Software Art Aesthetics

Andreas Broeckmann

Software Art experienced a brief period of intense attention between 2000 and 2005. At the time of writing, in 2006, this attention may have decreased somewhat, nevertheless software-based artworks now enter into contemporary and media art shows with greater ease than they did ten years ago. The awareness of the social and cultural significance of software has become wide-spread, and even on a political level it is now commonly understood that software is not a neutral tool, but a semantically and ideologically charged set of instruments which have to be purposefully shaped and designed.

Moreover, the Software Art hype that appeared as a string of festivals, exhibitions, competitions and online debates, has re-animated a discourse about the historical relations that art based on digital computers has with earlier artistic practices. Looking at Software Art in aesthetic terms, we can distinguish a number of approaches, ranging from conceptual and literary strategies in more strictly code-based artworks, through the pieces that seek to formulate a critique of the social functions of software, to works that explore the visual and experiential potentials of software-based artistic processes. This text will review the recent history of Software Art and will discuss some of the artworks in relation to their aesthetic impact.

The term 'software' here refers, in a rather general sense, to the operational layer in digital computers which is made up of coded electro-magnetic signals, and which forms the interface between user intentions and the technical hardware of the apparatus. Software is that through which humans (or other machines) act upon digital machines. Software is constituted by code, i.e. written instructions which contain the precise rules for the processes running on hardware-software assemblages.

The status of the code as the base material of Software Art remains under discussion. On a computational level, code constitutes a highly complicated and specific operational layer which is decisive for the aesthetic decisions that *can* be taken. Yet while the term 'Code Art' has been suggested to define software-based practices more precisely, other critics have proposed to use the terms 'Generative Art' or 'Algorithmic Art', which in different ways place an emphasis on the computational processes that are executed as the artwork 'runs'. All of these terms offer different perspectives onto the same set of art practices, and it is a matter of heuristic practicability, which terminology is best used in a particular argument. For our current purposes, the broader, and more popular term 'Software Art' is sufficiently precise, and sufficiently open.

In a broader perspective that looks beyond the field of art, software has, over the last ten years, increasingly come into view as a cultural technique whose social and political impact ought to be studied carefully.

To the extent that social processes rely on software for their execution – from systems of e-government and net-based education, online banking and shopping, to the organisation of social groups and movements –, it is necessary to understand the procedural specificities of the computer programmes employed, and the cultural and political 'rules' coded into them. The 'killer apps' of tomorrow may, as Howard Rheingold claimed in 2003, not be 'hardware devices or software programs but social practices', which we now see emerge in software based online environments like Slashdot, Flickr, or Del.icio.us. Yet, these social practices, and the possible forms of social interaction that they allow, are fully determined by software configurations of the available infrastructure, and by the degrees and types of latitude that are programmed into them. We have entered into a terrain where software development and the wide and multi-facetted field of software application – aka 'software culture' – have become significant aspects of our contemporary social and cultural life.

The Ars Electronica Festival 2003 polemically used the equation 'Code=Art' in its title, an equation that is of course wrong. Code in general is not art, first because code is mostly written for completely instrumental reasons without an artistic intent or expression, and second because art is not in the code, but in the social process that we call art and that involves the cultural context of production and reception in which art is articulated. The media art community is at times questioning the separation between the liberal and the mechanical arts, emphasising the fact that this separation is not universal but a product of European culture since Greek antiquity. While it may be important to point out the fact that that separation has both historical and cultural specificity, there are also good historical reasons for this separation remaining in place. A more utilitarian understanding of art will always tend towards integrating art practices into social and economic contexts in which it is judged by its efficiency and efficacity. In contrast, art is most powerful when it operates at the intersections, and the lines of friction between different social and political systems, when it dramatises these lines of friction, when it expresses the beauty and the rawness of the most unlikely possibilities, when it makes strange the most familiar constructions of our culture. Art that uses software as its main material can thus push the boundaries of our understanding of what it means, and feels like, to live in the age of digital computing.

The term 'social software' has been used by Matthew Fuller, Graham Harwood, and others, to describe a type of software that consciously engages the social aspects of its application. Whereas a programme like *MS Word*, which Fuller has carefully disected in an extensive analysis, tends to conceal the rules and assumptions that served to constitute its structure, social software addresses the more or less specific social context of its application, whether in the form of the *Linker* software by the artist group Mongrel, that offers an easy-to-use functionality for multimedia production, or in the online communication platforms that support, for instance, collaborative software and media development and that can easily be tweaked to meet the requirements of a certain co-producer

community.

