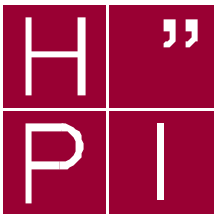
HASSO- PLATTNER - INSTITUT
für Softwaresystemtechnik an der Universität Potsdam



Auf dem Weg zu einem Softwareingenieurwesen

Prof. Dr. -Ing. Siegfried Wendt

Technische Berichte Nr. 1
des Hasso-Plattner-Instituts
für Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik
an der Universität Potsdam

Auf dem Weg zu einem Softwareingenieurwesen

Prof. Dr. -Ing. Siegfried Wendt

Potsdam 2004

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

Die Reihe *Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam* erscheint aperiodisch.

Herausgeber: Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam

Redaktion: Prof. Dr. -Ing. Siegfried Wendt
Email: Siegfried.wendt@hpi.uni-potsdam.de

Vertrieb: Universitätsverlag Potsdam
Postfach 60 15 53
14415 Potsdam
Fon +49 (0) 331 977 4517
Fax +49 (0) 331 977 4625
e-mail: ubpub@rz.uni-potsdam.de
http://info.ub.uni-potsdam.de/verlag.htm

Druck: allprintmedia gmbH
Blomberger Weg 6a
13437 Berlin
email: info@allprint-media.de

Heft 1 (2004)
ISBN 3-937786-37-6
ISSN 1613-5652

Auf dem Weg zu einem Softwareingenieurwesen

Die bewusste Wahrnehmung eines Problems ist der halbe Weg zu seiner Lösung. Die vorliegende Sammlung von Aufsätzen aus den Jahren 1998 bis 2004 hat den Zweck, ein Problem ins Bewusstsein zu rücken, dessen Lösung den Beginn der Entstehung eines ernstzunehmenden Softwareingenieurwesens kennzeichnet.

Meine beiden Kernthesen sind:

(1) Die thematische Fülle des Gebietes "Software" ist viel zu groß, als dass sie innerhalb einer einzigen akademischen Disziplin bewältigt werden könnte.

(2) Innerhalb eines Softwareingenieurwesens kommt der Methodik der Systemmodellierung eine Schlüsselstellung zu.

Meine These (1) möchte ich durch eine Gegenüberstellung der in den beiden Bildern 1 und 2 dargestellten Tabellen untermauern. In der Tabelle in Bild 1 vervielfacht sich von Zeile zu Zeile die Anzahl der beteiligten Atome. Man könnte alle in den Zeilen betrachteten Gebilde "als Haufen von Atomen" ansehen. Dies wäre aber sicher unangemessen, denn das Wesen der Gebilde wird keineswegs ausschließlich durch die Art und die Anzahl der beteiligten Atome bestimmt, sondern ganz besonders durch die Art ihrer Konfiguration. Deshalb ist auch fast in jeder Zeile eine andere zuständige akademische Disziplin eingetragen.

Gegenstand der Betrachtung	Zuständige Fachdisziplin
Atom	Physik
Molekül = viele Atome in bestimmter Konfiguration	Chemie
Zelle = viele Moleküle in bestimmter Konfiguration	Biologie oder Medizin
Organ = viele Zellen in bestimmter Konfiguration	Biologie oder Medizin
Mensch = viele Organe in bestimmter Konfiguration	Medizin oder Psychologie
Unternehmen = viele Menschen in bestimmter Konfiguration	Betriebswirtschaftslehre
Wirtschaftssystem = viele Unternehmen in bestimmter Konfiguration	Volkswirtschaftslehre

Bild 1 Zunahme von Quantität schafft neue Qualitäten: Atome

Gegenstand der Betrachtung	Zuständige Fachdisziplin
Symbol	Informatik
Codezeile = viele Symbole in bestimmter Konfiguration	Informatik
Programmkonstrukt = viele Codezeilen in bestimmter Konfiguration	Informatik
Algorithmus = viele Programmkonstrukte in best. Konfiguration	Informatik
Modul = viele Algorithmen in bestimmter Konfiguration	Informatik oder SW-Ingenieurwesen
Softwaresystem = viele Module in bestimmter Konfiguration	SW-Ingenieurwesen
Vernetzte Softwaresysteme = viele SW-Systeme in best. Konfig.	SW-Ingenieurwesen

Bild 2 Zunahme von Quantität schafft neue Qualitäten: Symbole

Von den "Haufen von Atomen" gehen wir nun über zu "Haufen von Symbolen", die in den Zeilen der Tabelle in Bild 2 eingetragen sind. Hier vervielfacht sich von Zeile zu Zeile die Anzahl der beteiligten Symbole. Obwohl es hier möglich ist, mehrere Zeilen durch die gleiche akademische Disziplin abzudecken, wäre es doch Hybris, Dummheit oder opportunistische Verlogenheit zu behaupten, alle Zeilen gehörten in die Zuständigkeit einer einzigen Disziplin, und dies sei die Informatik.

Meine These (2) gibt einen Hinweis auf die erforderliche Schwerpunktsetzung in der Ausbildung von Softwareingenieuren. In der konventionellen Informatikausbildung wird üblicherweise auf die Anfangsjahre der beruflichen Tätigkeit der Absolventen geschaut, wo diese selbstverständlich noch keine echten Ingenieursaufgaben bekommen, bei denen sie planend oder koordinierend für die arbeitsteilige Realisierung komplexer Systeme verantwortlich sind. Sie werden anfangs zum überwiegenden Teil als Softwareentwickler eingesetzt, für deren Arbeit die routinierte Beherrschung bestimmter Programmiersprachen und -konzepte erforderlich ist. In einer im Hinblick auf diese Anfangsjahre geplanten Ausbildung wird deshalb der überwiegende Teil der Ausbildungszeit auf die Vermittlung von Kenntnissen verwendet, die ein Softwareentwickler braucht (siehe Bild 3 oben). Dagegen wird in einer auf die späteren Berufsjahre zielenden Ausbildung der größte Teil der Ausbildungszeit den Problemen gewidmet, die von Softwarearchitekten und IT-Projektleitern bewältigt werden müssen (Bild 3 unten). Dass dabei für die Vermittlung der in den ersten Berufsjahren benötigten Kenntnisse nur noch ein kleiner Teil der Ausbildungszeit zur Verfügung steht, ist unkritisch, denn die Praxis hat gezeigt, dass sich die so ausgebildeten Absolventen bei Bedarf die noch fehlenden Kenntnisse schnell autodidaktisch aneignen können, wobei man noch bedenken muss, dass der jeweilige Bedarf sehr stark vom konkreten Arbeitsumfeld abhängt, in das sich die Absolventen einfügen müssen und das von der Ausbildungsinstitution nicht vorhergesehen werden kann.

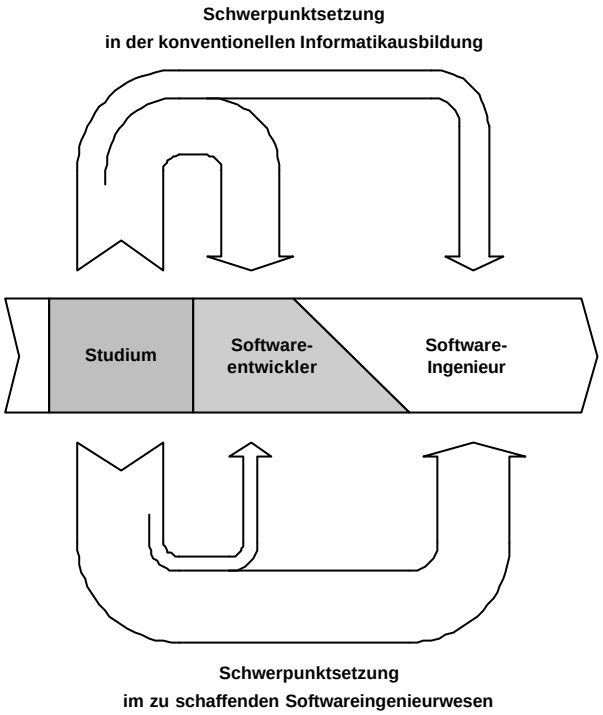


Bild 3 Alternative Schwerpunktsetzungen in software-bezogenen Lehrplänen

Es folgen nun sechs Aufsätze, worin zusätzliche Informationen zur Erhärtung und Konkretisierung der obigen beiden Thesen geliefert werden:

1. Über die Notwendigkeit, die bisherige Informatik in eine Grundlagenwissenschaft und eine Ingenieurwissenschaft aufzuspalten

2. Was ist Ingenieurskultur?

3. Das Kommunikationsproblem der Informatiker und ihre Unfähigkeit, es wahrzunehmen

4. Besonderheiten des Softwareingenieurwesens im Vergleich mit den klassischen Ingenieurdisziplinen

5. Softwareingenieurspläne können auch für Nichtfachleute verständlich sein.

6. Principles for Planning Curricula in Software Engineering

(1)

Über die Notwendigkeit, die bisherige Informatik in eine Grundlagenwissenschaft und eine Ingenieurwissenschaft aufzuspalten

(aus "Potsdamer Industrieblätter" Nr.1 des Industrieclubs Potsdam)

Das Softwarewesen ist gegenwärtig noch in einem Zustand der Unreife. Ob es uns gelingt, diesen zu überwinden, wird mit entscheiden, ob Deutschland ein Land mit Ingenieurskompetenz bleiben kann oder nicht. Hierfür braucht man nicht vorherzusehen, welche zukünftigen Produkte und Leistungen auf der Grundlage von Software kommen werden. In der Informationsverarbeitung ist fast alles technisch machbar, und es ist nur eine Frage, ob es sich wirtschaftlich durchsetzen wird.

Zunächst: was ist Software überhaupt? Hardware ist stets materiell, d. h. aus Materie aufgebaut. Durch Wandlung materieller und energetischer Strukturen kann Hardware Leistungen erbringen, die von den Menschen als nützlich erlebt werden. Software kommt ins Spiel, wenn die Hardware parametrisierbar ist, d. h. wenn der Hersteller der Hardware ihre Leistung noch nicht endgültig festlegen muss. Diese Festlegung erfolgt erst durch den späteren Nutzer, indem dieser der Hardware seinen Willen aufzwingt. Hierfür gibt es in der Hardware sogenannte Speicher, in die der Nutzer seinen Willen in Form bestimmter Strukturmuster einbringen kann. Software nennen wir die Information, die der Nutzer als seine Willensäußerung gegenüber der Hardware formulieren kann. Von Software kann also nie Leistung erbracht werden, obwohl man manchmal von der Software so spricht, als wäre sie ein Akteur, der irgendetwas richtig oder falsch machen kann. Mit der Erfindung des Computers ist eine ungeheuer mächtige parametrisierbare Hardware in die Welt gekommen. Diese Parametrisierbarkeit hat den Raum der Lösungsmöglichkeiten für technische Probleme ins schier Unermessliche erweitert. Früher musste man bei der Herstellung technischer Apparate ihre Endfunktion bereits bei der Herstellung festlegen. Denken Sie beispielsweise an die Form der Schriftzeichen, die bei den früheren Schreibmaschinen in metallische Form gegossen waren. Heute kann man die Form der gewünschten Schriftzeichen als geäußerten Willen der Maschine übermitteln und auf diese Weise mühelos sehr viel mehr Schriftzeichen nutzen, als sie früher in den Setzkästen der Druckereien verfügbar waren. Diese gewaltige Erweiterung der Funktionalität hat allerdings ein Problem mit sich gebracht, welches heute als das schwierigste Problem der Softwaredisziplin gewertet werden muss: Die Form, in der die Software für die Maschine formuliert werden muss, ist i.a. so kryptisch, dass selbst derjenige, der sie ursprünglich ausgedacht und formuliert hat, schon nach recht kurzer Zeit große Mühe aufwenden muss, um sie wieder zu verstehen.

Weil man Software nicht sieht, ist es bisher noch gar nicht ins Bewusstsein der breiten Öffentlichkeit gelangt, dass Software schon längst als eine zivilisationstragende Technologie gleichgewichtig neben Stahl und Strom getreten ist. Während es allen bewusst ist, dass sie auf Schritt und Tritt auf die Verfügbarkeit von Stahl und elektrischer Energie angewiesen sind, meinen immer noch die meisten, dass sie mit Software nicht in Berührung kämen, solange sie keine Computer benutzen. Es ist ihnen gar nicht klar, dass gar nichts mehr in ihrem technisierten Alltag funktionieren würde, also auch die Energieversorgung und das Verkehrswesen nicht, wenn die in diesen Systemen verborgen arbeitenden Computer ihre Arbeit nicht korrekt machen würden. Die Computer, die wir in unserem Alltag unmittelbar erleben, bilden nur einen sehr geringen Bruchteil von unter 2 % aller in Betrieb befindlichen Computer.

Das Vermittlungssystem im Telefonnetz, die Abfüllmaschine in der Molkerei, die Steuerung von Signalen und Weichen bei der Bahn, der Kraftstoff sparende Motor im Auto oder die Navigation der Flugzeuge in der Luft beruhen heute alle auf Software.

Auch die Erfolge der Gentechnologie sind ohne intensive Nutzung von Software nicht möglich. Wie tiefgreifend die durch Software verursachten Veränderungen im Wirtschaftsleben bereits sind, kann man an der Tatsache erkennen, dass selbst der Elektrokonzern Siemens bereits mehr als die Hälfte seiner Wertschöpfung aus der Software gewinnt.

