Aufwandsabschätzung (1)

„Die Bank GuterKunde GmbH will ein Online-Banking umsetzen. Es soll all die Funktionen haben, die ein Standard-Online-Banking bietet. Wie lange brauchen Sie dafür?“

- Einfache Frage, komplexe Antwort!
  - Aufwand (Kosten)
  - Entwicklungsdauer (Zeit)

Aufwandsabschätzung (2)

- Warum ist Aufwandsabschätzung so schwierig?
  - findet zu einem Zeitpunkt statt, zu dem i. d. R. noch sehr wenige Informationen bekannt sind
  - Selten schon Pflichtenheft, oft nur Anforderungsliste
  - Welche Schätzmethode ist gut?
  - Welche Schätzgrundlage wird verwendet?
- Dilemma: mit wenig Information ein brauchbares Ergebnis erhalten

Aufwandsabschätzung (3)

- Brauchbares Ergebnis ist sehr wichtig!
- auf der Basis der Aufwandsabschätzung wird das **Angebot kalkuliert**
- und der **Liefertermin fixiert**
- Trend: Fixpreiskalkulationen statt Abrechnung „nach Aufwand“

Aufwandsabschätzung: Bedeutung (1)

- Qualität der Aufwandsabschätzung hat einen hohen Einfluss
  - zunächst ob das Projekt / die Entwicklung überhaupt beauftragt wird
  - auf den finanziellen Erfolg der Projekts
  - die erfolgreiche und termingerechte Fertigstellung

Aufwandsabschätzung: Bedeutung (2)

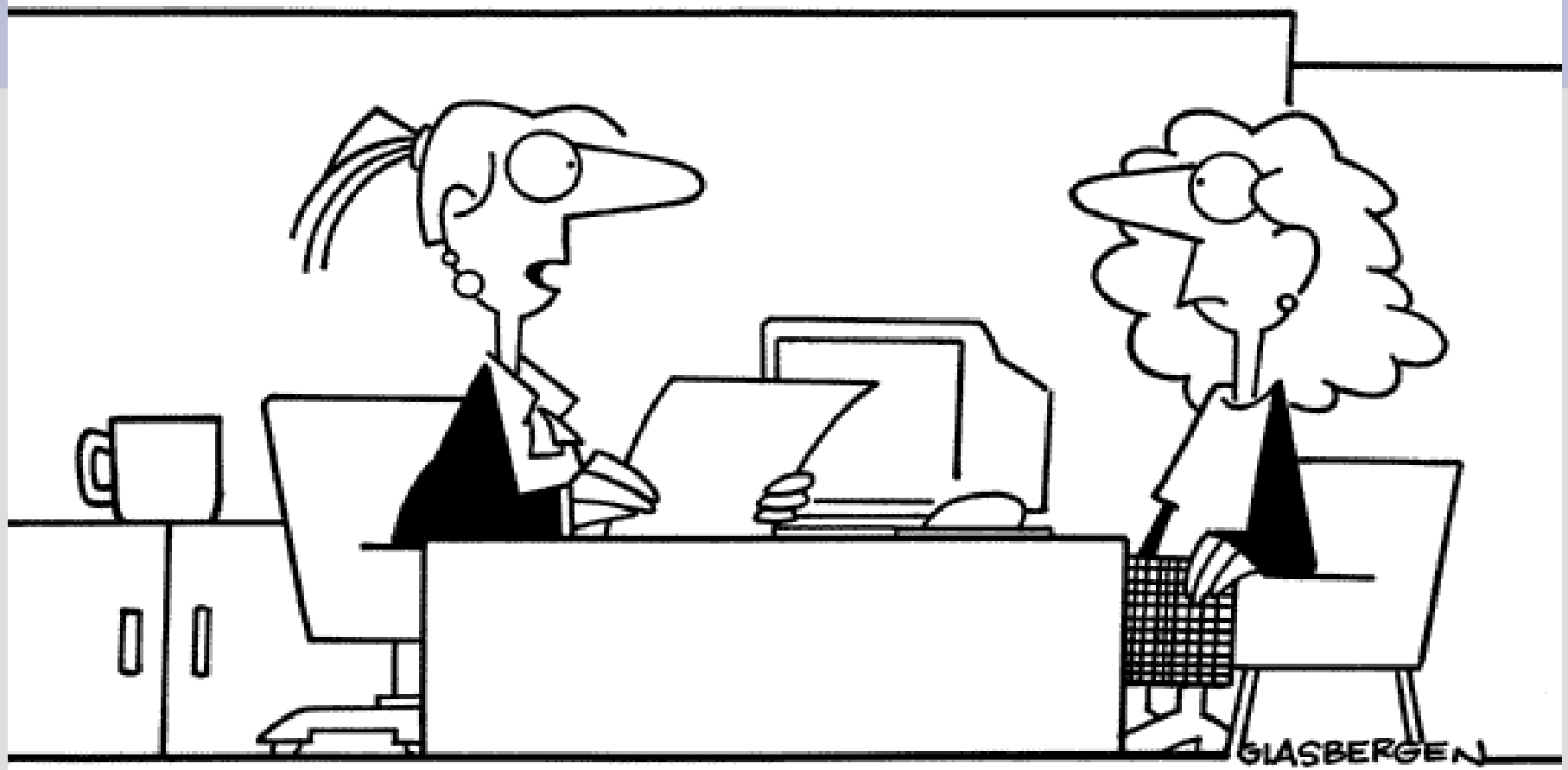
- „Nachverhandlung“ mit dem Auftraggeber ist in der Regel immer problematisch
- Je besser und detaillierter die Anforderungsanalyse, desto besser kann der Aufwand geschätzt werden.
- Leider wird oftmals gerade diese sehr schlampig durchgeführt!