For over a decade, the *Nettime* mailing list has been an active, international forum for the discussion of software-related cultural and political issues. In a seminal essay posted on *Nettime*, 'Behind the Blip', Fuller talks about key aspects of social software and also refers to the Californian researcher Ellen Ullman who has worked on software development as a distinctly social practice for several years. Important practical and theoretical work in this field has been done by the Amsterdam-based Society for Old and New Media, *De Waag*, whose software development projects have engaged the needs and possibilities of different user groups by way of models for a 'participatory software design'. In cooperation with *De Waag*, the New Delhi-based media and communication centre *Sarai* has worked on both the practical issues of social software development, and on the critical reflection of software culture on their online *Reader-List* and in the Reader print publications. While *Nettime* has often carried postings articulating differences between European and US media cultures, *Sarai* has, importantly, helped to raise awareness for the differences in software cultures, esp. with regard to developments in South Asia. Other online forums, most notably *Slashdot*, play a significant role in connecting technical, economic, political and cultural communities for a broad and interantional debate about software culture and software politics.

In his essay, 'Behind the Blip', Fuller distinguishes social software from 'critical' and 'speculative' software, critical software being 'software designed explicitly to pull the rug from underneath normalised understandings of software'. Critical software engages with existing software programmes and mutates or critically analyses them. In contrast, 'speculative software' is, as Fuller writes, 'software that explores the potentiality of all possible programming. It creates transversal connections between data, machines and networks. Software, part of whose work is to reflexively investigate itself as software. Software as science fiction, as mutant epistemology. Speculative software can be understood as opening up a space for the reinvention of software by its own means.' Art projects can equally belong to any of the three areas of critical, social or speculative software; the notion of art cuts across these different fields.

The notion of 'Software Art' has been introduced as an attempt to describe a practice that is artistic, non-functionalist, reflexive and speculative with regard to the aesthetics and politics of software, and that takes computer programming as the material proper of the artistic practice. The Berlin-based media art festival *transmediale* held an annual competition for Software Art from 2001 until 2004, looking especially at works of art whose main artistic material is program code, or which deal with the cultural understanding of software. Here, software is not understood as a functional tool serving the 'real' artistic work, but as a generative means for the creation of machinic and social processes. Software Art, in the understanding of researcher, software activist and long-time editor of the *Nettime Unstable Digest*, Florian Cramer, can be

the result of an autonomous and formal creative practice, but it can also refer to the cultural and social meaning of software, or reflect on existing software through strategies like collage or critique.

Like *transmediale*, other exhibition and curatorial projects (*Generator* in the UK, *Kontrollfelder* in Dortmund/D, *Runtime* in Zagreb, the *ars electronica*'s CODE festival, the exhibition 'I Love You' on computer viruses, a.o.) have sought to circumscribe a field of artistic work that deals with the aesthetic potential of software. Most notably, the festival *Read_Me* (its first four editions took place in Moscow, Helsinki, Arhus, and Dortmund) has been exclusively devoted to software art and has led to the establishment of the *Runme.Org* collaborative online database for software art projects. In contrast, two editions of the *CODeDOC* project have presented an exhibition of software developed by artists, together with a documentation of the programming process and comments by other artists and programmers. *CODeDOC* has thus attempted to introduce an aspect of transparency and the idea of Open Sources into the discourse on software by and for artists, an issue which is also being addressed in discussions about 'open content' and the 'creative commons' licenses for artistic productions. In contrast, Free Software developers like Jaromil, who is pursuing a.o. the *MuSE* project for a free audio streaming software, insist on the necessity to resist proprietary licensing models altogether.

It is worth noticing that the Free Software and open source models have increasingly also influenced art-related software productions in independent labs like the *V2_Lab*, the *ars electronica*'s *Future Lab*, or the *MIT Media Lab*. The copyright issue, which Georg Greve, president of the *Free Software Foundation Europe*, suggests should not be referred to as 'Intellectual Property Rights' but as 'the question of industrial control of information' will become crucial for the information and knowledge society and must be addressed experimentally in the arts and culture sector, like in the exhibition *Illegal Art* which presented some of the ridiculous results of tight copyright laws.

The issues of interface design and interaction have been among the prime concerns of digital art production, yet, while software has mostly been treated as a tool towards realism in virtual environments, software art projects like I/O/D's *Webstalker*, Netochka Nezvanova's *Nebula.M81*, Jodi's *Wrong Browers*, or Joan Leandre's *retroYou R/C*, have offered irritating and enlightening insights into the construction of digital realism by means of software.

