7. ESTIMERING

Estimering er egentlig en del af planlægningen, men da estimeringsaktiviteterne "fylder" meget i it-projekter, bliver de her beskrevet i et kapitel for sig selv. Og dette kapitel er lidt specielt i forhold til så mange af de andre fordi vi her bliver nødt til at introducere lidt mere matematik. Det må du bære over med – eller springe over.

Tidsestimering drejer sig om at vurdere hvor lang tid ting tager. Dette er en af de vanskeligste dele af projektledelsen. Det skyldes at estimeringen ikke – som de andre aktiviteter i systemudviklingen – opfattes som håndværk. Mange mener at estimering bygger på erfaring og intuition (selv om der findes udmærkede metoder). Så der er basis for forbedring på dette område.

Vi skelner mellem gæt, kvalificeret gæt og estimat.

- ET GÆT er en udtalelse på basis af en formodning. Det er ikke nogen særligt videnskabelig fremfærd, og de fleste udviklere er da heller ikke meget for ligefrem at gætte. Altså at udtale sig om noget de ikke føler de har en direkte viden om. Skal det lyde lidt finere kaldes det også lidt humoristisk: et *gæstimat* (eng.: guesstimate).
- ET KVALIFICERET GÆT er derimod baseret på erfaring og viden inden for området. Så det er altså ikke blot baseret på rene formodninger, men dog heller ikke underbygget af nogen bestemt metode eller teknik. Det er det kvalificerede gæt der oftest bruges når udviklere skal udtale sig om hvor lang tid et givet it-projekt vil tage. Det kvalificerede gæt kaldes også *ekspertgæt*.
- ET ESTIMAT er et kvalificeret gæt som er underbygget af en metode, og som derigennem giver et resultat, samt en bestemt usikkerhed som det pågældende resultat er behæftet med.

Regel nummer ét er at hvis vi ikke ved hvad vi skal lave, kan vi heller ikke vide hvor lang tid det tager. Vi kan selvfølgelig have en mening om det – et gæt – men det vil være behæftet med en voldsom usikkerhed. Først når vi har fastlagt kravene og bedømt de involverede risici, kan vi indsnævre usikkerheden. Eller sagt på en anden måde: Usikkerheden på estimatet er ligefrem proportional med usikkerheden på det vi estimerer. Men usikkerheden bliver kun mindre – den forsvinder ikke. Selvom vi kender kravene i detaljer, kan vi ikke give et helt præcist estimat – det er derfor det hedder et estimat.

Regel nummer to er at estimatet skal dokumenteres, dvs. det skal være gennemsigtigt hvorfor vi netop kommer frem til det pågældende estimat. Det medvirker også til

at gøre estimatet troværdigt. Vi kan f.eks. hænge det op på noget konkret. Som vi senere skal se kan vi bruge mange ting til at hænge estimatet op på: krav, use cases, funktioner, skærmbilleder, tabeller, kodelinjer osv. Det vigtige er at vi hænger estimatet op på nogle enheder som kan tælles (som de netop nævnte). Tælleniveauet kan være forskelligt: vi kan tælle hele applikationer, vi kan tælle skærmbilleder, eller vi kan tælle antallet af kontroller på skærmbillederne.

Hvornår estimeres?

Estimeringen foretages ad flere omgange. Første gang er som regel umiddelbart efter at kravspecifikationen foreligger. Ud fra de opstillede krav foretages et såkaldt *grov-estimat*. Usikkerheden på dette estimat er relativt stor, men det kan dog bruges til at udarbejde en grov-tidsplan, og det kan indgå i beslutningsgrundlaget for hvorvidt projektet skal startes.

Desværre ser man også at projektlederen afkræves et estimat *før* kravspecifikationen foreligger (sådanne estimater kaldes spøgefuldt for SPT-estimater, Slag På Tasken). Her skal man træde varsomt som projektleder. Du skal vide at det første tal der forlader dine læber, vil blive husket til evig tid. Hvis du derfor bliver afkrævet et gæt på dette meget tidlige tidspunkt hvor usikkerheden er stor, er det et godt tidspunkt at indkalkulere denne usikkerhed. Hvis sælgeren selv har et gæt, så bed ham om at bruge det. Hvis han ikke har et gæt, og du i øvrigt er usikker, så lad dit gæt afspejle den store usikkerhed ved at gange dit første gæt med en faktor 3-10. Du kan altid bagefter nedjustere. Det er derimod langt sværere at opjustere.

På grundlag af kravspecifikationen kan estimeres med enten Function Points-teknikken eller Use Case-Points-teknikken som vi begge vender tilbage til senere i dette kapitel (side 225).

Næste gang der estimeres, er efter uddybningsfasen, planlægningsfasen eller hvad den hedder i de forskellige udviklingsmodeller. Nu sker det på grundlag af en dybere forståelse, og der kan tages hensyn til den reelle bemanding. Usikkerheden på estimatet bliver således langt mindre.

Vi vil i det følgende beskrive:

- Politiske estimater
- Et par populære måder at estimere på
- Hjælpeteknikker som bruges under estimeringen
- Hvordan organiseres estimeringen, dvs. hvem foretager den?
- Hvordan opsamles erfaringerne fra estimeringen til senere brug?
- Konkrete estimeringsteknikker

7.1 Politiske estimater

Man skelner skarpt mellem de teknisk funderede estimater og de politiske "estimater" (også kaldet politiske diktater).

Politiske estimater er estimater der udspringer af andre forhold end det rent tekniske i projektet, f.eks.:

- VINDENDE PRIS, altså et estimat der bygger på hvor meget man højst må bruge hvis man overhovedet vil have kontrakten. Er vi for dyre så løber konkurrenterne med den.
- KAPLØB MED TIDEN. Konkurrenceforhold kan afgøre hvor lang tid vi må bruge før produktet ikke længere er konkurrencedygtigt.
- FAST DEADLINE. Der kan være tale om betonmurs-deadlines, f.eks. hvis produktet *skal* præsenteres på en messe der holdes på et bestemt tidspunkt. Kommer produktet ikke med, er en masse markedsføring spildt. Eller produktet skal måske være færdigt til produktionsstart på et bestemt tidspunkt. I den iterative terminologi svarer det til time boxing.
- FASTE RESURSER. Dette er estimatet som bygger på hvor mange resurser vi har til rådighed. Det svarer til når børn i slikbutikken spørger: – Hvad kan jeg få for en femmer? Eller måske til at pege på et bestemt stykke (dyrt) slik og sige: – Jeg vil have det der – men jeg har kun en femmer. I den iterative terminologi svarer det til money boxing.
- INVESTERING. Endelig er der satsninger på at hvis vi tilbyder udviklingen til denne alt for lave pris, så kan vi senere tjene det ind igen på vedligeholdelse og betalt fejlretning. En slags "vi kan tjene ind på gyngerne hvad vi har mistet på karrusellen".

