FREDERICK P. BROOKS JUN.

VOM MYTHOS DES MANN-MONATS

ESSAYS ÜBER SOFTWARE-ENGINEERING

mitp

mitp

Hinweis des Verlages zum Urheberrecht und Digitalen Rechtemanagement (DRM)

Vom Mythos des Mann-Monats

Frederick P. Brooks jun.

Vom Mythos des Mann-Monats

Essays über Software-Engineering

Übersetzt aus dem Amerikanischen von Arne Schäpers

Bibliografische Informationen 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.

ISBN 978-3-95845-877-2
1. Auflage 2019

Printed in Germany

Übersetzung aus dem Amerikanischen: Arne Schäpers
Übersetzung der ursprünglichen Ausgabe von 1975: Arne Schäpers, Armin Hopp
Lektorat: Katja Völpel
Korrektorat: Petra Heubach-Erdmann
Satz und Layout: Arne Schäpers
Umschlaggestaltung: Christian Kalkert

Über den Autor

Frederik P. Brooks jun. ist Professor Emeritus für Computerwissenschaften an der Universität von North Carolina in Chapel Hill und bekannt als „Vater des IBM System/360“, dessen Entwicklung er zunächst als Projektmanager leitete; später war er Leiter des Software-Projekts Operating System/360 während seiner Designphase. Für diese Arbeit erhielt er zusammen mit Bob Evans und Erich Bloch im Jahre 1985 die *National Medal of Technology*. Zuvor war er einer der Architekten des IBM Stretch und des IBM Harvest-Computers.

In Chapel Hill gründete Dr. Brooks das Department of Computer Science, dem er von 1964 bis 1984 vorstand. Er war sowohl Mitglied des National Science Board als auch des Defense Science Board. Seine aktuellen Lehr- und Forschungsgebiete sind Computer-Architektur, molekulare Grafik und virtuelle Umgebungen.

Lernaea-Hydra: Sie wurde in dem Lerna, einem Pfuhle in Argolien auferzogen und verwüstete das flache Land weit und breit, weil sie Menschen und Vieh hinwegraubete. Sie war von einer ungeheuern Größe, hatte neun Köpfe, unter welchen der mittelste unsterblich war. Diese zu erlegen, wurde dem Herkules von dem Eurysiheus anbefohlen, der sich denn mit dem Jolaus auf seinem Wagen dahin machte. Er traf sie auf einem Hügel, bey der Amymone Brunnen, in ihrem Lager an, ließ den Wagen stehn, griff sie mit feurigen Pfeilen an, und zwang sie also, sich hervor zu machen. Er fassete sie darauf selbst mit den Händen: sie schlug sich aber dagegen um so genauer um seine Beine herum. Allein, indem er anfing, mit seiner Keule ihr die Köpfe abzuschlagen, so sah er, daß an eines Stelle sofort zween andere hervorkamen. Dabey kam ihr noch ein ungeheurer Krebs zu Hülfe, der ihn gewaltig in die Füße kipp. Er erschlug also erst diesen und rief darauf den Jolaus zu Hülfe, welcher denn einen nahestehenden Wald anzündete und mit den Bränden sogleich über den Hals der Hydra herfuhr, wenn ein Kopf angeschlagen war. Dadurch verhinderte er dann, dass keine andere an dessen Stelle wuchsen, und Herkules also das Ungeheuer endlich erlegen konnte. Den unsterblichen Kopf nahm er und vergrub ihn in die Erde, legete auch zu mehrer Sicherheit, einen gewaltigen Stein auf denselben. Dabei tunkete er seine Pfeile in ihr Blut, welche sodann ganz unheilbare Wunden machten.

Hederich, Benjamin: Gründliches Mythologisches Lexikon, Leipzig, Gleditsch 1770; Reprogr. Nachdruck, Darmstadt, Wissenschaftliche Buchgesellschaft 1996, S. 1454 f.

Widmung der Ausgabe von 1975

Zwei Menschen gewidmet, die meine Jahre bei IBM in ganz besonderer Weise bereichert haben:
Thomas J. Watson, Jr., dessen liebevolle Sorge um Menschen nach wie vor die Firma durchdringt, und Bob O. Evans, unter dessen kraftvoller Führung Arbeit zum Abenteuer wurde.

Widmung der Ausgabe von 1995

Für Nancy, Gottes Geschenk an mich.

Vorworte der Übersetzer

Als F. P. Brooks diese Serie von Essays schrieb – mittlerweile ist mehr als ein Jahrzehnt vergangen – war die Programmierung von Computern die Domäne eines kleinen Spezialistenzirkels. Auch wenn die Firma Intel bereits im Jahre 1972 den ersten Mikroprozessor auf den Mark gebracht hat: Die Vorstellung eines PC auf jedermanns Schreibtisch wäre selbst vom abgebrühtesten Verkaufsmanager als wilde Phantasterei abgetan worden.

Die Programmierung der ersten Mikrocomputer (des PET 2001 und des Apple II) *war* das Werk von „Garagenduos“, auf die der Autor mit dem nachsichtigen Lächeln eines Großrechnerspezialisten anspielt (Kapitel 1). Auch aus heutiger Sicht ist er dabei vollkommen im Recht.

Unnötig zu sagen, dass sich die Zeiten geändert haben. Für die Wochenmiete von 200 KByte zusätzlichem Speicher einer IBM 360 bekommt man einen kompletten Computer mit 4 MByte RAM, Programmierarbeit ohne symbolische Prozedurnamen ist auch auf einem Homecomputer unvorstellbar geworden – und Betriebssysteme von 200 KByte und mehr sind die Regel.

