Die Weitsicht des Software-Entwicklers

Jan Witt, München

1 Zusammenfassung

Der Software-Entwickler bewegt sich in einer sich rasch verändernden Welt. Es wird versucht, einige Fragen hierzu zu beantworten: Gibt es stabile Gesetze, die diese Welt beherrschen? Wird der Software-Entwickler entbehrlich? Kann Software-Entwicklung ähnlich inhumane Resultate hervorbringen wie die Auswüchse des modernen Städtebaus? Bringt die künstliche Intelligenz die Lösung?

2 Einleitung -Weltsicht oder Weitsicht

Der Titel meines Vortrages ist das Ergebnis eines "Lapsus styli ", eines fast Freudschen Verschreibers. Ursprünglich war nämlich die **Weltsicht** des Software-Entwicklers, nicht seine **Weitsicht** gemeint. Dann aber auf den Spuren der Prinzen von Serendip wandelnd, die zu finden verstanden, was sie nicht gesucht hatten, fand auch ich: es zeigte sich immer mehr der neue, unbeabsichtigte Titel als die eigentliche "Trouvaille", als die wichtige Findung.

Ist nicht Weitsicht etwas, was die Träger des technischen Fortschritts und der persönlichen Verantwortung, die diese Fortschritts-Trägerschaft mit sich bringt, gut gebrauchen können, gar bitter nötig haben?

Was ist mit "Weitsicht"gemeint? Der Begriff hat, wie viele andere, eine zeitliche und eine räumliche Dimension:

Er bezeichnet eine Tugend, eine positive Eigenschaft, vorzugsweise einer einzelnen Person, die zugleich **Rücksicht**, **Vorsicht** und **Voraussicht**, aber auch die Fähigkeiten der pluralistischen Toleranz und der Offenheit gegenüber dem Zeit- und Regionalgenossen und schließlich auch Gelassenheit gegenüber den hochfrequenten Überschwingern des Tagesgeschehens und ephemeren Wunderheilungen umschließt.

Weitsicht als Weltsicht verlangt nicht einen illusionären Glauben an eine "Heile Welt", wohl aber die Überzeugung, daß es Regeln und Gesetzmäßigkeiten gibt. Der **Software-Entwickler**, wie im Grunde jeder "Profi" muß den Anspruch erheben, die "ehernen"Gesetze zu kennen, kennen zu wollen, kennen zu können, die **seiner** täglichen Arbeit zu Grunde liegen.

3 Was macht der Software-Entwickler morgen?

Wenn man über die Tätigkeit des Software-Entwicklers diskutiert, muß man sich natürlich auch die Frage stellen, was denn nun der unverlierbare Kern dieser Tätigkeit sein könnte, und welche Berufsbilder mit welchen Populationen hier mittel- und langfristig entstehen bzw. verbleiben werden.

So wie der Schneider heute häufig nur noch ein Änderungsschneider ist, der Schuster ein Flickschuster, so muß sich z. B. die Frage stellen, ob der Software-Entwickler der Zukunft nicht eher ein Software-Flicker und -Reparierer sein wird als ein Neuhersteller von Software.

Wenn man das neue Schlagwort von der **Software-Wiederverwendung** , dem **"Software reuse"** , ernst nimmt, so muß man ja auch an den **Gebraucht-Software** -Markt, ja an den **Second-hand-Softwareladen** denken.

Hier stellt sich dann auch die Frage, was denn mit der existierenden Software geschehen wird. Im Prinzip ist Software ja keinem Verschleiß unterworfen, muß, wie wir wissen aber trotzdem "gewartet" werden.

Wird man zukünftig wartungsarme, gar wartungsfreie Software entwickeln können? Wird diese auf den Paradigmen des logischen Programmierens und/oder des objektorientierten Entwurfes basieren?

Wird neue Software die alte völlig substituieren?

4 Rationalisiert sich der Software-Entwickler selbst hinweg?

Es ist vielfach darauf hingewiesen worden, daß der Software-Entwickler sich selbst möglicherweise ums Brot bringt, sich und seinesgleichen wegrationalisiert. Sollte sich herausstellen, daß es sich in der Tat vermeiden läßt, daß mal wieder einer zum 155ten Male das fünfeckige Rad erfindet, so wäre dies in der Tat ein harter Schlag für den betreffenden, sofern er sich auf nichts anderes versteht als auf das Erfinden eckiger Räder. (Man könnte ihm vielleicht einen Posten als Heizer auf einer Diesellok vermitteln).

