Softwarequalität aus Nutzersicht und ihre wirtschaftliche Bewertung

Albert Endres

Informatikerinnen und Informatiker haben eine besondere Verantwortung für die von ihnen entwickelten Produkte, gleichgültig ob sie aus Hardware oder aus Software bestehen. Zu dieser Verantwortung müssen sie stehen.

Leider werden viele Diskussionen über Softwarequalität sehr oberflächlich geführt. Deshalb möchte ich versuchen, einige Konzepte und Zusammenhänge zu verdeutlichen. Anschließend stelle ich eine Studie vor, die den wirtschaftlichen Schaden ermittelt, der einzelnen Branchen bzw. einer ganzen Volkswirtschaft infolge fehlerhafter Software entsteht.

Wirtschaftliche Bedeutung von Software

Eigentlich braucht man hierzu kaum noch ein Wort zu verlieren. Es gibt inzwischen viele überzeugende Aussagen, die geradezu gebetsmühlenhaft wiederholt werden. Dennoch soll auf zwei quantitative Angaben kurz Bezug genommen werden. Das Bundesministerium für Bildung und Forschung (BMBF) hat im Jahre 2001 einen Bericht zur Lage der Softwareentwicklung in Deutschland vorgelegt [2]. Darin wird die Wertschöpfung durch Software im Jahre 2000 auf 25 Mrd. Euro beziffert, geleistet von insgesamt 19.000 Firmen. In einer kürzlich vom amerikanischen Normungsinstitut (NIST) veröffentlichten Studie [5] wird der Umsatz an Softwareprodukten in den USA im Jahre 2000 mit 180 Mrd. US-Dollar angegeben, d. h. etwa siebenmal so viel wie in Deutschland. Dahinter stünden etwa 1,3 Mio. hoch bezahlte Fachkräfte, teils bei Entwicklern, teils bei Anwendern.

In beiden Berichten wurde nur diejenige Wertschöpfung durch Software berücksichtigt, die sich in zuordnenbaren Umsätzen niederschlug. Ein Großteil der Erträge aus Individualanwendungen oder eingebetteten Systemen sind vermutlich nicht darin enthalten, ebenso wenig die viel diskutierte „freie“ Software (Open Source).

Softwarequalität – nicht nur Stoff für Anekdoten

Populärwissenschaftliche oder mediengerechte Darbietungen schmücken sich gerne mit individuellen Horrorgeschichten über Softwareprobleme. Die wegen ihrer Wissenschaftlichkeit angesehene Fachgesellschaft ACM hat dafür ein weltbekanntes Archiv geschaffen, bestehend aus Zeitungsschnipseln und E-Mail-Fragmenten [6]. Diese Sammlung von Anekdoten muss bei jeder passenden und unpassenden Gelegenheit herhalten. Dass man ganz gut auf dieser Welle reiten kann, zeigt ein Beispiel, das ich vor kurzem in einer bekannten Wochenzeitung fand. In einem öffentlichen Vortrag in einer deutschen Großstadt gab ein aus den USA angereister Fachkollege seine Erfahrungen mit einem bestimmten Textverarbeitungssystem zum Besten. Das System wolle partout seinen Firmennamen, der mit einem Kleinbuchstaben beginnt, großschreiben. Also sei das System schlecht und der Hersteller zu rügen. Niemand widersprach dem Redner oder wies ihn darauf hin, dass man die Vorschläge der Rechtschreibprüfung abschalten oder seinen Firmennamen in das Benutzerwörterbuch einfügen kann. Die kuriose Erkenntnis dieses sog. Fachmanns war das Einzige, was über die Veranstaltung berichtet wurde (natürlich mit genauem Produkt- und Firmennamen).

Ich möchte annehmen, dass professionell ausgebildete Informatiker sich von derlei Anekdoten nicht beeindrucken lassen. Sie wissen, dass sie Softwarequalität nicht auf die leichte Schulter nehmen

