

# Management von Softwaresystemen



## *Aufwands- und Kostenschätzung*

**Betreuer:** Bernhard Schätz

**Bearbeiterin:** Meltem Akyazi

# Aufwands- und Kostenschätzung

Bevor mit der eigentlichen Durchführung eines Projektes begonnen werden kann, ist es im Normalfall notwendig, eine möglichst genaue Schätzung des benötigten Aufwands durchzuführen. Die Aufwandsschätzung ist ein wichtiger Teil im Rahmen der Projektplanung. Sie dient der Abschätzung der für die Projektdurchführung benötigten Kapazitäten und ist daher ein wesentliches Element bei der Bestimmung der Projektkosten. Die Fähigkeit, den Entwicklungsaufwand abschätzen zu können, ist eine wesentliche Voraussetzung, um Softwareprojekte erfolgreich durchzuführen bzw. mit Softwareentwicklung Geld zu verdienen.

Um eine Schätzung über den Aufwand und und Zeit eines Projektes ermitteln zu können, müssen erst mal die folgenden Fragen beantwortet werden:

- 1.** Wie viel Aufwand ist für die Erledigung einer Aufgabe erforderlich?
- 2.** Wie viel Zeit ist für die Erledigung erforderlich?
- 3.** Wie hoch sind die für eine Aufgabe anfallenden Gesamtkosten?

Projektaufwandsschätzung und -zeitplanung erfolgen gemeinsam. Allerdings sollten die Kosten zu Beginn eines Projekts grob eingeschätzt werden können, bevor detaillierte Pläne aufgestellt werden. Diese Schätzungen dienen zum Festlegen eines Projektbudgets bzw. zur Ermittlung des Softwarepreises.

Drei Parameter sind für die Berechnung der Gesamtkosten eines Softwareentwicklungsprojekts von Bedeutung:

- Hardware- und Softwarekosten, einschließlich Wartung
- Reise- und Schulungskosten
- Personalkosten

Die Hard- und Software, die eine ausreichende Leistung für die Softwareentwicklung erbringen, sind relativ günstig. Die Reise- und Schlungskosten lassen sich durch die Verwendung von elektronischen Kommunikationsmitteln im großen Maße verringern. Den größten Kostenfaktor stellen hierbei die Personalkosten dar.

Kostenfaktoren wie

- 1.** Kosten für die Bereitstellung, Beheizung und Beleuchtung der Büroräume
- 2.** Kosten für unterstützendes Personal
- 3.** Kosten für Netz und Kommunikation
- 4.** Kosten für zentrale Einrichtungen wie Bibliotheken, Erholungseinrichtungen
- 5.** Kosten für Sozialversicherungen, Krankenversicherungsanteile und Mitarbeiterzuwendungen wie Erfolgsbeteiligungen und Betriebsrenten

sind auch Teil des Gesamtaufwands.

Bei der Preisfestlegung für ein Projektangebot für den Kunden sollten neben den eigentlichen Kosten und dem erhofften Gewinn umfangreichere, betriebliche, wirtschaftliche, politische und geschäftliche Überlegungen in Betracht gezogen werden. Die zu berücksichtigenden Faktoren sind wie folgt aufgelistet:

- **Marktchancen:** Eine Entwicklungsorganisation könnte einen niedrigen Preis anbieten, weil sie in ein neues Segment des Softwaremarkts eindringen möchte.
- **Unsicherheit der Aufwandsschätzung:** Bestehen in einer Organisation Unsicherheiten bei der Aufwandsschätzung, könnte sie beim Festlegen ihres Preises einige unvorhergesehene Ausgaben berücksichtigen, so dass dieser über dem üblichen Preis liegt.
- **Vertragsbedingungen:** Ein Kunde ist möglicherweise bereit, dem Auftragnehmer das Eigentum am Quellcode zu überlassen, damit dieser ihn in anderen Projekten erneut verwenden kann und der in der Rechnung gestellte Preis niedriger ausfällt.
- **Unbeständigkeit der Anforderungen:** Ist eine Änderung der Anforderungen wahrscheinlich, könnte eine Firma den Preis senken, um einen Auftrag zu erhalten. Nach der Auftragserteilung könnten für geänderte Anforderungen höhere Preise in Rechnung gestellt werden.
- **Wirtschaftliche Bonität:** Eine Entwicklungsfirma in finanziellen Schwierigkeiten könnte ihre Preise senken, um einen Auftrag zu erhalten.

## Produktivität

Um die Produktivität in einem Fertigungssystem messen zu können, wird die produzierte Stückzahl durch die Anzahl der für die Produktion erforderlichen Personenstunden geteilt. Die Produktivität wird von vielen verschiedenen Faktoren beeinflusst. Einige von denen sind:

- Erfahrung im Anwendungsgebiet
- Prozessqualität
- Projektgröße
- Technische Unterstützung
- Arbeitsumgebung

Zur Produktivitätsschätzung werden einige Softwareattribute verwendet, deren Wert an der Entwicklung des erforderlichen Gesamtaufwands in zwei Arten von Maßen beteiligt.

**1. Größenspezifische Maße:** Diese beziehen sich auf die Größe der sich aus einer Aufgabe ergebenden Softwaregröße. Die meisten Modelle basieren auf dem geschätzten Umfang des zu erstellenden Software-Produktes in »Anzahl der Programmzeilen« bzw. in **Lines of Code** (LOC), was das gebräuchlichste größenspezifische Maß ist.

- Bei höheren Sprachen werden alle Vereinbarungs- und Anweisungszeilen geschätzt.
- Der geschätzte Umfang wird durch einen Erfahrungswert für die Programmierproduktivität (in LOC) eines Mitarbeiters pro Monat geteilt.

Je nach Programmiersprache ändert sich die Produktivität. Je anschaulicher eine Programmiersprache ist, desto geringer ist die sichtbare Produktivität. Dies kann man in einem Beispiel erläutern.

