Wat bepaalt de kosten van software?

Wat bepaalt de kosten van software?

Ir. F.J. Heemstra

De ervaring leert dat het schatten van de kosten en de duur van een softwareproject moeilijk is. Terwijl in de bouwwereld een kostenoverschrijding van 10% op het budget als een behoorlijke misser wordt ervaren is dit in de softwarewereld een staaltje van vakmanschap op begrotingsgebied.
In dit artikel wordt aangegeven waarom het geven van betrouwbare kostenschattingen zo moeilijk is. Zo zal o.a. besproken worden:
– het grote aantal factoren dat de softwarekosten beïnvloedt (de cost drivers),
– de moeilijk te kwantificeren relatie tussen softwarekosten en kostenbepalende factoren,
– de stand van zaken met betrekking tot onderzoeken op dit gebied,
– het gebrek aan historische projectgegevens, die als basis dienen voor nieuwe, te begroten, softwareprojecten,
– het manco aan een eenduidige definiëring van begrippen, zoals omvang van software, complexiteit enzovoort.

1 Inleiding

Het is meer dan tien jaar geleden dat Wolverton in een baanbrekend artikel [Wol74] een eerste aanzet gaf voor een methode om de kosten voor het vervaardigen van een softwareprodukt te schatten en Walston en Felix [Wal77] pioneerswerk verrichtten met hun studie over het meten van de produktiviteit bij software-ontwikkeling. Ondanks een groeiende toevloed van publikaties over dit soort onderwerpen, staat het schatten van softwarekosten nog in de 'kinderschoenen'
Het plannen en begroten van softwareprojecten levert veel problemen op. Budgetten worden voortdurend overschreden en tijdsplannen moeten voortdurend worden bijgesteld. Als gevolg hiervan wordt de druk om het softwareproduktieproces te beheersen wat betreft benodigde tijd, kosten, mankracht en hulpmiddelen steeds groter. Zo zijn onder deze druk, de laatste jaren diverse modellen ontwikkeld voor het schatten van softwarekosten. In dit speciaal nummer van het tijdschrift *Informatie* wordt in het artikel van Van Vliet [Vli87] een korte beschrijving gegeven van enkele van deze modellen.

Wat zal een schattingsmodel de projectmanager moeten bieden? Is een uitkomst in termen van aantal mensmaanden voldoende of dient er ook een prognose geleverd te worden voor een aantal andere zaken? Een projectmanager zal, met name bij aanvang van een softwareproject, op diverse vragen een antwoord willen hebben. Vragen zijn ondermeer:
– hoeveel gaat het softwareprodukt kosten,
– hoeveel tijd is er nodig om het produkt te maken,
– hoeveel mensen zijn daarvoor nodig en om welke mensen gaat het (welke ervaring, welke opleiding, enz.),
– welke middelen (apparatuur, programmatuur) zijn er nodig,
– hoe zal de verdeling zijn van geld, mensen, tijd en middelen over de verschillende fasen van het softwareproject,
– welke factoren zijn van invloed op de kosten en doorlooptijd van een softwareproject, en hoe groot is die invloed,
– wat zal de omvang van het produkt worden,
– hoeveel documentatie moet er gemaakt worden,
– moet de software zelf ontwikkeld of aangeschaft worden,
– welke delen van de software (ontwerp, code) is nieuw en welke reeds bestaande delen kunnen hergebruikt worden?
– wat zullen de kosten c.q. inspanningen zijn om de ontwikkelde software te onderhouden.

Deze lijst met vragen is niet uitputtend. Wel geeft deze lijst een goed beeld van de complexiteit van het software schattings- en beheersingsproces. In het verdere verloop van dit artikel zullen we ons voornamelijk concentreren op de vraag: welke factoren zijn in welke mate van invloed op de hoogte van de kosten van de ontwikkeling van software?

In dit artikel zal ik een overzicht geven van de meest dominante kostenbepalende factoren. Per factor zal aangegeven worden tegen welke problemen men aanloopt bij het bepalen van de relatie tussen de betreffende factor en de softwarekosten.

2 Kostenbepalende factoren

Er zijn vele factoren van invloed op de softwarekosten. Er zijn onderzoeken die melding maken van maar liefst 1200 verschillende factoren [Not84]. Andere onderzoeken beperken zich tot een of enkele factoren. Het zal duidelijk zijn dat het schatten van de softwarekosten een bijna ondoenlijke zaak wordt als men bij het schatten met duizenden factoren rekening moet houden. Het zal derhalve noodzakelijk zijn zich te beperken tot de meest dominante kostenbepalende factoren. Men verschilt in de literatuur en in de praktijk overigens van mening over wat nu wel of niet dominante factoren zijn. Het volgende overzicht (tabel 1) van diverse modellen en de factoren die hierin een rol spelen moge dat demonstreren [Boe84].

INFORMATIE JAARGANG 29 EXTRA EDITIE
PAG. 569 T/M 648 1987

FACTOR	SDC. 1965	TRW. 1972	PUTNAM SLIM	DOTY	RCA. PRICES	IBM	BOEING 1977	GRC. 1979	COCOMO	SOFCOST	DSN	JENSEN
SOURCE INSTRUCTIONS			x	x		x	x		x	x	x	x
OBJECT INSTRUCTIONS	x	x		x	x							
NUMBER OF ROUTINES	x				x					x		
NUMBER OF DATA ITEMS						x			x	x		
NUMBER OF OUTPUT FORMATS								x			x	
DOCUMENTATION				x		x				x		x
NUMBER OF PERSONNEL			x			x	x			x		x
TYPE	x	x	x	x	x	x	x			x		
COMPLEXITY		x	x		x	x			x	x	x	x
LANGUAGE	x		x				x	x		x	x	
REUSE			x		x		x	x	x	x	x	x
REQUIRED RELIABILITY			x		x				x	x		x
DISPLAY REQUIREMENTS				x						x		x
TIME CONSTRAINT		x	x	x	x	x	x	x	x	x	x	x
STORAGE CONSTRAINT			x	x	x	x		x	x	x	x	x
HARDWARE CONFIGURATION	x				x							
CONCURRENT HARDWARE DEVELOPMENT	x			x	x	x			x	x	x	x
INTERFACING EQUIPMENT, S/W										x	x	
PERSONNEL CAPABILITY			x		x	x			x	x	x	x
PERSONNEL CONTINUITY						x					x	
HARDWARE EXPERIENCE	x		x	x	x	x		x	x	x	x	x
APPLICATIONS EXPERIENCE		x	x		x	x	x	x	x	x	x	x
LANGUAGE EXPERIENCE			x		x	x		x	x	x	x	x
TOOLS AND TECHNIQUES			x		x	x	x		x	x	x	x
CUSTOMER INTERFACE	x					x				x	x	
REQUIREMENTS DEFINITION	x			x		x				x	x	x
REQUIREMENTS VOLATILITY	x			x	x	x		x	x	x	x	x
SCHEDULE			x		x				x	x	x	x
SECURITY						x				x	x	
COMPUTER ACCESS			x	x		x	x		x	x	x	x
TRAVEL/REHOSTING/MULTI-SITE	x			x	x					x	x	x
SUPPORT SOFTWARE MATURITY									x		x	

Tabel 1: Diverse modellen en nun kostenbepalende factoren

In tabel 1 wordt van diverse modellen aangegeven met welke kostenbepalende factoren rekening wordt gehouden. De modellen staan vermeld in de kolomhoofden, de kostenbepalende factoren staan in de eerste kolom. De kruisjes in de overige kolommen geven per kolom (= model) de bijbehorende cost drivers weer. Een haakje achter een aantal kruisjes betekent dat in het betreffende model de met deze kruisjes corresponderende factoren te zamen als een factor worden beschouwd. Een korte beschrijving van de verschillende modellen wordt gegeven in [Boe81].

Zoals uit tabel 1 blijkt worden in het ene model niet alleen meer, maar ook andere dominante factoren onderscheiden dan in het andere model. De verschillende modellen zijn daarnaast weinig eensluidend over de invloed van de individuele factoren op de hoogte van de softwarekosten. Zo komt het voor dat factor x in model A als de meest dominante geldt, terwijl diezelfde factor in model B als vele male minder dominant wordt gezien. In het verdere verloop van dit artikel zullen hier nog meer voorbeelden van worden gegeven en zullen verklaringen worden gegeven van deze meningsverschillen.

Voor de overzichtelijkheid delen we de kostenbepalende factoren in het navolgende in een aantal categorieën. We onderscheiden factoren met betrekking tot:
a. het te ontwikkelen softwareprodukt zelf (WAT),
b. de middelen waarmee het produkt ontwikkeld wordt (WAARMEE),
c. het personeel dat de software ontwikkelt (DOOR WIE),
d. projecteigenschappen, (HOE),
e. eigenschappen van de organisatie waarvoor de software ontwikkeld wordt (VOOR WIE).

In tabel 2 wordt een overzicht gegeven van mijns inziens de meest dominante kostenbepalende factoren, verdeeld over de hierboven genoemde vijf categorieën. In de volgende secties van dit artikel zullen deze factoren een voor een besproken worden. Het accent zal hierbij vooral gelegd worden op de problemen die er zijn bij het bepalen van de invloed van de verschillende factoren op de hoogte van de softwarekosten.

3 Produktafhankelijke factoren

Omvang van het software produkt
Algemeen is men het erover eens dat de kosten van het te ontwikkelen softwareprodukt zullen toenemen naarmate de omvang van het produkt toeneemt. Deze constatering is onafhankelijk van de maatstaf die men hanteert voor de factor omvang. Men treft in de literatuur verschillende methoden aan om de omvang van een softwareprodukt te bepalen [Cra84]:
a. het aantal coderegels,
b. de 'Halstaed' methode,
c. de waarde van de gebruikersfunctie.