Albert Endres
Sindelfingen

Relevanz	Kriterium	Definition
Nutzer	Funktionalität	Umfang der von Nutzern erwarteten Dienste
	Verfügbarkeit	Leichter und dauernder Zugang
	Zuverlässigkeit	Niedrige Problemrate
	Effizienz	Sparsamer Verbrauch von Betriebsmitteln
	Installierbarkeit	Leichtes und schnelles Hochfahren in der Nutzerumgebung
	Nutzbarkeit	Den Fähigkeiten und Vorlieben der Nutzer gut angepasst
	Robustheit	Sanfte Reaktion auf Nutzer- und Gerätefehler
	Sicherheit	Geringer Schaden bei unachtsamer oder böswilliger Nutzung
Beide Seiten	Testbarkeit	Gute Diagnose-Hilfsmittel, Dokumentation und Struktur
	Wartbarkeit	Leichte Änderbarkeit, gute Lesbarkeit
Entwickler	Portabilität	Geringe Abhängigkeit von technischer Umgebung
	Regionalisierbarkeit	Anpassbar an nationale und regionale Erfordernisse
	Wiederverwendbarkeit	Hohe Modularität und Vollständigkeit

Abb. 1 Wichtige Qualitätskriterien für Software

können. Sie kennen (hoffentlich) die einschlägigen Definitionen, z. B. die in ISO 9126 [3]. Sie wissen daher, dass Softwarequalität leider ein mehrdimensionales Problem darstellt. Gemeint ist, dass mehrere Aspekte gleichzeitig bedacht werden müssen, die nicht selten miteinander zusammenhängen oder gar im Widerspruch zueinander stehen.

In Abb. 1 sind die (leicht abgewandelten) ISO-Kriterien nach der Relevanz für Nutzer und Entwickler gruppiert. Diese Unterscheidung verdeutlicht, dass der Schwerpunkt auf der Nutzerseite liegt. Sämtliche Kriterien im Einzelnen zu diskutieren, möchte ich mir ersparen. Nur so viel: Das Kriterium Zuverlässigkeit wird oft stellvertretend für alle benutzt, ist aber nur eines von vielen. Einige Leser werden in dieser Liste das Kriterium Korrektheit vermissen. Es ist hier nicht aufgeführt, weil es im Gegensatz zu allen anderen ein binäres Kriterium ist. Es trifft entweder zu oder nicht. Um seine Erfüllung zu prüfen, muss eine mathematisch präzise, unabhängig erstellte Spezifikation gegeben sein, was in der Praxis selten der Fall ist. Außerdem ist es, abgesehen von trivialen Beispielen, für Praktiker recht aufwendig und schwierig, den Vergleich von Spezifikation und Programm durchzuführen.

Ein erfolgreicher Ansatz, um sich über Qualität Klarheit zu verschaffen, besteht darin, sich Gedanken über potenzielle Abweichungen zu machen. So kommt man von den in Abb. 1 gelisteten Kriterien zu einer ersten Aufteilung von Softwarequalitätsmängeln. Es gibt Funktionalitätsprobleme, Verfügbarkeitsprobleme, Zuverlässigkeitsprobleme, Effizienzprobleme usw. Sehr hilfreich ist auch eine Aufteilung nach dem Entstehungszeitpunkt von Fehlern innerhalb des Entwicklungsprozesses. Dann unterscheidet man zwischen Anforderungsfehlern, Entwurfsfehlern, Implementierungsfehlern usw. Seit über 30 Jahren ist bekannt, dass die in frühen Phasen entstandenen Fehler nicht nur zahlenmäßig überwiegen, sondern auch die gravierendsten sind. Bei manchen Projekten sind über 60% aller Fehler Anforderungs- und Entwurfsfehler.

In der Praxis messbare, nutzerbezogene Softwarequalität

Nutzer sehen keine Fehler (engl.: faults), sondern nur Probleme (engl.: failures), sagte einst Harlan Mills. Ein aus der Nutzerperspektive relevantes Maß ist z. B. Probleme pro Nutzermonat (engl.: problems per user month, abgekürzt PUM). Es ist dies auch in der Literatur durchaus bekannt. Eine umfassende Behandlung bringt beispielsweise Kan [4]. Die zahlenmäßige Umrechnung von Problemen in Fehler ist meist recht schwierig. Der entsprechende Zusammenhang ist in Abb. 2 dargestellt. Eine kurze Erklärung dazu: Der Kreis in der Mitte steht für die Menge der einem Hersteller oder Serviceanbieter in Bezug auf ein bestimmtes Produkt von allen Kunden gemeldeten (angeblichen) Probleme. Falls es eine kundeninterne Systemunterstützung gibt, wird bereits ein großer Teil von Problemen vorher gelöst. Diese treten nicht als Servicefälle in Erscheinung. Die extern berichteten Probleme entfallen auf zwei Hauptgruppen, Duplikate und nicht reproduzierbare Probleme. Duplikate sind die Probleme, die früher bereits bei anderen Kunden aufgetreten waren. Oft können auch mehrere Probleme, die ein einzelner Nutzer hat, durch denselben Fehler verursacht sein. Nicht reprodu-

