Humboldt-Universität zu Berlin
Institut für Informatik
Informatik in Bildung und Gesellschaft

Ästhetik & Software

Oliver Stadie[1] Stephan Otto[2]

August 21, 2008

[1] o.stadie@googlemail.com
[2] otto.stephan@berlin.de

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Der vorliegende Text entstand im Sommersemester 2008 in Gruppenarbeit als Ausarbeitung zu einem am 22. Mai 2008 im Seminar „Dimensionen der Software-Entwicklung“ an der Humboldt-Universität zu Berlin gehaltenen Vortrag.[1] Unsere Betrachtungen zur ästhetischen Dimension der Software-Entwicklung speisen sich einerseits aus der Sichtung von Literatur und ähnlicher Quellen zum Thema und andererseits aus der telefonischen Befragung von vier Software-Entwicklern. Für letztere Aktivität wurde von uns ein Fragenkatalog erstellt.

Die vier Software-Entwickler sollen unter Wahrung von Anonymität nun kurz vorgestellt werden:

- Entwickler A ist diplomierter Informatiker und entwickelt seit mehr als 10 Jahren im SAP-Umfeld sowie in reinen JAVA-Entwicklungsprojekten und ist seit einigen Jahren “Themen-Manager” in der Entwicklungsabteilung einer international agierenden Unternehmensberatung.
- Entwickler B studierte Landschaftsarchitektur und wurde im Rahmen einer Qualifizierungsmaßnahme zum Software-Entwickler umgeschult. Er arbeitet seit etwa 5 Jahren ausschließlich in SAP-Entwicklungsprojekten und ist in der Entwicklungsabteilung einer international agierenden Unternehmensberatung angestellt.
- Entwickler C ist diplomierter Bergbauingenieur und wurde vor etwa 15 Jahren im Rahmen einer Qualifizierungsmaßnahme zum Software-Entwickler umgeschult. Seither arbeitet er ausschließlich in SAP-Entwicklungsprojekten im Bereich Product Lifecycle Management und ist in der Entwicklungsabteilung einer international agierenden Unternehmensberatung angestellt.
- Entwickler D ist diplomierter Physiker und entwickelte vor seinem Wechsel zu einer Spieleentwicklungsfirma unter anderem für Banken und den Musiksender MTV Web-Präsenzen. Seit etwa 10 Jahren ist er nun bei einer Spieleentwicklungsfirma, die vor allem internetbasierte Spiele entwickelt, angestellt. Dort arbeitete er zunächst an Projekten mit Web-Technologien und unter Einsatz von JAVA mit. Seit einigen Jahren übt er auch die Funktion eines Projektleiters aus.

Diese Entwickler wurden mit einem von uns vorbereiteten Interview konfrontiert, dessen Resultat überall in dieser Arbeit Verwendung fand; der jeweilige Befragte ist dann jeweils mit seinem Buchstaben (A bis D) in runden Klammern gekennzeichnet. Des weiteren schöpften wir auch Erkenntnisse aus unseren eigenen Erfahrungen, die wir in Studium, Beruf und Freizeitaktivitäten sammeln konnten.

2 Zum Begriff der Ästhetik

Der Begriff der Ästhetik entzieht sich einer eindeutigen Definition. In den durchgeführten Befragungen wurden recht unterschiedliche Dinge unter „Ästhetik“ verstanden.

[1] Vgl. [Koubek 2008] zum Begriff „Dimension“.

2.1 Ästhetik als Ergebnis von Sozialisation

Unterschiedliche Auffassungen der Befragten zum Begriff der Ästhetik können unter anderem in ihren unterschiedlichen Sozialisationen begründet sein. Sozialisation verstehen wir als „Prozess der Persönlichkeitsentwicklung in wechselseitiger Abhängigkeit von den körperlichen und psychischen Grundstrukturen [des Individuums] und den sozialen und physikalischen Umweltbedingungen“.[2] Gemäß der kanonischen Einteilung der Sozialisationsinstanzen werden primäre, sekundäre und tertiäre Sozialisationsinstanzen unterschieden. Die primären Sozialisationsinstanzen sind zum Beispiel Familie und Freunde; diesen kam gemäß den Antworten auf unseren Fragenkatalog aus Sicht aller Befragten eine wichtige Rolle bei der Ausprägung des individuellen Verständnisses von Ästhetik zu. In zwei Fällen wurden vor allem die Eltern genannt, die als Künstler (B) beziehungsweise Architekten (C) großen Einfluss ausübten. Daneben wurde von einem weiteren Befragten (A) die Ehefrau als Trägerin eines sehr starken Einflusses genannt. Schließlich antwortete ein weiterer Befragter (D), er sei von allen Seiten sehr stark geprägt worden, könne aber lediglich die Ergebnisse in Form seines Kleidungsstils oder Musikgeschmacks anführen. Welche Personen aus seinem Umfeld in welcher Weise dazu beigetragen haben, könne er nicht ermitteln.[3]

Schule und Universität, die zu den sekundären Sozialisationsinstanzen zählen, wurden hingegen von zwei Befragten (A und D) als gänzlich ohne Einfluss bezeichnet. Die beiden anderen Befragten machten hingegen einen großen Einfluss geltend, vor allem für das Studium der Landschaftsarchitektur (B), welches für Kompositionen, Formen und Farben sensibilisiert habe, bzw. das Grundlagenstudium der Ingenieurwissenschaften (C), welches Geometrie und Symmetrie näher gebracht habe. Die Tatsache, dass der Schule durch die Befragten fast jeglicher Beitrag zur Entwicklung der eigenen Auffassung von Ästhetik abgesprochen wurde, könnte bei diesen auf den verhältnismäßig langen Zeitraum seit dem Austritt aus der Schule zurückzuführen sein.

Es stellt sich die Frage, wieso eine Institution, die Schüler mit Kunst- und Musikunterricht versorgt und den Versuch unternimmt, Kontakt zur Literatur herzustellen, bei einer verhältnismäßig langen Verweildauer der Schüler in ihr – nicht zuletzt angesichts der immer mal wieder geführten „Kanon-Diskussionen“ – keinen Einfluss auf die Auffassung von Ästhetik beanspruchen können soll.[4]

2.2 Ästhetik als Teil der Wissenschaften

Auch im Wissenschaftsbetrieb wird mit verschiedenen Ästhetik-Begriffen operiert. Spätestens seit der Antike wird Ästhetik als Theorie der Kunst ausgeübt. Verlangte die klassische Definition von Kunst, dass diese die Natur – im Sinne von Mimesis – nachahmen solle, so ist dies in modernen Definitionen nicht mehr der Fall. Kunst wird dann

[2] Quelle: [Hurrelmann 2002], S. 26.

[3] Vgl. etwa für die deutschsprachige Literatur [Stifter 2005], wo auf mehr als 700 Seiten gerade die Bildung von ästhetischen Auffassungen und der Einfluss der Sozialisation literarisch dargereicht sind – selbstverständlich in der Tradition Jean-Jacques Rousseaus.

[4] Zu “Kanon-Diskussionen” vgl. etwa [Buß 2003], S. 145 ff.

etwa als „eine Weise, die Welt verstehend zu erschließen, uns in ihr zurechtzufinden“ bezeichnet.[5]

Ein weiterer Ansatz ist es, Ästhetik interdisziplinär zu betreiben. Der Versuch etwa, Erkenntnisse der Ästhetik als Teil der Philosophie mit solchen der Psychologie zu verknüpfen, könnte zur Formulierung einer Metatheorie führen – und damit zu einer gänzlich neuen Grundlage der Ästhetik.[6] Ein solches Vorgehen wäre ohne das Aufstreben von Psychophysik und Psychologie seit der zweiten Hälfte des 19. Jahrhunderts kaum vorstellbar, wenn man Schönheit z. B. anhand der Gehirnaktivität von Menschen messen möchte.[7] So glauben Forscher sowohl Hirnregionen die für die objektive, als auch die Regionen für die subjektive Ästhetik, gefunden zu haben.[8] Von letzterer soll auch der folgende Abschnitt 2.3 handeln. Im Zuge dieser Forschung hat sich ein neues Teilgebiet der Wissenschaft herrauskristalisiert – die sogenannte “Neuroästhetik”. Diese Wissenschaft steckt zum Zeitpunkt dieser Ausarbeitung jedoch noch so weit in den Kinderschuhen, dass dazu zwar schon erste eigenständige Kurse an Universitäten angeboten werden, jedoch noch keinerlei wissenschaftliche Veröffentlichungen zu finden sind.[9]

Noch umfassender scheint allerdings der Versuch, ästhetische Begriffe in einer Systemtheorie des Menschen abzubilden.[10]

Janker untersucht des weiteren, warum Maschinen in der Wissenschaft der Künstlichen Intelligenz (kurz: KI) bisher nicht in der Lage sind, Ästhetik zu empfinden.[11] Sie führt es vor allem auf die fehlenden Gefühle (Gefühle entstünden nicht durch bloße Berechnungen) von Maschinen zurück, da ästhetisches Empfinden direkt aus Gefühlsregungen resultiere (siehe hierzu auch 2.4 auf der nächsten Seite). Eine ähnliche Grenze sieht auch Warnke, der die Theoretische Informatik zu Rate zieht.[12] Er begründet, dass die Turing-Berechenbarkeit eine Grenze darstelle, welche menschliche Kreativprozesse von den maschinellen scheidet (vgl. auch hierfür 2.4).

2.3 Ästhetik als individuelle Kategorie

Für die Befragung der einzelnen Software-Entwickler nutzen wir eine „naive“ Definition von Ästhetik. Da die Ansichten, was unter Ästhetik zu verstehen sei, bei den Befragten stark divergierten, verstanden wir während der Befragung unter Ästhetik jeweils das, was der Befragte darunter verstand.

Zur Beantwortung der Frage, was er unter Ästhetik verstünde, erklärte ein Befragter (A), diese sei gegeben, wenn etwas „gut aussieht“ oder „sich gut anfühlt“ bzw. „die Sinne anspricht“. Ein anderer (B) sah für „Sachen, die Wohlbefinden erzeugen“ – vornehm-

[5] Quelle: [Gethmann-Siefert 1995], S. 120.
[6] Vgl. [Janker 2002].
[7] Vgl. etwa [Kittler 1995], S.266.
[8] Quelle: [Di Dio 2008].
[9] Das Universitätsklinikum Jena bietet diesen Kurs beispielsweise erstmalig im Sommersemester ’08 an. (Siehe auch `http://www.association-of-neuroesthetics.org/` für aktuellere Informationen zu diesem Thema.)
[10] Vgl. [Janker 2002], S. 23, die ihren Doktorvater Dietrich Dörner für genau diesen Versuch benennt.
[11] Quelle: [Janker 2002, Abschnitt 2.1].
[12] Vgl. [Warnke 2005].

lich auf visuell Wahrnehmbares bezogen – Ästhetik als gegeben an. Ähnlich antwortete ein weiterer Befragter (D), hier ginge es um „Sachen, die man spontan als angenehm empfindet“, was gegenüber dem Zweitgenannten wiederum nicht auf lediglich einen Sinn rekurriert. Schließlich sah einer der vier Befragten (C) Ästhetik als die „Lehre von der Schönheit der Dinge“ an. Stellt man die Antworten auf die Frage nach dem eigenen Verständnis von Ästhetik mit denen nach dem Einfluss der Sozialisationsinstanzen Schule und Hochschule beziehungsweise Eltern, Verwandte, Freunde und Kollegen gegenüber, so scheint es, als wurde die erste Frage umso konkreter beantwortet, je expliziter die Befragten den Einfluss der Sozialisationsinstanzen bejahten und schilderten. Die beiden Befragten, die für Schule und Hochschule jeglichen Einfluss verneinten (A und D), jedoch einen sehr starken Einfluss ihres sozialen Umfeldes als gegeben ansahen, machten ein Verständnis von Ästhetik geltend, dass eben „was gut aussieht“ oder „sich gut anfühlt“ (A) bzw. „Sachen, die man spontan als angenehm empfindet“ (D) enthält. Der Befragte (C), der hingegen Ästhetik als die „Lehre von der Schönheit“ ansah, gab an, dass zu seinem Verständnis von Ästhetik sowohl die (Hoch-) Schule – vor allem im Grundlagenstudium der Ingenieurwissenschaften – als auch sein soziales Umfeld – etwa seinen als Architekt tätigen Vater – „mit Sicherheit“ beigetragen habe.

Dies aufnehmend, stellt sich die Frage, was schön ist, wenn man Ästhetik als „Lehre von der Schönheit der Dinge“ betrachtet.[13] Diese Frage markierte den Abschluss unseres Fragenkataloges, mit dem wir die Software-Entwickler konfrontierten. Der Befragte, der Ästhetik als „Lehre von der Schönheit der Dinge“ ansah, antwortete hier, dass er unter „schön“ eine „Empfindung“ oder ein „gutes Gefühl“ verstehe, welches „Freude bereitet“ und „durch Ästhetik ausgedrückt“ wird. Bei den anderen Befragten fielen die Antworten stärker mit ihrem zuvor geäußerten Verständnis von Ästhetik zusammen. Unter Ästhetik wird also von verschiedenen Menschen etwas anderes verstanden; dies muss im Weiteren beachtet werden.

