Paul Feyerabend und die Softwaretechnologie

Gregor Snelting
Abteilung Softwaretechnologie
Technische Universität Braunschweig

Zusammenfassung. Der folgende Beitrag ist ein Plädoyer für stärkere methodische Standards in der Softwaretechnologie. Nicht immer werden solche wissenschaftstheoretischen Grundsätze beachtet wie das Prinzip, daß Aussagen falsifizierbar sein sollten. Dies scheint die These der sog. Konstruktivisten zu bestätigen, daß nämlich "objektive Wahrheiten" in Wirklichkeit soziale Konstrukte seien. Der Beitrag plädiert deshalb für eine stärkere empirische Fundierung der Softwaretechnologie.

1 Einleitung: Der sog. Konstruktivismus

Im offiziellen Frauenförderungsbericht des Niedersächsischen Wissenschaftministeriums findet sich neben einem Vorwort der Ministerin ein Kapitel über jede Wissenschaft. Im Kapitel über Mathematik, Physik und Informatik heißt es unter dem Stichwort "Erschütterung des Objektivitätspostulates": "Viele sogenannte wissenschaftliche Tatsachen, die bisher als objektiv galten, werden womöglich als hierarchiefixierte, herrschaftsorientierte, lebensfeindliche, weil subjektiv-androzentrische Interpretationen erkannt werden." ([MWK93], S. 117). Mit anderen Worten: Die sogenannten Naturgesetze sind eine Erfindung der Männer, um die Frauen zu unterdrücken. In der Tat geht ja aus Newtons kürzlich aufgefundenem Geheimtestament hervor, daß er das Fallgesetz nur erfunden hat, um seine Lebensabschnittspartnerin am Höhenflug zu hindern.

Aber Scherz beiseite. Die Aussagen der Niedersächsischen Frauenförder-Schrift sind nur die feministische Variante einer erkenntnistheoretischen Haltung, die als Konstruktivismus bezeichnet wird und u.a. von P. Feyerabend mitbegründet wurde [Fe74]. Konstruktivisten glauben nicht an kontextunabhängiges Wissen, schon gar nicht an Naturgesetze und ihre (näherungsweise) Erkennbarkeit durch den Menschen. Vielmehr halten sie die "Naturgesetze", grob gesagt, für das Resultat einer gesellschaftlichen Übereinkunft, für das Ergebnis komplexer Kommunikationsprozesse, kurz: für ein soziales Konstrukt. Daß die "Konstruktion" der "Naturgesetze" interessengesteuert erfolgt, versteht sich von selbst; Professoren, die sich als Wahrheitssucher bezeichnen, verteidigen in Wirklichkeit nur ihre Pfründe.

Feyerabend wollte zeigen, "wie leicht es ist, die Menschen im Namen der Vernunft an der Nase herumzuführen"; daß es philosophische Fundamente zuverlässiger Erkenntnis geben könnte, bestritt er mit dem Slogan "anything goes". Sein ultimatives Verdikt konnte man 1991 im amerikanischen "Who's Who" nachlesen: "Alle Naturwissenschaftler sind Verbrecher". In den USA rief das nur ein müdes Lächeln hervor; in Deutschland hingegen sind konstruktivistische Ansätze in den Geisteswissenschaften weit verbreitet. Auch viele gebildete Laien glauben, daß 1. Einstein bewiesen hat, daß alles relativ ist, und daß 2. Gödel bewiesen hat, daß man nichts beweisen kann.

Nun könnte man einwenden, daß Wissenschaft doch offensichtlich *funktioniert* – Flugzeuge fliegen, Glühlampen brennen, Atombomben explodieren. Wie können da die zugrundeliegenden Theorien nur soziale Konstrukte sein? In einem neueren Interview, das der amerikanische Wissenschaftsjournalist J. Horgan mit Feyerabend führte [Ho96], wird klar, daß Feyerabend in Wirklichkeit die Erkenntniskraft der Naturwissenschaften nicht bestreitet – er fürchtet sich vielmehr vor der "Tyrannei der Objektivität", die für menschliche Kreativität und Vielfalt keinen Platz mehr lasse. Nur aus dieser Angst heraus wird Feyerabends dadaistische Wissenschaftstheorie verständlich.

