Schwerpunkt

Die Gestaltung des Unsichtbaren

Reinhard Keil-Slawik*

Die Qualität von Software erweist sich erst im Einsatz. An dieser einfachen Wahrheit hat sich in den letzten zwanzig Jahren nichts geändert. Seit auf der NATO-Konferenz 1968 in Garmisch-Partenkirchen der Begriff Software Engineering als Ausdruck der Hoffnung geprägt wurde, es möge doch eine ingenieurmäßige Herangehensweise zur Softwareproduktion geben, sind zwar viele Defizite im Bereich der systematischen Programmentwicklung abgebaut worden, doch als eine wissenschaftlich begründete Ingenieurdisziplin kann die Softwaretechnik auch heute nicht charakterisiert werden.

Sowohl bezüglich der empirischen Fundierung als auch bezüglich der theoretischen Orientierung hat dieses Fachgebiet erhebliche Defizite aufzuweisen. Die Folge: immer noch gehören „Softwarekatastrophen" zum Alltag des Softwaretechnikers, wie dies u. a. die vielen Beispiele in dem Buch „Softwarereflexionen" von R. L. Baber belegen. Dabei dürfte die Dunkelziffer erheblich größer sein, denn sein Scheitern öffentlich zuzugeben, nagt an der Reputation.

Die rasante Erschließung neuer Einsatzbereiche sowie die schnelle technologische Entwicklung im Hardware-Bereich haben dazu geführt, daß es keine ausreichenden empirischen Untersuchungen gibt, anhand derer Entwicklungshypothesen systematisch überprüft werden könnten. Häufig wird beispielsweise die Entwicklung eines Produktes als „Beweis" für die Richtigkeit der Annahmen gewertet, die den dabei verwendeten Methoden und Techniken zugrunde liegen. Zu welch verheerenden Fehleinschätzungen dies führen kann, wird insbesondere im militärischen Einsatzkontext u. a. am Beispiel von SDI deutlich.

Die Ausarbeitung theoretischer Grundlagen hat sich nicht aus den Erfahrungen der Herstellung und des Einsatzes von Software ergeben, sondern die wissenschaftliche Fundierung wurde schlichtweg mit der Entwicklung mathematischer Beweisverfahren und Spezifikationstechniken ineins gesetzt. So kritisierte M. M. Lehmann 1979 im Infotech State of the Art Report, daß vieles, was zum damaligen Zeitpunkt als Verkörperung der Informatik betrachtet wurde, vollkommen losgelöst von jeglicher phänomenologischen Grundlage auf einer abstrakten mathematischen Ebene initiiert worden sei.

Trotz der großen Lücke, die heute noch zwischen Theorie und Praxis klafft, besteht im ökonomischen wie im wissenschaftlichen Bereich Einigkeit in der vorherrschenden Sichtweise, die Software vorrangig als ein eigenständiges Produkt ohne Bezug zum Herstellungs- und Einsatzkontext betrachtet. Oder ausgedrückt mit den Worten von Harry Sneed: „Um als Wirtschaftsgut zu gelten, muß ein geistiges Gut personenunabhängig, reproduzierbar, übertragbar und meßbar sein".

Wenn man aber die Aufmerksamkeit darauf lenkt, zu überprüfen, was denn bei der Herstellung und Benutzung von Software eigentlich passiert, dann wird deutlich, daß praktisch eingesetzte Software keine dieser Bedingungen erfüllt. Erste Hinweise erhält man bereits, wenn man sich die wesentlichen Unterschiede zwischen Software und anderen, ingenieurmäßig hergestellten Produkten vor Augen führt.

Nicht Fisch, nicht Fleisch

Software unterscheidet sich in vielerlei Hinsicht von anderen technischen Gebilden wie Brücken, Autos, oder auch Fernsehgeräten:

- Software besteht aus einem einheitlichen und zudem abstrakten Baustoff. Das Verständnis ist nur über beschreibende Texte und Graphiken oder die Erprobung eines bereits existierenden Produktes möglich.
- In der Softwaretechnik gibt es bisher keine universell verwendbaren Bausteine. Technische Gebilde werden aber nur selten vollkommen neu entwickelt; sie bauen in erheblichem Maße auf bestehende Teillösungen auf.
- Traditionelle ingenieurmäßige Probleme bestehen meist darin, eine neue technische Lösung für eine bereits bekannte Funktion zu entwickeln. Beispielsweise hat sich die Funktionalität eines Autos in vielen Jahrzehnten kaum geändert, wohl aber die technische Realisierung. Bei der Softwareentwicklung jedoch müssen die funktionellen Anforderungen überhaupt erst ermittelt werden.