Zusammenfassend lässt sich sagen, dass die Befragten zwar alle ihre individuelle Vorstellung vom Begriff der Ästhetik hatten, jedoch implizierte der Ästhetikbegriff bei allen vier Entwicklern sinnlich Wahrnehmbares (siehe dazu auch 2.5 auf der nächsten Seite).

2.4 Ästhetik als Resultat eines Arbeitsprozesses

Eine weitere Frage, die sich zum Thema Ästhetik stellt, wenn man es unter dem Aspekt der Softwareentwicklung betrachtet, ist die nach ihrer Entstehung. Ist die Erschaffung ästhetischer Werke automatisierbar? Ist Ästhetik nur das Resultat eines algorithmischen Prozesses, oder steckt zu viel kreative Leistung dahinter, um es zu automatisieren?

Einer unserer Softwareentwickler erwiderte auf die Frage, was er von der Idee Software als zweckfreie Kunst zu produzieren hällt, dass es für ihn nur schwer vorstellbar wäre. Er argumentierte damit, dass Softwareentwicklung ein handwerklicher Prozess sei, dem jeder Bezug zur Kunst fehle (abgesehen von der Entwicklung von User Interfaces).

Anders sah dies Frieder Nake, der sich ausführlich damit beschäftigt hat.[14] Er unter-

[13] Vgl. [Adorno 1973], S. 82: „So wenig ist das Schöne zu definieren, wie auf seinen Begriff zu verzichten, eine strikte Antinomie.“

[14] Vgl. [Nake 1974].

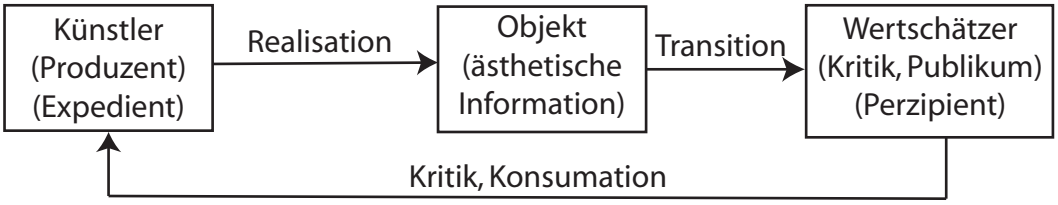

Abbildung 1: Nach Frieder Nake: “Einfaches Modell des ästhetischen Prozesses”

nimmt den Versuch, den Prozess der Kreation von Kunstwerken in einzelne Schritt zu zergliedern. In Abb. 1 sieht man, wie er beginnt, den Prozess zu zergliedern. Ein Künstler realisiert ein Kunstwerk (Objekt mit ästhetischer Information) und ein Wertschätzer (z. B. der Künstler selbst, der einen Schritt von der Leinwand zurück macht, um sein Bild aus der Ferne zu betrachten) kritisiert das Werk, worauf hin der Künstler aufs neue am Werk arbeiten kann, um diese Kritik auszumerzen. Diese einzelnen Prozesse bricht er dann in weitere Subprozesse herunter, wie in Abb. 2 auf der nächsten Seite angedeutet werden soll. So versucht er durch das Zerlegen der Probleme auf kontrolierbare und für Computer lösbare Teilprobleme zu kommen.

Doch bei genauerem Hinsehen fallen einige Mängel an diesem Prozess auf. Zum Einen bewertet der Computer seine eigenen Werke, wodurch natürlich Werke zustande kommen würden, die der Computer als ästhetisch “empfindet” und keine vom Menschen als ästhetisch empfundenen. Zum Anderen fehlt dem Computer eine Zielstellung, ohne die keine Bewertung des erschaffenen Werkes möglich ist. So kann man dem Computer dann entweder Bewertungsrichtlinien und Zielstellungen vorgeben, womit er am Ende nur noch zu einem Wekzeug verkommt und der Mensch wieder das kreative Glied des Prozesses wäre, oder ihn ziellos irgendetwas vor sich hin rechnen lassen.

Scheinbar lässt sich Ästhetik also nicht so einfach algorithmisieren, ob es jedoch möglich ist oder nicht lässt sich aufgrund der Unklarheit und Abstraktheit des ästhetischen Begriffes nur schwer beweisen.

2.5 Angewandte Ästhetik

Nachdem der Begriff der Ästhetik aus verschiedenen – u. a. wissenschaftlichen aber auch individuellen – Perspektiven betrachtet wurde, stellt sich die Frage, welche Dinge nach ästhetischen Maßstäben wahrgenommen werden. In den Antworten der Software-Entwickler fanden sich hier überwiegend visuelle wahrnehmbare Dinge wieder. Da aber konkret „Autos, Gebrauchsgegenstände, Werkzeuge“ sowie „Sachen aus dem Haushalt“ genannt wurden, liegt es nahe, dass auch die Haptik der Dinge bei der Betrachtung nach ästhetischen Maßstäben wichtig ist. Ein Befragter (A) sprach auch von „Gärten“, die er einer ästhetischen Betrachtung unterzöge. Interessanterweise war jener Befragte nicht identisch mit jenem (B), der vor seiner Tätigkeit in der Software-Entwicklung ein Studium der Landschaftsarchitektur absolviert hatte. Dieser antwortete nämlich lediglich, er betrachte alles, was er „mit dem Auge erfasse“ unter ästhetischen Maßstäben. Dennoch scheint die Betrachtung von „Gärten“ ein interessantes Beispiel zu sein, da hier mögli-

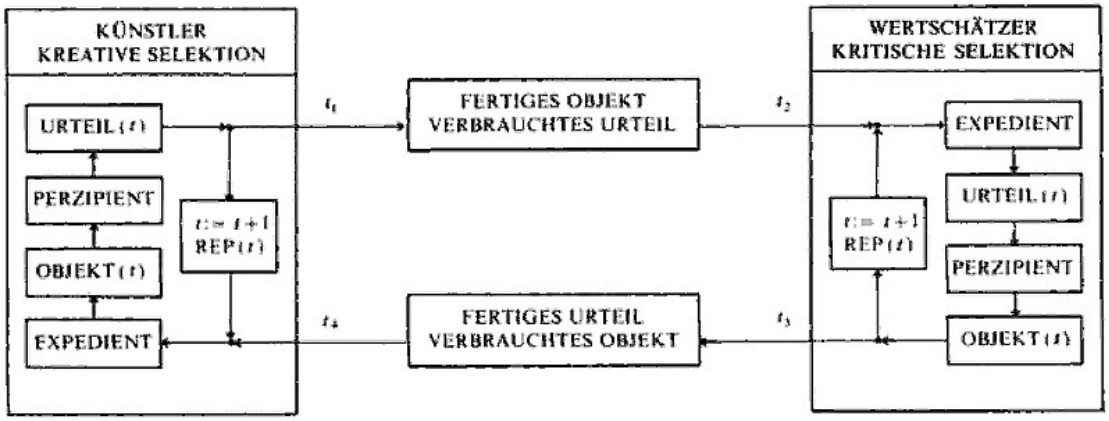

Abbildung 2: Frieder Nakes' "Verfeinertes Modell des ästhetischen Prozesses"

cherweise nicht nur Optik und Haptik als Sinn eine Rolle spielen; ein Garten hat eine charakteristische Geräuschkulisse und besitzt Gerüche.[15]

Wir gehen hier davon aus, dass sich grundsätzlich alle Dinge unter ästhetischen Gesichtspunkten betrachten lassen. Dominant ist dabei die Optik, obwohl auch anderen Reizen eine Beteiligung grundsätzlich zugesprochen werden muss, wie wir lediglich andeuten wollten. Die befragten Software-Entwickler schränkten sich demgegenüber aber hauptsächlich auf Visuelles ein; die Beteiligung anderer Sinne lässt sich allenfalls implizit aus den Antworten ableiten, wird also vermutlich bei den Befragten nicht reflektiert.

Zusätzlich zur sinnlichen Wahrnehmung konnten wir indirekt noch ein weiteres Kriterium für ästhetisches Empfinden ableiten – die Assoziationen zu betrachteten Objekten. So erwähnte jener Entwickler (C), für welchen Ästhetik "die Lehre der Schönheit der Dinge" ist, dass er Ästhetik auch in seinen Mitmenschen sehe, bzw. diese nach ästhetischen Maßstäben beurteile. Dies lässt darauf schließen, dass Ästhetik nicht nur von den fünf Sinnen abhängig ist, sondern, dass die Gedanken und Assoziationen der wahrnehmenden Person einen erheblichen Einfluss auf die ästhetische Wahrnehmung haben. Diese Vermutung wird noch unterstützt von der in Abschnitt 3.3 gewonnen Erkenntniss, dass Dokumente, Modelle und Quellcode eine "klare Gedankenstruktur" haben müssten oder "angenehm zu lesen sein" müssten um als ästhetisch angesehen zu werden. Bedenkt man die Wichtigkeit und Wirkung der benanneten Facette auf die Ästhetik, so lässt sich darüber streiten, wie einfach oder kompliziert ästhetische Kunst mithilfe simpler automatisierter Arbeitsprozesse, wie in 2.4 beschrieben zu erschaffen sind, bzw. ob dies überhaupt machbar ist.

[15] Vgl. wiederum [Stifter 2005], wo die Geräuschkulisse von Gärten aufgrund der angesiedelten Singvögel thematisiert wird.

3 Ästhetik für Software-Entwickler

Im Folgenden soll speziell auf die Bedeutung ästhetischer Kategorien in der Softwareentwicklung eingegangen werden. Zunächst erfolgt hierzu eine Annäherung an den Gegenstand mittels Auswertung der Befragung zur eigenen Einstellung und zum Arbeitsplatz der Softwareentwickler. Anschließend wird unter Bezugnahme auf den so genannten Software Lyfe Cycle betrachtet, welche ästhetischen Kategorien für die Resultate (z. B. Dokumente) der verschiedenen Phasen jenes Lebenszyklus' festzustellen sind. Dieser zweite Schritt wurde in der Befragung ausführlich thematisiert, so dass auch hierfür die Antworten von 4 Software-Entwicklern nutzbar sind.

3.1 Software-Entwicklung als Handwerk

Ohne hier auf das noch zu erörternde Thema „Software und Kunst“ vorzugreifen, steht zu Beginn dieses Abschnitts die Äußerung des ersten Befragten (A), Software-Entwicklung sei ein „handwerklicher Prozess“ (und nicht etwa ein künstlerischer).

Unser Interview-Design bot – ohne dass diese Äußerung vorhersehbar war – für diese Betrachtungsweise eine eigene Frage. Die Frage sollte ermitteln, was das „Material“ und was die „Werkzeuge“ (des Software-Entwicklers) wären, wenn „Software ein Möbelstück“ und der Entwickler ein „Handwerker“ wäre, der es produzieren muss.

Lediglich ein Befragter (C) beantwortete die Frage – wie von uns intendiert – mit Begriffen aus der Software-Entwicklung. Die anderen Befragten fühlten sich stattdessen zunächst in die Rolle des klassischen Handwerkers ein. Dann wurde als Material sofort „Holz“ (A, B und C) und nur von einem zusätzlich „Metall“ (B) genannt. Als Werkzeuge wurden „Handwerkzeuge“ (A) oder – expliziter – „Feilen, Hammer, Schmiedeeisen, Feuer, Säge“ (B) erwähnt.

Darüber hinaus unterschied wiederum der erste Befragte (A) dieses – zunächst auch von ihm geäußerte – idealisierte Bild von einem seine Arbeitsrealität angemessen widerspiegelnden Bild. Für die „Realität“ sah er sich mit „Spritzgussplastik“ und „vorgefertigten Teilen“ handwerken.

Zwei Befragte (B und D) wurden nach Schilderung ihrer Handwerksphantasien auf die eigentlich von uns intendierte Bedeutung der Frage hingewiesen und antworteten dann auch noch mit Begriffen aus der Software-Entwicklung.

Die einzige Antwort (C), die sich nur auf solche Begriffe bezog, führte als Material „Anforderungen und eigene Ideen“ und als Werkzeuge die „Entwicklungsumgebung und die technische Plattform“ auf. Ein weiterer Entwickler (B) sah seinen „Kopf“ als Material und „Tastatur, Rechner, Software, Compiler“ als Werkzeuge an. Schließlich rundete die Antwort (D), Werkzeuge seien „Programme, Software“ und das Material sei „Luft beziehungsweise Nichts“ den Reigen der variierenden Antworten ab.