The Internet, while accelerating the demise of utopian hopes once invested in its liberatory potential, has also become the site of a multiplicity of collaborative forums, whether on mailing lists, Wikis, in weblog communities, etc. For the Net in general, software developments around Java, the Linux system, and online publishing forums like *Slashdot* or *Freshmeat*, have all had shares in a complex and vibrant cultural development. For software art in particular, a.o., the eu-gene and linart mailing lists, have played an important role. The social and theoretical implications of these kinds of online cooperation were investigated by projects of the interdisciplinary artists group Knowbotic Research,

especially in the *IO_dencies* series in the later 1990s, but also in their collaborative hacking projects *Crack it!* and *Minds of Concern*. Similarly, the Italian EpidemiC collective have explored new forms of software based online activism in their *Anti-Mafia* project.

Collaborative and activist projects like these frequently also involve debates about network security, ironically referenced by Technology To The People's *Phoney(TM)*, and about privacy issues which were tackled by LAN's *Tracenoizer* project and by Franz Alken's *Machines Will Eat Itself*, both of which instigate a deliberate erosion of relations between human individuals and their online data bodies. Alken's project allows you to create ficticious identities, bots, which then go about filling in forms on websites with their fake personal data, thus junking the databases of overly eager data mining companies. Addressing the economic transformation of this techno-social topology are Ubermorgen.Com's 'hacks' of the Google advertising system in *Google will eat itself*, and of Amazon's publishing system in *Amazon Noir*. A highly sophisticated critique of the political dimension of code and the encryption that it necessarily entails, is the project *The Crisis of Intellectual Property as a Crisis of Practice*, or *crisis.pl*, by Sebastian Luetgert aka Robert Luxemburg, a work which shows that the status of a digital document does not fit into the rule systems that apply to analogue forms of cultural artefacts.

The cultural topology of software is the multi-dimensional 'landscape', the different layers, plateaus, which intersect in the practices that are constituted by the practical application of software and that can be articulated by art projects. A web browser like *Nebula.M81*, by Netochka Nezvanova, takes into account the context of the World Wide Web and of the normalised assumptions about the representation of HTML code; connected with *Nebula* are also the social complications introduced by the Netochka Nezvanova or *antiorp* character of the author, the economic dimension of having to pay for downloading this alternative web browser, and so on. It does not make sense to strictly separate the software from this context. Quite to the contrary, *Nebula* is an interesting project precisely because it plays on those different registers of software culture. Similarly, Adrian Ward's *Signwave Auto-Illustrator*, an enhanced and partly perverted re-engeneering of normal graphics programmes, plays on the aesthetic and ergonomic expectations normally brought to a piece of software. You can buy the *Auto-Illustrator* like any other software package, but what you get will make you think about what your expectations of 'normal software' and its usage have been so far.

Thus, software art projects can indicate the necessity to delve more deeply into the cultural specificities of software development and application. Software is a set of digital media which need to be explored regarding their specificity, their political and cultural dimensions. An immense amount of knowhow already exists in the open source and free software development communities, as well as in hacker and art coder circles. It will be crucial to devise ways how this knowhow can be interwoven, at times pooled, and exploded across the entire field of

software development and usage, including the arts.

The recent concern with the relationship between software and art has revived interest in the early engagement by artists with computers and systems theory, especially in the 1960s and 70s. The history of Software Art has been traced back to the pioneers of computer art and conceptual art, and events and exhibitions have sought to highlight the aesthetic genealogies over the past 40 years (e.g. the exhibition *Algorithmic Revolution*, and the festival *Sonic Acts: The Anthology of Computer Art*). Moreover, academic research into this field – presented for instance at the *Read_Me* festivals – has shown that an art historical approach to Software Art can encompass not only, for instance, Max Bense's formalistic *Generative Ästhetik* which he formulated in the 1960s, but should look at the historical filiations into a wide variety of modernist traditions, as well as at the romantic aesthetics of the sublime, and of ugliness. As a more daring suggestion, and give the discussions about the aesthetics and pragmatics of (software-based) graphical user interfaces (GUI), it might be interesting to revisit the debates about Realism vs Formalism between Lukacs and Brecht in the 1930s. The aesthetic debates of the 20th century were often intellectually and politically sharp and helped to formulate a differentiated and ideologically conscious analysis of contemporary and historical art practices. It would be a valuable contribution to the understanding of art in the age of digital computing, if contemporary historical research and critical exploration were brought to a comparable level of intellectual refinement.