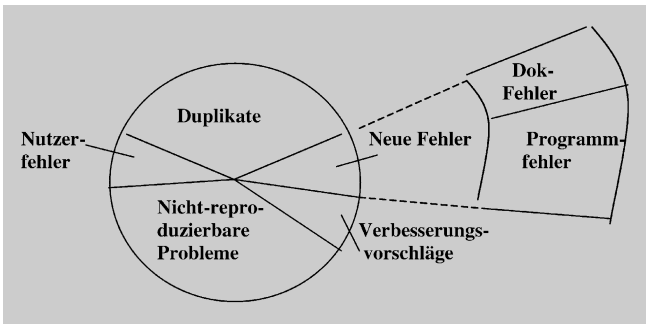

Abb. 2 Von Problemen zu Fehlern

zierbar sind die Probleme, die nur beim Zusammentreffen von bestimmten Zeitbedingungen oder Systemlasten auftreten. Es war in diesen Fällen nicht möglich, dieselbe Situation mit sinnvollem Aufwand wieder herzustellen.

Die restlichen drei kleinen Gruppen enthalten Nutzerfehler, Verbesserungsvorschläge und neu aufgetretene Fehler. Letztere zerfallen in Dokumentationsfehler und Programmfehler. Als Probleme (im engeren Sinne), um die sich der Serviceanbieter kümmern muss, gelten Duplikate, nicht reproduzierbare Probleme und neue Fehler. Nicht dazu rechnen Nutzerfehler und Verbesserungsvorschläge, aus denen natürlich wertvolle Information gewonnen werden kann.

Die relative Aufteilung stammt zwar aus dem Großsystembereich, wird sich aber im PC-Bereich nicht völlig geändert haben. Wichtig ist zu verstehen, dass die beiden größten Problemgruppen nicht direkt von der inhärenten Zuverlässigkeit des Produkts abhängen, sondern eher von seiner Testbarkeit und der Wartungsstrategie. Das Kriterium Testbarkeit soll hier die Fähigkeit mit einschließen, Produktprobleme leicht zu reproduzieren und zu diagnostizieren. Die Wartungsstrategie ist dafür verantwortlich, wie Korrekturen angeboten und installiert werden. Werden Korrekturen nur an solche Kunden geliefert, die das betreffende Problem berichtet haben, ist es unvermeidbar, dass die Zahl der Duplikate anwächst. Wird angenommen, dass alle Kunden früher oder später alle Funktionen des Systems benutzen, so werden sämtliche Nutzer auch nach und nach mit allen Problemen konfrontiert. Werden jedoch alle Korrekturen möglichst frühzeitig an alle Kunden verteilt und diese ermutigt oder gedrängt, die Korrekturen zu installieren, auch wenn sie das betreffende Problem nicht hatten (auch vorbeugende Wartung genannt), dann lässt sich die Zahl der Duplikate niedrig halten.

Ob die Installation von Korrekturen sehr schnell und ohne Risiko vonstatten geht, hängt sehr davon ab, in welcher Form die Korrekturen angeboten werden. Ganze Moduln in Objektform zu ersetzen, ist ein großer Fortschritt gegenüber der Verteilung einzelner Quellcodezeilen. Das in diesem Falle erforderliche Kompilieren und Montieren kostet Zeit und verursacht Nutzerfehler.

Das Gesagte soll keinesfalls davon ablenken, dass es enorm wichtig ist, die Fehlerdichte von Softwareprodukten zu reduzieren. Dass dies auch in der Industrie möglich ist, belegt Abb. 3. Es handelt sich dabei um die tatsächlich erreichten Werte für ein etwa zwei Millionen Programmzeilen umfassendes Produkt (Betriebssystem DOS/VSE von IBM), an dessen Entwicklung der Autor beteiligt war. Wie zu sehen, wurde die Fehlerdichte, gemessen in Feldfehlern pro 1000 Quellcodezeilen (KLOC), um eine Größenordnung reduziert. Etwa alle zwei Jahre wurde eine neue Systemversion ausgeliefert. Gezählt wurden die bei Kunden gefundenen neuen Fehler innerhalb der ersten 48 Monate nach Verfügbarkeit der jeweiligen Produktversion. Die zur Qualitätsverbesserung benutzten Methoden sind als erfolgreiche Methoden seit langem bekannt. Es ist die Kombination von Inspektion und Testen.