Die Zeit scheint nur in einem einzigen Punkt stehen geblieben zu sein: Probleme der Organisation großer Programmierprojekte haben sich nicht etwa geändert (geschweige denn: sind gelöst worden) – mit der Popularisierung integrierter Programmpakete, deren reiner Codeumfang bei mehreren hundert KByte liegt, haben sie sich lediglich auf den PC verlagert und dabei vervielfacht. Während die Vorschläge und Überlegungen Brooks' im Großcomputerbereich Allgemeingut sind, ist die Verspätung angekündigter Programme auf dem PC-Markt zur Regel geworden (für die im amerikanischen Sprachgebrauch denn auch der bezeichnende Satz „available real soon now“ entstand).

Einige Aussagen (PL/1 als Programmiersprache der Wahl) und Postulate (interaktive Terminals anstelle von Zeilendruckern, Symbole anstelle von Speicheradressen) sind nur noch vor historischem Hin-

tergrund von Interesse – der Kernpunkt des Buchs, die Organisation großer Programmierprojekte, ist dagegen aktueller denn je.

München, April 1987 Arne Schäpers/Armin Hopp

Ein Buch über Programmierung 16 Jahre später erneut auf dem Tisch zu haben, ist für sich schon absolut einmalig. Und wieder – diesmal nicht zehn, sondern „nur“ acht Jahre nach Erscheinen des Originals – zeigt sich der prophetische Charakter dieses Werks, sei es nun bei der komponentenbasierten Programmierung (Delphi und Visual Basic steckten 1995 in den Kinderschuhen, von C# erst gar nicht zu reden), der Forderung nach anwendungsübergreifenden Scriptsprachen (VBA) oder des zur Einheitlichkeit zwingenden, ins System integrierten Werkzeugkastens, wie ihn Microsoft zur Zeit mit der .NET-Initiative in nie gekannter Geschlossenheit vorexerziert. Eine weitere Ausgabe dieses Werks im Jahre 2015 vorauszusagen – dazu braucht es keine prophetischen Gaben.

München, Juli 2003 Arne Schäpers

Vorwort zum 20. Jubiläum der Originalausgabe

Zu meiner Überraschung und Freude ist *Vom Mythos des Mann-Monats* 20 Jahre lang populär geblieben; über 250 000 Exemplare allein der amerikanischen Originalausgabe wurden gedruckt. Ich werde oft gefragt, zu welchen der von mir im Jahre 1975 geäußerten Meinungen und Vorschläge ich nach wie vor stehe, was sich geändert hat – und was immer noch gilt. Von Zeit zu Zeit bin ich in Vorlesungen auf diese Fragen eingegangen, und der Wunsch, all dies einmal schriftlich niederzulegen, wurde lange gehegt.

Peter Gordon, mittlerweile assoziierter Hersteller bei Addison-Wesley, hat meine Arbeiten seit 1980 mit Geduld und hilfreicher Unterstützung begleitet. Von ihm stammt der Vorschlag einer Geburtstagsausgabe, für die wir beschlossen, das Original (mit Ausnahme trivialer Korrekturen) unverändert wiederzugeben und es um aktuelle Überlegungen zu ergänzen.

Kapitel 16 ist ein Abdruck von „No Silver Bullet: Essence and Accidents of Software Engineering“, eines für die IFIPS 1986 von mir publizierten Papiers, das meine Erfahrungen als Leiter einer Studie des Defense Science Board zu militärischer Software zusammenfasst. Die Koautoren dieser Studie und unser Generalsekretär, Robert L. Patrick, haben Unschätzbares dazu beigetragen, mich wieder auf Tuchfühlung mit großen Software-Projekten in der realen Welt zu bringen. Das Papier wurde 1987 von der Zeitschrift *IEEE Computer* abgedruckt und erreichte so eine breite Leserschaft.

„Keine Silberkugel“ ist ein provokativer Text, der voraussagt, dass es für die nächsten zehn Jahre keine einzige Technologie geben wird, die für sich allein genommen Verbesserungen der Produktivität bei Software um wenigstens eine Größenordnung erreicht. Diese Dekade steht ein Jahr vor ihrem Ablauf, und die Vorhersage scheint sich zu bestätigen. „Keine Silberkugel“ hat mehr und engagiertere Diskussionen in der Fachliteratur angeregt als *Vom Mythos des*

Mann-Monats. Aus diesem Grund geht Kapitel 17 auf die öffentlich geäußerte Kritik ein und stellt die 1986 geäußerten Thesen in einen aktuelleren Kontext.

Bei der Vorbereitung des Rückblicks auf *Vom Mythos des Mann-Monats* und seine Aktualisierung hat mich vor allem erstaunt, wie wenige der von mir aufgestellten Thesen durch spätere Studien der Disziplin und praktische Erfahrungen kritisch hinterfragt, tatsächlich bewiesen oder widerlegt worden sind. Für mich erwies es sich als nützlich, diese Vorschläge und Vorhersagen einmal als dürre Liste zusammenzufassen, bar jeglicher unterstützender Argumentation oder den dazugehörigen Daten. In der Hoffnung, dass diese nackten Statements zur Diskussion anregen, das Beibringen von Fakten provozieren, die sie stützen, ad absurdum führen, durch Aktuelleres ersetzen oder auch nur verfeinern, habe ich dieses Grundgerüst als Kapitel 18 in dieses Buch eingesetzt.