2 Konstrukt und Realität

Ich möchte in diesem Beitrag die These aufstellen, daß es in der *Softwaretechnologie* gelegentlich durchaus so zugeht, wie von Feyerabend und den niedersächsischen Damen behauptet. Ich werde diese These an einigen Beispielen zu erläutern suchen, um dann methodische Konsequenzen zu formulieren. Dazu müssen wir zunächst einen Blick auf einige erkenntnistheoretische Grundlagen und ihren Bezug zur Informatik tun.

Wenden wir uns zuerst der Frage zu, wie man denn objektives Wissen von Konstrukten unterscheiden kann. Viele glauben, daß das Wesentliche an Wissenschaft in ihrer *Erklärungskraft* liegt: Aufgabe der Forschung sei es, ein verständliches, am besten mechanisches Modell eines Sachverhaltes zu liefern. Tatsächlich liefern erfolgreiche Theorien und Methoden natürlich immer auch Erklärungen der beobachteten Phänomene. Trotzdem können Erklärungen allein niemals Konstrukte von sicherem Wissen trennen. Die Menschheit hat in ihrer Geschichte schon viel "erklärt"; Erklärungen sind wohlfeil und nichts wert. Warum geht die Sonne im Osten auf? Weil sie sich um die Erde dreht. Warum fühle ich mich schlecht? Weil Außerirdische von mir Besitz ergriffen haben. Was hilft dagegen? Bachblütentherapie.

Das entscheidende Kriterium ist ein anderes, nämlich *Vorhersagekraft*. Wenn meine Theorie sagt, daß der Mond in genau einem Jahr am Punkt x stehen wird, und sich diese Voraussage ein Jahr später bewahrheitet, so bedeutet das, daß das theoretische Modell sehr plausibel oder "zuverlässig" ist; zuverlässig genug, um technische Systeme darauf aufzubauen. Auch in anderen Bereichen wie Medizin oder Psychologie ist Vorhersagekraft der

entscheidende Indikator (entsprechende Vorhersagen sind z.B. “Das Medikament wird helfen”, “Der Sexualstraftäter wird nicht rückfällig”). Daß Vorhersagen oft nur statistischer Natur sind, wie in Psychologie und Quantenmechanik, ist kein Manko – wenn ich voraussagen kann, daß sich von 100 Personen 10 soundso verhalten werden, oder von 1000 Elektronen eines tunnelt, habe ich eine zuverlässige Theorie, auch wenn die Vorhersagen nur für Kollektive und nicht für Individuen gelten.

Um Vorhersagen überhaupt überprüfen zu können, müssen diese durch Experimente widerlegbar sein. Dies ist das bekannte *Falsifizierbarkeitskriterium* von Popper. Nicht falsifizierbare Voraussagen sind nach Popper per se unwissenschaftlich. Freilich darf man dies nicht zu dogmatisch sehen. Popper selbst gab in einem Interview kurz vor seinem Tode zu, daß er Falsifizierbarkeit nicht als absolutes Prinzip, sondern als methodische Richtschnur betrachtet [Ho96].

Oft wird eingewandt, daß menschliche Erkenntnis notwendig durch a) die Struktur des Erkenntnisapparates und b) durch den sozialen Kontext (z.B. Prägungen, Vorurteile) begrenzt sei. Kontextabhängige Aussagen können in der Tat dem Kriterium der Vorhersagekraft nicht standhalten; Kontextabhängigkeiten sind gerade typisch für Konstrukte. Wer aber die Möglichkeit kontextunabhängigen Wissens bestreitet, plädiert in Wirklichkeit für Flugzeuge, die vom Weltbild ihrer Erbauer statt von den Gesetzen der Aerodynamik getragen werden. Das hatten wir schon einmal – im Mittelalter.

