Kunst als öffentliche kulturelle Produktion (Software){

Christiane Paul

Seit einigen Jahren wird Software zunehmend als Kunstform betrachtet – eine Sichtweise, die gewissermaßen als logische und unvermeidbare Folge des eigentlichen Wesens des digitalen Mediums und des Einflusses seiner Strukturen und Regeln aufgefasst werden kann. Software ist die treibende Kraft digitaler Medien – ein in ein kommerzielles System integriertes, kulturell und politisch „codiertes" kreatives Mittel.

Software wird allgemein als von einem Computer ausführbares Set formaler Befehle definiert. Es gibt jedoch keine Form der digitalen Kunst, die nicht eine Code- oder Algorithmusebene aufweist, die mittels formaler Befehle in einer bestimmten Anzahl von Schritten ein „Ergebnis" generiert. Selbst wenn die physischen und visuellen Manifestationen digitaler Kunst die Ebene der Datensätze und Codes in den Hintergrund drängen, wird jedes „digitale Bild" letztlich über Befehle und die zur Herstellung oder Manipulation des Bildes verwendete Software erzeugt. Die Ebene des „Codes" und der Befehle schafft einen konzeptuellen Rahmen, der an die Tradition früherer künstlerischer Werke wie Fluxus' oder Dadas Experiment mit formalen Variationen oder die, auf der Ausführung von Befehlen basierten, konzeptuellen Werke von Duchamp, Cage und Sol LeWitt anknüpft.

Es ist allerdings wichtig, Datenkonstrukte wie digitalisierte Bilder oder Texte von Algorithmen zur Steuerung von generativen Prozessen zu unterscheiden. Es muss eine klare Unterscheidung zwischen Kunst, die digitale Technologien als Hilfsmittel für den künstlerischen Schaffungsprozess einsetzt und in einem „traditionellen" Kunstobjekt (Druck, Fotografie, Gemälde, Skulptur) resultiert, und Kunst, die zur Schaffung, Speicherung und Präsentation digitale Technologien als Medium nutzt, getroffen werden. Nur letztere kann potenziell generative Prozesse in Echtzeit aufzeigen. Code sollte auch nicht als von den übergeordneten Strukturen isolierbare Einheit betrachtet werden. Wie Adrian Ward, Alex McLean und Geoff Cox in *The Aesthetics of Generative Code*[1] aufgezeigt haben, ist die schriftliche Form des Codes – als „vom Computer ausgeführte Notation einer internen Struktur, über die Ideen, Logik und Entscheidungen zum Ausdruck gebracht werden" – eine „computerlesbare Notation einer logischen Abfolge" und nicht die vom Computer tatsächlich ausgeführte Operation. Die eigentliche Ausführung der Befehle erfolgt in verschiedenen Ebenen der Interpretation und Kompilierung.

Software als Kunst wird im größeren Umfeld der Generativen Kunst (beispielsweise bei *generative.net*[2] und seit einiger Zeit auch im Rahmen von Festivals wie *Read_me*[3] (dieses Festival widmet sich explizit dieser Kunstform), dem Softwarepreis der *Transmediale*[4] oder dem *runme software art repository*[5], einer im Januar 2003 gegründeten offenen, moderierten Datenbank, diskutiert. Im Einführungstext zur *runme*-Website wird Softwarekunst als Crossover zwischen zwei scheinbar nicht miteinander verbundenen Bereichen – Software und Kunst – definiert: Während Softwarekultur als „lebende Masse" betrachtet wird, die sich primär über das Internet entwickelt und verschiedene kulturelle Sphären in sich vereint und durchdringt, wird Kunst traditionellerweise im Rahmen von Ausstellungen in Galerien und Museen oder auf Festivals präsentiert.[6] Die Fusion von Software und Kunst zur „Softwarekunst" führe daher Softwarekultur in die Kunstwelt ein und bewirke gleichzeitig eine Erweiterung der Kunst über ihre institutionellen Grenzen hinweg.

Wird Software als Kunstform aufgefasst, stellt sich unweigerlich eine Reihe von Fragen: Kann

jede Form von Software als Kunst betrachtet werden und, falls nicht, wo liegt die Grenze zwischen Software als Kunst und Software als „rein" kommerziellem Produkt? Inwieweit wird die Identität der so genannten „Neuen Medien-Kunst" oder der digitalen Kunst durch ihr Wesen als code-basierte Kunst definiert? Worin besteht die Ästhetik von Softwarekunst und wie kann diese von traditionellen, kunstimmanenten Kriterien erfasst werden (bzw. sollte sie mittels derartiger Kriterien erfasst werden)?