## Beispiel:

Analyse Entwurf Codierung Testen Dokumen-  
tation

Assemblercode	3 W	5 W	8 W	10 W	2 W
Höhere Programmier- sprache	3 W	5 W	8 W	6 W	2 W

	<u>Größe</u>	<u>Aufwand</u>	<u>Produktivität</u>
Assemblercode	5000 Zeilen	28 W	714 Zeilen/Monat
Höhere Programmier- sprache	1500 Zeilen	20 W	300 Zeilen/Monat

Wie man auch aus dem Beispiel ablesen kann, ist ein System, das in 5000 Zeilen Assemblercode geschrieben werden könnte, könnte in 1500 Zeilen Code einer höheren Programmiersprache geschrieben werden. Während sich die Produktivität des Assemblerprogrammierers auf 714 Zeilen/Monat beläuft, ist die des Programmierers der hochentwickelten Sprache weniger als die Hälfte. Damit wird auch gezeigt, dass sowohl die Entwicklungskosten als auch die Entwicklungszeit für das in einer höheren Programmiersprache programmierte System geringer sind.

**2. Funktionsspezifische Maße:** Diese beziehen sich auf die Gesamtfunktionalität der gelieferten Software. Die Produktivität an diesen Maßen entspricht der innerhalb eines bestimmten Zeitraums produzierten nutzbringenden Funktionen. Function Points und Object Points sind die bekanntesten Maße dieser Art.

• **Function Points:** Diese Methode wurde von A.J. Albrecht von IBM begründet. Die Idee dabei war, dass die Größe eines Informationssystems durch eine geeignete Funktionalität gemessen werden kann. Bei der Function-Point-Methode wird der Umfang nicht durch LOC ausgedrückt, sondern aus einer Kombination von Programmeigenschaften ermittelt. Die Gesamtanzahl der Function Points eines Programms wird berechnet, indem man erst die folgenden Programmfunktionen misst oder schätzt:

- Externe Ein- und Ausgaben
- Benutzerinteraktionen
- Externe Schnittstellen
- Vom System verwendete Dateien

Als zweites wird jede Funktion in die Klassen „einfach, mittel oder komplex“ eingestuft und mit einem relativen Wert versehen, der zwischen 3 für einfache externe Eingaben und 15 für komplexe interne Dateien schwankt. Dafür können entweder die von Albrecht vorgeschlagenen gewichteten Werte oder auf firmeninternen Erfahrungen basierende Werte verwendet werden.

<b>Gewichtungsfaktoren</b>	<b>Komplexität niedrig</b>	<b>Komplexität mittel</b>	<b>Komplexität hoch</b>
Dateneingaben	3	4	6
Datenausgaben	4	5	7
Benutzerinteraktionen	3	4	6
Externe Schnittstellen	5	7	10
Vom System verwendete Dateien	7	10	15

Der sogenannte Function-Point-Rohwert (**U**nadjusted **F**unction **P**oint **C**ount ) wird berechnet, indem die einzelnen Häufigkeiten mit dem geschätzten Gewicht multipliziert und alle Werte aufsummiert werden.

$$\text{UFC} = \sum (\text{Anzahl der Elemente eines bestimmten Typs}) * \text{Gewicht}$$

Dieser Wert wird mit den Faktoren, wie Wiederverwendbarkeit, Installation und Konvertierung, Anpassbarkeit, Leistungsfähigkeit usw., für die Projektkomplexität multipliziert, um die endgültige Funktionspunktzahl zu errechnen. Daraus ergibt sich der Wert von „Adjusted Function Point Count“:

$$\begin{aligned} \text{Adjusted Function Point Count} \\ = \text{Korrekturfaktor} * \text{„Unadjusted Function Point Count“} \end{aligned}$$

Eigentlich müsste der Korrekturfaktor = 1 sein, wenn alle Faktoren durchschnittlichen Einfluss haben. Das wäre der Fall, wenn der durchschnittliche Einfluss mit 2.5 statt mit 3 bewertet würde. Aus praktischen Gründen hat man sich aber für den Wert 3 entschieden, was dazu führt, dass der Korrekturfaktor bei lauter durchschnittlichen Einflüssen den Wert 1.07 hat.

Der wesentliche Vorteil der Methode ist, dass die erforderliche Schätzung der Anzahl erforderlicher Befehle durch die Analyse der erforderlichen Funktionen ersetzt wird. Und diese Feststellung liegt weit vor dem Zeitpunkt, wo Anzahl der Befehle feststeht. Eine Schätzung ist ungleich leichter möglich, und eine genaue Kostenermittlung zum Zeitpunkt der Systemanalyse reicht für die Wirtschaftlichkeitsüberlegungen im Allgemeinen aus. Denn bis zu diesem Zeitpunkt sind noch nicht zu viele Kosten angefallen.

Der Nachteil dieser Methode ist, dass die Function Points eines Programms vom Schätzer abhängen. Verschiedene Personen haben unterschiedliche Vorstellungen von Komplexität. Je nach Beurteilung des Schätzenden bestehen große Unterschiede in der Zählung der Function Points. Aber trotz ihrer Mängel vertreten viele Benutzer die Ansicht, diese Methode in der Praxis effektiv ist.

## **Zusammenfassung**

### **Vor- und Nachteile der *Function Point*-Methode:**

#### **Vorteile:**

- + Ausgangspunkt: Produkthanforderungen, nicht LOC
- + Anpassbar an verschiedene Anwendungsbereiche (Änderung der Kategorien)
- + Anpassbar an neue Techniken (Änderung der Einflussfaktoren und der Einflussbewertung)
- + Anpassbar an unternehmensspezifische Verhältnisse (Änderung der Einflussfaktoren, der Einflussbewertung und der Gewichtungsfaktoren)
- + Verfeinerung der Schätzung entsprechend dem Entwicklungsfortschritt (iterative Methode)

- + Erste Schätzung bereits zu einem frühen Zeitpunkt möglich (Planungsphase).
- + Festgelegte methodische Schritte
- + Leicht erlernbar
- + Benötigt nur einen geringen Zeitaufwand
- + Gute Transparenz
- + Gute Schätzgenauigkeit
- + Werkzeugunterstützungen verfügbar.

### **Nachteile:**

- Es kann nur der Gesamtaufwand geschätzt werden
- Umrechnung auf einzelne Phasen muss mit der Prozentsatzmethode erfolgen
- In der Originalform von Albrecht personalintensiv und nicht automatisierbar
- Zu stark funktionsbezogen
- Q-Anforderungen werden nicht berücksichtigt
- Mischung von Projekt- und Produkteigenschaften bei den Einflussfaktoren
- Ursprüngliche Einflussfaktoren heute überholt
- Neigt zur Unterschätzung, da Anforderungen oft lückenhaft
- Methodische Mängel.