Die Komplexität heutiger Softwaresysteme ist eine zwangsläufige Konsequenz der seit Jahrzehnten exponentiell gestiegenen und immer noch weiter steigenden Leistungsfähigkeit der Hardware, sowohl was die Verarbeitungsgeschwindigkeit als auch was die Speicherkapazität angeht. Deshalb wird der Entwicklungsbedarf für Software und der Umfang der Softwaresysteme in den kommenden Jahrzehnten noch stark ansteigen. Es gibt jedoch eine große Diskrepanz zwischen der wissenschaftlichen Reife des Hardwaresektors im Vergleich zum Softwaresektor. Die Disziplinen, die für die Hardware zuständig sind, also die Naturwissenschaften Physik und Chemie sowie die Ingenieurwissenschaften Elektrotechnik und Maschinenbau, waren schon vor der Erfindung des Computers gereifte Wissenschaften, so dass die weitere Entwicklung auf einem soliden Fundament aufbauen konnte. Im Gegensatz hierzu entstand die Softwarewissenschaft erst nach der Erfindung des Computers. Mit der rasanten Hardwareentwicklung konnte der Reifungsprozess der Softwarewissenschaft nicht mithalten. Allerdings leugnen - vermutlich aus einem plausiblen Selbstschutzreflex heraus - die meisten in der Softwarebranche Tätigen diese Tatsache, und unsere politische und wirtschaftliche Führungselite ist meist auch nicht in der Lage, das Problem und die damit verbundenen Risiken klar zu sehen. Bei den oft genug erhobenen Forderungen nach mehr staatlicher Förderung der Informatikwissenschaft wird fast immer mit Defiziten argumentiert, die das eigentliche Problem so gut wir gar nicht berühren. Man verweist immer auf die Möglichkeit, die Leistungsfähigkeit der Systeme um wünschenswerte und verblüffende Funktionen zu erweitern, was man unbedingt realisieren müsse, um nicht im Wettbewerb abgehängt zu werden. Man stachelt sich gegenseitig an, "immer noch höhere Türme" zu bauen, was auf Grund der wachsenden Leistungsfähigkeit der Hardware tatsächlich immer möglich ist. Was aber fast niemandem aufzufallen scheint, ist die Tatsache, dass von den Erfahrungen, welche auf den Softwarebaustellen gesammelt werden, ein viel zu großer Teil in den Köpfen der Turmbauer eingeschlossen bleibt und nicht an die Kollegen oder die nachfolgende Generation weiter gegeben wird. Hierin äußert sich die Tatsache, dass die Informatikwissenschaft mit der exponentiell gewachsenen Komplexität der Systeme nicht Schritt halten konnte. Man baut heutzutage Systeme, bei denen der Umfang der enthaltenen Software die Schwelle von 100 Millionen Programmzeilen längst überschritten hat. Demgegenüber steht aber die wissenschaftliche Begriffsbildung erst bei einem Umfang von ungefähr 10.000 Programmzeilen. Die Problematik wird in einer Analogie anschaulich: Jeder Themenbereich in der Reihe Zelle-Organ-Mensch-Abteilung-Firma braucht seine spezifische Begriffswelt. Man kann nicht mit den Begriffen der Organebene angemessen über ein Unternehmen reden.

Der Rückstand der Informatikwissenschaft ist kein spezifisch deutsches, sondern ein globales Problem. Die Entscheidung, Software in Billiglohnländern wie Indien entwickeln zu lassen, löst das Problem nicht, sondern verschärft es nur noch. Denn wenn man noch nicht einmal innerhalb eines Teams aus 5 bis 10 Leuten, die zusammen in einem Raum sitzen, angemessen über Softwaresysteme kommunizieren kann, muss die Kommunikation über den halben Erdball hinweg zwangsläufig noch unbefriedigender sein.

Wenn dagegen einmal die Begriffsbildung und damit der Kommunikationswirkungsgrad ein angemessenes Niveau erreicht haben, gibt es tatsächlich keinen Grund mehr, der gegen eine Verteilung der Softwareentwicklung über den ganzen Globus sprechen würde. Denn im Gegensatz zur verteilten Produktion materieller Güter bringt ja die Verteilung der Softwareentwicklung praktisch keine Transportzeiten und -kosten mit sich.

Die wirtschaftliche Bedeutung von Software wird weiter zunehmen. Für die wirtschaftliche Zukunft Deutschlands wird es deshalb immer wichtiger, die Weichen so zu stellen, dass Deutschland im Softwareingenieurwesen eine vergleichbare Souveränität gewinnt, wie wir sie im Maschinenbau und dort speziell im Bereich der Kraftfahrzeugtechnik haben. Die Durchdringung aller technischen Bereiche mit Software macht es in Zukunft unmöglich, noch in irgendeiner Ingenieurdisziplin souverän zu bleiben, wenn die Souveränität im Softwareingenieurwesen fehlt. Diese zu erlangen muss ein strategisches Ziel der Wirtschaft und der Politik werden. Als erster und wichtigster Schritt muss die Trennung zwischen Naturwissenschaften und Ingenieurwissenschaften, die ein wesentlicher Grund für unsere Erfolge im Maschinenbau und der Elektrotechnik sind, analog auf den Softwarebereich übertragen werden. Das bedeutet konkret, dass die Informatik als Grundlagenwissenschaft behandelt werden muss, die sich auf Probleme konzentriert, welche durch Formalisierung und Algorithmisierung lösbar sind. Daneben muss ein unabhängiges Softwareingenieurwesen gesetzt werden, dessen Thema die arbeitsteilig zu realisierenden Softwaresysteme sind (siehe Bild 1).

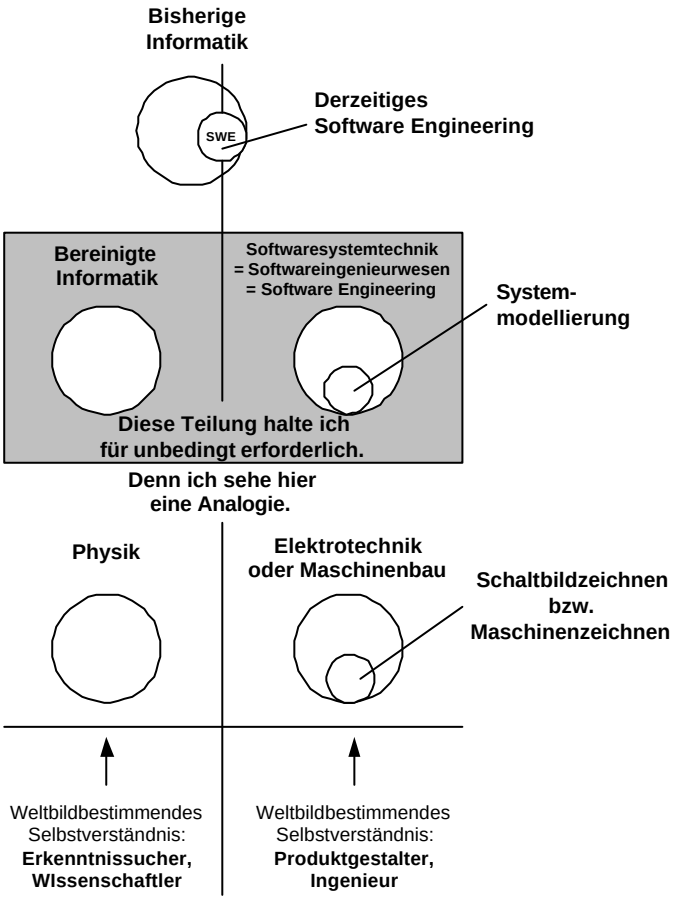


Bild 1 Gegenüberstellung von Grundlagenwissenschaft und Ingenieurwissenschaft

(2)

Was ist Ingenieurskultur?

(bisher unveröffentlichter Entwurf einer Rede, 2002)

Als Mission des HPI wurde ursprünglich einmal formuliert: „Das HPI hat die Mission, die Entstehung und Verbreitung einer Ingenieurskultur in der Softwareindustrie zu fördern." Kann man das erreichen an einer Universität, die keine Ingenieursfakultäten hat? Auf diese Frage werde ich zum Schluss meiner Ausführungen zurückkommen.

Häufig kann man die Gliederung eines Aufsatzes dadurch gewinnen, dass man jedem interessanten Begriff, der im Titel vorkommt, einen eigenen Abschnitt widmet. Im vorliegenden Falle sind es die Begriffe *Ingenieur* und *Kultur*. Der Kulturbegriff ist dabei der umfassendere, dem ich mich deshalb zuerst zuwende, damit ich ihn anschließend durch die Verbindung mit dem Ingenieursbegriff einschränken kann.

Wenn man beliebige Leute aus den sog. gebildeten Kreisen nach der Bedeutung des Wortes „Kultur" fragt, wird man vermutlich häufig nur ein hilfloses Drumherumgerede hören wie in vielen anderen Fällen auch, wenn man nach der Bedeutung eines alltagsüblichen Begriffs fragt. Es fällt mir und vermutlich vielen anderen schwer, das Wort „Kultur" kontextunabhängig zu interpretieren, denn es wird in unterschiedlichen Bedeutungen benutzt, und nur wenn man den Kontext der aktuellen Benutzung kennt, weiß man oder kann es zumindest ahnen, was gemeint ist. Wenn man von jemandem sagt, er habe keine Kultur, versteht man unter dem Wort Kultur sicher etwas anderes als wenn man sagt, im Kultusministerium liege das Feld der Kultur in den Händen eines Staatssekretärs.

In dem Aufsatz „Universität ohne Kultur?" des Trierer Philosophen Hinske findet man die folgenden Aussagen:

> Kultur meint eine bestimmte Qualität menschlicher Tätigkeiten, und zwar insbesondere solcher Tätigkeiten, die nicht auf die unmittelbare Herstellung eines Produkts gerichtet sind.
>
> Kultur bedeutet nicht die bloße durchschnittliche Ausführung einer Handlung, sondern deren Steigerung zur größtmöglichen Vollkommenheit.
>
> Kultur hat, wer eine Handlung so auszuführen weiß, wie es an sich der Sache entspricht. In diesem Sinne sprechen wir heute von Esskultur, Wohnkultur oder Lesekultur.

Es wird in diesen Aussagen herausgestellt, dass es bei der Kultur um eine größtmögliche Vollkommenheit einer Handlung geht – also beispielsweise des Essens, des Wohnens oder des Lesens, und nicht um die Vollkommenheit eines möglichen Ergebnisses der Handlung. Bezogen auf die Ingenieurskultur heißt das, dass es nach Hinske um die größtmögliche Vollkommenheit des Ingenieurseins geht und nicht um die Vollkommenheit der von den Ingenieuren hergestellten Produkte.

Das kann uns allerdings erst etwas bedeuten, nachdem wir geklärt haben, was denn das Typische des Ingenieurseins ist.

Zuvor aber sollen noch weitere Aussagen zum Kulturbegriff betrachtet werden, denn was Hinske meint, muss nicht unbedingt konsistent sein mit dem, was andere meinen.

Wenn man vor der Aufgabe steht, in einem Aufsatz oder einer Rede die Bedeutung bestimmter Wörter klar zu stellen, wird man nicht lange zögern, sondern zuerst einmal in einer oder mehreren Enzyklopädien nachlesen. Die beiden von mir herangezogenen Enzyklopädien waren Brockhaus und Meyers. Die Erklärungen zum Wort Kultur umfassen im Meyers zwei und im Brockhaus vier Seiten. In unseren Kontext passen vor allem folgende Aussagen:

Zuerst zwei Zitate aus dem Meyers:

> Kultur ist das von Menschen zu bestimmten Zeiten in abgrenzbaren Regionen aufgrund der ihnen vorgegebenen Fähigkeiten in Auseinandersetzung mit der Umwelt und ihrer Gestaltung in ihrem Handeln in Theorie und Praxis Hervorgebrachte.
>
> In seiner Anwendung als Begriff kritischer Wertung wird der Kulturbegriff auf die am Maßstab der Vernünftigkeit und des ethisch und ästhetisch Vertretbaren gemessenen und positiv bewerteten Kulturleistungen, Handlungsformen und Normensysteme der Menschen eingegrenzt.

Und nun zwei Zitate aus dem Brockhaus:

> In einem engeren Sinne bezeichnet Kultur die Handlungsbereiche, in denen der Mensch auf Dauer angelegte und den kollektiven Sinnzusammenhang gestaltende Produkte, Produktionsformen, Lebensstile, Verhaltensweisen und Leitvorstellungen hervorzubringen vermag, weswegen dieser Kulturbegriff nicht nur das jeweils Gemachte, Hergestellte und Künstliche betont, sondern auch das jeweils moralisch Gute der Kultur anspricht.
>
> Im jeweiligen besonderen Kontext meint Kultur eine spezifische, von anderen Gruppen und Verhaltensnormen unterscheidbare Menge gemeinsamer Verhaltensweisen und Sachverhalte, die für eine bestimmte Teilgruppe der Gesellschaft oder eine ganze Gesellschaft typisch ist.

Es ist genau diese letzte Definition, die im Wort „Ingenieurskultur“ gemeint ist. Es geht also um Verhaltensweisen und Sachverhalte, die für die Teilgruppe der Gesellschaft, auf die der Ingenieurbegriff passt, typisch sind und in denen sie sich vom Rest der Gesellschaft unterscheidet. Wenn dem HPI die Mission vorgegeben ist, die Entstehung und Verbreitung einer Ingenieurskultur in der Softwareindustrie zu fördern, dann heißt dies, dass die Entstehung und das Wachsen einer Gruppe von Fachleuten gefördert werden soll, deren hauptsächliches Tätigkeitsfeld in der Softwareindustrie liegt und auf die Verhaltensweisen und Sachverhalte zutreffen, die traditionell den Ingenieursbegriff kennzeichnen.