Etwas ernsthafter fragend gelangt man zu drei Varianten:

1. Der Bedarf an Software-Entwicklern nimmt ab, weil alle benötigte Software schon existiert und via Wiederverwendung genutzt werden kann.
2. Das Software-Entwickeln kann von jedermann selbst ausgeübt werden. (Vgl. die Berufe Telephonist, Photograph, Chauffeur.)
3. Die " künstliche Intelligenz " hat die Software-Entwicklung überflüssig gemacht.

Zunächst zu den ersten zwei Möglichkeiten, die dritte werden wir in der Schlußbemerkung kurz behandeln.

Falls eine der beiden ersten Varianten tatsächlich eintreten sollte, hängt es von der Bandbreite des Bildungs- und Erfahrungshintergrundes des Einzelnen zusammen, ob er deshalb um seinen Job fürchten muß.

Es geht dabei vor allem um die Frage, ob sich der Entwickler als Universalist oder als Spezialist versteht und betätigt.

Die Frage, ob Informatik und Software-Technik überhaupt weiterhin eine kohärente Disziplin darstellen oder in Zukunft in mehrere Teile zerfallen werden, ist sicherlich genauso schwer zu beantworten und letzlich müßig wie die Fragen, ob **der Physiker** ein Universalist oder ein Spezialist ist, und ob **die Physik** etwa eine oder viele Wissenschaften darstellt.

Die Frage, ob es sich bei der Informatik der Zukunft oder ihren Nachfolgedisziplinen um Wissenschaften oder Ingenieur-Disziplinen handelt, soll an dieser Stelle nicht weiter erörtert werden.

Wichtig ist sicherlich, daß die gegenwärtige Software-Technik darauf abzielt, wie so viele andere Disziplinen auch, die Dinge ein für allemal zu lösen, und zwar dadurch, daß an die Stelle des konkreten Arbeitsablaufes das Verfügbarmachen eines Verfahrens tritt.

Wenn wir sagen, " **ein für allemal lösen** ", so lohnt es sich auch, noch ein bißchen über den Unterschied zwischen der " eigentlichen " Informatik, d.h. der Kerninformatik im engeren Sinne, und der Informationstechnik oder Informationstechnologie im weiteren Sinne nachzudenken.

Es gibt heute sicherlich in der Welt nur wenige Leute, die die Chance haben, einen Automobilmotor neu zu konstruieren, der dann tatsächlich in einer größeren Anzahl von Autos eingesetzt wird.

Gleichwohl ist es überaus nützlich, gelernt zu haben, wie man einen solchen Motor konstruiert und zu wissen, warum die landesüblichen Konstruktionen in welcher Weise entwickelt wurden. Viele technische Produkte wie z. B. auch das Fahrrad erfahren nach ihren stürmischen Anfangsjahren eine gewisse Stabilisierung, die für alle Personen, die mit diesem technischen Gegenstand umgehen, Verkäufer, Käufer, Benutzer, Reparateure eine faktische Situation festschreibt, der kaum noch ausgewichen werden kann.

Sind diese strukturellen Verkrustungen nun die zuvor erwähnten ehernen Gesetze?

Die Tierart **Frosch** hat sich in den letzen 60 Millionen Jahren genetisch praktisch nicht mehr verändert, etwa im Gegensatz zum Menschen, dessen Vorfahren nach derzeit herrschender Meinung erst vor weniger als drei Millionen Jahren mit dem Neocortex, den " **kleinen grauen Zellen** " ausgestattet wurden, die es Agatha Christie gestatteten, sich den Hercule Poirot auszudenken.

Wenn der Frosch als Resultat einer natürlichen Evolution und das moderne Fahrrad als Resultat der Koevolution von Mensch, Asphaltstraße und synthetischem Gummi jeweils einen stabilen Zustand für weitere 10 Millionen Jahre bzw. weitere 10 Jahre erreicht haben sollten, so bedeutet dies nur relative Stabilität gegenüber dem rascher Flüchtigen, wenig mehr.

Ich glaube, es wird schon zu einer gewissen allmählichen Ausdünnung der Nur-Informatiker kommen, so wie die Frösche mit der Trockenlegung der Sümpfe auch weniger werden.

Dies bedeutet aber zunächst noch keine Krise der Ausbildungspläne und Curricula sofern diese sich bemühen, verstärkt langlebiges und anwendungsneutrales Wissen in den Vordergrund zu schieben.

