Softwareentwicklung als Realitätskonstruktion

Christiane Floyd
TU Berlin
Sekr. FR 5-6
Franklinstr. 28/29
D-1000 Berlin 10

Realität = Gemeinschaft
/v. Foerster 73/

1. Einleitung

Im folgenden will ich eine Sicht der Softwareentwicklung skizzieren, die im wesentlichen mit erkenntnistheoretischen Auffassungen des *Radikalen Konstruktivismus*[1] übereinstimmt. Es handelt sich hier um einen ersten Versuch, die Ergebnisse meiner Forschungstätigkeit der letzten beiden Jahren in schriftlicher Form vorzustellen[2]. Ich möchte sie als Vorschlag für eine weitergehende Diskussion einbringen und entsprechend vorsichtig formulieren.

Mein Anliegen war, Softwareentwicklung zu verstehen. Es ergab sich aus der langjährigen Beschäftigung mit Softwareentwicklungsmethoden in Forschung, Lehre und Praxis[3]. Dabei sind für mich die etablierten Denkschemata der Disziplin Software Engineering als ausschließliche Fundierung der wissenschaftlichen Arbeit fragwürdig geworden.

Das betrifft insbesondere folgende Grundannahmen des Faches: die Sicht der Softwareentwicklung als Produktion von Programmsystemen aufgrund fester Vorgaben, die Trennung zwischen Produktion und Einsatz bzw. Wartung, die Einteilung der Produktion in linear zu durchlaufende Phasen, die dabei fast ausschließliche Verwendung von Zwischenergebnissen in Form von Dokumenten, die Sicht von Methoden als feste Regelwerke, die menschen- und

---

1 Der Kürze halber spreche ich von jetzt ab nur von "Konstruktivismus", meine damit aber durchgehend den radikal konstruktivistischen Ansatz (z.B. /v. Foerster 73/, /v. Glasersfeld 81/, /Maturana, Varela 87/, /Watzlawick 81/, /Schmidt 87/).

2 Meine Forschung wurde von der Stiftung Volkswagenwerk in zweifacher Weise gefördert: durch ein Akademie-Stipendium im Wintersemester 1987/88 für einen Forschungsaufenthalt in Palo Alto zum Thema "Erkenntnistheoretische Grundlagen der Softwareentwicklung" und durch Finanzierung der Tagung "Software Development and Reality Construction" vom 26.-30.9.1988 in Schloß Eringerfeld. Ich möchte der Stiftung hiermit herzlich danken.

3 Diese Arbeit fand seit 1978 an der TU Berlin in einem kooperativ gestalteten Forschungsmilieu statt, und wurde von meinen Mitarbeitern wesentlich mitgetragen. Insbesondere danke ich Reinhard Keil-Slawik, Jürgen Pasch, Fanny-Michaela Reisin und Gerhard Schmidt.

situationsunabhängig eine standardisierte Arbeitsweise festlegen und die einseitige Betonung von Formalisierung unter Wegfall von Kommunikation, Lernen und Evolution[1].

Ich gebe gerne zu, daß kaum ein mir bekannter Autor die genannten Annahmen heute noch ohne Einschränkungen vertritt. Sie werden vielmehr als Idealvorstellungen eingeschätzt, die in der Praxis nur approximiert werden können. Es gibt aber noch wenige Ansätze zu Alternativen für die etablierte Denktradition des Software Engineering[2].

Die Fragwürdigkeit dieser Tradition folgt für mich zum einen aus den eklatanten Widersprüchen zwischen ihren Postulaten und dem, was in Projekten in der Industrie wie auch an der Hochschule tatsächlich stattfindet, obwohl viele dieser Projekte im Sinne der etablierten Tradition angeleitet werden. Mir steht fern zu behaupten, die Produktionssicht der Softwareentwicklung wäre irrelevant, sie scheint mir aber immer nur jeweils stückweise bezogen auf wohldefinierte Teilziele zu greifen, während sie der Softwareentwicklung als Ganzem nicht gerecht wird.

Zum anderen berücksichtigt die etablierte Sicht nicht das Ringen um Qualität[3] bei der Softwareentwicklung. Letztlich bietet sie keine Aufsatzpunkte für eine menschengerechte Systemgestaltung.

Hier scheint eine reichhaltigere Sichtweise notwendig. Das hat mich veranlaßt, nach adäquaten erkenntnistheoretischen Grundlagen für die Softwareentwicklung zu suchen[4]. Sie müssen uns helfen, spezifische, von Gemeinschaften getragene, koordinierte Erkenntnisprozesse zu verstehen, bei denen verschiedene Realitätsbereiche aufeinanderstoßen, abstrakte und gleichzeitig sehr komplexe Ergebnisse erarbeitet werden, die entstehende technische Realität der Software mit der sozialen Realität ihrer Herstellung und Nutzung verwoben ist, und sämtliche Prozesse vor dem Hintergrund gesellschaftlicher Widersprüche stattfinden.

Nach meiner Auffassung geht es bei der Softwareentwicklung primär um eine spezifische Ausprägung von *Design*. Unter Design verstehe ich insgesamt den kreativen Vorgang, in

---

1 Die daraus resultierende Kritik des Software Engineering findet sich in /Floyd 81/, /Floyd, Keil 83/, /Floyd 84/, /Floyd 85a/ und /Floyd 87/.

2 Nach meiner Auffassung ist der tragfähigste alternative Denkansatz der von Peter Naur über Programmierung als "Theoriebildung" (/Naur 84/). Meine eigene Arbeit im Software Engineering wurde durch Peter Naur maßgeblich beeinflußt (siehe auch meine Zusammenfassung seiner Kritik einer formalistischen Programmiermethodik in /Floyd, Mehl, Reisin, Schmidt, Wolf 87/).

3 "Quest for Quality", eine Wortschöpfung von Donald Knuth im Zusammenhang mit seiner Reflexion der Entwicklung des TeX-Systems und der dabei von ihm gemachten Fehler (/Knuth 87/).

4 Meine Suche wurde unmittelbar ausgelöst durch /Floyd 85b/, wo ich mich erstmals mit einem Versuch anderer Autoren, das Konzept autopoietischer Systeme auf Design anzuwenden, auseinandergesetzt habe und dabei auf die Komplementarität von Prozessen und Produkten gestoßen bin.

dem das Problem erschlossen, eine zugehörige Lösung erarbeitet und in menschliche Sinnzusammenhänge eingepaßt wird. Treffend dafür ist die Feststellung von Peter Naur: "Software development is an activity of overall design with an experimental attitude" /NAUR 74/.

In der vorliegenden Arbeit beziehe ich mich zur Fundierung von Design ausschließlich auf erkenntnistheoretische Einsichten, die dem konstruktivistischen Diskurs entstammen (siehe /SCHMIDT 87/[1]). Sie erscheinen mir in besonderer Weise geeignet, um die Softwareentwicklung als Design zu verstehen.

Zunächst will ich in Abschnitt 2 Postulate des etablierten Software Engineering aufzeigen, die ich insgesamt als *Produktionssicht* der Softwareentwicklung bezeichne. Dabei will ich deutlich machen, daß diese Postulate perspektivischer Natur sind. Sie wurden von einem Blickwinkel aus für einen bestimmten Betrachtungsschwerpunkt "erfunden", blenden aber andere aus. Damit sind wir bereits bei einer Schlüsselerkenntnis des Konstruktivismus

In Abschnitt 3 werden wichtige Elemente des Konstruktivismus eingeführt und auf unser Thema bezogen. Für uns sind seine Einsichten von Interesse, die das Entstehen von Wissen in verschiedenen Bereichen betreffen. Unsere Aufgabe ist dann, den spezifischen Typ von Erkenntnisprozessen, die bei der Softwareentwicklung maßgeblich sind, durch sie zu erklären.

In Abschnitt 4 wird ein für die Softwareentwicklung passender Design-Begriff ausgearbeitet. Im Deutschen ist er auf das Wortpaar "Entwurf" und "Gestaltung" abzubilden. Design ist nicht primär an vorweg festgelegte Ziele gebunden, sondern wird durch das Suchen nach Qualität geleitet.