### **Aufwandschätzung mit Function Points**

Sollen Function Points zur Aufwandschätzung verwendet werden, so muss der Aufwand pro Function Point bekannt sein.

Faustregeln zur Aufwandberechnung:

$$\mathbf{Kalenderzeit [Monate] = FP^{0.4}}$$

$$\mathbf{Anzahl Mitarbeiter = FP/150}$$

$$\mathbf{Aufwand [Personenmonate] = Kalenderzeit * Anzahl Mitarbeiter}$$

Sollen projektspezifische Faktoren (z.B. die Fähigkeiten der MitarbeiterInnen) berücksichtigt werden, so müssen zusätzliche Korrekturfaktoren verwendet werden.

### Weitere Faustregeln mit Fps:

- 1 Function Point = 320 Statements in Basic Assembler
- 1 Function Point = 213 Statements in Makro Assembler
- 1 Function Point = 128 Statements in C
- 1 Function Point = 107 Statements in COBOL
- 1 Function Point = 107 Statements in FORTRAN
- 1 Function Point = 80 Statements in PL/I
- 1 Function Point = 71 Statements in ADA83
- 1 Function Point = 53 Statements in C++
- 1 Function Point = 15 Statements in Smalltalk

~ Für prozedurale Sprachen: 1 FP = 100 Statements

~ Für objekt-orientierte Sprachen: 1 FP = 20 Statements

- **Dokumentationsumfang:** Anzahl Seiten =  $FP^{1.15}$
- **Schleichende Anforderungen** wachsen während der Design bis zur Codierungsphase durchschnittlich um 2% pro Monat.
- **Anzahl benötigter Test Cases** =  $FP^{1.2}$
- **Anzahl benötigter Personen für die Wartung** der Software =  $FP/750$
- **Object Points:** Bei der Anzahl der Object Points eines Programms handelt es sich um eine gewichtete Schätzung folgender Faktoren:
  1. *Die Anzahl der angezeigten Bildschirmmasken.* Je nach Komplexität der Masken ändern sich die Anzahl der Object Points zwischen 1 und 3.
  2. *Die Anzahl der erzeugten Berichte.* Je nach Komplexität der Berichte ändern sich die Anzahl der Object Points zwischen 2 und 8.
  3. *Die Anzahl der 3GL-Module, die zur Ergänzung des 4GL-Codes verwendet werden müssen.* Jedes 3GL-Modul zählt 10 Object Points.

Im Vergleich zu Function Points sind die Object Points bei einer groben Softwarespezifikation leichter zu schätzen.

## Schätzverfahren

Der Aufwand, der für die Entwicklung eines Softwaresystems erforderlich ist, ist nicht leicht zu schätzen. In einem frühen Projektstadium sind weder die Fähigkeiten der Mitarbeiter noch die zu verwendende Entwicklungstechnologie bekannt. Daher ist sehr schwierig, in einem frühen Projektstadium eine Schätzung der Systementwicklungskosten vorzunehmen.

Um den Aufwand und die Kosten für eine Software schätzen zu können, werden eines oder mehrere der unten beschriebenen Verfahren verwendet:

- *Analogieschätzung*: Die Kosten für ein neues Produkt werden analog zu den auf dem Anwendungsgebiet bereits abgeschlossenen Projekten geschätzt.
- *Parkinsons Gesetz*: Parkinsons Gesetz stellt fest, dass sich die Arbeit immer so ausdehnt, dass sie genau die Zeit braucht, die man für sie erübrigen kann. Die Kosten werden nach den verfügbaren Ressourcen und nicht nach einer konkreten Schätzung ermittelt. Muss die Software in zwölf Monaten geliefert werden und stehen fünf Mitarbeiter zur Verfügung, wird der erforderliche Aufwand auf 60 Personenmonate geschätzt.
- *price to win*: Die Softwarekosten werden auf das dem Kunden für das Projekt zur Verfügung stehende Budget geschätzt. Der geschätzte Aufwand hängt vom Budget des Kunden und nicht von der Funktionalität der Software ab.
- *Expertenbeurteilung*: Mehrere auf dem Gebiet erfahrene Experten liefern eine Schätzung der Projektkosten ab. Diese Schätzungen werden miteinander verglichen und so lange wiederholt, bis eine Einigung über die Software erzielt ist. Der Vorteil dieses Verfahrens ist, dass es einfach und billig ist. Die durch dieses Verfahren erzielte Schätzung kann aber sehr ungenau sein. Die Qualität der Schätzung ist von der Erfahrung der Schätzer und der Qualität der Erfahrungsdaten abhängig.

Bei der Schätzung der Softwarekosten müssen die Unterschiede zwischen früheren und künftigen Projekten berücksichtigt werden. Die Einführung neuer Methoden und -verfahren, wie

~ Objektorientierte anstelle funktionsorientierter Entwicklung

~ Client/Server-Systeme anstelle von Großrechnersystemen

~ Verwendung von Standardsoftwarekomponenten anstelle der Entwicklung von Komponenten

~ Entwicklung zur und unter Wiederverwendung anstelle von Neuentwicklung aller Systemteile

~ Verwendung von CASE-Werkzeugen und Programmgeneratoren anstelle einer Softwareentwicklung ohne Unterstützung

erschweren den Experten die genauere Schätzung des Aufwands.

- *Algorithmisches Aufwandsmodell*: Es wird ein Modell entwickelt, das historische Aufwandsdaten verwendet, die eine bestimmte Softwaremetrik (normalerweise die Größe) zu den Projektkosten in Bezug setzt.

