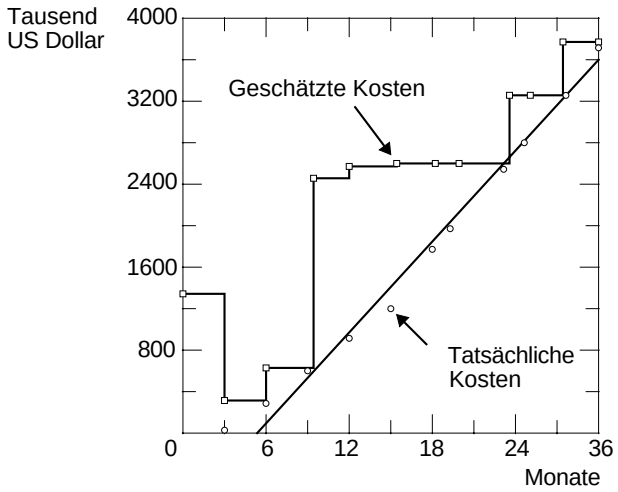
5. Software-Aufwandschätzung

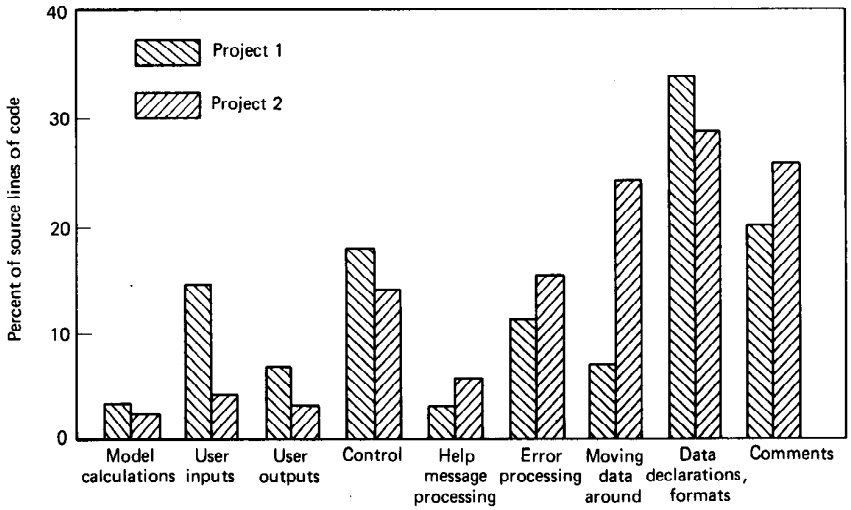
5.1 Allgemeines

Bild 5.1 zeigt, wie man es nicht machen sollte: Eine erste Aufwandschätzung wird zunächst nach unten korrigiert, um den Auftrag zu bekommen. Anschließend wird die Schätzung jedes Mal dann nach oben angepasst, wenn sie von den tatsächlichen Kosten eingeholt worden ist. Die Schätzung nach sieben Monaten ist offenbar die erste, die wirklich seriös durchgeführt wurde. Sie zeigt, dass schon die erste Schätzung viel zu optimistisch war, ganz zu schweigen von der politischen „Schätzung“ nach drei Monaten.



nach Boehm (1981)

BILD 5.1. Unrühmliches Beispiel für die Kostenentwicklung eines Projekts



Quelle: Boehm (1981)

BILD 5.2. Woraus ein Programm besteht

Es gibt eine Reihe von Gründen für solche krassen Fehleinschätzungen, zum Beispiel:

- Nur ein kleiner Teil einer Software trägt die eigentliche Funktionalität. Der Rest (und das wird oft übersehen) geht in Verwaltung, Fehlerbehandlung, Benutzerschnittstelle, etc. (Bild 5.2).

- Software-Erstellung ist weitgehend Kopfarbeit und daher stark von der Leistungsfähigkeit dieser Köpfe abhängig. Sie kann um mehr als eine Größenordnung schwanken.
- Erfahrungen aus kleinen Projekten werden linear extrapoliert (Bild 1.7).
- Es wird nicht nach sachlichen, sondern nach politischen Kriterien geschätzt.

5.2 Empirische Schätzverfahren

5.2.1 Expertenschätzung

Bei der so genannten Expertenschätzung schätzen Personen auf Grund ihrer persönlichen Erfahrungen und durch Vergleich mit bisher abgewickelten Projekten die zu erwartenden Aufwendungen. Die Bezeichnung „Expertenschätzung“ ist insofern etwas irreführend als sie die gesamte Spannbreite von seriösen Schätzungen durch wirkliche Experten bis zur groben Peilung über den Daumen durch beliebig unerfahrene Leute umfasst. Die Güte der Schätzung hängt entscheidend ab von der Erfahrung der Schätzenden, der Verfügbarkeit von Daten über abgewickelte Projekte und der Verwendung geeigneter Techniken für die Durchführung der Schätzung (siehe 5.2.3 unten). Krasse Fehler ergeben sich oft dann, wenn Erfahrungen fehlen oder Erfahrungen mit kleinen Projekten auf große Projekte extrapoliert werden (Bild 1.7). Expertenschätzung ist das in der Praxis am häufigsten verwendete Schätzverfahren (Moløkken und Jørgensen 2003).

FESTSTELLUNG 5.1. Expertenschätzung ist ein einfaches und billiges Schätzverfahren, das dann gut funktioniert, wenn die Schätzenden genügend Erfahrung haben und über eine Datenbasis mit den Aufwendungen für vergleichbare, in der Vergangenheit abgewickelte Projekte verfügen. Andernfalls können die Prognosen sehr ungenau sein.

5.2.2 Delphi-Methode

Die Delphi-Methode ist eine objektivierte Expertenschätzung: Mehrere Personen geben unabhängig voneinander eine begründete Schätzung ab. In einer nächsten Runde erhält jede(r) Beteiligte eine Zusammenfassung der ersten Schätzungen. Alle erstellen daraufhin eine neue Schätzung, wobei Abweichungen vom Mittelwert der vorhergehenden Runde zu begründen sind. Dies geht über mehrere Runden, bis alle Schätzungen einigermaßen beieinander liegen.