Dann wird in Abschnitt 5 der Design-Raum aufgespannt. Er besteht aus den ineinandergreifenden Realitätsbereichen Anwendung, Methoden und Realisierungsmittel, die im Design konstruiert werden. Im Unterschied zu der Produktionssicht wird hier kein phasenspezifischer, zeitlicher Übergang von einem Realitätsbereich zum anderen angenommen, sondern es handelt sich um ein pulsierendes Gebilde aus immer neu und immer feiner zu treffenden Unterscheidungen, das sozusagen in der Zeit "tanzt".

In Abschnitt 6 werden die beim Design stattfindenden Erkenntnisprozesse als Geflecht von Entscheidungen charakterisiert, die die für Design maßgeblichen Realitätsbereiche verknüpfen. Die Brauchbarkeit einer Design-Entscheidung erweist sich durch ihre

---

[1] Dieses Buch gibt eine ausgezeichnete Einführung in den Konstruktivismus. Es war mir beim Schreiben dieses Papiers sehr hilfreich. Ich habe auch, soweit möglich, die Positionen der Original-Literatur aus einzelnen Beiträge dieses Buchs hier zitiert.

Beurteilung. Wo eine Rückwirkung der Beurteilung auf den Design-Prozess zugelassen wird, kommt es zur Schließung, indem Ergebnisse von Design wieder Grundlage für die Weiterführung von Design werden. Erfolg im Design bedeutet Stabilisierung des Geflechtes von Design-Entscheidungen trotz Revisionen.

Während die bisherige Behandlung auch für Design-Prozesse gilt, die von einzelnen vollzogen werden, wird in Abschnitt 7 die Möglichkeit einer dialogischen Orientierung aufgezeigt, bei der das Ich und Du der Softwareentwicklung in den Grundbeziehungen "Ich entwickle Software mit Dir" und "für Dich" anerkannt, und in dialogisch gestalteter Zusammenarbeit umgesetzt wird. Diese Orientierung bezieht unsere Verantwortung unmittelbar in die technische Arbeit ein.

Abschließend werden Konsequenzen aus dieser Sichtweise für Ausbildung, Projektgestaltung, sowie die Entwicklung von Methoden und Werkzeugen für die Softwareentwicklung aufgezeigt.

Insgesamt ergibt sich eine Sicht von Design als ineinandergreifenden, lebenden Prozessen, die von uns getragen werden, die verkümmern, entarten oder sich entfalten können. Ihre Entfaltung setzt zum einen eine genügende Autonomie des Designs voraus, zum anderen auch die Fähigkeit und Bereitschaft der Beteiligten zur multiperspektivischen Reflexion.

2. Softwareentwicklung als Produktion - eine Sicht und ihre Grenzen

In diesem Abschnitt will ich die für mich maßgebliche Fragestellung umreißen, und meinen Weg zu ihrer Bearbeitung aufzeigen[1]. Die erkenntnistheoretische Fundierung der Softwareentwicklung wird heute von vielen Wissenschaftlern und Praktikern als wichtiges Anliegen anerkannt und von verschiedenen Blickwinkeln aus vorangetrieben[2]. Der bekannteste Beitrag in diese Richtung ist zur Zeit das vieldiskutierte Buch "Understanding Computers and Cognition" /WINOGRAD, FLORES 86/, in dem es insgesamt um das Zusammenspiel von informationstechnischen Systemen mit unserem Denken und kooperativen Arbeiten geht.

In diesem Buch wird ein gangbarer Weg für erkenntnistheoretische Untersuchungen in unserem Bereich abgesteckt. Die etablierte Sicht, so argumentieren die Autoren, erscheint uns nur so lange selbstverständlich, wie wir uns innerhalb der rationalistischen Tradition

---

1 Ich möchte an dieser Stelle meinen Kollegen Albrecht Biedl, Dirk Siefkes und Walter Volpert sowie allen beteiligten Studenten und Studentinnen für die Zusammenarbeit bei einer Reihe von erkenntnistheoretisch ausgerichteten Seminaren an der TU Berlin danken, die für mich eine wichtige Diskussionsplattform darstellten.

2 Das hat sich für mich am deutlichsten bei der Tagung "Software Development and Reality Construction" gezeigt (siehe /Budde, Floyd, Keil-Slawik, Züllighoven 88/ und /GMD 88/; ein Buch über die Ergebnisse der Tagung ist in Vorbereitung).

bewegen, die sie durch ihre Postulate und zugrundeliegenden Annahmen charakterisieren. Diese Tradition, wie jede andere, bringt aber eine "Blindheit" mit sich, indem sie den Blick auf die ihr zugrundeliegenden Annahmen verstellt.

Konkret auf die hier interessierende Softwareentwicklung bezogen, rechtfertigt die rationalistische Tradition folgende Annahmen[1]:

- Es gibt eine vorgegebene Realität "dort draußen", die wir bei der Softwareentwicklung vorfinden. Durch Analyse ihrer Gegebenheiten erhalten wir Anforderungen an die Software.

- Die wesentliche Aufgabe von Softwareentwicklern ist es, ausgehend von dem in der Realität vorgegebenen Problem eine korrekte Lösung in Form eines Programmsystems zu ermitteln.

- Es ist möglich, die Herstellung von Software von ihrem Einsatz zu trennen. Software Engineering behandelt die Herstellung von Software aufgrund fester Vorgaben.

- Grundlage für die Softwareherstellung sind Modelle, die die Realität abbilden. Modelle sollen der Realität genau entsprechen.

- Der gesamte Vorgang ist weitgehend menschenunabhängig. Für ein und das selbe Problem sollten unterschiedliche Entwickler auch die selben Ergebnisse erbringen. Mitarbeiter sollten austauschbar sein.

- Die Kommunikation soll eingeschränkt und über feste Schnittstellen geregelt werden. Die Arbeitsteilung kann nach Bedarf erfolgen. Je nach technischer Machbarkeit können beliebige Anteile der Herstellungsarbeit automatisiert werden.

- Die Verantwortung der Entwickler betrifft – nur – die ordnungsgemäße Herstellung des Produktes nach den Vorgaben. Weitergehende ethische Überlegungen sind von der sachlichen Arbeit getrennt.

Die mit diesen Annahmen verbundene Sicht der Softwareentwicklung ist zweifellos nützlich. Sie hat entscheidend zu den beeindruckenden Fortschritten in der Programmiermethodik, zu kontrollierbaren Modellen für die Abwicklung von Projekten und darauf aufbauender Entwicklung von Werkzeugen beigetragen. Sie gestattet es im Vorfeld, wichtige Aspekte der

---

[1] Winograd und Flores konkretisieren die "Rationalistische Tradition", indem sie aus ihr Annahmen bezogen auf verschiedene Gegenstandsbereiche ableiten. Eigentlich geht es nicht nur um die rationalistische Tradition als Lehre über unser Denken, sondern auch über die realistische Tradition als Auffassung über die Wirklichkeit.

Softwareentwicklung zu verstehen oder im Nachhinein mehr oder weniger abgeschlossene Projekte aufzuarbeiten (siehe auch /PARNAS, CLEMENTS 85/).

Sie ist aber nicht brauchbar, um die in der jeweiligen Situation tatsächlich ablaufenden Prozesse der Softwareentwicklung zu verstehen, die insgesamt das Zustandekommen von Einsichten in die Funktionalität, Realisierung und Nutzungsmöglichkeiten von Programmen betreffen. Insbesondere führt uns die Produktionsssicht irre, idem sie uns nahelegt, daß wir bei der Softwareentwicklung von festen Gegebenheiten ausgehen können (müssen), und daraus (im Idealfall nach festen Regeln) ein Programmsystem ableiten können.

Die Produktionssicht beleuchtet einen wichtigen Betrachtungsschwerpunkt der Softwareentwicklung. Sie verstellt andere. Nach meiner Auffassung verstellt sie die Sicht auf Design. Die übergreifende Natur und damit die Schlüsselstellung des Design wird auch von anderen Autoren gesehen. So ist der Untertitel des zitierten Buchs von Winograd und Flores: "A New Foundation for Design".

