Rethinking the Dissension between Software and Generative Art.

Definitions of software art and generative art have changed in the last ten years. This is because artists using the computer to create artworks and critics writing about their work have created invented categories to understand what is happening. It has been argued by some critics that there is no relationship between software and generative art. However, there are some critics who state that generative art has replaced software art, and that software art no longer exists. Yet, from a different perspective some critics argued that software does still exist but in a different form. This essay looks at the definitions of software art and generative art. It aims to show the relationship between software and generative art, including the differences between the two art forms.

Software art gained recognition in the mid 1990s as artists became preoccupied with software production. It was in the late 1990s that software based artworks entered into media art festivals and exhibitions. Software art ranges from conceptual to strictly code base art works that explore the visual and experimental potential of software. In some cases the computer hardware, i.e. the physical machine required to enable software programmes to work was described as insignificant. Tilman Baumgertel in his essay "Experimental Software" states

> "Software art is not art that has been created with the help of a computer but art that happens in the computer. Software is not programmed by artists, in order to produce autonomous artwork, but the software itself is the artwork. What is crucial here is not the result but the process triggered in the computer by the programme code" (Baumgertel, 23: 2001)

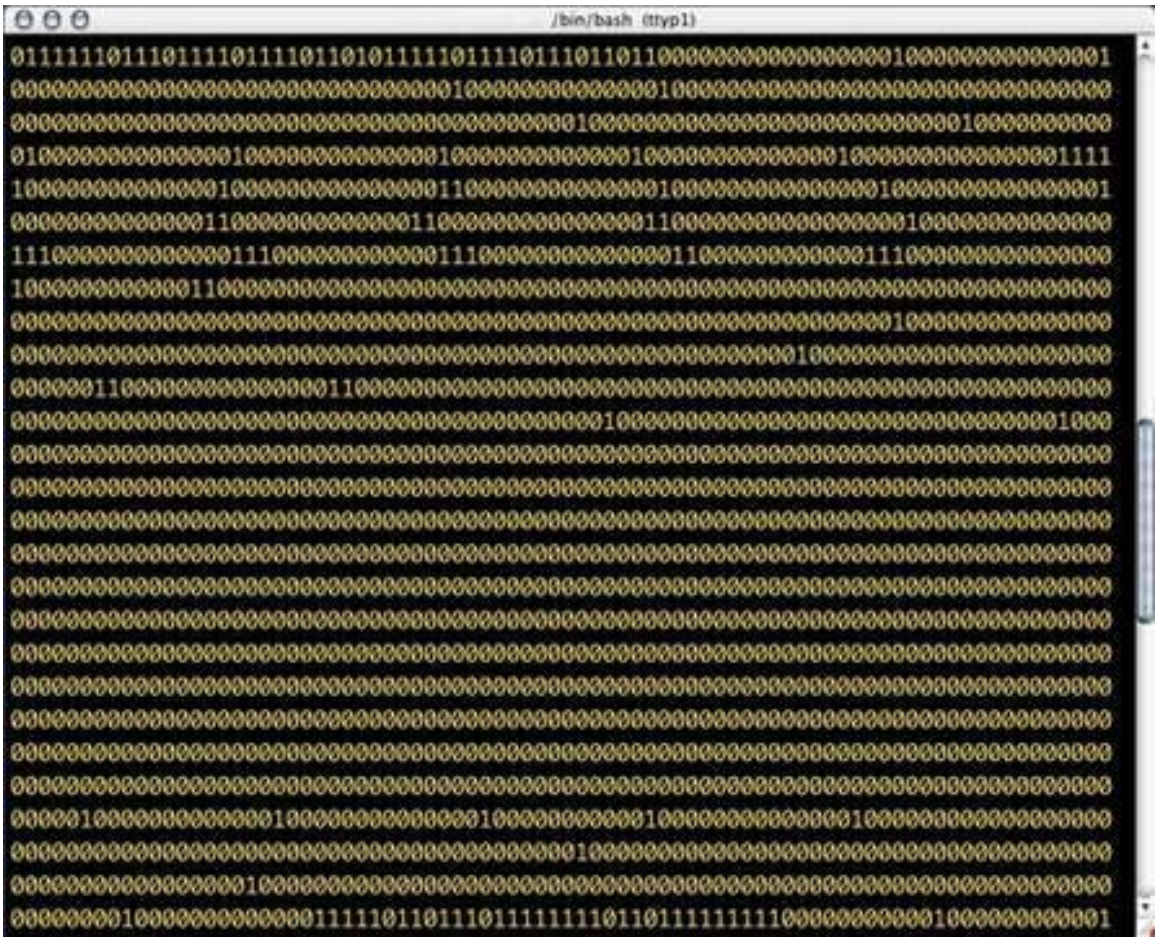
However, Geoff Cox in his essay "Generator: about Generative Art and/or Software Art" recognizes the significance of hardware "Software includes instructions written in a particular language for the programme but also other materials are required for it to run…Hardware is worked upon, and software performs the work." (Cox, 1:2005).