Es bedeutet eher schon eine Krise der im Berufsleben stehenden Software-Entwickler, und zwar im Hinblick auf die Fragen,

1. ob das was sie in ihrer Ausbildung gelernt haben für den tatsächlich ausgeübten Beruf nützlich und relevant ist
2. ob und in welcher Weise sie hier eine Möglichkeit des hierarchischen Aufstiegs in der traditionellen Form des Karrieremachens sehen.

Zu 2. ist nicht so sehr viel zu sagen: Es ist bekannt, daß in der technisch orientierten Industrie eine deutliche Tendenz besteht, vorhandene tiefgeschachtelte Hierarchien abzuflachen und Informationswege zu verkürzen.

Dies gilt ganz besonders im Bereich der Innovation technischer Artefakte, wo möglicherweise solche Hierarchien auf fachlicher Ebene nie ernsthaft wirksam waren.

Zu 1, zu den zu vermittelnden Inhalten gibt es allerdings einiges zu sagen. Ein paar grundsätzliche Tendenzen in der Berufswelt des Software-Entwicklers von heute und morgen muß man einfach zur Kenntnis nehmen:

1. Software besteht zu einem immer kleineren Teil nur aus von Computern ausführbaren einzelnen Programmen. Sie umfaßt vielmehr komplexe Systeme von solchen Programmen, zusammen mit riesigen Datenbeständen, Normen, Vereinbarungen, organisatorischen Randbedingungen und Maßnahmen.

2. Erfolgreiche, sehr weit verbreitete Software wurde meist nicht am Reißbrett designed und geplant, sondern wuchs organisch aus einer Keimzelle, möglicherweise gestützt auf eine Vision oder ein allgemeines Prinzip.

3. Der Erfolg von Software stellt sich in der Evolution und in der Bewährung im Wettbewerb heraus, er ist nur bedingt planbar und hängt, wie jede Innovation, von den sich öffnenden und sich schließenden **Zeitfenstern** ab.

4. Software wird (aus Anwendersicht) nicht gemacht, sondern gekauft. Die Schwierigkeit besteht nur darin, zu entscheiden, was man kaufen soll. Der berühme **Papiertiger**, der nur als Glanzbroschüre existiert, hat keine Chance mehr. Der Kunde will "**hands on** " Erfahrung sammeln. Oft heißt daher das Gebot der Stunde **Prototyping**.

5. Software hängt in zunehmenden Maße von technischen Gegebenheiten ab, die nicht einfach als grössere, schnellere und billigere Computer zu klassifizieren sind, nämlich z.B. von Informationsnetzen und anderen umfangreichen technischen Einrichtungen, die ihre sehr spezifischen Eigengesetzlichkeiten haben.

5 Die ehernen Gesetze und die Wissensbasen

Die Redeweisen der künstlichen Intelligenzler, haben uns gelehrt, zwischen Regeln und Fakten , die der Objektorientierten, zwischen Klassen und ihren exemplarischen Ausprägungen zu unterscheiden.

Die Gesetze, von denen zuvor die Rede war, stellen bei einem wissensbasierten System herkömmlicher Prägung Anschreibungen ohne jeglichen Realitätsbezug dar, die geeignet sein sollen, die Fakten d.h. die auftretenden Phänomene zu klassifizieren bzw. sie wirken als einschränkende Bedingungen (Constraints), die es gestatten, nur für solche Situationen Vorkehrungen zu treffen, die tatsächlich eintreten können.

Wissensbasierte Systeme befassen sich in der Regel nicht mit der Frage, wie Gesetze gelernt werden bzw. wie sie gefunden werden, noch wie die Wahrheit von Fakten geprüft werden soll und schließlich auch nicht, wie weit der Beobachtungsapparat die beobachteten Fakten selbst beinflußt. (Vgl. [Bro78]).

6 Wie ehern und ewig sind die Gesetze?

Gesetzmäßigkeiten sind begrenzt räumlich über den Grad der Allgemeinheit, den Geltungsbereich im Begriffsraum oder im Raum des Vorfindlichen, und zeitlich über die Fortdauer der Gültigkeit von Beziehungen im Lauf der Veränderungen.

Der Begriff des Gesetzes, (griech. **nomos**), konnotiert sehr stark Beharrlichkeit, Beständigkeit und fortdauernde Gültigkeit. [1]

Seit Hammurabis Zeiten wird der Gesetzesbegriff verknüpft mit der Frage, ob Gesetze menschliche Setzungen, oder unauflösliche Vorgaben der Götter oder der Natur, also Naturgesetze sind, bzw. umgekehrt, ob auch das reversibel "gesetzte" gleichwohl "Gesetz" genannt werden soll.