Interessanter als der in absoluten Zahlen ausgedrückte Qualitätsfortschritt ist in diesem Beispiel der dafür benötigte Zeitraum. Die Ergebnisse wurden erzielt durch harte, zielgerichtete Arbeit, und dies nicht über Monate, sondern über Jahre hinweg. Dahinter steckte ein organisatorischer Lern- und Reifeprozess, weniger ein technischer Durchbruch oder ein singulärer Kraftakt. Nicht nur waren die Kosten auf der Entwicklerseite beträchtlich, auch mussten andere Ziele, vor allem der geplante Zuwachs an Funktionalität, etwas zurückgenommen werden. Im Endeffekt konnten auch die PUMs erheblich reduziert werden. Sie fielen, wenn ich mich richtig erinnere, von 0,5 (bei einem Nutzer alle zwei Monate ein Problem oder bei je zwei Nutzern jeden

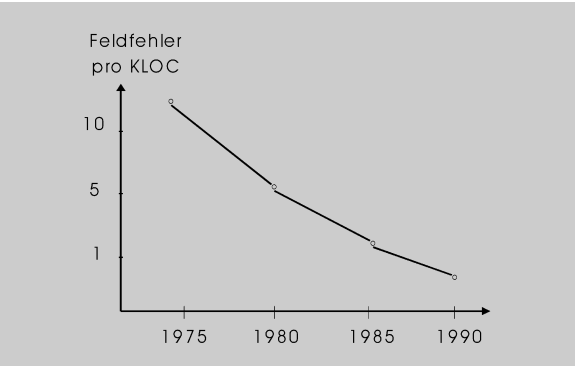

Abb. 3 Beispiel industrieller Qualitätsverbesserungen

Monat ein Problem) auf 0,04 (bei einem Nutzer ein Problem alle 24 Monate oder bei je 24 Nutzern jeden Monat ein Problem). Dabei hatte die Umstellung der Wartungsstrategie in Richtung einer vorbeugenden Wartung mindestens eine gleich große Wirkung wie die Qualitätsverbesserungen am Produkt. Allein auf der Herstellerseite wurden Wartungskosten im Bereich von mehreren Millionen US-Dollar eingespart. Die finanziellen Vorteile auf der Nutzerseite waren sicherlich höher, wurden aber nicht erfasst.

Oft werden Aussagen über große Softwaresysteme gemacht von der Art „Das System X soll 10.000 Fehler enthalten!“ Das schafft natürlich Verwirrung bei Laien und Nichtpraktikern. In Wirklichkeit wird damit nur eine Aussage getroffen über die Größe des Systems. Legt man eine Rate von einem Fehler pro KLOC zugrunde (eine Zahl, die bei gewissen Produktklassen realistisch ist), so heißt dies, dieses System umfasst vermutlich etwa 10 Mio. Programmzeilen. Da der einzelne Nutzer vielleicht 15% des Codes überhaupt nutzt und einen Teil davon erst nach zwei Jahren, wird er auch nur von einem Bruchteil der Fehler jemals betroffen sein. Nutzungsbezogene Aussagen, seien es PUMs oder eine Weiterentwicklung davon, wären da viel sinnvoller. Am nützlichsten sind Maße und Methoden, die das Qualitätsempfinden jedes einzelnen Nutzers messen und bewerten. In diese Richtung gehen die Kundenbefragungen, die von der Industrie regelmäßig durchgeführt werden. Auch diese sind bei Kan [4] ausführlich beschrieben.

Bemerkt sei, dass auch die mit Hilfe mathematischer Modelle abgeleiteten Qualitätsprognosen vielfach keine neuen Erkenntnisse liefern. Mit ihrer Hilfe wird in der Regel von den Ergebnissen diverser interner Tests auf das spätere Feldverhalten hochgerechnet. Solche Modelle ignorieren sehr oft den in Abb. 2 dargestellten Unterschied zwischen Problemen und Fehlern. Manchmal gehen sie auch von einer Gleichverteilung von Fehlern über alle Komponenten und Moduln aus. In der Praxis findet man meist eine sog. Pareto-Verteilung, etwa der Art, dass 80% der Fehler sich in 20% der Moduln befinden.