Was sind denn nun diese ingenieurstypischen Verhaltensweisen und Sachverhalte? Wenn man Ingenieure fragt, was denn das Wesen des „Ingenieurseins“ sei, erhält man i.a. keine philosophisch befriedigende klare Antwort, denn Ingenieure werden i.a. nie gefragt, was denn das Typische sei, weswegen ihre Tätigkeit als Ingenieurstätigkeit bezeichnet wird. Da erhält man oft von Nichtingenieuren viel klarere Antworten, weil man dort eher zu wissen glaubt, was

das Ingenieurtypische sei. Interessanterweise wurde der Begriff „Software Engineering“ im Jahre 1968 von Nichtingenieuren, insbesondere von Mathematikern geprägt.

Zum Wort „Ingenieur“ fand ich in den beiden Enzyklopädien die folgenden Erklärungen:

> Bis ins 18. Jahrhundert war „Ingenieur“ vor allem die Bezeichnung für den Kriegsbaumeister. Die zivile Bedeutung des Wortes bildete sich im Laufe der Industrialisierung als Sammelbezeichnung für die höheren technischen Berufe verschiedener Fachrichtungen heraus.
>
> Aufgabe des Ingenieurs ist es, auf der Grundlage natur- und technikwissenschaftlicher Erkenntnisse und unter Berücksichtigung wirtschaftlicher und gesellschaftlicher Belange technische Werke zu planen und zu konstruieren sowie die Ausführung des Geplanten leitend anzuordnen und zu überwachen.

Diese letzte, von einem Herrn Tuchel stammende Definition erscheint mir sehr treffend bezüglich der Ingenieure in den traditionellen Disziplinen Bauwesen, Maschinenbau und Elektrotechnik. Deren Grundlagen sind zweifellos natur- und technikwissenschaftliche Erkenntnisse, und ihre Aufgabe ist es, technische Werke zu planen und ihre Ausführung zu leiten.

Die Übertragung dieser Definition auf die Softwareingenieure ist jedoch problematisch, denn deren Tätigkeit beruht nicht auf naturwissenschaftlichen Erkenntnissen. Nur wenn man Softwaresysteme als technische Werke einstuft und wenn man die Erkenntnisse, die der Tätigkeit von Softwareingenieuren zugrundeliegen, als technikwissenschaftliche Erkenntnisse klassifiziert, kann man die Definition des Herrn Tuchel unverändert für die Softwareingenieure übernehmen - was ich für äußerst wünschenswert halte. Da halte ich es sogar für zweckmäßig, nötigenfalls den traditionellen Technikbegriff ein wenig zu erweitern.

Im Brockhaus und im Meyers findet man zum Wort „Technik“ u.a. die folgenden Aussagen:

> Man versteht heute unter der Technik im eigentlichen Sinne die Gesamtheit aller Objekte (Werkzeuge, Geräte, Maschinen u.a.), Maßnahmen und Verfahren, die vom Menschen durch sinnvolle, zielgerichtete Ausnutzung der Naturgesetze und –prozesse sowie geeigneter Stoffe hergestellt bzw. entwickelt werden und sich zweckmäßig und in einem jeweils als nützlich betrachteten Ausmaß insbesondere bei der Arbeit und in der Produktion, aber auch im Bereich des Informations- und Kommunikationswesens anwenden lassen.
>
> Die neuere interdisziplinäre Technikforschung bevorzugt einen Technikbegriff, der beinhaltet:
> (1) die Menge der nutzenorientierten künstlichen materiellen Gebilde, der sog. Sachsysteme;
> (2) die Menge menschlicher Handlungen und Einrichtungen, in denen Sachsysteme entstehen;
> (3) die Menge menschlicher Handlungen, in denen Sachsysteme verwendet werden.
>
> Die gelegentlich als neuartige „abstrakte“ oder „transklassische“ Technik aufgefasste Programmierung elektronischer Datenverarbeitungsgeräte lässt sich dem dritten Bereich der Definition zuordnen, da sie eine besondere Fertigkeit für die Verwendung von Computern darstellt.

Diese Zitate aus den Enzyklopädien zeigen, dass der Technikbegriff bereits so aktualisiert wurde, dass er umfassend genug ist für unseren Zweck, die Tuchelsche Ingenieurdefinition auch auf Softwareingenieure anwenden zu dürfen. Ich wiederhole diese Definition noch einmal:

> *Aufgabe des Ingenieurs ist es, auf der Grundlage natur- und technikwissenschaftlicher Erkenntnisse und unter Berücksichtigung wirtschaftlicher und gesellschaftlicher Belange technische Werke zu planen und zu konstruieren sowie die Ausführung des Geplanten leitend anzuordnen und zu überwachen.*

An dieser Stelle möchte ich noch einmal auf die Festlegung von Hinske zurückkommen, der sagte, dass nur dort Kultur sei, wo das menschliche Handeln in größtmöglicher Vollkommenheit geschieht. Damit man dem Ingenieur zu Recht eine Ingenieurskultur zugestehen kann, genügt es also nicht, dass er seine Aufgabe irgendwie erfolgreich zu Ende bringt, sondern er muss so handeln, dass man sich nicht mehr vorstellen kann, wie er noch wesentlich besser hätte handeln können.

Es sind zwei sehr unterschiedliche Felder, auf denen sich die Arbeit von Ingenieuren hauptsächlich abspielt, und die Frage nach der höchstmöglichen Vollkommenheit des Handelns muss für jedes der beiden Felder getrennt beantwortet werden. Das eine Feld ist charakterisiert durch die Anwendung von Mathematik zum Zwecke der Optimierung der geplanten Werke und der Optimierung des Ausführungsprozesses. Das andere Feld ist charakterisiert durch die Anwendung von Abstraktionskonzepten und Darstellungsformen zum Zwecke der optimalen Kommunikation mit den am arbeitsteiligen Planungs- und Ausführungsprozess beteiligten Personen, damit diese optimale Beiträge zum Prozess liefern können. Hierzu zähle ich auch die Klassifikation von Aufgabenstellungen und Lösungsalternativen durch angemessene Abstraktion. Ich möchte im Folgenden das erste als das Mathematikfeld und das zweite als das Kommunikationsfeld bezeichnen.

Solange auf beiden Feldern die Distanz zur Vollkommenheit noch sehr groß ist, darf man einer technischen Disziplin sicher noch keine Ingenieurskultur zusprechen. Auch die Frage, ob es bereits eine Ingenieurskultur gibt, wenn nur auf einem Feld eine Nähe zur Vollkommenheit erreicht ist, möchte ich verneinen. Weder die Kombination eines vollkommenen Mathematikfeldes mit einem mangelhaften Kommunikationsfeld noch die Kombination eines mangelhaften Mathematikfeldes mit einem vollkommenen Kommunikationsfeld geben Anlass, von einer Ingenieurskultur zu sprechen.

Die Entstehung einer Ingenieurskultur in den traditionellen technischen Bereichen Bauwesen, Maschinenbau und Elektrotechnik vollzog sich in einheitlicher Weise derart, dass die Nähe zur Vollkommenheit zuerst auf dem Kommunikationsfeld erreicht wurde und erst danach auf dem Mathematikfeld.

An der Technischen Universität Karlsruhe gibt es einen Hörsaal, der den Namen Redtenbacherhörsaal trägt. Jacob Ferdinand Redtenbacher lebte von 1809 bis 1863 und war von 1841 bis zu seinem Tode Professor für Maschinenbau am damaligen Karlsruher Polytechnikum, welches später zur Technischen Hochschule wurde. Im Ehrenhof der Hochschule, auf den man gelangt, wenn man durch das Tor des Hauptgebäudes geht, steht eine Büste dieses Mannes. Viele ehrende Schriften nennen Professor Redtenbacher den „Begründer des Maschinenbaus als Ingenieurwissenschaft“.

Sein Beitrag zur Entstehung einer Ingenieurskultur im Maschinenbau bestand darin, dass er das Mathematikfeld, welches vor ihm im Maschinenbau praktisch gar nicht existierte, zu einer

erstaunlichen Reife brachte. Vor ihm wurden zwar auch schon viele Maschinen gebaut, aber Berechnungen gab es praktisch nicht. Aus seinem umfangreichen Werk seien hier nur drei besonders typische Bücher genannt:

Theorie und Bau der Turbinen
Theorie und Bau der Wasserräder
Die Gesetze des Lokomotivbaues

Bezüglich des Kommunikationsfeldes fand Redtenbacher schon eine vollkommenheitsnahe Situation vor, denn die Praxis des technischen Zeichnens war längst gereift und es bestand bereits eine angemessene Begriffswelt zur Kommunikation über Maschinen.

Wenn ich den Maschinenbau mit der Softwarebranche vergleiche, sehe ich hier eine umgekehrte Entwicklung: Damit in der Softwarebranche eine Ingenieurskultur geschaffen wird, muss nicht das Mathematikfeld zur Reife gebracht werden, denn dieses wurde meiner Einschätzung nach durch die Bemühungen vieler hervorragender Informatiker bereits nahe an die Vollkommenheit herangeführt. Der große Schritt, der zu einer Ingenieurskultur in der Softwarebranche führt, muss auf dem Kommunikationsfeld getan werden. Dessen Reifegrad muss heute noch als äußerst mangelhaft beurteilt werden.

Zu Beginn dieses Aufsatzes zitierte ich die Formulierung der Mission des HPI: „Das HPI hat die Mission, die Entstehung und Verbreitung einer Ingenieurskultur in der Softwareindustrie zu fördern.“ Und ich stellte die rhetorische Frage, ob man dies an einer Universität erreichen könne, die keine Ingenieursfakultäten hat. Nun kann ich diese rhetorische Frage beantworten: Selbstverständlich kann man das erreichen, denn auch in einer Universität ohne Ingenieursfakultäten kann die Notwendigkeit einer Ingenieurskultur erkannt werden, und die Probleme, deren Lösung zu einer Ingenieurskultur in der Softwarebranche führt, können verstanden werden. Wenn das Problem klar erkannt ist, wird die Förderung der Lösungsbemühungen nicht ausbleiben.

(3)

Das Kommunikationsproblem der Informatiker und ihre Unfähigkeit, es wahrzunehmen

(aus "Grundlagenstudien aus Kybernetik und Geisteswissenschaft", Band 39, 2/1998)

1. Die Kritiker in der Minderheit

Wer aus der frischen Luft des Waldes kommend einen miefigen Saal betritt, wird selbstverständlich die Luft im Saal als schlecht bezeichnen, wogegen diejenigen, die sich schon lange in diesem Mief aufgehalten haben, gar nicht wahrnehmen können, dass es hier stinkt. Sie werden allerdings auch nicht kämpferisch behaupten, die Luft im Saal sei besonders gut und besser als die im Wald.

So behaupten auch die Informatiker nicht, dass sich ihr Kommunikationsverhalten durch besonders hohe Qualität auszeichne, aber sie sind durchaus der Meinung, dass die Art und Weise, wie sie kommunizieren, ganz in Ordnung sei. Somit kann es für sie auch keinen Anlass geben, über eine Verbesserung der Qualität ihres Kommunikationsverhaltens nachzudenken.

Derjenige, der von draußen reingekommen ist und dabei gemerkt hat, wieviel schlechter die Luft im Saal als die im Wald ist, steht vor der Frage, ob er die schlechte Luft im Saal als gegeben hinnehmen muss, oder ob er nach Möglichkeiten suchen soll, die Luft im Saal zu verbessern. Dabei kann er verständlicherweise keine Unterstützung von den Leuten im Saal erwarten, denn diese sind ja mit der derzeitigen Qualität der Luft ganz zufrieden. Es ist dagegen sogar möglich, dass sie seinem Bemühen, den Saal zu lüften, Widerstand entgegensetzen, weil dadurch ihr gemütliches Beisammensein unnötigerweise gestört wird. Derjenige, der weiß, wie schlecht die Luft im Saal ist und wieviel besser sie sein könnte, wird möglicherweise versuchen, die Leute im Saal zu überreden, für eine kurze Zeit den Saal zu verlassen und mit ihm an die frische Luft zu gehen. Er ist nämlich überzeugt, dass die Leute, wenn sie aus der frischen Luft wieder in den Mief des Saales zurückkehren, von ganz alleine dafür sorgen werden, dass der Saal gelüftet wird. Aber warum sollte jemand, der gemütlich im Saal sitzt und sich wohlfühlt, auf jemanden hören, der ihn bittet, mit ihm einmal kurz nach draußen zu gehen?

Eine klare schlüssige Begriffswelt und die dieser Begriffswelt angemessenen Darstellungsmittel stellen in dieser Analogie die Entsprechung zur frischen Luft dar. Da die Informatiker aber gar keinen Grund sehen, nach einer klareren Begriffswelt und besseren Ausdrucksmitteln zu suchen, nehmen sie entsprechende Angebote gar nicht zur Kenntnis oder, falls sie sie doch zur Kenntnis nehmen, machen sie sich nicht die Mühe, sie genau zu studieren. Sie verhalten sich wie jemand, dem man die Konstruktionspläne für ein Perpetuum Mobile vorlegt, der doch weiß, dass es ein Perpetuum Mobile gar nicht geben kann und dass es deshalb eine verlorene Mühe wäre, die Pläne genau zu studieren.

Derjenige, der den Unterschied zwischen frischer Luft und Mief kennengelernt hat, kann vorläufig nur in den Sälen, für deren Luft er selbst verantwortlich ist, den Mief vertreiben. Hinsichtlich der anderen Säle aber, worin die anderen Leute zur Zeit noch so selbstzufrieden hocken, braucht er nur zu warten, bis dort die Mangelerscheinungen so offenkundig werden, dass man dann auch dort gerne die Existenz frischer Luft zur Kenntnis nehmen wird.

