SOFTWARE ENGINEERING

Elite Graduate Program

Projektmanagement: Schätzverfahren

Martin Wirsing

Institut für Informatik
Ludwig-Maximilians-Universität München

WS 2006/07

Ludwig Maximilians Universität München

Ziele

- Generelles Vorgehen bei Schätzungen kennen lernen
- Grundlegende Schätzmuster und Maße zur Systemkomplexität kennen lernen
- Schätzverfahren für den Aufwand eines Software-Projekts kennen lernen und beurteilen
  - **Delphi-Methode**
  - **Function Points**
  - **CoCoMo**

Schätzverfahren

„Was man nicht misst, das kann man nicht steuern.“

[Tom de Marco, *Controlling Software Projects*, 1982.]

- **Messung von Softwaresystemen:**
  - Zu Projektbeginn: Projektplanung (Schätzung)
  - Bei Durchführung: Projektstatus/Projektsteuerung
- **Ziel des Schätzens:**
  - Bestimmung des Entwicklungsaufwands
    - Realisierungsaufwand
    - Realisierungszeit
  - Abhängig von Systemkomplexität und Produktivität
  - Möglichst vor Systemrealisierung
- **Prinzipielle Strategie beim Schätzen: per Analogie**
- Schätzungen sind „unfair“:
  - Passieren zu einem sehr frühen Zeitpunkt
  - Man weiß wenig über die Aufgabe
  - Auf die Zahlen wird man später festgenagelt

Herangehensweisen

- Von den Arbeitspaketen zur Aufwandszahl
  - **Schätzung der einzelnen Arbeitspakete**
  - **Summe ergibt Gesamtaufwand**
  - ➢ **Schätzung als Grundlage für Aufwand**

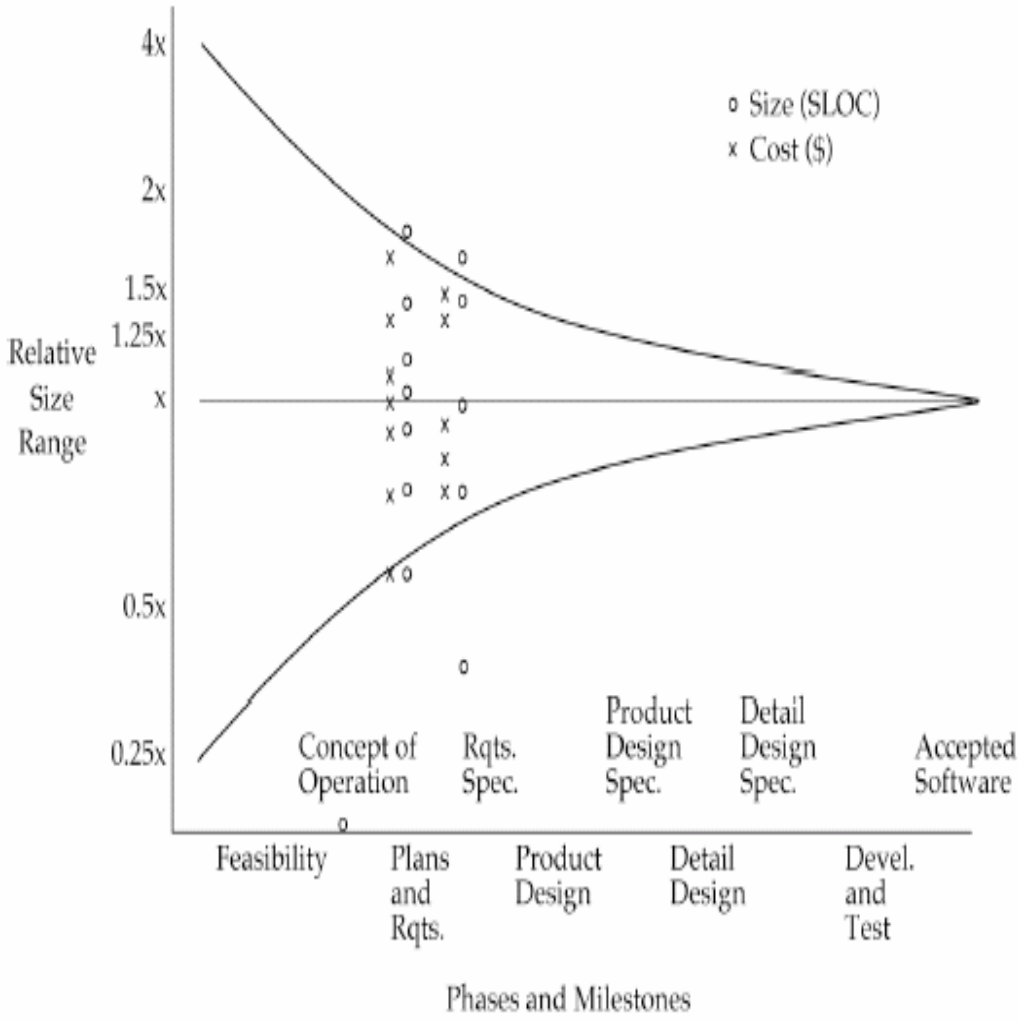
- Vom fixierten Aufwand zu den Arbeitspaketen
  - **Frage: Was darf es denn insgesamt kosten?**
  - **Projekt so aussteuern, dass es im Kostenrahmen bleibt**
  - **Aufgaben werden nur so „gut“ erledigt, wie Geld da ist**
  - ➢ **Schätzung zur Prüfung der Machbarkeit**

Auswirkungen einer „falschen“ Schätzung

- Zu hohe Schätzung
  - **Ist kaum festzustellen. Jedes Projekt schöpft den Kostenrahmen voll aus.**
  - **Gefahr: Business Case rechnet sich nicht, bzw. Projekt wird an Mitbewerber verloren.**
- Zu geringe Schätzung
  - **Geld reicht nicht, Projekt kann im Budget nicht durchgeführt werden**
- Das Projektmanagement hat extrem großen Einfluss auf die verbrauchten Aufwände eines Projekts. Im Nachhinein ist schwer festzustellen, ob der geschätzte Kostenrahmen realistisch war.