Sehr schwer ist es auch, die Güte der Testfälle im Modell zum Ausdruck zu bringen. Schließlich ist weder das Argument einleuchtend, dass eine Komponente oder ein System umso zuverlässiger ist, je mehr Fehler während des Testens auftreten, noch das umgekehrte Argument, dass eine geringe Zahl aufgedeckter Fehler auf eine hohe Qualität hindeutet. Dass einige Prognosemodelle dennoch Erfolg haben, mag damit zusammenhängen, dass hin und wieder auch da Korrelationen festzustellen sind, wo es keine Kausalität gibt. Für eine Diskussion weiterer empirisch gewonnener Einsichten sei auf Endres und Rombach [1] verwiesen.

Durch Softwaremängel verursachter volkswirtschaftlicher Schaden

Der betriebswirtschaftliche Schaden, der für einen individuellen Anwender infolge von Softwareproblemen entsteht, kann sehr beträchtlich sein. Wenn eine Bank Buchungen falsch ausführt oder ein Versandhaus während der Vorweihnachtszeit nicht am Netz sein kann, hat dies weit reichende Auswirkungen. Soweit mir bekannt ist, wurden bisher noch nie für eine Branche oder gar für eine Volkswirtschaft als Ganzes Schätzungen des Effekts von Softwarefehlern vorgenommen. Genau dies wird in der bereits erwähnten NIST-Studie [5] versucht. Die Zahlen, die dabei entstanden, sind für deutsche Leser sekundär.

Interessant ist jedoch die Methodik, die verwandt wurde. Am Beispiel der Automobil- und Flugzeugindustrie soll diese vorgestellt werden. Die für diese Branche maßgebliche Software sind graphische Entwurfssysteme, die unter der Bezeichnung CAD/CAM-Systeme bekannt sind. Der Markt für diese und verwandte Produkte hatte im Jahre 2000 in den USA einen Umfang von etwa 11 Mrd. US-Dollar, also 6% des gesamten Softwaremarkts. Eine weitere Branche, die ähnlich ausführlich untersucht wurde, ist das Finanzwesen, d. h. Banken und Versicherungen.

Für die ausgewählte Branche wurden die durch fehlerhafte Software verursachten Kosten getrennt

Aktivität	Anford., Entwurf	Codieren, Modultest	Integration, Systemtest	Betatest	Wartung (n.Auslief.)	Gesamtkosten
Kosten/Fehler	2	2,4	4,1	6,2	13,1	(5,6)
Fehlerverteilung						
– Ideal	16	80	4	0	0	240
– Aktuell	5	40	45	5	5	387
– Ziel	5	60	30	3	2	322

Abb. 4 Vergleich von Entwicklerkosten (in Stunden)

nach Entwicklern und Nutzern ermittelt. In beiden Fällen wird ein spezielles Kostenmodell zugrunde gelegt, dessen Parameter durch Befragung ermittelt wurden. Die Anzahl der ausgefüllten Fragebögen war allerdings relativ klein (10 Entwickler, 179 Nutzer). Für die Entwicklerseite wurde das in Abb. 4 angedeutete Modell verwandt. Die Aufteilung der Aktivitäten in Phasen ist recht grob und durchaus verbesserbar.