Gerade heute muß man sich stets aufs neue die Frage stellen, was denn zufällige (d.h. nicht notwendige) menschliche Setzungen auszeichnet, vor welchem Hintergrund eines möglichen Andersseins sie zu sehen sind. Heute einmal deshalb, weil der **Konnektionismus** , das

[1] Der griech. Wortstamm ist ja auch Bestandteil des Wortes **Ergonomie**.

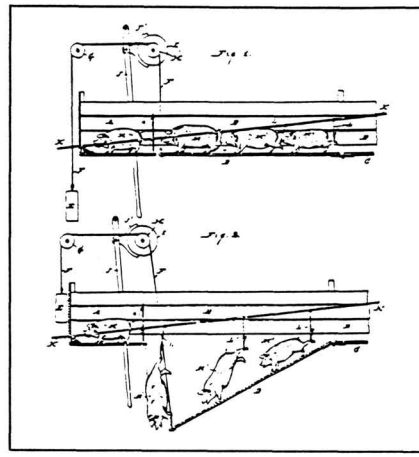

Abbildung 1: Apparat zum Aufhängen von Schweinen, 1882 nach [Gie87]

Kehren wir zurück zur Gebäude-Archtiktektur-Analogie: Der Software-Entwickler von heute ist in der Mehrzahl der Fälle - wir wiesen bereits darauf hin - eigentlich nicht so sehr ein Architekt, der auf der grünen Wiese (bisher Bauerwartungsland) Neues entwickelt, sondern er ist meist damit befaßt zu modernisieren, zu sanieren und auch zu integrieren, d.h. Verbindungen zwischen Komponenten herzustellen, die vorher nicht bestanden.

Wenn einige vermeintlich glückliche System-Designer die Möglichkeit bekommen, einmal ganz von Grund auf neu "Software-Städteplanung" zu betreiben oder zumindest ein Stadtviertel oder einen Gebäudekomplex zu planen und dann zu realisieren, dann führt das in der Regel zu einem Desaster.

Dies Phänomen ist ja auch aus dem Bereich der echten Städteplanung bekannt, man denke an Brasilia, an das märkische Viertel in Berlin, an die Vielzahl der "aus der Tube gequetschten" Vorstädte, Kulturzentren etc. Warum kommt es eigentlich immer wieder zu diesen Dilemmas? Warum entstehen Piranesische unbewohnbare und unbewohnte Hallen anstelle von freundlichen bewohnbaren Strukturen? (Abbildung 3). Auf diese Fragen gibt es sicher viele verschiedene Antworten. Es fällt schwer, hier Position zu beziehen, ohne zumindest teilweise in Kulturpessimismus zu verfallen, bzw. zumindest die Mißbeschaffenheit der Epoche zu beklagen. vgl.[III73, III78, I+79, Pad84]) Die Ursachen sind sicher vielgestaltig: Betrachten wir zunächst den hohen Stellenwert der persönlichen Erfahrung gegenüber dem bloß angelesenen Wissen. So mancher hält sich für einen Designer und bewährt sich auch, solange er von einer bewährten Grundstruktur nicht allzuweit abweicht, versagt aber, sobald der Entwurfsspielraum allzu groß und vieldimensional wird.

Wahrscheinlich liegen die Ursachen aber noch tiefer:

Umgehen mit automatischen, "lernenden" Klassifikatoren auf Basis der "künstlichen neuronalen Netze" neue Zugänge zum Verständnis von Begriff, Sprache und Lernen liefern und zum anderen zur Beantwortung der Frage, welcher ethische Rahmen der menschlichen Freizügigkeit, Gesetze zu setzen , denn als Grenze zukünftig zu dienen habe, welche denn die großen kosmischen und säkularen Invarianten sind, die zu verändern dem Menschen nicht gegeben ist, es sei denn um den Preis der Hybris, und was denn der Preis dieser Hybris, die Strafe des Frevels wider die Götter sein möchte, wenn eben diese Invarianten verletzt werden.

Entdecken oder Schaffen wir unsere tägliche Wirklichkeit? Und - ist das Beobachtete, zusammen mit den Möglichkeiten der Beobachtbarkeit nicht auch vorgeprägt durch die beabsichtigte spätere Interpretation? Wie stark prägt die Sprache unser Denken und Handeln?

