Feyerabend – zwei Jahre später

Gregor Snelting
Universität Passau

1 Feyerabend

Als vor zwei Jahren mein Artikel „Paul Feyerabend und die Softwaretechnogie“ im Informatik-Spektrum erschien ([6], englische Fassung in [7]), bekam ich viele Leserbriefe. Ich hatte überwiegend Kritik erwartet, insbesondere Kritik von Vertretern intellektueller Moden des letzten Jahrhunderts wie dem sog. Konstruktivismus und dem sog. Feminismus. Für diesen Fall hatte ich auch schon etwas in der Schublade. Überraschenderweise äußerten die diversen Emails und persönlichen Kommentare fast nur Zustimmung. Dies stimmte mich nachdenklich, denn einige der „Fans“ vertreten eigentlich die softwaretechnischen Ansätze, die in dem Artikel hinterfragt wurden.

Inzwischen sind zwei Jahre vergangen, und die Informatik-Welt hat sich geändert. Die Y2k-Umstellung haben „wir Softwaretechniker“ problemlos bewältigt; die Politik hat die Bedeutung der Informatik erkannt; die Anfängerzahlen in Informatik steigen exponentiell; Bindestrich-Studiengänge und Bachelor-Abschlüsse sprießen aus dem Boden; alle reden über das Internet; und es wird sogar mehr Geld für Forschung und Lehre in Aussicht gestellt. In der Softwaretechnik ist der Siegeszug der Objektorientierung unaufhaltsam, in der Grundlagenforschung gibt es neue Impulse, und viele Standorte haben die softwaretechnische Ausbildung intensiviert. Hat sich die Welt also verbessert? Paul Feyerabend würde der Softwaretechnik auch heute wieder sein „Anything goes“ entgegenwerfen. Dies jedenfalls vermute ich und möchte es im Folgenden näher ausleuchten.

2 Methodik

Feyerabend hatte behauptet, dass es „objektives Wissen“ nicht geben kann, sondern dass wissenschaftliche Theorien und Methoden lediglich „soziale Konstrukte“ seien, erfunden zur Stabilisierung der herrschenden Machtverhältnisse [2]. Wir wissen heute – auch von Feyerabend selber [1] –, dass diese erkenntnistheoretische Haltung als Waffe im politisch-psychologischen Kampf der Alt-68er konstruiert wurde, so dass der Konstruktivismus ein wunderbares Beispiel von Selbstanwendbarkeit lieferte. „Wahrheit ist die Erfindung eines Lügners“ – sagte der Lügner.

Trotzdem ist der Begriff des „Konstrukts“ nützlich, der auf kontextabhängige Pseudoerklärungen verweist und im Gegensatz zu zuverlässigem Wissen steht. Zu den Kriterien, die man gemeinhin als Indikatoren für zuverlässiges Wissen ansieht, gehört insbesondere die *Vorhersagekraft* wissenschaftlicher Aussagen. Parnas, Wirth, Dijkstra und andere machten softwaretechnische Vorhersagen, die auch Jahrzehnte später nicht widerlegt, sondern immer weiter untermauert wurden. Diese Eigenschaft geht manchen anderen softwaretechnischen Arbeiten ab. Behauptungen und Konstrukte, die weder empirisch noch theoretisch abgesichert sind, stattdessen Einflüssen der Wissenschaftsmode entspringen, sind in der softwaretechnischen Forschung nicht unbekannt.

Tichy und andere hatten deshalb eine stärkere empirische Fundierung der Softwaretechnik verlangt. Insbesondere methodische Aussagen sollten experimentell so abgesichert werden, wie es z.B. in der Arzneimittelforschung üblich ist. Dies wurde manchmal dahin misinterpretiert, dass kreative Ideen nun unerwünscht und nur noch überdimensionale Feldstudien methodisch korrekt seien. Heute aber muss man feststellen, dass die seinerzeitige Initiative die Situation verbessert hat. Empirische Validierungen neuer Ergebnisse sind quasi zur Pflicht geworden für methodenorientierte Beiträge, die sich schlecht analytisch behandeln lassen. Sogar in Bereichen, die Korrektheits- und Komplexitätsbeweisen zugänglich sind, werden Ergebnisse nur noch in Kombination mit Benchmarks anerkannt. Reine $O(.)$-Aussagen etwa abstrahieren zuviel weg, Benchmarks allein hingegen sind aus bekannten Gründen fragwürdig.

Dieses Umdenken hatte zur Folge, dass Konstrukte ohne Realitätsbezug in softwaretechnischen Beiträgen selten geworden sind. Beiträge, die nur noch ein Konzept mit „Kästchen und Pfeilen“ vorstellen, ohne eine analytische oder empirische Untermauerung mitzuliefern, werden auf großen Konferenzen kaum mehr angenommen. Gleiches gilt übrigens für Beiträge, die nur formal-mathematische Fingerübungen sind, ohne sich der Realität der Softwaretechnik zu stellen. Statistik genießt heute Wertschätzung bei empirischen Softwaretechnikern und in der stochastischen Qualitätssicherung.