Das Modell basiert auf der empirisch gesicherten Erkenntnis, dass die Kosten für einen Fehler umso höher sind, je größer seine Lebensdauer ist und je später er behoben wird. In der ersten Zeile sind die Kosten eines Fehlers in Personenstunden angegeben, und zwar in Abhängigkeit von der Projektphase, in der er behoben wird. Der Durchschnitt beträgt 5,6 Stunden. In den Zeilen darunter sind Fehlerverteilungen angegeben, und zwar in Prozent. Die Idealverteilung gibt an, wie viele Fehler nach Aussage der Entwickler derzeit in welchen Phasen entstehen. Wenn sie alle in der gleichen Phase behoben werden könnten, wäre dies das Beste, was erreicht werden könnte. Dass man auch die Anzahl der entstandenen Fehler durch bessere konstruktive Maßnahmen (z. B. bessere Anforderungs- und Entwurfsmethoden) reduzieren könnte, ist außer Betracht gelassen. Die aktuelle Verteilung entspricht der augenblicklichen Aufdeckungsrate der befragten Entwickler. Als Ziel ist eine Verteilung angegeben, die realistischerweise erreicht werden könnte, wenn bessere Methoden und Werkzeuge für Verifikation und Testen zur Verfügung stünden und eingesetzt würden. Welche das sind, soll hier nicht weiter ausgeführt werden. Der augenblickliche Prozess ist um 147 Stunden (61%) teurer als der ideale Prozess. Verbesserte Methoden und Werkzeuge könnten 65 Stunden einsparen. Man läge dann immer noch 82 Stunden (34%) über dem Ideal. Dass in der Branche nur 16% statt der oben erwähnten 60% aller Fehler als Anforderungs- und Entwurfsfehler angesehen werden, lässt sich vielleicht dadurch erklären, dass in diesem Falle die Entwickler mehr mit dem Anwendungsgebiet als mit den Entwicklungsmethoden vertraut waren. Bei einer anderen Verteilung wären die Einsparungen wesentlich höher und damit die in den Zahlen enthaltene Botschaft noch deutlicher.

Völlig anders, aber nicht weniger interessant, ist der Ansatz bei den Nutzerkosten. Wie in Abb. 5 dargestellt, wurden diejenigen Kosten untersucht, die einem typischen Anwender, nämlich einer Firma mit 10.000 Mitarbeitern, für das Produkt entstehen, und zwar zusätzlich zu den Anschaffungskosten. Die aus der Umfrage gewonnenen (und in der ersten Zeile angegebenen) aktuellen Kosten betragen insgesamt 754.000 US-Dollar. In den zwei folgenden Zeilen ist die geschätzte Ersparnis angegeben, wenn die Fehlerverteilung so aussieht, wie in der letzten Zeile von Abb. 4 angenommen. In ihrer Summe ergeben sich Ersparnisse von 281.000 US-Dollar (37%). Bei den Kosten der Produktauswahl sind alle vor der Installation der Anwendung liegenden Aktivitäten inbegriffen. Dabei sind auch Kosten berücksichtigt, die dadurch entstehen, dass sich die Entscheidung für ein Produkt in der Regel um mehrere Monate verschiebt, da es nur unzureichende Informationen über dessen Qualität gibt. Hier werden auch die höchsten Einsparmöglichkeiten (42%) gesehen. Die zusätzliche Gerätekapazität wird erforderlich, weil die erwartete Effizienz nicht erreicht wird oder weil Ausweich- oder Rückfallmöglichkeiten für den Fehlerfall bereitgehalten werden. Auffallend ist, dass die Ersparnisse für die eigentliche Wartungsaktivität mit 14% am niedrigsten ausfallen. Auch hier ist die Beschränkung auf nur vier Kostenarten eine Vereinfachung. Außer im Falle von Effizienz ist es daher schwer, einen eindeutigen Bezug zu den verschiedenen Qualitätskriterien aus Abb. 1 herzustellen. Das gilt insbesondere für die Kriterien Nutzbarkeit, Robustheit und Sicherheit.

Aktivität	Produktauswahl	Installation, Einführung	Wartung	Zusätzl. Kapazität	Gesamtkosten
Aktuelle Kosten	500	160	77	17	754
% Ersparnis	42	27	14	100	(37)
Zielkosten	290	117	66	0	473

Abb. 5 Vergleich von Nutzerkosten (in 1000 US-Dollar)

Mit beiden Modellen wird zuerst auf die ganze Automobil- und Flugzeugbranche hochgerechnet. In den Entwicklerfirmen für die in Betracht gezogenen Softwareprodukte sind 85.000 Mitarbeiter beschäftigt. Als Anwender sind große und kleine Firmen getrennt behandelt worden, mit insgesamt 2,6 Mio. bzw. 0,5 Mio. Mitarbeitern. Die für die Branche im Jahre 2000 durch fehlerhafte Software verursachten Kosten (gegenüber dem Idealfall in Abb. 4) wurden auf 1,8 Mrd. US-Dollar errechnet. Das entspricht etwa 16% des Softwareumsatzes der Branche. Etwa ein Viertel davon betrifft die Entwickler und Dreiviertel die Anwender. Die Einsparmöglichkeiten aufgrund besserer Methoden und Werkzeuge (was dem Zielfall in Abb. 4 entspricht) wurden auf 590 Mio. US-Dollar oder 5% des Softwareumsatzes der Branche geschätzt.