Motivation für eine Schätzung

- Wenn eine Schätzung so fehlerbehaftet und schwierig ist, warum schätzt man dann überhaupt?
  - **Auch eine „falsche“ Schätzung ist besser als ein kompletter Blindflug**
  - **Auch Schätzen ist ein Prozess: sobald erste Erfahrungen und ermittelte Aufwände im Projekt vorliegen, wird nachgeschätzt und die Schätzung korrigiert**
  - **Die Schätzung wird im Laufe des Projekts immer genauer.**



Einflussfaktoren auf den Aufwand (wichtigste Faktoren)

- **Umzusetzende Fachlichkeit (funktional)**
  - Masken
  - Druckstücke
  - Berechnungen
  - zu berücksichtigende Fehlersituationen
  - Migrationen aus Altsystemen
  - Abhängigkeit von anderen Systemen
- **Technologische Umsetzung,**
  - nicht-funktionale Anforderungen
    - Performance, Antwortzeitverhalten
  - Architektur
  - Systemplattform, Basis-Technologien
- **Projektmanagement-Faktoren**
  - Team
    - Mitarbeiterqualifikation
    - Erfahrung
    - Eingespieltes Team
  - Projektorganisation
  - Projektvorgehen, Methodik
  - Unterstützung durch Tools
- **Sonstige Faktoren**
  - Auftraggeber
  - Aufwände steigen mit Größe der Aufgabe

Generelles Vorgehen bei der Schätzung

- Eine gute Vorbereitung ist elementar für die Schätzung
  - **Komplettieren und Strukturieren der Schätzgrundlage**
    - **(osintots – oh shit, I never thought of this)**
  - **Sammeln aller Faktoren**
- Nachschätzen und aus Projekterfahrungen lernen ist ein stetiger Kreislauf während des Projekts

Schätzmuster

- Schätzungen werden fast immer aus einer Kombinationen der folgenden grundlegenden **Schätzmuster** hergeleitet
- Insbesondere beruhen auch ausgefeilte Schätzmethoden wie Function Points, CoCoMo oder Delphi auf diesen Mustern, meist ergänzt um konkrete Zahlenwerte für Schätzformeln.

- **Schätzmuster:**
  - **Schätzen durch Vergleich**
  - **Schätzen durch Zerlegung**
    - **Anforderungen**
    - **des Entwurfs**
  - **Expertenschätzung**
  - **Schätzen mit Korrekturfaktoren**
  - **Schätzen mit Stellvertretergröße**

Schema für Schätzmuster:

- **Problem/Kontext**
- **Vorgehensidee**
- **Vorteile, Nachteile**
- **evtl. Varianten, Anmerkungen**

Schätzen durch Vergleich

- **Problem/Kontext:**

  Wenn keine bessere Schätzmethode verfügbar ist,
  aber Erfahrungen mit Entwicklung ähnlicher Dinge
  und deren Aufwände **noch** bekannt sind.

- **Vorgehensidee:**

  Wähle aus dem Erfahrungsschatz das ähnlichste Ding aus und benutze dessen Aufwand als Schätzung

- **Vorteile:** Einfach, schnell
- **Nachteile:** Evtl. zu wenig Erfahrung; Ähnlichkeitsmaß unklar
- **Variante:**

  Wähle mehrere Ähnlichste und mittele die Schätzung

- **Anmerkung:**

  Alle anderen Schätzverfahren basieren letzten Endes auf Vergleich (explizit oder intuitiv)

Schätzen durch Zerlegung (Anforderungen)

- **Problem/Kontext:**

  Wenn Anforderungen wenig voneinander abhängen,

  d.h. das System nur einen kleinen Kern aufweist
- **Vorgehensidee:**

  Schätze Aufwand pro Anforderung einzeln; summiere
- **Vorteile:**

  Starke Reduktion der Komplexität
- **Nachteile:**

  Nur in wenigen Anwendungsgebieten sinnvoll,

  Meist ist der Kern für den Aufwand sehr relevant,

  Aufwandsmessungen liegen selten in dieser Form vor.
- **Anmerkung:**

  Das „**Function Point**" Schätzverfahren beruht auf diesem Muster und ist für einfache Informationssysteme sehr bewährt.

Schätzen durch Zerlegung (Entwurf)

- **Problem/Kontext:**

  Wenn die Architektur des Systems bereits erkennbar ist

- **Vorgehensidee:**

  Schätze den Aufwand für die Subsysteme und addiere

- **Vorteile:**

  Im Prinzip ist eine sehr genaue Zerlegung und Schätzung möglich

- **Nachteile:**

  Irrtümer über Architektur möglich (insbes. Teile übersehen)

  Vernachlässigt Aufwand für Integration und nichtfunktionale Anforderungen

- **Anmerkung:**

  Zumindest grobe Zerlegung wird in der Praxis fast immer eingesetzt.

  Oft läuft ein großer Teil der Zerlegung auf bloße Zählung hinaus, z.B. Anzahl "Dialoge" (Bildschirmmasken)

Schätzen durch Experten

- **Problem/Kontext:**
  Wenn es Erfahrungen und Aufwandsdaten nicht schriftlich,
  wohl aber im Kopf von Experten gibt
- **Vorgehensidee:**
  Bitte jede/n Experten/in separat um eine Schätzung
  Lasse die Experten Ihre Schätzungen und Begründungen diskutieren und modifzieren
  Resultat: Konsens über einen geschätzten Aufwandsbereich
- **Vorteile:**
  Sehr breitbandig, bezieht enorm viele Faktoren ein
  Unsicherheit der Schätzung wird ggf. klar sichtbar
- **Nachteile:**
  Kann bei "Groupthink" zu dramatisch falschen Ergebnissen führen, die scheinbar vertrauenswürdig sind
- **Variante:**
  Black-box-Schätzungnur eines/r Experten/in
- **Anmerkung:**
  Das „**Delphi**" Schätzverfahren beruht auf diesem Muster.