Diese letztgenannte Vorstellung, das Material des Software-Entwicklers sei „Luft beziehungsweise Nichts“, fiel etwas aus dem Rahmen. Der Befragte (D) nannte außerdem lediglich „Programme, Software“ als Werkzeuge und setzte hinzu, daraus würde „Sourcecode“. Gedanken fielen also hier nicht unter die Kategorie Material, was vielleicht daher rührt, dass der klassische Handwerker mit Holz, Metall und Handwerkzeugen ja

schließlich auch Gedanken auf seine Arbeit verwendet, die nicht explizit in seiner Materialstückliste aufgeführt sind. Eine andere Möglichkeit bestünde darin, zu vermuten, dass der Entwickler seine Gedanken in diesem Prozess für so unbedeutend hielt, dass er diese für nicht erwähnenswert erachtete, da er auch auf die Frage, ob er sich seiner Sache sicher sei, auf seinem Standpunkt beharrte.

Aus der Bandbreite der Antworten könnte geschlossen werden, dass die Analogie des Handwerkers wohl den Beruf des Software-Entwicklers nicht ganz umfänglich abbilden kann.

Möglicherweise spielt hier auch die Tatsache eine Rolle, dass die Software-Entwicklung aus Sicht der unmittelbar mit ihr beschäftigten Menschen deutlich stärker zergliedert ist, als dies gemeinhin von der Herstellung von Möbelstücken angenommen wird; den meisten Befragten wird kein Äquivalent des Software Life Cycle für Möbelstücke bekannt gewesen sein, während sie über den Prozess der Software-Entwicklung und die damit einhergehende Zergliederung schon aufgrund firmeneigener Richtlinien informiert waren.

3.2 Ästhetik am Arbeitsplatz

Der Betrachtung von Ästhetik im Software-Entwicklungsprozess vorangestellt soll hier auf die Gestaltung des Arbeitsplatzes eingegangen werden. Neben der Grundausstattung von Bürostuhl, Schreibtisch und Computer richten sich Mitarbeiter an ihren Arbeitsplätzen ein – vom Familienbild bis zur Überraschungseifigur.

Auf die Frage, inwieweit der persönliche „Arbeitsplatz seine eigene Ästhetik“ hat und wie der Befragte diese beschreiben würde, wurde erwidert, der Arbeitsplatz sei „eher funktional als sinnlich“ (A), „ohne Ästhetik, nicht nach ästhetischen Maßstäben“ eingerichtet (B) bzw. „weniger ästhetisch, nicht schön“ (C). Der vierte Kandidat (D) antwortete, sein Arbeitsplatz sei „ein Mac“ und der Rest sei „zweckorientiert“.

Man kann also behaupten, dass bei allen Befragten Ästhetik am Arbeitsplatz nicht bewusst eine Rolle spielt. Insofern wurde die Fragestellung von allen mit Überraschung aufgenommen. Teilweise wurde bedauert, dass die Arbeitsplätze nicht nach ästhetischen Maßstäben eingerichtet sind. Offensichtlich resultiert die Ästhetik des Arbeitsplatzes zu einem großen Teil aus Entscheidungen des Arbeitgebers. In vielen Firmen gibt es Richtlinien für die Arbeitsplatzgestaltung; außerdem existieren hierfür in Deutschland gesetzliche Regelungen.[16] Während unseres Vortrages zu diesem Thema, kam aus dem Publikum eine Anmerkung, dass einige Arbeitgeber, sogar vorgeben, wie man seinen Laptop/Desktop einzurichten hat. So gibt es sogar Corporate Designs für Desktop-Hintergründe und sonstige Layouts, die von den Firmen vorgegeben werden.

Zudem macht die Antwort, dass der Arbeitsplatz “ein Mac” sei, auf ein weiteres Phänomen der Softwareentwicklung aufmerksam: In der Softwareentwicklung spielt sich ein Großteil der Arbeit viel mehr “im” Computer ab, als “um ihn herum”, wodurch die Räumlichkeit in der sich die Arbeit abspielt, weniger bewußt oder (im Fall des Befragten sogar) überhaupt nicht wahrgenommen wird. Dies ist scheinbar ein weiterer Grund

[16] Hier sind u. a. die Arbeitsstättenverordnung (ArbStättV) und das Gesetz über die Durchführung von Maßnahmen des Arbeitsschutzes zur Verbesserung der Sicherheit und des Gesundheitsschutzes der Beschäftigten bei der Arbeit (ArbSchG) zu nennen.

dafür, dass keine Priorität auf die Schaffung von Ästhetik im Arbeitsraum gesetzt wird.

Letztlich werden ästhetische Gesichtspunkte von den meisten Arbeitgebern wie auch Arbeitnehmern im Software-Bereich für die Gestaltung des Arbeitsplatzes nicht umfassend reflektiert. Allerdings ist es auch fraglich, ob eine stärkere Berücksichtigung solcher Aspekte wirklich zu einer Steigerung der Produktiv der Arbeitnehmer führen würde.

3.3 Ästhetik im Software-Entwicklungsprozess

Geht man von einem Software Life Cycle aus, wie er in der Industrienorm IEEE 12207 beschrieben ist, so beziehen sich die Tätigkeiten von Software-Entwicklern auf einen Teil dieses Lebenszyklus´, der als Software-Entwicklungsprozess bezeichnet wird.[17] Vereinfacht besteht der Software-Entwicklungsprozess im Wesentlichen aus den folgenden Phasen:

1. Anforderungsanalyse
2. Design
3. Implementierung
4. Test

Zwar sind beispielsweise in der IEEE 12207 noch deutlich mehr zum Entwicklungsprozess zählende Phasen ausdifferenziert, jedoch spiegelt dies lediglich ein umfassendes Konzept der Phaseneinteilung wider, welches in den meisten Unternehmen nicht in der gleichen Umfänglichkeit angewandt wird. Die vier von uns angeführten Phasen bilden den Teil dieses Konzeptes, der nach Erfahrung der Verfasser dieser Arbeit – im Gegensatz zur IEEE 12207 – mehr oder weniger vollständig in der Praxis verwendet wird. Die IEEE 12207 stellt Softwareentwicklung nämlich als reinen Handwerksprozess dar, als welchen wir ihn jedoch nicht ansehen wollen, da diese Betrachtung Kreativprozesse zu sehr vernachlässigt. Die Sicht auf – mit ästhetischen Zielen betriebene – Software-Entwicklung als Handwerksprozess wurde bereits in 2.4 diskutiert.

Für die Befragung der Software-Entwickler haben wir uns daher nicht so sehr an der IEEE 12207 orientiert. Wir wollten herausfinden, welche Rolle Ästhetik in den einzelnen Phasen des Software-Entwicklungsprozesses spielt. Zum Einen haben wir daher gezielt nach der Rolle von Ästhetik in den Resultaten des Software-Entwicklungsprozesses gefragt; unter Resultaten verstehen wir dabei:

- Dokumente (Design, Dokumentation u. a.)
- Modelle (etwa UML)
- Quellcode
- User Interfaces

[17] Eine Übersicht über die in IEEE 12207 enthaltenen Phasen gibt es etwa in [Rico 2003].

Außerdem wollten wir von den Befragten wissen, welche Rolle Ästhetik in der *Kommunikation* (mit Kollegen, Kunden, Testern u. a.) spielt. Anders als etwa Dokumente oder User Interfaces besitzt Kommunikation in Projekten nicht immer eine fassbare Materialität, was sie von den oben aufgeführten Resultaten des Software-Entwicklungsprozesses also unterscheidet. Andererseits sollte als Resultat des Ablaufs einer jeden Phase im Software-Entwicklungsprozess unbedingt Kommunikation – sei es in druckbaren Mails oder in Flurgesprächen – erfolgen. Somit haben wir also unserer Meinung nach zu Recht eine fünfte Kategorie identifiziert.

3.3.1 Dokumente

In Dokumenten spielten für die Befragten hauptsächlich Struktur, Gliederung und Kapselung, Lesbarkeit eine Rolle. Dies bezieht sich nicht so sehr auf Typografie (die ohnehin in den Firmen durch Standardschriftarten oder Dokumentenvorlagen meist festgelegt ist), sondern auf die Strukturierung der Inhalte – etwa bei der Einteilung der Kapitel.

Eine besondere Rolle haben o. g. ästhetische Merkmale in Teamarbeit oder auch in Phasen der direkten Kommunikation mit Kunden. Werden Spezifikationsphase, Design Reviews sowie Übernahme und Abnahme zusammen mit dem Kunden gestaltet und durchgeführt, so müssen nicht nur die Entwickler mit den Dokumenten arbeiten können, sondern auch die auf der Kundenseite beteiligten Personen; schließlich kann aus einem Software-Entwicklungsprojekt z. B. auch ein Betriebshandbuch hervorgehen.[18]

Als wissenschaftliche Disziplin zur Analyse solcher Texte erscheint die Linguistik – und hier speziell die Gebrauchstextanalyse – geeignet. Operiert man mit den Begriffen Kohärenz (als Zusammenhang des Gesamttextes) und Kohäsion (als Sammlung von Mitteln um Kohärenz zu erzeugen), so lässt sich zeigen, mit welchen Mitteln der Text – neben so augenscheinlichen Dingen wie der Kapiteleinteilung – strukturiert ist und woraus möglicherweise Verständnisschwierigkeiten resultieren.[19]

3.3.2 Modelle

Unter dem Begriff „Modelle“ verstehen wir hier nicht etwa Strukturen aus der Theoretischen Informatik sondern zumeist graphische Darstellungen von Sachverhalten im Software-Entwicklungsprozess – etwa in Diagrammen – die zu ähnlichem Zwecke erstellt werden können, wie sich beispielsweise Architekten maßstabsgetreuer Modelle von Gebäuden bedienen.

Unsere Modelle sind also Visualisierungen und unterscheiden sich hierdurch von den Texten in den zuvor thematisierten Dokumenten. Gleichwohl können Modelle natürlich auch in Dokumente eingebunden sein und somit durch ihre Eigenschaften wie durch Platzierung im konkreten Dokument die Struktur des Dokumentes und den Eindruck des Lesers beeinflussen.

Daneben gilt aber für einige Modelle (UML, Workflow, Ereignisgesteuerte Prozessketten u. a.), dass sie nicht nur in Dokumente eingebunden sind, sondern durch eine

[18] Man kann sicher auch ästhetische Maßstäbe der Literaturwissenschaft anlegen (vgl. [Eagleton 1994]).
[19] Vgl. [Willkop 2001], S. 314 ff.

eigenständige Existenz einen spezifischen Wert besitzen. Sie müssen nicht immer dokumentarischen Charakter besitzen; seit einiger Zeit werden diese in andere Dokumente, Modelle oder Quellcode transformiert, etwa wenn aus Klassen-Diagrammen Coding generiert wird.

Nach Meinung der Befragten sollten Diagramme „optisch ansprechend“ (A) und „direkt wahrnehmbar“ (B) sein. „Lesbarkeit“ wurde mehrmals (A und B) als wichtiges Kriterium genannt. Optisch ansprechend könne ein Diagramm oder Modell sein (C), indem auf „schöne Farben“ und „Geometrie“ zurückgegriffen würde. UML-Diagramme sollten „Grundbestandteile“ (D) hervorheben. Fürderhin wurde in einem Falle (D) das Kriterium des eigenen „Feeling[s]“ genannt, das bei der ästhetischen Bewertung von Modellen eine Rolle spiele.

3.3.3 Quellcode & Software

Nach Anforderungsanalyse und Design findet in einem Software-Entwicklungsprojekt heutzutage irgendwann auch die Implementierungsphase statt. Als Ergebnis dieser Phase entsteht Quellcode, also ein Text oder eine Sammlung von Texten; geschrieben in einer oder mehreren Programmiersprachen. Somit ist der Software-Entwickler bei der Gestaltung des Quellcodes bereits durch die verwendete(n) Programmiersprache(n) eingeschränkt, denn diese legt bzw. legen ggf. Schlüsselwörter, für Variablen und andere Namen verwendbare Zeichen, Syntax von Befehlen – und somit so etwas wie die Strukturierung auf Zeilenebene – und anderes mehr fest.

Gleichwohl es auch Programmiersprachen mit größerem Freiheitsgrad gibt, zeichnen die oben angeführten Restriktionen die Charakteristik heutzutage – und insbesondere bei den Befragten – häufig verwendeter Programmiersprachen nach. Sieht man von diesen Restriktionen ab, so ist die Gestaltung des Quellcodes fürderhin beschränkt durch die der Implementierung vorangegangenen Phasen im Software-Entwicklungsprozess. In Anforderungsanalyse und Design bzw. den entsprechenden Spezifikations- und Designdokumenten sind üblicherweise bereits Vereinbarungen getroffen worden, die bei der Implementierung lediglich zu übernehmen sind. Dies kann die Benennung von Entwicklungsobjekten und andere Aspekte der Strukturierung des Quellcodes betreffen. Nicht selten kommen interne oder vom Kunden vorgegebene Programmier- oder Benennungsrichtlinien hinzu. Letztlich obliegt aber die Umsetzung der Richtlinien dem Entwickler und dieser kann sich grundsätzlich meist auch über Richtlinien hinwegsetzen, während die Restriktionen der jeweiligen Programmiersprache vom Syntax-Check durchgesetzt werden.