Der erste Einwand wiegt dagegen schwerer und wurde schon von Kant vorgebracht. In der Tat wird, wie man heute weiß, menschliche Wahrnehmung vom Gehirn aktiv konstruiert. Daraus darf man aber nicht folgern, daß zwischen Realität und “Innenwelt” nur eine schwache Kopplung besteht. Die evolutionäre Erkentnistheorie [Vo92] weist zu Recht darauf hin, daß das menschliche Gehirn sich in Anpassung an die Realität entwickelt hat; würde unser Erkenntnisapparat zu viele falsche Vorhersagen machen, hätte die Menschheit gar nicht überlebt. Und so ist es sicher richtig, daß ein Mensch einen Raum ganz anders erlebt als, sagen wir, eine Fledermaus. Genauso richtig ist aber auch, daß Mensch und Fledermaus lediglich verschiedene mentale Repräsentationen derselben realen Raumtopologie bilden.

3 Informatik und Erkenntnistheorie

Abstraktion als Standardmethode

Wie ist es nun in der Informatik? Informatik ist keine Naturwissenschaft. Vielmehr schafft sich die Informatik ihre Realität zum Teil selbst. “Die durch Abstraktion entstandenen Konstrukte der Informatik als Bedingungen möglicher Information sind zugleich die Bedingungen der möglichen Gegenstände der Information in den Anwendungen” sagte einst H. Wedekind in Anlehnung an eine Aussage Kants aus der “Kritik der reinen Vernunft”. Einfacher gesagt: Informatiker erfinden (konstruieren) abstrakte Konzepte; diese ermöglichen (oder begrenzen) wiederum die spezifischen Anwendungen.

In der Tat ist das *Finden guter Abstraktionen* eine der herausragenden Kompetenzen von Informatikern; dies gilt besonders für die Softwaretechnologie. Jeder Algorithmus ist bereits eine Abstraktion (von konkreten Eingabewerten). Als Softwaretechnologe denke man etwa an abstrakte Datentypen, Softwarearchitekturen, Objektorientierung, generische OO-Klassen, parametrisierbare Softwarekomponenten, Entwurfsmuster.

Selbstverständlich müssen die entwickelten Abstraktionen nebst darauf basierenden Theorien *konsistent* sein, denn aus inkonsistenten Theorien kann man stets alles folgern (also auch alle Voraussagen über das Verhalten von Software). Beispielhaft ausgeleuchtet wurde der Konsistenzbegriff in der Theorie der abstrakten Datentypen: Konsistenz kann durch die initiale Semantik algebraischer Spezifikationen genau charakterisiert werden.

Übrigens ist es wünschenswert, daß Abstraktionen auch *elegant* sind. Die Erfahrung aus den Naturwissenschaften zeigt, daß elegante Theorien eine höhere Korrektheitswahrscheinlichkeit haben. Die Definition “Ein Betriebssystem ist ein 17–Tupel” ist nicht elegant, ebensowenig wie viele Konzepte von Fortran und Cobol. Leider zeigt die Entwicklung der Programmiersprachen, daß sich manchmal auch häßliche Ideen durchsetzen.

Voraussage und Experiment

Was ist nun die Informatik-Version von Voraussagen nebst falsifizierendem Experiment? Voraussagen werden in der Informatik (wie in anderen Fächern) aus Abstraktionen und Theorien gewonnen. Jeder Informatiker kennt Voraussagen über das Verhalten und die Komplexität von Algorithmen, die mit analytischen Methoden (Programmverifikation, Komplexitätsanalyse) gewonnen werden. Diese werden aber nicht immer experimentell überprüft, und manche Informatiker halten das auch für überflüssig. So gibt Dijkstra bekanntlich nur widerstrebend zu, daß Programme überhaupt ausgeführt werden; die Aufgabe des Informatikers besteht gemäß Dijkstra – neben der Konstruktion des Programms – vielmehr im Durchführen des Korrektheitsbeweises [Di89].