*Wissenschaftler an der TH Berlin

Software-Entwicklung

Seit zwanzig Jahren geistert der Begriff des Software Engineering durch Tagungen, Kongresse, Bücher und Marketing-Broschüren. Allen Anstrengungen der Theoretiker und Software-Häuser zum Trotz bleibt die Software-Qualität und damit die Unzufriedenheit der Anwender, ihre Klagen über unkalkulierbare Risiken bei der Software-Entwicklung, ein Dauer- und Krisenthema. Neue Gedanken, neue Erkenntnisse, neue Ansätze werden in diesem Software-Magazin vorgestellt. Keine Rezepte, keine aufgebauschten Erfolgsmeldungen, keine Illusionen, sondern Nachdenkenswertes.

Redaktion: H. Haug

● Das Einsatzumfeld von Software ist in der Regel nicht klar umrissen, Anforderungen sind z. T. widersprüchlich und verändern sich mit der Zeit.

● Etwa die Hälfte des Codes praktisch eingesetzter Software besteht aus Benutzungsschnittstellen, Fehlermeldungen, Ausnahmebehandlungen und Kommentierungen; Aspekten also, die nicht aus softwareimmanenten Qualitätseigenschaften ableitbar und damit auch nicht formal „überprüfbar" sind, sondern nur unter Bezugnahme auf das Entwicklungs- und Einsatzumfeld definiert werden können.

● Software zeichnet sich durch eine enorme Fülle diskreter Betriebszustände aus und weist in der Regel nur eine sehr schwache repetitive Struktur (d. h. Wiederverwendung ein und desselben Bausteins) auf. Dadurch ist es schwierig, wie D. Parnas feststellt, die Zuverlässigkeit sicherzustellen, denn die herkömmliche technische Mathematik der kontinuierlichen Funktionen könne nicht zur Verifikation herangezogen werden. Dieser grundlegende Unterschied kann auch durch technologische Verbesserungen nicht aufgehoben werden.

● Softwareentwicklung ist in hohem Maße ein rückbezüglicher Prozeß. Beispielsweise verändert sich das einer Entwicklung zugrundeliegende Modell (bzw. die damit zusammenhängenden Anforderungen) des Einsatzkontextes durch den Einsatz des fertigen Produktes. Ein anderes Beispiel ist die Einführung eines Leistungsmerkmals bei der Programmentwicklung, wie z. B. die Anzahl der Programmzeilen, das unmittelbar auf den Programmierstil zurückwirkt. Damit verliert jeweils das Modell bzw. das Leistungsmerkmal seine Aussagekraft.

Software kann als Vergegenständlichung von Handlungsanweisungen aufgefaßt werden: der Entwickler legt fest, was zur Laufzeit des Programms wie, in welcher Reihenfolge und unter welchen Umgebungsbedingungen ausgeführt werden soll. Insofern kommt eine Auffassung, die Software als Plan bzw. eine Ansammlung von Plänen betrachtet, der Sache näher als die produktorientierte Sichtweise.

Mit dieser Sichtweise finden wir aber auch wieder den Anschluß zur traditionellen Ingenieurwissenschaft, denn Planungsprozesse offenbaren auch dort dieselben Probleme, die wir aus der Softwareentwicklung kennen: Kosten werden nicht eingehalten, Termine werden überschritten, das Ergebnis entspricht oft nicht den Vorstellungen der Auftraggeber und der Betroffenen und nicht alle Ereignisse, die die Ausführung des Plans beeinflussen, können vorhergesehen werden. Was ein Plan alles beinhaltet, kann man ihm nicht ansehen, sondern dies erschließt sich einem erst in der Umsetzung des Planes.

Dies würde auch erklären, warum die ansonsten im ingenieurwissenschaftlichen Bereich unerläßlichen Maße und Metriken in der Softwaretechnik keine vergleichbare Bedeutung haben; bis heute ist es nicht gelungen, einen Satz von allgemein anerkannten und brauchbaren Meßmethoden zu entwickeln. In der Einleitung zu einem Tutorium über quantitative Modelle betont Victor R. Basili, daß jedes der von ihm zusammengestellten Modelle vom Manager ausreichende Erfahrung und genaue Kenntnisse im Umgang mit dem Modell verlangt, bevor er beurteilen kann, inwieweit er den (Meß-)Ergebnissen vertrauen kann bzw. wie er sie zu interpretieren hat.