Software = Kunst /
(Produkt + (Formalismus / Kulturalismus))

Die *Read_me 1.2*-Jury definierte Softwarekunst sehr weit gefasst als Kunst, die auf aus formalen Befehlen bestehendem Code basiert oder eine kulturelle Reflexion von Software erlaubt – Definitionen, die ein sehr breites Spektrum abdecken. Werden die auf der *runme repository*-Website angeführten Unterkategorien herangezogen, eröffnet sich eine Landschaft, die in ihrer Topografie relativ verwirrend erscheinen mag, allerdings dennoch einige wichtige Unterscheidungen präsentiert und mittels der oben angeführten Definitionen grob zusammengefasst werden kann. Etikettierungen wie algorithmisches Verständnis, Generative Kunst, Code-Poesie, Datentransformation oder digitale Folklore und Kunsthandwerk (z. B. Ascii-Kunst oder Bildschirmschoner) scheinen das Hauptaugenmerk auf die Ästhetik formaler Befehle zu legen. Die Namen, mit denen aktuelle Softwaremanipulationen (Cracks, Patches oder Plug-Ins) oder politische und aktivistische Software (z. B. Cease-and-Desist-Ware und Software-Widerstand) bezeichnet werden, betonen hingegen die Rolle von Softwarekunst als kritische Reflexion des kulturellen Status und der politischen oder wirtschaftlichen Agenda von Software. Spiele, künstlerische Hilfsmittel und konzeptuelle Software können, je nach Ausführung des jeweiligen Projekts und dem Gewicht, das auf formale Aspekte bzw. kritische Reflexion gelegt wird, zu beiden Gruppen gezählt werden.
Es ließe sich schwerlich argumentieren, dass jede Form von Software – von Adobe Photoshop bis Maya – als Kunst betrachtet werden kann. Der „künstlerische Aspekt" zeigt sich in künstlerisch gestalteter Software (Konzept, „Schreibstil", Ergebnisse der Ausführung von Befehlen etc.) oder der Umschreibung / Umformung bestehender Software als Prozess, der die dem originalen Konstrukt inhärente Ästhetik und Agenda untersucht und dadurch zur Diskussion stellt. Diesem Vorgang liegen die Hoffnung und das Versprechen zugrunde, dass Softwareproduktion im größeren Umfeld der kulturellen Produktion akzeptiert wird, oder, wie Pit Schultz es formulierte, „dass das Schreiben von Code einen tieferen Sinn hat, als nur ein Programm zum Laufen zu bringen oder es abstürzen zu lassen oder gute Verkaufszahlen zu erzielen".[7]
Paraphrasiert man die oben angeführte Definition von Softwarekunst – aus formalen Befehlen bestehender Code oder kulturelle Reflexion – als Manifestation von Formalismus vs. Kulturalismus, stellt sich die Frage, ob diese beiden Definitionen zwei entgegengesetzte Pole eines Spektrums vertreten bzw. ob Softwarekunst mittels dieser Kategorien klassifiziert werden kann. In *Concepts, Notations, Software, Art* zeigt Florian Cramer auf, dass ein großer Teil der zeitgenössischen Softwarekunst zwei unterschiedliche Ansätze zu Softwarekunst und Softwarekritik verfolgt: „Software als primär kulturell und politisch codiertes Konstrukt" oder Ansätze, die „die formale Poetik und Ästhetik des Softwarecodes und die in Algorithmen ausgedrückte individuelle Subjektivität" betonen.[8] Die inhärenten Gefahren, die Cramer bei beiden Zugängen sieht, verdienen nähere Beachtung: Eine Reduktion von Softwarekunst auf ersteren Ansatz könnte diese, so Cramer, zu „einer kritischen Fußnote zum Microsoft-Desktop-Publishing" degradieren, die das Potenzial formaler Explorationen nicht erkennt; der zweite Zugang hingegen könnte in „einem neoklassizistischen Verständnis von Softwarekunst als schönem und elegantem Code" resultieren. Der zweite Zugang lässt sich anhand der vom Informatiker Donald