In der Praxis läuft ein Programm aber in einer realen technischen Umgebung, und deshalb ist eine besondere Form des Experiments – der *Test* – unverzichtbar. Die formale Korrektheit erfaßt nämlich nur einen Teil des realen Verhaltens eines Algorithmus, ebenso wie die Newtonsche Mechanik das reale Himmelsgeschehen nur näherungsweise beschreibt. Auch Komplexitätsaussagen (Beispiel: $O(n^{2.81})$ ist besser als $O(n^3)$) bewahrheiten sich in der Praxis nicht, wenn der “break-even-point” jenseits praktikabler Problemgrößen liegt.

Wie alle Experimente, kann man aber das Testen nur zum Widerlegen einer Vorhersage verwenden: Dijkstras Diktum “Testen kann nur Fehler nachweisen, nicht Korrektheit garantieren” ist ein Spezialfall von Poppers Falsifikationsprinzip. Tests – und nichts anderes – können Fehler in Modellbildung und Verifikation aufdecken (Hingegen kann die gängige Methode, mit Tests nach Implementierungsfehlern zu suchen, zumindest im Prinzip durch Korrektheitsbeweise ersetzt werden).

Andere Voraussagen in der Informatik betreffen nicht konkrete Algorithmen, sondern die Entwicklungsmethodik selbst. Manche bekannte Voraussagen wurden zunächst heftig angegriffen und haben sich erst Jahrzehnte später als "nicht falsifizierbar" etabliert. Als softwaretechnologische Beispiele seien genannt:

1. "GOTOs erhöhen die Codeentropie" (Dijkstra)
2. "Strenge Typisierung reduziert die Laufzeitfehler" (Wirth)
3. "Gute Modularisierung reduziert die Wartungskosten" (Parnas)

Da die Vorteile der GOTO-freien, streng typisierten, modularen Softwareentwicklung sich nicht unmittelbar, sondern oft erst viel später erweisen, im Gegenteil zunächst höhere Kosten entstehen, kann der "Test" derartiger Voraussagen erhebliche Zeit in Anspruch nehmen.

Es kommt also nicht nur darauf an, daß ein Algorithmus oder eine Methodik theoretisch korrekt und "unkomplex" ist, sondern auch das Verhalten im realen Entwicklungs- und Einsatzumfeld zählt. "Zu Effektivität und Effizienz tritt als dritte Bestimmungsgröße die Performanz auf. Sie ist in der Informatik die entscheidende Größe, und sie ist letztlich nur im utilitaristischen, d.h. ökonomischen Kontext testbar" [Ko97].

Axiomatik vs. Empirie

Neue Abstraktionen werden in der Informatik oft axiomatisch formuliert, müssen dann aber ihre Nützlichkeit empirisch nachweisen. Ein herausragendes Beispiel sei hier einmal angeführt, nämlich Polymorphismus in funktionalen Sprachen. Entstanden aus zwei sich offenbar widersprechenden praktischen Anforderungen, nämlich hohe Typsicherheit einerseits und hohe Flexibilität und Wiederverwendbarkeit von Code andererseits, garantiert der Damas/Milner Typkalkül Laufzeitfehlerfreiheit auch für Code, der in verschiedenen Kontexten einsetzbar ist. Die Axiome sind gerade so konstruiert, daß die Typisierbarkeit entscheidbar bleibt (in der Praxis ist sie sogar effizient, obwohl sie theoretisch NP-hart ist), aber trotzdem Funktionen mit flexiblem Argumenttyp möglich sind. Polymorphismus ist heute ein wichtiges Prinzip, und Milner erhielt für sein Typsystem (und andere Leistungen) den Turing Award.