Kombination von Schätzungen

- **Problem/Kontext:**

  Wenn mehrere Schätzungen verfügbar sind, deren Qualität aber unklar ist

- **Vorgehensidee:**

  Kombiniere die Schätzungen zu einer (durch Bereichsbildung, Mittelung, Zurückweisung von Ausreißern etc.)•

- **Vorteile:**

  Erhöht die Robustheit der Schätzung

- **Nachteile:**

  Gibt es systematische Fehler in vielen der Schätzungen, dann führt das Kombinieren in die Irre

- **Anmerkung:**

  Das „**Delphi**" Schätzverfahren wendet dieses Muster an.

Schätzen mit Korrekturfaktoren

- **Problem/Kontext:**

  Wenn ein ähnliches Ding zum Vergleich verfügbar ist

  es aber zum jetzigen Ding (identifizierbare) Unterschiede gibt

- **Vorgehensidee:**

  Benutze den Aufwand des ähnlichen Dings und korrigiere ihn für jeden Unterschied um einen prozentualen Faktor herauf/herunter.

  Die einzelnen Faktoren werden wiederum geschätzt oder aus Erfahrungen entnommen.

- **Vorteile:**

  Recht flexibel

- **Nachteile:**

  Bei zu vielen Korrekturen wird das Ergebnis dubios.

  Bei zu wenig Erfahrung über einzelnen Korrekturfaktor ebenfalls.

- **Anmerkung:**

  Das „**CoCoMo**" Schätzverfahren beruht hauptsächlich auf diesem Muster.

Schätzen mit Stellvertretergrößen

- **Problem/Kontext:**

  Wir besitzen Aufwandsdaten nur über andere Erfahrungsgrößen als die, die wir schätzen wollen,

  z.B. Produktivität in Anzahl an Zeilen von Code (Lines of Code, LoC) pro Personenmonat,

  aber diese Erfahrungsgröße (oder etwas Verwandtes) lässt sich schätzen

- **Vorgehensidee:**

  Schätze nicht den Aufwand, sondern die schätzbare Größe und rechne dann um.

- **Vorteile:**

  Evtl. ist Stellvertretergröße anschaulicher und besser zu schätzen

- **Nachteile:**

  Evtl. wird Kontextabhängigkeit übersehen

  z.B. Abhängigkeit der Produktivität an Zeilen von Code von der Programmiersprache

- **Anmerkung:**

  Das „**CoCoMo**" Schätzverfahren verwendet „Anzahl der Zeilen von Code" als Ausgangspunkt

Exkurs: Messen von Systemkomplexität

- **Komplexitätsfaktoren**
  - Größe: Anzahl der Bausteine
  - Diversität: Unterschiedlichkeit der Bausteine
  - Vernetzung: Abhängigkeit der Bausteine
- **Grundlagen der Softwareschätzung**
  - Messung/Schätzung von Systemkomplexität
  - Komplexität, Aufwand, Laufzeit
- **Softwaremetrik Code**
  - Ansatz: Software = Programmcode
  - Einheit: Anzahl der Programmzeilen (Lines of Code – LoC)
  - Messverfahren: Softwareumfang = Anzahl Programmzeilen
  - Vorzug: Einfache Messung
- **Verschiedene Maße in verschiedenen Phasen**
  - Implementierung: Codezeilen
  - Feinentwurf: Abstrakter Code
  - Analyse: Funktionspunkte

Exkurs: Messen von Systemkomplexität
Lines-Of-Code (LoC) und Varianten

- **Lines-of-Code (LoC)**
  - zähle Zeilen, subtrahiere leere Zeilen
- **Non-Comment-Lines-of-Code (NCLoC)**
  - zähle LOC, subtrahiere reine Kommentarzeilen
- **Kommentardichte (KD)**
  - abgeleitet, daher kein eigenes Messverfahren nötig
  - KD(P) = CLoC(P) / NCLoC(P)
- **Verteilung Deklarationen/ausführbare Befehle**
  - unterscheide NCLoC in DecLoC und ExecLoC
  - z.B. Deklarationsanteil(P) = DecLoC / (DecLoC + NCLoC)

Exkurs: Messen von Systemkomplexität
Lines-Of-Code (LoC) Produktivitätsparadoxon

- **Die Benutzung einer höheren Programmiersprache scheint eine geringere Produktivität nach sich zu ziehen: für das gleiche Geld/in der gleichen Zeit bekommt man weniger Ergebnis.**

Sprache	Assembler		Cobol	
Resultat	10.000	LoC	3.000	LoC
Aufwand	500	PT	300	PT
Kosten	125.000	€	75.000	€
Produktivität	12,50	€/Zeile	25,50	€/Zeile
	0,05	PT/Zeile	0,1	PT/Zeile
	20	Zeile/PT	10	Zeile/PT

- **Die Abstraktionsebene der Programmiersprache geht in diese Berechnung nicht ein – in den Nutzen für den Anwender aber schon.**

- **Daher das LoC-Maß nur für relative Produktivitätsbetrachtungen geeignet, also z.B. bei der Prozessverbesserung.**

Exkurs: Messen von Systemkomplexität
Alternativen zu Lines-Of-Code (LoC)

- **Non-Comment-Source-Statements (NCSS)**
  - baue abstrakten Syntaxbaum auf, zähle geeignete Knotentypen
  - z.B. im Eclipse-Plugin

- **Volumen**
  - Wird gemessen in Bit
  - kein eigenes Messverfahren, Wert wird abgeleitet: $V=N \bullet \log_2 \eta$
  - wobei
    - N=Größe des Vokabulars
    - η=benutzte Symbole

- **Es zeigt sich aber empirisch:**
  Für viele realistische Programme p gilt: V(p) = O(LoC(p)).