Quellcode-Formatierung und die Einhaltung formaler Richtlinien (etwa bei der Wahl der Namen für Variablen und Parameter) waren den meisten befragten Entwicklern (A, B und C) sehr wichtig. Weitere Formatierungselemente, die genannt wurden, waren Abstände (A) und Groß- und Kleinschreibung (C). Kommentare wurden nur einmal (B) explizit erwähnt – und zwar in dem Kontext von Strukturierung, Kapselung und Modularisierung. Durch Nennung von „Modularisierung“ treffen Ästhetik und Software Engineering (als genuin der praktischen Informatik zugehöriger Bereich) zusammen, da die Modularisierung von Quellcode ja nicht nur aus ästhetischen Gründen praktiziert

wird, allerdings offenbar eine ästhetische Qualität besitzt. Des Weiteren wurde geäußert (D), der Quellcode sei wichtig für die Dokumentation.

Darüber hinaus würde es mehr Spaß machen (D), „elegante Implementierungen“ zu lesen. Lediglich diese letztgenannte Äußerung lässt auf Algorithmen schließen, die vielleicht als „elegant“ bezeichnet werden. Es fällt auf, dass bei keinem Befragten der Begriff des Algorithmus explizit verwendet wurde. Quellcode wurde hier ausschließlich nach formalen Gesichtspunkten analysiert – insofern entspricht das Vorgehen der Befragten weitgehend einer syntaktischen Sichtweise. Auf die Semantik des Quellcodes gingen die Befragten nicht oder kaum ein.[20] Das könnte so interpretiert werden, dass die Kategorie der Algorithmen eher der Designphase zugeordnet werden und nicht so sehr dem Quellcode bzw. der Implementierungsphase. Insofern hätte die Frage nach der Rolle von Ästhetik in Dokumenten im Software-Entwicklungsprozess möglicherweise für die verschiedenen Dokumentenarten feiner differenziert werden sollen.

3.3.4 User Interface & Verhalten der Anwendung

Der Quellcode stellt das Ergebnis der Implementierungsphase (und der vorangegangenen Phasen) dar, welches in erster Linie der Software-Entwickler begutachten kann. Für den Kunden sowie die Anwender ist dies selten möglich; sie sehen als Ergebnis des Software-Entwicklungsprozesses (soweit eines erstellt wurde) ein User Interface oder jedenfalls ein bestimmtes Verhalten der Anwendung.

User Interfaces sind aufgrund ihres graphischen Charakters sehr wesentlich an ästhetische Maßstäbe gebunden. Auch hier gibt es aber – ähnlich wie in der Implementierung durch die verwendete Programmiersprache für die Gestaltung von Quellcode – Restriktionen für die Gestaltung des UIs aufgrund der verwendeten Programmierumgebung. Insbesondere in der Entwicklungsumgebung des SAP-Systems unter Benutzung von ABAP sowie so genannten Dynpros ist aufgrund des SAP Style Guides sowie der äußerst beschränkten Zahl an verwendbaren Eingabeelementen und Controls der Gestaltungsfreiraum des Entwicklers sehr beschränkt.[21] Dieser Sachverhalt wurde von zwei (A und B) der drei befragten SAP-erfahrenen Entwickler implizit bzw. explizit dargelegt. Der einzige Entwickler (D), der über keinerlei SAP-Erfahrung verfügte und oft mit Web-Technologien entwickelte, zeigte eine weitere Restriktion für die UI-Entwicklung auf, da er die Vorgaben für die Gestaltung von Websites von einem Designer erhalte und insofern sein einziger Gestaltungsspielraum sei, zu kommunizieren, ob das Umsetzen dieser Vorgaben technisch möglich oder nicht möglich sei.

Schließlich sei – nach einem der befragten SAP-Entwickler (C) – der „Käuferwille“ entscheidend. Während er versuche „Verständlichkeit herzustellen“, besitze der Kunde bzw. Benutzer meist eine etwas andere Wahrnehmung, da Entwickler – und also auch er selbst – „meist zu kompliziert denken“ würden. Die einzige Aussage eines Befragten (A),

[20] Vgl. hierzu auch [Kernighan/Plauger 1976], S. 28, die schreiben: “But our primary tool for writing good programs is to strive to make them readable. In our experience, readability is the single best criterion of program quality: if a program is easy to read, it is probably a good program; if it is hard to read, it probably isn't good.”

[21] Vgl. [SAP 2001].

die wirklich auf die ästhetische Qualität des UI einging, war, dass Ästhetik für Web-Oberflächen eine Rolle spiele und solche UIs optisch ansprechend sein sollten. Es bleibt also festzuhalten, dass Kundenwünsche sowie weitere Beteiligte und die benutzte Entwicklungsplattform den Software-Entwickler bei der Gestaltung von UIs einschränken. Es mag auch ein Spezifikum der betriebswirtschaftlich orientierten Software-Entwicklung (wie bei SAP) sein, dass ästhetische Maßstäbe nicht in sonderlich hohem Maße an das UI angelegt werden. Bei richtigen Spiele-Entwicklern wären die Antworten hier also möglicherweise ganz anders ausgefallen, wenngleich die Wahrscheinlichkeit hoch ist, dass wiederum auf die Vorgaben von Designern verwiesen worden wäre.

3.3.5 Kommunikation im Software-Entwicklungsprozess

Die letzte abgefragte Kategorie des Software-Entwicklungsprozesses war im engeren Sinne kein Produkt einer oder mehrerer Phasen, sondern umfasste die Kommunikation im Software-Entwicklungsprozess. Damit war die Kommunikation mit Kollegen, Kunden, Testern u. a. gemeint. Von den Befragten wurde dem mehrheitlich keine ästhetische Relevanz attestiert.

Ein Entwickler (A) bemerkte allerdings, er würde Kollegen zur Code-Hygiene anhalten und mit Kunden über UIs kommunizieren.

Insofern bildet diese Kategorie eine Metaebene, die vor allem Dokumente und Modelle einschließt.

3.3.6 “Dem Wandel voraus” - eine kritische Würdigung der SAP AG

Wie weiter oben deutlich geworden ist, wurden die Antworten von drei der befragten Entwickler maßgeblich dadurch beeinflusst, dass sie vor allem mit der Entwicklungsumgebung der SAP AG entwickelten. Daher soll hier kurz auf SAP eingegangen werden, damit auch Leser ohne einschlägige Erfahrungen ein entsprechendes Hintergrundwissen beziehen können. SAP ist das größte deutsche Software-Unternehmen und entwickelt betriebswirtschaftliche Standard-Software, die vor allem in großen und mittleren – zunehmend aber auch in mittelständischen – Unternehmen zur Durchführung betriebswirtschaftlicher Prozesse eingesetzt wird. Man spricht auch von ERP-Software (Enterprise Ressource Planning). Die verbreitete Software SAP R/3 und ihre Nachfolger sind fast gänzlich in der SAP-eigenen Programmiersprache ABAP geschrieben. Software-Entwickler bei SAP und in Beratungsunternehmen oder bei Kundenunternehmen tätige Software-Entwickler nutzen daher ABAP bzw. die sogenannte ABAP Workbench als Entwicklungsumgebung. Zu bedenken ist, dass eine große Zahl von SAP-Entwicklern, die in Unternehmensberatungen und Softwarehäusern beschäftigt sind, zum Teil jahrelang an die SAP verliehen und an der Standardentwicklung beteiligt waren.

Um die (unternehmens-)kulturellen Unterschiede zwischen SAP-Software-Entwicklern und auf andere Plattformen spezialisierten Entwicklern zu erkennen, empfiehlt sich durchaus ein Blick in das Buch „Dem Wandel voraus“, in dem Hasso Plattner, der letzte SAP-Gründer, der das operative Geschäft der SAP – durch den Wechsel vom Vorstand in den Aufsichtsrat im Jahre 2003 – verließ, im Jahre 1997 mit August-Wilhelm Scheer,

Siegfried Wend und Daniel S. Morrow spricht, zu rezipieren. Es mag irritieren, dass es sich um Mitschriften von Interviews bzw. Diskussionen handelt. Es mag noch mehr irritieren, wenn derjenige, der das Interview führt, dem segelbegeisterten Plattner nach Erzählen einer seiner plätschernden Regatta-Geschichten mit einem „Wow! [sic!]“ sekundiert. Nach Rezeption des Buches lässt sich also erahnen, welcher Personenkult bei SAP um Hasso Plattner stattfand. Eine nette Verbindung zur Frage, ob Software-Entwicklung als handwerklicher Prozess aufzufassen ist, folgt:

> “[...] vielleicht ist der Ingenieur gar nicht so sehr das beste Bild, sondern eher der Vergleich zwischen Bauingenieur und Architekt. Ich würde den heutigen Informatiker mehr als den Bauingenieur sehen, und wir wollen Architekten ausbilden. Da kommern sehr viele Dinge zusammen; Design ist eine ganz wesentliche Komponente. Selbstverständlich muss auch ein Architekt Statik belegen und verstehen, dass ein Haus bestimmten physikalischen Grundanforderungen genügen muss. Nachher aber beim Realisieren des Bauprojekts benötigt er natürlich Leute, die auf Statik spezialisiert sind. Und er braucht viele andere, die auf vielen anderen Gebieten spezialisiert sind. Aber nur weil er diesen Überblick hat, kann er seine Archtitektenrolle ausüben. Und das ist eigentlich das, was wir im Sinn haben. Wir wollen Architekten im Bereich Softwaresystemtechnik ausbilden.”[22]

3.4 Restriktionen im Software-Entwicklungsprozess

Wir haben bereits Restriktionen im Software-Entwicklungsprozess angeschnitten. Restriktionen für die ästhetische Dimension sind Erfordernisse, die aus anderen Dimensionen der Software-Entwicklung stammen und die die Freiheit von Software-Entwicklern bei der Ausübung ihrer Tätigkeit beschränken.

3.4.1 Wirtschaftliche Restriktionen

Alle befragten Software-Entwickler arbeiteten in Unternehmen. Die Zugehörigkeit dieser Unternehmen zum Wirtschaftssystem stellt eine wesentliche Anforderung an Software-Entwicklungsprojekte: sie müssen wirtschaftlich sein, d. h. der – für die ausführende Firma in Geldeinheiten gemessene – Nutzen sollte größer sein als der Aufwand.

Verkauft die ausführende Firma ein Softwarepaket (etwa als „Standard“-Software), so sollten die erzielten Einnahmen die Ausgaben übersteigen. Im Falle von Individualentwicklungen wie auch im SAP-Umfeld wird hingegen häufig nach Aufwand abgerechnet, d. h. alle Stunden, in denen Software-Entwickler des ausführenden Unternehmens an einem Projekt arbeiten, werden dem Kunden in Rechnung gestellt. Da die meisten Kunden jedoch nicht das Risiko eines unbegrenzten Projekt-Budgets eingehen möchten, wird häufig vor der Auftragserteilung der Aufwand geschätzt und ein maximales Budget – in Tagen oder in Geldeinheiten – festgelegt. In diesem Falle kann der Software-Entwickler also sehr leicht selbst feststellen, wie wirtschaftlich er arbeitet, indem er die von ihm

[22] Quelle: [Plattner et. al. 2000], S. 195.

benötigte Arbeitszeit und die Schätzung vergleicht. Voraussetzung dafür sind natürlich realistische – und nicht etwa zu niedrig angesetzte – Schätzungen.

Diese Ausführungen zur Wirtschaftlichkeit von Software-Entwicklungsprojekten weisen weiters auf etwas hin: Versucht der Software-Entwickler sein Produkt besonders „ästhetisch“ zu gestalten, so muss er sicherstellen, dass der hierdurch ggf. verursachte Mehraufwand nicht zur Überschreitung des Budgets führt. Die wirtschaftliche Dimension der Software-Entwicklung beeinflusst in konkreten Projekten also wiederum die ästhetische Dimension. Zugleich ist dies aber auch in der Gegenrichtung der Fall, denn wenn ein Software-Entwickler eine Stunde Mehraufwand erzeugt, weil er den Quellcode auf besondere Weise einrückt, um seine ästhetischen Maßstäbe zu erfüllen, wird zwar kein Controller den Entwickler rügen, aber die Wirtschaftlichkeit des Projektes wird hierdurch wahrscheinlich vermindert, es sei denn diese Einrückungen würden z. B. die Handhabbarkeit des Quellcodes erhöhen und somit an anderer Stelle den Aufwand wieder reduzieren.