### **Die Aufwandsschätzung kann mit der Verwendung eines Top-down- oder Bottom-up Ansatzes realisiert werden.**

- **Der Top-down Ansatz**: Dieser Ansatz setzt auf Systemebene ein. Bei der Top-Down Schätzung wird der Gesamtaufwand für ein Projekt mit Hilfe mathematischer Algorithmen geschätzt und dann auf die Arbeitsphasen und -pakete des Projekts verteilt. Die Kosten für die auf Systemebene anfallenden Aufgaben, wie zum Beispiel Integration, Konfigurationsmanagement und Dokumentation, finden dabei Berücksichtigung.
- **Der Bottom-up Ansatz**: Dieser Ansatz beginnt auf Komponentenebene. Bei der Bottom-

up Schätzung werden für jedes Arbeitspaket eines Projekts die Aufwände getrennt ermittelt. Der Gesamtprojektaufwand ergibt sich als Summe dieser Einzelaufwände. Dieser Ansatz beginnt auf Komponentenebene.

Die Nachteile des Bottom-up Ansatzes sind gleichzeitig die Vorteile des Top-down Ansatzes und umgekehrt. Während die Top-down Schätzung die Kosten für schwierige, mit bestimmten Komponenten verbundene technische Probleme unterschätzen, finden sie bei der Bottom-up Schätzung Berücksichtigung. Allerdings werden die Kosten für systembezogene Tätigkeiten, zum Beispiel die Integration, beim Bottom-up Ansatz unterschätzt. Außerdem ist die Bottom-up Schätzung aufwendiger als die Top-down Schätzung.

Jedes Schätzverfahren verfügt über eigene Stärken und Schwächen. Die Ergebnisse verschiedener Verfahren müssen miteinander verglichen werden. Wenn die Ergebnisse sich voneinander gravierend unterscheiden, müssen die Schätzungen wiederholt werden, bis eine gute Annäherung der Ergebnisse gefunden wird.

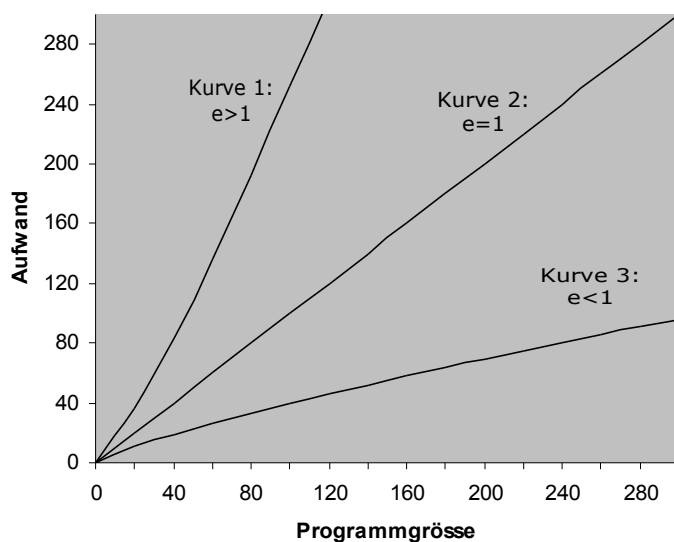
## Algorithmische Schätzverfahren

Ein algorithmisches Kostenmodell bedient sich zur Ergebnisermittlung immer einer Formel bzw. eines Formelgebildes, dessen Struktur und Konstanten empirisch und teilweise mit mathematischen Methoden, wie z.B. der Regressionsanalyse, bestimmt worden sind. So wird aufgrund empirisch gefundener Korrelation die Abhängigkeit des Aufwands von einer bestimmten Ergebnisgröße als Kurvenverlauf dargestellt, aus dem dann für ein künftiges Entwicklungsvorhaben - unter Berücksichtigung bestimmter Einflussparameter - der zu erwartende Aufwand abgeleitet werden kann.

Die meisten algorithmischen Schätzmodelle verwenden eine Exponentialkomponente. Daraus kann man folgern, dass die Kosten normalerweise nicht linear mit dem Projektumfang steigen. Der Aufwand für die Erstellung von Software steigt mit wachsender Produktgröße überproportional an. Je größer das Projekt ist, desto mehr Schnittstellen sind zu programmieren. Die Anzahl an Mitarbeitern ist dann auch dementsprechend zu erhöhen. Das hat zur Folge, dass es größere Administration- und Kommunikationsaufwand entsteht.

Aufwand zur Beherrschung der wachsenden Komplexität wächst überproportional. Je größer der Projektumfang ist, desto mehr Kosten entstehen durch die Unkosten der Kommunikation in größeren Teams, das komplexere Konfigurationsmanagement, die schwierige Systemintegration usw.

Der folgende Graph verdeutlicht das Verhalten zwischen dem Aufwand und der Projektgröße.



Die Schätzung der Softwarekosten kann mit Hilfe eines Algorithmus wie folgt ausgedrückt werden:

$$\text{Aufwand} = A * \text{Größe}^B * M$$

A ist ein konstanter Faktor, der von den eigenen Praktiken der Organisation und der Art der von ihr entwickelten Software abhängt. Größe kann entweder als eine Schätzung der Größe des Softwarecodes oder als eine Schätzung der Funktionalität nach Function oder Object Points interpretiert werden. Der Wert von B liegt zwischen 1 und 1,5. M ist ein Multiplikator, der sich aus einer Kombination verschiedener Prozess-, Produkt- und Entwicklungsmerkmale ergibt.

Die Schätzung der *Größe* ist zu Beginn der Arbeit, wo nur die Spezifikation zur Verfügung steht, sehr schwierig. Die Verwendung der Function- und Object-Point-Schätzungen anstatt der Codegröße weisen auch Ungenauigkeiten auf. Die Schätzung von B und M schwankt auch je nach Hintergrundwissen und Erfahrungen der einzelnen Personen.

Die in den meisten algorithmischen Kostenmodellen verwendete Grundmetrik ist die Anzahl der Quellcodezeilen eines Systems. Die Größe kann durch Umwandeln der Function Points in Codegröße oder durch den Vergleich mit den anderen ähnlichen schon abgeschlossenen Projekten oder einfach durch die Beurteilung erfahrener Entwickler abgeschätzt werden.

Die Codegröße lässt sich von den Faktoren wie die Verwendung von kommerziellen Datenbanken für eine komplexe Datenverwaltung oder die Wahl der verwendeten Programmiersprache beeinflussen. Zum Beispiel eine Sprache wie Java hat mehr Umfang an der Anzahl des Programmcodes im Vergleich zu C.