Etwa seit dem Anfang der 60er Jahre weisen Philosophen, Wissenschaftler und Techniker verstärkt darauf hin, daß wir nicht nur manifeste physische Artefakte um uns herum gruppieren, sondern daß auch viele unserer "**Entdeckungen**" zu einem großen Teil als **willkürliche Kreationen** betrachtet werden können. (vgl. z.B. Kuhn [Kuh62],Nelson Goodman [Goo77, Goo78], Winograd und Flores [WF86] u.a.)

Dem Gesetzesbegriff eigentümlich sind neben der erwähnten Persistenzeigenschaft auch

- der Anspruch auf eine räumlich und zeitlich begrenzte Ausdehnung des Gültigkeitsbereiches. (d.h. es ist ihnen eigentümlich, daß sie zu anderer Zeit und an anderem Orte auch nicht gelten könnten.)
- der Anspruch der Unmißverständlichkeit, Genauigkeit.
- der Anspruch auf Formalisierbarkeit, Kodifizierbarkeit oder zumindest auf Aufschreibbarkeit.

Gesetze sind **Gebote** oder **Verbote** und dementsprechend sind sie, in der Sprache der Informatik, weder rein **prozedural** noch rein **deskriptiv**, sie haben überwiegend den Character von "**constraints**". Sie erzwingen Sachverhalte und schließen andere Sachverhalte aus, dort wo sie und dann wenn sie gelten. Sie erlauben auch das Herleiten quantitativer Zusammenhänge.

Nicht nur der allgemeine Wissensdrang des Menschen, sondern auch seine wirtschaftlichen Interessen machen gerade das Auffinden bzw. Festlegen der zeitlichen und räumlichen Grenzen von Gesetzen außerordentlich interessant.

Formulierbarkeit von Projektkosten, Lebensdauer von Software, Bedarf an Software-Spezialisten , Wiederverwendung von Software, Wohlverhalten im Umgang mit Software, etc. können nur auf der Basis von Gesetzmäßigkeiten sichergestellt werden.

Es geht also einmal darum, weltausschnitts-spezifische "quasi-ewige" Invarianten zu finden dazu aber auch einigermaßen stabile (extrapolierbare) Trends, d.h. zeitliche Veränderungen, die sich formal packen lassen. In diesen Bereich gehören die Versuche, Software-Metriken, "Software-Physics", Software-Arbeitsmethoden und, last not least, auch Gesetzmäßigkeiten der Software-Ergonomie zu finden.

Zum anderen ist empirisches **Weltwissen**, das sich der Formalisierung entzieht und stets hochgradig erfahrungsabhängig bleibt, in geeigneter Weise zu klassifizieren, zu sammeln und zumindest heuristisch beherrschbar und fortschreibbar zu machen.

Die Weitsicht des Software-Entwicklers müßte sich also auf beides stützen:

- (hoffentlich langlebige) monolithische axiomatische Theorien
- detailreiche feingeflochtene Erfahrungsmuster ohne faßbare axiomatische Basis.

Diesem Wunsch stehen entgegen:

- Die Widersprüche zwischen empfohlener Technik und geübter Praxis (wie finden Software-Design und -Implementierung tatsächlich statt?)
- Die Begrenztheit der persönlichen Erfahrungen des Einzelnen und die Schwierigkeit, diese weiterzugeben. (Vgl. [Pol62, Pol67, Pol69].)

7 Software-Entwicklung ist wie Städtebau

Im Bereich der Informatik ist die Analogie zum Häuserbauen oder besser gesagt zur Architektur ja sehr verbreitet. Man spricht von Computer-Architektur von der Architektur eines Systems usw.

Bei der Computer-Architektur ist aber meist eher das gemeint, was beim Häuserbau der Statik entspricht, der technischen Funktionstüchtigkeit, vielleicht auch der Wirtschaftlichkeit, nicht so sehr der Benutzbarkeit.

Ähnliches ist bei der "Architektur" eines Anwendungssystems gemeint. Selten ist die Rede von der Bewohnbarkeit eines so entworfenen und realisierten Gebäudes und dem eigentlichen Leben, das sich dort auf die Dauer abspielt. Man denke an Mitscherlichs "Unwirtlichkeit der Städte " ", oder man vergleiche hierzu auch z.B. die Ausführungen von Tzonis [Tzo76] über den Entwurfsprozeß des **"vor-rationellen Menschen"** .

