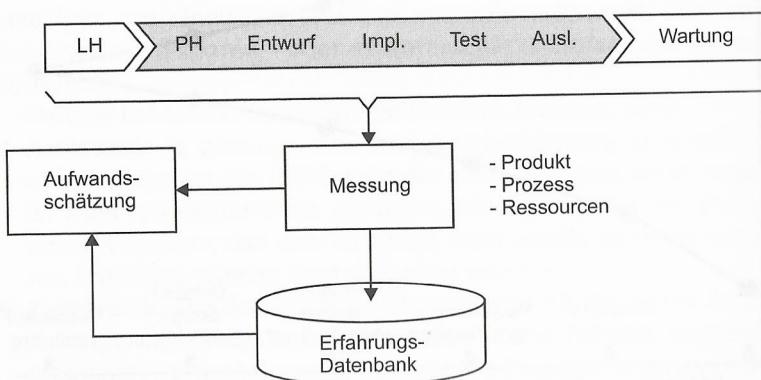


aber auch organisatorische (z. B. Schwierigkeiten bei der Kommunikation mit dem Kunden) oder personelle Schwierigkeiten (Chefentwickler kündigt) sein.

In jedem Fall sollten die der Schätzung zugrunde liegenden Annahmen dokumentiert werden. So kann die Schätzung leichter an neue Entwicklungen angepasst werden, und es ist später nachvollziehbar, warum sie möglicherweise falsch war.

23.4 Schätzverfahren

Alle Schätzmethoden beruhen auf Erfahrung. Es müssen also Daten aus vergangenen Projekten erhoben worden sein und zur Verfügung stehen, aus denen dann auf den Aufwand eines neuen Projekts geschlossen werden kann. Die Abb. 23.4-1 zeigt den grundlegenden Prozess. Während des Entwicklungsprozesses werden Messungen durchgeführt. Deren Ergebnisse bilden zum einen die Basis für eine bestmögliche Aufwandsschätzung zum aktuellen Zeitpunkt, zum anderen werden sie nach Abschluss des Projekts in eine Erfahrungsdatenbank eingespeist. Diese Daten können dann für eine Aufwandschätzung zusätzlich herangezogen werden. Da die Eigenschaften der Entwicklungsorganisation den Aufwand stark beeinflussen, bringen eigene Erfahrungswerte hier den größten Nutzen. Erfahrungswerte anderer Organisationen sind dagegen nur begrenzt einsetzbar.



Es gibt verschiedene Arten von Aufwandsschätzmethoden:

- »Analogiemethode«, S. 523
- »Expertenschätzung«, S. 523
- »Bottom-up-Methode«, S. 524
- »Prozentsatzmethode«, S. 525
- »Algorithmische Schätzung«, S. 526
- »Faustregeln«, S. 526

23.4.1 Analogiemethode

Eine recht einfache Kostenschätzungs methode ist die **Analogiemethode**. Dabei wird ein bereits abgeschlossenes Projekt identifiziert, das dem abzuschätzenden Projekt möglichst ähnlich ist. Man kann davon ausgehen, dass dann auch der Aufwand für die Realisierung des neuen Projekts ähnlich sein wird. Eventuell hat man aus dem abgeschlossenen Projekt gelernt, z. B. Domänenwissen oder anderes Spezialwissen aufgebaut – dann ist der Aufwand möglicherweise geringer. Auch wenn Wiederverwendung möglich ist, reduziert sich der Aufwand.

Die Unterschiede zum gewählten Referenzprojekt müssen in jedem Fall genau betrachtet und für diese Teile der abweichende Aufwand dazuaddiert bzw. abgezogen werden. Das Verfahren arbeitet um so genauer, je ähnlicher die beiden Projekte sind (auch bezüglich der beteiligten Mitarbeiter). Es ist folglich nur anwendbar, wenn überhaupt ein vergleichbares Projekt existiert.

Eine eng verwandte Methode ist die **Relationsmethode**, bei der die Abweichungen vom Referenzprojekt über Faktorlisten und Richtlinien zur Anpassung der Schätzung führen. So kann es beispielsweise Faktoren für den Wechsel von einer bestimmten Programmiersprache zu einer anderen geben. Der Umgang mit Abweichungen ist bei dieser Methode also genau spezifiziert.

Beispiel
In Unternehmen soll ein Buchungssystem für einen Kreuzfahrtveranstalter erstellen. Es hat bereits mehrere ähnliche Systeme für andere Reiseveranstalter realisiert. Nur die für diesen Veranstalter spezifische Funktionalität (z. B. Kreuzfahrt statt Pauschalreisen) muss nun als Neuentwicklung geschätzt werden. Die übrige Funktionalität ist sehr ähnlich zu den vorigen Systemen, daher wird der zu deren Realisierung nötige Aufwand ebenfalls ähnlich sein. Werden Teile eines alten Systems wiederverwendet, sinkt der Aufwand für diese Teile, beispielsweise auf 25 % des Neuerstellungsaufwands.

23.4.2 Expertenschätzung

Eine artverwandte und weit verbreitete Methode ist die **Expertenschätzung**. Hier beurteilt ein Experte das zu schätzende Projekt. Aufgrund seiner langjährigen Erfahrung mit vielen verschiedenen Softwareprojekten ist er in der Lage, den Aufwand intuitiv genauer anzugeben als jemand, der nicht über diese Erfahrung verfügt. Die Schätzung wird also nicht aus Messwerten früherer Projekte berechnet, sondern findet allein im Kopf des Experten statt. Sein Kopf bildet die Erfahrungsdatenbank, und die Messung erfolgte durch seine Mitarbeit in Projekten.

Diese Methode kann weiter verfeinert werden, indem mehrere Experten befragt werden. Bei stark differierenden Schätzungen sollten die Experten dann zusammengebracht und ein Meinungsaustausch angestoßen werden. Da verschiedene Experten unterschiedliche Aspekte bei ihrer Schätzung berücksichtigen werden, sollten durch deren Kombination alle wesentlichen Aspekte einfließen und das Ergebnis entsprechend gut sein.

Experten können eigene erfahrene Entwickler oder Projektleiter sein, aber auch externe Berater mit entsprechendem Hintergrund. Der Vorteil des internen Experten ist dabei, dass er auch die firmenspezifischen Besonderheiten berücksichtigen kann.

Eine weitere Verfeinerung der Expertenschätzung ist das **Dreipunktverfahren**. Dabei werden drei Schätzungen abgegeben: Je- weils eine unter Annahme von optimistischen, pessimistischen und wahrscheinlichen Bedingungen. Daraus kann dann ein gewichteter Mittelwert (wie in PERT¹) berechnet werden:

$$T_E = (T_{\text{optimistisch}} + 4 \cdot T_{\text{wahrscheinlich}} + T_{\text{pessimistisch}}) / 6$$

Die **Wideband-Delphi-Methode** [Boeh81] ist ein Beispiel für eine systemisierte Expertenschätzung. Eine Gruppe von 3–7 Mitarbeitern trifft sich zwei Mal. Beim ersten Treffen wird das Projekt in ca. 10–20 Teilaufgaben zerlegt. Diese werden von jedem Mitarbeiter separat geschätzt, wobei die jeweils getroffenen Annahmen dokumentiert werden. Dann treffen sich die Mitarbeiter zum zweiten Mal, tauschen ihre Schätzungen aus und diskutieren diese unter Anleitung eines Moderators. Unterschiedliche Annahmen und Unklarheiten werden zusammengeführt und geklärt. So nähern sich die Schätzungen in mehreren Iterationen aneinander an, bis deren Spannweite für jede Teilaufgabe als akzeptabel angesehen wird.