Durch das Infragestellen der rationalistischenTradition entsteht zunächst ein Leerraum, den unterschiedliche erkenntnistheoretische Denkschulen ausfüllen könnten. Beim Aufzeigen ihrer neuen Fundierung beziehen sich Winograd und Flores insgesamt auf eine Mischung von Gesichtspunkten, die sie der Hermeneutik (nach Heidegger und Gadamer), dem biologisch fundierten Konstruktivismus (nach Maturana), und der Sprachphilosophie (nach Searle) entnehmen. Dies erscheint mir unbefriedigend.

Auch ich habe ansatzweise die Relevanz anderer Denkschulen ausgelotet, die derzeit zur Fundierung der Softwareentwicklung diskutiert werden[1]. Ihre gemeinsame Berücksichtigung überzeugt mich jedoch nicht, solange nicht geklärt ist, wie diese ganz unterschiedlichen, teils komplementären, teils konträren Traditionen zusammengebracht werden können, ohne daß daraus ein philosophischer Eintopf mit fadem Nachgeschmack entsteht.

Im Gegensatz zu dieser Herangehensweise beziehe ich mich im folgenden nur auf konstruktivistische Auffassungen. Sie passen in hervorragender Weise zu dem von mir gewählten Betrachtungsschwerpunkt Design.

Ferner halte ich sie auch auf einer anderen Ebene als Bezugsrahmen für eine eventuelle spätere Zusammenführung von Einzelbehandlungen anderer Betrachtungsschwerpunkte der Softwareentwicklung im Sinne unterschiedlicher Denkschulen für wünschenswert.

---

[1] Der skandinavische Autor Pelle Ehn stellt gar als theoretische Fundierung drei Design-Philosophien in einem Kapitel zusammen: eine Heidegger'sche, eine Wittgenstein'sche und eine Marx'sche (/Ehn 88/), deren Beziehung zueinander er vorsichtshalber offen läßt.

3. Einstieg in den konstruktivistischen Diskurs

Die Beschäftigung mit konstruktivistischen Ansätzen ist nicht einfach. Sie erfordern ein sehr weitgehendes Umdenken, das ich nur schrittweise vollziehen kann. Ferner ist die verfügbare Original-Literatur stückhaft und heterogen, und behandelt vorwiegend Gegenstandsbereiche, die mir fern liegen. Dies hängt damit zusammen, daß es sich hier um eine Metatheorie über das Zustandekommen von Erkenntnis handelt, deren Einsichten unter anderem aus der Biologie und der Entwicklungspsychologie gewonnen , und von dort aus in den Bereich des Sozialen übertragen worden sind. Dies nachzuvollziehen, ist nicht unser Anliegen. Auch können wir hier nicht die Grenzen der Übertragbarkeit ausloten.

Da es keine klar ausformulierte Gesamtposition gibt, sondern eine Vielfalt von im Einzelnen durchaus voneinader abweichenden Auffassungen, die durch einen gemeinsamen Geist (nach /BATESON 82/) gekennzeichnet sind, kann man zu Recht von einem konstruktivistischen "Diskurs" sprechen (/SCHMIDT 87/). Dabei sind einzelne Autoren in ihre fachwissenschaftliche Diskussion eingebunden, dringen aber über das "Was" des jeweils behandelten Gegenstandes hinaus zum "Wie" der dabei zustande kommenden Erkenntnis vor, wobei sie in unterschiedlichen Disziplinen verwandte Muster vorfinden.

Es übersteigt meine Möglichkeiten bei weitem, im Rahmen dieses Papiers eine adäquate Einführung in den Konstruktivismus zu geben. Trotzdem will ich versuchen, seine wichtigsten Eckpunkte wenigstens zu benennen, um mich anschließend auf sie beziehen zu können.

Unsere Fähigkeiten zur Erkenntnis sind nach konstruktivistischer Auffassung letztlich in der biologischen Natur des Menschen und seiner Koevolution mit allen Lebewesen begründet. Bateson, ein wichtiger Wegbereiter des Konstruktivismus, nimmt eine Wesensgleichheit von Geist und Evolution an (/BATESON 82/). Maturana setzt das Zustandekommen von Erkenntnis (Kognition) mit Leben gleich (/MATURANA, VARELA 87/).

Geist in diesem Sinne zeichnet nicht nur den einzelnen Menschen aus, sondern wird auch in anderen lebenden Systemen angetroffen. Geist ist auch immer auf etwas bezogen. So kennzeichnet Geist auch die Art des Zusammenwirkens von Gemeinschaften bezogen auf gemeinsame Anliegen. Dabei kommen tiefergehende Einsichten durch differenzierte Gegenüberstellung und Koordination von Einzelbeiträgen zustande.

Im Konstruktivismus werden Erkenntnistheorie und Ontologie getrennt (/v. GLASERSFELD 81/). Der Konstruktivismus lehrt, daß unsere Erkenntnis durch Konstruktion zustandekommt, er macht damit keine Aussage über das Seiende. Bei der Erkenntnis geht es nicht um

Abbilder, die einer vorgegebenen Realität entsprechen, wir konstruieren vielmehr unsere Erkenntnisse so, daß sie in für uns brauchbarer Weise passen.

Wesentlich ist dabei die Einführung des Beobachters. Erkenntnis ist prinzipiell an einen Beobachter gebunden (siehe z.B. /MATURANA, VARELA 87/). Beobachter können nur das erkennen, wozu sie in der Lage sind. Die Verständigung zwischen Beobachtern findet in konsensuellen Bereichen statt, die beiden zugänglich sind.

Damit hängt auch das Konzept von Perspektiven zusammen. Eine Perspektive ist die Gesamtheit von Annahmen über relevante Aspekte eines interessierenden Gegenstandsbereichs aus einem gemeinsamen Blickwinkel. Perspektiven sind nicht an Personen gebunden. Eine Person nimmt zeitlich verschoben unterschiedliche Perspektiven ein. Zwischen zwei oder mehreren bilden sich gemeinsame Perspektiven (/PASK 75/, siehe auch /BRÅTEN 78/).

Perspektivität ist prinzipiell mit Blindheit verbunden. Ich sehe nicht, was ich aus meiner Perspektive nicht sehen kann. Die Blindheit kann niemals ausgeschaltet werden. Voraussetzung für das Entstehen von tieferen Einsichten ist Selbstreferenz (s.u.) und die Interaktion (Kreuzung) von Perspektiven.

Konstituierend für Erkenntnis ist das Treffen von Unterscheidungen und Benennungen (/SPENCER BROWN 69/). Komplexe Erkenntnisprozesse bestehen in Geflechten von ineinandergreifenden Unterscheidungen und Wiederzusammenführungen, die sich auf jeweils unterschiedliche Perspektiven beziehen (/PASK 75/).

Ein Schlüsselkonzept ist die Selbstreferenz. Während sie in der Logik zu Paradoxien führt, ist sie grundlegend für das Verständnis des Lebendigen. Selbstreferenz erfordert operationale Geschlossenheit von Systemen. Das bedeutet zunächst nur, daß die Ergebnisse einer Operation wieder Elemente des Systems sind. Es liefert die Voraussetzung zur rekursiven Anwendung von Operationen.

Bei operational geschlossenen und energetisch offenen Systemen ist das Verhalten durch rekursive Kopplung von Operationen bestimmt, wobei mehrere Betrachtungsebenen ineinandergreifen. Das Verhalten von Systemen stabilisiert sich anhand von Eigenwerten, die zu Eigenverhalten Anlaß geben. Dadurch kommt es zur Selbstorganisation, die zum Zustandkommen höherer Ordnungen in Systemen führt ("Ordnung durch Störung"-Prinzip /v. FOERSTER 60/).

Lebende Systeme sind dadurch charakterisiert, daß sie im Verlauf ihres Bestehens ihre Organisation kontinuierlich selbst reproduzieren (Autopoiese nach /MATURANA,

VARELA, URIBE 74/). Autopoiese findet in einem Medium statt. Das autopoietische System und das Medium bedingen sich gegenseitig[1].