Genauigkeit der Schätzung des Aufwands ist abhängig von

- der Genauigkeit der Eingangsgrößen
- der Qualität der Kalibrierung (Anpassung an die jeweilige Entwicklungsumgebung)

Werden zur Schätzung der Projektkosten Algorithmusmodelle verwendet, sollten sie anhand bekannter Daten an den zu beurteilenden Projekttyp angepasst werden, wobei die gewonnenen Ergebnisse dennoch mit Vorsicht zu genießen sind. Anstelle einer Einzelschätzung sollte der Schätzer mehrere Schätzungen für den ungünstigsten, den erwarteten und den günstigsten Fall vornehmen und dabei immer dieselbe Kostenberechnungsformel anwenden.

## Das COCOMO-Modell (Constructive Cost Model)

Dieses Modell geht von der Abschätzung der erforderlichen Befehle für die zu erstellenden Programme aus. Nur differenziert es wesentlich feiner nach den erforderlichen Qualitätsmerkmalen und hebt auch die Leistungsfähigkeit der Mitarbeiter ab. Das COCOMO-Modell wurde abgeleitet, indem eine große Anzahl Daten von Softwareprojekten gesammelt und die Daten anschließend analysiert wurden, um die Anzahl der Quellcodezeilen und Formeln zu bestimmen, die mit den Beobachtungen am besten in Klang standen. COCOMO beruht auf einer Kombination von Gleichungen, statistischen Modellen, und Schätzungen von Parameterwerten. Die erste Version des COCOMO-Modells ist ein Drei-Stufen-Modell, in dem die Stufen die Ausführlichkeit einer Aufwandsschätzung widerspiegeln.

Ausgangspunkte der Schätzung sind:

- Ermittlung der LOC/KDSI
- Ermittlung der Berechnungsfaktoren

## **1. Stufe: Basic COCOMO**

Den Entwicklungsaufwand errechnet man über eine Basisgleichung (Aufwand in Personenmonaten und benötigte Projektdauer ausgehend von einer Schätzung der Produktgröße), ohne dass irgendwelche Einflussparameter in die Aufwandsbestimmung miteinbezogen werden. Die Produktgröße wird dabei durch die Anzahl an Codezeilen (LOC) angegeben. Die Hauptformel lautet:

$$\text{Aufwand[in Personenmonaten]} = A * \text{Größe[in KDSI]}^B$$

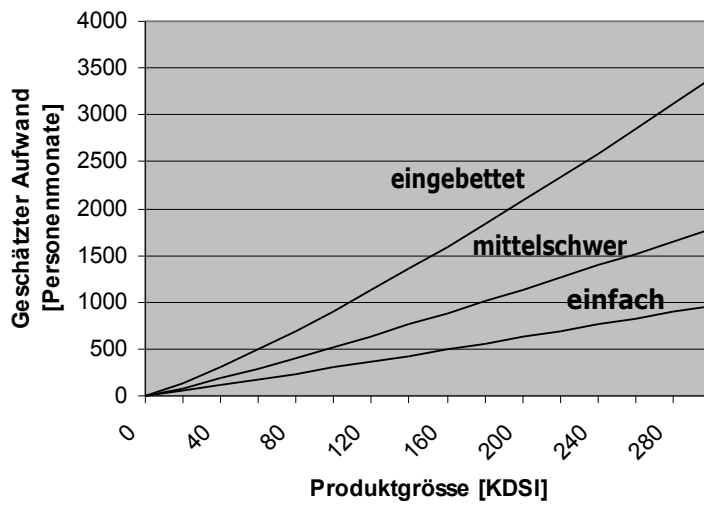
$$\text{Benötigte Projektdauer[in Monaten]} = C * \text{Aufwand}^D$$

Je nach Komplexität des Projekts ändern sich die Werte von A, B, C und D wie folgt:

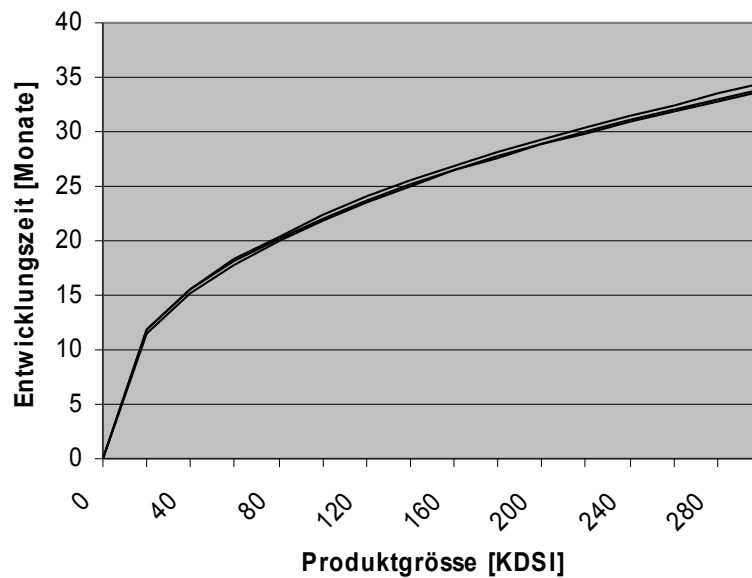
<b>Komplexität des Projekts</b>	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>
<b>Einfach („Organic Mode Projects“)</b>	<b>2.4</b>	<b>1.05</b>	<b>2.5</b>	<b>0.38</b>
<b>Mittelschwer (Semi-detached Mode Projects“)</b>	<b>3.0</b>	<b>1.12</b>	<b>2.5</b>	<b>0.35</b>
<b>Eingebettet („Embedded Mode Projects“)</b>	<b>3.6</b>	<b>1.2</b>	<b>2.5</b>	<b>0.32</b>

- **Einfache Projekte:**
  - Kleine Projektteams in vertrauter Umgebung
  - Stabile Entwicklungsumgebung, wenige Änderungen an der Hardware/Software-Umgebung
  - Geringer Zeitdruck
- **Mittelschwere Projekte:**
  - Komplexe Projekte, bei denen die Teamglieder über begrenzte Erfahrungen mit entsprechenden Systemen verfügen könnten.
- **Eingebettete Projekte:**
  - Entwicklung unterliegt starken Beschränkungen
  - Systemumgebung verändert sich sehr stark
  - Hoher Termindruck

$$\text{Aufwand [in Personenmonaten]} = A * \text{Größe [in KDSI]}^B$$



$$\text{Benötigte Projektdauer [in Monaten]} = C * \text{Aufwand}^D$$



## 2. Stufe: Intermediate COCOMO

### **Vorgehen:**

- Der „Entwicklungsmodus“ des Projekts identifizieren
- Die Größe des Projekts (in Anzahl Codezeilen) schätzen
- Die 15 „Cost Driver Attributes“ bestimmen