Was könnte Feyerabend wohl dagegen einwenden? Er würde vielleicht Churchill zitieren, der bekanntlich gesagt hat, dass man nur den Statistiken glauben soll, die man selbst gefälscht hat. In der Tat lassen sich statistische Methoden auf gegebene Daten überanpassen, manchmal so, dass sie die Vorhersagen liefern, die man sich wünscht. Das Kriterium der Vorhersagekraft ist davon jedoch nicht berührt: wenn *a posteriori* stochastische Vorhersagen nicht widerlegt werden können, sondern dauerhaft bestätigt werden, hat man eine objektive Aussage.

Sind also Fortschritte bei der empirischen Unterfütterung von Softwaretechnik zu bemerken, so kann man auch Fortschritte in softwaretechnisch relevanten Theoriebereichen feststellen. Neue Validierungsverfahren wie Model Checking haben – kombiniert mit Programmanalyse – die Grundlagen für eine weitere Qualitätsverbesserung gelegt und werden deshalb ja auch in den USA massiv gefördert (worüber noch zu reden sein wird).

Im kommerziellen Bereich werden die neuen Verfahren allerdings kaum eingesetzt, sei es, weil sie zu teuer sind, sei es, weil sie für kommerzielle Riesensysteme (noch) nicht anwendbar sind, sei es, weil man sie nicht versteht. Eine positive Ausnahme sind Firmen wie Microsoft, AT&T, IBM, die massiv in neue Softwaretechnologie investieren.

Diese Beispiele zeigen auch, dass die Entfremdung zwischen Theorie und Praxis, die sich typischerweise in gegenseitigen Beschimpfungen äußerte, ihren Höhepunkt überschritten hat. Der größte Erfolg der neueren Softwaretechnik, die objektorientierte Entwicklung, wird in der Industrie inzwischen überall akzeptiert (sogar bei Banken und Versicherungen). Auch die softwaretechnischen Ergebnisse etwa zum Architekturentwurf, zur Komponententechnologie oder zur Prozessmodellierung werden in vielen Softwarefirmen rezipiert. Verstärkt hat sich die Einsicht durchgesetzt, dass wir eine neue Synthese von Theorie und Praxis brauchen – es gibt heute keine „theoretische“ oder „praktische“ Informatik mehr, sondern nur noch gute oder schlechte.

Wir sind also ein Stück vorwärtsgekommen. Aber es gibt immer noch zu viele Softwaretechniker, die nur Kästchen und Pfeile malen. Immer noch gibt es Firmen, die im Stil der 70er Software entwickeln. Immer noch gibt es „Wissenschaftler“, die Softwareentwicklung primär als sozialen Prozess auffassen. Immer noch ist die Qualität von softwaretechnischen Konferenzen nicht mit der von programmiersprachlichen Konferenzen vergleichbar. Immer noch hinkt Deutschland den USA hinterher, was die neue Synthese von Theorie und Praxis anbelangt.

3 Ausbildung

Die Anfängerzahlen in Informatik explodieren, aber von diesen Anfängern wird höchstens die Hälfte überleben. Viele studieren heute Informatik, weil sie gelesen haben, dass man als Informatiker schnell reich werden kann. Wenn sie dann das erste ϵ in der Analysis-Vorlesung sehen, fallen sie ohnmächtig vom Sessel.

Eine Abhilfe kann der Bachelor sein. Ich war früher ganz der Meinung von Jochen Ludewig: „Der Bachelor ist Selbstmord aus Angst vor dem Tode“ [3]. Aber die Gesellschaft braucht heute viele praktische Informatiker (und gar nicht so viele Informatik-Wissenschaftler). Diese zu produzieren, ist eigentlich Aufgabe der Fachhochschulen; angesichts der Realität kann der Bachelor an Universitäten aber ein sinnvoller Ausstieg sein. Die Universitäten dürfen auf keinen Fall zu Kastraten werden, indem man die Theorie aus dem Studium herausnimmt; für den wissenschaftlichen Nachwuchs brauchen wir eigentlich noch mehr Theorie. Für Studenten ohne mathematisches Verständnis kann man aber die theoretischen Anforderungen reduzieren. Ein wohlstrukturiertes 120-SWS-Bachelor-Paket kann so ohne Mehraufwand implementiert werden.