Im Bereich der Realisierung von Computer-Anwendungen beobachtet man immer wieder noch eine ausufernde Begeisterung für das technisch Machbare, wie sie zu Beginn des Maschinenzeitalters mit der Apotheose des Zahnrades Hand in Hand ging. Bekannte Kritiker wie Joseph Weizenbaum oder die Brüder Dreyfus bemängeln vor allem den Verlust von Lebensqualität, aber auch die materiellen Schäden, die aus dem Ausschöpfen der technischen Möglichkeiten resultieren können. Ich meine aber auch, daß es hier auch noch eine unbewußte Verrohung, ein genüßliches Auskosten von unerfreulichen Begleitphänomenen der technischen Schöpfung gibt. Ich möchte hierzu zwei Beispiele als Analogie aus der frühen Maschinenzeit geben, beide aus [Gie87] entnommen. Das erste (Abbildung 1) ist eine technische Einrichtung, die es gestattet, Schweine "am laufenden Band" zu schlachten. Natürlich ist das Schlachten von Schweinen sicher in keinem Fall eine besonders ästhetische Angelegenheit. Hier wird jedoch ein lebendes Schwein nach dem anderen an einem Hinterfuß kopfüber aufgehängt und so zu seinem Tode transportiert, wobei der Zeichner durch geöffnete Mäuler noch andeutet, daß die Schweine erschreckt aufquieken. Das zweite Beispiel (Abbildung 2) ist ein mechanischer Zahnarztstuhl, aus der gleichen Zeit, sicherlich zweckmäßig gebaut, den heutigen Stühlen dieser Art nicht unähnlich. Die Darstellung konzentriert sich darauf, zu zeigen, mit Hilfe welcher Justierschrauben der Patient für den Behandelnden in optimale Lage gebracht und dort fixiert werden kann: Technik als Mittel der Ausübung (wohltätigen) physischen Zwanges.

Mit diesen beiden Beispielen sollte natürlich nicht gesagt werden, daß der Software-Entwickler ein "armes Schwein" sei, oder daß es sich empfehle, ihm "auf den Zahn zu fühlen".

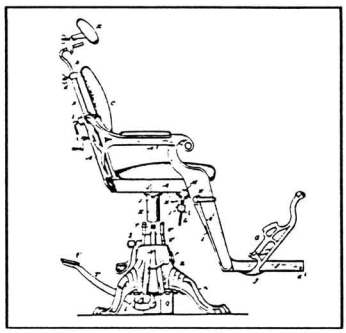

Abbildung 2: Zahnarztstuhl, 1879 nach [Gie87]

Abbildung 3: Piranesi: Carceri - vor 1750

8 Arbeitsteilige Software-Entwicklung

Bevor wir mit den ehernen Gesetzen fortfahren, ist es vielleicht angemessen, einmal über die Frage nachzudenken, wen wir eigentlich mit dem Software-Entwickler gemeint haben.

Es gab eine Zeit, sie liegt, Gott sei Dank, schon eine Weile zurück, in der man glaubte, daß es jeweils einen Chef und viele Untergebene bei den Software-Entwicklern gibt. Der Chef gibt die Befehle und die Mitarbeiter setzen sie in die Tat um.

Man darf das nicht belächeln, denn es gab auch eine Zeit, in der ein gewisser Dawson in den USA die ersten Verkehrsflugzeuge (aus Sperrholz) baute, wobei er selbst - wie ein Kapitän auf der Brücke stehend - Kommandos gab, die ein Steuermann dann in die Tat umsetzte. (Leider stürzte eines seiner Flugzeuge mit sehr vielen Prominenten an Bord dann in den Sumpf. Einige meinten, es habe an der Kommandostruktur gelegen).

Aber wie soll man denn eine Horde ungebärdiger Hacker im Schach halten, wenn nicht durch eine paramilitärische Organisationsform?

Eine Kompromißformel bot sich Anfang der 70er Jahre in der Form des legendären Chief-

Programmer-Teams an, wobei allerdings damals noch niemand wußte, daß das vielgepriesene New York Times-Projekt, bei dem das Chief-Programmmer-Team angeblich die wundervollsten Wirkungen vollbrachte, selbst ein ziemlicher Flop war, dessen Ergebnisse nie zum praktischen Einsatz kamen ([Bak72, BM73]).

In diesem Zusammenhang kann man sicherlich über Sinn und Unsinn von Arbeitsteiligkeit philosophieren, aber auch über die Fragen von Qualifikation und der Rolle, die ein Entwickler in einem Projekt spielt.

Am einen Ende der Skala steht sicher nach wie vor der geniale halbverrückte Hacker, äußerlich unter die "great unwashed" einzuordnen, der die Nacht zum Tage macht, seine Instruktionen direkt von den Göttern bekommt, dafür aber unübertrefflich perfekte Software produziert, und am anderen Ende der begnadete, peitschenknallende Manager, der auch dem armseligsten Würstchen noch Höchstleistungen abpreßt, ohne selbst von der Software-Entwicklung mehr zu verstehen als der Software-Entwickler vom Käsehandel.

