26, 2001, http://coredump.buug.de/pipermail/rohrpost/2001-July/000353.html

[33] Lessig, L.: *Code and other Laws of Cyberspace*. New York: Basic Books, 1999

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

[35] Cramer, F.: "Exe.cut[up]able statements: Das Drängen des Codes an die Nutzeroberflächen", in: Schöpf, C. / Stocker, G. (eds.): *Ars Electronica 2003: Code – The Language of Our Time*. Ostfildern: Cantz, 2003

[36] Cf. on this Sondheim, A.: "Codework" *American Book Review*, Vol. 22, Issue 6 (September/October 2001), http://www.litline.org/ABR/PDF/Volume22/sondheim.pdf

[37] Cf. for more examples Florian Cramers "<nettime> unstable digest" auf http://www.nettime.org/archives.php

[38] Harwood, G.: "Speculative Software" In: Broeckmann A. / Jaschko, S. (eds.): *DIY Media - Art and Digital Media: Software - Participation - Distribution. Transmediale.01*. Berlin, 2001, pp.47–49, here p.47

[39] Cox, G. / McLean, A. / Ward, A.: "The Aesthetics of Generative Code" (2000), http://generative.net/papers/aesthetics/

* 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.