- **Für eine Reihe weiterer Größenmaße gilt das gleiche.**

Exkurs: Messen von Systemkomplexität
Entwurfsmetriken

- **OO-Metriken:**
  - Attribute, Methoden, Anzahl Vererbungsstufen, Abhängigkeiten

- **Graph-basierte Maße (gut bei UML-Modellen)**
  - **Zusammenhalt (auch: Kohäsion)** eines Teilgraphen G: Anteil der Kanten innerhalb von G im Vergleich zur Anzahl aller Kanten, an denen G beteiligt ist.
  - **Kopplung** zwischen zwei Teilgraphen G1 und G2 eines Graphen: Anteil existierender Verbindungen zwischen Knoten von G1 und G2 im Vergleich zur Anzahl aller möglichen Verbindungen
  - **McCabe‘s Cyclomatic Complexity (bei Flussgraphen** : Anzahl der linear unabhängigen Pfade (= #Kanten - #Knoten +2)

Schätzverfahren: Delphi-Methode

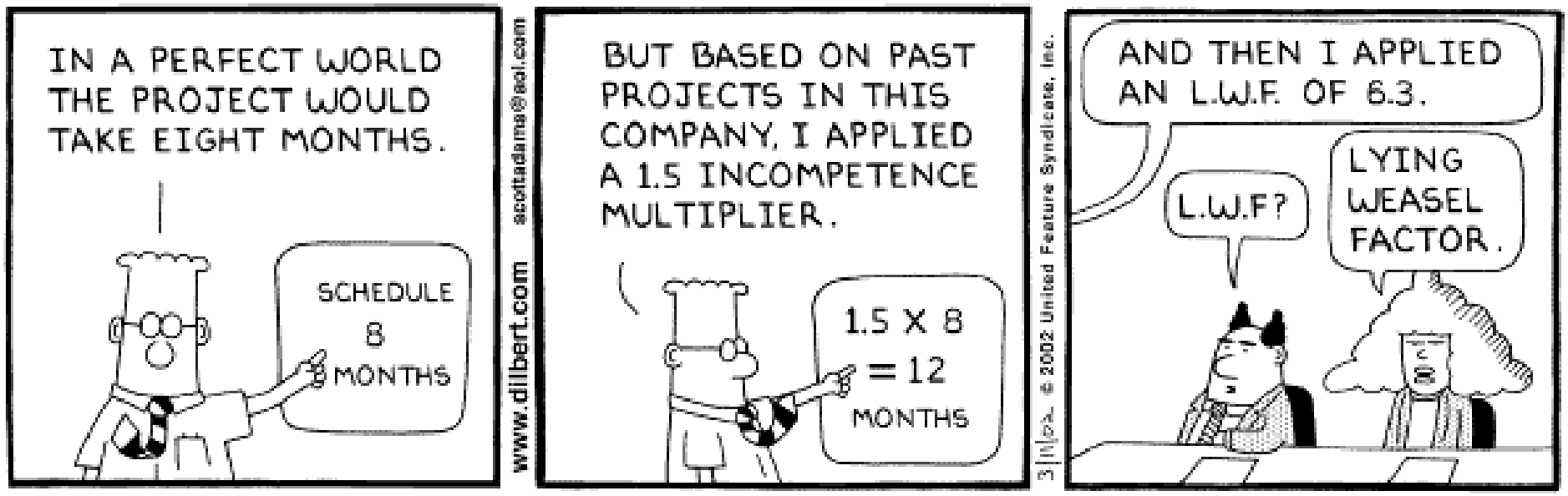
- Charakteristikum:
  - **Systematische Befragung von mindestens zwei Experten, die aus Erfahrung Voraussagen über den Zeit/Ressourcen-Bedarf einzelner Projektaktivitäten machen.**
- Varianten:
  - **Standard Delphi-Verfahren:**
    - **Befragung anonym**
  - **Breitband Delphi-Verfahren:**
    - **Schätzergebnisse werden gegenseitig bekanntgegeben, damit Resultate diskutiert und ggf. korrigiert werden können**
    - **Häufig Schätzung im Rahmen einer Schätzklausur**

Schätzklausur

- **Phase 1: Auftragsdefinition**
  - Auftraggeber, Kunde, Verantwortliche und Durchführende festlegen
  - Ziel definieren
  - Aufgaben analysieren (Prozesszerlegung)
- **Phase 2: Schätzungsvorbereitung**
  - Geeignete Leute einladen
  - Fachleute, Verantwortliche, Durchführende
  - 2 Stunden ansetzen
  - Excel-Sheet vorbereiten
- **Phase 3: Schätzung**
  - Projekt und Projektplan vorstellen, Aufwands- und Zeitplan austeilen, ggf. korrigieren (lassen)
  - Jeder schreibt seine Schätzungen auf Ansage auf
  - Jeder sagt seine Schätzungen auf Ansage an
  - Extrema diskutieren, ggf. entfernen
  - Mittelwerte bilden
  - Summe bilden

Schätzklausur

**Meistens gibt es anschließend noch eine Phase 4,
in der die Schätzung interessengeleitet verändert wird.**



Zwischenfazit

Sind Schätzungen notorisch ungenau – wieso?

In wohlgeordneten Prozessen sind Schätzungen von einem zum nächsten Mal vergleichbar.

Man kann also aus Fehlern lernen.

Nicht so bei ungeordneten Prozessen.

Hier werden Fehler versteckt oder verdeckt.

Schätzverfahren: Funktionspunkte (Function Points - FP)

- **Historie:**

  Eingeführt 1979 von Alan Albrecht (IBM).

  Seit 1986 verwaltet und normiert durch die International Function Point User Group (IFPUG);

  Gegenstand der ISO-Standards 10926 und 14143-1:1998.