Selbstreferenz spielt auch eine zentrale Rolle beim Zustandekommen von Erkenntnis. Perspektivische Blindheit kann durch Selbstreferenz überwunden werden ("Wenn ich sehe, daß ich blind bin, kann ich sehen"[2]). Selbstreferenz ist mathematischer Behandlung zugänglich (/VARELA 75/, /VARELA 87/).

Ein wesentlicher und immer wieder betonter Aspekt des Konstruktivismus ist, daß Ethik von der Betrachtung von Erkenntnis und Handeln niemals losgelöst werden kann (z.B. /v. FOERSTER 73/, /v. GLASERSFELD 84/, /MATURANA, VARELA 87/). Das geschieht nicht durch explizite Angabe von Normen darüber, was man tun soll, sondern das Ethische ist von vornherein mit "eingewoben", da Erkenntnis und Handeln immer auf mich bezogen sind. In /v. FOERSTER 73/ findet sich der ethische Imperativ: Handle stets so, daß die Anzahl der Wahlmöglichkeiten größer werden[3].

Den Anderen anzuerkennen, erfordert nach /v. FOERSTER 73/ eine Entscheidung. Sie veranlaßt uns aus dem Monolog herauszutreten und in den Dialog einzutreten. Dialog bedeutet, die Perspektive des Anderen anzunehmen. Schließung erfolgt dann über den Anderen ("sich selbst mit den Augen des Anderen sehen" /v. FOERSTER 87/).

Dieser Schritt zum Dialogischen[4] ist von ausschlaggebender Bedeutung bei der Umsetzung von konstruktivistischen Vorstellungen im Umgang mit anderen. Er führt zu der als Motto für das vorliegende Papier verwendeten Sicht "Realität = Gemeinschaft" (/v. FOERSTER 73/). Nach /BRÅTEN 88/ ist unsere Erkenntnisfähigkeit insgesamt dialogisch angelegt.

Ich weiß nicht, ob ich Ihnen bei dieser rudimentären und naturgemäß lückenhaften Parforce-Tour durch den Konstruktivismus ein bißchen von der Freude vermitteln konnte, die ich bei seiner allmählichen Erschließung erlebt habe. Ich fühle mich mit diesem Gedankengut wohl. Es paßt zu meinen eigenen Erfahrungen im Alltagsleben und in der Berufspraxis und scheint mir weitreichende Möglichkeiten zu einer wünschenswerten Gestaltung unseres Zusammenlebens aufzuzeigen.

---

1 Ich vermeide im folgenden eine Stellungnahme darüber, wie sich dieses Konzept in brauchbarer Weise auf Soziale Systeme übertragen läßt (siehe dazu die scharf kontrastierten Theorieentwürfe in /Maturana, Varela 87/, /Luhmann 87/ und /Hejl 87/).

2 V. Foerster bezieht sich immer wieder auf eine für ihn prägende Erfahrung aus der therapeutischen Tätigkeit von Viktor Frankl (/v. Foerster 87/).

3 Dies läßt sich nach meiner Auffassung unmittelbar als Leitfaden zur Gestaltung von Softwareentwicklungsprojekten sowie von computergestützten Systemen anwenden.

4 Der Begriff wird hier im Sinne von Buber verwendet (siehe /Buber 84/).

Mein eigentliches Thema beginnt dort, wo wir erkennen, daß die Sicht der Softwareentwicklung als Produktion erfunden ist. So erweist sich die vorherrschende Auffassung über den Gegenstand der Disziplin Software Engineering als konstruierte Realität. Sie ist brauchbar, um bestimmte Aspekte der Softwareentwicklung zu verstehen, und versagt bei anderen. Daher ist es wichtig, ihr andere Sichten gegenüberzustellen.

4. Softwareentwicklung als Design

Zunächst ist zu klären, was ich hier unter Design verstehen will. Es wird nicht ausreichen, einen vorgegebenen Begriff von Design zu verwenden. Vielmehr wird es mir darum gehen, den Begriff Design passend zu den für mich maßgeblichen Anliegen zu konstruieren. Das heißt: ich möchte mit Ihnen gemeinsam eine Reihe von Unterscheidungen treffen, um Design in der Softwareentwicklung brauchbar zu charakterisieren.

Mit Design meinen wir eine spezielle Art von Erkenntnisprozessen, die auf machbare und wünschenswerte Ergebnisse in einem interessierenden Bereich ausgerichtet sind. Die interessierenden Bereiche können ganz unterschiedlich sein. Wir sprechen in der Regel nur dann von Design, wenn es Anliegen gibt, die man erfüllen möchte, begrenzte Ressourcen, die zur Verfügung stehen, und verschiedene Möglichkeiten zur Realisierung.

Design in der Softwareentwicklung ist spezifisch und unterliegt besonderen Bedingungen. Software ist durch ein Zusammenspiel mehrerer unüblicher Eigenschaften gekennzeichnet, die sowohl die Art des Produktes (siehe z.B. /PARNAS 86/ und /KEIL-SLAWIK 88/) als auch seine Einbettung in menschliche Sinnzusammenhänge (z.B. /EHN 88/) betreffen. Software weist eine ungeheure Komplexität auf, erfordert also ebenso komplexe Herstellungprozesse. Sie besteht aus einem einheitlichen, abstrakten Baustoff, ist daher beliebig formbar und prinzipiell uneingeschränkt revisionsfähig. Sie muß maschinell verarbeitbar, das heißt bis ins Einzelne vollständig, konsistent und formal fehlerfrei sein. Sie ist nicht der sinnlichen Wahrnehmung zugänglich, kann also letztlich nur beim Einsatz beurteilt werden. Sie schafft soziale Kontexte für menschliche Handlungen, die durch die technischen Eigenschaften des Produktes geprägt werden.

Design verknüpft somit verschiedene Welten: die soziale Welt der jeweils maßgeblichen Anwendung, die technische Welt der Realisierungsmittel und die formale Welt der Methoden und Konzepte.

Es ist nicht einfach, die gemeinte Bedeutung auf Deutsch auszudrücken. Um die erforderliche Reichhaltigkeit zu gewährleisten, müssen wir "Design" auf das Paar "Entwurf" und "Gestaltung" abbilden. "Entwurf" allein genügt nicht, weil wir in der Regel Entwurf und Realisierung trennen. Wir entwerfen etwas. Das Ergebnis (der Entwurf) wird anschließend

realisiert. "Gestaltung" meint zwar auch Realisierung, erscheint aber im Zusammenhang mit rein technischen Aspekten künstlich.

Verwenden wir statt dessen das englische Wort im Deutschen, so denken wir hauptsächlich an "das Design" als Ergebnis eines Gestaltungsprozesses, das vorwiegend äußere Merkmale eines Gegenstandes betrifft. Das ist jedoch zu eng. Design ist hier prozessual zu verstehen: die Ergebnisse des Design werden in die Gestaltungsprozesse eingeordnet, aus denen sie hervorgehen. Design betrifft nicht nur äußerliche Merkmale, sondern auch die Funktionalität des zu erstellenden Programmsystems sowie seine Einbettung in menschliche Handlungskontexte. Im Design ist auch die Bereitstellung von geeigneten Werkzeugen und Methoden für die jeweils spezifische Softwareentwicklung sowie die Projektgestaltung enthalten.

Ich möchte jetzt Design in der Softwareentwicklung von verschiedenen Seiten her beleuchten. Dabei bitte ich im voraus um Vergebung für die abstrakten Formulierungen. Sie kommen zum einen zustande, weil diese Gedanken für mich neu sind[1]. Zum anderen charakterisiere ich hier aber auch Design so, daß sowohl der von Einzelnen als auch der von Gemeinschaften getragene Design darunter subsumiert werden können. Die Differenzierung zwischen diesen beiden erfolgt erst in Abschnitt 7.