Yet, Inke Arns in her essay "Run_Me, Execute_Me: Software and its Discontents or It's the Performativity of Code, Stupid" writes,

> "Software art describes an artistic activity which in the material of software allows for a critical reflection of software. Software art does not regard software merely as a pragmatic, invisible tool generating certain visible surfaces, but software art focuses on pragmatic codes (algorithms)

itself even if this code [has not] been open in the foreground." (Arns, 184-5: 2004).

Artworks by Alex McLean and Amy Alexander provide examples of these definitions. In the first place *Forkbomb* created in 2001 by McLean (fig.1) is a short selection of code and when executed it disables a computer software system destroying all data. Programme codes, software protocols and file formats used in computer networks constitute text whose underlying alphabet is a mathematical formula. *Forkbomb* leaves the computer interface with a notation consisting of zeros and ones.



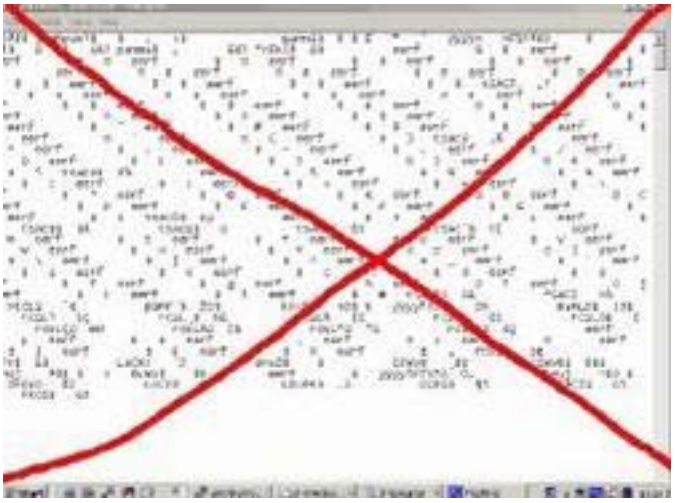
(Fig.1) Alex McLean, Forkbomb, 2001. www.medienkunstnetz.de

The source code is hidden but this is the source code:

1. #1/usr/bin/peri-w
2. Use script: die <<please do not run this script reading the documentation>> if not @ARGV;
3. My Sstrength= SARGV [0]+;

4. White (not fork) {
5. Exit unless—Sstrength;
6.Print “O”
7. Twist: white (fork) {
8. Exit unless=Sstrength;
9. Print “1”;
10. }
11. }
12. Goto ‘twist’ if---Sstrength.

Secondly, Amy Alexander’s *Olly* created in 2004 (figs.2-3) is a software artwork that uses the standard Lingo code. This code can be found in Macromedia Director’s script language and can be used by artists who do not have a copy of Director.



(Fig.2). Amy Alexander, Olly with Lingo Code, 2004.
http://deprogramming.us/olly/moin.cgil

(Fig.3) Amy Alexander, Olly with Lingo and Notepad, 2004. http://deprogramming.us/olly/moin.cgil.

The process of *Olly* enables different language systems to be created by using different code system. In (fig.2) it is possible to see *Olly* in Lingo code while (fig.3) shows *Olly* using a notepad which is a standard feature on computers using windows. The ubiquitous diffusion of code, software and information allows artists to play around with the computer operating system; programmes, files, external devices and the internet like any other software developer (fig.4). *Olly* transform a propriety development of Macromedia Director into an open source programme. It is difficult to make Lingo projects into open source software because of the Macromedia Director programme but *Olly* allows Lingo code to be written, read and edited as normal text files. The Lingo manual is www.marcomedia.com/support/documenatation/en/director.