- **Idee:**

  Schätzung des Arbeitsaufwands in Mann-Stunden anhand der vom Kunden gewünschten Funktionen

- **Vorgehen:**
  - Zähle und klassifiziere (als einfach, mittel, komplex) wenige Arten von Anforderungen, vergebe dafür jeweils **Funktionspunkte** (FP), summiere diese und bestimme Aufwand daraus per empirischem Umrechnungsfaktor (**unangepasste FP**)

    FP ist also ein **Stellvertreterverfahren**
  - Anschließend kann man noch Projektumstände berücksichtigen, um die Schätzung anzupassen (**angepasste FP**)

    Hinzu kommt also ein **Korrekturfaktorverfahren**

Unangepasste FP (UFP)

- **Messverfahren: Klassifiziere Anforderungen in**
  - Eingaben — Eingabemasken, für die Datensätze eingegeben werden können
  - Ausgaben — Datenpräsentationsdarstellung ("Report")
  - Abfragen — Eingabe eines Suchkriteriums plus Anzeige des Gefundenen
  - Datenbestände — Intern verwalteter Datenbestand (z.B. DB-Tabelle)
  - Referenzdaten — Schnittstelle/Datenformat zur Anbindung an anderes System

- **Bewerte jedes Element als *einfach*, *mittel* oder *komplex* und bilde die gewichtete Summe nach folgender Tabelle:**

Item	einfach	mittel	komplex
Eingaben	3	4	6
Ausgaben	4	5	7
Interaktionen	3	4	6
Datenbestände	7	10	15
Referenzdaten	5	7	10

Angepasste FP

- FP = UFP * TK

  wobei der Korrekturfaktor **Technische Komplexität (TK)** gebildet wird durch

  $$TK = 0.65+0{,}01 \sum_{i=1..14} F_i \qquad \text{(d.h. } 0.65 <= TK <= 1.35\text{)}$$

- und **F_i Korrekturwerte** gemäß dem **Schwierigkeitsgrad gewisser nichtfunktionaler Anforderungen** darstellen:

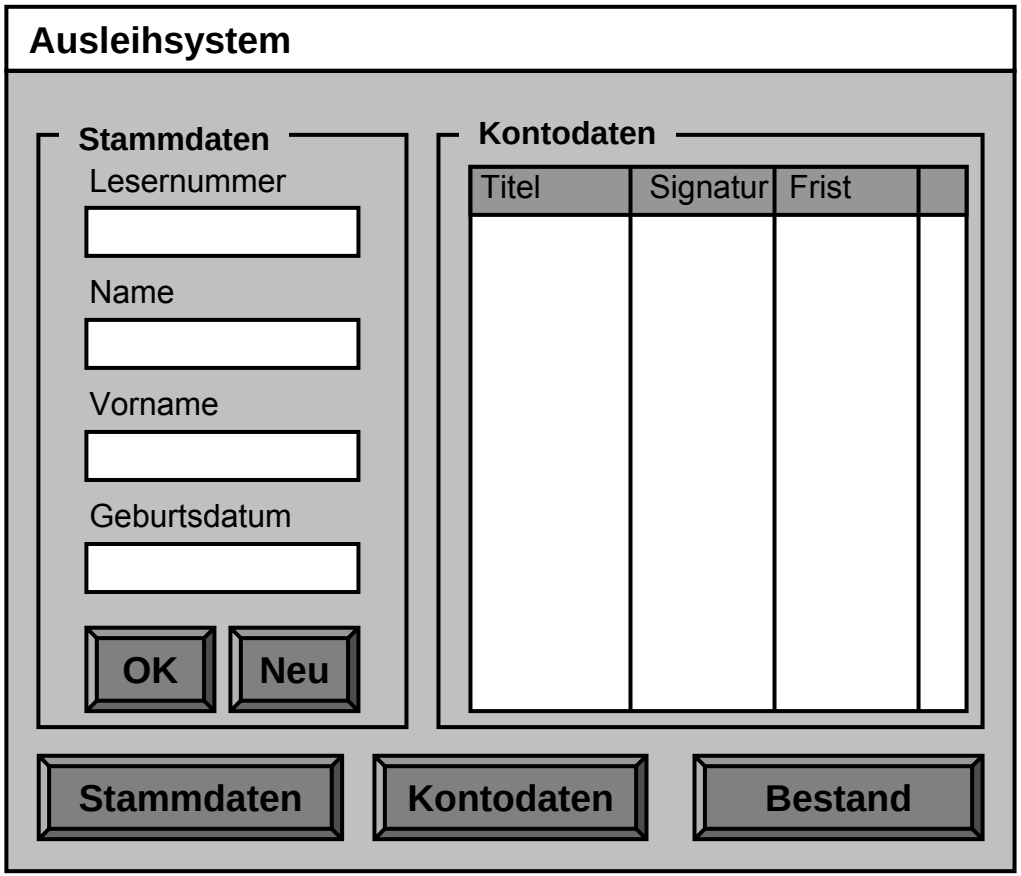
F_1 Reliable back-up and recovery	F_2 Data communications
F_3 Distributed functions	F_4 Performance
F_5 Heavily used configuration	F_6 Online data entry
F_7 Operational ease	F_8 Online update
F_9 Complex interface	F_{10} Complex processing
F_{11} Reusability	F_{12} Installation ease
F_{13} Multiple sites	F_{14} Facilitate change

  haben je einen Wert zwischen 0 und 5 (für ***nicht*** bzw. ***sehr stark vorhanden***).

Technische Korrekturfaktoren
Übersetzung und Erläuterung

Faktor	Originalname	Erläuterung
F1	Reliable back-up & recov.	Anforderungen an die Datenintegrität
F2	Data communications	Datenaustausch mit Umsystemen
F3	Distributed functions	Applikationen auf mehr als einem Rechner
F4	Performance	Leistungsanforderungen
F5	Heavily used config.	Starke Konfigurationsabhängigkeiten
F6	Online data entry	Interaktive Benutzung
F7	Operational ease	Interaktionsqualität
F8	Online update	unmittelbare Aktualisierung (WYSIWYG)
F9	Complex interface	komplexe Benutzerschnittstelle
F10	Complex processing	algorithmisch komplexe Verarbeitung
F11	Reusability	geplante Wiederverwendung/Wartung
F12	Installation ease	Installationskomfort
F13	Multiple sites	mehrere (unterschiedliche) Installationen
F14	Facilitate change	Unterstützung der Anpassung in der Wartung