Wirtschaftliche Restriktionen wurden jedenfalls von allen Befragten als die entscheidenden Restriktionen im Projektalltag angegeben. So wurde geäußert „meist [sei] nicht genug Zeit“ vorhanden (A), die „Ergebnisse [seien] budgetgetrieben“ (C) bzw. „wirtschaftliche Faktoren [hemmten] die Ästhetik“ (D), weil „z.B. der Wunsch nach Aufhübschung des Codings meist“ (D) nicht mehr zu erfüllen sei.[23]

3.4.2 Richtlinien

Bereits im Abschnitt zu Quellcode & Software wurden Programmier- und Benennungsrichtlinien erwähnt, die aus der organisatorischen Dimension der Software-Entwicklung den Software-Entwickler in seiner Freiheit einschränken. Sieht man einmal davon ab, dass noch weitere Restriktionen existieren, die wir hier nicht thematisieren, so steht auch nicht a priori fest, ob Restriktionen sich unbedingt negativ auf die ästhetischen Gesichtspunkte in Software-Entwicklungsprozess auswirken. Programmierrichtlinien z. B. können bei Entwicklern, die über ein geringes eigenes ästhetisches Empfinden verfügen, eine gewisse ästhetische Qualität der Produkte befördern. So wurde von zwei Befragten explizit als Sinn von Programmierrichtlinien angegeben, sie seien „bei Teamarbeit“ sinnvoll (D) bzw. als „Marschrichtung für Leute mit geringeren ästhetischen Kompetenzen“ (B).

3.4.3 Wegfall von Restriktionen

Eine weitere Frage an die Software-Entwickler war, ob die Ergebnisse ihrer Arbeit im Sinne ihrer Ästhetik besser wären, wenn einzelne Restriktionen nicht vorhanden wären. Zwei der vier Entwickler (B und D) bejahten dies; einer (B) führte dazu an, er könne dann ja „nach eigenem Gutdünken“ entwickeln. Differenzierter sahen dies die beiden anderen Befragten. In einem Fall (A) wurde gesagt, dies sei „nicht allein“ ausreichend, sondern

[23] Vgl. auch die Studie in [Lepold/Böhret 1987], S. 133: „Bei der Befragung der Manager und Teamleiter kamen die Worte ‚Zeit, Zeitdruck, Terminprobleme’ öfters zur Sprache.“; Software-Entwicklung fand auch vor mehr als 20 Jahren unter Zeitdruck statt.

es bedürfe zunächst Schulungen, um ein „gemeinsames [ästhetisches] Verständnis“ zu finden. Der andere Befragte (C) äußerte sich so, dass die Software in formaler Hinsicht (Programmierrichtlinien) nicht besser werden könne, da er darauf „ohnehin“ achte. Ansonsten tue sich hier ein Spannungsfeld auf, weil er „zu einem hohen Vollendungsgrad neige“.

3.4.4 Sind Restriktionen sinnvoll?

Ähnlich wie bei der Frage nach den Auswirkungen eines Wegfalls von Restriktionen waren auch die Antworten auf die Frage, ob „diese Restriktionen trotzdem sinnvoll“ gefunden würden, gespalten. Diejenigen, die eine ästhetische Verbesserung ihrer Ergebnisse bei Wegfall der Restriktionen bejaht hatten, sahen die Restriktionen als sinnvoll an. Allerdings anscheinend eher für ihre Kollegen, denn einmal wurde hinzugefügt, hierin sei „eine Marschrichtung für Leute mit geringeren ästhetischen Kompetenzen“ (B) gegeben, und im anderen Fall wurde der Sinn der Restriktionen für „Teamarbeit“ bestätigt; „bei Alleinarbeit [aber könne] Weglassen sinnvoll sein“ (D).

Die beiden anderen Entwickler, die eine ästhetische Verbesserung ihrer Ergebnisse bei weggefallenen Restriktionen nicht bejaht hatten, fand sich die Haltung, Restriktionen seien „halt so“ und vom „Markt“ vorgegeben (A) bzw. sie würden „als Sachzwänge hin[genommen]“ (C). Auch bei dieser letzten Antwort folgte auf die „Sachzwänge“ noch der Verweis darauf, dass Restriktionen sinnvoll seien, „weil nicht alle willens [seien], von sich aus strukturiert [oder] ästhetisch zu arbeiten“.

Die drei Fragen zu Restriktionen mögen den Einfluss der Restriktionen auf die ästhetische Dimension für unsere vier Software-Entwickler nicht erschöpfend geklärt haben, sie scheinen aber einen guten Einblick in das Selbstverständnis der Befragten zu geben.

4 Software-Entwicklung und Gesellschaft

Nachdem unser Ästhetik-Begriff und der Software-Entwicklungsprozess in Bezug auf Ästhetik behandelt wurden, wollen wir uns in diesem Kapitel einer Betrachtung der Software-Entwicklung aus der Perspektive der Gesellschaft schrittweise annähern. Schließlich wird sich zeigen, inwiefern diese Veränderung des Standortes unserer Betrachtungen den Blick auf weitere Aspekte von Software im Zusammenhang mit Ästhetik lenkt.

4.1 Das Verhältnis von Realität und Software

Die Software, die die Entwickler spezifizieren, „designen“, implementieren, testen, wird im Kontext des Wirtschaftssystems nicht zum Selbstzweck hergestellt, sondern soll bestimmte Funktionen erfüllen. Ist der Software-Entwicklungsprozess ausreichend erfolgreich, so wird die Software einer – mehr oder weniger offensichtlichen – Nutzung durch Anwender zugeführt. Hierbei kann Software direkt an Vorgängen der gesellschaftlichen Realität beteiligt sein, ob es sich nun um eine Software zum Handeln von Wertpapieren oder zur Überwachung der Blutzuckerwerte handelt.

Bevor wir auf unsere Befragung eingehen, wollen wir Floyd und Klischewski zitieren, die schreiben:

> „1. Modelle sind nicht als Realitätsabbilder zu verstehen, sondern sie rekonstruieren im wesentlichen die Begriffe und die Sprache, mit der im Anwendungskontext Sinnzusammenhänge beschrieben und kommuniziert werden. Modelliert wird nicht die objektive Welt, sondern die Sicht der beteiligten Akteure.
>
> 2. Modelle vergegenständlichen Abstraktionen und wirken dabei normativ. Durch diese Integrationsfunktion konstruieren sie Realitäten. Modelle tragen damit zur Aufrechterhaltung einer sozial konstruierten Realität bei und kontrollieren Handlungen, indem sie die Wahrnehmung, Begründungszusammenhänge, usw. kanalisieren.
>
> 3. Modelle sind abhängig von Vorurteilen und dem Vorverständnis, mit dem die Modelle entwickelt werden. Diese Vorurteile lassen sich nur durch einen reflektierten Dialog [Agenda des Buches] transparent machen.
>
> 4. Die Modellierung verändert den Bezugsrahmen für die Handlungsorientierung und greift damit in die Interessen der beteiligten Akteure ein. Formale Modelle sind keine isolierten Gebilde, sondern werden in sozialen und kulturellen Prozessen konstituiert und anschließend in diesen Feldern wirksam. Als selbst ablaufende Modelle haben sie direkte Auswirkungen auf die Realität und verändern die Bedingungen sozialen und kulturellen Handelns.“[24]

Wir fragten die vier Software-Entwickler, wie sie „das Verhältnis der Realität“ zu ihrer Software einschätzen und ob Software ihrer Meinung nach die Realität verändert. Zweimal wurde hierauf geantwortet, dies sei „projektabhängig“ (B) bzw. „sehr stark projekt-/budgetabhängig“ (D). Eine etwas differenziertere Sicht offenbarte sich in der Antwort eines anderen Entwicklers (C), der behauptete, in Kundenprojekten sei der Bezug zur Realität größer als in der Standardentwicklung – also bei der Entwicklung von Standard-Software für viele Abnehmer; der Bezug sei als „umso größer, je näher am Kunden“ der Entwickler sei. Ein anderer Entwickler (A) antwortete, er „versuche das [Anm.: den Bezug zur Realität] im Hinterkopf zu behalten“. Drei der vier Befragten bejahten, dass Software die Realität verändert, ein Entwickler (D) äußerte sich hierzu lediglich nicht.

4.2 Feedback

Wenn nun Software in Realität und Gesellschaft Veränderungen herbeiführt, wie verhält sich die Gesellschaft zur Software? Eine teilweise Antwort auf diese Frage ist, dass zwei der 4 Befragten angaben, stolz auf ihr Produkt zu sein, wenn ein „positives Feedback“ (A) bzw. „Kundenlob“ (C) von Anwender-/ Kundenseite zurückgegeben werde. Auch wenn die anderen beiden Entwickler Stolz ganz individuell verspürten, wenn sie ihrer

[24] Quelle: [Floyd/Klischewski 1998], S. 3.

„Meinung nach das Beste herausgeholt“ hatten (D) bzw. die Software ihren „Ansprüchen gerecht“ (B) würde – letzteres übrigens auf das Erscheinungsbild bezogen (!). Hier wird dennoch klar, dass mindestens ein Rückkanal von der Gesellschaft zum Software-Entwickler existiert.

Die Befragung vertieften wir durch die Frage, welchen Stellenwert die Produkte der Software-Entwickler ihrer Meinung nach „in unserer Kultur“ hätten. Hierauf wurde zunächst in drei von vier Fällen mit Ratlosigkeit reagiert, indem „schwierig“ (A), „keinen“ (B) bzw. „fällt mir nichts ein“ (C) geantwortet wurde. Der soeben als erster zitierte Befragte (A) ergänzte dann die Software mache „Leuten das Leben leichter“, befreie „von Routineaufgaben“ und lasse so mehr „Zeit für kreativer Aufgaben“. Der vierte Befragte (D) antwortete direkt aber eingeschränkt auf „Webdesign“, dies sei diesbezüglich „sehr breit“ aufgestellt; der Stellenwert reiche hier von „informativ“ bei Webpräsenzen von Banken bis zu einem eigenen „popkulturellen Wert“ bei bestimmen Webpräsenzen wie derjenigen des Musiksenders MTV; außerdem habe er Versuche von Kunst im Bereich der Software-Entwicklung gesehen.

Ein noch genaueres Bild der Rückkanäle aus der Gesellschaft hätte eine Anwenderbefragung – auch und gerade unter Einbeziehung ästhetischer Begriffe – ergeben können, die jedoch den Rahmen dieser Arbeit bei Weitem gesprengt hätte.

4.3 Kunst & Software

Christiane Paul schrieb für “die Ars Electronica in Linz, das wichtigste Medienkunstfestival in Europa”:[25]

> “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.”[26]

Dieser Abschnitt soll die Beziehung von Software und Kunst näher erläutern. Dabei soll in 4.3.1 zu aller erst geklärt werden, was Kunst überhaupt mit Ästhetik zu tun hat, um diesem Abschnitt seine Daseinsberechtigung in einer Arbeit über Ästhetik zu geben. Danach versuchen wir in 4.3.2 die Software selbst, also Sourcecode, Dokumentation usw. als eigenständige Kunstwerke zu betrachten. Da – vor allem für Nicht-Informatiker – jedoch auch oder sogar gerade die sichtbaren Interfaces der Programme, einen künstlerischen Wert haben, sollen in 4.3.5 auch diese erörtert werden. Zuletzt soll dann auch noch Software als Werkzeug, sprich als “Pinselersatz” dienen (Abschnitt 4.3.6) und dadurch vollkommen neue Möglichkeiten eröffnen, sich künstlerisch zu verwirklichen (Abschnitt 4.3.7).

[25] Quelle: [Warnke 2005, S. 17].
[26] Quelle: [Paul 2003].

4.3.1 Beziehung von Kunst und Ästhetik

In Meyers' online Lexikon ist über *Kunst* folgendes zu lesen:

> "Kunst [althochdeutsch, zu können], 1) im weitesten Sinn jede auf Wissen und Übung gegründete Tätigkeit (z. B. Reitkunst, Kochkunst); 2) in einem engeren Sinn die Gesamtheit des vom Menschen Hervorgebrachten (Gegensatz: Natur), das nicht durch eine Funktion eindeutig festgelegt ist oder sich darin erschöpft (Gegensatz: Technik). [...] Im heutigen Verständnis ist die Kunst in die Teilbereiche Literatur, Musik, darstellende Kunst sowie bildende Kunst gegliedert (in der Moderne sind Grenzüberschreitungen häufig); [...]"

Dabei ist es nur ein naiver Begriff von Kunst, der hier verwendet werden soll. Das Thema "Was ist Kunst?" füllt bis heute ganze Bücher und ist immernoch nicht hinreichend geklärt, daher nehmen wir hier eine Vereinfachung für den Kunstbegriff an.[27] Im Folgenden sei Kunst sowohl das Erschaffen selbst, als auch das Erschaffene Objekt, dass bewusst unter der Zielstellung erschaffen wurde, etwas zu kreieren, dass auf das ästhetische Empfinden abzielt – egal ob positiv oder negativ.

Ästhetische Wahrnehmung wiederum umfasst die Kunst und geht noch darüber hinaus. Es gibt auch Definitionen die besagen, *alles* sei Kunst, wobei nach dieser Definition ästhetische Wahrnehmung natürlich nicht über diese hinaus gehen könnte. Um derartige Missverständnisse auszuschließen, haben jedoch wir unseren naiven Kunstbegriff eingeführt. So umfasst Ästhetik nun nach unseren Definitionen auch Objekte und Sachverhalte, die nicht unter künstlerischem Vorwand geschaffen wurden.