FESTSTELLUNG 5.2. Die Delphi-Methode liefert zuverlässigere Schätzungen als die Expertenbeurteilung, weil sie Ausreißer eliminiert. Als Nachteil muss ein erheblich höherer Schätzaufwand in Kauf genommen werden.

5.2.3 Techniken für die Expertenschätzung

Die beiden wichtigsten Techniken der Expertenschätzung sind die direkte Analogieschätzung und die Schätzung durch Zerlegung.

Bei der *direkten Analogieschätzung* wird ein abgewickeltes Projekt herangezogen, dessen Aufwanddaten bekannt sind und das dem zu schätzenden Projekt möglichst ähnlich ist. Durch Analyse der Unterschiede zwischen dem vorliegenden und dem Referenzprojekt werden Mehr- oder Minderaufwendungen gegenüber dem Referenzprojekt geschätzt und aufsummiert.

Bei der *Schätzung durch Zerlegung* wird das zu schätzende Projekt zerlegt, beispielsweise in Teilaufgaben (Komponenten, Systemfunktionen, Anwendungsfälle o.ä.) oder in Teilschritte (entsprechend den Teillieferungen eines Wachstumsmodells, vgl. Kapitel 3.2.3). Die resultierenden Teile werden einzeln geschätzt und die Aufwendungen aufsummiert. Da das Verhältnis von Produktgröße und Aufwand nicht linear ist (vgl. Bild 1.7), wird der Aufwand bei der Schätzung

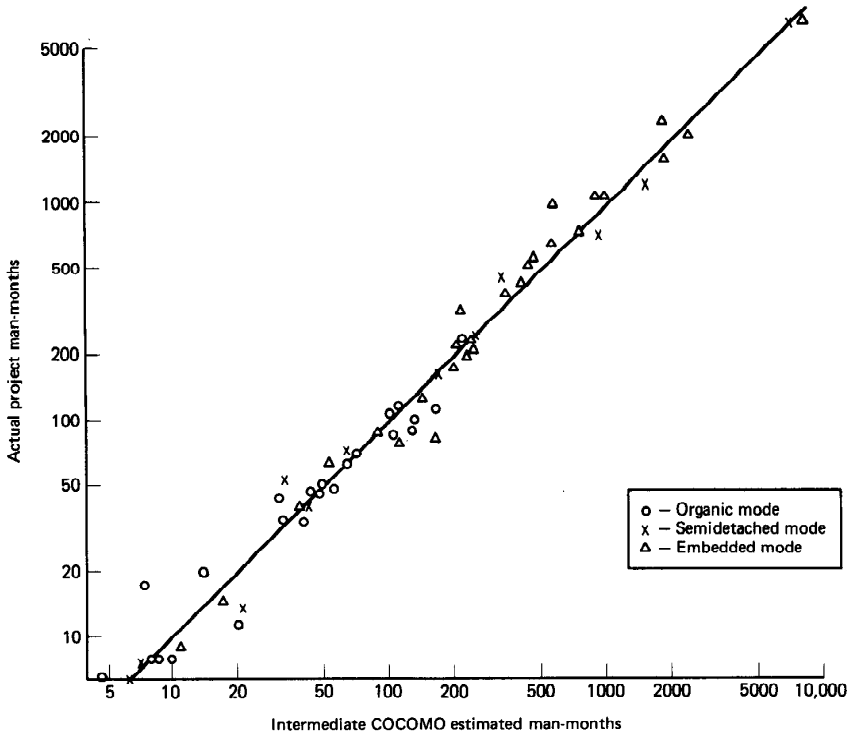
durch Zerlegung systematisch unterschätzt, wenn die Summe der Einzelaufwendungen nicht nachkorrigiert wird, indem der zusätzliche Aufwand für Kommunikation, Koordination, Problemkomplexität, Projektführung, etc. geschätzt und hinzugerechnet wird. Die Schätzung durch Zerlegung hat den Vorteil, dass aus den geschätzten Aufwendungen für Teilaufgaben Meilensteine für die Projektabwicklung abgeleitet werden können.

Da der tatsächliche Aufwand in Personentagen stark von der Produktivität der Projektbeteiligten abhängt (Boehm (1981) hat Produktivitätsunterschiede von bis zu 1:4 bei Entwicklergruppen beobachtet; die individuellen Unterschiede sind noch größer, vgl. Kapitel 12.3), muss die Fähigkeit des vorgesehenen Projektteams in die Schätzung einbezogen werden.

Ferner ist zu beachten, dass die geschätzten Größen faktisch nicht feste Zahlen, sondern statistische Verteilungen sind, die wir durch ihre Mittelwerte abschätzen. Dabei muss man sich stets vor Augen halten, dass selbst bei exakter Schätzung in 50 Prozent aller Fälle der tatsächliche Aufwand größer oder bestenfalls gleich dem geschätzten Mittelwert ist! Nimmt man noch die Tatsache dazu, dass Menschen bei Schätzungen zum Optimismus neigen, so ist es nicht mehr groß verwunderlich, dass die Mehrzahl aller Projekte mehr Aufwand beansprucht als ursprünglich geschätzt.

5.3 Algorithmische Schätzverfahren

Algorithmische Methoden bestehen aus einem oder mehreren Algorithmen zur Berechnung einer Kosten- bzw. Durchlaufzeit-Funktion aus einer Reihe von Variablen. Bei hinreichend genauen Eingaben und stabilem Umfeld ergeben sich erstaunlich präzise Prognosen (Bild 5.3).



Quelle: Boehm (1981)

BILD 5.3. Genauigkeit von COCOMO-Schätzungen

Die Genauigkeit aller algorithmischen Methoden hängt jedoch entscheidend von zwei Dingen ab:

- die *Eingangsgrößen* der Kostenfunktion (z.B. Anzahl Instruktionen) müssen einigermaßen *zutreffend geschätzt* werden

- das Modell muss *kalibriert* werden, d.h. der Wert der einzelnen Kostenattribute muss an die jeweilige Entwicklungsumgebung angepasst werden. Eine solche Kalibrierung ist nur möglich, wenn genügend Messwerte von durchgeführten Projekten vorliegen.