Projektdeltagerne behøver ikke at blive sure over de politiske estimater. Der kan være særdeles gode grunde til at lave dem. Man skal blot være klar over at de altså ikke er estimater i teknisk forstand.

Disse politiske estimater skal ikke foretages af projektgruppen. Det er projektgruppens opgave at foretage teknisk velfunderede estimater. På baggrund af disse kan salgschef, produktchef, udviklingschef osv. så lave de politiske estimater.

Disse hænger i sagens natur ikke altid sammen med de tekniske estimater, og der skal derfor foretages en revurdering af disse. F.eks. ved at nedsætte ambitionsniveauet, udvide projektgruppen osv. Man må *ikke* blot affinde sig med at arbejde ud fra det politiske estimat – i stedet skal opgaven ændres så den kan løses inden for det politiske estimat. Øvelsen går altså ud på at få det tekniske estimat til at falde sammen med det

politiske. Alt andet er strudsetaktik hvor man lukker øjnene og først opdager problemerne til sidst.

I nogle organisationer er man meget klar over denne skelnen. Her har man to adskilte budgetter: et salgsbudget og et produktionsbudget. Salgsbudgettet tager højde for de salgsmæssige og konkurrencemæssige faktorer, inkl. en eventuel senere indtjening, produktionsbudgettet er det teknisk korrekte budget for udviklingen.

I øvrigt kan udviklerne også præstere politiske estimater. De findes i to udgaver:

- GUF DE LUXE. Hvis udviklerne meget gerne vil have igangsat et spændende, nyt projekt med en spændende, ny teknologi eller et spændende, nyt værktøj, så kunne de måske godt finde på at estimere en anelse lavere. Sæt nu ledelsen ikke ville acceptere projektet fordi det blev for dyrt.
- SKODOPGAVE. Eller hvis de omvendt meget nødigt vil lave et bestemt projekt fordi det er en kedelig opgave, en besværlig kunde etc. Så kunne de fristes til at give et alt for højt estimat.

Denne adfærd med skjult dagsorden er kritisabel, men den forekommer.

7.2 Trepunktsestimering

Denne teknik er meget populær. Den går ud på at estimere tre tal for hver aktivitet:

- a: det *korteste* man kan forestille sig det kan laves på
- b: det mest *sandsynlige*
- c: det *længste* det kan komme til at tage

På denne måde får vi indkalkuleret den usikkerhed der altid er i et estimat.

Nogle går i baglås over at skulle gætte på værdierne for "det korteste" og "det længste". De siger f.eks.: – Det korteste er hvis vi har det hele i forvejen, så det er 0! Eller: – Det længste er hvis virksomheden brænder, og vi skal begynde helt forfra. Men det er selvfølgelig ikke meningen. Det korteste er hvis udviklingen kører lige efter en snor helt uden uheld. Og det længste er hvis "de normale" ting går galt. Men naturkatastrofer, guddommelig indgriben og lignende tæller altså ikke med. Det er her vi har brug for resultatet af risikoanalysen.

Ud fra disse tre tal beregnes så middelværdien, m, standardafvigelsen (spredningen), s, og variansen, v, for hver enkelt aktivitet således:

$$m = \frac{a + 3b + c}{5} \qquad s = \frac{c \div a}{5} \qquad v = s^2$$

Hvis du er statistisk interesseret, så ser formlerne sådan ud fordi der er tale om en Erlang-fordeling (se f.eks. [Lichtenberg00]).

Man fortsætter nedbrydningen af de indgående aktiviteter til standardafvigelsen kommer ned på et acceptabelt niveau. Standardafvigelsen afslører nemlig de steder hvor usikkerheden er stor. Teknikken kaldes også successiv kalkulation (eller successivprincippet), netop fordi vi gennem en række successive nedbrydninger laver en trinvis forfining af estimatet.

Det totale estimat for hele projektet, M, fås da ved at summere de enkelte middelværdier:

$$M = \sum m_i$$

og den tilsvarende standardafvigelse, S, fås med:

$$S = \sqrt{\sum v_i}$$

Det kan f.eks. se således ud:

Fig. 7.1: Eksempel på trepunktsestimat

Akvivitet	**a**	**b**	**c**	**m**	**s**	**v**
A	1	4	10	4,6	1,8	3,24
B	10	14	25	15,4	3,0	9,0
C	8	12	27	14,2	3,8	14,4
D	3	9	15	9,0	2,4	5,76
I alt				**43,2**		**32,4**
				S:	**5,7**	

Nogle projektplanlægningsværktøjer har indbygget trepunktsestimering og kan lave denne beregning automatisk. Ellers er det let at lave i et regneark.

Som bekendt siger standardafvigelsen noget om med hvilken sandsynlighed den faktiske størrelse vil ligge inden for en vis afstand af estimatet.

Med 68% sandsynlighed vil den faktiske størrelse ligge inden for en afstand af 1 × standardafvigelsen fra estimatet. Eller med andre ord: sandsynligheden for at

$$M \div S < \text{faktisk} < M + S$$

er 68%. Det pågældende interval kaldes også "68% konfidensintervallet".

Tilsvarende er 95 % konfidensintervallet

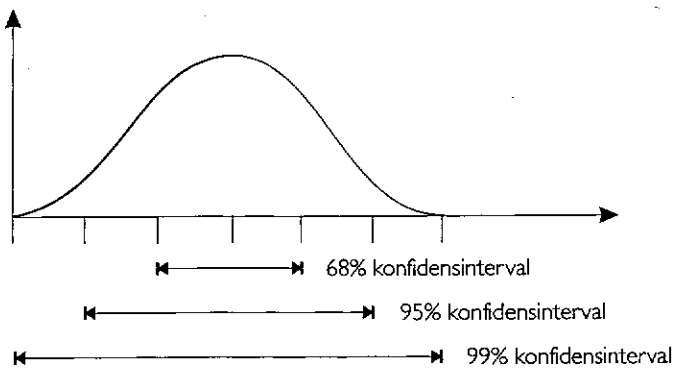
$$M \div 2\,S < \text{faktisk} < M + 2\,S$$

og endelig er 99% konfidensintervallet

$$M \div 3\,S < \text{faktisk} < M + 3\,S$$

Illustreret grafisk ser det således ud:

Fig. 7.2: Konfidensintervaller



Man kan også gå den anden vej: Hvis man ønsker at lægge sig fast på et bestemt konfidensinterval kan man beregne de tilhørende værdier. Hvis man f.eks. ønsker at konfidensintervallet skal være 85%, svarer det til:

$$M \div 1{,}44 \times S < \text{faktisk} < M + 1{,}44 \times S$$

Dette er dog ikke helt trivielt at beregne og kræver lidt statistikkendskab.