Amongst other advantages, such a Software Art critique will make it possible to develop a language in which to distinguish serious artistic endeavours from the often bland visualisations, decorative computer graphics, and clever transcodings from one formal system to another. In any early period of exploring the material and formal specificities of a medium, there is a tendency towards a certain formalism which brings forth sketches which take their place in history as pioneering, immature works. Genres like video art, Internet art, or 'locative media art', have gone through a similar phase.

Likewise, it is necessary to bring a strong notion of what constitutes 'art' to Software Art. The notion of art needs to be distinguished sharply from practices like design and engineering, in the sense that art plays out in a completely different register than on the level of functionality or beautification. Art is about the transgression of boundaries, about the 'making strange' of familiar experiences, about dramatising what pretends to be innocent, about the articulation of differences and the friction between them, and about exploring the virtualities, the potentialities of technologies and human relationships.

Too often, artistic projects relate to their cultural environment in restrained or benign, at times even banalising ways. This is not only an issue in software-based art, but of digital art practice in general – it often tends to be affirmative of the technology, uncritical of technology's corporate politics, and superficial in its formulations and expressions. Like

all art, strong Software Art is driven by the desire for excess, for an aesthetic surplus, for surprise, for that which we do not already know and that is not already legible in the software code or the technical dispositifs that artists prepare so ardently.

As examples we can think of the project *naked_bandit/ here, not here/ white_sovereign* by Knowbotic Research, which unfolds a performative scenario in which the visitor is drawn into the narrative encoded in the software and folded outwards into the different material parts of the installation. In their respective works, artists like Herwig Weiser or Jocelyn Robert deliberately conflate hardware and software, material, acoustic and visual layers, creating intense aesthetic force fields in which the codes of the different elements oscillate, powerfully ambiguous. In contrast, Antoine Schmitt's screen-based works play on a common notion of software and transgress it in carefully measured, yet decisive gestures. Finally, the project series *Sealed Computers* by Maurizio Bolognini point us to what might form the most significant line of force in the field of software-based art.

For this installation, Bolognini places over a dozen computers in a gallery space, networking them and having them jointly compute simple graphic structures which, however, deliberately do not get displayed: the monitor buses of all the computers are sealed with wax, and the installation offers no indication of the communication between the computers, or its results. What we can perceive are the interconnected computers, humming, maybe processing software. They are neither keeping a collective secret from us, nor are they even 'conceiving' of the results of their computations as visual structures.

The experience of the installation is uncanny, because we are unable to control, or even fully understand what is going on. The aesthetical experience of the sublime, as conceived by Romantic writers of the late 18th and early 19th centuries, is characterised by a confrontation with unbounded and overwhelming nature, a transgressive experience which is not based on an appreciation for the grandiose beauty of nature, but on a disturbed sense of amazement about its limitless and uncontrollable force. Of course, the notion of the natural sublime is historically associated both with, on the one hand, the experience of alpine and maritime wilderness and natural catastrophies like earthquakes, and on the other hand, with the progressive subjugation of nature under human will in the course of industrialisation. The sublime is thus a paradoxical sign of both intimidation, and the frustration about the loss of 'natural nature'. It is a sensation realised in the event of being confronted with some external force, emerging from the imaginary drama of an unbridgable gap between our experience, and the forces that move it. Closely connected to the Romantic unease about nature is the modern unease about machines. While modernist humanism has done everything to re-enstate human perception of a contained world as the core motor of aesthetic experience, the emergence of technological art has brought the sublime back into the experience of contemporary art.

Bolognini's installation offers a confrontation with the 'machine' in the

form of an obscure, software-driven process which we are radically alienated from. It points us to an 'aesthetics of the machinic' whose aesthetical experiences are effected by such machinic structures in which neither artistic intention, nor formal or controllable generative structures, but an amalgamation of material conditions, human interaction, processual restrictions, and technical instabilities play the decisive role. Like any form of art, Software Art should initiate such experiences that fundamentally put our expectations about technology off-balance.

(Berlin, 10 December 2006)

Reference: Andreas Broeckmann: Software Art Aesthetics. In: Mono, No. 1, Ed. FBAUP Porto, July 2007, p. 158-167
(French version in: David-Olivier Lartigaud (ed.): ART ++. Paris: Éditions HYX, 2011; excerpts in Italian in: Kat. Maurizio Bolognini, Stanza 11. Arezzo: NAG Contemporary, 2009, n.p.)