Die Hochrechnung von einzelnen Branchen auf die gesamte Volkswirtschaft ergab dann die astronomisch anmutenden Zahlen von 59,5 bzw. 22,2 Mrd. US-Dollar. In beiden Fällen sind die absoluten Zahlen wegen der Größe des US-Marktes zwar beeindruckend, die relativen Zahlen sind es aber nicht. Geht man nämlich von der Einsparung aufgrund besserer Methoden aus, könnte man auch hier zu dem Schluss kommen, dass es leichter ist, den Ertrag oder den Nutzen der entsprechenden Produkte um 5–10% zu steigern, als die Kosten um 5–10% zu senken. Obwohl man manchen Softwareherstellern solche Überlegungen gerne unterstellt, können wir nicht umhin, die durch die Nutzer erfahrene Qualität unserer Produkte stetig zu steigern. Nur so lässt sich das Renommee der Softwareindustrie langfristig verbessern. Obwohl diese Studie im zweiten Teil der Sichtweise einzelner Nutzer sehr entgegenkommt, sind auch hier bessere Maße und Methoden vorstellbar.

Fazit

Die Qualität von Softwareprodukten ist ein zu ernstes Problem, um es Anekdotenerzählern zu überlassen. Hier systematische Fortschritte zu erzielen, ist eine Herausforderung, der nicht allein mit Hilfe rigoroser technischer Maßnahmen begegnet werden kann. Dabei ist die Nutzersicht entscheidend und nicht die Entwicklersicht. Beide in diesem Beitrag skizzierten Ansätze bilden zwar eine Basis für eine sachgemäße Betrachtung, regen aber zu vielen Verbesserungen an. An diesen zu arbeiten, wäre nützlich.

Um in einer gegebenen Praxissituation Aussagen zu treffen, sind tiefer gehende Kenntnisse von relevanten Produkten und Prozessen unabdinglich, sowie die Beachtung des dem Nutzer tatsächlich entstehenden Schadens. Wenn Zahlen benutzt werden, dann sollte man dies sehr sorgfältig tun. Vergleiche zwischen Firmen und Branchen sind fast immer problematisch. Mit Pauschalurteilen und tendenziösen Statistiken ist niemandem gedient. Nur eine differenzierende Betrachtung macht Sinn. Wie die erwähnte Studie zeigt, ist der durch fehlende Softwarequalität verursachte volkswirtschaftliche Schaden zwar erheblich, jedoch weder horrend noch unermesslich. Qualität ist aber mehr als nur ein wirtschaftliches Ziel. Es sollte das Anliegen einer gewissenhaften Informatikausbildung sein, diese Einsichten zu vermitteln und zu fördern.

Literatur

1. Endres, A., Rombach, H.D.: A handbook of software and systems engineering. Empirical observations, laws and theories. Harlow: Addison Wesley 2003
2. Friedewald, M., Rombach, H.D., Stahl, P., et al.: Softwareentwicklung in Deutschland. Informatik Spektrum 24(2), 81–90 (2001) (s. auch BMBF-Studie: Analyse und Evaluation der Software-Entwicklung in Deutschland, Dezember 2000; http://www.dlr.de/IT/IV/Studien/evasoft_abschlussbericht.pdf)
3. ISO International Standard ISO/IEC 9126. Information technology. Software product evaluation. Quality characteristics and guidelines for their use. International Organization for Standardization, International Electrotechnical Commission, Geneva 1991. http://www.cse.dcu.ie/essiscope/sm2/9126ref.html
4. Kan, S.H.: Metrics and models in software quality engineering. Reading/MA: Addison Wesley 1995
5. National Institute of Standards and Technology (NIST): The economic impacts of inadequate infrastructure for software testing. Planning report 02–3, prepared by RTI Health, Social, and Economic Research, Research Triangle Park, NC, May 2002. http://nist.gov/director/prog-ofc/report02–3.pdf
6. Neumann, P.: Risk to the public (ständige Rubrik seit etwa 1992). In: Software Engineering Notes; Newsletter of ACM Special Interest Group on Software Engineering (s. auch http://www.csl.sri.com/users/neumann/)