Med tallene fra ovenstående eksempel får vi følgende værdier:

Fig 7.3: Eksempel på konfidensintervaller

Interval	Min	M	Max	x	x × s
68%	37,5	43,2	48,9	1	5,7
85%	35,0	43,2	51,4	1,44	8,2
95%	31,8	43,2	54,6	2	11,4
99%	26,1	43,2	60,3	3	17,1

Du kan komme ud for at nogle bliver irriterede, når du siger at: Med 68% sandsynlighed tager dette mellem 37 og 49 timer. De svarer måske: – Kan du ikke bare give mig et tal? Her har du en pædagogisk udfordring i at forklare at dette altså ikke er spådomskunst som med usvigelig sikkerhed kan ramme det rigtige tal. Det er et forsøg på at udsige noget rimeligt pålideligt om fremtiden. Og det *er* altså kompliceret, det her.

Denne trepunktsestimering blev først brugt i PERT-teknikken som vi vender tilbage til på side 248 i kapitlet om Planlægning, og du kan læse mere om den i [Lichtenberg00].

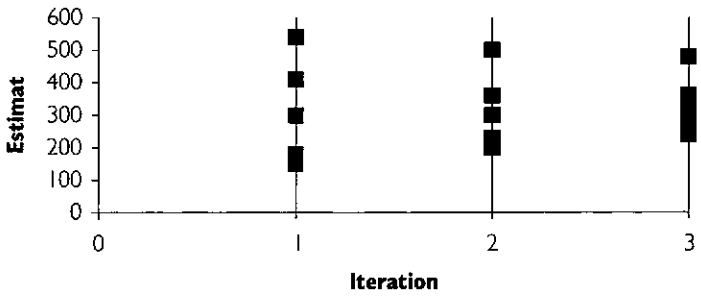
7.3 Ekspertgæt

Denne teknik går ud på at nogle eksperter finder frem til deres bedste bud på omfanget. Eksperterne vurderer først anonymt og hver for sig hvorefter de kan se hinandens estimater og eventuelt prøve igen i et antal iterationer hvor de mødes for at diskutere sig frem til et godt estimat. Bemærk at der *aldrig* bør opnås enighed ved at tage gennemsnit af eksperternes estimater. Det skyldes at hvis nogen har "ramt ved siden af", så er det ofte fordi der er noget de ikke ved. Hvis de har skudt for højt, kan de f.eks. være uvidende om at en given komponent allerede findes. Og hvis de har skudt for lavt, kan der være problemer ved domænet eller opgavens sværhedsgrad de ikke har erkendt. Og begge dele er lige slemt. Af samme grund skal eksperterne have kendskab til den tidligere foretagne risikoanalyse – og gerne selv have deltaget i udarbejdelsen.

Lykkes det ikke at få indsnævret forskellen mellem estimaterne ved et passende antal iterationer, må man enten få fat i en opmand som kan dømme nogle af estimaterne "ude". Eller man må erkende at usikkerheden på estimatet altså *er* så stor.

Man kan plotte resultaterne fra hver iteration ind på en graf for at give deltagerne mulighed for at se hvor de ligger i forhold til de andre:

Fig. 7.4: Ekspertvurderinger



Ekspertvurdering kaldes også *Delfi-metoden* efter det klassiske Grækenlands *Oraklet i Delfi* hvor præstinderne gav (ofte flertydige!) svar på de spørgsmål de fik stillet om hvad der ville ske i fremtiden. Metoden har altså intet at gøre med programmeringssproget af samme navn. Som estimeringsmetode blev den opfundet hos *Rand Corporation* i USA allerede i 1948. Senere er den blevet forfinet af bl.a. Barry Boehm (se f.eks. [Boehm81], Wideband Delphi) hvor bl.a. muligheden for at eksperterne kunne mødes for at diskutere estimaterne, blev indført.

Ekspertgæt er ikke en egentlig estimeringsteknik. Vi vil dog alligevel se på hvordan et ekspertgæt kan gøres en lille smule mere "eksakt".

Ud fra kravspecifikationen laves en funktionel nedbrydning (vi vender senere tilbage til dette i afsnittet om Aktivitetsnedbrydning på side 239). Der er en øvre grænse for omfang som ingen aktivitet bør overskride – den kaldes estimeringstærsklen. På store projekter kan denne grænse gå op til både 60 og 90 persontimer (svarende til 2-3 personuger), mens man i mindre projekter formentlig vil holde sig under 20 persontimer. Nedbrydningen fortsættes indtil de enkelte komponenter vurderes til maksimalt denne estimeringstærskel.

Bemærk at det *kan* være et advarselstegn hvis eksperternes estimater er ækvidistante (f.eks. i multipla af 10, 30 eller 100 persontimer). Det kan muligvis være et tegn på at usikkerheden på estimatet er for stor – og at de i virkeligheden har brugt en anden målestok (f.eks. personuger eller personmåneder).

Ofte estimeres kun de tekniske aktiviteter. De administrative aktiviteters omfang beregnes derefter som en fast procentsats af de tekniske aktiviteters omfang, f.eks. 25%. I nogle estimeringsteknikker er den administrative tid dog også med i estimatet.

Det følgende eksempel viser et lille udsnit af et skema til ekspertgæt for et system, foretaget ud fra kravspecifikationen. De angivne tider er persontimer.

Fig. 7.5: Nedbrydning

Tekst		Subtotal	Timer
Funktion 1			**8**
1.1		1	
1.2		2	
1.3		5	
Funktion 2			**11**
2.1		3	
2.2		8	
Funktion 3			**4**
3.1		4	
Hjælp-system			1
Installation			**24**
Installation Shield		3	
Udarbejdelse af vejledning		9	
Installation hos kunden		12	
Kurser			6
Kodning i alt			54
Arkitektur, design	15% af kodning		**8**
Implementation i alt			62
Projektledelse	10% af implementation		6
Systemtest	25% af implementation		15
SW-dokumentation	10% af implementation		6
Accepttest, overdragelse			8
Nedlukning, arkivering			2
Garantiforpligtelser	10% af implementation		6
Antal timer i alt			**105**

Ved udarbejdelse af ekspertgæt kan man med fordel benytte sig af et almindeligt, pc-baseret regnearksprogram. Man kan f.eks. gætte på tiderne til selve kodningen, hvorefter man lader regnearksprogrammet foretage summeringen samt udregne tiden til design, test, administration osv. som faste procentsatser.