23.4.3 Bottom-up-Methode

Eine Alternative zur direkten Schätzung der Gesamtkosten eines Projekts ist es, das Projekt in **kleine Einzelteile** zu zerlegen, die Kosten für die Entwicklung der Einzelteile zu schätzen und diese dann zusammenzuaddieren. Hinzu kommt der Integrationsaufwand. Dies ist die sogenannte **Bottom-up-Methode**. Diesem Verfahren liegen folgende Ideen zugrunde:

1 Bei den Schätzungen der Einzelteile kommen Abweichungen nach oben und nach unten in etwa gleich häufig vor und heben sich damit gegenseitig auf.

2 Kleinere Teile sind besser überschaubar und damit besser abschätzbar, d.h. die Genauigkeit dieser Teilschätzungen wird relativ gut sein.

3 Bei der Schätzung können die späteren Entwickler einbezogen werden, die ihren eigenen Aufwand vermutlich besser einschätzen können als Personen, die nicht an der Entwicklung beteiligt sind. Außerdem sind sie in der Regel Experten auf ihrem Spezialgebiet (z.B. Datenbanken oder Benutzungsoberflächen). Es handelt sich hier um eine Art **Expertenschätzung** im Kleinen, auf anderer Ebene.

Allerdings können gerade beim Zusammenbau und Zusammenspiel der Einzelkomponenten die größten Probleme auftreten, die bei diesem Verfahren leicht übersehen oder unterschätzt werden.

Watts Humphrey entwickelte für den persönlichen Softwareprozess **PROBE** (*Proxy Based Estimating*) – eine Methode zur Bestimmung des Entwicklungsaufwands einer einzelnen Komponente [Hump95]. Jeder Entwickler baut eine Datenbank mit Informationen über Komponenten auf, die er in der Vergangenheit entwickelt hat. Die Komponenten werden nach ihrem Typ (z.B. Berechnung, Daten, Logik) und ihrer Größe klassifiziert. Mittels linearer Regression werden diese Daten für die Schätzung des Entwicklungsaufwands zukünftiger Komponenten der gleichen Klasse benutzt. Da die Komponenten von einer einzelnen Person entwickelt werden, kann hier ein linearer Zusammenhang angenommen werden.

PROBE

23.4.4 Prozentsatzmethode

Die **Prozentsatzmethode** basiert auf der Annahme, dass die gleichen Phasen von Projekten (z.B. Anforderungsspezifikation, Entwurf, Implementierung, Test) immer einen ähnlichen Anteil am Gesamtaufwand haben. Dann kann aufgrund historischer Daten vom Aufwand einer einzelnen (z.B. der ersten) Phase auf den zu erwartenden Aufwand der anderen Phasen sowie den Gesamtaufwand geschlossen werden. Allerdings müssen für die Anwendbarkeit dieser Methode ebenfalls gewisse Anforderungen an die Ähnlichkeit der Projekte und Prozesse erfüllt sein. Die Verteilung ist unter anderem abhängig von der Größe des Projekts (Tab. 23.4-1).

Für ein System wurde bereits die Anforderungsspezifikation erstellt. Dafür wurden 10 **Personenmonate** (PM) benötigt, und aus der Spezifikation wurde ein Umfang von ca. 1.000 **Function Points** (siehe »Die Function-Points-Methode«, S. 527) ermittelt. Dann wird (Tab.

Beispiel

¹Die *Program Evaluation and Review Technique* ist eine ereignisorientierte Netzplantechnik, die für die Planung von Projekten, die mit hohen Unsicherheiten behaftet sind, entwickelt wurde. Der Zeitbedarf für jeden Vorgang wird nicht durch eine feste Größe, sondern über eine Beta-Wahrscheinlichkeitsverteilung geschätzt. Dabei kommt die

23 Schätzen des Aufwands

Aktivität	10 FP	100 FP	1.000 FP	10.000 FP	100.000 FP
Spezifikation	5	5	7	8	9
Entwurf	5	6	10	12	13
Implementierung	50	40	30	20	15
Test	27	29	30	33	34
Change Management	1	4	6	7	8
Dokumentation	4	6	7	8	9
Projektmanagement	8	10	10	12	12

23.4-1) ein Gesamtaufwand von ca. $10 \text{ PM} \cdot 100\% / 7\% = 143 \text{ PM}$ erwartet, von denen jeweils ca. 43 PM (=30%) für Implementierung und Test benötigt werden.

23.4.5 Algorithmische Schätzung

Bei der algorithmischen Schätzung (»Berechnung aus früh bekannten Größen«) werden eine Vielzahl von Fragen zum Projekt und zur Organisation gestellt und aus den Antworten auf den Aufwand geschlossen. Auch hier basiert die Schätzung auf Erfahrung aus abgeschlossenen Projekten: Entsprechende Zusammenhänge zwischen den Antworten (einer daraus resultierenden Punktzahl) und dem tatsächlichen Realisierungsaufwand müssen zuvor aus empirischen Daten ermittelt worden sein. Die zwei bekanntesten algorithmischen Verfahren sind:

- »Die Function-Points-Methode«, S. 527
- »COCOMO II«, S. 536

Die Kalibrierung dieser Verfahren erfolgt üblicherweise per Regressionsanalyse. Daneben wurden in den vergangenen 20 Jahren auch Ansätze aus der künstlichen Intelligenz für das Lernen des Verhältnisses zwischen Anforderungen und Aufwand benutzt. Dabei kommen unter anderem neuronale Netze, Bayes'sche Netze, unscharfe Logik oder Entscheidungsbäume zum Einsatz. COCOMO II wurde beispielsweise mit einem Bayes-Ansatz kalibriert, und das PROBE-Verfahren (siehe »Bottom-up-Methode«, S. 524) ist der unscharfen Logik zuzuordnen.

23.4.6 Faustregeln

Die meisten Schätzverfahren sind recht aufwendig. Alternativ gibt es auch einige sehr einfache Faustregeln, die ebenfalls auf Erfahrung basieren, jedoch aufgrund ihrer starken Vereinfachung nur zu sehr ungenauen Ergebnissen führen können. Sie basieren alle auf **DLOC**, der Anzahl der ausgelieferten produktiven Codezeilen.

23.5 Die Function-Points-Methode IV

DLOC schließt Code, der nur für Tests benötigt wird, nicht mit ein. In der Tab. 23.4-2 sind einige Faustregeln für prozedurale Sprachen in Abhängigkeit des Gesamtumfangs angegeben.

Größe [DLOC]	DLOC/PM	DLOC/PS
1.000	833	5,76
10.000	588	4,13
100.000	400	2,60
1.000.000	267	2,02

Tab. 23.4-2:
Faustregeln für
prozedurale
Sprachen
(PM=Personenmonat,
PS=Personenstunde)
[Jone07].

Beispiel

Ein System, das auf 200.000 DLOC geschätzt wird, kann (interpoliert) mit einer Produktivität von ca. 360 DLOC pro Personenmonat rechnen. Somit ergibt sich der erwartete Gesamtaufwand zu $200.000 / 360 = 556 \text{ PM}$.

Entsprechende Regeln existieren auch für *Function Points* (siehe »Die Function-Points-Methode«, S. 527). In jedem Fall wird auch hier eine existierende Schätzung des Umfangs vorausgesetzt.

[Kras05]

Weiterführende
Literatur

23.5 Die Function-Points-Methode

Eine verbreitetes algorithmisches Schätzverfahren ist die im Jahr 1979 von Alan J. Albrecht (IBM) veröffentlichte **Function-Points-Methode** [Albr79]. Sie ist auch heute noch eines der wenigen Standardschätzverfahren für Software und hat daher eine entsprechende Bedeutung und Verbreitung. Seit 2003 ist sie standardisiert, unter anderem in der ISO/IEC-Norm 20926².