2. Die Mangelerscheinungen

Nachdem bisher das Problem nur anhand einer Analogie charakterisiert wurde, soll es nun konkretisiert werden, d.h. es sollen nun die Mangelerscheinungen aufgeführt werden, die man mühelos feststellen kann, wenn man die Kommunikationsformen der Informatik an Maßstäben misst, die man aus anderen Disziplinen gewohnt ist.

Man betrachte die folgenden Beispiele aus Informatiktexten:

- Rechenvorgänge sind aus Rechenschritten, den Operationen zusammengesetzt. Eine Operation ist eine Anweisung oder ein Klartext.

- Das Prädikat für die Eingabe ist read(x). Dieses Ziel liest einen Term vom aktuellen Eingabestrom.

- Durch die Auftrennung eines Objekts in einen Objekttyp und einen Objektnamen lassen sich bequem Systeme beschreiben.

- Das Ereignis beschreibt das Eingetretensein eines Zustands, der eine Folge bewirkt.

- Not every object required to carry out an operation flows through its trigger.

- The control condition does not continuously check if the condition is true.

- Each activity in the chart may contain a control activity whose role is to supervise the behaviour of its siblings. The internal descriptions of control activities are given by statecharts. A statechart starts activities through actions on transitions. When an activ ity with a control activity is started, the control activity and its associated statechart are started. (Die hier gestiftete Verwirrung entspricht derjenigen, die entstünde, wenn in einem Text der Unterschied zwischen einem Sänger, seinem Gesang, seinen Noten und dem Konzertsaal verwischt würde.)

Man muss selbstverständlich akzeptieren, dass es disziplinspezifische Fachsprachen gibt, die nicht jeder Laie verstehen kann. Wer keine Noten lesen kann, gehört nicht zum vorgesehenen Leserkreis eines Aufsatzes über den Kontrapunkt. Man muss also fragen, ob die oben angeführten Beispielstexte in einer Informatikfachsprache abgefasst sind, so dass sie nur dem Laien konfus erscheinen, aber für die Informatiker klar verständlich sind. Diese Frage lässt sich leicht beantworten, indem man die Texte überdurchschnittlich begabten und sehr gut ausgebildeten Informatikern vorlegt und sie um eine Interpretation bittet. Dann stellt sich heraus, dass diese Texte auch von den Informatikern als konfus erlebt werden und sie auf Vermutungen bezüglich der Interpretation angewiesen sind.

Nun könnte man natürlich einwenden, dass es in jeder Disziplin möglich sei, Beispielstexte von unfähigen Autoren zu zitieren. Deshalb komme den angeführten Beispielen keinerlei Beweiskraft zu. Gegen einen solchen Vorwurf kann hier tatsächlich kein wissenschaftliches Argument vorgebracht werden, denn dieses könnte ja nur darin bestehen, dass man auf sehr viele unabhängige andere Wissenschaftler verweist, die in ähnlicher Weise wie der Autor über zwei Jahrzehnte hinweg das Kommunikationsverhalten der Informatiker studiert haben und dabei erkennen mussten, dass in dieser Disziplin ein unbeholfenes "Schauderwelsch" nicht die Ausnahme, sondern die Regel ist.

Wer die Texte Hegels mit den Texten Schopenhauers vergleicht, ist geneigt, Schopenhauer Recht zu geben, der behauptet, Hegel könne nicht klar schreiben, und daraus müsse man zwangsläufig schließen, dass er auch nicht klar denken könne.

Darf man also aus den konfusen Texten der Informatiker schließen, dass sie nicht klar denken können? Dies wäre sicher kein logisch zwingender Schluss. Es ist durchaus denkbar, dass sich in einer Disziplin die Gepflogenheit herausgebildet hat, sehr klare Sachverhalte sehr umständlich auszudrücken gemäß der Redewendung "Warum etwas einfach machen, wenn es auch umständlich geht." Man kann nie sicher sein zu wissen, was die Leute zu dem assoziieren, was sie reden oder schreiben. Dennoch glaubt der Autor aufgrund seiner jahrelangen intensiven Beobachtung des Kommunikationsverhaltens der Informatiker zu der Vermutung berechtigt zu sein, dass die Schwierigkeiten der Informatiker daher kommen, dass sie sich bei ihrer Kommunikation grundsätzlich auf einer niederen Abstraktionsebene bewegen, auch wenn es um Sachverhalte geht, die erst auf einer höheren Abstraktionsebene überhaupt ausdrückbar werden. Man könnte sagen, dass sie den Wald vor lauter Bäumen nicht sehen, genauer gesagt, dass sie nur den Begriff Baum und das dazugehörige Wort kennen, aber nicht den Begriff Wald und dementsprechend auch kein zugehöriges Wort. Eine Fortführung dieser Überlegungen wird weiter unten angeschlossen; an dieser Stelle geht es ja nur um die Charakterisierung von Mangelerscheinungen.

Jeder kennt aus seinem elementaren Mathematikunterricht die Bedeutung des Gleichheitszeichens. Wenn also jemand die Form

$$x = x + 2$$

sieht, wird er denken, es handle sich um eine Gleichung mit einer Unbekannten, deren Auflösung zu einer Zahl führen müsste, die ihren Wert nicht ändert, wenn man 2 hinzuaddiert. Da es eine solche Zahl nicht gibt - außer in der Welt der Kardinalzahlen für unendliche Mengen -, wird der Leser diese Gleichung als unlösbar bezeichnen.
Wenn er dagegen den Ausdruck

$$x := x + 2$$

sieht, wird er entweder nicht wissen, wie er diesen Ausdruck interpretieren soll, oder aber er wird - falls er programmieren kann - damit die Vorstellung der Anweisung

$$x_{nachher} = x_{vorher} + 2$$

verbinden, die verlangt, dass auf den aktuellen Inhalt der Speicherzelle x die Zahl 2 hinzuaddiert werden soll - was nur möglich ist, wenn die Speicherzelle vorher schon eine Zahl enthält. Er wird zumindest aber nicht auf die Idee kommen, es könne sich um eine Gleichung handeln, denn die Verbindung des Doppelpunkts mit einem Gleichheitszeichen erzeugt eine Vorstellung von Unsymmetrie, die sich mit der intuitiven Vorstellung der Symmetrie der beiden Waagschalen auf den beiden Seiten einer Gleichung nicht verträgt.

Während in der Programmiersprache FORTRAN noch das Gleichheitszeichen als Zuweisungsoperator verwendet wurde, hat man bereits in der Programmiersprache ALGOL60 die unsymmetrische Symbolkombination := für den Zuweisungsoperator festgelegt. Zweifellos bedeutete es einen Rückschritt, als man bei der Schaffung der Programmiersprache C wieder zu der in FORTRAN üblichen Symbolisierung zurückkehrte. Damit hat man nicht nur die Unsymmetrie des Zuweisungsoperators verwischt, sondern man hat auch die Notwendigkeit geschaffen, das Gleichheitszeichen dort, wo es eigentlich hingehört, durch etwas anderes zu ersetzen. In Bedingungsprädikaten, wo man eigentlich den Text

```
IF  j = 5  THEN  ...
```

erwarten würde, findet man nun

```
IF  j = = 5  THEN  ...
```

Häufig kommt es vor, dass ein C-Programmierer versehentlich

```
IF  j = 5  THEN  ...
```

schreibt, was vom Compiler noch nicht einmal als Fehler festgestellt werden kann, weil in C bei der Berechnung von Prädikaten auch Zustandsänderungen als Seiteneffekte akzeptiert werden.

Man kann der weltweiten Informatikergemeinde den Vorwurf nicht ersparen, dass sie das Qualitätsbewusstsein bezüglich menschenfreundlicher Kommunikationsformen entweder nie gehabt oder inzwischen längst verloren hat.

Wenn ein Architekt ein Gebäude entwirft, ein Maschinenbauer eine Getränke-Abfüllanlage konstruiert oder ein Elektroingenieur eine Automatisierungselektronik konzipiert, zeichnen sie Pläne - Baupläne, Konstruktionspläne oder Schaltpläne. Ihre Tätigkeit bezeichnen sie als entwerfen, konstruieren oder planen, aber selbstverständlich nicht als dokumentieren, obwohl die Pläne ja auch Dokumente sind.

In der Informatik unterscheidet man zwischen Mitdokumentation und Nachdokumentation, um auszudrücken, ob während oder erst nach der Softwareentwicklung dokumentiert wird. Durch diese Vorsilben wird dem Wort Dokumentation Gewalt angetan, denn umgangssprachlich bezeichnet Dokumentation immer eine nachträgliche Zusammenstellung von Dokumenten. Man denke an eine Dokumentation der französischen Revolution oder der Gründung der heutigen Humboldt-Universität in Berlin. Die Erstellung von Dokumentationen ist normalerweise eine Aufgabe für Historiker. Das Wort Dokumentation gibt keinerlei Hinweis auf die Art der verwendeten Dokumente - es können Texte sein, Fotographien, Zeichnungen und sogar Gegenstände wie beispielsweise die Amtskette eines Universitätsrektors.

Wenn man demgegenüber von Plänen spricht, verbindet man damit unwillkürlich die Vorstellung bestimmter Plantypen, d.h. also von Darstellungen unter Verwendung bestimmter genormter Formen. In der Informatik hat sich über viele Jahre praktisch niemand die Frage gestellt, wie denn die Pläne für ein Betriebssystem oder ein Textverarbeitungssystem aussehen könnten. Es sind zwar etliche Systeme zur rechnerunterstützten Dokumentation mit Hilfe graphischer Darstellungen auf den Markt gekommen - die Kürzel SA, SADT, SART oder UML stehen für derartige Darstellungen -, aber dass selbst die Anbieter solcher Systeme ihren Plantypen keine große Bedeutung beigemessen haben, erkennt man daran, dass die Softwaresysteme zur Edition, Verwaltung und Transformation solcher Pläne selbst nicht mit dieser Art von Plänen entworfen wurden.

Die Nichtexistenz allgemein akzeptierter und brauchbarer Pläne musste zwangsläufig zur Entstehung sogenannter Gurus führen. Ein Guru ist ein Softwareentwickler, der die inneren Strukturen eines großen Softwaresystems im Kopf hat und zu dem jeder kommen muss, der etwas Spezielles über das Innere des Systems wissen will. In den traditionellen Ingenieurwissenschaften gibt es kein derartiges Guruwesen, denn dort werden die Fragen anhand von Plänen beantwortet, die jeder Ingenieur lesen kann. Da alle Gurus eine Machtstellung haben, die sie nicht gerne verlieren möchten, bringen die Gurus immer sehr viele Gründe vor, weshalb die Einführung von Plänen in der Informatik nicht möglich sei.

Die Informatik hat ihre wissenschaftlichen Qualitätskriterien aus der Mathematik bezogen, wo die Konzentration auf das Formale bzw. Formalisierbare zweifellos gut begründet ist. Nun sind aber die Gegenstände der Informatik keine mathematischen Gegenstände, sondern technische Systeme, obwohl der renommierte Informatiker Hoare [Hoare-86] sagt:

"Programs are mathematical expressions." Eine solche Aussage entspricht in ihrem Nutzen der Aussage: Ein Auto ist nur ein Haufen von Elementarteilchen. Die Aussage ist zwar richtig, aber es gibt keinerlei praktische Konsequenzen, die man aus ihr ziehen könnte. Es gibt allerdings sehr viel unsinnige Konsequenzen, die man aus einer solchen Aussage ziehen kann und die leider in der Informatik auch gezogen werden.

Es ist durchaus verständlich, weshalb jemand, der sich einmal in der Welt der Mathematik eingenistet hat, diese Welt nicht mehr gerne verlässt. Die Welt der Mathematik ist eine viel klarere reinere Welt als die Welt der Ingenieure, denn in der Mathematik geht es immer um die Entscheidung zwischen wahr und falsch, während es bei den Ingenieuren um Entscheidungen zwischen angemessen und unangemessen oder zweckmäßig und unzweckmäßig geht, und da sind die Entscheidungskriterien selbstverständlich nicht mehr eindeutig.

Die Unterschiede zwischen der mathematischen und der nichtmathematischen Welt fallen schon jedem Schüler auf, der Schwierigkeiten hat, Textaufgaben zu lösen, obwohl er doch seine Mathematik gut beherrscht. Er kann rechnen, aber er kann ein Problem nicht in Formeln überführen. Das Nichtformale bzw. die Brücke zwischen dem Nichtformalen und dem Formalen ist aber eigentlich das interessantere, denn das Formale alleine kann man der Maschine übergeben, die dann die sturen Rechenschritte durchführt. Wer sich auf das Formale konzentriert, sieht somit seinen primären Kommunikationspartner nicht in anderen Menschen, sondern in der Maschine. Deshalb ist er auch schnell bereit, sich mit einer hieroglyphenähnlichen Symbolik zufriedenzugeben, solange sie nur eindeutig ist. Deshalb ist es kein Wunder, dass der Quellcode häufig das einzige ist, was von einem großen Softwaresystem als Dokumentation zur Verfügung steht.

Welche Bedeutung das Nichtformale für das menschliche Verständnis hat, erkennt man schon an sehr einfachen Beispielen: In Bild 1 sind die fünf Peano'schen Axiome sowohl in der üblichen prädikatenlogischen Symbolik als auch in einer graphischen Veranschaulichung dargestellt. Es wäre völlig absurd anzunehmen, es könne jemand, der die natürlichen Zahlen als Begriff noch nicht kennt, zu diesem Begriff geführt werden, indem man ihm einfach die prädikatenlogischen Formeln vorsetzt. Es ist selbstverständlich, dass auch Peano selbst den Begriff der natürlichen Zahl schon kannte, bevor er anfing, über ihre axiomatische Festlegung nachzudenken. Aufgrund der graphischen Darstellungen in Bild 1 erkennt jeder leicht, dass die ersten vier Axiome nicht ausreichen, die natürlichen Zahlen eindeutig festzulegen; hätte man aber diese Graphen nicht, sondern nur die prädikatenlogischen Formeln der ersten vier Axiome, hätte man vermutlich große Schwierigkeiten zu begründen, weshalb man noch ein fünftes Axiom braucht.