Demgegenüber verwischt das rein axiomatische Vorgehen in der Mathematik, so wie von Bourbaki exemplarisch durchgeführt, daß Mathematik auch einen Realitätsbezug hat und nicht nur ein superschlaues Glasperlenspiel ist. Neoplatoniker wie Penrose sehen mathematische Gebilde wie die Schrödingergleichung sogar für das "Ding an sich" an – die unglaubliche Kraft der mathematischen Modellbildung kann nach Penrose nicht nur das Resultat formalistischer Sandkastenspiele sein.

4 Einige Beobachtungen

Empirische Studien werden kaum durchgeführt

1994 untersuchte Walter Tichy 400 Informatik-Publikationen aus referierten Journalen und Konferenzen des Jahres 1993 [Ti95]. Dabei stellte er folgendes fest:

- 43% aller untersuchten Informatik-Arbeiten enthielten keine experimentellen Untersuchungen;
- Nur 31% aller Arbeiten widmeten mehr als 20% ihres Umfangs Fallstudien oder experimentellen Validierungen;
- In den IEEE Transactions on Software Engineering waren 55% der Artikel der Präsentation neuer Konzepte ohne jede Validierung gewidmet;
- in den ACM Transactions on Programming Languages and Systems immerhin nur 45%;
- Nur 20% der Software-Engineering(SE)-Artikel enthielten mehr als 20% experimentelle Untersuchungen;
- Vergleichsstudien aus den Bereichen Optical Engineering und Neural Computation zeigten, daß dort 70% der Arbeiten einen substantiellen experimentellen Anteil enthielten.

Tichy zieht daraus alarmierende Schlüsse: "The results suggest that large parts of computer science may not meet standards long established in the natural and engineering sciences". Da die Informatik inzwischen ein halbes Jahrhundert alt ist, kann jugendliche Unreife nicht der Grund sein. Tichy befürchtet, daß sich die Informatik als Wissenschaft mittelfristig diskreditieren wird, was erhebliche Auswirkungen auf Fördermittel, öffentlichen Einfluß usw. haben dürfte. "When comparing this situation with the way the medical profession adopts new treatment or drugs, I can't help seeing the software field as dangerously close to quackery", ergänzte er kürzlich [WoW97].

Nun kann man gewiß nicht von jedem Artikel umfangreiche Fallstudien oder gar flächendeckende Experimente wie in der Arzneimittelforschung verlangen. Auch Parnas, Wirth und Dijkstra gaben keine experimentelle Überprüfung für ihre Voraussagen an. Gewöhnlich Sterbliche, die in einem deutschen Informatik-Institut sitzen, haben meist weder Zeit noch Geld, ihre Arbeiten experimentell zu validieren. Manche Falsifikationsversuche sind übrigens wirklich überflüssig: Der Nutzen etwa von graphischen Benutzerschnittstellen gegenüber ASCII-Kommandozeilen springt meist unmittelbar ins Auge. Dennoch besteht ganz klar ein empirisches Defizit im Software Engineering.

Die Kluft zwischen Theorie und Praxis wächst

Im kürzlich erschienen Beitrag "Strategic directions in software quality" heißt es: "The chasm between research and practice seems particularly wide and increasingly inculturated, to the detriment of both communities. Practice is not as effective as it must be, and research suffers from the validation of good ideas and redirection that inevitably results from serious use" [Os96].

Von der Industrie wird den akademischen Softwaretechnologen regelmäßig vorgeworfen, sie seien viel zu abgehoben und würden reale Probleme ignorieren. Oft werden Professoren rundheraus für Traumtänzer gehalten – während selbige anmerken, daß Manager ein initiales Modell nicht von einem Pappmodell unterscheiden kön-

nen. Für dieses Phänomen werden verschiedene Gründe angeführt. David Notkin glaubt, daß akademische SE-Forscher sich nicht für Anwendungen interessieren oder Praktiker verachten, während gleichzeitig die Industrie verlangt, die Forschung möge sich ihren spezifischen Tagesproblemen widmen [WoW97]. Tichy verweist auf die fehlende empirische Fundierung, die es Praktikern verunmöglicht, den Wert neuer Ergebnisse einzuschätzen.