Kostenbepalende factoren m.b.t. het				
Wat (produkt)	*Waarmee* (middelen)	*Door wie* (personeel)	*Hoe* (project)	*Voor wie* (gebruiker)
omvang van de software	beperkingen wat betreft:	kwaliteit	eisen, gesteld aan duur project	participatie
kwaliteit	– executie tijd,	ervaring	project beheersing	aantal gebruikers
omvang van de database	– responsietijd,	verloop personeel		mate waarin specificaties veranderen
complexiteit van de software	– geheugencapaciteit	kwaliteit management		scholing opleiding
documentatie	Tools			
hergebruik	moderne programmeer technieken			

Tabel 2: Overzicht van de kostenbepalende factoren

ad a. Het aantal coderegels
Dit is de meeste bekritiseerde, maar tevens meest verbreide methode. Velen wijzen terecht op het nadeel dat de ene coderegel de andere niet is. Zo kost het maken van commentaarregels of het invoegen van blanco regels minder tijd dan het opstellen van een programmaregel! Een geschatte omvang van 10 000 programmaregels zegt op zichzelf dus weinig over de ontwikkeltijd en -kosten. Dit zal onder meer worden bepaald door de definitie die men hanteert voor het begrip omvang. Dat de meningen hierover verdeeld zijn blijkt uit tabel 3. In deze tabel wordt een beeld gegeven van de verschillende definities die door verschillende onderzoeken voor het begrip omvang worden gebruikt [Thi81].

naam onderzoek	*definiëring omvang software*
aerospace	aantal regels in het programma
Farr en Zagorski	aantal machine instructies
SLIM	aantal statements
Wolverton	aantal machine instructies
Doty	aantal (executeerbare) regels code
PRICE-S	aantal (executeerbare) instructies
Telecote	aantal machine instructies

Tabel 3: Definiëring van de omvang in verschillende onderzoeken

Een gevolg van het ontbreken van een eenduidige definitie voor de omvang van een programma is dat men op grond van deze methode voorzichtig moet zijn om een uitspraak te doen over de produktiviteit van een programmeur. Door met name blanco regels en commentaar regels mee te tellen bij het aantal coderegels bestaat het gevaar dat de programmeur aangemoedigd wordt kunstmatig veel van dergelijk soort regels in zijn programma op te nemen. Op deze wijze wekt hij de illusie van een hoge produktiviteit [Con86]. Bovendien wordt een vergelijking van de produktiviteit van programmeurs moeilijk als er sprake is van verschillende programmeertalen, bijvoorbeeld APL versus FORTRAN of COBOL.

ad b. de 'Halstead' methode
Deze methode telt niet het aantal coderegels maar bepaalt de omvang van een programma [Hal77] op basis van:
– het aantal unieke operatoren (gereserveerde woorden),
– het aantal unieke operanden (denk aan variabelen en constanten) en
– het aantal malen dat deze operatoren en operanden in het programma worden gebruikt.

Ook bij deze telmethode bestaat het eerder genoemde definitie probleem. Er bestaat geen algemeen geaccepteerde opvatting over de beste manier om operatoren en operanden te klassificeren en te tellen. Het telresultaat blijkt bovendien sterk afhankelijk te zijn van de programmeertaal die wordt gebruikt. Een voordeel van deze methode is dat, gegeven een bepaalde programmeertaal, de spreiding in de telresultaten minder is dan bij het tellen van het aantal coderegels.

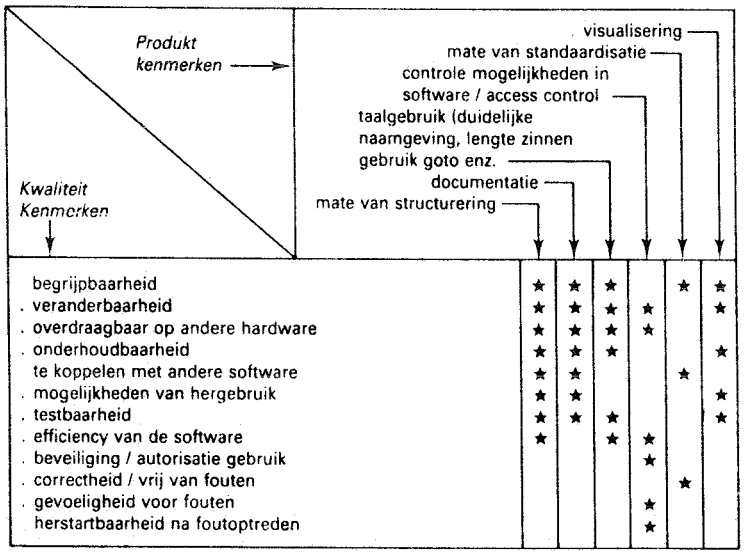
Ad c. de 'waarde' van de gebruikersfunctie
Ook bij deze methode gaat het niet om het aantal coderegels. Als alternatief voor het aantal coderegels als maat voor de omvang wordt de hoeveelheid functies bepaald die een software-produkt vervult, in termen van de data die het programma gebruikt en genereert. Een uitwerking hiervan is Albrecht's functie punt analyse waarvan in de loop der tijd allerlei varianten en verfijningen zijn ontwikkeld [Alb79]. Deze methode om de omvang van een softwareprodukt te schatten wordt veel gebruikt, ondanks de onduidelijkheid die er is hoe men precies die functies (invoer, uitvoer, bestanden en opvragingen) moet bepalen en waarderen. Bovendien schiet de methode tekort bij rekenintensieve toepassingen, waarbij men over het algemeen te maken heeft met weinig I/O, weinig bestanden wor-

INFORMATIE JAARGANG 29 EXTRA EDITIE
PAG. 569 T/M 648 1987

den gebruikt en het aantal opvragingen gering is. Dergelijke programma's vervullen derhalve een gering aantal functies en zouden per definitie een beperkte omvang moeten hebben. Deze methode is daarom alleen toepasbaar bij administratieve toepassingen. [Con86].
Hoewel de onderzoekers het onderling eens zijn over de positieve relatie tussen produktomvang en produktkosten, ongeacht de maat die ze gebruiken om de omvang te bepalen, verschillen zij sterk van mening over de mate waarin de omvang de kosten bepaalt. Over het algemeen neemt men aan dat de kosten meer dan proportioneel toenemen bij een toename van de produktomvang. Deze progressie treedt vooral op als het aantal ontwikkelaars en dus ook het communicatieverkeer in een project toeneemt. Bovendien gaat een toename in omvang veelal gepaard met een toename in complexiteit van het programma. Hoe sterk die progressie is, verschilt van auteur tot auteur. Er is echter ook een beperkt aantal onderzoekers die aantonen dat de kosten degressief toenemen bij een toename van de omvang [Cro79], [Wal77].

Kwaliteit van het softwareprodukt

De kwaliteit van een te ontwikkelen softwareprodukt wordt een steeds belangrijkere kostenfactor. Men zal niet alleen oog moeten hebben voor de functionele eisen waaraan een softwareprodukt moet voldoen, maar ook rekening moeten houden met prestatie-eisen. Zo zal het produkt aan bepaalde kwaliteitseisen moeten voldoen. Het is moeilijk softwarekwaliteit te definiëren en te operationaliseren. Nog moeilijker is het om de invloed ervan op de softwarekosten (kwantitatief) te bepalen.
H. Willmer geeft in haar proefschrift een aanzet om tot een definiëring van softwarekwaliteit te komen [Wil85]. Zij maakt daarbij onderscheid in kwaliteits- en produktkenmerken van software. Voorbeelden van kwaliteitskenmerken zijn aanpasbaarheid en testbaarheid (zie tabel 4). Dit soort kenmerken manifesteert zich bij het gebruik van de software en moeten voorafgaande aan de ontwikkeling vastgelegd worden. Validatie ervan vindt plaats bij gebruik. Verschillen tussen gestelde en gerealiseerde kwaliteit kunnen slechts met aanzienlijke inspanningen gecorrigeerd worden. Deze eigenschappen geven niet aan hoe de ontwikkelaar de vereiste kwaliteit kan bereiken. Daarvoor is het nodig produktkenmerken te onderscheiden. Bij de ontwikkeling van software kan men er gericht naar streven dit soort kenmerken te realiseren. Men kan hierbij denken aan kenmerken als toegankelijkheid van de documentatie en de mate van structurering van de programmatuur (zie tabel 4). Dit soort kenmerken kan direct na de realisering ervan gevalideerd worden; men hoeft hiermee niet te wachten tot bij gebruik. Produktkenmerken beïnvloeden de softwarekwaliteit. Aan de hand van produktkenmerken is het mogelijk richtlijnen te bepalen waarmee de kwaliteit van software doelgericht gerealiseerd kan worden en de evaluatie ervan in een vroeg stadium mogelijk wordt. Daarvoor is het nodig de samenhang te kennen tussen beide soorten kenmerken. Willmer beschrijft in haar onderzoek deze samenhang waarvan de essentie in tabel 4 wordt weergegeven.
De studie van Willmer maakt duidelijk hoe moeilijk het is een antwoord te geven op vragen als: wat is softwarekwaliteit, wanneer is er sprake van goed onderhoudbare programmatuur, hoe kan dat gerealiseerd worden, hoe hangen de kwaliteitskenmerken onderling samen en hoe zijn ze gerelateerd aan de produktkenmerken. Wordt onderhoudbaarheid geëist, dan moet ook testbaarheid worden geëist en dienen er specifieke eisen gesteld te worden aan documentatie, de structurering van de programmatuur enz. Pas als dergelijk soort vragen beantwoord zijn wordt het mogelijk de cruciale vraag 'Wat is de invloed van kwaliteit op de kosten van software' te beantwoorden. Ook gedegen onderzoeken van onder andere Willmer [Wil85] en Boehm [Boe78] op dit gebied komen niet verder dan het geven van een kwalitatieve relatie tussen software-kwaliteit en kosten. Kwantitatieve onderzoeksresultaten zijn slechts sporadisch aanwezig. In het bekende onderzoek van Walston en Felix [Wal77] wordt kwaliteit zelfs niet als kostenbepalende factor onderscheiden.



Tabel 4: De invloed van de produktkenmerken op de kwaliteitkenmerken van software. Een kruisje in tabel geeft een relatie tussen beide kenmerken aan. Een voorbeeld: om te kunnen voldoen aan de eis dat de software hergebruikt kan worden, moet bij de ontwikkeling van de software aandacht worden besteed aan de structurering, de documentatie en de visualisering. Wat die aandacht betekent staat uitvoerig beschreven in [WIL85].

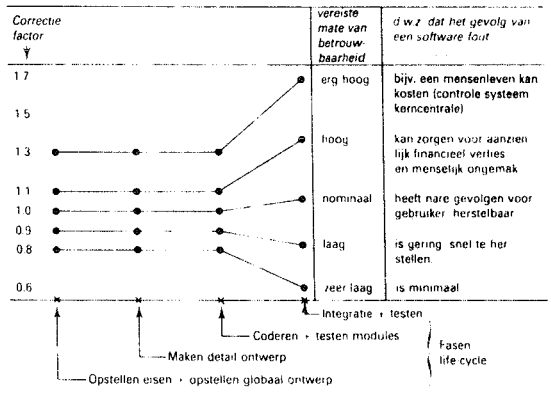
DeMarco [Dem82] definieert kwaliteit als:
'software quality is the absence of spoilage',

waarbij spoilage gedefinieerd wordt als de inspanning om softwarefouten op te sporen en op te heffen. Als kwantitatieve maatstaf definieert hij:

$$\text{Kwaliteit} = \frac{\text{Totale kosten om fouten op te sporen en op te heffe}}{\text{Omvang van het programma}}$$

Een beperkte definiëring, waarbij het accent ligt (uitgaande van het model van Willmer) op kwaliteitskenmerken die pas bij gebruik zichtbaar worden. DeMarco geeft niet aan welke inspanningen (en dus kosten)

er *bij de ontwikkeling* van het softwareprodukt geleverd moeten worden om de 'spoilage' te minimaliseren c.q. op een acceptabel niveau te houden.
Boehm [Boe81] heeft het in zijn COCOMO (COnstructive COst MOdel) niet over de kwaliteit maar over de betrouwbaarheid van software. Betrouwbaarheid wordt door hem gedefinieerd als de kans dat de software op een bevredigende wijze voldoet aan de gestelde eisen gedurende de volgende verwerkingsgang. Boehm geeft toe dat pogingen om deze kans te kwantificeren zijn mislukt en dat men zich voorlopig tevreden moet stellen met kwalitatieve maten voor betrouwbaarheid en met een kwalitatieve relatie tussen geëiste betrouwbaarheid en softwarekosten. Een uitsplitsing in deelfactoren, zoals Willmer dat doet, wordt niet uitgevoerd. De relatie tussen de factor betrouwbaarheid en de kosten volgens het model van Boehm wordt in figuur 1 in beeld gebracht.