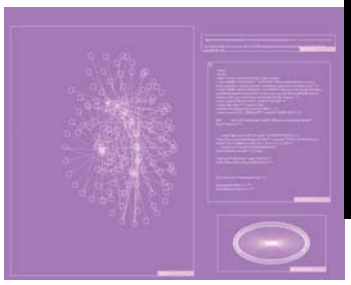

Abb. 1: I/O/D: *Web Stalker*

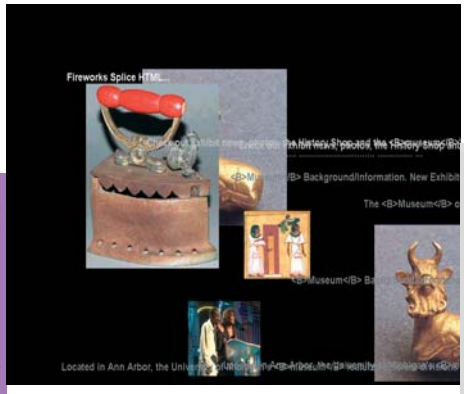

Abb. 2: Maciej Wisniewski: *netomat ™*

```
my $strength = $ARGV[0] + 1;

while (not fork) {
  exit unless --$strength;
  print 0;
  twist: while (fork) {
    exit unless --$strength;
    print 1;
  }
}
goto 'twist' if --$strength;
```

Abb. 3:
Alex McLean:
forkbomb.pl

Kuhn erstellten Bewertungskriterien veranschaulichen, der bereits seit den 1970er-Jahren von „Programmieren als Kunst" spricht:[9] Zu diesen Kriterien zählen Korrektheit, Wartungsfreundlichkeit, Klarheit, Möglichkeiten der Benutzerinteraktion und Lesbarkeit – was den *International Obfuscated C Code Contest*[10] als Fehlschlag der Programmierkunst klassifiziert. Cramer verbindet die „Software als kulturelles Konstrukt"-Schule mit Programmierkünstlern wie Matthew Fuller und der I/O/D-Gruppe oder Graham Harwood und der Gruppe Mongrel. I/O/D „schufen" mit ihrem *Web Stalker*[11] [Abb. 1] im Alleingang alternative Browser als Kunstform. Der Benutzer zeichnet „Rahmen" in einem leeren Fenster und wählt jene Informationen aus, die in diesen Rahmen angezeigt werden sollen – beispielsweise eine grafische Karte der Website mit allen Einzelseiten und Links, Text aus einer URL und den Quellcode der HTML-Seite, eine Liste von URLs, die gespeichert werden sollen. Obwohl der *WebStalker* keine Grafiken anzeigt, erweiterte er die Funktionalität bestehender Browser auf eine Art und Weise, die die Paradigmen der konventionellen Informationspräsentation und Internet-„Architektur" in Frage stellt. Maciej Wisniewskis *netomat ™* [12] [Abb. 2], der das Seitenformat traditioneller Browser aufgibt und das Internet als einzige große Datenbank unzähliger Einzeldateien betrachtet, schlägt zwar einen anderen Weg ein, würde jedoch zur selben Kategorie gezählt werden. Mittels einer audiovisuellen Sprache, die speziell zur Erforschung des bislang unerforschten Internet entwickelt wurde, demonstriert *netomat ™,* wie das kontinuierlich wachsende Web kulturelle Konzepte und Themen interpretiert und re-interpretiert und Benutzern Zugang zum „Unterbewusstsein" des Internets verschafft.

Cramer zufolge zeigt sich der Zugang der Formalisten in den Arbeiten der Gruppen um Adrian Ward und Alex McLean und den Abonnenten der Mailingliste „eu-gene". Adrian Wards *Auto-Illustrator*[13] (Gewinner des Transmediale.01-Preises für Softwarekunst) ist eine Grafikdesign-Anwendung, die Benutzern das Ausprobieren einer Vielzahl von Prozeduren zur Erstellung ihres persönlichen Grafikdesigns ermöglicht. Bei Alex McLeans *forkbomb.pl*[14] (Gewinner des Transmediale.02-Preises für Softwarekunst) [Abb. 3] handelt es sich hingegen um ein Perl-Script, das ein künstlerisches Abbild des Computersystems unter Belastung schafft (indem wiederholt neue Prozesse in einer solchen Geschwindigkeit gestartet werden, dass es letztendlich zum Systemstillstand kommt).

Selbst wenn die Grundausrichtung der oben erwähnten Projekte (bzw. der Absicht ihrer „Erfinder") sich einem Ende der Formalismus/Kulturalismus-Skala zuneigt, dürfen die subtileren Anzeichen für eine Bewegung in die Gegenrichtung, die sich in jedem dieser Projekte zeigen, nicht ignoriert werden. Der *Web Stalker* mag Browser als kulturell codierte Konstrukte betrachten, kann jedoch nicht seine ausgeprägte Ästhetik und die klaren kunsthistorischen Referenzen verleugnen. In seinem Essay *Visceral Facades: taking Matta-Clark's crowbar to software*[15] schafft Matthew Fuller von der I/O/D-Gruppe eine Verbindung zwischen dem Verständnis des *WebStalker* für die Informationsarchitektur und der Technik des amerikanischen Künstlers Gordon

Matta-Clark der buchstäblichen „Aufsplitterung“ der Architektur von Gebäuden, einer aus formalen Prozeduren bestehenden Anwendung zur Identifikation der strukturellen Eigenschaften von Gebäuden. Der „Dekonstruktivismus“ und die „Anarchitektur“, die in Matta-Clarks Arbeiten bzw. im *WebStalker* zu finden sind, sind ebenso Stellungnahmen gegen bestimmte soziale Bedingungen, wie sie auch ästhetische Handlungen zwischen der Rekonstruktion des Zerstörten und der Zerstörung der Abgeschlossenheit sind. Wards *Auto-Illustrator* ist vielleicht eine Anwendung zur Untersuchung der Schönheit und Eleganz eines grafischen Designs, gleichzeitig jedoch auch eine Stellungnahme zu den Konventionen und der Standardisierung kommerzieller Grafikdesign-Anwendungen. Bei genauerer Analyse des Oeuvres eines Künstlers ist es oft schwer, diesen einem bestimmten Lager zuzuordnen. Mark Napiers *FEED*[16] [Abb. 4], das Webseiten in einen Pixelstrom dekonstruiert, die auf neun unterschiedlichen Anzeigen grafisch dargestellt und geplottet werden, kann gleichsam als Automatisierung ästhetischer Strukturen vom Abstrakten Expressionismus hin zum Minimalismus betrachtet werden. Napiers *Riot*[17] ist hingegen ein alternativer Browser, der Text, Bilder und Links aus den drei letzten URLs, die *Riot*-Benutzer weltweit in einem Browserfenster aufgerufen haben, vermischt und territoriale Konventionen wie Domains, Sites und Webseiten zusammenbrechen lässt. Indem aufgezeigt wird, wie das Internet traditionellen Vorstellungen von Territorialität, Besitztum und Autorität widersteht, wird die Politik der Verschlüsselung von Informationen hinterfragt. Wenn Software im Allgemeinen nicht neutral, sondern kulturell kodiert ist, existiert ein beständiges Zusammenspiel von formalen und kulturellen Aspekten, das je nach Schwerpunkt eines spezifischen Projekts gewisse Variationen aufweist.

Ästhetik der Wahrnehmung/Poetik der Konstruktion

Die Kategorisierung von künstlerischer Software scheint je nach konkreter Manifestation des Kunstwerks permanenten Veränderungen ausgesetzt zu sein. Wie bereits erwähnt, umfasst jedes digitale Kunstwerk eine Ebene des Codes. Es scheint wichtig anzumerken, dass digitale Kunstinstallationen – selbst wenn sie von künstlerischer Software gesteuert sind – nur selten als Softwarekunst betrachtet werden. Obwohl die Bewegungen und Reaktionen von Robotern und Objekten (bzw. die von Sensoren ausgelösten Reaktionen) von künstlerischer Software gesteuert oder verarbeitet werden, wird im Allgemeinen den konzeptuellen Aspekten, den kulturellen Auswirkungen oder der „Eleganz“ der Software selbst wenig Aufmerksamkeit zuteil; die Software bleibt eine versteckte Kraft im Hintergrund und generiert häufig