Alan J. Albrecht

Die Idee dieser Methode ist es, den funktionalen Umfang eines zu entwickelnden Softwaresystems auf Basis der funktionalen Anforderungen zu messen. Die Anforderungen (laut Lasten- oder Pflichtenheft) definieren, welche Funktionalität aus Sicht des Benutzers vom System erwartet wird. Dies wird in der Regel durch eine Beschreibung der nötigen Ein- und Ausgaben sowie der beteiligten Daten erreicht. Ist diese Information vorhanden, so kann sie systematisch ausgewertet werden und gibt ein relativ genaues Bild vom Umfang der Software – zumindest aus Benutzersicht. Aus dem Resultat kann dann unter Ausnutzung historischer Daten auf den personellen Aufwand für die Umsetzung geschlossen werden. Die Function-Points-Methode zielt dabei besonders auf klassische Informationssysteme ab (bzw. wurde dafür entwickelt), bei denen Informationen in ent-

² ISO/IEC 20926 ist die IFFUG-Variante (siehe unten) der Function Points-Methode. Auch andere Versionen sind standardisiert, etwa COSMIC in der ISO/IEC-Norm 1976.

23 Schätzen des Aufwands

sprechenden Masken eingegeben und über Berichte ausgewertet und ausgegeben werden. Sie ist jedoch prinzipiell unabhängig von der späteren technischen Realisierung des Systems.

Der Vorteil der Function-Points-Methode ist, dass sie zu einem relativ frühen Zeitpunkt im Projektverlauf schon relativ genaue und objektive Aussagen zum funktionalen Umfang der zu entwickelnden Software erlaubt. Voraussetzung dafür ist allerdings die Verfügbarkeit eines Lastenhefts, das die wesentlichen Anforderungen schon berücksichtigt und möglichst detailliert beschreibt. Daneben muss genau definiert sein, wie gezählt wird, um tatsächlich zu einem objektiven und vergleichbaren Ergebnis zu kommen. Entsprechende Zählregeln werden beispielsweise von der IFPUG (*International Function Point Users Group*) festgelegt. Deren Einhaltung kann dann zu einer besseren Austauschbarkeit beitragen.

Der grobe Ablauf der Function-Points-Methode ist wie folgt:

1 Zähltyp festlegen. Hierbei wird zwischen einer Neuentwicklung, einer Weiterentwicklung und der Zählung eines bestehenden Systems unterschieden.

2 Systemgrenzen festlegen. In diesem Schritt wird genau definiert, welche Teile zum System gehören und welche außerhalb des Systems liegen. Hieraus ergeben sich die Schnittstellen (Ein-/Ausgaben) und die Unterscheidung zwischen internen und externen Daten/Datenbanken. Dies ist in der Abb. 23.5-1 am Beispiel von SemOrg veranschaulicht (siehe »Fallstudie: SemOrg – Die Spezifikation«, S. 107). Die Systemgrenze ist als Rechteck dargestellt, die Schnittstellen durch Pfeile.

3 Identifizieren von Funktionstypen einschließlich Datenbeständen. Hier erfolgt eine Zerlegung in Elementarprozesse und deren Zuordnung zu einem von fünf Funktionstypen. Dies wird im Folgenden genauer beschrieben.

4 Bewertung der Komplexität der einzelnen Funktionstypen. Je nach Funktionstyp und Umfang der jeweils beteiligten Daten wird die Anzahl zu zählender *Function Points* bestimmt.

5 Ermittlung der gewichteten Function Points (optional). Hierbei werden weitere Einflüsse berücksichtigt, die sich auf den Umfang des Systems auswirken, aber nicht in Ein-/Ausgaben und Datenbeständen zum Ausdruck kommen.

Im dritten Schritt wird der Funktionsumfang der zu bewertenden Anwendung in Elementarprozesse zerlegt. Ein **Elementarprozess** ist eine atomare und einzigartige Aktivität des Systems aus Anwendersicht. **Atomar** bedeutet, dass der Elementarprozess die kleinste aus fachlicher Sicht sinnvolle in sich abgeschlossene Aktivität ist, die mit dem System durchführbar ist. **Einzigartig** bedeutet, dass der Elementarprozess durch die ein- oder ausgegebenen Daten oder

23.5 Die Function-Points-Methode IV

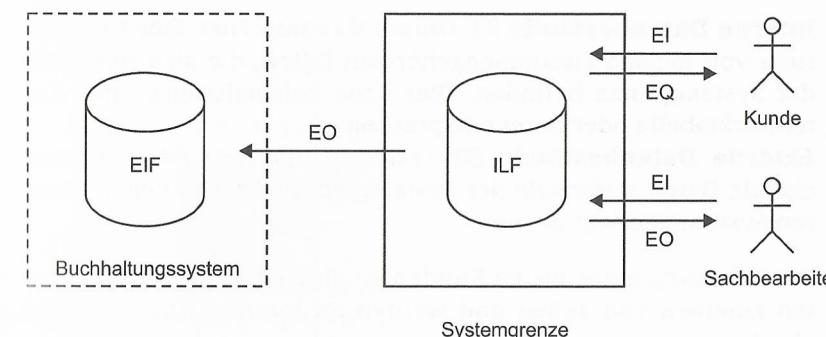


Abb. 23.5-1:
Systemgrenze am
Beispiel SemOrg.
Extern: Buchhal-
tungssystem (EIF),
Kunde, Kunden-
sachbearbeiter.
Intern:
Datenbestand (ILF)
Seminare,
Veranstaltungen,
Kunden, Dozenten,
Firmen,
Buchungen.
Transaktionen (EI,
EO, EQ) bewegen
Informationen
über die
Systemgrenze.

durch die Verarbeitungslogik aus Sicht des Anwenders unterscheidbar ist. Durch Anwendung dieser Prinzipien erreicht die Function-Points-Methode eine eindeutige und nachvollziehbare Zerlegung.

Beispiele für Elementarprozesse:

- a Die Erfassung eines neuen Kunden /LF20/.
- b Die Anzeige einer Veranstaltungsliste /LF10/.
- c Der Ausdruck von Versandpapieren /LF30/.
- d Die Übermittlung von Buchungsdaten an das Buchhaltungssystem /LF80/.

Jeder Elementarprozess lässt sich – auf Basis seines Hauptzwecks – einem **Funktionstyp** zuordnen:

- 1 Transaktions-Funktionstypen:** Hierbei werden Daten über die Systemgrenze bewegt.
- a **Eingaben EI (External Input):** Daten werden von außen ins System gebracht und im System gespeichert (als Datenbestand oder Systemzustandsänderung).
 - b **Ausgaben EO (External Output):** Daten werden vom System generiert und nach außen gegeben. Bei dieser Transaktion müssen entweder die Daten vor der Ausgabe transformiert (abgeleitet) oder der Systemzustand bzw. die internen Daten geändert werden.
 - c **Abfragen EQ (External Inquiry):** Eingabedaten gehen ins System, werden dort aber *nicht* gespeichert, und Ausgabedaten werden vom System nach außen gegeben. Hierbei handelt es sich im Gegensatz zur **Ausgabe** um eine reine Abfrage von Daten – ohne weitere Verarbeitung und ohne Seiteneffekte. Ein Beispiel für diesen Transaktionstyp ist eine Suchanfrage. Die Eingabedaten enthalten die Suchkriterien.
- 2 Daten-Funktionstypen:** Dies sind Sammlungen von Daten, die sich entweder innerhalb oder außerhalb der Systemgrenze befinden. Sie werden durch die Transaktions-Funktionstypen gepflegt.