Figuur 1: De invloed van de factor betrouwbaarheid op de softwarekosten. De betekenis van de correctiefactor is de volgende: om de invloed van de factor betrouwbaarheid in de totale ontwikkelkosten te betrekken, moeten de nominale kosten (i.c. het aantal mensmaanden) met deze factor worden vermenigvuldigd.

In het PRICE-S model [Fre79] worden naast betrouwbaarheid als kwaliteitsindicator de factoren overdraagbaarheid, mate van structurering, testbaarheid en de wijze van documentatie onderscheiden. Dit model gaat ervan uit dat de invloed van de kwaliteit op de softwarekosten veel groter is dan volgens COCOMO
Het verschil is deels te verklaren door verschillende definities van het begrip kwaliteit en deels doordat de modellen gebaseerd zijn op verschillende historische projectgegevens.

Een evaluatie van de onderzoeken op dit gebied laat zien dat er geen onderzoeksresultaten beschikbaar zijn die een kwantitatieve relatie tussen kwaliteit en softwarekosten leggen. De weinige onderzoeken die een kwalitatieve relatie bepalen, zijn het er unaniem over eens dat de kosten toenemen naarmate er hogere kwaliteitseisen aan de te ontwikkelen software worden gesteld. De meningen lopen uiteen als het gaat om een definitie van kwaliteit of als het gaat om een beschrijving van hoge, gemiddelde en lage kwaliteit.
Ook blijkt men van mening te verschillen over de stijging van de kosten bij een toename van de kwaliteit van bijvoorbeeld gemiddeld naar hoog.

Men dient bij al deze overwegingen ermee rekening te houden dat hogere kwaliteitseisen en dus hogere kosten bij de ontwikkeling van een softwareprodukt gevolgen heeft voor de totale life cycle van het produkt. Hogere eisen ten aanzien van bijvoorbeeld onderhoudbaarheid betekent een extra investering in het eerste gedeelte van de life cycle, maar leiden tot besparingen in de fase onderhoud. Besparingen die de extra kosten wel eens vele malen zouden kunnen overtreffen.

Grootte van de database
De omvang en de complexiteit van de database blijken een duidelijke invloed te hebben op de kosten van het te ontwikkelen softwareprodukt [Boe81]. In veel schattingsmodellen wordt deze factor niet als afzonderlijke cost driver onderscheiden en is deze verweven met de factor omvang van de software.
Dit onderscheid wordt wel gemaakt in onder andere het onderzoek van Walston en Felix [Wal77] en het schattingsmodel van Boehm [Boe81]. Zo kan men in de studie van Walston en Felix de factor 'aantal categorieën items in een database per 1000 coderegels' als een maat voor de grootte van de database beschouwen. Het effect van deze factor is aanzienlijk; zo blijkt de produktiviteit van de ontwikkelaar met een factor 1.73 af te nemen als het aantal categorieën items in de database per 1000 coderegels toeneemt van laag naar hoog. Ook Boehm legt bij de definitie van de grootte van de database een relatie tussen de omvang van de database en de omvang van het programma. In COCOMO gaat het echter om de totale hoeveelheid data ten opzichte van de totale hoeveelheid coderegels. De maximale verandering in produktiviteit blijkt volgens dit model 1.44 te bedragen. Op het eerste gezicht een geringe afwijking met het resultaat van Walston en Felix. In absolute zin komt dat op jaarbasis evenwel neer op een niet gering verschil van circa vier mensmaanden.
In het PRICE-S model wordt de omvang van de database niet als aparte kostenbepalende factor onderscheiden. Ook het SPQR model (Software Productivity, Quality and Reliability model) van C. Jones [Jon86] kent deze factor niet als afzonderlijke cost driver.

Complexiteit van het softwareprodukt
Wat is complexiteit van software? Het blijkt moeilijk te zijn dit begrip te definiëren en de invloed ervan op de ontwikkelkosten van een softwareprodukt te bepalen.
In de literatuur wordt veelal een kwalitatieve beoordeling van complexiteit gegeven door het te ontwikkelen produkt in te schalen in een comlexiteitsklasse. Dit inschalen kan gebeuren op basis van onder andere onderstaande criteria [Her84]:
- de hoeveelheid interacties tussen de verschillende componenten/modules van het softwareprodukt;
- de mate van standaardisatie;
- het soort toepassingsgebied van het te ontwikkelen produkt;
- de structuur van het softwareprodukt;
- de moeilijkheid van de invoer/uitvoer.

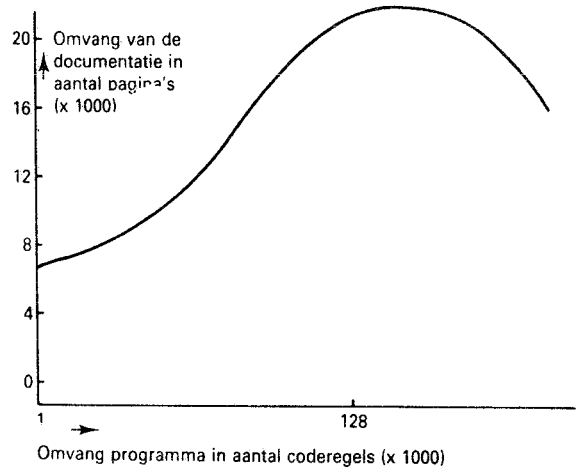
Naam onderzoeker	*criterium voor complexiteit*	*kostenfactor*
Aron [Aro69]	frequentie van interacties	4
Boehm [Boe75]	type applicatie	2.5
Brooks [Bro80]	mate van standaardisatie	9
Chen [Che81]	moeilijkheid invoer/uitvoer	5.65
Walston en Felix [Wal77]	type applicatie	1.7
Wolverton [Wol74]	mate van nieuwheid	5

Tabel 5: De invloed van een toename van de complexiteit op de kosten van software. De kostenfactor geeft aan met welk getal de software-kosten moeten worden vermenigvuldigd als de complexiteit (voor wat betreft het bewuste criterium) toeneemt van laag naar hoog [Her84]

Aan een kwalitatieve beoordeling kleven bezwaren. Een zekere mate van subjectiviteit speelt bij de beoordeling een rol. Dit geldt voor het samenstellen van de verschillende complexiteitsklassen, maar ook voor het onderbrengen van een te vervaardigen produkt in een van deze klassen.
Er is een gebrek aan kwantitatieve beoordelingscriteria. Bovendien blijkt dat de bestaande kwantitatieve maatstaven voor complexiteit pas achteraf, na voltooiing van het produkt, te bepalen zijn.
Men moet er echter rekening mee houden dat de mate van complexiteit van het uiteindelijke softwareprodukt niet overeen hoeft te komen met de prognoses die in de allereerste fasen van de software-ontwikkeling zijn gemaakt. Deze prognoses zijn veelal gebaseerd op de complexiteit van de oorspronkelijke opdrachtformulering of de eisen van de gebruiker. Als de gebruiker evenwel niet goed in staat is die eisen te formuleren en als de ontwikkelaar bovendien niet vertrouwd is met het bewuste type applicatie of zijn ervaring met software-ontwikkeling in zijn algemeenheid gering is, dan kan dat een reden zijn van het verschil in werkelijke en verwachte complexiteit.
Alle onderzoeken over complexiteit zijn eensluidend in de richting van hun resultaten. Hoe hoger de complexiteit hoe hoger de softwarekosten. In de bewuste onderzoeken wordt echter niet altijd dezelfde definitie van complexiteit gebruikt. Een vergelijking van de onderzoeksresultaten is daardoor moeilijk te maken (zie tabel 5).

Hoeveelheid documentatie

Documentatie is een belangrijk deelprodukt van elk softwareproject. Het begroten van documentatiekosten vormt een essentieel onderdeel van het kostenschattingsproces. Een veel gebruikte maat om de hoeveelheid documentatie in uit te drukken is het aantal pagina's. Walston en Felix [Wal77] hebben bepaald wat de relatie is tussen het aantal coderegels en het aantal pagina's documentatie. Tot documentatie werd door hen gerekend de functionele specificaties, de gebruikershandleidingen, het functionele ontwerp, de testresultaten, de programmastroomschema's en delen van de programmalisting (in zoverre deze als onderdeel in handleidingen zijn opgenomen). Uit hun onderzoek blijkt dat er een lineaire relatie bestaat tussen de omvang van de programmatuur (aantal coderegels) en de omvang van de documentatie (aantal pagina's). Haaks op deze bevindingen staan de onderzoeksresultaten van Jones [Jon86]. Hij toont aan dat deze relatie niet lineair is. Voor programma's met een omvang van 1000 tot 128 000 coderegels blijkt de omvang van de documentatie sneller te groeien dan de omvang van de software. Boven de 128 000 regels neemt de hoeveelheid documentatie geleidelijk minder snel toe en neemt op een gegeven moment zelfs af (zie figuur 2).



Figuur 2: De relatie tussen de omvang van de software en de hoeveelheid documentatie

Het is blijkbaar moeilijk een eenduidige kwantitatieve relatie te leggen tussen de omvang van de software en de daarbij behorende hoeveelheid documentatie.
Er is een aantal factoren die van invloed zijn op deze relatie. Zo blijkt dat onder meer de factor 'type applicatie' een belangrijke invloed heeft op de hoeveelheid documentatie per 1000 coderegels. Zo zal de hoeveelheid documentatie bij programmatuur voor militaire toepassingen en voor projecten als bemande ruimtevaartvluchten omvangrijker zijn dan bij programmatuur van dezelfde omvang voor bijvoorbeeld administratieve toepassingen. Aan software van de eerste categorie zullen hogere eisen qua betrouwbaarheid, toegankelijkheid, beveiliging e.d. gesteld worden. Een en ander heeft directe gevolgen voor de omvang van de documentatie.
Het wel of niet gebruik maken van hulpmiddelen die delen van de documentatie automatisch genereren, is ook een factor die bepalend is voor de invloed van de hoeveelheid documentatie op de softwarekosten.
Naast de kosten die verbonden zijn aan het maken van de documentatie dient men ook onderscheid te

maken in kosten tengevolge van gebrekkige documentatie.
Weinig aandacht voor goede documentatie drukt weliswaar de ontwikkelkosten, maar bemoeilijkt het onderhoud en het gebruik van de software. Het zoek- en speurwerk in gebrekkige documentatie, het vergaderen over onduidelijkheden en fouten kan aanzienlijk zijn en bovendien leiden tot tal van irritaties. Korte termijnbesparingen leiden aldus tot aanzienlijke kosten in de toekomst.