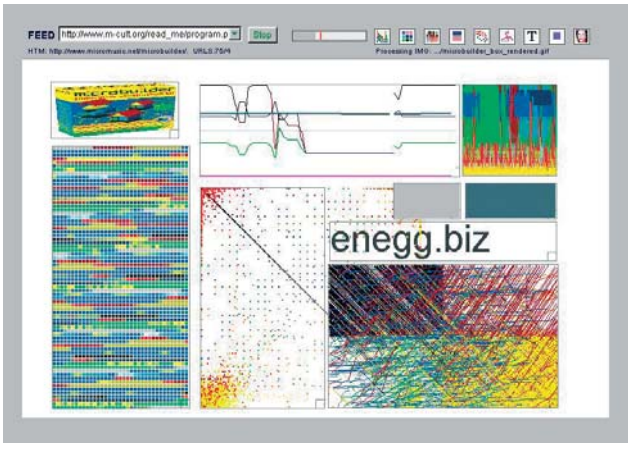

Abb. 4: Mark Napier: *FEED*

derart komplexe Interaktionen, dass ihr „Schreibprozess“ schwerer zugänglich ist als ein Kunstwerk der Code-Poesie. Ein wesentliches Element bei der Identitätsfindung digitaler Kunst ist das Verständnis der grundlegenden Natur und Sprache von digitaler Kunst und ihres Ursprungs in Algorithmen und Programmcode.

Was heute allgemein als „Softwarekunst“ akzeptiert wird, unterscheidet sich stark in der Ausrichtung und Manifestation dieser Kunstform. Werke der Softwarekunst können sich in vielfältiger Gestalt präsentieren, von visuellen Manifestationen (gesteuert von einer großteils verdeckten Ebene des künstlerischen Codes) bis hin zum schriftlichen Code selbst. Code-Poesie wie etwa Graham Harwoods *London.pl* (von William Blake)[18] wäre ein Beispiel für letztere Kategorie. Programmieren als Kunstform und künstlerische Ausdrucksweise bleibt weiterhin stark unterbewertet und unterschätzt. Florian Cramer zufolge ist der Fokus auf einer rein perzeptuellen Ästhetik der Kunst „[…] eine klare Fortsetzung der romantischen Philosophie und der Privilegierung der Aisthesis (Wahrnehmung) gegenüber der Poeisis (Konstruktion), herabgewürdigt zu einem eingeschränktem Verständnis von Kunst als das, was angreifbar, hörbar und sichtbar ist“.[19]

Was Softwarekunst von anderen Kunstpraktiken unterscheidet, ist die Tatsache, dass der Künstler – anders als bei allen Formen der visuellen Kunst – eine rein verbale Beschreibung seiner Arbeit liefern muss (die dann oft hinter den daraus resultierenden Prozessen verborgen bleibt). In den meisten traditionellen Kunstformen manifestiert sich die „Unterschrift“ und „Stimme“ eines Künstlers in der Ästhetik visueller Eindrücke und der Ausführung. Die Ästhetik von Künstlern, die ihr eigenes Quellprogramm schreiben, zeigt sich sowohl in der Poesie des Codes als auch der daraus resultierenden visuellen Ergebnisse. Der Künstler John F. Simon Jr. hat den Code des Öfteren als Form des kreativen Schreibens bezeichnet, bei der von der Auswahl der „Geschichte“ bis zur Sprache der Narration und Handlung alle Einzelschritte die „Stimme“ des Künstlers zum Ausdruck bringen. Obwohl Cramer der Ansicht ist, dass Ward, McLean und Cox primär die Ausführung des Programms forcieren, betonen sie dennoch, dass eine Trennung des Codes von den daraus resultierenden Aktionen zu einer Einschränkung der ästhetischen Erfahrung[20] führen würde und beide Seiten des Spektrums Beachtung finden müssten.

Für die Untersuchung und Kritik von Softwarekunst ist es nötig, sowohl über die Ästhetik der Konstruktion von nachgeschalteten Rechnern als auch die (multisensorische) Wahrnehmung auf Seiten der Benutzerrechner Bescheid zu wissen. Das folgenschwere Dilemma von Softwarekunst könnte durchaus sein, dass die Untersuchung der „Nachrechnerseite“ stets ein Randbereich bleiben wird, der nicht in den Mainstream der wahrnehmungsorientierten Kunstkritik integriert wird. Der Zusammenhang zwischen dem schriftlichen Code und den daraus resultierenden Prozessen wirft auch die Frage auf, wie transparent die Beziehung zwischen diesen zwei Aspekten sein kann oder sollte bzw. ob dies ein Kriterium für die Bewertung von Kunst sein kann. Ist Softwarekunst erfolgreicher, wenn man die Algorithmen, die hinter der Entfaltung von visuellen/klanglichen Phänomen stehen, „sehen“ kann und direkte Verbindungen zwischen der Benutzerseite und dem dahinter liegenden Code herstellen kann? Kunst, die die Herstellung dieser Verbindung erlaubt, wird sicher einem größerem Publikum zugänglich sein. Es bleibt jedoch fraglich, ob die Transparenz von Ursache-Wirkung-Beziehungen ein geeignetes Kriterium für die Qualität von Kunst darstellt. Es scheint sich hier um eine Frage der Referenz zu handeln, die in eine weiter gefasste Diskussion hinsichtlich des Status der Repräsentation von digitaler Kunst eingebettet ist. Wird Repräsentation als „Ähnlichkeit“ oder Abbild eines externen Referenten (als „Objekt“ oder „Rahmen“ im weitesten Sinn) definiert, repräsentiert digitale Kunst letztlich stets Daten – seien es externe Daten oder eine selbst erzeugte Datenquelle. Während dies auf digitale Kunst im Allgemeinen zutrifft, konzentriert sich Softwarekunst stärker auf den generativen Konstruktionsprozess der Datenrepräsentation. Es liegt daher die Schlussfolgerung nahe,