Funktions-
typen

23 Schätzen des Aufwands

- a Internen Datenbestände ILF (Internal Logical File):** Eine Gruppierung von logisch zusammengehörigen Daten, die sich *innerhalb* der Systemgrenze befinden. Dies kann beispielsweise einer Datenbanktabelle oder Datei entsprechen.
- b Externe Datenbestände EIF (External Interface File):** Entsprechende Daten *außerhalb* der Systemgrenze, die von einem anderen System gepflegt werden.
- c** Die Erfassung eines neuen Kunden ist eine Eingabe, denn die Daten kommen von außen und werden im internen Datenbestand abgelegt.
- d** Die Anzeige einer Veranstaltungsliste ist eine Abfrage. Die Daten werden über die Systemgrenze bewegt und unverändert angezeigt, und durch die Abfrage wird der Systemzustand nicht beeinflusst.
- e** Der Ausdruck von Versandpapieren ist eine Ausgabe, da die Daten über die Systemgrenze zum Drucker geschickt werden und der erfolgte Druck in der Datenbank vermerkt wird (/F120/). Somit erfolgt eine dauerhafte Änderung des internen Datenbestands.
- f** Die Kundendaten werden innerhalb des Systems abgelegt, also in einem ILF (*Internal Logical File*) (/F70/).

Nachdem so die Elementarprozesse klassifiziert wurden, wird als nächstes ihre Komplexität bewertet. Dazu müssen jeweils zwei der folgenden Elementtypen gezählt werden:

- Ein **Datenelement DET (Data Element Type)** ist ein für den Benutzer erkennbares, eindeutig bestimmtes, nicht-rekursives Feld. Dazu gehören zum Beispiel Eingabefelder in einer GUI, aber auch Datenfelder einer Datei. DETs werden für jeden Funktionstyp gezählt.
- Ein **referenzierter Datenbestand FTR (File Type Referenced)** ist ein Datenbestand (ILF oder EIF), der von einer Transaktion (EI, EO, EQ) verwendet wird.
- Eine **Feldgruppe RET (Record Element Type)** ist eine vom Benutzer erkennbare, logisch zusammengehörige Gruppe von Datenelementen innerhalb eines Datenbestands. Die Feldgruppen werden für jeden internen oder externen Datenbestand (ILF oder EIF) gezählt.

Aus der Kombination von DETs mit FTRs (für Transaktionen) oder RETs (für Datenbestände) werden dann aus einer von fünf Komplexitätsmatrizen die *Function Points* abgelesen. Den Aufbau und den Inhalt der Matrizen zeigen die Abb. 23.5-2, die Abb. 23.5-3 und die Abb. 23.5-4.

23.5 Die Function-Points-Methode IV

DETs			
ET	1-5	5-15	≥16
FTRs	1	3	3
	2	3	4
	≥ 3	4	6
			6

Abb. 23.5-2: Komplexitätsmatrix Externe Eingaben (EI): Datenelemente (DETs) – referenzierte Datenbestände (FTRs).

DETs			
EO	1-5	6-19	≥20
FTRs	0-1	4	4
	2-3	4	5
	≥ 4	5	7
			7

DETs			
EQ	1-5	6-19	≥20
FTRs	0-1	3	3
	2-3	3	4
	≥ 4	4	6
			6

Abb. 23.5-3: Komplexitätsmatrixen Externe Ausgaben (EO), externe Abfragen (EQ): Datenelemente (DETs) – referenzierte Datenbestände (FTRs).

DETs			
ILF	1-19	20-50	≥51
RETs	1	7	7
	2-5	7	10
	≥ 6	10	15
			15

DETs			
ELF	1-19	20-50	≥51
RETs	1	5	5
	2-5	5	7
	≥ 6	7	10
			10

Abb. 23.5-4: Komplexitätsmatrixen Interne Datenbestände (ILF), externe Datenbestände (EIF): Datenelemente (DETs) – Feldgruppen (RETs).

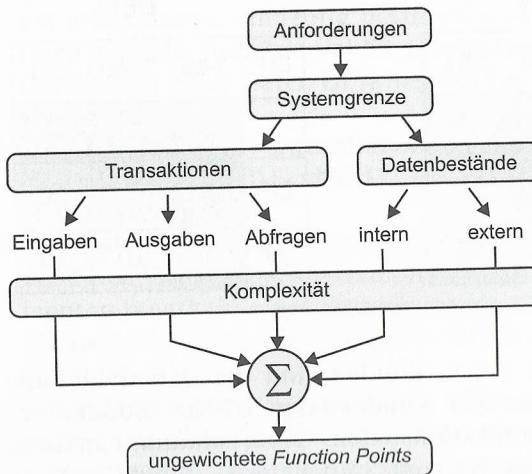
- a** Bei der Erfassung des neuen Kunden sind die Datenfelder die einzelnen Eingabefelder der Kundendaten (/F70/: Kunden-Nr., Name, Adresse, Kommunikationsdaten, Geburtsdatum, Funktion, Umsatz, Kurzmitteilung, Notizen, Info-Material, Kunde seit). In diesem Fall würden daher 11 DETs und 1 FTR (die Kundentabelltabelle) gezählt, was laut EI-Matrix (Abb. 23.5-2) 3 FPs ergibt (2. Spalte, 1. Zeile).

Beispiel 1c

23 Schätzen des Aufwands

- b** Die Anzeige der Veranstaltungsliste betrifft die logischen Datencontainer Seminar und Veranstaltung (2 FTRs) und liest bis zu 17 DETs an Veranstaltungsdaten (/F100/) und 13 DETs an Seminar- daten (/F90/), was in der Summe 30 DETs ergibt. Laut EQ-Matrix (Abb. 23.5-3, rechts) ergibt dies 6 FPs (3. Spalte, 2. Zeile).
- c** Beim Drucken ist die Anzahl der DETs und FTRs von den für den Druck zu lesenden Daten abhängig. Dies ist für SemOrg nicht genauer spezifiziert. Vermutlich werden die Daten über das gebuchte Seminar bzw. die Veranstaltung sowie die Daten des Teilnehmers bzw. Kunden und dessen Firma benötigt. Dies sind im Höchstfall 5 FTRs und 51 DETs, was laut EO-Matrix 7 FP ergibt – die höchstmögliche Wertung für Ausgabetransaktionen.
- d** Die Kundendaten (/F70/) bestehen aus 11 Feldern (DETs), die zu 2 logisch zusammengehörigen Feldern (RETs) zusammengefasst werden können: Stammdaten und Annotationen. Daraus ergeben sich laut ILF-Matrix (Abb. 23.5-4, links) insgesamt 7 FP.

Die so ermittelten *Function Points* werden über alle Elementarprozesse aufsummiert und ergeben so die **ungewichteten Function Points** (UFP). Diese stellen den Umfang der Anwendung dar. Der Prozess ist noch einmal in der Abb. 23.5-5 zusammengefasst. Sofern Zählregeln eingehalten werden, ist das Ergebnis einigermaßen eindeutig und vergleichbar. So stellte Kemerer bei Anwendung der IF-PUG 3.0-Zählregeln Abweichungen zwischen zwei Zählern von maximal 11 Prozent fest [Keme93].



5:
er
en
ts
er
n.

- 1d** Für die Verwaltung der Kundendaten sind die Elementarprozesse gemäß der Tab. 23.5-1 zu berücksichtigen. Es ergibt sich für diesen Teil der Software ein Umfang von 26 *Function Points*.