7.4 Hjælpeteknikker

Til brug ved estimeringen kan anvendes en række hjælpeteknikker, som f.eks.:

- NEDBRYDNING. Programmet der skal estimeres, brydes ned i en række mindre enheder, idet det er lettere at ramme rigtigt på små enheder. Man kan enten nedbryde i funktionelle enheder (som i afsnittet om Ekspertgæt på side 215) eller i software-enheder. Hvis estimeringen foretages umiddelbart efter kravspecifikationsfasen, må man nedbryde i funktionelle enheder, mens man kan nedbryde i software-enheder (klasser eller moduler) efter procesdesignfasen. Nedbrydningsteknikken kaldes også bottom/up-estimering, i modsætning til COCOMO og Function Points-teknikkerne som kaldes top/down-estimering. Nedbrydning er mere detaljeret beskrevet på side 239 i kapitlet om Planlægning.
- ITERATION. Består i at gentage estimeringen. Det er specielt nyttigt hvis flere personer har lavet hver deres estimat. På denne måde får de andres estimat indflydelse på ens eget estimat. Estimeringen bør også gentages hen igennem projektforløbet, idet estimatet bliver nøjagtigere, jo længere man kommer frem i processen.
- ANVENDELSE AF FLERE ESTIMERINGSTEKNIKKER. Ved at anvende flere teknikker kan man opveje en enkelt tekniks svagheder. Man kan f.eks. estimere det samme modul både med fremskrivning, analogi og parametriske metoder (se nedenfor). Får man (næsten) samme resultat, er alt godt. Får man derimod meget afvigende estimater, bør man tænke sig om en ekstra gang.

Desuden er det en særdeles god ide at anvende egentlige softwareværktøjer til brug for estimeringen. I [Jones96] og [Jones98] dokumenteres det at estimaterne bliver væsentligt bedre ved anvendelse af værktøjer. Kan du ikke få bevilget et estimeringsværktøj, så begynd for dig selv med et regneark.

7.5 Estimeringsteknikker

Estimeringsteknikker kan groft inddeles i følgende kategorier:

- fremskrivning,
- analogivurdering og
- parametriske modeller.

7.5.1 Fremskrivning

Fremskrivning er en teknik hvor man ser på hvor langt man er nået i projektforløbet, og hvor stor en procentdel af resurserne dette plejer at tage. Herudfra ekstrapolerer man så. Hvis man f.eks. har overstået kravspecifikationsfasen på 2 uger, og denne fase plejer at tage 10% af den samlede udviklingstid, estimerer man hele projektet til 20 uger og dermed *resten* til 18 uger.

Teknikken kan først bruges når man er kommet et stykke hen i forløbet, og estimatet bliver bedre, jo længere man er kommet hen.

Der findes mange forsøg på at sætte procenter på faserne. Sådanne procenter skal altid tages med et gran salt, idet de er afhængige af det pågældende firma, den benyttede udviklingsmodel, typen af projekter osv. De kan f.eks. se således ud:

Kravfase	20%
Design	35%
Kodning	15%
Test	30%

Men det skal endnu engang understreges at disse procenter kun er vejledende, og at man selv, gennem opsamling af data fra firmaets egne projekter, skal justere tallene.

Brug af fremskrivning forudsætter ensartethed i projekterne, og den er derfor vanskelig at bruge hvis man laver mange forskellige typer af projekter.

7.5.2 Analogivurdering

Analogi benytter erfaringer fra ét eller flere tidligere, konkrete aktiviteter der "ligner" den aktuelle aktivitet. Hvis du f.eks. skal forudsige morgendagens vejr, så gæt på at det vil blive det samme som i dag. Der er faktisk 70% sandsynlighed for at du rammer rigtigt! ([Beck01])

Ved analogivurdering kigger man på hvor lang tid det/de pågældende aktiviteter faktisk tog, og vurderer herudfra hvor meget den aktuelle aktivitet vil tage. Det er naturligvis vigtigt at man bruger det faktiske tidsforbrug og ikke det oprindelige estimat.

Man skal tage hensyn til hvor meget den aktuelle aktivitet ligner den aktivitet man sammenligner med. Og om der eventuelt er andre forskelle, f.eks. hvilke personer der skal udføre aktiviteten denne gang.

Ved analogivurdering er det bedst hvis man har en erfaringsbase, men det vigtigste er dog erfaringen og intuitionen hos den der foretager estimatet, idet man dels skal vide hvad "ligne" vil sige, dels have kendskab til de tidligere projekter.

Der kan eventuelt blive tale om at skalere det tidligere projekts forbrug. Hvis du i det første projekt skulle konvertere halvt så mange tabeller som i dette, så tager dette måske dobbelt så lang tid.

Sammenlign i øvrigt med diskussionen om overtiden kontra idealtiden som blev omtalt i afsnittet om Læreprocesserne (side 52). Vi bliver bedre og bedre når vi bevæger os inden for *det samme* område.

Analogivurdering kan i øvrigt også bruges sammen med iterative modeller. I eXtrem Programmering anbefales det således at hvis man ikke umiddelbart har noget at sammenligne med kan man lave såkaldte *spikes*, dvs. hurtige, målrettede eksperimenter for at afklare et område.

7.5.3 Parametriske modeller

Parametriske modeller er fællesbetegnelsen for en række teknikker hvor man vurderer en række af de parametre der hver for sig har indflydelse på estimatet. Herefter beregner man det endelige estimat ved hjælp af en formel.

Parametre benyttes i mange "officielle" estimeringsmetoder, f.eks. Barry Boehms COCOMO-metode (se næste afsnit). Disse metoder er tiltrækkende fordi de ser "eksakte" ud: man sætter nogle tal ind i en formel, og ud kommer et færdigt estimat.

Erfaringen viser at de er svære at komme i gang med, og at de kræver kalibrering til de faktiske forhold (også her er en erfaringsbase altså af betydning). Endvidere viser erfaringen at uanset om man bruger det færdige estimat eller ej (uanset om man "tror" på det eller ej), så er selve det at *bruge* en metode af denne type af betydning. Det skyldes at man bliver tvunget til at lave en detaljeret vurdering af de indgående parametre og dermed bliver bevidst om deres betydning for det endelige estimat.