Des weiteren wohnt der Kunst anheim, dass sie lediglich den "Selbstzweck" zu erfüllen habe. Das soll heißen, dass Kunst keinen anderen Zweck haben soll, als Kunst zu sein. Da jedoch ein stilvoll verzierter Hammer sowohl unter dem Aspekt, auf ästhetische Wahrnehmung abzuzielen, als auch unter dem Fremdzweck, als Werkzeug zu fungieren, geschaffen wird, soll der "Selbstzweck" in unserer Betrachtung kein notwendiges, aber dennoch ein nicht außer Acht gelassenes Merkmal sein.

4.3.2 Software als eigenständige Kunstwerke

Sourcecode

Zunächst wollen wir den Quellcode als eigenständige Kunst betrachten. Code wird normalerweise unter folgenden Kriterien entworfen:[28]

- Korrektheit
- Wartbarkeit
- Lesbarkeit

[27] Vgl. z. B. *Was ist Kunst?: Positionen der Ästhetik von Platon bis Danto* von Michael Hauskeller.
[28] Quelle: [Trueg 05, S. 23].

- Übersichtlichkeit, Klarheit
- Gutmütiger Umgang mit dem Benutzer
- Effizienz

Diese sind nicht unbedingt als ästhetische Kriterien anzusehen. Jedoch kann Sourcecode auch aus diesem Rahmen fallen und so gänzlich neue Kategorien der Kunst, mit teilweise oder völlig eigenen Kriterien, bilden.

Sofern der Code selbst das Kunstwerk darstellen soll, ist es sehr wahrscheinlich, dass die Kategorie, in die sich das Kunstwerk einordnen lässt, entweder viel Wert auf die 6 oben genannten Kriterien legt und diese selbst als ästhetische Maßstäbe ansieht, oder dass der Code "frei" von diesen Restriktionen ist, dann jedoch unter Umständen nicht einmal einen Sinn erfüllen muß. Im ersten Fall stimmt die Auffassung von Ästhetik im Quellcode mit der unserer Softwareentwickler überein (siehe dazu auch Abschnitt 3.3.3). In zweiterem Falle eröffnen sich viele neue Möglichkeiten des künstlerischen Ausdrucks, derer wir hier, zur Gewinnung eines besseren Eindruckes, zwei aufzeigen wollen:

Softwaregedichte sind Gedichte die in der Syntax einer bestimmten (Programmier-) Sprache geschrieben werden, so dass der Scanner des Compilers dieser Sprache sie erkennen würde. Es wird sich also streng an die Syntax der Sprache gehalten, jedoch steht einem die Form offen. Hier als Beispiel ein Gedicht von Sharon Hopkins's – "listen":[29]

```
APPEAL:
listen (please, please);
open yourself, wide;
    join (you, me),
connect (us, together),
tell me.
do something if distressed;
    @dawn, dance;
    @evening, sing;
    read (books,$poems,stories) until peacefull;
    study if able;
write me if-you-please;
sort your feelings, reset goals, seek (friends, family, anyone);
            do*not*die (like this)
            if sin abounds;
keys (hidden), open (locks, doors), tell secrets;
do not I-beg-you, close them, yet.
                    accept (yourself, changes),
                    bind (grief, despair);
require truth, goodness if-you-will, each moment;
select (always), length(of-days)
```

[29] Quelle: [Trueg 05, S. 25].

Ob dieses Programm ausführbar ist oder gar einen Zweck erfüllt, steht hierbei nicht zur Debatte. Der Quellcode dient einzig und allein dem zuvor besprochenen Selbstzweck. Damit unterscheiden sich diese Gedichte schematisch nicht von klassischen Gedichten wie z.B. Haiku, da diese auch nur bestimmten "syntaktischen Regeln" zu folgen haben.

Als zweites wollen wir hier die sogenannten "Einzeiler" vorstellen. Das sind Programme, die nur aus einer einzigen Codezeile bestehen sollen, aber dennoch bei Ausführung eine (mehr oder weniger) sinnvolle Funktion erfüllen sollen. Hier ein Beispiel des Gewinners des "Best One Liner" Award 1987 – Dave Korn:[30]

```
main() { printf(&unix["\021%six\012\0"], (unix)["have"]+"fun"-0x60);}
```

Ob es sich bei diesen Programmen nun um Kunst oder um einfache "Witze" handelt, ist fragwürdig. Jedoch schlagen sie eine Brücke zu Abschnitt 4.3.5, da bei ihnen sowohl die Form des Quellcodes, als auch die Ausgabe / das Interface beim Ausführen des Quellcodes von Bedeutung ist.

Programmidee

Wenn man die interne Struktur von Programmen unter ästhetischen Gesichtspunkten betrachtet, so ist Sourcecode nicht die einzige Möglichkeit der Analyse.

Wie schon in Abschnitt 2.5 erläutert, lässt sich die Ästhetik von Software nicht nur durch sinnlich Wahrnehmbares erfassen, sondern auch mithilfe abstrakterer Sachverhalte, wie die "klaren Gedankenstrukturen".

So schreibt Thomas Dreher zur sogenannten "Konzeptuellen Kunst" beispielsweise:

> "Sich selbst beschreibende und sich selbst dokumentierende beziehungsweise sich selbst replizierende Codes, sogenannte Quines, sind Modelle zur Funktion von Programmiersprachen und als solche konzeptuelle Gegenstücke zu Algorithmischer Ästhetik: Es werden nicht Regeln zur Erzeugung von visuellen Effekten in Ausgabemedien nach ästhetischen Kriterien, sondern Reflektionsmodelle erstellt, die ein Vergnügen an der Art provozieren können, wie sie Probleme des Konstruierens von Programmcodes an Hand von Selbstdokumentationen vorführen."[31]

Ein Auszug aus der Definition für diese Quines:

> ":quine: /kwi:n/ /n./ [from the name of the logician Willard van Orman Quine, via Douglas Hofstadter] A program that generates a copy of its own source text as its complete output. Devising the shortest possible quine in some given programming language is a common hackish amusement."[32]

[30] Quelle: [Trueg 05, S. 26].
[31] Quelle: [Dreher 2005].
[32] Quelle: [Thompson 1999].

Hierbei handelt es sich also um Programme, deren Sinn darin besteht, mithilfe von möglichst wenig Sourcecode eine Ausgabe-Datei mit dem eigenen Sourcecode zu erzeugen.

Diese Kunst ließe sich auch im vorherigen Teilabschnitt zum Sourcecode einordnen, da auch hier der Sourcecode – ähnlich dem Einzeiler – möglichst knapp gehalten sein soll. Auch ließe er sich in den Abschnitt 4.3.5 einsortieren, in dem es unter anderem um die Outputs von Programmen gehen soll, da es bei Quines darauf ankommt, eine ganz bestimmte Datei auszugeben. Doch die eigentliche kreative und ästhetische Leistung steckt natürlich in der Idee die sich hinter dem geschriebenen Quines verbirgt, weswegen wir sie in diesem Abschnitt anbringen.

Quines können auch einen gewissen Grad an "Selbstzweck" vorweisen, denn abgesehen vom Zweck des "common hackish amusement", dienen sie keinem wirklichen Zweck als zu existieren und sich selbst zu reproduzieren.

Sich selbst vermehrende Programme sind natürlich nur ein Tropfen im Meer der künstlerischen Möglichkeiten im Gebiet der "Konzeptuellen Kunst", doch sollen sie genügen, um einen Eindruck zu bekommen in welcher Art und Weise eine *Programmidee* ein Kunstwerk werden kann.

4.3.3 Software-Entwicklung oder auch nur "Programmierung als künstlerische Praktik"

Nachdem belegt wurde, dass Software als Kunst fungieren kann, soll kurz auf die Person der Künstler eingegangen werden. Hierbei fungiert also der Künstler zugleich als Software-Entwickler bzw. der Software-Entwickler als Künstler. Künstlern, die Software-Entwicklung – oder zumindest die Programmierung – verfügbar zu machen, ist der Hintergedanke von [Trogemann/Viehoff 2005]. Die Agenda ist klar und wird von Gerrit Kohlke in der leider vergriffenen Publikation "Software Art" so beschrieben:

> „Softwarekunst ist keine Verfahrenstechnik zur Erzeugung anziehender Kunstoberflächen. Sie ist der realistische künstlerische Zugriff auf Software als allgegenwärtiges Material unserer Gesellschaft. Der unausgesprochene Vorwurf der Softwarekunst an die Kunst betrifft deshalb keinen Wettstreit innovativer Formen, sondern einen blinden Fleck der Kunst. Denn was kann Kunst können, wenn sie die informatorischen, genetischen, ökonomischen Codesprachen nicht beherrscht? [...] ‚Software Art' ist eine Reportage über den Code. Denn Softwarekunst ist keinesfalls Kunst für Programmierer. Softwarekunst ist die Lesbarmachung des Codes."[33]

Trogemann und Viehoff konstatieren folgendes:

> „Programmierende Künstler begreifen den Code als Material und damit wesentliche Komponente ihrer Projekte. [...] Die allermeisten Einführungen in die Programmierung konzentrieren sich auf die reine Fertigkeit der Codierung und glauben, damit sei schon alles Notwendige gesagt. Doch Softwarestrukturen sind keine technische Errungenschaft der letzten 50 Jahre,

[33] Zitiert nach: [Trogemann/Viehoff 2005], S. 11.

sondern stehen in einer kontinuierlichen Denktradition, die so alt ist, wie die Menschheit selbst.“[34]

Und deshalb besteht Trogemanns und Viehoffs Versuch, Künstler an die Software-Kunst heranzuführen, aus einer mit Kultur- und Medientheorie verknüpften Einführung in die Programmierung mit Java. Dieser – sicher begrüßenswerte – Versuch beinhaltet jedoch (noch) nicht Software-Entwicklung in einer Prozesshaftigkeit, wie in IEEE 12207 oder der beruflichen Praxis vieler Software-Entwickler üblich (siehe 3.3). Insofern stellt sich die Frage, ob Software-Kunst (irgendwann) nach ähnlichen Regeln und vor allem mit den nämlichen Schritten im Entwicklungsprozess produziert wird wie etwa Spiele oder betriebswirtschaftliche Software.

4.3.4 Entwickler-Meinungen

Die von uns befragten Entwickler hatten unterschiedliche Meinungen zu der “Idee, Software als Kunst ohne einen bestimmten Zweck zu produzieren”. In einem Falle (A) wurde die Umsetzbarkeit einer solchen Idee als “schwer vorstellbar” in Zweifel gezogen, da “abgesehen von User Interfaces” Software-Entwicklung ein “handwerklicher Prozess” sei, dem der “Bezug zur Kunst” fehle. Zwei andere Entwickler waren da schon aufgeschlossener, in dem sie von einer “interessanten Möglichkeit” (B) sprachen bzw. äußerten, sich dies “gut vorstellen” zu können (C). Ein Befragter (C) ergänzte:

> “Außerhalb meines zweckgebundenen Tuns habe ich für mich auch Testprogramme geschrieben, die nur für mich waren.”

Der vierte Befragte (D) konnte uns schließlich sogar ein Beispiel nennen, nachdem er bekundet hatte, dass er Software als Kunst “toll” findet.

4.3.5 User Interfaces als Kunstwerke

Dieser Abschnitt soll untersuchen, inwieweit das Userinterface und die Ausgaben eines Programmes – also das, was der nicht in Informatik ausgebildete “Ottonormalverbraucher” sieht – als Kunstwerke anzusehen sind und welche Formen diese Kunst annimmt. Da die Richtungen, in die sich diese Kunst entwickelt, aufgrund des hohen kreativen Freiheitsgrades, sehr vielseitig sind, wollen wir, wie schon im Abschnitt 4.3.2 nur anhand einzelner Beispiele aus möglichst verschiedenen Bereichen argumentieren, um daraus eine umfassende und anschauliche Darstellung ohne Anspruch auf Vollständigkeit zu erhalten.

Peripherie

Derzeit spielt sich die Software-Kunst primär in den akustischen und optischen Bereichen ab, da die Peripheriegeräte standardmäßg nur Bild und Ton auszugeben erlauben.

visuell: Um optischen Output zu erzeugen, werden bisher vor allem Monitore benutzt, welche im Normalfall einfach mit Hilfe der “RGB”-Farben arbeiten. Des weiteren können Beamer, Drucker, Plotter oder vllt. auch speziell zum Erschaffen des Kunstwerkes

[34] Quelle: [Trogemann/Viehoff 2005], S. 12.

selbstgebaute Hardware verwendet werden. Für letzteres wäre z. B. ein ein Programm denkbar, das eine Paintball-Waffe Farbbälle auf eine Leinwand schießen lässt.

auditiv: Für den akustischen Output verwendet man bislang natürlich primär Lautsprecher und Kopfhörer (welche auch nur kleine Lautsprecher sind), diesen wird elektronisch mitgeteilt, wie stark und schnell sie ausschwingen sollen, um mit einem luftverdrängenden schwingenden Teller Wellen im hörbaren Bereich zu erzeugen. Das ist derzeit die einzige Methode für akustischen Output und da damit das komplette hörbare Frequenzspektrum abgedeckt ist, ist auch keine weitere Variante nötig. Natürlich sind auch hier wieder Selbstbauten für den Output möglich, wie auch bei allen anderen Sinnen.

olfaktorisch, gustatorisch & haptisch: In diesen drei Bereichen hat sich bisher noch kein Peripheriegerät etabliert, jedoch sind in allen drei Bereichen experimentelle Selbstbauten vorstellbar. Beispielsweise könnte man ein Gerät bauen, dass einem, ähnlich dem RGB-Prinzip, die 4 Geschmäcker zusammenmischt, auch wenn es vielleicht nicht das volle Geschmacksspektrum abdecken wird. Auch wurden schon vereinzelt Peripherie-Geräte aus dieser Kategorie erfolgreich vermarktet (z. B. Joypads mit Vibration), jedoch bis heute keines, das mehr als eine Modeerscheinung zu sein schien.