- Den Projektaufwand in Personenmonaten und die Projektdauer in Monaten berechnen

Das Zwischenmodell berücksichtigt die 15 Kostenfaktoren, allerdings ohne Differenzierung nach Entwicklungsphasen. Dieses Schätzmodell ist daher nicht mehr rein parametrisch, weil in die Bestimmung der Einflussgrößen rechnerisch nicht belegbare Expertenerfahrungen einfließen. Für jeden Kostenfaktor wird ein Erfahrungswert angegeben, der sich zwischen den vorgegebenen Min- und Max-Werte bewegt.

Die Berechnungsformel lautet wie folgt:

$$\text{Aufwand[in Personenmonaten]} = (K1 * \dots * K15) * \text{Aufwand[aus Basic COCOMO]}$$

gleichbedeutend mit:

$$\text{Aufwand[in Personenmonaten]} = A * \text{Größe[in KDSI]}^B * M$$

wobei  $M = K1 \dots K15$  = das Produkt der Kostenfaktoren ist.

#### **Vorteile:**

- Transparent („man versteht das Modell“)

#### **Nachteile:**

- Modell ist anfällig auf falsches Zuweisen des Entwicklungsmodus.
- Cost Driver Attributes und insbesondere die Software-Grösse müssen gut geschätzt werden.

#### ***Einige dieser Kostenfaktoren sind wie folgt aufgelistet:***

- **Produkt-abhängige Attribute**

-RELLY: Erforderliche Systemzuverlässigkeit zwischen 0.75-1.40

- **Rechner-abhängige Attribute**

-VIRT: Software Entwicklungsumgebung zwischen 0.87-1.30

- **Personen-abhängige Attribute**

-LEXP: Erfahrung mit der Programmiersprache zwischen 0.95-1.14

- **Projektumgebung-abhängige Attribute**

-MODP: Erfahrung mit modernen Programmiermethoden zwischen 0.82-1.24

### **3. Stufe: Detailed COCOMO**

Das Detailmodell berücksichtigt 15 Einflussfaktoren sowie die Abweichungen der anteilmäßigen Aufwände einzelnen Entwicklungsphasen.

Im Detailed COCOMO können die Cost Driver Attributes auf einzelne Phasen oder einzelne Subsysteme anstatt nur auf das System als Ganzes angewendet werden.

Das Schätzen der benötigten Projektdauer, wird wie auch schon beim Intermediate COCOMO

mit der selben Formel wie Basic COCOMO berechnet. Der Aufwand für die Implementierung und Anwendung des Detailed COCOMO Modells ist deutlich höher als bei Intermediate COCOMO.

## COCOMO 2.0

Im Bereich der Softwareentwicklung ergeben sich immer wieder Änderungen. Software kann durch die Zusammensetzung wiederverwendbarer Komponenten und die Verknüpfung dieser Komponenten über eine Skriptsprache erstellt werden. Die Entwicklung von Prototypen und ihre Weiterentwicklung sind weit verbreitete Ablaufmodelle. In vielen Fällen werden käufliche Subsysteme eingesetzt. Vorhandene Software wird umgestellt, um neue Software herzustellen. Für die meisten Vorgänge im Softwareprozess steht jetzt außerdem Unterstützung durch CASE-Werkzeuge zur Verfügung. Um diesen Änderungen Rechnung zu tragen, kennt das Modell COCOMO 2 verschiedene Ansätze der Softwareentwicklung, wie zum Beispiel das Entwickeln mit Prototypen, die Entwicklung auf der Basis der zusammengestellter Komponenten, die Verwendung von 4 GLs usw.

COCOMO 2 besteht aus 3 Stufen:

### 1. Die frühe Prototypenstufe:

Die frühe Prototypenstufe wurde eingeführt, um die Aufwandsschätzung für Prototypen und Projekte zu unterstützen, bei denen die Software aus einer Kombination vorhandener Komponenten entwickelt wurde.

- Die Größenschätzungen basieren auf Object Points, die durch eine Standardzahl für die geschätzter Produktivität dividiert wird.
- Zur Schätzung des erforderlichen Aufwands wird eine einfache Größen-/Produktivitätsformel verwendet.

In dieser Phase ist die Wiederverwendung üblich, so dass die Anzahl der in der Schätzung des Zeitplans verwendeten Object Points so angepasst wird, dass der erwartete prozentuale Wiederverwendungsanteil (%reuse) Berücksichtigung findet.

$$PM = (NOP * (1 - \%reuse / 100)) / PROD$$

Bei *PM* handelt es sich um den Aufwand in Personenmonaten, bei *NOP* um die Anzahl der Object Points und bei *PROD* um die Produktivität.

### 2. Die frühe Entwurfsstufe:

- Diese Stufe entspricht der abgeschlossenen Festlegung der Systemanforderungen und einem ersten Entwurf
- Die Schätzungen basieren auf Function Points, die dann in die Anzahl der Quellcodezeilen umgewandelt werden.

$$PM = A * Größe^B * M + PM_m$$

$$M = PERS * RCPX * RUSE * PDIF * PREX * FCIL * SCED$$

$$PM_m = (ASLOC * (AT / 100)) / ATPROD$$

Der Multiplikator  $M$  beruht auf einer vereinfachten Reihe von Projekt- und Prozessfaktoren, welche die Produktzuverlässigkeit und -komplexität ( $RCPX$ ), den erforderlichen Wiederverwendungsgrad ( $RUSE$ ), den Schwierigkeitsgrad der Plattform ( $PDIF$ ), die Mitarbeiterfähigkeiten ( $PERS$ ), die Erfahrungen des Personals ( $PREX$ ), den Zeitplan ( $SCED$ ) und unterstützende Einrichtungen ( $FCIL$ ) betreffen.  $PM_m$  findet Verwendung, wenn ein signifikanter Anteil des Codes automatisch erstellt wird. Bei  $ASLOC$  handelt es sich um die Anzahl der automatisch erstellten Zeilen des Quellcodes und bei  $ATPROD$  um den Produktivitätsgrad für diese Art der Codeerstellung. Allerdings ist ein gewisser Aufwand erforderlich, um eine Schnittstelle zwischen dem erzeugten Code und dem restlichen System herzustellen. Dieser hängt vom prozentualen Anteil ( $AT$ ) automatisch erstellten Codes an gesamten Systemcode ab.