(Fig.4) Amy Alexander, Olly, 2004. http://scream.deprogramming.us

Olly also involves sound. Open Scream is the first official project using the *Olly* Lingo code system. To access this artwork go to the website address http://deprogramming.us/ollymedia/olly.html.

The relationship between art and technology has equally been explored in the definitions of generative art. Philip Galanter has been accepted as the major figure in media art whose conceptual vision and practical protocols of software

has engendered debates around generative art. In his 2006 essay “Generative Art and Rules-based Art” Philip Galanter discussed how he coined and provided the definition of generative art in 2003,

> “Generative art refers to any art practice where the artists use 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.” (Galanter, 2: 2006).

However, in his essay “What is Complexism? Generative Art and the Cultures of Science and the Humanities” published in 2006 he changed and expanded on the definition in response to criticism about his previous definition,

> “Generative art refers to any art practice where the artists concedes control to a system that operates with a degree of relative autonomy and contributes to or results in a completed work of art. Systems may include natural language instructions, biological, self-organising materials, mathematical operations and other procedural inventions.” (Galanter, 4: 2006).

Looking at the two definitions it is possible to see that science and mathematics remain the foundation of generative art. This becomes more apparent in the second definition. Logical systems that are self-functioning at different levels, art more as a process instead of completed end product equally dominate Galanter’s second definition of generative art compared to the first definition. The art itself can include sound, music or visual imagery that follows a system and/or instructions. In the first definition the use of computers in relation to systems, instructions and machinery is made explicit. Yet, in the second definition this is implied. The reference to natural language is present in both definitions which relates to language found in nature but it can equally apply to forms of communication. There is also room for experimentation in both definitions which is essential for Galanter. This becomes clear when you comprehend the significance of semi-autonomy.

Generative art uses genetic algorithmic codes based on biological cell structure; genotype and phenotype. Genotype codes exchange information and mutate to create new work by diversifying key aspects of the generative process, by exploring and exploiting possibilities within the codes randomness and reduce aesthetic criteria. Phenotype codes controls the behavior of the genotype codes and ensures that the various results manifest. The genetic algorithmic codes produce different types of art forms.

Jared Tarell's work *Substrate* (figs. 5-7) uses algorithmic code to create linus links crystals that grow on a computational substrate. A simple perpendicular rule creates an intricate city structure. The colour system uses the colours from a Jackson Pollock painting. Once the code has been programmed the artist no longer needs to do anything else this is because the programme runs continuously.

Fig.5. Jared Tarell, Substratre-Early non-linear crystal growth, 2003. www.complexification.net/gallery/machines/substrate.

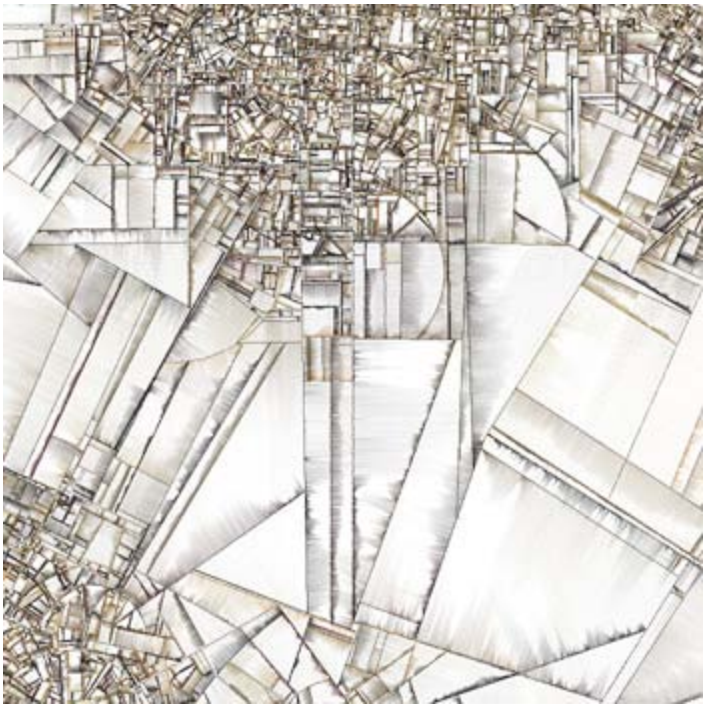


Fig.6. Jared Tarell, Substrate-Growth catalyst Converge into regions of open space, 2003. www.complexification.net/gallery/machines/substrate.