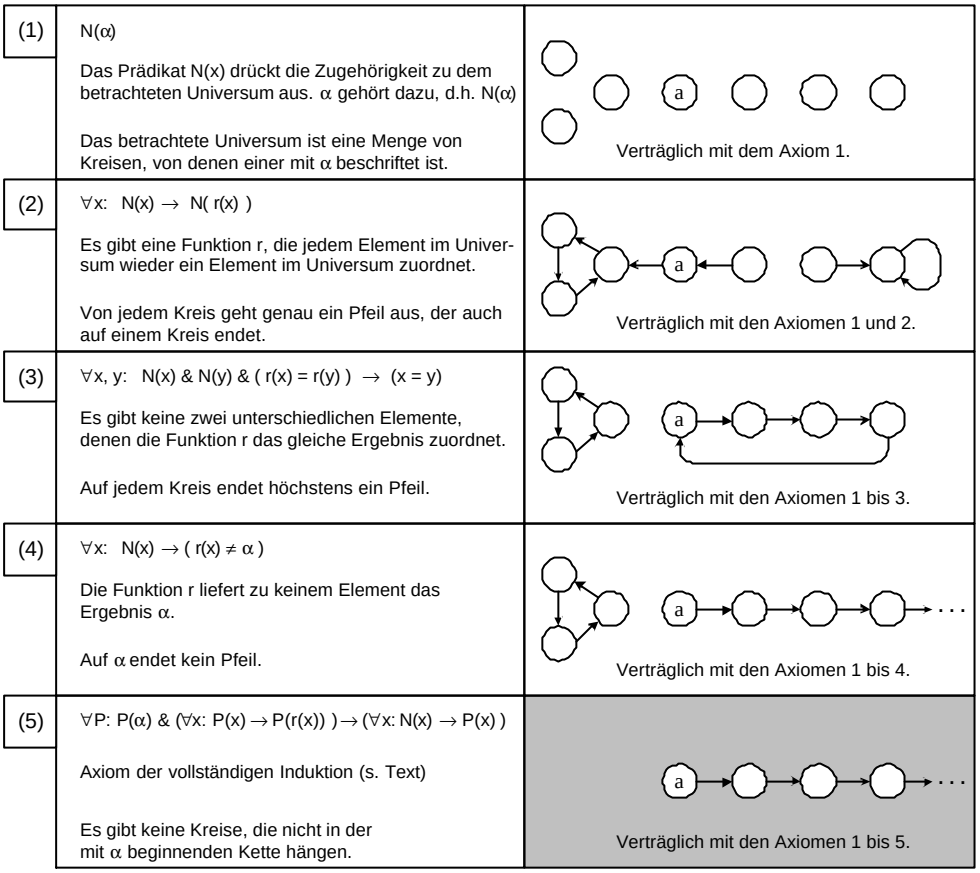


Bild 1 Die Axiome von Peano

Da die Informatiker über mathematische Sachverhalte untereinander und mit anderen sehr gut kommunizieren können, ist ihre Kommunikationsschwäche eine Konsequenz der Tatsache, dass sie überwiegend über nichtmathematische Sachverhalte kommunizieren müssen, sie dies aber bisher nicht als Problem erkannt haben oder ihm zu wenig Aufmerksamkeit widmeten.

3. Die Besonderheiten des Problems und seiner Lösung

Die Produkte, über die in den traditionellen Ingenieurwissenschaften kommuniziert werden muss, sind immer gegenständliche Produkte, die man sehen und anfassen kann. Deshalb ähneln die Darstellungen auf den Plänen auch immer den Ansichten, die man sehen kann, wenn man das Produkt von einer bestimmten Seite oder in einer bestimmten aufgeschnittenen Form betrachtet. Bei Softwareprodukten dagegen gibt es solche Ansichten nicht, denn dort ist das einzig konkret Wahrnehmbare der Quellcode und die Ein- Ausgabeerscheinungen auf dem Bildschirm oder auf dem Druckerpapier. Man muss also in der Informatik eine größere Abstraktionsleistung erbringen als in den traditionellen Ingenieurwissenschaften, wenn man die Individuen finden will, die man auf den Plänen symbolisiert. Dass dieses Problem in der Anfangszeit der Informatik fast von niemandem und bis heute nur von recht wenigen erkannt wurde, ist zwar dem Autor, der dieses Problem schon vor über 20 Jahren in Angriff nahm, nicht wirklich verständlich, aber er versucht wenigstens, Argumente zu sammeln, die das Versäumnis entschuldbar erscheinen lassen. In der Anfangszeit der Informatik waren die Softwaresysteme so klein, dass es möglich war, alles Wesentliche in Form des Quellcodes zu vermitteln. Solange man nur eine Hundehütte zusammennagelt, braucht man sich keine Gedanken über die Gestaltung von Bauplänen zu machen.

Das Problem ist erst im Laufe der Jahre dringlicher geworden, als die Programmsysteme immer komplexer wurden. Dass dennoch fast niemand das Problem erkannte, lässt sich möglicherweise durch eine Analogie erklären: Es wurde experimentell nachgewiesen, dass sich Leute, die in einem anfänglich wohltemperierten Raum sitzen, dessen Temperatur mit der Zeit langsam erhöht wird, erst bei sehr hoher Temperatur eines Problems bewusst werden und anfangen zu überlegen, wie man die Situation verbessern könne.

Die "Temperatur im Informatikraum" ist nun inzwischen doch so hoch geworden, dass immer mehr Informatiker anfangen, über eine Abhilfe nachzudenken. Inzwischen gibt es nämlich Softwaresysteme, an denen über tausend Entwickler etliche Jahre gearbeitet haben. Die Manager, welche die Verantwortung für derart komplexe Systeme tragen, sehen immer deutlicher die Gefahren, die darin bestehen, dass die vielen Beteiligten über diese Systeme nicht mehr angemessen kommunizieren können.

Der Autor, der das Kommunikationsproblem vor über 20 Jahren zum Schwerpunkt seiner Forschungsarbeit gemacht hat, hat in der Zwischenzeit selbstverständlich nicht nur das Problem analysiert, sondern er hat unter Mithilfe mehrerer Generationen junger Wissenschaftler eine Begriffswelt und die zugehörigen Darstellungsmittel entwickelt, die das Problem auf befriedigende Weise lösen. Die Angemessenheit der Begriffswelt und der Darstellungsmittel konnte inzwischen in mehreren großen Projekten mit der Industrie nachgewiesen werden.

Vermutlich war der Autor gegenüber den meisten Informatikkollegen bezüglich der Lösungssuche dadurch im Vorteil, dass er über die Systemtheorie und nicht über die Programmierung zur Informatik kam. Somit konnte er Software als Information über eine Klasse dynamischer informationeller Systeme ansehen, woraus er den Schluss zog, dass nicht die Software der primäre Kommunikationsgegenstand sei, sondern das dynamische informationelle System. Er brauchte also nur nach geeigneten Begriffen zur Modellierung dynamischer informationeller Systeme zu suchen. Während er noch vor Jahren wegen seines Ansatzes von Wissenschaftskollegen ausgelacht wurde, konnte er zu seiner Genugtuung vor kurzem von einem renommierten Informatiker [Jackson-98] die Aussage lesen: "A program is a description of a machine: a general-purpose computer accepts the description and, by executing it, becomes the machine described."

Da die weltweite Informatik erst jetzt zu dieser Einsicht gelangt ist, hatte sie selbstverständlich noch keine Zeit, daraus die angemessenen Konsequenzen zu ziehen. Und da in der Informatik die Neigung vorherrscht, jedes Rad, welches man in den traditionellen Ingenieurwissenschaften schon lange kennt, noch einmal zu erfinden, ist zu vermuten, dass man trotz der nun vorliegenden Einsicht nicht auf die Idee kommen wird, in den alten Schriften der Systemtheoretiker nachzuschauen, was man von dort übernehmen könne.

4. Betrachtung des Problems unter Bezug auf das Sprachwertedreieck

Das in [Frank-96] dargestellte Sprachwertedreieck geht von einer Klassifikation der Sprachen in die drei Klassen Sachsprachen, Ritualsprachen und Marktsprachen aus und stellt deren Merkmale - Genauigkeit, Verständlichkeit und Glaubwirksamkeit - zueinander in Beziehung.

Das hier betrachtete Problem kann nur durch eine angemessene Sachsprache gelöst werden. Bei der Entwicklung einer solchen Sachsprache musste der Autor Semantik und Syntax gleichzeitig schaffen, d.h. es ging nicht nur darum, einen bekannten Semantikbereich mit einer angemessenen Syntax zu verbinden.

Abgesehen von der mathematischen Begriffswelt, wo die Informatiker eine bewährte Sachsprache verwenden, haben sich die Informatiker ein Sprachverhalten angewöhnt, bei dem sowohl die Genauigkeit als auch die Verständlichkeit fehlen. Ihre Sprache erfüllt sicher das Kriterium einer Ritualsprache, denn sie vermittelt zweifellos den typischen Informatikerstallgeruch, und außerdem hat sie viele Kennzeichen einer Marktsprache, denn sie wird intensiv dazu genutzt, Produkt- und Produzentenmängel zu verschleiern.

5. Literatur

[Frank-96] Helmar Frank: Bildungskybernetische Sachsprache im Sprachwertedreieck. Grundlagenstudien aus Kybernetik und Geisteswissenschaft - grkg, Band 37, 1996, Heft 4, S. 184-195.

[Hoare-86] C.A.R. Hoare: The Mathematics of Programming. Clarendon, Oxford, 1986.

[Jackson-98] Michael Jackson: Will there ever be Software Engineering? IEEE Software, Februar 1998, S. 36-39.

[Wendt-91] Siegfried Wendt: Nichtphysikalische Grundlagen der Informationstechnik. 2. Auflage. Springer-Verlag, Berlin; 1991.

(4)

Besonderheiten des Softwareingenieurwesens im Vergleich mit den klassischen Ingenieurdisziplinen

(aus "Das ist Informatik", Jörg Desel (Hrsg.), Springer Verlag, 2001)

Ingenieurwissenschaften werden erforderlich, wenn das Basteln technischer Produkte nicht mehr zu befriedigenden Ergebnissen führt. Durch die Ingenieurwissenschaften wird die Entwicklung und die Herstellung technischer Produkte auf eine solide wissenschaftliche Basis gestellt. Diese ist gekennzeichnet durch eine angemessene Begriffswelt, dazu passende Darstellungsformen und Methoden für das systematische Gewinnen von Ergebnissen. Die Ingenieurwissenschaft läuft zwangsläufig immer der Bastelei hinterher: Die Thermodynamik kam erst nach der Dampfmaschine.

Wer nicht weiter darüber nachdenkt, könnte voreilig zu dem Schluss kommen, jede Ingenieurdisziplin sei im Grunde angewandte Physik und angewandte Mathematik. Zweifellos erledigt ein Ingenieur in den traditionellen Disziplinen immer wieder Aufgaben der angewandten Physik und der angewandten Mathematik. Dennoch wäre es nicht angemessen, einen sogenannten angewandten Physiker oder einen angewandten Mathematiker als Ingenieur zu bezeichnen. Das wesentliche Kennzeichen des Problembereichs der Ingenieure ist die Befassung mit komplexen Systemen. Und dabei geht es gerade um die nicht mathematisch behandelbaren Aspekte dieser Systeme, also nicht um Erkenntnisse und Entscheidungen, die man berechnen kann, sondern um Entscheidungen, die man nur mehr oder weniger zweckmäßig oder unzweckmäßig fällen kann. Es geht dabei nicht immer nur um die Konstruktion neuer Systeme, sondern auch um Fragen ihrer evolutionären Veränderung oder ihrer Kopplung mit anderen Systemen.

Die Lehrinhalte eines ingenieurwissenschaftlichen Studienganges müssen immer von der Frage her gefunden werden, was die Absolventen gelernt haben sollten, damit sie den Aufgaben, denen sie in der Industrie begegnen, gewachsen sind. Die Frage, ob es sich dabei auch um Lehrinhalte handelt, die für den Hochschullehrer wissenschaftlich interessant sind, sollte immer erst an zweiter Stelle gestellt werden.

Die Komplexität der Systeme bringt es zwangsläufig mit sich, dass die Systeme nur arbeitsteilig erstellt und verändert werden können. Da der Ingenieur eines Tages durchaus in der Lage sein sollte, die Verantwortung für den gesamten Prozess der Systemgestaltung und Pflege zu übernehmen, muss er primär im Hinblick auf diesen arbeitsteiligen Prozess ausgebildet werden und nicht im Hinblick auf die Optimierung einzelner Schritte dieses Prozesses. Der Ingenieur kann immer davon ausgehen, dass er Spezialisten für Spezialaufgaben zur Verfügung haben wird. Er muss zuallererst einmal als „Kommunikator technischer Inhalte" ausgebildet werden, der seine Planungen unmissverständlich weitergeben kann und der mit seinen Spezialisten optimal kommunizieren kann. Es gehört unbedingt zum Weltbild des Ingenieurs, dass ihm immer der Unterschied zwischen einem technischen Produkt und dem Ergebnis einer Bastelei, und sei sie auch noch so genial, bewusst bleibt. Labormuster und technische Produkte wird ein Ingenieur nie verwechseln.

Das Typische der Ingenieursarbeit wird nicht durch den Komponentenbau bestimmt, sondern durch die Systemkonzeption. Hierfür gibt es keine formale Spezifikation und demzufolge keine formal unterstützbaren Lösungswege.