De næste afsnit handler om parametriske metoder. Først beskrives *COCOMO II* som bygger på en vurdering af det forventede antal kodelinjer, samt en vurdering af en række forskellige parametre som hver især har indflydelse på udviklingstiden.

Dernæst beskrives *Function Point-metoden* som bygger på at kravene fra kravspecifikationen tildeles point som derefter justeres.

Endelig beskrives *Use Case-Points-teknikken* som baserer sig på at kravspecifikationen er beskrevet med Use Cases.

7.6 COCOMO

I dette afsnit beskrives en af de tungere estimeringsteknikker. Hvis du er helt nybegynder på estimeringsområdet, vil jeg derfor anbefale at du simpelthen springer dette afsnit over og gemmer det til en senere lejlighed. At jeg alligevel har valgt at tage det med i denne bog, skyldes dels at teknikken er meget omtalt, dels at nogle af dens ideer rummer megen indsigt i hvad der påvirker et estimat.

Estimeringsteknikken *COCOMO* ("COnstructive COst MOdel") er opfundet af Barry Boehm og er først beskrevet i [Boehm81]. Denne oprindelige model kaldes i dag *COCOMO81*. Den er senere blevet opdateret, og den seneste udgave kaldes *COCOMO II* og er beskrevet i [Boehm00].

COCOMO er ikke særligt udbredt i Danmark hvilket formentlig skyldes at mange anser den for uoverskuelig at gå i gang med – måske parret med en skepsis over for om man nu kan sætte et estimat på formler. De gange hvor jeg har set den anvendt, har den imidlertid kunnet opvise imponerende præcise resultater.

Teknikken går ud på at vurdere tre ting:

- Programmets forventede "størrelse", udtrykt ved det forventede antal kodelinjer eller antal funktionspoint.
- 5 skalafaktorer.
- 17 indsatsfaktorer der hver for sig har indflydelse på det endelige estimat.

Ideen i at adskille disse ting og vurdere dem hver for sig er at det alt andet lige er lettere at vurdere en størrelse, når der ikke skal tages hensyn til sideeffekter fra andre steder.

Herefter indsættes programstørrelse og faktorer i en formel hvorved man får den forventede indsats, målt i personmåneder.

I de følgende afsnit vil vi kigge på:

- programstørrelsen
- skalafaktorerne
- indsatsfaktorerne
- beregningsudtrykkene

7.6.1 Programmets størrelse

Den forventede programstørrelse måles ved antal tusind kodelinjer, kaldet *KSLOC* ("Kilo Source Lines Of Code").

For at alle kan vurdere på samme måde, skal man have defineret præcist hvad der menes med en kodelinje. Forskellige programmørers "stil" skal ikke have indflydelse på estimatet, så f.eks. blanke linjer og kommentarer tæller *ikke* med.

I praksis vil man estimere kodestørrelsen på grundlag af en nedbrydning af kravspecifikationen i mindre dele og altså estimere de enkelte dele hver for sig, for så at summere dem til sidst.

Kritikere har hævdet at det er næsten lige så svært at vurdere størrelsen målt i antal kodelinjer som det er at vurdere tiden direkte. I nogle forsøg har det da også vist sig at selv garvede it-udviklere kan skyde op til en faktor 3 for lavt på antal kodelinjer. Dette kan imidlertid lige så godt skyldes manglende erfaring med at vurdere i *kodelinjer* i stedet for *timer*. For at få denne erfaring kan det være en god ide at måle kodestørrelsen på gamle programmer. Vil man ikke gøre helt så meget ud af det, kan man nøjes med compilerens oplysninger om hvor mange linjer programmet består af.

7.6.2 Omfang og varighed

Følgende forkortelser (Boehms egne) anvendes i nedenstående formler:

PM: Person Month, dvs. resurseforbruget i personmåneder. Kaldes herefter *omfang*.

TDEV: Time DEVelopment schedule, dvs. projektvarighed i kalendermåneder. Kaldes herefter *varighed*.

I formlerne indgår nogle konstanter (A, B, C og D). Disse findes ved at kalibrere modellen ud fra faktiske projektforløb. Konstanterne der indgår i COCOMO II.2000-kalibreringen er:

$A = 2{,}94 \quad B = 0{,}91 \quad C = 3{,}67 \quad D = 0{,}28$

Omfanget fås som:

$$PM = A \times KSLOC^{E}$$

hvor $E = B + 0{,}01 \times \sum Sf_i$

Sf_i (i=1..5) er 5 skalafaktorer som omtales i det følgende.

Benyttes funktionspoint til beregning af størrelse, kan der omregnes til KSLOC (se [Boehm00]).

Varigheden fås som:

$$TDEV = C \times PM^F$$

hvor $F = D + 0{,}2 \times 0{,}01 \times \sum Sf_i = D + 0{,}2 \times (E - B)$

I [Boehm00] påvises i øvrigt at det stort set er umuligt at presse varigheden ned under 75% af TDEV, altså den beregnede nominelle varighed (kaldes "den umulige region"). Alene dette faktum burde være grund nok til at beregne TDEV.

7.6.3 Faktorer

Eksponenten E i ligningerne herover udtrykker om projektet udviser stordriftsfordele. Så hvis $E < 1$, vil en fordobling af projektets størrelse (målt i kodelinjer) ikke betyde en fordobling af omfanget.

Skalafaktorerne er:

- KENDSKAB. Hvor tit har vi lavet sådan noget før?
- FLEKSIBILITET. *Skal* vi tilpasse os stramme krav og eksterne grænseflader?
- AFKLARETHED. Hvor godt er de mulige risici afklarede?
- PROJEKTGRUPPEN. Hvor godt sammentømret er projektgruppen?
- MODENHED. Hvor god en udviklingsproces har vi (med udgangspunkt i CMMI – se side 61)?

Skalafaktorerne omsættes til tal ifølge en omsætningstabel (se [Boehm00]). Værdierne ligger fra 5-8 hvis indflydelsen er meget lav og op til 0 hvis den er ekstra høj. Skalafaktorerne udtrykker hvor meget de aktuelle omstændigheder er værre end idealsituationen.