### **3. Die Stufe nach dem Architekturentwurf:**

- Die Schätzungen basieren auf LOC.
- Die erste Aufwandsberechnung wird verfeinert.
- Es werden 17 Attribute anstelle von 7 Attributen verwendet.
- Die Schätzung der Gesamtanzahl der Zeilen an Quellcode wird so angepasst, zwei wichtige Projektfaktoren Berücksichtigung finden:
  - ~ **Die Unbeständigkeit der Anforderungen:** Diese Schätzung wird als Anzahl der zu ändernden Zeilen im Quellcode ausgedrückt und der anfänglichen Größenschätzung hinzuaddiert.
  - ~ **Der Umfang einer möglichen Wiederverwendung:** Umfangreiche Wiederverwendung bedeutet, dass die Anzahl an Zeilen des tatsächlich entwickelten Quellcodes berichtigt werden muss.

Im Vergleich zum ursprünglichen COCOMO-Modell wird hier der Exponent unter Berücksichtigung von Skalierungsfaktoren geschätzt:

- 1. Vorhandensein:** Frühere Erfahrungen mit ähnlichen Projekten
- 2. Entwicklungsflexibilität:** Keine Einbeziehung des Kunden, Freiheit der Gestaltung des Entwicklungsprozesses
- 3. Architektur/Risikoauflösung:** Umfang der durchgeführten Risikoanalyse
- 4. Teamzusammenhalt:** Vertrautheitsgrad der Entwickler untereinander
- 5. Prozessausgereiftheit:** Eine gewisse Prozesssteuerung ist vorhanden.

### **Beispiel für die Auswirkung von Skalierungsfaktoren auf die Aufwandsschätzung:**

Exponent	1.17
Systemgröße(einschließlich der Faktoren für die Wiederverwendung und die Unbeständigkeit der Anforderungen)	128,0 DSI <sup>a</sup>
<b>Erste COCOMO-Schätzung ohne Kostenfaktoren</b>	<b>730 PM</b>

Zuverlässigkeit	Sehr hoch
Komplexität	Sehr hoch
Speicherbeschränkungen	Hoch
Werkzeugverwendung	Gering
Zeitplan Beschleunigt	
<b>Angepasste COCOMO-Schätzung</b>	<b>2306 PM</b>

Zuverlässigkeit	Sehr gering
Komplexität	Sehr gering
Speicherbeschränkungen	Keine
Werkzeugverwendung	Sehr hoch
Zeitplan	Normal
<b>Angepasste COCOMO-Schätzung</b>	<b>295 PM</b>

In diesem Beispiel werden den wichtigsten Kostenfaktoren Höchst- und Mindestwerte zugewiesen, um zu zeigen, wie sie die Aufwandschätzung beeinflussen. Die verwendeten Werte entsprechen den in COCOMO 2-Referenzhandbuch (Böhm, 1997) genannten Werten. In diesem Beispiel ist es sehr anschaulich, dass hohe Werte für die Kostenfaktoren zu einer Aufwandsschätzung führen, die mehr als dreimal so hoch wie die Ausgangsschätzung, während niedrige Werte die Schätzung etwa 1/3 der ursprünglichen Schätzung reduzieren. Dies unterstreicht die großen Unterschiede zwischen verschiedenen Projekttypen und die Schwierigkeiten bei der Übertragung der Erfahrungen von einem Anwendungsgebiet auf ein anderes.

#### **Vorteile von COCOMO:**

- + Geeignet für schnelle, grobe Schätzungen der anfallenden Kosten
- + Gute Ergebnisse bei kleineren Projekten, die in einer bekannten Entwicklungsumgebung durchgeführt werden (Vergleichbarkeit mit bereits durchgeführten Projekten ist gegeben)
- + Abdeckung des Gesamtprojekts angefangen bei der Designphase bis hin zur Testphase (z.T. durch Erfahrungswerte wie 10% Management, 10% Infrastrukturaufwand)

#### **Nachteile von COCOMO:**

- Vergleichbarkeit mit bereits durchgeführten Projekten nicht immer gegeben.
- Viele im voraus zu bestimmende Einflussfaktoren
- Erfahrungen zeigen Abweichungen der Schätzungen vom tatsächlichen Aufwand um den Faktor 4

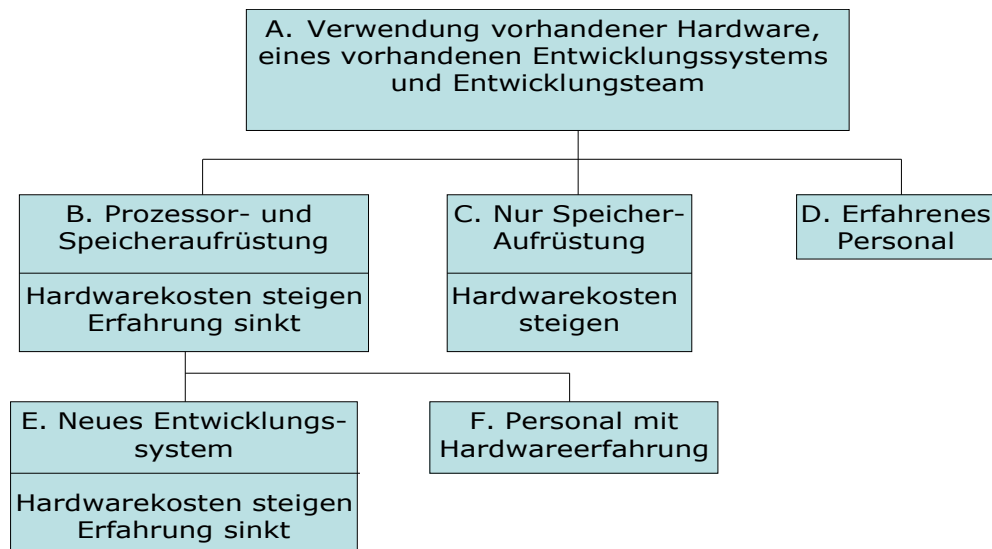
## **Algorithmische Kostenmodelle bei der Projektplanung**

Projektmanager können ein algorithmisches Kostenmodell verwenden, um verschiedene Investitionsarten zur Reduzierung der Projektkosten zu vergleichen. Dies ist insbesondere wichtig, wenn eine Abwägung der Hardware- gegen die Softwarekosten erfolgen muss und neue Mitarbeiter mit bestimmten Projektkenntnissen eingestellt werden müssen.