Irgendwie kommt es hierbei natürlich darauf an, was eigentlich getan werden soll, und daß, falls hier wieder ein Kapitän auf der Brücke steht und Kommandos ruft, eigentlich sichergestellt ist daß das, was angeblich getan werden soll überhaupt sinnvoll ist zu tun.

Mit dem Aussterben der "grüne-Wiese"-Projekte verschwindet auch der Wunsch, große Teams zu bilden, und damit (hoffentlich) auch der Software-Kommandant.

9 Die " Software-Zunftmeister "

Seit längerem etablierte Wissenschaftszweige oder Ingenieur-Disziplinen verfügen neben dem inhaltlich stabilen Wissensgebäude, das die herrschende Lehrmeinung reflektiert, auch über Stile und Konventionen wie mit diesem Wissensgebäude umzugehen ist.

Die Professionalität wird oft daran gemessen, in welchem Umfang diese Konventionen eingehalten werden. Wer sich diesen Konventionen nicht fügt, wird im wissenschaftlichen Bereich als Hochstapler, im Bereich der Ingenieur-Disziplinen als Pfuscher oder zumindest als unprofessionell angesehen.

Hierüber wachen Berufsverbände, Standesorganisationen, Hochschullehrer und Herausgeber von Zeitschriften als strenge Zunftmeister.

Bei sehr neuen oder sich rasch verändernden Disziplinen und Ingenieur-Methoden, wie sie ja im Bereich der Informationstechnik die Regel darstellen (wenigstens bis jetzt), ist zu unterscheiden zwischen (schon) dogmatisierten d.h. reglementierten Themenbereichen und Themenbereichen, die ihre endgültige feste Form noch nicht gefunden haben.

Unreglementiert sind naturgemäß vor allem Bereiche, in denen auch noch keine gültigen Standards existieren, und in denen subjektiv gefärbte Mutmaßungen stellvertretend für gesicherte Tatsachen einspringen müssen.

Die Zunftmeister überwachen in erster Linie die fest dogmatisierten Bereiche und erklären auch, was als experimentell und daher tolerabel und was als falsch dogmatisiert, d.h. als ketzerisch zu gelten hat.

Für die Welt des Software-Entwicklers gibt es (zur Zeit noch) einige unabhängig dogmatisierte Hochburgen, die noch nicht gar so lange diplomatische Beziehungen miteinander aufgenommen haben. Es sind dies

- Die strenggläubigen KI-Forscher.(" hard ai ", besondere Eiferer: die Anhänger von COMMON LISP, CLOS etc., sowie der mehrfachen Vererbung als Bestandteil der Schöpfungsgeschichte.)

- Die strenggläubigen nicht objektorientierten Algorithmiker. (Besonders asketisch: Anhänger des **Functional Programming** : Sie wollen als Programmiersprache neben Miranda eigentlich nur Haskell gelten lassen.)
- Die strenggläubigen (nicht objektorientierten) Datenbankler. (Den Vertretern des Entity-Relationship-Glaubens ist es bei Strafe der Exkommunikation verboten, das Wort " Wissensrepräsentation " auszusprechen.)
- Die strenggläubigen Software-Project-Manager. (sie glauben, Strand 88 sei nur eine Adresse in London, Eiffel nur ein scheußlicher Turm in Paris.)
- Die strenggläubigen Konnektionisten. (Das Leben ist Lernen.)
- Die gemäßigt rechtgläubigen Objektorientierten. (Sie glauben nicht an die klassenlose Gesellschaft.)
- Die strenggläubigen Software-Metriker.(Im Aussterben.)
- Die strenggläubigen Psycholinguisten. (Auch sie dürfen das Wort "Wissensrepräsentation nicht gebrauchen, sich sich aber mit Begriffen wie "Deep semantic Model" helfen.)

Für (grundsätzliche) Fragen der Dogmatisierung siehe auch [Bar73, Cas46, MCL66, Pol62].

Wir erwähnten bereits die Frage nach der Gültigkeit, Absolutheit und Verbindlichkeit von Gesetzen. In diesen Bereich gehört auch die Treffsicherheit von Prozeduren d.h. die Planbarkeit der Zukunft vor dem Hintergrund von Determinismus und Entscheidbarkeit auf der einen und Chaos-Theorie auf der anderen Seite.