Formen der Kunst

Die Softwarekunst ist natürlich durch die Grenzen der oben genannten Peripheriegeräte beschränkt, jedoch kombinieren viele Kunstwerke verschiedene Peripherie (z. B. Monitor und Lautsprecher), wodurch sie sich schwer aufgrund ihrer verwendeten Hardware kategorisieren lassen. Des weiteren gehört natürlich die Hardware nicht wirklich zur Software, jedoch setzt die Betrachtung der Softwarekunst natürlich diese Hardware vorraus, weswegen wir sie hier als Teil – oder wenigstens als Hilfsmittel – der Software betrachten wollen.

Aufgrund der Vielfalt von Kunstwerk-User Interfaces wollen wir uns hier wieder auf zwei Beispiele beschränken, um einen Eindruck zu vermitteln, wie die Kunst in diesem Gebiet sich ausdrücken kann, da eine vorgegebene Kategorisierung immer einzelnen Werken nicht gerecht werden würde.

Unser erstes Beispiel haben wir einem der befragten Softwareentwickler (D) zu verdanken. Auf die Frage hin, was er von der Idee Software als zweckfreie Kunst halte, wies er uns auf `www.Balldroppings.com` hin:

> "BallDroppings is an addicting and noisy play-toy. [...] Balls fall from the top of the screen and bounce off the lines you are drawing with the mouse. The balls make a percussive and melodic sound, whose pitch depends on how fast the ball is moving when it hits the line."[35]

Hierbei handelt es sich also um ein Softwaretool ohne wirklichen Zweck. Man kann im zweidimensionalen Raum Linien mit der Maus zeichnen, und fallende Bälle erzeugen verschiedene Geräusche wenn sie von diesen Linien abprallen. Beim Benutzen der Software kann man einige intressante Unterschiede zur Physik der echten Welt feststellen.

[35] Siehe dazu auch: [Nimoy 2004].

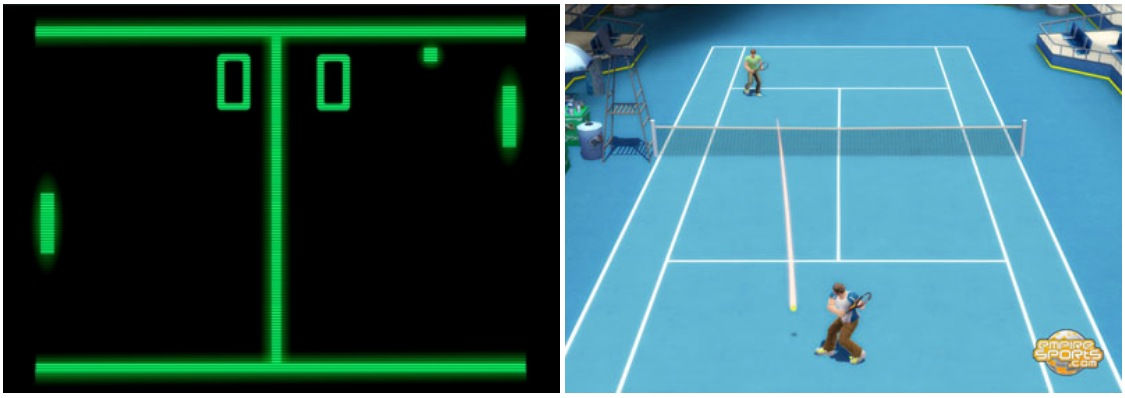

Abbildung 3: links das alte Tennisspiel “Pong”, rechts ein moderneres 3D-Tennisspiel. Das Spielprinzip ist das gleiche, die Ästhetik der Grafik jedoch stark verändert.

Abgesehen von dem Fakt, dass Balldroppings zweidimensional ist, ist das Universum von Balldroppings beispielsweise reibungsfrei. Diese Reibungsfreiheit ist etwas, dass in der echten Welt nicht realisierbar ist, weswegen dieses Kunstwerk *nur* als Software exisieren kann und nie mithilfe herkömlicher Kunstformen nachgebaut werden könnte. Auf die durch Softwarekunst eröffneten neuen Möglichkeiten wollen wir in Abschnitt 4.3.7 näher eingehen.

Unser zweites Beispiel kann kein solches Maß an Zweckfreiheit vorweisen, ist dafür jedoch auch etwas allgemeiner: die Rede ist von Spielen. Abgesehen von deren kommerziellen Zwecken, welche – so könnte man sagen – deren Wert als Kunstwerk negieren, genießen Spiele einen größeren Bekanntheits- und auch Beliebtheitsgrad als zum Beispiel Balldroppings. Im Vergleich zum ersten Beispiel stellen Spiele an den Spieler Herausforderungen, welche der Spieler durch seine interaktive Einflussnahme lösen muss. So könnte man sagen das die Interaktive Komponente bei Spielen – im Gegensatz zu anderer Softwarekunst – unverzichtbar ist.

An der geschichtlichen Entwicklung von Spielen ist zu sehen, dass die Ästhetik eine entscheidende Rolle für den Benutzer zu spielen scheint. Während beim ersten Computerspiel – Pong – nur zwei rechteckförmige Schläger auf und ab bewegt wurden, um einen Quadratischen “Ball” zum Gegner zurück zu schlagen, werden in modernen Tennissimmulationen dreidimensionale Tennisspieler in einem mit Beifall klatschenden Menschen gefüllten Stadion nach links und recht gesteuert (Vgl. auch Abb. 3). Während sich das Spielprinzip und die Herrausforderung an den Spieler also im Grunde kaum geändert hat, spielt die Ästhetik der Grafik offenbar eine entscheidende Rolle. So reizen moderne Spiele oft die Möglichkeiten der beim Spieler gegebenen Hardware bis an ihre Grenzen aus, um möglichst “gut aussehende” Ergebnisse zu erzielen.

4.3.6 Software als Werkzeug für Kunst

In den vorherigen Abschnitten ging es um die Art der Softwarekunst, die Programmier- oder anderer Formen von Fachkenntnissen bedürfen. In diesem Abschnitt, soll es nun um fertige Software gehen, welche dazu genutzt werden kann, um mit ihrer Hilfe Kunstwerke zu schaffen. Dabei sollen keine Programmierkenntnisse beim Benutzer vorrausgesetzt werden, da er lediglich die Benutzeroberfläche des Programmes bedienen muss, um seine Kunstwerke zu erschaffen. Die Software nimmt in diesem Abschnitt also die Rolle des Pinsels ein, mit dem der Künstler sein Bild malt, oder die Rolle des Instruments mit dem er sein Lied spielt. In anderen Worten, die Software ist das *Werkzeug* zum Erschaffen des Werkes.

In Abschnitt 4.3.5 wurde auf die, durch den technischen Stand der Hardware, möglichen Arten der Darstellung der Kunstwerke eingegangen. Will man nun ein generisches Werkzeug schaffen, so sollte man darauf achten, dass es Peripherie für die ausgegebenen Werke gibt. Daher haben sich auch als Werkzeuge bisher nur jene, die Kunst im visuellen oder akustischen Bereich schaffen können, etabliert. Die visuelle Kunst kann man nun wiederrum in zeitabhängig animierte und zeitunabhängig stillstehende unterteilen, was bei der akustischen Kunst nicht möglich ist. Im Gegensatz zu Bildern können Klänge nämlich ausschließlich über einen Zeitraum hinweg erfasst werden und nicht zu einem einzelnen Zeitpunkt. Versucht man einen zeitlich stillstehenden Ton zu erzeugen, so ist dieser nicht hörbar, da akustische Wahrnehmung eine zeitabhängige Schwingung erfodert.

Stillstehende Bilder

Tools und Programme, die sich auf die Erzeugung und Bearbeitung stillstehender Bilder spezialisiert haben, sind in vielen Fällen vorrangig nur Simulationen von Stift, Pinsel, Papier, Farbtopf und Leinwand. Der Benutzer kann auswählen, ob er Stift oder Pinsel benutzen möchte, kann dann auswählen, welche Form und Größe dieser haben soll und welche Farbe er damit auftragen möchte und kann diese dann aufs Bild auftragen, als würde er einen echten Pinsel bzw. Stift benutzen. Wirft man Blicke ins Detail, offenbaren viele Programme jedoch schnell, dass ihre Funktionalität weit über eine einfache "Pinselsimulation" hinaus geht. So können Effekte angewendet werden, welche das ganze Bild in einem anderen Stil erscheinen lassen, Zonen mit bestimmten Farbgebungen durch einfaches Klicken einzeln bearbeitet werden (z. B. kann man den blauen Himmel rot färben, ohne die Bäume im Vordergrund mit dem Pinsel zu berühren). Ein bedeutender Unterschied gegenüber der klassischen Malerei sind wohl auch "Rückgängig"-Funktionen und die Möglichkeit, Farben einfach weg zu radieren oder zu übermalen. Diese Möglichkeit bieten zwar auch einige herkömliche Medien (z. B. kann man Bleistift radieren), jedoch weder alle, noch eines in diesem Umfang.

Eine weitere Form von Tools für stillstehende Bilder, sind jene für die 3D-Modelierung. Diese simulieren oft auch nur – ähnlich den 2D-Programmen – Bildhauerei bzw. Töpferei, auch wenn dies in andere Begrifflichkeiten verpackt sein könnte.

Animierte Filme

Jene Programme, welche sich auf das Erstellen und Bearbeiten von Filmen spezialisiert haben, bieten zum einen Möglichkeiten vorhandene Filme im Nachhinein zu bearbeiten als auch Möglichkeiten, mittels eines relativ geringen Aufwandes neue Filme zu kreieren. Für ersteres werden beispielsweise rückwirkend Filter auf die Filme gelegt, so dass bestimmte Farben besonders hervorstechen oder gedämpft werden. Für zweiteres gibt es auch viele Möglichkeiten. Eine ist die Keyframetechnologie. Bei dieser erstellt der Benutzer einfach zwei Bilder, woraus das Tool dann eine Animation, welche Bild A in Bild B überführt, interpoliert.

Es gibt natürlich auch Programme welche 3D-Bilder (siehe oben) auf die gleiche Weise animieren, oder diese in fertig aufgenommene Filme einbinden können. Auch Filmschnittprogramme, welche dazu dienen, vorhandene Bildfolgen zu zerschneiden und zusammenzufügen, um so einzelne Segemente zu entfernen oder an anderer Stelle erscheinen zu lassen, sind ein wichtiger Bestandteil der Animations- und Filmtools.

Sounds und Melodien

Da die Peripherie aus dem akustischen Bereich das vollständige hörbare Klangspektrum abdecken kann, ist natürlich auch Software vonnöten, die dieses Potential ausschöpfen kann. Und natürlich ist diese Art von Software auch vorhanden. Neben Programmen welche, wie auch in den anderen beiden Kategorien, einfach Filter auf fertige Sounds legen (z. B. um diese "dumpfer" klingen zu lassen), gibt es auch Tonschneidungs- und Erzeugungstools. Erstere funktionieren genau wie die Filmschnittprogramme, nur dass sie auf Tönen statt auf Bildfolgen arbeiten. Zweitere dienen zum Erzeugen von Tönen und Melodien. Dies kann auf völlig unterschiedliche Art und Weise erfolgen. Während einige Programme den Benutzer das zu simulierende Instrument auswählen lassen und nach eingegebener Melodie die Geräusche eines Instrumentes synthetisieren, auf dem diese Melodie gespielt wird, gibt es andererseits zum Beispiel Software-Synthesizer, bei welchen der Benutzer direkt die Form, Amplitude, Frequenz und Modulation der Wellen einstellt, welche am Ende hörbar werden sollen.

Hybride

Viele Programme lassen sich nicht klar in eine der drei Kategorien einteilen, sondern sind Hybride aus diesen Kategorien. So wäre es z. B. umständlich, wenn man beim Schneiden einer Filmszene den Sound und das Bild separat schneiden müsste, weshalb der Film und der Sound stattdessen einfach als Einheit betrachtet und dann beschnitten wird.