Bei der Aufwandsschätzung für ein Projekt sind drei Komponenten zu berücksichtigen:

1. Die Kosten der Zielhardware, auf der das System ausgeführt werden soll.
2. Die Kosten der Plattform, auf der das System entwickelt wird.
3. Die Kosten des für die Softwareentwicklung erforderlichen Aufwands.

Das folgende Schema zeigt einige Möglichkeiten, die in Betracht gezogen werden können. Dazu gehört, mehr für die Zielhardware auszugeben, um die Softwarekosten zu reduzieren, oder in bessere Entwicklungswerkzeuge zu investieren.



Zusätzliche Hardwarekosten könnten in diesem Fall akzeptabel sein, da es sich um ein spezielles System handelt, das nicht aus einer Massenproduktion stammen muss. Wird die Hardware in Verbraucherprodukte eingebettet, ist es selten annehmbar, mehr in die Zielhardware zu investieren, um die Softwareprodukte zu senken, da sich damit die Stückkosten des Produkts erhöhen.

### Kosten der Managementoptionen:

Option	RELY	STOR	TIME	TOOLS	LTEX	PM	SK(\$)	HK(\$)	GK(\$)
A	1,39	1,06	1,11	0,86	1	63	949.393	100.000	1.049.393
B	1,39	1	1	1,12	1,22	88	1.313.550	120.000	1.402.025
C	1,39	1	1,11	0,86	1	60	895.653	105.000	1.000.653
D	1,39	1,06	1,11	0,86	0,84	51	769.008	100.000	897.490
E	1,39	1	1	0,72	1,22	56	844.425	220.000	1.044.159
F	1,39	1	1	1,12	0,84	57	851.180	120.000	1.022.706

$$SK = \text{Aufwandsschätzung} * RELY * TIME * STOR * TOOL * EXP * 15.000\$/PM$$

Option A stellt die Kosten für den Aufbau des Systems mit vorhandener Ausstattung und vorhandenem Personal dar. Sie bildet die Grundlage für einen Vergleich. Alle anderen Optionen

sind entweder mit höheren Hardwarekosten oder der Einstellung neuen Personals (mit den damit einhergehenden Kosten und Risiken) verbunden. Option B zeigt, dass eine Aufrüstung der Hardware nicht unbedingt die Kosten senkt, da der Erfahrungsmultiplikator bedeutender ist. Es ist eigentlich kostenwirksamer, anstelle der gesamten Computerkonfiguration den Speicher aufzurüsten.

Option D scheint für alle grundlegenden Schätzungen die niedrigsten Kosten anzubieten. Es entstehen keine zusätzlichen Hardwarekosten, allerdings neue Personal für das Projekt abgestellt werden. Ist dieses bereits im Unternehmen vorhanden, ist es wahrscheinlich am günstigsten, diese Option zu wählen. Andernfalls müssen externe Mitarbeiter eingestellt werden, was mit beträchtlichen Kosten und Risiken verbunden ist. Das könnte bedeuten, dass die Kostenvorteile dieser Option weitaus geringer ausfallen, als es die obige Tabelle nahe legt. Die Option C bietet eine Kostenersparnis von fast 50.000 \$ und zwar praktisch ohne ein damit verbundenes Risiko. Konservative Projektmanager würden wahrscheinlich diese Option anstelle der Riskanteren Option D wählen.

## Projektdauer und Personalplanung

Genauso, wie sie den für die Entwicklung eines Softwaresystems erforderlichen Aufwand und die Gesamtkosten für diesen Aufwand abwägen, müssen Projektmanager auch schätzen, wie viel Zeit die Entwicklung der Software in Anspruch nehmen wird und wann Mitarbeiter für die Projektarbeit benötigt werden.

Das COCOMO-Modell enthält eine Formel zum Schätzen des Entwicklungszeitraums, der für die Fertigstellung eines Projekts erforderlich ist. Die Zeitberechnungsformel ist in allen COCOMO-Stufen dieselbe:

$$\text{Entwicklungszeit} = 3 * (\text{PM})^{(0.33+0.2)*(B-1.01)}$$

Die Nenn-Entwicklungszeit entspricht nicht unbedingt der für den Projektplan erforderlichen Zeit. Das wird im COCOMO 2-Modell berücksichtigt.

$$\text{Entwicklungszeit} = 3 * (\text{PM})^{(0.33+0.2*(B-1.01))} * \text{SCEDPercentage} / 100$$

Bei *SCEDPercentage* handelt es sich um den prozentuellen Anstieg oder die Verkürzung der Nenn-Entwicklungszeit. Unterscheidet sich die kalkulierte Zahl dann signifikant vom Zeitplan, bedeutet dies auf ein bestehendes hohes Risiko hin, dass Probleme mit der pünktlichen Lieferung der Software entstehen werden.

## **Literaturangabe:**

1. B.W. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.
2. E. Horowitz und andere, *Software Cost Estimation with COCOMO II*
3. [sunset.usc.edu/research/COCOMOII/index.html](http://sunset.usc.edu/research/COCOMOII/index.html)
4. [www.spr.com](http://www.spr.com) (Software Productivity Research Inc., Unternehmen von Capers Jones)
5. Helmut Balzert, *Lehrbuch der Software Technik*
6. Ian Sommerville, *Software Engineering*
6. [www.software-kompetenz.de](http://www.software-kompetenz.de)