Abb. 5: Jodi: *Untitled Game*

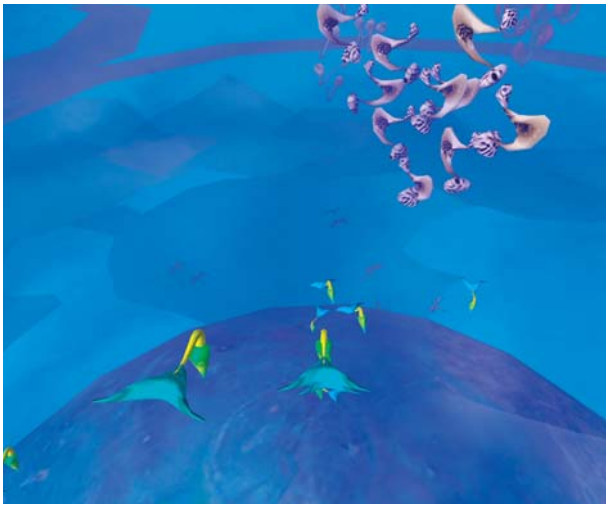

Abb. 6: John Klima: *Ecosystem*

dass wir eine umfassende Transformation des Status der Repräsentation selbst erleben, die zu einem Prozess der Konvergenz von Sprache und Mathematik wird, was umgekehrt wiederum das Potenzial zur Steuerung eines multisensorischen „Displays" birgt.
Diese Transformation der Repräsentation impliziert auch, dass Softwarekunst in stärkerem Maße kontextabhängig ist, da ihre Datenquelle stets in einen spezifischen Kontext eingebettet ist. Bei Softwarekunst, die primär auf Datenvisualisierung abzielt – die Schaffung von visuellen Modellen für bestimmte Datensätze –, ist eine geeignete Kontextualisierung besonders wichtig. Jeder Datensatz kann auf verschiedene Arten visualisiert werden; diese Visualisierungsmöglichkeiten eignen sich wiederum für eine Neukonfiguration. Dadurch verändert sich der Kontext: Der Kontext zur Erschließung des Sinns eines gewissen Datensatzes ist zu einem großen Teil von der Dynamik des Interface bestimmt. In diesem Fall kann ein externer Referent, ein Datensatz, Teil seiner eigenen Repräsentation werden. Auch für den Erfolg des Visualisierungsprozesses per se ist der Kontext entscheidend. Während Veränderungen der Datensätze stets beachtet und entsprechend reflektiert werden müssen, verblasst das visuelle Modell häufig zu einer Art von „Tapete" (wie schön diese auch immer sein mag), wenn der größere Kontext der Datensätze außer Acht gelassen wird.
Dieser Prozess wird zusätzlich dadurch erschwert, dass Softwarekunst weder in einem kunsthistorischen und kulturellen Vakuum existiert noch ein in sich klar abgrenzbares Gebiet darstellt. Das Label „Softwarekunst" kann lediglich ein charakteristisches Merkmal eines Kunstwerks sein, das in verschiedene Kontexte eingebettet ist. John F. Simon Jr.' Farbbildserie (bestehend aus eigens angepasster Hard- und Software) weist beispielsweise viele kunsthistorische Referenzen auf. *Color Panel v 1.0* [21] ist eine zeitbasierte Farbstudie, in der Simons Software die Möglichkeiten und „Regeln" von Farben untersucht, wie dies von Bauhaus-Künstlern wie Klee und Kandinsky vorgeschlagen wurde – eine Untersuchung der Farbtheorie in Bezug auf Bewegung und Zeit. Es wäre problematisch, das Projekt als eine Fortsetzung der Bauhaus-Tradition aufzufassen, ohne entsprechendes Augenmerk auf die nachfolgenden aus dem Code resultierenden Aktivitäten zu legen, die über einzelne Momente in Zeit und Raum hinausgehen und zu kontinuierlichen Kontextveränderungen führen.
Es existiert eine breite Palette von Softwarekunst, die Bezug auf Spiele oder die Spielkultur nimmt – entweder durch die Programmierung von neuen Spielen oder das Umschreiben und