Berechnung am Beispiel „Ausleihmaske“



Mögliche Interaktionen:
1) Ausweis scannen
2) Konto- und Stammdaten abfragen durch Eingabe der Lesernummer
3) Stammdaten eingeben

Berechnung am Beispiel „Ausleihmaske“

1.) Elemente zählen

	(1)	(2)	(3)
Eingaben	1x einfach	-	1x mittel
Ausgaben	-	1x mittel	-
Abfragen	-	-	-
Referenzdaten	1x einfach		
Datenbestände	3x einfach		

Berechnung am Beispiel „Ausleihmaske“

2.) UFP berechnen

	(1)	(2)	(3)
Eingaben	**1x 3**	**-**	**1x 4**
Ausgaben	**-**	**1x 5**	**-**
Abfragen	**-**	**-**	**-**
Referenzdaten	**1x 5**	**-**	**-**
Datenbestände		**------ 3x 7 ------**	

UFP = 3 + 4 +5 + 5 +21 = 38

Berechnung am Beispiel „Ausleihmaske“

3.) Technische Faktoren schätzen

2	F_1 Reliable back-up & recovery
1	F_2 Data communications
2	F_3 Distributed functions
1	F_4 Performance
0	F_5 Heavily used configuration
3	F_6 Online data entry
5	F_7 Operational ease
3	F_8 Online update
2	F_9 Complex interface
0	F_{10} Complex processing
1	F_{11} Reusability
1	F_{12} Installation ease
4	F_{13} Multiple sites
1	F_{14} Facilitate change

Technische Komplexität

TK = $0{,}65+0{,}01 \sum_{i=1..14} F_i$

hier also

TK = 0,65+0,01 * 26

= 0,91

FP = TK * UFP

Also im Beispiel

FP = 0,91 * 38 = 34,58

Falls in einer Firma 1 FP 2,1 PT entspricht, erhält man ca. 72 PT;

d.h. bei 18 PT pro Monat (220 PT pro Jahr) einen Entwicklungsaufwand von etwa 3 PM

Ist das Produktivitätsparadoxon behoben?

Sprache	LoC / FP		
Komplexität	**niedrig**	**mittel**	**hoch**
Assembler	200	320	450
C	60	128	170
Fortran	75	107	160
Cobol	65	107	150
C++	30	53	125
Smalltalk	15	21	40

LoC / Mannmonat relativ konstant.

=> In einer höheren Programmiersprache schafft ein Programmierer mehr Funktionalität pro Zeiteinheit.

=> Es gibt Produktivitätsunterschiede zwischen Programmiersprachen.

Damit ist ein Teil des Produktivitätsparadoxons behoben, aber es gibt noch andere Einflussgrößen: Das Verhältnis LoC/FP hängt z.B. auch von der Systemkomplexität insgesamt ab. Aber FPs sind schon deutlich besser als LoC.

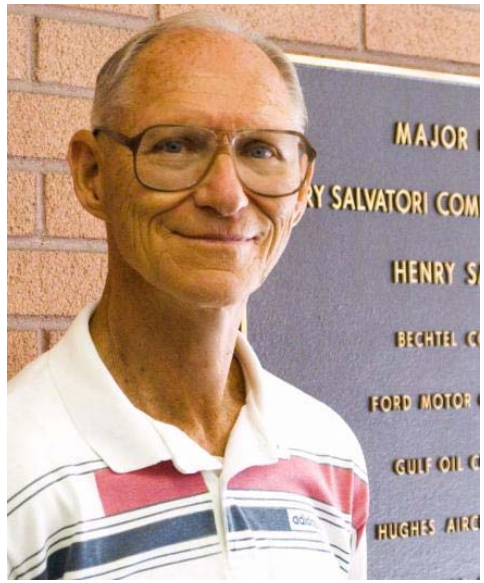
Erfahrungswert: Die Erstellung eines Function Points kostet etwa 1.500 €.

Schätzverfahren: CoCoMo (Constructive Cost Modeling)

- Entwickler: Barry Boehm
- 1981 CoCoMo I, ab 1995 CoCoMo II
- **Idee:**
  - **Schätzung der Projektgröße Size in LOC** (Lines of Code) bzw. KDSI (Kilo Delivered Software Instructions), d. h. ohne Kommentare.
  - Gleichungen:

    Aufwand: $PM_{DEV} = a * Size^{b} * m$

    Laufzeit: $T_{DEV} = 2{,}5 * MM_{DEV}^{d}$
    - PM_{DEV} Entwicklungsaufwand in PM und
    - T_{DEV} Projektlaufzeit
    - a, b, d, m unternehmensspezifisch zu kalibrierende Faktoren
    - b beschreibt überproportionale Auswirkung großer Projekte
    - m dient zur Berücksichtigung von Attributen für Produkt, Vorgehen und die Qualität von Entwicklern