Kapitel 19 stellt die eigentliche Aktualisierung dar. Der Leser sollte sich bewusst sein, dass die dort geäußerten Meinungen nicht annähernd so gut durch eigene Erfahrungen mit täglicher Kärrnerarbeit abgesichert sind wie das von mir im originalen Buch Dargelegte: Ich habe an einer Universität gearbeitet, nicht in der Industrie, und kleine Projekte anstelle großer realisiert. Seit 1986 lehre ich Software Engineering, anstatt auf diesem Gebiet praktisch zu forschen. (Tatsächlich haben sich meine universitären Forschungsarbeiten stark auf virtuelle Umgebungen und ihren Einsatz konzentriert.)

Für diese Retrospektive musste ich deshalb die Hilfe von Freunden in Anspruch nehmen, die auch heute noch im Bereich des praktischen Software Engineering arbeiten. Für ihre wunderbare Bereitschaft, mich an ihren Ansichten teilhaben zu lassen, das Manuskript gründlich zu hinterfragen und mein eigenes Wissen auf den neuesten Stand zu bringen, schulde ich vor allem den folgenden Leuten Dank: Barry Boehm, Ken Brooks, Dick Case, James Coggins, Tom DeMarco, Jim McCarthy, David Parnas, Earl Wheeler und Edward Yourdon. Und natürlich Fay Ward, die sich souverän um die Produktion der neuen Kapitel gekümmert hat.

Ich danke Gordon Bell, Bruce Buchanan, Rick Hayes-Roth, meinen Kollegen von der Defense Science Board Task Force on Military Software und vor allem David Parnas für ihre Einsichten und Anregungen zu dem hier als Kapitel 16 abgedruckten Papier, und Rebekah Bierly für dessen technische Produktion. Die Analyse des Software-Problems in Termini von Essenz und Akzidenz wurde übrigens von Nancy Greenwood Brooks inspiriert, die dieselbe Perspektive in einem Papier über die Pädagogik der Suzuki-Violine verwendete.

Die hauseigene Politik von Addison-Wesley hat es mir nicht gestattet, im Vorwort der Ausgabe von 1975 den Verlagsmitarbeitern zu danken, die damals Schlüsselrollen innehatten. Speziell die Beiträge zweier Menschen sollten nicht unerwähnt bleiben: des damaligen Cheflektors Norman Stanton und des damaligen Art Directors Herbert Boes, der für den eleganten Stil verantwortlich zeichnet, über den eine Rezension schrieb: „große Abstände, sprechender Einsatz von Schriftarten und des Layouts“. Wichtiger war mir sein Vorschlag, jedes Kapitel mit einem eigenen Bild einzuleiten. (Ich hatte ursprünglich nur den Teersumpf und die Kathedrale vom Reims.) Die Suche nach weiteren Bildern hat mich ein zusätzliches Jahr Arbeit gekostet, für den Ratschlag werde ich aber ewig dankbar sein.

Soli Deo gloria – Allein Gott die Ehre.

Chapel Hill, N.C. F.P.B. Jr.
März 1995

Vorwort zur ersten Ausgabe

Das Management großer Computerprojekte zeigt mehr Gemeinsamkeiten mit dem anderer vergleichbar großer Unternehmen, als die meisten Programmierer glauben. Andererseits gibt es aber auch eine Anzahl fundamentaler Unterschiede, die die meisten Projektmanager kaum erwarten werden.

Inzwischen weiß man schon einiges über dieses Problemfeld. Es wurden verschiedene Symposien abgehalten, einige Sitzungen auf AFIPS-Konferenzen beschäftigten sich ebenfalls mit diesem Thema. Parallel dazu sind einige Bücher und Aufsätze in die Diskussion eingebracht worden. Allerdings kann man nicht sagen, dass sich schon zu diesem Zeitpunkt Umrisse abzeichnen, die die Abfassung eines Lehrbuchs rechtfertigen würden. Ich fühle mich dennoch berufen, mit diesem Buch meinen eigenen Standpunkt darzulegen.

Obwohl mein ursprüngliches Betätigungsfeld auf dem Gebiet der Computerprogrammierung lag, war ich während der Jahre 1956–1963, in denen das autonome Kontrollprogramm und der Hochsprachencompiler entwickelt wurden, hauptsächlich mit Hardware-Architektur beschäftigt. Als ich dann 1964 Manager des Operating System/360 wurde, fand ich eine im Zuge des Fortschritts stark veränderte Programmierwelt vor. Die Leitung der Entwicklung des OS/360 war für mich eine große Erfahrung, obwohl wir natürlich auch frustrierende Rückschläge erlitten. Das Team hat einschließlich meines Nachfolgers F. M. Trapnell allen Grund, auf das Geleistete stolz zu sein – sie haben ein ausgezeichnetes System auf die Beine gestellt. Es hat sowohl in seinem Design als auch in der Ausführung unbestreitbare Vorzüge und ist deshalb auf dem Markt sehr erfolgreich. Bestimmte Ideen (hervorzuheben sind hier besonders die Umleitungsmöglichkeiten der Ein- und Ausgabe sowie die Verwaltung externer Funktionsbibliotheken) stellten seinerzeit technische Neuerungen dar, die heute fast überall Verwendung finden. Das System

ist mittlerweile recht zuverlässig, hinlänglich effizient und sehr vielseitig.