Aufwand vs. Entwicklungsaufwand

- Aufwand bedeutet dabei nicht nur Entwicklungsaufwand

- Beinhaltet

  - Anforderungsanalyse und Pflichtenheft
  - Konzeptionsphase
  - Implementierungsphase
  - Tests
  - Organisation und Projektmanagement
  - Dokumentation



"We're only asking you to work 20 hours a week. To get that much done, you'll need to be here 80 hours a week."

8h Arbeitszeit sind nicht gleich 8h Arbeit!

„Das programmieren Sie doch locker an einem Nachmittag runter!“

- an einem Nachmittag programmiert (13-19 Uhr): **6 h**
- Programmierzeit entspricht aber nur 30 – 40 % des Gesamtaufwands hochgerechnet mit Konzeption, Test, Dokumentation **20h**
- 1h ungestörte Arbeitszeit entspricht 1,2 – 3h Anwesenheit (Telefon, Meeting, Service, Tagesgeschäft) – pauschal Faktor 2 **40h**

Einflussfaktoren (1)

- Quantitative Faktoren
  - Die tatsächliche Größe des Projekts, der Umfang der Funktionalitäten.
  - „Masseinheiten“:
    - LOC
    - function points
    - Personentage / -monate / -jahre

Einflussfaktoren (2)

- Qualitative Faktoren
  - Technische Komplexität, z.B. schwierige Algorithmen
  - Fachliche Komplexität, z.B. hochspezialiserte oder detaillierte Geschäftsprozesse, die gegebenfalls noch fachliche Einarbeitung erfordern

Einflussfaktoren (3)

- „Human resources“
  - Leistungsfähigkeit und Erfahrung der Mitarbeiter
  - Qualifikation und Potential der Mitarbeiter
  - Synergieeffektive in Teams: wie gut arbeiten die Teams zusammen?
- Sehr wichtige Faktoren, aber extrem schwer quantifizierbar!