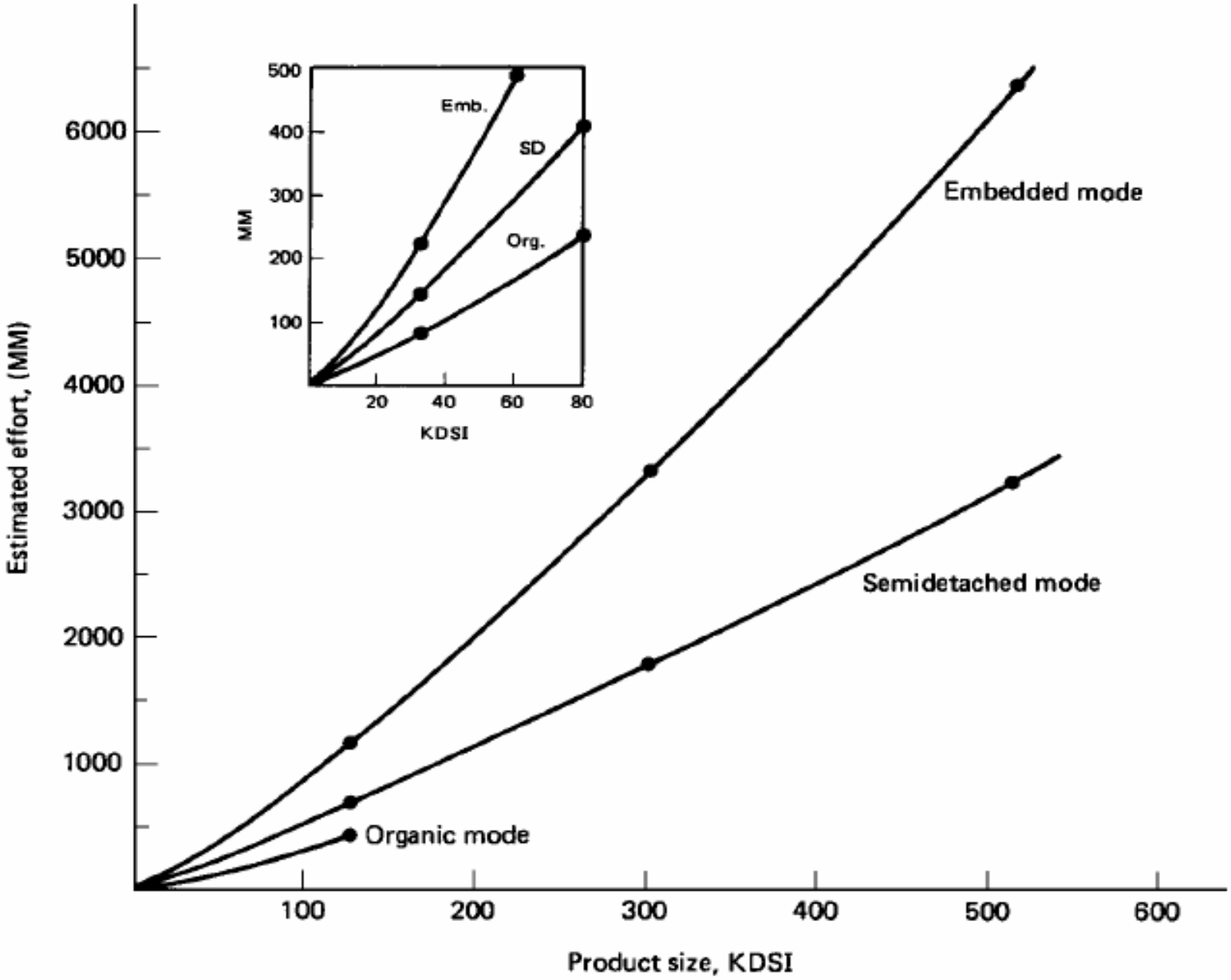
Barry Boehm
***1935**
PhD UCLA,
Prof. USC,
Erfinder von
V- und
Spiralmodell,
CoCoMo

CoCoMo I

Differenzierung in Projekt-Klassen

- **Einfach (Organic Mode)**
  - einfache Anwendungssoftware, Größe <50 KDSI
  - eingespieltes Team, bekannte Umgebung, wenig Neuland
  - $PM_{DEV} = 2{,}4 * (KDSI)^{1{,}05} * m$
  - $T_{DEV} = 2{,}5 * PM_{DEV}^{0{,}32}$
- **Mittelschwer (Semi-detached Mode)**
  - mittelschwere Projekte, Größe typischerweise <300 KDSI
  - $PM_{DEV} = 3{,}0 * (KDSI)^{1{,}12} * m$
  - $T_{DEV} = 2{,}5 * PM_{DEV}^{0{,}35}$
- **Eingebettete Systeme (Embedded Mode)**
  - schwierige Projekte, beliebige Größe
  - starker Kosten-, Termindruck, viel Neuland
  - $PM_{DEV} = 3{,}6 * (KDSI)^{1{,}2} * m$
  - $T_{DEV} = 2{,}5 * PM_{DEV}^{0{,}38}$

COCOMO: Größe vs. Aufwand



Einflussfaktoren, Kostentreiber

- **Produkt**
  - RELY: geforderte Zuverlässigkeit der Software
  - DATA: Größe der Datenbasis
  - CPLX: Komplexität des Produktes
- **Computer**
  - TIME: benötigte Rechenzeit
  - STOR: Nutzung verfügbarer Speicherplatz
  - VIRT: Änderungshäufigkeit der Systembasis
  - TURN: Bearbeitungszyklus
- **Projekt**
  - MODP: Verwendung moderner Entwicklungsmethoden
  - TOOL: Verwendung von Tools
  - SCED: Anforderungen an die Entwicklungszeit
- **Personal**
  - ACAP: Analysefähigkeit der Projektmitarbeiter
  - AEXP: Erfahrung der Mitarbeiter im Arbeitsgebiet
  - PCAP: Programmierfähigkeit der Mitarbeiter
  - VEXP: Erfahrung der Mitarbeiter in der Systemumgebung
  - LEXP: Erfahrung der Mitarbeiter in der Programmiersprache

Der Korrekturfaktor m dient zur Berücksichtigung unternehmensspezifischer und projektspezifischer Kostenfaktoren (cost drivers):

$\mathbf{m} = m1 * m2 * \ldots * m15$

$MM_{DEV} = a * KDSI^{b} * \mathbf{m}$

Kostenfaktoren **m** = m1 * m2 * … * m15

Cost Drivers	Ratings					
	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	

CoCoMo II

- **COCOMO II** (Boehm et al. 2000) ist eine Weiterentwicklung von COCOMO zu einem 3-Stufen Modell, das bei Entwicklungsfortschritt immer genauere Schätzungen ermöglicht.