FESTSTELLUNG 5.3. Algorithmische Methoden liefern die besten Schätzungen. Sie können aber nur nach entsprechenden Vorarbeiten (Kalibrierung) überhaupt eingesetzt werden. Außerdem sind sie stark abhängig von der Genauigkeit, mit der die Eingangsgrößen bestimmt werden können (garbage-in-garbage-out-Problem).

Die zwei wichtigsten algorithmischen Schätzverfahren, COCOMO und Function Point, werden in den folgenden Unterkapiteln vorgestellt.

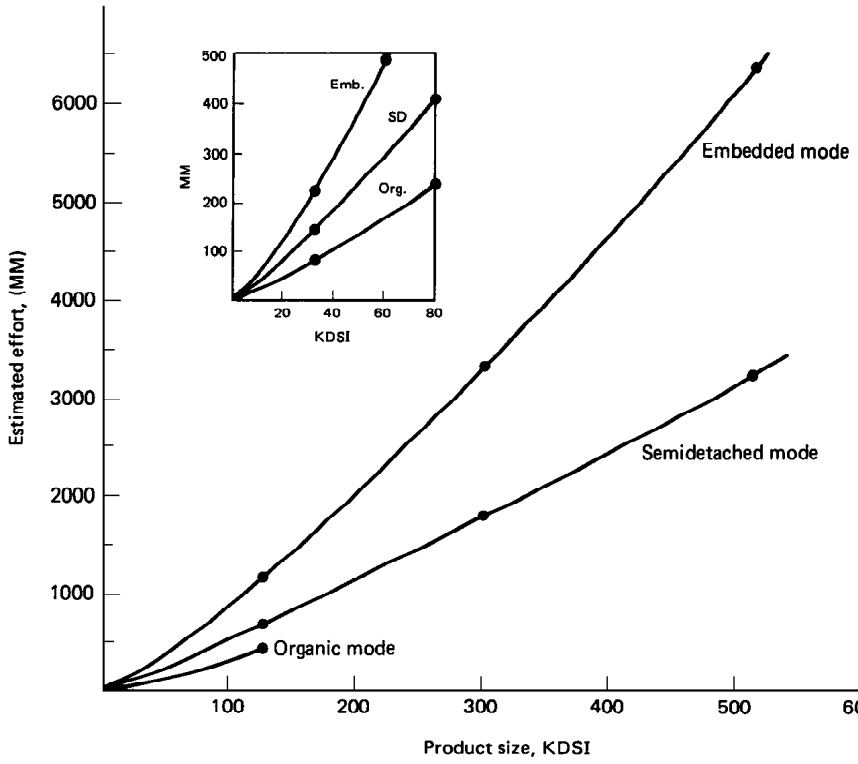
5.3.1 COCOMO

COCOMO (Constructive Cost Model) ist das Kostenschätzverfahren von Boehm (1981). Es geht aus von einer Schätzung der Produktgröße in KDSI (Kilo lines of delivered source instructions). Aus diesem Grundwert und einer Reihe von Kosten-Multiplikatoren werden Kosten und Durchlaufzeit berechnet. Die Grundgleichungen lauten:

(5.1) $MM = 2.4\ KDSI^{1.05}$ (MM = man month)

(5.2) $TDEV = 2.5\ MM^{0.38}$ (TDEV = time to develop)

Diese Gleichungen gelten für einfache *Anwendungsprogramme* (organic mode). Für *Programmsysteme* (semidetached mode), bei denen ein erheblicher Anteil an Interaktion mit Betriebssystem, Gerätetreibern, etc. hinzukommt und für *eingebettete Systeme* (embedded mode), deren Erstellung nochmals erheblich schwieriger ist, gelten andere Faktoren. Bild 5.4 zeigt eine graphische Veranschaulichung der Kosten über der Produktgröße.



Quelle: Boehm (1981)

BILD 5.4. Aufwandsschätzung nach COCOMO

COCOMO-Grundgleichungen für Programmsysteme

(5.3) $MM = 3.0\ KDSI^{1.12}$

(5.4) $TDEV = 2.5\ MM^{0.35}$

COCOMO-Grundgleichungen für eingebettete Systeme

(5.5) $MM = 3.6\ KDSI^{1.2}$

(5.6) $TDEV = 2.5\ MM^{0.32}$

Die mit den Grundgleichungen ermittelten Nominalwerte können nun noch erheblich genauer gemacht werden, indem man sie mit einer Reihe von Kostenfaktoren (cost drivers) multipliziert. Tabelle 5.1 zeigt die COCOMO-Kostenfaktoren. Die aktuellen Werte aller Kostenfaktoren für ein Projekt werden unabhängig voneinander geschätzt und anschließend alle miteinander multipliziert. Der Nominalwert für den Projektaufwand in Personenmonaten wird dann mit diesem Produktfaktor multipliziert.

(5.7) MM_{Korr} = Produkt der Kostenfaktoren · $MM_{nominal}$

TABELLE 5.1. COCOMO-Kostenfaktoren

	Ratings					
Cost Drivers	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY Required software reliability	.75	.88	1.00	1.15	1.40	
DATA Data base size		.94	1.00	1.08	1.16	
CPLX Product complexity	.70	.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME Execution time constraint			1.00	1.11	1.30	1.66
STOR Main storage constraint			1.00	1.06	1.21	1.56
VIRT Virtual machine volatility[a]		.87	1.00	1.15	1.30	
TURN Computer turnaround time		.87	1.00	1.07	1.15	
Personnel Attributes						
ACAP Analyst capability	1.46	1.19	1.00	.86	.71	
AEXP Applications experience	1.29	1.13	1.00	.91	.82	
PCAP Programmer capability	1.42	1.17	1.00	.86	.70	
VEXP Virtual machine experience[a]	1.21	1.10	1.00	.90		
LEXP Programming language experience	1.14	1.07	1.00	.95		
Project Attributes						
MODP Use of modern programming practices	1.24	1.10	1.00	.91	.82	
TOOL Use of software tools	1.24	1.10	1.00	.91	.83	
SCED Required development schedule	1.23	1.08	1.00	1.04	1.10	

Quelle: Boehm (1981)

Eine ausführliche Beschreibung von COCOMO, inklusive der Definition, was bei ihm ein Personenmonat und eine Zeile gelieferter Code ist, findet sich in Boehm (1981).