Eine zweckmäßige Systemkonzeption kann dem Ingenieur nur gelingen, wenn er auf zwei unterschiedlichen Bereichen über Erfahrungen verfügt: Zum einen muss er klare Vorstellungen über die funktionelle Vielfalt machbarer Komponenten haben, und zum anderen muss er viele aktuell in der Vergangenheit realisierte Systeme kennen gelernt haben.

In den traditionellen Ingenieurwissenschaften wie Maschinenbau und Elektrotechnik gibt es selbstverständlich eine Fülle von Katalogen, in denen eine Vielfalt von Komponenten in funktioneller Klassifikation zusammengestellt sind. Für die vorliegende Betrachtung sind dabei nicht die Kataloge für die elementaren Komponenten wie Schrauben, Zahnräder, Widerstände oder Transistoren interessant, sondern die Kataloge für Komponenten, die bereits aus elementaren Bauteilen aufgebaut wurden. Man denke an Getriebe, Pumpen, Verstärker oder Antennen. Entsprechend kann sich ein Softwareingenieur auf Kataloge für Algorithmen stützen. Die Ausbildung im Softwareingenieurwesen muss dahin fuhren, dass der Ingenieur mühelos die Kataloge verstehen kann, um darin zielgerichtet nach geeigneten Komponenten für sein System zu suchen. Die Ausbildung hat nicht primär das Ziel, ihn in die Lage zu versetzen, neue Beiträge für die Kataloge zu schaffen.

Ein System entsteht immer als Ergebnis einer Iteration von Entwürfen. Ein Entwurf wird im Grunde immer von einem einzelnen Menschen geschaffen - dies gilt nicht nur für den Bereich der Ingenieurentwürfe, sondern überall, wo das Wort Entwurf einen Sinn hat, also beispielsweise auch für Gesetzesentwürfe oder für Entwürfe von Tapetenmustern. Die Entscheidung, ob ein Entwurf realisiert werden soll, wird nicht allein von demjenigen gefällt, der den Entwurf geschaffen hat, sondern diese Entscheidung ist die Konsequenz einer Diskussion mehrerer Fachleute. Deshalb müssen die Entwürfe so leicht fassbar dargestellt werden, dass die zu beteiligenden Fachleute überhaupt in die Lage kommen, in der Diskussion eines vorgelegten Entwurfs konstruktive Beiträge zu leisten.

Die zentrale Rolle der Systemdarstellung ist in Bild l veranschaulicht. Es sind drei Aktivitäten gezeigt, die zyklisch voneinander abhängen. Damit man sich ein Systemkonzept ausdenken kann, sollte man vorher schon Systeme, die in der Vergangenheit realisiert wurden, kennen gelernt haben. Damit man Systeme kennen lernen kann, müssen diese leicht fassbar dargestellt worden sein. Und damit man ein System darstellen kann, muss dieses zuvor von jemandem erdacht worden sein.

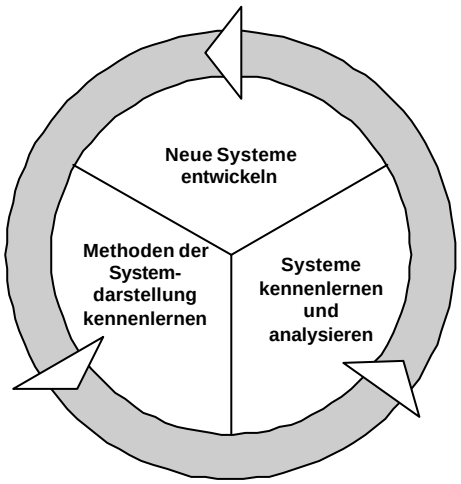


Bild 1 Inhaltskategorien der Ingenieurausbildung

Unter den drei in Bild 1 aufgeführten Aktivitäten begründet nur eine die Existenzberechtigung der Ingenieure, und das ist die Entwicklung neuer Systeme. Daraus darf man aber nicht den Schluss ziehen, die Ausbildung von Ingenieuren müsse sich auf diese Aktivität konzentrieren, weil die Ingenieure besonders befähigt werden sollten, neue Ideen und neue Lösungen zu finden. Man kann niemandem beibringen, gute Einfälle zu haben. Da das eine Feld in Bild l also nicht direkter Gegenstand der Lehre sein kann, muss sich die Lehre auf die anderen beiden Felder konzentrieren. Man muss daher den Studenten beibringen, wie Systemkonzepte darzustellen sind, und man muss ihnen sehr viele unterschiedliche Systeme aus der Vergangenheit vorstellen. Letzteres ist nur möglich, wenn man zuvor die Darstellungsmethoden gelehrt hat.

In den traditionellen Ingenieurwissenschaften wirft das Darstellen von Systemkonzepten keine besonderen Schwierigkeiten auf, denn im Bereich gegenständlicher Systeme ist das Darstellungsproblem fast trivial. Das Bild 2 veranschaulicht den grundsätzlichen Unterschied der Darstellungsproblematik anhand einer Gegenüberstellung der traditionellen Ingenieurwissenschaften und dem Softwareingenieurwesen.

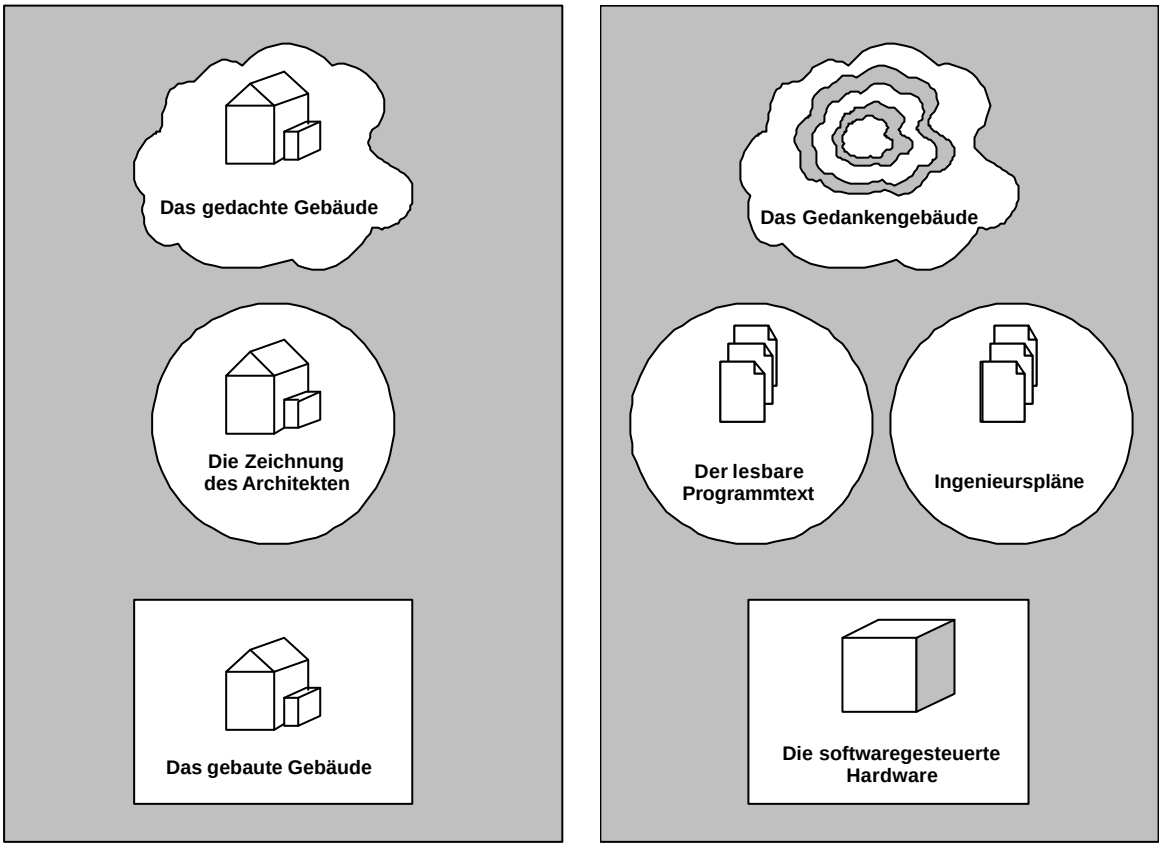


Bild 2 Konsequenzen natürlicher Sichtbarkeit bzw. Unsichtbarkeit von Ingenieursprodukten

Links geht es um ein Gebäude, das zuerst nur vor dem geistigen Auge des Architekten steht. Anschließend erstellt der Architekt eine Zeichnung, damit nun jedermann sehen kann, was zuvor nur der Architekt sah. Und wenn dann schließlich das Gebäude steht, sieht es so aus wie auf der Zeichnung.

In der rechten Hälfte der Grafik geht es um ein informationstechnisches System. Dieses wird dadurch realisiert, dass sehr viel Softwaretext erstellt wird, der dann ein Computersystem dazu bringt, ein benutzbares Gerät mit bestimmter Funktionalität zu werden. Im Gegensatz zum Gebäudearchitekten sieht der Ingenieur der Softwaresystemtechnik vor seinem geistigen Auge keine reale Struktur. Im Bild ist dies durch die Zeichnung eines wolkenartigen Gebildes zum Ausdruck gebracht. Ganz unten ist in Form eines Würfels das reale System skizziert. Man muss sich hier die Computergehäuse, die Kabel, die Bildschirmgeräte, die Tastaturen und andere Geräte, die zum Computersystem gehören, vorstellen. Links über diesem Würfel sieht man die Dokumente, auf denen der Programmtext steht, der viele Millionen Zeilen umfassen kann. Wollte man den zu einem komplexen Softwaresystem gehörenden Programmtext auf Papier ausdrucken, erhielte man häufig Papierstapel von der Höhe von Wolkenkratzern. Rechts neben dem Programmtext findet man die Zeichnungen der Ingenieure des Softwareingenieurwesens, aus denen die restlichen Strukturen überhaupt erst verständlich werden. Die begrifflichen Kategorien und Konzepte, die hinter diesen Ingenieurszeichnungen der Softwaresystemtechnik stehen, sind nicht trivial und müssen den Studenten erst über mehrere Semester hinweg beigebracht werden.

Die Reife einer Darstellungsmethodik zeigt sich daran, dass technologische Revolutionen keine wesentlichen Änderungen der Darstellungsmittel erfordern. So kann man beispielsweise die Reife der Konzepte des Bauzeichnens daran erkennen, dass der Übergang von einem Barockschloss zum Empire State Building keine grundsätzlichen Änderungen dieser Konzepte erforderte.

(5)

Softwareingenieurspläne können auch für Nichtfachleute verständlich sein.

(aus "Betriebswirtschaftliche Blätter", 53. Jahrgang, 3/2004)

Die Häufigkeit scheiternder oder alle Aufwandsschätzungen sprengender IT-Projekte ist eine Folge des Fehlens bewährter Methoden. Führungskräfte, die IT-Projekte ganz oder teilweise delegieren müssen, können noch nicht einmal die Durchführung zeitnah kontrollieren, sondern müssen oft blind vertrauen. Das liegt vor allem daran, dass Darstellungsformen fehlen, die es Außenstehenden ermöglichen würden, erreichte Zwischenzustände im Projektverlauf zu verstehen und zu bewerten. Der nachfolgende Beitrag skizziert einen möglichen Weg aus der Misere.

Das Verhältnis von Vertrauen und Kontrolle

„Vertrauen ist gut, Kontrolle ist besser." Dieser Spruch, der von Lenin stammen soll, kennzeichnet das Spannungsfeld, worin sich jeder Manager bewegen muss. Der Erfolg umfangreicher Projekte hängt von der guten Arbeit vieler unterschiedlicher Fachleute ab, an die die Verantwortung für die Erledigung von Teilaufgaben delegiert werden muss. Derjenige, der eine Aufgabe delegiert, muss darauf vertrauen, dass die Aufgabe erfolgreich in seinem Sinne erledigt wird. Aber wie kann er wissen, ob sein Vertrauen gerechtfertigt ist? In den Fällen, in denen es bewährte Verfahren zur Erledigung der Aufgaben gibt, sind im Allgemeinen auch Fachleute bekannt, die ihre Fähigkeit zur erfolgreichen Anwendung dieser Verfahren bereits in ausreichendem Maße nachgewiesen haben. Diesen Fachleuten kann der Auftraggeber „blind vertrauen", ohne dadurch ein bedenkliches Risiko einzugehen. Es gibt aber viele Fälle, in denen ein blindes Vertrauen mit einem sehr hohen Risiko für den Auftraggeber verbunden ist. Je „klarer die Sicht" des Auftraggebers ist, d.h. je mehr Kontrollmöglichkeiten er nutzen kann und nutzt, umso kleiner ist sein Risiko. Am sichersten kann er sich fühlen, wenn er zu jedem beliebigen Zeitpunkt vor Abschluss der Aufgabe den jeweils erreichten Zwischenzustand beobachten und bewerten kann. Wenn er diese Möglichkeit in ausreichend kurzen Zeitabständen nutzt, kann er im Bedarfsfall rechtzeitig korrigierend eingreifen und den mit Fehlentwicklungen verbundenen Schaden gering halten.

Die Möglichkeit ausreichender Kontrolle ist an die Voraussetzung geknüpft, dass die vom Kontrolleur zu bewertenden Zwischenzustände der Aufgabenerledigung in angemessener Form beobachtbar sind. Die Frage nach dieser Form im Falle von IT-Projekten bildet den Kern des vorliegenden Aufsatzes.