4.3.7 Neue Möglichkeiten des künstlerischen Ausdrucks

Beim Sprung von klassischer analoger Kunst auf digitale Kunst wurde weit mehr erreicht, als dass nur die gleiche Kunst auf einem anderen Medium stattfände. Vielmehr erweitert die Software die künstlerischen Möglichkeiten um Weiten. Primär ist diese Änderung wohl in der Filmkunst spürbar, doch auch in anderen Teildiziplinen eröffnen sich neue Perspektiven. Dieser Abschnitt soll genauer darauf eingehen, welche Möglichkeiten sich bieten und wie diese genutzt werden können. Wir wollen auch zeigen, dass diese Möglichkeiten viel mehr durch die Hardwarebeschränkungen eingegrenzt werden, als es

die Software tut und wollen dann auch den Faden weiterspinnen, um zu sehen dass die Softwarekunst bei angenommener vorhandener Hardware dennoch nicht grenzenlos ist. Erst wollen wir jedoch die neuentstandenen Mittel an einigen gegebenen Beispielen untersuchen, um dann etwas allgemeinere Aussagen zu treffen.

Film, die animierte Kunst

Zunächst einmal bietet die neue Kunst natürlich Vereinfachungen während der Produktion von Filmen, während man früher z. B. die Filter schon während des Drehens aufs Objektiv stecken musste, kann man dies nun mittels Filmbearbeitsungsprogrammen leicht im Nachhinein tun. Auch bietet die Software Möglichkeiten, welche den herkömlichen Werkzeugen qualitativ überlegen ist. So müssen Zuschauer sich in Sciencefiction-Filmen keine Raumschiffe aus Pappmaschee, welche an einer Angelsehne vor der Kamera hin und her schaukeln, mehr ansehen, sondern können hochwertig animiertere und auch detailiertere Raumschiffe ansehen, die auch Flugverhalten vorweisen, als würden sie sich im Vakuum des Weltalls bewegen.

Doch das sind nicht wirklich Neuerungen sondern lediglich Feinschliffe schon vorhandener Möglichkeiten. Die wirklichen Neuerungen findet man wohl eher in den physikalisch unmöglichen Ideen. Angenommen wir wollen beispielsweise eine Kugel mitten in einem Raum erscheinen lassen, die erst unsichtbar klein ist und dann immer größer wird. Dies war soweit auch noch in der klassischen Kunst möglich indem man sie selbst in allen Größenstufen selbst "zeichnet", ähnlich wie in Trickfilmen. Soweit wäre es zwar schon ziehmlich aufwändig, scheint jedoch noch machbar. Fordern wir nun jedoch zusätzlich dass die Kugel eine den Raum relektierende metallische Oberfläche haben soll, so versagt die herkömliche Methode um Längen, sofern der "Zeichner" dieser Szene nicht sein halbes Leben mit dem Zeichnen zubringen soll. Scheinbar gibt es nur einen weichen Übergang vom bereits Möglichen zum neu Ermöglichten, wenn man von qualitativen Details absieht. Dennoch kann man bei allen Dingen, die nicht einfach so, wie sie im Film zu sehen sein sollen, vor die Kamera zu bekommen sind – also vor allem bei fiktiven Dingen oder seltenen bzw unerreichbaren Erscheinungen – mittels Software viel ansehlichere Ergebnisse mit deutlich weniger Aufwand erzeugen. Darin liegt wohl die Hauptleistung der Software in Bezug auf die Filmkunst.

Musik und Sound, die hörbare Kunst

Anders als in der Filmkunst war in der Soundtechnik bereits das komplette hörbare Spektrum mittels elektronischer Instrumente, wie Synthesizern u. a. ansteuerbar. Diesen mangelte es jedoch an der Fähigkeit, die Harmonien und die melodischen Tonfolgen analoger Instrumente synthetisieren zu können. Die Softwarekunst gibt Ersteren die Fähigkeit Zweiterer, zumindest bis zu einem gewissen Maß. So kann man beispielweise die Klangstrukturen beim Anspielen verschiedener Töne und Harmonien einer Gitarre analysieren und diese dann so in ein Programm bauen, dass beim Eingeben von Noten genau diese Klänge für die richtige Dauer erzeugt werden. Auch kann man ganze Tonfolgen einfach *speichern*, um sie später – beispielsweise bei einem Live-Auftritt – genauso abspielen zu können. Diesen synthetischen Sounds mangelt es – so sagen manche – jedoch noch an dem "Gefühl" das beim Spielen einer analogen Gitarre vermittelt werden kann.

Grenzen der Hard- und Software

Wie bereits im Abschnitt 4.3.5 aufgezeigt, begrenzt die Peripherie die realisierbare Software-Kunst. So kann man wohl kaum eine Software schreiben, welche Gerichte mit bestimmten Geschmäckern zubereiten soll, wenn es keine Geräte gibt, um dem Benutzer Geschmäcker zu vermitteln. Genauso kann man schlecht dreidimensionale Animationen wirklich in 3D darstellen, solange der visuelle Output auf 2D-Monitoren stattfindet. Versucht man sich diese Grenzen wegzudenken und auch die, die einzig durch die Geschwindigkeit der Hardware gegeben sind, so eröffnet sich einem, was die Software wirklich in der Lage ist zu leisten und was nicht.

So kann Software alles darstellen und erzeugen, was vollständig und eindeutig parametrisierbar ist. Dies trifft vermutlich auf alles, was wir mit unseren 5 Sinnen wahrnehmen können zu und was Software leisten kann, geht sogar noch über die Grenzen dieser Sinne hinaus.

Erwartet man von Software jedoch, dass sie Gefühle so ausdrückt, wie es ein Mensch könnte (wie im Beispiel mit der Gitarre), oder das sie ästhetische Kunst erzeugt, deren Ästhetik in der Gedankenstruktur liegt, so wird man vermutlich enttäuscht. Heutige Software ist auch durch die Turingberechenbarkeit begrenzt und ob abstrakte Größen wie Ästhetik oder Gefühle jenseits dieser Grenzen liegen oder nicht, lässt sich wohl nicht ohne weiteres berechnen.

Man kann also sagen, dass Software sowohl Kunst als auch Ästhetik bereits stark bereichert und auch viele neue Persepektiven eröffnet, jedoch auch noch ebenso viele Grenzen in Sicht sind, die verdeutlichen, dass Software kein “Allheilmittel” ist.

5 Schlusswort

Die ästhetische Dimension der Software-Entwicklung bietet, wie wir im Vortrag gesehen und nun vielleicht noch gelesen haben, viele Möglichkeiten zum Anknüpfen an klassische Software-Entwicklung, Gesellschaft und Gesellschaftswissenschaften, das Wirtschaftssystem und letztlich auch an die Kunst. Insofern – und weil nach unserer Auffassung oft vernachlässigt bzw. nicht reflektiert – kann Ästhetik in der Software-Entwicklung nur interdisziplinär, d.h. unter Nutzung von Methoden und Begriffsinventar verschiedener wissenschaftlicher Disziplinen, untersucht werden. Die vorliegende Arbeit zeigt die dabei auftretenden Probleme – etwa an Hand unterschiedlicher Terminologien.

Die Antworten der befragten Entwickler wiesen auf ein allgemein großes Interesse an Ästhetik hin, wenngleich Ästhetik in der Software-Entwicklung meist wie ein blinder Fleck erscheint.

Der Siegeszug digitaler Medien mag schließlich die Lücke zwischen Ästhetik und Software fürderhin immer mehr schließen. Gleichwohl ist nicht absehbar, ob und wie die Software-Entwicklung für das Wirtschaftssystem in Zukunft stärker an ästhetischen Maßstäben orientiert sein wird.

Literatur

[Adorno 1973] Adorno, Theodor W.: *Theorie der Ästhetik.* Herausgegeben von Gretel Adorno und Rolf Tiedemann. 1. Auflage. Frankfurt am Main: Suhrkamp 1973

[Buß 2003] Buß, Angelika: *Kanonprobleme.* In: Kämper-van den Boogaart, Michael (Hrsg.): Deutsch-Didaktik: Praxishandbuch für die Sekundarstufe I und II. Berlin: Cornelsen, 2003.

[Di Dio 2008] Di Dio, Cinzia / Macaluso, Emiliano / Rizzolatti, Giacomo: *The Golden Beauty: Brain Response to Classical and Renaissance Sculptures*, 2007. Internet: `http://www.plosone.org/article/info:doi%2F10.1371%2Fjournal.pone.0001201` [August 2008]

[Dreher 2005] Dreher, Thomas: *Konzeptuelle Kunst und Software Art: Notationen, Algorithmen und Codes*, 2005. Internet: `http://iasl.uni-muenchen.de/links/NAKS.html` [August 2008]

[Eagleton 1994] Eagleton, Terry: *Ästhetik. Die Geschichte ihrer Ideologie.* Aus dem Engl. von Klaus Laermann 1. Auflage. Stuttgart, Weimar: Metzler, 1994.

[Floyd/Klischewski 1998] Floyd, Christiane / Klischewski, Ralf: *Modellierung – ein Handgriff zur Wirklichkeit. Zur sozialen Konstruktion und Wirksamkeit von Informatik-Modellen.* 1998. Internet: `http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-9/Floyd.ps` [August 2008]

[Gethmann-Siefert 1995] Gethmann-Siefert, Annemarie: *Einführung in die Ästhetik.* 1. Auflage. München: Fink, 1995.

[Hurrelmann 2002] Hurrelmann, Klaus: *Sozialisationstheorie.* 8. Auflage. Weinheim, Basel: Beltz, 2002.

[Janker 2002] Janker, Stefanie: *Das Geheimnis des Schönen. Ein Modell der psychischen und mentalitätsgeschichtlichen Strukturen von „Ästhetik“.* Diss. Bamberg: 2002.

[Kernighan/Plauger 1976] Kernighan, Brian W. / Plauger, P. J.: *Software Tools.* Reading, Massachusetts: Addison-Wesley 1976.

[Kittler 1995] Kittler, Friedrich A.: *Aufschreibesysteme 1800 . 1900.* 3. vollst. überarbeitete Neuauflage. München: Fink 1995.

[Koubek 2008] Koubek, Jochen: *Dimensionen der Software-Entwicklung.* Seminarkonzeption. Präsentation. Berlin 2008.

[Lepold/Böhret 1987] Lepold, Frank / Böhret Peter.: *Der Mensch in der Software-Entwicklung. Eine organisationspsychologisch orientierte Beschreibung von Individuum und Gruppe in der Software-Entwicklung.* Idstein: Schulz-Kirchner 1987.

[Nake 1974] Nake, Frieder: *Ästhetik als Informationsverarbeitung.* 1. Auflage. Wien, New York: Springer, 1974.

[Nimoy 2004] Nimoy, Josh: *www.balldroppings.com*, 2004. Internet: `http://www.balldroppings.com/` [August 2008]

[Paul 2003] Paul, Christiane: *Kunst als öffentliche kulturelle Produktion (Software)*, 2003. Internet: `http://www.aec.at/de/archives/festival_archive/festival_catalogs/festival_artikel.asp?iProjectID=12501` [August 2008]

[Plattner et. al. 2000] Plattner, Hasso et. al.: Dem Wandel voraus. Hasso Plattner im Gespräch mit August-Wilhelm Scheer, Siegfried Wendt und Daniel S. Morrow. 1. Auflage. Bonn: Galileo Press, 2000.

[Purgathofer 2003] Purgathofer, Peter: *designlehren. zur gestaltung interaktiver systeme.* Habil. Wien: 2003.

[Rico 2003] Rico, David F.: *IEEE 12207 Software Life Cycle. Architecture, Phases, Products, Evaluations, Records, Audits, Reviews and Baselines.* 2003. Internet: `http://kur2003.if.itb.ac.id/file/ieee-12207.pdf` [August 2008]

[SAP 2001] SAP AG: BC - SAP Style Guide. Release 4.6C. Walldorf: 2001. Internet: `http://help.sap.com/printdocu/core/Print46c/de/data/pdf/BCDWBERG/BCDWBERG.pdf` [August 2008]

[Stifter 2005] Stifter, Adalbert: *Der Nachsommer.* Ditzingen: Reclam, 2005.

[Thompson 1999] Thompson II, Gary P.: *The Quine Page*, 1999. Internet: http://www.nyx.net/~gthompso/quine.htm [August 2008]

[Trogemann/Viehoff 2005] Trogemann, Georg / Viehoff, Jochen: *CodeArt : eine elementare Einführung in die Programmierung als künstlerische Praktik.* 1. Auflage. Wien, New York: Springer, 2005.

[Trueg 05] Trüg, Florian: *Programmieren als Kunst.* Referat. Präsentation. WS 2005/06 an der Universität Kassel.

[Warnke 2005] Warnke, Martin: *Kunst aus der Maschine – Informationsästhetik, Virtualität und Interaktivität, Digital Communities.* Lüneburg: 2005.

[Willkop 2001] Willkop, Eva-Maria: *Linguistische Analyseverfahren von Texten.* In: Helbig, Gerhard (Hrsg.): Deutsch als Fremdsprache. HSK 19.1, Berlin, New York: de Gruyter, 2001.