5.3.2 COCOMO II

Ende der 90er Jahre wurde COCOMO überarbeitet und verfeinert. Das Resultat ist unter dem Namen COCOMO II in Boehm et al. (2000) publiziert. An Stelle von drei Klassen von Grundgleichungen treten zwei universelle Grundgleichungen, die über einen *Wachstumsfaktor* (effort growth factor) B an unterschiedliche Gegebenheiten angepasst werden.

(5.8) $\text{Aufwand} = 2{,}45 \cdot \text{KSLOC}^{B} \cdot \prod_{i=1}^{17} EM_i$

(5.9) $\text{Durchlaufzeit} = 2{,}66 \cdot \text{Aufwand}^{0{,}33 + 0{,}2\,(B - 1{,}01)}$

KSLOC steht für “Kilo source lines of code”. Der Wachstumsfaktor B berechnet sich aus

(5.10) $B = 1{,}01 + 0{,}01 \cdot \sum_{i=1}^{5} SF_i$

Die EM_i sind 17 *Kostenfaktoren* (effort multipliers; siehe Tabelle 5.2), welche eine überarbeitete Version der COCOMO Kostenfaktoren (vgl. Tabelle 5.1) darstellen.

In die Berechnung des Wachstumsfaktors B gehen fünf *Skalierungsfaktoren* SF_i (scaling factors; Tabelle 5.3) ein. *Neuartigkeit* (precedentedness) ist ein Maß sowohl für die Neuartigkeit des zu lösenden Problems als auch die Vertrautheit der beteiligten Entwickler und Anwender mit dem Problem. *Flexibilität* (development flexibility) gibt an, wie viel Spielraum bei der Erfüllung der Anforderungen besteht. *Risiko* (risk resolution) berücksichtigt sowohl die Projektrisiken als auch Umfang und Qualität des Risikomanagements im Projekt. *Zusammenarbeit* (team cohesion) misst, wie gut die Projektbeteiligten (Auftraggeber, Endbenutzer, Entwickler, Projektleitung, etc.) zusammenarbeiten. *Prozessreife* (process maturity) schlussendlich ist ein Maß dafür, wie gut die entwickelnde Organisation ihre Software-Prozesse beherrscht.

TABELLE 5.2. COCOMO II: Kostenfaktoren (effort multipliers)

Factor	Very low	Low	Nominal	High	Very High	Extra high
Reliability required	0.75	0.88	1.00	1.15	1.39	
Database size		0.93	1.00	1.09	1.19	
Product complexity	0.75	0.88	1.00	1.15	1.30	1.66
Reuse required		0.91	1.00	1.14	1.29	1.49
Documentation required	0.89	0.95	1.00	1.06	1.13	
Execution time constraint			1.00	1.11	1.31	1.67
Storage constraint			1.00	1.06	1.21	1.57
Platform volatility		0.87	1.00	1.15	1.30	
Analyst capability	1.50	1.22	1.00	0.83	0.67	
Programmer capability	1.37	1.16	1.00	0.87	0.74	
Personnel continuity (turnover)	1.24	1.10	1.00	0.92	0.84	
Application experience	1.22	1.10	1.00	0.89	0.81	
Platform experience	1.25	1.12	1.00	0.88	0.81	
Language and tool experience	1.22	1.10	1.00	0.91	0.84	
Use of software tools	1.24	1.12	1.00	0.86	0.72	
Team co-location and communications support	1.25	1.10	1.00	0.92	0.84	0.78
Required development schedule	1.29	1.10	1.00	1.00	1.00	

TABELLE 5.3. COCOMO II: Skalierungsfaktoren

Faktor	Sehr gering	Gering	Nominal	Hoch	Sehr hoch
Neuartigkeit	4,05	3,24	2,43	1,62	0,81
Flexibilität	6,07	4,86	3,64	2,43	1,21
Risiko	4,22	3,38	2,53	1,69	0,84
Zusammenarbeit	4,94	3,95	2,97	1,98	0,99
Prozessreife	4,54	3,64	2,73	1,82	0,91

5.3.3 Das Function Point-Verfahren

Function Points sind ein *relatives Maß zur Bewertung der Funktionalität*, d.h. des Leistungsumfangs eines Systems. Verfügt ein Unternehmen über Erfahrungszahlen, wie viel Aufwand pro Function Point im Mittel benötigt wird, um Software zu entwickeln bzw. zu pflegen, so können Function Points zur Aufwandschätzung herangezogen werden. Wird in laufenden oder abgeschlossenen Projekten der mittlere Zeitbedarf pro Function Point bestimmt, so bekommt man ein Produktivitätsmaß.

Das Function Point-Verfahren wurde von Albrecht (1979) bei IBM entwickelt und seither von verschiedenen Autoren bzw. Gremien ergänzt und weiterentwickelt (Seibt 1987, Symons 1988, IFPUG 1994). Das Verfahren basiert auf der Idee, die folgenden Größen eines Software-Systems in geeigneter Weise zu zählen und zu bewerten (in Klammern die Terminologie von Albrecht):

- Dateneingaben (External input)
- Datenausgaben (External output)
- Anfragen (External inquiry)
- Schnittstellen zu externen Datenbeständen (External interface file)
- Interne Datenbestände (Logical internal file)

Dateneingaben, Datenausgaben und Anfragen werden an Hand der logischen Transaktionen, die das untersuchte System ausführen soll, gezählt. Tauchen die gleichen Eingabe- bzw. Ausgabedaten in verschiedenen Transaktionen mit unterschiedlicher Verarbeitungslogik auf, werden sie mehrmals gezählt. Bei den Datenbeständen werden logische Dateien bzw. logische Datengruppen (Entitätstypen, Relationen) in Datenbanken gezählt. Die Werte können alle bereits in der Anforderungsspezifikation gezählt werden, sofern diese hinreichend vollständig und detailliert ist.