Umformen bestehender Spiele – und als Zusammenspiel zwischen dem kontextuellen Rahmen (Spiele), dem Code und den daraus folgenden Handlungen betrachtet werden muss. Jodis „Dekonstruktion" des originalen Wolfenstein-Spiels in *SOD*[22] oder die Dekonstruktion von *Quake I* in *Untitled Game*[23] [Abb. 5] kann ohne grundlegende Kenntnisse über Videospiele (Ästhetik, Architektur, Benutzer/System-Interaktion etc.) nicht „gelesen" werden. *SOD* ersetzt die Repräsentationselemente des Originalspiels durch schwarze, weiße und graue geometrische Formen und schafft eine neue Architektur, die sowohl Orientierung als auch Navigation zu einer Herausforderung werden lässt. *Untitled Game* löst die Originalstruktur von *Quake* auf, programmiert seine Struktur und Interaktivität neu und nutzt die originale Game-Engine als Mittel zur Schaffung abstrakter Kunst. Cory Arcangel schuf mehrere Arbeiten (darunter *I Shot Andy Warhol* und *Landscape Study #4* [24]), die auf rückentwickelten Cartridges des Nintendo-Spiels *Super Mario Brothers* basieren; die Originalchips werden von einer *Super Mario*-Cartridge ausgelötet und durch selbst erzeugte Chips ersetzt. Man könnte annehmen, dass die Akzeptanz dieser Werke sich zumindest bis zu einem gewissen Grad auf die „Retro"-Ästhetik ihres Originalkontextes stützt. *Landscape Study #4* verbindet traditionelle Landschaftsfotografie mit Spielästhetik und schafft eine Szenerie, die sehr wirksam jene Medien, aus denen Elemente übernommen werden, durchdringt und sich zu einer neuen Form des Pop-Art-Genres zu entwickeln scheint. Die Softwarearbeiten von John Klima – darunter *glasbead, Go, Fish* und *ecosystm*[25] [Abb. 6], die globale Währungsdaten in einer 3D-Simulation präsentieren, in der die Population und das Verhalten von insektenartigen „Vogel"-Schwärmen (diese symbolisieren die „Währungen" der einzelnen Länder) vom Wert ihrer jeweiligen Währung im Vergleich zum Dollar bzw. der täglichen/jährlichen Volatilität ihrer Währung beeinflusst wird – sind alle stark von den Paradigmen der Spielszene und der Spielästhetik geprägt. Während die oben erwähnten Künstler alle im Bereich der Softwarekunst bzw. im Spielbereich tätig zu sein scheinen, unterscheiden sich doch ihre Zugänge zu Programmierung und Ästhetik; das Oeuvre dieser verschiedenen Künstler erfordert einen unterschiedlichen kontextuellen Rahmen.