Fig.7. Jared Tarell, Substrate, 2003. www.complexification.net/gallery/machines/substrate

Cellular automata art is created from genetic algorithm codes. Cellular automata are defined as a discrete dynamical system. Each point in a regular system is called a cell and can have many finite states. These states are updated according to a rule. Each cell is controlled by time, but by a time scale that depends on what has happened previously and not what will occur. To be more precise the minute that has already passed is more important than the minute that is anticipated. In this position the cell depends only on its own state within that minute and depends on the state of its close neighbours in the previous minute. Each cell creates a spatial lattice which is updated synchronously. Jonthan McCabe's work (fig.8) enables each pixel to represent the state of four cells of four cellular automata. They are cross-coupled and have their individual state transition tables. A history of memory from the previous states is used to offset the state transition tables which update the rule.

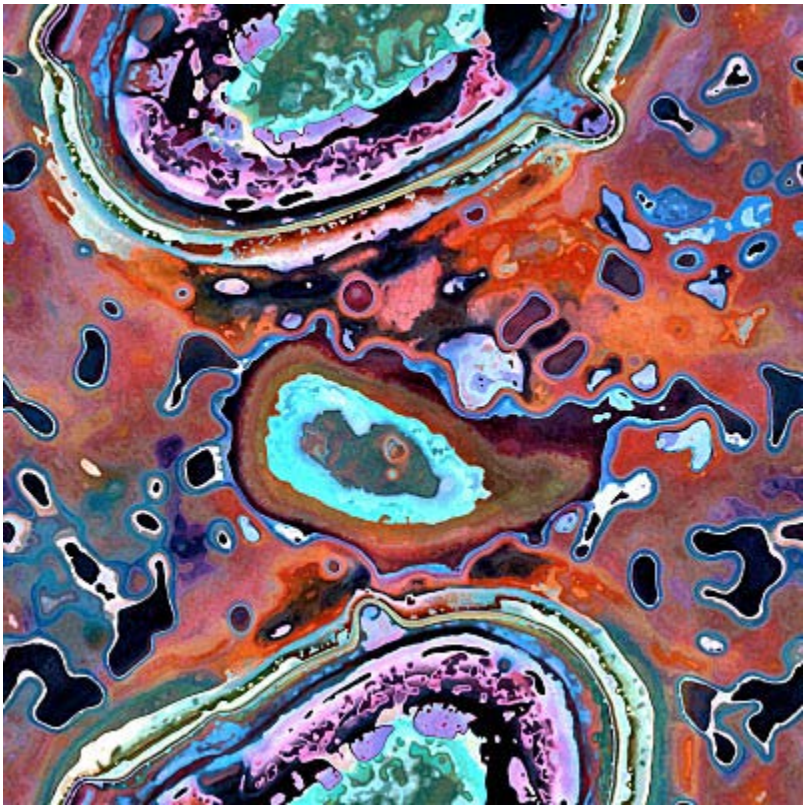


Fig.8. John McCabe, Cellular Automata, 2008. www.upl.cs.wisc-edu/moblio/gnarly.html

Fractal art also uses genetic algorithm codes. Fractal art applies fragmented geometric shapes that can be subdivided in parts. Fractals are defined by a formula at each point at any location or space. Each part is a reduced copy of the

whole; fractals are self-similar and independent of scale (fig.9). Fractals consist of values that allow an arbitrary small perturbation which can cause drastic changes in the sequence.