Trotzdem kann das Ergebnis all der Anstrengungen nicht ganz überzeugen, wird doch jedem Benutzer des OS/360 schnell klar, um wie viel besser es sein könnte. Die Mängel in Ausführung und Design liegen hauptsächlich im Betriebssystem selbst, weniger in den Compilern. Die meisten Unzulänglichkeiten haben ihre Wurzeln in den Jahren 1964–1965 und fallen folglich unter meine Verantwortung. Überdies konnte der projektierte Zeitrahmen nicht eingehalten werden, wodurch das Produkt mit Verspätung auf den Markt kam. Es brauchte mehr Speicherplatz als eigentlich geplant, kostete ein Mehrfaches der veranschlagten Summe und gab in den ersten Versionen kein gutes Bild ab. 1965 verließ ich IBM, um nach Chapel Hill an die Universität von North Carolina zu gehen, wie es schon vereinbart worden war, als ich OS/360 übernahm. Dort begann ich, meine gesammelten Erfahrungen zu analysieren. Es ging mir darum, herauszufinden, welche administrativen und technischen Lehren man daraus ziehen könnte. Eine Klärung der bemerkenswerten Unterschiede, die bei der Entwicklung der System/360-Hardware im Gegensatz zu der der OS/360-Software zu Tage traten, lag mir dabei besonders am Herzen. Dieses Buch ist also eine reichlich verspätete Reaktion auf Tom Watsons Gretchenfrage, warum die Kunst des Programmierens denn eine so schwierige sei.

Bei der Suche nach einer Antwort habe ich aus langen Gesprächen mit R. P. Case, dem stellvertretenden Manager der Jahre 1964–1965, und F. M. Trapnell, dem Projektleiter von 1965–1968, viele Erkenntnisse gewonnen. Meine Schlussfolgerungen konnte ich schließlich mit den Leitern anderer ähnlich großer Programmierprojekte diskutieren. Unter ihnen waren F. J. Corbato vom M.I.T., John Harr und V. Vyssotsky von den Bell Telephone Laboratories, A. P. Ershov von den Computerlaboratorien der Sibirischen Abteilung der Akademie der Wissenschaften der UdSSR und A. M. Pietrasanta von IBM.

Die Sammlung meiner Erkenntnisse findet sich in den folgenden Essays, die für Berufsprogrammierer, Manager und besonders für Manager von Berufsprogrammierern bestimmt sind.

Obwohl dieses Buch aus voneinander unabhängigen Essays besteht, ist es doch um einen zentralen Gedanken herum aufgebaut, der sich besonders in den Kapiteln 2 bis 7 wiederfindet. Kurz gesagt bin ich zu der Überzeugung gelangt, dass sich bei großen Programmierprojekten wegen des Maßes an Arbeitsteilung andere Probleme aufwerfen als bei kleineren. Deswegen muss ein Hauptbestandteil der Programmierarbeit in der Bewahrung eines einheitlich durchgehenden Produktkonzepts selber liegen. Die genannten Kapitel untersuchen sowohl die Schwierigkeiten beim Versuch, eine solche Struktur durchgängig zu bewahren, als auch die dazu notwendige Methodik. Die folgenden Kapitel behandeln dann weitere Aspekte des Managements der Software-Entwicklung.

Die Literatur für dieses Problemfeld ist zwar nicht gerade dünn gesät, aber weit gestreut. Deswegen habe ich versucht, Hinweise zu geben, die bestimmte Punkte zusätzlich beleuchten und dem interessierten Leser weitere Arbeiten zum Thema zugänglich machen sollen. Viele meiner Freunde haben das Manuskript gelesen und konnten mir mit ihren Kommentaren oft weiterhelfen; wo mir diese wertvoll erschienen, aber den Fluss des Texts behinderten, habe ich sie in die Anmerkungen aufgenommen.

Da dieses Buch eine Sammlung von Essays enthält und keinen durchgehenden Text, wurden alle Hinweise und Anmerkungen ans Ende verbannt, wodurch der Leser gezwungen ist, sie zunächst außer Acht zu lassen.

Zu tiefstem Dank verpflichtet bin ich Miss Sara Elizabeth Moore, Mr. David Wagner und Mrs. Rebecca Burris für ihre Hilfe bei der Erstellung des Manuskripts und Professor Joseph C. Sloane für seine Ratschläge zur Illustration des Buchs.

Vorwort des Verlags

Im Vorwort zur 20-jährigen Jubiläumsausgabe schreibt der Übersetzer Arne Schäpers im Jahr 2003: »Eine weitere Ausgabe dieses Werks im Jahre 2015 vorauszusagen – dazu braucht es keine prophetischen Gaben.«

Nun, ganz so hat es nicht geklappt, denn mittlerweile schreiben wir das Jahr 2019. Aber das vorliegende E-Book, das 15 Jahre nach der ersten deutschen Audio-Veröffentlichung und ganze 44 Jahre nach der Erstausgabe von Brooks´ »Vom Mythos des Mann-Monats« nun im mitp-Verlag allen interessierten Lesern neu zur Verfügung gestellt wird, ist und bleibt einfach ein Klassiker.

Schwierigkeiten beim Managen komplexer Projekte hat es zu jeder Zeit gegeben und dies wird wohl auch immer so bleiben. Frederik P. Brooks analysiert und seziert in seinen Essays die Zutaten für ein erfolgreiches Projektmanagement mit sowohl historischem Bezug als auch zeitlosem Scharfsinn.

Ob Sie aus rein geschichtlicher Neugierde den Ausführungen von Brooks folgen wollen oder einige der auch in den heutigen Zeiten von Android und iOS gültigen Kernaussagen auf Ihr Software-Projekt übertragen möchten, Sie werden sicherlich einige wertvolle Ansätze in diesem Buch finden.