Das Interesse an der Entwicklung von Kriterien zur Untersuchung und Kritik von Softwarekunst nimmt stetig zu, doch stellt sich weiterhin die Frage, welche Auswirkungen diese Kunstform sowohl auf den Bereich der zeitgenössischen Kunst als auch die Gesamtkultur haben wird. Es besteht die Hoffnung, dass ein wachsendes Bewusstsein hinsichtlich der kunsthistorischen "Verwandtschaft" von Konzeptkunst und Softwarekunst zu einer größeren Akzeptanz der Medienkunst im Umfeld der zeitgenössischen Kunst führt. Wie Matthew Fuller aufzeigt, beschäftigt die zeitgenössische Kunst sich bereits mit Netzwerken und Computertechnologie, indem wesentliche Merkmale dieser beiden Bereiche, wie z. B. eine „relationale Ästhetik", erforscht werden; selten jedoch erstreckt sich dieses Interesse auf spezifische digitale Technologien.[26] Es stellt sich nicht nur die Frage, was Softwarekunst für die Kunstwelt und ihre Institutionen bringen könnte, sondern wie diese Institutionen Softwarekünstler unterstützen könnten. Die Förderung von Software- und Programmierkenntnissen ist überaus wichtig, wenn die Rolle von Software im Rahmen der kulturellen Produktion erweitert werden soll. Initiativen wie beispielsweise *Processing,* ein offenes Projekt von Ben Fry und Casey Reas (siehe Seite 210),[27] das eine geeignete Umgebung zum Erwerb von grundlegenden Kenntnissen des Programmierens schafft und als elektronisches Skizzenbuch zur Entwicklung von Ideen gedacht ist, sind ein erster Schritt in diese Richtung. In der gegenwärtigen Situation muss allerdings die Anerkennung von Software als Kunstform und kulturelles Ausdrucksmittel stärker als bisher forciert werden, um eine Ebene der Akzeptanz zu erreichen, auf der Software als mehr als nur ein Produkt aus dem Regal, das primär an seiner Effizienz gemessen wird, betrachtet wird.

}

Aus dem Amerikanischen von Sonja Pöllabauer

1 Ward, Adrian, McLean, Alex und Cox, Geoff: *The Aesthetics of Generative Code, http://generative.net/papers/aesthetics/*
2 *http://www.generative.net/*
„Generative Kunst dient als Bezeichnung für Arbeiten, die sich mit den Prozessen zur Schaffung eines Kunstwerks beschäftigen, die üblicherweise (wenn auch nicht ausschließlich) durch Einsatz einer Maschine oder eines Computers oder durch mathematische oder pragmatische Befehle zur Definition der Regeln, mittels derer diese Kunstwerke ausgeführt werden, automatisiert werden." (Adrian Ward)
„Generative Kunst umfasst jede Form von Kunst, bei der der Künstler einen Prozess auslöst, wie beispielsweise ein Set von natürlichen Sprachregeln, ein Computerprogramm, eine Maschine oder andere prozedurale Phänomene, der dann mit einem gewissen Grad von Autonomie in Bewegung gesetzt wird und zu einem vollendeten Kunstwerk beiträgt oder in einem solchen resultiert." (Philip Galanter)
3 *http://www.m-cult.org/read_me/*
4 *http://www.transmediale.de*
5 *http://www.runme.org;* entwickelt von Amy Alexander, Florian Cramer, Matthew Fuller, Olga Goriunova, Thomax Kaulmann, Alex McLean, Pit Schultz, Alexei Shulgin und The Yes Men
6 *http://www.runme.org/about.tt2*
7 *QuickView on Software Art, runme.org/project/+quickview*
8 Cramer, Florian: *Concepts, Notations, Software, Art,* 2002, *http://userpage.fu-berlin.de/~cantsin/homepage/writings/software_art/concept_notations//concepts_notations_software_art.html*
9 Knutz, Donald E.: „Literate Programming", in *CSLI Lecture Notes,* Number 27, Center for the Study of Language and Information, Stanford, CA 1992
10 *http://www.ioccc.org/*
11 I/O/D: *Web Stalker, http://www.backspace.org/iod/*
12 Wisniewski, Maciej: *netomat™, http://www.netomat.net*
13 Ward, Adrian: *Auto-Illustrator, http://www.auto-illustrator.com*
14 McLean, Alex: *forkbomb.pl, http://www.slab.org*
15 Fuller, Matthew: *Visceral Facades: taking Matta-Clark's crowbar to software, http://www.backspace.org/iod/Visceral.html*
16 Napier, Mark: *FEED,http://www.potatoland.org/feed/*
17 Napier, Mark: *Riot, http://www.potatoland.org/riot/*
18 Harwood, Graham: *London.pl* von William Blake, *http://www.runme.org/project/+londonpl/*
19 Ibid. [8]
20 Ibid. [1]
21 Simon, John F., Jr.: *Color Panel v 1.0, http://www.numeral.com/panels/colorpanelv1.0.html*
22 Jodi: *SOD, http://sod.jodi.org/*
23 Jodi: *Untitled Game, www.untitled-game.org*
24 Arcangel, Cory: *I Shot Andy Warhol,* Landscape Studies, *http://beigerecords.com/cory*
25 *http://www.cityarts.com*
26 Ibid. [7]
27 *http://proce55ing.net/*