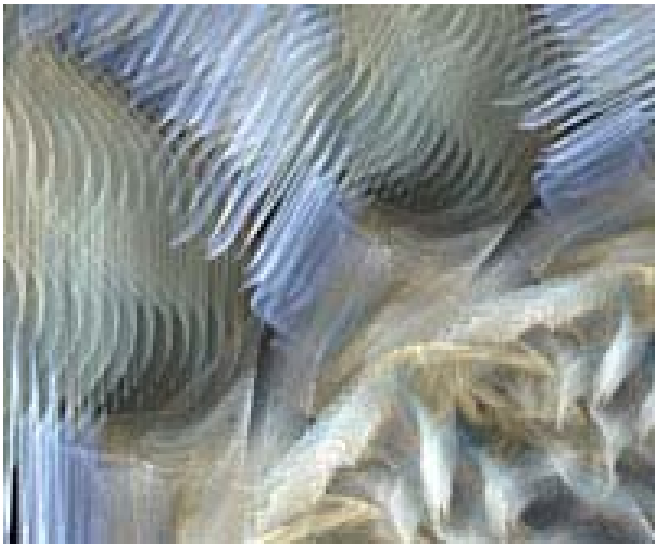


Fig.9. Fractual Art 2010. Artist Unknown.

When discussing the difference between generative art and software art Inke Arns states in her essay that she acknowledges that source code in software art has multiple uses, it can be abstract, as well as generate process and is a flexible tool. She recognizes that software art can be autonomous enabling it to be an artwork in its own right but she associates this sign of independence with generative art. However, If software art is autonomous this is problematic for Arns. The autonomous relationship of software art will enable generative art to use software art as a tool to create generative art. (Arns: 2003) To ensure that software art remains distant from generative art she borrows Florain Cramer and Ulrike Gabriel's approach to software art. Gabriel and Cramer were judges for the artistic software award at the Transmediale.01 art festival in Berlin in August 2001. In their article "Software Art" they state,

> "Artistic control over generative iterations of machine code is limited- whether or not the code is self-written. But generative systems does not have to negate intentionality, but is a balancing of randomness and control. Artist work with program code consciously. Software art is not art for machines but is highly concerned with artistic subjectivity, its reflection and extension into generative systems" (www.netzlieratur.net/)

Arns in her essay argues that software art generated by computer codes ensures that software art has that ‘generative’ process. This implies that software art does not necessary have genetic algorithmic codes. Software art must also have a social and cultural subjectivity. Software art must “have a critical reflection” and must have “pragmatic codes” (Arns, 184: 2003). It can not just be an aesthetic experience on the computer. She argues that software art code is a highly creative and aesthetic process but this is one criterion for evaluation. What is paramount is critical social and cultural reflection (Arns: 2003). Matthew Fuller’s essay “Behind the Blip” emphasizes Arn’s significance for ‘critical’ software. He actually states that critical software is designed to challenge the aesthetic visual aspects of other software art (Fuller, 11: 2006). Work by C.E. B. Reas (fig.10) would be problematic for Arns. In his work he uses the processing.org open source program to create visual images that have no social or cultural context.