Voraussetzung ist allerdings, dass schon im Grundstudium umfangreiche softwaretechnische Veranstaltungen und Praktika, auch im Team, stattfinden. Dies ist in manchen Informatik-Studiengängen nicht der Fall und auch nicht überall durchsetzbar; Ludewig hat ja deshalb seinen Softwaretechnik-Studiengang gegründet. Aber viele Standorte haben in den letzten Jahren ihr Grundstudium umstrukturiert, die Mathematik reduziert und speziell für Informatiker ausgerichtet und umfangreiche Softwareprojekte in das Grundstudium integriert. Eigenständige Softwaretechnik-Studiengänge halte ich deshalb nicht für essentiell.

Trotzdem haben auch eigenständige Studiengänge ihre Existenzberechtigung, so etwa die Ausbildung zum Software-Systemingenieur am Hasso-Plattner-Institut. Dort ist das angepeilte „Endprodukt" nicht der Wisssenschaftler und auch nicht der Laboringenieur, sondern der Projektingenieur; dieser ist „nicht für jeden kleinen Sortieralgorithmus" (Plattner), sondern für die Architektur und den Entwicklungsprozess im Großen zuständig [9].

Ich denke, solche Ausbildungsmodelle werden von Kerninformatikern zu oft als Konkurrenz empfunden; dabei besteht ein klarer Bedarf an solchen Absolventen, und auch Hasso Plattner wird nicht die Kerninformatik abschaffen. Ich sehe im Gegenteil in einer Profilbildung wie am HPI die Möglichkeit, dass einige Universitäten sich stärker auf wissenschaftliche Informatik konzentrieren könnten, weil sie nicht mehr Studenten betreuen müssten, die in Wirklichkeit Software-Projektingenieur und nicht Wissenschaftler werden wollen (und umgekehrt).

Nicht erkennnen kann ich dagegen einen Bedarf an „getypten" Softwaretechnikern. Softwaretechnik ist ein universelles Querschnittsfach, das nach meiner Überzeugung alle Informatiker kennenlernen müssen. Noch weniger vermag ich mich mit Softwaretechnik für Bindestrichinformatiker anzufreunden. „Softwaretechnik für Sportinformatiker" oder „UML für Apotheker" – so etwas ist nicht nur nutzlos, sondern schädlich.

4 Forschung

Insbesondere in den USA ist eine neue Synthese zwischen Theorie und Praxis zu beobachten. Aufregende theoretische Entwicklungen machen ihren Weg in die Praxis. Der Renner im Moment ist Model Checking, ein Verfahren zur automatischen Verifikation gewisser Systemaspekte; in den letzten Jahren gab es große Fortschritte bei der Anwendung auf Software. Auch die klassischen formalen Spezifikationen lassen sich durch die Fortschritte im automatischen Beweisen heute auf reale Systeme anwenden. Ein anderes Beispiel ist Programmanalyse: heute gibt es viele Analysen, die z.B. Programmverstehen oder Sicherheitsprüfungen unterstützen.

Richtig mächtig werden die neuen Verfahren, wenn man sie kombiniert. Mit Model Checking und Programmanalyse hat die NASA in der Software der Deep-Space-Sonde Fehler gefunden, die der vorangegangenen, intensiven „klassischen" Qualitätssicherung entgangen waren. (Bei Raumsonden gab es auch beeindruckende Pannen – wie die fehlende Umrechnung von Zentimeter in Inch bei der letzten Marssonde – aber dort kamen aus Kostengründen die verfügbaren Technologien nicht zum Einsatz).

Diese Entwicklung kommt nicht ohne Grund. Noch vor 10 Jahren waren die großen Forschungszentren der USA voll auf Anwendungsorientierung gedrillt: nur was sofort Geld brachte, wurde gemacht. Inzwischen wurde umgedacht: Microsoft hat ein exzellentes Forschungszentrum gegründet, und es gibt umfangreiche Förderprogramme für die softwaretechnologische Grundlagenforschung wie die PITAC-Initiative. Schaut man sich die Liste der insgesamt ca. 100 PITAC-Projekte an [8], so sieht man, dass die meisten Projekte sich mit Programmanalyse, Model Checking, Formalen Methoden und ähnlichen Dingen befassen. Auffällig ist die Abwesenheit von Prozessmodellierung, Architekturentwurf und ähnlichen softwaretechnischen Lieblingsthemen, auffällig ist auch die starke Verwobenheit mit programmiersprachlichen Themen und allgemein eine strategische Bündelung der einzelnen Projekte.