IT-Projekte gehören zur Zeit noch nicht zu den Aufgaben, für die es bewährte Verfahren der Aufgabenerledigung gibt. Man erkennt dies an der großen Zahl bekannt gewordener Misserfolge – man denke hier beispielsweise an den aktuellen Fall Toll-Collect –, zu der noch eine viel größere Dunkelziffer hinzuaddiert werden muss. Es gibt somit keine Fachleute, an die man IT- Projekte in blindem Vertrauen delegieren kann, ohne dadurch ein sehr großes Risiko einzugehen. Deshalb liegt es im Interesse der verantwortlichen Auftraggeber, die Projekte in ausreichendem Maße kontrollieren zu können. Zur Zeit aber müssen sie es noch resignierend hinnehmen, dass ihnen die nötige Kontrolle nicht möglich ist, weil die Zwischenergebnisse der Aufgabenerledigung nicht in angemessener Form vorliegen. Es bleibt ihnen also gar nichts anderes übrig, als blind zu vertrauen und auf ein gutes Ende zu hoffen.

Verständlichkeit für Nichtfachleute als Bewertungskriterium

Wer delegiert ist gegenüber dem, an den er delegiert, im Allgemeinen in der Rolle des Nichtfachmannes. So kann man beispielsweise ein Haus bauen lassen, obwohl man nicht im Bauwesen ausgebildet wurde. Selbstverständlich kann der Bauherr auch als Nichtfachmann den Projektfortschritt kontrollieren, weil er die Zwischenergebnisse unmittelbar sehen und verstehen kann. Er kann die für ihn relevanten Informationen aus den Bauplänen entnehmen und er kann mühelos das Wachsen des Gebäudes verfolgen. Auch die neben der Erscheinung des Gebäudes für ihn besonders relevanten Informationen über Kosten und Termine kann er den Plänen entnehmen und auf Plausibilität prüfen. Diese ideale Situation, in der der Nichtfachmann in ausreichendem Maße Kontrolle ausüben kann, ist in der natürlichen Sichtbarkeit des angestrebten Ergebnisses, also des Gebäudes, begründet. Die Baupläne sind geringfügig abstrahierende Zeichnungen dessen, was man am Projektende als konkretes Ergebnis sehen kann. Im Falle von IT-Projekten ist die Situation bekanntermaßen nicht so einfach, denn hier äußert sich nur ein sehr kleiner Teil des angestrebten Endergebnisses in sichtbarer Form. Der weitaus größere Teil sind die in der Software verborgenen gedachten Strukturen, und die haben, wie alle Gedanken, keine natürliche Sichtbarkeit. Der größte Teil dessen, was Gegenstand einer Kontrolle sein müsste, befindet sich „nur in den Köpfen“ der Systemgestalter und entzieht sich somit jeglicher Kontrolle von außen.

Diese Problematik ist besonders heikel, weil die jeweils wissenden Fachleute mehrheitlich gar keine Notwendigkeit sehen, mit Nichtfachleuten über fachliche Themen zu kommunizieren. Deshalb sehen sie auch keinen Mangel darin, dass sie zu einer solchen Kommunikation gar nicht fähig sind, zumindest nicht mit einem akzeptablen Wirkungsgrad. Besonders brisant wird die Problematik noch durch eine starke Korrelation, die bisher kaum jemandem aufgefallen zu sein scheint: In seinen jahrzehntelangen Beobachtungen des Kommunikationsverhaltens in IT-Projekten hat der Autor festgestellt, dass der Wirkungsgrad der Wissensweitergabe durch IT-Fachleute unabhängig davon ist, ob der jeweilige Adressat ein Fachkollege oder ein Nichtfachmann ist. Das heißt, dass ein IT-Fachmann, dem es nicht gelingt, einem Nichtfachmann in angemessener Zeit ein bestimmtes für den Adressaten relevantes Wissen zu vermitteln, es auch nicht oder nur mit inakzeptabel hohem Aufwand schafft, dieses Wissen an einen Fachkollegen weiterzugeben. Damit offenbart sich der Kommunikationswirkungsgrad mit Nichtfachleuten als zentraler Schlüssel für Produktivität und Qualität im Softwarewesen.

Systemarchitekten als Vermittler

Man darf nicht annehmen, dass es möglich sei, der Gesamtheit der IT-Fachleute die Motivation und Fähigkeit zur allgemeinverständlichen Darstellung von Zwischenergebnissen ihrer kreativen Arbeit zu vermitteln. Die meisten von ihnen muss man mit Künstlern vergleichen, die zwar zum Teil geniale Werke schaffen, aber nicht motiviert oder fähig sind, ihren Schöpfungsakt als eine Menge zeitlich partiell geordneter Schritte zu strukturieren, die über verständlich dargestellte Zwischenergebnisse zum fertigen Werk führen. Da aber sowohl ihre Art kreativer Arbeit als auch die allgemeinverständlichen Darstellungen der Zwischenergebnisse gebraucht werden, müssen in einem IT-Projekt zwei unterschiedliche Typen von Fachleuten zusammen arbeiten. Die Systemarchitekten garantieren durch ihre Darstellungen den Auftraggebern, also den Nichtfachleuten, die Kontrollierbarkeit der Projekte, und sie organisieren auf der Grundlage dieser Darstellungen die arbeitsteilige Systemrealisierung.

Die Programmentwickler hingegen schaffen das Endprodukt in Form von Programmtexten. Auch ihre Arbeitsergebnisse sind Informationsdarstellungen, aber nur solche, die von Nichtfachleuten nicht verstanden werden müssen.
In Abbildung 1 sind die Kommunikationswege zwischen den drei Typen von Projektbetroffenen – den Nichtfachleuten, den Systemarchitekten und den Programmentwicklern – veranschaulicht. Die genaue Erklärung der verwendeten symbolischen Notation folgt weiter unten.

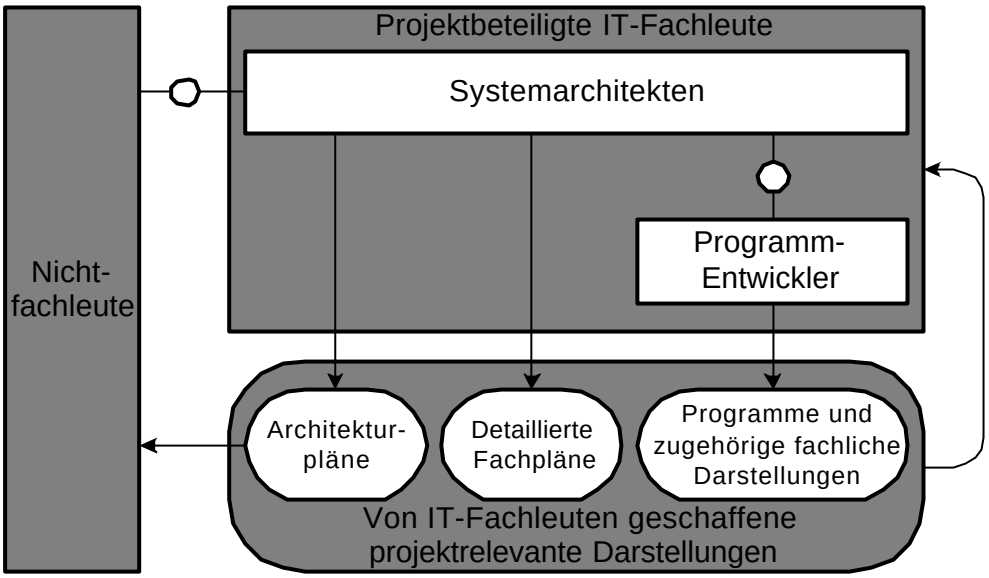


Abb. 1 Kommunikationswege in IT-Projekten

Die Konzepte der Systemkartographie

Wenn Nichtfachleute im Zusammenhang mit Software das Wort Strukturplan hören, denken die meisten spontan an Ablaufdiagramme, denn diese werden in der Anfangsphase fast aller Programmierkurse eingeführt. Ablaufpläne können aber nicht die Darstellungen sein, an denen sich die Projektkontrolle primär orientiert, denn Ablaufpläne stellen in der Praxis immer etwas dar, was nur einen Systemausschnitt und nicht das ganze System betrifft. In einem komplexen System wie beispielsweise dem System R/3 von SAP gibt es keinen sinnvollen Ablauf, der das ganze System beschäftigt, so wenig wie es in einer Großstadt wie beispielsweise Berlin eine Buslinie gibt, welche die ganze Stadt befährt. In der Analogie zwischen IT-Systemen und Großstädten müssen die gesuchten Darstellungen den Stadtplänen entsprechen. Ein Stadtplan zeigt selbstverständlich nicht alles, was es über die Stadt zu wissen gibt, aber er stellt die primäre Referenz für sämtliche weiteren Darstellungen dar, die zusätzliches Wissen über die Stadt vermitteln. So werden beispielsweise die Darstellung der Linienführung und des Fahrplanes einer städtischen Buslinie selbstverständlich unter Bezug auf den Stadtplan erstellt.

Die bewährten Plantypen, die man in den traditionellen Ingenieursdisziplinen als primäre Referenzpläne findet wie Bauzeichnungen im Bauwesen, Konstruktionszeichnungen im Maschinenbau und Schaltpläne in der Elektrotechnik geben den Hinweis, was der Inhalt der gesuchten Referenzpläne für IT-Systeme sein sollte: Es muss ein statischer Systemaufbau gezeigt werden.

Da IT-Systeme keine sichtbare Struktur haben, muss eben ein virtueller, d.h. denkbarer Aufbau gezeigt werden. Dabei eignen sich weniger die Bau- und Konstruktionszeichnungen als Vorbilder für IT-Systempläne, sondern die Stadtpläne und elektrischen Schaltbilder. Diese zeigen Wege, auf denen etwas zwischen Orten fließen kann, wobei an den Orten etwas verweilen, entstehen, verschwinden oder verändert werden kann. Im Falle der Stadtpläne sind dieses „etwas" die Verkehrsteilnehmer und im Falle der elektrischen Schaltbilder ist es der elektrische Strom, die elektrische Energie oder es sind die nachrichtentechnischen Signale. Was dieses Fließende, Verweilende, Erzeugbare, Eliminierbare und Veränderbare im Falle von IT-Systemen sein muss, liegt nahezu auf der Hand: Es muss Information sein.

Ein Beispiel eines solchen Plans, der Wege zwischen Informationserzeugern, -bearbeitern und -verbrauchern zeigt, wurde bereits mit der Abbildung 1 eingeführt. Die großen rund berandeten und beschrifteten Knoten sind die Symbole für die Informationsspeicher. Die beschrifteten Rechtecke sind als Akteure zu verstehen, die Information erzeugen und verarbeiten und miteinander kommunizieren können, indem sie Informationen verschicken und empfangen. Die Wege, auf denen die Kommunikationsinformation fließen kann, also die so genannten Kanäle, sind durch kleine unbeschriftete Kreise dargestellt. Alle rund berandeten Knoten, also sowohl die Speicher als auch die Kanäle, können als Beobachtungsorte gedeutet werden, an denen Information liegt oder vorbeifließt. Diese Orte sind die Aktionsfelder, auf denen sich das Agieren der Akteure äußert.

So sagt der Plan unter anderem aus, dass die Systemarchitekten Architekturpläne erzeugen, die von Nichtfachleuten angeschaut und verstanden werden können, und dass die Systemarchitekten mit den Nichtfachleuten und den Programmentwicklern über Kanäle – man denke an direkte oder telefonvermittelte Gespräche – kommunizieren können. Der Plan zeigt auch, dass zwischen den Entwicklern und den Nichtfachleuten kein Kontakt besteht, weder über Speicher noch über Kanäle.

In dem Plan der Abbildung 1 kommt ein Pfeil vor, der zwei Zusammenfassungsknoten miteinander verbindet: Alle Speicher, in denen irgendwelche Darstellungen liegen, die im Laufe des Projektes von IT-Fachleuten geschaffen werden, sind zu einem umfassenden Speicher zusammengefasst worden. Und alle projektbeteiligten IT-Facheute sind zu einem einzigen Akteur abstrahiert worden, wobei die Unterscheidung zwischen Systemarchitekten und Entwicklern und damit auch das Wissen um den Kanal zwischen ihnen verloren gehen musste. Der die beiden „höheren Knoten" verbindende Pfeil besagt, dass irgendwelche IT-Fachleute irgendwelche projektrelevanten Darstellungen anschauen und verstehen können.

So wie die Einhaltung der Rechtschreibregeln die Lesbarkeit von Texten fördert, wird auch die Lesbarkeit der hier vorgestellten Strukturpläne gefördert, wenn man auch bei der Bildung höherer Knoten die Regel einhält, dass keine Verbindungslinien zwischen zwei Rundknoten oder zwei Rechteckknoten vorkommen dürfen.

Die Bildung höherer Knoten ist ein Konzept, welches auch bei der Gestaltung von Stadtplänen und elektrischen Schaltbildern genutzt wird. Dieses Konzept schafft die Möglichkeit, vergröberte oder verfeinerte Pläne zu zeichnen. Ohne dieses Konzept könnte man die Pläne immer nur in der feinsten Auflösung zeichnen, in der jedes Detail sichtbar wird. Dadurch würden die Pläne für komplexe Systeme unüberschaubar groß werden und möglicherweise sogar die Fläche eines Fußballfeldes überschreiten.