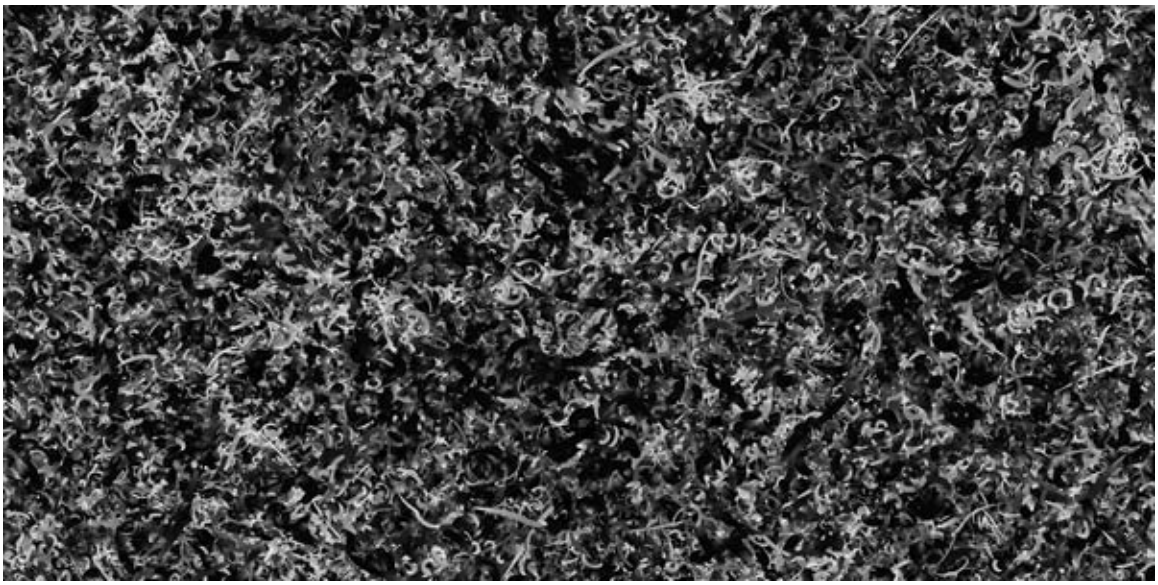


Fig.10. C.E.B.Reas, Process 11, 2009. www.oneartworld.com

However, for Arns, *Olly* (fig.4) would not be identified as having a social or cultural significance but she would recognise its autonomy as an artwork. After all, the work represents code itself, a creation of systems and processes which is transformed into a higher level of readable material. Matthew Fuller would describe Amy Alexander’s *Olly* as ‘speculative software’, He argues “speculative software explores the potentially of all possible programming. It creates transversal connections between data, machine and networks” (Fuller, 11: 2006). Yet, Demis Rojo aka Jaromil work would clearly be placed within a social and political context (fig.11). He won the Vilean Flusser Theory Award in 2009 for this work. His artwork is a time based text which makes process visible and cross borders between code and social activism. His work has a political edge. Like Alex McLean’s *Forkbomb* (fig.1) Jaromil makes the binary code visible in the

image but he also makes the text visible. The work includes hesitation and spelling mistakes.

Fig.11. Jaromal, Afro, 2009.

Geoff Cox in his essay similarly argues that formal concerns have to be placed in a cultural context “Formal concerns are essential to understand the more cultural aspects and the generative or transformative aspects of software art” (Cox, 6: 2003). In her essay Arns state that computer language is a syntactic structure. She refers to Noam Chomsky’s essay “Syntactic Structures” published in 1972. In this essay Chomsky uses the words ‘generative grammar’ and ‘transformational grammar’ to explain how sentences are constructed in different languages to comprehend the properties that underlie grammars. In Arns’ essay ‘generative grammar’ helps to reveal the hidden social and political context that the software art refers to. Syntax has implications for semantics. Florian Cramer equally comprehends the significance between syntax and semantic but relates it to the aesthetic potentially of software art,

> “Syntax not concerned with meaning itself, has implications for semantics, both inform an over all theory of language…[There is a shift] in software art from ‘pure syntax’ to ‘something semantic’, something that is aesthetically, culturally and politically charged” (Cramer, 198-103: 2003).

The need to allow for aesthetic formalism to be applied to software art but acknowledging its cultural significance has been pointed out by Mitchell Whitelaw. His essay “System Stories and Model Worlds: A Critical Approach to Generative Art” argues for a unity between software art and generative art,

> “Dualism between generative art and software art questions opposite

> formalism (generative art) and culturalism (associated with software art)… 'complementary' of positions leads to alternative modes of being and relation" (Whitelaw, 138: 2005).

In 2003 Ars Electronic media festival Code: The Language of our Time looked at the role and influence of code within and upon art and society. There where three themes to this festival, code's relationship to law, code's relationship to art and code's relationship to society. Before 2005 a different perspective about code was emerging.

For software art not to be trapped in a social or cultural content the source code as to be accepted as a visual image in its own right. Transmediale art festival in 2001 introduced the category software art for the first time which was dominated by source code and artists were writing their own software programmes. Software art as process and programme code dominated the Transmediale art until 2004. In 2002 the Whitney Museum of American Art hosted the exhibition CoDeDoc which was curated by Christanne Paul. She invited artist-programmers to produce a piece of work based on a particular instruction. Artist-programmers used Java, Pearl and Lingo to create the work http://artport.whitney.org/commissions/codedoc/index.shtml. In the exhibition article "Public Cultural Production Art" Christanne Paul makes her intentions clear,

> "In software art, the 'materiality' of the written instructions mostly remains hidden. In addition, these instructions and notations can be instantaneously activated; they contain further layers of processing and are the artwork itself. While one might claim that the same holds true for a work of conceptual art that consists of written instructions, this work would still have to be activated as a mental or physical event by the viewer and cannot instantaneously transform, transcend and generate its own materiality" (Paul, 129-130: 2003).

The apparent dualism between software art and generative art is evident in the previous paragraphs. Even Geoff Cox states that Arns is only concerned about process for software art and Galanter is more interested in an end product "But to Inke Arns, (Galanter definition of generative art is) the problem as the definition is far too inclusive, applied across many fields of practice that focus attention on the end-product of a process" (Cox, 3: 2005). Arns' criticism of inclusiveness of Galanter's first definition of generative art is acceptable but she does not recognize that process is crucial for genetic algorithmic codes used in generative art. Galanter in both of his definitions of generative art does refer to process "(Generative art is something) which is set into motion with some degree of

autonomy contributing to or resulting in a completed work of art" (Galanter, 2: 2006) and "(Generative art has) a degree of relative autonomy and contributes to or results in a completed work of art" (Galanter, 4: 2006). In some case the process is continuous or aids to complete an artwork. To this extent process is equally important to software and generative art which unites their art technique. Generative systems can be built on non-generative platforms.

Stanza's *Genomixer* (fig.12) and *Life Forms* (fig.13) uses generic algorithmic codes. Both pieces are created from his dna which scientists extracted from his blood before they were turned into computer codes. *Genomixer* is code represented by code creating a visual DNA of the artist. *Genomixer* has elements of randomness as the codes change and operates semi-autonomously and creates audio visual portraits. Look at the web address www.genomixer.com/genomixer/genomixergensm.htm while *Life Forms* investigated genetic codes as they mapped and reassembled themselves. The work creates a cross reference of all codes on the genome sequence which is mixed into new audio visual self-portraits. Both *Genomixer* and *Life Forms* can be looked at the website addresses www.genomixer.com and www.stanzia.co.uk/muatations/three/mutations3.htm.

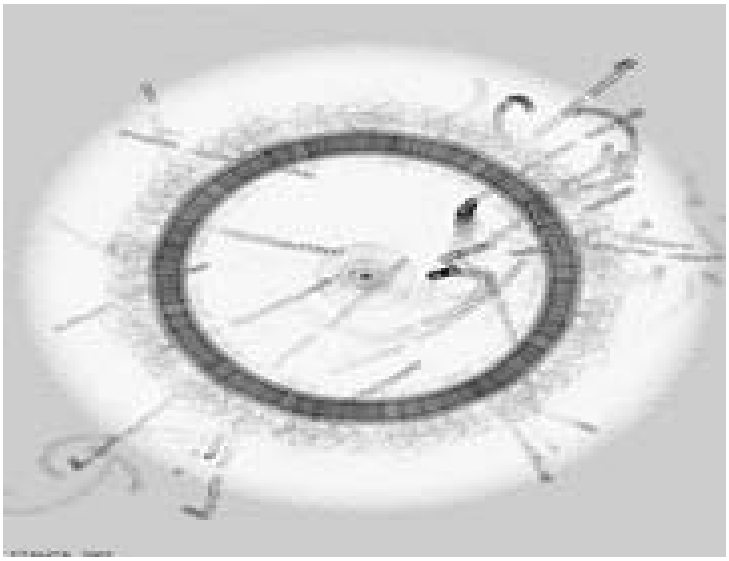


Fig.12. Stanza, Germinator 2009. www.genomixer.com

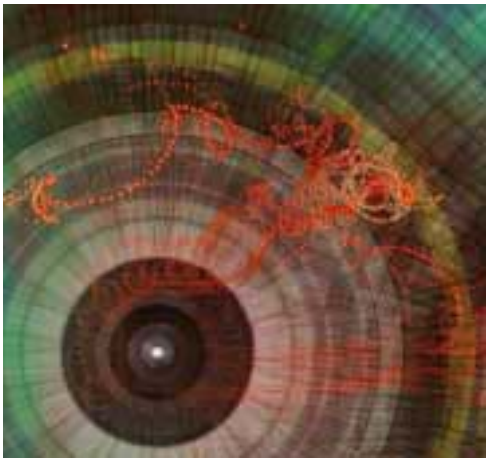


Fig.13. Stanza, Life Forms. 2009.
www.Stanza.co.uk

Stanza is interested in generative codes but refers to his work as software art and John McCormick's definition of software art clearly relates to Stanza's

> "Software art as 'genotype' (DNA cells) as machine code and 'phenotype' (the higher level form of behavior) is what happens when the programmer. …sets the parameters that define the fitness, and the software evolves 'autonomously' (Brown, 5: 2003).

While Mitchell Whitelaw's in his essay "System Stories and Model Worlds: A Critical Approach to Generative Art" argues that the unity between software art and Generative art should combine an emergence and transformative properties that reflect the social complexity and software cultural engagement. This he calls 'critical generativity' (Whitelaw, 152: 2005). Stanza's works are about the invisible human body which is visualized by computer codes.

Philip Galanter also considers how generative art can have a transformative quality. It is in his 2006 essay "What is Generative Art? Complexity as a Context for Art Theory" where he introduces his theory of complexism for generative art. Complexism relates to systems that have a large number of smaller components that interact with small components. These systems are self-organising and react to changes and are called complex adaptive systems. They often develop in a unpredictable way which sometimes appear random. Complex systems allows an understanding of simple systems. According to Galanter, generative artists use randomization in computer codes, while complexity scientists describe complex system as chaos because complex systems are nonlinear and difficult to predict even if the complex systems are machines that follow a strict sequence. Galanter suggest that for the process of generative art to develop it should adopt a system of artificial chaotic systems instead of artificial random systems. From this

perspective, it appears that Galanter is attempting to make the gap between software art and generative art larger. If artists like Stanza can use random code systems to make software art the question is there a difference between generative art and software art? Yet, surprising Galanter becomes interested in an aesthetic formalism when he encounters the aesthetic potential of generative art. In his 2009 essay "What is Emergence? Generative murals as experiments in the philosophy of Complexity" he argues that emergence derives from the field of complexity science and philosophy of science. 'Emergent behaviours' evolves out of complex systems. In genetic complex systems an aesthetic 'formal beauty' becomes apparent and 'Emergence is offered by as a les mysterious, purely mechanistic, explanation of creation" (Galanter, 3: 2009).

From the various discussions about software art and generative art it appears that that artistic experimentation of both relies heavily on process despite the different outcomes of that process. While some people would prefer software art to have a social and cultural context there are other who would prefer to consider it aesthetic qualities. This is the opposite for generative art. Where generative art embraces it social and cultural context its aesthetic qualities are becoming more significant as criteria for evaluation. Yet the 'social-aesthetics' becomes a coherent form aesthetically shifting between social and cultural formations. However a different perspective is need. Online forms of software and generative art will have to consider user participation and new changes in network conditions. There needs to a structure where it is possible to engage with shifting aesthetic criteria and artistic intentionality without succumbing to notions of nostalgia. Computer codes take on a new significance through the extension of technology and aesthetic phenomena does operate within that.

Bibliography.

Arns, I. (2004) "Read_Me, Run_Me, Execute_Me: Software and its Discontents, or it's The Performativity of Code" in Read_Me: Software Art and Cultures in Goriunova, O and Shulgin, A (eds) USA. Arkus: Digital Aesthetics Research Centre.

Baumgartel. T. (2001) "Experimental Software" Germany. Tedopolis. http://www.heisode/tp/deutsch/inhelt.sa/9908/1.html

Cox, G. (2003) "Generator: About Generative Art and/or Software Art" www.generative.net/generator.html

Cramer . F and Gabriel. U. (2001) "Software Art" www.netzlieratur.net and "Software Art" in Broeckmann. A and Jaschko, S (eds.)

(2003) DIY Media-Art and Digital Media: Software-Participation-Distribution. Germany.

Galanter, P. (2003) “What is Generative Art? Complexity Theory as a Context for Art Theory” www.philipgalanter.com/pages/acad/media/gaz3%2proceedings%2opaper.pdf and (2006) What is Complexism?: Generative Art and the Cultures of Science and the Humanities” , “What is Emergence? Generative Murals as Experiments in the Philosophy of Complexity” www.academia/index.html

McCormack, J (2003) “Generative Computation and the Arts” in Digital Creativity. Vol.14. No.1

.

Motte, W. Jr. (ed) (1998) Oulipo: A Primer of Potential Literaure. USA. Dalkey Archive Press.

Paul, C. (2003) “Public Cultural Production Art (Software) in Code-The Language of Our Time by Stocker, G and Schopf, C.

Whitelaw, M. (2005) “System Stories and Model Worlds: A Critical Approach to Generative Art” in Goriunova. Olga, Readme 100: Temporary Software Art Factory. Germany.