23.5 Die Function-Points-Methode IV

Funktion/ Datenbestand	Typ	Anzahl RETs/FTRs	Anzahl DETs	FP
Kundendaten	ILF	3	11	7
Neuer Kunde	EI	2	7	4
Kundendaten aktualisieren	EI	2	7	4
Kundendaten löschen	EI	1	1	3
Kundenliste anzeigen	EQ	3	11	4
Kundendaten anzeigen	EQ	3	11	4
Summe				26

Tab. 23.5-1:
Function Points für die Fallstudie SemOrg.

Gewichtete
Function Points

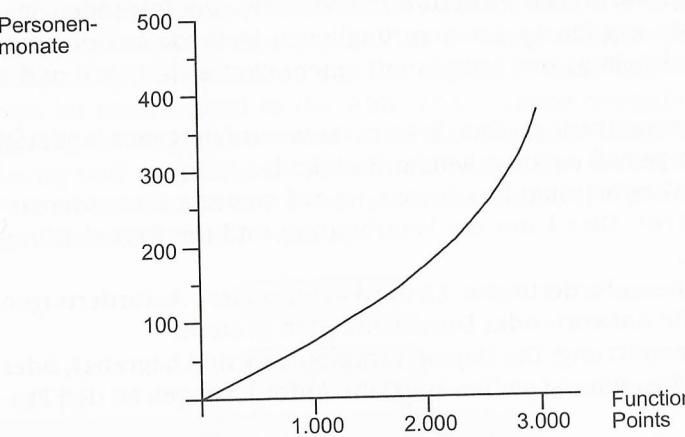
In einem weiteren (optionalen) Schritt werden zusätzliche Einflüsse anhand von 14 Systemmerkmalen berücksichtigt. Diese umfassen vor allem technische Faktoren, aber auch nichtfunktionale Anforderungen. Sie können die UFP um $\pm 35\%$ verändern und ergeben die so genannten **gewichteten Function Points** (GFP). Die folgenden Systemmerkmale werden in der ursprünglichen Methode bei der Korrektur berücksichtigt und jeweils mit einem Wert zwischen 0 und 5 bewertet:

- Datenkommunikation: Das System kommuniziert mit anderen Systemen gemäß entsprechender Protokolle.
- Verteilte Verarbeitung: Das System ist auf mehrere Computersysteme verteilt. Dies kann die Verarbeitung und die Datenhaltung betreffen.
- Performanceanforderungen: Es werden besondere Anforderungen an schnelle Antwort- oder Durchlaufzeiten gestellt.
- Ressourcennutzung: Die Hardwareressourcen sind begrenzt, oder Teile der Anwendung stellen spezielle Anforderungen an den Prozessor.
- Transaktionsrate: Es muss mit einer so hohen maximalen Transaktionsrate gerechnet werden, dass spezielle Maßnahmen erforderlich sind, um diese verarbeiten zu können.
- Online-Dateneingabe: Das System soll interaktive Dateneingaben erlauben (im Gegensatz zum Batch-Betrieb).
- Benutzungsfreundlichkeit: Interaktive Funktionen sollen benutzerfreundlich realisiert werden, z. B. durch Einsatz von Navigationshilfen, Popup-Fenstern oder Mehrsprachigkeit.
- Online-Update: Die ILFs (*Internal Logical Files*) sollen zur Laufzeit interaktiv geändert werden können. Dabei sind unter Umständen Vorkehrungen gegen Datenverlust zu treffen.
- Komplexe Verarbeitung: Dazu gehören beispielsweise komplexe logische, mathematische oder Fehlerbehandlungs-Operationen.
- Wiederverwendbarkeit: Der Code soll in zukünftigen Systemen wiederverwendbar sein.

23 Schätzen des Aufwands

- Migrations-/Installationshilfen: Bestehende Daten sollen ins neue System konvertiert werden können.
- Betriebshilfen: Vollautomatischer Betrieb mit entsprechenden Datensicherungs- und Wiederherstellungsfunktionen wird unterstützt.
- Mehrfachinstallationen: Das System soll in verschiedenen Hard- und Softwareumgebungen lauffähig sein.
- Änderungsfreundlichkeit: Das System unterstützt flexible Abfragen und Berichte. Steuerdaten können vom Benutzer konfiguriert werden.

Das Ergebnis kann schließlich als Eingabe für COCOMO II (siehe »COCOMO II«, S. 536) dienen (ungewichtete FP) oder per Erfahrungskurve in Aufwand umgerechnet werden (gewichtete FP). Die Abb. 23.5-6 zeigt die 1983 von IBM veröffentlichte Umrechnungskurve. Diese sollte keinesfalls direkt angewendet werden, veranschaulicht aber das typische Aussehen.



Erst durch Erstellung und Verwendung eigener Erfahrungskurven lassen sich brauchbare Schätzungen erstellen. Alternativ dazu können die *Function Points* auch per Daumenregel direkt in Personenmonate oder Codezeilen umgerechnet werden. So entspricht 1 *Function Point* etwa 128 Zeilen C-Code oder 53 Zeilen Java-Code. Umrechnungsfaktoren für weitere Sprachen sind in der Tab. 23.5-2 angegeben [Jone95].

Weitere Faustregeln, basierend auf gewichteten *Function Points* (GFP) [Jone07]:

$$\text{Entwicklungsduer} = \text{GFP}^{0,4}$$

$$\text{Personen} = \text{GFP} / 150 \text{ (aufgerundet)}$$

$$\text{Aufwand (PM)} = \text{Personen} \cdot \text{Entwicklungsduer}$$

$$\text{Wartungspersonal} = \text{GFP} / 750$$

$$\text{Produktivität: 10 FP pro PM}$$

23.6 Object Points/Application Points IV

Tab. 23.5-2:
Faustregeln für
LOC pro Function
Point [Jone95].

Sprache	LOC/UFP	Sprache	LOC/UFP
Access	38	HTML 3.0	15
Ada 95	49	Java	53
Assembler	320	Pascal	91
C	128	PERL	27
C++	55	Shell Script	107
Cobol	91	SQL	13
Tabellenkalkulation	6	Visual Basic 5	29
4GL	20	Visual C++	34

Kosten: 1.200 \$ pro FP

Die Function-Points-Methode wird – trotz ihrer Abstammung aus der IT-Welt von vor 30 Jahren – auch heute noch verbreitet eingesetzt. Sie wird von der IFPUG weiterentwickelt und standardisiert. Daneben existieren knapp 40 weitere Varianten der Function-Points-Methode, die teilweise im Hinblick auf bestimmte Domänen (etwa Echtzeitsysteme) erweitert wurden. Ein Beispiel für eine solche Variante ist COSMIC (*Common Software Measurement International Consortium*). Die Notwendigkeit dieser Varianten ist allerdings umstritten [Jone07].

- + Die Erstellung eigener Erfahrungskurven und Einhaltung einheitlicher Zählregeln vorausgesetzt, lassen sich mit der Function-Points-Methode sehr gute Ergebnisse erzielen [Keme93].
- Nachteile sind der recht hohe Aufwand für die Zählung und die subjektive Bewertung der Systemmerkmale.