Die hier angeführten Merkmale haben zwangsläufig Konsequenzen für die Möglichkeiten und Grenzen von Methoden, Formalismen und Werkzeugen und erfordern daher auch eine geeignete theoretische Fundierung, die die Wesensmerkmale geistiger Arbeit und sozialer Prozesse in den Vordergrund der Betrachtung rücken muß.

Obwohl Programme im Prinzip leichter zu ändern sind als elektronische Bauteile, die jeweils neu gefertigt werden müssen, können große Softwaresysteme kaum noch geändert werden, weil die damit verbundenen Effekte nur schwer oder gar nicht zu durchschauen sind. Generell gilt, daß Computer die meiste Zeit nicht rechnen, sondern damit beschäftigt sind, Daten hin und her zu schaufeln und zu analysieren. Die Regeln, nach denen dies geschieht, können nicht in Form mathematischer Funktionen oder Gleichungen beschrieben werden, da sie meist nur eine komplizierte Ansammlung ad hoc gebildeter Regeln und Algorithmen repräsentieren, von denen man annimmt, daß sie das Problem korrekt modellieren.

Restrukturierung und Identität

Beim Entwurf von Software kommt es wesentlich darauf an, alle Sonderfälle und Ausnahmebedingungen vollständig zu erfassen und ordnungsgemäß zu verarbeiten. Entscheidend ist, inwieweit die Softwareingenieure das zu lösende Problem mit all seinen Randbedingungen durchschauen. Programmieren kann nach Peter Naur daher nicht in erster Linie als Produktion von Programmen und zugehörigen Texten betrachtet werden, sondern als ein Prozeß, bei dem die Programmierer eine Theorie darüber entwickeln, wie die vorhandenen Probleme durch die Programmausführung gelöst werden können. Da aber nicht alle bei der Systementwicklung auftretenden Probleme und Entscheidungen mit all ihren Wechselbezügen dokumentiert werden können, ist diese Theorie nur in den Köpfen der Entwickler vorhanden: „das Wiederherstellen der Theorie lediglich aufgrund der Dokumentation ist gänzlich unmöglich" (Naur, Übersetzg. d. A.).

Was aber in den Köpfen der Menschen ist, ist nicht sichtbar; erst ihr Handeln offenbart die Theorie und dies auch nur zu einem kleinen Teil. Dies wird besonders deutlich, wenn bereits geschriebene Programme erweitert oder an neue Einsatzbedingungen angepaßt werden müssen. Jede Änderung führt dazu, daß die Struktur des Programmes schlechter und damit der Einarbeitungsaufwand für die Programmierer größer wird. Schließlich ist es billiger, statt Änderungen vorzunehmen, ein neues Programm zu schreiben.

Ebensowenig wie es möglich ist, die hinter einem Programm stehende Theorie lediglich aufgrund von Dokumenten zu rekonstruieren, ist es unmöglich, alle Anforderungen von vornherein zu ermitteln und schriftlich niederzulegen, um dann – gemäß der Idealvorstellung des Phasenmodells – das Produkt nur unter Bezugnahme auf dieses Dokument zu entwickeln. Letztlich erfolgt die Entwicklung von Software immer in Zyklen von Analyse, Entwurf, Einsatz und Auswertung. Dabei sind die Fehlerbewertung wie auch die Bewertung der Angemessenheit eines Systems soziale d. h. kommunikative Prozesse, die nicht formalisierbar sind.

Ludwig Wittgenstein hat dargelegt, daß es nicht möglich ist, nur für sich allein einer Regel zu folgen, weil man dann keine Gewißheit haben kann, ob man auch korrekt gehandelt hat. Daran anknüpfend hat Jürgen Habermas in seiner „Theorie des kommunikativen Handels" festgestellt, daß die Identität von Bedeutungen letztlich darauf gründe, daß sich mindestens zwei Individuen, die sowohl über die Kompetenz zu regelgeleitetem Verhalten als auch zur kritischen Überprüfung dieses Verhaltens verfügten, sich verständigten.

Wissen ist nicht ein Zustand oder etwas, das man hat oder nicht hat. Wissen ist auch nicht das Geschriebene; Wissen manifestiert sich in Kommunikationen, in denen sich das Geschriebene entfaltet. Insofern kann man nur aufgrund eines Dokumentes bzw. Programms nicht entscheiden, um welches Produkt es sich handelt.

Software wird im Laufe der Entwicklung und des Einsatzes mehrfach geändert, erweitert und restrukturiert. Die Frage, ob es sich bei zwei Programmen um zwei Versionen desselben Produktes handelt oder um zwei verschiedene Produkte,

kann auch nicht aufgrund formaler oder softwareimmanenter Eigenschaften entschieden werden. In der Regel werden Entwickler und Benutzer auch bei weitreichenden Änderungen von demselben Produkt sprechen, wenn die Gründe für solche Änderungen nicht in einem Wechsel der ursprünglich gesetzten Absichten und Ziele zu suchen sind, sondern als Lernprozeß charakterisiert werden können, um diesen Zielen näherzukommen. Lernen im Kontext sozialer Verständigung ist die wesentliche Voraussetzung für intelligentes Verhalten; es ist die Voraussetzung für kreative Softwareentwicklung schlechthin.

Kreieren statt Generieren

Die Unterstützung von Kommunikations- und Lernprozessen ist bislang in der Softwaretechnik – von wenigen Ausnahmen abgesehen – nicht thematisiert worden. Insbesondere hat man sich bei der Entwicklung von Methoden und Werkzeugen selten auf die kognitiven Fähigkeiten des Menschen bezogen, sondern vielmehr auf die formalen Eigenschaften und Merkmale, die das mithilfe dieser Mittel zu erstellende Dokument bzw. Programm aufweisen soll, wie z. B. Vollständigkeit, Eindeutigkeit, Widerspruchsfreiheit, Konsistenz, Korrektheit usw.
Aus der Tatsache, daß Programme formal interpretierbar sind (eben durch die Maschine), wird häufig abgeleitet, daß auch die Verständigung über Programme und ihre Eigenschaften auf einer formalen Grundlage erfolgen muß. Dabei wird unterschlagen, daß diese Eigenschaften erst am Ende einer Entwicklung gegeben sind; zwischendurch muß man mit Inkonsistenzen, Unvollständigkeiten, Widersprüchen, unterschiedlichen Sichten und Interessen leben. Werden sie zu früh beseitigt oder läßt man sie gar nicht erst aufkommen, ist der Mißerfolg quasi vorprogrammiert. Dies gilt erst recht angesichts der Vielfalt unterschiedlicher Rollen und Interessen, die im Rahmen der Systementwicklung von Bedeutung sind.

Zu sehr hat sich die Softwaretechnik darum bemüht, den Entwicklungsprozeß durch Regelwerke organisatorischer und technischer Art zu beherrschen. Das fortwährende Scheitern von Projekten wurde darauf zurückgeführt, daß formale Verfahren noch nicht genügend eingesetzt würden, Methoden nicht strikt genug angewendet würden oder die Werkzeuge noch nicht umfassend genug den gesamten, alle Aspekte umfassenden Entwicklungsprozeß unterstützen. Das Ergebnis hat Ernst Denert im letzten Jahr im computer magazin als „Software-Bürokratie der 80er Jahre" charakterisiert.
Zwar helfen Methoden und Werkzeuge bei der Überprüfung und bei der Erzeugung von Dokumenten, doch je mehr sie dem Menschen Verhalten vorschreiben und je weniger sie eine individuelle und flexible Vorgehensweise erlauben, desto weniger unterstützen sie die Entwicklung qualitativ hochwertiger Software. Die wenigen bisher vorliegenden empirischen Untersuchungen deuten darauf hin, daß die Güte von Programmen mit der Möglichkeit in Zusammenhang steht, durch flexible Regeln die individuelle Entfaltung der am Herstellungsprozeß beteiligten Menschen zu ermöglichen, indem je nach Problemsituation unterschiedliche und nur zum Teil formalisierte Hilfsmittel eingesetzt werden.
Gestaltung des Unsichtbaren heißt folglich, Methoden und Werkzeuge so zu gestalten, daß sie die Prozesse der Herstellung und Benutzung angemessen unterstützen. Wir werden diese Prozesse nicht mit Vorschriften und Regeln beherrschen können, weil soziale Systeme selbstorganisierend sind. Wir können soziale Prozesse nicht steuern aber beeinflussen. Wir können kooperatives Verhalten nicht durch Vorschriften erzwingen aber durch unser Verhalten fördern. Und wir können kreative Lösungen nicht aus Methoden ableiten oder mit Werkzeugen generieren. Was wir aber machen können ist, mit Hilfe technischer und organisatorischer Maßnahmen das Entwicklungsmilieu so zu gestalten, daß einige Ereignisse wahrscheinlicher eintreten als andere.