Ud over programmets funktionalitet – udtrykt i den forventede kodestørrelse – er estimatet afhængigt af en række indsatsfaktorer, såkaldte "cost drivers", der udtrykker forskellige forholds indvirkning på estimatet. Hver af indsatsfaktorernes indflydelse vurderes på en skala: meget lav, lav, nominel, høj, meget høj (og for enkeltes vedkommende endda "ekstra høj"). Hele skalaen anvendes dog ikke nødvendigvis for alle faktorerne. I det følgende gennemgås kort de 17 indsatsfaktorer. Selv om du ikke bruger COCOMO-modellen, indeholder disse 17 indsatsfaktorer megen relevant viden om hvad der egentlig påvirker estimatet, og kan derfor være gode at kende.

Produkt-faktorer

- PÅLIDELIGHED. Hvilken pålidelighed kræves for det pågældende produkt? Er der f.eks. tale om et måleapparat til brug i eget laboratorium hvor en programfejl stort set

ingen betydning har? Eller er der tale om styring af atomkraftværker, medicinsk udstyr eller våbenkontrolsystemer hvor fejl kan betyde tab af menneskeliv?

- DATABASENS STØRRELSE drejer sig om den effekt det har på udviklingen, hvis der kræves en stor mængde testdata.
- PRODUKTETS KOMPLEKSITET. Hvor komplekst er det produkt der skal udvikles? Der laves et vægtet gennemsnit hvor de forskellige moduler vægtes efter deres forventede størrelse.
- UDVIKLING TIL GENBRUG. Faktoren udtaler sig om den ekstra indsats der skal til, når man udvikler noget der skal kunne genbruges. Sådanne komponenter skal ofte være mere generelle. Rangerer lige fra intet krav om genanvendelighed til krav om genbrug inden for flere produktlinjer.
- DOKUMENTATIONENS EGNETHED. Hvor godt egnet er dokumentationen til det den skal bruges til i de forskellige faser? Rangerer fra at der er mange udækkede behov i de forskellige faser, til at den er særdeles udmærket til at dække behovet.

Platformsfaktorer

- EKSEKVERINGSHASTIGHED. Er der særlige krav til eksekveringshastigheden? Er det helt lige meget hvor lang tid programmet er om udførelsen? Eller er der en krævet svartid på 0,5 msek i en maskine der i forvejen er hårdt belastet?
- LAGERBEGRÆNSNINGER. Er der begrænsning på den mængde lagerplads programmet har til rådighed i maskinen? Er det lige meget hvor meget programmet fylder, fordi det er alene i maskinen, eller det bare kan swappe? Er der kun en bestemt – evt. lille – mængde lager til rådighed, samtidig med at swap er udelukket?
- PLATFORMENS USTABILITET. Sker der tit ændringer med den platform (dvs. maskine, styresystem, databasesystem osv.) der udvikles på? Kan der hyppigt forventes nye releases af maskinel eller programmel som udviklingen er afhængig af?

Person-faktorer

- DESIGNERNES EVNER. Hvor gode er analyse- og designfolkene (dvs. dem der arbejder med kravspecifikation, arkitektur og design)? Og hvor gode er de til at samarbejde? Er de dygtige til at skrue systemer sammen? Bemærk at det er hele gruppen der vurderes under ét. Er der indbyrdes forskel, må man prøve at finde et gennemsnit.
- PROGRAMMØRERNES EVNER. Hvor gode er programmørerne? Og hvor gode er de til at samarbejde? Er de dygtige til at få noget kode, der virker, banket ned? Eller plejer det at tage en frygtelig tid, og så er der altid masser af fejl? Bemærk at det er hele gruppen der vurderes under ét.

- UDVIKLERSTYRKENS KONTINUITET. Hvad er omsætningen (dvs. udskiftningen) blandt udviklerne?
- ERFARINGER MED EMNEOMRÅDET. Hvor stor erfaring har projektgruppen inden for det pågældende emneområde? Er det første gang gruppen laver et produkt som dette? Eller har den tidligere lavet lignende produkter? Denne faktor kan være problematisk ved længerevarende projekter, idet der naturligt sker en oplæring i løbet af projektet.
- ERFARINGER MED PLATFORMEN. Har projektgruppen stor erfaring med platformen (compiler undtaget)? Eller er det første gang man bruger den pågældende platform? Også her drejer det sig om hele gruppen.
- ERFARINGER MED SPROG OG VÆRKTØJER. Har projektgruppen erfaring med det pågældende programmeringssprog og de benyttede udviklingsværktøjer? Eller er det første gang de bruges?

Projekt-faktorer

- UDVIKLINGSVÆRKTØJER. Bruges der udviklingsværktøjer ved udviklingen (CASE-værktøjer eller lignende)? Eller har gruppen kun en compiler til rådighed? På dette punkt arbejdes med en yderligere nedbrydning, da der sker meget på værktøjsområdet hele tiden.
- GEOGRAFISK ADSKILT UDVIKLING. Når udviklings-teamet arbejder geografisk adskilt på flere fysiske lokationer, tager det længere tid, end hvis de sidder tæt sammen.
- TIDSKRAV TIL UDVIKLING. Er gruppen under tidspres, dvs. kræves projektet gennemført på meget kort kalendertid? Eller er der den nødvendige tid til rådighed? Bemærk at faktoren også stiger, selv om der er længere tid end den nominelle til rådighed.

Vurderingerne af de enkelte faktorer (meget lav, lav, nominel osv.) omsættes til et tal (i området fra 0,73 til 1,74) ifølge en omsætningstabel. Herefter multipliceres værdierne for alle indsatsfaktorerne nu med hinanden til en fælles indsatsfaktor som ganges på omfanget (PM) for at få det justerede omfang.

7.7 Function Points

En anden parametrisk metode er Function Points-metoden. Den blev opfundet af Allan J. Albrecht (se [Albrecht83]) hos IBM. Metoden går ud på at estimere funktionaliteten ud fra en række karakteristika ved programmet som kan læses i kravspecifikationen.

Metoden minder om det som håndværkere foretager sig, når de skal afgive et overslag over en opgave. En malermester vil f.eks. opmåle antallet af kvadratmeter væg, antallet af kvadratmeter loft, antal meter fodpaneler, antal vinduer osv. Når han så ved hvor lang tid de enkelte operationer plejer at tage, kan han beregne en pris ud fra det. Altså en slags "projekt i metermål".

I et it-projekt tæller man hvor mange input, output, forespørgsler, datafiler og interfaces der er. Derefter ganges med bestemte vægte, og til sidst summeres for at få det samlede antal Function Points.

Fig. 7.6: Eksempel på Function Points

Data	Antal	×	Vægt	=	Function Point
Input	4	×	4	=	16
Output	5	×	5	=	25
Forespørgsel	7	×	4	=	28
Datafil	2	×	10	=	20
Interface	1	×	7	=	7
I alt					**96**

Tælles det samlede antal Function Points som i ovenstående eksempel sammen til 96, og man erfaringsmæssigt ved at man er 8 persontimer om at producere 1 FP, så omfatter systemet altså 8×96 = 768 persontimers arbejde.

Dette er det *ujusterede* antal Function Points. De kan justeres med omgivelsesfaktorer analogt til COCOMOs.

Bemærk at Function Points altså mere er et funktionalitetsmål end en egentlig estimeringsteknik: vi skal vide hvor mange timer vi er om at producere 1 Function Point, før vi kan estimere tidsforbruget. Derfor kan to organisationer som byder på den samme opgave (hvor antal Function Points er kendt) altså godt komme frem til to forskellige priser.

Function Points-metoden er en anerkendt og udbredt metode til estimering (også i

Danmark). Man kan melde sig ind i IFPUG (International Function Points User Group, www.ifpug.org), og der findes internationalt anerkendte tællemanualer. Desuden kan man blive certificeret Function Points-tæller. Det er nemlig ikke helt trivielt at lære at tælle Function Points.

I øvrigt kan man også lave sin egen mini-udgave af Function Points-metoden hvis man synes den er for omfattende. Sådanne metoder kaldes også Action Points (eller Object Points eller Application Points). Her kan man f.eks. tælle skærmbilleder (evt. opdelt i simple, avancerede, meget komplekse), kontroller på skærmbilleder osv. Man kan også tilføje sine egne vægte. Laver man mange ens systemer, vil dette ofte være en udmærket metode.

Du kan læse mere om Function Points-metoden i [Garmus00].

7.8 Use Case-points

Denne teknik baserer sig på at kravspecifikationen er beskrevet som en række Use Cases. Kort fortalt inddeles Use Case-beskrivelsens aktører i tre grupper:

- Simple, f.eks. et andet system med en veldefineret grænseflade (API)
- Middel, f.eks. et andet system som tilgås via en bestemt protokol, eller en person som tilgår systemet via en simpel tekstgrænseflade
- Komplekse, f.eks. en person som tilgår systemet via en grafisk grænseflade

Derefter inddeles de beskrevne Use Cases i tre grupper, afhængigt af hvor mange transaktioner de indeholder:

- Simple (3 eller derunder)
- Middel (4-7)
- Komplekse (8 eller derover)

Nu kan det ujusterede antal Use Case-Points beregnes som:

$$UUCP = AktørFaktor + UseCaseFaktor, \text{ hvor}$$

$$AktørFaktor = 1 \times SimpleAktører + 2 \times MiddelAktører + 3 \times KompleksAktører$$

$$UseCaseFaktor = 5 \times SimpleUseCases + 10 \times MiddelUseCases + 15 \times KompleksUseCases$$

Du kan læse mere om Use Case-Points i [Schneider98].

7.9 Organisering af estimeringen

Estimeringsprocessen kan organiseres på flere måder:

- DELTAGERVURDERING hvor de enkelte projektdeltagere hver især estimerer deres egne aktiviteter. Dette forudsætter at de enkelte projektdeltagere er i besiddelse af tilstrækkelig erfaring og intuition, samt at de har indsigt i projektet. Eventuelt kan de individuelle estimater diskuteres i projektgruppen, eller det kan være projektgruppen der i fællesskab laver estimaterne.
- EKSPERTVURDERING hvor én eller flere "eksperter" (personer med stor erfaring og god intuition) foretager estimeringen som beskrevet ovenfor.

I de fleste tilfælde er ekspertvurdering langt at foretrække. Ved deltagervurdering er der nemlig dels en optimistfaktor involveret, dels ofte en manglende erfaring.

Optimistfaktoren skyldes at man som deltager er tilbøjelig til ikke at være tilstrækkeligt objektiv i sine vurderinger. Nogle har lidt i spøg sagt at også her gælder Paretos 80/20-regel: udviklerne estimerer de lette 80% (som tager 20% af tiden at udvikle), mens de undervurderer de svære 20% (som tager 80% af tiden at udvikle). Og desuden føler man mange gange ikke man kan være "bekendt" at sige at det tager *så* mange timer. Dette har fået nogle til bagefter at multiplicere det færdige estimat med et gæt på denne "optimistfaktor" (oftest bruges det berømte π, dvs. man ganger estimatet med 3). Dette er dog højst uvidenskabeligt! Her kan det være en ide at huske udviklerne (og dig selv) på TTT-reglen: "Ting Tager Tid".

Det andet problem med deltagervurderinger (den manglende erfaring) skyldes at de enkelte projektdeltagere forholdsvist sjældent starter på nye projekter, og derfor lige så sjældent får luftet deres evner som estimatorer. Derfor har de ofte en ringe erfaring i at vurdere korrekt.

Benyttes derimod eksperter, vil disse langt oftere foretage disse vurderinger og derfor være i bedre træning. Da eksperter sjældent er direkte involverede i udviklingen af det pågældende produkt, har de også større mulighed for at være objektive i deres vurderinger.

Ved at benytte eksperter opnås en yderligere fordel, nemlig at erfaringsopsamlingen (se næste afsnit) bliver lettere. Eksperterne har en naturlig interesse i at blive bedre til at vurdere rigtigt, så de er for det første mere motiverede for at erfaringsopsamlingen faktisk bliver foretaget. For det andet er estimaterne foretaget af de samme personer hvorved de bliver lettere sammenlignelige.

Eksperterne kan eventuelt tilføje en faktor, således at de vurderer som om det er dem

selv der skal lave arbejdet, hvorefter udviklerne får tildelt en faktor der fortæller hvor hurtig den pågældende er i forhold til eksperten. Jeg hørte engang om en virksomhed hvor eksperterne skulle vurdere som om det var deres fætter der skulle gennemføre opgaven. Derfor blev faktoren kaldt en *fætterfaktor*. Bruges sådanne faktorer, skal de løbende justeres. Typisk bliver udviklerne hurtigere og hurtigere gennem projektet.

Folk bliver tit overraskede over hvor stor forskel der egentlig kan være på udviklernes produktivitet. Men undersøgelser dokumenterer faktisk produktivitetsforskelle på op til en faktor 10 mellem en dygtig udvikler og en mindre rutineret (se f.eks. [Boehm00]). Derfor skal der tages højde for det ved estimeringen. Af samme grund er det vanskeligt at lave et nøjagtigt estimat før projektet er bemandet. Her kan man kun basere sig på en vis ensartethed blandt udviklerne. Og så i øvrigt de store tals lov som siger at eventuelle unøjagtigheder udjævner hinanden når der er mange nok.

7.10 Erfaringsopsamling og kalibrering

Ved alle de nævnte teknikker er det af afgørende betydning at man har en erfaringsbase fra tidligere projekter at trække på. Man skal derfor ikke alene arkivere de estimater der foretages undervejs, men også de faktiske udviklingstider. Ud over dette talmateriale er det også af betydning at man har andre oplysninger om de tidligere projekter, f.eks. antal, størrelse og kompleksitet af de indgående moduler. Denne information kan f.eks. opsamles i statusrapporter eller evalueringsrapporter.

Det er lettest hvis man har et automatiseret timeregistreringssystem til at holde styr på estimater og faktisk forbrugt tid.

Datamaterialet bruges også til løbende at kalibrere estimeringsmodellen, sådan at den hele tiden passer til firmaets forhold.

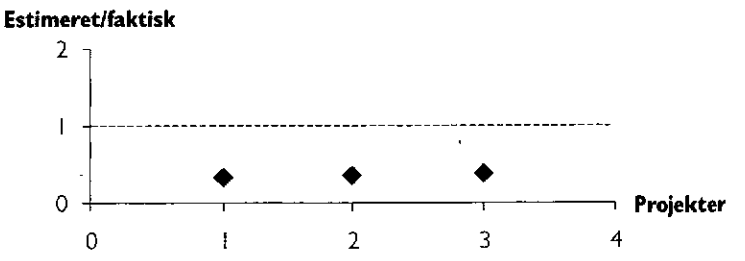
Estimeringsnøjagtighed

Hvor godt rammer du når du estimerer? De fleste vil have svært ved at svare på dette spørgsmål. Men det kan være rigtig godt at vide. Forestil dig f.eks. at du plotter værdierne for:

$$\frac{\textit{Estimeret antal timer}}{\textit{Faktisk forbrugt antal timer}}$$

for et antal projekter ind i et regneark og tegner en graf over det. Det kunne f.eks. se således ud:

Fig. 7.7: Estimeringsnøjagtighed



Y-aksen er altså estimeringsnøjagtigheden mens X-aksen er de forskellige projekter. For hvert af de tre projekter kan vi nu se hvor nøjagtigt det blev estimeret. Værdierne bør ideelt set ligge på 1, svarende til at du har ramt lige i øjet hver gang.

Med ovenstående eksempel vil det være nærliggende at antage at det næste projekt, du skal i gang med, vil have godt af en mindre justering før du offentliggør dit estimat. Men bemærk at du ikke nødvendigvis behøver justere så du kommer helt op på 1. Tag det gerne i mindre bidder ad gangen. F.eks. kan du næste gang gange med 1,5. Faren er nemlig at du kommer til at estimere for højt – og det er også galt, jf. Parkinsons lov som siger at ting tager den tid der er afsat til dem.

Jeg hørte engang om en virksomhed hvor en af de ansatte var fantastisk god til at gætte *forkert*. Hans estimater lignede kurven ovenfor: de lå altid på en ret linje (svarende til at han altså gættede præcis *det samme* forkert hver gang). Det udnyttede projektlederne til at justere estimatet ind.

For god ordens skyld er det dog ikke altid at disse værdier vil ligge på linje. Hvis du laver forskellige *typer* af projekter, kan du komme ud for at estimeret/faktisk varierer noget mere. Men måske er faktoren nogenlunde ens inden for hver projekttype.

I øvrigt vil usikkerheden på estimatet afhænge af hvor tidligt der estimeres. Hvis kulturen i jeres virksomhed er sådan at du altid bliver hængt op på dit første estimat (på trods af evt. forbehold), og du så tvinges til et tidligt estimat, så må du skyde vildt for højt. F.eks. med en faktor 3-10. Hvis kulturen derimod er som den bør være, så kan du give dit teknisk velfunderede estimat og samtidig sige at usikkerheden er så og så stor. Og så vil det blive respekteret. Din overbevisningseffekt bliver i øvrigt større hvis du kan fremvise jeres statistik over hidtidig estimeringsnøjagtighed.

7.11 Hovedpunkter

Kapitlet fremhæver følgende hovedpunkter:

- Estimeringen danner grundlaget for planlægningen og er derfor en aktivitet der skal tages meget alvorligt.
- Adskil omfang og varighed. Omfang er det forventede resurseforbrug målt i persontimer. Varigheden er kalendertiden.
- Politiske estimater er de estimater som udspringer af andre forhold end de rent tekniske (f.eks. "vindende pris"). Politiske estimater foretages ofte af ledelsen og skal tages lige så alvorligt som de tekniske estimater.
- Trepunktsestimering er en praktisk anvendelig teknik som man meget hurtigt kan komme i gang med. Her estimeres det sandsynligste, det korteste og det længste aktiviteten kan tage, og der laves et vægtet gennemsnit. Der nedbrydes de steder hvor usikkerheden er for stor.
- Ekspertgæt (Delfimetoden) er en teknik hvor et antal eksperter sætter sig sammen for at finde frem til et godt estimat. Det er ofte bedre end at udviklerne selv estimerer, idet mange udviklere har en tendens til at underestimere. Til gengæld skal ekspertenes estimat justeres efter hvilken udvikler der skal udføre opgaven.
- Fremskrivningsteknikken kan bruges hvis I arbejder efter en fasemodel og ved hvor lang tid de forskellige faser tager relativt, i forhold til hinanden.
- Analogivurdering kan bruges hvis I har et erfaringsmateriale i form af tidligere projekter som du så kan sammenligne det aktuelle med.
- COCOMO er en parametrisk model som beregner estimatet ud fra formler. Specielt dens 17 indsatsfaktorer rummer megen indsigt i hvad der er afgørende for et estimat.
- Function Points er en anerkendt og udbredt estimeringsteknik som egner sig specielt til administrative systemer. Der findes detaljerede tællemanualer, og man kan blive certificeret til at tælle Function Points.
- Erfaringsopsamling i form af timeregistrering på projektet (og gerne dets enkelte dele) er afgørende for at man kan blive bedre til at estimere.

Gode estimater kræver:

- En grundig forståelse af kravene.
- En dokumentation af hvordan de er fremkommet.
- Erfaringsdata så vi kan sammenligne med hvad vi tidligere har præsteret.
- Metoder til at vurdere størrelse og kompleksitet af det som skal estimeres.