[ÖzMe06], [PoBo05]

Weiterführende Literatur

23.6 Object Points/Application Points

Der hohe Aufwand für eine exakte Function-Points-Zählung führte im Rahmen von COCOMO II (siehe »COCOMO II«, S. 536) zur Einführung einer vereinfachten Methode – der Zählung von *Object Points*. Diese werden auch *Application Points* genannt, um Assoziationen zur objektorientierten Programmierung zu vermeiden. Diese Methode ist für 4GL-Projekte³ gedacht, das heißt für Projekte, in denen Berichts- und Eingabeformulargeneratoren eingesetzt werden, um die Menge des von Hand zu erstellenden Codes zu verringern. Die Zählung betrachtet im Wesentlichen die Anzahl verschiedener Eingabeformulare (*Screens*) und Berichte (*Reports*) sowie die Anzahl zusätzlich erforderlicher traditioneller – ausprogrammierter – 3GL-Module⁴. Für

³4GL: Programmiersprachen der 4. Generation, im Wesentlichen Sprachen zur Abfrage von Datenbanken, z. B. SQL.

⁴3GL: Programmiersprachen der 3. Generation, das sind prozedurale Programmiersprachen wie z. B. Pascal, Ada oder C.

jedes Formular und jeden Bericht wird die Komplexität bewertet – zum Beispiel aufgrund der Menge und unterschiedlichen Zugehörigkeit oder Herkunft (Datenbanken bzw. Tabellen) der Daten. Aus der Tab. 23.6-1 können dann direkt die entsprechenden *Object Points* abgelesen werden. Die zusätzlichen 3GL-Module werden jeweils konstant mit 10 *Object Points* gezählt. Die Gesamtsumme ergibt ein Maß für den Umfang der Anwendung.

	Geringe Komplexität	Mittlere Komplexität	Hohe Komplexität
Formular	1	2	3
Bericht	2	5	8
3GL-Komponente			10

Enthält das zu erstellende Produkt 7 Eingabeformulare mittlerer Komplexität, 2 Formulare hoher Komplexität, 8 Berichte mittlerer Komplexität sowie 4 3GL-Module, so ergeben sich:
 $7 \cdot 2 + 2 \cdot 3 + 8 \cdot 5 + 4 \cdot 10 = 100 \text{ Object Points}$

Mit *Object Points* wird für 4GL-Projekte mit etwa halbem Aufwand eine zu *Function Points* vergleichbare Präzision erreicht [BKK91]. *Object Points* werden vor allem in der frühen Prototypenstufe von COCOMO II eingesetzt.

23.7 COCOMO II

Das Kostenschätzungsmodell COCOMO (*Constructive Cost Model*) wurde 1981 von Barry Boehm vorgestellt [Boeh81]. 1995 entwickelte er eine überarbeitete Variante, die die Veränderungen in der Softwareentwicklungswelt berücksichtigt [BCH95], [Boeh+00]. Diese zweite Version wird hier vorgestellt.

COCOMO II bietet drei verschiedene Modelle an, die sich nach dem Zeitpunkt der Schätzung unterscheiden und für spätere Zeitpunkte zunehmend genauer werden:

- Frühe Prototypenstufe
- Frühe Entwurfsstufe
- Stufe nach Architekturentwurf

Die **frühe Prototypenstufe** (*Application Composition Model*) basiert auf einer Zählung der *Object Points* (siehe »Object Points/Application Points«, S. 535), einer Einschätzung der Produktivität (Entwickler und Werkzeuge) sowie dem aus früheren Projekten wiederverwendeten Anteil. Aus diesen Größen wird der Aufwand ermittelt, indem der Umfang zur Produktivität in Beziehung gesetzt wird:

$$\text{PM} = \text{NOP} / \text{PROD}$$

$$\text{NOP} = \text{OP} \cdot (100 - \% \text{ reuse}) / 100$$

Dabei sind OP die gezählten *Object Points*, %reuse ist der Anteil, der durch Wiederverwendung abgedeckt werden kann, NOP sind die *New Object Points* (OP korrigiert bezüglich Wiederverwendung), PROD ist die Produktivität und PM ist der geschätzte Aufwand in Personenmonaten. Die angenommene Produktivität PROD kann zwischen 4 und 50 NOP pro Personenmonat variieren – also um bis zu einem Faktor 12. Sie wird anhand der Tab. 23.7-1 abgeschätzt.

	--	-	o	+	++
Erfahrung/Fähigkeiten der Entwickler	4	7	13	25	50
Reife/Fähigkeiten der CASE-Werkzeuge	4	7	13	25	50

Tab. 23.7-1:
Ermittlung der Produktivität. Die angegebenen Werte sind NOP pro PM. PROD ergibt sich als Durchschnitt der beiden Einzelfaktoren.

Wird ein Projekt zur Erstellung eines Prototypen auf 100 *Object Points*, 25 % Wiederverwendungsanteil und eine mittlere Produktivität (PROD = 13) geschätzt, so ergibt sich:

$$\text{New Object Points NOP} = \text{OP} \cdot (100\% - \% \text{ reuse}) = 75$$

$$\text{PM} = \text{NOP} / \text{PROD} = 5,8$$

Der erwartete Aufwand beläuft sich demnach auf knapp 6 Personenmonate.

Diese Stufe ist mit größter Vorsicht zu benutzen. Bei der Softwareentwicklung steigt mit zunehmender Entwicklerzahl der Kommunikationsaufwand überproportional an. Ein lineares Modell kann daher keine brauchbare Prognose liefern. Auch ist die Annahme, dass Wiederverwendung ohne jeglichen Aufwand erfolgen kann, unrealistisch.

In der **frühen Entwurfsstufe** wird von den *Function Points* bzw. daraus abgeleiteter Codegröße ausgegangen. Die Formel zur Berechnung der **Personenmonate** PM sieht so aus:

$$\text{PM} = A \cdot \text{KLOC}^E \cdot \text{EM} + \text{PM}_m$$

KLOC (*Kilo Lines of Code*) ist dabei der erwartete Umfang der Software (wie viele 1.000 Zeilen Code), PM_m ist ein Korrekturfaktor für generierten Code, und $A=2,94$ ein konstanter Erfahrungswert, den Boehm über eine große Anzahl von Projekten ermittelt hat (alle Werte aus der letzten Kalibrierung in 2000, siehe [Boeh00]). In den Exponent E gehen Informationen über die Erfahrung mit der Domäne, die Flexibilität des Entwicklungsprozesses, die Qualität des Risikomanagements und Entwurfs, den Teamzusammenhalt und die Prozessreife ein. Diese fünf Kriterien (*Cost Driver*) werden – mit Hilfe von weiter aufgegliederten Fragestellungen – jeweils mit einem von sechs Werten zwischen sehr gut/viel und sehr schlecht/wenig beurteilt, die Bewertung per Tabelle in einen Faktor w umgesetzt.

Frühe Entwurfsstufe

23 Schätzen des Aufwands

und alle Bewertungen aufsummiert. Die Tab. 23.7-2 gibt einen Überblick über die Faktoren und Wertebereiche. Die genauen Kriterien und Zwischenwerte sind dem »COCOMO II Model Definition Manual« zu entnehmen [Boeh00]. Der Exponent ergibt sich dann wie folgt:

$$E = B + \sum w_i / 100$$

E kann derzeit (Kalibrierung 2000) mit $B=0,91$ Werte bis 1,23 annehmen und setzt für Werte größer 1 den mit der Problemgröße superlinear steigenden Aufwand um. Für sehr gute Bewertungen und damit Werte kleiner 1 kann der Aufwand allerdings auch sublinear steigen.