Jede Dateneingabe, Datenausgabe, Anfrage, etc. wird als „einfach“, „mittel“ oder „komplex“ bewertet und mit einem entsprechenden Gewicht versehen. Tabelle 5.4 zeigt als Beispiel die Gewichtungskriterien für Dateneingaben; Tabelle 5.5 zeigt ein Schema für die Berechnung des Function Point-Rohwerts. Hat also beispielsweise ein System 37 logische Transaktionen mit Dateneingaben, von denen 12 als einfach, 16 als mittel und 9 als komplex bewertet werden, so ergibt sich ein Wert von $12 \cdot 3 + 16 \cdot 4 + 9 \cdot 6 = 154$ als Wert für die Dateneingaben.

TABELLE 5.4. Gewichtungskriterien für Dateneingaben (IFPUG 1994)

Anzahl bearbeiteter Datenbestände	Anzahl unterscheidbarer Datenelemente in der Eingabe		
	1–4	5–15	>15
0–1	einfach	einfach	mittel
2	einfach	mittel	komplex
>2	mittel	komplex	komplex

TABELLE 5.5. Schema zur Berechnung des Function Point Rohwerts UFP (IFPUG 1994)

Element	Schwierigkeitsgrad			
	einfach	mittel	komplex	Summe
Dateneingaben	_____ x 3 = _____	_____ x 4 = _____	_____ x 6 = _____	_____
Datenausgaben	_____ x 4 = _____	_____ x 5 = _____	_____ x 7 = _____	_____
Anfragen	_____ x 3 = _____	_____ x 4 = _____	_____ x 6 = _____	_____
Ext. Schnittstellen	_____ x 5 = _____	_____ x 7 = _____	_____ x10 = _____	_____
Int. Datenbestände	_____ x 7 = _____	_____ x10 = _____	_____ x15 = _____	_____
Function Point Rohwert (Unadjusted Function Points, UFP)				

Der Rohwert wird mit Gewichtungsfaktoren multipliziert, welche die technische Komplexität des Systems reflektieren. Der so genannte Wertkorrekturfaktor (VAF, value adjustment factor[1]) berechnet sich nach der Formel

(5.11) $VAF = 0{,}65 + 0{,}01 \cdot TDI$

TDI ist der "total degree of influence", der gemäß Tabelle 5.6 berechnet wird. Die Formel ist so konstruiert, dass VAF in einem Bereich zwischen 0,65 und 1,35 liegt. Die Function Points FP eines Systems berechnen sich dann zu

(5.12) $FP = UFP \cdot VAF$

Sollen Function Points zur Aufwandschätzung verwendet werden, so muss der zu erwartende Aufwand pro Function Point bekannt sein. Entsprechende Kurven oder Tabellen sind im Umlauf (Bild 5.7), sind aber mit Vorsicht zu genießen. Der Aufwand pro Function Point ist stark von projektspezifischen und unternehmensspezifischen Faktoren abhängig, beispielsweise vom Können der Leute im betreffenden Unternehmen, der verwendeten Entwicklungsumgebung und dem Stellenwert von Qualität (z.B. Umfang der verlangten Dokumente, mittlerer Testaufwand). Ferner muss geklärt werden, welche Aufgaben in der Aufwandschätzung enthalten sind (beispielsweise, ob der Aufwand für die Anforderungsspezifikation eingeschlossen ist oder nicht). Wie Bild 5.5 zeigt, können schon innerhalb desselben Unternehmens signifikant unterschiedliche Umrechnungskurven resultieren.

TABELLE 5.6. Schema zur Bestimmung der Einflussfaktoren (IFPUG 1994)

Nr.	Faktor	Wert
1	Datenkommunikation	
2	Verteilte Funktionen	
3	Leistungsanforderungen	
4	Belastung der Hardware	
5	Verlangte Transaktionsrate	
6	Online-Dateneingabe	
7	Effiziente Benutzerschnittstelle	
8	Online-Datenänderungen	
9	Komplexe Verarbeitungen	
10	Wiederverwendbarkeit	
11	Einfache Installation	
12	Einfache Benutzbarkeit	
13	Installation an mehreren Orten	
14	Änder- und Erweiterbarkeit	
	Summe der Faktoren (TDI)	

Einzusetzen sind Werte zwischen 0 und 5

0 nicht vorhanden, kein Einfluss
1 unbedeutender Einfluss
2 mäßiger Einfluss
3 durchschnittlicher Einfluss[1)]
4 erheblicher Einfluss
5 starker Einfluss

[1)]Von der Konstruktion des Maßes her müsste eigentlich VAF = 1 gelten, wenn alle Faktoren durchschnittlichen Einfluss haben. Das wäre der Fall, wenn durchschnittlicher Einfluss mit 2.5 statt mit 3 bewertet würde.
Aus Praktikabilitätsgründen hat man sich aber für den Wert 3 entschieden, was dazu führt, dass VAF bei lauter durchschnittlichen Einflüssen den Wert 1.07 hat.

Jones (1996) gibt Faustregeln zur Berechnung des Aufwands aus Function Points an.

REGEL 5.1. *Faustregeln von Jones zur Aufwandberechnung.*

A. Durchlaufzeit [in Monaten] = $FP^{0,4}$

[1] In manchen Publikationen heißt dieser Faktor auch TCF (Technical Complexity Factor)

B. Anzahl Mitarbeiter = FP / 150

C. Aufwand = Durchlaufzeit · Anzahl Mitarbeiter = $FP^{0,4} \cdot FP / 150$

Sollen projektspezifische Faktoren (zum Beispiel die Fähigkeit der beteiligten Personen, ihre Vertrautheit mit Problemstellung und die Vertrautheit mit der Lösungsplattform) berücksichtigt werden, so müssen bei der Umrechnung von Function Points in Personenmonate zusätzliche Korrekturfaktoren zur Anwendung kommen.

Das Function Point-Verfahren muss folglich wie jedes andere algorithmische Verfahren kalibriert werden, wenn damit brauchbare Prognosen erzielt werden sollen. Das bedeutet, dass abgeschlossene Projekte nachkalkuliert werden müssen. Aus diesen Daten müssen unternehmens- und eventuell auch projektartspezifisch die Zusammenhänge zwischen Function Points und Aufwendungen hergeleitet werden.

Es gibt unterschiedliche Zählregeln für Function Points. In diesem Text werden die Regeln der „International Function Point Users Group“ (IFPUG 1994) verwendet. In IBM (1983) und Seibt (1987) werden andere Gewichtungskriterien und nur sieben Einflussfaktoren zur Berechnung des Wertkorrekturfaktors herangezogen. Noth und Kretzschmar (1984) verwenden wie IFPUG vierzehn Einflussfaktoren, gewichten aber den Faktor 9 (technische Komplexität) mit 0-50 statt nur mit 0-5. Symons (1988) kritisiert das Albrechtsche Zählverfahren und schlägt ein Zählverfahren vor, das statt kompletter Ein- und Ausgaben die einzelnen Datenfelder zählt (Mark II Function Points).

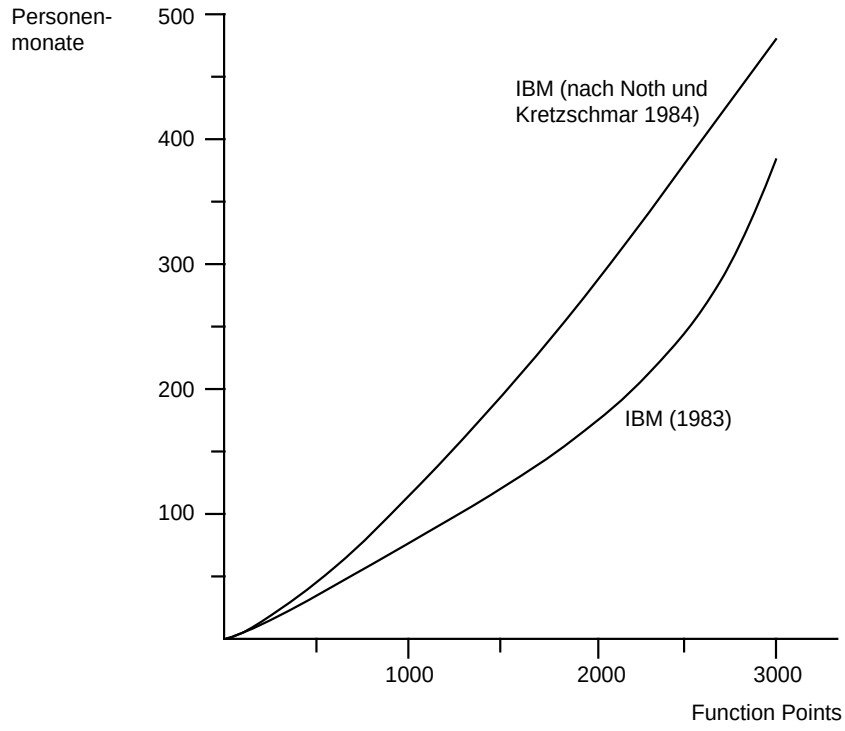


BILD 5.5. Bestimmung des Aufwands aus den Function Points

Jones (1995) gibt Erfahrungswerte für die Umrechnung von Function Points in Codezeilen an (Tabelle 5.7). Die Codezeilen verstehen sich ohne Leerzeilen und ohne Kommentar.

Das Function Point-Verfahren hat den großen Vorteil, dass die benötigten Eingangsgrößen sich in den frühen Phasen eines Projekts leichter bestimmen lassen als beispielsweise die Größe des erwarteten Resultats bei COCOMO. Zudem sind mit Function Points Produktivitätsmessungen möglich, die weniger leicht verfälschbar sind als Messungen auf der Grundlage der erzeugten Programmzeilen.

TABELLE 5.7. Anzahl Codezeilen pro Function Point

Sprache	Mittlere Anzahl Codezeilen
Assembler	320
C	128
FORTRAN	107
COBOL	197
Pascal	91
C++	53
Java	35
Smalltalk	21
SQL	12

Allerdings ist das Verfahren auf Informationssysteme zugeschnitten und daher auf andere Arten von Software (zum Beispiel Prozessautomatisierung) nur beschränkt anwendbar. Ein weiterer Nachteil ist, dass das Function Point-Maß nicht additiv ist: Die Summe der Function Points von n logisch zusammenhängenden Teilsystemen ist größer als die Anzahl der Function Points des Gesamtsystems. Dies liegt daran, dass die Schnittstellen zwischen je zwei Teilsystemen auf beiden Seiten als Schnittstellen zu externen Datenbeständen gezählt werden, während diese Daten bei Betrachtung des Gesamtsystems als *ein* interner Datenbestand gezählt werden. Es ist daher nicht möglich, bei einem großen System die Function Points pro Teilsystem zu bestimmen und diese zu addieren.

5.4 Sonstige Methoden

Es gibt eine Reihe weiterer gängiger Methoden zur Kostenschätzung, z.B. so schätzen, dass man auf jeden Fall den Auftrag bekommt, so viel schätzen, wie der Auftraggeber zu zahlen bereit ist, Schätzung nach dem Parkinson'schen Gesetz: „Das Projekt kostet soviel Arbeitskapazität wie vorhanden ist", usw.

5.5 Schätzung von Pflegekosten

Aus Messungen über Entwicklungs- und Pflegekosten von Software ergeben sich die folgenden beiden Faustregeln für Pflegekosten.

REGEL 5.2. Das Kostenverhältnis zwischen Entwicklung und Pflege eines Software-Produkts liegt im Bereich von 30:70 bis 50:50. Je länger ein Produkt lebt und je schlechter seine Qualität ist, desto höher ist der Kostenanteil für die Pflege.

REGEL 5.3. Die Kosten für die Pflege eines Software-Produkts verteilen sich etwa wie folgt: 60% Verbesserungen, 20% Anpassungen und 20% Fehlerbehebung.

Tabelle 5.8 zeigt verschiedene Schätzwerte für die Anzahl Codezeilen, die ein Vollzeit-Pflegeprogrammierer betreuen kann. Die Aussage 20 $KDSI/FSP_M$[1] bedeutet beispielsweise, dass zur Pflege eines Programms mit 20000 Codezeilen die volle Arbeitskraft einer Person (über das ganze Jahr hinweg) erforderlich ist.

[1] $KDSI/FSP_M$ steht für **k**ilo **d**elivered **s**ource **i**nstructions per **f**ull-time **s**oftware **p**erson (for **m**aintenance).

Sind die Function Points der gepflegten Systeme bekannt und werden die Pflegeaufwendungen über längere Zeit erfasst, so kann daraus der mittlere Pflegeaufwand pro Function Point für das betreffende Unternehmen bestimmt werden. Eine solche Zahl kann dann auch für Prognosen über zu erwartende Pflegeaufwendungen eines neuen Systems verwendet werden. Jones (1996) gibt dazu folgende Faustregel an.

REGEL 5.4. *Faustregel von Jones für Pflegeaufwand.* Zur Pflege eines Systems von N Function Points sind N/500 Personen erforderlich.

TABELLE 5.8. Zeile/Person-Raten in der Software-Pflege

Source	Application Type	$(KDSI/FSP)_M$
COCOMO, lowest		3
[Wolverton, 1980]	Aerospace	8
[Ferens–Harris, 1979]	Aerospace	10
COCOMO, 25th percentile		10
[Daly, 1977]	Real-time	10–30
[Griffin, 1980]	Real-time	12
[Elliott, 1977]	Business	20
[Graver and others, 1977]	Business, HOL	20
[Graver and others, 1977]	Business, MOL	22
COCOMO, median		25
[Lientz–Swanson, 1980]	Business, 487 installations	32
COCOMO, 75th percentile		36
[Daly, 1977]	Support software	30–120
COCOMO, highest		132

Quelle: Boehm (1981)

5.6 Einfluss der Schätzung auf den Aufwand

Es gibt Hinweise dafür, dass Schätzung und tatsächlicher Aufwand keine voneinander unabhängigen Größen sind. Dies ist das durch das Gesetz von Parkinson (Der Aufwand passt sich der verfügbaren Zeit an) beschriebene Phänomen (Bild 5.6). Abdel-Hamid und Madnick (1986) haben in Simulationen eine signifikante Korrelation zwischen der Schätzung und dem tatsächlichen Aufwand gefunden. Diese Ergebnisse dürfen allerdings nicht zu falschen Schlüssen verleiten. Das Parkinson-Phänomen tritt vor allem dann auf, wenn in zu knapp kalkulierten Terminplänen Luft geschaffen wird. Dann werden nämlich all die Arbeiten tatsächlich gemacht, die bei ordnungsgemäßer Entwicklung eigentlich gemacht werden sollten (z.B. sorgfältige Dokumentation und Prüfung), die aber sonst der Termin-Guillotine zum Opfer fallen. In Bild 5.6 beispielsweise

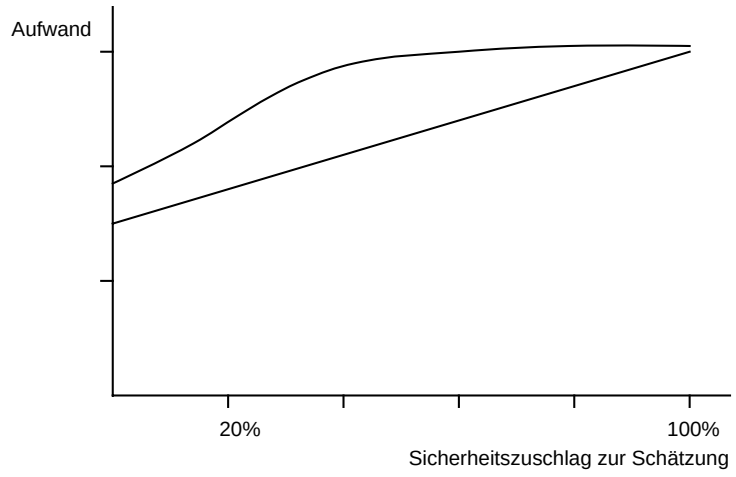


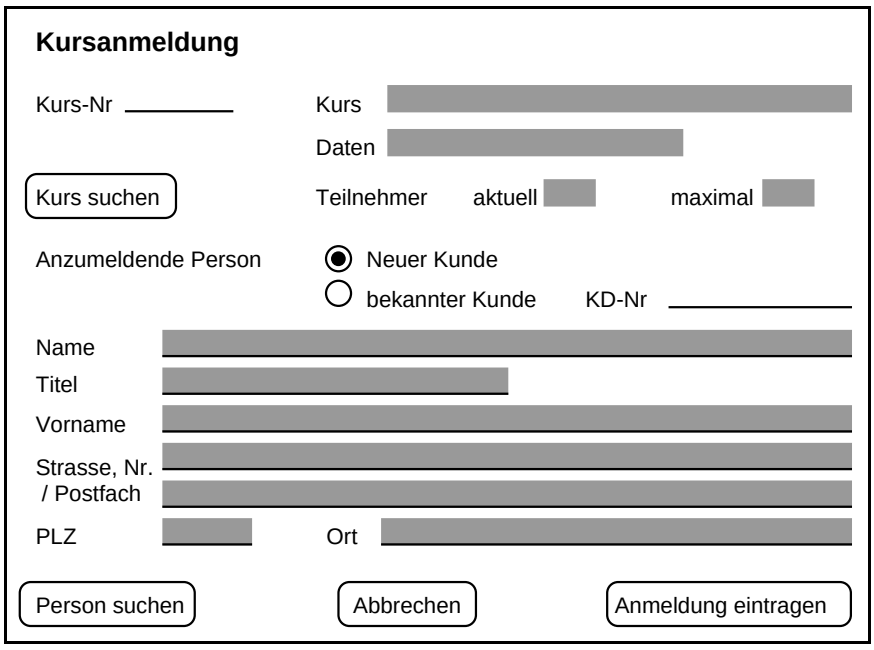
BILD 5.6. Der Parkinson-Effekt: Korrelation von geschätztem und effektivem Aufwand

beeinflusst die Schätzung ab etwa dem 1,6 fachen des ursprünglichen Werts den tatsächlichen Aufwand kaum noch.

Aufgaben

5.1 Sie sind Mitarbeiter(in) in der Informatikabteilung eines Unternehmens. Im Rahmen des unternehmensweiten Qualitätsmanagementsystems nehmen Sie in Ihrer Abteilung die Rolle des Qualitätsbeauftragten ein.
Ihr Chef hat auf einer Konferenz von COCOMO gehört. Da es mit der Aufwandschätzung in Ihrer Abteilung im Argen liegt, hat er beschlossen, das Gehörte in die Tat umzusetzen und ab sofort alle den Aufwand für alle neuen Projekte mit COCOMO schätzen zu lassen. Bevor er seinen Entschluss verkündet, fragt er Sie als Qualitätsbeauftragte(n) noch um Ihre Meinung. Was raten Sie ihm?

5.2 Im Rahmen der Entwicklung eines Informationssystems für die Teilnehmeradministration eines Schulungsunternehmens ist eine Anwendung zur Anmeldung von Kursteilnehmern zu erstellen. Die Anwendung wird über untenstehende Eingabemaske angesprochen. Unterstrichene Felder sind Eingabefelder, graue Felder sind Ausgabefelder, Schaltknöpfe lösen Verarbeitungen und Datenbankzugriffe aus. Die Anwendung verwendet die Gegenstandstypen KURS, KUNDE und BUCHUNG aus der Datenbank.



Berechnen Sie die Function Points für diese Anwendung. Beachten Sie folgende Hinweise:

- Die Maske enthält neben der eigentlichen Transaktion (Anmeldung) zwei weitere eingebettete Transaktionen (Kurs suchen, Person suchen). Diese sind mitzuzählen, wenn sie nicht anderswo bereits als selbständige Transaktionen existieren. Gehen Sie davon aus, dass das hier der Fall ist, d.h. dass beide eingebetteten Transaktionen gezählt werden müssen.
- Anfragen, die nicht mehr als einen Datenbestand betreffen und nicht mehr als 15 Ein- oder Ausgabedaten umfassen, werden als „einfach“ bewertet.
- Sie können davon ausgehen, dass die beteiligten Datenbestände alle als „einfach“ zu bewerten sind.

- Bei den Einflussfaktoren können Sie von folgenden Annahmen ausgehen: Belastung der Hardware: unbedeutend, komplexe Verarbeitungen: nicht vorhanden, Wiederverwendbarkeit: sehr wichtig. Alle übrigen Faktoren haben durchschnittlichen Einfluss.

Ergänzende und vertiefende Literatur

Boehm (1981) ist die Standardreferenz für COCOMO. Boehm et al. (2000) ist das entsprechende Standardwerk für COCOMO II.

IFPUG (1994) enthält detaillierte Zählregeln für Function Points und illustriert diese anhand von Beispielen. Garmus und Herron (2000) haben eine Monographie über Function Point Schätzung auf der Basis der IFPUG Zählregeln verfasst.

Knöll und Busse (1991) beschreiben verschiedene, in der Praxis eingesetzte Verfahren, u.a. auch COCOMO und Function Points.

Wellman (1992) und Jones (1998) sind Monographien über Software-Kosten; letztere präsentiert den Stand der Technik in umfassender Weise.

Zitierte Literatur

Abdel-Hamid, T.K., S.E. Madnick (1986). Impact of Schedule Estimation on Software Project Behavior. *IEEE Software* **3**, 4 (July 1986). 70-75.

Albrecht, A.J. (1979). Measuring Application Development Productivity. *Proceedings Guide/Share Application Development Symposium* (Oct. 1979). 83-92

Boehm, B. (1981). *Software Engineering Economics*. Englewood Cliffs, N.J.: Prentice Hall.

Boehm, B., C. Abts, A. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachi, D. Reifer, B. Steerce (2000). *Software Cost Estimation with Cocomo II.* Pearson Education.

IBM (1983). *Die Function Point-Methode: Eine Schätzmethode für IS-Anwendungs-Projekte*. Broschüre GE 12-1618-0. IBM Deutschland GmbH.

IFPUG (1994). *Function Point Counting Practices Manual, Release 4.0*. International Function Point Users Group. Westerville, Ohio (USA).

Garmus, D., D. Herron (2000). *Function Point Analysis: Measurement Practices for Successful Software Projects.* Reading, Mass.: Addison-Wesley.

Jones, T.C. (1995). Backfiring. Converting Lines of Code to Function Points. *IEEE Computer* **28**, 11 (Nov 1995). 87-88.

Jones, T.C. (1996). Software Estimating Rules of Thumb. *IEEE Computer* **29**, 3 (March 1996). 116-118.

Jones, T. C. (1998). *Estimating Software Costs*. New York: McGraw-Hill.

Knöll, H.-D., J. Busse (1991). *Aufwandschätzung von Software-Projekten in der Praxis*. Reihe Angewandte Informatik Band 8, Mannheim, Wien Zürich: BI-Wissenschaftsverlag.

Moløkken, K., M. Jørgensen (2003). A Review of Surveys on Software Effort Estimation. *IEEE International Symposium on Empirical Software Engineering (ISESE'03)*. 223-230.

Noth, T. und M. Kretzschmar (1984). *Aufwandschätzung von DV-Projekten*. Berlin, etc.: Springer.

Seibt, D. (1987). Die Function-Point-Methode: Vorgehensweise, Einsatzbedingungen und Anwendungserfahrungen. *Angewandte Informatik* 1/1987. 3-11

Symons, C.R. (1988). Function Point Analysis: Difficulties and Improvements. *IEEE Transactions on Software Engineering* **14**, 1 (Jan. 1988). 2-11.

Wellman, F. (1992). *Software Costing*. Hemel Hempstead: Prentice Hall International (UK).