Darstellungen ergänzender Aspekte

Ein Aufbauplan sagt nur über einige interessierende Aspekte des Systems etwas aus. Weitere Aspekte müssen durch zusätzliche andersartige Darstellungen vermittelt werden, wobei diese aber stets auf den Aufbauplan Bezug nehmen. Insbesondere sind es zwei zusätzliche Aspekte, zu denen man neben der Kenntnis des Aufbaus noch Informationen benötigt, um das System im Prinzip zu verstehen. Der eine Aspekt betrifft die Frage nach den Eigenschaften des Fließenden bzw. Verweilenden und Veränderbaren. Im Falle des Stadtplans wäre dies die Frage nach den Eigenschaften der Verkehrsteilnehmer, im Falle der IT-Systeme ist es die Frage, welche Informationen in den Speichern liegen oder auf den Kanälen fließen können. Der zweite Aspekt betrifft die Frage nach den Prozessen, die sich im Aufbau abspielen können. Dieser Aspekt wird durch die bereits erwähnten Ablaufpläne erfasst.

Weil die Abläufe nicht notwendigerweise kausal sequentiell sind, braucht man ein Darstellungskonzept, welches sich sowohl zur Darstellung sequentieller als auch nicht sequentieller, so genannter „nebenläufiger“ Prozesse eignet. Hier nutzt man das seit langem bekannte Konzept der Petrinetze (s. Abbildung. 2). Der Name verweist auf den Erfinder dieser Netze, den Mathematiker Carl Adam Petri. Ein Petrinetz ist ein Netz aus sog. Stellen (=Kreisknoten) und sog. Transitionen (=Rechteckknoten). Die beschrifteten Transitionen stellen die einzelnen Schritte des Ablaufs, d.h. die Aktionen der im Aufbau vorkommenden Akteure dar, deren zeitliche Ordnung sich aus dem Verlauf der sogenannten Markierung der Stellen ergibt. Die Markierung einer Stelle stellt man sich als Belegung der Stelle mit einer Marke vor wie die Belegung eines Tellers mit einer Münze. Ein Ablaufschritt kann auftreten, wenn alle seine Eingangsstellen belegt sind. Durch den Schritt werden alle seine Eingangsmarken „verbraucht“, und auf alle seine Ausgangsstellen werden neue Marken hingelegt. Eine Stelle kann Eingangsstelle mehrerer Transitionen sein, wie beispielsweise die Stelle nach dem ersten Prüfungsschritt in Abbildung 2, wodurch Entscheidungen bezüglich der Schrittfolge erforderlich werden.

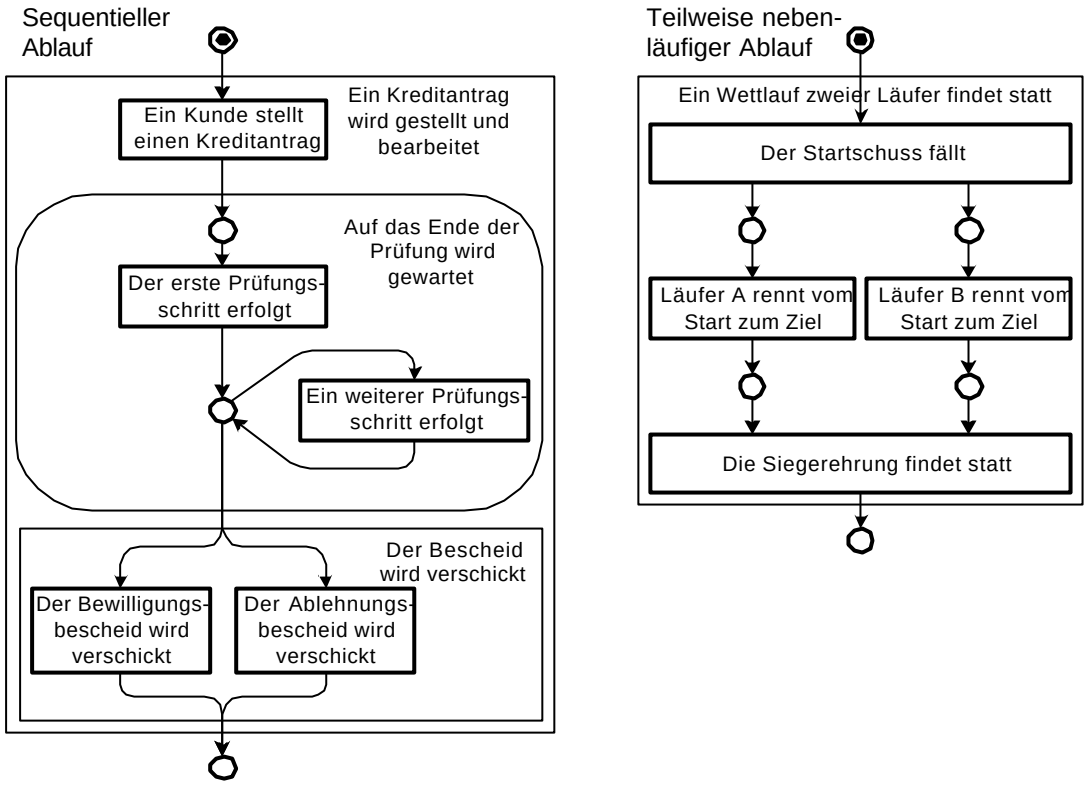


Abb. 2 Zwei Beispiele von Ablaufplänen in Form von Petrinetzen

Auch die Pläne zur Darstellung des im System fließenden, verweilenden oder verarbeiteten „Materials“, also die Informations- oder Datensemantikpläne, sind graphische Darstellungen mit Rund- und Rechteckknoten. Ein Datensemantikplan ist eine „formalgraphische Übersetzung“ eines Textes, der einen Informationstyp beschreibt. Subjekte und Objekte und werden in Rundknoten übersetzt. Sie stehen für Klassen konkreter oder abstrakter Dinge; man spricht von Entitäten. Prädikate beschreiben Relationen zwischen den Entitäten; sie werden in Rechteckknoten übersetzt. Adjektive und Relativsätze, welche den Subjekten oder Objekten beigegeben sind, äußern sich als Attributangaben in den runden Entitätsknoten. Entsprechend äußern sich Adverben und adverbiale Bestimmungen, die den Verben in den Prädikaten beigegeben sind , als Attributangaben in den rechteckigen Relationsknoten. Ein Beispiel ist in Abbildung 3 gezeigt. Solche Strukturpläne zur Erfassung von Datensemantik sind schon lange unter der Bezeichnung „Entity Relation Diagrams (ERD)“ bekannt.

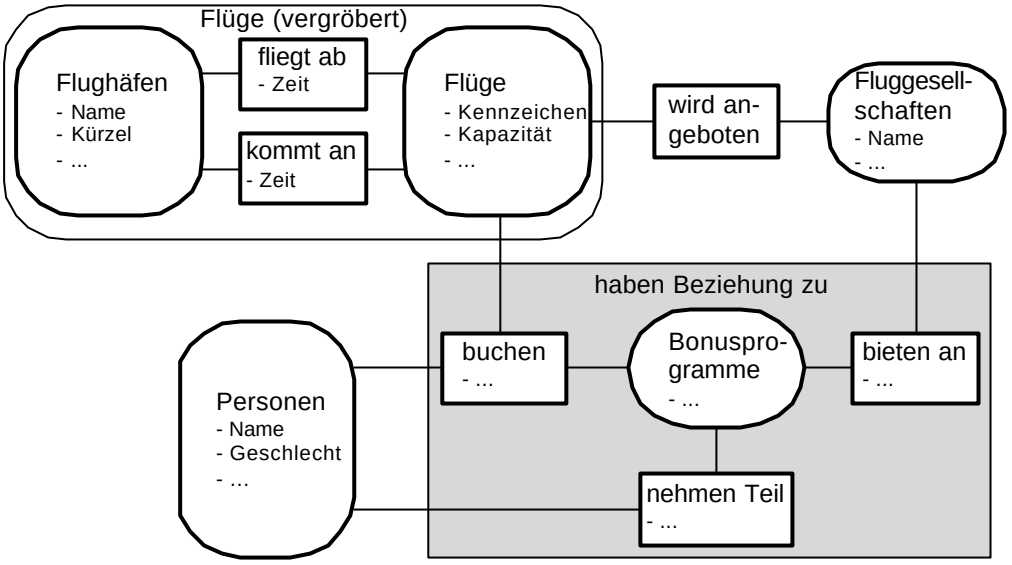


Abb. 3 Beispiel eines Entity-Relationship-Diagramms

Nichtfachleute müssen die Initative ergreifen

Zu beweisen, dass projektverantwortliche Nichtfachleute schon heute IT-Projekte wirksam kontrollieren können, war aus Platzgründen im Rahmen des Beitrags nicht möglich. Denn für einen überzeugenden Beweis wäre es nötig, ein konkretes komplexes Projekt als Beispiel vorzustellen. Außerdem wurde gar nicht gezeigt, dass die vorgestellte FMC-Modellierung („Fundamental Modeling Concepts“) mit den drei Plantypen für Aufbau, Ablauf und Datensemantik tatsächlich ein optimaler Ausgangspunkt für die Entwickler ist, die zu den Planelementen passende Programmabschnitte konstruieren müssen. Der Autor hofft aber, dass es ihm wenigstens gelungen ist, bei denjenigen Lesern Zweifel zu wecken, die bisher überzeugt waren, sie müssten das hohe Risiko der Nichtkontrollierbarkeit von IT-Projekten fatalistisch als ein in der Natur der Sache liegendes Phänomen hinnehmen. Die Kontrollierbarkeit wird ihnen allerdings nicht in den Schoß fallen, d.h. sie werden sie nicht bekommen, wenn sie einfach nur darauf warten, dass ihnen die IT-Fachwelt ein entsprechendes Geschenk macht. Vielmehr müssen die bisher in Unmündigkeit Gehaltenen ihre wirtschaftliche und wirtschaftspolitische Macht einsetzen, um die IT-Fachwelt zu einer entsprechenden Kulturänderung zu zwingen.

(6)

Principles for Planning Curricula in Software Engineering

(Bisher unveröffentlichte Vorlage für den Wissenschaftlichen Beirat des HPI, 2001)

This paper is a consequence of my observation that todays software engineers have not learned to give high quality reports on complex software systems.

My expectations concerning the quality of system descriptions is determined by my experience as an electrical engineer. I am used to apply certain standards of quality when I grade engineering level system descriptions. To make it clear: I am not talking about the quality of the systems, but about the quality of the descriptions. My attention had been focussed on this problem since 1974. In all these years, I had not seen or heard a single description of a complex software system, outside of the sphere of my own academic influence, meeting my quality standards for descriptions of engineering products. And I am convinced that I know why software developers can't make good descriptions presenting their knowledge: They have never seen good examples. And the reason for all the complaints about missing or poor documentation is quite clear to me: It's the consequence of the missing ability to report. (I make a strict distinction between reporting and documentation: The need for reporting is an actual need for communication while the need for documentation is a potential need for information in the future.)

Dissatisfaction with computer science curricula is not a new phenomenon: Complaints about a lack of practice orientation could already be heard or read in the seventies. Consequently, more emphasis was put on programming professionalism and on subjects called "software engineering". But this did not eliminate the deficiencies I am complaining about.

Not everybody assigns the same meaning to the word "engineering". For me, engineering mainly means planning and coordinating a process of divided work for producing a complex technical system. And the basis of this is the ability to report efficiently on complex systems - in our case software systems - and to produce high quality system descriptions. Certainly, there are many other tasks performed by engineers, but many of them I would not call engineering but applied math or applied physics. Doubtlessly, the deficiencies I am talking about do belong to the field of engineering as I define it.

In an exchange of arguments we had a couple of years ago, Dave Parnas wrote to me: "You can't set up a program for only educating chiefs, you also have to care for the Indians." On the other hand, James Larus from Microsoft Research said at Dagstuhl's 10th Anniversary Symposium: "Educate engineers, not mechanics. Best students from best schools have skills of mechanics - excellent programmers." My experience is that engineers can more or less manage to do the job of mechanics, but that mechanics in most cases fail when they must do the job of engineers. Therefore, our education must be focused on the special subjects which are the basis of engineering.

Every new textbook on software systems provides more evidence of the fact that the inability to produce high quality system descriptions is implicitly accepted as state of the art. When we look into these textbooks, what shall we find?

Excellent descriptions of the pieces, i.e. of algorithms, data structures, protocols, user interfaces, source code structures, inheritance trees, application programming interfaces, etc. But certainly no satisfying engineering level view of the whole system. Compared with mechanical engineering this is as if we only knew how to make engineering drawings of nuts and bolts, but not of trucks or airplanes!

There is certainly an urgent need for a change.

When I studied some software engineering programs, I found that the goals, though well justified, were different from mine. I could not find programs aiming at the results I am looking for. I am convinced that by analyzing traditional engineering programs, we can find many basic concepts which could be used successfully in a new approach of establishing a curriculum in software engineering.

Dijkstra once said (This is no exact quotation!): "Computers and software are so different from anything humans had to deal with in the past, that it is not helpful to transfer categories and concepts from the old world to this completely new world. It is better to forget our old experience and to start from scratch." I am not the only one who does not share this opinion. On the contrary, I see a lot of categories and concepts which can be transferred from the traditional engineering world into the world of software based systems.

When I look at my own engineering education, I see right away three characteristics which should be transferred:

- The program I went through was not designed with the goal of ensuring that we would be productive in a certain kind of industry on the day after our graduation. The program was designed with the goal of ensuring that we could grow into a wide variety of engineering jobs.

- Our "engineering mind-set" was built up in the first four semesters. This means that it was made quite clear to us that our contribution as engineers to building complex systems would always be only a very small fraction of the whole effort. And as a consequence of this, we learned that it is more important to know how to efficiently communicate with all the others involved in the process in order to understand their problems and decisions, than to know every method for optimizing the details. We got the confidence that we could study such methods later on our own in case a specific need would come up.

- Courses and exercises in design and construction had their place in the second half of the program because they were planned for students whose engineering mind-set had already been built up. The majority of the courses - I would say more than 70% - were focussed on modelling and analyzing, not on synthesizing.