Einflussfaktoren (4)

- Organisatorische Faktoren
  - gutes Projektmanagement
  - Belastung durch Tagesgeschäft
  - Möglichkeit, „Kern“ - Mitarbeiter vom Tagesgeschäft „freizuschaufeln“
  - Entwicklungsdauer, Projektgrösse: jeder zusätzliche Mitarbeiter verursacht zusätzlichen administrativen Aufwand (höherer Kommunikationsaufwand)

Schätzverfahren

- Selten: ein einziges Schätzverfahren
- i.d.R. Kombination mehrerer Verfahren

- Beispiele
  - Analogie- / Relationsmethode
  - „Aufwand-pro-Einheit“ - Methode
  - Prozentsatzmethode
  - Formelbasierte Schätzverfahren
    - COCOMO
    - Function Points

Analogie- / Relationsmethode

- umzusetzendes Programm weist hohe Ähnlichkeiten zu bereits durchgeführten Entwicklungen auf

- Erfahrungswerte
  - für Programmteile, die weitgehend übernommen werden können: ca. 25% Aufwand von Neuentwicklung

Analogiemethode Beispiel

- abgeschlossen: Internet-Browser in C++ unter Windows in 20 Monaten
- neues Produkt: Internet-Browser für Linux
  - 50 % des Codes wiederverwendbar
  - 50 % des Codes müssen überarbeitet werden
  - 20 % zusätzliche Neuentwicklung
- Schätzung:
  - 50 % wiederverwendbar: ¼ * 10 = 2,5 Monate
  - 50 % neu: 10 Monate
  - 20 % zusätzlich: 4 Monate
  - Sicherheits/Komplexitätszuschlag 2 Monate

- Summe: 18,5 Monate

Analogiemethode

- Probleme:
  - schwer nachvollziehbar
  - sehr grobe Schätzung
  - wie kommen die „50 % Neuentwicklung“ zustande – auch hier muss in irgendeiner Form geschätzt werden

- oftmals als Teilbestandteil für kleine Einheiten bei der „Aufwand-pro-Einheit“-Methode

„Aufwand-pro-Einheit“ - Methode

- Das System wird mit Hilfe der Anforderungen soweit es geht in kleinere Teile aufgeteilt – so lange, bis die Teile mit einem Aufwand bewertet werden können.
- Einzelteile oftmals mit Analogie- / Relationsmethode bewertet
- Aufwand = Summe des Aufwands der Einzelteile

„Aufwand-pro-Einheit“ - Beispiel Online-Banking (1)

- Anforderung „Durchführung einer Überweisung“
- Teilanforderung 1:
  - Der Bankkunde kann eine Überweisung von seinem Konto auf ein beliebiges anderes Konto durchführen.
- Teilanforderung 2:
  - Die zu verarbeitenden Daten entsprechen denen eines herkömmlichen Überweisungsbelegs (Kontoinhaber, Kontonummer des Empfängers, Empfänger, ...).
- Teilanforderung 3:
  - Die Oberfläche soll identisch aussehen wie ein herkömmliches Überweisungsformular.

„Aufwand-pro-Einheit“ - Beispiel Online-Banking (2)

- Einheiten bilden und Schätzen
  - Design der Oberfläche im HTML-Editor: 2 Tage
  - Konsistenzprüfungen auf der Oberfläche: 1 Tag
  - Konsistenzprüfungen auf der Serverseite: 1 Tag
  - Geschäftslogik: Durchführen der Überweisung: 3 Tage
  - Berücksichtigung der Mehrsprachigkeit: 2 Tage
  - Test: 5 Tage
  - Dokumentation: 2 Tage
  - .....

„Aufwand-pro-Einheit“ - Vorteile

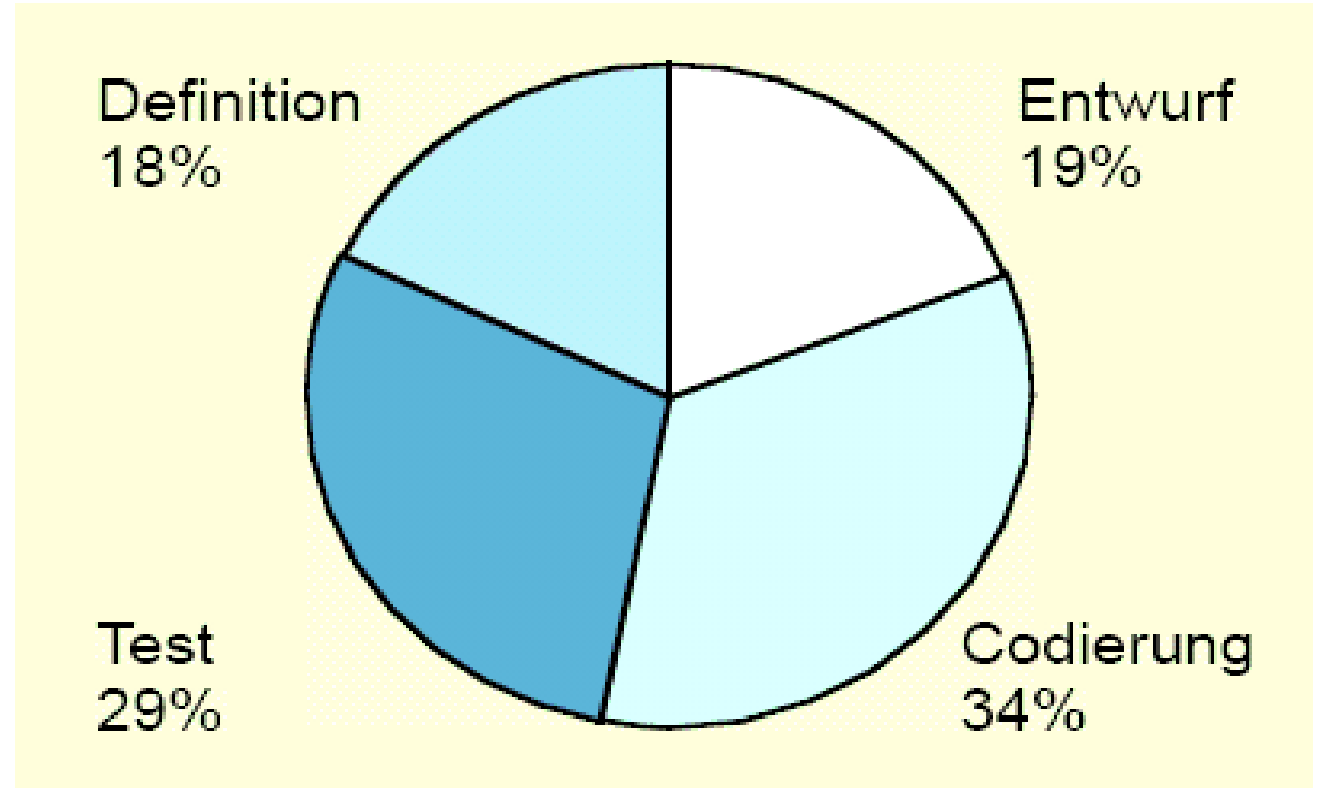
- detaillierte Betrachtung des Aufwands
- Analogiemethode für die Teilbausteine gut anwendbar
- Bereits im Vorfeld intensive Analyse der Anforderungen

„Aufwand-pro-Einheit“ - Nachteile

- Baukastenprinzip vernachlässigt Abhängigkeiten und Synergien zwischen Programmteilen
- Anforderungsänderungen sind nicht eingearbeitet
- bei großen Projekten hoher Aufwand für Schätzung

Prozentsatzmethode (1)

- Aus früheren Projekten: Ermittlung, wie der Aufwand sich auf die einzelnen Entwicklungsphasen verteilt hat



Prozentsatzmethode (2)

- Neuentwicklungen:
  - entweder eine Phase zunächst vollständig abschließen und dann den Aufwand hochrechnen
  - oder detaillierte Schätzung einer Phase und dann Hochrechnung auf die anderen Phase
- Vorteil: frühzeitig einsetzbar
- Nachteile:
  - sehr ungenau
  - unterschiedliche Komplexität und Anforderungen in Projekten → Verschiebung des Aufwands für die Phasen

Formelbasierte Schätzverfahren COCOMO (Basis) (1)

- Constructive Cost Model (Barry Boehm, University of Southern California)
- Unterscheidung der Projekte in drei Kategorien
  - einfach
  - mittelschwer
  - komplex

COCOMO (Basis) (2)

- Der Aufwand wird in Abhängigkeit von LOC angegeben:

$$A = C * KLOC^{B}$$

mit A = Aufwand in Monaten

	C	B
Einfach	2,4	1,05
Mittel	3,0	1,12
Komplex	3,6	1,2

COCOMO (Basis) Beispiel

- Aufwand von 80 Tagen (= 4 Monate),
  - implementiert serverseitig in Java
  - clientseitig in Delphi
  - Mittelschweres Projekt

- Java: ca. 4000 LOC
- Delphi: ca. 1600 LOC => Summe **5600 LOC** (logical)
- zzgl. Oberflächendesign „ohne Code“ ca. 5 Tage.

- Aufwandsberechnung:

$$3{,}0 * 5{,}6^{1{,}12} = 20{,}65 \text{ Monate}$$

- Annahme: einfaches Projekt:

$$2{,}4 * 7{,}4^{1{,}05} = 14{,}65 \text{ Monate ????}$$

COCOMO (Basis) Probleme (1)

- Woher kommen die Faktoren, wieso sind sie allgemeingültig?
- LOC überhaupt noch geeignetes Kriterium?
  - Codephase teilweise unter 40 % des Gesamtaufwands, vor 20 Jahren noch über 60 %
  - Oberflächendesign
  - Komponentenbasierte Software: Konfiguration statt Codierung (Delphi, JavaBeans)

COCOMO (Basis) Probleme (2)

- LOC überhaupt noch geeignetes Kriterium?
  - Automatisierung: Tools übernehmen Generierung von einfachem Code (Objektcontainer, Datenbankzugriffe, Persistenzmodelle)
  - „physikalische“ (alles ohne Leerzeilen und Kommentare) oder „logische“ (tatsächliche Anzahl an Anweisungen) LOC?

COCOMO (Basis) Probleme (3)

- LOC überhaupt noch geeignetes Kriterium?
  - mehrere physikalische LOC in einer logischen LOC möglich
  - mehrere logische LOC in einer physikalischen LOC möglich
  - LOC müssen auch geschätzt werden. Wie diese schätzen? Und warum dann nicht gleich den Zeitaufwand mit anderen Methoden direkt schätzen, sondern den Umweg über LOC (Mehraufwand, doppelte Schätzung – höhere Ungenauigkeit?) ?

COCOMO II

- umfangreiches Modell (Basis COCOMO)

- Aufwand ist abhängig von vielen Einzelfaktoren
- Erfahrungswerte der Programmierer
  - Datenbankgröße
  - Ergonomie bei der Arbeit
  - Einsatz von Tools
  - ....
- Formel:

$$A = C * KLOC^{B}$$

$$C = C_1 * ..... * C_k$$

  - statistische Schätzverfahren für die Ermittlung der C_i

COCOMO II Probleme

- hoher Aufwand für die Ermittlung der Koeffizienten
- keine ausreichende Stichprobe für die Schätzung
  - Zeitnahe Projekte
  - in Struktur ähnlich
  - nur bei Großkonzernen ist das einigermaßen gewährleistet.
- Für kleinere Unternehmen mathematisch nicht aussagekräftig
- Problem LOC nicht gelöst – siehe oben

Function Points (1)

- Nicht LOC sind Basis der Schätzung, sondern die Anforderungen
- Vorgehensweise
  - Kategorisierung jeder Anforderung in 5 Gruppen (Eingaben, Ausgaben, Abfragen, Applikationsdaten, Referenzdaten)
  - Klassifizierung der Anforderung in einfach, mittel, schwer
  - Bewertung jeder Anforderung mit Gewichtungsfaktoren