Festzustellen bleibt, daß viele theoretisch orientierte Softwaretechnologen sich für Anwendungen nur mäßig interessieren. Es ist jedoch nicht ausgeschlossen, daß theoretische Resultate etwa zu formalen Spezifikationen auch praktisch nützlich sind, und der Autor möchte davor warnen, die Grundlagenforschung zu demontieren, nur weil sich Kategorientheorie nicht direkt in Dollar transformieren läßt. Die Zahlentheorie galt lange Zeit als esoterisch, bis sie sich plötzlich als Grundlage von Verschlüsselungsverfahren entpuppte. In letzter Zeit bemüht sich übrigens die theoretische SE-Gemeinde verstärkt um Fallstudien und industrielle Anwendungen [ABL96].

Aber auch in der Industrie gibt es massive Barrieren. Was soll man davon halten, wenn sehr große und einflußreiche Softwarehersteller gesicherte Erkenntnisse der Softwaretechnologie einfach ignorieren? Objektorientierung ist eine bemerkenswerte Ausnahme, die aber eher auf Marketing als auf echte Einsicht zurückgeht. Moderne Softwaretechnologie wird eigentlich nur im Hochsicherheitsbereich (Raumfahrt) und in großen, forschungsorientierten Firmen angewandt, und auch da nicht immer. Im kommerziellen Bereich wurde die gesamte Informatik eigentlich konsequent als irrelevant abgetan; stattdessen setzte man auf Cobol und jene EDV-ler, die in Kurzkursen elementare Programmierung erlernten. Erst in letzter Zeit merken auch die kommerziellen Mainfraimer, daß sie ein Problem haben, und sind geneigt, z.B. objektorientierte Methoden in Betracht zu ziehen.

Konstruktivismus in der Softwaretechnologie

Theorieorientierte Forscher werden zu Unrecht für den fehlenden Praxisbezug der Softwaretechnologie verantwortlich gemacht. Theoretische Modelle (z.B. formale Sprachen) haben gewaltig zum Fortschritt der Informatik beigetragen. Weder formale Spezifikationen noch funktionale Programmiersprachen, Model Checking oder andere "Esoterika" sind an der gegenwärtigen Situation der Softwaretechnologie schuld. Theoretische Modelle lassen sich an der Realität messen; manch theoretisches Verfahren bewährt sich in der Praxis.

Nein, das Problem kommt gerade von jenen *praxisorientierten* Wissenschaftlern, die Theorie ignorieren, sich aber gleichzeitig der empirischen Validierung entziehen. Solche Forscher praktizieren Konstruktivismus, und davon gibt es in der Softwaretechnologie eine ganze Menge. Ein Beispiel mag das erläutern und zugleich dem Bekenntnis eigener Jugendsünden dienen.

Vor 10 Jahren waren sprachspezifische Programmierumgebungen der große Renner in der Forschung. Weltweit wurden an die hundert Systeme entwickelt, die versprachen, die Softwareentwicklung dadurch zu vereinfachen, daß die Entwicklungswerkzeuge Syntax und Semantik der verwendeten Sprache kennen. Die meisten dieser Umgebungen zentrierten sich um einen Strukturediteor. Tatsache ist, daß Strukturediteoren allenfalls für Anfänger oder für Spezialsprachen nützlich sind; dem normalen Programmierer haben sie nichts zu bieten.