Diese Initiative hat kein Gegenstück in Deutschland. Dringend notwendig wäre es aber, etwa in Form eines besonderen, strategisch fokussierten DFG-Programms Grundlagenforschung auch in der Softwaretechnik zu fördern. Die Autoren des sog. Walberberg-Memorandums weisen darauf hin, dass ohne eine solche Förderung die absolute Vormachtstellung der USA in moderner Softwaretechnik dauerhaft erhalten bleibt [5]. Die seinerzeitige Initiative des BMBF zur Förderung der Softwaretechnik ist nicht vergleichbar, da dort nur anwendungsnahe Verbundprojekte gefördert wurden. In der Tat ist der BMBF noch voll in der Theorie-Praxis-Dichotomie verfangen, wie man auch an der Diskussion über die Fraunhofer/GMD-Fusion sieht. Die Position der GI ist da nicht viel besser, da sie nur plump die Grundlagenforschung gegen die angewandte Forschung ausspielt. Von neuer Synthese ist also hierzulande nicht viel zu sehen.

Aber auch in den USA gibt es rückwärtsgewandte Initiativen. Unter dem Motto „We won't take it any more“ stellten Lee Osterweil und andere das Impact-Project vor, dass zum Ziel hat, die Bedeutung softwaretechnischer Forschung für die Praxis nachzuweisen [4]. Nun, ein bisschen Public Relations für die Softwaretechnik schadet sicher nicht. Es ist aber immer befremdlich, wenn eine Forschungscommunity sich selbst evaluieren will, und dies „nach höchsten akademischen Standards“. Erste Ergebnisse wurden übrigens auf der FSE in San Diego vorgestellt, und zwar wurden uns als besonders einflussreich und erfolgreich präsentiert: Inspektionen und Konfigurationsmanagement. Beide sind nachweislich wirksame Techniken.

Aber das Impact-Projekt will ja gerade das Ansehen der Softwaretechnik bei den Nachbardisziplinen erhöhen. Stellen wir uns also den Compilerbauer, Theoretiker oder Physiker vor, der fragt, was das bedeutendste Ergebnis der Softwaretechnik ist. Inspektionen? Was macht man da, löst man da stochastische Differentialgleichungen zur Bestimmung der Fehlerwahrscheinlichkeit? Nein, man macht Besprechungen und sieht sich den Code an? Aha, so ist das also in dieser Wissenschaft!

Nun, es müssen nicht immer Differentialgleichungen sein. Der Erfolg von Design Patterns und Refactoring beruht darauf, dass sie mehr sind als nur Kästchen und Pfeile (auch wenn sie so daherkommen); sie repräsentieren tiefes semantisches Wissen über guten Softwareentwurf, das empirisch gut abgesichert ist. Und zur neuen Synthese gehört auch, zu gegebenen Qualitäts- und Kostenvorgaben jeweils die adäquate Technologie zu verwenden.

5 Schluss

Softwaretechniker brauchen sich nicht mit überflüssigen Formalisierungen zu schmücken, nur um als Wissenschaftler anerkannt zu werden. Formal basierte Verfahren können sehr durchschlagskräftig sein, aber auch Einsichten über Softwarearchitektur und den Entwicklungsprozess sind gute Wissenschaft, wenn sie methodisch sauber erzielt werden. Rein bürokratische Ansätze wie ISO900x führen dagegen nicht weiter. In der Ausbildung ist ein ausgewogenes Verhältnis zwischen Theorie und Praxis wichtig, gekoppelt mit umfangreichen Praktika, nicht jedoch überspezialisierte Sub-Softwaretechniken für spezielle Geschäftsprobleme. Für die praktische Softwaretechnik gibt es inzwischen sehr gute Lehrbücher wie die von Balzert. Hätten wir gute Bücher über die „neue Synthese“, dann wäre es mit Feyerabends „Anything goes“ in der Softwaretechnik endgültig vorbei.

Literatur

[1] P. Feyerabend: Killing Time: The Autobiography of Paul Feyerabend. University of Chicago Press, 1995.

[2] P. Feyerabend: Against Method. Revised edition, Verso, London 1988.

[3] J. Ludewig: Erfolg macht Spaß: Softwaretechnik an der Universität Stuttgart. Proc. Informatik 2000, Springer 2000, S. 20-31.

[4] L. Osterweil et al.: The Impact Project: Determining the Impact of Software Engineering upon Practice. Proc. FSE'00, ACM 2000, S. 108-109.

[5] D. Schmid, M. Broy: Das Walberberg-Memorandum zur Förderung der IT-Forschung. Informatik-Spektrum, April 2000, S. 109-117.

[6] G. Snelting: Paul Feyerabend und die Softwaretechnologie. Informatik-Spektrum, Oktober 1998, S. 273-276.

[7] G. Snelting: Paul Feyerabend and Software Technology. Software Tools for Technology Transfer, November 1998, S. 1-5.

[8] W. Tracz, W. Stidolph (Eds.): NSF Software Engineering and Language Program Summaries. ACM Software Engineering Notes, Januar 2000, S. 30-94.

[9] S. Wendt: Softwaresystemtechnik – eine Informatik-Ingenieurdisziplin. `http://www.hpi.uni-potsdam.de/deu/sst-ingenieur/index.html`