5. Der Design-Raum aus ineinander verschränkten Realitätsbereichen

Die Produktionssicht legt uns nahe, die Softwareentwicklung als Folge von Phasen zu betrachten, die im Idealfall linear zu durchlaufen sind. Sie haben zum Gegenstand, zuerst die Anforderungen der Anwendung zu ermitteln, darauf aufbauend eine Spezifikation des zu erstellenden Systems zu erarbeiten, die festlegt, was gemacht werden soll, ohne zu bestimmen, wie das System arbeiten wird, und daraus das Programm abzuleiten.

Hier treten die für Design maßgeblichen Realitätsbereiche implizit auf:

- die Welt der Anwendungen, deren Anliegen für die Softwareentwicklung maßgeblich sind, und aus der wir Anforderungen an die Software ableiten,
- die Welt der Realisierungsmittel, in unserem Falle informationstechnische Systeme einschließlich vorhandener Software,

[1] Nicht nur mir geht es so: "Die Gedanken hier sind so neu, daß ihre Beschreibung noch nicht durch ständige und wiederholte Bemühungen die Glätte eines von Wind, Wasser und Sand polierten Kieselsteins besitzt" (/v. Foerster 87/).

- die Welt der Methoden und Konzepte, die wir wie Landkarten verwenden, um uns bei der Verknüpfung von Anliegen mit Realisierungsmitteln zurecht zu finden.

Im Phasenmodell wird genau ein Weg durch diese Welten aufgezeigt: von den festen Anforderungen nach festgelegten Methoden zur Realisierung auf einem vorgegebenen System. Dieser Weg ist im Idealfall einmal bezogen auf das gesamte Produkt Software zu durchlaufen.

Die Design-Sicht bringt hier ein Umdenken mit sich. Der zeitliche Fortschritt ist von den Realitätsbereichen zu trennen. Sie werden nicht zeitlich nacheinander bearbeitet, sondern sie sind zu jedem Zeitpunkt gegenwärtig und verknüpft. Ferner ist kein Realitätsbereich vorgegeben, sondern sie werden im Design konstruiert. Das bedeutet:

- Wir analysieren nicht Anforderungen, sondern wir konstruieren sie aus unserer Perspektive.(siehe auch /Reisin 88/) Diese wird bestimmt durch unsere eigenen Prioritäten und Werte, durch die von uns als Landkarte verwendeten Methoden, und durch unsere Interaktion mit anderen, die Anforderungen aus ihrer Sicht konstruieren. Anforderungen sind perspektivisch. Sie reflektieren meist Differenzen der Perspektiven und unterliegen zeitlichem Wandel.

- Wir wenden nicht fest vorgegebene Methoden, sondern wir konstruieren sie aufgrund der Gegebenheiten. Methoden als solche gibt es nicht, es geht vielmehr stets um Prozesse der situationsbedingten Methodenentwicklung und -anwendung. Wir wählen Methoden und passen sie an. Letztlich entwickeln wir im Verlaufe von Design unsere eigenen Methoden.

- Wir beziehen uns nicht auf feste Realisierungsmittel, die erst spät und im Detail bei Entscheidungen zum Tragen kommen. Vielmehr konstruieren wir den sinnvollen Einsatz von Realisierungsmitteln durch Erproben, Auswahl oder Ergänzung dessen, was verfügbar ist.

Die Annahme eines vorweg definierten Wegs durch diese Welten ist irreführend. Dieser Weg wäre nur dann gangbar, wenn alle relevanten Entscheidungen bereits getroffen wären. Dann aber kann kein Design stattfinden.

6. Design als Geflecht von Entscheidungen

Design ist *in Anliegen verankert*. Sie geben Anlaß zum Setzen von Zielen, die mithilfe von bestimmten Mitteln erreicht werden sollen. Mit der Differenzierung zwischen Anliegen und Zielen will ich hier von vornherein die mögliche Diskrepanz zwischen dem,was vorgeblich

erreicht werden soll und dem, was sich als wünschenswert erweist, einbeziehen[1]. Design wird im Hinblick auf gesetzte Ziele ins Leben gerufen, wobei der Ausgangspunkt ein bereits getroffenes Geflecht von Entscheidungen ist, die die für relevant gehaltenen Anliegen im Hinblick auf zu erreichende Ziele mit vorläufig anvisierten Realisierungsmitteln verknüpfen.

Dennoch setzt Design nicht auf einer festen Grundlage auf, und wird auch nicht durch vorgegebene Ziele determiniert. Die Anliegen können sich während des Design-Prozesses wandeln. Die Realisierungsmittel können sich als nicht ausreichend erweisen. Die vorgegebenen Ziele können als irreführend oder nicht mehr gültig erkannt werden. In diesem Sinne kann Design als *zielfrei* angesehen werden. Design schafft sich seine eigenen Grundlagen und setzt sich seine eigenen Ziele.

Design erfordert ein Zusammenspiel unterschiedlicher Fähigkeiten: neben der Beherrschung von Methoden sind dies ein Gespür für das Potential der Realisierungsmittel und Sensitivität für die sich wandelnden Anliegen der Anwendung.

Design setzt einen (Spiel)raum mit *Wahlmöglichkeiten* voraus und den (Frei)raum für spielerisches Vorgehen zum Ausloten dieser Möglichkeiten. Das bedeutet: Design kann nur dort zustandekommen, wo ein ausreichendes Repertoire an Möglichkeiten besteht, um relevante Unterscheidungen zu treffen. Design erfordert Autonomie, um eine echte Auswahl treffen zu können.

Design besteht aus einem Geflecht von *Designentscheidungen*, die in ihrer Gesamtheit einen Lösungsvorschlag ausmachen[2]. Sie verknüpfen Anliegen mit Mitteln im Hinblick auf das Erreichen von jeweils gültigen Zielen. Dabei werden komplexe Strukturen von miteinander verwobenen Entscheidungen aufgebaut. Sie müssen in sich kohärent, und insgesamt wünschenswert sein. Ihr Zustandekommen ist *für den individuellen Design-Prozess spezifisch*, es ist nicht vom vorgegebenen Problem determiniert. Vielmehr wird auch das Problem im Design erschlossen. Design ist durch die Perspektive seiner Träger, und durch die ihnen auferlegten Vorgaben bestimmt.

Was wünschenswert ist, orientiert sich an mehreren Gesichtspunkten, die häufig zu gegenläufigen führen: ob die Entscheidungen zu den Anliegen passen, ob das Geflecht von Entscheidungen alle als wesentlich erachteten Elemente des Problems überdeckt, ob die

---

1 Mit Zielen ist nach konstruktivistischer Sicht vorsichtig umzugehen. /V. Glasersfeld 82/ spricht von "ehrlichen Zielen", die sich aus der Aufrechterhaltung der autopoietischen Organisation ableiten lassen. Ähnlich sind hier Anliegen gemeint.

2 Nicht alle erforderlichen Designentscheidungen werden bewußt getroffen. Häufig zeigt erst die Beurteilung des Lösungsvorschlags, welche Entscheidungen erforderlich wären, und welche Konsequenzen nicht getroffene Entscheidungen implizieren. Die Bedeutung von expliziten Entwurfsentscheidungen wurde besonders von /Parnas 72/ hervorgehoben. Allerdings betrachtet er nur Entwurfsentscheidungen als Grundlage für die Modularisierung. Zum Design gehört natürlich eine Fülle von weiteren Entscheidungen.

verfügbaren Mittel sinnvoll eingesetzt werden, ob die gesetzten Ziele erreicht werden. Diese Unterscheidungen werden durch einen Beobachter getroffen.

Design beruht somit auf einer Fülle von aufeinander Bezug nehmenden Unterscheidungen darüber, was "gut" (wünschenswert) ist. Dabei ergibt sich ein Wechselspiel zwischen Lösungsvorschlägen und ihrer Beurteilung. Was "gut" ist, erweist sich im Prozess dadurch, daß die Beteiligten es für "gut" halten. Unterscheidungskriterien ergeben sich aus den für den Design maßgeblichen Anliegen.

Design kann sich nur dann voll entfalten, wenn bereits getroffene Entscheidungen aufgrund ihrer Beurteilung revidiert werden können; das heißt, wenn die Ergebnisse von Design selbst wieder zum Ausgangspunkt für Design werden. Dadurch kommt *Schließung* im Design zustande.

Das Treffen von Entscheidungen, die Beurteilung und die Schließung finden, ineinander verschränkt, auf verschiedenen Ebenen statt: beim Einzelnen informell, beim Erarbeiten und Überprüfen eines Lösungsvorschlages, bei gemeinsamer kritischer Würdigung, beim Umsetzen einer Entscheidung in die Realisierung, beim Testen, beim Einsatz.

Schließung erfordert, Fehler zuzugeben und daraus zu lernen (/KNUTH 87/), konstruktive Kritik zu geben und zu nehmen, von fehlgesetzten Zielen abzugehen und sich wandelnde Anliegen anzuerkennen. Schließung bedeutet Weiterführung des Design.

Design ist insgesamt erfolgreich, wenn sich das Geflecht von Designentscheidungen im Verlauf von Revisionen stabilisiert, das heißt, wenn es Beurteilungen standhält, und trotz sich wandelnder Anliegen von den Beteiligten als "gut" anerkannt wird.

Design ist somit immer *multiperspektivisch*, auch wenn er von einzelnen getragen wird. Dies ergibt sich aus der Verknüpfung von Anliegen, Realisierungsmitteln und Methoden, aus unterschiedlichen Beurteilungskriterien, sowie aus dem Wechselspiel zwischen Entwurf, Realisierung und Benutzung, das maßgeblich für die Schließung ist. Design erfordert multiperspektivische Reflexion.

7. Dialogische Orientierung im Design

In diesem Abschnitt geht es um Softwareentwicklung mit anderen und für andere, das ist der Normalfall für gesellschaftlich wirksame Softwareentwicklung. Sie soll hier als *potentiell dialogisch* angesehen werden. Das Ich und Du der Softwareentwicklung verbirgt sich in Grundbeziehungen wie "Ich entwickle Software mit Dir" und "Ich entwickle Software für Dich".Wenn Softwareentwicklung potentiell eine dialogische Aktivität ist, so können wir dies

anerkennen und uns darauf orientieren. Wir können uns für das DU entscheiden und ihm Raum geben. Dann findet das Wesentliche zwischen mir und dem anderen ab, wir entwickeln *gemeinsam*. Wir suchen nach Denkweisen, um unsere gemeinsame Wirklichkeit zu verstehen und nach Arbeitsformen, um zu ihrer vollen Entfaltung beizutragen.

Angesichts der vorherrschenden Praxis der Softwareentwicklung mag dies absurd klingen.Wie wir wissen, wird Softwareentwicklung im großen Ausmaß als Instrument zum Ausüben von Macht und Kontrolle benutzt, Softwareprojekte werden bürokratisch angeleitet und über Werkzeuge gesteuert, Teamarbeit ist durch Rivalitäten und Aneinander-Vorbeiarbeiten gekennzeichnet. Dies nicht zu sehen, wäre tatsächlich absurd Wir sind aber nicht gezwungen, dies als unabänderlich anzusehen.

Ganz im Gegenteil halte ich tiefgehende Veränderungen der Praxis im Hinblick auf Qualität und auf menschengerechte Systemgestaltung für dringend geboten. Wir haben deshalb an der TU Berlin schon seit mehreren Jahren an einer Reorientierung der Softwareentwicklung auf menschengerechte Systemgestaltung gearbeitet (/FLOYD 86/), die zu Projektlehrveranstaltungen, Methodenentwicklung, Pilotprojekten und interdisziplinärer Zusammenarbeit mit anderen Wissenschaftlern geführt haben.

Ich halte diese Reorientierung nicht nur aus "humanitären Gründen" erforderlich - so als ob die sachliche Arbeit ohne Berücksichtigung des Anderen mindestens ebenso gut geleistet werden könnte. Vielmehr kann sich Design in Gemeinschaften nach meiner Auffassung ohne diese Reorientierung nicht entfalten.

Im folgenden will ich die für unsere Erfahrungen maßgeblichen Denkansätze und Arbeitsformen in der Sprechweise der Konstruktivisten ausdrücken.

Ich gehe dabei davon aus, daß Gruppenarbeit bei der Softwareentwicklung als ineinander verwobenes Geflecht der oben genannten Grundbeziehungen angesehen werden kann, und spreche daher von einer dialogischen Orientierung[1]. Das bedeutet nach /BUBER 84/, den anderen anzunehmen, und nicht zu instrumentalisieren. "Dialogische Orientierung im Design" ist ein Weg, um die Zusammenarbeit mit anderen Entwicklern und mit Benutzern gemeinsam zu charakterisieren. Nach unseren Erfahrungen liegen solche Gemeinsamkeiten vor. Wir müssen aber auch zwischen den beiden Fällen differenzieren können. Wir unterscheiden dann zwischen

---

[1] Diese Sichtweise habe ich von Stein Bråten übernommen, und halte sie für eine wichtige Bereicherung. Falls sie Ihnen nicht natürlich scheint, wird es für die praktische Arbeit wenig ändern, dabei einfach an kooperative Softwareentwicklung zu denken.

- dialogischem Entwurf, damit meine ich das mit anderen Entwicklern gemeinsame Erarbeiten eines Lösungsvorschlages (siehe auch /PASCH 89/), und

- dialogischer Gestaltung, damit meine ich das mit Benutzern gemeinsame Schaffen von computergestützten Handlunskontexten (siehe auch /REISIN, SCHMIDT 89/).

Im Dialogischen Design geht es darum, daß ein wünschenswerter Lösungsvorschlag zwischen mir und anderen entsteht, daß Designentscheidungen gemeinsam getroffen werden und Schließung unter Berücksichtigung der Perspektiven aller Beteiligten erfolgt. Das heißt: anstatt mein Modell nach meinen Beurteilungskriterien zu entwickeln, diese nach Möglichkeit zu verobjektivieren und durchzusetzen, gilt es, mich den Perspektiven der anderen zu öffnen. Anstelle eines von mir verwalteten Modellmonopols (/BRÅTEN 73/), dem sich die anderen anpassen müssen, geht es darum, die Perspektiven aller aufzugreifen, und in Interaktion zu bringen.

Dies wird von Methoden nur wenig unterstützt. Die meisten mir bekannten Methoden sind monologisch (genau genommen postulieren sie eine Pseudo-Objektivität und erkennen die Perspektivität des Designers nicht an[1]). In einem dialogischen Design müssen wir davon ausgehen, daß jeder ausgearbeitete Beitrag vorläufigen Charakter hat, daß zusammenarbeitende Designer unterschiedliche Erwartungen an den gesamten Prozeß und unterschiedliche Prioritäten und Beurteilungskriterien mitbringen. In einem dialogischen Design müssen wir auch bestehende Konflikte anerkennen und gemeinsam bewältigen.

Für Design im Dialog muß es gelingen, diese Perspektiven so zu vernetzen, daß das im Design entstehende Geflecht von Entscheidungen gemeinsam getragen wird. Das bedeutet, in sinnvoller Weise zwischen individueller und gemeinsamer Arbeit zu alternieren, konstruktive Kritik zu geben und zu nehmen, die Konsequenzen von vorgeschlagenen Designentscheidungen auszuloten, Ergebnisse multiperspektivisch zu bewerten und gemeinsam zu revidieren, so daß allmählich ein gemeinsam getragener Lösungsvorschlag sich stabilisiert.

Die Voraussetzung für Design im Dialog ist Vertrauen. Es kann nur dort entstehen, wo die Interessen der Beteiligten berücksichtigt werden, und verlangt von allen Beteilgten, besonders vom Projektleiter, die Bereitschaft zur Schaffung und Aufrechterhaltung eines sozial stützenden Milieus.

Dialogische Gruppenarbeit kann von keinen impliziten Voraussetzungen ausgehen, sondern muß die Grundlagen für die gemeinsame Arbeit selbst legen. Das bedeutet gemeinsame Etablierung des Projektes, der Übernahme von Verantwortungen, der Aufgabenteilung und

---

[1] Eine Ausnahme bildet SADT mit dem Blickwinkel-Konzept und dem Autor-Kritiker-Zyklus (siehe meine Bewertung in /Floyd 84/). Hier gibt es aber kein Verfahren, die unterschiedlichen Perspektiven zur Interaktion zu bringen.

Synchronisation, der Konventionen und Standards für die gemeinsame Arbeit. Es bedeutet auch gemeinsame Erarbeitung einer für das Projekt maßgeblichen Sicht der Grundlagendokumente und der jeweils gültigen Anliegen, Prioritäten und Beurteilungskriterien.

Design im Dialog erfordert die bewußte Bildung einer gemeinsamen Projektsprache, die die maßgeblichen Realitätsbereiche in einer für alle nachvollziehbaren Weise verknüpft.

Gemeinsame Ziele müssen im laufenden Prozeß aufgrund der jeweiligen Situation gesetzt und revidiert werden.

Gemeinsame Arbeit erfordert gemeinsam zugängliche und aufrecht erhaltene externe Stützen wie Projektordner oder Tagebücher, die die geltenden Arbeitsgrundlagen und die gemeinsam getroffenen Entscheidungen festhalten, so daß der Entscheidungsprozeß bei Revisionen nachvollzogen werden kann.

Im Gegensatz zu den Annahmen der Produktionssicht erfordert Design im Dialog reichhaltige Kommunikation zwischen allen Beteiligten. Information muß kontinuierlich gesammelt und gestreut werden, neue Blickwinkel eingebracht und aus immer wieder neuen Perspektiven beurteilt werden.

Die Arbeitsteilung muß stets vor dem Hintergrund einer gemeinsam getragenen Gesamtsicht erfolgen, getrennt erarbeitete Ergebnisse müssen gemeinsam überprüft werden. Technisch kann dieser Prozeß durch Prototyping unterstützt werden, wobei gemeinsames Lernen anhand der realisierten Vorversionen im Vordergrund steht.

Insgesamt orientieren sich die beschriebenen Maßnahmen daran, daß sich im Projekt Geist entfaltet, und eine gemeinsam getragene Perspektive zustandekommt.

8. Schlußbemerkungen

Obwohl ich hier nur eine Skizze von Softwareentwicklung als Design geben konnte, möchte ich noch aufzeigen, was für Konsequenzen sich ergeben, wenn wir diese Sicht anerkennen und ihr Raum geben wollen. Sie bedeutet eine bewußte Orientierung für unsere wissenschaftliche und praktische Arbeit.

Bei der Ausbildung wird sie uns nahelegen, Studenten die für Design erforderlichen Fähigkeiten mitzugeben .

Bei der Methodenentwicklung wird es darum gehen, flexibel adaptierbare Konzepte und Verfahren auszuarbeiten, die gemeinsames Arbeiten mit anderen Entwicklern und Benutzern

unterstützen. Gemeinsames, flexibel strukturiertes und inkrementelles Arbeiten wird auch maßgeblich bei der Werkzeugentwicklung sein.

Die Projektgestaltung wird die Zusammenarbeit im Design fördern. Das bedeutet möglichst hohe Autonomie, so daß wir Verantwortung einbringen können; die Schaffung eines dialogisch orientierten Arbeitsmilieus, so daß gemeinsame Perspektiven gebildet werden können; die Arbeitsteilung so, daß unsere gemeinsamen Designentscheidungen zustandekommen und beurteilt werden können; letztlich die bewußte Einbeziehung von Revisionen, so daß Design sich schließen kann.

Ein Anerkennen und Umsetzen dieser Sicht bedeutet somit, Design selbst zu gestalten ("Designing Design"). Zweifellos sind wir sehr weit von gesellschaftlichen Bedingungen entfernt, in denen das im großen Ausmaß möglich ist[1]. Das macht es um so wichtiger, als Beitrag zu wünschenswerten gesellschaftlichen Veränderungen, gangbare Wege für Design, orientiert an gemeinsam erlebter Qualität und menschengerechter Gestaltung, aufzuzeigen, zu erproben und ihre Tauglichkeit unter Beweis zu stellen.

Danksagung

Diesem Abschnitt kommt ein wichtiger Stellenwert zu, da die hier geäußerten Gedanken dialogisch entstanden sind. Das heißt, sie sind durch Zusammenarbeit, Gespräche und geistige Auseinandersetzungen mit einer Reihe von Personen, die ich nicht vollständig aufzählen kann, ermöglicht worden und zustandegekommen.

Wichtige Einsichten entstammen meinem Forschungsaufenthalt in Palo Alto 1987/88 sowie der Vorbereitung und Durchführung der Tagung "Software Development and Reality Construction". Dafür danke ich in besonderer Weise meinen Mitveranstaltern Reinhard Budde, Reinhard Keil-Slawik und Heinz Züllighoven, sowie allen anderen Organisatoren und Teilnehmern.

Eine wertvolle Gesprächspartnerin beim Schreiben dieses Papiers war mir Fanny-Michaela Reisin; durch wiederholtes und engagiertes Kreuzen unserer Perspektiven habe ich viel gelernt.

Die Möglichkeit, hiermit versuchsweise in den konstruktivistischen Diskurs einzusteigen, verdanke ich der geistigen Führung, Ermutigung und liebevollen Unterstützung durch drei

---

[1] Die Meister im Design finden sich in den skandinavischen Ländern. Aus ihren wissenschaftlichen und methodischen Ansätzen wie auch den Strategien zur gesellschaftlichen Umsetzung können wir wichtige Einsichten für unsere eigene Arbeit gewinnen (siehe /Floyd, Mehl, Reisin, Schmidt, Wolf 87/).

Wissenschaftler: Stein Bråten hat mir den Zugang zu dieser Denkwelt erschlossen und mir unschätzbare Orientierungshilfen gegeben. Gordon Pask hat offenkundig die Theorie, die ich brauche, meine bisher allerdings vergeblichen Versuche, sie zu verstehen, haben mir dennoch zentrale Einsichten vermittelt. Heinz v. Foerster hat die vorliegende Arbeit in ihren Kernpunkten inspiriert und wesentlich geprägt.

Literatur

Bateson 82: G. Bateson: *Geist und Natur. Eine notwendige Einheit*, Suhrkamp, Frankfurt a. M., 1982.

Bråten 73: S. Bråten: *Model Monopoly and Communication: Systems Theoretical Notes on Democratization*. In: Acta Sociologica 1973, Vol. 16 -No. 2.

Bråten 78: S. Bråten: *System Research and Social Science*. In: G. Klir (Ed.): Applied Systems Research: Recent Deveopments and Trends, New York, 1978.

Bråten 88: S. Bråten: *Between Dialogical Mind and Monological Reason: Postulating the Virtual Other*. In: M. Campanella (Ed.): Between Rationality and Cognition – Policy-Making under Conditions of Uncertainty, Complexity and Turbulence, Turin, 1988.

Buber 84: M. Buber: *Das Dialogische Prinzip*, Verlag Lambert Schneider, Heidelberg, 1984.

Budde, Floyd, Keil-Slawik, Züllighoven 88: R. Budde, C. Floyd, R. Keil-Slawik, H. Züllighoven (Eds.): *Software Development and Reality Construction* (Conference Preprints), GMD, Bonn, 1988.

Ehn 88: P. Ehn: *Work-Oriented Design of Computer Artifacts*, Almquist & Wiksell International, Stockholm, 1988.

Floyd 81: C. Floyd: *A Process-Oriented Approach to Software Development*. In: Systems Architecture. Proc. of the 6th European ACM Regional Conference, Westbury House, 1981.

Floyd, Keil 83: C. Floyd, R. Keil: *Adapting Software Development for Systems Design with the User*. In: U. Briefs, C. Ciborra, L. Schneider (Eds.): Systems Design For, With and By the Users, North-Holland, Amsterdam, 1983.

Floyd 84: C. Floyd: *Eine Untersuchung von Software-Entwicklungsmethoden*. In: H. Morgenbrod, W. Sammler (Hrsg.): Programmierumgebungen und Compiler, Berichte des German Chapter of the ACM 18, B.G Teubner, Stuttgart, 1984.

Floyd 85a: C. Floyd: *On the Relevance of Formal Methods to Software Development*. In: H. Ehrig, C. Floyd, M. Nivat, J. Thatcher (Eds.): Proc. of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT), Vol. 2: Formal Methods and Software Development, Springer-Verlag, Berlin Heidelberg New York Tokyo, 1985.

Floyd 85b: C. Floyd: *Comments on* "Giving back some Freedom to the System Designer" *by F. De Cindio, G. De Michelis and C. Simone – Design Viewed as a Process*. In: Systems Research Vol. 2, No. 4, pp. 281-283, 1985.

Floyd 86: C. Floyd: *STEPS – eine Orientierung der Softwaretechnik auf sozialverträgliche Technikgestaltung*. In: E. Riedemann, U. von Hagen, K.-D. Heß, W. Wicke (Hrsg.): 10

Jahre Informatik und Gesellschaft – eine Herausforderung bleibt bestehen, Universität Dortmund, Forschungsbericht Nr. 227, 1986.

Floyd 87: C. Floyd: *Outline of a Paradigm Change in Software Engineering*. In: G. Bjerknes, P. Ehn, M. Kyng (Eds.): Computers and Democracy – a Scandinavian Challenge, Gower Publishing Company Ltd., Aldershot, England, 1987.

Floyd, Mehl, Reisin, Schmidt, Wolf 87: C. Floyd, M. Mehl, F.-M. Reisin, G. Schmidt, G. Wolf: *SCANORAMA – Methoden, Konzepte, Realisierungsbedingungen und Ergebnisse von Initiativen alternativer Softwareentwicklung und -gestaltung in Skandinavien*, Werkstattbericht Nr. 30, Ministerium für Arbeit, Gesundheit und Soziales des Landes Nordrhein-Westfalen, 1987.

v. Foerster 60: H. von Foerster: *Über selbstorganisierende Systeme und ihre Umwelten*. In: ib.: Sicht und Einsicht: Versuche zu einer operativen Erkenntnistheorie, Vieweg, Braunschweig, Wiebaden, 1985.

v. Foerster 73: H. von Foerster: *Über das Konstruieren von Wirklichkeiten*. In: ib.: Sicht und Einsicht: Versuche zu einer operativen Erkenntnistheorie, Vieweg, Braunschweig, Wiebaden, 1985.

v. Foerster 85: H. von Foerster: *Sicht und Einsicht: Versuche zu einer operativen Erkenntnistheorie*, Vieweg, Braunschweig, Wiebaden, 1985.

v. Foerster 87: Heinz von Foerster: *Erkenntnistheorien und Selbstorganisation*. In: S. Schmidt (Hrsg.): Der Diskurs des radikalen Konstruktivismus, Suhrkamp, Frankfurt a. M., 1987.

v. Glasersfeld 81: E. von Glasersfeld: *Einführung in den radikalen Konstruktivismus*. In: P. Watzlawick (Hrsg.): Die erfundene Wirklichkeit, R. Piper & Co., München, 1981.

v. Glasersfeld 82: E. von Glasersfeld im Gespräch mit NIKOL: *Siegener Gespräche über Radikalen Konstruktivismus* (1982, 1984). In: S. Schmidt (Hrsg.): Der Diskurs des radikalen Konstruktivismus, Suhrkamp, Frankfurt a. M., 1987.

GMD 88: R. Budde, C. Floyd, R. Keil-Slawik, H. Züllighoven: *Bericht über die Arbeitskonferenz zu erkenntnistheoretischen Grundlagen für die Entwicklung und Nutzung von Software-Systemen*. In: Der GMD-Spiegel 4/88, 56-58.

Hejl 87: P. Hejl: *Konstruktion der sozialen Konstruktion: Grundlinien einer konstruktivistischen Sozialtheorie*. In: S. Schmidt (Hrsg.): Der Diskurs des radikalen Konstruktivismus, Suhrkamp, Frankfurt a. M., 1987.

Keil-Slawik 88: R. Keil-Slawik: *Die Gestaltung des Unsichtbaren*. In: Computer Magazin 7-8/88, 39-41.

Knuth 87: D. Knuth: *The Errors of TEX* (First Draft), Computer Science Dept.,Stanford University, Stanford, 1987.

Luhmann 87: N. Luhmann: *Soziale Systeme: Grundriß einer allgemeinen Theorie*, Suhrkamp, Frankfurt a. M., 1987.

Maturana 87: H. Maturana: *Kognition*. In: S. Schmidt (Hrsg.): Der Diskurs des radikalen Konstruktivismus, Suhrkamp, Frankfurt a. M., 1987.

Maturana, Varela 87: H. Maturana, F. Varela: *Der Baum der Erkenntnis – Die biologischen Wurzeln des menschlichen Erkennens*, Scherz Verlag, Bern, München, Wien, 1987.

Maturana, Varela, Uribe: H. Maturana, F. Varela, R. Uribe: *Autopoiesis, the Organization of Living Systems: Its Characterization and a Model*. In: Biosystems 5, 187.

Naur 74: P. Naur: *Concise Survey of Computer Methods*, Studentlitteratur, Lund, Sweden, 1974.

Naur 84: P. Naur: *Programming as Theory Building*, Datalogisk Institut, University of Copenhagen, 1984.

Parnas 72: D. Parnas: *On the Criteria To Be Used in Decomposing Systems into Modules*. In: Comm. ACM 15 (1972), 1053.

Parnas 86: D. Parnas: *Software Wars*. In: Kursbuch 83, Kursbuch/Rotbuch Verlag, Berlin, 1986.

Parnas, Clements 85: D. Parnas, P. Clements: *A Rational Design Process: How and Why to Fake It*. In: H. Ehrig, C. Floyd, M. Nivat, J. Thatcher (Eds.): Proc. of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT), Vol. 2: Formal Methods and Software Development, Springer-Verlag, Berlin Heidelberg New York Tokyo, 1985.

Pasch 89: J. Pasch: *Mehr Selbstorganisation in Softwareentwicklungsprojekten*, Manuskript, Technische Universität Berlin, 1989.

Pask 75: G. Pask: *Conversation, Cognition and Learning*, Elsevier, Amsterdam Oxford New York, 1975.

Reisin 88: F.-M. Reisin: *Anticipating Reality Construction: A Reference Scheme for Software Development*. In: R. Budde, C. Floyd, R. Keil-Slawik, H. Züllighoven (Hrsg.): Software Development and Reality Construction, Springer-Verlag, erscheint im Herbst 1989.

Reisin, Schmidt 89: F.-M. Reisin, G. Schmidt: *STEPS – ein Ansatz zur evolutionären Systementwicklung*. In: K.-D. Jansen, U. Schwitalla, W. Wicke (Hrsg.): Beteiligungsorientierte Systementwicklung, Beiträge zu Methoden der Partizipation bei der Entwicklung computergestützter Arbeitssysteme, Westdeutscher Verlag, 1989.

Schmidt 87: S. Schmidt (Hrsg.): *Der Diskurs des radikalen Konstruktivismus*, Suhrkamp, Frankfurt a. M., 1987.

Spencer Brown 69: G. Spencer Brown: *Laws of Form*, George Allen and Unwin, London, 1969.

Varela 75: F. Varela: *A Calculus of Self-Reference*. In: Int. J. General Systems 2, 5-24.

Varela 87: F. Varela: *Autonomie und Autopoiese*. In: S. Schmidt (Hrsg.): Der Diskurs des radikalen Konstruktivismus, Suhrkamp, Frankfurt a. M., 1987.

Watzlawick 81: P. Watzlawick (Hrsg.): *Die erfundene Wirklichkeit*, R. Piper & Co., München, 1981.

Winograd, Flores 86: T. Winograd , F. Flores: *Understanding Computers and Cognition*, Ablex Publishing Corporation, Norwood, New Jersey, 1986.