Type applicatie
Een van de eerste vragen die men zich dient te stellen bij het schatten van de softwarekosten is: Voor welk toepassingsgebied moet de programmatuur worden ontwikkeld?
De ervaring leert dat de factor type applicatie een belangrijke invloed heeft op de hoogte van de softwarekosten. Bovendien bestaat bij de auteur het (nog niet hard gemaakte) vermoeden dat per type applicatie een beperkt aantal en qua samenstelling steeds verschillende verzameling dominante cost drivers is aan te geven. In de literatuur treft men talloze classificaties van software aan. Zo onderscheidt Stanley [Sta84] de volgende (beperkte) typologie van applicaties:
a. real-time systemen,
b. operating systemen,
c. self contained real-time projects,
d. stand alone non real-time applications,
e. modifications to an existing software system,
f. rewrite of an existing system for a new target machine.

Het is mogelijk aan een softwareproject een moeilijkheidsgraad toe te kennen, afhankelijk van de mate waarin deze onder te brengen is in een van deze type applicaties. Een enigszins afwijkende en m.i. genuanceerdere indeling wordt opgesteld door C. Jones [Jon86]. Hij maakt een indeling in zogenaamde 'programming classes'; een indeling die gebaseerd is op het doel waarvoor het programma wordt ontwikkeld. Hij onderscheidt elf klassen (zie tabel 6).
De eerste conclusie van Jones is, dat het frappant is dat in bijna geen enkel onderzoek een indeling op basis van programma klassen wordt gemaakt, terwijl een dergelijke typologie bij het bepalen van de softwarekosten van veel nut kan zijn. Bij het merendeel van de typologieën ligt het accent op het programmeringsaspect (programmering van batch applicaties, artificial intelligence programmering, grafische programmering, nonprocedurele programmering enz). Bij een dergelijke indeling is het veelal niet mogelijk een programma tot één type te rekenen; dit in tegenstelling tot het gebruik van programma-klassen.
Jones legt bij de beschrijving van de programmaklassen vooral de nadruk op documentatiekosten en (in mindere mate) op de vereiste bekwaamheden van het ontwikkelteam. Zo concludeert hij dat de documentatiekosten sterk toenemen, gaande van klasse 1 naar klasse 11 en dat iedere klasse zijn eigen specifieke kenmerken heeft die in belangrijke mate de software kosten determineren.
In het onderzoek van Walston en Felix wordt gesproken over de complexiteit van de applicatie. Een precieze definitie ontbreekt. Bij een lage complexiteit wordt een gemiddelde produktiviteit van 349 coderegels per maand gemeten, bij een hoge complexiteit nog niet de helft. Boehm [Boe81] heeft de factor type applicatie niet als aparte cost driver in het model COCOMO opgenomen, omdat deze factor een overlap heeft met factoren als complexiteit van de software, omvang van de database enz. en omdat het niet mogelijk bleek een rangorde in moeilijkheid van applicaties aan te brengen.
Concluderend kan men stellen dat de meeste onderzoeken wijzen op de belangrijke invloed die de factor type applicatie op de hoogte van de software kosten heeft. De produktiviteit van het personeel is direct gerelateerd aan de moeilijkheidsgraad van het te ontwikkelen softwareprodukt. Hoe moeilijker het toepassingsgebied, hoe lager de produktiviteit en hoe hoger de personeelskosten (de meest dominante kostenpost).
Verder blijkt elk schattingsmodel zijn eigen typologie van software te hanteren. Van een standaardisatie is geen sprake. Deze verschillen geven tevens een verklaring voor de verschillen tussen de modellen over de (kwalitatieve) invloed van soort toepassing op de softwarekosten.

Hergebruik
Bij het ontwikkelen van een nieuw softwareprodukt zal de volgende vraag centraal moeten staan: welke delen ervan zijn echt nieuw en welke delen niet. Is het mogelijk van reeds eerder ontwikkelde software elementen opnieuw te gebruiken bij de ontwikkeling van een nieuw programma.
C. Jones [Jon84] maakt, als hij het heeft over reusability, onderscheid in:

1. Programma's ontwikkeld voor persoonlijk gebruik.
2. Programma's ontwikkeld voor gebruik *binnen één* organisatie.
3. Programma's ontwikkeld voor *veel* gebruikers binnen *dezelfde* organisatie en vanuit het oogpunt van hergebruik.
4. Programma's ontwikkeld t.b.v. intern gebruik binnen een organisatie, maar op beperkte schaal ook te gebruiken door externen.
5. Programma's ontwikkeld als public domain.
6. Programma's die klanten kunnen leasen maar niet kunnen kopen.
7. Programma's die als geïntegreerd geheel met hardware worden verkocht.
8. Programma's ontwikkeld uit puur commercieel oogpunt (denk hierbij aan software die in computer-shops gekocht kan worden, of per post besteld kan worden).
9. Programma's ontwikkeld op basis van een overeenkomst/contract.
10. Programma's ontwikkeld voor de overheid.
11. Programma's ontwikkeld voor militaire doeleinden.

Tabel 6: een indeling van software in klassen (een typologie)

– *reusable data* (een eerste vereiste voor hergebruik is een standaard data format, zodat verschillende applicaties dezelfde data kunnen gebruiken);
– *reusable architecture* (de verzameling aannames waarop hergebruik is gebaseerd, bijvoorbeeld een standaardopbouw van programma's, standaardnaamgeving e.d.);
– *reusable design* (raamwerk ontwerpen, referenties voor de meest gangbare applicaties);
– *reusable programs* (bijvoorbeeld standaardpakketten),
– *reusable modules* (Macros, subroutines, programma generatoren, vierde generatietalen e.d.).

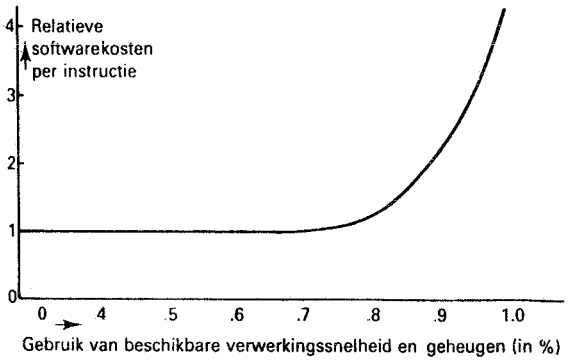
Het zijn niet alleen de directe kostenbesparingen die optreden doordat het eindprodukt sneller gemaakt kan worden. Ook zijn er de besparingen die inherent zijn aan het gebruik van produktcomponenten, die reeds in het verleden getest zijn en door het gebruik hun waarde bewezen hebben. De onderhoudskosten voor dergelijke componenten zullen daardoor laag zijn.
Tegenover deze besparingen staan onder andere de volgende kosten:
– er moet een catalogussysteem opgezet worden waarin alle reeds eerder ontwikkelde produktcomponenten vermeld worden. De software-ontwikkelaar moet gemakkelijk uit die vermelding kunnen afleiden of de betreffende component wel of niet bruikbaar is;
– er moet een database opgezet worden waarin de verschillende softwarecomponenten opgeslagen zijn;
– een deel van de ontwikkeltijd zal nu gaan zitten in het raadplegen van het catalogussysteem en het zoeken naar bruikbare componenten;
– ontwikkelt men software met als nevendoel de mogelijkheid van hergebruik, dan zal bij de ontwikkeling een zwaarder accent komen te liggen op een modulaire aanpak, een zodanige opsplitsing in componenten dat deze voor een eventueel hergebruik bruikbaar zijn;
– niet alleen het selecteren van bruikbare (bestaande) componenten zal tijd vergen. Het onderlinge koppelen, het inpassen in de nieuw ontwikkelde software zal een extra tijd- en dus kostenfactor zijn.

De voordelen die met behulp van hergebruik behaald kunnen worden, zullen de bovengenoemde (nieuwe en noodzakelijke) kosten overtreffen. C. Jones [Jon86] verwacht dat in de toekomst circa 80% van elk nieuw te ontwikkelen softwareprodukt door hergebruik van reeds eerder ontwikkelde componenten tot stand zal komen. Dit heeft enorme kostenbesparingen tot gevolg: een verkorting van de doorlooptijd voor grote projecten van 36 maanden naar 8 maanden, kostenbesparingen van meer dan 50% per project, en bijkomende voordelen als het terugdringen van de application backlog en het (geleidelijk) ontstaan van een verzameling standaardcomponenten. Dit laatste voordeel heeft verstrekkende gevolgen. Zo zal er binnen de verzameling standaardcomponenten slechts *een* module bestaan om de programma functie A uit te voeren. Als in een nieuwe applicatie deze functie A voorkomt, dan zal de software-ontwikkelaar verplicht zijn de betreffende module te gebruiken. Het gevolg hiervan zal zijn dat er niet een wildgroei van allerlei privé modules binnen een organisatie ontstaat. De applicaties worden op deze wijze toegankelijker; ze zijn immers voor een belangrijk deel opgebouwd uit bekende modules. Een en ander heeft weer directe consequenties voor de onderhoudbaarheid van deze applicaties. Dit wordt nog eens versterkt door de hogere eisen die aan de documentatie gesteld worden. Dit alles zal op den duur leiden tot een standaardisatie van software-ontwikkeling.
Terwijl Jones de factor hergebruik als een van de meest dominante kostenbepalende factoren beschouwt en ook het PRICE-S model het belang ervan erkent door de factoren % new design en % new code, is het des te opmerkelijker dat zowel Walston en Felix alsook Boehm deze factor niet noemen. Vermoedelijk dat de nieuwheid van het verschijnsel hergebruik hier een verklaring voor is. Dit vermoeden wordt bevestigd door uitspraken van Boehm in meer recente publikaties [Bos83 en Boe84]. Hierin pleit hij voor een aanpassing van schattingsmodellen met onder meer de factor hergebruik.

4 Middelen-afhankelijke factoren

Tot deze categorie van factoren dienen gerekend te worden:
a. beperkingen voor wat betreft de executietijd, de responsetijd en de geheugencapaciteit;
b. het gebruik van vierde generatietalen, programma- en applicatie generatoren, ontwikkelomgevingen enz. (*Tools*);
c. het gebruik van moderne programmeertechnieken.

ad a. Executietijd, responsetijd en geheugencapaciteit
In vrijwel alle modellen voor het schatten van de softwarekosten worden deze cost drivers onderscheiden. Dat de invloed van dergelijke factoren aanzienlijk kan zijn blijkt uit figuur 3, ontleend aan het PRICE-S model [Sie86].