Kürzel	Beschreibung	Min.	Max.
PREC	Ähnlichkeit zu vorigen Projekten	0	6,20
FLEX	Flexibilität in der Entwicklung bzgl. Anforderungen, externen Schnittstellen, Zeitplan	0	5,07
RESL	(a) Risikomanagement (b) Aufwand für Architektur	0	7,07
TEAM	Teamzusammenhalt und Kooperationsbereitschaft	0	5,48
PMAT	CMM-Level oder mittlerer Erfüllungsgrad der 17 KPAs (Key Process Areas)	0	7,80

Ein Projekt wird von einer Firma bearbeitet, die schon einige sehr ähnliche Projekte abgewickelt hat (PREC = 1,24). Es gibt in dem Projekt sehr genaue Vorgaben, die kaum Abweichungen erlauben (FLEX = 4,05). Risikomanagement wird praktiziert (RESL = 2,83). Das Projektteam arbeitet an einem Ort zusammen und kennt sich bereits (TEAM = 2,19). Die Abteilung hat einen geringen Entwicklungsprozessreifegrad (PMAT = 6,24). Dann ergibt sich der Exponent E zu:

$$E = B + (\text{PREC} + \text{FLEX} + \text{RESL} + \text{TEAM} + \text{PMAT}) / 100 \\ = 0,91 + (1,24 + 4,05 + 2,83 + 2,19 + 6,24) / 100 = 1,0755$$

Im Multiplikator EM (*Effort Multiplier*) gehen Abschätzungen über Produktgüte und -komplexität, Plattformkomplexität, Fähigkeiten und Erfahrung des Personals, Zeitplan und Infrastruktur ein. Auch diese Kriterien sind zwecks besserer Bewertbarkeit weiter in Unterpunkte gegliedert. Die bis zu 6-stufige Bewertung wird in Tabellen nachgeschlagen. Der Normalfall wird jeweils mit 1,0 bewertet. Die Tab. 23.7-3 gibt auch für diese Faktoren einen Überblick. Die Faktoren für die frühe Entwurfsstufe sind jeweils fett gedruckt, darunter befinden sich die feiner aufgegliederten Faktoren, die in der Stufe nach Architekturentwurf zur Anwendung kommen. In der Tabelle sind auch jeweils die Mindest- und Höchstwerte angegeben. Sie geben einen Eindruck davon, wie groß der potenzielle Einflussgrad des jeweiligen Faktors auf den Aufwand ist. So haben die nichtfunktionalen Anforderungen insgesamt den größten Einfluss, während

23.8 Bewertung und weitere Aspekte IV

die geforderte Wiederverwendung vergleichsweise geringe Auswirkungen zeigt. Durch Multiplikation aller dieser Einzelfaktoren erhält man schließlich den Gesamtwert für EM.

Die Entwicklungszeit TDEV in Monaten ergibt sich aus folgender Formel:

$$TDEV = C \cdot PM^{(D + 0,2 \cdot (E-B))}$$

mit $C = 3,67$ und $D = 0,28$.

Die Anzahl benötigter Entwickler N ergibt sich damit zu:

$$N = PM / TDEV$$

Bei einem Projekt mit geschätzten 100.000 LOC ist der Aufwand im Beispiel bestmöglichen Fall ($E = B$, $EM = 1,0$):

$$PM = 2,94 \cdot 100^E \cdot EM = 2,94 \cdot 100^{0,91} \cdot 1,0 = 194 \text{ PM}$$

$$TDEV = 3,67 \cdot PM^{(0,28 + 0)} = 16 \text{ Monate}$$

$$N = PM / TDEV = 12 \text{ Entwickler}$$

Steht nun weniger Zeit zur Verfügung als eigentlich gebraucht wird, so können die Auswirkungen über den SCED-Parameter analysiert werden. Angenommen es stehen nur 12 statt 16 Monate zur Verfügung, so bedeutet dies eine Straffung um 25 %. Es ergibt sich (laut Tabelle) ein SCED-Faktor von 1,43, der zur Korrektur des nominalen Modells noch mit dem errechneten Aufwand multipliziert werden muss, und damit:

$$PM = 2,94 \cdot 100^{0,91} \cdot 1,43 = 278 \text{ PM}$$

$$N = PM / TDEV = 278 \text{ PM} / 12 \text{ Monate} = 23 \text{ Entwickler}$$

Die Verkürzung um 25 % erfordert also eine Verdoppelung der eingesetzten Ressourcen und bedeutet eine Steigerung des Aufwands um 43 %.

In der letzten Stufe von COCOMO II – der **Stufe nach Architekturentwurf** – werden zusätzlich die Auswirkungen erwarteter Änderungen von Anforderungen, das Ausmaß und der Aufwand für Wiederverwendung und Codegenerierung sowie weiter verfeinerte lineare Einflussfaktoren berücksichtigt. Die verfeinerten Faktoren finden sich in der Tab. 23.7-3 jeweils unter den zugehörigen zusammengefassten Faktoren der frühen Entwurfsstufe.

[BoVa08]

Stufe nach Architektur

Weiterführende Literatur

23.8 Bewertung und weitere Aspekte

Es wurden eine ganze Reihe von Schätzmethoden vorgestellt. Welches ist nun die Methode der Wahl, oder unter welchen Umständen ist welche Methode zu bevorzugen?

Eine allgemeingültige Empfehlung lässt sich dazu nicht aussprechen. Es hängt vom Zweck der Schätzung und von den für die Schätzung verfügbaren Ressourcen ab, welche Schätzung im konkreten

23 Schätzen des Aufwands

23.8 Bewertung und weitere Aspekte IV

Kürzel	Beschreibung	Min.	Max.
RCPX	Produkteigenschaften	0,49	2,72
RELY	Erforderliche Zuverlässigkeit	0,82	1,26
DATA	Datenbankgröße für vollständigen Test	0,90	1,28
CPLX	Produktkomplexität	0,73	1,74
DOCU	Qualität der Dokumentation (gesamter Software-Lebenszyklus)	0,81	1,23
RUSE	Geforderte Wiederverwendbarkeit	0,95	1,24
PDIF	Plattform	0,87	2,61
TIME	Rechenzeitbeschränkungen	1,00	1,63
STOR	Hauptspeicherbeschränkungen	1,00	1,46
PVOL	Plattform-Volatilität	0,87	1,30
PERS	Personalfaktoren	0,50	2,12
ACAP	Fähigkeiten der Analysten (Spezifikation, Entwurf)	0,71	1,42
PCAP	Fähigkeiten der Programmierer	0,76	1,34
PCON	Personalfluktuation	0,81	1,29
PREX	Erfahrung der Entwickler	0,62	1,59
AEXP	Erfahrung mit Anwendungstyp	0,81	1,22
PLEX	Erfahrung mit Plattform	0,85	1,19
LTEX	Erfahrung mit Sprache/Werkzeugen	0,84	1,20
FCIL	Projektfaktoren	0,62	1,43
TOOL	Fähigkeiten/Fortschrittlichkeit der eingesetzten Werkzeuge	0,78	1,17
SITE	Verteilte Entwicklung (Kommunikationsmittel)	0,80	1,22
SCED	Straffheit des Zeitplans	1,00	1,43

Fall geeignet ist. Im Idealfall empfiehlt es sich, **mehrere der Verfahren** anzuwenden und so zu voneinander unabhängigen Ergebnissen zu kommen. Sind diese ähnlich, so steigt das Vertrauen in die Schätzung. In der Praxis werden die Kosten für ein derartiges Vorgehen aber zu hoch sein.