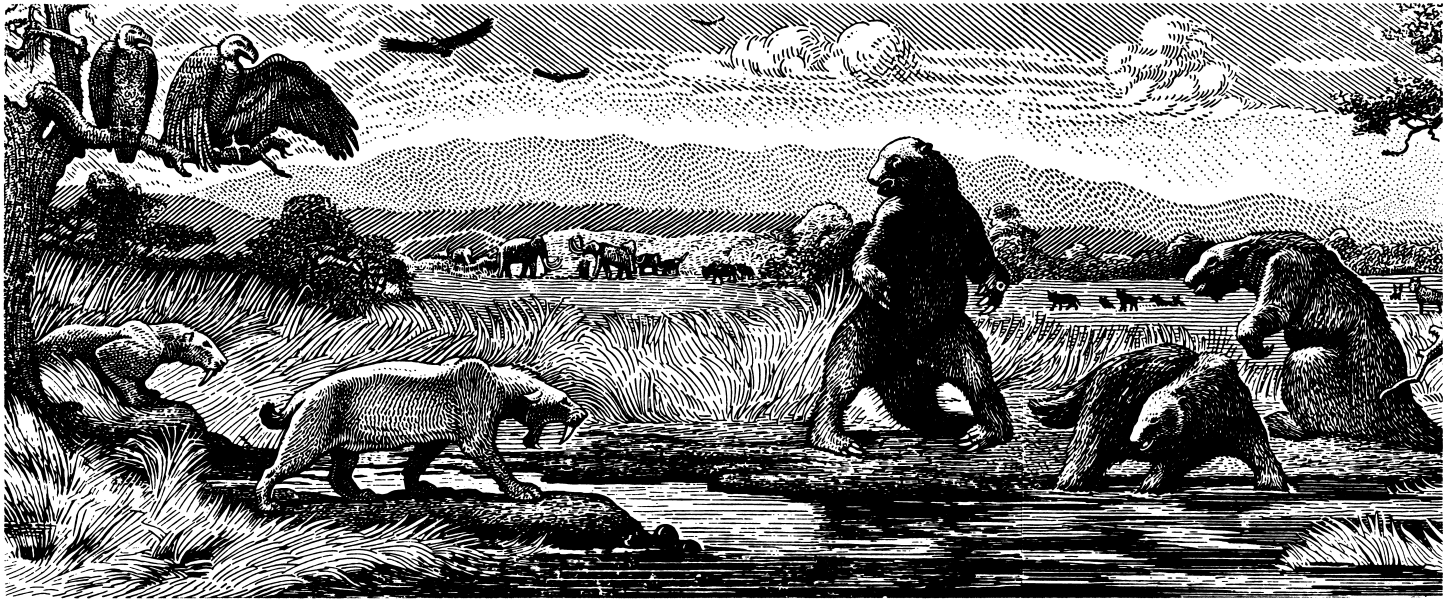
Wir wünschen Ihnen ein angenehmes Lese-Erlebnis!

Ihr mitp-Verlag

Inhalt

1

Der Teersumpf



1

Der Teersumpf

Een ship op het strand is een baken in zee.
(Ein gestrandetes Schiff ist eine Bake zur See hinaus.)

Niederländisches Sprichwort

C.R. Knight, Mural of La Brea Tar Pits
Naturhistorisches Museum von Los Angeles, Fotografische Abteilung

Kein Bild aus prähistorischer Zeit ist nur annähernd so beeindruckend wie das vom Todeskampf riesiger Bestien in den urweltlichen Teersümpfen. Vor dem geistigen Auge erscheinen Dinosaurier, Mammuts und Säbelzahntiger im Kampf gegen den Sog des Sumpfes. Je heftiger sie sich wehren, desto unnachgiebiger wird der Griff des Teers, und auch all ihre Kraft und Gewandtheit können sie nicht davor bewahren, letztendlich doch zu versinken.

Die Programmierung großer Systeme war während des letzten Jahrzehnts ein solcher Teersumpf, in dem viele große und starke Bestien wild um sich geschlagen haben. Die meisten haben funktionierende Systeme hervorgebracht – wenige haben Zielsetzungen, Zeitvorgaben und Kostenrahmen einhalten können. Ob umfangreich oder schmalbrüstig, in großem oder kleinem Rahmen, ein Team nach dem anderen hat sich in den Sumpf ziehen lassen. Es scheint unmöglich, für dieses Phänomen einen besonderen Grund zu isolieren. Genauso wie die versinkende Bestie, die jede einzelne ihrer Pfoten wieder aus dem Sumpf ziehen kann, schließlich aber doch untergehen muss, ist auch das Programmierteam dem Sog hilflos ausgeliefert. Die Anhäufung gleichzeitig auftretender und sich gegenseitig beeinflussender Faktoren schränkt die Bewegungsfähigkeit immer weiter ein. Jedermann zeigt sich überrascht von der Unzugänglichkeit des Problems, dessen Natur offensichtlich schwer zu ergründen ist. Aber wenn wir es lösen wollen, müssen wir versuchen, es verstehen zu lernen.

So wollen wir damit beginnen, die Kunst (das Handwerk?) des Programmierens von Systemen zu beschreiben, mitsamt den ihr innewohnenden Freuden und Leiden.

Das Produkt „Programmiersystem“

Dann und wann liest man in den Zeitungen Berichte über zwei Programmierer, die in einer umfunktionierten Garage ein bedeutendes Programm entwickelt haben, das die Anstrengungen ganzer Teams in den Schatten stellt. Und jeder Programmierer ist nur allzu gern bereit, derlei Märchen Glauben zu schenken, ist er doch davon überzeugt, er könne *jedes* Programm wesentlich effizienter und schneller

schreiben als die Teams in der Industrie, deren Durchschnittsleistung bekanntlich bei 1000 Befehlen pro Programmierer und Jahr liegt.

Warum werden dann nicht einfach alle in der Industrie beschäftigten Programmierteams durch solche Garagenduos ersetzt? Dazu muss man genauer betrachten, *was* eigentlich produziert wird.

In Bild 1.1 findet sich oben links das *Programm*. In sich selbst ist es vollständig und kann von seinem Autor auf dem System benutzt werden, auf dem es geschrieben wurde. Das ist es, was gewöhnlich in Garagen entwickelt wird, und daran misst der einzelne Programmierer seine eigene Produktivität.

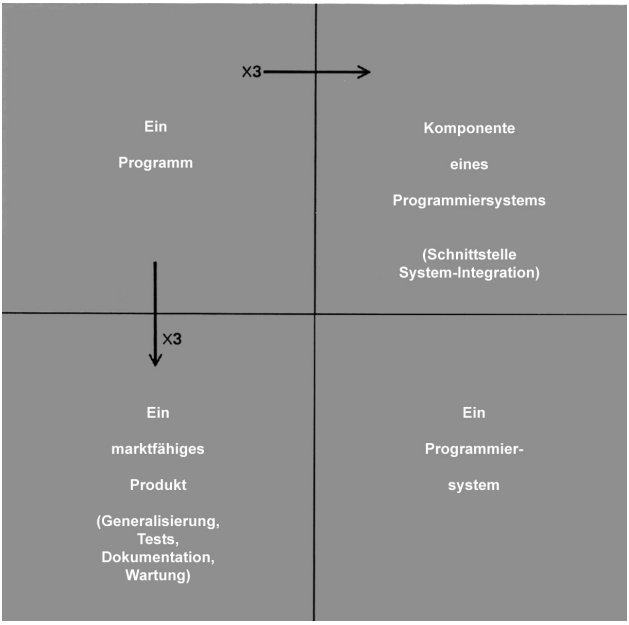


Bild 1.1 Die Evolution eines Programmierprodukts

Es gibt zwei Wege, um ein Programm in etwas Nützlicheres umzuwandeln – beide sind allerdings mit erheblich höheren Kosten verbunden. Diese beiden Möglichkeiten werden im Diagramm durch zwei Linien dargestellt. Unterhalb der horizontalen Linie sehen wir, wie aus einem Programm ein *marktfähiges Produkt* wird. Ein sol-

ches Programm kann von jedermann benutzt, abgefragt, instandgesetzt und erweitert werden. Man kann es in mehreren Systemkonfigurationen und für viele verschiedene Arten von Datensätzen benutzen. Hat man sich ein solches generell verwendbares Produkt zum Ziel gesetzt, muss auch das Programm selber in einer generalisierten Form gehalten sein.

Besonders Umfang und Form der Eingaben verlangen nach weitestgehender Generalisierung, so weit, wie es der zu Grunde liegende Algorithmus vernünftigerweise zulässt. Schließlich muss das Programm einer gründlichen Prüfung unterzogen werden, bis seine Zuverlässigkeit gewährleistet werden kann. Dies bedeutet, dass eine große Anzahl von Prüfverfahren zur Untersuchung des Umfangs der Eingaben und ihrer Grenzen entwickelt, angewandt und festgehalten werden muss. Letztendlich muss das zum marktfähigen Produkt avancierte Programm noch von einer gründlichen Dokumentation begleitet werden, damit ein jeder es verwenden, reparieren und erweitern kann. Als Faustregel gehe ich davon aus, dass ein solches Produkt mindestens das Dreifache eines fehlerbereinigten Programms mit vergleichbarer Arbeitsweise kostet.

Mit dem Sprung über die vertikale Linie wird das Programm zur *Komponente eines Programmiersystems*. Dabei handelt es sich um eine Zusammenstellung interagierender Programme, deren jeweilige Funktionen koordiniert und in eine Struktur eingebracht werden. Aus dieser Montage entsteht dann eine Arbeitseinheit zur Lösung umfangreicher Aufgaben. Damit ein Programm zur Komponente eines Programmiersystems werden kann, muss es so geschrieben sein, dass seine Ein- und Ausgaben syntaktisch und semantisch präzise festgelegten Definitionen entsprechen. Darüber hinaus soll das Programm so angelegt sein, dass es die verfügbaren Ressourcen nur in einem genau festgelegten Ausmaß in Anspruch nimmt – sowohl was Rechenzeit und Speicherplatz als auch die Peripheriegeräte betrifft. Schließlich muss das Programm mit anderen Systemkomponenten getestet werden, und zwar in allen zu erwartenden Kombinationen. Ein solcher Prüfvorgang ist aufwendig – einfach schon deshalb, weil die Zahl der Testfälle mit der der möglichen Kombinationen steigt. Er ist zeitaufwendig, da das Zusammenspiel schon feh-

lerbereinigter Komponenten völlig unerwartet zu neuen Fehlfunktionen führen kann. Aus all diesen Gründen kostet die Komponente eines Programmiersystems mindestens das Dreifache eines einfachen Programms gleicher Funktion. Wenn das System aus vielen Komponenten besteht, können die Kosten noch höher liegen.

Unten rechts in Bild 1.1 finden wir das Produkt *Programmiersystem*, das sich in allen oben erwähnten Punkten von einem einfachen Programm unterscheidet. Es kostet das Neunfache. Aber es ist das einzige wirklich brauchbare Produkt, das angestrebte Ziel fast aller Anstrengungen auf dem Gebiet der Systemprogrammierung.

Die Freuden des Handwerks

Warum macht Programmieren Spaß? Welche Freuden kann ein Jünger unseres Handwerks als Lohn erwarten?

Zuerst ist da die schiere Freude an der Herstellung von Dingen an sich. Wie ein Kind sich an seinen Sandkuchen erfreut, so delektieren Erwachsene sich am Zusammenbau aller möglichen Dinge, besonders an den von ihnen selbst erdachten. Ich glaube, diese Freude muss ein Abbild der Freude Gottes an der Schöpfung der Dinge sein, einer Freude, die sich in der Einzigartigkeit und Schönheit eines jeden Blattes und einer jeden Schneeflocke offenbart.

Zum zweiten verschafft es ein großes Maß an Befriedigung, für andere Menschen nutzbringende Gegenstände zu schaffen. Tief in uns wollen wir unsere Arbeit von anderen genutzt und als hilfreich empfunden sehen. In dieser Hinsicht besteht kein wesentlicher Unterschied zwischen Programmiersystemen und dem ersten selbst gemachten tönernen Aschenbecher eines Kindes für den Wohnzimmertisch der Eltern.

Zum dritten ist da die Faszination an der Gestaltung komplexer, puzzlegleicher Objekte aus ineinander greifenden, bewegten Teilen und an der Betrachtung der Funktion ausgetüftelter Kreisläufe, wobei Konsequenzen von Prinzipien ausgespielt werden, die bereits von Anfang an eingebaut worden waren. Der programmierte Computer verfügt über die ganze Faszination eines Flipperautomaten

oder des Mechanismus einer Musikbox – nur, dass er das Nonplusultra der technischen Entwicklung darstellt.

Der vierte Punkt ist die Freude am ständigen Lernen, die in der nonrepetitiven Natur der Aufgabe begründet liegt. Auf die eine oder andere Art ist das Problem immer neu, und wer es angeht, lernt etwas: manchmal praxisbezogen, manchmal theoretisch, und manchmal etwas von beidem.

Zu guter Letzt bereitet es Freude, in einem so leicht formbaren Medium zu arbeiten. Der Programmierer, wie auch der Dichter, arbeitet kaum im Bereich des stofflich Fassbaren. Aus dem Nichts baut er sich Luftschlösser, ist kreativ aus dem Einsatz, seiner Vorstellungskraft heraus. Wenige Medien gewähren dem kreativen Menschen ein solches Maß an Flexibilität, sind so leicht zu be- und überarbeiten, erlauben so willig die Realisierung großer konzeptueller Strukturen. (Wie wir sehen werden, hat aber gerade auch diese Formbarkeit ihre eigenen problematischen Seiten.)

Allerdings ist das zum Programm gewordene Gedankengebäude, anders als die Worte des Dichters, wirklich, in dem Sinne, dass sich etwas bewegt und arbeitet. Es werden sichtbare Ausgaben erzeugt, die sich von dem Gedankengebäude selber unterscheiden. Ein Programm kann Ergebnisse ausdrucken lassen, Grafiken anfertigen, Geräusche erzeugen, Maschinenteile in Bewegung setzen. Die Zauberkunst aus Mythos und Legende ist zu unserer Zeit Wirklichkeit geworden. Man gibt über eine Tastatur die richtige Zauberformel ein und ein Bildschirm wird lebendig, Dinge erscheinen, die es nie vorher gegeben hat und geben konnte.

Schließlich und endlich bereitet das Programmieren Freude, weil es unsere innersten kreativen Bedürfnisse befriedigt und Sinne anspricht, die wir mit allen Menschen gemeinsam haben.

Die Schattenseiten des Handwerks

Nicht alles ist die reine Freude – aber das Wissen um die innewohnenden Leiden macht es leichter, sie zu ertragen, wo immer sie auftauchen mögen.

Zuerst einmal muss man perfekt arbeiten. Auch in dieser Hinsicht erinnert der Computer an die Zauberkunst aus alten Legenden: Wenn nur ein einziges Zeichen der Zauberformel nicht genau die richtige Form hat, funktioniert der ganze Zauber nicht. Der Mensch ist nicht dazu geschaffen, perfekt zu sein, und nur wenige Bereiche menschlicher Aktivitäten verlangen dies. Die Gewöhnung an die Notwendigkeit zum perfekten Handeln stellt, so glaube ich, die größte Schwierigkeit beim Erlernen des Programmierens dar.[1]

Dazu kommt, dass es andere Leute sind, die einem die Zielsetzungen vorgeben, Ressourcen zur Verfügung stellen und Informationen liefern. Selten kontrolliert man selber das Umfeld seiner Arbeit, ganz zu schweigen von ihrem Ziel. Um es in die Terminologie des Managements zu fassen: Die Befugnisse des Programmierers sind im Verhältnis zu der ihm auferlegten Verantwortung nicht ausreichend. Es scheint allerdings in allen Bereichen so zu sein, dass die Jobs, in denen die Aufgaben tatsächlich angegangen werden, nie die ihrer Verantwortung entsprechenden formalen Befugnisse mit sich bringen. Wirkliche Befugnisse aber (im Gegensatz zu den formalen) werden in der Praxis durch die Kraft des Geleisteten erworben.

Die Abhängigkeit von anderen hat einen speziellen Aspekt, der für den Systemprogrammierer besonders schmerzlich ist. Er ist abhängig von den Programmen anderer Leute, die oft schlecht aufgebaut, unzulänglich implementiert, spärlich dokumentiert sind und dazu häufig unvollständig (d.h. ohne Quellcode und Prüfroutinen) geliefert werden. Deshalb muss er sich stundenlang mit dem Studium und der Instandsetzung von Dingen herumschlagen, die in einer idealen Welt vollständig, verfügbar und benutzbar wären.

Und damit nicht genug – die Freude des Programmierers an seiner Tätigkeit wird noch durch andere Widrigkeiten überschattet: Der Entwurf großer Konzepte macht sicherlich viel Spaß, die Suche nach verborgenen, kleinen Fehlern dagegen bedeutet einfach Arbeit. Mit jeder kreativen Aktivität kommen eintönige Stunden langweiliger und mühseliger Arbeit – das Programmieren macht hier keine Ausnahme.