Figuur 3: De invloed van zgn. hardware beperkingen op de software-kosten. Naarmate men meer tegen de 'grenzen' van de hardware aanloopt, nemen de kosten sterk toe

Over de grootte van de invloed lopen de meningen aanzienlijk uiteen. Dit wordt geïllustreerd aan de hand van tabel 7 [Boe81]. Hierin wordt voor een aantal onderzoeken/modellen, de invloed van beperkingen in executietijd en geheugencapaciteit op de produktiviteit gegeven.

naam onderzoek/ model	*invloed van beperkingen in executietijd op de produktiviteit*	*invloed van geheugenbeperkingen op de produktiviteit*
TRW studie [Wol74]	3.00	–
BOEING [Bla77]	3.33 - 6.67	–
DOTY [Her77]	1.33 - 1.77	1.43
GRC [Car79]	1.60	7.00
GTE [Dal79]	1.25	1.25
Walston + Felix [Wal77]	1.37 - 1.77	2.03
COCOMO (Boehm) [Boe81]	1.66	1.56

Tabel 7: De invloed van beperkingen in executietijd en geheugencapaciteit op de produktiviteit. De getallen in de tabel geven de spreiding van de produktiviteit weer; de verhouding in het aantal benodigde mensmaanden bij geringe en bij sterke beperkingen. De verschillen zijn soms aanzienlijk. Oorzaken hiervan zijn o.a. verschillen in definiëring van geringe en sterke beperkingen en verschillen in historische software-projectgegevens, die als basis voor de betreffende onderzoeken dienden

De responsietijd blijkt een duidelijke kostenbepalende factor te zijn, hoewel er een gebrek is aan cijfermateriaal om deze bewering hard te maken. Bij responsietijd kan men onderscheid maken in responsietijd bij gebruik en responsietijd bij ontwikkeling. Responsietijd in de eerste betekenis behoort tot de categorie produkt afhankelijke cost drivers. In deze sectie richten we ons op responsietijd bij de ontwikkeling van een programma. Snelle responsietijden door met name interactief programmeren leiden tot aanzienlijke verbeteringen van de produktiviteit [Wal77] in de codeer- en testfase.

ad b. Het gebruik van hulpmiddelen (TOOLS)
Deze hulpmiddelen kunnen variëren van eenvoudige debuggers tot zeer uitgebreide en geavanceerde hulpmiddelen voor informatie analyse en ontwerp. Het gebruik van met name vierde generatie talen bij de ontwikkeling van software zal in de toekomst een aanzienlijke kostenbesparing tot gevolg hebben. Als men toekomstprofeten mag geloven dan is een reductie van 90% in programmeertijd en een produktiviteitstijging van 1000% realiseerbaar [Jon86 en Mar84]. De toekomst zal het moeten uitwijzen. De stand van zaken op dit moment leert ons dat:
– het merendeel van dergelijke hulpmiddelen ingezet wordt in de codeerfase van de software-ontwikkeling. Een fase die gemiddeld slechts 20% van de totale ontwikkeltijd in beslag neemt. Een sterke reductie van de codeertijd heeft aldus een beperkte invloed op de totale ontwikkeltijd;
– er nog maar weinig hulpmiddelen beschikbaar zijn, die de ontwikkelaar ondersteunen in de beginfase van het ontwikkelproces; de fasen die over het algemeen de meest tijdrovende zijn;
– er praktisch geen hulpmiddelen zijn die het totale ontwikkeltraject ondersteunen. De bestaande hulpmiddelen richten zich op een of een beperkt aantal fasen [Bov84]:
– bij veel hulpmiddelen het accent ligt op een ondersteuning bij het documenteren. Het aantal geautomatiseerde hulpmiddelen dat een deel of delen van het ontwikkelproces ondersteunen is minder en het aantal dat een deel of delen van het proces van de ontwikkelaar overnemen (de zogenaamde 'intelligente hulpmiddelen') is nog minder;
– er momenteel veel energie gestoken wordt in hulpmiddelen die primair gericht zijn op een inhoudelijke ondersteuning van de software-ontwikkeling – het *hoe* –. De ontwikkeling van hulpmiddelen die de projectmanager ondersteunen bij de uitvoering van zijn taak – het *wat* – blijft hierbij achter. Denk hierbij aan hulpmiddelen voor plannings- en calculatie doeleinden, maar ook kostenschattingsmodellen;
– dat er veel hulpmiddelen beschikbaar zijn, het gebruik ervan tegen valt en het nog maar de vraag is in hoeverre ze daadwerkelijk helpen.

Zetten de ontwikkelingen, die hierboven impliciet zijn genoemd door ('intelligente' hulpmiddelen, integrale ondersteuning, projectmanagement ondersteuning enz.) dan zijn de eerder genoemde besparingen inderdaad te verwachten. Het onderzoek van Mizuno [Miz83] wijst al in die richting. Na de factor personeelskwaliteit leidt deze factor tot de grootste besparingen in softwarekosten.

ad c. Het gebruik van moderne programmeertechnieken
In het onderzoek van Walston en Felix [Wal77] worden vier van deze moderne programmeertechnieken onderscheiden: gestructureerd programmeren, controle van ontwerp en code, top-down ontwikkeling en de inschakeling van een hoofdprogrammeur team. In het COCOMO model [Boe81] wordt hieraan toegevoegd het gebruik van een programma bibliotheek en het gebruik van schematechnieken. COCOMO onderscheidt niet de inschakeling van hoofdprogrammeur teams. Een ander verschil is dat Walston en Felix het effect van de vier technieken op de produktiviteit afzonderlijk bepalen, terwijl COCOMO het gezamenlijk effect ervan aangeeft. Ondanks deze verschillen ontlopen de resultaten elkaar niet veel (zie tabel 8).

naam onderzoeker	*factoren*	*invloed*
Walston en Felix	gestructureerd programmeren	1.78
	controle van ontwerp en code	1.54
	top-down ontwikkeling	1.64
	inschakeling hoofdprogrammeur-team	1.86
Boehm	gebruik moderne programmeertechnieken	1.49

Tabel 8: De invloed van het gebruik van moderne programmeertechnieken op de produktiviteit/softwarekosten. De getallen in de laatste kolom geven de spreiding in produktiviteit/kosten aan. Bijvoorbeeld: Walston en Felix laten zien dat de gemiddelde produktiviteit van 169 coderegels per maand toeneemt tot 301 coderegels bij een toename van weinig naar veel gestructureerd programmeren. De verandering in produktiviteit is: 301 – 169 = 133, de spreiding 301/169 = 1.78

Vraag welke van de volgende technieken gebruikt U?	*niet*	*in overweging*	*wel*	*totaal*	*% wel*
hoofd programmeur team	134	307	224	665	34
doorlopend testen	51	288	307	646	47.5
top down ontwerp	43	329	332	704	47
gestructureerd programmeren	37	351	412	800	51
HIPO	139	278	188	605	31
programma bibliotheek	109	286	237	632	37
interactief programmeren	86	320	280	686	41
Vraag: als u wel gebruik maakt van een of meerdere van bovengenoemde technieken wat is dan het effect hiervan op	*groot*	*gering*	*geen*	*negatief*	*totaal*
projectbeheer	63	294	206	8	571
communicatie met gebruikers	89	227	252	3	571
stabiliteit organisatie	47	193	303	10	553
nauwkeurigheid van ontwerp	166	297	107	3	573
kwaliteit van de code	206	287	94	2	589
vroeg ontdekken fouten	213	276	87	4	580
produktiviteit	165	350	80	6	601
onderhoudskosten	178	272	108	11	569
gedrag/motivatie ontwikkelaar	108	292	160	20	580

Tabel 9: Resultaten van een enquête onder IBM gebruikers naar het gebruik van moderne programmeertechnieken

In een meer recent artikel (Boe83) noemt Boehm als een van de zeven basisprincipes van software engineering het gebruik van moderne programmeertechnieken. Voordelen hiervan zijn meer inzicht in het ontwikkelproces, snellere detectie van fouten, beter te onderhouden programmatuur enz. Uit een enquête gehouden onder IBM gebruikers [Gui79] blijkt eveneens het belang van moderne programmeertechnieken (zie tabel 9). Aan hen werd gevraagd in hoeverre zij gebruik maken van moderne programmeringstechnieken en zoja, wat het effect daarvan is.

Uit de antwoorden blijkt dat gestructureerd programmeren, testen en top-down ontwerpen het meest worden gebruikt (in circa 50% van de gevallen). Als men gebruik maakt van een van de genoemde technieken dan blijken deze het grootste effect te hebben op de kwaliteit van de code en op het in een vroeg stadium ontdekken van fouten. Hoewel dergelijke tabellen een aardige indicatie geven van het gebruik en belang van moderne programmeringstechnieken moet men voorzichtig zijn met het nemen van te absolute conclusies. Zo kan men zich afvragen in hoeverre de ondervraagden in staat zijn het effect van de verschillende technieken te meten. Helpen de technieken werkelijk of denkt men dat ze helpen? Er bestaat in deze immers veel religie!
Evenals James Martin [Mar84] verdedigt Boehm het standpunt om fouten in een zo vroeg mogelijk stadium van het ontwikkelingsproces te ontdekken en te herstellen (to get the errors out early) en hiermee niet te wachten tot de testfase na het coderen. De meeste fouten zijn namelijk al gemaakt voordat aan het coderen wordt begonnen en hoe langer men wacht met het ontdekken en herstellen van een fout, hoe duurder het corrigeren ervan wordt. Boehm beschouwt de techniek van het doorlopend valideren/testen dan ook als een van de zeven basis principes van de software-engineering. Hierbij kan men gebruik maken van onder meer prototyping, simulatie, geautomatiseerde hulpmiddelen, gebruikersdocumentatie van analyse- en ontwerpresultaten enz.