COCOMO II unterscheidet:

- **Frühe Entwicklungsphasen (Early prototyping level)**
  - Schätzung für Projekte mit Prototyperstellung und hoher Wiederverwendung basierend auf Anwendungspunkten (application points) und einfacher Formel für die Aufwandschätzung
- **Frühe Entwurfsphase (Early design level)**
  - Schätzung nach **abgeschlossenen Festlegung der Systemanforderungen und einem ersten Entwurf** basierend auf Funktionspunkten, die in LOC übersetzt werden.
- **Nach-Architektur-Phase (Post-architecture level)**
  - Schätzung nach **Erstelleung der Architektur** basierend auf LOC

Nach-Architektur-Phase (Post-architecture level)

- Wir betrachten hier **nur die Post-Architektur-Phase** (für eine kurze Beschreibung der anderen Phasen siehe Anhang).
- Neue universelle Grundgleichungen
  - **Aufwand** $PM = 2{,}45 * KSLOC^{B} * M$
  - **Entwicklungszeit** $T = 2{,}66 * PM^{\,0{,}33 + 0{,}2 \cdot (B - 1{,}01)}$

    wobei
    - **KSLOC: Kilo Source Lines of Code**
    - **Wachstumsfaktor $B = 1{,}01 + 0{,}01 \, \Sigma \, SF_i$**
    - **SF_i sind insgesamt fünf Skalierungsfaktoren (scaling factors)**
    - **M ist das Produkt der insgesamt 17 Kostenfaktoren (effort multipliers)**

COCOMO II: Skalierungsfaktoren (scaling factors)

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

- **Erfahrung:** Vertrautheit des Entwicklungsteams mit dem Produkt
- **Flexibilität** bezüglich der Einhaltung von Anforderungen / Vorgaben
- **Risiko-Umgang:** Qualität des Risikomanagements
- Güte der **Zusammenarbeit** zwischen allen Projektbeteiligten
- **Reife** des Entwicklungsprozesses

Beispiel

- **Eine Firma will ein Projekt in einem neuen Gebiet durchführen. Der Kunde hat keinen speziellen Entwicklungsprozess gefordert und Zeit für die Risikoanalyse eingeplant. Die Firma besitzt eine Prozessreife der Stufe 1 (nach CMM – siehe später).**

- **Skalierungsfaktoren:**

Erfahrung:	neues Projekt	S. gering (4,05)
Prozessflexibilität	Kunde lässt Freiheit	S. hoch (1,21)
Architecture/risk resolution	Keine Risikoanalyse	S. gering (4,22)
Teamzusammenhalt	Neues Team	Nominal (2,97)
Prozessreife	CMM Level 1	Gering (3,64)
Summe		**16,19**

- **Der Skalierungsfaktor ist also 1,01 + 0,16 = 1.17**

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	

Beispiel: 2,45 * KSLOC$^{1.17}$ * M

Exponent value	1.17
System size (including factors for reuse and requirements volatility)	128, 000 **SLOC**
Initial COCOMO estimate without cost drivers	**730 person-months**
Reliability	Very high, multiplier = 1.39
Complexity	Very high, multiplier = 1.3
Memory constraint	High, multiplier = 1.21
Tool use	Low, multiplier = 1.12
Schedule	Accelerated, multiplier = 1.29
Adjusted COCOMO estimate	**2306 person-months**
Reliability	Very low, multiplier = 0.75
Complexity	Very low, multiplier = 0.75
Memory constraint	None, multiplier = 1
Tool use	Very high, multiplier = 0.72
Schedule	Normal, multiplier = 1
Adjusted COCOMO estimate	**295 person-months**

Schätzverfahren in der Praxis

- Oft sind Schätzungen gefragt, noch bevor man auch nur halbwegs den Projektinhalt versteht
  - **und dennoch werden die Schätzergebnisse anschließend verbindlich**
- Viele Firmen sammeln Aufwandsdaten nicht
  - **Dann hat man sie nur in den Köpfen der Entwickler/Projektleiter**
  - **Das ist sehr ungenau!**
- Oft wird das Ergebnis der Schätzung nicht akzeptiert
  - **sondern es gibt politischen Druck, die Schätzung zu verringern,**
  - **aber die künstlich verminderte Schätzung wird trotzdem zur verbindlichen Vorgabe an das Projektteam.**
  - **Dramatischste Ausprägung: "Todesmarsch-Projekte„**
    - **Schätzung liegt 50% oder mehr unterhalb "vernünftiger" Werte**

Zusammenfassung (1)

- **Ziel des Schätzens ist die Bestimmung des Entwicklungsaufwands abhängig von Systemkomplexität und Produktivität und zwar möglichst vor Systemrealisierung**
- **Typische Maße zur Systemkomplexität sind Lines-Of-Code (LoC) und Non-Comment-Source-Statements. Weitere Komplexitätsmaße sind Zyklomatische Zahl, Kopplung/Kohäsion, Vererbungstiefe.**
- Schätzungen werden fast immer aus einer Kombinationen der grundlegenden **Schätzmuster** hergeleitet
  - **Schätzen durch Vergleich**
  - **Schätzen durch Zerlegung (Anforderungen oder Entwurf)**
  - **Expertenschätzung**
  - **Schätzen mit Korrekturfaktoren**
  - **Schätzen mit Stellvertretergröße**

Zusammenfassung (2)

- **Wichtige Schätzverfahren sind das Delphi-Verfahren, Function Points und CoCoMo**
  - Das Delphi-Schätzverfahren ist eine Expertenschätzung, bei der Experten aus Erfahrung Voraussagen über den Zeit/Ressourcen-Bedarf einzelner Projektaktivitäten machen.
  - Mit Function Points schätzt man den Arbeitsaufwand in Mann-Stunden anhand der vom Kunden gewünschten Funktionen; dieses Verfahren ist bei einfachen (Informations-) Systemen gut anwendbar.
  - CoCoMo dient zur Schätzung der Projektgröße Size in LOC (Lines of Code) bzw. KDSI (Kilo Delivered Software Instructions),
    - CoCoMo I beruht auf der Schätzgleichung

      $$MM_{DEV} = a * Size^{b} * m$$

    - COCOMO II ist ein 3-Stufen Modell, das bei Entwicklungsfortschritt immer genauere Schätzungen ermöglicht