10 Schlußbemerkung

Software-Entwicklung sollte nach Inhalt und Methode (Software-Ergonomie) eine Wissenschaft nicht nur von den Dingen, sondern auch vom und für den Menschen sein, und zwar von dem Menschen, der Vorhandenes lernt und weiter vervollkommnet, also vom Homo faber, vom menschlichen Schmied im besten Sinne. Der Software-Schmied ist grundsätzlich mit beidem befaßt: mit dem Hammer, der den Nagel schmiedet und dem Hammer, der den Nagel einschlägt. Die Hervorbringungen der künstlichen Intelligenz können bei beidem hilfreich sein, beim Nagelschmieden und beim Nageln. Sie helfen jedoch wenig beim Üben der menschlichen Kunst des sich etwas Wünschens und auch nicht bei den Fragen der Akzeptierbarkeit des Herbeigewünschten, das nur allzu oft alsbald zum Teufel verwünscht wird wie im Märchen von den drei Wünschen die Wurst, die der Bäuerin an der Nase angewachsen war. Die Besorgnis, unser Software-Entwickler könnte über kurz oder lang von der künstlichen Intelligenz auf dem Wege des " Automatic Programming " arbeitslos gemacht werden, erweist sich dort als unbegründet, wo klar wird, daß es keine **Wunschmaschine** (sondern allenfalls eine Wunscherfüllungsmaschine) geben kann, wenngleich der Computer immer wieder als eine solche angepriesen wird ([Tur84]).

Die Weitsicht des Software-Entwicklers sollte ihn natürlich nicht dazu verleiten, allzu weit in die Ferne zu schauen, ohne auf das zu achten, was auf seinem Schreibtisch liegt. Insbesondere sollte er Fachliteratur, Fachzeitschriften und eventuell "Network News"-Dienste in Anspruch nehmen um im Auge zu behalten, was weltweit in der Informationstechnik vor sich geht, sonst wird er am Ende vielleicht doch noch ein "armes Schwein".

Literatur

[Bak72] Terry F. Baker. Chief programmer team management of production programming. *IBM Systems Journal*, 11(2):56–73, 1972.

[Bar73] Roland Barthes. *Mythologies.* orig: Editions du Seuil, 1957, engl. tr. Paladin, Grafton, London, Glasgow, 1973.

[BM73] Terry F. Baker and Harlan D. Mills. Chief programmer teams. *Datamation*, 19(12):58–61, 1973.

[Bro78] Jacob Bronowski. *The Origins of Knowledge and Imagination.* Yale University Press, New Haven, Ct., 1978.

[Cas46] Ernst Cassirer. *Language and Myth.* Harper and Brothers, 1946.

[Gie87] Sigfried Giedion. *Die Herrschaft der Mechanisierung.* Europäische Verlagsanstalt, Frankfurt am Main, 1987.

[Goo77] Nelson Goodman. *The Structure Of Appearance.* Reidel, Dordrecht, Holland, 1977.

[Goo78] Nelson Goodman. *Ways of Worldmaking.* Hachett Publishing Company, 1978.

[I+79] I.D. Illich et al. *Entmündigung durch Experten.* rororo aktuell, 1979.

[Ill73] I.D. Illich. *La Convivialité.* 1973.

[Ill78] I.D. Illich. *Fortschrittsmythen.* rororo aktuell, 1978.

[Kuh62] T.S Kuhn. *The Structure of Scientific Revolutions.* 1962.

[MCL66] Marshal MCLuhan. *Understanding Media.* 1966.

[Pad84] Hanspeter Padrutt. *Der epochale Winter. - Zeitgemäße Betrachtungen -.* Diogenes, Zürich, 1984.

[Pol62] Michael Polanyi. *Personal Knowledge.* Routledge and Kegan Paul, 1962.

[Pol67] Michael Polanyi. *The Tacit Dimension.* Routledge and Kegan Paul, 1967.

[Pol69] Michael Polanyi. *Knowing and Being.* Routledge and Kegan Paul, 1969.

[Tur84] Sherry Turkle. *Die Wunschmaschine - vom Entstehen der Computerkultur -.* Rowohlt, 1984.

[Tzo76] Alexander Tzonis. *Vers un environment non-oppressif.* Pierre Mardaga, Brussels, 1976.

[WF86] Terry Winograd and Fernando Flores. *Understanding Computers and Human Cognition.* 1986.

Anschrift des Verfassers:
Dr. Jan Witt c/o PCS Computer Systeme GmbH
Pfälzer Wald Straße 36
D-8000 München 90