Der Grund für den seinerzeitigen Aktionismus war, daß sprachspezifische Werkzeuge plötzlich technisch möglich wurden, indem Compilertheorie inkrementalisiert und auf interaktiven Workstations realisiert wurde. Viel Geld ist in die Entwicklung von Strukturediteoren geflossen – ausgegeben für ein Konstrukt, nämlich eine Problemlösung für ein nichtexistentes Problem. Dabei hatten einige der entwickelten Systeme hohe theoretische Ansprüche, und allein die dafür entwickelte Technologie, z.B. inkrementelle oder generische Analyse-, Inferenz- und Interpretationsverfahren, rechtfertigen im Nachhinein den Mitteleinsatz.

Heute gibt es andere Modethemen, z.B. Frameworks, Reengineering oder Software Process Modelling. Reengineering etwa ist sehr wichtig, gerade auch im Hinblick auf das Jahr 2000. Viele Reverse Engineering/Reengineering-Verfahren setzen jedoch Heuristiken ein, um Strukturinformation aus alter Software zu rekonstruieren. Solche Heuristiken enthalten stets freie Parameter, die an echter Altsoftware geeicht werden müssen. Dies ist nur über empirische Studien möglich. Das methodische Minimum auch für forschungsorientierte Artikel, die neue Verfahren vorstellen, sind deshalb eine oder mehrere Fallstudien. Reiner Implementierungsaktionismus läßt methodische Zweifel aufkommen.

Immerhin: falls neue Konzepte nicht gut formalisierbar sind, ist das mindeste, was man verlangen kann, daß eine prototypische Implementierung existiert, die man zur Evaluierung verwenden kann. In der Softwaretechnologie kommt es aber vor, daß Ideen veröffentlicht werden, die weder mathematisch faßbar noch implementiert sind. Nach meiner Erfahrung beträgt der Anteil solcher "freischwebenden" Beiträge bei softwaretechnologischen Konferenzen bis zu 20%.

Entsprechend setzt sich in der Softwaretechnologie oft nicht die beste Idee durch, sondern die, die genügend Marktmacht und/oder Einfluß auf Geldgeber hinter sich hat. Dieses Phänomen ist in der Informatik stärker ausgeprägt als in anderen Fächern, und das liegt zum Teil daran, daß sich die Informatik – wie oben dargelegt – ihre Realität teils selbst schafft, so daß es meistens mehr als eine technische Lösung gibt.

Zum Teil liegt es aber auch an methodischen Mängeln. Hatte Popper noch angemahnt, man möge jeden Morgen vor dem Spiegel versuchen, die eigene Lieblingstheorie zu falsifizieren, so bemühen sich manche Softwaretechnologen (und –innen), das eigene Lieblingsspielzeug um jeden Preis zu "verkaufen", aber eine Validierung zu verhindern. Dies führt genau dazu, daß die Glaubwürdigkeit verlorengeht und konstruktivistische Kritik laut wird.