Function Points (2)

- Vorgehensweise (Forts.)
  - Bewertung der Einflussfaktoren
  - Ermittlung der Summe der Function Points
  - Ablesen des Aufwands aus einer Tabelle
  - Bei Projektende: Schätzgrundlagen anpassen mit aktuellen Werten

Kategorie	Klassifizierung	Gewichtung	Anzahl	Betrag
Eingaben	einfach	3		=
	mittel	4		=
	komplex	6		=
Ausgaben	einfach	4		=
	mittel	5		=
	komplex	7		=
Abfragen	einfach	3		=
	mittel	4		=
	komplex	6		=
Applikationsdaten	einfach	7		=
	mittel	10		=
	komplex	15		=
Referenzdaten	einfach	5		=
	mittel	7		=
	komplex	10		=
			Summe E1der ungewichteten FP	=
Einflussfaktoren	1. Backup / Recovery erforderlich (0-5)			=
	2. Datenkommunikation erforderlich			=
	3. Verteilte Verarbeitung			=
	4. Kritische Performance			=
	5. Komplexe, stark benutzte Nutzungsumgebung			=
	6. Online Dateneingabe benötigt			=
	7. Transaktionsverarbeitung			=
	8. Online-Update benötigt			=
	9. Komplexe Ein-/Ausgaben, Abfragen			=
	10. Komplexe interne Logik			=
	11. Wiederverwendbarkeit des Codes			=
	12. Konvertierung und Installation wichtig			=
	13. Mehrere Installationen, mehrere Kunden			=
	14. Benutzerfreundlichkeit			=
			Summe E2 der Einflüsse	=
			Einflussfaktor E3 = 0.65 + 0.01 x E2	=
			***Function Points* FP= E1 * E3**	=

(Beispiel von M. Kropp, FH Solothurn)

Function P.	IBM-PM	VW-PM	Function P.	IBM-PM	VW-PM	Function P.	IBM-PM
50	2,3	–	1200	145,2	207,8	3300	547
100	5,6	–	1300	161,3	237,8	3400	568,8
150	9,5	–	1400	177,7	273,2	3500	590,8
200	13,9	11,7	1500	194,6	319,1	3600	613,1
250	18,6	19,3	1600	211,7	–	3700	635,5
300	23,6	27,1	1700	229,3	–	3800	658,1
350	28,9	35	1800	247,1	–	3900	680,9
400	34,4	43	1900	265,3	–	4000	703,9
450	40,1	51,1	2000	283,7	–	4100	727
500	46,1	59,6	2100	302,4	–	4200	750,4
550	52,2	68,2	2200	321,5		4300	773,9
600	58,5	77	2300	340,7		4400	797,5
650	65	86,1	2400	360,3		4500	821,4
700	71,6	95,3	2500	380,1		4600	845,4
750	78,4	104,8	2600	400,1		4700	869,6
800	85,3	114,6	2700	420,4		4800	893,9
850	92,4	124,7	2800	441		4900	918,4
900	99,6	135,2	2900	461,7		5000	943,1
950	106,9	146	3000	482,7		5100	967,9
1000	114,4	157,3	3100	503,9		5200	992,8
1100	129,6	181,3	3200	525,4			

(Aus "Lehrbuch der Software Technik", Bd. 2, Helmut Balzert)

Function Points
Vorteile

- klar definierte Vorgehensweise
- gute Vergleichsmöglichkeit
- Anforderungen im Vordergrund, keine LOC

Function Points
Nachteile

- mathematisch nicht solide fundiert: gaukelt eine Exaktheit vor, die so gar nicht existiert
- Anforderungen nicht immer eindeutig einer Kategorie zuordenbar
- bei großen Projekten hoher Aufwand für Schätzung
- Kategorienaufgliederung veraltet - nach modernen OO-Kriterien nicht nachvollziehbar
- Baukastenprinzip vernachlässigt Abhängigkeiten und Synergien zwischen Programmteilen
- Anforderungsänderungen sind nicht eingearbeitet