Dazu kommt zwangsläufig die Erkenntnis, dass die Fehlerbeseitigung bestenfalls einen linearen Verlauf nimmt, obwohl man eigentlich gegen das Ende der Arbeit hin eine Art quadratischer Steigerung erwarten sollte.

Die letzten Leidensmomente, und das bringt dann das Fass manchmal wirklich zum Überlaufen, erwarten den Programmierer, wenn das Produkt, an dem er so lange gearbeitet hat, zum Zeitpunkt (oder sogar vor) seiner Fertigstellung bereits obsolet zu sein scheint. Kollegen und Konkurrenten verfolgen längst neue und bessere Ideen. Schon ist die Ablösung des eigenen Abkömmlings nicht nur im Gespräch, sondern geplant.

Das scheint auf den ersten Blick immer schlimmer zu sein, als es dann wirklich ist. Das neue und bessere Produkt ist gewöhnlich nicht verfügbar, wenn man sein eigenes fertig stellt; es wird nur darüber geredet. Es wird ebenso Monate der Entwicklung brauchen. Gegen den Papiertiger sieht der echte immer etwas schlecht aus, es sei denn, wirkliche Brauchbarkeit ist gefragt.

Natürlich entwickelt sich die technologische Basis, auf die man sich stützt, immer weiter. Sobald man eine Entwicklung einfriert, ist sie veraltet, was ihr Konzept betrifft. Aber die Implementierung wirklicher Produkte verlangt nach Abstimmung und Abwägung. Die Veraltung einer Implementierung muss an schon existierenden gemessen werden, nicht an unrealisierten Konzepten. Die Herausforderung und die Aufgabe besteht darin, mit verfügbaren Ressourcen im Rahmen wirklicher Zeitpläne wirkliche Lösungen für wirkliche Probleme zu finden.

Das also ist Programmieren, sowohl ein Teersumpf, in dem viele Bemühungen untergegangen sind, als auch eine kreative Tätigkeit mit den ihr eigenen Freuden und Leiden. Für viele überwiegen die Freuden bei weitem, und für sie soll mit dem Rest dieses Buches der Versuch unternommen werden, einige Wege durch den Teer aufzuzeigen.

2

Der Mythos Mann-Monat

Restaurant Antoine

Fondé En 1840

AVIS AU PUBLIC

Faire de la bonne cuisine demande un certain temps. Si on vous fait attendre, c'est pour mieux vous servir, et vous plaire.

ENTREES (SUITE)

Côtelettes d'agneau grillées 2.50
Côtelettes d'agneau aux champignons frais 2.75
Filet de boeuf aux champignons frais 4.75
Ris de veau à la financière 2.00
Filet de boeuf nature 3.75
Tournedos Médicis 3.25
Pigeonneaux sauce paradis 3.50
Tournedos sauce béarnaise 3.25
Entrecôte minute 2.75
Filet de boeuf béarnaise 4.00
Tripes à la mode de Caen (commander d'avance) 2.00
Entrecôte marchand de vin 4.00
Côtelettes d'agneau maison d'or 2.75
Côtelettes d'agneau à la parisienne 2.75
Fois de volaille à la brochette 1.50
Tournedos nature 2.75
Filet de boeuf à la hawaïenne 4.00
Tournedos à la hawaïenne 3.25
Tournedos marchand de vin 3.25
Pigeonneaux grillés 3.00
Entrecôte nature 3.75
Châteaubriand (30 minutes) 7.00

LÉGUMES

Epinards sauce crême .60
Broccoli sauce hollandaise .80
Pommes de terre au gratin .60
Haricots verts au berre .60
Chou-fleur au gratin .60
Asperges fraîches au beurre .90
Carottes à la crème .60
Pommes de terre soufflées .60
Petits pois à la française .75

SALADES

Salade Antoine .60
Salade Mirabeau .75
Salade laitue au roquefort .80
Salade de laitue aux tomates .60
Salade de légumes .60
Salade d'anchois 1.00
Fonds d'artichauts Bayard .90
Salade de laitue aux oeufs .60
Tomate frappée à la Jules César .60
Salade de coeur de palmier 1.00
Salade aux pointes d'asperges .60
Avocat à la vinaigrette .60

DESSERTS

Gâteau moka .50
Méringue glacée .60
Crêpes Suzette 1.25
Glace sauce chocolat .60
Fruits de saison à l'eau-de-vie .75
Omelette soufflée à la Jules César (2) 2.00
Omelette Alaska Antoine (2) 2.50
Cerises jubilé 1.25
Crêpes à la gelée .80
Crêpes nature .70
Omelette au rhum 1.10
Glace à la vanille .50
Fraises au kirsch .90
Pêche Melba .60

FROMAGES

Roquefort .50
Camembert .50
Liederkranz .50
Gruyère .50
Fromage à la crême Philadelphie .50

CAFÉ ET THÉ

Café .20
Café brulôt diabolique 1.00
Café au lait .20
Thé glacé .20
Thé .20
Demi-tasse .15

EAUX MINERALES—BIERE—CIGARES—CIGARETTES

White Rock
Vichy
Cliquot Club
Bière locale
Canada Dry
Cigares
Cigarettes

Roy L. Alciatore, Propriétaire

713-717 Rue St. Louis — Nouvelle Orléans, Louisiane

2

Der Mythos Mann-Monat

Die gute Küche braucht ihre Zeit. Sollten Sie einmal warten müssen, dann nur, weil wir Sie besser bedienen und zufrieden stellen wollen.

Speisekarte des Restaurant Antoine, New Orleans