5 Personeelsafhankelijke factoren

Het is moeilijk personeelsafhankelijke factoren als ervaring, vaardigheid e.d. eenduidig te definiëren en de invloed ervan op de software-kosten te bepalen. In tabel 10 is een overzicht gegeven van de personeelsafhankelijke factoren die in een aantal bekende onderzoeken worden onderscheiden. In het verdere verloop van deze sectie wordt aangegeven dat er behalve deze nog meer factoren zijn; factoren die in sommige gevallen een grote invloed kunnen hebben op de softwarekosten.
In het model van Boehm blijkt de factor bekwaamheid (van *alle* factoren) veruit de grootste invloed te hebben op de softwarekosten en de ervaring met de toegepaste programmeertaal de minste invloed te hebben. Bij Walston en Felix daarentegen blijkt de factor ervaring met de toegepaste programmeertaal een van de belangrijkste cost drivers te zijn, terwijl de factor bekwaamheid weliswaar een belangrijke maar zeker niet de meest belangrijke factor blijkt te zijn.
C. Jones geeft geen kwantitatieve relaties tussen zijn personeels afhankelijke factoren en de software kosten en over de inhoud van het model PRICE-S is geen documentatie beschikbaar.
Ondanks de grote invloed van deze factoren op de softwarekosten [Boe81, Wal77, Miz83, Jen84, Jon86] is hier weinig onderzoek naar gedaan. De meeste onderzoeken komen niet veel verder dan een aanzet om factoren als bekwaamheid en ervaring te definiëren en

Walston en Felix	*COCOMO/ Boehm*	*SPQR/ C.Jones*	*PRICE-S/RCA*
algehele ervaring en capaciteit	bekwaamheid analist	ervaring met software ontwikkeling	algehele vaardigheden
ervaring met operationele computer	ervaring met hardware	ervaring met software onderhoud	bekendheid met project
ervaring met programmeertalen	ervaring met program.taal	ervaring met klant	
ervaring met toepassingen van soortgelijke of grotere omvang en complexiteit	ervaring met applicatie bekwaamheid programmeur	persoonlijke (werk)omstandigheden	

Tabel 10: De personeelsafhankelijke factoren in verschillende onderzoeken/modellen

objectief te meten. Bovendien wijzen de onderzoeken op de complexe relaties tussen de personeelsafhankelijke factoren onderling (iets wat ook geldt voor andere cost drivers). Zo zal de bekwaamheid van een analist en programmeur onder meer afhankelijk zijn van zijn opleiding, ervaring, motivatie en vaardigheden om met anderen samen te werken en te communiceren. Een ander voorbeeld van zo'n complexe relatie: de opleiding mag dan nog zo goed zijn en de ervaring nog zo breed, de invloed ervan wordt volledig teniet gedaan als er gewerkt moet worden onder onprettige omstandigheden of als er geen begrip of zelfs tegenwerking van het management is. Rivaliteit, een vijandige sfeer, intern politiek gekonkel zijn dodelijk voor het welslagen van een softwareproject. Jones [Jon86] beweert dat 10 tot 15% van de softwareprojecten juist om die redenen op een mislukking uitlopen.
Couger en Zawacki [Cou78] trachten een antwoord te geven op de hamvraag: Wat is dat, een goede programmeur/analist?
Zij beweren dat persoonskenmerken, zoals het streven naar zelfactualisatie en het willen afleveren van een hoogwaardig intellectueel produkt, veel belangrijker voor de programmeur en analist zijn dan over het algemeen erkend wordt. Salaris blijkt meestal niet een primaire drijfveer. In het onderzoek van Laughery [Lau85] worden deze bevindingen nogmaals bevestigd.

In alle onderzoeken komt ook naar voren dat de factor *ervaring* een belangrijke kostenbepalende factor is. Bij ervaring dient men evenwel onderscheid te maken in:
- ervaring met soortgelijke applicaties;
- ervaring met de betreffende programmeertaal, met programmeren;
- ervaring met soortgelijke machineconfiguraties;
- ervaring met soortgelijke gebruikers.

Grofweg kan gesteld worden: hoe meer ervaring, hoe minder ontwikkelkosten. Een nuancering is hier noodzakelijk. Zo zal de factor ervaring met soortgelijke applicaties het grootste effect hebben aan het begin van het ontwikkeltraject. Geringe ervaring op dit gebied resulteert onder meer in uitgebreider testen, langduriger analyses, minder gebruik van bestaande ontwerpen en code (aspect hergebruik). Onderzoeken noemen een periode van vijf jaar als overgangspunt van weinig naar veel ervaring op dit gebied.
Voor de factor ervaring met programmeren ligt dit punt (althans voor ervaring met de taal COBOL) na ongeveer een jaar. Bovendien blijkt dat na twee tot drie jaar deze ervaringsfactor een verzadigingspunt bereikt. Extra opleiding en nog meer programmeerervaring blijken weinig of geen effect te hebben. Een dergelijk verzadigingspunt wordt ook bereikt bij de factor ervaring met soortgelijke configuraties, operating systems, DBMS, enz. Is die ervaring gering omdat er sprake is van een geheel nieuwe configuratie, dan gaat veel tijd verloren met extra opleiding en (vooral in het begin) veel vertragingen als gevolg van inleerperikelen. De invloed van de factor ervaring met soortgelijke gebruikers is aanzienlijk. Zo zal het verschil uitmaken of men voor de eerste of tiende keer als ontwikkelaar te maken heeft met bijvoorbeeld een onderwijsinstelling. Het leren kennen van de organisatie, het lokaliseren van knelpunten, het communiceren met de gebruiker zal sneller gaan als men vertrouwd is met het betreffende type organisatie.
Kwantitatieve onderzoeksresultaten om deze beweringen hard te maken zijn nauwelijks voorhanden en blijkbaar moeilijk te realiseren, getuige de bevindingen in tabel 11.

Een factor die in de overspannen markt van vraag en aanbod van softwarepersoneel niet verwaarloosd mag worden is het *verloop* van softwarespecialisten tijdens de realisatie van een produkt. In de meeste gevallen zal de projectplanning drastisch moeten worden aangepast als een of meer (moeilijk vervangbare) projectmedewerkers halverwege het project afhaken. Het is lastig bij het bepalen van de softwarekosten hiervan een prognose te maken. In de regel is het werken met een vast toeslagpercentage op de totale softwarekosten hier het meest zinvol.

Ondanks de problemen om de personeelsafhankelijke factoren eenduidig te definiëren, objectief te meten en de invloed ervan op de softwarekosten (kwantitatief) te bepalen, worden in de meeste onderzoeken c.q. schattingsmodellen deze factoren als de meest dominante cost drivers beschouwd. Gezien het belang van dergelijke factoren poneert Boehm [Boe83] de stelling: 'Gebruik beter en minder personeel!'. Dit betekent dat:
- men niet moet trachten door het inzetten van extra personeel opgelopen achterstand in te halen. Meer personeel betekent een meer dan evenredige toename

INFORMATIE JAARGANG 29 EXTRA EDITIE
PAG. 569 T/M 648 1987

Opleiding	*jaren ervaring met* COBOL	*gemiddelde*	
lagere technische opleiding of diploma inleiding programmeren/informatica	1		2.1
	4		7.5
		gemiddeld	3.9
voltooide middelbare schoolopleiding	1		8.7
	2		8.5
	3		6.8
	4 of meer		12.3
		gemiddeld	10.1
voltooide hogere beroepsopleiding	1		8.9
	2		7.1
	3		9.7
	4 of meer		6.3
		gemiddeld	8.1
voltooide universitaire opleiding computerkunde/informatica	1		8.6
	2		14.8
	4 of meer		5.1
		gemiddeld	10.6

Tabel 11: De invloed van opleiding en ervaring op de produktiviteit. Het onderzoek is uitgevoerd door Jeffery en Lawrence [Jef85] op basis van uitgebreid empirisch materiaal. De tabel laat zien dat de genoemde relatie verwarrend is en dat op basis van deze cijfers een toename van opleiding en ervaring niet duidelijk leidt tot een produktiviteitstoename.

van communicatie, vooral ook omdat nieuw personeel verteld moet worden wat er van hen verwacht wordt [Bro75];
- zet in het begin van het project niet te veel personeel in;
- stel voor ontwikkelaars een carrièreplanning op en laat goede prestaties in de salariëring tot uitdrukking komen;
- laat slechte ontwikkelaars niet in een project deelnemen;
- maak gebruik van geautomatiseerde hulpmiddelen waardoor minder personeel ingezet hoeft te worden en dus minder tijd verloren gaat aan communicatie.

6 Projectafhankelijke factoren

Tot deze categorie kostenbepalende factoren worden gerekend:
a. eisen die gesteld worden aan de projectduur,
b. beheersing van het project.

ad a. Eisen die gesteld worden aan de projectduur
Hierbij staat de vraag centraal: wat zijn de gevolgen voor de softwarekosten als een project sneller klaar moet zijn of langer mag duren dan wat reëel haalbaar is. Jones [Jon86] noemt vier redenen waarom vaak te krappe tijdsplanningen worden opgesteld:
1. commerciële overwegingen (price to win). Om een opdracht binnen te halen wordt vaak een oplevertermijn genoemd waarvan iedereen weet dat die veel te optimistisch is;
2. een groot aantal projecten blijkt halverwege de rit toch omvangrijker dan bij aanvang werd gedacht. Specificaties blijken niet juist of volledig te zijn. Contractueel zit de software leverancier echter vast aan een bepaalde leverdatum;
3. toepassing van de wet van Parkinson. Het management is in zo'n geval geneigd een strakke tijdplanning aan te houden om de organisatie zo efficiënt mogelijk te laten functioneren;
4. Schattingsfouten. Als een leveringscontract wordt opgesteld door niet deskundige en onervaren software-ontwikkelaars is de kans op een niet haalbare levertermijn groot.

In de modellen COCOMO en PRICE-S wordt rekening gehouden met de invloed op de software-kosten van een noodzakelijke versnelling (compression) of vertraging (stretch out) van de projectduur. Vooral de invloed van compression blijkt volgens het PRICE-S model aanzienlijk te zijn [Sie86]. Er gaan dan factoren een rol spelen als overwerk, werken in het weekend, werken onder grote tijdsdruk, werken met de hete adem van de projectleider in je nek, stress e.d.
In COCOMO blijkt de invloed hiervan echter gering te zijn, een (maximale) kostentoename van 23% bij een compression van 75% t.o.v. de nominale projectduur. De invloed van stretch out blijkt nog minder te zijn: een (maximale) kostentoename van 10% bij een stretch out van 160% t.o.v. de nominale projectduur. Bij stretch out treden verschijnselen op als het overdreven perfectioneren van de programmatuur. Er zijn ook modellen die geen rekening houden met deze factor, met name het model van Jensen [Jen84] en de studie van Walston en Felix [Wal77].

ad b. Beheersing van het project
Hiertoe moeten gerekend worden alle administratieve activiteiten, die inherent zijn aan software-ontwikkeling, onkostendeclaraties, kosten t.b.v. het gebruik van ruimtes, reiskosten, opleiding, managementkosten enz. In veel gevallen wordt deze onkostenpot over het hoofd gezien. In situaties waarbij de toekomstige ge-

bruiker, de opdrachtgever, een geografisch breed vertakte organisatie is, kunnen met name de reiskosten in de eerste en laatste fasen van de software-ontwikkeling aanzienlijk zijn.

7 Gebruikersafhankelijke factoren

Deze categorie factoren heeft betrekking op de invloed van de gebruikersorganisatie op de softwarekosten.
Men kan hierbij denken aan factoren als:
– in welke mate wordt de gebruiker betrokken bij de ontwikkeling van de software (*gebruikersparticipatie*);
– voor hoeveel verschillende gebruikers/klanten moet de betreffende software worden gemaakt (*aantal*);
– *invoering*. Bij een ruime interpretatie van het begrip software-ontwikkeling, zal men in het schattingsproces rekening moeten houden met kosten die gepaard gaan met opleiding en omscholing, met de installatie van de software, met conversie, met schaduwdraaien, met (in vele gevallen) aanpassen van de bestaande procedures, enz.;
– wat is de *dynamiek* van de betreffende organisatie, m.a.w. hoe stabiel zijn eenmaal opgestelde specificaties.

In het model COCOMO van Boehm wordt met deze factoren geen rekening gehouden. Walston en Felix daarentegen onderscheiden de factoren:
– complexiteit klanteninterface;
– deelname van de klant bij definitie van eisen;
– door klant opgestelde wijzigingen in programmaontwerp en
– ervaring klant met toepassingsgebied.

De eerste twee factoren blijken in dit onderzoek de meest dominante cost drivers te zijn. De gemiddelde produktiviteit blijkt van 491 coderegels per maand te dalen naar 205 als de deelname van de gebruiker bij het opstellen van de eisen toeneemt van weinig naar veel. Bij een hoge complexiteit van de klanteninterface ligt de gemiddelde produktiviteit van de ontwikkelaar op 124 coderegels, bij een lage complexiteit op 500. Een toename met meer dan een factor vier.

De betrokkenheid van de gebruiker, met name in de beginfasen van de software-ontwikkeling is in vele gevallen een voorwaarde voor een effectief produkt. Daarnaast zal deze effectiviteit onder meer afhankelijk zijn van de ervaring van de ontwerper met soortgelijke projecten, de ervaring van de gebruiker met automatisering, het type applicatie enz. Maar al te vaak blijkt bij oplevering dat de programmatuur niet voldoet aan de wensen van de gebruiker [Bem81]. Het onderschatten c.q. verwaarlozen van de fasen informatie-analyse en specificatie, zijn hier meestal de oorzaak van. Hierdoor behaalde tijd- en kostenbesparingen worden volledig tenietgedaan door enorme herstel/onderhoudsactiviteiten, die nodig zijn om de software achteraf met veel kunst en vliegwerk alsnog af te stemmen op de eisen van de gebruiker. Dergelijke hersteloperaties nemen over het algemeen meer tijd in beslag dan alle daaraan vooraf gaande fasen tezamen. Dit is onder meer een verklaring voor vaak gehoorde cijfers als: 70% van de software lifecycle bestaat uit onderhoud en een belangrijk deel hiervan komt voor rekening van de eerder genoemde hersteloperaties [Mar83].

Indien hetzelfde softwareprodukt voor meerdere gebruikers ontwikkeld moet worden, zullen de ontwikkeltijd en -kosten toenemen. Het zoeken naar de 'de grootste gemene deler' in de specificaties, het nagaan of eis x van klant A dezelfde betekenis heeft als eis y van klant B, vergroten de complexiteit van het ontwikkelproces. Zo wordt in de praktijk vaak de vuistregel gehanteerd dat de projectduur met een factor twee toeneemt als het aantal klanten niet een maar twee is. Er is in deze een gebrek aan betrouwbare onderzoeksresultaten.
Als de softwarespecificaties gedurende het ontwikkeltraject vaker wijzigen zal dat een aanzienlijk effect op de softwarekosten hebben. Boehm [Boe81] haalt in dit verband een onderzoek van Schluter [Sch77] aan. Hierin is sprake van een toename in kosten met een factor vier omdat de software, door steeds wijzigende eisen, aangepast moest worden. Vooral naarmate de onzekerheid over de informatiebehoeften groter is, de ontwikkelaar weinig of geen ervaring heeft met het type te ontwikkelen applicatie en de gebruiker ook niet vertrouwd is met automatisering, is dit gevaar reëel aanwezig. Een andere benadering van softwareontwikkeling, te weten prototyping, waarbij de traditionele (lineaire) lifecycle aanpak wordt doorbroken, lijkt in dergelijke gevallen de meest geschikte. Een onderzoek uitgevoerd door Boehm, Gray en Seewaldt toont aan dat met behulp van prototyping ongeveer 40% minder coderegels en 45% minder mankracht ten opzichte van de traditionele benadering nodig is om een softwareprodukt met eer redelijk gelijke performance te realiseren [Bsg84]. Dit 'laboratorium'experiment is nadien bevestigd door ander onderzoek [Jon86].

8 Conclusies en aanbevelingen

In dit artikel is een overzicht gegeven van de belangrijkste factoren die van invloed zijn op de kosten en tijd om software te ontwikkelen. De nadruk is gelegd op de problemen die hierbij optreden. Het is mogelijk een onderscheid te maken in zes soorten problemen.

1. *Het definitieprobleem.*
Er zijn geen eenduidige definities voor begrippen als produktomvang, softwarekwaliteit, ervaring en bekwaamheid van personeel enz.

2. *Het kwantificeringsprobleem.*
Een aantal kostenbepalende factoren is moeilijk te kwantificeren. Men maakt in die gevallen veelal gebruik van kwalitatieve maatstaven als veel, gemiddeld, weinig.

3. *Het objectiviteitsprobleem.*
Voor die factoren waarbij kwantificeringsproblemen optreden, bestaat het gevaar van subjectiviteit. Wat bijvoorbeeld bij de ene ontwikkelaar valt onder de categorie veel, wordt door een andere ontwikkelaar wel-

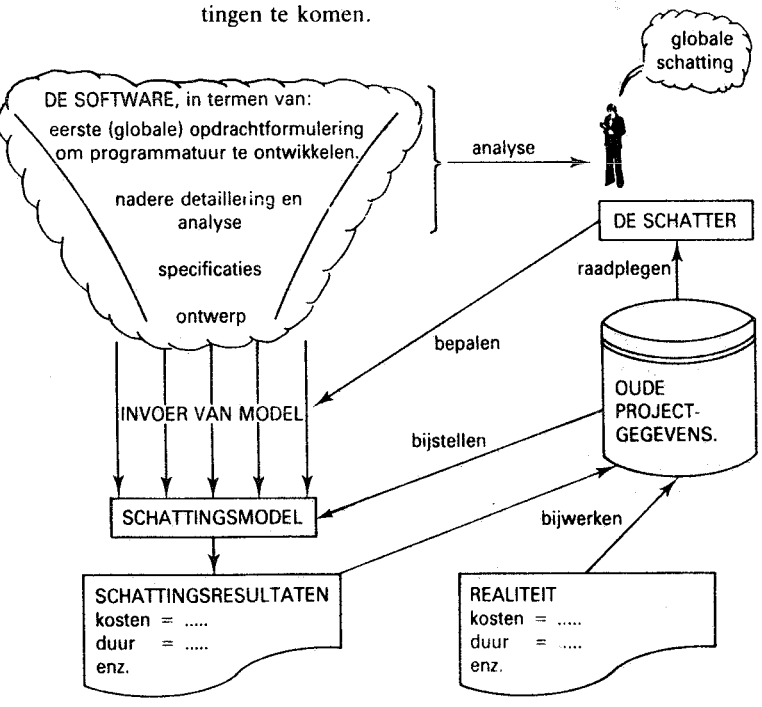
licht gerekend tot de categorie gemiddeld. Factoren die wel gekwantificeerd kunnen worden, kunnen veelal moeilijk objectief worden gemeten.

4. *Het beïnvloedingsprobleem.*
In dit artikel is naar voren gekomen dat het lastig is de invloed van een bepaalde kostenfactor op de software-kosten te bepalen; zelfs voor die factoren die goed gedefinieerd, gemeten en gekwantificeerd kunnen worden.

5. *Het correlatieprobleem.*
Bij de bespreking van de kostenbepalende factoren is aangegeven dat de factoren elkaar wederzijds beïnvloeden. Zo kan een, op het eerste gezicht weinig dominante cost driver, door zijn invloed op andere cost drivers indirect de hoogte van de softwarekosten in belangrijke mate bepalen.

6. *Het schattingsprobleem.*
Bij de ontwikkeling van een nieuw programma zal men vooraf een schatting willen maken van de kosten en duur. Daarvoor is het onder meer nodig schattingen te maken van de waarden van de kostenbepalende factoren. Bijvoorbeeld: hoeveel regels code zal het programma gaan tellen, of hoeveel functiepunten, wat zal de complexiteit zijn enz. Naast de eerder genoemde problemen speelt nu ook de onzekerheid over de waarden van de factoren een belangrijke rol.

Alle schattingsmodellen kampen met deze problemen. Er moet nog veel onderzoek plaatsvinden voordat men in staat is de kosten van het ontwikkelen van software goed te schatten.
In figuur 4 wordt een model van het kostenschattingsproces gegeven. Aan de hand van dit model zal een aantal suggesties worden gedaan om tot betere schattingen te komen.



Figuur 4: Een model van het schattingsproces van softwarekosten

In dit model worden onderscheiden:
a. de software in verschillende stadia van ontwikkeling;
b. het projectteam dat de schatting opstelt en zich daardoor 'verplicht' deze schatting waar te maken (in het figuur 'de schatter');
c. de invoer voor het schattingsmodel;
d. een (parametrisch) schattingsmodel;
e een databank met gegevens over oude projecten;
f. de schattingsresultaten;
g. de werkelijke kosten.

Al in een vroeg stadium van de software-ontwikkeling is het van belang uitspraken te doen over de kosten en doorlooptijd van het project. Over het algemeen heeft men dan nog weinig inzicht in het gewenste produkt. Over zaken als omvang, complexiteit, benodigd personeel e.d. heerst in deze fasen nog onzekerheid. Het heeft dan ook weinig zin om een schattingsmodel zoals bijvoorbeeld COCOMO hier te gebruiken. De onzekerheid over de invoerwaarden (de cost drivers) van een dergelijk model is zo groot dat van betrouwbare schattingsresultaten geen sprake kan zijn. In deze fase is het zinvoller dat de schatter nagaat of er in het verleden software is ontwikkeld die grote overeenkomst vertoont met het gewenste produkt in kwestie. Om dat mogelijk te maken is het noodzakelijk te beschikken over oude projectgegevens. Bovendien moet het mogelijk zijn om op basis van globale specificaties in deze projectgegevens te zoeken naar analogieën. In de praktijk blijkt juist aan dit soort gegevens een gebrek te zijn. Veelal blijft het vastleggen van projectgegevens beperkt tot urenregistratie.
Het schatten van kosten moet meermalen gedurende een project worden uitgevoerd. Naarmate de projectvoortgang vordert zal men steeds meer duidelijkheid krijgen in de gewenste software. In figuur 4 wordt dit in beeld gebracht door de trechtervorm linksboven. Zo zal men na het opstellen van het logisch ontwerp beter in staat zijn te bepalen wat het belang is van de verschillende kostenbepalende factoren dan tijdens de eerste oriëntatie van de opdracht. Behalve de eerder genoemde analogiemethode is het nu zinvol het schatten te ondersteunen door een parametrisch model. De schatter moet de invoerwaarden van het model bepalen door antwoord te geven op vragen als:
- wat is de omvang van de software (in regels, code of functiepunten);
- wat is de complexiteit;
- wat is de vereiste kwaliteit;
- wat voor een type applicatie is het;
- wat is de kwaliteit/ervaring van uw personeel, enz.

Bij het beantwoorden van deze vragen loopt de schatter tegen de problemen aan die in het begin van deze sectie zijn genoemd. Het raadplegen van oude projectgegevens kan bij het bepalen van de invoerwaarden een goed hulpmiddel zijn. Hierbij kan men de volgende aanpak volgen:
1. het onderbrengen van het te ontwikkelen softwareprodukt in een klasse van gelijksoortige produkten. Dit vereist een software klassificatiesysteeem (een typologie);

INFORMATIE JAARGANG 29 EXTRA EDITIE
PAG 569 T/M 648 1987

2. het karakteriseren van het te ontwikkelen produkt (dit is het bepalen van de invoerwaarden van het model) met behulp van de kenmerken van de betreffende klasse en de specifieke (afwijkende) kenmerken van het gewenste produkt.

Zo'n klassificatiesysteem is niet statisch maar zal met behulp van oude (geregistreerde) projectgegevens doorlopend onderhouden moeten worden. Een voordeel van het werken met een dergelijk systeem is dat men per klasse kan aangeven welke bijvoorbeeld de vijf meest dominante kostenfactoren zijn. Deze kunnen per klasse verschillen. Het heeft m.i. weinig zin veel cost drivers in het schattingsproces te betrekken. Hiermee wordt een schijn van nauwkeurigheid gesuggereerd.
Nadat de invoerwaarden van de meest dominante factoren zijn bepaald, kan met behulp van een parametrisch model een schatting van de softwarekosten worden gemaakt. Dergelijke modellen berekenen op basis van de invoerwaarden het benodigd aantal mensmaanden. Vandaar is het dan mogelijk de doorlooptijd en de kosten te bepalen. Het model doet niets anders dan aan geven wat het (kwantitatieve) effect is van de verschillende cost drivers bij gegeven invoerwaarden op het benodigd aantal mensmaanden. Om die relatie te kunnen leggen moet het model gebaseerd zijn op een groot aantal historische projecten. Deze projecten moeten representatief zijn voor de betreffende organisatie. Een en ander houdt in dat het model voor gebruik in een bepaalde omgeving gecalibreerd moet worden. In het artikel van Cuelenaere, Heemstra en Van Genuchten [Cul87] in dit speciale nummer van *Informatie* wordt nader op calibratie ingegaan.
Het berekenen van het aantal benodigde mensmaanden is een mechanische bezigheid die gemakkelijk door een computer kan worden overgenomen. Er is inmiddels een aantal geautomatiseerde versies van onder meer COCOMO, het Jensen model, het model van Putnam enz. commercieel op de markt verkrijgbaar [Set86]. De schattingen en de werkelijk gerealiseerde kosten dienen opgenomen te worden bij de oude projectgegevens. Afwijkingen hiertussen moeten worden gebruikt om het model en of het klassificatiesysteem bij te stellen.

In deze laatste sectie komt duidelijk het belang van een registratie van oude projectgegevens naar voren. Hoe meer historie ten grondslag ligt aan een model en hoe meer deze historie aansluit bij de actuele ontwikkelingsomgeving, des te betrouwbaarder zullen de resultaten van de kostenschattingen zijn.
In de meeste gevallen ontbreekt het binnen organisaties aan gestructureerd cijfermateriaal over reeds ontwikkelde software. Bovendien blijkt het materiaal dat al voorhanden is, vaak verspreid te liggen over verschillende afdelingen, ontwikkelteams e.d. en blijkt de wijze waarop de gegevens zijn verzameld en vastgelegd onderling te verschillen. Dit laatste wordt nog versterkt door de complicatie dat het niet strookt met de geaardheid van de software-ontwikkelaar om nauwkeurig en in detail verslag te leggen van zijn activiteiten. Een veel gehoorde klacht onder programmeurs en analisten is dan ook dat zij hun kostbare tijd veel beter kunnen besteden met het bouwen van een systeem dan met het invullen van de meest uiteenlopende documentatie ten behoeve van projectmanagement [Daw85].
Wil men met enig succes het begroten van software kunnen uitvoeren dan is het beschikken over database met historische produkt- en projectgegevens een voorwaarde.

REFERENTIES

[Alb79] Albrecht, A.J., 'Measuring application development productivity', *The proceedings of the joint SHARE/GUIDE/IBM application development symposium*, okt. 1979.

[Aro69] Aron, J.D., *Estimating resources for large programming systems*, NATO science committee, Rome Italy, okt. 1969.

[Bem81] Bemelmans, Th.M A., de Boer, J.G., 'Ontwikkelingsmethoden 1: het ontwikkelen van informatiesystemen', *Informatie*, jrg. 23, nr. 1 (februari 1981).

[BGS84] Boehm, B.W., Gray, T.E., Seewaldt, T., 'Prototyping versus specifying: a multiproject experiment', *IEEE transactions on software engineering*, vol se-10, no 3, mei 1984.

[Boe75] Boehm, B.W., Holmes, C.E., Katkus, G.R., Romanos, J.P., McHenry, R.C., Gordon, E.K., 'Structured programming: A quantitative assessment', *Computer*, juni 1975.

[Boe78] Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., Mcleod, G.J., Merritt, M.J. *Characteristics of software quality*, Elsevier North-Holland Publishing Company, New-York, 1978.

[Boe81] Boehm, B.W., *Software engineering economics*, Prentice Hall, 1981.

[Boe83] Boehm, B.W., 'Seven principles of software engineering', *The journal of Systems and Software*, no 3, blz. 3-24, 1983.

[Boe84] Boehm, B.W., 'Software engineering economics', *IEEE transactions on software engineering*, vol. se-10, no. 1, jan. 1984.

[Bos83] Boehm, B.W., Standish, T.A., 'Software technology in the 1990's: using an evolutionary paradigm', *IEEE computer*, 1983.

[Bov84] Boer de, J.G., Vonk, R., 'Help .... de informatie-analist verzuipt', *Informatie*, jrg. 26, nr. 12, 1984.

[Bro75] Brooks, F.P., *The mythical man-month*, Addison-Wesley, Reading MA, 1975.

[Bro80] Brooks, W.D., *Software technology payoff: some statistical evidence*, IBM-FSD, Bethesda, MD, april 1980.

[Che81] Chen, E., Zelkowitz, M.V., 'Use of cluster analysis to evaluate software engineering methodologies', *Proceedings fifth international conference on software engineering, IEEE*, maart 1981.

[Con86] Conte, S.D., Dunsmore, H.E., Shen, V.Y., Thebaut, S.M., 'Cost estimation for software development', *Onderzoeksvoorstellen van het center for software engineering research (CSER)*, 1986.

[Con86] Conte, S.D., Dunsmore, H.E., Shen, V.Y., *Software engineering metrics and models*, Benjamin Cummins, 1986.

[Cou78] Couger, J.D., Zawacki, R.A., 'What motivates DP professionals?', *Datamation*, september 1978.

[Cra84] Craenen, G, 'Begroten van automatiseringsprojecten', *Informatie*, jaargang 26 nr. 3 pag. 173 t/m 268, maart 1984.

[Cro79] Crossman, T.D., 'Taking the measure of progammer productivity', *Datamation*, vol. 25, no. 5, blz. 144-147, 1979.

[Cul87] Cueleneare, A., Heemstra, F.J., Genuchten van, M., 'Een expert systeem voor het gebruik van een software begrotingsmodel', *Informatie*, jrg 29, special 1987.

[Daw85] Dawes, B., 'Management techniques to control software development', *Data Processing*, vol 27 no 9, nov. 1985.

[Dem82] DeMarco, T., *Controlling software projects*, Yourdon Press, 1982.

[Fre79] Freiman, F.R., Park, R.E., 'The Price software cost model', *IEEE transactions on software engineering*, 1979

[Gui79] GUIDE, Inc., 'GUIDE survey of new programming technologies', *GUIDE proceedings*, GUIDE, Inc., Chicago Il, 1979.

[Her84] Herrman, O., 'Analyse der Einfluszfaktoren auf die kosten von Softwareentwicklungen', *Angewandte Informatik*, no. 4, 1983

[Jef85] Jeffery, D.R., Lawrence, M.J, 'Managing programming productivity', *The journal of systems and software* 5, 1985.

[Jen84] Jensen, R.W , 'A comparison of the Jensen and COCOMO schedule and cost estimation models', *The proceedings of the ISPA sixth annual conference*, blz 96 t/m 106, mei 1984.

[Jon84] Jones, C., 'Reusability in programming: A survey of the state of the art', *IEEE transactions on software engineering*, vol. SE-10, no 5, september 1984

[Jon86] Jones, C , *Programming productivity*, McGraw-Hill, 1986

[Lau85] Laughery, K R jr , Laughery, K R.sr , 'Human factors in software engineering: a review of the literature', *The journal of systems and software* 5, 1985

[Mar83] Martin, J., McClure, C , *Software maintenance the problems and its solutions*, Prentice-Hall inc, Englewood Cliffs, New Jersey, 1983.

[Mar84] Martin, J , *Information systems manifesto*, Englewood Cliffs, New Jersey, 1984

[Miz83] Mizuno, Y., 'Software quality improvement', *IEEE computer*, 1983.

[Not84] Noth, Th., Kretzschmar, M., *Aufwandschaetzung von D-projekten*, Springer-Verlag, Berlin, 1984.

[Put79] Putnam, L , Fitzsimmons, 'Estimating software costs, parts 1, 2, 3', *Datamation*, sept. to dec. 1979

[Rub85] Rubin, H A., 'The art and science of software estimation: fifth generation estimators', *The proceedings of the ISPA seventh annual conference*, blz 56 t/m 72, mei 1985.

[Set86] Setzer, R., 'Spacecraft software cost estimation: striving for excellence through parametric models (a review)', *The proc of the ISPA 8th annual conference*, 1986

[Sie86] Siereveld, H , *Voordracht op de Technische Universiteit Eindhoven over het schattingsmodel PRICE-S*, september 1986

[Sta84] Stanley, M., 'Software cost estimating', *The proceedings of the ISPA 6th annual conference*, 1984

[Thi81] Thibodeau, R , *An evaluation of software cost estimating models*, RADC-TR-81-144, 1981

[Vli87] van Vliet, J.C., 'Wat kost dat nou, zo'n programma?', *Informatie*, jrg. 29, special 1987

[Wil85] Willmer, H., *'Systematische software-qualitätsicherung anhand von qualitäts- und produktmodellen*, Springer-Verlag, informatik-fachberichte, 1985.

[Wal77] Walston, C.E., Felix, C.P., 'A method of programming measurement and estimating', *IBM Systems Journal* 16, 1977.

[Wol74] Wolverton, R.W., 'The cost of developing large-scale software', *IEEE transactions on computers*, vol. c-23, no 6, june 1974.

Ir F J Heemstra is als universitair docent verbonden aan de Technische Universiteit Eindhoven, faculteit Bedrijfskunde, vakgroep Besturlijke Informatiesystemen en Automatisering (BISA).