5 Einige Konsequenzen

- Es gibt eine Proliferation von Konferenzen und Workshops, wobei gleichzeitig die Qualität vieler Veranstaltungen immer dünner wird. Hier sollte gegengesteuert werden.
- Oft wird dieselbe Idee mehrmals in leicht variierter Form publiziert (sog. "cut-copy-paste-Artikel"); dies schadet auf die Dauer dem Ansehen der Wissenschaft.
- Beiträge, die lediglich eine Idee präsentieren, ohne ein theoretisches Modell oder eine Implementierung anzubieten, sind methodisch fragwürdig.
- Werkzeuge und Verfahren sollten prototypisch implementiert werden, um eine Beurteilung zu ermöglichen.
- Prototypen sind keine Produkte, und Universitätsinstitute keine Softwarefirmen. Deshalb kann man professionelle Qualität von Forschungsprototypen kaum verlangen.
- Forschungsergebnisse müssen reproduzierbar sein; mithin sollten auch Prototypen – trotz ihrer Unvollkommenheit – öffentlich zugänglich sein.
- Wenn SE-Erkenntnisse bei der Werkzeugentwicklung angewendet würden, wäre die Qualität der Werkzeuge besser. Selbstanwendung ist ein Standardverfahren der Informatik!
- Beiträge, die neue Konzepte vorstellen und theoretisch untersuchen, sind absolut legitim, wenn das theoretische Modell überprüfbare Vorhersagen hergibt.
- Theorie ist sogar dann sinnvoll, wenn bestimmte Phänomene einfacher beschrieben oder besser erklärt werden können, ohne daß sich neue Vorhersagen (sprich: mögliche praktische Konsequenzen) ergeben.
- Theorie ist in der Softwaretechnologie hingegen nicht sinnvoll, wenn sie zum Selbstläufer ohne Anwendungsbezug wird.
- Was fehlt, sind Untersuchungen, die den tatsächlichen Wert neuer Konzepte, Theorien, Methoden und Werkzeuge evaluieren.
- Zwar sind umfangreiche empirische Studien nur selten möglich, aber eine Fallstudie sollte immer Teil der Arbeit sein.
- Zusammenarbeit mit der Industrie ist oft sinnvoll, um neue Ideen in einem echten Anwenderkontext auszuprobieren.
- In der Lehre sollten die Studenten in umfangreichen Praktika Softwaretechnologie tatsächlich einüben.
- Informatik-Absolventen, die nach Hackerart entwickeln oder gar nicht anständig programmieren können, heben unser Ansehen nicht.
- Wenn die empirischen Fundamente besser werden, wird auch die Industrie stärker auf die Ergebnisse der akademischen SE-Forschung hören.
- Die Softwaretechnologie wird im Vergleich zu KI, Telematik usw. wieder mehr Fördermittel erhalten, wenn sie ihre Methodik überprüft hat.

6 Schluß

Die Naturwissenschaften haben von jeher hohe methodische Standards gesetzt, und nach 2000 Jahren Erkenntnistheorie haben wir gute Kriterien, um zuverlässige Aussagen von Konstrukten zu unterscheiden. Es wäre schön, wenn softwaretechnologische Ergebnisse sich ähnlicher Solidität erfreuen könnten.

Denn dann bliebe dem Konstruktivismus nur noch ein Betätigungsfeld: die Selbstanwendung (ein Standardverfahren der Informatik). Wir überlassen es dem Leser als Übungsaufgabe, das Ergebnis solcher Selbstanwendung zu formulieren.

7 Literatur

[ABL96] Abrial, J. , Börger, E., Langmaack, H. (Eds.): Formal methods for industrial applications. Lecture Notes in Computer Science, VOl. 1165. Springer 1996.

[Di89] Dijkstra, E.D.: On the cruelty of really teaching computer science. Communications of the ACM 32 (12), S. 1398 – 1404 (Dezember 1989).

[Fe74] Feyerabend, P.: Against method: outline of an anarchistic theory of knowledge. Atlantic Highlands 1974.

[Ho96] Horgan, J.: The end of science. Addison Wesley 1996.

[Ko97] Kornwachs, K.: Um wirklich Informatiker zu sein, genügt es nicht, Informatiker zu sein. Informatik-Spektrum 20, 2 (April 1997), S. 79 –86.

[MWK93] Frauenförderung ist Hochschulreform – Frauenförderung ist Wissenschaftskritik. Herausgegeben vom Niedersächsischen Ministerium für Wissenschaft und Kunst, 1993.

[Os96] Osterweil, L. etc al.: Strategic directions in software quality. In: Strategic directions in computing research. ACM Computing Surveys 28(4), Dezember 1996.

[Ti95] Tichy, W. et al.: Experimental evaluation in computer science: a quantitative case study. Journal of Systems and Software 28, 1 (Januar 1995), S. 9 –18.

[Vo92] Vollmer, G.: Evolutionäre Erkenntnistheorie. Hirzel Wissenschaftliche Verlagsgesellschaft, Stuttgart 1992.

[WoW97] Tracz, W. et al.: ICSE-19 Window on the World. Volume 3 No. 1–3. http://www.webjammers.com/wow