Die Verfahren unterscheiden sich durch unterschiedlichen Aufwand sowie verschiedene Voraussetzungen für ihren Einsatz von einander. So kann die Experten- oder Analogie-Schätzung gut und günstig sein, wenn ein entsprechender Experte verfügbar ist bzw. Daten von sehr ähnlichen Projekten vorliegen. Die Bottom-up-Schätzung setzt die Zerlegung des Systems in Subsysteme oder Komponenten voraus, die ihrerseits sehr aufwendig oder zu spät verfügbar

sein kann. Die algorithmischen Verfahren sind mit Abstand am aufwendigsten, liefern aber auch die objektivsten Ergebnisse. Auf die Faustregeln sollte aufgrund ihrer Ungenauigkeit nur im Notfall zurückgegriffen werden.

Üblicherweise wird bei der Aufwandsschätzung nur an die Entwicklung der Software bis zur Auslieferung gedacht. Die Wartungsphase wird oft völlig außer acht gelassen, obwohl diese 60–80 % des Aufwands im Softwarelebenszyklus ausmacht [Snee97]. Daher ist es angebracht, sich auch mit der Schätzung des Wartungsaufwands zu beschäftigen. Dieser muss zumindest teilweise in die Projektkosten einkalkuliert werden: Die Fehlerbehebung macht ca. 20 % des Wartungsaufwands aus [a.a.O.]. Kosten für zusätzlich gewünschte Funktionen oder Anpassungen werden dagegen in der Regel separat beauftragt. Diese können mit einer der vorgestellten Methoden geschätzt werden. Diese Schätzung ist sogar einfacher als die eines neuen Projekts, denn es liegen in der Regel bereits umfangreiche Erfahrungen mit dem Kunden und der Domäne vor, es sind historische Daten über den Projektverlauf verfügbar, und die neuen Anforderungen sind meist recht konkret und von geringem Umfang. Andererseits schließen Wartungsarbeiten sehr viele unterschiedliche Aspekte ein, die unterschiedlich gehandhabt werden müssen [Jone07]. Und schließlich wird die Wartung um so aufwendiger, je älter das Produkt wird. Dies liegt vor allem daran, dass die ursprünglichen Entwickler nicht mehr verfügbar sind, die eingesetzten Techniken möglicherweise nicht mehr aktuell und damit bekannt sind, und sich die Qualität der Software durch ständige Wartungsarbeiten verschlechtert hat.

Eine Schätzung ist im Allgemeinen nur dann realistisch möglich, wenn es sich nicht um etwas völlig Neues handelt. Wenn man etwas völlig Neues macht, von dem noch nicht einmal bekannt ist, ob es überhaupt realisierbar ist, handelt es sich um Forschung. Hier gelten andere Regeln. Eine Möglichkeit für solche Projekte ist es, zunächst nur den Rahmen für den Aufwand bzw. die Kosten zu stecken – was genau das Ergebnis ist, ergibt sich dann erst im Laufe des Projekts in Abstimmung der beteiligten Parteien. Es bietet sich bei diesem Modell eine enge Zusammenarbeit und ein inkrementeller Entwicklungsprozess an. Diese Vorgehensweise ist auf eine vertrauensvolle Zusammenarbeit zwischen Auftraggeber und Entwicklungsorganisation angewiesen. Sie wird häufig zwischen Forschungseinrichtungen und Unternehmen praktiziert. Ähnliche Ideen werden beispielsweise auch beim *Extreme Programming* angewandt.

[Snee03], [BSB08]

Wartung

Forschung

Weiterführende Literatur

23 Schätzen des Aufwands

23.9 Zusammenfassung

In den Beschreibungen der verschiedenen Schätzverfahren wurde deutlich, dass sie alle auf **Erfahrung** basieren. Es ist also notwendig, Erfahrungen zu sammeln. Die Schätzung hängt stark von der Entwicklungsorganisation ab, z.B. von ihrer Produktivität, ihren Prioritäten und Zielen. Konkret bedeutet dies, dass die Messung und Auswertung von Projekten der eigenen Organisation nötig ist, damit diese als Basis für zukünftige Schätzungen dienen können. Ist dies nicht der Fall, so ist man auf die undokumentierte Erfahrung von Experten angewiesen.

Gute Schätzungen sind *nicht* billig zu haben. Verfahren wie die Function-Points-Zählung sind teuer, und die Zählung bereits abgeschlossener Projekte ist zunächst einmal eine Investition in die Qualität zukünftiger Schätzungen. Und auch eine Expertenschätzung bindet die Produktivität hochbezahlter Mitarbeiter. Billige Alternativen wie die Faustregeln liefern entsprechend auch nur sehr vage Ergebnisse.

Eine Erkenntnis, die ausgenutzt werden sollte, ist die steigende Genauigkeit der Schätzung mit fortschreitendem Projektverlauf. Die initiale Schätzung bzw. die ihr zugrunde liegenden Annahmen sollten daher regelmäßig überprüft und angepasst werden, um Abweichungen vom Zeitplan und damit drohende Budgetüberschreitungen rechtzeitig erkennen zu können. Diese Maßnahme kann somit das Risikomanagement unterstützen.

Insgesamt lässt sich feststellen, dass systematische Aufwands schätzung sehr nützlich sein kann. Um sie zu ermöglichen, muss allerdings kontinuierlich ein gewisser Aufwand betrieben werden, um Projektdaten zu erfassen und eine Erfahrungsdatenbank aufzubauen. Mittels dieser Daten lassen sich dann neue Projekte recht verlässig abschätzen – wenn sie nicht völlig anders als alle bereits abgeschlossenen Projekte sind.

24 Anforderungen priorisieren

Bei der Anforderungsermittlung werden in der Regel *viele* Anforderungen aufgestellt. Aber nicht jede Anforderung ist gleich wichtig.

Nach welchen Kriterien können Prioritäten für Anforderungen vergeben werden? Frage

Ein Kriterium ist sicher die **Wichtigkeit** einer Anforderung. Wichtigkeit kann aber Verschiedenes bedeuten: Wichtigkeit bezogen auf die Funktionalität des Systems, Wichtigkeit bezogen auf die Benutzungsfreundlichkeit, Wichtigkeit bezogen auf die frühzeitige Verfügbarkeit usw. Antwort

Anforderungen können auch nach Kosten, Volatilität, Risiko hinsichtlich der Akzeptanz des Systems, Schaden bezogen auf die Nachteile, wenn die Anforderung nicht berücksichtigt wird, usw. gewichtet werden. Eine Priorisierung anhand eines Kriteriums ist in der Regel relativ einfach möglich. Werden mehrere Kriterien berücksichtigt, dann steigt der Aufwand mit jedem zusätzlichen Kriterium.

Klassifikation nach einem Kriterium

Häufig werden Anforderungen nach einem Kriterium priorisiert. In [IEEE830, S. 13] wird als Kriterium die **Notwendigkeit** (*necessity*) mit folgenden Ausprägungen vorgeschlagen:

- **Essenziell** (*essential*): Die Software wird nicht akzeptiert, wenn die Anforderung nicht in der geforderten Weise realisiert wird.
- **Bedingt notwendig** (*conditional*): Die Anforderung wertet die Software auf. Wird sie nicht realisiert, dann wird die Software aber nicht unakzeptabel.
- **Optional** (*optional*): Die Realisierung der Anforderung kann wertvoll sein, muss es aber nicht. Die Anforderung gibt dem Auftragnehmer die Möglichkeit, über die vorhandenen Anforderungen hinauszugehen.

Erfahrungen haben gezeigt, dass die Verwendung dieser Ausprägungen dazu führt, dass die meisten Anforderungen mit **essenziell** gekennzeichnet werden, während optionale Anforderungen nur selten vorkommen.

Die Kano-Klassifikation

N. Kano hat festgestellt, wie Kundenwünsche und Produkteigenschaften zusammenhängen [KTS+84]. Die Produkteigenschaften werden in Kategorien eingeteilt: