Hoe duur is programmatuur? : begroten en beheersen van software-ontwikkeling

Hoe duur is programmatuur?

Begroten en beheersen van software-ontwikkeling

Fred J. Heemstra

informatiemanagement

Hoe duur is programmatuur?

Hoe duur is programmatuur?

begroten en beheersen van software-ontwikkeling

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven
op gezag van de rector magnificus
Prof. ir. M. Tels
voor een commissie aangewezen door het college van dekanen
in het openbaar te verdedigen op
vrijdag 23 juni 1989 om 16.00 uur

door
FEDDERIK JELLE HEEMSTRA
geboren te Heerlen

Dit proefschrift is goedgekeurd door de promotoren

Prof.dr. T.M.A. Bemelmans,
Prof.dr. J.C. van Vliet

voor Marianne en Pien

ISBN 90 267 1371 1

INHOUD

HOOFDSTUK 3. BEGROTINGSMETHODEN

HOOFDSTUK 4. BEGROTINGSMODELLEN

HOOFDSTUK 5. KOSTENBEPALENDE FACTOREN.

HOOFDSTUK 6. BEGROTEN EN BEHEERSEN.

HOOFDSTUK 7. EEN TYPOLOGIE VAN HET BEHEERSEN VAN SOFTWARE-ONTWIKKELING.

VOORWOORD

De aanleiding tot het onderzoek naar het beheersen begroten van software-ontwikkeling vormden de alarmerende berichten over forse overschrijdingen van budgetten en levertijden bij het ontwikkelen van software. Een empirisch onderzoek, uitgevoerd bij een groot aantal organisaties in Nederland, bevestigde deze berichten. In deze studie zijn de bestaande theorieën over en methoden voor het begroten van software-ontwikkeling in kaart gebracht, zijn de belangrijkste begrotingsmodellen beschreven en geëvalueerd en is een uiteenzetting gegeven van factoren die een belangrijke invloed hebben op ontwikkelkosten, doorlooptijd en inspanning. Verder zijn er richtingen aangegeven, die kunnen leiden tot een betere beheersing c.q. begroting van software-ontwikkeling. Zo is een typologie van beheerssituaties ontwikkeld. De essentie van deze typologie is dat de wijze waarop software-ontwikkeling beheerst en begroot dient te worden situatie-afhankelijk is. Verder is in een raamwerk aangegeven welke activiteiten moeten worden uitgevoerd bij het beheersen en begroten van software-ontwikkeling, in welke volgorde dit moet gebeuren en welke invoer en uitvoer bij de verschillende activiteiten behoren. In deze studie wordt een warm pleidooi gehouden voor het registreren van voltooide projectgegevens. Dergelijke gegevens kunnen een belangrijk hulpmiddel vormen bij het beheersen en begroten.

Fred J. Heemstra

1. BEGROTEN EN BEHEERSEN VAN SOFTWARE-ONTWIKKELING : PROBLEEMSTELLING

1.1. INLEIDING

Bij de opdracht tot het bouwen van een huis, het opnieuw inrichten van een badkamer of de aanleg van een nieuwe tuin, verwacht men een nauwkeurige opgave van de hiermee gepaard gaande tijd, inspanning en kosten. Een architect kan op grond van onder andere het aantal gewenste kubieke meter inhoud van een huis, de gewenste inrichting, het aantal vierkante meters ramen en soortgelijke informatie, een eerste begroting opstellen. Deze wordt vervolgens in een verdere dialoog, voordat de eerste steen gelegd wordt, preciezer uitgewerkt (van Vliet 1987). Een aannemer zal bij het bouwen van een huis uitgaan van tekeningen en op basis hiervan benodigde materialen en gereedschap reserveren. Voordat met het werk wordt begonnen, zullen opdrachtgever en aannemer door middel van contracten de gemaakte afspraken vastleggen. De aannemer weet in welke volgorde de verschillende bouwactiviteiten uitgevoerd moeten worden; het metselen van de muren kan niet voorafgaan aan het storten van de fundering. Tijdens de bouw heeft de aannemer voortdurend contact met de opdrachtgever en/of architect. Bij eventuele tussentijdse wijzigingsverzoeken van de opdrachtgever zal hij kosten, inspanning en tijd hiervan calculeren en zonodig de bouwtekeningen aanpassen.

Bij het begroten en beheersen van de bouwkosten van een huis, kan men een projectmatige aanpak verwachten. Doorgaans zijn bij aanvang van de bouw de specificaties redelijk bekend. Wijzigingen hierop tijdens de bouw probeert men te vermijden en zijn alleen tegen vooraf gespecificeerde meerkosten aan te brengen. De aannemer kan aangeven welke activiteiten in welke volgorde uitgevoerd moeten worden en welke middelen daarvoor nodig zijn. Met andere woorden, er bestaat grote duidelijkheid over het te realiseren produkt en het uit te voeren proces.

Begroten en beheersen van kosten, tijd en inspanning ligt anders als het gaat om het realiseren van produkten, waarvan vooraf de specificaties niet precies kunnen worden aangegeven. Daarbij komt het meer dan eens voor dat de specificaties tijdens de realisatie wijzigingen. Het betreft hier doorgaans projecten met een groot technisch, financieel en organisatorisch risico. Bekende voorbeelden van dergelijke projecten zijn de aanleg van de Deltawerken, de bouw van onderzeeboten, vliegtuigen en tal van andere (produkt-)innovatieprojecten. Terwijl bij de bouw van een huis een kostenoverschrijding van 10% op het budget als fors wordt ervaren, zijn overschrijdingen bij genoemde risicovolle projecten met een factor 2 tot 3 geen uitzondering.

In dit onderzoek richt ik mij op het begroten en beheersen van software-ontwikkeling. Wie bij de opdracht voor het ontwikkelen van software eenzelfde rationele en projectmatige aanpak als bij de bouw van een huis verwacht, wordt in het algemeen teleurgesteld. In veel gevallen blijkt het ontwikkelen van software overeen te komen met de eerder genoemde risicovolle projecten. Veel software-ontwikkeltrajecten starten zonder dat duidelijk is vast te leggen wat het uiteindelijke resultaat moet zijn. Meer dan eens gebeurt het dat software wordt ontwikkeld, terwijl er niet vooraf een nauwkeurige analyse van de problematiek heeft plaatsgevonden. Vaak ziet men dat wijzigingen in het systeem worden aangebracht. Deze wijzigingen worden eveneens veelal niet vastgelegd in specificaties en ontwerp. Dit heeft tot gevolg dat het eindprodukt en de specificaties steeds verder uit elkaar komen te liggen.

Het is vanuit deze achtergrond gezien, niet verwonderlijk dat het management problemen heeft met het beheersen van de kosten, inspanning en doorlooptijd van software-ontwikkeling. Het komt in de praktijk voor dat na realisatie van de software de kosten driemaal zo hoog blijken als oorspronkelijk geraamd en de doorlooptijd tweemaal zoveel als gepland. Budgetten worden overschreden en plannen moeten voortdurend worden bijgesteld. Het begroten van software-ontwikkeling is een nogal onontgonnen terrein, waar maar al te vaak "met de natte vinger" wordt gewerkt. Figuur 1.1. geeft hiervan een voorbeeld. Yourdon (1987) onderstreept het voorgaande door erop te wijzen dat slechts circa 10% van de projecten wordt voltooid zonder kostenoverschrijding. Ongeveer 25% van de opgestarte software-ontwikkeltrajecten wordt zelfs nooit beëindigd. Tot soortgelijke uitspraken komt Jones (1986). Hij stelt dat:

- 25% van grootschalige software-ontwikkeltrajecten voortijdig wordt gestopt,

- 60% een significante overschrijding van de begrote kosten heeft,
- en software-ontwikkeling gemiddeld een tijdvertraging kent van een jaar en een overschrijding van de begrote kosten met een factor twee.

...ten nu eenmaal vooral in Latijns Amerika.

Automatisering van studiefinanciering drie maal zo duur

Van onze verslaggever

DEN HAAG — De kosten van de automatisering van de studiefinanciering zijn ruim drie keer zo hoog als was verwacht. Tot 1991 is minister Deetman van Onderwijs een kwart miljard extra kwijt aan automatiseringskosten. In plaats van dertig miljoen gulden, zal Deetman volgend jaar ruim negentig

Figuur 1.1 : Een voorbeeld van een uit de hand gelopen project (Volkskrant, 24 juni 1987).

Het is eigenlijk alarmerend dat organisaties niet of nauwelijks in staat blijken te zijn de ontwikkeling van software in de hand te houden.
In dit eerste hoofdstuk zullen we het voorgaande toelichten en verder onderbouwen. In paragraaf 1.2 zal de groeiende betekenis van software en de noodzaak van een juiste begroting worden aangegeven. In paragraaf 1.3 zal vervolgens een aantal redenen worden opgesomd voor het falen van software-ontwikkeling en wel in het bijzonder wat betreft het overschrijden van kosten en doorlooptijd. In paragraaf 1.4 wordt de probleemstelling omschreven die als basis voor het hier beschreven onderzoek heeft gediend. Bovendien wordt een aantal begrippen nader gedefinieerd en worden uitgangspunten van dit onderzoek geformuleerd. In paragraaf 1.5 tenslotte wordt een verantwoording gegeven van de gevolgde werkwijze.

1.2. HET BELANG VAN BETROUWBARE BEGROTINGEN

Overschrijdingen van budgetten en doorlooptijden in de orde van grootte zoals genoemd in de vorige paragraaf, zorgen binnen organisaties voor veel frustraties en problemen. Voorbeeld: een opdrachtgever heeft op basis van het overeengekomen tijdstip van levering plannen opgesteld voor personele reorganisaties, in gebruikname van nieuwe machines, afstoten van verouderde apparatuur, integratie van nieuwe procedures in het bestaande systeem enz. Het is voor zo'n opdrachtgever frusterend en soms zelfs rampzalig voor de continuïteit van het bedrijf om geconfronteerd te worden met grote overschrijdingen op automatiseringsgebied.

Op tijd leveren van software kan bijzonder kritiek zijn. Voorbeeld: als de programmatuur voor de logistieke beheersing van een nieuwe productielijn te laat geleverd wordt, heeft dit verstrekkende gevolgen. Elke dag niet kunnen produceren betekent immers een direct verlies. Miscalculaties resulteren soms in financiële rampen voor leveranciers en opdrachtgever, die beiden veel geld hebben uitgegeven voor de ontwikkeling van software die bij oplevering niet voldoet aan de gewekte verwachtingen.
De behoefte aan een oplossing voor de problemen bij het begroten van software-ontwikkeling zal in de toekomst alleen maar groter worden. Het toepassingsgebied van software wordt immers steeds groter. Software zal een steeds belangrijkere plaats in onze samenleving gaan innemen. Het navolgende geeft een algemeen overzicht van toepassingsgebieden van software.

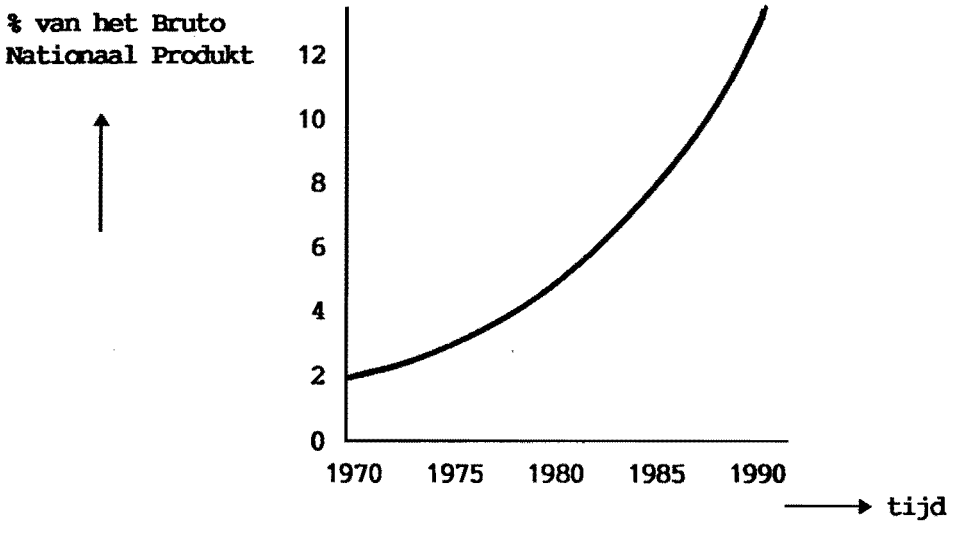
1. *Bestuurlijke informatiesystemen en kantoorautomatisering.*
   In eerste instantie zijn automaten ingezet voor het verwerken van grote hoeveelheden gegevens ten behoeve van administratieve toepassingen. In de jaren zeventig zijn daaraan toegevoegd allerlei beslissingsondersteunende systemen (bestuurlijke toepassingen). De laatste jaren geven ontwikkelingen te zien richting geïntegreerde kantoorautomatisering, waarin secretariële, administratieve, professionele en bestuurlijke toepassingen worden geïntegreerd. Kennis- en expertsystemen kan men in deze beschouwen als pogingen om met name meer inhoud te geven aan professionele toepassingen op gebieden zoals plannen, ontwerpen en diagnostiseren.
2. *Fabrieksautomatisering.*
   Sinds de jaren zeventig is automatisering meer en meer toegepast in fysieke fabricage-, opslag-, en transportprocessen. Ideaal voor menigeen is de "onbemande fabriek", waarin fabricage, opslag en

transport geheel door robots wordt afgehandeld.

3. *Produkt/dienst automatisering.*
   De laatste jaren wordt software meer en meer toegepast in fysieke eindprodukten en voor de produktie van allerlei diensten aan afnemers. Voorbeelden van produktautomatisering, ook wel aangeduid met de term "embedded software", zijn toepassingen voor de besturing van allerlei apparaten (compact disc speler, video apparatuur, huishoudelijke apparatuur). Voorbeelden van dienstenautomatisering zijn publieke databanken en electronische berichtenuitwisseling en -afhandeling (Electronic Data Interchange, afgekort EDI).

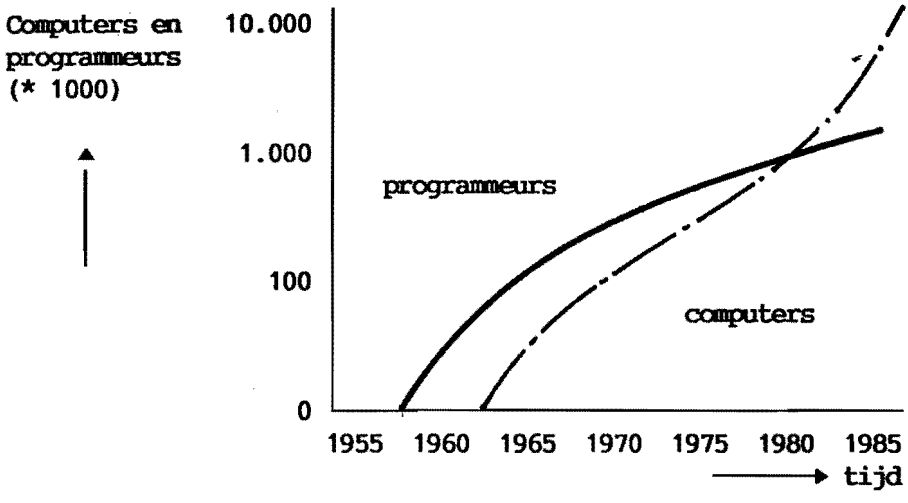
Hoewel de kenmerken van de verschillende categorieën software verschillen, zijn de problemen bij het begroten identiek. Dit onderzoek richt zich dan ook op software in het algemeen. De meeste voorbeelden die aangehaald worden, hebben evenwel betrekking op software ten behoeve van bestuurlijke informatiesystemen, omdat mijn referentiekader vooral in deze categorie ligt.

Het overzicht van de verschillende categorieën software laat zien dat er een breed toepassingsgebied voor software is. Dit zal naar verwachting alleen nog maar toenemen. Software wordt steeds belangrijker als industrieel produkt. Zoals uit figuur 1.2 blijkt is het



Figuur 1.2 : Het aandeel van de software-industrie in het Bruto Nationaal Produkt van de Verenigde Staten.

aandeel van de software-industrie in het Bruto Nationaal Produkt van de Verenigde Staten van 2% in 1970 toegenomen tot circa 10% in 1987. In 1990 zal dat naar verwachting 13% zijn (Musa 1985).De uitgaven aan software nemen jaarlijks toe met een groeipercentage van circa 12%. Volgens globale ramingen (Boehm 1987) waren deze uitgaven over de hele wereld genomen in 1980 90 miljard dollar, in 1985 140 miljard en zullen ze rond de eeuwwisseling 800 miljard dollar zijn.
De stijgende betekenis van software blijkt eveneens uit figuur 1.3, waarin de groei van de software-industrie in de USA in termen van het aantal geïnstalleerde computersystemen en aantal programmeurs in beeld is gebracht (Phister 1985). Een soortgelijke tendens valt waar te nemen in Europa.



Figuur 1.3 : Het aantal geïnstalleerde computers en het aantal programmeurs in de Verenigde Staten.

De explosief groeiende markt voor software is ongetwijfeld ook het gevolg van het feit dat de prijs-prestatie verhouding van de hardware elk jaar beter wordt. Het aantal toepassingen waarvoor het werken met een computer een geschikte en economische oplossing is, neemt hierdoor enorm toe. Studies (Boehm 1981) hebben aangetoond dat men bij de uitvoering van het dagelijks werk gemiddeld reeds voor circa 40% aangewezen is op hardware en software.

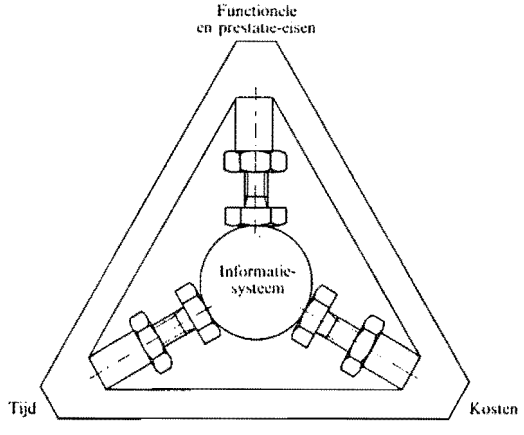
Uit het voorgaande kan men afleiden dat automatisering steeds meer invloed zal krijgen op ieders persoonlijke leven: opslag van persoonlijke gegevens, het nationale betalingscircuit, belastingdienst, luchtverkeersleiding, verzekeringen enz. Software zal bovendien een steeds belangrijker onderdeel gaan vormen van industriële produkten en consumentenartikelen zoals auto's, compact-disc spelers, medische apparatuur enz. Het is momenteel al niet meer uitzonderlijk dat meer dan 40% van de totale ontwikkelkosten van een nieuw produkt voor rekening komt van (embedded) software in zo'n produkt.Deze groeiende betekenis van software vormt een enorme uitdaging voor software-ontwikkelaars. Zij zullen software moeten ontwikkelen die aan steeds hogere kwaliteitseisen moet voldoen. Hierin schuilt een belangrijk probleem. Behalve dat op dit moment de ontwikkelkosten veelal aanzienlijk hoger zijn dan geraamd, laat de kwaliteit van de geleverde software in vele gevallen te wensen over. Niet alleen gebeuren er ongelukken en ongelukjes, de ontwikkelde software voldoet meer dan eens niet aan de wensen van de gebruiker. Van Vliet (1987a) geeft hiervan een aantal sprekende voorbeelden.

- In de software die gebruikt werd voor de besturing van de eerste raket naar Venus stond de volgende programmaregel:
  DO 100 I=1.3
  Er had echter moeten staan
  DO 100 I=1,3
  In plaats van deze herhalingsopdracht stond er nu een regel in de software die opgevat werd als een toekenningsopdracht. Het gevolg was dat de raket nooit op Venus is aangekomen.
- Bij de ontwikkeling van een groot luchtverdedigingssysteem in de Verenigde Staten werd tijdens een van de laatste tests groot alarm geslagen. Een van de computers detecteerde een onbekend projektiel. Het bleek de maan te zijn. Met die mogelijkheid was geen rekening gehouden.

De oorzaak van kwaliteitsgebreken in beide voorbeelden is van totaal verschillende aard. In het eerste voorbeeld is er sprake van een codeerfout. In plaats van een komma is er een punt gecodeerd. In het tweede voorbeeld zijn er fouten gemaakt tijdens de probleemanalyse. De definitie van eisen is onjuist of onvolledig geweest, wellicht ten gevolge van communicatieproblemen tussen klant en analist.

Software-ontwikkelaars worden niet alleen gedwongen programmatuur te ontwikkelen van een steeds betere kwaliteit, maar moeten deze ook economischer te produceren. Er zijn veel manieren waarop organisaties kunnen reageren op deze druk tot efficiency-verbetering. Mintzberg (1983) noemt in dit kader verschillende vormen van stan-

daardisatie en coördinatie. Galbraith (1973) wijst ondermeer op organisatorische maatregelen als het overgaan op een matrixorganisatie of een projectorganisatie. In hoofdstuk 6 wordt nader ingegaan op dit soort maatregelen. In dit onderzoek ligt het accent op het beheersen van software-ontwikkeling als mogelijkheid tot efficiency-verbetering. Bij de ontwikkeling van software zal meer aandacht geschonken moet worden aan activiteiten als begroten, plannen, bewaken van de voortgang en bijsturen van budgetten, plannen en middelen. Zaken als geld, tijd en kwaliteit moeten beheerst worden. Hierbij is het een voortdurend balanceren tussen de kwaliteit (wat kan de software en hoe goed kan de software dat) enerzijds en de kosten en inspanning om dat te realiseren anderzijds. Het spanningsveld waarin de te ontwikkelen software zich bevindt, kan uitgebeeld worden zoals in figuur 1.4 (Bemelmans, van der Pool en Zwaneveld 1984, pag 524). In hoofdstuk 5 wordt nader op deze problematiek ingegaan.



Figuur 1.4 : Het spanningsveld van kwaliteit, tijd en kosten bij de ontwikkeling van software.

Een verhoging van de productiviteit is noodzakelijk om een antwoord te geven op de groeiende vraag naar software. De kloof tussen vraag en aanbod van software is immers groot. De vraag naar nieuwe toepassingen is groter dan wat met de beschikbare menskracht gerealiseerd kan worden. Tussen 1965 en 1985 is de productiviteit jaarlijks met slechts gemiddeld 3% toegenomen, wat neerkomt op een verhoging met een factor 1,8 in twintig jaar. Het aantal programmeurs is in

diezelfde periode met een factor 10 toegenomen.
Een en ander betekent dat de totale output van software met ongeveer een factor 18 is toegenomen. Dit cijfer staat in schril contrast met de toename van de vraag naar nieuwe applicaties met een factor 100 in diezelfde periode. Bij gelijkblijvende ontwikkeling zal deze "software backlog" alleen maar groter worden.Ondanks deze verontrustende ontwikkelingen krijgt het onderzoek op het gebied softwareproductiviteit en begroten van software niet die aandacht die het verdient en krijgt het een plaatsje op de achterbank als het gaat om het verdelen van tijd en geld voor de ontwikkeling van nieuwe methoden, tools en technieken.

In deze paragraaf is aangegeven dat software een steeds grotere betekenis krijgt in onze samenleving, dat er steeds meer, steeds complexere en kwalitatief hoogwaardigere programmatuur ontwikkeld zal gaan worden. De financiële offers die hiervoor nodig zijn, zullen groot zijn. Beter beheersen en begroten van software-ontwikkeling zal om die redenen meer dan noodzakelijk worden. We zijn evenwel nog ver verwijderd van de ideale situatie, getuige de vele voorbeelden van uit de hand gelopen projecten. In de volgende paragraaf wordt aangegeven waarom het afgeven van betrouwbare begrotingen en beheersen moeilijk zijn.

1.3. PROBLEMEN BIJ HET BEGROTEN VAN SOFTWARE-ONTWIKKELING

Zoals in de voorgaande paragraaf aangegeven, is het in de praktijk geen uitzondering dat na voltooiing van software-ontwikkeling de werkelijke kosten en doorlooptijd aanzienlijk hoger blijken te zijn dan oorspronkelijk begroot. Voor dit probleem kan een aantal oorzaken worden genoemd. Hieronder volgt een lijst van oorzaken die in de literatuur de belangrijkste worden genoemd. Deze lijst is niet uitputtend. Wel heb ik getracht een rangorde aan te brengen in die zin dat de belangrijkste oorzaken het eerst genoemd worden (Heemstra 1986):

1. een gebrek aan ervaringsgegevens over reeds voltooide projecten,
2. de wijzigingen in de specificaties,
3. de specifieke kenmerken van software en software-ontwikkeling,
4. het grote aantal factoren dat tijdens de ontwikkeling van invloed is op kosten en doorlooptijd,

5. het inschatten van de invloed van nieuwe ontwikkelingen op het gebied van de software-ontwikkeling, projectbeheersing, informatie-technologie enz.,
6. een gebrekkige definitie van begrippen als software, software-ontwikkeling, informatiesysteemontwikkeling, automatiserings-project en dergelijke,
7. een systematische onderschatting van benodigde inspanning en doorlooptijd en daarmee van de kosten van software-ontwikkeling,
8. de foutieve veronderstelling dat er een lineair verband is tussen tijd en capaciteit.

In de rest van deze paragraaf worden de bovengenoemde oorzaken kort toegelicht.

ad. 1. *een gebrek aan ervaringsgegevens.*

Een voorspelling van kosten en doorlooptijd van een nieuw software-project dient mede gebaseerd te zijn op kennis en ervaring die is opgedaan met oude projecten. Over het algemeen vindt er echter geen systematische registratie plaats van deze kennis en ervaring en worden slechts vanuit financieel oogpunt gegevens vastgelegd. Dit soort gegevens heeft ondermeer betrekking op de totale kosten aan programmatuur, analyse en ontwerp, reis- en verblijfkosten, apparatuur-kosten e.d. en dienen primair voor het doorbelasten van kosten aan gebruikers c.q. opdrachtgevers. Voor het opstellen van een begroting zijn alleen financiële gegevens echter niet toereikend. Zo is het bij het begroten belangrijk dat men inzicht heeft in de invloed op de ontwikkelkosten van zaken als ervaring en kwaliteit van het ontwikkelpersoneel, complexiteit en omvang van de programmatuur, hoeveelheid documentatie, enz. De invloed van dergelijke factoren op kosten, tijd en inspanning blijkt immers, zoals in hoofdstuk 5 zal worden gesignaleerd, aanzienlijk te zijn. Bij het begroten van een nieuw project zal men bovendien veel steun ondervinden van gegevens van oude projecten die betrekking hebben op begrote en gerealiseerde kosten, tijd en inspanning en op oorzaken van eventuele afwijkingen tussen begroting en realisaties. Het is daarom van belang dat men naast financiële gegevens ook zaken registreert zoals hierboven aangegeven. In de literatuur wordt door een groot aantal auteurs (Boehm 1981, DeMarco 1982, Silverman 1985, Noth 1987, Cheatham 1984, Heemstra 1987) het verzamelen van gegevens over lopende en voltooide projecten als noodzakelijk gezien. Deze gegevens kunnen bij vergelijkbare projecten worden benut bij het opstellen van een begroting.

In de hoofdstuk 8 wordt aangegeven welk soort gegevens van voltooide projecten moeten worden geregistreerd en wordt uitvoerig ingegaan op de wijze waarop een dergelijke kennisbank van voltooide projecten een waardevolle ondersteuning kan zijn bij het begroten en beheersen van software.

ad 2. *de dynamiek van de specificaties.*

Een kenmerk van software-ontwikkeling is in vele gevallen een onduidelijke en onvolledige beschrijving van het projectresultaat, zeker in de beginfase. Een gevolg hiervan is dat tijdens de ontwikkeling de specificaties voortdurend worden bijgesteld en aangescherpt. De ontwikkelaar wordt daardoor met het probleem geconfronteerd dat delen van het ontwerp en soms zelfs de code aangepast moeten worden. Een en ander heeft wederom konsekwenties voor kosten en ontwikkeltijd. Het is moeilijk en wellicht onmogelijk alle eisen bij aanvang van een project duidelijk in kaart te brengen (Van Vliet 1987a). Toch zijn het juist de specificaties die het fundament vormen van de te ontwikkelen software. Een onderzoek naar het slagen en falen van software-ontwikkeling, uitgevoerd binnen een software-ontwikkelafdeling van Philips (Haarhuis 1988), toont aan dat gebrekkige specificaties bij aanvang van het project een belangrijk knelpunt zijn. De bij het onderzoek betrokken ontwikkelaars zijn unaniem in hun uitspraak. Allen noemen deze factor als of de meeste dominante of tenminste als één van vijf meest dominante oorzaken voor het overschrijden van kosten en doorlooptijd en voor het realiseren van niet-effectieve software. In tabel 1.1 wordt dit onderzoeksresultaat in beeld gebracht.

De gevolgen van gebrekkige specificaties op ontwikkeltijd en -inspanning kunnen rampzalig zijn. Fouten en onvolkomenheden worden pas in een latere fase ontdekt. Wil de software aansluiten bij de verwachtingen van de toekomstige gebruikers dan zijn hersteloperaties noodzakelijk; specificaties moeten worden aangepast, ontwerpen moeten worden bijgesteld enz. Uit onderzoek blijkt (Martin 1984) dat de hiermee gepaard gaande kosten progressief oplopen naarmate het ontwikkeltraject verder is voortgeschreden. Eveneens indicatief is een onderzoek, uitgevoerd door van der Stelt (1987), waaruit blijkt dat de herstelkosten van een specificatiefout in de codeerfase 23 maal hoger te zijn dan bij herstel in de specificatiefase. In de testfase kan dit zelfs oplopen tot een factor 82.

Vonk (1987) beweert op basis van zijn onderzoeksresultaten dat in een gemiddeld geval tussen de 35% en de 50% van de totale life-cycle

kosten (ontwikkel- en onderhoudskosten tezamen) van software veroorzaakt worden door de lage kwaliteit van specificaties. Hij baseert zich hierbij op een onderzoek van Strausz (1983) waarin beschreven staat dat 60 tot 80% van de onderhoudskosten een direct gevolg is van vage specificaties.

Tabel 1.1 : Gebrekkige specificaties als belangrijkste oorzaak voor het overschrijden van kosten en tijdstip van levering van software-ontwikkeling. In de eerste kolom staan de oorzaken vermeld, die door de betreffende ontwikkelaars zijn genoemd. In de tweede kolom staat aangegeven welk percentage van de ontwikkelaars deze oorzaak heeft genoemd. Opmerking: een ontwikkelaar kon (maximaal) vijf oorzaken noemen.

OORZAKEN	%
gebrekkige specificaties	100
ongeformaliseerde werkwijze	47
onvoldoende beschikbaarheid hardware	27
wijzigingen in hardware	27
communicatieproblemen met opdrachtgever	27
communicatieproblemen binnen ontwikkelteam	20
personeelswijzigingen binnen ontwikkelteam	20
fouten in analyse en/of ontwerp	20
dagelijkse verstoringen (telefoon, vergaderingen)	20
gebrek aan bouwstenen	20
besluiteloosheid	13
te lange projectduur	7
gebrek aan gezag	7
te zeer in details treden.	7

In hoofdstuk 6 wordt nader ingegaan op het belang van een goede specificaties, dat wil zeggen op een juiste beschrijving van het resultaat van de software-ontwikkeling.

ad 3. *kenmerken van software en software-ontwikkeling.*

Een belangrijke oorzaak voor de problemen bij het beheersen en begroten van software is inherent aan het produkt dat gerealiseerd moet worden en het proces dat hiervoor nodig is.
Er zijn een aantal belangrijk verschillen te noemen tussen het ontwikkelen van software en de ontwikkeling van fysieke produkten. Zo gaat het bij software gaat om de ontwikkeling van een abstract produkt. Bij fysieke produkten gaat het naast werkvoorbereiding om het inkopen van grondstoffen en onderdelen, het vervaardigen van halffabrikaten en het assembleren van het eindprodukt. Grondstoffen, onderdelen, halffabrikaten en eindprodukt zijn tastbaar en zichtbaar, evenals het fabricageproces. Dit geldt in mindere mate voor software. Van grondstof is geen sprake; onderdelen en halffabrikaten zijn vergelijkbaar met softwaremodules; het eindprodukt, de code, is niet tastbaar. Bij het realiseren van software ligt het accent de ontwikkeling, fabricage is ondergeschikt. Het inrichten van een produktiestraat en het vervaardigen van duurzame produktiemiddelen spelen bij de ontwikkeling van software weinig of geen rol. In tegenstelling tot tal van fysieke produkten komt een groot gedeelte van de ontwikkeltijd van software voor rekening van het proces van informatiebehoeftebepaling en het specificeren en ontwerpen. Ook bij het testen en onderhouden van software treden duidelijke verschillen op met fysieke produkten. Het aantal toestanden in software is vaak dusdanig groot dan het testen van alle toestanden onbegonnen werk is. Zo komt het voor dat een programma, na uitvoerig testen en een aantal jaren probleemloos functioneren, alsnog fouten oplevert. De software komt in een, niet geteste en nog nooit eerder bereikte toestand, waarin toevallig een fout optreedt. Bij het herstellen van fouten heeft software, in tegenstelling tot fysieke produkten de vervelende eigenschap, dat een verbetering van de ene fout de introductie van andere fouten tot gevolg kan hebben. Bovendien kan men bij software minder eenvoudig een "versleten" onderdeel vervangen door een nieuw. Een belangrijk verschil ten opzichte van fysieke produkten is ook de mogelijkheid van hergebruik van programma's, modules, subroutines. Eenmaal ontwikkeld kan software zonder extra ontwikkeltijd in andere programma's worden hergebruikt.

Een belangrijk kenmerk van software-ontwikkeling is de problematiek om de specificaties vooraf volledig in kaart te brengen. Het gaat er bij de ontwikkeling van software om de lijst van eisen en wensen tijdens het ontwikkeltraject duidelijk en volledig te maken. Stap voor stap groeit het inzicht in het gewenste resultaat en is men in staat meer realistische begrotingen te maken. Een ander kenmerk dat hier

nauw bij aansluit is de instabiliteit van specificaties. Als de werkelijkheid, waarvoor de software is gemaakt, verandert, zal de software aangepast moeten worden. Brengt een organisatie veranderingen aan in haar salariëringssysteem dan zal als gevolg daarvan de betreffende programmatuur aangepast moeten worden. Dergelijke veranderingen en de hiermee gepaard gaande kosten zijn vooraf moeilijk of zelfs niet in te schatten.
De problemen die men met behulp van automatisering tracht op te lossen zijn doorgaans complex. De programma's die hiervoor geschreven moeten worden zijn dus groot in omvang. Bijvoorbeeld: de programmatuur voor de Apollo in 1972 telde 10 miljoen regels code, voor de Space Shuttle in 1982 was dit aantal 40 miljoen regels en voor de Space Station in 1992 zal dit aantal naar verwachting 80 miljoen zijn. Dergelijk grote programma's worden door teams van soms honderden ontwikkelaars gemaakt. Dit stelt grote eisen aan de wijze van organiseren, toekennen van verantwoordelijkheden en bevoegdheden, documenteren enz. Verder staan software-projecten als gevolg van de eerder genoemde backlog onder grote tijdsdruk.

Deze kenmerken en vooral de combinatie ervan maken het geven van betrouwbare schattingen van kosten en tijdstip van levering moeilijk.

ad 4. *beïnvloedende factoren.*

Het aantal factoren dat van invloed is op kosten, inspanning en doorlooptijd blijkt groot te zijn. Denk hierbij bijvoorbeeld aan factoren als kwaliteit, complexiteit en omvang van het eindprodukt, en aan factoren als betrokkenheid van gebruikers, dynamiek van de specificaties, ervaring personeel enz. In hoofdstuk 5 zal uitvoerig worden ingegaan op dergelijke factoren.

Voor het merendeel van deze factoren bestaan geen eenduidige definities. Zelfs voor een op het eerste gezicht eenvoudig te definiëren begrip als software-omvang bestaat geen duidelijke omschrijving. In het ene onderzoek wordt omvang omschreven als het aantal regels programmacode inclusief commentaarregels, in een ander onderzoek is dit exclusief deze commentaarregels (Thibodeau 1981). In weer andere gevallen wordt omvang gedefinieerd als de hoeveelheid uit te voeren funkties in termen van aantal raadplegingen, bestanden, uitvoerlijsten enz.

Het is over het algemeen moeilijk kostenbepalende factoren als kwaliteit, betrokkenheid van het management en complexiteit te operationaliseren. In vele gevallen zal men terug moeten vallen op maten op een ordinale schaal zoals veel, gemiddeld en weinig. Wat door de ene

ontwikkelaar ervaren wordt als een grote betrokkenheid van het projectmanagement, kan door een ander gerekend worden tot de categorie gemiddeld. Zelfs die faktoren die wel eenvoudig geoperationaliseerd kunnen worden, kunnen over het algemeen moeilijk objectief gemeten worden. Zo is het goed mogelijk iemands ervaring met een programmeertaal uit te drukken in het aantal jaren dat door de betreffende persoon geprogrammeerd wordt in die taal. Maar wanneer heeft iemand nu veel ervaring met COBOL, na hoeveel jaren, na het schrijven van hoeveel programma's van een zekere moeilijkheidsgraad ?
Verder levert het problemen op om de invloed van een bepaalde factor op de inspanning en doorlooptijd te bepalen. Dit geldt zelfs voor factoren die goed gedefinieerd en geoperationaliseerd kunnen worden. Daarbij komt dat de relatie tussen de waarden van de factoren en de hoogte van de kosten bepaald wordt door de omgeving waarin de software wordt ontwikkeld. Algemeen geldende uitspraken in deze zijn daarom niet te geven. Dit wordt ondermeer bevestigd als men de drie basisformules voor de relatie tussen software-omvang en inspanning uit drie bekende onderzoeken met elkaar vergelijkt. In tabel 1.2 zijn deze drie formules opgenomen (Vliet 1987).

Tabel 1.2 : Drie basisformules voor de relatie tussen software-omvang en inspanning.

Oorsprong	Basisformule	Mensmaanden bij KLOC = 1000
Halstead	$E = 0.7*KLOC^{1,50}$	22.136
Walston en Felix	$E = 5.2*KLOC^{0,91}$	3.390
Boehm	$E = 2.4*KLOC^{1,05}$	2.793

De factor omvang wordt in deze formules uitgedrukt in KLOC (Kilo Lines Of Code), wat staat voor het aantal regels programmacode gedeeld door 1000. Voor de benodigde inspanning (E = Effort) wordt het aantal mensmaanden genomen. De basisformule van Walston en Felix (1977) laat zien dat de kosten per regel code afnemen naarmate de omvang toeneemt, terwijl uit de formules van Boehm (1981) en Halstead (1977) het tegenovergestelde blijkt.

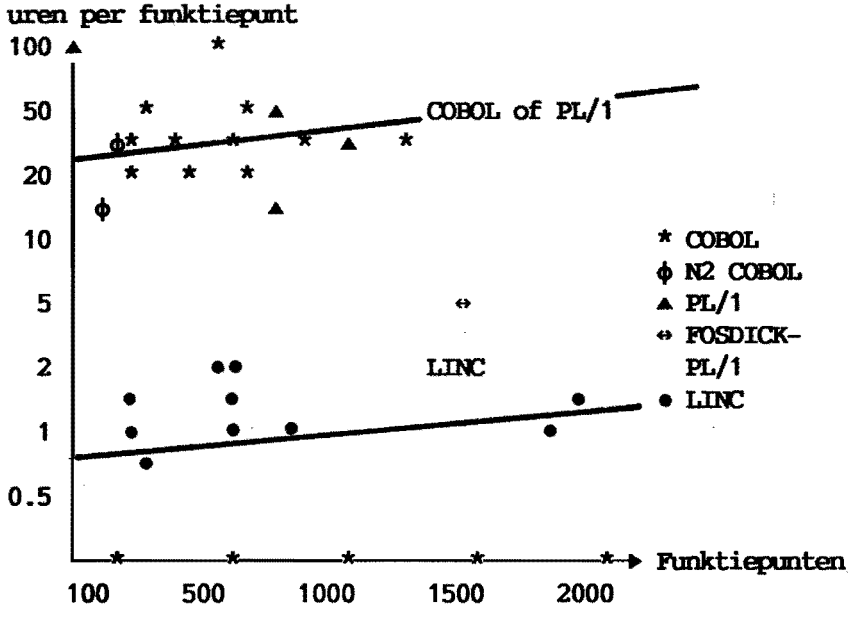
Het is moeilijk de invloed van één individuele factor op de kosten te bepalen. Naarmate er hogere eisen worden gesteld aan de software zal de complexiteit ervan veelal toenemen alsmede de omvang, de hoeveelheid documentatie enz. Bovendien zal er hoger gekwalificeerd personeel ingezet moeten worden en zal er een intensiever contact met de opdrachtgever nodig zijn.
Bij de start van een ontwikkelproject zal de opdrachtgever ondermeer duidelijkheid willen hebben over de prijs en het tijdstip van levering. De projectmanager zal moeten inschatten hoeveel tijd, geld, personeel en middelen in de opeenvolgende fasen van ontwikkeling nodig zijn. Daarvoor is het ondermeer noodzakelijk schattingen te maken van de waarden van de kostenbepalende factoren. Bijvoorbeeld: hoeveel regels code zal het programma gaan bevatten, wat zal de hoeveelheid documentatie zijn en wat de mate van gebruikersparticipatie. Behalve de eerder genoemde problemen speelt nu ook de onzekerheid over deze waarden een belangrijke rol. In (Heemstra 1987a) wordt uitvoerig op deze problematiek ingegaan.

ad 5. *nieuwe ontwikkelingen.*

De snel ontwikkelende technologie en de veranderingen in de methodologie van software ontwikkeling maken het opstellen van betrouwbare en realistische begrotingen extra lastig. Voorbeelden van die ontwikkelingen en veranderingen zijn er te over; denk hierbij aan vierde generatie hulpmiddelen, workbenches, prototyping, end-user computing enz. Door de snelheid waarmee nieuwe methoden, technieken en gereedschappen elkaar opvolgen, zal de ervaring die een ontwikkelaar met een specifiek hulpmiddel opdoet over het algemeen gering zijn, terwijl het gebruik van deze gereedschappen, technieken en methoden bij nieuwe projecten (naar men verwacht) aanzienlijk zal zijn. In een onderzoek van Rudolph (1983) constateert deze dat met de vierde generatietaal LINC van Burroughs per tijdseenheid twintig tot vijftig keer meer funktiepunten (een maat voor de omvang van software) geproduceerd worden dan met COBOL (zie figuur 1.5).

Ook Conte, Dunsmore en Shen (1986) verwachten dat tussen nu en wellicht tien jaar de productiviteit om programmatuur te ontwikkelen met een factor 2 tot 10 zal toenemen. Deze verbetering zal volgens hen voor een belangrijk deel toe te schrijven zijn aan een toename in het gebruik van vierde generatie hulpmiddelen. Ondanks deze en ook andere uitspraken over een spectaculaire toename van de produktiviteit blijkt dat dergelijke hulpmiddelen in de praktijk voorlopig nog slechts in beperkte mate worden gebruikt en dat de grote produktiviteitsverbeteringen sterk wordt bepaald door ondermeer de complexiteit

van de te ontwikkelen software en het ervaringsnivo van de ontwikkelaars. In hoofdstuk 2 zal nader worden ingegaan op het gebruik van vierde generatiehulpmiddelen en in hoofdstuk 5 worden enige kanttekeningen geplaatst bij mogelijke produktiviteitsverbeteringen.



Figuur 1.5 : Per tijdseenheid geproduceerde aantal funktiepunten meteen derde en een vierde generatie taal (zie ook Vonk 1987).

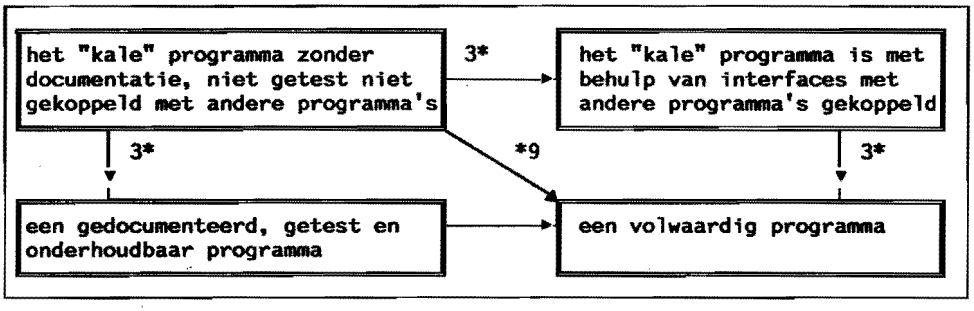
ad 6. *gebrekkige definities.*

Het blijkt dat een deel van de problemen bij het inschatten van ontwikkelkosten terug te voeren is tot een onduidelijke omschrijving van zaken als software, software-ontwikkeling, informatiesysteem ontwikkeling, automatiseringsproject enz. (Heemstra 1985).

Zo is het ontwikkelen van software meer dan alleen het produceren van code. Veel tijd wordt in beslag genomen door de eerste fasen van ontwikkeling, met name het vaststellen en analyseren van het objectsysteem, het achterhalen van de informatiebehoeften, het opstellen van specificaties en het maken van een ontwerp. Thibodeau

en Dodson (1981) tonen aan dat men bij de verdeling van ontwikkeltijd kan spreken van de 40-20-40 regel. 40% van de tijd komt voor rekening van analyse en ontwerp, 20% voor coderen en 40% voor testen.

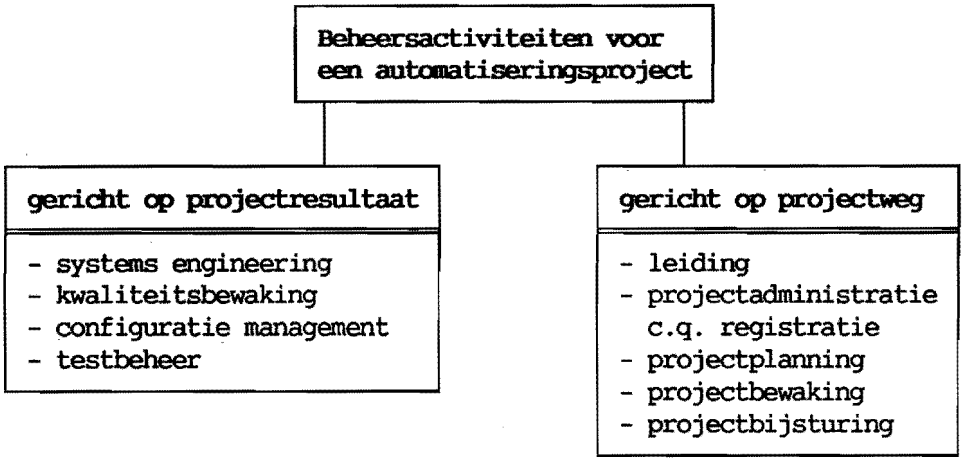
Bovendien dient men bij het calculeren van de ontwikkelkosten rekening te houden met activiteiten als conversie, opleiding en het maken van de documentatie. Zo toont Brooks (1975) aan wat de gevolgen van een onzorgvuldige definitie van software kunnen zijn op de ontwikkeltijd ervan. In figuur 1.6 wordt dit in beeld gebracht.



Figuur 1.6 : Verschillende omschrijvingen van een programma en de gevolgen daarvan voor de benodigde kosten, inspanning en doorlooptijd. De symbolen 3* en 9* geven aan dat de ontwikkelkosten met respectievelijk een factor 3 en 9 toenemen.

Ervaringscijfers leren dat het maken van een gedocumenteerd, getest en onderhoudbaar programma ongeveer driemaal zoveel inspanning kost als het maken van een "kaal" programma. Voor het koppelen met andere programma's moet de inspanning nogmaals met een factor drie verhoogd worden. Men moet de verhalen over programmatuur die in een recordtijd is geconstrueerd dus met de nodige argwaan aanhoren. In bijna alle gevallen betreft het hier software die onder andere op geen enkele manier is gedocumenteerd en daardoor moeilijk overdraagbaar en nauwelijks onderhoudbaar is. Verder is het veelal programmatuur die niet geïntegreerd is met andere software en niet of niet volledig is uitgetest. Als ook deze noodzakelijke ontwikkelactiviteiten worden ingevuld, zal de ontwikkeltijd en -inspanning met sprongen toenemen. Het voorafgaande duidt erop dat software-ontwikkeling meer is dan alleen maar coderen.

Zaken die als gevolg van een onzorgvuldige omschrijving van een project vaak over het hoofd worden gezien zijn de noodzakelijke beheersactiviteiten. Het gaat hierbij om activiteiten die van belang zijn tijdens de volledige uitvoering van een project en die een niet te verwaarlozen deel vormen van de totale kosten (Pressman 1987). In figuur 1.7 wordt aangegeven om welk soort activiteiten het hier gaat (Noth 1987).



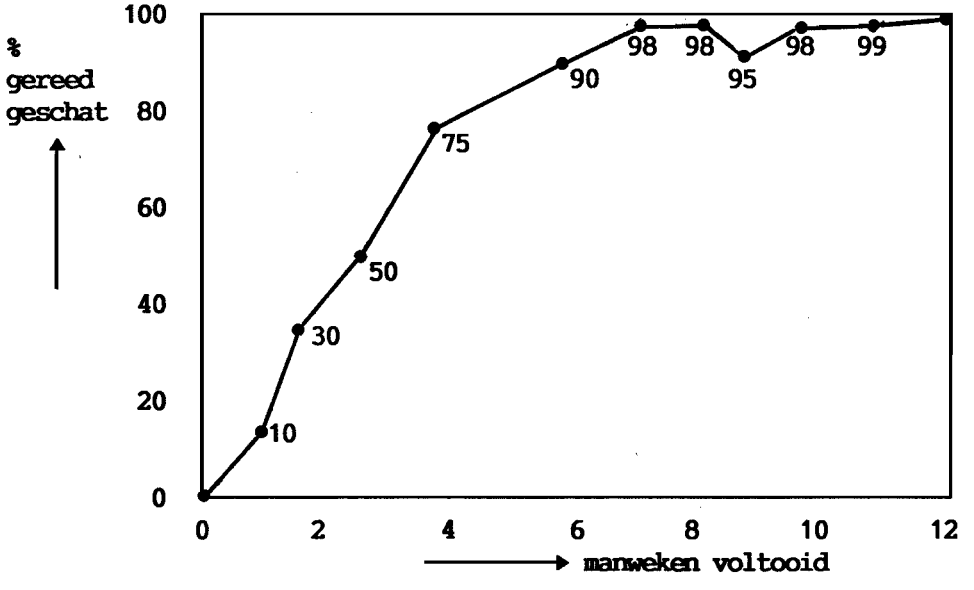
Figuur 1.7 : Activiteiten die uitgevoerd moeten worden voor het beheersen van software-ontwikkeling.

Hierbij is een onderscheid gemaakt in activiteiten die gericht zijn op het projectresultaat en activiteiten die gericht zijn op de projectweg. Een en ander betekent dat bij het begroten van softwarekosten en doorlooptijd niet alleen rekening gehouden moet worden met de inhoudelijke ontwikkelactiviteiten. Ook het beheer van die activiteiten kost tijd en geld.

ad 7. *onderschatting van inspanning en doorlooptijd.*

Systeemontwikkelaars blijken over het algemeen veel te optimistisch te zijn over de hoeveelheid tijd en inspanning die zij nodig denken te hebben voor de ontwikkeling van een systeem (Brooks 1975). Een verschijnsel waarbij dit optimisme tot uitdrukking komt is het be-

kende 90% syndroom (Abdel-Hamid 1988). In figuur 1.8 wordt de essentie hiervan in beeld gebracht.
Uit de figuur valt af te leiden dat na circa vijf weken de optimistische ontwikkelaar schat dat 90% van het project gereed is en dat voor de laatste 10% nog slechts een paar dagen nodig is. De ervaring leert evenwel dat juist die laatste "10%" die nodig is om het project af te ronden, nog veel tijd en inspanning kost, zodat men de facto na vijf weken nog pas 40% van de software gereed heeft. Een verklaring hiervoor is reeds in figuur 1.6 gegeven. Na vijf weken is het "kale programma" min of meer gereed. De tijd die nodig is om hiervan een volwaardig programma te maken, wordt hierbij zwaar onderschat. Daarnaast is men geneigd, bewust of onbewust naar een bepaalde uitkomst "toe te rekenen".



Figuur 1.8 : Het 90% syndroom (Abdel-Hamid 1988).

Dit geldt zeker als men als belanghebbende nauw betrokken is bij de te ontwikkelen software (Bemelmans 1987). In paragraaf 3.2 worden een aantal begrotingsmethoden behandeld waaronder de methode Price-to-Win en de "capaciteitsmethode". Beide methoden illustreren op welke gronden een onderschatting kan plaatsvinden.

ad 8. *de foutieve veronderstelling dat er een lineair verband is tussen tijd en capaciteit.*

Putnam en Fitzsimmons (1979) wijzen erop dat er geen lineair verband bestaat tussen benodigde capaciteit per tijdseenheid en beschikbare tijd. Een lineair verband zou betekenen dat de inspanning om software te ontwikkelen het produkt zou zijn van mensen en tijd. Een programma dat door 25 mensen in twee jaar gerealiseerd wordt, zou volgens deze redenering met 50 mensen in een jaar gereed zijn. De praktijk heeft uitgewezen dat deze veronderstelling onjuist is. Fred Brooks, projectleider voor de ontwikkeling van het operating system van de IBM 360, heeft dit verschijnsel uitvoerig en kleurrijk beschreven. Naar hem is de volgende wet genoemd:

"Adding people to a late project only makes it later".

De meest waarschijnlijke verklaring hiervoor is de volgende: naarmate het aantal projectmedewerkers toeneemt, zal het aantal menselijke interacties toenemen. Steeds meer tijd worden aan onderlinge communicatie en coördinatie worden besteed. Nieuwe medewerkers moeten met het systeem vetrouwd raken (inleertijd). Er blijft voor iedere medewerker steeds minder tijd over om daadwerkelijk aan het project te werken. Een en ander heeft tot gevolg dat er zelfs een punt bereikt wordt waarbij het inzetten van meer personeel niet leidt tot een verkorting maar tot een verlenging van de doorlooptijd. Ondanks deze wetmatigheid komt het maar al te vaak voor dat als oplossing voor een dreigende overschrijding van de levertijd in eerste instantie aan het inzetten van extra personeel wordt gedacht.

1.4. PROBLEEMSTELLING

Over kostenbeheersing en in het bijzonder begroten van softwareontwikkeling is in recente jaren veel gezegd maar relatief weinig onderzocht en geschreven. Zoals in de vorige paragrafen is aangegeven bestaan er nog volop misverstanden en onduidelijkheden over dit onderwerp. De literatuur biedt hier vooralsnog weinig aanknopingspunten. Het merendeel van de literatuur richt zich primair op het instrumentele aspect van kostenbegroting en beperkt zich tot een gedetailleerde behandeling van de vele parametrische modellen en de ontwikkeling van uiteenlopende metrieken. Storend daarbij is dat deze modellen en de bijbehorende literatuur in vele gevallen gebruik maken

van een telkens verschillend definitiestelsel en afwijkende uitgangspunten. Een en ander heeft geleid tot terminologische verwarring.

In de voorgaande paragrafen is een beeld geschetst van de noodzaak om betrouwbare en realistische begrotingen voor het ontwikkelen van software op te stellen. Tevens is aangegeven welke problemen hierbij kunnen optreden. De kosten van de software die geleverd wordt, zijn dikwijls aanzienlijk hoger dan oorspronkelijk geraamd. Dit gegeven heeft geleid tot de volgende doelstelling voor dit onderzoek.

> *Het ordenen en evalueren van de bestaande opvattingen en theorieën over en methoden voor het beheersen en begroten van software-ontwikkeling om vanuit bedrijfskundig perspectief theorieën en methoden te ontwikkelen voor een beter beheersen en in het bijzonder begroten van software-ontwikkeling.*

Alvorens aan te geven voor welke werkwijze gekozen is om deze doelstelling op te lossen, zullen eerst een aantal in dit onderzoek belangrijke begrippen nader worden omschreven.

Een centrale vraag die bij de ontwikkeling van software beantwoord zal moeten worden, is de volgende:

> ***Hoeveel geld, tijd, mensen, middelen (hardware, software-tools en dergelijke) moeten beschikbaar zijn (per fase c.q. activiteit van het project) om software te ontwikkelen die voldoet aan gestelde funktionele eisen en prestatie eisen.***

In deze omschrijving van begroten komt naar voren dat:

1. begroten een veel ruimere betekenis heeft dan het begroten van de kosten van een project. Zo zal er ondermeer een antwoord gegeven moeten worden op de vraag hoeveel personeel en wat voor een soort personeel (wat betreft ervaring, opleiding, vaardigheden) ingezet moeten worden op welke activiteiten en voor hoelang. In hoofdstuk 5 wordt een overzicht gegeven van de belangrijkste factoren waarover in een begroting een uitspraak moet worden gedaan.

2. In de omschrijving wordt tevens aangegeven dat het gaat om zaken nodig voor de *ontwikkeling* van software. Voor ontwikkeling wordt in dit onderzoek een ruime definitie gehanteerd; startend bij

de analyse van het objectsysteem, het in kaart brengen van de probleemsituatie (in ISAC termen: de veranderingsanalyse (Lundeberg 1982)) en eindigend bij de in gebruikname van het geteste eindprodukt. De fase informatieplanning valt buiten het kader van dit onderzoek. Datzelfde geldt voor het schatten van de kosten van onderhoud en exploitatie. Ontwikkelkosten en onderhoudskosten c.q. exploitatiekosten kunnen evenwel niet los van elkaar beschouwd worden. Zo zal een uitvoerige en grondige analyse van de probleemsituatie en een zorgvuldige uitvoering van de specificatiefase in eerste instantie leiden tot hogere ontwikkelkosten en dus een duurder eindprodukt. Deze meerkosten kunnen echter worden terugverdiend als gevolg van minder noodzakelijk onderhoud.

3. In de omschrijving wordt gesproken over fasen in de ontwikkeling van software. Impliciet wordt een gefaseerde aanpak van software-ontwikkeling verondersteld. In hoofdstuk 7 wordt deze veronderstelling genuanceerd. In dat hoofdstuk wordt beargumenteerd dat een decompositie van het ontwikkelproces in fasen en van de fasen in activiteiten en deze op hun beurt in taken een voorwaarde is voor een succesvolle ontwikkeling van software. Behalve een decompositie van het proces wordt in dit hoofdstuk ook nader ingegaan in een decompositie van het ontwikkelprodukt.

4. In de omschrijving wordt gesproken over software-ontwikkelprojecten. Door de definitie van Wijnen, Storm en Rennes (1984) van een project toe te spitsen op dergelijke projecten onstaat de volgende omschrijving:

   - Het is een geheel van onderscheidbare activiteiten.
   - Het is gericht op een of meer concrete resultaten; in dit geval op het ontwikkelen van software.
   - In deze resultaten stellen een of meer zelfstandige personen (ontwikkelaars, opdrachtgevers, gebruikers) duidelijk belang.
   - De resultaten dienen binnen een bepaalde tijdsperiode en met beperkte middelen (geld, hardware, software, personeel e.d.) bereikt te worden.
   - De resultaten omvatten voor de betrokken personen een of meer geheel nieuwe elementen.

   Verder gaat het bij dergelijke projecten over meestal grote projecten. In de praktijk van de automatisering is het geen uitzondering als dergelijke grote projecten de volgende kenmerken vertonen (Stehouwer 1988):

- de ontwikkeltijd bedraagt meer dan 50 mensjaren,
- de doelstellingen van het project zijn veelal vaag, bovendien zijn er dikwijls veel doelstellingen geformuleerd,
- bij de ontwikkeling van het eindprodukt wordt gebruik gemaakt van nieuwe technieken,
- de specificaties van het systeem zijn veelal instabiel. Bovendien worden tijdens de ontwikkeling van het systeem nieuwe specificaties toegevoegd,
- de projectorganisatie verandert tijdens de looptijd van het project,
- de doorlooptijd is meer dan twee jaar,
- bij het project zijn meerdere afdelingen en meerdere disciplines betrokken,
- vaak is er een gebrek aan ervaring of deskundigheid bij de ontwikkelaar en/of de opdrachtgever c.q. gebruiker,
- het nieuw ontwikkelde systeem noodzaakt tot conversies, koppeling met andere systemen en nieuwe werkmethoden,
- bevoegdheden en verantwoordelijkheden zijn over het algemeen onduidelijk geregeld. Een en ander leidt tot onduidelijke beslissingsprocessen.

5. In de omschrijving is de toevoeging opgenomen "die voldoet aan de gestelde functionele eisen en prestatie eisen", met ander woorden die voldoet aan zekere kwaliteitseisen. Functionele eisen geven aan welk soort gegevens door de software geleverd moet kunnen worden. Prestatie eisen geven aan onder welke voorwaarden deze gegevens geproduceerd moet worden. Qua voorwaarden kan man denken aan hoe snel (responsiesnelheid), hoe vaak (frequentie), hoe flexibel en aanpasbaar, in welke mate beveiligd, hoe betrouwbaar, hoe onderhoudbaar enz. (Bemelmans 1987). In de praktijk zal het vaker voorkomen dat de eisen in de beginfasen van het project niet volledig en duidelijk zijn. In zo'n geval wordt vaak een bepaald budget en tijd beschikbaar gesteld. Bijvoorbeeld: een informatie-analist krijgt 2 maanden de tijd om een haalbaarheidsstudie uit te voeren. Het principe van deze benadering is om uitgaande van een bepaalde hoeveelheid beschikbaar gestelde middelen, het resultaat te maximaliseren. In hoofdstuk 7 wordt nader ingegaan op verschillende benaderingswijzen bij het begroten van software-ontwikkeling.

1.5. VERANTWOORDING VAN DE GEVOLGDE WERKWIJZE

In dit onderzoek kunnen, in navolging van H. Simon (1964) de volgende fasen worden onderscheiden:

- intelligence,
- analysis,
- design.

Analoog aan deze fasering zijn in dit onderzoek drie delen te onderkennen.

- Het eerste gedeelte: "INTELLIGENCE".

  Het eerste gedeelte van dit onderzoek heeft voornamelijk een verkennend karakter. Het wordt beschreven in de hoofdstukken 1, 2 en voor een deel 3 en 4. Het accent ligt op een verkenning van het onderzoeksobject : het begroten van software-ontwikkeling. In dit eerste hoofdstuk is aandacht besteed aan het toenemende belang van software. Verder is gewezen op de problemen die er bestaan bij beheersen en in het bijzonder begroten. Door middel van een uitgebreid empirisch onderzoek onder Nederlandse organisaties die zich bezig houden met het ontwikkelen van software, is nagegaan of de alarmerende signalen over budget- en levertijdoverschrijdingen op waarheid berusten. Dit onderzoek en de resultaten ervan worden beschreven in hoofdstuk 2. De resultaten laten zien dat bij een belangrijk deel van de organisaties die aan dit onderzoek hebben deelgenomen niet of nauwelijks sprake is van kostenbeheersing van software-ontwikkeling. Zo worden budgetten en doorlooptijden inderdaad sterk overschreden; door 35% van de organisaties die aan het onderzoek hebben deelgenomen wordt geen begroting opgesteld, 50% van hen registreert niets van een project en 57% calculeert niet na. Het is voor de eerste keer dat op deze wijze een representatieve weergave wordt gegeven van praktijkopvattingen over dit onderwerp. Soortgelijke onderzoeken zijn veel fragmentarischer van opzet, beperken zich tot een of enkele organisaties en zijn veelal van Amerikaans origine. De resultaten van het veldonderzoek bevestigen de uitspraken uit de literatuur en geven meer dan voldoende redenen voor nader onderzoek. In het verkennende gedeelte van het onderzoek is verder een beschrijving van de bestaande begrotingsmethoden (paragraaf 3.2) en van de belangrijkste begrotingsmodellen (paragrafen 4.2 tot en met 4.8) opgenomen.

- Het tweede gedeelte : **"ANALYSIS".**

Het tweede gedeelte van het onderzoek heeft een sterk ordenend karakter. Zo worden de belangrijkste begrotingsmodellen geëvalueerd (paragrafen 4.2 tot en met 4.8), vergeleken (paragraaf 4.9) en binnen een typologie geplaatst (paragraaf 4.11). Verder is er een ordening gemaakt van factoren die van invloed zijn op kosten, inspanning en doorlooptijd. Een analyse van deze factoren leert dat ze betrekking hebben op een van de volgende vijf onderwerpen: het te ontwikkelen produkt, het ontwikkelpersoneel, de hulpmiddelen, de projectorganisatie en de gebruiker.
Alvorens in de "design fase" in te gaan op mogelijke verbeteringen in het begroten en beheersen van software-ontwikkeling, wordt in hoofdstuk 6 nader ingegaan op het beheersen c.q besturen van software-ontwikkeling. Hierbij wordt ondermeer gebruik gemaakt van het besturingsparadigma (de Leeuw 1974). In dit hoofdstuk worden de voorwaarden voor effectieve besturing volgens dit paradigma vertaald naar software-ontwikkeling. Verder wordt nader ingegaan op een aantal aspecten van besturen, zoals de aard van het besturingsprobleem, de manier van leidinggeven en coördineren, de ontwikkelstrategie en de funkties van een begroting.

- Het derde gedeelte : **"DESIGN".**

Vanuit deze inventarisatie en ordening zijn theorieën en methoden ontwikkeld die de basis moeten vormen voor een beter beheersen en begroten van software-ontwikkeling. In het derde gedeelte van het onderzoek zijn deze theorieën en methoden ontwikkeld.
Als basis hiervoor heeft met name het P-B-I model (Bemelmans 1986) gediend. In hoofdstuk 7 is dit model gebruikt als analyse-instrument voor het proces van software-ontwikkeling en als ontwerphulpmiddel van een beheersingsconcept voor software-ontwikkeling. Uitgangspunt van het P-B-I model is dat de karakteristieken van software-ontwikkeling (P) bepalend zijn voor de wijze van besturing (B). De wijze van besturing op zijn beurt determineert welke informatie (I) hiervoor noodzakelijk is. De P is beschreven aan de hand van een uitgebreid aantal kenmerken. Deze zijn uitgesplitst naar produkt-, proces- en middelenkenmerken. Omdat de zekerheid wat betreft deze kenmerken van ontwikkelsituatie tot ontwikkelsituatie kan verschillen, zal ook de wijze van beheersen en begroten variëren. Om hierin een ordening aan te brengen, is in hoofdstuk 7 een typologie van beheersen en begroten ontwikkeld. Uitgangspunt hierbij is dat elk proces van software-ontwikkeling onder te brengen is in vier te onderscheiden ideaaltypen. Bij elk ideaaltype behoort een bepaalde aanpak van beheersing en begro-

ting. De mate van produkt-, proces- en middelenzekerheid bepaalt tot welke ideaaltype een bepaalde software-ontwikkeling gerekend moet worden.
In hoofdstuk 8 wordt inhoud gegeven aan de informatie (I) die nodig is om software te begroten. Zo wordt in paragraaf 8.5 een raamwerk voor het begroten van software gepresenteerd. In het raamwerk zijn alle stappen opgenomen die voor het beheersen en begroten van software-ontwikkeling nodig zijn. Bovendien is bij elke stap aangegeven wat de vereiste invoer en de uitvoer is. Op een onderdeel van dit raamwerk wordt in paragraaf 8.6 nader ingegaan: een databank van voltooide projecten. Aangegeven wordt welke soort gegevens in een dergelijke databank opgenomen moeten worden.

In deze studie is geen poging ondernomen het zoveelste begrotingsmodel te ontwikkelen. Dit zou zeker een uitdaging zijn geweest, gezien de gebrekkige kwaliteit van de bestaande modellen. Uit de verkennende en ordenende fase van het onderzoek kwam echter naar voren dat er allereerst een gebrek is aan een duidelijke visie van de rol van begroten binnen het beheersen van projecten. Vandaar de centrale positie van de typologie van beheersings- en begrotingssituaties en van het raamwerk.

2. BEGROTEN EN BEHEERSEN VAN SOFTWARE-ONTWIKKELING : EEN EMPIRISCH ONDERZOEK

2.1. INLEIDING

Door de literatuur over begroten van software en door de dagelijkse praktijk van software-ontwikkeling wordt het beeld geschapen, dat de problemen met betrekking tot kostenbeheersing en in het bijzonder met betrekking tot het onderdeel begroten, groot zijn. In hoofdstuk 1 is gewezen op de relevantie van deze problematiek.
In dit hoofdstuk zal nagegaan worden of dit beeld reëel is en of de diverse uitspraken, die veelal niet onderbouwd zijn met empirisch materiaal, terecht zijn. Een tweede doel dat in dit hoofdstuk zal worden nagestreefd, is het geven van een overzicht van de stand van zaken bij beheersing c.q. begroting van software-ontwikkeling in Nederland. Zoals uit het verdere verloop zal blijken, verschillen de empirische gegevens niet wezenlijk van de geldende opvattingen en wordt de argumentatie versterkt voor onderzoek naar het begroten van software-ontwikkelprojecten.

De resultaten van het empirisch onderzoek laten zien dat het matig gesteld is met automatiserend Nederland. Zo geeft 35% van de responderende organisaties te kennen niet een begroting op te stellen, 57% calculeert niet na en 50% registreert niets van een project.

In paragraaf 2.2 worden de doelstelling en de opzet van het empirisch onderzoek nader toegelicht. Paragraaf 2.3 gaat in op de wijze waarop het onderzoek is uitgevoerd en bevat informatie over de enquetering. Paragraaf 2.4 geeft een overzicht van de stand van zaken wat betreft kostenbeheersing in Nederland. In paragraaf 2.5 worden de hypothesen getoetst, die in paragraaf 2.2 geformuleerd zijn. Het hoofdstuk wordt afgesloten met conclusies (paragraaf 2.6).
Terzijde dient vermeld te worden dat dit het eerste grootschalige empirische onderzoek is dat de stand van zaken op het gebied van begroten van software-ontwikkeling in Nederland in kaart brengt.

Vergelijkbare onderzoeken over dit onderwerp zijn over het algemeen veel kleinschaliger van opzet en richten zich veelal op één specifiek aspect van begroten.

2.2. HET ONDERZOEK

Het onderzoek waar het hier om gaat, heeft voornamelijk een beschrijvend en toetsend karakter. Bij de aanvang van het onderzoek zijn allereerst diverse hypothesen geformuleerd. Deze zijn met behulp van de ontvangen enqueteresultaten getoetst. Het doel van het beschrijvende gedeelte is het geven van een overzicht van de stand van zaken bij het beheersen c.q. begroten van software-ontwikkelprojecten in Nederland. Om deze doelstelling te realiseren zijn in de enquete vragen opgenomen die betrekking hebben op:

- de positionering van de responderende organisaties:
  "wie is men"
- het begroten, registreren en nacalculeren door de responderende organisaties:
  "wat doet men"
- budget- en doorlooptijdoverschrijdingen bij de responderende organisaties:
  "hoe goed beheerst men software-ontwikkeling"
- ervaringen en meningen van de responderende organisaties over kostenbeheersing van software-ontwikkeling:
  "wat denkt men"

Op basis van literatuuronderzoek, gesprekken met software-ontwikkelaars en ideeën die leefden bij de researchgroep "Begroten van Automatiseringsprojecten" van de Technische Universiteit Eindhoven, zijn in het toetsend gedeelte zijn de volgende hypothesen getoetst:

1. Hoe meer een organisatie bij het opstellen van een begroting zich laat leiden door commerciële motieven, dat wil zeggen gebruik maakt van de methode Price-to-Win, of bij het begroten uitgaat van de beschikbare capaciteit, dat wil zeggen gebruik maakt van de "capaciteitsmethode", des te hoger is de overschrijding van het budget en levertijd.
2. Hoe vaker een organisatie voor een en hetzelfde project een begroting opstelt, des te lager is de overschrijding van het budget en levertijd.

3. Hoe gedetailleerder een organisatie een begroting opstelt, des te lager is de overschrijding van het budget en de levertijd.
4. Hoe vroeger een organisatie in een project voor de eerste keer een begroting opstelt, des te groter is de overschrijding van het budget en de levertijd ten opzichte van die eerste begroting.
5. Hoe intensiever een organisatie gebruik maakt van begrotingsmodellen, des te lager is de overschrijding van budget en de levertijd.
6. Hoe intensiever een organisatie gebruik maakt van vierde generatiehulpmiddelen, des te lager is de overschrijding van het budget en levertijd.

2.3. DE ENQUETE

De enquete is in december 1987 en januari 1988 voorgelegd aan functionarissen die binnen hun organisatie verantwoordelijk zijn voor de beheersing en begroting van software-ontwikkeling. Per deelnemende organisatie is één enquete verstuurd. Bij het achterhalen van de adressen is gebruik gemaakt van een geselecteerd adressenbestand. Overwegingen die een rol hebben gespeeld bij de selectie van het meest geschikte adressenbestand waren:

- *selektiemogelijkheden* binnen het bestand. Is het mogelijk uit een adressenbestand de gewenste doelgroep, dat wil zeggen de organisaties in Nederland die ondermeer software ontwikkelen, te selecteren,
- *representativiteit* van het bestand. Geeft het adressenbestand wat betreft branche, bedrijfsomvang en omvang van de automatiseringsafdeling een soortgelijke verdeling als de landelijke verdeling volgens de cijfers van het Centraal Bureau Statistieken.
- *beschikbaarheid* van het bestand. Hierbij spelen zaken een rol als levertijd, prijs en omvang van het bestand.

Op grond van het voorgaande is gekozen voor een bepaald adressenbestand. Hieruit zijn allereerst de adressen van de doelgroep geselecteerd. Dit aantal bleek te groot, vandaar dat er aselect 2659 adressen uit de geselecteerde doelgroep zijn getrokken. Hiervan hebben 597 organisaties gereageerd, hetgeen een respons van ruim 22% betekent. Deze respons is voor een schriftelijke enquete bevredigend (Fink en Kosecof, 1985). De enquete is op een zodanige wijze opgesteld en verwerkt dat de *anonimiteit* van de responderende organisaties ge-

waarborgd kon worden.

Voordat de vragenlijst is verstuurd, heeft een testronde plaatsgevonden. Aan een aantal ervaren projectmanagers is gevraagd de vragenlijst in te vullen en te becommentariëren. Als gevolg daarvan is de lijst op een aantal punten aangepast.

De eerste stap bij het verwerken van de 597 reacties was het onderzoeken of de respons als representatief beschouwd mocht worden ten opzichte van de verzonden enquetes. Met andere woorden zijn er bijvoorbeeld branches die een significant kleinere of grotere respons hebben dan de andere.
In tabel 1 en 2 is een overzicht gegeven van de verdeling van de terugontvangen vragenlijsten over de branches en bedrijfsomvang.

Tabel 2.1 : Overzicht terugontvangen vragenlijsten naar branches.

BRANCHE	AANDEEL IN % VERZONDEN ENQUETES	AANDEEL IN % TERUGONTVANGEN ENQUETES
overheid en semi-overheid	35,3	24,3
industrie	20,1	20,3
overige dienstverlening	1,5	14,3
handel, horeca, reparatiebedrijven	12,5	10,8
bank en verzekeringswezen	11,3	7,8
bouwnijverheid, installatiebedrijven	5,6	6,0
transport, opslag, communicatiebedrijven	4,1	4,8
openbare nutsbedrijven	1,4	4,4
softwarehuizen/systeemhuizen	6,9	4,1
landbouw en visserij	1,0	3,0
delfstoffenwinning	0,3	0,2
	100	100

tabel 2.2 : Overzicht terugontvangen vragenlijsten naar bedrijfsomvang.

BEDRIJFSOMVANG	AANDEEL IN % VERZONDEN ENQUETES	AANDEEL IN % TERUGONTVANGEN ENQUETES
minder dan 20 personen	1,0	1,4
20 t/m 49 personen	17,8	8,9
50 t/m 99 personen	27,5	21,0
100 t/m 199 personen	23,5	24,5
200 t/m 499 personen	19,6	21,0
meer dan 500 personen	10,6	22,2
	100	100

Na toetsing bleek er een significant verschil te zijn tussen de verdeling van verzonden en terugontvangen vragenlijsten bij zowel branche als bedrijfsomvang. Bij branche is het aantal responderende organisaties in de categorie Overige Dienstverlening in de terugontvangen enquetes significant hoger ten opzichte van de verzonden enquetes. Slechts 1,5% van het totaal aantal enqueteformulieren is verzonden naar organisatie uit de categorie Overige Dienstverlening, terwijl van de terugontvangen enquetes 14,3% zich tot deze branche rekende. Er zijn zelfs deze branche zelfs meer enquetes terugontvangen dan verzonden (81 terug ten opzichte van 38 verzonden).
Als verklaring kan gelden dat een aantal organisaties zichzelf liever rekenden tot de branche Overige Dienstverlening, terwijl deze in het adressenbestand in andere categorieën waren ingedeeld.
Bij de branche Overheid en Semi-overheid bleek dat 24,3% van de responderende organisaties zich tot deze branche rekende, terwijl 35,3% van de enquetes verzonden is naar deze branche.

Wat betreft bedrijfsomvang valt op dat de responderende organisaties uit de grootste categorie (meer dan 500 personen) oververtegenwoordigd zijn. Van de terugontvangen enquetes bleek 22,2% van de responderende organisaties tot deze categorie te behoren, terwijl slechts 10,6% van de enquetes is verzonden naar organisaties uit die categorie. Een sluitende verklaring is hier niet voor gevonden. Het vermoeden bestaat dat bij grote bedrijven het probleem van kostenbeheersing van software-ontwikkeling meer leeft en men daardoor eerder bereid is medewerking te verlenen. Wellicht bestaat er bij grotere bedrijven ook meer ruimte voor het afhandelen van enquetes en is men bovendien meer gewend aan enqueteringen.

De terugontvangen vragenlijsten zijn verwerkt met behulp van het softwarepakket SPSS. Bij het interpreteren van de tabellen dient men rekening te houden met het feit dat er weliswaar 597 enquetes zijn terugontvangen, maar dat niet elk antwoord geschikt was voor verwerking. Dit kan zijn omdat een vraag niet, niet juist of onduidelijk beantwoord is (de zogenaamde missing values). Hierdoor kan het voorkomen dat het aantal geldige antwoorden van vraag tot vraag kan verschillen en dus ook de totalen in de verschillende tabellen.

2.4. STAND VAN ZAKEN : FEITEN EN MENINGEN

Zoals in paragraaf 2 is aangegeven zijn in de enquete vier categorieën vragen opgenomen om een beeld te krijgen van de stand van zaken wat betreft de kostenbeheersing van software-ontwikkelprojec-

ten in Nederland. Deze vier categorieën hebben voor de responderende organisaties betrekking op:

- de positionering,
- de beheersing van kosten en levertijd,
- de overschrijdingen,
- de ervaringen.

De Paragrafen 2.4.1 tot en met 2.4.4 zullen gewijd zijn aan de behandeling van deze categorieën.

2.4.1 POSITIONERING VAN DE RESPONDERENDE ORGANISATIES

Allereerst diende duidelijkheid te bestaan over:

- wie zijn de responderende organisaties (kortweg: respondenten),
- welk soort software-ontwikkelprojecten worden in deze organisaties uitgevoerd.

respondenten

Meer dan de helft van de functionarissen die de enquete hebben ingevuld, blijkt een leidinggevende functie te hebben (62%). Het percentage functionarissen dat een uitvoerende c.q. adviserende func

Tabel 2.3 : Terrein van werkzaamheden van de functionarissen die de vragenlijst hebben ingevuld. Door 91 van de 597 responderende organisaties is deze vraag niet beantwoord.

TERREIN VAN WERKZAAMHEDEN	aantal	%
controlling/administratie	126	25
beheer hardware/systeemsoftware	115	23
projectmanagement	97	19
systeemanalyse/programmering	49	10
informatieanalyse	32	6
overige	87	17
TOTAAL	506	100

tie heeft is respectievelijk 19% en 17%.
Verder blijkt dat het merendeel van de terugontvangen enquetes afkomstig is van organisaties waarvan de afdeling automatisering tussen de 2 en 10 medewerkers heeft (51%). Ruim 27% van de terugontvangen enquetes komt van organisaties waar slechts één persoon zich direct met automatisering bezig houdt. Bij 13% van de respondenten is er sprake van automatiseringsafdelingen met meer dan twintig medewerkers.
In tabel 2.3 is aangegeven wat de primaire werkzaamheden zijn van de functionarissen die de vragenlijst hebben ingevuld.

softwareprojecten

Uit de antwoorden op de vragen over softwareprojecten blijkt dat bij de meeste organisaties het grootste gedeelte van de ontwikkelinspanning, uitgedrukt in aantal mensmaanden, toegeschreven moet worden aan kleine en middelgrote projecten (een project wordt als klein beschouwd als er minder dan 12 mensmaanden ontwikkelinspanning nodig is. Middelgrote projecten vragen tussen de 12 en 48 mensmaanden). Gemiddeld gaat per geënquêteerde organisatie 60% van de totale ontwikkelinspanning naar de eerste categorie projecten en 28% naar de tweede. Negen procent wordt besteed aan grote projecten, dat wil zeggen projecten tussen de 48 en 200 mensmaanden. Slechts drie procent komt voor rekening van zeer grote projecten (meer dan 200 mensmaanden).
Verder blijkt dat per geënquêteerde organisatie gemiddeld 81% van de softwareprojecten gericht is op de ontwikkeling van administratieve applicaties en dat het grootste gedeelte van de software, namelijk 82%, ontwikkeld is voor intern gebruik.

2.4.2 BEGROTEN, REGISTREREN EN NACALCULEREN

In de enquete is een groot aantal vragen opgenomen die betrekking hebben op de onderwerpen begroten, registreren en nacalculeren. De antwoorden op deze categorie vragen beogen inzicht geven in wat door de responderende organisaties aan kostenbeheersing wordt gedaan.

Begroten

De vragen over dit onderwerp hebben betrekking op:

a. begroten in zijn algemeenheid
b. begrotingsmethoden

ad a. ***begroten in zijn algemeenheid***

Dit zijn vragen als:

- wordt voor elk softwareproject een begroting gemaakt.
- met welke zaken worden hierbij rekening gehouden.
- wie worden betrokken bij het opstellen van een begroting en wanneer en hoe vaak wordt een begroting opgesteld.

Deze categorie vragen geeft een indicatie over de wijze waarop en de kwaliteit waarmee een begroting wordt opgesteld.

Uit de antwoorden blijkt dat 65% van de responderende organisaties een begroting opstelt voor elk project. Dit komt erop neer dat maar liefst 35% te kennen geeft *geen* begroting te maken. Dit percentage blijkt voor bedrijven uit de branches bouwnijverheid/installatiebedrijven en openbare nutsbedrijven nog veel hoger te liggen. In tabel 2.4 zijn de meest opvallende cijfers hierover weergegeven.

Tabel 2.4 : Overzicht van organisaties per branche die wel/niet een begroting opstellen. Van de 597 responderende organisaties hebben er 40 deze vraag niet beantwoord.

	wel begroten		niet begroten	
BRANCHE	aantal	%	aantal	%
bouwnijverheid/installatiebedrijven	12	35	22	65
openbare nutsbedrijven	10	42	14	58
softwarehuizen/systeemhuizen	22	96	1	4
alle responderende organisaties	364	65	193	35

Uit de resultaten blijkt dat van de softwarehuizen, die gereageerd hebben, 96% wel begroten. In deze branche was er slechts één bedrijf dat te kennen gaf geen begrotingen op te stellen. Voor de overige branches wijken de cijfers nauwelijks af van de genoemde 65% in tabel 2.4.

Van de 364 bedrijven die te kennen hebben gegeven dat in hun organisaties *wel* een begroting wordt opgesteld, wordt in tabel 2.5 aangegeven over welke zaken in de begroting een uitspraak wordt gedaan.

Tabel 2.5. : Overzicht van onderwerpen die in een begroting worden opgenomen. De betreffende vraag is door 364 organisaties beantwoord. Bij het invullen van vraag konden meerdere onderwerpen worden aangekruist.

ONDERWERP DAT IN DE BEGROTING WORDT OPGENOMEN	aantal	%
benodigd budget in geld	305	82
benodigde hardware	279	75
geplande doorlooptijd	274	74
benodigde software	248	67
benodigde inspanning (mensmaanden)	211	57
benodigd personeel per categorie	172	46
overhead (reiskosten e.d.)	84	23
overige zaken	42	11

De enqueteresultaten in tabel 2.5 laten zien dat een aantal organisaties, die zeggen wel te begroten, de meest relevante zaken, zoals het benodigd budget (gemeten in geld) en de geplande doorlooptijd, niet in de begroting opnemen. Een verklaring hiervoor zou kunnen zijn, dat deze organisaties begroten beschouwen als een capaciteitsprobleem. Deze veronderstelling is getoetst en verworpen.
Een en ander kan betekenen dat een begroting in dergelijke organisaties voornamelijk een voorwaardenscheppende funktie heeft, met andere woorden dient voor het veilig stellen van middelen. Een begroting kan niet als stuurinstrument optreden. Vergelijken van begroting met realisatie en gericht bijsturen op basis van geconstateerde verschillen hiertussen is niet mogelijk als geplande doorlooptijd en budget niet zijn vastgelegd. Deze vooronderstelling kon niet worden gecontroleerd, omdat in de vragenlijst geen vragen naar de funktie van een begroting zijn opgenomen. In hoofdstuk 6 wordt nader ingegaan op mogelijke funkties van een begroting.
Bij het opstellen van een begroting zijn de projectleider en het

management verreweg de belangrijkste betrokkenen. Respectievelijk 60 en 80% van de responderende organisaties geeft dit te kennen. Uit de antwoorden blijkt verder dat er binnen 17 (= 5%) organisaties een aparte functionaris bestaat, die belast is met de taak begroten.

Bij het beantwoorden van de vraag "Wanneer wordt een automatiseringsproject voor de eerste keer begroot" kon gekozen worden uit drie mogelijkheden:

1. na een eerste globale omschrijving,
2. na het opstellen van de specificaties,
3. na de afronding van het ontwerp.

Bovendien moest de responderende organisatie bij het geven van het antwoord rekening houden met de omvang van het softwareproject. Uit de antwoorden blijkt dat naarmate een project in omvang toeneemt, er vaker een begroting wordt opgesteld. Bovendien neemt het detailleringsniveau toe bij een toename van de projectomvang. In de tabellen 2.6 en 2.7 is dit weergegeven.

Tabel 2.6 : Overzicht van het aantal malen dat een project wordt begroot, afhankelijk van de projectomvang (MM =mensmaanden).

	PROJECTOMVANG							
	< 12 MM		12 - 48 MM		49 - 200 MM		> 200 MM	
BEGROTINGSFREQUENTIE	aantal	%	aantal	%	aantal	%	aantal	%
nooit	112	37	38	15	12	10	0	0
1 maal	145	48	95	37	30	25	7	18
2 maal	22	8	66	26	34	29	9	23
3 tot 5 maal	16	5	54	20	29	25	15	39
meer dan 5 maal	5	2	5	2	13	11	9	22
TOTAAL	300	100	258	100	118	100	39	100

Er blijkt hierbij een duidelijk verschil te bestaan tussen kleine projecten (minder dan 12 mensmaanden) en de overige projecten.

Voor kleine projecten wordt in het merendeel van de gevallen slechts eenmaal een begroting voor het totale project gemaakt en is er geen opsplitsing in de begroting gemaakt naar de verschillende

fasen c.q. activiteiten. Een verklaring hiervoor kan zijn dat de financiële risico's bij kleine projecten geringer zijn en de te realiseren produkten minder complex zijn.

Tabel 2.7 : Overzicht van het detailleringsnivo van een begroting, afhankelijk van de projectomvang (MM = mensmaanden). De totalen in de kolommen aantal geven aan het aantal organisaties dat te kennen geeft projecten uit te voeren van een omvang < 12 MM, 12-48 MM, etc. Deze totalen moeten in tabel 2.6 en 2.7 gelijk zijn. Dit is echter niet het geval voor het totaal aantal projecten kleiner dan 12 mensmaanden (300 versus 291) en projecten met een omvang tussen de 12 en 48 mensmaanden (258 versus 269). De verklaring voor deze verschillen is dat een aantal organisaties een van beide vragen niet of niet juist heeft beantwoord (de eerder genoemde missing values).

DETAILLERINGSNIVO	PROJECTOMVANG							
	< 12 MM		12 - 48 MM		49 - 200 MM		> 200 MM	
	aantal	%	aantal	%	aantal	%	aantal	%
het totale project	232	79	122	45	32	27	8	21
per fase van project	31	11	111	42	54	46	18	45
per activiteit per fase	28	10	36	13	32	27	13	34
TOTAAL	291	100	269	100	118	100	39	100

Uit de antwoorden blijkt verder dat er een verschil is tussen kleine projecten enerzijds en middelgrote, grote en zeer grote projecten anderzijds wat betreft het moment waarop voor de eerste keer voor een project een begroting wordt opgesteld. Bij de middelgrote, grote en zeer grote projecten geldt een verdeling 40 - 40 - 20; dat wil zeggen dat in 40 % van de gevallen een eerste begroting plaats vindt na een eerste globale omschrijving, in 40% na het opstellen van de specificaties en in 20% van de gevallen na de afronding van het ontwerp. Voor kleine projecten is deze verdeling 60 - 30 - 10. Het is moeilijk een passende verklaring te geven voor deze verschuiving. Een mogelijkheid kan zijn dat men bij kleine projecten het eerder aan-

durft een begroting op te stellen, ondanks de onzekerheden die er bij aanvang van een project bestaan. De financiële risico's bij kleine projecten zijn dan ook een stuk minder.

ad b. ***begrotingsmethoden***

Over dit onderwerp zijn vragen gesteld als:

- Welke begrotingsmethoden worden gebruikt,
- Maakt men gebruik van een begrotingsmodel en zoja, welk model,
- Wordt elk project, ongeacht de omvang, met behulp van een model begroot,
- Welke problemen worden ervaren bij gebruik van modellen.

Deze vragen beogen een indicatie te geven van de mate waarin begrotingsmethoden c.q. modellen in Nederland worden gebruikt. Bovendien krijgt men een idee hoe serieus men bezig is bij het opstellen van een begroting. De achterliggende veronderstelling is dat gebruikers van begrotingsmodellen serieus nagedacht hebben over het onderwerp begroten. Dergelijke modellen dwingen de gebruiker uitspraken te doen over een groot aantal factoren die van invloed zijn op de kosten.

De meest gebruikte begrotingsmethoden zijn: "afgaan op intuïtie en ervaring" en "gebruik maken van project- c.q. begrotingsgegevens van soortgelijke projecten uit het verleden". Beide methoden worden door 60% van de responderende organisaties genoemd. Het inschakelen bij het begroten van een expert wordt door 26% van de respondenten genoemd. Een groot aantal organisaties ziet het begroten als een vraagstuk van beschikbaarheid van capaciteit. Met andere woorden: hoeveel mensen kunnen we vrijmaken voor een project. Deze methode wordt 77 maal (= 21%) genoemd. Er zijn maar weinig organisaties die zeggen zich te laten leiden door commerciële motieven bij het opstellen van een begroting (8%).

51 organisaties (= 14%) geven te kennen dat zij gebruik maken van een model bij het opstellen van een begroting. In tabel 2.8 wordt het gebruik van de begrotingsmethoden door de responderende-organisaties weergegeven.
In hoofdstuk 3 komen de verschillende begrotingsmethoden uitgebreid aan de orde en zal nader worden ingegaan op begrotingsmodellen.

Tabel 2.8 : Overzicht van gebruikte begrotingsmethoden. Bij het invullen van de vraag waren meer antwoorden toegestaan.

BEGROTINGSMETHODE	%
- afgaan op intuïtie en ervaring	62
- gebruik maken van project- c.q. begrotingsgegevens van soortgelijke projecten uit het verleden	61
- een expert op dit gebied inschakelen	26
- begroten zien als een capaciteitsvraagstuk: Hoeveel mensen kunnen we vrijmaken voor een project ?	21
- gebruik maken van begrotingsmodellen	14
- overige methoden	9
- de begroting alleen laten afhangen van commerciële motieven (price to win)	8

Uit de antwoorden op de vraag "Welk begrotingsmodel gebruikt U" blijkt dat Funktie Punt Analyse (FPA) verreweg het meest wordt gebruikt. Verrassend is dit resultaat niet als men rekening houdt met het feit dat 82% van de responderende organisaties voornamelijk administratieve applicaties ontwikkeld en dat FPA juist gericht is op het begroten van dit soort toepassingen. In tabel 2.9 is een overzicht gegeven van de gebruikte modellen.

Tabel 2.9. : Overzicht van gebruikte begrotingsmodellen. Het totaal aantal organisaties dat de betreffende vraag heeft beantwoord, is 51. Bij het invullen van deze vraag waren meerdere antwoorden toegestaan.

MODEL	aantal	%
Funktie Punt Analyse	45	52
overige modellen	36	42
COCOMO	4	5
ESTIMACS	4	5
Putnam/SLIM	3	4
PRICE-S	2	2
SPQR	0	0

Uit dit overzicht blijkt dat de groep "overige modellen" bijzonder groot is. Het was bij het analyseren van de antwoorden niet mogelijk na te gaan of het hier ging om zelf ontwikkelde modellen of commercieel verkrijgbare modellen. Het vermoeden bestaat dat het hier voornamelijk moet gaan om zelf ontwikkelde modellen, omdat de meest gangbare (commerciële) modellen in de enquete zijn genoemd. Er blijkt een duidelijke relatie te bestaan tussen het wel of niet gebruiken van een model en de omvang van het project. Naarmate een organisatie grotere projecten uitvoert, neemt het gebruik van begrotingsmodellen toe. Zo maakt 70% van de responderende organisaties, die te kennen hebben gegeven hoofdzakelijk projecten van meer dan 12 mensmaanden uit te voeren, gebruik van een begrotingsmodel. Ook blijkt dat modellen meer worden toegepast naarmate de omvang van de organisaties toeneemt.

In de hoofdstukken 3 en 4 zal nader in worden gegaan op begrotingsmethoden en -modellen. Hierop anticiperend, is in de vragenlijst de vraag opgenomen: "welke twee problemen ervaart als de meest belangrijke bij het opstellen van een begroting". Door de modelgebruikers worden de volgende drie antwoorden het meest gegeven:

1. subjectiviteit bij het inschatten van de kostenbepalende factoren (28%),
2. bepalen van de omvang van de software (21%)
3. onvoldoende inzicht in het project om de modelparamaters te kunnen invullen (19%)

De modelgebruikers geven aan dat zij minder problemen ervaren met het afstemmen van een begrotingsmodel op de eigen organisatie (calibreren). Wellicht dat men zelfs niet eens toekomt aan dit calibreren.

registreren.

Tot deze categorie behoren vragen als: vindt er registratie van projectgegevens plaats, en zoja, wat wordt er dan geregistreerd. Met behulp van deze vragen kan nagegaan worden hoe serieus de responderende organisatie zich bezig houden met het beheersen c.q. begroten van softwareprojecten. De achterliggende gedachte is dat beheersen en begroten slechts mogelijk is als tijdens de uitvoering van een project registratie plaats vindt en deze vastgelegde gegevens tijdens het project worden gebruikt voor de voortgangsbewaking en na afloop van een project geaggregeerd worden tot gegevens over voltooide

projecten. In hoofdstuk 8 zal nader worden ingegaan op de relatie tussen beheersen, begroten en registreren.
Bij 50% van de responderende organisaties vindt géén registratie van projectgegevens plaats. Dit blijken 276 organisaties te zijn. Uit een verdere analyse van de antwoorden blijkt dat 63 organisaties uit deze groep beweert toch een begroting op te stellen op basis van gegevens van afgesloten projecten. Blijkbaar gaat men in dergelijke gevallen af op intuïtie en ervaring of op het oordeel van een expert, en maakt men hierbij impliciet gebruik van (niet geregistreerde) projectervaring. Van de 23 softwarehuizen die deze vraag beantwoord hebben, geven 3 (= 13%) te kennen geen projectgegevens vast te leggen.
In tabel 2.10 is aangegeven welke projectgegevens worden geregistreerd.

Tabel 2.10 : Overzicht van de onderwerpen waarover registratie plaats vindt. Het totaal aantal organisaties dat registreert is 276. 45 organisaties gaven geen (juist) antwoord op de vraag "registreert u projectgegevens". Bij het invullen van deze vraag waren meerdere antwoorden toegestaan.

ONDERWERP	aantal	%
doorlooptijd	234	82
besteed budget in geld	219	77
personeelsinzet	209	73
oorzaken van afwijkingen in tijd/kosten	208	73
kwaliteit van het eindprodukt	185	65
omvang van het eindprodukt	178	62
gebruikte apparatuur, tools	156	55
verdeling middelen naar fasen	137	48
verdeling middelen naar activiteiten	121	42
kwaliteit personeel	120	42
hergebruik van software	102	36

77% van de registrerende bedrijven legt geen gegevens vast over het bestede budget in geld. Verder is het opvallend dat maar weinig organisaties gegevens vastleggen over de mate waarin gebruik is gemaakt van bestaande code, ontwerpen, specificaties e.d. In de literatuur (Jones 1984, Martin 1984, Boehm 1988) wordt hergebruik

een factor genoemd die een belangrijke invloed heeft op de productiviteit en de kosten van software-ontwikkeling. Registratie van de mate van hergebruik kan een organisatie hierover informatie leveren. Dit soort informatie is waardevol bij het opstellen van een begroting.

Bij de interpretatie van tabel 2.10 dient men ermee rekening te houden dat deze cijfers betrekking hebben op de organisaties die *wel* registreren. Een en ander betekent bijvoorbeeld dat van alle organisaties die gereageerd hebben niet 77% vastlegt hoeveel geld uitgegeven is voor de uitvoering project, maar slechts 37,5%. Immers 50% van de responderende organisaties registreert niets.

Nacalculeren

Een belangrijk onderdeel van het beheersen van software-ontwikkeling is, behalve plannen en begroten, het onderdeel nacalculeren. Met andere woorden: wat mag er besteed worden, wat is er besteed en wat is de oorzaak van eventuele verschillen hiertussen. Uit de antwoorden op de vragen over dit onderwerp komt naar voren dat slechts 43% van de organisaties nacalculeert. Dit geeft een indicatie van de kwaliteit van het beheersen c.q. begroten van projecten in dergelijke organisaties. Gesteld kan worden dat van beheersing geen sprake is als men na afloop geen idee heeft hoeveel middelen men besteed heeft.

> Voor meer dan de helft van de organisaties die de vragenlijst hebben ingevuld, geldt dit.

Een uitsplitsing naar branche geeft een opvallende uitschieter: 13% van de softwarehuizen calculeert niet na. Voor de overige branches geldt dat er geen significante afwijking is van de eerder genoemde 43%.

2.5. OVERSCHRIJDINGEN

Een belangrijk onderwerp uit de vragenlijst is de vraag naar de omvang van de overschrijding van het budget en de geplande doorlooptijd. Met behulp van deze vraag wordt getracht een indicatie te krijgen van de kwaliteit van de begroting. In tabel 2.11 is een overzicht gegeven van de budget- en doorlooptijdoverschrijdingen.

Tabel 2.11 : Overschrijdingen op het begrote budget en doorlooptijd.

MATE VAN OVERSCHRIJDING	BUDGET		DOORLOOPTIJD	
	alle projecten %	zeer grote projecten %	alle projecten %	zeer grote projecten %
geen	30	17	20	16
minder dan 10%	41	28	32	20
tussen de 10 en 50%	21	25	38	33
tussen de 50 en 100%	6	19	8	12
meer dan 100%	1	11	2	19

Uit de antwoorden komt naar voren dat de overschrijdingen minder groot zijn dan in de literatuur wordt gesuggereerd (Martin en McClure 1983, Zelkowitz e.a. 1979).
Een nadere analyse van de antwoorden leert evenwel dat er een significant verschil is tussen kleine en zeer grote projecten wat betreft het overschrijden van kosten en doorlooptijd. Uit tabel 2.11 blijkt dat de relatieve overschrijdingen bij zeer grote projecten (meer dan 200 mensmaanden) groter zijn.De cijfers die in de literatuur genoemd worden hebben veelal betrekking op dergelijke grote projecten. Een en ander betekent dat alleen een vergelijking mogelijk is tussen de cijfers van grote projecten uit dit empirisch onderzoek met de cijfers uit de literatuur. Wordt dit gedaan dan blijken de onderzoeksresultaten en de literatuur minder ver uit elkaar te liggen. Bij kleine en middelgrote projecten worden in 37% van de gevallen geen budget- en doorlooptijden overschreden en in 42% van de gevallen minder dan 10%. Blijkbaar zijn organisaties in staat voor projecten van dit soort omvang redelijk nauwkeurige begrotingen op te stellen en zich te houden aan een afgegeven prijs en een tijdstip van levering.

2.6. ERVARINGEN

De laatste categorie vragen hadden geen betrekking op feitelijkheden uit de betreffende organisaties, maar vragen naar de mening van de functionarissen die de vragenlijst hebben ingevuld over een aantal moeilijk meetbare zaken. Zo zijn de volgende twee vragen gesteld:

- Welke vijf factoren hebben naar uw mening de grootste invloed op de kosten van software-ontwikkeling ?
- Welke maatregelen zullen naar uw mening het grootste effect hebben op een verbetering van het beheersen c.q begroten van software-ontwikkeling ?

In tabel 2.12 zijn de antwoorden op de eerste vraag in kaart gebracht.

Tabel 2.12 : Overzicht van de kostenbepalende factoren, in volgorde van dominantie. Bij het invullen van deze vraag waren vijf antwoordmogelijkheden.

GENOEMDE KOSTENBEPALENDE FACTOREN	aantal	%
omvang van de te ontwikkelen software	328	55
complexiteit van de te ontwikkelen software	288	48
wijzigen specificaties tijdens ontwikkelen van de software	250	41
kwaliteit personeel	230	38
kwaliteitseisen gesteld aan de te ontwikkelen software	210	35
mate van gebruikersparticipatie bij de ontwikkeling	174	29
kwaliteit van het management	154	26
scholing en opleiding van de gebruikers	141	24
ervaring van het personeel	126	21
vereiste documentatie	103	17
prestatie-eisen gesteld aan de nieuwe aan te schaffen hardware	84	14
beperkingen van de aanwezige hardware	80	13
gebruik van ontwikkelgereedschap (tools)	73	12
aantal verschillende gebruikers	67	11
eisen gesteld aan de doorlooptijd van het project	49	8
gebruik van moderne programmeertechnieken	39	7
verloop van het personeel	22	4
mate van hergebruik	22	4

De responderende organisaties konden voor het beantwoorden van deze vraag kiezen uit een lijst van 20 factoren. Hierbij is gebruik gemaakt van een lijst van factoren, die in hoofdstuk 4 nader wordt toegelicht. Welke factoren de grootste invloed hebben op kosten en doorlooptijd is in belangrijke mate afhankelijk van de omgeving waarin de software wordt ontwikkeld. In de lijst zijn die factoren opgenomen die in de meeste onderzoeken, beschreven in de literatuur, als de meest dominante worden beschouwd (Heemstra 1987 en 1988a).

Uit een nadere analyse van de antwoorden kwam naar voren dat er een significant verband is tussen de grootte van de organisatie en de

vijf factoren die genoemd worden. Zo blijkt dat, naarmate de omvang van een organisatie toeneemt, de factor "kwaliteitseisen die gesteld worden aan de te ontwikkelen software" in betekenis afneemt. Verder blijkt de betekenis van de factor "kwaliteit van het personeel" toe te nemen bij een toename van de omvang van de organisatie. Voor deze verbanden heb ik geen verklaring kunnen vinden. In de literatuur wordt van dergelijke afhankelijkheden geen melding gemaakt.

Wat betreft de ideeën over mogelijke verbeteringsmaatregelen voor kostenbeheersing c.q begroting komt naar voren dat met name in de relatie tussen ontwikkelteam en opdrachtgever/klant verbeteringen gezocht moeten worden.
In tabel 2.13 wordt een overzicht gegeven van de geopperde verbeteringsmaatregelen.

Tabel 2.13 : Overzicht van geopperde verbeteringsvoorstellen. Bij deze vraag waren twee antwoordmogelijkheden.

MOGELIJKE VERBETERINGSVOORSTELLEN	aantal	%
meer discipline in de relatie met klant/opdrachtgever	208	21
meer betrokkenheid van het management	150	15
uitvoerige registratie/bewaking van projecten	132	13
expliciete toekenning van verantwoordelijkheden en bevoegdheden aan projectmedewerkers	123	12
gebruik van vierde generatiehulpmiddelen	108	11
beter personeel	68	7
gebruik van begrotingsmodellen	58	6
hergebruik van bestaande code, ontwerpen e.d.	45	5
meer onderzoek op dit gebied	36	4
gebruik van expertsystemen en KI-technieken	15	2
overige maatregelen	41	4

Uit de antwoorden komt duidelijk naar voren dat volgens de responderende functionarissen verbeteringen vooral gezocht moeten worden op organisatorisch gebied: zoals betere afspraken met de klant, grotere betrokkenheid van het management en meer verantwoordelijkheden leggen bij projectmedewerkers. Het is opmerkelijk dat het vastleggen van gegevens over projecten als een belangrijke verbetering wordt gezien, maar dat aan de andere kant in slechts 50% van de organisaties zo'n registratie plaatsvindt. Uit de analyse van de antwoorden blijkt verder dat men een grotere betekenis toekent aan registreren naarmate de omvang van het project toeneemt.

2.7. TOETSINGEN

In paragraaf 2.3 zijn een zestal hypothesen geformuleerd die in dit onderzoek getoetst zijn. In elke hypothese wordt een verband verondersteld tussen een bepaalde grootheid en budget- en levertijdoverschrijdingen. Voor de zes hypothesen zijn deze grootheden successievelijk:

- gebruik van de methode Price-To-Win of de "capaciteitsmethode"
- begrotingsfrequentie
- detailleringsnivo van de begroting
- moment van de eerste begroting
- gebruik begrotingsmodellen
- gebruik van vierde generatiehulpmiddelen

Met behulp van regressie analyse is onderzocht of deze verbanden significant zijn. Op de resultaten van de regressie zijn toetsen uitgevoerd met een betrouwbaarheid van 95%.

Alvorens in te gaan op de resultaten van de regressie-analyse een opmerking vooraf. Uit het beschrijvende gedeelte van het onderzoek kwam het beeld naar voren dat het merendeel van de responderende organisaties niet in staat geacht moet worden een betrouwbare uitspraak te doen over overschrijdingen van budgetten en doorlooptijden.

Immers, 35% van de responderende organisaties geeft te kennen geen begroting op te stellen, 50% registreert niets van een project en 57% calculeert niet na.

Bij het toetsen van de zes hypothesen is het mijns inziens niet juist deze categorie respondenten te betrekken. Vandaar dat voor het toetsen van de hypothesen alleen maar gebruik gemaakt is van de antwoorden van die responderende organisaties die wel inzicht kunnen hebben in hun overschrijdingen en hierover uitspraken kunnen doen. Om deze groep respondenten te selecteren, is uitgegaan van de veronderstelling dat een organisatie aan de volgende voorwaarden moet voldoen:

- er wordt in de betreffende organisatie een begroting opgesteld,
- er vindt registratie plaats van minimaal kosten en doorlooptijd en van de oorzaken van eventuele afwijkingen tussen begroting en werkelijk bestede middelen,
- in de betreffende organisatie wordt nagecalculeerd.

Van organisaties die voldoen aan deze voorwaarden, wordt aangenomen dat ze zich serieus bezighouden met het begroten van software-ontwikkeling en in staat zijn betrouwbare uitspraken te doen over overschrijdingen van budgetten en doorlooptijden. Na het uitvoeren van de selectie bleek dat 160 organisaties (26,5% van de responderende organisaties) aan de drie genoemde voorwaarden voldeden.

Overschrijdingen en price-to-win of de "capaciteitsmethode"

Organisaties die begroten beschouwen als een capaciteitsvraagstuk of zich bij het opstellen van een begroting sterk laten leiden door commerciële motieven, hebben geen grotere overschrijdingen van budgetten en/of levertijden dan organisaties die gebruik maken van andere begrotingsmethoden. Een verklaring zou kunnen zijn dat bij gebruik van Price-To-Win of de "capaciteitsmethode" de kwaliteit van de afgeleverde software geringer is. Kosten en overschrijdingen dienen immers ook afgemeten te worden aan de kwaliteit van de opgeleverde software. In het onderzoek is daarom eveneens getoetst of er bij organisaties die genoemde begrotingsmethoden toepassen een groter deel van de totale ontwikkelinspanning gaat zitten in onderhoud dan in organisaties die niet gebruik maken van deze budgetmethode of begroten zien als een capaciteitsvraagstuk. Dit bleek niet het geval te zijn. Concluderend kan men stellen dat het begroten van software-ontwikkeling nog in de kinderschoenen staat als het wat betreft de mate van kostenoverschrijdingen "niets uitmaakt" of men wel of niet gebruik maakt van de methode Price-To-Win of de "capaciteitsmethode".
In paragraaf 3.2 wordt nader ingegaan op de methode Price-To-Win en de "capaciteitsmethode".

Overschrijdingen en begrotingsfrequentie

Uit het onderzoek blijkt dat er een significant verband bestaat tussen het aantal malen dat begroot wordt en de hoogte van de budgetoverschrijdingen. Wel tenderen de uitkomsten ernaar dat de overschrijdingen toenemen naarmate er vaker voor een project een begroting wordt opgesteld. Een verklaring hiervoor kan zijn dat voor grotere projecten meerdere malen een begroting moet worden opgesteld omdat bij grotere projecten ook grotere budgetoverschrijdingen dreigen. Om deze verklaring te verifiëren, is het verband tussen begrotingsfrequentie en budgetoverschrijding onderzocht, rekening houdend met de projectomvang. Van de 160 geselecteerde organisaties

zijn er 20 die zeer grote projecten uitvoeren, 56 die grote projecten uitvoeren, 100 met middelgrote en 112 met kleine projecten. Voor de duidelijkheid merken we op dat een organisatie projecten kan uitvoeren in een of meerdere omvangsklassen. Per omvangsklasse is vervolgens onderzocht of er een significant verband bestaat tussen het aantal malen dat begroot wordt en de hoogte van de budgetoverschrijdingen. Dit bleek *alleen* het geval te zijn voor de klasse grote projecten (tussen de 49 en 200 mensmaanden). Een verklaring voor deze samenhang, kon op basis van de enquetegegevens niet worden gegeven.

Overschrijdingen en detailleringsnivo van de begroting

Er blijkt een significant verband te bestaan tussen de mate van detaillering van de begroting en de hoogte van de budget- overschrijdingen. Hoe gedetailleerder een begroting wordt opgesteld, des te groter de overschrijdingen op het budget zijn. Door rekening te houden met de projectomvang kon worden aangetoond dat dit verband alleen geldt voor zeer grote projecten. Een verklaring voor dit verband is niet gevonden. De nul-hypothese was zelfs dat naarmate de detaillering toenam, budgetoverschrijdingen zouden afnemen.

Overschrijdingen en moment van begroten

Er blijkt geen verband te bestaan tussen de hoogte van overschrijdingen en het moment waarop een organisatie voor de eerste keer een begroting opstelt. Dit betekent dat met behulp van deze enqueteresultaten niet kan worden aangetoond dat organisaties die pas na het ontwerp voor de eerste keer een begroting opstellen lagere budget- en doorlooptijdoverschrijdingen hebben, dan organisaties die al op basis van een globale omschrijving voor de eerste keer begroten.

Overschrijdingen en begrotingsmodellen

Na toetsing is gebleken dat er geen significante afhankelijkheid bestaat tussen het gebruik van een begrotingsmodel en de hoogte van de budget- en levertijdoverschrijdingen. Voor modellenmakers is dit een nogal teleurstellend resultaat. Een verklaring voor het gevonden toetsresultaat is niet gevonden. Wellicht dat het toetsresultaat aangeeft dat de beschikbare modellen maar een zeer beperkte gebruikswaarde hebben.
Een mogelijke verklaring kan zijn dat modellen door sommige organi-

saties onzorgvuldig worden gebruikt en/of deze organisaties software-ontwikkeling onzorgvuldig begroten en beheersen. Zo bleek bijvoorbeeld dat van de 45 organisaties, die gebruik maken van het model Funktie Punt Analyse, er 2 zijn die niets registreren, er 12 zijn die niet nacalculeren en er 3 zijn die het toepassen voor situaties waarvoor het niet geschikt is.

Overschrijdingen en vierde generatiehulpmiddelen

In de literatuur komt naar voren dat het gebruik van vierde generatiehulpmiddelen (in de toekomst) naar alle waarschijnlijkheid een grote invloed zal hebben op de produktiviteit van software-ontwikkelaars en als gevolg daarvan op de hoogte van de kosten van software-ontwikkeling (Boehm 1988, Jones 1984, Howard 1988, van der Ven 1988, Martin 1985). Met behulp van vragen als: in welke mate maakt u gebruik van vierde generatiehulpmiddelen en wat is naar uw mening het belangrijkste effect ervan, werd gepoogd inzicht te verkrijgen in het gebruik en vermeende effect ervan. Door de antwoorden te correleren met de antwoorden over budget- en doorlooptijd overschrijdingen werd getoetst of er een afhankelijkheid bestaat.

Uit het onderzoek bleek allereerst dat vierde generatiehulpmiddelen slechts op beperkte schaal worden gebruikt. 61% van de responderende organisaties geeft aan dat zij bij geen enkel project van dergelijke hulpmiddelen gebruik maakt. Slechts 17% gebruikt ze in meer dan de helft van de projecten. Evenals het gebruik van begrotingsmodellen blijkt de toepassing van vierde generatiehulpmiddelen toe te nemen bij een toename van de omvang van de organisatie. Van de 160 geselecteerde organisaties gebruiken er 64 geen vierde generatiehulpmiddelen (40%), 54 (=34%) organisaties gebruiken ze in minder dan de helft van de projecten en 36 (=22,5%) organisaties in meer dan de helft van de projecten. 6 organisaties hebben de betreffende vraag niet (juist) beantwoord.

Op de vraag "Wat is naar uw mening het belangrijkste effect van vierde generatiehulpmiddelen" antwoordt 46% van de respondenten: verkorting van de doorlooptijd, 30% verlaging van de ontwikkelkosten en 18% verlaging van de onderhoudskosten. Hierbij maakt het geen verschil uit in welke mate de betreffende organisatie gebruik maakt van dergelijke hulpmiddelen.

Bij het toetsen van de hypothese is de volgende vooronderstelling gemaakt: een organisatie maakt *veel* gebruik van vierde generatiehulpmiddelen als in meer dan de helft van de projecten hiervan gebruik

wordt gemaakt. Zoals hierboven aangegeven zijn dat 36 organisaties. De hypothese zelf luidde: hoe meer een organisatie gebruik maakt van vierde generatiehulpmiddelen, hoe lager de budgetoverschrijdingen zullen zijn. Na toetsing blijkt dat de hypothese verworpen moet worden. Er blijkt geen significant verband te bestaan tussen de mate waarin organisaties gebruikt maken van vierde generatiehulpmiddelen en de hoogte van budget- en levertijdoverschrijdingen.

2.8. CONCLUSIES

De belangrijkste conclusies uit dit onderzoek zijn:

- 35% van de responderende organisaties geeft te kennen dat in hun organisatie voor softwareprojecten geen begroting wordt opgesteld. Voor projecten die meer dan 200 mensmaanden kosten wordt evenwel in 100% van de gevallen een begroting opgesteld,

- 65% van de responderende organisaties zegt wel te begroten, 62% baseert de begroting mede op intuïtie en ervaring,

- 50% van de organisaties die gereageerd hebben, registreren niets van een project in uitvoering,

- 57% van de responderende organisaties geeft te kennen niet na te calculeren. Dit betekent dat meer dan de helft van de organisaties geen idee heeft van de bestede middelen. Mochten zij al een begroting hebben gemaakt, dan kunnen zij geen betrouwbare uitspraken doen in hoeverre er al dan niet een afwijking van de begroting heeft plaatsgevonden,

- er blijkt een duidelijk verschil te bestaan tussen kleine en grote projecten wat betreft het overschrijden van budgetten en doorlooptijden. Bij zeer grote projecten ligt de gemiddelde overschrijding rond de 50%. Dit blijkt significant hoger te liggen dan bij kleine projecten,

- als er een begroting wordt opgesteld, dan blijkt deze veelal niet gedifferentieerd te zijn naar de verschillende fasen c.q. activiteiten van een project. Daarnaast stelt het merendeel van de organisaties slechts éénmaal een begroting op voor een project. Voor organisaties die zich hoofdzakelijk richten op de uitvoering van zeer grote

projecten (meer dan 200 mensmaanden) blijkt dit significant anders te liggen. Deze organisaties stellen voor meer dan de helft van dergelijk grote projecten meer dan drie keer tijdens zo'n project een begroting op. Voor 80% van deze projecten splitsen zij de begroting uit naar fasen en activiteiten,

- een begroting wordt primair opgesteld door het management en/of de projectleider. Projectmedewerkers en opdrachtgever worden slechts sporadisch hierbij betrokken,

- in de gevallen dat er een begroting wordt opgesteld, blijkt deze zich vooral te richten op de geschatte besteding van geld, tijd en personeel. Over een aantal eveneens belangrijke onderwerpen, zoals het verwachte percentage hergebruik, de benodigde middelen in termen van software, hardware e.d wordt beduidend minder een uitspraak gedaan in een begroting,

- er blijken 65 organisaties te zijn die zeggen te begroten op basis van gegevens van voltooide projecten maar daarnaast eveneens te kennen geven niets te registreren van projecten,

- slechts 16% van degenen die begroten, maken gebruik van een begrotingsmodel. 75% van de modelgebruikers combineren het gebruik van een model met de analogiemethode (bij een begroting gebruik maken van project- c.q. projectgegevens van soortgelijke projecten uit het verleden). 15% van de modelgebruikers gebruikt naast een model de expertmethode. Uit de enquete kan niet opgemaakt worden op welke wijze zo'n model gebruikt wordt. Uit een verder analyse van de antwoorden kan wel opgemaakt worden dat van de 51 modelgebruikers, er 18 zijn die niet nacalculeren. Een dergelijk cijfer zegt iets over de wijze waarop men een begroting hanteert in deze organisaties: wél een begrotingsmodel gebruiken maar niet nacalculeren, betekent dat het model niet anders dan intuïtief wordt gevalideerd.

Het bovenstaande geeft geen rooskleurig beeld van de stand van zaken over het begroten van software-ontwikkelprojecten in Nederland. De resultaten blijken een bevestiging te zijn van de wat negatief getinte uitspraken die over dit onderwerp in de literatuur worden gedaan en rechtvaardigen nader onderzoek over dit onderwerp.

3. BEGROTINGSMETHODEN

3.1. INLEIDING

Het is inmiddels meer dan 10 jaar geleden dat Wolverton in een baanbrekend artikel (1974) een aanzet gaf voor een methode om de kosten voor het vervaardigen van software te schatten en Walston en Felix (1977) pionierswerk verrichtten met hun studie over het meten van de produktiviteit bij software-ontwikkeling. Het aantal en de aard van de onderzoeken die hieraan zijn voorafgegaan, is zeer beperkt (onder andere Nelson 1966, Norden 1963). Ondanks een groeiende aantal publikaties in de laatste jaren over dit soort onderwerpen, staat het begroten van software-ontwikkeling nog in de "kinderschoenen".

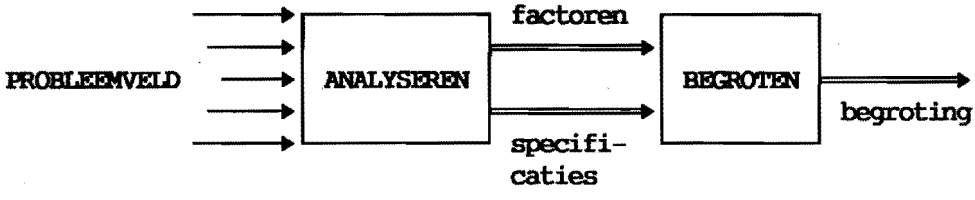
Het beheersen en begroten van software-ontwikkeling levert veel problemen op. In hoofdstuk 2 is dit nog eens bevestigd. Budgetten worden veelal overschreden en plannen moeten vaak worden bijgesteld. Als gevolg hiervan wordt de druk om het ontwikkelproces beter te beheersen wat betreft benodigde tijd, geld, mankracht en hulpmiddelen steeds groter. Zo zijn door deze druk de laatste jaren diverse methoden en modellen ontwikkeld voor het begroten van softwareprojecten. In dit hoofdstuk wordt een overzicht gegeven van de meest voorkomende methoden van begroten.

Wat zal een begrotingsmethode de projectmanager moeten bieden? Is een uitkomst in termen van aantal mensmaanden voldoende of dient er ook een prognose gemaakt te worden van een aantal andere zaken? Als men de activiteit begroten op een eenvoudige wijze weergeeft, zal een projectleider een antwoord willen hebben op vragen als:

1. hoeveel gaat het programma kosten,
2. hoeveel tijd is er nodig om de software te maken,
3. hoeveel mensen zijn daarvoor nodig en om welke mensen gaat het (welke ervaring, welke opleiding enz.),

4. welke middelen (apparatuur, programmatuur) zijn er nodig,
5. hoe zal de verdeling zijn van geld, mensen, tijd en middelen over de verschillende fasen van het project,

Om dergelijke vragen te kunnen beantwoorden zal men inzicht moeten hebben in de software die ontwikkeld moet worden en de faktoren die de kosten beïnvloeden. Figuur 3.1 geeft daarvan een sterk vereenvoudigde weergave.



Figuur 3.1 : Een eenvoudige weergave van het begroten van software.

Zo zal bijvoorbeeld nagegaan moeten worden wat naar verwachting de omvang van de software zal zijn. Ook zal men een schatting moeten maken van de complexiteit van de software, van de mate van gebruikersparticipatie, van het effect van te gebruiken vierde generatiehulpmiddelen, van de hoeveelheid documentatie en ga zo maar door. De antwoorden op dergelijk soort vragen vormen de invoer voor de uitvoering van een begroting. Hoofdstuk 5 is volledig gewijd aan de faktoren die de softwarekosten bepalen. Welke vragen beantwoord moeten worden, of anders geformuleerd, welke faktoren relevant zijn, kan van situatie tot situatie verschillen. Bij de ontwikkeling van software voor de besturing van een compact-disc speler spelen andere zaken een rol dan bij de ontwikkeling van een factureringsprogramma. Beide projecten verschillen qua type, omvang, complexiteit en dergelijke. Bij het ieder project zullen specifieke kostenbepalende factoren een rol spelen. Zo zal men bij de compact-disc speler moeten zorgen dat de ontwikkeling van de software gelijke tred houdt met de ontwikkeling van de hardware. Dergelijke overwegingen spelen niet bij het begroten van een factureringsprogramma. Welke faktoren relevant zijn en wat de waarde van de verschillende faktoren is kan alleen achterhaald worden door het analyseren van de probleemsituatie en door na te gaan wat de eisen zijn die aan de software worden gesteld. Dit betekent dat een uitgebreide analyse aan de activiteit begroten vooraf dient te gaan.

Het bovenstaande geeft een idee van vragen die bij en voorafgaande aan het begroten beantwoord moeten worden. De reeks vragen is niet uitputtend, maar geeft wel een beeld van de complexiteit van het begroten van een project.

In paragraaf 3.2 worden de belangrijkste begrotingsmethoden beschreven. Op een van deze methoden, namelijk parametrische modellen, wordt in 3.2.5 dieper ingegaan. In paragraaf 3.3 wordt een aantal manieren aangegeven om begrotingsmodellen te typeren. Omdat het aanbod van begrotingsmodellen de laatste jaren sterk toeneemt, wordt de vraag actueel welk model in een bepaalde situatie de voorkeur geniet. In paragraaf 3.4 wordt een methode beschreven voor het beoordelen van begrotingsmodellen. Het hoofdstuk wordt afgesloten met conclusies (paragraaf 3.5).

3.2. BEGROTINGSMETHODEN

In de literatuur treft men een groot aantal methoden aan voor het begroten van software-ontwikkeling. Over het algemeen zijn deze terug te voeren tot een van de volgende vijf basismethoden of een combinatie daarvan:

1. Beoordeling door een expert,
2. Begroten op basis van analogieën,
3. De methode Price-To-Win,
4. De "capaciteitsmethode",
5. De methode gebaseerd op parametrische modellen.

In de paragrafen 3.2.1 tot en met 3.2.5 zullen deze methode successievelijke beschreven worden.

3.2.1. BEOORDELING DOOR EEN EXPERT

Bij deze basismethode gaat men af op het advies van een expert of meerdere experts. Een expert baseert zich bij het begroten van het project op hun ervaring en inzicht in het op stapel staande project. De kwaliteit van de afgegeven schattingen is ondermeer afhankelijk van de mate waarin het nieuwe project aansluit bij de ervaring van

de expert en van het vermogen van de expert om zich relevante zaken van oude projecten te herinneren. In de meeste gevallen zijn de schattingen kwalitatief van aard en veelal niet objectiveerbaar. De enqueteresultaten in hoofdstuk 2 laten zien dat deze methode in de praktijk veel wordt toegepast. 62 % van de responderende organisaties geeft te kennen zich bij het opstellen van een begroting mede te baseren op intuïtie en ervaring. Men gaat in dergelijke gevallen met name af op eigen expertise. Ook haalt men expertise herhaaldelijk van elders. 26 % van de organisaties zegt namelijk dat zij bij een begroting een expert inschakelen.

Het belangrijke probleem bij de expertmethode is dat het voor iemand anders lastig is de ervaring en kennis van de expert te reproduceren en te gebruiken. Het gevaar bestaat dat vuistregels die een expert hanteert binnen een organisatie de status krijgen van algemeen geldende regels en vervolgens toegepast worden in situaties en omgevingen waar deze niet toepasbaar zijn. Deze bezwaren zouden voor een goed deel weggenomen kunnen worden als de expert zijn kennis en ervaringen over oude projecten meer zou kunnen objectiveren en vastleggen. De "gemiddelde" ervaring van een expert is overigens relatief gering als men ervan uitgaat dat projecten een doorlooptijd hebben van een tot enkele jaren zodat een expert zijn ervaring slechts kan opbouwen aan de hand van een beperkt aantal projecten. Door de toenemende verscheidenheid aan projekten wordt de kans dat de ervaring van een expert niet representatief genoeg is voor een nieuw projekt steeds groter. Daarbij komt dat door de steeds sneller voortschrijdende technologie en veranderingen in de methodologie van software-ontwikkeling het voor de expert alsmaar moeilijker wordt de nodige expertise op te bouwen ten aanzien van nieuwe methoden, technieken en gereedschappen.

Van de expertmethode zijn allerlei varianten in omloop. Zo beschrijft Londeix (1987) een procedure om via brainstorming van een aantal experts tot een begroting te komen. Een Delfi-achtige aanpak is een verdere verfijning hiervan. Deze aanpak komt voor een deel tegemoet aan de hierboven genoemde bezwaren. In een aantal rondes komen de deelnemende experts tot één gezamenlijke begroting. De volgende stap kan nu zijn om in een volgende ronde de deelnemers te vragen hoe "hard" deze begroting is, of zij op basis van de opgegeven kosten en doorlooptijd het project zouden uitvoeren. Op deze wijze achterhaald men wat de "hardheid" van een begroting is en welke betekenis men eraan mag toekennen.

Ondanks de nadelen wordt de "Expert-methode" vaak gebruikt in situaties waarin men snel een eerste indicatie van benodigde tijd,

geld, personeel en middelen wil hebben. Met name in de allereerste fase van software-ontwikkeling, waarin de specificaties van het produkt nog vaag zijn en voortdurend bijgesteld moeten worden, is het gebruik van deze snelle en efficiënte methode te verkiezen boven de nog te bespreken parametrische schattingsmodellen, die veelal om een gedetailleerde en kwantitatieve aanpak vragen.

3.2.2. BEGROTEN OP BASIS VAN ANALOGIEEN

Bij het schatten op basis van analogieën baseert de degene die de begroting opstelt zich expliciet op gegevens van vergelijkbare oude projecten of vergelijkbare delen c.q. modules hiervan. In hoofdstuk 2 is aangetoond dat een groot gedeelte van de organisaties die een begroting opstellen mede gebruik maken van deze methode (61 %). Wil men in staat zijn een overeenkomst te vinden tussen het nieuwe project en een of meerdere oude projecten dan is een registratie en klassificatie van oude projecten noodzakelijk. Veelal ontbreekt een dergelijk registratie- en klassificatiesysteem en is kennis van een project of delen ervan ongestruktureerd aanwezig in de hoofden van enkelen. Voor anderen, die geconfronteerd worden met schattingsproblemen is deze kennis vrijwel niet toegankelijk. Het is moeilijk aan te geven in hoeverre een oud project representatief is voor het nieuwe project en wat het effect is van bepaalde verschillen. Door het onderhouden en raadplegen van een (geautomatiseerd) registratiesysteem van voltooide projecten wordt een deel van deze moeilijkheden weggenomen; er ontstaat een vollediger beeld van het verleden. Wat moeilijk blijft, is het klassificeren van een nieuw project. Op basis van deze klassificatie wordt er immers gezocht naar overeenkomsten.

3.2.3. DE METHODE PRICE-TO-WIN

De methode Price-To-Win is nauwelijks een begrotingsmethode te noemen. Het zijn voornamelijk commerciële motieven die een projectmanager verleiden tot deze methode. Voorbeelden hiervan zijn te vinden in de literatuur (van Vliet 1987):

> "Over een half jaar is de efficiency beurs. We moeten koste wat het kost ons nieuwe software-produkt daar presenteren".

"Willen we deze opdracht binnenhalen dan moeten we de prijs laten zakken beneden de fl. 2.000.000. Dan zitten we zeker beneden de prijs van onze concurrenten".

Deze methode wordt echter toch gebruikt, ondanks het feit dat de afgegeven budgetten en levertermijnen vaak weinig realistisch zijn en het gevaar voor overschrijding levensgroot aanwezig is. De resultaten van de enquete in hoofdstuk 2 verbazen in dit kader. Organisaties die te kennen geven, zich mede te laten leiden door commerciële motieven, schijnen of lijken niet slechter te "scoren" dan organisaties die met behulp van een andere methode begroten.

3.2.4. DE "CAPACITEITSMETHODE"

Bij deze methode wordt het begroten beschouwd als een capaciteitsvraagstuk. Het uitgangspunt is dat de beschikbare middelen de beperkende factor vormen. De doelstelling van het project is om met de beschikbaar gestelde middelen het maximaal mogelijke resultaat te bereiken. Een voorbeeld van deze aanpak is de volgende:

"Gezien onze capaciteitsplanning kunnen we de komende acht maanden voor de uitvoering van dit project drie mensen vrij maken. De werktijd van het project bedraagt dus 24 mensmaanden".

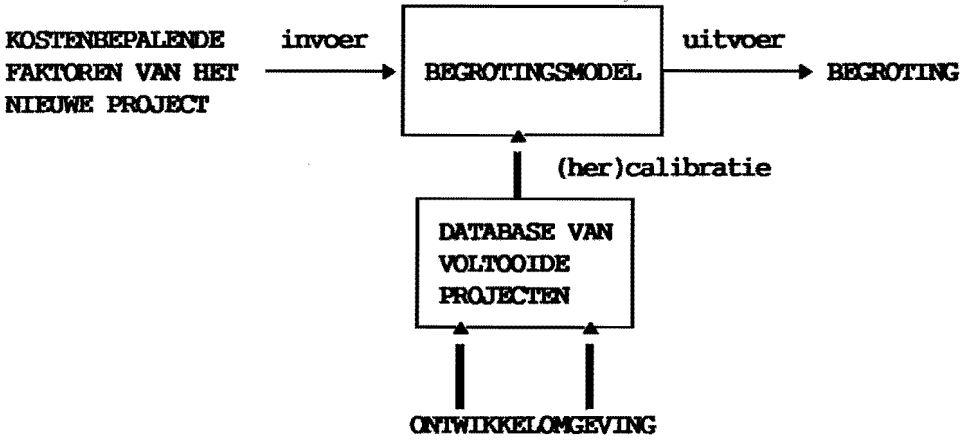
Met name bij softwareprojecten waarbij het niet voldoende duidelijk is hoe het eindresultaat er uit zal gaan zien, kan deze methode gekozen worden. Een ongewenst uitvloeisel van deze methode is, dat ook in situaties, waarin achteraf beschouwd, de beschikbaar gestelde middelen te ruim blijken te zijn, deze toch volledig gebruikt worden. In dergelijke situaties uit zich dat in het verfraaien van de software met (niet gevraagde) "toeters en bellen". Dit verschijnsel is gebaseerd op een van de wetten van Parkinson. Deze wet luidt: *"Work expands to fill the available volume"*.

Uit de enquete (zie hoofdstuk 2) blijkt dat de methode bij gebruik "succesvoller" is dan men op voorhand verwacht. De enqueteresultaten laten zien dat 14 % van de organisaties zich bij het opstellen van een begroting laat leiden door deze aanpak. In veel gevallen wordt een prognose gebaseerd op beschikbare capaciteit door het projectteam gezamenlijk gedaan. Men is op deze manier verzekerd van het commitment van de teamleden. Juist zo'n commitment blijkt een zeer

belangrijke drijfveer te zijn om zich te houden aan de afgegeven prognoses. Ik kom daar later meer uitvoerig op terug.

3.2.5. DE METHODE GEBASEERD OP PARAMETRISCHE MODELLEN.

Bij gebruik van parametrische modellen wordt de schatting van benodigde tijd, geld en middelen weergegeven als een funktie van een aantal variabelen. Deze variabelen representeren de belangrijkste kostenbepalende factoren (cost drivers). In figuur 3.2 is het begroten met behulp van een model in beeld gebracht.



Figuur 3.2 : Een begrotingsmodel in relatie met de omgeving.

De invoer van een model bestaat uit waarden van kostenbepalende factoren. In hoofdstuk 5 zal worden aangegeven dat aantal en aard van deze factoren per model verschillen. De uitvoer van een model is uiteraard een begroting. De kern van een begrotingsmodel bestaat uit een aantal algoritmen en parameters. Hierin komt de relatie tussen de kostenbepalende factoren en de uitvoer tot uitdrukking. De waarden van de parameters en het soort algoritmen zijn mede gebaseerd op een verzameling gegevens van afgesloten softwareprojecten. Voordat een model voor de eerste keer in een bepaalde ontwikkelomgeving toegepast kan worden, dient het eerst gecalibreerd te worden. De

omgeving waarin het model is ontwikkeld en de projectgegevens die ten grondslag liggen aan het model zullen immers in de meeste gevallen verschillen van projectkenmerken van de specifieke ontwikkelomgeving. Om dit calibreren, het op maat maken van modelparameters, mogelijk te maken is het noodzakelijk dat men beschikt over de gegevens van representatieve afgesloten projecten.

De literatuur over begrotingsmodellen is eensluidend in haar oordeel dat calibratie van ongeacht welk model noodzakelijk is. Calibreren is echter niet een eenmalige activiteit, maar moet periodiek herhaald worden. Door technologische en methodologische veranderingen bij software-ontwikkeling en door personele en organisatorische wijzigingen kunnen de kenmerken van een ontwikkelomgeving in de loop der tijd wijzigen. Opnieuw calibreren wordt dan noodzakelijk. Door de introductie van weegfactoren is het mogelijk de invloed van projecten op de calibratie te laten toenemen naarmate het project bijvoorbeeld van recentere datum is.

Bij het gebruik van het merendeel van de parametrische modellen volgt degene die de begroting opstelt een min of meer gelijke aanpak. Dit betekent dat de volgende zeven stappen genomen moeten worden.

stap 1. Calibreer het model,
stap 2. Bepaal de omvang van de software,
stap 3. Vertaal de schatting van de software-omvang in een schatting van de benodigde inspanning,
stap 4. Stel de benodigde inspanning bij door rekening te houden met kostenbepalende factoren,
stap 5. Verdeel de totale inspanning over de verschillend fasen van het project,
stap 6. Bepaal de doorlooptijd
stap 7. Voer gevoeligheid- en risico-analyses uit.

De verschillende stappen zullen in de rest van deze paragraaf worden toegelicht. Evaluatie van toepassingen van modellen leert overigens dat er slechts weinig toepassingen zijn die aan alle stappen voldoende aandacht besteden.

1. *Calibreer het model*

Zoals hiervoor al aangegeven, moet een model gecalibreerd worden, voordat het voor de eerste keer in een bepaalde ontwikkelomgeving gebruikt kan worden. Door Cuelenaere, Heemstra en van Genuchten (1987 en 1987a) wordt de noodzaak van calibratie toegelicht. Verder

is erop gewezen dat calibreren niet een eenmalige activiteit is, maar vaker herhaald moet worden. Gesteld kan worden dat een model zonder calibratie weinig waarde heeft. Onderzoeken van ondermeer Heemstra, van Genuchten, Kusters (1988a), van Kemerer (1987) en Miyazaki en Mori (1985) tonen dit aan. In hoofdstuk 4 wordt hier nader op ingegaan.
Het verzamelen, structureren en vastleggen van voltooide projectgegevens ten behoeve van calibratie zou men als stap 0 kunnen beschouwen.

2. ***Bepaal de omvang van de software***

De volgende stap is het bepalen van de omvang. Veelal wordt deze omvang uitgedrukt in het aantal regels code van de taal waarin het te ontwikkelen programma wordt geschreven of het aantal funtiepunten.

3. ***Vertaal de schatting van de omvang naar een schatting van de benodigde inspanning***

Over het algemeen wordt bij deze omrekening gebruik gemaakt van produktiviteitsmaten. Een voorbeeld hiervan is de vergelijking van het type "inspanning per 1000 instructies". Deze heeft de vorm:

$$\text{INSPANNING} = a * \text{OMVANG}^b \qquad (1)$$

De benodigde INSPANNING wordt daarbij uitgedrukt in het aantal benodigde mensmaanden of geld om het programma te ontwikkelen. De variabelen a, b krijgen een waarde door middel van regressie-analyses op gegevens van voltooide projecten. Naarmate er
-1- meer oude projecten geregistreerd zijn
-2- de gegevens per project vollediger zijn en
-3- de totale projectregistratie meer aansluit bij de ontwikkelomgeving voor het nieuwe programma, zal men beter in staat zijn de waarden van deze variabelen te bepalen.

4. ***Stel de benodigde inspanning bij door rekening te houden met kostenbepalende factoren***

De benodigde inspanning voor een project wordt door een groot aantal factoren bepaald. Door Heemstra (1987) wordt hiervan een overzicht gegeven. Vergelijking (1) wordt als gevolg daarvan als volgt aangepast:

$$\text{INSPANNING} = a * \text{OMVANG}^b * \text{correctiefactor} \qquad (2)$$

waarbij de correctiefactor staat voor:

$$\prod_{i=1}^{i=n} \text{Invloedsfactor(i)} \qquad (3)$$

Invloedfactor (i) geeft de invloed van de kostenbepalende factor i op de ontwikkelkosten weer. Stel bijvoorbeeld dat de eis "kwaliteit van de te ontwikkelen programmatuur" hoog is. Invloedfactor "kwaliteit" krijgt dan de waarde die overeenkomt met de invloed van hoge kwaliteitseisen op de ontwikkelkosten. Door de waarden van de afzonderlijke invloedsfactoren met elkaar te vermenigvuldigen, krijgt men de correctiefactor.
Degene die een begroting opstelt, zal moeten aangeven wat de waarden van de kostenbepalende factoren zijn voor het nieuwe project. Hij zal dus moeten inschatten wat bijvoorbeeld de complexiteit, vereiste kwaliteit, de mate van gebruikersparticipatie e.d. is. Dit is geen gemakkelijke taak, die de nodige expertise vereist en waarbij het beschikken over goede specificaties noodzakelijk is. Het inschatten van genoemde grootheden is moeilijk in die gevallen waarin de te ontwikkelen software zo specifiek is, zo afwijkend is van het gangbare assortiment, dat de verzameling voltooide projectgegevens te weinig aanknopingspunten biedt. Ook kan het zijn dat de wijze waarop de software gemaakt gaat worden anders is dan in het verleden. Men kan zich dan niet alleen baseren op (geregistreerde) ervaringsfeiten. De invloed van genoemde factoren op de kosten worden dan vaak intuïtief bepaald; ervaringsgegevens ontbreken immers.

5. ***Verdeel de totale inspanning over de verschillende fasen van het project***

Het percentage van de totale inspanning dat toegewezen wordt aan de verschillende ontwikkelfasen moet empirisch worden bepaald met behulp van opnieuw oude projectgegevens. Een studie van Thibodeau en Dodson toont aan (1981) dat deze verdeling in belangrijke mate wordt bepaald door de karakteristieken van het softwareproject. In tabel 3.1 wordt de verdeling van de ontwikkelinspanning over de fasen weergegeven. Deze cijfers zijn afkomstig van een aantal grote projecten. Verder zijn in de tabel ook verdelingen opgenomen die door een aantal bekende onderzoekers/bedrijven wordt gehanteerd.
Hoewel de verdelingen onderling verschillen, is er een duidelijke trend waar te nemen die wijst in de richting van de vaak gehanteerde 40-20-40 regel. Dit wil zeggen 40% voor analyse en ontwerp, 20% voor coderen en 40% voor testen.

tabel 3.1 : Verdeling van de inspanning over de fasen in de ontwikkeling van software (Thibodeau en Dodson 1981).

	procentuele verdeling over fasen		
BRON	ANALYSE + ONTWERP	CODEREN	TESTEN
PROJECTEN			
Apollo	31	36	33
DAIS	38	15	47
Gemini	36	17	47
NTDS	30	20	50
OS/360	33	17	50
SAGE	39	14	47
Saturnus V	32	24	44
SETS/BL	42	18	40
Skylab	38	17	45
Titan III	33	28	39
X-15	36	17	47
ONDERZOEKERS/BEDRIJVEN			
Aron	30	20	50
Boehm	34	19	48
Brandon	32	28	40
Brooks	33	17	50
Farr	35	18	47
GRC	30	20	50
Krauss	47	16	37
Raytheon (Business)	44	28	28
RCA	32	21	47
TRW	44	26	30
Wolverton	46	20	34
GEMIDDELD	37	20	43

6. *Bepaal de doorlooptijd*

Behalve het begroten van de benodigde inspanning voor het ontwikkelen van software, zal ook een prognose gemaakt moeten worden van

de tijd om de software te ontwikkelen en een verdeling hiervan over de fasen c.q. activiteiten. Het begroten hiervan zal op een soortgelijke wijze moeten gebeuren als het bepalen van de benodigde inspanning. In een aantal onderzoeken is aangetoond dat de ontwikkeltijd vrij nauwkeurig gerelateerd kan worden aan de inspanning via de volgende vergelijking:

$$\text{DOORLOOPTIJD} = a * \text{INSPANNING}^b \qquad (4)$$

waarbij de waarden van a en b empirisch worden vastgesteld met behulp van alweer gegevens van voltooide projecten. Als INSPANNING wordt weergegeven in mensmaanden en ONTWIKKELTIJD in maanden dan liggen in de meeste gevallen de waarden van a tussen 2 en 4 en van b tussen 0.25 en 0.4 (Birrell en Ould 1985).

In sommige parametrische modellen wordt aangegeven hoe de ontwikkelinspanning verandert als men gaat afwijken van de nominale waarde voor de doorlooptijd, zoals die in (4) is bepaald. Zo gaat het model van Putnam (1978) ervan uit dat een geringe verkorting van de doorlooptijd een meer dan proportionele stijging van de inspanning tot gevolg heeft. Andere modellen zoals COCOMO van Boehm (1981) gaan ervan uit dat de waarde van de doorlooptijd voor een project niet beneden een zekere ondergrens kan komen, ongeacht de hoeveelheid extra inspanning die men aan het project toevoegt. Deze extra inspanning gaat verloren in overhead, met name in toenemende communicatie tussen het stijgend aantal leden van het projectteam. De verdeling van de doorlooptijd over de verschillende fasen c.q. activiteiten zal wederom moeten gebeuren op basis van afgesloten projectgegevens.

7. *Voer gevoeligheids- en risico-analyses uit*

Slechts weinig modellen ondersteunen de mogelijkheid voor het uitvoeren van *gevoeligheidsanalyses*. Uitzondering hierop zijn ondermeer de modellen Estimacs (Computer Associates 1986) en Before You Leap (1986). Door het met kleine stapjes laten wijzigen van de waarde van een (of meerdere) kostenbepalende factor(en) kan men nagaan hoe gevoelig een begroting is voor bepaalde invloeden. Bijvoorbeeld: wat is het effect van het verkorten van de doorlooptijd met 2 maanden, het wijzigen van de responsiesnelheid van 5 naar 2 seconden, enz.

Een belangrijke stap bij het gebruik van een begrotingsmodel behoort de *risico-analyse* te zijn. Het projectmanagement dient tijdens het ontwikkelingstraject op de hoogte te zijn van de risico's en onzekerheden van het project. Slechts weinig modellen bieden deze mogelijk-

heid. Een uitzondering hierop is het model Estimacs. In dit model is een uitvoerige risicomodule opgenomen.

3.3. EEN ORDENING VAN BEGROTINGSMODELLEN

Op dit moment kan de gebruiker kiezen uit een groot aantal begrotingsmodellen. In hoofdstuk 4 wordt hiervan een overzicht gegeven. In deze paragraaf geef ik een manier om begrotingsmodellen te ordenen. Nadat de modellen in de paragrafen 4.2 tot en met 4.8 beschreven en geëvalueerd zijn, worden ze in paragraaf 4.11 met behulp van de hier beschreven aanpak geordend. Bij de ordening heb ik gebruik gemaakt van de volgende vijf invalshoeken:

1. ***Modellen gebaseerd op aantal regels code of modellen niet gebaseerd op aantal regels code.***
   Toelichting:
   Een groot probleem bij het begroten is het bepalen van de omvang van de omvang van de software. Dit geldt des te sterker naarmate in een vroeger stadium van software-ontwikkeling een begroting gemaakt moet worden. Van Vliet (1987a) trekt terecht de parallel met het schrijven van een roman. Hoe is de auteur op voorhand in staat de omvang ervan te bepalen als hij nog geen duidelijk zicht heeft op het aantal personen, aantal lokaties en de tijdspanne waarin het verhaal zich afspeelt. Bij het bepalen van de omvang gaat het merendeel van de modellen uit van het schatten van het aantal regels code. Er is echter een trend waar te nemen dat steeds meer modellen een andere maatstaf hanteren voor de omvang. Zo is de maatstaf die de omvang uitdrukt in aantal funktiepunten, sterk in opmars. Hierbij wordt uitgegaan van de functionaliteit van de te ontwikkelen software. Deze is programmeertaalonafhankelijk.

2. ***Modellen die primair gericht zijn op het bepalen van de omvang versus modellen die zich richten op het bepalen van de produktiviteit.***
   Toelichting:
   Zoals in paragraaf 3.2.5 aangegeven is een van de eerste stappen in een begrotingsmodel het bepalen van de omvang. Het resultaat kan uitgedrukt worden in aantal regels code, aantal statements, aantal funktiepunten, aantal functionele primitieven enz. Omdat "de ene regel code moeilijker is dan de andere, de ene ontwikkelaar meer ervaring heeft dan de andere, met het ene hulpmiddel

sneller ontwikkeld kan worden dan met een ander hulpmiddel, enz.", moet de waarde voor de omvang, zoals gebruikt in een model, gecorrigeerd worden voor dit soort productiviteitsafhankelijke factoren. Modellen die zich in hoofdzaak richten op het schatten van de omvang zijn vaak zwak in het bepalen van productiviteitsafhankelijke factoren. Modellen die zich primair richten op het bepalen van de produktiviteit bieden gewoonlijk de begroter weinig houvast als het gaat om het bepalen van de omvang.

3. ***Modellen gebaseerd op parameters versus modellen gebaseerd op expertkennis versus modellen gebaseerd op analogieën.***
   Toelichting:
   Het is mogelijk dat ervaring en feiten zijn verwerkt in de parameters van het model, in regels, of in geregistreerde voltooide projecten.
   In de vorige paragrafen is gewezen op het feit dat de basis van begrotingsmodellen dient te bestaan uit gegevens van ***eigen*** voltooide projecten. In de volgende hoofdstukken zal dit nog sterker worden benadrukt. De meest bekende vorm is het condenseren van deze gegevens in algoritmen en parameters van het model. Deze aanpak heeft ook model gestaan in paragraaf 3.2.5. Een andere mogelijkheid is om gegevens, vuistregels en heuristieken vast te leggen in regels. Een derde mogelijkheid is dat het model bestaat uit een zoekmechanisme dat op basis van karakteristieken van de te ontwikkelen software in een databank van voltooide softwareprojecten zoekt naar vergelijkbare projecten, projectdelen, modulen.

4. ***Modellen gebaseerd op een top-down aanpak versus modellen gebaseerd op een bottom-up aanpak.***
   Toelichting:
   Bij een top-down benadering wordt begonnen met een begroting voor het totale systeem. Deze wordt vervolgens verdeeld over de verschillende fasen, activiteiten en modules.
   Bij een bottom-up benadering wordt per fase een begroting gemaakt. Door middel van extrapolatie wordt een begroting voor het totaal gemaakt.

5. ***Modellen met een gedefinieerd toepassingsgebied versus modellen met een niet-gedefinieerd toepassingsgebied.***
   Toelichting:
   Een belangrijk onderscheid in dit kader is het onderscheid in modellen die zich primair richten op het begroten van bestuurlijk informatiesystemen (bijvoorbeeld FPA), modellen die zich richten op embedded software (PRICE-S) en modellen die pretenderen

geschikt te zijn voor het begroten van ongeacht welk type software (Estimacs, Before You Leap).

3.4. EEN METHODE VOOR DE BEOORDELING VAN BEGROTINGSMODELLEN

3.4.1 INLEIDING

Het toenemende belang van software en de problemen die ondervonden worden bij het begroten hebben ertoe geleid dat de laatste jaren een groot aantal begrotingsmodellen, al dan niet geïmplementeerd in een geautomatiseerd hulpmiddel, zijn ontwikkeld. Door het stijgend aanbod van deze modellen wordt voor een organisatie de volgende vraag actueel:

aan welke eisen moet een begrotingsmodel voldoen om een geschikt hulpmiddel te zijn bij het schatten van tijd en kosten voor het ontwikkelen van software.

Om deze vraag te beantwoorden wordt in deze paragraaf een methode gepresenteerd voor het beoordelen van begrotingsmodellen. Deze methode is ontwikkeld door een researchgroep "Begroten van Automatiseringsprojecten" van de Technische Universiteit Eindhoven. In het kader van contractresearch is de methode toegepast om een aantal modellen te beoordelen (Heemstra, van Genuchten en Kusters 1988). In de methode is een groot aantal eisen opgenomen waaraan een begrotingsmodel moet voldoen. Deze eisen zijn het resultaat van literatuuronderzoek (Boehm 1981, Noth en Kretzschmar 1984, Heemstra 1987b) en van praktische ervaringen, opgedaan met begrotingsmodellen. De eisen zijn gegroepeerd in een van de volgende drie hoofdcategorieën:

- ***Modeleisen***
  deze eisen hebben betrekking op de ideeën en concepten die ten grondslag liggen aan het model.

- ***Toepassings-eisen***
  deze eisen hebben betrekking op de geparametriseerde versie van het model. Geparametriseerd wil zeggen dat de parameters van een model door middel van calibratie een waarde hebben gekregen.

Nadat calibratie heeft plaatsgevonden, kan worden nagegaan in hoeverre een model aan dit soort eisen voldoet.

- *Implementatie-eisen*
  deze eisen hebben betrekking op de geïmplementeerde versie van een model. Voor de meeste begrotingsmodellen zijn een of meer geautomatiseerde versies (pakketten) beschikbaar. Dit soort eisen concentreert zich vooral op dergelijke pakketten.

In tabel 3.2 is een overzicht gegeven van de eisen die in de beoordelingsmethode zijn opgenomen. In paragrafen 3.4.2 tot en met 3.4.4 worden deze eisen nader toegelicht.

Tabel 3.2 : Eisen voor de beoordeling van begrotingsmodellen.

EISEN		
model-eisen	toepassings-eisen	implementatie-eisen
- gekoppeld aan beheersingsmethode - vroeg toepasbaar - evoluerend - aanpasbaar aan doelstellingen - toepasbaarheid	- calibratie - nauwkeurigheid	- gebruiksvriendelijkheid - gevoeligheidsanalyse - risico-analyse - inzichtelijkheid - eenduidigheid - volledigheid uitvoer

Uiteraard zullen niet alle eisen in elke situatie even zwaar wegen. Zo zal een organisatie die een model primair wenst te gebruiken in een vroeg stadium van software-ontwikkeling een groot gewicht toekennen aan de eis "vroeg toepasbaar". Aan sommige eisen, zoals de mogelijkheid tot calibratie, moet in alle gevallen voldaan worden ongeacht de situatie waarin het model gebruikt zal gaan worden. Een en ander heeft tot gevolg dat binnen de beoordelingsmethode de eisen ondergebracht zijn in een van de volgende drie "gewichtsklassen":

- *dwingende eisen.*
  Voldoet een model niet aan één van deze eisen, dan krijgt het een negatieve beoordeling, ongeacht of het model wel of niet aan de andere eisen voldoet.

- *belangrijke eisen.*
  Dit soort eisen zijn weliswaar belangrijk, maar vormen geen onoverkomelijk beletsel vormen bij het accepteren van een model, in het geval de toetsing op een dergelijke eis negatief uitvalt.
- *wenselijkheden.*
  Het is minder belangrijk of een model aan dit soort eisen voldoet. Om het populair te formuleren: "het zou mooi meegenomen zijn als .....".

Binnen de beoordelingsmethode is een verfijning aangebracht waarmee aan elke eis een waardering in de vorm van een cijfer gegeven kan worden. Deze waardering is afhankelijk van de wijze waarop en de uitgangspunten waarmee de betreffende organisatie begrotingsmodellen wil gaan gebruiken.
Bij de toepassing van de beoordelingsmethode in de eerder genoemde contractresearch zijn deze waarderingen gebruikt. In dit onderzoek zal niet verder worden ingegaan op de waardering van de eisen, omdat de gekozen waarden in belangrijke mate worden bepaald door de betreffende organisatie.

Als het model aan de dwingende eisen voldoet dan komt het eindoordeel tot stand door de waarderingen van de individuele eisen bij elkaar op te tellen. De betreffende organisatie zal afspraken moeten maken over de wijze van interpretatie van dit eindoordeel, met andere woorden over de vertaling van een cijfermatig eindoordeel in een uitspraak over de bruikbaarheid van een model. Door Heemstra, van Genuchten en Kusters (1988) wordt hier uitvoerig op ingegaan.

In hoofdstuk 4 worden een aantal begrotingsmodellen beschreven en geëvalueerd. De eisen uit de beoordelingsmethode zullen in de paragrafen 4.2 tot en met 4.5 en paragraaf 4.7.5 worden gebruikt bij de evaluatie van de modellen.

3.4.2. MODEL-EISEN

Een begrotingsmodel dient ondersteuning te bieden bij de beheersing c.q. begroting van de ontwikkeling van software.
Bij aanvang van het project bestaat er nog op tal van punten onzekerheid. In dit stadium van het project zal men behoefte hebben aan een model dat ondersteuning biedt bij de uitvoering van gevoeligheids- en risico-analyses. Het is belangrijk dat het model de gevoeligheden van bepaalde invloedsfactoren aangeeft. Voor het projectmana-

gement vormt dit een waarschuwing welke zaken bij de uitvoering en de bewaking van het project scherp in de gaten gehouden moeten worden. Een begroting heeft in dit stadium een indicatief karakter en dient hoofdzakelijk ter ondersteuning van de beslissing betreffende de haalbaarheid van het project. Naarmate een project vordert, worden de onzekerheden doorgaans minder, stijgt het inzicht in het vereiste eindresultaat en worden plannen steeds gedetailleerder. Het wordt daardoor mogelijk begrotingen bij te stellen en steeds nauwkeuriger te maken. Wat een en ander betekent voor een begrotingsmodel, zal worden toegelicht aan de hand van de eisen die vallen onder de categorie "model-eisen".

Eis: gekoppeld aan beheersingsmethode.
Men kan bij de projectuitvoering gebruik maken van diverse benaderingswijzen. Men kan hierbij denken aan prototyping, pakketselectie (al dan niet met aanpassingen), incrementeel ontwikkelen, spiraal aanpak etc. Vaak zal een bepaalde benaderingswijze samenhangen met een bepaalde verdeling van het project in fasen, of in ieder geval met een ander verdeling van de inspanning over de fasen of met een wijziging in de verwachte productiviteit per fase. Een pakket dat de inspanning en doorlooptijd wil begroten zal rekening moeten houden met de verschillende benaderingswijze om tot een goede begroting te kunnen komen.
Het is voor een begrotingsmodel belangrijk dat het gekoppeld kan worden met een projectbeheersingsmethode (bijvoorbeeld SDM, PRODOSTA, PARAET, PROMPT). Het is opmerkelijk dat er binnen de bestaande begrotingsmodellen zo weinig aandacht wordt besteed aan een dergelijke koppeling. Zo'n koppeling betekent ondermeer dat bij aanvang van elke fase c.q. activiteit door de beheersingsmethode het gebruik van een begrotingsmodel wordt voorgeschreven. Een ander gevolg van een koppeling is dat begrippen, definities e.d. op elkaar zijn afgestemd.
Punten van beoordeling:
- maakt het model gebruik van een fasering
- zijn meerdere faseringen mogelijk
- is het mogelijk de verdeling van de inspanning over de fasen aan te passen
- is het mogelijk een eigen fasering in te brengen.
- kan het model wat betreft definitiestelsel en naamgeving van begrippen worden aangepast
- ondersteunt het model het begroten van projecten die volgens verschillende benaderingen uitgevoerd worden (bijvoorbeeld prototyping, incrementeel ontwikkelen enz.)
- ondersteunt het model pakketselectie.

Eis: vroeg toepasbaar.
Het model moet aan het begin van het ontwikkeltraject bruikbaar zijn. Dit wil zeggen dat alle vragen van het model te beantwoorden moeten zijn op grond van een beschrijving uit bijvoorbeeld een informatieplan en een eventueel gesprek met de opdrachtgever. Bij de beoordeling op dit punt moet bij elke invoervraag van het model gekeken worden of die vraag op redelijke wijze aan het begin van het ontwikkeltraject te beantwoorden is.

Eis: evoluerend.
Naarmate de voortgang van een project vordert, neemt het inzicht in het gewenste eindresultaat toe en krijgt men de beschikking over steeds gedetailleerdere en betrouwbaardere informatie. Een model moet gebruik maken van dit stijgend inzicht en van deze extra informatie door het stellen van nieuwe en gedetailleerdere vragen.
Punten van beoordeling zijn:
- worden naarmate het project vordert:
  * andere vragen gesteld
  * meer gedetailleerde vragen gesteld
  * vragen gesteld die gebruik maken van tussentijds verkregen informatie
  * consistente definities gebruikt.

Eis: aanpasbaar aan doelstellingen.
In eerste instantie zullen de meeste modellen een schatting geven van de inspanning en ontwikkeltijd die de ontwikkeling van een produkt onder "normale" omstandigheden zou kosten. Het zal echter regelmatig voorkomen dat de resulterende ontwikkeltijd te lang is, of dat men over minder personeel beschikt dan volgens de begroting nodig is. Een model moet, met inachtneming van deze nieuwe doelstellingen ook in staat zijn een begroting te maken, waarmee de gevolgen van dergelijke beperkingen op tijd, kwaliteit en inspanning zichtbaar gemaakt kunnen worden.
Punten van beoordeling zijn:
- ondersteunt het model dergelijke randvoorwaarden, te weten:
  * gevolgen van de verkorting van de doorlooptijd
  * gevolgen van beperking personeel
  * gevolgen van gewijzigde kwaliteitseisen.

Eis: toepasbaarheid.
Uit de literatuur valt duidelijk op te maken dat voor verschillende soorten applicaties verschillende soorten modellen geschikt zijn. Het idee, dat men met slechts één model alle uiteenlopende soorten appli-

caties kan begroten, lijkt dus uiterst twijfelachtig. Ergo "verschillende modellen voor verschillende toepassingsgebieden". Per model zal dan ook moeten worden aangegeven voor welk toepassingsgebied het geschikt is. Een model dat claimt voor alle toepassingsgebieden geschikt te zijn, moet met enig wantrouwen benaderd worden.
Punten van beoordeling zijn:
- is het toepassingsgebied gegeven
- zo niet, valt dan af te leiden op welk toepassingsgebied het model specifiek gericht is
- is bekend op welk soort projecten het model gebaseerd is.

3.4.3. TOEPASSINGS-EISEN

Deze hoofdeis heeft betrekking op de vereiste kwaliteit van de resultaten van een model. Vanuit het oogpunt van de gebruiker betekent dit primair dat de uitvoer (de begroting) nauwkeurig moet zijn. De nauwkeurigheid wordt in belangrijke mate bepaald door de mate waarin het model gecalibreerd is met voltooide projectgegevens uit de omgeving waarin het model wordt toegepast.

Eis: calibratie.
Indien een model calibratie niet ondersteunt en niet de mogelijkheid biedt om op systematische wijze gegevens van oude projecten op te slaan om te gebruiken voor calibratie, dan moet het model op dit punt als onvoldoende worden beoordeeld.

Eis: nauwkeurigheid.
Bij de beoordeling van de nauwkeurigheid van een begroting spelen twee aspecten een belangrijke rol, namelijk gemiddelde voorspelfout van een begroting en de variantie ervan. Een belangrijke eis die hieruit af te leiden is, is die van zuivere schatting. Hieronder wordt verstaan dat op de lange duur het gemiddelde van de voorspelfouten, klein is. Om een zuivere schatting te realiseren is calibratie noodzakelijk. Wordt aan de eis van zuiverheid voldaan dan wil dat nog niet zeggen dat de schatting ook bruikbaar is. Het zal ook noodzakelijk zijn dat de variantie van de voorspelfout voldoende klein is. Wat "voldoende klein" is zal de projectmanager voor een bepaalde situatie zelf moeten bepalen. Hij zal zelf moeten aangeven welke afwijkingen hij nog acceptabel acht in het stadium van softwareontwikkeling waar het getoetste model wordt gebruikt.

3.4.4. IMPLEMENTATIE-EISEN

Eis: gebruiksvriendelijk.
Het merendeel van de begrotingsmodellen is beschikbaar in een of meer geautomatiseerde versies. Bij het gebruik van dergelijke pakketten speelt het aspect gebruiksvriendelijkheid een belangrijke rol. Hierbij spelen zaken een rol als het bedieningsgemak, de inzichtelijkheid en de leerbaarheid van het pakket (Bemelmans 1987). Bedieningsgemak heeft betrekking op allerlei ergonomische aspecten zoals de eenvoud van bediening, de mogelijkheid interactief gegevens in te voeren en te veranderen enz. Inzichtelijkheid heeft te maken met de opzet van het pakket. Voor de gebruiker moet deze logisch overkomen. Leerbaarheid wil zeggen dat het pakket eenvoudig en snel door de gebruiker te leren is. Deze leerbaarheid van een pakket wordt in belangrijke mate bepaald door de volledigheid en duidelijkheid van de documentatie en de tijdsduur om het pakket te leren gebruiken.

Eis: gevoeligheids-analyse.
Het begrotingspakket moet het uitvoeren van gevoeligheidsanalyses ondersteunen. Het moet eenvoudig mogelijk zijn, na te gaan wat het effect op het eindresultaat is van het wijzigen van een of meerdere invoerparameters. Het geeft de projectleider/begroter inzicht in de voor het eindresultaat meest belangrijke factoren. De aandacht wordt aldus gericht op die zaken die bij de voortgangsbewaking een belangrijke rol spelen. Bovendien geeft het de mogelijkheid in te schatten wat de mogelijke gevolgen zijn van wijzigingen in invoerparameters waarvan de waarde nog niet helemaal vaststaat.
In principe is een gevoeligheidsanalyse altijd mogelijk. Het zal echter duidelijk zijn dat het uitvoeren van een gevoeligheidsanalyse niet aantrekkelijk is als men voor het veranderen van een invoerparameter, alle invoervragen nogmaals moet doorlopen.
Punten van beoordeling zijn:
- ondersteunt het pakket gevoeligheidsanalyses
- is de gevoeligheidsanalyse eenvoudig uitvoerbaar
- gaat het uitvoeren van een gevoeligheidsanalyse snel

Eis: Risico-analyse.
Bij het begroten van softwareprojecten speelt onzekerheid een belangrijke rol. De specificaties zijn vaak onduidelijk en onvolledig en veranderen meer dan eens tijdens de ontwikkeling van de programmatuur. Het is van belang dat in dergelijke moeilijke begrotingssitua-

ties risico-analyses uitgevoerd worden. Het projectmanagement dient tijdens het totale systeemontwikkelings-traject op de hoogte te zijn van de risico's van het betreffende project. Dit geldt in het bijzonder bij aanvang van een project waarbij de onzekerheden c.q. risico's het grootst zijn. De SarBachets Analys geeft een voorbeeld van zo'n risico-analyse. In deze aanpak worden vragen gesteld die betrekking hebben op de:

1. *projectomvang.* Staat de omvang van het project in een redelijke verhouding tot reeds eerder uitgevoerde projecten ?
2. *automatiseringsnivo binnen het bedrijf.* Hoe is het kennis- en ervaringsnivo van het eigen personeel met betrekking tot de te ontwikkelen software ?
3. *technologie.* Hoe vertrouwd is het ontwikkelteam, de gebruikersorganisatie en de leverancier/fabrikant met de technologie die voor het project wordt gekozen ?
4. *projectorganisatie.* Hoe wordt het project georganiseerd en bemand?
5. *projectomgeving.* Onder welke voorwaarden wordt het project ontwikkeld ?

Door het kritisch doorlopen van alle vragen op deze vijf gebieden en het analyseren van het verzamelde materiaal, wordt een risico-analyse van het gehele project uitgevoerd.
Punten van beoordeling zijn:
- ondersteunt het pakket het uitvoeren van risico-analyse
- is de betekenis van deze analyse voldoende toegelicht
- is bekend wat de samenstellende factoren in deze risico-analyse zijn.

Eis: inzichtelijkheid.
Het vertrouwen in het begrotingspakket en de mate van acceptatie van de resultaten zullen toenemen naarmate de gebruiker meer inzicht heeft in de werking ervan. De inzichtelijkheid van het pakket draagt er toe bij dat nagegaan kan worden hoe de begrotingsresultaten worden bereikt. Zijn de resultaten controleerbaar (verklaarbaar, na te trekken) dan is het mogelijk bij verschillen tussen begroting en realisatie na te gaan waaraan de verschillen te wijten zijn. De begroting is immers vastgesteld op basis van een aantal geregistreerde vooronderstellingen. Dit is een essentiële voorwaarde waaraan voldaan moet zijn wil men kunnen komen tot een beheersing van een project. Is hier namelijk niet aan voldaan, dan is vrijwel nooit aan te geven wat de oorzaak is van een afwijking tussen begroting en realisatie. Hierdoor is het vrijwel onmogelijk gericht actie (bijsturing) te ondernemen.

Naarmate het pakket toegankelijker is en de mogelijkheid biedt direct het effect van veranderingen in de invoer op de uitvoer te traceren, zal de inzichtelijkheid van de resultaten toenemen, zal het vertrouwen van de gebruiker in het pakket stijgen en zal ook het inzicht van de gebruiker in het begrotingsproces vergroot worden. Het gevolg van dit alles maakt een betere beheersing van het project mogelijk.
Bij de beoordeling van deze eis moet men zich realiseren dat inzichtelijkheid geen op zich zelf staande eis is. Er kan alleen aan voldaan worden als ook aan andere eisen voldaan is. De belangrijkste hiervan is eenduidigheid.

Eis: eenduidigheid invoer.
De wijze waarop de invoervariabelen van het pakket gewaardeerd worden, moet eenduidig zijn. Alleen op deze wijze is men in staat de vooronderstellingen waarop het model is gebaseerd eenduidig vast te leggen. Zoals we zagen, is dit een noodzakelijke voorwaarde voor de eis verklaarbaarheid. Men kan zich voorstellen dat tussen alle gebruikers van het pakket eensluidende afspraken worden gemaakt over de interpretatie van bepaalde invoervragen. Het is noodzakelijk dat het pakket een dergelijke aanpak ondersteunt. Deze ondersteuning kan plaatsvinden door een formulering van heldere, niet ambigue vragen, of wat nog beter is, door vragen over zaken die objectief meetbaar zijn.
Punten van beoordeling zijn:
- is de gevraagde invoer meetbaar op een eenduidige wijze
- indien dit niet het geval is, is de vraag dan in ieder geval zo helder en duidelijk gesteld dat misverstanden zoveel mogelijk voorkomen kunnen worden.

Eis: volledigheid en detail van uitvoer.
Bij de beheersing van een project heeft men niets aan vage en algemene begrotingen. De uitvoer van een pakket zal uitgesplitst moeten zijn naar fasen c.q. activiteiten van het project en naar modulen van het produkt. Verder zal naarmate het project vordert, de mate van detail moeten toenemen. De kennis waarover men beschikt neemt immers ook toe.
Punten van beoordeling zijn:
- uitvoer, onderscheiden naar fasen van het project
- uitvoer, onderscheiden naar modulen van het produkt
- uitvoer, onderscheiden naar soorten benodigd personeel
- toename mate van detail uitvoer naarmate project vordert.

3.5. CONCLUSIES

In dit hoofdstuk is een overzicht gegeven van de belangrijkste begrotingsmethoden. Een van deze methoden, te weten begrotingsmodellen, is uitvoeriger beschreven. Zo is de aanpak bij dergelijke modellen beschreven en is de relatie van een model en zijn omgeving in beeld gebracht.
Door het stijgend aanbod van modellen wordt een organisatie voor het keuzeprobleem geplaatst: welk model is voor onze organisatie het meest geschikte. In dit hoofdstuk is een methode gepresenteerd voor het beoordelen van begrotingsmodellen. De eisen die in deze methode worden genoemd, zullen in hoofdstuk 4 worden gebruikt bij de evaluatie van een aantal modellen.

4. BEGROTINGSMODELLEN.

4.1. INLEIDING

De afgelopen jaren zijn er een groot aantal modellen voor het begroten van software ontwikkeld. Deze tendens wordt in de hand gewerkt door de ontwikkelingen die in de hoofdstukken 1 en 2 zijn geschetst.

In dit hoofdstuk wordt een aantal modellen beschreven. Tevens worden deze modellen kritisch geëvalueerd. Hierbij zal gebruik worden gemaakt van de eisen die in de beoordelingsmethode in het vorige hoofdstuk worden genoemd. Voor een aantal modellen is het niet of nauwelijks na te gaan in hoeverre zij aan deze eisen voldoen. Dit komt omdat deze modellen commerciële produkten zijn. De betreffende leveranciers houden de achterliggende concepten en algoritmen geheim. Daarnaast zijn er maar van weinig modellen empirische begrotingsresultaten beschikbaar c.q. gepubliceerd.

Bij de beschrijving van de modellen heb ik een onderscheid gemaakt in:

- ***uitvoerige beschrijving.***
  Dit geldt voor de modellen COCOMO (paragraaf 4.2), PRICE (paragraaf 4.3), Putnam (paragraaf 4.4.), Funktie Punt Analyse (paragraaf 4.5) en Walston en Felix (paragraaf 4.6). Deze modellen worden uitvoerig beschreven omdat daarover via de literatuur veel bekend is en zij belangrijk zijn, hetzij vanuit historisch perspectief, hetzij gezien de huidige actualiteit.
- ***beknopte beschrijving.***
  Een aantal modellen worden beknopt beschreven (paragraaf 4.7). Hiervoor is gekozen omdat een deel van deze modellen een combinatie is van de uitvoerig beschreven modellen of varianten daarvan zijn. Bovendien is een aantal modellen, vanuit commerciële overwegingen, weinig toegankelijk.
- ***geen beschrijving.***
  Een aantal modellen wordt niet beschreven maar slechts genoemd

(paragraaf 4.8). De reden hiervoor is dat deze modellen verouderd zijn, of er weinig of niets van bekend is. Een andere reden kan zijn dat deze modellen zich nog in een ontwikkel/prototype-fase bevinden.

De evaluaties van de modellen verschillen in diepgang. Dit heeft te maken met de algemene bekendheid met het betreffende model. Een aantal modellen dat beknopt wordt beschreven, wordt toch uitvoerig geëvalueerd. Dit geldt voor de modellen Before You Leap, Estimacs, SPQR-20 en BIS-estimator. De reden voor juist deze keuze is dat deze vier modellen de huidige en toekomstige ontwikkelingen op het gebied van begrotingsmodellen goed illustreren.

Na de beschrijving en evaluatie van begrotingsmodellen rijst de vraag welk model het meest geschikt is. Dat deze vraag moeilijk te beantwoorden is, zal in paragraaf 4.9 worden beschreven. Er wordt een aantal onderzoeken aangehaald waarbij modellen op een aantal aspecten onderling zijn vergeleken. In deze paragraaf zal in de vorm van een samenvatting een overzicht gegeven worden van de mate waarin de geëvalueerde modellen voldoen aan de eisen uit de eerder beschreven beoordelingsmethode.

In paragraaf 4.10 wordt aangegeven wat het belang van modellen is, maar vooral wat het belang in de toekomst kan worden, bij het begroten van software. Een dergelijke beschouwing lijkt mij noodzakelijk omdat uit de enqueteresulaten blijkt dat op dit moment modellen maar weinig in de praktijk worden toegepast en dat begrotingen met behulp van modellen vooralsnog niet tot betere resultaten leiden. Bovendien komt uit de evaluaties naar voren dat de huidige kwaliteit van modellen te wensen over laat. Om een typering aan te brengen in de beschreven modellen zullen in de paragraaf conclusies (4.11) de modellen ingedeeld worden naar de eigenschappen die in het vorige hoofdstuk (3.3) zijn genoemd.

4.2. COCOMO (COnstructive COst MOdel)

Er is geen model dat op zo'n duidelijke en toegankelijke manier is vastgelegd als dit model (Boehm 1981 en Boehm 1984). In het kort zal beschreven worden hoe met behulp van dit model een begroting van de ontwikkelkosten wordt uitgevoerd. Tevens zal worden aangegeven dat de kwaliteit van een model als COCOMO valt of staat met

de beschikbaarheid van een uitgebreide verzameling van voltooide projectgegevens.

COCOMO omvat drie versies, die onderling verschillen in de mate van detaillering.

In de meest eenvoudige versie, **BASIC COCOMO**, wordt de benodigde inspanning en ontwikkeltijd berekend als een funktie van de programma-omvang uitgedrukt in het geschatte aantal regels code, waarin het programma wordt geschreven.

INTERMEDIATE COCOMO, de tweede versie, gaat een stap verder in detaillering door naast de omvang nog 15 andere factoren mee te nemen bij het schatten van de kosten. Voor alle versies van COCOMO wordt op basis van historische projectgegevens een verdeling van de inspanning over de ontwikkelfasen van een project gemaakt. BASIC en INTERMEDIATE COCOMO gaan uit van een top-down benadering. Zij beschouwen het ontwikkelproject in zijn totaliteit, bepalen hiervoor de kosten en ontwikkeltijd en verdelen deze tenslotte over de fasen.

DETAILED COCOMO, de meest verfijnde versie van COCOMO, gaat uit van een bottom-up benadering. Een nieuw softwareprodukt wordt met behulp van een Work-Breakdown-Structure aanpak gedecomponeerd in een zogenaamd drie hiërarchie-nivo (systeem - subsystemen - modules). De totale inspanning en ontwikkeltijd wordt stap voor stap opgebouwd door eerst de schattingsalgoritmen op module nivo toe te passen, vervolgens op het nivo van subsystemen en tenslotte voor het totale programma. Bovendien wordt in DETAILED COCOMO rekening gehouden met het feit dat een kostenbepalende factor per module en/of fase een verschillende invloed op kosten en tijd kan hebben.

Van INTERMEDIATE COCOMO zal een beschrijving worden gegeven. Bij gebruik van deze versie van COCOMO moeten de volgende stappen genomen worden:

stap 1. Bepaal de ontwikkelomgeving

In COCOMO worden drie ontwikkelomgevingen onderscheiden. Gebruik makend van Boehm's terminologie zijn dat:

- *de organic mode.*
  De ontwikkeling vindt plaats in een klein team, dat software in een voor de ontwikkelaars vertrouwde omgeving ontwikkelt. Over het algemeen heeft het betreffende ontwikkelteam veel ervaring met soortgelijke projecten. Het project is veelal niet groot en kan min of meer onafhankelijk van andere projecten of reeds

bestaande systemen worden uitgevoerd. Er hoeven weinig initiële kosten te worden gemaakt, men kan snel aan de slag.
- *de Embedded mode.*
  In dit geval heeft men te maken met het ontwikkelen van complexe software, met hoge eisen ten aanzien van integreerbaarheid met andere software. Het eindresultaat zal ingebed moeten worden in een bestaande software-omgeving (bijvoorbeeld: luchtverkeersgeleidingsysteem).
- *de Semi-detached mode.*
  Een tussenvorm van organic en embedded.

stap 2. Bepaal de nominale inspanning

Bij de opzet van COCOMO heeft Boehm gebruik gemaakt van gegevens van 63 voltooide softwareprojecten. Met behulp van deze gegevens is de nominale ontwikkelinspanning (in mensmaanden) voor een project afgeleid als een funktie van de te schatten omvang en de in stap 1 bepaalde ontwikkelomgeving. In tabel 4.1 is deze relatie weergegeven.

Tabel 4.1 : De nominale inspanning als funktie van de ontwikkelomgeving en het aantal regels code. MM(nom) staat voor het aantal nominale mensmaanden. TONT(nom) staat voor de nominale ontwikkeltijd. KDSI is het aantal delivered source instructions / 1000 (aantal regels code gedeeld door duizend).

ONTWIKKELOMGEVING	MM(nom)	TONT(nom)
Organic	$3.2*KDSI^{1,05}$	$2.5*MM(nom)^{0,38}$
Semi-detached	$3.0*KDSI^{1,12}$	$2.5*MM(nom)^{0,35}$
Embedded	$2.8*KDSI^{1,20}$	$2.5*MM(nom)^{0,32}$

Stel dat in stap 1 bepaald is dat de ontwikkelomgeving semi-detached is en dat in stap 2 de omvang van het programma geschat is op 100.000 regels code dan kunnen aan de hand van tabel 4.1 de nominale waarden voor inspanning en ontwikkeltijd worden bepaald.

stap 3. Karakteriseer het project aan de hand van kostenbepalende factoren.

In COCOMO worden 15 factoren (cost drivers) onderscheiden die de inspanning en ontwikkeltijd beïnvloeden. De mate van de invloed van elke afzonderlijke factor wordt bepaald door de waarde van die factor. Zo zal er meer inspanning nodig zijn naarmate de kwaliteitseisen die aan het systeem worden gesteld, toenemen.
Aan de hand van een uitgebreide analyse van de gegevens van de 63 historische projecten en het oordeel van experts (met behulp van de Delphi-methode) is in COCOMO voor elke waarde van de 15 factoren de invloed op de nominale waarden kwantitatief bepaald. In tabel 4.2 is hiervan een overzicht gegeven.

Tabel 4.2 : De invloed van de verschillende cost drivers op de nominale waarden van ontwikkelinspanning en tijd.

	waarde factoren					
KOSTENBEPALENDE FACTOREN	erg laag	laag	gemiddeld	hoog	erg hoog	extra hoog
betrouwbaarheid software	.75	.88	1.00	1.15	1.40	
omvang database		.94	1.00	1.08	1.16	
complexiteit software	.70	.85	1.00	1.15	1.30	1.65
beperkingen executietijd			1.00	1.11	1.30	1.66
beperkingen werkgeheugen			1.00	1.06	1.21	1.56
vervangingsgraad computer		.87	1.00	1.15	1.30	
responsietijd		.87	1.00	1.07	1.15	
kwaliteit analisten	1.46	1.19	1.00	.86	.71	
ervaring met applicaties	1.29	1.13	1.00	.91	.82	
kwaliteit programmeurs	1.42	1.17	1.00	.86	.70	
ervaring met apparatuur	1.21	1.10	1.00	.90		
ervaring met prog. taal	1.14	1.07	1.00	.95		
gebruik moderne prog.tech.	1.24	1.10	1.00	.91	.82	
gebruik software tools	1.24	1.10	1.00	.91	.83	
eisen aan projectduur	1.23	1.08	1.00	1.04	1.10	

Voor elke kostenbepalende factor geeft Boehm in de beschrijving van COCOMO (Boehm 1981) uitvoerig aan wat de betekenis is van een zeer lage, lage, gemiddelde enz. factorwaarde. In deze stap zal de schatter de software aan de hand van de 15 factoren moeten karakteriseren. In welke mate zal er gebruik worden gemaakt van hulpmiddelen, wat is de kwaliteit van de programmeurs die ingezet worden op het project, wat is de complexiteit van de programmatuur enz. Af-

hankelijk van die karakterisering geeft COCOMO de bijbehorende beïnvloedingswaarden.

stap 4. Bepaal de ontwikkelinspanning.

De totale ontwikkelinspanning voor een nieuw programma wordt berekend door de waarde van de nominale inspanning uit stap 2 te vermenigvuldigen met het produkt van de 15 beïnvloedingswaarden. De totale inspanning wordt dan bepaald met behulp van de volgende vergelijking:

$$MM = MM(nom) * \prod_{i=1}^{n} F(i) \qquad (1)$$

waarbij F(i) staat voor de invloed van cost driver i op de inspanning.

stap 5. Schat de ontwikkeltijd en bepaal de verdeling van de inspanning en tijd over de verschillende fasen.

De schatting van de ontwikkeltijd wordt afgeleid van de bepaling van de inspanning; namelijk met behulp van de volgende vergelijking (zie tabel 4.1):

$$TONT = 2.5 * (MM)^{a} \qquad (2)$$

waarbij de waarde van a afhankelijk is van de soort ontwikkelomgeving. Voor de verdeling van het aantal mensmaanden en de tijd over de verschillende fasen en aktiviteiten per fase biedt INTERMEDIATE COCOMO uitgebreide "verdelingstabellen".

EVALUATIE VAN COCOMO.

COCOMO is afgeleid van een databank van 63 projecten die tussen de jaren 1964 en 1979 werden uitgevoerd door het Amerikaanse bedrijf TRW Systems Inc. De projecten verschillen onderling sterk. Wat omvang betreft lopen de programma's uiteen van 2000 tot 1.000.000 regels code. De gebruikte programmeertalen zijn ondermeer assembler, COBOL, FORTRAN en PL/1. Een breed spectrum van soorten applicaties wordt afgedekt; administratieve en wetenschappelijke toepassinger maar ook applicaties voor onder andere procescontrole. Juist die grote verscheidenheid maken de toepassingsmogelijkheden van deze databank voor een schattingsmodel als COCOMO twijfelachtig ondanks het grote aantal projecten dat geinventariseerd is. Zo moet in COCOMO de invloed van de 15 cost drivers (met omvang erbij 16) worden

bepaald. Dat moet gebeuren voor drie ontwikkelomgevingen en voor maximaal 6 waarden voor de verschillende factoren (erg laag, laag, nominaal, hoog, erg hoog en extra hoog). Uit tabel 4.2 valt af te leiden dat er 54 verschillende invloedswaarden zijn bepaald. Het is niet waarschijnlijk dat elke afzonderlinge waardebepaling gebaseerd is op een voldoende grote steekproef. Zo zijn slechts zeven van de 63 projecten ontwikkeld in een semi-detached omgeving. Van deze zeven heeft er maar één betrekking op een administratieve toepassing, geschreven in PL/1, met een omvang van 132.000 regels code en met waarden voor de cost drivers die over het algemeen weinig van de gemiddelde waarden afwijken. Het is nog maar de vraag in hoeverre de overall-waarden uit tabel 4.3 toegepast mogen worden op een nieuw project, eveneens in een semi-detached omgeving maar dan veel omvangrijker en met factorwaarden die aanzienlijk afwijken van die van dat ene geregistreerde project (bijvoorbeeld hoge complexiteit, hoge inzet van tools enz.). De conclusies die uit het bovenstaande getrokken mogen worden, zijn dat de databank te klein is en teveel uiteenlopende projecten omvat om zoveel verschillende waarden betrouwbaar te kunnen bepalen. Het is aan te bevelen (zie ook Conte, Shen en Dunsmore 1986) het aantal cost drivers te beperken tot uitsluitend de meest relevante en in de databank de gegevens van een homogenere groep projecten vast te leggen.

Indien men voor het model COCOMO nagaat in hoeverre voldaan wordt aan de eisen van de gepresenteerde beoordelingsmethode, dan geeft dit het volgende beeld:

MODEL-EISEN:

- *gekoppeld aan beheersingsmethode.*
  COCOMO maakt gebruik van een eigen fase-indeling, van een eigen begrippen en definitiestelsel. Verder is het model sterk geënt op een traditionele wijze van software-ontwikkeling. Het is niet mogelijk binnen COCOMO een andere fasering aan te brengen. Bovendien ondersteunt het model niet pakketselectie en andere benaderingen om software te ontwikkelen, zoals prototyping, incrementeel ontwikkelen e.d. Het is wel mogelijk de verdeling van de inspanning over de fasen aan te passen.
- *vroeg toepasbaar.*
  Voor het bepalen van de omvang maakt COCOMO gebruik van het aantal regels code, waarin het te ontwikkelen programma wordt geschreven. In een vroeg stadium van software-ontwikkeling is het schatten hiervan uiterst moeilijk. In een experiment met 15 ervaren projectleiders was het merendeel niet bereid in een vroeg stadium (aan het begin van de definitiestudie) een schatting te maken van het aantal regels code. Dit kon wel voor het aantal funktiepunten

(Heemstra, van Genuchten, Kusters 1988a). COCOMO laat de begroter bovendien in de kou staan als het gaat om een manier om deze schatting uit te voeren. De kritiek uit de hoek van de aanhangers van modellen gebaseerd op funktiepunten is terecht als zij beweren dat COCOMO er wat betreft praktische toepasbaarheid op vooruit zou gaan als voor de bepaling van de omvang gebruik gemaakt zou worden van funktiepunten. Het model Before You Leap is een voorbeeld van een COCOMO implementatie met een schatting van de omvang in funktiepunten.

- *Evoluerend.*
  Met name DETAILED COCOMO biedt een beperkte mogelijkheid om de invloed van de cost drivers per fase te variëren. Men blijft echter gebonden aan de vaste verzameling factoren.
- *Aanpassing doelstellingen.*
  Hiermee houdt COCOMO rekening. De factoren "vereiste betrouwbaarheid", "kwaliteit/ervaring personeel" en "eisen aan projectduur" kunnen hiervoor aangepast worden.
- *Toepasbaarheid.*
  COCOMO richt zich voornamelijk op embedded software.

TOEPASSINGS-EISEN:

- *Calibratie.*
  COCOMO biedt geen mogelijkheden voor calibratie. Wil men het model toepassen in de eigen organisatie en eigen projectgegevens als basis voor het model gebruiken, dan is men verplicht een eigen COCOMO versie te ontwikkelen. In zo'n geval zal nagegaan moeten worden of nieuwe cost drivers toegevoegd en/of bestaande verwijderd moeten worden. Bovendien moeten de beïnvloedingswaarden opnieuw worden bepaald.
- *Nauwkeurigheid.*
  Er zijn weinig empirische begrotingsresultaten met COCOMO beschikbaar. Resultaten die er wel zijn hebben veelal betrekking op niet gecalibreerde toepassingen. Dit wil zeggen dat de parameterwaarden die gebruikt worden rechtstreeks afkomstig zijn uit de beschrijving van COCOMO, met andere woorden betrekking hebben op de ontwikkelomgeving van TRW in de USA. Deze waarden zijn niet aangepast aan de ontwikkelomgeving waarin het model is gebruikt. Deze resultaten laten zien dat zonder calibratie enorme miscalculaties kunnen worden gemaakt. In paragraaf 4.9 wordt hierop nader ingegaan.

IMPLEMENTATIE-EISEN:

- *Gebruiksvriendelijk.*
  COCOMO is inzoverre gebruiksvriendelijk dat de documentatie uitstekend is. Aan een implementatie in de eigen organisatie zitten

echter de nodige haken en ogen.
Er bestaan verschillende geautomatiseerde versies van COCOMO. De meest bekende zijn (Sierevelt 1985):

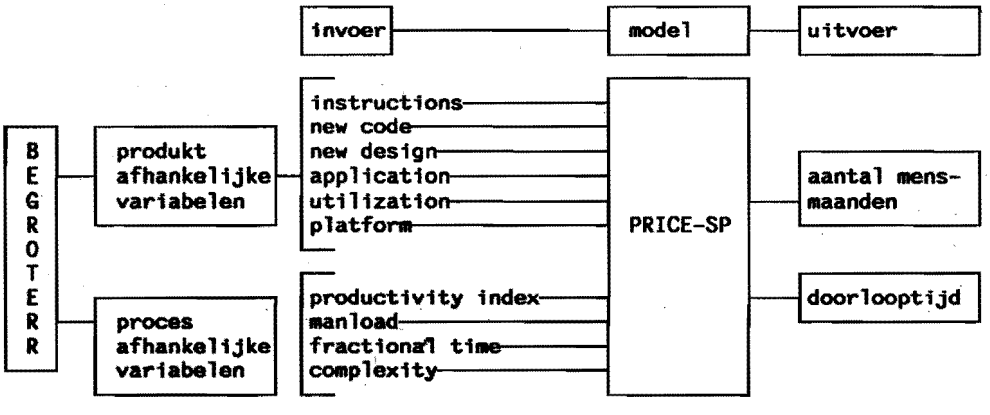
* WICOMO
  (Demshki c.s. 1982) (Ligett 1985), ontwikkeld door het Wang Institute in Tyngboro. Visser (1987) beschrijft een evaluatie van een praktische toepassing van WICOMO,
* GECOMO
  (Rooks 1985a en b, GECOMO 1985) van GEC Software Limited in Londen,
* PCOC of PC-COCOMO
  (Royce 1985) van Electric Systems in Torrance (V.S.),
* GTCOCOMO
  (Price 1985) van het US army electronics research en development command forth monmounth in de V.S.,
* SECOMO
  (Goethert 1988) van het ITT research institute,
* ADA COCOMO (Royce 1988).

- *Gevoeligheids-analyse.*
  Dit wordt niet door COCOMO ondersteund.
- *Risico-analyse.*
  Hoewel Boehm in het standaardwerk "software engineering economics" (1981) uitgebreid ingaat op risico-analyse, wordt aan deze eis wordt door COCOMO niet voldaan.
- *Inzichtelijkheid en duidelijkheid invoer.*
  In Boehm (1981) wordt het model tot in de finesses beschreven. Gebruik van COCOMO dwingt de gebruiker gestructureerd na te denken over begroten. De documentatie in de vorm van het genoemde boek is daarbij een uitstekend hulpmiddel. De achterliggende concepten en algoritmen zijn volledig bekend en is het altijd mogelijk de resulaten na te trekken. Een sterk punt van COCOMO is de zorgvuldige wijze waarop de cost drivers geoperationaliseerd zijn.
- *Volledigheid en detail van uitvoer.*
  Bij BASIC en INTERMEDIATE COCOMO is die beperkt tot slechts de benodigde inspanning en tijd voor het totale project. Bij DETAILED COCOMO vindt er een uitsplitsing plaats naar fasen. Van een uitsplitsing naar benodigde middelen is geen sprake.

4.3. DE PRICE MODELLEN

PRICE staat voor Programmed Review of Information Costing and Evaluation. De modellen zijn ontwikkeld en worden ondersteund door

RCA PRICE Systems, een onderdeel van de Radio Corporation of America. RCA is ongeveer vijftien jaar geleden begonnen met het ontwikkelen van modellen, aanvankelijk alleen voor het begroten van hardware kosten. Aangemoedigd door het succes heeft RCA een soortgelijk model ontwikkeld voor het begroten van softwarekosten. Weer later zijn hieraan toegevoegd modellen voor het schatten van onderhoudskosten en voor het bepalen van de omvang van een produkt in aantal regels code. Het meest recente model is PRICE-SP voor het begroten van het aantal benodigde mensmaanden en de doorlooptijd van softwareprojecten (PRICE 1985).
Een belangrijk nadeel ten opzichte van COCOMO is dat de werking van het model geheim is. Ook voor de gebruikers blijft het model een black box. De gebruiker van PRICE stuurt zijn invoergegevens naar een time sharing computer in de Verenigde Staten, Groot Brittannië of Frankrijk en krijgt vrijwel direct de gevraagde begroting terug. Ondanks dit nadeel en de hoge huurprijs wordt het model met name in de Verenigde Staten veel gebruikt. Enkele gebruikers daar zijn: Boeing, General Dynamics, Het Amerikaanse ministerie van Defensie van defensie, IBM, General Electric en Texas Instruments. Een belangrijke drijfveer voor een groot aantal bedrijven om het model te gebruiken komt voort uit de eis van het Amerikaanse ministerie dat elke offerte die bij haar wordt ingediend betreffende een softwareproject, gebaseerd dient te zijn op een begroting met PRICE. In Nederland is Philips de enige gebruiker.
Over de werking van het model valt, vanwege de eerder genoemde geheimhouding, weinig te zeggen. De invoer en uitvoer van het model is weergegeven in figuur 4.1.



Figuur 4.1 : Schema PRICE-SP.

De uitvoer van het model bestaat in essentie uit een begroting van het aantal benodigde mensmaanden en de doorlooptijd van het project. Dit zijn de te verklaren variabelen. De invoer van het model is een karakterisering van het te begroten project. Deze karakterisering bestaat uit tien variabelen, de zogenaamde verklarende variabelen. Deze variabelen en hun definities zijn opgenomen in tabel 4.3.

Tabel 4.3 : De invoervariabelen van PRICE-SP.

Variabele	Omschrijving uit PRICE manual	toelichting
instructions	is the total number of deliverable, executable machine level instructions	grootte van programma
new code	is the amount of new code	tussen 0 en 1
new design	is the amount of new design	tussen 0 en 1
application	summarizes the application mix of instructions	soort toepassing
utilization	fraction of available hardware life cycle time or total memory capacity used	tussen 0.8 en 11
platform	describes the planned operating environment for the software	waarde tussen 0.6 en 2.5
productivity index	is an empirically derived parameter that serves as a productivity, skill level experience and efficiency index	
manload	average number of software personnel involved in the software project over the entire project	
fractional time	is the average fractional time dedicated to the software job	fractie v/h aan-uren, besteed aan project
complexity	describes the relative effect of complicating factors as product familiarity, personnel software skills, hardware/software design interactions as they effect manpower costs.	

De definities zijn letterlijk overgenomen uit de Engelstalige PRICE manual (PRICE 1985). Omdat ze soms niet erg verhelderend zijn, is waar nodig een korte toelichting gegeven.

De tien invoervariabelen zijn onder te verdelen in twee groepen (zie figuur 4.1). De eerste groep van zes variabelen beschrijft het te ontwikkelen produkt. De tweede groep van vier variabelen beschrijft het ontwikkelproces dat het gewenste eindprodukt moet gaan opleve-

ren.

De variabele *"instructions"* is een maat voor de grootte van het programma. De gebruiker moet het aantal regels machinecode opgeven.

De variabelen *"new code"* en *"new design"* geven aan welk deel van het produkt geheel opnieuw ontwikkeld moeten worden. Het is mogelijk dat een deel van de code en het ontwerp uit de literatuur of uit een vorig project gehaald kan worden. De waarde van deze variabelen zullen in dat geval kleiner zijn dan 1.

De variabele *"application"* beschrijft de soort toepassing. De gebruiker wordt geacht de waarde van deze variabele te bepalen door uit een gegeven klassificatie die klasse te kiezen die het beste aansluit bij de bewuste applicatie. Dit is geen makkelijke opgave door de soms vage omschrijving van de klassen.

Eventuele hardware beperkingen worden beschreven in de variabele *"utilization"*. Men kan hierbij denken aan de beperkte geheugenruimte van de computer waarop het te ontwikkelen softwareprodukt moet gaan functioneren.

De laatste variabele die het project karakteriseert is *"platform"*. Platform beschrijft de omgeving waarin het te ontwikkelen produkt gebruikt gaat worden. Over het algemeen zal voor een afdeling de waarde van deze variabele voor de verschillende projecten gelijk zijn. De gebruiker moet de waarde van platform bepalen door uit een gegeven tabel die omgeving te kiezen, die de meeste overeenkomsten vertoont met zijn eigen omgeving.

"productivity index" is een variabele die door calibratie wordt bepaald op grond van een aantal voltooide projecten.

De variabel *"manload"* is opgenomen, omdat mensen en tijd niet onderling uitwisselbaar zijn (zie hoofdstuk 1).

De variabele *"fractional time"* beschrijft het verschijnsel dat versnippering van de aandacht leidt tot een lagere productiviteit (Sierevelt 1986). Een in de softwarewereld veel voorkomend verschijnsel is dat de individuele ontwikkelaar zich deels bezig moet houden met de ontwikkeling van een of meerdere nieuwe produkten en deels met het onderhoud van bestaande systemen.

De variabele *"complexity"* tot slot beschrijft projectkenmerken die vooral van invloed zijn op de doorlooptijd. De standaardwaarde is 0.1. Eventuele afwijkingen van die standaardwaarde moet de gebruiker bepalen met behulp van de documentatie van PRICE-SP.

EVALUATIE VAN PRICE-SP.

Een nadeel van de PRICE modellen in vergelijking met COCOMO is het feit dat de inhoud van de modellen geheim is. Juist om die reden is het lastig na te gaan in hoeverre het model PRICE-SP voldoet aan de eisen uit de beoordelingsmethode. Ondanks dit bezwaar is dit model toch uitvoerig beschreven en geëvalueerd, omdat ermee ervaring is opgedaan (Cuelenaere, van Genuchten, Heemstra 1987).

MODEL-EISEN:

- *Gekoppeld aan beheersingsmethode.*
  PRICE-SP voldoet niet aan deze eis.
- *Vroeg toepasbaar.*
  Om dezelfde reden als COCOMO is PRICE-SP niet geschikt om in een vroeg stadium van software-ontwikkeling te gebruiken. Vermeld moet echter worden dat binnen de PRICE modellen een "sizer"-module bestaat, die speciaal gericht is op het schatten van de omvang. Wat dat betreft geven de PRICE-modellen, in tegenstelling tot COCOMO, de begroter een ondersteuning bij het bepalen hiervan.
- *Evoluerend.*
  Het model biedt hiervoor geen mogelijkheden.
- *Aanpassen aan doelstellingen.*
  PRICE-SP biedt hiervoor beperkte mogelijkheden. Er kan geanalyseerd worden wat het effect op kosten en doorlooptijd is als de software eerder gereed moet zijn of er minder personeel beschikbaar is dan in eerste instantie gepland.
- *Toepasbaarheid.*
  Hierover doet het model geen uitspraak.

TOEPASSINGS-EISEN:

- *Calibratie.*
  PRICE-SP biedt hiervoor beperkte mogelijkheden. Calibratie gebeurt door een aantal voltooide projecten met behulp van het model te beschrijven. De invoer bestaat nu, behalve de bekende waarden van de invoervariabelen, ook uit het aantal (gerealiseerde) mensmaanden en de (gerealiseerde) doorlooptijd. De variabele "productivity index" behoort bij de calibratie niet tot de invoer, maar is de uitvoer. Omdat het model als het ware omgekeerd wordt gebruikt, heeft RCA de calibratietoepassing van het model ECIRP genoemd. Deze vorm van calibratie is inzoverre beperkt dat de aanpassing aan de eigen ontwikkeling in één variabele wordt ondergebracht. Toevoegen en/of verwijderen van cost drivers is niet mogelijk. Veranderen van de invloed van de cost drivers is ook niet mogelijk.

- *Nauwkeurigheid.*
  RCA claimt dat begrotingen het model, mits gecalibreerd, gemiddeld genomen slechts 6% afwijken van de realiteit. Gegevens over de bij dit getal behorende standaardafwijking en betrouwbaarheidsintervallen worden door RCA niet gegeven. Helaas ontbreekt het aan voldoende empirische gegevens om deze cijfers te verifiëren.

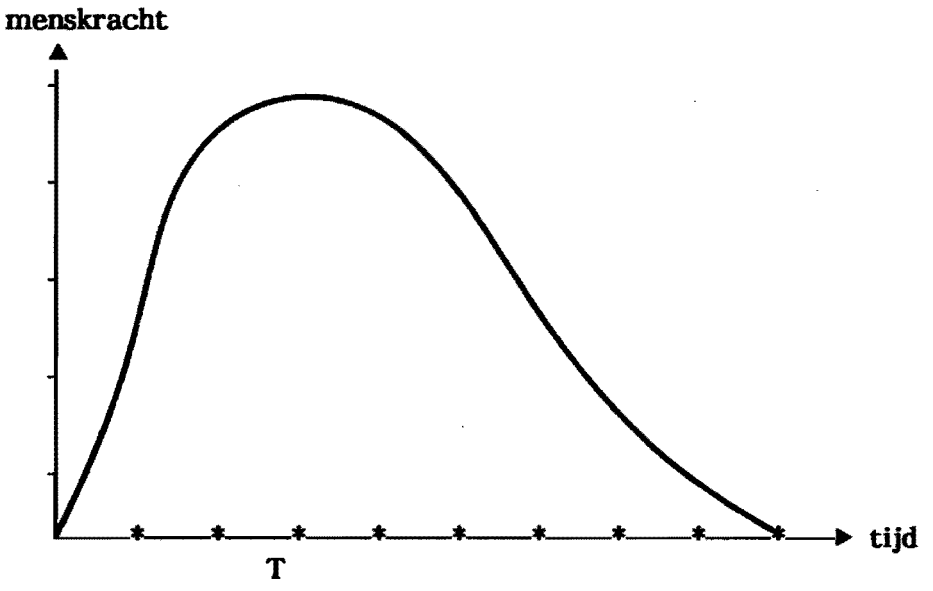
IMPLEMENTATIE-EISEN:

- *Gebruiksvriendelijk.*
  Zoals in de beschrijving is aangegeven, is het model door RCA geïmplementeerd op een mainframe computer. Het enige voordeel voor de begroter is dat de responsietijd kort is. Voor de rest is het model een black box. De documentatie is soms moeilijk te interpreteren. Een en ander heeft ertoe geleid dat Philips (als gebruiker) het model, en dan met name het invoergedeelte, op een aantal punten heeft aangepast (Sierevelt 1986).
- *Gevoeligheidsanalyse.*
  Het model claimt te beschikken over uitgebreide faciliteiten om gevoeligheidsanalyses uit te voeren. Het manipuleren met de waarden van de invoervariabelen en het uitvoeren van "What-If" analyses blijkt weliswaar mogelijk te zijn, maar blijkt in de praktijk een moeilijke en tijdrovende bezigheid te zijn.
- *Risico-analyse.*
  Aan deze eis voldoet het model niet.
- *Inzichtelijkheid.*
  Omdat de inhoud van het model geheim is het niet mogelijk na te gaan hoe de begrotingsresultaten worden bereikt. Het gevaar bestaat dat de ontwikkelaar het model ervaart als een toverdoos. Een gevolg van die geheimhouding is dat de gebruiker weinig gevoel krijgt voor de onderlinge samenhang van de cost drivers en voor de invloed van een individuele cost driver op de kosten. Slechts door langdurig experimenteren met behulp van gevoeligheidsanalyses kan men enig inzicht krijgen in die samenhang.
- *Eenduidigheid invoer.*
  De tabellen in de documentatie zijn vaak moeilijk te interpreteren. Dit heeft tot gevolg dat het bepalen van de waarden van de invoervariabelen nog moeilijker wordt dan het al is. In de praktijk ziet men dan ook dat alleen zeer ervaren modelgebruikers in staat zijn de tabellen juist te interpreteren en de waarde van de invoervariabelen met redelijk succes in te schatten.
- *Volledigheid en detail van uitvoer.*
  PRICE-SP biedt een breed scala van uitvoermogelijkheden. Hierbij kan een onderscheid worden gemaakt in de uitvoer van:

* **Kosten.**
  De kosten (in een gewenste munteenheid en aantal mensmaanden, mensuren, enz.) uitgesplitst naar fasen en activiteiten.
* **Invoergegevens.**
  Alle invoervariabelen, de opgegeven waarden en de achterliggende cost drivers kunnen in een overzicht worden weergegeven.
* **Additionele informatie.**
  Deze informatie heeft vooral betrekking op de betrouwbaarheid en consistentie van de invoer.
* **Grafische weergave.**
  Naar believen kan de uitvoer grafisch gepresenteerd worden.
* **Gevoeligheidsmatrix.**
  (zie eis gevoeligheidsanalyse)
* **Planningsoverzicht.**
  Hierin wordt een vergelijking gegeven tussen het plan zoals dat door PRICE wordt bepaald en het plan zoals dat door de gebruiker is gespecificeerd. Bovendien wordt aangegeven wat de gevolgen op de kosten zijn bij aanpassing van de plannen.
* **Kostenverdeling.**
  Een overzicht van de werklast en kosten over het totale project.

4.4. HET MODEL VAN PUTNAM

Dit schattingsmodel werd in 1974 ontwikkeld door Putnam, destijds directeur van de softwaredivisie van het Amerikaanse ministerie van defensie (Putnam 1978). Hij bouwde hiermee verder op het werk van Norden (1963). Norden bestudeerde de verdeling van het aantal ontwikkelaars tijdens de ontwikkeling van geautomatiseerde systemen bij IBM en kwam tot de conclusie dat hierin een zeer regelmatig patroon te herkennen was. Hij maakte frequentie-verdelingen voor elk project, waarin hij liet zien hoeveel mensen er vanaf het begin tot aan het einde aan een project waren toegewezen. Deze verdeling van menskracht in de tijd gaf een beeld zoals weergegeven in figuur 4.2. De karakteristieke curve leek veel op een kansverdeling. Norden vergeleek deze curve met bekende kansverdelingen en vond er een die er wonderwel goed bij aansloot, te weten de Rayleigh curve. Norden's waarnemingen waren louter empirisch. Er waren geen duidelijke aanwijsbare redenen waarom de verdeling van mankracht in de tijd zich zou gedragen volgens een Rayleigh curve. Het bleek gewoon zo te zijn (DeMarco 1982).



Figuur 4.2 : Verdeling van menskracht in de tijd volgens Norden.

Putnam heeft op basis van deze veronderstelling een model ontwikkeld waarin de menskracht op tijdstip t weergegeven wordt door:

$$Mk(t) = 2\ K\alpha t\ ^{-\alpha t^2} \qquad (1)$$

waarbij Mk(t) staat voor benodigde menskracht op tijdstip t, α een parameter is die de hellingshoek van de curve bepaalt en K gelijk is aan de oppervlakte van het gebied onder de curve over het interval $[0,\infty]$.
Een bijzonder geval is $Mk(\infty)=K$. Dit correspondeert met de totale benodigde "life-cycle" inspanning (inclusief ontwikkeling en onderhoud). Als T het tijdstip is waarop de Rayleigh-curve een maximum bereikt, met andere woorden de benodigde menskracht maximaal is, geldt dat de parameterwaarde α gelijk is aan $1/(2T^2)$. Dit tijdstip T zal dicht liggen bij het tijdstip waarop de software-ontwikkeling gereed is c.q. de programmatuur wordt opgeleverd. De oppervlakte onder de curve tussen t=0 en t=T is aldus een goede benadering voor de benodigde inspanning tot aan het moment van oplevering van de software. Wordt T gesubstitueerd voor t in vergelijking (1) dan krijgen we als schatting van de totale ontwikkelinspanning I:

$$I = Mk(T) = 0{,}3945K \qquad (2)$$

Een en ander betekent dat van de totale "life-cycle" inspanning 39,45% voor rekening komt van ontwikkeling en 60,55% voor onderhoud. Deze cijfers komen erg goed overeen met empirische gegevens over de verdeling van de totale inspanning over ontwikkeling en onderhoud (zie ook hoofdstuk 3).

In de theorie van Putnam speelt de moeilijkheidsgraad (MG)

$$MG = K/(T^2) \qquad (3)$$

een belangrijke rol bij het bepalen van de ontwikkelinspanning. Uit een analyse van 50 softwareprojecten, alle uitgevoerd bij de NASA, signaleerde Putnam dat voor software die eenvoudig te ontwikkelen was MG een lage waarde had. Voor moeilijk te ontwikkelen software, bleek MG een hoge waarde te hebben. Op basis van deze constatering stelde Putnam de hypothese dat er een relatie moet zijn tussen de moeilijkheidsgraad en de produktiviteit (P), te weten:

$$P = \text{constante} * MG^{\beta} \qquad (4)$$

Uit een nadere analyse van de eerder genoemde 50 softwareprojecten constateerde Putnam dat de waarde van ß goed overeenkwam met $^2/_3$. Produktiviteit is ook uit te drukken als funktie van omvang en ontwikkelinspanning, namelijk:

$$P = O/I \qquad (5)$$

O staat voor omvang en wordt uitgedrukt in aantal regels code. De ontwikkelinspanning I is volgens formule (2) gelijk aan Mk(T). Door in formule (5) de waarde van produktiviteit uit formule (4) te substitueren ontstaat Putnam's zogenaamde "softwarevergelijking":

$$O = P * I = \text{constante} * MG^{-2/3} * (0{,}3945K) \qquad (6)$$

Substitueren we in deze formule voor MG de waarde K/T^2 uit formule 5 dan geeft dit:

$$O = TC * (K^{1/3}) * (T^{4/3}) \qquad (7)$$

TC is een zogenaamde technologie constante, die gebruikt wordt om de karakteristieken van de ontwikkelorganisatie te beschrijven, zoals hardware faciliteiten, gebruik van moderne tools en technieken e.d.).

Deze constante moet bepaald worden voor elke specifieke organisatie c.q. afdeling door het model te calibreren met behulp van gegevens van voltooide projecten.

Verder wordt in het model van Putnam een zogenaamde moeilijkheidsgradiënt (G) onderscheiden. De waarde van deze gradiënt is afhankelijk van de moeilijkheid om een bepaald programma te ontwikkelen. G is constant voor een bepaald project. Putnam geeft aan dat er 6 verschillende waarden van deze gradiënt te onderscheiden zijn. De minimale waarde 7,3 correspondeert met complexe software; de maximale waarde 89,0 hoort bij eenvoudige software c.q. software waarin bij de ontwikkeling in hoge mate gebruik is gemaakt van reeds bestaande code. De moeilijkheidsgradiënt legt de relatie vast tussen de ontwikkelinspanning en -tijd:

$$G = I / T^3 \qquad (8)$$

Op basis van bovenstaande formules kan worden afgeleid dat er sprake is van een laagst toegestane ontwikkeltijd (TMIN). Als voor de ontwikkeling van een bepaald programma TC en G bekend is en O is geschat, dan kunnen de volgende twee vergelijkingen worden opgelost voor TMIN en voor I.

$$(I^{1/3}) * (TMIN^{4/3}) = O/TC \quad \text{en} \quad I/(T^3) = G \qquad (9)$$

Willen we een schatting maken van de benodigde inspanning voor de ontwikkeling van software, dan kan formule (7) worden geschreven als:

$$K = (O^3) / (TC^3 * T^4) \qquad (10)$$

Deze vergelijking impliceert dat in het model van Putnam wordt uitgegaan van de veronderstelling dat een gewenste versnelling van de ontwikkeltijd van een project zeer hoge kosten met zich meebrengt. Bij de evaluatie van Putnam's model zal dit aan de hand van een cijfervoorbeeld worden toegelicht.

EVALUATIE VAN HET MODEL VAN PUTNAM.

Zoals in de beschrijving van het model is gesignaleerd, brengt een gewenste versnelling van de ontwikkeltijd van een project wel erg hoge kosten met zich mee. Het volgende voorbeeld, ontleend aan

Shen, Conte en Dunsmore (1986) moge dit verduidelijken. In het voorbeeld wordt aangenomen dat het gaat om de ontwikkeling van software met een geschatte omvang van 500.000 regels code. De technologie constante bedraagt 5000 en de moeilijkheidsgradiënt is 89. In tabel 4.4 is aangegeven hoe de benodigde inspanning verandert als de ontwikkeltijd wijzigt.

Tabel 4.4 : Relatie tussen ontwikkeltijd en inspanning volgens het model van Putnam.

ONTWIKKELTIJD (in jaren)	INSPANNING (in mensjaren)
5,0	1600
4,0	3906
3,5	6664
3,0	12346
TMIN = 2,7	14125

Zoals uit de tabel blijkt, betekent een verkorting van de ontwikkeltijd van 5 naar 4 jaar een kostentoename met een factor 2,4 en een verkorting van 4 naar 3 jaar een toename met een factor 7.7. De hierboven geïllustreerde wetmatigheid is niet in die mate terug te vinden bij andere modellen. Zo stelt Boehm dat een verkorting van de doorlooptijd met 25% een toename van de inspanning met 23% tot gevolg heeft.

Het model blijkt goed bruikbaar te zijn voor grote softwareprojecten (Conte, Shen en Dunsmore 1986, DeMarco 1982). Groot wil hier zeggen meer dan 30 mensjaren. Voor kleine projecten blijkt het model veel minder geschikt te zijn. Volgens Noth en Kretzschmar (1984) is de verdeling van menskracht tijdens de life cycle van een softwareprodukt voor kleine projecten afwijkend.

Net zoals bij COCOMO en PRICE-SP is voor het model van Putnam onderzocht inhoeverre het tegemoet komt aan de eisen uit de beoordelingsmethode.

MODEL-EISEN:
- *Gekoppeld aan beheersingsmethode.*
  Aan deze eis wordt niet voldaan.
- *Vroeg toepasbaar.*
  Om dezelfde reden als de twee vorige modellen lijkt dit model niet geschikt om in een vroeg stadium van software-ontwikkeling te gebruiken.
- *Evoluerend.*
  Aan deze eis wordt niet voldaan.
- *Aanpassen aan doelstellingen.*
  Uit de beschrijving van het model blijkt dat in bescheiden mate aan deze eis wordt voldaan. Zo is het mogelijk na te gaan wat het effect is van het inkorten van de ontwikkeltijd en van het wijzigen van de toegewezen capaciteiten.
- *Toepasbaarheid.*
  Hierover worden geen duidelijke uitspraken gedaan.

TOEPASSINGS-EISEN:
- *Calibratie.*
  Het model biedt geen mogelijkheden voor calibratie. De gebruiker kan niet aantal en soort cost drivers veranderen. Het model is hoofdzakelijk gebaseerd op de invloed van de cost drivers omvang en beperkingen wat betreft doorlooptijd. De invloed van andere cost drivers wordt niet meegenomen in de schattingen. Het model blijkt erg gevoelig te zijn voor de variabele "technologie constante". In tegenstelling tot de aannamen in het model, blijkt de technologie constante te variëren met de omvang van het project.
- *Nauwkeurigheid.*
  Er is een gebrek aan empirische gegevens om hier een uitspraak over te doen. Toetsing van het model door Conte, Shen en Dunsmore (1986) en door Wingfield (1982) resulteerde in slechte begrotingsresultaten.

IMPLEMENTATIE-EISEN:
- *Gebruiksvriendelijkheid.*
  Het model is goed gedocumenteerd. Bovendien is het model geïmplementeerd in een geautomatiseerde versie onder de naam SLIM (Software LIfe-cycle Methodology) (Putnam 1980).
- *Gevoeligheidsanalyse en risicofactor.*
  Hierin wordt niet voorzien.
- *Inzichtelijkheid en eenduidigheid invoer.*
  Het model is in de literatuur uitvoerig beschreven (Putnam 1978 en 1980, Putnam en Fitzsimmons 1979, Londeix 1987). Het achterliggende concept en de algoritmen zijn openbaar. Het is daarom mogelijk de begrotingsresultaten te verklaren. De begrippen die in

het model worden gebruikt, zijn goed gedefinieerd.
- *Volledigheid en detail van uitvoer.*
  De uitvoer van het model is beperkt tot een schatting van de ontwikkelkosten, -tijd en -inspanning (uitgesplitst naar soorten personeel). De geautomatiseerde versie van het model (SLIM) geeft uitgebreidere uitvoermogelijkheden (o.a. een verdeling van kosten en inspanning over de fasen).

4.5. DE FUNKTIE PUNT ANALYSE METHODE

Het kenmerkende van de Funktie Punt Analyse methode is dat bij de bepaling van de omvang van de programmatuur niet het aantal regels code als uitgangspunt wordt genomen maar de hoeveelheid "funkties" die de software dient te vervullen.
De methode is ontwikkeld door Albrecht van IBM, in samenwerking met de gebruikersgroepen GUIDE en SHARE. In 1979 (Albrecht 1979) verscheen de eerste publicatie over deze aanpak. Albrecht zocht naar mogelijkheden om de productiviteit bij software-ontwikkeling te meten. Voor dat doel ontwikkelde hij de Funktie Punt Analyse (FPA) methode als een alternatief voor het schatten van aantal regels code. De methode is niet gericht op een specifieke programmeertaal of een applicatiegenerator (de Kater 1985). De methode is in de loop der jaren verfijnd (Rudolph 1983, 1983a, Albrecht en Gaffney 1983, Symons 1988). Door middel van een groot aantal publicaties is er een grote bekendheid aan gegeven (ondermeer de FPA proceedings 1988). De werking van de methode is in principe eenvoudig en is gebaseerd op het aantal "functies" dat een programma dient te vervullen, in termen van de typen/soorten gegevens die het systeem gebruikt en genereert. Zo wordt bij deze methode de te ontwikkelen software getypeerd aan de hand van de volgende vijf funkties:

- Invoer : het aantal verschillende typen invoergegevens,
- Uitvoer : de verschillende soorten output,
- Bestanden : het aantal benodigde permanente bestanden en hulpbestanden,
- Triggers : stuurgrootheden voor de uitvoering van een gegevensverwerkend programma,
- Interfaces : aantal en soort samenhangen met overige programma's, waaronder de noodzakelijke gegevensuitwisseling.

Van elke gebruikersfunctie die in de te ontwikkelen software wordt onderscheiden, wordt de moeilijkheidsgraad bepaald. Deze kan eenvoudig, gemiddeld of complex zijn. Afhankelijk van de moeilijkheidsgraad krijgt de gebruikersfunktie een waarde. Deze waarden zijn empirisch bepaald. In tabel 4.5 is een overzicht gegeven van de waarderingen per gebruikersfunktie.

Tabel 4.5 : Waarderingen voor de vijf gebruikersfuncties in relatie tot de moeilijkheidsgraad van het te ontwikkelen programma.

GEBRUIKERSFUNKTIE	EENVOUDIG	GEMIDDELD	COMPLEX
invoer	3	4	6
uitvoer	4	5	7
bestanden	7	10	15
triggers	3	4	6
interfaces	5	7	10

De volgende stap is het toekennen van funktiepunten aan het te ontwikkelen programma op basis van deze tabel. Een voorbeeld. Stel dat men voor de te ontwikkelen software vijf soorten uitvoer verwacht, waarvan er twee als eenvoudig, twee als gemiddeld en een als complex worden ingeschat. Het totaal aantal funktiepunten voor de gebruikersfunktie "uitvoer" wordt dan

$$4 + 4 + 5 + 5 + 7 = 25$$

In een verdere verfijning van de methode introduceert Rudolph hulpmatrices, waarmee de waarden van de gebruikersfunkties nauwkeuriger kunnen worden bepaald. In deze matrices wordt aangegeven wat verstaan moet worden onder een eenvoudige, gemiddelde en complexe gebruikersfunktie. Zo is dit voor de funktie uitvoer afhankelijk van het aantal velden en het aantal benodigde bestanden. In tabel 4.6 is een voorbeeld van zo'n hulpmatrix weergegeven.

Tabel 4.6 : Hulpmatrix voor het klassificeren van de gebruikersfunktie uitvoer (Craenen 1984).

AANTAL VELDEN				
1-5	6-19	>19		
eenvoudig	eenvoudig	gemiddeld	1	AANTAL BESTANDEN
eenvoudig	gemiddeld	complex	2-3	
gemiddeld	complex	complex	>3	

De samenhang tussen de wegingsfactoren uit tabel 4.5 en de classificatie met de hulpmatrix uit tabel 4.6 wordt toegelicht aan het geïntroduceerde voorbeeld. Stel dat de vijf soorten uitvoer als volgt geclassificeerd worden:

```
uitvoer 1 : 2  velden , 1 bestand   dus categorie : eenvoudig en weging = 4
uitvoer 2: 5  velden , 2 bestanden dus categorie : eenvoudig en weging = 4
uitvoer 3: 30 velden , 1 bestand   dus categorie : gemiddeld en weging = 5
uitvoer 4: 8  velden , 3 bestanden dus categorie : gemiddeld en weging = 5
uitvoer 5: 9  velden , 7 bestanden dus categorie : complex   en weging = 7
Het totaal aantal funktiepunten is 25.
```

Voor alle gebruikersfunkties wordt op deze wijze het aantal funktiepunten geschat. Door het optellen van deze aantallen krijgt men het bruto aantal funktiepunten voor een programma. De volgende stap is het omzetten van dit brutototaal naar het zogenaamde netto werkprodukt (netto aantal funktiepunten). Dit doet men door het brutototaal te vermenigvuldigen met een korrektiefactor. In deze faktor wordt de invloed verwerkt die veertien cost drivers kunnen hebben op de complexiteit van het softwareproject en dientengevolge op de benodigde inspanning. Gevraagd wordt de invloed van deze cost drivers in te schatten, waarbij van de onderstaande schaal gebruik moet worden gemaakt.

- niet van toepassing of geen invloed = 0
- af en toe van invloed = 1
- matige invloed = 2
- normale invloed = 3

- belangrijke invloed = 4
- zwaarwegende invloed = 5

In tabel 4.7 wordt een overzicht gegeven van de cost drivers die in de methode worden onderscheiden.

Tabel 4.7 : Overzicht van de cost drivers in de FPA (van Straten 1987).

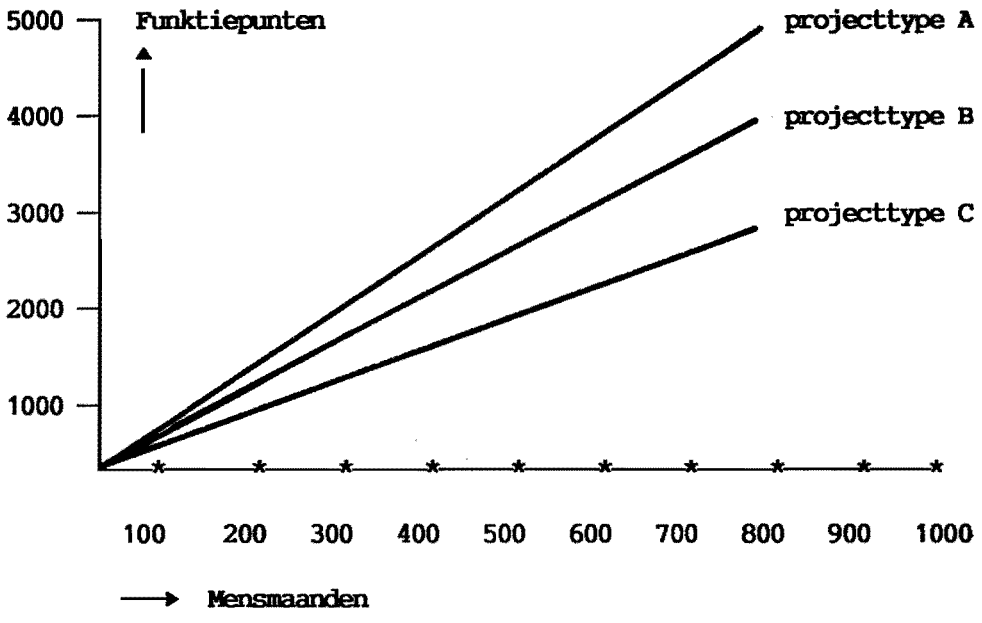
1.	invoer/uitvoer via datacommunicatie
2.	systeem maakt gebruik van "distributed data processing"
3.	eisen te stellen aan responsietijd
4.	eisen te stellen aan geheugengebruik en gebruik processing time
5.	aantal transakties
6.	het informatiesysteem werkt met on-line data entry
7.	invoer/uitvoer is conversationeel
8.	bestanden/databank worden on-line bijgewerkt
9.	complexiteit van het informatiesysteem
10.	mate van hergebruik
11.	complexiteit conversie en ingebruikname
12.	bedieningsgemak
13.	geinstalleerd op meerdere plaatsen
14.	aanpasbaarheid van het systeem.

De cost drivers kunnen tesamen een invloed hebben van minimaal 0 * 14 en maximaal 5 * 14. De korrektiefaktor wordt berekend op basis van de formule 0,65 + (totale beïnvloeding)/100. Het netto aantal funktiepunten wordt nu als volgt bepaald:

Netto aantal funktiepunten = brutototaal * korrektiefactor.

De laatste stap die gezet moet worden is de omrekening van netto aantal funktiepunten naar inspanning. Hierin wordt binnen FPA niet voorzien.
Om deze vertaalslag te maken, zal een organisatie van alle softwareprojecten moeten registreren uit hoeveel funktiepunten de gerealiseerde programmatuur bestaat en hoeveel inspanning hiervoor nodig is geweest. Op deze wijze beschikt een organisatie na verloop van tijd over de mogelijkheid om op basis van historische cijfers uitspraken te doen als een funktiepunt kost x uren werklast, een analist met die ervaring produceert per dag y funktiepunten. Bij voldoende grote aantallen historische projecten is het voor een organisatie mogelijk

verbanden zoals weergegeven in figuur 4.4 voor verschillende soorten projecten te realiseren.



Figuur 4.4: grafieken voor het omrekenen van funktiepunten (FP) naar mensmaanden (MM).

EVALUATIE FUNKTIE PUNT ANALYSE.

Zoals uit de enquete in hoofdstuk 2 blijkt is de Funktie Punt Analyse methode in Nederland het meest toegepaste begrotingsmodel. Van de 51 organisaties die bij het begroten mede gebruik maken van een begrotingsmodel zijn er 45 FPA gebruikers. Een verdere analyse van de enqueteresultaten laat zien dat het merendeel van deze gebruikers (55%) terug te vinden is in grote organisaties met een grote afdeling automatisering. Verder blijkt FPA het meest te worden toegepast bij het begroten van kleine projecten (48%). 82% van de projecten die met behulp van FPA worden begroot, liggen in de administratieve sfeer. Door de FPA gebruikers wordt consequent geregistreerd (96%) en in mindere mate nagecalculeerd (73%). Bij het analyseren van de

enqueteresultaten heb ik de volgende hypothese getoetst: organisaties die gebruik maken van FPA hebben lagere budgetoverschrijdingen dan organisaties die niet gebruik maken van FPA. Om dit na te gaan ben ik, op de zelfde wijze als in hoofdstuk 2, uitgegaan van die organisatie die begroten, registreren en nacalculeren. In totaal zijn er 160 organisaties die hieraan voldoen. Van deze 160 organisaties maken er 32 gebruik van FPA. Vervolgens heb ik de budgetoverschrijdingen van deze 32 FPA gebruikers vergeleken met de budgetoverschrijdingen van de niet-FPA gebruikers van de geselecteerde 160 organisaties. In tabel 4.8 is voor de 32 FPA en 128 niet-FPA gebruikers een overzicht gegeven van de mate van budget-overschrijdingen, uitgesplitst naar kleine, middelgrote, grote en zeer grote projecten. De 128 niet FPA-gebruikers worden in de tabel aangeduid met OVERIGE. Om aan te geven hoeveel organisaties tot de verschillende categorieën behoren, is per categorie het aantal organisaties vermeld. Bijvoorbeeld: Er zijn 24 organisaties die FPA gebruiken voor het begroten van kleine projecten. Van deze 24 organisaties geeft 25% te kennen geen budget-overschrijdingen te hebben, 41,7% hebben overschrijdingen op het budget van minder dan 10%, enz. Budgetoverschrijdingen van meer dan 50% komen niet voor.

Tabel 4.8 : Budgetoverschrijdingen bij de 32 FPA gebruikers en niet-FPA gebruikers.

	KLEINE PROJECTEN		MIDDELGROTE PROJECTEN		GROTE PROJECTEN		ZEER GROTE PROJECTEN	
	FPA	OVERIGE	FPA	OVERIGE	FPA	OVERIGE	FPA	OVERIGE
0%	25,0%	32,8%	-	15,9%	5,6%	11,3%	9,5%	19%
< 10%	41,7%	46,7%	43,5%	35,5%	22,1%	38,7%	23,8%	28,6%
10 t/m 50%	33,3%	18,0%	56,5%	41,1%	61,1%	41,9%	42,9%	28,6%
50 t/m 100%	-	2,5%	-	4,7%	5,6%	3,2%	23,8%	14,3%
> 100%	-	-	-	2,8%	5,6%	4,8%	-	9,5%
TOTAAL %	100%	100%	100%	100%	100%	100%	100%	100%
aantal organisaties	24	122	23	107	18	62	21	21

Met behulp van de Wilcoxon Rank Sum W test is getoetst of de budgetoverschrijdingen bij FPA gebruikers geringer is dan bij niet FPA gebruikers. Na toetsing bleek dat de hypothese verworpen moest worden. Een nadere analyse liet zien dat er een significant negatief

verband is tussen het gebruik van FPA en budgetoverschrijdingen. Dit betekent dat FPA gebruikers hogere budgetoverschrijdingen hebben dan niet FPA-gebruikers. Een verdere analyse gaf aan dat dit verband uitsluitend geldt voor grote projecten. Een mogelijke verklaring voor dit teleurstellend resultaat is, dat FPA voornamelijk gebruikt wordt voor het begroten van grote projecten, die, zoals gesignaleerd in hoofdstuk 2, duidelijk grotere budgetoverschrijdingen kennen dan kleine projecten. Na toetsing bleek deze veronderstelling inderdaad te kloppen. Organisaties die FPA gebruiken, passen deze methode significant meer toe bij het begroten van grote projecten. Een andere mogelijke verklaring wordt gegeven door De Haas (1988) en Rabbers (1988). Zij wijzen erop dat de invoering en het gebruik van FPA geen sinecure is. Veelal kost het een aantal jaren voordat het gebruik van FPA vruchten begint af te werpen. Volgens de Haas en Rabbers wordt dit door veel FPA gebruikers niet onderkend en wordt de methode vaak onzorgvuldig toegepast.

Voor Funktie Punt Analyse is onderzocht in hoeverre voldaan wordt aan de eisen uit de beoordelingsmethode.

MODEL-EISEN:

- *Gekoppeld aan beheersingsmethode.*
  FPA is op geen enkele wijze gekoppeld aan een beheersingsmethode.
- *Vroeg toepasbaar.*
  FPA is in tegenstelling tot de eerder besproken modellen niet gebaseerd op het aantal regels code als maat voor de omvang. Een gevolg hiervan is dat FPA in een vroeg stadium van het softwareproject gebruikt kan worden.
- *Evoluerend.*
  Uitgangspunt bij FPA is dat de gegevens en het inzicht die gaande het project toenemen, geen additionele informatie meer toevoegen aan een eenmaal opgestelde begroting.
- *Aanpassen aan doelstellingen.*
  Hierin voorziet FPA niet.
- *Toepasbaarheid.*
  Het model is uitsluitend toepasbaar voor het begroten van administratieve toepassingen. Voor rekenintensieve applicaties en voor embedded software, die veelal gekenmerkt worden door weinig invoer, weinig bestanden en weinig uitvoer, is deze aanpak niet bruikbaar.

TOEPASSINGS-EISEN:

- *Calibratie.*
  Het model biedt geen (actieve) ondersteuning voor calibratie. De waarden in de FPA matrices en aantal en aard van de cost drivers kunnen door de gebruiker bijgesteld worden. Net zoals bij COCOMO betekent calibratie hier een langdurig en moeizaam proces van bijstellen.
- *Nauwkeurigheid.*
  Evenals bij de eerder besproken modellen is het ook voor FPA lastig voldoende empirische gegevens te verzamelen om over de nauwkeurigheid van het model betrouwbare uitspraken te doen. Bemelmans (1987) merkt terecht op dat de aanpak op het eerste gezicht meer lijkt op een verzameling verstandige vuistregels dan op een theoretisch goed onderlegd model.

IMPLEMENTATIE-EISEN:

- *Gebruiksvriendelijk.*
  De aanpak van het model is eenvoudig. Bovendien worden er talloze FPA-cursussen georganiseerd, is het model goed gedocumenteerd en in vele artikelen uitvoerig beschreven en geanalyseerd (Craenen 1984, de Kater 1985, van Straten 1987 en 1988, Noth en Kretzschmar 1984, Albrecht en Gaffney 1983, Vonk 1985, NGI 1988, Albrecht 1984, FPA-proceedings 1988). Een groot aantal andere modellen, waaronder ESTIMACS (Rubin 1985), Before You Leap (1986) en ESTIMATE/1 (1987), zijn gebaseerd op een "funktiepuntachtige" benadering. Van FPA zijn verschillende geautomatiseerde versies in omloop. Van FPA zijn meerdere varianten in omloop. Als een variant van FPA heeft Symons (1988) het "MARK II" Function Point model ontwikkeld.
- *Gevoeligheidsanalyse en risicofactor.*
  Aan beide eisen wordt door FPA niet voldaan.
- *Inzichtelijkheid.*
  Doordat het model openbaar is, is het mogelijk de begrotingsresultaten te verklaren/na te trekken.
- *Eenduidigheid invoer.*
  Bij het tellen van de gebruikersfunkties blijkt in de praktijk een grote spreiding op te treden. Verder bestaat bij het waarderen van de gebruikersfunkties het gevaar dat subjectiviteit een rol speelt. Door gebruik te maken van de eerder genoemde hulpmatrices kan dit gevaar voor een deel ondervangen worden. Het gebruik van geavanceerde hulpmiddelen kan de wel erg versimplificeerde indeling in eenvoudig, gemiddeld en complex danig aantasten; een mogelijkheid waarin binnen FPA niet is voorzien. Bij het inschatten van de invloed van de 14 cost drivers speelt subjectiviteit een nog veel grotere rol.

- *Volledigheid en detail van uitvoer.*
  De uitvoer van FPA bestaat uit het aantal netto funktie punten dat nodig is om een programma te maken. FPA doet bijvoorbeeld geen uitspraken doet over de verdeling van funktiepunten en dus ook van uren, inspanning en kosten naar fasen c.q activiteiten. De uitvoer is beperkt.

4.6. HET MODEL VAN WALSTON EN FELIX

Walston en Felix (1977) zijn al in 1972 begonnen met een uitvoerig onderzoek naar factoren die van invloed zijn op de produktiviteit bij software-ontwikkeling. Tussen 1972 en 1977 zijn de gegevens van 60 voltooide softwareprojecten verzameld en opgeslagen in een databank. Gemiddeld over alle projecten vonden zij de onderstaande basisrelatie tussen inspanning en omvang (in KLOC):

$$\text{Inspanning} = 5.2*\text{KLOC}^{0,91}$$

Zoals ook het geval is in COCOMO en FPA moet deze basisvergelijking bijgesteld worden met een correctiefaktor. In deze correctiefaktor is de invloed van 29 productiviteitsbeïnvloedende factoren verwerkt. In tabel 4.9 zijn de meest dominante faktoren, in volgorde van belangrijkheid, met hun mogelijke invloed op de produktiviteit weergegeven. Voor elk van deze faktoren kunnen drie nivo's van beïnvloeding worden onderscheiden: hoog, gemiddeld en laag. De bijbehorende waarden zijn bepaald met behulp van de resultaten van een uitgebreide vragenlijst. De betekenis van tabel 4.9 zal toegelicht worden aan de hand van de eerste faktor uit de tabel. De geënqueteerde projectleider werd gevraagd voor zijn projecten aan te geven of de mens-machine interface erg, weinig of gemiddeld complex was. Vervolgens werd de gemiddelde produktiviteit bepaald van alle projecten met een lage complexiteit van de mens-machine interface. Deze produktiviteit bleek 500 regels code per mensmaand te zijn. Hetzelfde werd gedaan voor de projecten met een hoge en gemiddelde complexe mens-machine interface. De bijbehorende produktiviteitswaarden bleken 124 en 295 regels code per mensmaand te zijn. De verandering in produktiviteit bij een overgang van een lage naar een hoge complexiteit van de mens-machine interface is dus 500 - 124 = 376 regels code per mensmaand. Dit is de waarde in de laatste kolom.De eerder genoemde korrektiefaktor voor een specifiek project wordt als volgt bepaald:

$$\text{Korrektiefaktor} = \sum_{i=1}^{29} \text{Gewicht(i)}*\text{Antwoord(i)}$$

waarbij Gewicht(i) gedefinieerd wordt als:

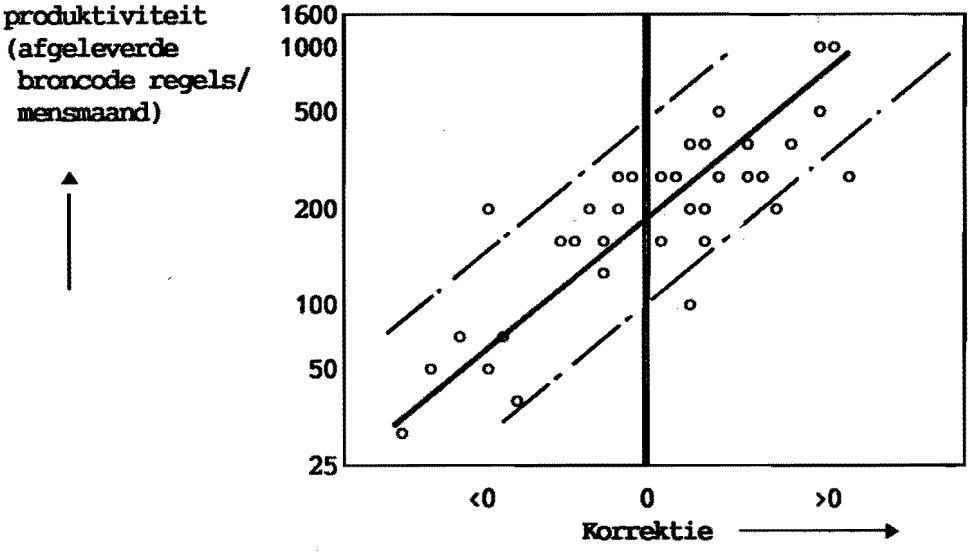
Gewicht(i) = 0,5log*[PV(i)]
PV(i) stelt voor de produktiviteitsverandering van faktor i.

en waarbij Antwoord(i) een waarde kan krijgen van -1, 0 en +1, afhankelijk of de betreffende faktor een lage, gemiddelde en hoge produktiviteit tot gevolg heeft.

Tabel 4.9 : De belangrijkste produktiviteitsbeinvloedende faktoren uit het model van Walston en Felix (1977). PV staat voor verandering in produktiviteit. Dit is |hoge productiviteit - lage produktiviteit|.

FAKTOR	GEMIDDELDE PRODUKTIVITEIT			PV
Complexiteit mens-machine interface	<normaal 500	normaal 295	>normaal 124	376
Deelname gebruikers bij definitie van eisen	geen 491	beetje 267	veel 205	286
Algehele ervaring en capaciteiten van projectmedewerkers	laag 132	gemiddeld 257	hoog 410	278
Ervaring met toepassingen van soortgelijke omvang	minimaal 146	gemiddeld 221	uitgebreid 410	264
Ervaring met benodigde programmeer talen	minimaal 122	gemiddeld 225	uitgebreid 385	263
% programmeurs dat deelnam in opstellen logische spec's en tevens codeerde	<25% 153	25-50% 242	>50% 391	238
Beperkingen voor programma ontwerp t.a.v. hoofdgeheugen	laag 391	gemiddeld 277	hoog 193	198
Werken met "chief programmer team"	0-33% 219	34-66% -	>66% 408	189
Complexiteit van de toepassing in het algemeen	laag 349	gemiddeld 345	hoog 168	181

Walston en Felix hebben deze korrektiefactor voor 51 projecten berekend. Een plot van de gerealiseerde produktiviteit voor elk project in vergelijking met de berekende korrektiefactor en een toepassing van de kleinste kwadratenmethode op deze relatie is weergegeven in figuur 4.5. De standaardfout van de schattingen is aangegeven als stippellijnen.



Figuur 4.5 : Relatie produktiviteit en korrektiefaktor.

In de beschrijving van het model (Walston en Felix 1977) wordt de bij deze relatie behorende vergelijkingen niet gegeven.
Andere verbanden die Walston en Felix door middel van analyses van de gegevens van de 60 voltooide softwareprojecten ontdekten en in vergelijkingen vastlegden, zijn:

- Relatie tussen documentatie en software-omvang: $D = 49*KLOC^{1.01}$ waarbij D = pagina's documentatie en KLOC = aantal regels code gedeeld door duizend.

- Relatie tussen projectduur en software-omvang: $PD = 4.1*KLOC^{0.36}$ waarbij PD = duur in mensmaanden en KLOC = aantal regels code gedeeld door duizend.

- Relatie tussen projectduur en inspanning: PD = $2.47*I^{0.35}$ waarbij PD = duur in mensmaanden en I = inspanning in mensmaanden.

- Relatie tussen gemiddelde personeelsbezetting en inspanning: P = $054*I^{0.6}$, waarbij P = gemiddeld aantal bij project betrokken personen en I = inspanning in mensmaanden.

Om het model van Walston en Felix te gebruiken om de benodigde inspanning voor een nieuw projekt te bepalen moeten de volgende stappen worden genomen.

1. Beantwoord de door Walston en Felix samengestelde vragenlijst om de gewichten van de 29 faktoren te bepalen.
2. Ken het cijfer +1, -1 of 0 toe aan elk antwoord afhankelijk of de faktor een positief, negatief of geen effekt heeft op de produktiviteit.
3. Bereken de korrektiefaktor.
4. Ga na met behulp van figuur 4.5 welke produktiviteitswaarde bij deze korrektiefaktor hoort. Hierover worden geen details gegeven in (Walston en Felix 1977).
5. Maak een schatting van het aantal te produceren regels code.
6. Bereken de benodigde inspanning met behulp van de eerder genoemde basisvergelijking.

EVALUATIE VAN HET MODEL VAN WALSTON EN FELIX.

Het model van Walston en Felix is vooral vanuit historisch perspectief van belang. Het is een uitgangspunt geweest voor later ontwikkelde modellen. Het model wordt heden ten dage in de praktijk nergens toegepast. Om deze reden heb ik de evaluatie ervan beperkt gehouden.

Het model van Walston en Felix is gebaseerd op gegevens van voltooide projecten, die onderling sterk verschillen in omvang, toepassingsgebied en gebruikte programmeertalen. De geldigheid van het model zou beter zijn indien de vergelijkingen gebaseerd zouden zijn op een uniformere verzameling van voltooide projecten. Het is de vraag wat twee volkomen verschillende softwareprojecten zoals het ontwikkelen van een eenvoudig batch geörienteerd voorraadprogramma en een uiterst complex programma voor geleide projectielen gemeen hebben. Alle onderzochte projecten zijn gerealiseerd in het begin van de zeventiger jaren. De beschikbaarheid van geautomatiseerde hulpmiddelen was in die tijd laag.

Van Vliet (1987) merkt op dat het aantal produktiviteits-bepalende faktoren erg groot is. Verder wordt niet duidelijk in hoeverre de verschillende faktoren elkaar wederzijds beïnvloeden. Zo zullen de vier faktoren "gestructureerd programmeren", "ontwerp en code inspectie", "gebruik chief programmer team" en "top down ontwikkelen" nauw met elkaar samenhangen. Het ligt voor de hand dat een projectleider van alle vier gebruik zal maken als hij een fervent voorstander is van gestructureerd programmeren. Het is daarom moeilijk te achterhalen welk deel van een stijging van de produktiviteit toegeschreven moet worden aan welke faktor of aan de combinatie van faktoren. Bovendien lijkt het aantal mogelijke waarden per faktor, namelijk drie, in de praktijk te weinig mogelijkheden te bieden (Conte, Shen en Dunsmore 1986).

Ondanks de genoemde bezwaren heeft het pionierswerk van Walston en Felix veel bijgedragen aan het onderzoek over het begroten van softwareprojecten.

4.7. EEN AANTAL OVERIGE MODELLEN

In de afgelopen 15 jaren is een groot aantal modellen ontwikkeld. Van een vijftal modellen is hierboven een uitvoerige beschrijving en een kritische analyse gegeven. In deze paragraaf wordt een korte beschrijving gegeven van de volgende, recente begrotingsmodellen:

- Before You Leap (BYL)
- Estimacs
- SPQR-20
- BIS-Estimator
- SOFCOST
- het model van Jensen
- het model van DeMarco

De reden waarom voor deze modellen is gekozen is dat ik met de vier eerst genoemde modellen ruime ervaring heb opgedaan, en dat deze door het researchteam "Begroten van Automatiseringsprojecten" zijn getest en geanalyseerd (BYL, Estimacs, SPQR-20, BIS-estimator). De beschrijvingen zullen zich beperken tot het geven van de belangrijkste kenmerken van de modellen. De overige drie modellen zijn opgenomen omdat deze in de literatuur over begrotingsmodellen en in de praktijk van begroten een steeds belangrijkere plaats gaan innemen.

Van de eerste vier modellen zal worden onderzocht in welke mate zij voldoen aan de eisen genoemd in de beoordelingsmethode. Deze evaluaties zullen beknopt zijn. Een uitvoerige beschrijving ervan is te vinden in Heemstra, van Genuchten en Kusters (1988 en 1988a). Er volgt nu eerst een beschrijving van de modellen BYL, Estimacs, SPQR-20 en BIS-Estimator, vervolgens de evaluaties van deze modellen en tenslotte een beschrijving van de overige modellen.

4.7.1. HET MODEL BEFORE YOU LEAP

Het model Before You Leap (BYL) is ontwikkeld door de Gordon Group (BYL 1986) en is onder dezelfde naam geïmplementeerd in een geautomatiseerde versie. BYL maakt bij het begroten gebruik van een integratie van Funktie Punt Analyse en COCOMO. Met behulp van FPA wordt het netto aantal funktiepunten van het te ontwikkelen systeem bepaald. Vervolgens vindt er een omrekening plaats van het aantal funktiepunten naar het aantal regels code. Bij deze omrekening is de keuze van de programmeertaal de bepalende factor. Het model biedt de mogelijkheid te kiezen uit 48 talen. Bij elke taal behoort een omrekeningscoëfficiënt. Voor COBOL is deze 105, voor LISP 64 en voor APL 32. De omrekening gebeurt met behulp van de volgende formule:

KDSI = (aantal funktiepunten) * (coefficient) / 1000

De waarde voor omvang in KDSI dient als invoer voor COCOMO. Door het uitvoeren van de verschillende stappen van COCOMO wordt een schatting van kosten en ontwikkeltijd gemaakt. De parameterwaarden in het model zijn gebaseerd op meer dan 50.000 mensjaren aan projectervaring. Hoewel het idee van een combinatie van FPA als schatter voor de omvang en COCOMO als schatter van de invloed van cost drivers goed is, roept de toepassing ervan in het model BYL vragen op. Zo wordt het, via FPA bepaalde, aantal netto funktiepunten omgerekend is aantal regels code en als waarde van de omvang genomen bij het bepalen van de nominale inspanning. Deze transformatie is niet juist, omdat binnen FPA door middel van de correctiefactoren al een aanpassing heeft plaatsgevonden. Op deze wijze wordt de nominale inspanning voor sommige factoren twee maal aangepast. Een voorbeeld hiervan is de factor "eisen wat betreft responsiesnelheid". Deze factor wordt zowel binnen FPA als COCOMO meegenomen als onderdeel van de correctiefactor. Op deze wijze krijgen een aantal factoren een meer dan evenredig gewicht. Om deze dubbele aanpassing te vermijden is men genoodzaakt de invloed van de betreffende variabele in of het FPA deel of het COCOMO deel niet

mee te tellen.

4.7.2. HET MODEL ESTIMACS

Het model ESTIMACS is ontwikkeld door H. Rubin van het Hunter College (Rubin 1983, 1984 en 1985, Computer Associates 1986) en is als geautomatiseerde versie beschikbaar. Het model is gebaseerd op een funktiepuntachtige benadering. Binnen ESTIMACS worden de volgende negen modules onderscheiden:

- *Effort*: voor het bepalen van het aantal benodigde mensmaanden voor ontwikkeling en onderhoud,
- *Staffing en costs*: voor het bepalen van het aantal en de kwaliteit van de benodigde medewerkers per fase en voor het bepalen van de kosten en doorlooptijd,
- *Hardware*: voor het bepalen van de vereiste hardware faciliteiten,
- *Risk Analysis*: voor het uitvoeren van risico analyses,
- *Portfolio*: voor het optimaliseren van het beslag op middelen bij het uitvoeren van meerdere projecten,
- *Maintenance*: voor het bepalen van de onderhoudsinspanning,
- *Financial analysis*: voor onder andere het bepalen van de maande lijkse cashflows, totale cashflow, break even point,
- *Function Point*: voor het bepalen van het aantal funktiepunten,
- *Small Project* : voor het begroten van kleine projecten.

De belangrijkste en omvangrijkste module is Effort. De modelgebruiker moet bij toepassing van deze module 25 vragen beantwoorden. Deze vragen hebben deels betrekking op de complexiteit van de betrokken organisatie(s) en deels op de complexiteit en omvang van de te ontwikkelen applicatie zelf. Het model vraagt niet om het aantal regels code als maat voor de omvang, maar gaat uit van een funktie-punt-achtige benadering. Hoe ESTIMACS op basis van de antwoorden tot een voorspelling van het aantal mensmaanden komt is niet duidelijk. De onderliggende modelvergelijkingen zijn niet toegankelijk.

4.7.3. HET MODEL SPQR

SPQR staat voor Software Productivity, Quality en Reliability. Het model is ontwikkeld door C. Jones (1984). SPQR pretendeert toepas-

baar te zijn voor het totale spectrum van softwareprojecten. Behalve het schatten van tijd, kosten en inspanning om een programma te ontwikkelen, begroot het model ook onderhoudskosten. Funktie Punt Analyse wordt binnen SPQR gebruikt als hulpmiddel bij het schatten van de omvang van de te ontwikkelen software in regels code. Het model is gebaseerd op een omvangrijke databank van voltooide projecten. Er bestaan een viertal geautomatiseerde versies van het model, te weten SPQR 10, 20, 50 en 100 (de getallen staan voor het aantal vragen die door de modelgebruiker beantwoord moeten worden en zijn een indicatie voor de detaillering van de verschillende versies). Alleen het model 20 is commerciëel verkrijgbaar.

4.7.4. HET MODEL BIS-ESTIMATOR

BIS-Estimator is een begrotingsmodel dat op een volkomen andere wijze in elkaar zit dan de eerder beschreven modellen. Volgens de documentatie (Bis 1987) gaat het om een "knowledge-based tool". Met zekerheid kan dit door mij niet gesteld worden, omdat de werking van het model grotendeels geheim is. De wijze van benadering is volkomen afwijkend van die van de drie hiervoor genoemde modellen. Er wordt op (veel te) eenvoudige wijze een schatting gemaakt van de benodigde inspanning en doorlooptijd van het totale project. Dit is een zogenaamde "zachte" schatting. Daarna wordt per fase een zogenaamde "harde" schatting gemaakt. Op basis van de schattingsresultaten per fase wordt, door middel van een soort extrapolatie, een nieuwe schatting verkregen van het totale project. Het is de bedoeling dat de harde schatting van een fase pas gemaakt wordt tijdens of zelfs na de voorafgaande fase. Tenslotte geeft het model nog de mogelijkheid voor het maken van een begroting op basis van een directe vergelijking met een aantal door de gebruiker te selecteren afgesloten projecten. De gebruiker moet voor het huidige project dan aangeven hoe het qua omvang gerelateerd moet worden met de historische projecten. Het model geeft schattingen voor doorlooptijd vanaf de fase "feasibility" tot en met de fase "implementation". Binnen elke fase worden weer een groot aantal activiteiten onderscheiden, waarin in ieder geval management en documentatie een prominente plaats krijgen.

4.7.5. EVALUATIE VAN DE MODELLEN BYL, ESTIMACS, SPQR-20 EN BIS-ESTIMATOR

Zoals aan het begin van deze paragraaf is aangekondigd, volgt er nu een beknopte evaluatie van de vier hierboven beschreven modellen. Per eis van de beoordelingsmethode wordt aangegeven in welke mate de modellen eraan voldoen.

MODEL-EISEN:

- *Gekoppeld aan beheersingsmethode.*
  BYL, BIS-Estimator en SPQR-20 gaan alle uit van een eigen vaste faseverdeling. Het is voor de gebruiker niet mogelijk hier iets aan te wijzigen. Estimacs ondersteunt elke fasering die de gebruiker wenst in te brengen. Verder ondersteunt dit model benaderingen als prototyping en pakketselectie.
- *Vroeg toepasbaar.*
  BYL, Estimacs en SPQR-20 zijn alle drie gebaseerd op een FPA-achtige benadering. Een en ander betekent dat deze modellen in een redelijk vroeg stadium van software-ontwikkeling toegepast kunnen worden. Dit geldt het meest voor Estimacs. De vragen die dit model stelt zijn zodanig dat ze alle gemakkelijk aan het begin van een project beantwoord kunnen worden. BIS-Estimator is er niet op gericht in het begin van het project een begroting voor het totaal te maken.
- *Evoluerend.*
  BIS-Estimator houdt als enig model rekening met dit aspect. De begroting wordt per fase van de life cycle opgezet. In elke fase worden nieuwe vragen gesteld die expliciet rekening houden met het soort kennis dat, tot aan deze fase, verzameld zou moeten zijn.
- *Aanpassen aan doelstellingen.*
  In BYL en BIS-Estimator is dit niet mogelijk. SPQR-20 ondersteunt expliciet de mogelijkheid dat verschillende doelstellingen actueel zijn (bijvoorbeeld: laagste kosten, met extra personeel; kortste doorlooptijd en maximale kwaliteit; en nog vele andere). Estimacs voldoet in bescheiden mate aan deze eis. Het geeft hierbij de volgende mogelijkheden: beperkingen in doorlooptijd: beperkingen in de hoeveelheid beschikbaar personeel.
- *Toepasbaarheid.*
  De modellen BYL, SPQR-20 en Estimacs beweren geschikt te zijn voor alle soorten programma's. BIS-Estimator is expliciet gericht op administratieve projecten en beperkt zich ook bewust hiertoe.

TOEPASSINGS-EISEN:

- *Calibratie.*
  Bij BYL is calibratie alleen mogelijk op het eindresultaat. Dit wil zeggen dat na afloop van het project de werkelijk benodigde inspanning en doorlooptijd ingevoerd kunnen worden. Door deze waarden te confronteren met de geschatte inspanning en doorlooptijd berekent BYL een correctiefactor voor toekomstige schattingen. Ook kunnen de COCOMO cost drivers veranderd worden alsmede de gewichten. De enige mogelijkheid voor calibratie die Estimacs geeft is dat de produktiviteit per fase aangepast kan worden. BIS-Estimator ondersteunt de mogelijkheden tot calibratie uiterst beperkt. Bij SPQR-20 is het vrijwel niet mogelijk het model te calibreren.
- *Nauwkeurigheid.*
  Van alle modellen zijn te weinig empirische gegevens beschikbaar om hier betrouwbare uitspraken over te doen. BYL claimt accuraat te zijn binnen 20% met een betrouwbaarheid van 68% en binnen 30% met een betrouwbaarheid van 90%. BIS-Estimator geeft geen informatie betreffende de nauwkeurigheid. De voorspellingen worden gegeven in de vorm van een onder- en bovengrens. Ditzelfde doet ook Estimacs. De leverancier claimt een nauwkeurigheid van +/- 15%. Hiervoor zijn nog geen bevestigingen in de literatuur te vinden. SPQR-20 geeft in haar documentatie geen vermelding van de nauwkeurigheid. In paragraaf 4.9 wordt een aantal onderzoeken aangehaald waarin een aantal modellen (waaronder ook Estimacs) op het aspect nauwkeurigheid worden beoordeeld.

IMPLEMENTATIE-EISEN:

- *Gebruiksvriendelijk.*
  Alle vier modellen zijn beschikbaar in een geautomatiseerde versie. Vooral het model BYL springt eruit wat betreft gebruiksvriendelijkheid. Dit geldt in iets mindere mate voor SPQR-20 en Estimacs en in nog mindere mate voor BIS-Estimator. In dit laatste model zijn een aantal vragen opgenomen die niet of nauwelijks te begrijpen zijn, ook na raadpleging van de documentatie.
- *Gevoeligheidsanalyse.*
  De modellen BYL en Estimacs ondersteunen het uitvoeren van gevoeligheidsanalyses. Deze faciliteit worden door BIS-Estimator en SPQR-20 ook geboden. Het uitvoeren van what/if analyses gaat echter niet snel en is niet eenvoudig.
- *Risico-analyses.*
  De modellen BYL en BIS-Estimator geven geen indicatie van het bij het project behorende risico. Estimacs beschikt over een gedegen module voor risico-analyse. Bij SPQR-20 wordt een schatting gegeven van de kans dat het project volledig mislukt. Er is echter niets bekend over de wijze waarop de kans wordt bepaald en over

de factoren die deze risicokans bepalen.

- *Inzichtelijkheid.*
  De inhoud van de modellen Estimacs en SPQR-20 zijn geheim. De produktieregels van BIS-Estimator worden niet verborgen gehouden. Op elk moment kan men opvragen welke regels gebruikt zijn. Het aantal regels en de wijze van presentatie zijn echter niet bevorderlijk voor de begrijpelijkheid van het model. Het model is wel open, maar de gebruiker moet er echter veel tijd en moeite in investeren om de werking ervan te doorgronden. BYL is een volkomen open model en de resultaten zijn daarom volledig verklaarbaar.
- *Eenduidigheid invoer.*
  Voor BYL, Estimacs, SPQR en in mindere mate voor BIS geldt dat weliswaar niet alle invoervariabelen meetbaar zijn maar dat de definities van deze variabelen zo duidelijk zijn dat de modelgebruiker op dit punt geen problemen zal ondervinden.
- *Volledigheid en detail van uitvoer.*
  Alle modellen geven allerlei uitvoer-mogelijkheden en voldoen meer dan voldoende aan deze eis.

4.7.6. HET MODEL SOFTCOST

Het model SOFTCOST is ontwikkeld door Tausworthe (1981) van het JET Propulsion Laboratorium en verder verfijnd door Reifer Consultants. Het uitgangspunt was om in dit model het beste van een aantal bestaande modellen te verenigen. Zo wordt in het model ondermeer gebruik gemaakt van de 29 factoren van Walston en Felix en van de 7 factoren uit het GRC model (Carriere en Thibodeau 1979). Bovendien is het principe van het Putnam model gebruikt om de haalbaarheid van de geschatte verdeling van menskracht in de tijd te toetsen. Het model bevat 68 parameters die betrekking hebben op zaken als produktiviteit, projectduur, ervaringsnivo medewerkers, documentatie en hardware middelen. SOFTCOST is verkrijgbaar in de vorm van een softwarepakket. Dit pakket stelt de gebruiker 47 vragen. Uit de antwoorden herleidt het model waarden voor de parameters. De uitvoer van het model bestaat uit een begroting van de ontwikkelinspanning en -tijd, verdeeld over zelf te kiezen fasen. Helaas is SOFTCOST tot op heden niet getest met behulp van een uitgebreide databank. Bovendien kan gesteld worden dat het model onnodig complex is. Door factoren van verschillende modellen in een model te combineren, wordt over het hoofd gezien dat een aantal factoren uit beide modellen onderling sterk correleren.

4.7.7. HET MODEL VAN JENSEN

Dit model is ontwikkeld door R. Jensen en staat uitvoerig beschreven in een aantal artikelen van de ISPA conferenties (International Society of Parametric Analysts) (Jensen 1983, 1984, Jensen en Lucas 1983). Het model vertoont grote overeenkomsten met het model van Putnam, maar leunt ook sterk op de ideeën van COCOMO. In tegenstelling tot Putnam gaat Jensen ervan uit dat bij een verkleining van de gewenste doorlooptijd de kosten veel minder sterk zullen stijgen. Een aantal tekortkomingen in het model van Putnam worden door Jensen aangepakt. Zo wordt er in het model wel rekening gehouden met het effect van een aantal belangrijke cost drivers, waaronder personeelsafhankelijke factoren. Bij het onderkennen van de cost drivers is duidelijk de invloed van COCOMO te bespeuren. Het model is verkrijgbaar als het geautomatiseerd hulpmiddel JS-3. In Rampton (1984) wordt dit hulpmiddel uitvoerig beschreven.

4.7.8. HET MODEL VAN DEMARCO

Het bijzondere van de aanpak van DeMarco (1982 en 1984) is dat hij de omvang van een project bepaald op een volkomen andere wijze dan de eerder behandelde modellen. Zijn methode BANG bepaalt de "net usable funtion from the user's point of view" of zoals Van Vliet (1987) het noemt, het aantal functionele primitieven in de mens-machine interface van het systeem. Als basis voor de schatting van de omvang worden de dataflowdiagrammen uit de ontwikkelingsmethode Structured Systems Analysis van Gane en Sarson (1979) gebruikt. Deze diagrammen worden omgezet in functionele primitieven (FP). Een FP wordt door DeMarco omschreven als "a trivial piece, too small to justify further partitioning". Op dit laagste nivo onstaat een netwerk van aan elkaar gekoppelde funktionele primitieven. Het tellen van het aantal FP levert een maat voor de omvang. De ene FP kan echter omvangrijker en complexer zijn dan de andere. Vandaar dat er twee correctie slagen nodig zijn. Als maat voor de omvang van een FP wordt het aantal relaties (invoer en uitvoer) met andere FP genomen. Voor het bepalen van de complexiteit van een FP wordt elke FP ondergebracht in een complexiteitsklasse. Met elke klasse correspondeert een correctiefactor. De samenhang tussen omvang en functionele primitieven is nu als volgt weer te geven:

Omvang is een functie van: aantal FP, aantal relaties per FP, de correctiefactor per FP.

De relatie tussen het aantal FP en de inspanning is door DeMarco niet verder uitgewerkt maar berust evenals de eerder behandelde modellen op de vergelijking:

$$\text{inspanning} = a*(FP)^b$$

DeMarco geeft zelf aan dat zijn methode Bang primair van toepassing is voor procesgeörienteerde systemen. Voor software die sterk datagedreven zijn heeft hij een variant ontwikkeld. In een dergelijk geval dient men uit te gaan van de objecten in de database. Het gewicht van elk object wordt bepaald door het aantal relaties met andere objecten. Als maat voor de omvang wordt nu de som van de gewogen objecten genomen. Voor systemen die zowel proces- als datageörienteerd zijn stelt De Marco een combinatie voor van beide methoden. Een invulling hiervan wordt echter niet gegeven. Praktische toetsing ontbreekt vooralsnog, zodat de waarde van dit model onbekend is (Van Vliet 1987). Verner en Tate (1987) merken terecht op dat de functionele primitieven op zo'n detail nivo beschikbaar moeten zijn dat het te betwijfelen valt of de methode BANG is een vroeg stadium van systeemontwikkeling al toegepast kan worden. De aanpak is sterk gekoppeld aan de systeemontwikkelingsmethode Structured Systems Analysis.

4.8. OVERIGE MODELLEN

In deze paragraaf wil ik een overzicht geven van modellen die niet in dit onderzoek behandeld zijn. Dit overzicht heeft niet de pretentie volledig te zijn.

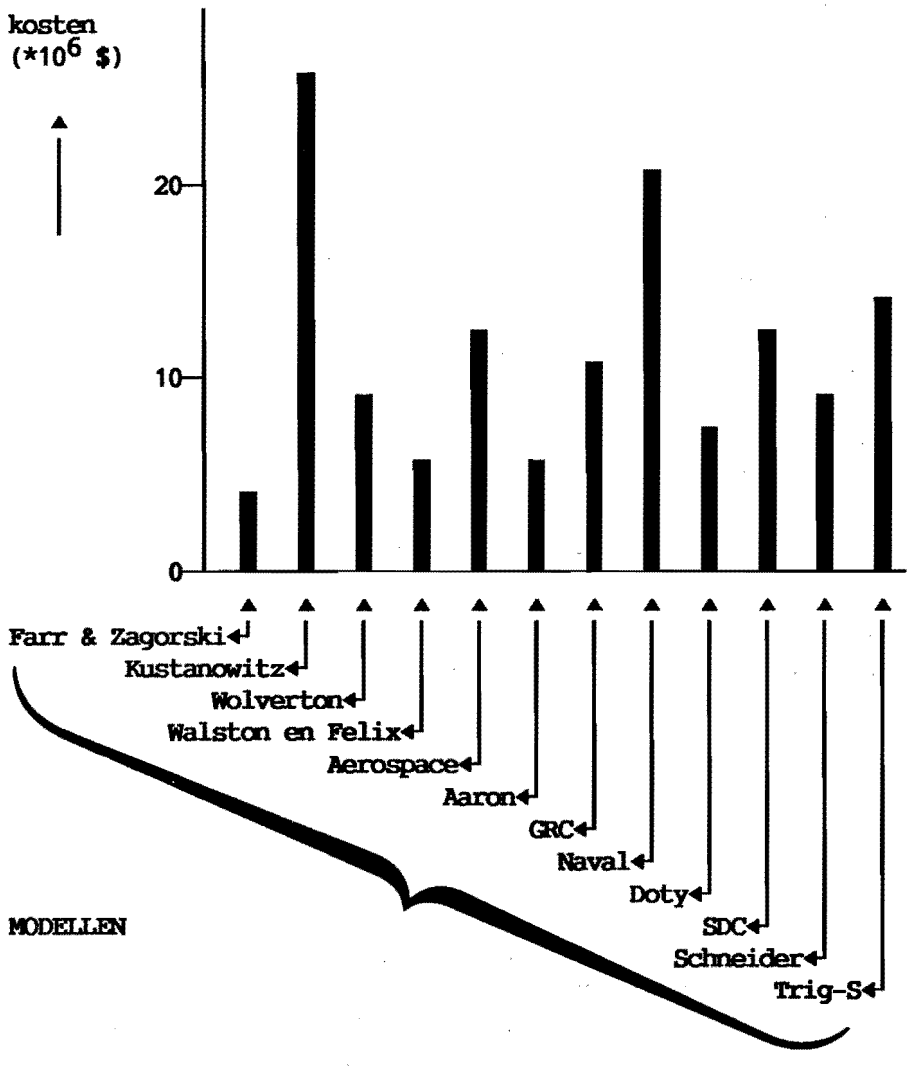
Zoals uit de enquete in hoofdstuk 2 blijkt, gebruikt een niet onaanzienlijk deel van de organisaties een zelf-ontwikkeld begrotingsmodel. Deze modellen zijn uiteraard niet in dit overzicht terug te vinden. Modellen die in de literatuur genoemd worden, zijn in tabel 4.10 gepresenteerd.

Tabel 4.10 : Overzicht van begrotingsmodellen.

NAAM MODEL	LITERATUUR
Het SDC model	(Nelson 1965)
Het TRW Wolverton model	(Wolverton 1974)
Het model TELECOTE	(Frederic 1974)
Het Boeing model	(Black c.s. 1977)
Het Doty model	(Herd c.s. 1977)
Het ESD1 model	(Duquette en Bourbon 1978)
Het model van Surbock	(Surbock 1978)
Het GRC model	(Carriere en Thibodeau 1979)
Het Slice model	(Kustanowitz 1980)
Het Baily-Basili Meta-model	(Baily en Basili 1981)
Het model FAST	(Freiman 1981)
Het model COPMO	(Thebaut en Shen 1984)
Het model ESTIMATE/1	(Arthur Anderson 1987)
het model van Jeffery	(Jeffery 1987)
Het Grumman model	(Sandler en Bachowitz)

4.9. MODELLEN VERGELEKEN

De verschillende besproken modellen wijken behoorlijk van elkaar af. Experimenten die zijn uitgevoerd, laten zien dat begrotingen voor een en hetzelfde project opgesteld met behulp van verschillende modellen onderling sterk verschillen. Bovendien wijken de begrotingen sterk af van de gerealiseerde ontwikkelkosten en -tijd. en leveren totaal verschillende uitkomsten. Praktisch geen enkel model is in staat om de ontwikkelkosten voor een geautomatiseerd systeem redelijk nauwkeurig in te schatten (Abdel-Hamid en Madnick 1987). Deze uitspraak is een bevestiging van een interessant onderzoek, uitgevoerd door Mohanty (1981). Hij onderzocht in hoeverre met verschillende modellen hetzelfde kostenbedrag begroot werd voor één gegeven softwareproject. Voor het definiëren van zijn hypothetisch project maakte hij gebruik van alle cost drivers, die in een of meerdere modellen voorkwamen (in totaal 49 cost drivers en 12 modellen). De begrotingen van de kosten zijn weergeven in figuur 4.6.



Figuur 4.6 : Kostenbegrotingen van één softwareproject met behulp van 12 verschillende modellen (Mohanty 1981).

Zoals uit de figuur blijkt is er een grote spreiding in de kosten; 362.500 dollar als laagste begroting en 2.766.667 dollar als hoogste bedrag. Mohanty geeft twee redenen voor deze verschillen. De eerste is terug te voeren tot de kwaliteit van het eindresultaat/de programmatuur. De tweede heeft betrekking op de omgevingsverschillen. Met dit laatste wordt bedoeld dat elk model gebaseerd is op gegevens van voltooide projecten die uitgevoerd zijn in een specifieke ontwikkelom-

geving.
Deze gegevens representeren enerzijds het specifieke karakter van die organisatie met zijn eigen werkmethoden, managementstijl en normen en standaarden en anderzijds de karakteristieken van de gerealiseerde projecten. Al deze kenmerken zijn in ingedikte vorm in de parameters van het model terug te vinden. Modellen zijn dus verschillend en geven daarom ook verschillende uitkomsten. Het is derhalve voor een organisatie, die gebruik wil gaan maken van een begrotingsmodel, van belang na te gaan of het model aansluit bij zijn ontwikkelomgeving. Een voorwaarde voor de bruikbaarheid van een model is daarom de mogelijkheid het model aan te passen met behulp van gegevens van *eigen* voltooide projecten. In hoofdstuk 8 wordt nader ingegaan op het belang van het creëren van een dergelijke gegevensverzameling.

Ook Kemerer (1987) laat zien dat de begrotingsresultaten van verschillende modellen aanzienlijk kunnen verschillen. Voor alle door hem onderzochte modellen zijn de verschillen tussen werkelijke en begrote kosten enorm. In zijn onderzoek maakt hij gebruik van de modellen COCOMO, ESTIMACS, FPA en Putnam. Deze modellen worden gebruikt om het aantal benodigde mensmaanden te schatten van 15 reeds voltooide projecten. Alle projecten hebben betrekking op omvangrijke administratieve applicaties. Behalve het feit dat Kemerer niet gebruik heeft gemaakt van gecalibreerde modellen, moet worden aangetekend dat de historische projectgegevens waarop de modellen van Putnam en Boehm zijn gebaseerd niet of nauwelijks betrekking hebben op administratieve toepassingen. In tabel 4.11 zijn de belangrijkste resultaten weergegeven.

Tabel 4.11 : Begroting van het aantal mensmaanden en het werkelijk bestede aantal mensmaanden.

Modellen	gemiddeld over alle projecten:		
	werkelijk aantal mensmaanden	begroot aantal mensmaanden	(begroting gedeeld door werkelijkheid) * 100%
COCOMO	219,25 (15 projecten)	1291,75	607,85
Putnam	219,25 (15 projecten)	2060,17	771,87
FPA	260,30 (14 projecten)	533,23	167,29
ESTIMACS	287,97 (12 projecten)	354,77	85,48

Evenals Mohanty noemt Kemerer het achterwege laten van calibreren van de modellen aan de specifieke karakteristieken van de eigen organisatie de belangrijkste reden voor de geconstateerde verschillen. Modellen kunnen niet straffeloos overgeplant worden van de ene organisatie naar de andere.

Een soortgelijk onderzoek is uitgevoerd door Rubin (1985). Hij heeft een vergelijking gemaakt tussen de modellen Jensen, Putnam, GECOMO en (zijn eigen) ESTIMACS. Met behulp van deze vier modellen is een begroting gemaakt van het aantal mensmaanden en de ontwikkeltijd voor een bepaald te ontwikkelen (administratief) programma. Uit tabel 4.12 valt af te lezen dat de begrotingen onderling sterk verschillen.

Tabel 4.12 : Begrotingen met behulp van vier verschillende modellen voor één en hetzelfde project.

	Jensen	Putnam	GECOMO	ESTIMACS
Inspanning	940 MM	200MM	363MM	17100 uur
Duur	31 m	17 m	23 m	16 m

Ook Rubin geeft als verklaring hiervoor dat de modellen gebaseerd zijn op verschillende databanken van voltooide projectgegevens. Ook andere vergelijkbare onderzoeken tonen aan dat begrotingsresultaten flink uiteenlopen (Thibodeau 1981, Kitchenham en Taylor 1985, Noth en Kretzschmar 1984, Jensen 1984). Allen zijn ook eensluidend in hun oordeel dat een begrotingsmodel pas dan een zinvolle ondersteuning bij het bepalen van kosten, tijd en middelen kan zijn als het model gecalibreerd is, op maat gemaakt is voor de eigen organisatie. Om dit te realiseren is het van belang te beschikken over een eigen, uitgebreide verzameling van voltooide projectgegevens.

In hoofdstuk 8 wordt de centrale rol van zo'n gegevensverzameling bij het begroten toegelicht en wordt een indicatie gegeven van welk soort gegevens van voltooide projecten vastgelegd moeten worden. Deze paragraaf wordt afgesloten met een vergelijking van de geëvalueerde modellen. In tabel 4.13 wordt aangegeven in welke mate deze modellen voldoen aan de eisen van de beoordelings-methode.

Tabel 4.13 : De mate waarin de geëvalueerde modellen voldoen aan de eisen van de beoordelingsmethode.

EISEN ▼ / MODELLEN →	COCOMO	PRICE-SP	PUTNAM	FPA	BYL	Estimacs	SPQR	BIS
MODEL-EISEN								
gekoppeld aan beheersingsmethode	--	--	--	--	--	++	--	--
vroeg toepasbaar	--	--	--	+	+	++	+	-
evoluerend	+	--	--	--	--	--	--	++
aanpassen aan doelstellingen	+	+	+	--	--	+	++	--
toepasbaarheid	+	-	-	++	-	-	-	++
TOEPASSINGS-EISEN								
calibratie	-	--	--	-	+	+	-	-
nauwkeurigheid	?	?	?	?	?	?	?	?
IMPLEMENTATIE-EISEN								
gebruiksvriendelijkheid	++	-	+	+	++	+	+	+
gevoeligheidsanalyse	--	+	--	--	++	++	-	-
risico-analyse	--	--	--	--	--	++	+	--
inzichtelijkheid	++	--	++	++	++	-	-	+
eenduidigheid invoer	++	-	++	-	+	+	+	+
volledigheid en detail van uitvoer	+	++	-	-	++	++	++	++

Het teken ++ staat voor: voldoet goed. Een + betekent voldoende, een - betekent onvoldoende en -- betekent dat het model aan deze eis niet voldoet. Als geen uitspraak gedaan kan worden, staat er een ? in de tabel. Wat opvalt in de tabel zijn de weinige plussen. De conclusie is dan ook dat kwaliteit van de huidige modellen te wensen over laat en nog veel verbeteringen nodig zijn.

4.10. HET BELANG VAN BEGROTINGSMODELLEN

Uit de enqueteresultaten van hoofdstuk 2 blijkt dat begrotingsmodellen in de praktijk op de dag van vandaag weinig worden gebruikt. Van de 364 organisaties die begroten maken er slechts 51 gebruik van modellen. Deze 51 modelgebruikers "doen het slechter" dan de niet-model gebruikers. Zoals uit de enqueteresultaten bleek, hebben organisaties die gebruik maken van begrotingsmodellen significant budgetoverschrijdingen, althans voor grote projecten. Dat stelt op het

eerste gezicht teleur. Maar dat betekent mijns inziens niet dat men geen verdere inspanning in modellen moet plegen. De onderzoeken van Kemerer, Rubin en Mohanty, die in de vorige paragraaf beschreven zijn, benadrukken het belang van calibratie. Worden begrotingsmodellen niet aangepast aan de eigen ontwikkelomgevingen dan heeft dit tot gevolg dat de nauwkeurigheid van de afgegeven begrotingen gering is. De evaluaties van de modellen in dit hoofdstuk bevestigen dit. De modellen voldoen in beperkte mate aan de eisen die in de beoordelingsmethode zijn opgenomen. Ondanks de gebreken van de huidige modellen, is het zinvol de ontwikkeling van geschikte begrotingshulpmiddelen te bevorderen. In deze paragraaf zullen een aantal argumenten worden gegeven die pleiten voor het investeren van tijd en energie in de ontwikkeling van begrotingsmodellen.

Bij het opstellen van een begroting heeft men te maken met slecht geheel gestructureerde problemen, zijn er verschillende veelal conflicterende doelstellingen (kostenminimalisatie, maximaliseren van de kwaliteit, minimalisatie doorlooptijd, optimaal gebruik personeel enz.) en zijn er veel betrokkenen c.q. belanghebbenden (opdrachtgever, gebruiker, projectmanager, ontwikkelaars enz.). Verder speelt een groot aantal, moeilijk te kwantificeren en moeilijk meetbare factoren een rol bij het nemen van een beslissing over de te besteden middelen. De invloed van een eenmaal genomen beslissing is evenwel groot in termen van afgegeven doorlooptijd, toewijzing van personeel en te besteden budget. Voor het oplossen van dergelijk soort problemen kan een begrotingsmodel toch een geschikt hulpmiddel zijn (Bemelmans 1987). Een voorwaarde is wel dat zo'n model aan een aantal eisen voldoet (zie beoordelingsmethode). Is dat het geval, dan is een dergelijk model geen algoritme voor het berekenen van een optimale oplossing, maar biedt het model de begroter een aantal scenario's waaruit hij kan kiezen.

Bij het opstellen van een begroting, in het bijzonder bij de aanvang van het project, is er sprake van een grote onzekerheid en wazigheid. Men weet niet welke factoren en in welke mate deze factoren een rol spelen bij het bepalen van de kosten, inspanning en doorlooptijd. Bovendien heeft men geen inzicht in het waardebereik van de verschillende factoren. Voor de begroter c.q. projectleider is het in dergelijke onoverzichtelijke situaties belangrijk dat er nagagaan kan worden wat de effectiviteit is van bepaalde stuuracties. Het afgeven van puntschattingen in de vorm van "doorlooptijd is 320 mensmaanden waarvan 110 voor analyse, 70 voor ontwerp enz.", is van minder belang. Dergelijke exacte cijfers stroken niet met de aard van de problematiek. Voor een projectleider is het veel interessanter als er aangegeven kan worden hoe gevoelig een begroting is voor bepaalde

invloedsfactoren. Bijvoorbeeld: wat is het effect van de inzet van extra analisten op de doorlooptijd; hoe ontwikkelen zich de kosten als ik de projectduur aanzienlijk inkort; wat is het effect op de begroting als de complexiteit van het te ontwikkelen programma een nivo te hoog of te laag wordt ingeschat, en ga zo maar door. Door op zo'n wijze om te gaan met een begrotingsvraagstuk krijgt de projectleider meer gevoel voor en inzicht in oplossingsmogelijkheden. Bovendien ontstaat er op deze wijze een geschikte basis voor de voortgangsbewaking van een project. Blijkt een begroting bijzonder gevoelig te zijn voor wijzigingen in de waarde van een of meer kostenbepalende factoren, dan is dit voor een projectleider een waarschuwing juist deze factoren tijdens de uitvoering van het project in de gaten te houden.

Bij de opstelling van een begroting zal men in tal van gevallen geconfronteerd worden met een beslissingssituatie waarbij de uitkomst min of meer vastligt. Dat wil zeggen dat er weinig speelruimte is in opleverdatum, prijs en kwaliteit. In dergelijke gevallen zal men behoefte hebben aan een ondersteuning bij het kiezen van de waarden van de beslissingsvariabelen. Welke mogelijkheden zijn er om aan de gegeven doelstellingen te voldoen. Welk personeel in combinatie met welke hulpmiddelen en met behulp van welke wijze van projectorganisatie, zijn mogelijke alternatieve oplossingen. Het bovenstaande laat zien dat men bij het opstellen van een begroting niet zozeer behoefte heeft aan een rigide "rekenmodel". De probleemsituatie kenmerkt zich ondermeer door onzekerheid, wazigheid, weinig structurering en onduidelijke en onvolledige specificaties, die bovendien vaak tijdens de ontwikkeling van de programmatuur aan verandering onderhevig zijn. Daarom heeft men veel meer behoefte aan een model dat ondersteunt bij de beslissingsvoorbereiding en -analyse.

Een dwingende voorwaarde waaraan voldaan moet worden bij begroten van software is dat er afspraken worden gemaakt en eenduidige definities en standaards worden gehanteerd, geaccepteerd en nageleefd. Een en ander resulteert in het maken van afspraken en richtlijnen die betrekking hebben op ondermeer:

- het aantal malen dat een begroting in principe voor een project wordt opgesteld. Bijvoorbeeld: vijfmaal voor elk project dat meer dan 12 mensmaanden kost.
- de stadia in de projectuitvoering waarop wordt begroot. Bijvoorbeeld: tijdens de haalbaarheidsstudie, tijdens het opstellen van de specificaties en na afronding van het ontwerp.
- degenen die bij het begroten worden betrokken. Bijvoorbeeld de projectmanager, de opdrachtgever, vertegenwoordigers van het

ontwikkelteam.
- datgene dat wordt begroot. Bijvoorbeeld alle ontwikkel-activiteiten die betrekking hebben op de fasen vooronderzoek, specificeren, ontwerpen, enz., of alle activiteiten inclusief opleiding, documenteren, enz.
- de uitkomsten van een begroting. Bijvoorbeeld de kosten in guldens, de inspanning in mensmaanden en de doorlooptijd in maanden.
- de factoren die als de belangrijkste cost drivers worden beschouwd en waarover registratie moet plaatsvinden. Bijvoorbeeld de factoren omvang, betrouwbaarheid, type applicatie, kwaliteit personeel enz.
- de maatvoering die wordt gebruikt. Bijvoorbeeld: omvang wordt uitgedrukt in funktiepunten, deze worden vervolgens omgerekend naar aantal regels code (exclusief commentaar- en blancoregels).

Dit resulteert in een uitgebreide opsomming van maten en standaards die binnen een organisatie gehanteerd worden bij de ontwikkeling van software. Het is belangrijk dat een eenmaal gekozen maatvoering konsekwent wordt toegepast. Hierdoor onstaat op den duur een consistente verzameling van gemeten waarden.
Een begrotingsmodel dat voldoet aan eisen zoals een duidelijk definitiestelsel, meetbare en relevante cost drivers, evoluerend begroten enz. zorgt ervoor dat men bij begroten gestructureerd moet nadenken over het begrotingsvraagstuk.

Wat betreft de nauwkeurigheid van begrotingen, opgesteld met behulp van een model moet het volgende worden opgemerkt. Ook als begrotingen systematisch afwijken van de werkelijk bestede middelen, dan heeft men relatief nog een goed idee van kostenverhoudingen tussen projecten.

4.11. CONCLUSIES

In dit hoofdstuk is een overzicht gegeven van de belangrijkste modellen voor het begroten van softwareprojecten. Ondanks de korte historie van het onderwerp "begroten van softwareprojecten", is er de afgelopen vijftien jaar een aanzienlijk aantal modellen ontwikkeld. In paragraaf 3.3 is aangegeven op welke wijze een ordening in deze modellen gemaakt kan worden. In tabel 4.14 wordt deze ordening

Tabel 4.14. Een ordening van begrotingsmodellen. Een kruisje in de tabel wil zeggen dat een kenmerk gerelateerd is aan het betreffende model.

↓ KENMERKEN / MODELLEN →	COCOMO	PRICE-SP	PUTNAM	FPA	BYL	SPQR	Estimacs	BIS
bepaling omvang op basis van regels code	x	x	x					
bepaling omvang niet op basis van regels code				x	x	x	x	x
primair gericht op bepaling omvang				x	x	x	x	
primair gericht op bepaling produktiviteit	x	x	x		x	x	x	x
gebaseerd op formule die ervaringsfeiten geconcentreerd weergeeft	x	x	x	x	x	x	x	
gebaseerd op expertkennis								x
gebaseerd op analogie								
top-down benadering	x	x	x		x	x	x	x
bottom-up benadering	x							x
gericht op bepaald type applicatie	x			x				x

gebruikt voor het typeren van de modellen die in dit hoofdstuk zijn geëvalueerd. Uit de tabel valt ondermeer af te lezen dat bij merendeel van de modellen ervaringsfeiten van voltooide projecten zijn ondergebracht in een formule. Verder hanteren de meeste modellen een top-down benadering.
Werden modellen aanvankelijk voor intern gebruik in zeer grote organisaties ontwikkeld (bijvoorbeeld het Amerikaanse Ministerie van Defensie), de laatste jaren kan men een trend bespeuren dat steeds meer modellen op commerciële basis ook voor kleinere organisaties beschikbaar komen. Dat we hierbij pas bij het begin van een nieuwe ontwikkeling staan, moge blijken uit het geringe aantal organisaties dat vooralsnog hiervan gebruik maakt (zie hoofdstuk 2; resultaten enquete). Het zijn met name de implementatiemogelijkheden op een Personal Computer en het bedieningsgemak die deze hulpmiddelen aantrekkelijk lijken te maken. Bovendien wordt er door de negatieve publiciteit over falende softwareprojecten en aanzienlijke budget- en levertijdoverschrijdingen, een gewillige markt gecreëerd voor der-

gelijk soort modellen. Deze ontwikkeling is in zoverre verontrustend, omdat een uitgebreide toepassing van modellen vooralsnog ervan niet verantwoord is gezien de beperkte mogelijkheden voor calibratie, uitvoeren van gevoeligheids- en risico-analyses enz. In dit hoofdstuk zijn voldoende voorbeelden hiervan gegeven. Het gevaar bestaat dat door ondeskundig gebruik van modellen, een positieve ontwikkeling in het begroten van softwareprojecten ten onrechte schade wordt berokkend. Bij de evaluatie van Funktiepunt Analyse is gewezen op ondeskundig gebruik. Want positief is deze ontwikkeling zeker te noemen. Begrotingsmodellen hebben het voordeel dat men gestructureerd moet nadenken over de problematiek. Gebruik ervan dwingt tot uitspraken over een groot aantal zaken, die zonder gebruik van een model gemakkelijk over het hoofd gezien kunnen worden. Een zekere scepsis is echter noodzakelijk. Nagegaan moet worden of het model voor de betreffende organisatie inderdaad het meest geschikte hulpmiddel is. De gepresenteerde beoordelingsmethode kan hiervoor gebruikt worden.

Wil men met enig succes gebruik maken van een model, dan is het vastleggen van gegevens over voltooide softwareprojecten noodzakelijk. Hierdoor is men in staat zo'n model te calibreren, op maat te maken, voor de eigen organisatie. Registratie is trouwens steeds vereist ongeacht de begrotingsmethode die men gebruikt. Het is per slot van rekening riskant blind te varen op de resultaten van één model. Verstandig gebruik van een model wil zeggen dat de resultaten als een "tweede opinie" gebruikt moeten worden naast de begrotingen verkregen met behulp van bijvoorbeeld een analogie en/of expert methode. Ook is het wellicht raadzaam niet één maar meerdere modellen te gebruiken. Liggen de begrotingsresultaten ver uiteen dan is dit een extra waarschuwing om met omzichtigheid een uiteindelijke begroting te maken, waarin voldoende marges voor de diverse risico's zijn opgenomen.

5. KOSTENBEPALENDE FACTOREN

5.1. INLEIDING

Zoals in de hoofdstukken 3 en 4 is gesignaleerd, is een groot aantal factoren van invloed op kosten, tijd en inspanning om software te ontwikkelen. Bij het begroten is het van belang dat men inzicht heeft in *welke* factoren dit zijn, in *hoe groot* de invloed van deze factoren is, en in de *correlaties* tussen deze factoren . Naarmate dit inzicht groter is, zal men in staat zijn nauwkeuriger te begroten en het ontwikkelproces beter te beheersen.

In dit hoofdstuk zal een overzicht worden gegeven van dié kostenbepalende factoren, ook wel cost drivers genoemd, waarvan in de literatuur over het algemeen wordt gesignaleerd dat ze een belangrijke invloed kunnen hebben op de ontwikkelkosten van software. Belangrijke punten die in dit overzicht naar voren komen zijn ondermeer:

- er zijn erg veel factoren die van invloed zijn op de ontwikkelkosten van software;
- factoren blijken vaak onderling gecorreleerd te zijn;
- voor het merendeel van deze factoren bestaan geen eenduidige definities en is het lastig aan te geven wat de invloed is op de ontwikkelkosten;
- factoren die vallen onder de categorie "Human Factors" , zoals kwaliteit en ervaring van personeel, teamsamenstelling en dergelijke, hebben een grote invloed op de ontwikkelkosten;
- zoals bij de evaluatie van begrotingsmodellen in hoofdstuk 4 al is gesignaleerd, wordt in het overzicht van de kostenbepalende factoren het belang van calibratie van begrotingsmodellen onderstreept. Omdat de samenstelling van factoren en de invloed van een factor op de kosten per project kan variëren, is het belangrijk dat voor elk nieuw project wordt beoordeeld of het model bruikbaar is c.q. hercalibratie vereist is;

- een gedeelte van de kostenbepalende factoren heeft primair betrekking op werkwijze van software-ontwikkeling, dat wil zeggen op de wijze waarop mensen en middelen worden ingezet, en een ander gedeelte op de kenmerken van het te ontwikkelen produkt en de gebruiker waarvoor het produkt moet worden ontwikkeld.

Het doel van dit hoofdstuk is het aanbrengen van een ordening in de kostenbepalende factoren en het signaleren wat gevolgen zijn van het huidige gebrek aan kennis over deze factoren. Een dergelijke kennis is een belangrijke voorwaarde voor begroting en beheersing van software-ontwikkeling. Naarmate men immers met meer zekerheid uitspraken kan doen over samenstelling, onderlinge afhankelijk, invloed op ontwikkelkosten en -tijd enz., des te nauwkeurig zal men kunnen begroten en des te beheersbaarder zal het project zijn. In de hoofdstukken 6 en 7 wordt hierop nader ingegaan. In paragraaf 5.2 wordt een ordening gegeven van kostenbepalende factoren gegeven. Er worden vijf categorieën factoren onderscheiden. Elk van de volgende vijf paragrafen, 5.3 tot en met 5.7, is gewijd aan één categorie. Het hoofdstuk wordt afgesloten met conclusies (paragraaf 5.8).

5.2. EEN ORDENING VAN KOSTENBEPALENDE FACTOREN

Noth en Kretzschmar (1984) tonen aan dat het aantal factoren, dat van invloed is op de ontwikkelkosten van software, zeer groot kan zijn en deze factoren veelal sterk gecorreleerd zijn. In hun onderzoek komen zij in totaal tot maar liefst 1200 verschillende cost drivers. Andere onderzoeken beperken zich tot een of enkele factoren. Het zal duidelijk zijn dat het maken van een begroting 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. Hierbij moet worden opgemerkt, dat het niet mogelijk is in absolute termen te spreken over "de belangrijkste kostenbepalende factoren". Welke factoren bij de ontwikkeling van een programma in een bepaalde praktijksituatie kostendominant zijn, wordt in belangrijke mate bepaald door de omgeving waarin de software wordt ontwikkeld. Zo zal in de lijst van belangrijke cost drivers, de factor "vierde generatiehulpmiddelen" niet voorkomen, als in een betreffende organisatie niet of nauwelijks van dergelijke hulpmiddelen gebruik wordt gemaakt. In andere organisaties, die intensief dergelijke hulpmiddelen toepassen, is het goed mogelijk dat deze factor wel een belangrijke invloed heeft op de

Tabel 5.1 : Diverse modellen en hun kostenbepalende factoren.

MODELLEN → / FACTOREN	ESTIMACS	SPQR-20	COCOMO	BYL	BIS	SOFCOST	PRICE-S	SLIM	DOTY	JPL	BOEING	SLICE	JS-2/3	FPA	IBM
Omvang in regels code	*	*	*	*	*	*	*	*	*	*	*	*	*		*
Omvang in funktiepunten	*	*		*										*	
type applicatie		*				*		*	*		*				*
vereiste kwaliteit	*	*	*	*		*		*		*			*	*	
omvang database			*	*		*								*	*
complexiteit software	*	*	*	*	*	*	*	*				*	*		*
hoeveelheid documentatie		*				*			*			*	*		*
mate van hergebruik	*	*	*	*		*		*			*		*	*	
beperkingen (*)	*	*	*	*		*		*			*		*	*	
tools en/of technieken		*	*	*	*	*	*	*			*	*	*		*
kwaliteit personeel			*	*		*		*					*		*
ervaring personeel	*	*	*	*		*		*	*		*	*	*		*
kwaliteit management							*			*		*			
eisen aan projectduur		*	*	*		*		*		*			*		
projectbeheersing	*				*	*	*	*	*			*	*		
gebruikersparticipatie	*					*									*
dynamiek specificaties	*	*	*	*		*	*		*	*			*		*
opleiding en training															
aantal gebruikers		*			*										

Noot: een (*) betekent dat beperkingen staat voor beperkingen wat betreft executietijd en/of geheugenbeslag en/of responsietijd.

hoogte van de ontwikkelkosten. Tabel 5.1 brengt in beeld dat de samenstelling van de cost drivers sterk kan verschillen. In de tabel is een overzicht gegeven van welke kostenbepalende factoren in welke modellen voorkomen.

Bij analyse van begrotingsmodellen komt naar voren, dat de omschrijvingen van de factoren in de verschillende modellen sterk uiteenlopen. Zo hanteert bijvoorbeeld het ene model voor de factor "ervaring" een beperkte definitie, bijvoorbeeld ervaring met soortgelijke hardware, terwijl een ander model voor deze factor een veel ruimere interpretatie kent, bijvoorbeeld ervaring met soortgelijke applicaties, met de programmeertaal, enz. In de volgende paragrafen van dit hoofdstuk wordt uitvoerig ingegaan op de verschillen in definities van de factoren. Zoals uit tabel 5.1 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 ontwikkelkosten. Zo komt het voor dat factor x in model A als de meest dominante geldt, terwijl diezelfde factor in model B als veel minder dominant wordt ervaren. Om dit te illustreren wordt in tabel 5.2 aangegeven welke vijf cost drivers in zes verschillende onderzoeken de grootste invloed hebben op kosten en doorlooptijd. Bij vier van deze onderzoeken wordt eveneens de mate van de invloed aangegeven. De verschillen zijn voor een belangrijk deel terug te voeren tot de specifieke kenmerken van de omgeving waarin het model tot stand is gekomen c.q. het onderzoek is uitgevoerd. In de ene omgeving kunnen andere factoren, in een andere mate, relevant zijn dan in de andere omgeving. Voor een specifieke organisatie is het daarom van belang op basis van analyses van veel voltooide projecten te komen tot een "eigen" rangorde van cost drivers.
Wat verder opvalt in tabel 5.2 is de belangrijke invloed op ontwikkelkosten van "menselijke" factoren, zoals kwaliteit en ervaring van personeel. In praktisch alle door mij onderzochte studies wordt het belang van dergelijke factoren gesignaleerd. In paragraaf 5.5 wordt hier nader op ingegaan.

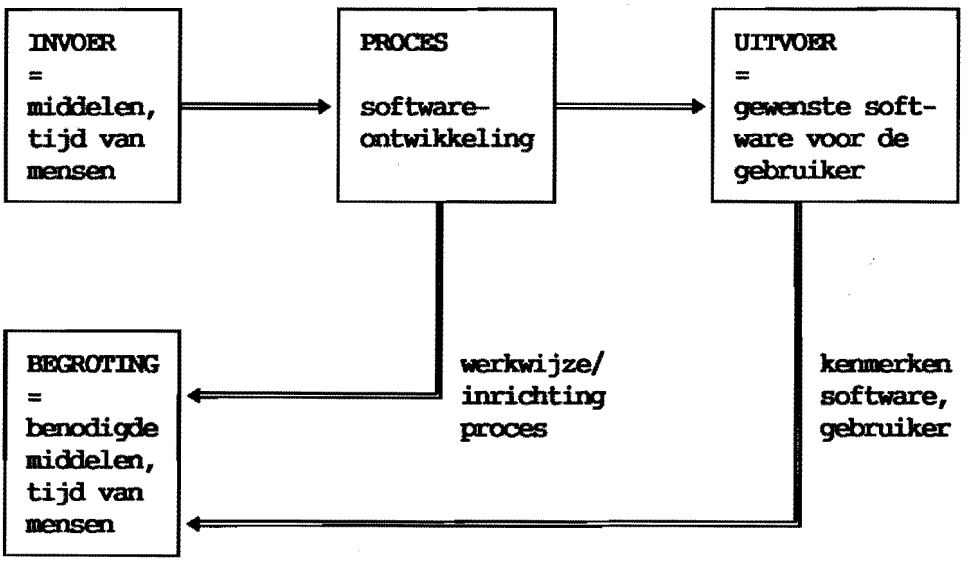
Gezien de hoeveelheid factoren en de vaak complexe relatie van deze factoren met de ontwikkelkosten is een ordening gewenst. Een mogelijke manier om het proces van software-ontwikkeling te beschouwen is de volgende: uitvoer is het gewenste produkt voor een bepaalde gebruiker; invoer is de tijd van mensen en de inzet van middelen die nodig zijn om dit produkt te realiseren; het proces is de eigenlijke software-ontwikkeling.

Tabel 5.2 : De 5 meest dominante cost drivers in 6 verschillende onderzoeken.

Onderzoek	Factoren (in volgorde van dominantie)	Invloed
COCOMO (Boehm 1981)	1. omvang van de software	
	2. bekwaamheid personeel/ontwikkelteam	4,18
	3. complexiteit van te ontwikkelen software	2,36
	4. vereiste betrouwbaarheid software	1,87
	5. beperkingen wat betreft responsietijd	1,66
Walston en Felix (1977)	1. complexiteit van de mens-machine interface	4,10
	2. ervaring met dezelfde programmeertaal	3,15
	3. algehele ervaring en kwaliteit van het personeel	3,10
	4. ervaring met soortgelijke omvang applicaties	2,88
	5. deelname gebruiker bij opstellen specificaties	2,86
Mizuno (1983)	1. menselijke factoren	2,22
	2. gebruikerscomplexiteit	1,81
	3. mate van gebruik ontwikkelingstools	1,72
	4. vereiste kwaliteit	1,50
	5. bekendheid van het type applicatie	1,42
Siskens, Heemstra, van der Stelt (1988)	1. omvang van het te ontwikkelen software	-
	2. complexiteit van de te ontwikkelen software	-
	3. wijzigingen in de specificaties	-
	4. kwaliteit personeel	-
	5. vereiste kwaliteit software	-
Jensen (1985)	1. reis en verblijfkosten	2,22
	2. complexiteit van de te ontwikkelen software	2,22
	3. bekwaamheid personeel	2,00
	4. vereiste betrouwbaarheid	1,98
	5. beschikbaarheid van resources	1,92
Druffel (1982) (voorspelling voor 1993)	1. mate van hergebruik	-
	2. gebruik van geautomatiseerde hulpmiddelen	-
	3. gebruik van moderne programmeringstechnieken	-
	4. beperkingen wat betreft executietijd	-
	5. beperkingen wat betreft geheugencapaciteit	-

noot: De getallen in de kolom "invloed" zijn een maat voor het verschil tussen de minimale en de maximale invloed van de corresponderende factor op de kosten. Bijvoorbeeld: de waarde 4,18 in de tweede regel betekent dat de ontwikkelkosten met een factor 4,18 zullen toenemen, als in plaats van bekwaam niet-bekwaam personeel wordt ingezet. In de onderzoeken van Siskens, Heemstra en van der Stelt (1988) en van Druffel (1982) is niet de mate van de invloed bepaald.

Bij het opstellen van een begroting wil men deze invoer bepalen. Als we uitgaan van de veronderstelling, dat het grootste gedeelte van de ontwikkelkosten van software personeelskosten zijn, dan betekent een en ander dat primair een schatting wordt gemaakt van de tijd die men nodig heeft om de uitvoer i.c. het gewenste produkt voor de betreffende gebruiker te maken. Deze schatting wordt doorgaans uitgedrukt in het aantal benodigde mensmaanden. Deze schatting is verder gebaseerd op de werkwijze die men zal gaan volgen bij het proces i.c. de software-ontwikkeling. Dat wil zeggen over de inrichting van het werk, de inzet van mensen en middelen. Het voorgaande kan schematisch als volgt worden weergegeven (figuur 5.1):



Figuur 5.1 : Begroten van software-ontwikkeling.

Een nadere analyse van figuur 5.1 laat zien dat de begroting van benodigde middelen en mensmaanden wordt bepaald door factoren die voor een deel zijn terug te voeren tot kenmerken van het gewenste produkt en de gebruiker in kwestie en voor een deel tot de manier waarop mensen en middelen worden ingezet om dat produkt te realiseren. Deze factoren worden voor de overzichtelijkheid in het navolgende ingedeeld in een aantal categorieën. We onderscheiden factoren met betrekking tot:

a. de te ontwikkelen software zelf
(WAT),
b. de middelen waarmee de software wordt ontwikkeld
(WAARMEE),
c. het personeel dat de software ontwikkelt
(DOOR WIE),
d. projekteigenschappen; de wijze van ontwikkeling
(HOE),
e. eigenschappen van de organisatie waarvoor de software wordt ontwikkeld
(VOOR WIE),

Factoren die vallen onder de categorieën WAARMEE, HOE en DOOR WIE hebben betrekking op de werkwijze van het proces van software-ontwikkeling. De WAT en VOOR WIE factoren hebben betrekking op het te realiseren produkt en de betreffende gebruiker. In tabel 5.3 wordt een overzicht gegeven van factoren die volgens de literatuur en volgens eigen onderzoek (Heemstra, van Genuchten en Kusters 1988, Haarhuis 1987) een belangrijke invloed hebben op kosten, tijd en inspanning voor de ontwikkeling van software.

Tabel 5.3 : Overzicht van de kostenbepalende factoren.

KOSTENBEPALENDE FACTOREN M.B.T. HET				
WAT (produkt)	WAARMEE (middelen)	DOOR WIE (personeel)	HOE (project)	VOOR WIE (gebruiker)
omvang van de software kwaliteit omvang van de database complexiteit software documentatie hergebruik type applicatie	beperkingen tav -executie tijd -responsie tijd -geheugencapaciteit gebruik van tools gebruik van moderne programmeertechnieken	kwaliteit personeel ervaring personeel kwaliteit management	eisen, gesteld aan projectduur wijze van projectbeheersing	participatie aantal gebruikers stabiliteit van de gebruikersorganisatie c.q. mate waarin specificaties veranderen ervarings/opleidingsnivo gebruiker

Deze cost drivers zijn in de tabel verdeeld naar de hierboven genoemde vijf categorieën. In de volgende paragrafen van dit hoofdstuk zullen deze factoren een voor een worden besproken. Het accent zal hierbij vooral worden gelegd op de problemen die er zijn bij de definitie van de factoren en bij de bepaling van de invloed van de verschillende factoren. Verder zal worden gesignaleerd dat de factoren met elkaar samenhangen en elkaar wederzijds beïnvloeden.

5.3. PRODUKTAFHANKELIJKE FACTOREN

omvang van de software

Algemeen is men het erover eens dat kosten, inspanning en tijd van de te ontwikkelen software zullen toenemen naarmate de omvang ervan toeneemt. Deze constatering is onafhankelijk van de maat die men hanteert voor de factor omvang.

Men treft in de literatuur verschillende methoden aan om de omvang te bepalen (Craenen 1984):

a. **het aantal regels code,**
b. **de 'Halstead' methode,**
c. **het aantal funktiepunten.**

ad a. **Het aantal regels code**

Dit is de meest verbreide, maar tevens meest bekritiseerde methode. Velen wijzen terecht op het feit dat de ene coderegel de andere niet is. Zo kost het maken van regels commentaartekst 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 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 5.4. In deze tabel wordt een beeld gegeven van de verschillende definities die in een aantal onderzoeken voor het begrip omvang worden gebruikt (Thibodeau 1981).

Een gevolg van het gebrek aan een eenduidige definitie voor de omvang van een programma is dat men op grond van een methode,

gebaseerd op omvang voorzichtig moet zijn met uitspraken over een begroting of over de produktiviteit van een programmeur. Door met name blanco regels en commentaarregels mee te tellen bij het aantal regels code bestaat het gevaar dat de programmeur wordt aangemoedigd kunstmatig veel van dergelijk soort regels in zijn programma op te nemen. Op deze wijze wekt hij de illusie van een hoge produktiviteit (Conte, Dunsmore en Shen 1986). Verder is een vergelijking van de produktiviteit van programmeurs moeilijk als er sprake is van verschillende programmeertalen, bijvoorbeeld APL versus FORTRAN of COBOL.

Tabel 5.4 : Definities van de omvang in verschillende onderzoeken.

NAAM ONDERZOEK(ER)	DEFINIERING VAN OMVANG
Aerospace	aantal regels code in programma (inclusief blanco- en commentaarregel, databeschrijving enz.)
Farr en Zagorski, Telecote, Wolverton	aantal machine instructies
Putnam	aantal statements
Doty	aantal (executeerbare) regels code
PRICE-S	aantal (executeerbare) instructies of aantal machine instructies

ad b. de Halstead methode

Deze methode, die bekend staat onder de naam "Software Science" (Halstead 1977) telt niet het aantal regels code maar bepaalt de omvang van een programma op basis van:

- het aantal unieke operatoren. (Operatoren geven een actie aan, bijvoorbeeld + , - , * enz.),
- het aantal unieke operanden (variabelen en constanten) en
- het aantal malen dat deze operatoren en operanden in het programma voorkomen.

Ook bij deze methode bestaat het eerder genoemde definitie probleem. Evenals bij het tellen van het aantal regels code, bestaat bij de Halstead methode het bezwaar, dat grootheden als de benodigde programeerinspanning en geschatte programmeertijd pas bepaald kunnen worden als de software is geschreven. Pas na een gedetailleerde analyse beschikt men doorgaans over voldoende inzicht om de waarde van deze grootheden redelijk nauwkeurig te bepalen. Tenslotte moet worden opgemerkt dat er geen algemeen geaccepteerde opvatting bestaat over de beste manier om operatoren en operanden te tellen.

ad c. **de "waarde" van de gebruikersfunctie**

Ook bij deze methode gaat het niet om het aantal regels code. Als alternatief voor het aantal regels code wordt de hoeveelheid functies bepaald als maat voor de omvang die de programmatuur vervult, in termen van de data die het programma gebruikt en genereert. Een uitwerking hiervan is Albrecht's Functie Punt Analyse (Albrecht 1979) waarvan in de loop der tijd allerlei varianten en verfijningen zijn ontwikkeld. Deze methode om de omvang te schatten wordt veel gebruikt, ondanks de onduidelijkheid die er is over hoe men precies die functies (invoer, uitvoer, bestanden, triggers en interfaces) moet bepalen en waarderen. In paragraaf 3.5.4 is deze aanpak uitvoerig aan de orde gekomen.

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 de onderlinge communicatie in een project toeneemt. Hierop is al in hoofdstuk 1 gewezen. Bovendien gaat een toename in omvang veelal gepaard met een toename in complexiteit van de programmatuur. Hoe sterk die progressie is, verschilt van auteur tot auteur. Er bestaan ook onderzoekers die menen, dat de kosten degressief toenemen bij een toename van de omvang (Crossman 1979 en Walston en Felix 1977). In deze onderzoeken wordt ervan uitgegaan dat de kosten van dure hulpmiddelen zoals rapport-generatoren, programmeeromgevingen, testgereedschap, over meer regels code verdeeld worden. Gaat men ervan uit dat bij een toename van de omvang de kosten per regel code stijgen, dan veronderstelt men dat grote projecten relatief duurder zijn; er is meer overhead nodig voor communicatie en leiding, de problemen worden doorgaans complexer, enz. (Van Vliet 1987).

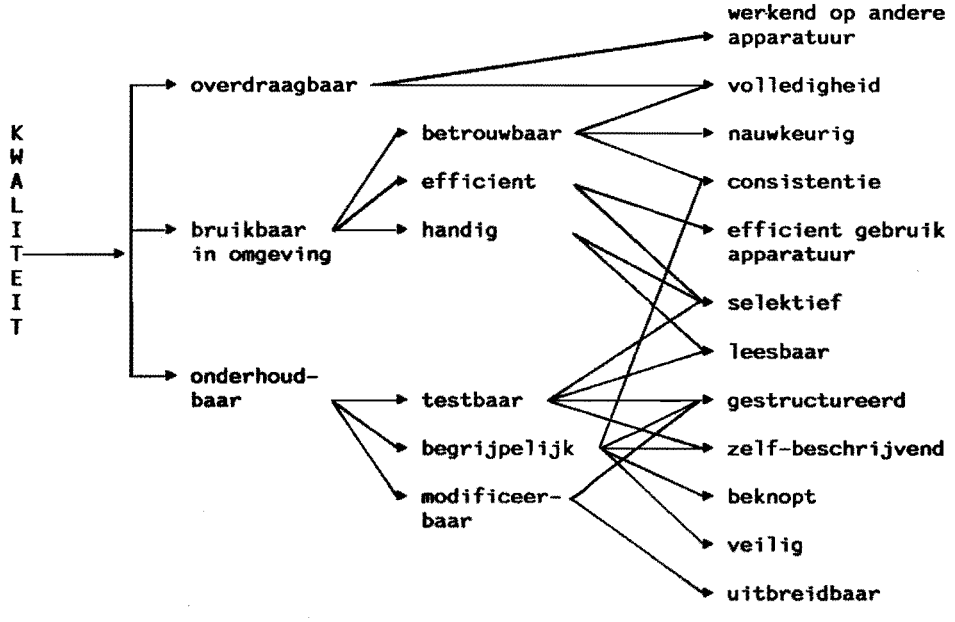
Uit het voorgaande kan men ondermeer concluderen dat er verschillende definities van omvang in omloop zijn en worden gebruikt in de diverse modellen. Vergelijking van onderzoeksresulaten over de relatie tussen omvang en ontwikkelkosten is daarom vaak moeilijk. Deze

moeilijkheid wordt nog groter omdat soms niet duidelijk is omschreven wat verstaan moet worden onder inspanning.

kwaliteit van de software

De kwaliteit van de te ontwikkelen software is een belangrijke 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, ofwel kwaliteitseisen. Het is echter moeilijk kwaliteit te definiëren en te operationaliseren. Nog moeilijker is het om de invloed ervan op de ontwikkelkosten (kwantitatief) te bepalen.
Er zijn slechts weinig geslaagde pogingen ondernomen om tot een werkbare omschrijving van softwarekwaliteit te komen. Als belangrijke bijdragen op dit gebied moeten de werken van Boehm c.s. worden genoemd (Boehm en anderen, 1978).

Boehm c.s (Boehm en anderen, 1978) definiëren een zogenaamde hiërarchie van kenmerken van softwarekwaliteit. Op een geaggregeerd nivo onderscheiden zij drie hoofdaspecten van softwarekwaliteit: bruikbaarheid, onderhoudbaarheid en overdraagbaarheid. In figuur 5.2 zijn deze kenmerken en hun onderlinge samenhang in beeld gebracht.



Figuur 5.2: Een hiërarchie van kwaliteitskenmerken (Boehm e.a. 1978).

H. Willmer (1985) borduurt in haar proefschrift verder op deze aanpak. Zij maakt daarbij onderscheid in kwaliteits- en produktkenmerken van software. In tabel 5.5 wordt een overzicht gegeven van deze kenmerken.

Tabel 5.5 : De invloed van de produktkenmerken op de kwaliteitkenmerken van een systeem.

KWALITEIT-KENMERKEN ▼ / PRODUKTKENMERKEN →	.visualisering	.documentatie	.mate van structurering	.mate van standaardisatie	.controle mogelijkheden in software access/control	.taalgebruik (duidelijke naamgeving, lengte zinnen, gebruik GOTO, enz.
.begrijpbaarheid	*	*	*	*		*
.veranderbaarheid	*	*	*		*	*
.overdraagbaarheid op andere hardware		*	*		*	*
.onderhoudbaarheid	*	*	*			*
.te koppelen met andere software		*	*	*		
.mogelijkheden van hergebruik	*	*	*			
.testbaarheid	*	*	*			*
.efficiency van de software			*		*	*
.beveiliging/autorisatie gebruik					*	
.correctheid/vrij van fouten				*		
.gevoeligheid voor fouten					*	
.herstartbaarheid na foutoptreden					*	

Noot : Een kruisje in de tabel geeft een relatie tussen beide kenmerken aan. Een voorbeeld: om te kunnen voldoen aan de eis "mogelijkheden van hergebruik", moet bij de ontwikkeling ervan aandacht besteed worden aan structurering, documentatie en visualisering. Wat die aandacht betekent, staat uitvoerig beschreven in (Willmer 1985).

Dit soort kenmerken manifesteert zich bij het gebruik van de programmatuur en moeten voorafgaande aan de ontwikkeling, dat wil zeggen bij het specificeren, vastgelegd worden. 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, volgens Willmer, nodig produktkenmerken te onderscheiden.

Bij de ontwikkeling van een programma 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. Dit soort kenmerken kan tijdens de ontwikkeling gevalideerd worden; men hoeft hiermee niet te wachten tot bij gebruik. Produktkenmerken beinvloeden de kwaliteit van het te ontwikkelen systeem. Aan de hand van produktkenmerken is het mogelijk richtlijnen te bepalen waarmee de kwaliteit doelgericht kan worden beïnvloed. Daarvoor is het nodig de samenhang te kennen tussen beide soorten kenmerken. Willmer beschrijft in haar onderzoek deze samenhang waarvan de essentie in tabel 5.5 wordt weergegeven.

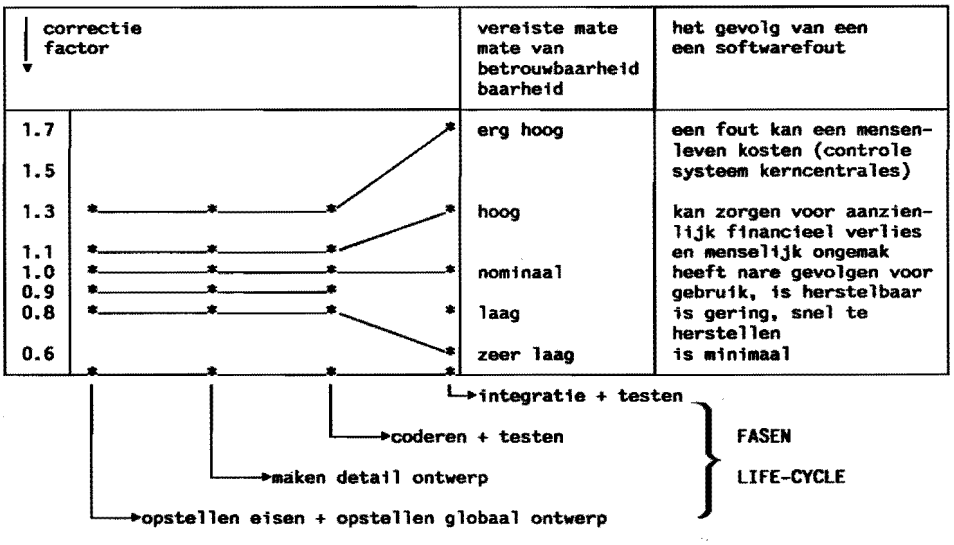
De studie van Willmer maakt opnieuw duidelijk hoe moeilijk het is antwoord te geven op vragen als: wat is softwarekwaliteit, wanneer is er sprake van goed onderhoudbare programmatuur, hoe kan dat gerealiseerd worden, hoe hangen kwaliteitskenmerken onderling samen en hoe zijn ze gerelateerd aan produktkenmerken. 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 vraag te beantwoorden: 'Wat is de invloed van kwaliteit op de kosten en doorlooptijd om software te ontwikkelen'. Ook de gedegen onderzoeken van onder andere Willmer (1985) en Boehm (1978) op dit gebied komen vooralsnog niet verder dan het geven van een kwalitatieve relatie tussen softwarekwaliteit en kosten. Kwantitatieve onderzoeksresultaten zijn slechts sporadisch aanwezig. In het bekende onderzoek van Walston en Felix (1977) wordt kwaliteit zelfs niet als kostenbepalende factor onderscheiden, wel een aantal daaraan gerelateerde factoren.

In het model COCOMO onderscheidt Boehm de cost driver betrouwbaarheid van software (required software reliability). Betrouwbaarheid wordt door hem gedefinieerd als de kans dat de programmatuur op een bevredigende wijze voldoet aan de gestelde eisen gedurende de

afgesproken verwerkingsgangen. COCOMO maakt gebruik van kwalitatieve maten voor betrouwbaarheid en met een kwalitatieve relatie tussen geëiste betrouwbaarheid en kosten. Een en ander betekent niet dat er geen kwantitatieve uitspraken gedaan kunnen worden over betrouwbaarheid (Musa, Iannino en Okumoto, 1987). Een uitsplitsing in deelfactoren, zoals Willmer dat doet, wordt in COCOMO niet uitgevoerd. De relatie tussen de factor betrouwbaarheid en de kosten volgens het model van Boehm wordt in tabel 5.6 in beeld gebracht.

Tabel 5.6 : De invloed van de factor betrouwbaarheid op de kosten.



Noot: 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 geval de vereiste betrouwbaarheid erg hoog is, betekent dit bijvoorbeeld dat het effect op de kosten het grootst is in de integratie- en testfase, namelijk een factor 1.7.

In het PRICE-S model (Freiman 1979) 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 kosten 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 voor alle factoren een kwantitatieve relatie tussen kwaliteit en kosten leggen. De weinige onderzoeken die een kwalitatieve relatie bepalen, geven unaniem aan dat de kosten toenemen naarmate er hogere kwaliteitseisen aan de te ontwikkelen software worden gesteld. De onderzoeken hanteren echter verschillende omschrijvingen van kwaliteit en van begrippen als een hoge, gemiddelde en lage kwaliteit. Ook moet men constateren dat in de verschillende onderzoeken verschillende opvattingen bestaan over de mate van 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 programma 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

In een aantal begrotingsmodellen (COCOMO, het model van Walston en Felix, Before You Leap, Sofcost, Funktie-Punt-Analyse) heeft de grootte van de database een duidelijke invloed op de kosten van de te ontwikkelen software (Boehm 1981). In veel begrotingsmodellen wordt deze factor niet als afzonderlijke cost driver onderscheiden en is deze verweven met de factor omvang van de software. Zo wordt bijvoorbeeld in het PRICE-S model de omvang van de database niet als aparte kostenbepalende factor onderscheiden. Ook het SPQR model van C. Jones (1986) kent deze factor niet als afzonderlijke cost driver.

De modellen die deze factor wel onderscheiden, hanteren vaak een verschillende definities. Door Walston en Felix wordt de grootte van

de database gedefinieerd als 'het aantal categorieën items in een database per 1000 regels code'. Het nadeel van deze definitie is dat de omvang van de database gerelateerd is aan het aantal entiteittypen. Een database met een aanzienlijk aantal entiteittypen en weinig occurences kan veel kleiner in omvang zijn dan een database met weinig entiteittypen, maar per type veel occurences. Het effect van deze factor in het onderzoek van Walston en Felix is trouwens 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 regels code" toeneemt van laag naar hoog.
In het model COCOMO wordt een definitie van de factor "grootte van de database" gebruikt, die volkomen afwijkt met de definitie van Walston en Felix. In COCOMO gaat het om de totale hoeveelheid occurrences ten opzichte van de totale hoeveelheid regels code. 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.

Binnen Funktie Punt Analyse wordt geen aparte factor onderscheiden die betrekking heeft op de invloed van de omvang van de database op de ontwikkelkosten. Funktie Punt Analyse richt zich evenwel, zoals in paragraaf 4.5 gesignaleerd, op het begroten van data-intensieve software. Bij de bepaling van de omvang van de software met behulp van FPA, zijn zaken als omvang en complexiteit van de data de meest bepalende factor. Deze omvang en complexiteit wordt bepaald aan de hand van ondermeer het aantal operaties met data (aantal invoer, uitvoer, bestanden enz.) en de complexiteit van die operaties (bijvoorbeeld: aantal geraadpleegde bestanden en records binnen die bestanden voor de realisatie van een uitvoer).

complexiteit van de software

Ook complexiteit wordt vaak genoemd als belangrijke cost driver. Opnieuw blijkt het moeilijk te zijn de invloed van deze factor op de ontwikkelkosten te bepalen.
Complexiteit heeft in dit verband betrekking op de inspanning die nodig is om software te maken. In de literatuur treft men veel verschillende complexiteitsmaten aan. Herrman (1984) geeft hiervan een overzicht. Hij onderscheidt maten die gebaseerd zijn op:

- de hoeveelheid interacties tussen de verschillende componenten /modules van de software;
- de mate van standaardisatie;

- het soort toepassingsgebied van de te ontwikkelen programmatuur;
- de structuur van de software;
- de moeilijkheid van de invoer/uitvoer.

Door Basili (1980) wordt een ordening aangebracht in de bestaande complexiteitsmaten. Hij beweert dat deze maten terug te brengen zijn tot maten die gebaseerd zijn op:

- **de omvang van de software.**
- **de structuur van de software.**

Bij complexiteitsmaten gebaseerd op de omvang wordt verondersteld dat er een duidelijk verband is tussen de omvang van de software en de complexiteit ervan. Naarmate de omvang toeneemt zal de complexiteit eveneens stijgen. Het probleem om de complexiteit van software te bepalen wordt op deze manier het probleem om de omvang te bepalen. Bij de beschrijving van de kostenbepalende factor is reeds gewezen op dit probleem. Verder dient men er rekening te houden dat de ene programmeur meer regels code voor de oplossing van een probleem met dezelfde complexiteit nodig heeft dan een andere programmeur. Bovendien zal men voor het coderen van dezelfde opdracht in de ene programmeertaal kunnen volstaan met een programma met een geringere omvang dan in een andere programmeertaal.
Complexiteitsmaten gebaseerd op de structuur van de software worden door Basili (1980) onder verdeeld in maten die zich oriënteren op de datastructuur van de software, op de controlestructuur of op een combinatie van deze twee. Een bekende complexiteitsmaat gebaseerd op de controlestructuur is de cyclomatische complexiteitsmaat van McCabe (1976).

Bij het begroten van de benodigde tijd, kosten en inspanning om een bepaald programma te ontwikkelen, zal men vooraf, dat wil zeggen bij aanvang van het ontwikkeltraject, een indicatie willen hebben van de complexiteit van dat programma. De bestaande complexiteitsmaten hebben het nadeel dat ze pas in een gevorderd stadium van het ontwikkeltraject, dat wil zeggen na het detailontwerp, redelijk nauwkeurig in te schatten zijn. In de eerste fase van software-ontwikkeling schieten deze kwantitatieve maten tekort en zal men terug moeten vallen op een kwalitatieve beoordeling van de complexiteit. Men loopt het gevaar dat de mate van complexiteit van het uiteindelijke produkt niet overeen komt met de prognoses die in de allereerste fasen van software-ontwikkeling zijn gemaakt. Deze prognoses zijn veelal gebaseerd op onvolledige en onduidelijke eisen van de gebruiker. Vaak komt het voor dat de gebruiker, gezien de aard van

de problematiek, niet goed in staat is die eisen te formuleren. Dit kan een reden zijn van een verschil in verwachte en gerealiseerde complexiteit.

Alle onderzoeken over de invloed van complexiteit op softwarekosten, zijn eensluidend in de richting van hun resultaten. Hoe hoger de complexiteit hoe hoger de kosten. In de bewuste onderzoeken wordt echter niet altijd dezelfde definitie van complexiteit gebruikt. Een vergelijking van de onderzoeksresultaten is daardoor moeilijk te maken. Een en ander wordt in tabel 5.7 weergegeven.

Tabel 5.7 : De invloed van een toename van de complexiteit op de ontwikkelkosten.

NAAM ONDERZOEKER	CRITERIUM VOOR COMPLEXITEIT	KOSTENFACTOR
Aron (1969)	frequentie van interakties	4
Boehm (1981)	type applicatie	2.5
Brooks (1980)	mate van standaardisatie	9
Chen (1981)	moeilijkheid invoer/uitvoer	5.65
Walston en Felix (1977)	type applicatie	1.7
Wolverton (1974)	mate van nieuwheid	5

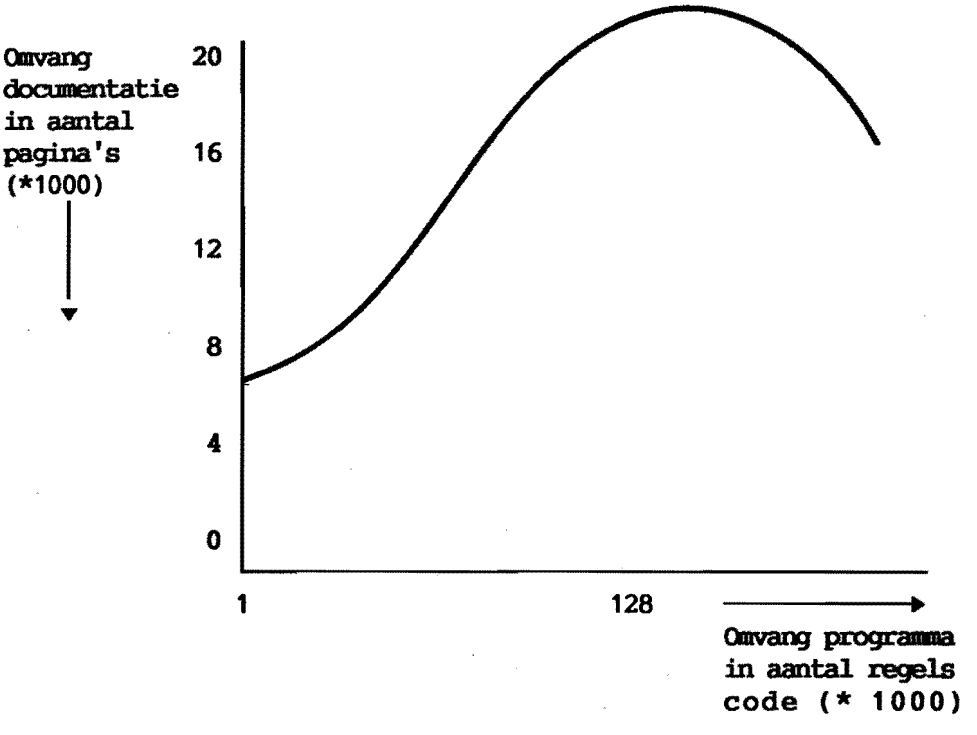
Noot : De kostenfactor geeft aan met welk getal de kosten moeten worden vermenigvuldigd als de complexiteit (voor wat betreft het bewuste criterium) toeneemt van laag naar hoog (Herrman 1984).

hoeveelheid documentatie

Documentatie is een belangrijk produkt van elk software- ontwikkelproject. Het begroten van documentatiekosten vormt derhalve een essentieel onderdeel van het kostenschattingsproces. Een veel gebruikte maat om de hoeveelheid documentatie uit te drukken, is het aantal pagina's. Walston en Felix (1977) signaleren in hun onderzoek dat de relatie tussen het aantal regels code en het aantal pagina's documentatie volgens de onderstaande vergelijking weergegeven kan worden:

$$D = KLOC^{1,01}$$

waarbij D staat voor aantal pagina's documentatie en KLOC staat voor aantal regels code in duizendtallen. Tot documentatie werd door hen gerekend de functionele specificaties, het functionele ontwerp, de testresultaten, de programmastroomschema's, delen van de programmalisting (in zoverre deze als onderdeel in handleidingen zijn opgenomen) en gebruikershandleidingen, . Uit hun onderzoek blijkt dat er bijna een lineaire relatie bestaat tussen de omvang van de programmatuur (aantal regels code) en de omvang van de documentatie (aantal pagina's). Haaks op deze bevindingen staan de onderzoeksresultaten van Jones (1986). In zijn onderzoek blijkt deze relatie niet lineair te zijn. Voor programma's met een omvang van 1000 tot 128.000 regels code 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 5.3).



Figuur 5.3 : De relatie tussen de omvang van de programmatuur en de hoeveelheid documentatie volgens Jones (1986).

Het blijkt opnieuw moeilijk een eenduidige kwantitatieve relatie te leggen tussen de omvang van de software en de daarbij behorende hoeveelheid documentatie. Er is een aantal factoren dat van invloed is op deze relatie. Zo blijkt dat onder meer de factor "type applicatie" een belangrijke invloed heeft op de hoeveelheid documentatie per 1000 regels code. Jones (1986) geeft aan dat de hoeveelheid documentatie bij programmatuur voor militaire toepassingen en voor projecten als bemande ruimtevaartvluchten omvangrijker zijn dan bij systemen van dezelfde omvang voor bijvoorbeeld administratieve toepassingen. Aan software van de eerste categorie zullen veelal 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 ontwikkelkosten.

Bij documenteren wordt men geconfronteerd met een afwegingsproces tussen enerzijds omvang en kwaliteit van de documentatie en anderzijds de benodigde tijd, geld en inspanning. Weinig aandacht voor c.q. besparingen op goede documentatie drukken weliswaar de ontwikkelkosten, maar bemoeilijken het onderhoud en het gebruik van de software. Korte termijn besparingen leiden aldus tot potentieel aanzienlijke kosten in de toekomst. Het zoek- en speurwerk in gebrekkige documentatie en de benodigde tijd voor besprekingen over onduidelijkheden en fouten kan aanzienlijk zijn en bovendien leiden tot tal van irritaties.

type applicatie

Een belangrijke vraag die men zich dient te stellen bij het schatten van tijd en kosten voor het ontwikkelen van software is: voor welk toepassingsgebied moet de software worden ontwikkeld? In een aantal begrotingsmodellen blijkt deze factor een belangrijke invloed te hebben op de hoogte van de kosten (zie tabel 5.1).

In de literatuur treft men talloze typologieën van software aan. Ook in hoofdstuk 1 is al een verdeling in "soorten" software gegeven. Het onderscheiden van soorten software is in zoverre van belang omdat software van het ene type moeilijker te maken is dan software van een ander type en er andere vaardigheden van de ontwikkelaar voor nodig zijn. Ook kan de produktiviteit, als men deze uitdrukt in het aantal regels code per tijdseenheid, per type applicatie erg verschil-

len.
Jones (1986) laat zien wat het een en ander betekent voor de ontwikkeling van twee verschillende typen applicaties. De ene applicatie betreft de ontwikkeling van een batchgeörienteerde toepassing van een salarisadministratie voor een klein bedrijf. De tweede applicatie heeft betrekking op de ontwikkeling van een real-time toepassing ten behoeve van een luchtverkeersgeleidingsysteem. Beide applicaties worden ontwikkeld in de programmeertaal C. Er wordt van uitgegaan dat het aantal regels code voor beide programma's 5000 is. De projecten worden uitgevoerd door ervaren ontwikkelaars; er worden geen externen bij betrokken; de eisen die gesteld worden aan de documentatie verschillen niet noemenswaardig. Jones berekent dat de totale inspanning voor het salarissysteem 24,1 mensmaanden is en de productiviteit 207 regels code per maand bedraagt. Voor het andere programma zijn deze waarden 36,1 mensmaanden en 150 regels code. De verschillen in productiviteit en benodigde inspanning zijn in dit geval, volgens Jones, terug te voeren tot verschillen in typen applicaties.

Boehm (1981) heeft de factor type applicatie niet als aparte cost driver in het model COCOMO opgenomen, omdat deze factor een overlap heeft met de 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 een aantal onderzoeken wijst op de belangrijke invloed die de factor type applicatie op de hoogte van de kosten heeft. De factor type applicatie blijkt verder sterk gerelateerd te zijn aan andere factoren. Dit werd duidelijk gesignaleerd door Cuelenaere, van Genuchten en Heemstra (1987). Zij hebben een expert systeem ontwikkeld om de begroter te ondersteunen bij de invoerbepaling van het model PRICE-SP. Type applicatie is daarbij een van de invoervariabelen. Bij het ontwikkelen van het expert systeem was een belangrijke stap het vergaren en structureren van de kennis over deze invoervariabele type applicatie. Hierbij bleek dat experts bij het bepalen van de waarde van deze factor impliciet rekening hielden met een groot aantal andere factoren. Ook elke expert een eigen typologie van software te hanteren. Van standaardisatie was geen sprake. Deze zaken geven deels een verklaring voor de verschillen tussen de modellen over de (kwalitatieve) invloed van soort toepassing op de ontwikkelkosten van software.

hergebruik

Bij de ontwikkeling van een nieuw programma zal men zich de volgende vraag stellen: welke delen zijn echt nieuw en welke delen niet. Is het mogelijk van reeds eerder ontwikkelde systemen elementen opnieuw te gebruiken bij de ontwikkeling van een nieuw systeem.

In de literatuur wordt op veel plaatsen een warm pleidooi gehouden voor hergebruik (onder andere: McIlroy 1968 Matsumoto 1984, Martin 1985, Jones 1984, Lanergan en Grasso 1984). Sikkel en Van Vliet (1988) maken een onderscheid in hergebruik van:

- **vaste softwarecomponenten** (hergebruik van code),
- **templates** (dit zijn de raamwerken van softwarecomponenten die nog niet af zijn),
- **ontwerpen** (hergebruik van ontwerpen is aan te bevelen in omgevingen waar vaak hetzelfde soort programma's wordt geschreven),
- **architectuur** (bij de compilerbouw vindt toepassing van deze vorm van hergebruik plaats.

C. Jones (1984) ziet, evenals Sikkel en Van Vliet mogelijkheden voor hergebruik op een aantal nivo's, te weten:

- hergebruik van data,
- hergebruik van architectuur,
- hergebruik van ontwerpen,
- hergebruik van modulen (Macros, subroutines, e.d.).
- hergebruik van programma's (bijvoorbeeld standaardpakketten).

Het zijn niet alleen de directe kostenbesparingen die optreden doordat het eindprodukt sneller kan worden gemaakt. 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 hebben bewezen. De onderhoudskosten voor dergelijke componenten zullen daardoor lager zijn.

Tegenover voorgaande voordelen, in termen van besparingen, staan onder andere de volgende nadelen c.q. kosten:

- er moet een catalogussyteem worden opgezet waarin reeds eerder ontwikkelde produktcomponenten vermeld worden. De ontwikkelaar moet gemakkelijk uit die catalogus kunnen afleiden of de betreffende component wel of niet bruikbaar is;
- er moet een database worden opgezet waarin de verschillende

softwarecomponenten zijn opgeslagen;
- een deel van de ontwikkeltijd zal nu gaan zitten in het raadplegen van het catalogussysteem en het zoeken naar bruikbare componenten;
- niet alleen de selectie van bruikbare, bestaande componenten zal tijd vergen, ook de onderlinge koppeling en inpassen in de nieuw ontwikkelde software zal een extra tijd- en dus kostenfactor zijn.
- ontwikkelt men systemen met als nevendoel de mogelijkheid van hergebruik, dan zal bij de ontwikkeling een zwaarder accent komen te liggen op modulaire aanpak, zodanig dat deze modules voor een eventueel hergebruik geschikt zijn;

De verwachtingen zijn dat de voordelen die met behulp van hergebruik kunnen worden behaald, de bovengenoemde kosten zullen overtreffen. Zo verwacht C. Jones (1986) dat in de toekomst circa 80% van elk nieuw te ontwikkelen programma door hergebruik van reeds eerder ontwikkelde componenten tot stand zal komen. Dit kan enorme kostenbesparingen tot gevolg hebben: een verkorting van de doorlooptijd voor projecten, kostenbesparingen van 50% of meer per project, en bijkomende voordelen zoals het terugdringen van de application backlog en het (geleidelijk) ontstaan van een verzameling standaardcomponenten. Dit laatste voordeel kan verstrekkende gevolgen hebben. Zo zal er binnen de verzameling standaardcomponenten bijvoorbeeld slechts *een* module bestaan om programmafunctie A uit te voeren. Als in een nieuwe applicatie deze functie A voorkomt, dan zal de ontwikkelaar verplicht zijn de betreffende module te gebruiken. Het gevolg hiervan zal zijn dat er niet een wildgroei van allerlei prive-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 en voor eisen die aan de documentatie worden gesteld. Dit alles kan op den duur leiden tot grotere standaardisatie van software-ontwikkeling. Een en ander wordt bevestigd door een onderzoek van Lanergan en Grasso (1984). Zij stellen dat er eigenlijk maar een paar verschillende basisoperaties te onderscheiden zijn in zakelijke COBOL-programma's. Na analyse van een groot aantal bestaande programma's, komen zij tot een zevental logische basisstructuren. Toepassing van deze structuren bij het maken van nieuwe software leerde dat het grootste voordeel ligt in het onderhoud. Bij onderhoud wordt de programmeur minder met onbegrijpelijke software geconfronteerd. Begrijpen en aanpassen van bestaande software wordt zo stukken eenvoudiger.

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 het opnemen van de factoren "% new design" en "% new code", is het opmerkelijk dat bijvoorbeeld Boehm deze factor niet noemt. Vermoedelijk dat de nieuwheid van het verschijnsel hergebruik hier een verklaring voor is. Dit vermoeden wordt bevestigd door uitspraken van Boehm in meer recente publicaties (Boehm en Standish 1983 en Boehm 1984). Hierin pleit hij voor een verfijning van schattingsmodellen met onder meer de factor hergebruik.
In het voorgaande is gesignaleerd dat de factor "hergebruik" een belangrijke invloed heeft op de productiviteit van software-ontwikkeling. De verwachtingen zijn dat die invloed in de toekomst sterk zal toenemen. Hergebruik is echter geen sinecure en vergt ondermeer een ander visie op software-ontwikkeling.

5.4. MIDDELEN-AFHANKELIJKE FACTOREN

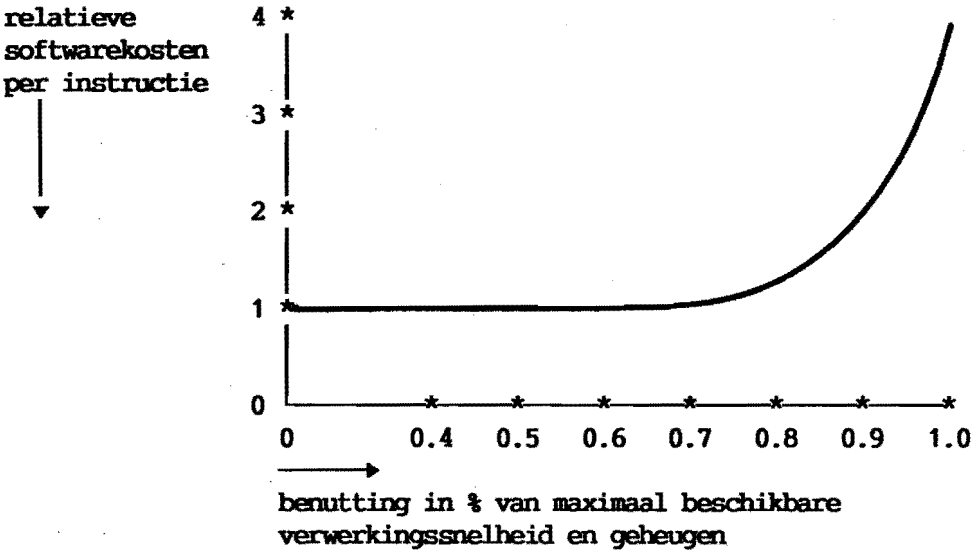
Tot deze categorie van factoren dienen te worden gerekend:

a. beperkingen voor wat betreft de executietijd, de responsietijd 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, responsietijd en geheugencapaciteit***

In vrijwel alle modellen voor het schatten van ontwikkelkosten van software worden deze cost drivers onderscheiden.
Dat de invloed van dergelijke factoren aanzienlijk kan zijn, blijkt uit figuur 5.4, ontleend aan het PRICE-S model (Sierevelt 1986).

Over de grootte van de invloed lopen de meningen opnieuw uiteen. Dit wordt geïllustreerd aan de hand van tabel 5.8 (Boehm 1981). Hierin wordt voor een aantal onderzoeken/modellen, de invloed van beperkingen in executietijd en geheugencapaciteit op de produktiviteit gegeven. De getallen in de tabel zijn een maat voor de verschillen in productiviteit weer. Bijvoorbeeld: in het model van Wolverton is de produktiviteit bij geen beperkingen in executietijd driekeer 3 zo hoog als bij sterke beperkingen. De verschillen zijn soms aanzienlijk.



Figuur 5.4 : De invloed van hardware beperkingen op de softwareontwikkelkosten. Naarmate men meer tegen de "grenzen" van de hardware aanloopt, nemen kosten sterk toe.

Oorzaken hiervan zijn onder andere verschillende definities van geringe en sterke beperkingen *en* verschillen in voltooide projectgegevens, die als basis voor de betreffende onderzoeken dienden.

Tabel 5.8 : De invloed van beperkingen in executietijd en geheugen-capaciteit op de productiviteit.

NAAM ONDERZOEK /MODEL	INVLOED VAN BEPERKINGEN IN EXECUTIETIJD OP DE PRODUCTIVITEIT	INVLOED GEHEUGENBEPERKINGEN OP DE PRODUCTIVITEIT
TRW (Wolverton 1974)	3,00	-
BOEING (Black c.s 1977)	3,33 - 6,67	-
DOTY (Herd c.s. 1977)	1,33 - 1,77	1,43
GRC (Carriere en thibodeau 1979)	1,60	7,00
GTE (DAL 1979)	1,25	1,25
Walston en Felix (1977)	1,37 - 1,77	2,03
COCOMO (Boehm 1981)	1,66	1,56

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. In een aantal onderzoeken (onder andere: Martin 1985, Jones 1986, van der Ven 1988, Howard 1988, Boehm 1988, Verner en Tate 1988, Verberne 1988, Bartelink 1988) wordt gesignaleerd dat het gebruik van met name vierde generatie talen bij de ontwikkeling van software een aantal besparingen oplevert: minder regels code, minder fouten, duidelijker programma's en minder moeilijk onderhoudbare programma's. Een en ander laat zich vertalen in een produktiviteitsstijging en als gevolg daarvan lagere ontwikkelkosten (van der Van 1988, Howard 1988). Boehm (1988) haalt in dit kader een onderzoek aan dat is uitgevoerd door UCLA. Hierin werd de productiviteit gemeten van beginnende en ervaren programmeurs. Beide groepen moesten met behulp van COBOL en het vierde generatiehulpmiddel FOCUS een aantal eenvoudige en een aantal complexe (administratieve) applicaties ontwikkelen. Uit het onderzoek bleek dat de niet-ervaren programmeurs vooral bij de ontwikkeling van complexe applicaties een grote productiviteitstoename realiseerden (een toename met een factor 8.1, voor eenvoudige applicaties lag deze factor tussen de 2.2 en 5.0). Bij ervaren programmeurs bleek juist het tegenovergestelde het geval: een sterke toename van de produktiviteit bij eenvoudige applicaties (met een factor 6.0) en een geringe bij complexe toepassingen (factor 1.5). Een conclusie die men hieruit mag trekken is dat vierde generatiehulpmiddelen niet zondermeer leiden tot een hogere productiviteit. De mate van produktiviteitsverandering wordt sterk bepaald door factoren als ervaringsnivo van de ontwikkelaar, omvang en complexiteit van de te ontwikkelen software enz.

Mizuno (1983) signaleert in zijn onderzoek dat de factor "gebruik van vierde generatiehulpmiddelen" na de factor "personeelskwaliteit" de grootste invloed heeft op de ontwikkelkosten.

Als men toekomstprofeten mag geloven, dan is een reductie van 90% in programmeertijd en een produktiviteitstijging van 100% realiseerbaar (Jones 1986 en Martin 1984). De toekomst zal dit moeten uitwijzen. De stand van zaken op dit moment leert ons evenwel dat:

- het merendeel van dergelijke hulpmiddelen wordt ingezet in de codeerfase van 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 relatief weinig hulpmiddelen beschikbaar zijn, die de

ontwikkelaar ondersteunen in de beginfase van het ontwikkelproces. Deze fasen zijn over het algemeen de meest tijdrovende.
- er praktisch geen hulpmiddelen zijn die het totale ontwikkeltraject ondersteunen. De bestaande hulpmiddelen richten zich op een of een beperkt aantal fasen (de Boer en Vonk 1984).
- bij veel hulpmiddelen het accent ligt op een ondersteuning bij het documenteren. Het aantal geautomatiseerde hulpmiddelen dat een deel of delen van het ontwikkelproces van de ontwikkelaar overneemt, is veel minder.
- er momenteel veel energie wordt gestoken in hulpmiddelen die primair zijn gericht op een inhoudelijke ondersteuning van de systeemontwikkeling - het HOE -. De ontwikkeling van hulpmiddelen die het projectmanagement ondersteunen bij de uitvoering van zijn taken - het WAT - blijft hierbij achter. Denk hierbij aan hulpmiddelen voor plannings- en calculatiedoeleinden, maar ook aan begrotingsmodellen.
- dat het gebruik van hulpmiddelen vooralsnog tegen valt en het nog maar de vraag is in hoeverre ze daadwerkelijk helpen. Deze bewering wordt ondersteund door de resultaten van een empirisch onderzoek, beschreven in hoofdstuk 2. 61% van de responderende organisaties geeft te kennen begrotingshulpmiddelen bij geen enkel project te gebruiken. Slechts 17% gebruikt ze in meer dan de helft van de projecten.

ad c. Het gebruik van moderne programmeertechnieken

In het onderzoek van Walston en Felix (1977) worden vier programmeertechnieken onderscheiden: gestructureerd programmeren, controle van ontwerp en code, top-down ontwikkeling en de opzet van een hoofdprogrammeur-team. In het COCOMO model (Boehm 1981) worden genoemd verificatie- en validatietechnieken, modulair ontwerpen, top-down ontwikkeling, gestructureerd programmeren, walk-throughs software-kwaliteit, programma bibliotheken en schematechnieken. COCOMO onderscheidt niet de opzet van hoofdprogrammeur teams. Boehm benadrukt, evenals James Martin (1984), dat het belangrijk is 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. 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 basisprincipes van software-engineering. De modellen van

Jensen en het model Before You Leap gebruiken dezelfde omschrijving van deze technieken als Boehm in COCOMO.
Jones (1986) noemt structureringsmethoden, als hij het heeft over moderne programmeertechnieken. Hieronder rekent hij structureringsmethoden ten behoeve van het specificeren en analyseren, van het ontwerpen en van het coderen. In tabel 5.9 worden verschillende omschrijvingen van de factor moderne programmeertechnieken in een aantal modellen gegeven. Tevens wordt het effect ervan op ontwikkelkosten gegeven. De getallen in de laatste kolom geven de spreiding in productiviteit/kosten aan. Bijvoorbeeld: Walston en Felix geven aan dat de gemiddelde productiviteit van 169 regels code per maand toeneemt tot 301 regels code bij een toename van weinig naar veel gestructureerd programmeren. De verandering in productiviteit is derhalve : 301 - 169 = 133, de verhouding derhalve 301/169 = 1,78.

Tabel 5.9 : De invloed van het gebruik van moderne programmeertechnieken op de productiviteit/ontwikkelkosten.

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	1,51
Jones	structureringsmethoden	1,38
Jensen JS-2 en 3	gebruik moderne programmeertechnieken	1,45

In recente artikelen (Boehm 1983 en 1987) noemt Boehm als een van de zeven basisprincipes van software engineering het gebruik van moderne programmeertechnieken. Behalve de technieken die in COCOMO worden onderscheiden, noemt hij tevens information hiding. Een belangrijk principe van information hiding is (Parnas 1972, Van Vliet 1988), dat bij het opsplitsen van software in modules, beslissingen over precieze implementatie van gegevens in een module verborgen worden en aldus niet zichtbaar zijn voor andere modulen van de software. Information hiding impliceert dat effectieve modularisatie kan worden gerealiseerd door het definiëren van een verzameling "onafhankelijke" modules. Tussen deze modules wordt slechts die

informatie onderling uitgewisseld die vereist zijn voor de functionaliteit van de software. De voordelen komen met name bij testen en onderhoud naar voren. Aanpassingen in een module blijven zo veel mogelijk beperkt tot die module. Welke gevolgen de toepassing van information hiding heeft op de produktiviteit c.q. de ontwikkelkosten van software, is vooralsnog niet duidelijk aan te geven. Boehm (1987) verwacht dat in de toekomst, door het toepassen van dergelijke technieken en in het bijzonder information hiding, produktiviteitsstijgingen van 40 tot 50% gerealiseerd kunnen worden.

Uit een enquete gehouden onder IBM gebruikers (Guide 1979) 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 vroegtijdig ontdekken van fouten. Dergelijke cijfers geven een aardige indicatie van het gebruik en belang van moderne programmeertechnieken.

Samenvattend kan men stellen dat in de bestaande begrotingsmodellen rekening wordt gehouden met de factor "gebruik van moderne programmeertechnieken". Vaak echter verschillen de genoemde technieken van model tot model. Dit komt ondermeer, omdat de periode waarin de diverse modellen zijn ontwikkeld, verschillen. Technieken van recentere datum treft men wel aan in recenter ontwikkelde modellen en niet in de oudere modellen. Een en ander maakt duidelijk dat aanpassing/calibratie van modellen noodzakelijk is.

5.5. PERSONEELSAFHANKELIJKE FACTOREN

Personeelsafhankelijke factoren als ervaring, vaardigheid e.d. hebben invloed op de ontwikkelkosten van software. In tabel 5.10 is een overzicht gegeven van de personeelsafhankelijke factoren die in een aantal bekende onderzoeken worden onderscheiden.

In het verdere verloop van deze paragraaf wordt aangegeven dat er behalve deze nog meer factoren zijn: factoren die in sommige gevallen een grote invloed kunnen hebben op kosten en doorlooptijd. In het model van Boehm blijkt van alle factoren, de factor "bekwaamheid" veruit de grootste invloed te hebben op de ontwikkelkosten en de factor "ervaring met de toegepaste programmeertaal" de minste invloed.

Tabel 5.10 : De personeelsafhankelijke factoren in verschillende onderzoeken/modellen.

Walston en Felix	COCOMO/Boehm	SPQR/Jones	PRICE-S/RCA
Algehele ervaring en capaciteit	bekwaamheid analist	ervaring met systeemontwikkeling	algehele vaardigheden
ervaring met operationele computer	ervaring met hardware	ervaring met software onderhoud	bekendheid met project
ervaring met programmeertalen	ervaring met programmeertaal	ervaring met type klant	
ervaring met toepassingen van soortgelijke of grotere omvang en complexiteit.	ervaring met applicatie bekwaamheid programmeur	persoonlijke werkomstandigheden	

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. Dat in het onderzoek van Walston en Felix "ervaring met de toegepaste programmeertaal" zo belangrijk blijkt te zijn, hoeft geen verbazing te wekken als men beseft dat de onderzoeksgegevens stammen uit de jaren 1972 tot en met 1976. In die tijd was kennis en ervaring met programmeren een belangrijker produktiviteitsfactor.
C. Jones geeft geen kwantitatieve relaties tussen zijn personeelsafhankelijke factoren en de kosten. Over de inhoud van het model PRICE-S is geen documentatie beschikbaar.

Dat de invloed van genoemde factoren op de ontwikkelkosten belangrijk is, blijkt ook uit het toenemend aantal publicaties over dit onderwerp (Esterling 1980, McKeithen en andere 1981). Laughery en Laughery (1985) geven een uitstekend overzicht van de onderzoeken waarin de relatie tussen personeelsafhankelijke factoren en softwarekosten wordt besproken. Zij komen tot de conclusie dat er nog groot gebrek aan kennis is omtrent dit onderwerp. Zij signaleren dat in weinig begrotingsmodellen rekening wordt gehouden met personeelsafhankelijke factoren, ondanks het feit dat de invloed ervan op de kosten aanzienlijk is (Boehm 1981, Walston en Felix 1977, Mizuno

1983, Jensen 1984 en Jones 1986). De modellen die wel rekening houden met personeelsafhankelijke factoren komen niet veel verder dan een aanzet om zaken als bekwaamheid en ervaring te definiëren en 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/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 goed zijn en de ervaring nog zo breed, de invloed ervan wordt teniet gedaan als er gewerkt moet worden onder onprettige omstandigheden of als er geen begrip of zelfs tegenwerking van bijvoorbeeld het management is. Rivaliteit, een vijandige sfeer, intern politiek gekonkel zijn funest voor het welslagen van een softwareproject. Jones (Jones 1986) beweert dat 10 tot 15% van de softwareprojecten juist om die redenen op een mislukking uitlopen. Matsumoto (1987) gaat nog verder door te beweren dat ideale werkomstandigheden voor analisten en programmeurs de belangrijkste voorwaarden zijn voor een succesvol software-ontwikkelproject. Ideale werkomstandigheden worden gecreëerd door aandacht te besteden aan onder meer werkruimte, management stijl, cultuur binnen de organisatie, salarisnivo, carrièreplanning, opleiding, beschikbaarheid van geautomatiseerde hulpmiddelen en de aanwezigheid van een duidelijke ontwikkelmethode.

Couger en Zawacki (1978) trachten een antwoord te geven op de vraag: Wat motiveert een programmeur/analist? Op basis van een uitgebreid onderzoek onder meer dan 600 analisten en programmeurs komen zij tot de conclusie dat persoonskenmerken, zoals het streven naar zelfactualisatie en het willen afleveren van een hoogwaardig intellectueel produkt, veel belangrijker voor een programmeur en analist zijn dan over het algemeen erkend wordt. Salaris blijkt meestal niet een primaire drijfveer. In het onderzoek van Laughery en Laughery (1985) worden deze bevindingen bevestigd. Dergelijke zaken, die kenmerkend zijn voor het uitvoeren van professionele taken, hebben gevolgen voor de wijze waarop software-ontwikkeling beheerst en begroot moet worden. In hoofdstuk 6 wordt hier nader op ingegaan.

In alle onderzoeken komt naar voren dat de factor "ervaring" een belangrijke kostenbepalende factor is. Wat onder ervaring verstaan moet worden, is niet altijd even duidelijk. Zo treft men in de literatuur de volgende omschrijvingen aan:

- ervaring met soortgelijke applicaties,
- ervaring met de betreffende programmeertaal (met programmeren),
- ervaring met soortgelijke machineconfiguraties,
- ervaring met soortgelijke gebruikers.
- ervaring in de breedte, dat wil zeggen dat de ervaring niet specifiek gericht is op een van de bovengenoemde specialismen, maar betrekking heeft op meerder onderwerpen.

Grofweg kan gesteld worden: hoe meer ervaring, hoe minder ontwikkelkosten. Laughery en Laughery (1985) signaleren evenwel dat er een verzadigingspunt bereikt wordt. Dit betekent dat meer ervaring opdoen, nadat een bepaald ervaringsnivo is bereikt, geen duidelijk effect meer heeft op zaken als produktiviteit, kwaliteit en vaardigheden. Voor de factor "ervaring met programmeren" wordt dit punt (althans voor ervaring met de taal COBOL) na ongeveer twee tot drie jaar 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.

Tabel 5.11 : De invloed van opleiding en ervaring op de productiviteit.

OPLEIDING	JAREN ERVARING MET COBOL	GEMIDDELDE PRODUKTIVITEIT
lagere technische opleiding of diploma inleiding programmeren/informatica	1 4	2,1 7,5
voltooide middelbare schoolopleiding	1 2 3 4 of meer	8,7 8,5 6,8 12,3
voltooide hogere beroepsopleiding	1 2 3 4 of meer	8,9 7,1 9,7 6,3
voltooide universitaire opleiding	1 2 4 of meer	8,6 14,8 5,1

De invloed van de factor "ervaring met soortgelijke gebruikers" is aanzienlijk. Zo zal het verschil uitmaken of men voor de eerste of voor de tiende keer als ontwikkelaar te maken heeft met bijvoorbeeld een onderwijsinstelling.Het 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 en betreffende type probleem. Kwantitatieve onderzoeksresultaten om deze beweringen hard te maken zijn helaas nauwelijks voorhanden, getuige de bevindingen van een onderzoek, waarvan de resultaten in tabel 5.11 zijn weergegeven. Het onderzoek is uitgevoerd door Jeffery en Lawrence (1985) op basis van uitgebreid empirisch materiaal. De tabel laat zien dat de relatie verwarrend is. Op basis van deze cijfers leidt een toename van opleiding en ervaring niet duidelijk tot een productiviteitstoename.

Een personeelsafhankelijke factor, die bepalend is voor de ontwikkeltijd en -kosten, heeft betrekking op de beschikbaarheid van een medewerker voor een project. Met andere woorden: welk percentage van zijn tijd kan hij besteden aan een specifiek project. Abdel-Hamid en Madnick (1987) signaleren in hun onderzoek dat de totale projectkosten toenemen naarmate projectmedewerkers hun tijd moeten verdelen over meerdere projecten. In tabel 5.12 wordt aangeven dat het inzetten van full-time of half-time medewerkers leidt tot een kostentoename van 22%.

Tabel 5.12 : Ontwikkelinspanning bij de inzet van medewerkers die 100% van hun tijd aan het project besteden en medewerkers die dat maar voor de helft van hun tijd doen.

BESCHIKBAARHEID VOOR PROJECT	BENODIGDE INSPANNING
volledig	3.795 mensdagen
helft van de tijd	4.641 mensdagen

Een verklaring voor dit verschijnsel ligt voor de hand. Wil men bij dezelfde doorlooptijd een project realiseren met half-time medewerkers, dan betekent dit de inzet van meer mensen. En een toename van medewerkers in een project leidt, zoals in hoofdstuk 1 is beargu-

menteerd, tot een toename van communicatie en afstemming.

Samenvattend geldt, dat personeelsafhankelijke factoren in de meeste onderzoeken c.q. begrotingsmodellen worden beschouwd als belangrijke, zo niet de belangrijkste kostenbepalende factoren. In tegenstelling tot bijvoorbeeld de produktafhankelijke factoren, zijn de waarden van de personeelsafhankelijke factoren door het projectmanagement beter te beïnvloeden. Bij het beheersen c.q. begroten van software is het daarom belangrijk beïnvloedbare zaken als opleidings- en ervaringsnivo van de medewerkers, kwaliteit van het projectmanagament, werkomstandigheden enz. te optimaliseren.

5.6. 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 ontwikkelkosten als een project sneller klaar moet zijn of langer mag duren dan wat nominaal geschat is. Jones (1986) noemt vier redenen waarom vaak te krappe tijdsplanningen worden opgesteld:

1. commerciële overwegingen (Price To Win). Om een opdracht binnen te krijgen wordt vaak een oplevertermijn genoemd 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. Deze wet zegt dat de tijd die voor de uitvoering van een project wordt uitgetrokken, altijd wordt benut, zelfs als men meer tijd beschikbaar krijgt dan strikt noodzakelijk. Om dit ongewenste effect te vermijden is het management in zo'n geval geneigd een strakke tijdplanning aan te houden om de organisatie zo efficient mogelijk te laten functioneren.

4. schattingsfouten. Als een leveringscontract wordt opgesteld door weinig deskundige en onervaren ontwikkelaars is de kans op een niet haalbare levertermijn groot.

In de modellen COCOMO en PRICE-S en het model van Putnam (1979) wordt rekening gehouden met de invloed op de ontwikkelkosten 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 (Sierevelt 86). 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 de nek", e.d. Ook in het model van Putnam is de invloed van compression enorm. In hoofdstuk 3 is dit reeds aan de orde geweest.
In COCOMO blijkt de invloed van compression veel geringer 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 overigens ook modellen die geen rekening houden met de factor "doorlooptijd", met name het model van Jensen (1984) en de studie van Walston en Felix (1977).

ad b. ***beheersing van het project***

Tot deze factor behoren zaken die betrekking hebben op de wijze waarop het softwareproject wordt beheerst. Hierbij kan men denken aan de manier waarop het softwareproject opgedeeld is in fasen, deze fasen verder opgedeeld zijn in aktiviteiten en deze op hun beurt weer in taken. Verder kan men denken aan de manier waarop in zo'n fasering duidelijke beslispunten zijn aangebracht, waarover op zo'n beslispunt een besluit genomen moet worden, wie die beslissingen neemt en wie daarbij betrokken wordt, welke beslisdocumenten door wie geproduceerd moeten worden en welke informatie deze documenten dienen te bevatten. Andere zaken die hierbij een rol spelen zijn projectdocumentatie en het gebruik van een databank van gegevens van afgesloten projecten. In de hoofdstukken 6 en 7 wordt nader ingegaan op het beheersen c.q. begroten van softwareprojecten. De hierboven genoemde zaken zullen in deze hoofdstukken uitvoeriger aan de orde komen. Concentreren we ons hier op de vraag wat de invloed is van de wijze van projectbeheersing op de ontwikkelkosten, dan blijkt dat in slechts enkele begrotingsmodellen rekening wordt

gehouden met een dergelijke samenhang. Dit geldt voor de modellen Estimacs en in het bijzonder voor Bis-Estimator. Toch ligt het voor de hand dat een juiste fasering van een project, een goede organisatiestructuur, goede procedures, goede communicatie en een juist veranderingsklimaat belangrijke voorwaarden zijn voor een succesvol software-ontwikkelproject. Wordt hieraan niet voldaan dan staat de deur wagenwijd open voor het uit de hand lopen van ontwikkelkosten en -tijd.

Als er in begrotingsmodellen al gesproken wordt over dit soort factoren dan hebben zij veelal betrekking op alleen 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 de kosten als gevolg van beheersactiviteiten over het hoofd gezien.

In begrotingsmodellen van Amerikaanse origine blijkt de factor "reiskosten" een belangrijke cost driver te zijn. Voor de Nederlandse situatie zal een dergelijke kostenfactor een veel mindere betekenis hebben, lettende op de geografische concentratie van organisaties.

Hoewel B.Boehm in (Boehm 1981, Boehm en Papaccio 1988) uitvoerig ingaat op de rol van software planning en -beheersing en de invloed hiervan op de kosten en doorlooptijd, wordt in het model COCOMO bij de projectgebonden factoren geen rekening gehouden met de invloed van wijze van projectbeheersing op de kosten. Ook in het model van Walston en Felix (1977) ontbreekt een dergelijke cost driver. Jones (1986) daarentegen noemt in dit verband de factoren gedistribueerde ontwikkeling en organisatiestructuur. Onder gedistribueerde ontwikkeling verstaat hij het aantal verschillende locaties waarop de software wordt ontwikkeld. In zijn model is de invloed van deze factor gering: een kostentoename van 13,5% wanneer het aantal locaties zes is in plaats van één. In dit geval zijn de locaties wel door middel van een netwerk met elkaar verbonden. Is dit niet het geval dan is de kostentoename aanzienlijk, namelijk 47%. Het effect van de factor organisatiestructuur wordt volgens Jones bepaald door de mate waarin gebruikt wordt gemaakt van een matrixorganisatie. In zijn model wordt verondersteld dat de ontwikkelkosten van een programma, ontwikkeld in een matrixorganisatievorm, 10,5% hoger zijn dan de kosten van hetzelfde programma, ontwikkeld in een hierarchische structuur. Met de interpretatie van dergelijke cijfers moet men voorzichtig zijn. De conclusie mag bijvoorbeeld niet zijn: een matrixorganisatievorm leidt tot hogere ontwikkelkosten, hanteer daarom deze organisatievorm niet. Een verklaring voor de hogere kosten kan zijn dat in een matrixorganisatie meer tijd gaat zitten in

coördinatie en communicatie. Deze extra tijd en kosten kunnen evenwel ten goede komen van een hogere effectiviteit van de te ontwikkelen software.

5.7. GEBRUIKERSAFHANKELIJKE FACTOREN

Deze categorie factoren heeft betrekking op de invloed van de gebruikersorganisatie op de ontwikkelkosten. Men kan hierbij denken aan factoren als:

- in welke mate wordt de gebruiker betrokken bij de ontwikkeling van de programmatuur (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 de 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.,
- hoe stabiel zijn eenmaal opgestelde specificaties.

In het model COCOMO van Boehm wordt slechts met de laatste van de genoemde factoren rekening gehouden. Walston en Felix onderscheiden de factoren:

- deelname van de klant bij definitie van eisen,
- door klant opgestelde wijzigingen in programma-ontwerp en
- ervaring klant met toepassingsgebied.

De eerste factor blijkt in het onderzoek van Walston en Felix de op een na 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. Tegenover deze afname van produktiviteit staat echter een toename van de effectiviteit van de software. Zoals al opgemerkt in de vorige paragraaf is er bij een aantal kostenbepalende factoren een spanningsveld tussen effectiviteit en efficiency. Streeft men bij software-ontwikkeling naar efficiency, dan zal men de gebruiker zo weinig mogelijk betrekken bij de ontwikkeling. Participatie kost immers tijd en geld. Streeft men naar effectiviteit, dan zal men

juist de gebruiker zoveel mogelijk bij de ontwikkeling betrekken. Hoge participatie vergroot immers de kans dat de software bij oplevering daadwerkelijk voldoet aan de eisen van de gebruiker en dat de software binnen de gebruikersorganisatie wordt geaccepteerd. De conclusie is, dat men bij het interpreteren van de invloed op de ontwikkelkosten van een factor als mate van betrokkenheid van de gebruiker, een onderscheid moet maken in het effect op de effectiviteit en efficiency van deze factor.

In tal van onderzoeken (Tait en Vessey 1988, Alter 1987, Swanson 1974) wordt de betrokkenheid van de gebruiker, met name bij aanvang van een project, een belangrijke voorwaarde genoemd voor effectieve software-ontwikkeling. Tait en Vessey (1988) hebben in hun onderzoek voor 59 software-ontwikkelprojecten, afkomstig uit 20 verschillende organisaties onderzocht of een verband bestaat tussen de mate van betrokkenheid en het welslagen van het project. Er blijkt inderdaad verband te bestaan: het succes van de software-ontwikkeling neemt toe, naarmate de betrokkenheid van de gebruiker toeneemt. Dat dit positieve verband niet voor elke situatie geldt, blijkt uit een ander onderzoek van Ives en Olson (1984). Zij hebben voor 22 projecten onderzocht of er sprake is van het bovengenoemd verband. Bij acht projecten bleek er een positief verband te bestaan tussen betrokkenheid van de gebruiker en het succes van software-ontwikkeling, voor zeven projecten kon geen verband worden gevonden en de bij resterende zeven bleek een negatief verband te bestaan. Een duidelijke verklaring konden zij hiervoor niet vinden.

De effectiviteit en efficiency van software-ontwikkeling zal ook 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. Bemelmans en de Boer (1981) signaleren dat het onderschatten c.q. verwaarlozen van de fasen informatie-analyse en specificatie een belangrijke oorzaak hiervan is. Gevolg: enorme herstel/onderhoudsactiviteiten, die nodig zijn om het projectresultaat 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: 60% van de software life-cycle bestaat uit onderhoud en een belangrijk deel hiervan komt voor rekening van de eerder genoemde hersteloperaties (Martin en McClure 1983).

Indien hetzelfde produkt voor meerdere gebruikers ontwikkeld moet worden, zullen de ontwikkeltijd en -kosten toenemen. Zoeken naar de grootste gemene deler in de specificaties en nagaan of eis x van klant A dezelfde betekenis heeft als eis y van klant B, vergroten de complexiteit van het ontwikkelproces. Helaas is er in deze een gebrek aan betrouwbare onderzoeksresultaten.

Als de specificaties gedurende het ontwikkeltraject vaker wijzigen zal dat een aanzienlijk effect op de kosten hebben. Boehm (1981) haalt in dit verband een onderzoek van Schluter (1977) 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 applicatie en de gebruiker niet vertrouwd is met automatisering, is dit gevaar reëel aanwezig. Een andere benadering van software-ontwikkeling, te weten prototyping lijkt in dergelijke gevallen de meest geschikte. Boehm, Gray en Seewaldt (1984) hebben in een experiment de prototyping aanpak vergeleken met de traditionele benadering van software-ontwikkeling. In het experiment moesten zeven groepen mensen de zelfde software ontwikkelen. Drie groepen maakten hierbij gebruik van een prototyping-aanpak en vier groepen hanteerde een traditionele benadering. De voornaamste conclusies waren dat met behulp van prototyping ongeveer 40% minder coderegels en 45% minder mankracht ten opzichte van de traditionele benadering nodig was voor de realisatie van hetzelfde programma. Van de andere kant bleek echter dat met de traditionele aanpak een robuuster en naar verwachting beter te onderhouden programma werd gerealiseerd. Dit 'laboratorium' experiment is nadien, wat betreft de besparing in aantal regels code, bevestigd door ander onderzoek (Jones 1986).

5.8. CONCLUSIES

In dit hoofdstuk is een overzicht gegeven van factoren die van invloed zijn op de ontwikkelkosten van software. Bij het opstellen van een begroting zal men dienen aan te geven welke middelen men moet aanwenden en welke mensen gedurende hoeveeltijd beschikbaar moeten zijn om de gewenste software voor (een) bepaalde gebruiker(s) te realiseren. Dit betekent dat men uitspraken moet doen over de waarden van kostenbepalende factoren; hoe groot wordt de omvang

van de software, welke kwaliteitseisen worden aan de software gesteld; welke tools worden ingezet; voor welke organisatievorm wordt gekozen; welke mensen met welk kwaliteits- en ervaringsnivo worden ingezet; enz. Behalve het onderscheiden van de dominante factoren in een bepaalde ontwikkelomgeving en het aangeven van de waarde van de factoren zal men moeten aangeven wat de invloed van die factoren op de ontwikkelkosten en -tijd zal zijn.
In dit hoofdstuk is naar voren gekomen dat dit geen eenvoudige opgave is. Wil men met succes deze opgave uitvoeren dan zal men oog moeten hebben voor een aantal zaken, te weten:

1. *definiëring*
   Er is een gebrek aan eenduidige definities voor begrippen als produktomvang, softwarekwaliteit, ervaring en bekwaamheid van personeel enz. Het is voor een organisatie belangrijk afspraken te maken over de definiëring van kostenbepalende factoren en zich te houden aan deze afspraken.

2. *kwantificering*
   Het merendeel van de kostenbepalende factoren is moeilijk te kwantificeren. Men maakt daarom veelal gebruik van kwalitatieve maatstaven als veel, gemiddeld, weinig.

3. *objectivering*
   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 wellicht gerekend tot de categorie gemiddeld. Factoren die wel gekwantificeerd kunnen worden, kunnen veelal moeilijk objectief worden gemeten. Het is belangrijk dat men afspraken maakt over het waardebereik van moeilijk te kwantificeren factoren. Bijvoorbeeld, wanneer is er sprake van zeer veel, veel, gemiddeld, enz gebruikersparticipatie. Boehm (1981) geeft een goed voorbeeld van dergelijke omschrijvingen.

4. *correlatie*
   Het is lastig de invloed van één bepaalde kostenfactor op de ontwikkelkosten te bepalen; zelfs voor die factoren die goed kunnen worden gedefinieerd, gemeten en gekwantificeerd.
   Bij de bespreking van de kostenbepalende factoren is aangegeven dat de factoren elkaar wederzijds beinvloeden. Zo kan een, op het eerste gezicht weinig dominante cost driver, door zijn invloed op andere cost drivers, indirect de hoogte van kosten in belangrijke mate bepalen.

5. *de relatie factor-kosten*
Bij de ontwikkeling van een nieuw programma zal men *vooraf* een schatting willen maken van de kosten en de duur. Daarvoor is het onder meer nodig schattingen te maken van de invloed van de kostenbepalende factoren op de ontwikkelkosten en -tijd. Bijvoorbeeld: als de omvang van de te ontwikkelen software wordt geschat op x regels code, de hoeveelheid documentatie op y, de complexiteit op "zeer hoog", de mate van gebruikersparticipatie als "laag", het ervaringsnivo van de analisten als "gemiddeld", enz. , dan zullen de ontwikkelkosten van de software p zijn.

6. *calibratie*
Het is niet mogelijk te spreken van "de belangrijkste kostenbepalende factoren". In de ene ontwikkelomgeving kunnen andere factoren kostendominant zijn dan in een andere omgeving. Het is daarom van belang dat een organisatie door middel van analyse van afgesloten software-ontwikkelprojecten nagaat welke factoren in welke mate de kosten beïnvloeden. De samenstelling en de invloed kan echter van project tot project variëren en zal in de loop van de tijd veranderen. Dit betekent dat voor elk nieuw project onderzocht moet worden in hoeverre kennis en ervaring van afgesloten projecten representatief is voor dit nieuwe project. Verder zal men voortdurend samenstelling en invloed van de factoren moeten actualiseren (hercalibreren).

7. *effectiviteit en efficiency*
In dit hoofdstuk is gewezen op het spanningsveld tussen effectiviteit en efficiency. Bij het begroten is het belangrijk een duidelijk onderscheid te maken tussen deze twee zaken. Vanuit efficiency-overwegingen lijkt het zinvol gebruikersparticipatie te minimaliseren, terwijl vanuit effectiviteitsoverwegingen men vaak streeft naar maximaliseren van gebruikersparticipatie.

8. *"human factors"*
In diverse onderzoeken wordt gesignaleerd dat de invloed op de ontwikkelkosten van menselijke factoren als kwaliteit en ervaring personeel, werkomstandigheden enz. belangrijk is. Investeren in "goed" personeel en "ideale" werkomstandigheden is belangrijk.

9. *hergebruik*
De invloed van hergebruik van modules, subroutines e.d. op de produktiviteit bij software-ontwikkeling zal steeds groter worden. In een aantal onderzoeken (Druffel 1982, Conte Dunsmore en Shen 1986) wordt hergebruik genoemd als de factor waarmee in de negentiger jaren veruit de grootste produktivitietswinst te behalen

valt.

Voor het beheersen en begroten van software-ontwikkeling is inzicht vereist in factoren die de ontwikkelkosten bepalen. Hoe groter dit inzicht, des te beter is men in staat software-ontwikkeling te beheersen en des te nauwkeuriger kan men begroten. Dit inzicht zal voor elke ontwikkelomgeving en voor elk softwareproject verschillen. In de ene situatie kan men bijvoorbeeld bij de start van de ontwikkeling het eindresultaat precies omschrijven. Het inzicht in de produktafhankelijke factoren is in zo'n situatie veel hoger dan wanneer de specificaties van de te ontwikkelen software onduidelijke en onvolledig zijn. In weer een andere situatie is de onzekerheid over de in te zetten middelen en de beschikbaarheid van personeel groot. Men heeft geen duidelijk idee op welke wijze men de software-ontwikkeling gaat uitvoeren. In zo'n situatie is er onduidelijkheid over ondermeer de project- en middelenafhankelijke factoren.
De conclusie die men hieruit kan trekken is dat de samenstelling van cost drivers, de invloed ervan op de kosten en de zekerheid waarmee men samenstelling en invloed kan aangeven, situatie-afhankelijk is. In hoofdstuk 7 worden vier verschillende situaties onderscheiden en wordt voor elke situatie aangegeven op welke wijze softwareprojecten beheerst en begroot kunnen worden.

6. BEGROTEN EN BEHEERSEN

6.1. INLEIDING

Een centraal thema in de eerste vijf hoofdstukken is het begroten van software-ontwikkeling. In hoofdstuk 1 is in dit kader een aantal problemen en oorzaken genoemd. Een en ander is geïllustreerd met resultaten van een empirisch onderzoek, die in het tweede hoofdstuk zijn beschreven. In de hoofdstukken 3 en 4 zijn begrotingsmethoden en -modellen behandeld, waarbij naar voren kwam dat deze methoden en modellen maar een beperkt hulpmiddel zijn voor het beheersen en begroten van software-ontwikkeling. In hoofdstuk 5 tenslotte is aangegeven dat een groot aantal faktoren van invloed is op de softwarekosten. Het is bepaald niet eenvoudig deze faktoren te identificeren, te definiëren en te operationaliseren.

Waren de eerste vijf hoofdstukken vooral inventariserend en probleembeschrijvend van aard, de hoofdstukken 6, 7 en 9 zijn voornamelijk oplossingsgericht. Het heeft mijns inziens op dit moment weinig zin aan de bestaande verzameling van methoden en modellen nieuwe toe te voegen. Alvorens men met succes methoden en modellen kan gebruiken, zal men allereerst een beter inzicht moeten hebben in de besturingssituatie van software-ontwikkeling. De hoofdstukken 6 en 7 spitsen zich toe op die besturingssituatie.

Hoofdstuk 6 omvat een beschrijving van een besturingsmodel voor software-ontwikkeling. Hierbij is gebruik gemaakt van het besturingsparadigma (de Leeuw 1974). Nadat in paragraaf 6.2 het besturingsparadigma is toegelicht, worden in paragraaf 6.3 voorwaarden genoemd waaraan voldaan moet worden, wil er sprake zijn van effectieve besturing. In de paragrafen 6.4 wordt nader ingegaan op de verschillende aspecten van besturing. Zo wordt een onderscheid gemaakt in een aantal verschillende besturingsproblemen en wordt aangegeven dat er verschillende managementstijlen en coördinatiemechanismen bij besturing mogelijk zijn. Verder wordt aangegeven dat bij software-ontwikkeling doorgaans sprake is van een project-shop en dat er

verschillende manieren zijn om een project-shop te besturen. Tenslotte wordt gesignaleerd dat bij besturing sprake kan zijn van verschillende manieren van begroten en dat de funktie van een begroting verschillend kan zijn. Het hoofdstuk wordt afgesloten met conclusies 6.5.

De hoofdstukken 6 en 7 zijn onlosmakelijk met elkaar verbonden. De analyse van besturing in hoofdstuk 6 zal worden gebruikt om in hoofdstuk 7 verschillende besturingsvormen voor software-ontwikkeling te onderscheiden.

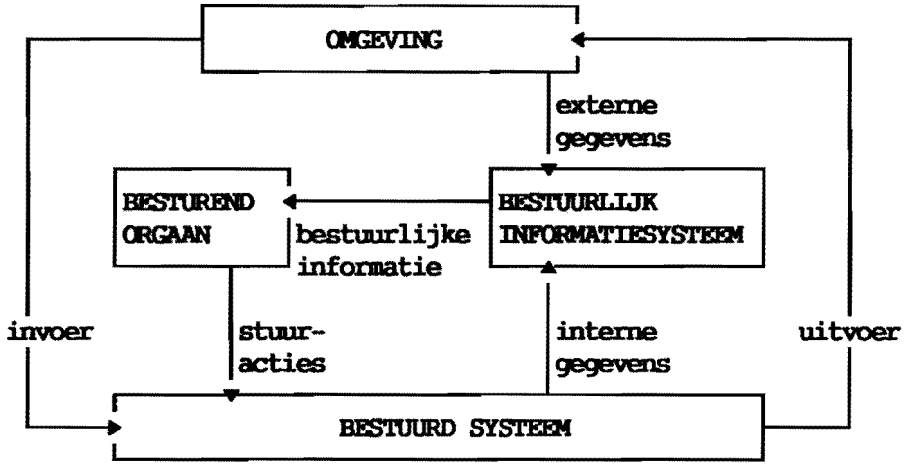
6.2. HET BESTURINGSPARADIGMA

Binnen de systeemleer is het gebruikelijk een besturingssituatie te beschrijven in termen van (de Leeuw 1974):

- een *Bestuurd Systeem* (BS). Dit is het systeem dat bestuurd moet worden.
- een *Besturend Orgaan* (BO) ook wel beslissingssysteem genoemd. Met dit systeem worden de, al of niet menselijke beslissers bedoeld, de organisatie van die beslisser en de beslisregels.
- een *Bestuurlijk Informatiesysteem.* Dit is het systeem dat de benodigde informatie verzamelt, verwerkt, bewaart en beschikbaar stelt met behulp waarvan het Besturend Orgaan beslissingen kan nemen en sturing kan uitoefenen. Een Bestuurlijk Informatiesysteem heeft twee gegevensbronnen: interne waarnemingen van het Bestuurd Systeem en externe waarnemingen van de omgeving van het Bestuurd Systeem.

In figuur 6.1 is de samenhang tussen de genoemde systemen weergegeven.

Wat men als Besturend Orgaan en Bestuurd Systeem wil beschouwen, is afhankelijk van het doel van het onderzoek. Het Bestuurd Systeem kan het totale software-ontwikkelproject c.q. projectorganisatie betreffen, maar kan ook beperkt blijven tot een bepaalde fase c.q. activiteit. Besturend Orgaan kan zijn het gehele besturingsapparaat ofwel onderdelen daarvan zoals een stuurgroep, een projectleider, een projectteam, een analist, enz.
Wil het Besturend Orgaan in staat zijn stuuracties uit te oefenen, dan dient het te beschikken over (bestuurlijke) informatie.



Figuur 6.1 : Schema van een besturingssituatie

Deze informatie wordt geleverd door het Bestuurlijk Informatiesysteem. Het Bestuurlijk Informatiesysteem heeft tot taak relevante gegevens van de omgeving en het Bestuurd Systeem te verzamelen, vast te leggen, op te slaan en te verwerken tot bestuurlijke informatie. Het Bestuurd Systeem tenslotte heeft als taak de invoer om te zetten in beoogde uitvoer.

Nemen we als voorbeeld de besturingssituatie van een projectleider als object van onderzoek. De projectleider is dan Besturend Orgaan en het software-ontwikkelingsproject dat uitgevoerd moet worden het Bestuurd Systeem. Afhankelijk van de bevoegdheden van de projectleider kunnen invoer van het Bestuurd Systeem zijn: toegewezen mankracht, het ziekteverzuim en het verloop van personeel, de automatiseringskennis van de gebruiker etc. Dit soort invloeden kunnen door het Besturend Orgaan niet of moeilijk beheerst worden. Men spreekt in dit verband van *irreguliere variabelen*. De uitvoer van het Bestuurd Systeem is de software, de documentatie, de opleiding enz. die geleverd moeten worden. De projectleider zal bij het besturen van het Bestuurd Systeem bepaalde doelstellingen nastreven, bijvoorbeeld maximalisatie van de kwaliteit of minimaliseren van de doorlooptijd. Dit soort variabelen die door een projectleider worden nagestreefd, worden *doelvariabelen* genoemd. De projectleider kan door middel van stuuracties invloed uitoefenen op het Bestuurd Systeem. Voorbeelden

hiervan zijn verandering van het aantal hulpmiddelen, de organisatie van het project, het reviewen van kostenresultaten etc. Dit worden *stuurvariabelen* genoemd. Voorbeelden van bestuurlijke informatie, op grond waarvan de projectleider sturing uitoefent, kunnen zijn de beslisdocumenten op de overeengekomen meetpunten, plannings-overzichten, voortgangsrapportages, overzichten van bestedingen qua geld en tijd enz. Dit soort bestuurlijke informatie wordt verkregen uit interne gegevens : bijvoorbeeld de projectadministratie (registratie van gegevens van het project in uitvoering), en uit de externe gegevens : produktiviteitsgegevens van een nieuw (geautomatiseerd) hulpmiddel, gegevens over de opdrachtgever enz.

Het is onmogelijk een rigide, steeds geldend onderscheid te maken tussen irreguliere variabelen, stuurvariabelen en doelvariabelen. Het is sterk afhankelijk van de betreffende situatie of een variabele bijvoorbeeld beschouwd moet worden als een irreguliere variabele of als een stuurvariabele. Gaat men uit van gegeven en uitgekristalliseerde specificaties, dan zal men door de inzet van adequaat personeel, hulpmiddelen, tijd en geld, invloed uitoefenen om de specificaties te realiseren. In de praktijk blijkt de geschetste situatie vaak niet zo duidelijk aanwezig te zijn. Tijdens de uitvoering van een project zullen, in overleg met de opdrachtgever, voortdurend bijstellingen in de functionaliteit en het prestatieniveau van de software plaatsvinden. Aspecten als geld en tijd kunnen dan irreguliere variabelen zijn. Men zal dan uitgaan van een gegeven budget en een gegeven periode voor ontwikkeling en zich vervolgens afvragen welke mogelijkheden er zijn, welke gebruikersinterfaces voor dat geld verwacht mogen worden, welke responsiesnelheid, welke betrouwbaarheid enz. In dit voorbeeld zijn tijd en geld dus irreguliere variabelen en zijn functionaliteit en kwaliteit stuurvariabelen.

Uit het voorgaande moge duidelijk zijn geworden zijn dat het voor beheersen en begroten van een specifiek softwareproject van belang is aan te geven wat de doelvariabelen, irreguliere en stuurvariabelen zijn. Het moet duidelijk zijn wat men wel en wat men niet kan beïnvloeden, waar wel en niet beheersing mogelijk is. Voor een aantal kostenbepalende factoren is dat meestal duidelijk. Bijvoorbeeld: bij de ontwikkeling van een programma kan men invloed uitoefenen op de betrokkenheid van de gebruiker en het aantal gebruikers. Men kan echter weinig of geen sturing (op korte termijn) uitoefenen op het ervaringsniveau van de gebruikers met automatisering. Zaken als overhead en eisen die aan de projectduur worden gesteld, zijn eveneens vaak minder makkelijk te beïnvloeden. Factoren als kwaliteit en ervaring van het in te zetten personeel zijn alleen op de lange termijn te reguleren. Bij de uitvoering van een specifiek project kan

men alleen door het kiezen van het juiste team voor dit project invloed uitoefenen op dergelijke personeelsafhankelijke factoren. Middelenafhankelijke factoren zijn over het algemeen eenvoudiger te beïnvloeden. Men kan bijvoorbeeld kiezen welke tools men wenst in te zetten en van welke methoden en technieken men gebruik wenst te maken.

In de hierboven beschreven besturingssituatie is de samenhang tussen het Bestuurd Systeem, het Besturend Orgaan en het Bestuurlijk Informatiesysteem aangegeven. Dit besturingsparadigma zal gebruikt worden als hulpmiddel om met name het Besturend Orgaan nader te analyseren. Besturing staat in dit paradigma centraal. Bij deze analyse zal ik gebruik maken van de voorwaarden voor effectieve besturing, zoals die in de systeemleer (de Leeuw 1974) worden geformuleerd. Het paradigma schiet te kort als het erom gaat besturing te zien als het resultaat van een contingentie benadering waarbij, de wijze van besturing wordt beschouwd als een ontwerpvraagstuk. De P-B-I benadering biedt hiervoor betere mogelijkheden. In hoofdstuk 7 wordt hierop nader ingegaan.

6.3. VOORWAARDEN VOOR BEHEERSEN EN BEGROTEN

De voorwaarden voor effectieve besturing, zoals die in de systeemleer worden geformuleerd, zijn:

- het Besturend Orgaan moet geïnformeerd zijn over de *doelstellingen* van het Bestuurd Systeem, omgevingsinvloeden en beslissingsalternatieven.
- het Besturend Orgaan moet beschikken over voldoende stuurmaatregelen (veelal aangeduid met voldoende *besturingsvariëteit*).
- het Besturend Orgaan moet beschikken over *informatie* over de toestand, de invoer en de uitvoer van het Bestuurd Systeem.
- het Besturend Orgaan moet beschikken over een *conceptueel besturingsmodel.* Men moet weten op welke wijze en in welke mate de diverse variabelen samenhangen en elkaar over en weer beïnvloeden. In zo'n model moet precies worden weergegeven welke effecten een bepaalde input en bepaalde stuurmaatregelen hebben.

Wanneer aan al deze voorwaarden in optima forma is voldaan, spreekt men van strikt rationele besturing. In zo'n besturingssituatie is geen sprake van onzekerheid, het Besturend Orgaan is volledig geïnfor-

meerd over alles wat van belang is. Het besturingsprobleem is geheel structureerbaar en formaliseerbaar. Sturen is in een dergelijke situatie relatief eenvoudig. In de praktijk van software-ontwikkeling wordt aan deze voorwaarden meestal niet voldaan. De doelstellingen zijn niet volledig bekend, zijn vaag en zijn veelal aan verandering onderhevig. Het Bestuurlijk Informatiesysteem functioneert slecht of onvoldoende en wordt niet of in onvoldoende mate gevoed met gegevens over het project in uitvoering. Er is gebrek aan voldoende stuurmogelijkheden en men heeft onvoldoende inzicht in de effecten van stuuracties. Afhankelijk van de mate waaraan voldaan wordt aan de genoemde voorwaarden, wordt in de literatuur een onderscheid gemaakt in rationele besturing, intuïtieve besturing en primitieve besturing (Bemelmans 1987). In tabel 6.1 is een korte samenvatting gegeven van deze soorten besturingssituaties.

Tabel 6.1 : Soorten besturingssituaties

Rationele besturing	Doelstellingen bekend, stuurmaatregelen beschikbaar, perfect informatiesysteem en besturingsmodel. Sturing vindt plaats op grond van rationele, formaliseerbare regels.
Intuïtieve besturing	Doelstellingen zijn wazig, besturingsmodel en/of informatiesysteem beperkt, gebrekkig inzicht in effecten van bijsturing. Sturing gebeurt op grond van intuïtie en ervaring.
Primitieve besturing	Doelstellingen zijn (deels) onbekend. Er is weinig inzicht in het besturingsmodel, het informatiesysteem is onvolledig. Sturing gebeurt op grond van (niet beargumenteerde) vuistregels.

In deze paragraaf zal ik nader ingaan op de voorwaarden voor effectieve besturing en zal ik aangeven in hoeverre bij de ontwikkeling van software wordt voldaan aan deze voorwaarden. Er zal worden

beargumenteerd dat er een andere vorm van beheersing nodig is afhankelijk van de mate waarin de doelstellingen van de programmatuur eenduidig zijn, men beschikt over informatie over de invoer, uitvoer en toestand van het ontwikkelproject, de resultaten meetbaar zijn, men meer mogelijkheden heeft tot sturing en inzicht heeft in de gevolgen van bijsturing.

6.3.1. DOELSTELLINGEN

Een belangrijke voorwaarde om een project te kunnen beheersen, is de aanwezigheid van duidelijke doelstellingen. Bij de ontwikkeling van software kan men een aantal conflicterende doelstellingen onderscheiden. Voorbeelden daarvan zijn:

- **Minimaliseer de doorlooptijd.** Het accent bij de ontwikkeling ligt op de leversnelheid. Een en ander wordt in de hand gewerkt doordat software in veel gevallen onder hoge tijdsdruk ontwikkeld moet worden. Dit wordt ondermeer veroorzaakt doordat de vraag naar software het aanbod ver overtreft. In hoofdstuk 1 is hier reeds op gewezen.
- **Maximaliseer de efficiency.** Staat de efficiency centraal, dan betekent dit dat zo goedkoop mogelijk ontwikkeld moet worden. Het accent ligt nu op een optimale benutting van de middelen.
- **Maximaliseer de kwaliteit van het eindresultaat.** Dit betekent dat de software zo goed mogelijk moet aansluiten bij de functionele- en prestatie eisen van de gebruikende organisatie.

Deze doelstellingen staan in een spanningsveld ten opzichte van elkaar. Ligt het accent op de leversnelheid, dat wil zeggen op een snelle afhandeling, dan zal men vanuit beheersoptiek overcapaciteit moeten aanhouden. Overcapaciteit bij software-ontwikkeling is, zoals in het verdere verloop van dit hoofdstuk zal blijken, echter kostbaar. Ligt het accent op de efficiency, dan zal men er juist naar streven overcapaciteit te vermijden en de middelen zo optimaal mogelijk te benutten. Tracht men beide doelstellingen tegelijkertijd te realiseren dan levert dit problemen op. Deze worden nog groter als men tevens een maximale kwaliteit van het eindresultaat tracht te realiseren. Wat betreft de derde doelstelling kan vermeld worden dat het doorgaans moeilijk is de eisen, waaraan de software moet voldoen, precies te omschrijven.

6.3.2. BESTURINGSVARIETEIT

De beheersbaarheid van een softwareproject neemt toe naarmate er meer mogelijkheden zijn voor bijsturing. Bij het ontwikkelen van software is de mate van besturingsvariëteit situatie-afhankelijk. Ik wil twee extreme situaties schetsen. In de realiteit zullen allerlei mengvormen voorkomen. In het ene geval zal de werkelijkheid neigen naar het ene uiterste van het continuüm en in het andere geval naar het andere uiterste.

Situatie 1.
begroting ligt vast. Maximaliseer het resultaat.
De specificaties zijn vaag en onvolledig. Het project moet vooralsnog uitgevoerd worden met een gegeven budget, ontwikkeltijd en middelen. Voor de ontwikkeling van bijvoorbeeld een geautomatiseerd studentenadministratiesysteem kan dit betekenen dat men zich afvraagt welke mogelijkheden er zijn, gegeven een budget van x gulden. Welke kwaliteit mag men voor dat bedrag verwachten. In deze situatie beschikt men over mogelijkheden om bij te sturen in de functionaliteit en kwaliteit van de software.

Situatie 2.
resultaat ligt vast. Optimaliseer gebruik middelen of minimaliseer levertijd.
Een geheel andere situatie treedt op als de specificaties vast liggen en men middelen, geld en tijd moet toewijzen om een programma conform de gestelde specificaties te realiseren. Sturing op kwaliteit is nu niet mogelijk. In deze situatie zal men sturen (indien mogelijk) door middel van de inzet van extra personeel van een bepaald niveau, beschikbaar stellen van hulpmiddelen e.d. Naarmate minder aan de kwaliteit van het eindresultaat getornd mag worden, budget en tijdstip van levering meer vast liggen en schuiven met capaciteit door bijvoorbeeld extra personeel aan het project toe te voegen minder mogelijk is, worden de mogelijkheden om het project te beheersen beperkter. Begroten is hier natuurlijk altijd mogelijk, maar heeft weinig zin als de uitkomst van een begroting slechts in beperkte mate kan leiden tot een of andere vorm van bijstelling.

In de praktijk treft men vele varianten aan van de hier geschetste (extreme) situaties.

6.3.3. BESTURINGSMODEL

Beheersen van softwareprojecten impliceert dat het mogelijk moet zijn effectief stuuracties uit te oefenen. Stuuracties kunnen nodig zijn als de voortgang van het project niet in overeenstemming is met het plan of als de werkelijke uitgaven en inspanning afwijken van de begroting of als de software niet voldoet aan de geformuleerde eisen. Effectief bijsturen betekent dat bijvoorbeeld bekend moet zijn wat het effect is van de inzet van extra personeel, van werkvoorbereidingsacties, van een verhoging of verlaging van het budget, van verkorting van de doorlooptijd of het gebruik van een vierde generatiehulpmiddel.
Bij de ontwikkeling van software blijkt het lastig te zijn de gevolgen van stuurmaatregelen juist in te schatten. Voor software geldt dat bijsturing van functionaliteit en prestatieniveau vaak het verschijnsel geeft dat een kleine bijsturing leidt tot grote gevolgen. Een ogenschijnlijk kleine toevoeging aan de lijst van eisen kan resulteren een veel extra werk. Verder blijkt doorgaans bij software dat een verbetering van de ene programmafout een reeks nieuwe fouten tot gevolg kan hebben.
Een besturingsmodel impliceert naast inzicht in de effecten van allerlei stuuracties, inzicht in de stand van zaken van een project. Waarom zijn budgetten en levertijden overschreden of waarom voldoet de software niet volledig aan de gestelde eisen? Voor de beantwoording van dergelijke vragen, moet het mogelijk zijn een diagnose van het uitgevoerde project te maken. Is er onvoldoende gekwalificeerd personeel ingezet, zijn bepaalde zaken (opleiding, conversie, projectmanagement) bij het opstellen van de begroting over het hoofd gezien, is de invloed van bepaalde cost drivers verkeerd ingeschat enz.

Vaak zijn organisaties niet in staat op dergelijk soort vragen een antwoord te geven. Deze handicap kan voor een deel worden ondervangen door tijdens de uitvoering van een project gegevens en ervaringen vast te leggen. In hoofdstuk 8 wordt dit punt uitgewerkt.

6.3.4. INFORMATIE

Zonder twijfel kan gesteld worden dat onzekerheid de belangrijkste reden vormt voor het overschrijden van begrotingen. Onzekerheid wordt door Galbraith (1973) gedefinieerd als het verschil tussen de hoeveelheid informatie om het werk uit te voeren en de hoeveelheid

informatie die men feitelijk bezit. Veel automatiseringsprojecten kenmerken zich door een grote mate van onzekerheid. Onzekerheid wat betreft de kwaliteit van de specificaties, onzekerheid over de juiste wijze van projectorganisatie, teamsamenstelling, management, te hanteren methoden en technieken, onzekerheid over de karakteriseren van het project, het gebruik van de meest geschikte begrotingsmethode c.q. model enz. Galbraith constateert dat organisaties te maken krijgen met steeds omvangrijkere en complexere besturingssituaties. Als gevolg van ondermeer een zich snel ontwikkelende technologie en een stijgende vraag naar complexere en moeilijk te formaliseren geautomatiseerde systemen (denk hierbij aan beslissingsondersteunende systemen, expert systemen enz.) nemen de hierboven genoemde onzekerheden toe. Het wordt voor een organisatie die belast is met het ontwikkelen van software daarom steeds moeilijker een antwoord te vinden op deze stijgende onzekerheden bij het beheersen en begroten. Galbraith onderscheidt twee strategieën om hieraan het hoofd te kunnen bieden.

1. *De eerste mogelijkheid is de behoefte aan coördinatie te reduceren door de bronnen van onzekerheid weg te nemen.*
   De meest eenvoudige en meest toegepaste manier hiervan is het introduceren van speling in de begroting, bijvoorbeeld een ruimere doorlooptijd nemen dan men strikt qua ondergrens nodig heeft, meer capaciteit claimen dan strikt noodzakelijk of op een hogere kwaliteit mikken dan vereist is. Naarmate de onzekerheid toeneemt, zal men grotere marges in moeten bouwen. Gebeurt dit niet dan is de kans dat het project uit de hand loopt aanzienlijk. Een andere maatregel die Galbraith noemt, is een zodanige invloed op de omgeving uitoefenen dat er minder verstoringen optreden. Bij het ontwikkelen van software wordt dit bijvoorbeeld gedaan met benodigde middelen. Als een extern deskundige nodig is voor bijvoorbeeld het ontwerp en bouw van een complex netwerk, en de beschikbaarheid van die deskundige is een constant probleem, dan kan de organisatie besluiten deze deskundige in dienst te nemen zodat hij of zij altijd beschikbaar is. Een ander voorbeeld is dat specificaties op een bepaald moment "bevroren" worden. De behoefte aan coördinatie kan ook verminderd worden door een project zodanig in te delen in deelprojecten dat deze min of meer autonoom van elkaar uitgevoerd kunnen worden.

2. *Verhoog de capaciteit om informatie te verwerken en uit te wisselen.*
   Deze strategie laat zich vertalen in onder meer de volgende maatregelen:
   - zorg voor een goede en snelle gegevensuitwisseling. Dit kan

bijvoorbeeld gebeuren door planning- en realisatiegegevens niet periodiek maar real time bij te werken c.q. beschikbaar te stellen. Een ander voorbeeld is het beschikbaar stellen van vastgelegde kennis en ervaring met voltooide projecten.
- creëer laterale relaties. Galbraith bedoelt hiermee het nemen van organisatorische maatregelen, bijvoorbeeld het overgaan op een matrixorganisatie of een projectorganisatie. Op deze wijze ontstaan overlegstructuren die dwars door de bestaande hiërarchische structuur lopen.

Beheersing van software-ontwikkeling impliceert meetbaarheid. Hierbij moet een onderscheid worden gemaakt in meetbaarheid van het project en meetbaarheid van het produkt.
Het is niet eenvoudig de voortgang van een project en de kwaliteit van het produkt tijdens de ontwikkeling te meten. Het is moeilijk zaken als betrouwbaarheid, gebruiksvriendelijkheid, overdraagbaarheid enz. te meten. Als er hogere eisen aan de softwarekwaliteit worden gesteld, zal een separate kwaliteitsbewakingsfunktie gecreëerd moeten worden. Deze funktie zorgt voor de uitvoering van risico-analyses, periodieke projectreviews, structured walkthroughs, speciale kwaliteitsreviews in de vorm van formal inspections, enz. Problemen met het meten van de projectvoortgang komen ondermeer tot uiting in het "95% gereed" syndroom.
Men kan twee dingen meten in een project: *produkten die gereed zijn*, zoals ontwerpen, opleidingen, codering van een bepaalde module, enz. en *gebruikte middelen*, zoals geld, inspanning, computertijd, enz. Het gebruik van middelen zegt echter weinig over de voortgang van het project. Een manier om greep te krijgen op de voortgang van een project is het opdelen van een project in fasen, activiteiten en taken en het inbouwen van meetpunten in een project. Naarmate het project zich kenmerkt door een grotere onzekerheid, zal men meerdere meetpunten moeten inbouwen. Informatie over de voortgang van het project impliceert ondermeer gereedmelding van produkten en een periodieke registratie van projectgegevens.

6.4. ASPECTEN VAN BESTUREN

Wil men in staat zijn het proces van software-ontwikkeling effectief te besturen, dan zal in ieder geval voldaan moeten worden aan de voorwaarden, die in de vorige paragraaf zijn genoemd. Het voldoen aan deze voorwaarden betekent echter niet dat besturing zondermeer

succesvol zal zijn. Hiervoor is ook nodig dat men weet hoe men moet besturen.

- Zo zal men bij besturing er rekening mee dienen te houden dat de besturingswijze afgestemd is op de aard van het **besturingsprobleem.**
- Bovendien is bij besturing **leidinggeven en coördinatie** noodzakelijk. Afhankelijk van de te besturen situatie zal de stijl van leidinggeven en de vorm van coördinatie verschillen.
- Voor het ontwikkelen van software kan men kiezen uit een aantal **ontwikkelstrategieën.** De wijze van besturen zal afhankelijk zijn van de gekozen ontwikkelstrategie.
- Een **begroting** kan een hulpmiddel zijn bij besturen. Afhankelijk van de geldende situatie zal een begroting op een andere manier als hulpmiddel bij besturen moeten worden toegepast.

In deze paragraaf zal nader worden ingegaan op de hierboven genoemde aspecten, waarmee men rekening dient te houden bij het besturen c.q. beheersen van software-ontwikkeling.

6.4.1. DE AARD VAN HET BESTURINGSPROBLEEM

Vanuit besturingsoptiek wil ik een onderscheid maken in een viertal problemen bij de beheersing van software-ontwikkeling. Deze problemen zijn:

- ***realisatieprobleem.***
  Het accent bij de ontwikkeling ligt op de realisatie, met andere woorden: hoe kan men, gegeven het pakket van eisen, het eindresultaat bereiken. De aandacht is niet primair gericht op het analyseren van de probleemsituatie, het inventariseren van de informatiebehoeften of het opstellen van de lijst van eisen, maar op de beheersing van de uitvoering van het project en op de bewaking van de produktkwaliteit.
- ***allocatieprobleem/capaciteitsprobleem.***
  Software-ontwikkeling is personeelsdominant en vraagt om specialistische vaardigheden. Een belangrijk probleem bij de ontwikkeling van software heeft betrekking op de beschikbaarheid van personeel. Gebrekkige beschikbaarheid kan kwantitatief en/of kwalitatief van aard zijn. Kwantitatief wil zeggen dat men niet voldoende personeel beschikbaar heeft om een specifiek project te bemannen en de bemanning niet eenvoudig op volle sterkte is te krijgen. Kwalitatief wil zeggen dat men weliswaar voldoende per-

soneel in huis heeft maar dat de ontwikkeling van de programmatuur in kwestie om specialistische vaardigheden vraagt, die niet aanwezig is bij het beschikbare personeel en moeilijk op relatief korte termijn te verwerven is.

- ***ontwerpprobleem.***
  Er is sprake van een ontwerpprobleem als niet duidelijk is op welke wijze men het ontwikkelproces moet invullen. Met ontwerpprobleem wordt bedoeld dat antwoord gegeven moet worden op vragen als: wordt het project wel of niet projectmatig aangepakt, welke beslispunten worden er in het project ingebouwd, welke documenten moeten wanneer opgeleverd worden, hoe moeten mensen ingezet worden, hoe worden verantwoordelijkheden en bevoegdheden verdeeld enz.
- ***exploratieprobleem.***
  Een probleem is exploratief van aard als men bij de ontwikkeling van software geen duidelijkheid heeft over het eindresultaat, men problemen heeft met de weg waarlangs men het eindresultaat wil bereiken en met de beschikbaarheid van de daarvoor benodigde middelen. Het accent ligt op het verkennen en formuleren van de probleemsituatie.

6.4.2. HET COÖRDINATIEMECHANISME EN DE MANAGEMENTSTIJL

De wijze van besturing van software-ontwikkeling is te verduidelijken met behulp van de vijf coördinatie-mechanismen uit het model van Mintzberg (1983) en met behulp van de vier managementstijlen uit het model van Reddin (1973).
Mintzberg gaat ervan uit dat te verrichten werk verdeeld moet worden in verschillende taken en dat deze taken gecoördineerd moeten worden. De vijf coördinatiemechanismen die hij onderscheidt zijn:

1. ***Mutual adjustment.*** Coördinatie wordt gerealiseerd door onderlinge afstemming. Deze vorm van coördinatie wordt toegepast in de meest eenvoudige én in de meest complexe werksituaties. Bij zeer omvangrijke en innovatieve projecten wordt het werk verdeeld over veel specialisten. Men is echter niet in staat precies aan te geven wat elke specialist moet maken en hoe hij dat moet doen. Het succes van het project is afhankelijk van de bekwaamheden van degenen die het werk uitvoeren om gezamenlijk op een niet gespecificeerde wijze een niet gespecificeerd resultaat te bereiken.

2. *Direct supervision.* Coördinatie berust bij aparte personen, die verantwoordelijk zijn voor het werk van anderen.

Uit te voeren werk kan ook gecoördineerd worden door middel van standaardisatie:

3. *Standardization of work processes.* De inhoud van het uit te voeren werk is volledig gespecificeerd. Het is mogelijk taken uitte voeren en te controleren aan de hand van voorgeschreven instructies.
4. *Standardization of work outputs.* Dit betekent dat het eindresultaat bekend moet zijn. De wijze waarop dit bereikt wordt, speelt hierbij in principe geen rol.

5. *Standardization of worker skills.* Als het niet mogelijk het uit te voeren werk en het resultaat te specificeren, moet coördinatie gerealiseerd worden door bijvoorbeeld opleiding en vaardigheden van degenen die het werk moeten uitvoeren, te specificeren.

Naarmate het uit te voeren werk complexer wordt en er minder zekerheid bestaat over het uit te voeren werk en het te bereiken resultaat verschuift de coördinatie van direct supervision naar standaardisatie, bij voorkeur van het proces, anders van het resultaat of anders van de middelen om tenslotte uit te komen bij mutual adjustment.

De theorie van Reddin (1973, Botter 1980) over stijlen van management sluit aan bij het model van Mintzberg. Reddin onderscheidt vier basisstijlen van management. Hij komt tot deze vier vormen door er van uit te gaan dat bij leiding geven twee dimensies te onderkennen zijn. Deze twee dimensies zijn:

- *relatiegerichtheid.* Dit heeft betrekking op de aandacht voor de mens en zijn relaties met andere mensen in de organisatie,
- *taakgerichtheid.* Hiermee wordt bedoeld de aandacht voor de resultaten die bereikt moeten worden en de wijze waarop dit resultaat bereikt moet worden.

Reddin gaat ervan uit dat zowel de relatiegerichtheid als de taakgerichtheid hoog en laag kan zijn. Door deze waarde te combineren ontstaan vier mogelijke combinaties: vier basisstijlen van leiding geven. In figuur 6.3 zijn deze vier stijlen weergegeven. Welke stijl het meest geschikt is, is afhankelijk van de situatie die zich voordoet.

		TAAKGERICHTHEID	
		laag	hoog
RELATIE-GERICHTHEID	laag	afscheidingsstijl (bureaucraat)	toewijdingsstijl (welwillend autocraat)
	hoog	relatiestijl (ontwikkelaar)	integratiestijl (bestuurder)

Figuur 6.3 : De vier basisvormen van leidinggeven volgens Reddin (1973)

- De ***afscheidingsbasisstijl*** blijkt doorgaans het meest effectief voor het leidinggeven aan werk met een sterk routinematig karakter. Efficiency staat centraal. Er is een voorkeur voor regels en procedures. Deze stijl sluit nauw aan bij Mintzberg's direct supervision in combinatie met standardization van work processes.

- De ***relatiebasisstijl*** is het meest effectief in situaties waarin mensen moeten worden gecoördineerd, gestimuleerd en opgeleid. Het uit te voeren werk is sterk persoonsgebonden. Er is niet sprake van routine-arbeid, maar van werk dat innovatief, specialistisch van aard is. Het coördinatiemechanisme dat hier het best bij aansluit is mutual adjustment.

- De ***toewijdingsbasisstijl*** is het meest effectief in situaties waar onder druk gewerkt moet worden. Van de manager wordt verwacht dat hij weet hoe hij onder deze omstandigheden het doel kan bereiken zonder wrevel te wekken. Deze stijl van leidinggeven sluit goed aan bij standardization of worker skills.

- De ***integratiestijl*** sluit het best aan bij situaties waarin sprake is van grote onzekerheid wat betreft het te bereiken resultaat. Het werk is exploratief van aard. Het is de taak van de manager zijn mensen te enthousiasmeren en hun commitment te krijgen. Coördinatie kan in een dergelijke situatie goed gerealiseerd worden door middel van mutual adjustment.

6.4.3. DE ONTWIKKELSTRATEGIE

De ontwikkeling van software vertoont duidelijke parallellen met een project-shop. In hoofdstuk 7 zal uitvoerig worden ingegaan op deze parallellen. In deze paragraaf wil ik mij concentreren op één kenmerk van een project-shop, namelijk de stapsgewijs realisatie van het eindresultaat. Wat betreft de ontwikkeling van software is het mogelijk hierbij een onderscheid te maken in de volgende ontwikkelstrategieën:

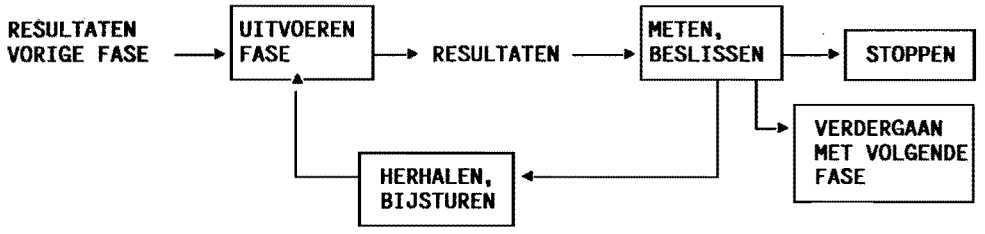
a. de grote-stapmethode ofwel de lineair strategie,
b. de kleine-stapmethode ofwel de incrementele strategie,
c. de prototypingstrategie (in feite een verbijzondering van de kleine-stapmethode).

Een centraal thema in deze paragraaf is de afstemming van de wijze van besturing op de gekozen ontwikkelstrategie.

Bij de vaststelling van de informatiebehoeften spelen twee zaken een belangrijke rol, te weten het bestaan van een verzameling duidelijke en volledige informatiebehoeften en de stabiliteit van deze behoeften. In paragraaf 7.3.1 wordt uitgebreid aandacht besteed aan het proces van informatiebehoeftebepaling. Bij de keuze van een ontwikkelstrategie ga ik ervan uit dat deze keuze in belangrijke mate wordt bepaald door de duidelijkheid en stabiliteit van de informatiebehoeften. In het nu volgende zal worden aangegeven dat een ontwikkelstrategie directe gevolgen heeft voor de wijze waarop software-ontwikkeling beheerst dient te worden.

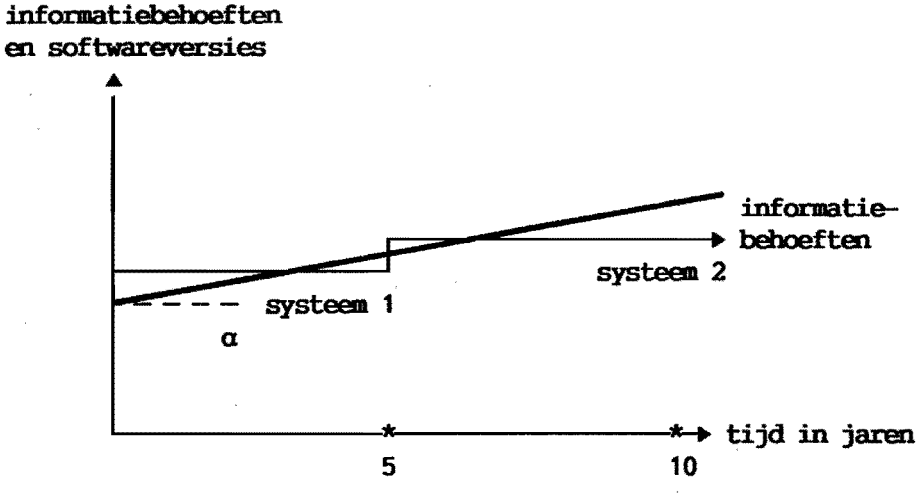
ad. a *lineaire ontwikkelstrategie.*

Kenmerkend voor de lineaire ontwikkel-strategie is de opdeling van het ontwikkeltraject in strikt achtereenvolgende fasen en de introductie van duidelijk identificeerbare meetpunten/mijlpalen. Van Vliet (1987a) stelt dat de belangrijkste drijfveer voor de lineaire aanpak de beheersbaarheid van een automatiseringsproject is. De voortgang van het ontwikkelproces wordt afgemeten aan het bereiken van die mijlpalen. In figuur 6.4 is de essentie van zo'n meetpunt weergegeven. De strategie is sterk document-driven. In paragraaf 7.3.3 wordt de fasenindeling nader toegelicht.



Figuur 6.4 : Het principe van de werking van een meetpunt.

De lineaire ontwikkelstrategie is het meest geschikt voor die situaties waarin sprake is van stabiele en duidelijke informatiebehoeften. Hiervoor geldt de volgende verklaring (zie ook figuur 6.5). Als de informatiebehoeften niet of niet noemenswaardig veranderen, kan de ontwikkelde software zonder noodzakelijke aanpassingen gedurende langere tijd geschikt zijn. In figuur 6.5 wordt de stabiliteit van de informatiebehoeften weergegeven door de hellingshoek α van de

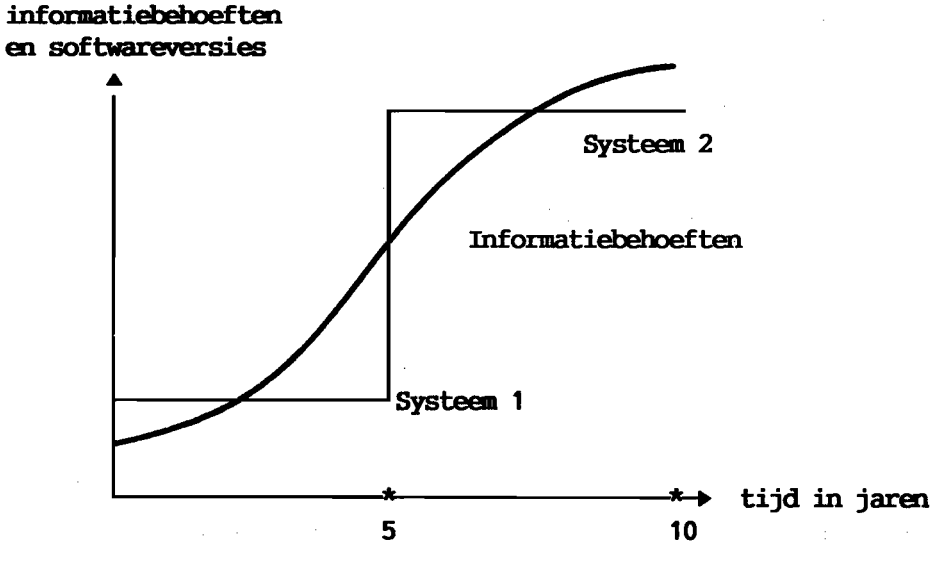


Figuur 6.5 : De grote-stapmethode bij min of meer stabiele informatiebehoeften.

betreffende lijn. Stabiele informatiebehoeften wil zeggen dat α gelijk is aan 0 of in ieder geval erg klein is.

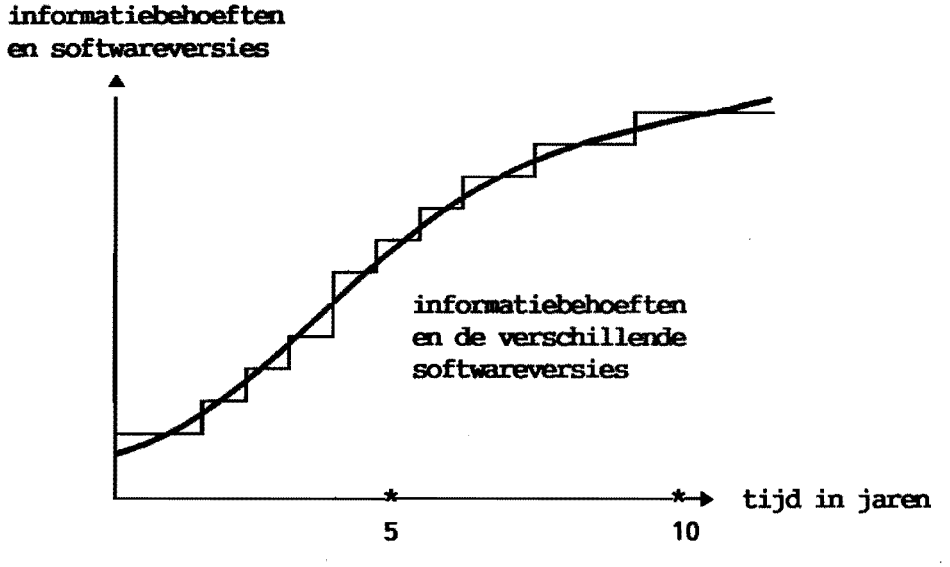
ad. b *de incrementele ontwikkelstrategie.*

Wordt software lineair ontwikkeld in het geval de stabiliteit van de informatiebehoeften gering is, dan zal het eindresultaat over het algemeen niet beantwoorden aan deze gewijzigde informatiebehoeften (Davis, Bersoff en Comer, 1988). Bij de ontwikkeling volgens de grote-stapmethode gaat men er immers van uit dat software zonder wijzigingen een langere periode meekan.
Wil men bij instabiele informatiebehoeften toch gebruik maken van de grote-stapmethode dat zal men bij de ontwikkeling moeten proberen de wijzigende informatiebehoeften te voorzien en hierop in te spelen. Een en ander betekent dat de software die men aldus ontwikkelt in het begin over een aanzienlijke "overcapaciteit" zal beschikken en aan het einde over een aanzienlijke "ondercapaciteit". In figuur 6.6 is dit in beeld gebracht.



Figuur 6.6 : De grote-stapmethode bij niet stabiele informatiebehoeften.

Anders ligt het als men te werk gaat volgens een incrementele ontwikkelstrategie. Hierbij stelt men zich op het standpunt dat software zich iedere keer weer met kleine stapjes moet aanpassen aan de veranderende informatiebehoeften. Een en ander betekent dat hoge eisen gesteld worden aan kwaliteitsaspecten als flexibiliteit, onderhoudbaarheid, aanpasbaarheid en uitbreidbaarheid van de software. Omdat de onzekerheid over toekomstige informatiebehoeften groot is, zal de tijdspanne tussen het moment van specificeren en realiseren zo kort mogelijk moeten zijn. De ontwikkeling verloopt in kleine stappen (incrementeel ontwikkelen). Bij iedere stap wordt de functionaliteit uitgebreid en aangepast aan de nieuwe eisen (McCracken en Jackson 1982). De essentie van de kleine-stapmethode is in figuur 6.7 weergegeven.



Figuur 6.7 : De kleine-stapmethode bij niet stabiele informatiebehoeften.

Op deze wijze worden de wijzigende informatiebehoeften op de voet gevolgd. Een tweede effect van deze aanpak is dat de duurzaamheid van de software groter is dan wanneer in deze situatie een lineaire ontwikkelstrategie toegepast zou zijn. De te volgen ontwikkelstrategie heeft duidelijke konsekwenties voor beheersen en begroten.

- Ligt bij de lineaire strategie het accent primair op het beheersen van het proces (ondermeer door te faseren), bij incrementeel ontwikkelen ligt het accent op het definiëren van increments. In iedere kleine stap is de fasering van analyse, ontwerp, realisatie, testen en implementatie terug te vinden. Omdat de increments qua omvang klein zijn en de ontwikkeltijd ervan kort is, zijn de ontwikkelstappen per increment beter te overzien en te beheersen. Gaande van lineair naar incrementeel zal men vaker moeten meten (meer meetpunten) of het produkt nog op de juiste wijze wordt gemaakt (verificatie) en het juiste produkt wordt gemaakt (validatie).
- Een tweede gevolg voor de beheersbaarheid heeft betrekking op het beslissen op de verschillende meetpunten. Naarmate de ontwikkeltijd van een softwareversie toeneemt, met andere woorden de stappen steeds groter worden, wordt de beslissing op een meetpunt om een project alsnog te stoppen tijdens de voortgang van de ontwikkeling steeds moeilijker. Op dit facet wil ik kort ingaan.

  De reden waarom het steeds moeilijker wordt een project alsnog te stoppen, zijn zowel psychologisch als economisch van aard. Vanuit psychologische optiek is het bepaald niet eenvoudig een project, waarin reeds veel is geïnvesteerd, alsnog af te "blazen". Men staat als het ware onder druk van de reeds geïnvesteerde bedragen en kan het vanuit die overweging moeilijk aan anderen "verkopen" dat het project toch een mislukking is en dus alsnog gestopt moet worden. Dit heeft niets te maken met economisch rationele besluitvorming, maar meer met "politieke" overwegingen.

  Wordt vanuit psychologische overwegingen de no/go beslissing steeds moeilijker, vanuit economische optiek wordt de go beslissing voor een project steeds eenvoudiger. Economisch rationeel handelen betekent dat men de nog te investeren bedragen afweegt tegen de verwachte baten. Bij die afweging zijn alle in het verleden gedane investeringen irrelevant, immers "gedane zaken nemen geen keer". Wanneer nu de verwachte baten niet al te dramatisch veranderen in negatieve zin, wordt een project steeds lucratiever naarmate de noodzakelijke investeringen reeds grotendeels achter de rug zijn. Het nog te investeren bedrag is dan immers relatief gering ten opzichte van de verwachte baten. Dit zal de go beslissing voor een project derhalve aanzienlijk vereenvoudigen.

Bovengenoemde problemen spelen bij de incrementele ontwikkel-strategie minder dan bij de lineaire. Na elke kleine stap wordt immers een operationeel produkt opgeleverd. De beslissingen in de meetpunten bij de ontwikkeling van zo'n produkt hebben een beperkte tijdshorizon en hebben betrekking op een relatief klein produkt.

ad. c *prototypingstrategie.*

Als de informatiebehoeften moeilijk te achterhalen zijn, verdient het aanbeveling door middel van prototyping te achterhalen aan welke eisen de software moet voldoen (Davis 1982, Vonk 1985 en 1987). Voor de begroting van software-ontwikkeling moet men in een dergelijk geval onderscheid maken in:

- *begroten van de ontwikkeling van een of meerdere prototypen.* Gezien de onzekerheid en wazigheid zal het moeilijk zijn betrouwbare kostenschattingen van de prototypen te maken. Aanbevolen wordt om een bepaalde hoeveelheid tijd, geld en middelen beschikbaar te stellen voor het ontwikkelen van prototypes. Vonk (1985) beweert in dit verband dat **afspraken** gemaakt moeten worden omtrent de duur van de verschillende prototypingfasen. Goede tools om snel een prototype te kunnen bouwen, moeten beschikbaar zijn.
- *begroting de software-ontwikkeling na afronding van de prototypingfasen.* Nadat de prototypingfasen zijn afgerond, behoren de informatiebehoeften (wat betreft functionaliteit) bekend te zijn. Afhankelijk van de stabiliteit van de informatiebehoeften kan men kiezen voor een lineaire- of incrementele ontwikkelstrategie.

6.4.4. DE WIJZE VAN BEGROTEN

Het vierde aspect van besturing waarop ik wil wijzen heeft betrekking op de wijze van begroten. Hierbij maak ik een onderscheid in:

a. begrotingsmethoden,
b. begrotingsfunkties,
c. begrotingseenheden,

ad. a *begrotingsmethoden.*

Hierbij maak ik gebruik van in paragraaf 3.2 besproken begrotingsmethoden, te weten:

- begrotingsmodellen,
- expertmethode,
- analogiemethode.

Een belangrijk hulpmiddel bij het begrotingsproces is een databank van voltooide projectgegevens. In situaties waarin duidelijkheid bestaat over het eindresultaat kan een dergelijke databank een belangrijk hulpmiddel zijn. In hoofdstuk 8 wordt dit onderwerp nader uitgewerkt.

ad. b *begrotingsfunkties.*

Een begroting kan meerdere funkties hebben (Hofstede 1967). Bij begroten van software-ontwikkeling kan ik volstaan met de volgende twee funkties:

1. De begroting als stuurinstrument voor het management,
2. De begroting als voorwaardenscheppend instrument.

Theeuwes (1987) geeft aan dat een begroting, wil deze als ***stuurinstrument*** voor het management voldoen, de volgende drie kenmerken moet hebben:

a. beheersbaar,
b. taakstellend,
c. betrouwbaar.

Beheersbaar impliceert dat een begroting gebaseerd moet zijn op een duidelijke en volledige lijst van programma-eisen. Er moeten duidelijke afspraken worden gemaakt wat er aan prestaties wordt verwacht en met welke inzet van mensen, middelen, hoeveelheid tijd en geld deze prestaties gerealiseerd dienen te worden. Door tijdige signalering van verschillen tussen begroting en werkelijkheid is het mogelijk sturing uit te oefenen wat betreft de inzet van produktiemiddelen en/of de te realiseren prestaties.

Taakstellend betekent in dit verband dat de begroting de betekenis heeft van een norm, van streefcijfers waaraan de uitgevoerde activiteiten en het gebruik van tijd, geld, mensen en middelen afgemeten kan worden. Bij een taakstellende begroting is het van belang de betrokken medewerkers te raadplegen voor dat deel van het project waar zij verantwoordelijk voor zijn. Men krijgt op deze wijze de medewerking die zo noodzakelijk is voor het welslagen van een project.

Het is van belang dat de normen/streefcijfers en de gegevens over de uitgevoerde activiteiten en bestede produktiemiddelen realistisch en

betrouwbaar zijn. Deze gegevens moeten bij voorkeur afkomstig zijn van de medewerkers in het softwareproject. Commitment zal in een positieve uitwerking hebben omdat de betrokken medewerkers dan eerder geneigd zijn realistische gegevens aan te leveren en deze te gebruiken bij de beheersing van het project. Naarmate de realiteitswaarde van de begroting toeneemt, zal het vertrouwen erin stijgen en zal de begroting meer worden gebruikt als echt stuurinstrument.

Een begroting heeft een ***voorwaardenscheppende*** betekenis, als het gaat om de vaststelling van de haalbaarheid van een project. Bij aanvang van een project is het van belang te onderzoeken in hoeverre de ontwikkeling van de betreffende software voor de organisatie interessant is. Een dergelijke begroting heeft slechts een indicatief karakter om onder andere het financiële risico in te schatten en dient als basis voor een go-no/go beslissing. Bij dit soort begrotingen gaat het vaak ook om het voorlopig veilig stellen van kosten, tijd, mensen en middelen. Harde afspraken over de inzet van deze produktiemiddelen en de uit te voeren activiteiten zijn in dit stadium van software-ontwikkeling nog niet te maken. Een dergelijke begroting kan derhalve niet dienen als instrument om het project te beheersen. Aan de voorwaarden voor beheersing is immers niet voldaan.

ad. c *begrotingseenheden.*

Begrotingseenheid heeft betrekking op de grootte van de stap bij de eerder genoemde kleine- en grote-stapmethode. Zo kan men bij het begroten kan een onderscheid maken tussen:

- *begroten van elke (kleine) stap.* Omdat elke stap relatief klein is, zal de onzekerheid bij het opstellen van een begroting ook relatief gering zijn.
- *begroten van een grote stap c.q. van het eindresultaat.* Omdat hier sprake is van een grote stap, zal het risico groter zijn en als gevolg daarvan de nauwkeurigheid van de begroting hiervan geringer zijn. Dit risico wordt nog groter naarmate de stabiliteit van de informatiebehoeften geringer wordt. Door in kleine stappen te ontwikkelen is een betere beheersing van tijd en kosten mogelijk. Na realisatie van elke stap kan op basis van de begroting van de volgende stap en een kostenindicatie van het eindresultaat een go-no/go beslissing genomen worden. Na elke stap beschikt men over een operationeel produkt. In tegenstelling tot de lineaire aanpak is de handicap van "de hypotheek uit het verleden" bij het nemen van een go-no/go beslissing minder.

6.5. CONCLUSIES

In de eerste vijf hoofdstukken heeft het accent gelegen op het begroten van software-ontwikkeling. Begroten is slechts een component bij het beheersen van het proces van software-ontwikkeling. Behalve begroten is het bij beheersen van belang te beschikken over duidelijke, volledige en meetbare doelstellingen. Deze doelstellingen vormen de basis voor een plan van uitvoering en een begroting. De voortgang van het ontwikkelproces c.q. de uitvoering en het resultaat van de uitvoering dient voortdurend gemeten te worden. Bij eventuele afwijkingen tussen plan/begroting en realisatie dient bijsturing plaats te vinden. Plan of begroting kan worden bijgesteld. In de werkelijkheid van de uitvoering kan worden bijgestuurd of de doelstellingen kunnen worden aangepast. De componenten van beheersen "formuleren doelstellingen, plannen en begroten, meten, besturen" zijn onlosmakelijke met elkaar verbonden. Bij het zoeken naar oplossingen voor de problemen van budget- en doorlooptijdoverschrijdingen is het niet voldoende uitsluitend de aandacht te richten op het begroten van software-ontwikkeling. In dit hoofdstuk en in de volgende hoofdstukken 7 en 8 is daarom het onderwerp van onderzoek verbreed van begroten naar beheersen van software-ontwikkeling.
Om het proces van software-ontwikkeling te analyseren en de rol van beheersen hierin aan te geven, is gebruik gemaakt van het besturingsparadigma. Verder is een aantal voorwaarden genoemd waaraan voldaan moet worden, wil er sprake zijn van effectieve besturing. Als bij het ontwikkelen van software voldaan wordt aan de genoemde voorwaarden, is dat nog geen garantie voor een succesvolle ontwikkeling. Men dient ook te weten op welke wijze men die voorwaarden zo goed mogelijk aan kan wenden c.q. op welke wijze men moet sturen. In deze paragraaf is daarom een aantal belangrijke aspecten behandeld, waarmee men rekening dient te houden bij het besturen van software-ontwikkeling. Hierbij komt naar voren dat de wijze van besturing afhankelijk is van kenmerken van software-ontwikkeling. Dit gegeven wordt nader uitgewerkt in hoofdstuk 7.

7. EEN TYPOLOGIE VAN BEHEERSSITUATIES

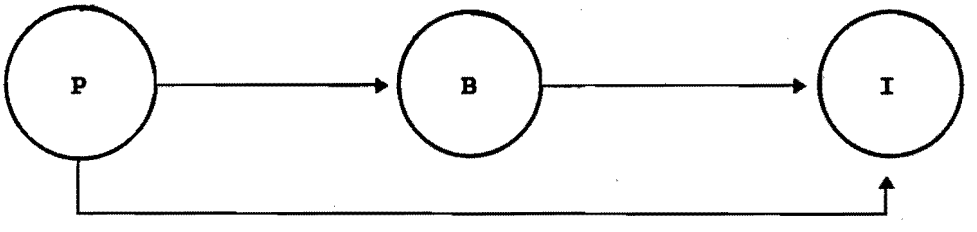
7.1. INLEIDING

In hoofdstuk 6 is het beheersen en begroten van software-ontwikkeling geanalyseerd met behulp van het besturingsparadigma. Dit paradigma is toegepast om aan te geven in welke mate bij het ontwikkelen van software wordt voldaan aan de voorwaarden voor effectieve besturing. Verder is in hoofdstuk 6 gewezen op de verschillende manieren waarop beheersen en begroten ingevuld kunnen worden. De concepten die in hoofdstuk 6 zijn aangedragen, zullen in hoofdstuk 7 worden gebruikt voor het ontwerpen van een typologie van beheerssituaties. Om tot verschillende beheerssituaties te komen, is het P-B-I model toegepast. Dit model wordt gebruikt om de samenhang tussen het proces van software-ontwikkeling (*P*), de wijze van beheersing (*B*) en de hiervoor noodzakelijke informatie (*I*) aan te geven. Het P-B-I model wordt in paragraaf 7.2 beschreven. Vervolgens worden in paragraaf 7.3 belangrijke kenmerken van *P* genoemd. Deze kenmerken worden geordend in produkt/markt kenmerken, produktiemiddelen kenmerken en proceskenmerken. In paragraaf 7.4 worden belangrijke kenmerken van *B* beschreven. Deze beschrijving wordt uitgevoerd vanuit vijf verschillende invalshoeken, te weten: de aard van het beheersprobleem, de primaire doelstelling bij de beheersing, het te hanteren coördinatiemechanisme, de ontwikkelstrategie en tenslotte de wijze van begroten. In paragraaf 7.5 wordt de samenhang tussen de P en de B weergegeven. Een en ander zal resulteren in vier ideaaltypische beheerssituaties. Tot welke situatie een specifiek proces van software-ontwikkeling gerekend moet worden, is afhankelijk van de mate waarin men kennis heeft van het te ontwikkelen produkt (produkt/markt), van de benodigde middelen (produktiemiddelen) en van ontwikkelproces (proces). In paragraaf 7.6 wordt, onder de titel "evolutionair begroten", aangegeven dat, naarmate het ontwikkelen vordert, risico's doorgaans kleiner worden en er meer "harde" gegevens beschikbaar komen voor een bijgestelde begroting. Als gevolg

daarvan zal een softwareproject van de ene beheerssituatie in de andere geraken. Het hoofdstuk wordt afgesloten met conclusies (paragraaf 7.7). In dit hoofdstuk worden, om in termen van het P-B-I model te spreken de P en de B beschreven. In hoofdstuk 8 wordt nader ingegaan op de I in het P-B-I model.

7.2. HET P-B-I MODEL

In paragraaf 6.2 is het besturingsparadigma beschreven. In dit paradigma staat het Besturend Orgaan centraal. Genoemd paradigma kan men ook anders interpreteren ten behoeve van het ontwikkelen van een bestuurlijk informatiesysteem. Het P-B-I model is gebaseerd op deze interpretatie. De basisgedachte daarbij is de volgende: besturing kan men niet willekeurig invullen maar is afhankelijk van het soort proces dat bestuurd moet worden. Anders geformuleerd: de besturingswijze, afgekort B, is afhankelijk van situatiegebonden factoren van het te beheersen proces, afgekort P. De benodigde informatie, afgekort I, wordt op zijn beurt weer bepaald door de wijze van besturing (B) en door het te beheersen proces (P). In figuur 7.1 wordt dit schematisch weergegeven.



Figuur 7.1 : Het P-B-I model.

De pijlen in de figuur geven aan wat door wat wordt bepaald. Vraag is nu welke karakteristieken van P min of meer determineren welke vorm van B gewenst is en vervolgens welke karakteristieken van B op hun beurt determineren welke I-concepten gekozen moeten worden. Het P-B-I model is van origine ontwikkeld als hulpmiddel

voor het ontwerp van een concept voor produktie-beheersing in industriële bedrijven. Bemelmans (1986) geeft aan dat het P-B-I model goed generaliseerbaar is naar andere situaties; bijvoorbeeld situaties waar geen fysieke produkten maar (immateriële) diensten worden geproduceerd. Daarbij zijn wel vertaalslagen noodzakelijk. In dit onderzoek zal het P-B-I model gebruikt worden als analyse-instrument voor het proces van software-ontwikkeling en als hulpmiddel om een beheersingsconcept voor software-ontwikkeling te ontwerpen. De genoemde vertaalslagen hebben betrekking op de aanpassing van het model aan begrippen die van toepassing zijn op de software-ontwikkeling. Bovendien blijkt het beschrijvingsmodel dat Bemelmans gebruikt voor de karakterisering van het primaire proces van een industrieel bedrijf niet zondermeer toepasbaar voor software-ontwikkeling. Het beschrijvingsmodel is aangepast, waarover meer in paragraaf 7.3. Ook bij de typering van B waren een aantal vertaalslagen nodig. Op dit punt wordt nader ingegaan in paragraaf 7.4. Na deze aanpassingen bleek het P-B-I model goed toepasbaar voor het karakteriseren van software en software-ontwikkeling.

Om de vraag te kunnen beantwoorden "welke kenmerken van P bepalen welke B gewenst is en vervolgens welke kenmerken van B bepalen de keuze voor welk I-concept" , zal ik in paragraaf 7.3 kenmerken van software en software-ontwikkeling (P) bespreken, in paragraaf 7.4 beheerskenmerken (B) van software-ontwikkeling beschrijven en vervolgens in paragraaf 7.5 aangeven op welke wijze de kenmerken van P bepalend zijn voor de wijze waarop software-ontwikkeling beheerst en begroot moet worden.

7.3. P: KENMERKEN VAN SOFTWARE-ONTWIKKELING

In deze paragraaf wordt een beschrijving gegeven van datgene wat beheerst moet worden en het proces dat moeten worden uitgevoerd. Een en ander zal gebeuren door het aangeven van belangrijke kenmerken van software-ontwikkeling. Bij de beschrijving van deze kenmerken heb ik mij laten inspireren door een beschrijvingsmodel c.q. een checklist van vragen die door Bemelmans (1986) is opgesteld ten behoeve van de karakterisering van het primaire proces van een industrieel bedrijf. Een deel van deze vragen richt zich op ***produkt /marktkenmerken*** (de software en de klant), een andere deel op ***produktie-middelkenmerken*** (primair het ontwikkel-personeel en de ontwikkelomgeving) en een derde deel heeft betrekking op het ***pro-***

duktieproces (de software-ontwikkeling, werkvoorbereiding, work-breakdown-structure, enz.). Het merendeel van de vragen uit de oorspronkelijke checklist kunnen gebruikt worden. Een aantal vragen, zoals vragen naar de voorspelbaarheid en het dynamisch karakter van de vraag naar eindprodukten, is bij de karakterisering van software buiten beschouwing gelaten. Bij software die bedoeld is voor hergebruik zijn dergelijke vragen wel relevant. Bij de karakterisering van software is evenwel niet nader ingegaan op software-ontwikkeling ten behoeve van hergebruik. Andere vragen zijn op sommige punten bijgesteld. Dit geldt ondermeer voor de vraag naar de specificiteit van het produkt. Deze is vertaald naar de mate van herbruikbaarheid van de software.
Er is een aantal belangrijke kenmerken van software te noemen die directe konsekwenties hebben voor de wijze waarop software-ontwikkeling beheerst en begroot kan worden.

7.3.1. PRODUKT/MARKTKENMERKEN

proces van informatiebehoeftebepaling.

Bij de vaststelling van de informatiebehoeften kan men verschillende situaties onderscheiden afhankelijk van de stabiliteit en de duidelijkheid van de informatiebehoeften. Duidelijkheid wil zeggen dat men in staat is precies te omschrijven aan welke eisen de te ontwikkelen software moet voldoen. Stabiliteit wil zeggen dat de informatiebehoeften, in de tijd gezien, niet of nauwelijks veranderen. Afhankelijk van die duidelijkheid en stabiliteit kan men een onderscheid maken in:

a. informatiebehoeften zijn duidelijk en stabiel;
b. informatiebehoeften zijn duidelijk en niet stabiel;
c. informatiebehoeften zijn niet duidelijk, maar wel stabiel;
d. informatiebehoeften zijn niet duidelijk en niet stabiel.

De hier geschetste situaties zijn ideaaltypisch. In werkelijkheid zal er sprake van een meer geleidelijke overgang van de ene situatie naar de andere.

Davis (1982) heeft het in dit verband over de onzekerheid in het proces van informatiebehoeftebepaling. De betreffende onzekerheid wordt met name bepaald door karakteristieken van de omgeving waarvoor de software wordt ontwikkeld. Voorbeelden van factoren die

de onzekerheid vergroten zijn ondermeer: ontbreken van duidelijke processen in de omgeving, weinig routine-werkzaamheden en weinig stabiliteit in de structuur van het desbetreffende deel van de organisatie. Naast onzekerheid over het bestaan van duidelijke systeemspecificaties en de stabiliteit van deze specificaties, noemt Davis ook de mate waarin gebruikers informatiebehoeften kunnen specificeren en analisten informatiebehoeften kunnen achterhalen en evolueren. Het grootste probleem om informatiebehoeften duidelijk te omschrijven, wordt veroorzaakt door de tekortkomingen van de mens als informatieverschaffer. Voor de vaststelling van de informatiebehoeften heeft dit een aantal konsekwenties. Zo heeft ieder mens de neiging "stokpaardjes te berijden". Dit betekent dat de gebruiker geneigd is zich primair te laten leiden door eigen voorkeur en belangen in plaats van door de belangen van de organisatie. Het gevaar bestaat dat hij daardoor relevante eisen over het hoofd ziet. De mens is verder beperkt om objectief en precies te zijn. Dit wil zeggen dat de gebruiker problemen heeft om exact te zijn bij het formuleren van zijn of haar behoefte aan gegevens. Vaak komt het voor dat eisen in vage bewoordingen geformuleerd zijn. Bij de vertaling van de eisen naar ontwerp en code ontstaan dan interpretatieproblemen. Ten slotte noemt Davis de eigenschap dat de mens zich meer laat beïnvloeden door gebeurtenissen naarmate deze recenter zijn. Een en ander betekent dat de gebruiker een groter belang hecht aan zaken die onlangs hebben plaats gevonden. De structurele informatiebehoefte blijft daardoor enigszins verborgen.

De totale onzekerheid in het proces van informatiebehoeftebepaling heeft volgens Davis directe gevolgen voor wijze waarop de informatiebehoeften bepaald moeten worden. Hij onderscheidt een aantal methoden van informatiebehoeftebepaling, variërend van interviews (bij weinig of geen onzekerheid) tot prototyping (bij grote onzekerheid).
Het voorgaande laat zien dat de mate van stabiliteit en duidelijkheid van de informatiebehoeften zal leiden tot een andere aanpak van beheersing, met name in het begin van het ontwikkelproces. In paragraaf 7.5 wordt hierop nader ingegaan.
Van Vliet (1987a) signaleert eveneens dat het specificeren van informatiebehoeften moeilijk is. Daarbij is het veelal niet voldoende de bestaande situatie als enige leidraad te gebruiken. Een belangrijke reden waarom een organisatie over wenst te gaan tot automatisering is immers in heel wat gevallen onvrede met de bestaande situatie. Een en ander betekent dat gebruiker en software-ontwikkelaar een beeld moeten krijgen van de gewenste toekomstige situatie. Dit is moeilijk, met name in die situaties, waarin de gebruiker weinig ervaring heeft met automatisering, de ontwikkelaar niet bekend is met het toepas-

singsgebied ("niet-materiekundig") en de innovatiegraad van de te ontwikkelen software hoog is.

Ongeacht of de informatiebehoeften wel of niet goed in kaart zijn te brengen, geldt dat bij veel applicaties de behoeften van de gebruikers evolueren (situaties b en d). Dit betekent dat de software die wordt ontwikkeld, afgestemd wordt op een zich steeds wijzigend doel. Vaak wordt dit verschijnsel bij de ontwikkeling van software niet voldoende onderkend. Tijdens de ontwikkeling worden dan pogingen ondernomen de software te laten voldoen aan nieuwe eisen, waarop het oorspronkelijke ontwerp niet was afgestemd. Een gevolg hiervan kan zijn dat kosten en doorlooptijd fors overschreden worden. Een ander uiterste is dat ontwikkelaars de eisen hebben bevroren en er niet van doordrongen zijn dat wijzigingen onvermijdelijk zijn. Gevolg daarvan kan zijn dat de software bij oplevering niet voldoet aan de verwachtingen van de gebruikers.

samengesteldheid produkt.

Bij software en met name bij omvangrijke en complexe software is de samengesteldheid meestal groot. Samengesteldheid wordt hier in de volgende twee betekenissen gebruikt:

- software-ontwikkeling is meer dan alleen maar het maken van code. Er moeten talrijke soorten documenten worden gemaakt, procedures worden opgesteld, testsets worden gemaakt, er moet worden gezorgd voor opleiding, conversie en auditing enz. Deze samengesteldheid heeft primair betrekking op het proces.
- software is veelal opgebouwd uit een groot aantal modules. Bij het ontwikkelen van software zal men erna dienen te streven het principe van koppeling en samenhang toe te passen. Yourdon en Constantine (1979) geven aan dat samenhang betrekking heeft op de activiteiten binnen een subsysteem en koppeling op de relaties tussen subsystemen. De koppeling-samenhang regel zegt dat men bij het decomponeren van een systeem moet streven naar een zo sterk mogelijke samenhang binnen subsystemen en een zo zwak mogelijke koppeling tussen de subsystemen onderling. De mate van koppeling heeft gevolgen voor onder andere de onderhoudbaarheid en uitbreidbaarheid van het eindresultaat en voor de beheersbaarheid van het ontwikkelproces. In paragraaf 7.4 wordt hier nader op ingegaan.

mogelijkheden van hergebruik.

In de gehanteerde checklist (Bemelmans 1986) wordt het aspect "specificiteit van eindprodukten" genoemd. Hiermee wordt bedoeld, de mate waarin het eindprodukt wel of niet uitwisselbaar is met andere produkten. Voor software is dit aspect minder geschikt voor het karakteriseren van de P. In plaats van dit aspect heb ik gebruik gemaakt van het aspect "mogelijkheden voor hergebruik". Voor software wil dit zeggen dat, naarmate de software toegesneden is op een specifiek applicatiedomein, de software minder herbruikbaar is.

complexiteit van het produkt.

Een kenmerk dat nauw gerelateerd is met het kenmerk samengesteldheid is complexiteit. Bij complexiteit dient men een onderscheid te maken in de complexiteit van het probleem en de complexiteit van de oplossing c.q. het produkt. Het is goed mogelijk dat voor een probleem van een geringe complexiteit een oplossing van een hoge complexiteit vereist is. Wat beschouwd moet worden als een eenvoudig of complex probleem valt buiten het kader van dit onderzoek. In de cognitieve psychologie wordt op dit gebied uitgebreid onderzoek uitgevoerd. Complexiteit van het probleem is aan subjectiviteit onderhevig; dat wil zeggen dat opdrachten die door experts als eenvoudig worden gekwalificeerd, door beginnelingen als complex worden ervaren. Dit ligt anders bij complexiteit van het produkt. Brooks (1987) signaleert dat software van nature complex is. Als oorzaak noemt hij ondermeer het feit dat in een programma geen twee identieke delen voorkomen (tenminste boven het nivo van een statement). Als twee delen gelijk zijn, dan worden ze samengevoegd in een routine. In dit opzicht verschilt software fundamenteel van computers, gebouwen of auto's, die opgebouwd zijn uit veel identieke elementen. Een tweede oorzaak van de hoge complexiteit van software heeft te maken met het grote aantal toestanden in software. Verder wijst Brooks er op dat de complexiteit van software meer dan lineair toeneemt bij een toename van de omvang.

In de beschrijving van het P-B-I model signaleert Bemelmans, dat de bestuurbaarheid ondermeer afneemt naarmate de complexiteit van het produkt toeneemt. Omdat software doorgaans complex is, zal de beheersing van de ontwikkeling ervan veelal lastig zijn.

economische waarde produkt.

De economische waarde van software is veelal hoog. Dit wil zeggen dat het bij de ontwikkeling van met name grote programma's, gaat om forse investeringen. Er is doorgaans sprake van hoge technische, organisatorische en financiële risico's. Het vereiste kwaliteitsniveau in termen van produktkwaliteit, levertijd, levertijdbetrouwbaarheid, bedieningsgemak enz. is in vele gevallen eveneens hoog. Hoe hoog dit niveau is, wordt mede bepaald door het doel waarvoor de software wordt ontwikkeld. Zo zullen bijvoorbeeld bij de ontwikkeling van software voor de besturing van een compact disc speler of een ruimteveer hoge eisen gesteld worden aan een oplevering op het afgesproken tijdstip en aan de kwaliteit van de geleverde software.

meetbaarheid/testbaarheid produkt.

Het laatste kenmerk van software dat ik in dit kader wil noemen heeft betrekking op meetbaarheid. Meetbaarheid is een aspect dat niet wordt genoemd in de gehanteerde checklist. Omdat meetbaarheid, of beter geformuleerd het gebrek aan meetbaarheid, zo'n kenmerkende eigenschap is van software, wil ik nader ingaan op dit aspect. Bovendien, zo zal in het verdere verloop van dit hoofdstuk blijken, heeft meetbaarheid directe gevolgen voor de wijze van beheersing.

Software is moeilijk meetbaar. Bij meetbaarheid moet een onderscheid worden gemaakt in de meetbaarheid gericht op het produkt en meetbaarheid gericht op het ontwikkelproces van software. Op de meetbaarheid van het proces wordt ingegaan in paragraaf 7.3.3 (proceskenmerken). Het is lastig de kwaliteit van software tijdens de ontwikkeling te meten c.q. te testen. Bijvoorbeeld: hoe meet je of het ontwerp voldoet aan de vereiste betrouwbaarheid ? Over testen van software merkt Parnas (1986) op dat het aantal toestanden in software dusdanig groot is dat testen van alle toestanden onbegonnen werk is. Verder signaleert hij dat voor software geldt dat kleine veranderingen in de specificaties vaak grote gevolgen hebben voor het eindprodukt. Verder geldt voor software dat door de correctie van een geconstateerde fout, vaak nieuwe fouten geïntroduceerd worden.

Vergelijken we software met een fysiek produkt, bijvoorbeeld hardware, dan geldt dat software niet aan technische slijtage onderhevig is. Voor hardware geldt in zijn algemeenheid dat het aantal storingen per tijdseenheid zich gedraagt volgens het badkuipmodel. Dit betekent dat als gevolg van "kinderziektes" direct na het in gebruik nemen het aantal storingen per tijdseenheid groot is. Na verloop van tijd neemt dit aantal geleidelijk af tot een min of meer constant niveau. Als

gevolg van onder andere slijtage, trillingen, stof, bloot staan aan extreme temperaturen e.d. zal vanaf een bepaald moment het aantal storingen weer toenemen. De hardware begint technisch te slijten. Zoiets geldt niet voor software. Het foutenverloop bij software zou men zich als volgt kunnen voorstellen (Pressman 1987): evenals bij hardware zijn er in het begin veel fouten. Wanneer deze fouten zijn hersteld en geen nieuwe fouten worden geïntroduceerd, neemt het aantal optredende fouten per tijdseenheid geleidelijk af. De hier geschetste situatie is echter erg gesimplificeerd. In werkelijkheid komt het vaak voor dat door herstel van oude fouten, nieuwe fouten worden geïntroduceerd. Verder wordt software aangepast als gevolg van veranderende eisen. Deze aanpassingen veroorzaken ook weer fouten. Voordat het aantal optredende fouten per tijdseenheid op een laag niveau is gebracht, komt er weer een nieuw verzoek om aanpassing. Het gevolg hiervan is dat het aantal optredende fouten per tijdseenheid in de loop van de tijd (afhankelijk van het aantal en de aard van de wijzigingen) in de meeste gevallen geleidelijk toeneemt tot een "onacceptabel" hoog niveau.
Evenals bij hardware wordt dan een punt bereikt waarop de software voor vervanging in aanmerking komt. In die zin "verslijt" ook software, al zijn de oorzaken anders.

7.3.2. KENMERKEN VAN DE PRODUKTIEMIDDELEN

Bij het ontwikkelen van software is personeel verreweg het belangrijkste produktiemiddel. Andere produktiemiddelen die genoemd kunnen worden, zijn ontwikkelhardware en ontwikkelsoftware (workbenches e.d.).

teamwork.

Kenmerkend bij software-ontwikkeling is dat met name bij grote programma's, de ontwikkeling ervan door verschillende teams wordt uitgevoerd. Naarmate de omvang van de te ontwikkelen software toeneemt, zullen er veelal meer mensen bij het project betrokken worden en zal de onderlinge communicatie c.q. het onderling afstemmen van activiteiten toenemen.

multidisciplinair, specialisten.

Een volgend kenmerk is dat experts van verschillende disciplines moeten samenwerken. Het aantal disciplines dat bij de ontwikkeling

wordt betrokken, kan groot zijn: analisten, ontwerpers, programmeurs, gegevensbeheerders, database ontwerpers, data communicatie- en netwerkontwerpers, gebruikers, opdrachtgevers, projectmanagers, stuurgroep, verkooppersoneel enz. Het rapport funktie-ordening van het NGI (1986) geeft een goed overzicht van de vele funkties op informaticaterrein. Het opleidingsniveau van de betrokken specialisten is doorgaans hoog. Het werk vereist hoge intellectuele en creatieve vaardigheden.

beschikbaarheid en economische waarde produktiemiddelen.

Een ander aspect waarmee rekening gehouden moet worden is het dynamisch karakter van het aanbod en de beschikbaarheid van automatiseringspersoneel. Door de snelle ontwikkelingen is het personeel genoodzaakt een groot deel van zijn werktijd te besteden aan opleiding en bijscholing. De beschikbaarheid van ontwikkelpersoneel is daardoor relatief laag. Omdat de economische waarde van automatiseringspersoneel hoog is, kan een organisatie zich veelal niet permitteren een grote overcapaciteit aan te houden.

7.3.3. KENMERKEN VAN HET PRODUKTIEPROCES

ontwikkelen in stappen.

Een belangrijk kenmerk van software-ontwikkeling betreft de levenscyclus van software. De basisgedachte is dat de ontwikkeling van software in een aantal stappen gebeurt. Enigszins gesimplificeerd onderscheiden we de stappen, die in figuur 7.2 in beeld zijn gebracht (Wijnen, Rennes en Storm 1984).

Figuur 7.2 : Het ontwikkelen van software in stappen.

Het initiatief (IDEE) tot de ontwikkeling van software komt veelal voort uit ofwel problemen die bij gebruikers leven ofwel uit mogelijkheden die de gebruikers zien. Nadat bepaald is welk onderwerp aangepakt wordt, volgt een nadere analyse hiervan om te komen tot een definitie van eisen (WAT). Uitgaande van de eisen wordt de te realiseren uitvoering/oplossing gekozen (HOE). Is eenmaal voor een oplossingsrichting gekozen, dan moet de uitvoering hiervan worden voorbereid (HOE TE DOEN) zodat de realisatie (DOEN) plaats kan vinden. De laatste stap (IN STAND HOUDEN) betreft de nazorg van de gerealiseerde software. Deze stap valt doorgaans buiten de ontwikkeling. Het principe van een stapsgewijze aanpak heeft ondermeer geleid tot een aantal fasemodellen voor software-ontwikkeling. De belangrijkste drijfveer voor deze fasegerichte aanpak is de beheersbaarheid. Vooral als de onzekerheid groot is, het project omvangrijk is, complex en innovatief, wordt de beheersbaarheid lastiger. In paragraaf 6.5 zijn de kleine en de grote stapmethoden beschreven als exponenten van een stapsgewijze aanpak van software-ontwikkeling.

meetpunten in het proces.

Een kenmerk dat direct gerelateerd is aan het vorige kenmerk, is het inbouwen van meetpunten in software-ontwikkeling. Op deze punten moeten worden nagegaan of het (deel)produkt op de juiste wijze wordt gemaakt alsmede of het juiste (deel)produkt wordt gemaakt. Een en ander kan leiden tot de beslissing om of verder te gaan met een volgende stap of om een voorgaande stap geheel of gedeeltelijk over te doen of om het projekt te stoppen. Het doel van het inbouwen van dergelijke meetpunten is dus de beheersing van de projectvoortgang en de produktkwaliteit. Beheersen en begroten van software-ontwikkeling wordt moeilijker naarmate er een grotere onzekerheid bestaat over datgene wat ontwikkeld moet worden en de wijze waarop dit moet geschieden. Wat de gevolgen hiervan zijn voor het inbouwen van meetpunten in het proces wordt in paragraaf 6.6 aangegeven.

meetbaarheid voortgang proces.

De voortgang van het ontwikkelproces is soms moeilijk meetbaar. Bijvoorbeeld: een ontwikkelaar kan een maand lang nadenken over oplossingsmogelijkheden. Ogenschijnlijk is er geen vooruitgang in de ontwikkeling, terwijl de oplossing in het hoofd van de ontwikkelaar reeds aanwezig is.

betrokkenheid gebruiker bij ontwikkelproces.

Kenmerkend voor het ontwikkelen van software is de rol van de gebruiker. Blokdijk en Blokdijk (1987) vergelijken de betrokkenheid van gebruikers bij het specificeren en ontwerpen van software met de inbreng van bewoners van een flat of een huis in de architectuurfase. Wat zou er met de bouwkosten en -tijd gebeuren als alle toekomstige bewoners van een flatgebouw betrokken zouden worden bij de vaststelling van de eisen en samen aan het ontwerp zouden werken. Anders ligt het wanneer een bewoner/eigenaar een architect opdracht geeft tot de bouw van een huis voor persoonlijk gebruik. De betrokkenheid van de toekomstige bewoner zal nu vele malen groter zijn. Er is een parallel te trekken met het ontwikkelen van software. Walston en Felix (1977) geven in hun onderzoek aan dat betrokkenheid van de gebruikers bij de opstelling van de software-eisen een belangrijke factor is. Zij signaleren dat een toename in deze betrokkenheid leidt tot een aanzienlijke toename in ontwikkelkosten, kwaliteit en acceptatiegraad. Winkelhage en Marock (1980) onderscheiden een aantal mogelijkheden van betrokkenheid. Voor welke mogelijkheid in een bepaalde situatie gekozen moet, is afhankelijk van het aantal toekomstige gebruikers. Dit aantal kan hoog zijn, bijvoorbeeld bij standaardsoftware. Er kan ook sprake zijn van slechts een of enkele gebruikers. In het eerste geval kan betrokkenheid het best gerealiseerd worden door vertegenwoordigers vanuit de gebruikersorganisatie of door consultatie van experts op dit terrein. Zijn er weinig gebruikers dan kunnen alle gebruikers bij de ontwikkeling betrokken worden. Deze laatste aanpak benadert de meest ideale vorm van betrokkenheid. Nadelen van deze benadering zijn, zoals uit het onderzoek van Walston en Felix blijkt, echter vaak hoge kosten en lange ontwikkeltijd. Dit nadeel gaat des te zwaarder wegen naarmate het aantal betrokkenen toeneemt.

7.4. B: KENMERKEN VAN BEHEERSING

7.4.1. TYPERING VAN B

In paragraaf 7.3 heb ik de kenmerken van software en softwareontwikkeling in beknopte termen toegelicht. Het doel van deze toelichting is het geven van een antwoord op de hamvraag van het P-B-I model: "Welke kenmerken van P determineren de wijze van B".

Alvorens nader in te gaan op het antwoord op deze vraag, wil ik een typering geven van B. Bemelmans (1986) geeft in de beschrijving van het P-B-I model aan dat men wat betreft B de volgende zaken kan onderkennen:

1. *doelstellingen.*
   Welke doelen streeft men na bij de beheersing van software-ontwikkeling. In paragraaf 6.3.1 is dit punt reeds beknopt toegelicht. Zo kan er sprake zijn van een situatie, waarbij de kwaliteit van de software centraal staat. Er gelden weinig of geen budgettaire beperkingen: "het produkt moet goed zijn ongeacht de kosten en tijd die daarmee gemoeid zijn". Een heel andere situatie treedt op als het accent bij de ontwikkeling op de efficiency ligt. De software moet in zo'n geval gerealiseerd worden met een zo minimaal mogelijke benutting van geld, tijd en middelen. Het kan ook zo zijn dat de levertijd van de software centraal staat. Concentreren we ons op de vraag welke konsekwenties dergelijke doelstellingen op de beheerswijze hebben, dan kunnen een aantal zaken worden opgemerkt. Allereerst geldt dat beheersing eenvoudiger wordt naarmate de doelstellingen minder dwingend geformuleerd zijn. Het meest extreme geval doet zich voor als er geen beperkingen zijn wat betreft geld, tijd, middelen en kwaliteit. Zodra er duidelijke voorwaarden worden gesteld aan de ontwikkeling treedt een beheersvraagstuk op. In paragraaf 6.3.1 is reeds gewezen op het spanningsveld tussen de doelstellingen onderling. Staat maximaliseren van de kwaliteit centraal dan kan vanuit beheersoptiek niet eveneens het accent worden gelegd op maximaliseren van de efficiency en minimaliseren van de doorlooptijd. Wil het proces beheersbaar blijven dan zullen er concessies gedaan moeten worden. Welke concessies dat zijn en door wie deze gedaan moeten worden, is een beheersvraagstuk. Samenvattend geldt dat elke mix van doelstellingen leidt tot een bepaalde wijze van beheersing en dus tot een specifieke manier van organiseren van beslissingscentra.

2. *ontkoppelde versus integrale beheersing.*
   De keuze tussen ontkoppelde of integrale beheersing wordt bij de ontwikkeling van software bepaald door de omvang van het uit te voeren werk, de samengesteldheid van het produkt en de specialisatie die bij de uitvoering van dat werk vereist is. Is er sprake van een omvangrijk project waarvoor een hoge mate van specialisatie en ook verschillende specialismen nodig is, dan vraagt dit om geïntegreerde beheersing. Omdat het werk omvangrijk en specialistisch is, wordt het uit te voeren werk opgesplitst in fasen, hoofdtaken, taken enz. en wordt het produkt opgesplitst in een aantal deelprodukten. Meerdere teams, elk gespecialiseerd in een bepaald

onderdeel van de te ontwikkelen software, worden ingeschakeld. Dit betekent dat een vorm van coördinatie tussen deze teams noodzakelijk is. Zo kan een team bij de uitvoering c.q. voortgang van zijn eigen werk afhankelijk zijn van een ander team. Bijvoorbeeld: team A kan pas beginnen met de ontwikkeling van haar module als ontwikkelteam B gereed is met een deel van haar werk. Treden er vertragingen op bij team B of stuit team A bij de ontwikkeling van haar module op tekortkomingen in de kwaliteit van het resultaat van team A of op onduidelijkheden in documentatie, dan heeft dit directe gevolgen voor de voortgang van team A. Brooks (1975) wijst op deze afstemmingsproblemen en geeft aan dat een belangrijk deel van de overschrijdingen van geplande ontwikkeltijden wordt veroorzaakt doordat teams "op elkaar wachten". Een andere vorm van "wachten" wordt veroorzaakt door de zogenaamde interfasetijden. Hiermee wordt bedoeld de tijd die verloopt tussen het afronden van een fase, het goedkeuren van de resultaten van die fase en het starten van de volgende fase. Deze interfasetijden kunnen soms zeer aanzienlijk zijn. Treden dergelijke lange wachttijden op dan is er sprake van een slechte integrale beheersing. In dit onderzoek zal ik niet nader ingaan op de wijze waarop integrale beheersing gerealiseerd kan worden.

De wijze van beheersing is van een geheel andere orde als het uit te voeren project klein is of het werk opgedeeld kan worden in onafhankelijke deelprojecten. Van afstemmingsproblemen is dan geen sprake. Beheersing hoeft niet door middel van geïntegreerde besturing gerealiseerd te worden, maar kan gebeuren via ontkoppelde besturing c.q. autonome werkgroepen.

3. *de soort beheersingsorganisatie van de produktie-middelen.*
In de literatuur over produktiebeheersing is het gebruikelijk twee extreme vormen van organisatie van produktiemiddelen te onderscheiden. Deze twee vormen zijn:

- de produktgeoriënteerde opstelling van capaciteiten,
- de funktiegeoriënteerde opstelling van capaciteiten.

Tussen deze twee uitersten is een volledig continuüm van mogelijke tussenvormen mogelijk (in 't Veld 1981). Bij de produktgerichte structuur of interne specialisatie worden alle mensen en middelen die nodig zijn om een bepaald produkt of dienst te maken in één afdeling bijeengebracht. Het materiaal voor dat ene produkt blijft dus binnen die ene afdeling. Hierdoor ontstaat de mogelijkheid de mensen en middelen op te stellen in de volgorde van de fabricagebewerkingen van dat ene produkt. Het summum

van de produktgerichte structuur is een continuproduktie, zoals bij sommige chemische processen. Andere vormen van een produktgerichte structuur zijn continue lijnproduktie en intermitterende lijnproduktie. Het accent ligt op een efficiënte, routinematige wijze van produceren. Het streven bij een produktgerichte structuur is om de produktiemiddelen volledig op elkaar af te stemmen en tussenvoorraden te vermijden, met andere woorden de produktielijn volledig uit te balanceren. Dit is niet altijd mogelijk. Soms worden de bewerkingstijden bij een bewerkingsstation bij het gevraagde produktietempo te lang of te kort om acceptabel te zijn. Hierdoor ontstaan allerlei varianten van een produktgerichte structuur zoals de lijnstop, groepentechnologie, batches in lijnen enz. Van een geheel andere situatie is sprake bij een funktiegeoriënteerde opstelling van de produktiemiddelen. De functionele structuur kenmerkt zich doordat men gelijksoortige bewerkingen in één afdeling of groep bij elkaar brengt. Het produktmateriaal doorloopt afhankelijk van de voor de transformatie noodzakelijke bewerkingen een deel van deze afdelingen. Omdat elk volgend te fabriceren produkt in meer of mindere mate verschilt van het voorgaande, kan men hier niet terug vallen op een vaste volgorde in de wijze van produceren. Voor elk nieuw produkt moet worden nagegaan welke capaciteiten in welke mate en in welke volgorde aangesproken moeten worden. Evenals bij de produktgerichte structuur treft men bij de funktionele structuur allerlei varianten aan. Planning en voortgangsbewaking zijn in een funktiegerichte structuur belangrijke componenten van beheersing.

Vertaalt men het onderscheid in produkt- en funktiegerichte structuur naar de ontwikkeling van software, dan zijn de volgende parallellen te trekken. Produktgericht wil in dit geval zeggen dat de ontwikkeling , "de fabriek" , georganiseerd is volgens een opsplitsing in fasen. Funktiegericht betekent dat men binnen een ontwikkelafdeling een onderscheid maakt in een aantal specialisaties, zoals een databasegroep, een netwerkgroep, een AI groep, een groep COBOL-applicatieontwerpers, enz.
Is er sprake van een grote zekerheid over het te realiseren produkt, dan is ver doorgevoerde specialisatie en efficiency mogelijk. Een produktgerichte structuur sluit hier het best bij aan. Anders ligt het als er onzekerheid bestaat over de te ontwikkelen software. Bij aanvang van het ontwikkeltraject wordt nagegaan welke specialismen men denkt in te moeten zetten. Volgens een geplande routing, een bepaalde volgorde, leveren de verschillende specialismen hun bijdragen.

In het verdere verloop van dit hoofdstuk zal ik mij, bij het aangeven van de samenhang tussen P en B, vooral concentreren op de gevolgen van de kenmerken van P voor de organisatie van de produktie-middelen. Ook Bemelmans (1986) legt bij de beschrijving van het P-B-I model en de typering van B de nadruk hierop. Ik zal laten zien dat bij de ontwikkeling van software situaties te onderscheiden zijn die goed aansluiten bij een funktiegerichte en een produktgerichte organisatie van de produktiemiddelen en dat de belangrijke beheersmodules bij beide organisatievormen goed te vertalen zijn naar de softwareontwikkeling. De andere twee bovengenoemde zaken die te onderkennen zijn bij een typering van B, doelstellingen en ontkoppelde versus integrale besturing, zullen zijdelings ter sprake komen bij het aangeven van de samenhang tussen P en B.

7.4.2. DE SAMENHANG TUSSEN P EN B

Om de samenhang tussen P en B te bepalen, kies ik voor de volgende aanpak. Uitgangspunt is het besturingsparadigma. Hierin is een aantal voorwaarden genoemd, waaraan voldaan moet worden, wil er sprake zijn van effectieve beheersing. De vraag is in hoeverre bij het ontwikkelen van software aan deze voorwaarden wordt voldaan. Om deze vraag te beantwoorden, is een analyse van de kenmerken van P noodzakelijk. De mate waarin aan genoemde voorwaarden wordt voldaan, met andere woorden de mate van beheersbaarheid van P wordt bepaald door variaties in kenmerken van P. Zo kan bijvoorbeeld de stabiliteit van de informatiebehoeften van geval tot geval sterk variëren. Deze stabiliteit, zo zal in deze paragraaf worden gesignaleerd, heeft directe gevolgen voor de wijze van beheersing. Niet bij alle kenmerken is er sprake van dergelijke sterke variaties. Dit zijn kenmerken die bij alle soorten software min of meer dezelfde zijn. Dergelijke kenmerken bepalen weliswaar de beheersbaarheid, maar leiden niet tot *verschillende* vormen van B. Dit geldt ondermeer voor het kenmerk "personeelsdominant". Bij de beantwoording van de vraag, welke kenmerken van P determineren de wijze van B, heb ik mij beperkt tot de meest dominante P kenmerken. Kenmerken zijn dominant als ze sterk in waarde kunnen variëren.

In paragraaf 7.3 zijn belangrijke kenmerken van P genoemd en uitgesplitst naar de categorieën produkt, proces en middelen. In deze paragraaf zal per categorie worden aangegeven welke kenmerken vanuit beheersingsoptiek dominant zijn. In dit verband zal ik spreken

over de mate van produktzekerheid, proceszekerheid en middelenzekerheid. Zo kan de produktzekerheid hoog of laag zijn. Hoge zekerheid betekent in dit geval onder andere dat de informatiebehoeften stabiel zijn. Door de mate van zekerheid van de drie categorieën te combineren ontstaan acht verschillende situaties van software-ontwikkeling, die elk vragen om een eigen wijze van beheersing. Een voorbeeld van zo'n combinatie is: produkt is zeker en proces en middelen zijn onzeker. Ik zal laten zien dat niet alle combinaties voor software-ontwikkeling van toepassing zijn. Een en ander leidt ertoe dat de acht combinaties terug te brengen zijn tot een viertal (ideaaltypische) beheerssituaties. Deze vier beheerssituaties zullen in deze paragraaf worden beschreven. Bij deze beschrijving zal blijken dat twee beheerssituaties duidelijk parallellen vertonen met de in paragraaf 7.4.1 genoemde funktie- en produktgeoriënteerde organisatie van produktiemiddelen. De twee andere beheerssituaties zijn mengvormen van deze twee beheersingsorganisaties. De beschrijving van de vier beheerssituaties is gemaakt vanuit vijf verschillende invalshoeken. Zo wordt per onderscheiden beheerssituatie aangegeven wat de aard van het beheersprobleem is, wat het primaire doel van de beheersing is, welk coördinatiemechanisme en welke managementstijl bij de beheersing gehanteerd dient te worden, welke ontwikkel- c.q. beheersstrategie het best aansluit bij elke situatie en tenslotte welke wijze van begroting de voorkeur geniet.

De dominante produkt/marktkenmerken die bepalend zijn voor de mate van ***produktzekerheid*** zijn:

1 - het bestaan van duidelijke en volledige informatiebehoeften,
2 - de stabiliteit van de informatiebehoeften,

De mate van ***proceszekerheid*** wordt bepaald door:

3 - de mogelijkheden om het proces bij te sturen,
4 - de meetbaarheid van het proces c.q. het inzicht dat men heeft in het proces. Hiermee wordt ook bedoeld kennen en meten van effecten van bijsturen.

De mate van ***middelenzekerheid*** wordt bepaald door:

5 - de economische waarde en de beschikbaarheid van het juiste personeel.

Bij het bepalen van de verschillende vormen van B, heb ik mij geconcentreerd op de voorgaande vijf kenmerken, omdat deze naar mijn idee het meest dominant zijn. De overige kenmerken, genoemd in paragraaf 7.3, zijn over het algemeen bij alle soorten software dezelfde, zijn te herleiden tot andere kenmerken of hebben geen duidelijke samenhang met de mate van zekerheid.

Door de vijf "dominante" P-kenmerken te combineren ontstaan er in principe 32 verschillende combinaties. De twee meest extreme combinaties zijn die, welke voldoen aan de volgende voorwaarden:

Combinatie 1:de informatiebehoeften zijn stabiel, precies te omschrijven, men "kent" het proces, er zijn voldoende mogelijkheden tot bijsturing en er zijn geen problemen met de beschikbaarheid van het personeel. Voor beheersen en begroten van software-ontwikkeling is dit is de meest ideale situatie.

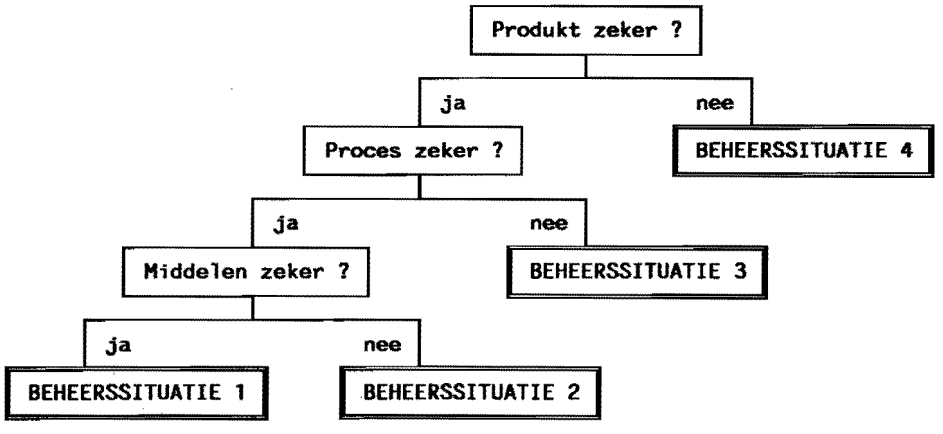
Combinatie 2:De informatiebehoeften zijn niet precies te omschrijven en zijn bovendien sterk aan verandering onderhevig. Procesonzekerheid is hoog, met andere woorden: er zijn relatief weinig mogelijkheden om het proces bij te sturen en de effecten van stuurmaatregelen te meten. Tenslotte zijn er problemen met de beschikbaarheid van personeel: dat wil zeggen grote middelenonzekerheid.

De overige mogelijke combinaties vormen een continuüm met deze twee combinaties als uitersten. Om een ordening aan te brengen in dit groot aantal combinaties heb ik de twee dominante produkt/marktkenmerken samengevoegd tot één variabele *produktzekerheid* en de drie dominante proceskenmerken tot één variabele *proceszekerheid.* Het dominante middelenkenmerk wordt aangeduid als de variabele *middelenzekerheid.* In dit onderzoek wordt er gemakshalve van uitgegaan dat zo'n variabele twee waarden kan hebben: de zekerheid is hoog of de zekerheid is laag. Door de mogelijke waarden van de drie variabelen te combineren onstaan acht mogelijkheden. In tabel 7.1 worden deze mogelijkheden genoemd. Niet alle mogelijkheden leiden tot geldige c.q. bestaande beheerssituaties. Mogelijkheden 5 en 6 vervallen omdat ik uitga van de aanname dat als er weinig of geen zekerheid bestaat over de te ontwikkelen software, er geen sprake kan zijn van hoge zekerheid omtrent het uit te voeren proces en de benodigde middelen. Als zodanig zijn mogelijkheid 5 en 6 niet interessant om verder te analyseren. Mogelijkheid 4 en 7 vervallen om een soortgelijke redenering. Als niet bekend is op welke wijze het ontwikkelproces uitgevoerd moet worden, is het niet mogelijk dat er

Tabel 7.1 : Acht mogelijke beheerssituaties.

VARIABELEN	MOGELIJKHEDEN 1 2 3 4 5 6 7 8
PRODUKTZEKERHEID	hoog hoog hoog hoog laag laag laag laag
PROCESZEKERHEID	hoog hoog laag laag hoog hoog laag laag
MIDDELENZEKERHEID	hoog laag laag hoog hoog laag hoog laag

zekerheid bestaat over de middelen die ingezet moeten worden bij dit proces. Door uit te gaan van de hiërarchie "produkt - proces - middelen" worden de acht mogelijkheden teruggebracht tot vier. Deze hiërarchie is in figuur 7.3 weergegeven.



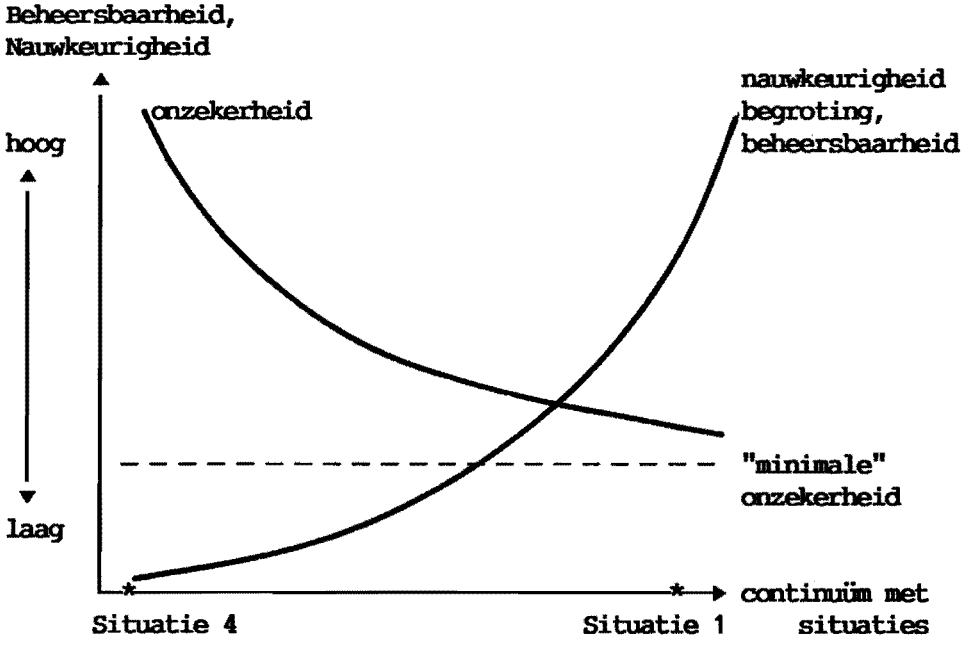
Figuur 7.3 : Een hiërarchie van beheerssituaties.

Deze vier mogelijkheden zullen verder aangeduid worden als de vier ideaaltypische beheerssituaties. De vier beheerssituaties die aldus ontstaan zijn:

Beheerssituatie 1: produkt zeker, proces zeker en middelen zeker (de hierboven genoemde combinatie 1);
Beheerssituatie 2: produkt zeker, proces zeker en middelen onzeker;

Beheerssituatie 3: produkt zeker, proces onzeker en middelen onzeker;
Beheerssituatie 4: produkt onzeker, proces onzeker en middelen onzeker (de eerder genoemde combinatie 2).

De vier ideaaltypische beheerssituaties zullen in paragraaf 7.5 nader worden toegelicht. Voor de overzichtelijkheid zal de toelichting van de vier beheerssituaties gebeuren vanuit vijf verschillende invalshoeken. Deze invalshoeken worden in paragraaf 7.4.3 beschreven. Gaande van situatie 1 naar situatie 4 neemt de produkt-, proces en middelenzekerheid geleidelijk af. Minder zekerheid betekent een afname van de beheersbaarheid en een afname van de nauwkeurigheid van de begroting. In figuur 7.4 is deze samenhang weergegeven.



Figuur 7.4 : De relatie tussen enerzijds produkt-, proces en middelenzekerheid en anderzijds de nauwkeurigheid van de begroting en de beheersbaarheid. De "minimale" onzekerheid heeft betrekking op kenmerken die inherent zijn aan software c.q. software-ontwikkeling.

7.4.3. INVALSHOEKEN VOOR DE BESCHRIJVING VAN DE KENMERKEN VAN B.

Bij de beschrijving van de karakteristieken van de beheersing van software-ontwikkeling wordt gebruikt van vijf invalshoeken. Deze invalshoeken zijn:

a. de aard van het beheersprobleem,
b. het primaire doel bij het beheersen,
c. het te hanteren coördinatiemechanisme c.q managementstijl bij beheersing,
d. de ontwikkelstrategie,
e. de wijze van begroten.

In hoofdstuk 6 zijn, bij de beschrijving van de besturingsconcepten, deze vijf onderwerpen aan de orde geweest. Ik zal daarom in deze paragraaf volstaan met het noemen van de essenties van de vijf invalshoeken.

Bij de aard van het beheersprobleem is een onderscheid gemaakt in een:
- relatieprobleem;
- allocatieprobleem/capaciteitsprobleem;
- ontwerpprobleem;
- exploratieprobleem.

Bij de beschrijving van mogelijke doelstellingen is een onderscheid gemaakt in:
- minimaliseer de doorlooptijd;
- maximaliseer de efficiency;
- maximaliseer de kwaliteit van het eindresultaat;
- gegeven de beperking van de middelen, optimaliseer werving, salariëring, opleiding, e.d.

Er zijn vijf coördinatiemechanismen en vier managementstijlen beschreven in paragraaf 6.4. De vijf genoemde coördinatiemechanismen zijn:
- mutual adjustment;
- direct supervision;
- standardization of work processes;
- standardization of work outputs;
- standardization of worker skills.

De vier genoemde stijlen van management zijn:
- de afscheidingsbasisstijl;
- de relatiebasisstijl;
- de toewijdingsbasisstijl
- de integratiestijl.

In paragraaf 6.5 zijn drie ontwikkelstrategieën beschreven, te weten:
- de grote stapmethode;
- de kleine stapmethode;
- prototyping.

Bij de wijze van begroten tenslotte is een onderscheid gemaakt naar:
- begrotingsmethoden;
- begrotingsfunkties;
- begrotingseenheden.

7.5. EEN TYPOLOGIE VAN BEHEERSSITUATIES

7.5.1. BEHEERSSITUATIE 1: PRODUKT ZEKER, PROCES ZEKER, MIDDELEN ZEKER

Begroten en beheersen van software-ontwikkeling is het eenvoudigst als zich de volgende situatie voordoet:

- de informatiebehoeften waaraan de software moet voldoen zijn precies te omschrijven,
- de informatiebehoeften wijzigen in de tijd niet of nauwelijks,
- het is bekend op welke wijze de software ontwikkeld moet worden; men "kent" het proces,
- men beschikt over voldoende besturingsvariëteit en bekend is wat de gevolgen zijn van stuuracties,
- de beschikbaarheid van produktiemiddelen is goed te reguleren.

In een dergelijke "ideale" situatie hebben we te maken met **rationele, routinematige** besturing. Omdat precies duidelijk is wat het eindresultaat moet zijn, bekend is op welke wijze dit resultaat bereikt moet worden en men exact kan aangeven welke capaciteiten op welk moment gebruikt gaan worden, sluit een **produktgerichte structuur** goed aan bij deze situatie.

Het accent bij de ontwikkeling ligt op de **realisatie**, met andere woorden: hoe kan men, gegeven het pakket van eisen, het eindresultaat op een zo **efficiënt** mogelijke wijze bereiken. De aandacht is niet primair gericht op de analyse van de probleemsituatie, de inventarisatie van de informatiebehoeften of het opstellen van de lijst van eisen, maar op de beheersing van de uitvoering van het project en op de bewaking van de produktkwaliteit.

In een dergelijke zekere en stabiele situatie kan men bij de ontwikkeling van software gebruik maken van een **lineaire** ontwikkelstrategie; dat wil zeggen de opdeling van de ontwikkeling in strikt achtereenvolgende fasen. Door de hoge mate van zekerheid is men goed in staat het project op te splitsen in fasen, activiteiten en taken, kan men per fase/aktiviteit/taak het resultaat precies specificeren en kan men aangeven welke personen met welke vaardigheden een en ander moeten uitvoeren.

Vertalen we deze situatie 1 naar de coördinatiemechanismen van Mintzberg dan geldt in zijn optiek dat coördinatie gerealiseerd dient te worden door **direct supervision**. Coördinatie wordt gerealiseerd via een lijnorganisatie. In termen van Mintzberg kenmerkt deze situatie zich namelijk doordat de "work outputs" gestandaardiseerd zijn. Het eindresultaat ligt immers vast. Hetzelfde geldt voor de "work processess" en de "workers skills". Ook deze liggen vast. Besturingsvariëteit moet gezocht worden in de verdeling van taken, de toewijzing van taken aan mensen, de inrichting van de communicatiestructuur, enz.

De stijl van leidinggeven die in deze situatie het meest effectief is de **afscheidingsbasisstijl**. Het uit te voeren werk ligt vast in regels, procedures en voorschriften, met andere woorden heeft een routinematig karakter. Voor de realisatie kan men volstaan met lager gekwalificeerd personeel. De taak van de leidinggever i.c. de projectmanager ligt vooral op het toekennen van opdrachten en het controleren van de uitvoering ervan.

Als de produkt-, proces en middelenzekerheid groot is kan men bij het opstellen van een begroting gebruik maken van een sterk geformaliseerde werkwijze. Dit betekent dat **begrotingsmodellen** hier goed bruikbaar zijn. Een dergelijke begroting kan dienen als instrument om een softwareproject te beheersen c.q. te **bewaken**. de begroting heeft primair een **taakstellende** funktie. In deze begrotingssituatie kan een **databank** met voltooide project-gegevens een geschikt hulpmiddel zijn. In hoofdstuk 7 zal hierop nader worden ingegaan.

7.5.2. BEHEERSSITUATIE 2: PRODUKT ZEKER, PROCES ZEKER, MIDDELEN ONZEKER

Deze situatie treedt op als de eisen waaraan de software moet voldoen duidelijk, volledig en stabiel zijn. Men heeft tevens een goed inzicht op welke wijze het eindresultaat bereikt moet worden. Men beschikt over voldoende besturingsvariëteit en kent de effecten van stuuracties. Er is echter onzekerheid wat betreft de produktie- middelen.
Het kernprobleem bij de ontwikkeling van software in een dergelijke situatie spitst zich toe op de **beschikbaarheid van personeel.** In paragraaf 7.2 is reeds gesignaleerd dat software-ontwikkeling personeelsdominant is en vraagt om specialistische vaardigheden. De arbeidsmarkt binnen het vakgebied automatisering kenmerkt zich door een nijpend tekort aan voldoende geschoold personeel. De vraag naar software-ontwikkelaars is vele malen groter dan het aanbod. Opleidingsinstituten schieten in kwantitatieve en kwalitatieve zin tekort. Dit valt ondermeer af te leiden uit een aantal bijdragen in het themanummer "onderwijs in de informatica en informatiekunde in Nederland en Vlaanderen" van het maandblad Informatie (Informatie 1988). Met name de sectie, gewijd aan bijdragen uit het bedrijfsleven, benadrukken het verschil tussen vraag en aanbod. Dit probleem wordt vergroot door de snelle veranderingen in het vakgebied. Enerzijds is er sprake van een sterke druk vanuit de technologie; steeds geavanceerdere methoden, technieken en hulpmiddelen volgen elkaar in een hoog tempo op, de technologische mogelijkheden van hardware en systeemsoftware nemen snel toe. Anderzijds heeft de softwareontwikkelaar ook te maken met een voortdurende verschuiving van zijn arbeidsterrein; de toepassingsmogelijkheden nemen sterk toe. Een en ander betekent dat er sprake is van een geringe stabiliteit in de vraag naar software en het aanbod van adekwaat personeel.

Beheersen tendeert naar het oplossen van een **capaciteitsvraagstuk.** Vragen die hierbij een rol spelen zijn ondermeer: hoe krijgt men het project bemand, waar haalt men geschikt personeel vandaan, hoe kan men met de beperkte middelen het gewenste eindresultaat leveren. Ik wil niet nader ingaan op mogelijkheden van scholing, salariëring en loopbaanplanning.

Een mogelijke strategie die gevolgd kan worden bij beschikbaarheidproblemen van adequaat personeel is het **uitbesteden** van het werk. In de optiek van Mintzberg zal men, in geval de stabiliteit van het personeel gering is, erna moeten streven het proces zoveel mogelijk te standaardiseren **(standardization of work processess).** Hierdoor wordt de uitwisselbaarheid c.q. vervangbaarheid vergroot. Richtlijnen

en procedures schrijven voor hoe de verschillende taken uitgevoerd moeten worden. Het gevolg hiervan is dat volstaan kan worden met lager geschoold personeel. Dergelijke maatregelen kunnen alleen effect hebben als de produkt- en procesonzekerheid gering is en de te ontwikkelen software bovendien niet te complex is. Naarmate minder aan deze voorwaarden wordt voldaan, nemen de risico's voor het project sterk toe.

Voor de begroting kan men evenals bij beheerssituatie 1 gebruik maken van **begrotingsmodellen** en **gegevens van voltooide projecten.** Omdat er onzekerheid bestaat omtrent de produktiemiddelen, zal men bij het opstellen van een begroting behoefte hebben aan **gevoeligheidsanalyses.** Met behulp van dergelijke analyses is men in staat na te gaan wat het effect op kosten en doorlooptijd is als er wijzigingen optreden in de in te zetten middelen. Bijvoorbeeld: hoe hoog worden de kosten als in plaats van 4 ontwerpers met een bepaald opleidings- en ervaringsnivo, 3 ontwerpers worden ingezet van een ander nivo.

7.5.3. BEHEERSSITUATIE 3: PRODUKT ZEKER, PROCES ONZEKER, MIDDELEN ONZEKER

Deze situatie treedt op als de informatiebehoeften precies omschreven kunnen worden en gedurende langere tijd stabiel zijn. Het is echter niet duidelijk op welke wijze het project uitgevoerd moet worden, dus hoe het resultaat bereikt moet worden. Men heeft geen inzicht in de gevolgen van mogelijke stuuracties.

Er zijn evenals in beheerssituatie 2 problemen wat betreft de beschikbaarheid van personeel. Er is hier sprake van een **ontwerpprobleem.** Er bestaan geen onzekerheden meer over het te ontwikkelen produkt, maar wel over het uit te voeren proces en de in te zetten middelen. Met ontwerprobleem wordt bedoeld dat antwoord gegeven moet worden op vragen als: wordt het project wel of niet projectmatig aangepakt, welke beslispunten worden er in het project ingebouwd, welke documenten moeten wanneer opgeleverd worden, hoe moeten mensen ingezet worden, hoe worden verantwoordelijkheden en bevoegdheden verdeeld enz.

Een en ander betekent dat van de kostenbepalende factoren (hoofdstuk 5) duidelijkheid bestaat over het *wat* en *voor wie.* Tijdens de ontwikkeling is sturing mogelijk door invloed uit te uitoefenen op de factoren met betrekking tot het *waarmee, hoe* en *door wie.* Wat de invloed van de beïnvloedbare factoren op de kosten en doorlooptijd is, is niet bekend. Dit betekent dan men onvoldoende inzicht

heeft in de gevolgen van het inzetten van extra personeel, andere hulpmiddelen, andere methoden en technieken enz. Het accent bij de uitvoering ligt op de **beheersing** van het ontwikkelproces.

Het coördinatiemechanisme van Mintzberg dat het beste aansluit bij deze situatie is het **standaardiseren van de output.** De uitvoer is immers gespecificeerd. Omdat de kwaliteit van de te ontwikkelen software "vast ligt", is sturing hoofdzakelijk mogelijk via sturing van het proces en de middelen. Over deze twee zaken bestaat echter onzekerheid.

Wil men de software-ontwikkeling in dergelijke onzekere omstandigheden beheersbaar houden, dan zal men met twee zaken rekening moeten houden. Allereerst zal men genoodzaakt zijn overcapaciteit in te bouwen. Wat betreft het proces laat zich dat ondermeer vertalen in **marges** in ontwikkeltijd en -budget. Voor middelen betekent dit bijvoorbeeld extra personeel inschakelen, parallelle ontwikkelteams inzetten enz. Omdat deze situatie zich kenmerkt door een geringe beschikbaarheid van personeel is het aanhouden van overcapaciteit hiervan in deze niet opportuun. Ten tweede zal men als gevolg van de onzekerheid het aantal meetpunten en de meetfrequentie groot moeten maken. In deze situatie ligt de overgang van de **lineaire ontwikkelstrategie** naar de **incrementele.** Naarmate de onzekerheid toeneemt wordt de voorkeur voor de incrementele strategie groter.

Om in een dergelijke situatie te kunnen begroten is het van belang dat degene die de begroting opstelt, beschikt over **gegevens van afgesloten projecten.** In zo'n gegevensverzameling moet gezocht worden naar softwareprojecten die wat betreft het *wat* en *voor wie* gelijkenis vertonen met het uit te voeren project. Aan de hand van deze informatie wordt inzicht verkregen in het gedrag van de *waarmee-, hoe-* en *wie-* factoren in vergelijkbare situaties. In hoofdstuk 8 wordt hierop nader ingegaan.

Evenals in beheerssituatie 2 zal men bij het opstellen van een begroting gebruik moeten maken van **gevoeligheidsanalyses.** Omdat de onzekerheid in deze derde situatie groter is dan in beheerssituatie 2 zal de noodzaak voor dergelijke analyses ook groter zijn. Het is belangrijk dat nagegaan kan worden wat de effectiviteit is van bepaalde stuuracties. Daarbij heeft men geen behoefte aan puntschattingen in termen van "doorlooptijd is 320 mensmaanden waarvan 110 voor analyse, 70 voor ontwerp enz.". Dergelijke exacte cijfers stroken niet met de aard van de problematiek. Voor een projectmanager is het veel interessanter als er aangegeven kan worden hoe gevoelig een begroting is voor bepaalde invloedsfactoren. Bijvoorbeeld: wat is het effect op de doorlooptijd van de inzet van twee extra analisten; hoe ontwikkelen zich de kosten als de projectduur met x dagen wordt ingekort; wat is het effect op de begroting als de complexiteit van

het te ontwikkelen systeem een niveau te hoog of te laag wordt ingeschat, enz. Door op zo'n wijze om te gaan met een begrotingsvraagstuk krijgt de project-manager meer gevoel voor en inzicht in oplossingsmogelijkheden.

7.5.4. BEHEERSSITUATIE 4: PRODUKT ONZEKER, PROCES ONZEKER, MIDDELEN ONZEKER

De moeilijkste situatie bij de beheersing van software-ontwikkeling en het opstellen van een begroting treedt op als er sprake is van een grote onzekerheid en wazigheid. Bij de ontwikkeling van software, met name in de beginfasen, treedt dit verschijnsel maar al te vaak op. Een en ander betekent dat er geen duidelijkheid bestaat over het eindresultaat, noch over de weg waarlangs dit eindresultaat bereikt moet worden en noch over de middelen die men hierbij kan inzetten. Ook heeft men geen duidelijk idee welke factoren bepalend zijn voor de kosten en doorlooptijd; men heeft geen inzicht in het waardenbereik van de verschillende factoren.

Als software-ontwikkeling op deze wijze gekarakteriseerd kan worden, dan sluit een funktiegerichte organisatievorm hier het beste bij aan. Vanwege de produktonzekerheid zal een belangrijk deel van de ontwikkelinspanning gaan zitten in een verkenning van de probleemsituatie en een analyse van de informatiebehoeften. Het uit te voeren werk is vooral **exploratief** van aard.

In de optiek van Mintzberg is coördinatie in een dergelijke situatie niet te realiseren met behulp van standaardisatie. Het eindresultaat is niet precies te omschrijven en is niet stabiel: coördinatie door het standaardiseren van de output is daarom niet mogelijk. De weg waarlangs het resultaat bereikt moet worden is niet bekend: coördinatie door standaardisatie van het proces vervalt. Beschikbaarheid van personeel is een probleem: coördinatie door standaardisatie van in te zetten personeel is moeilijk uitvoerbaar. Coördinatie kan in een dergelijke complexe en onzekere situatie het best gerealiseerd worden door **mutual adjustment.**

Verder signaleert Mintzberg dat voor de hier beschreven situatie **adhocracy** de beste structuurconfiguratie is. Dynamiek, innovatie, complexiteit, onzekerheid en risico's vormen de belangrijke uitgangspunten voor adhocracy. Adhocracy wil zeggen dat **experts** van diverse disciplines in projectgroepen samenwerken. Coördinatie via standaardisatie is niet of nauwelijks mogelijk, er is weinig formalisme. Be-

voegdheden en verantwoordelijkheden liggen bij de experts die het werk uitvoeren. Er is sprake van een **horizontale organisatiestructuur.**

Een belangrijke factor voor het slagen van een softwareproject in een dergelijke onzekere situatie wordt in belangrijke mate bepaald door **commitment** van de uitvoerenden. Het is niet mogelijk het werk op te splitsen in keurig afgebakende activiteiten en precies omschreven modules. Door de onzekerheid is een planning en begroting per definitie onnauwkeurig. De aanpak van het werk zal meer **improviserend** van aard zijn. Bij het leidinggeven zal een afscheidingsstijl averechts werken. Het zich houden aan een planning/begroting kan niet worden afgedwongen, maar zal gedragen moeten worden door alle betrokkenen. Het management zal de juiste voorwaarden moeten scheppen zodat het dure, hoog gekwalificeerd personeel haar werk kan uitvoeren. De **relatiestijl** sluit het beste aan in deze situatie. Het is van belang dat men snel kan inspelen op nieuwe, plotseling optredende omstandigheden.

De voorwaarden om het ontwikkelproces te beheersen zijn minimaal. Men kan immers niet duidelijk aangeven welke software ontwikkeld moet worden. De wijze waarop en de middelen waarmee de software gerealiseerd moet worden zijn als gevolg daarvan eveneens niet duidelijk. Om het uit te voeren werk alsnog beheersbaar te maken, is het aan te bevelen de middelen vast te leggen/zeker te maken. Het doel wordt dan om met deze **gegeven middelen het resultaat te maximaliseren.** Als toelichting het volgende voorbeeld: besloten wordt tot de ontwikkeling van een marketinginformatiesysteem. Het is niet duidelijk aan welke eisen dit systeem moet voldoen, wat het gaat kosten en in de toekomst gaat opleveren, wie het gaat gebruiken, op welke wijze het ontwikkeld moet worden enz. Met andere woorden, de onzekerheid is groot. Om deze te verminderen besluit men tot de volgende aanpak. Voer een voorstudie naar de haalbaarheid van het systeem uit, stel hiervoor x maanden en y gulden beschikbaar en laat de uitvoering ervan over aan hoog gekwalificeerd personeel. Om in dit voorbeeld de ontwikkeling beheersbaar te maken, zijn twee benaderingen gevolgd. Allereerst zijn tijd, geld en personeel geconditioneerd. Vervolgens is men niet begonnen met een plan voor de ontwikkeling van het totale systeem, maar de definiëring van een eerste kleine stap.

Omdat de informatiebehoeften niet duidelijk zijn, is het verstandig de ontwikkeling te beginnen met **prototyping.** De grote onzekerheid maakt het noodzakelijk dat het aantal meetpunten groot is. Hoe groter die onzekerheid, hoe vaker men zal moeten controleren of men het juiste produkt op de juiste wijze ontwikkeld. De **kleine-stapmethode** heeft in een dergelijke situatie de voorkeur.

Een onzekere situatie zoals hier beschreven, treft men vaak aan bij aanvang van software-ontwikkeling. De risico's zijn groot. Beheer-

sen en begroten zijn dan vrijwel onuitvoerbaar. Het is noodzakelijk de risico's te bepalen en te proberen deze te verlagen om beheersen en begroten beter mogelijk te maken. Voor het bepalen van risico's kan men gebruik maken van **risico-analyse.** Het projectmanagement dient tijdens het totale systeemontwikkelingstraject op de hoogte te zijn van de risico's van het betreffende project. De SarBachets Analys geeft een voorbeeld van zo'n risico-analyse. In deze aanpak worden vragen gesteld die betrekking hebben op de:

1. *projectomvang.* Staat de omvang van het project in een redelijke verhouding tot reeds eerder uitgevoerde projecten ?
2. *automatiseringsniveau binnen het bedrijf.* Hoe is het kennis- en ervaringsniveau van het eigen personeel met betrekking tot de te ontwikkelen software ?
3. *technologie.* Hoe vertrouwd is het ontwikkelteam, de gebruikersorganisatie en de leverancier/fabrikant met de technologie die voor het project wordt gekozen ?
4. *projectorganisatie.* Hoe wordt het project georganiseerd en bemand?
5. *projectomgeving.* Onder welke voorwaarden wordt het project ontwikkeld ?

Door het doorlopen van alle vragen op deze vijf gebieden en het analyseren van het verzamelde materiaal, wordt een risico-analyse van het gehele project uitgevoerd. Nadat men op deze wijze inzicht heeft gekregen in de risico's van het project, zal men deze **risico's,** ingeval ze hoog zijn, moeten **verlagen.** Dit kan op veel manieren, waarvan een aantal hierboven reeds zijn genoemd. Deze manieren zijn ondermeer:

- Men stelt de start van het project uit en tracht alsnog meer duidelijkheid te krijgen van het project.
- Er wordt extra tijd en geld uitgetrokken voor voorstudies, analyse van de probleemsituatie, vaststelling van de informatiebehoeften enz. Zo kan men bij bijzonder risicovolle projecten overwegen meerdere teams parallel een voorstudie te laten uitvoeren.
- Afhankelijk van de mate van onzekerheid worden er meer beslispunten geïntroduceerd. Dit betekent dat bij grote risico's vaker de haalbaarheid van het produkt overwogen moet worden en schattingen gemaakt moeten worden voor kosten, tijd en inspanning voor de uitvoering van het project. Het aantal go-no/go beslissingen wordt verhoogd en de tijdspanne tussen deze beslissingen wordt verkleind.
- Men maakt bij de ontwikkeling gebruik van ontwikkelstrategieën als prototyping en incrementeel ontwikkelen.

Bij het opstellen van een begroting heeft het weinig zin gebruik te maken van gegevens van voltooide projecten. Zoeken naar analoge projecten veronderstelt immers dat men weet wat men zoekt c.q. de specificaties van de te ontwikkelen software bekend zijn. Ook de sterk geformaliseerde aanpak met behulp van begrotingsmodellen zal om die zelfde reden weinig succes hebben. Bij het begroten zal men vooral gebruik moeten maken van de **expertmethode** en/of een **delfi-achtige** aanpak.

Tabel 7.2 : De vier beheerssituaties.

KENMERKEN VAN SOFTWARE/ SOFTWARE-ONTWIKKELING	produkt zeker proces zeker middelen zeker	produkt zeker proces zeker middelen onzeker	produkt zeker proces onzeker middelen onzeker	produkt onzeker proces onzeker middelen onzeker
	BEHEERSSITUATIE 1	BEHEERSSITUATIE 2	BEHEERSSITUATIE 3	BEHEERSSITUATIE 4
UITGANGS-PUNTEN ▼				
AARD PROBLEEM	-realisatie	-allocatie	-ontwerp	-exploratie
PRIMAIR DOEL BEHEERSING	-doel gegeven, optimaliseer middelen -efficiency en doorlooptijd	-werving, opleiding, scholing personeel -efficiënt gebruik middelen	-beheersing van het proces	-middelen gegeven maximaliseer resultaat -effectiviteit -verlagen risico
COÖRDINATIE MECHANISME, LEIDINGGEVEN	-standaardisatie produkt, proces, middelen -hierarchie, afscheidingsstijl	-standaardisatie produkt, proces	-standaardisatie proces	-mutual adjustment -commitment -relatiestijl
ONTWIKKEL-STRATEGIE	-lineair -projectmatig -rationaliteit	-uitbesteden -middelenexperts	-incrementeel -procesexperts	-incrementeel -prototyping -improviserend
BEGROTEN	-modellen -databank oude projectgegevens -taakstellend -bewakend	-modellen -databank oude projectgegevens -gevoeligheids-analyses	-databank oude projectgegevens -gevoeligheids-analyses	-expertmethode delfi-methode -risico-analyse -voorwaarde scheppend

Gezien de aard van de probleemsituatie - onzekerheid, wazigheid, weinig structurering - kan een specifiek systeem dat helpt bij de begroting/de beslissingsvoorbereiding een nuttige rol vervullen. Earl en Hopwood (1980) spreken in dit kader over een ideegenererend systeem.Omdat de onzekerheid groot is en als gevolg daarvan de onnauwkeurigheid van de begroting groot is, kan een begroting in deze situatie nooit een taakstellende funktie hebben. De betekenis van een begroting zal veel eerder **voorwaardenscheppend** zijn.

Als afronding van deze paragraaf worden in tabel 7.2 de vier beschreven beheerssituatie weergegeven.

7.6. EVOLUTIONAIR BEGROTEN

In de paragraaf 7.5 zijn vier ideaaltypische beheerssituaties beschreven. Hierbij is wellicht de indruk gewekt dat een specifiek ontwikkeltraject te typeren is met slechts één type situatie. Dit is doorgaans niet het geval. In de praktijk van software-ontwikkeling zal het erop neer komen dat de omstandigheden waaronder ontwikkeld wordt, in de loop van de tijd wijzigen. Produkt-, proces- en middelenzekerheid zijn zeker niet constant tijdens de ontwikkeling maar variëren. Bij aanvang van een softwareproject beschikt men doorgaans over een beperkte beschrijving van de probleemsituatie, een onvolledige verzameling van eisen en wensen, een gebrekkig inzicht in welke kostenbepalende factoren relevant zijn en welke van deze factoren wel en niet beïnvloedbaar zijn. Tijdens de ontwikkeling groeit het inzicht in het eindresultaat en krijgt men een steeds duidelijker en volledig beeld van de te ontwikkelen software. De gebruiker zal doorgaans steeds beter in staat zijn informatiebehoeften te formuleren. Het inzicht van de ontwikkelaar(s) in de probleemsituatie zal toenemen. Met andere woorden: de produktzekerheid neemt geleidelijk toe. Gaandeweg de ontwikkeling krijgt men een tevens een beter idee van de beheersmogelijkheden en van de effecten van bijstuurmaatregelingen. De projectresultaten worden steeds duidelijker en tastbaarder. Kortom, het inzicht in het project neemt toe bij iedere volgende fase van uitvoering; de proceszekerheid wordt steeds groter. In veel gevallen zal de hier geschetste situatie optreden. Het is evenwel ook mogelijk dat men een project start in de veronderstelling dat er weinig onzekerheden zijn. Gaande de ontwikkeling moet men constateren dat de mate van zekerheid te optimistisch is ingeschat. De informatiebehoeften blijken minder duidelijk dan oorspronkelijk werd

verondersteld en de wijze van aanpak blijkt eveneens minder duidelijk te zijn.
Een en ander betekent dat men zich bij de ontwikkeling van software voortdurend dient af te vragen welke beheerssituatie van kracht is. De ene beheerssituatie vraagt immers om een ander vorm van beheersen en begroten. In de meest ideale situatie zal er een geleidelijke verschuiving zijn van beheerssituatie 4 naar 1. Dit heeft ondermeer konsekwenties voor de wijze van begroten en voor de funktie van een begroting. Aanvankelijk zal de begroting voornamelijk een voorwaardenscheppende funktie hebben. Naarmate de uitvoering van het project vordert, stijgt het inzicht en komt er meer informatie beschikbaar om de begroting bij te stellen. De begroting zal daardoor steeds nauwkeuriger worden, dat wil zeggen dat het verschil tussen begroting en het uiteindelijk resultaat kleiner wordt. Daardoor neemt het taakstellend karakter van een begroting toe. De funktie van een begroting bij de beheersing zal om die reden afhankelijk zijn van de fase van uitvoering. Hoe nauwkeuriger de begroting, des te geschikter zal deze informatie zijn voor de beheersing van een project.

Na afloop van de fase informatieplanning zal op basis van een (globaal) projectplan een eerste kosten/baten analyse gemaakt worden voor het totale project. Deze eerste begroting dient een indicatie te geven of het voor de organisatie in kwestie economisch interessant is het project te starten. De funktie van een begroting is er een van een go-no/go beslissing. Een dergelijke begroting kan geen taakstellende funktie hebben en mag niet dienen als norm. Is het project haalbaar, dan dient de begroting verder voor een voorlopige allocatie van capaciteit. Pas op het moment dat de doelstellingen eenduidig zijn, dat wil zeggen als de opdrachtgever en ontwikkelaar hun goedkeuring hebben gegeven aan de lijst van eisen, is het mogelijk een begroting op te stellen, die kan dienen als instrument om een project te beheersen. Dit punt is veelal bereikt na afronding van het detailontwerp. De datastructuur, de mens-machine interfaces, de koppeling met andere systemen enz. moeten dan duidelijk zijn. De onzekerheden die een rol spelen bij het opstellen van een begroting behoren na die fase veel minder te zijn. Een begroting zal daarom nauwkeuriger zijn en kan gebruikt worden als instrument om het project te beheersen. Zolang niet aan deze voorwaarde is voldaan, mogen kostenramingen slechts een voorwaardenscheppende betekenis hebben.
Behalve dat er een kostenindicatie van het totale project wordt gemaakt, zal na afloop van de fase informatieplanning, afhankelijk van de gekozen ontwikkelstrategie, tevens een begroting worden opgesteld voor de fase definitiestudie of voor het volgende te ontwikkelen deelsysteem, increment. De begroting van een definitiestudie heeft voornamelijk een taakstellend karakter. In veel gevallen wordt

met de opdrachtgever een vast bedrag en doorlooptijd afgesproken; bijvoorbeeld een informatie-analist krijgt 20 dagen de tijd om een definitiestudie uit te voeren.
Maakt men gebruik van de incrementele ontwikkelstrategie dan zijn de eisen per stap precies omschreven en zal de begroting van zo'n stap als beheersinstrument dienen. In paragraaf 7.5.4 is een en ander toegelicht.

Het principe van meerdere malen begroten, van een toenemend inzicht in het produkt en de beheersmogelijkheden van het project en van een wijzigende begrotingsfunktie, noem ik evolutionair begroten.

7.7. CONCLUSIES

In dit hoofdstuk is een typologie van beheerssituaties voor software-ontwikkeling gepresenteerd. Hierbij is ondermeer gebruik gemaakt van het besturingsparadigma, beschreven in hoofdstuk 6, en van het P-B-I model. Dit model is gebruikt als analyse-instrument voor het proces van software-ontwikkeling en als hulpmiddel voor het ontwerpen van een beheersingsconcept voor software-ontwikkeling. Uitgangspunt van het P-B-I model is dat de karakteristieken van software-ontwikkeling (P) bepalend zijn voor de wijze van besturing. De wijze van besturing op zijn beurt determineert welke bestuurlijke informatie hiervoor noodzakelijk is. De P is beschreven aan de hand van een uitgebreid aantal kenmerken. Deze zijn uitgesplitst naar produkt-, proces- en middelenkenmerken. Omdat de zekerheid wat betreft deze kenmerken van ontwikkelsituatie tot ontwikkelsituatie kan verschillen, zal ook de wijze van beheersen en begroten variëren. Om hierin een ordening aan te brengen, is in dit hoofdstuk een typologie van beheersen en begroten ontwikkeld. Uitgangspunt hierbij is dat elk proces van software-ontwikkeling onder te brengen is in vier te onderscheiden ideaaltypen. Bij elk ideaaltype behoort een bepaalde aanpak van beheersing en begroting. De mate van produkt-, proces- en middelenzekerheid bepaalt tot welke ideaaltype een bepaalde software-ontwikkeling gerekend moet worden.

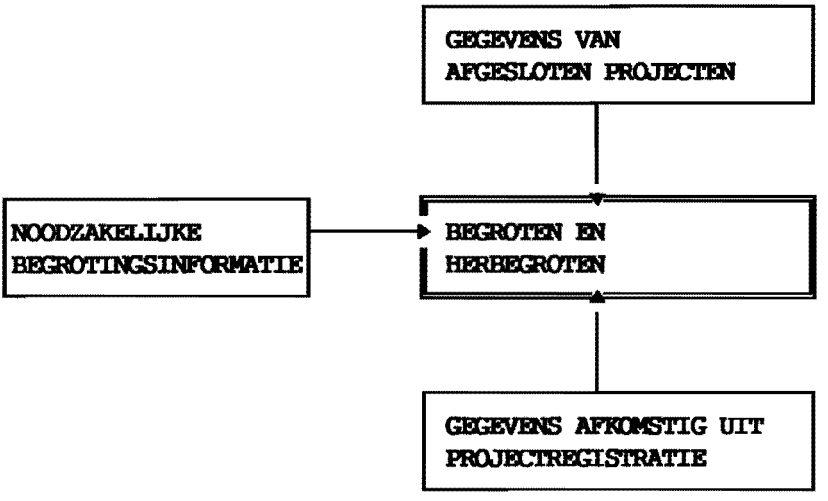
Tijdens de ontwikkeling van een programma verandert de mate van zekerheid geleidelijk. Aanvankelijk zullen de eisen waaraan de software moet voldoen vaag en onvolledig zijn. Men heeft weinig inzicht in de mogelijkheden om het proces te sturen, effecten van stuuracties zijn vaak niet goed te overzien. Naarmate de ontwikkeling van de software vordert nemen deze tekortkomingen langzaam maar zeker af. Men krijgt een steeds beter idee van het eindresul-

taat en is steeds beter in staat het ontwikkelproces te sturen. Een en ander betekent dat, vanuit beheers- en begrotingsoptiek, het ontwikkelen van software verschillende stadia doorloopt, die overeenkomen met de ideaaltypische beheerssituaties.
In hoofdstuk 8 wordt de aanpak volgens het P-B-I model doorgetrokken en wordt een nadere invulling gegeven aan de component "bestuurlijke informatie". Dit is alleen gedaan voor die beheerssituaties waarbij zekerheid bestaat over het eindresultaat.

8. EEN RAAMWERK VOOR HET BEGROTEN VAN SOFTWARE

8.1. INLEIDING

In de vorige hoofdstukken is meerdere malen gewezen op het belang van een zo juist en volledig mogelijke informatie bij het opstellen van een begroting. Er is ook op gewezen dat de juistheid en volledigheid van deze informatie afhankelijk is van de kenmerken van software-ontwikkeling en van de wijze van beheersing en begroting. Als het produkt dat ontwikkeld moet worden redelijk bekend is, dan vormt de lijst van eisen en wensen waaraan de programmatuur moet voldoen de belangrijkste bron van informatie voor een begroting. Anders ligt het als men geen duidelijk idee heeft van het te realiseren produkt. De noodzakelijke informatie heeft in een dergelijke situatie vooral betrekking op het budget, de tijd en de capaciteit die beschikbaar worden gesteld. Tussen deze twee uitersten zijn uiteraard allerlei combinaties mogelijk. Andere noodzakelijke informatie om te begroten en te herbegroten is afkomstig van ondermeer de voortgangsregistratie die tijdens de uitvoering van een project wordt gemaakt en de ervaring met reeds voltooide projecten. In figuur 8.1 is dit schematisch in beeld gebracht.
In hoofdstuk 7 is beargumenteerd dat duidelijke en volledige specificaties een belangrijke voorwaarde vormen voor het opstellen van een begroting, die kan dienen als beheersinstrument. Verder is in hoofdstuk 7 aangegeven dat begroten niet een eenmalige activiteit is. Het meten van de projectvoortgang kan verschillen tussen werkelijkheid en begroting/planning aantonen. Een gevolg hiervan kan zijn het bijstellen van de begroting (herbegroten). In de hoofdstukken 3 en 4 is meer dan eens gesignaleerd dat een begroting eveneens gebaseerd dient te zijn op gegevens van projecten, die in het verleden door de betreffende organisatie zijn gerealiseerd. Dit geldt ongeacht de methode die bij het opstellen van een begroting wordt gebruikt. Voor begrotingsmodellen is aangegeven wat de gevolgen zijn als aan deze voorwaarde niet wordt voldaan.



Figuur 8.1 : Noodzakelijke informatie om te begroten.

In de inleiding van hoofdstuk 7 heb ik aangegeven dat het P-B-I model het uitgangspunt vormt voor de behandeling van de hoofdstukken 7 en 8. In hoofdstuk 7 is de P, in casu het primaire proces van software-ontwikkeling beschreven en is nader ingegaan op de B, dat wil zeggen op de wijze van beheersen en begroten. In dit hoofdstuk zal het accent worden gelegd op de I, de bestuurlijke informatie, met dien verstande dat ik mij zal beperken tot die situaties waarin de produktzekerheid hoog is. Dit zijn de situaties die in hoofdstuk 7 aangeduid zijn met situatie 1, 2 en 3. Een en ander betekent dat het raamwerk dat in dit hoofdstuk wordt gepresenteerd uitsluitend bedoeld is voor dié situaties.

Bij het definiëren van de bestuurlijke informatie in dit hoofdstuk zal nader worden ingegaan op het verzamelen, vastleggen, analyseren en raadplegen (verder aangeduid als: het registreren) van inzichten, ervaringen en feiten met als doel het beheersen en begroten van software-ontwikkeling. Hierbij zal ik een onderscheid maken in:

- het registreren van projectgegevens met als primaire doelstelling deze te gebruiken bij het beheersen (waaronder herplannen en herbegroten) van het project in uitvoering,
- het registreren en aggregeren van projectgegevens, bedoeld voor het begroten van nieuwe projecten.

In dit hoofdstuk zal beschreven worden welke soort gegevens geregistreerd moeten worden en op welke wijze deze gebruikt kunnen worden bij het beheersen en begroten van software-ontwikkeling. Voor dat doel wordt in paragraaf 8.5 een raamwerk gepresenteerd voor het beheersen en begroten van software-ontwikkeling. De beperking die hierbij gemaakt wordt is dat het raamwerk van toepassing is op de begrotingssituaties 1, 2 en 3. In het vorige hoofdstuk is reeds beargumenteerd dat informatie verkregen via registratie niet bruikbaar is bij het opstellen van een politieke begroting. Voorafgaande aan het raamwerk wordt in paragraaf 8.2 aangegeven voor welke doeleinden gegevens van voltooide projecten aangewend kunnen worden. Ondanks de voordelen vindt er in de praktijk slechts op beperkte schaal registratie plaats. In paragraaf 8.3 wordt aangegeven wat hiervan de oorzaken zijn. In paragraaf 8.4 wordt een overzicht gegeven van het beperkt aantal databanken met gegevens over voltooide softwareprojecten. Na de presentatie van het raamwerk in paragraaf 8.5 wordt in paragraaf 8.6 welke soort gegevens in een dergelijke databank moeten worden opgenomen. Het hoofdstuk wordt afgesloten met conclusies (paragraaf 8.7).

8.2. HET DOEL VAN VOLTOOIDE PROJECTGEGEVENS

Het beschikken over historische projectgegevens wordt in de literatuur (Boehm 1981, DeMarco 1982, Silverman 1985, Baker 1985, van Vliet 1987, Heemstra 1987, Kitchenham en Smith 1985, Jeffery en Basili 1988) en in de praktijk van de automatisering steeds vaker genoemd als een zinvolle en zelfs noodzakelijke ondersteuning voor de beheersing van software-ontwikkeling. In deze paragraaf wordt aangegeven wat we ons moeten voorstellen bij deze ondersteuning en wordt gesteld dat dit soort projectgegevens kan dienen als hulpmiddel bij:

1. het calibreren van een begrotingsmodel,
2. het gebruik van begrotingsmethoden,
3. het beheersen van softwareprojecten,
4. het analyseren van software-ontwikkeling.

ad 1. *oude projectgegevens ten behoeve van het calibreren.*

De literatuur over begrotingsmodellen is eensluidend in haar oordeel dat calibratie van ongeacht welk model noodzakelijk is. In ondermeer

paragraaf 4.8 is gesteld dat begrotingsmodellen niet zo maar toegepast kunnen worden in andere omgevingen dan die waarin ze zijn ontwikkeld. De in paragraaf 4.9 genoemde grote afwijkingen tussen werkelijk bestede middelen en begrotingen, opgesteld met behulp van een model, zijn voor een belangrijk deel terug te voeren tot het achterwege blijven van calibratie. Dat het effect van calibratie op het begrotingsresultaat enorm is, wordt gesignaleerd door Miyazaki en Mori (1985). Zij hebben hierbij gebruik gemaakt van het model COCOMO en de gegevens van 33 voltooide automatiseringsprojecten. Deze verschilden van de COCOMO-projecten in zoverre dat ze over het algemeen ontwikkeld waren in een semi-detached omgeving, veelal geprogrammeerd waren in COBOL en gemiddeld genomen aanzienlijk groter waren in omvang dan de COCOMO-projecten. Evenals in de onderzoeken genoemd in paragraaf 4.9 wordt ook in dit onderzoek aangetoond dat bij het achterwege blijven van calibratie aanzienlijke miscalculaties van kosten en doorlooptijd worden gemaakt; gemiddeld een afwijking van 166% , slechts in 6% van de gevallen was die afwijking minder dan 20%. Op basis van deze resultaten passen Miyazaki en Mori het model COCOMO aan door rekening te houden met de specifieke eigenschappen van de omgeving waarin de 33 projecten tot stand zijn gekomen. Zij doen dit door ondermeer een aantal (in hun situatie) niet relevante cost drivers uit het model te elimineren. Bovendien stellen zij het model bij door de invloedswaarden van de verschillende cost drivers te veranderen. Het gevolg van deze calibratie is, dat de gemiddelde afwijking nog slechts circa 17% bedraagt.

Een soortgelijk onderzoek is uitgevoerd door Saalfrank, Schelle en Schnopp (1987). Zij beschrijven een procedure met de naam COKAL waarmee modellen van het type COCOMO gecalibreerd kunnen worden. In het onderzoek wijzen zij op het belang van calibratie. De nauwkeurigheid van de begrotingsresultaten wordt groter, naarmate het model COCOMO met behulp van de procedure COKAL wordt gecalibreerd met meer gegevens van oude projecten.

Heemstra, van Genuchten en Kusters (1988) signaleren in hun onderzoek eveneens de noodzaak van calibratie. Hiertoe laten zij 14 ervaren projectleiders een schatting maken van de benodigde inspanning voor de ontwikkeling van een geautomatiseerd informatiesysteem. Elke projectleider moet twee schattingen maken, een schatting met behulp van het begrotingsmodel Before You Leap en een schatting met behulp van het model Estimacs. In tabel 8.1 zijn de resultaten deze schattingen weergegeven.
Uit de resultaten van dit onderzoek komt naar voren dat de verschillen tussen werkelijk gebruikte en geschatte inspanning enorm is. Voor het pakket Before You Leap blijken de projectleiders gemiddeld genomen de inspanning meer dan 400% te ruim te schatten. Voor het pakket Estimacs ligt dit cijfer nog hoger, namelijk circa 750% Boven-

dien blijken de begrotingsresultaten van de individuele projectleiders ver uit elkaar te liggen. Bij beide modellen is er sprake van forse varianties.

Tabel 8.1 : Schattingsresultaten met behulp van de modellen Before You Leap en Estimacs (Heemstra, van Genuchten en Kusters 1988). Mm staat voor mensmaanden.

	Schatting van benodigde Inspanning	
Model	Gemiddeld	Variantie
Before You Leap	27.7 mm	14.0 mm
Estimacs	48.5 mm	13.9 mm
Werkelijk benodigde inspanning : 6.5 mm		

Opmerkelijk is verder het grote verschil tussen de twee gemiddelde schattingsresultaten.Hier zit bijna een factor twee verschil in, terwijl de varianties onderling nauwelijks afwijken. Niet alleen de grote verschillen tussen realisatie en schattingen wijzen op het belang van calibratie, maar vooral ook het feit dat de schattingsresulaten van twee verschillende modellen, door dezelfde projectleiders gebruikt voor het begroten van hetzelfde project, zover uiteen liggen.

De bovengenoemde onderzoeken geven aan dat ontwikkelomgevingen sterk kunnen verschillen en calibratie derhalve noodzakelijk is.

ad 2. ***Oude projectgegevens ten behoeve van het gebruik van begrotingsmethoden.***

Bij het opstellen van een begroting wordt gebruik gemaakt van gegevens van voltooide projecten. Bij de analogiemethode en begrotingsmodellen gebeurt dit expliciet. Dit geldt in mindere mate voor de expertmethode. Bij deze methode is de kennis van en ervaring met het ontwikkelen van software slechts aanwezig in de hoofden van

enkelen. Voor anderen die met soortgelijke problemen geconfronteerd worden, is het moeilijk zo niet onmogelijk deze niet overdraagbare kennis en ervaring te achterhalen. Dat bevat het risico in zich dat fouten worden herhaald. Een databank met gegevens over afgesloten projecten, waarin die kennis en ervaring expliciet worden gemaakt, kan het projectmanagement derhalve ondersteunen bij het schatten van de benodigde tijd, geld en middelen. Bij het gebruik van begrotingsmodellen kunnen oude projectgegevens het projectmanagement ondersteunen bij het bepalen van de invoer van die modellen. Over het algemeen vereist een dergelijke invoerbepaling veel ervaring in het gebruik van modellen en een ruime kennis en inzicht in het ontwikkelen van software. Zo zal de gebruiker van een model antwoord moeten geven op vragen als: hoeveel invoer- en uitvoerfuncties zijn er in het te ontwikkelen systeem te onderscheiden en hoeveel bestanden en inquiries, is de complexiteit hoog, gemiddeld of laag, hoe zwaar zijn de kwaliteitseisen, wat is het ervaringsniveau van de gebruiker op automatiseringsgebied en wat is de kwaliteit van de projectmedewerkers. Het merendeel van deze vragen is moeilijk te beantwoorden. Gegevens van voltooide projecten kunnen hierbij ondersteuning bieden. Op basis van de kenmerken van het nieuwe project, zoals type applicatie, aantal en soort klanten, te gebruiken programmeertaal e.d., is het mogelijk in een databank van oude projecten te zoeken naar projecten met soortgelijke kenmerken. Als van deze projecten de invoer voor het model is geregistreerd, dan kunnen deze gegevens een waardevolle ondersteuning vormen bij de invoerbepaling van het nieuwe projekt. Zijn bovendien evaluatie- c.q. validatiegegevens van de invoer in de vorm van expertise in de databank opgenomen, dan wordt het projectmanagement niet alleen gesteund door louter cijfermateriaal maar ook door ervaringsfeiten. Zo'n ervaringsfeit zou kunnen zijn dat bij soortgelijke projecten in de fase vooronderzoek in Y% van de gevallen de omvang met een factor X te hoog of te laag werd getaxeerd.

ad 3. ***Oude projectgegevens ten behoeve van projectbeheersing.***

Het is niet voldoende om alleen een begroting te maken van de hoeveelheid tijd, geld, personeel, hardware, software en overige middelen die in de verschillende fasen c.q. activiteiten ingezet moeten worden. Het ontwikkeltraject moet beheerst worden, dat wil zeggen er moet een plan en begroting worden gemaakt; er moet bewaakt worden of de uitvoering conform plan en begroting verloopt (monitoring) en er moet bij verschillen tussen plan en werkelijkheid (eventueel) worden bijgesteld. Oude projectgegevens kunnen hierbij een waardevol hulpmiddel zijn. Om dit te realiseren is het belangrijk dat van elk

projekt ondermeer wordt vastgelegd wat oorzaken zijn van verschillen tussen plan en werkelijkheid. Door van het analyseren van dit soort gegevens van oude projecten is het voor het projectmanagement mogelijk een antwoord te krijgen op vragen als welke zaken hebben in het verleden bij soortgelijke projecten als het nieuw op te starten project tot verstoringen in de ontwikkeling geleid, welke acties zijn vervolgens genomen en wat zijn de effecten hiervan geweest. Ook de resultaten van risico-analysegegevens van afgesloten projecten, zijn een waardevol hulpmiddel voor het projectmanagement. Het interpreteren van een risico-analyse krijgt meer betekenis als deze ondersteunt wordt door risico-analyse gegevens van soortgelijke projecten uit het verleden. Het projectmanagement neemt op verschillende meetpunten in een project beslissingen. De informatie waarop deze beslissingen gebaseerd zijn, is afkomstig van de projectuitvoering, van de projectdocumentatie, van normen en van doelstellingen. Bovendien zal het management impliciet gebruik maken van vuistregels en ervaring. Door het projectmanagement te ondersteunen met gegevens van oude projecten, bijvoorbeeld in de vorm van kengetallen, wordt deze ervaring en kennis expliciet gemaakt bij het beslissen op de meetpunten.

ad 4. ***Oude projectgegevens ten behoeve van analyses.***

Onder deze noemer kunnen een aantal toepassingen worden gerangschikt.

- Projectgegevens kunnen in de vorm van kengetallen gebruikt worden voor produktiviteitsanalyses (Hall, Gibbons en Knell 1986). Hierdoor is het mogelijk trends in de produktiviteit van een ontwikkelomgeving te analyseren. Ook kan men nagaan wat het effect op de produktiviteit is van het gebruik van vierde generatiehulpmiddelen, van organisatorische veranderingen, speciaal soort ontwikkelcomputers, team-samenstelling en andere produktiviteitsbeïnvloedende factoren.

- Door oude projectgegevens te analyseren is een organisatie in staat de volgende soort informatie te achterhalen. Men kan onderzoeken met welk type applicatie men de meeste ervaring heeft, in welke taal in hoofdzaak wordt geprogrammeerd, met welk type computers en operating systemen men veelal werkt, enz. Men is aldus in staat een overzicht te maken van zaken waarin men veel ervaring heeft c.q. sterk is en waarop, bijvoorbeeld, een gerichte benadering van de markt gebaseerd kan zijn. De databank die als basis diende voor het onderzoek van Walston en Felix (1977) was zo ingericht dat

dergelijk algemene vragen beantwoord konden worden. Dergelijk soort informatie kan tevens worden gebruikt om personeel aan projecten toe te wijzen. Zo kan de databank geraadpleegd worden welke ontwikkelaars de meeste ervaring hebben met het ontwikkelen van een bepaalde type software.

- Een niet te verwaarlozen resultaat van projectregistratie is het leereffect. Door alleen al impliciet aanwezige kennis, ervaring en feiten expliciet te maken is elk lid van een ontwikkelafdeling verplicht zich bewuster te verdiepen in een aantal inhoudelijke en beheersmatige zaken van software-ontwikkeling. Door het presenteren van statistische analyses op gegevens van afgesloten projecten wordt eveneens het inzicht in met name de projectmatige aspecten van software-ontwikkeling vergroot.

8.3. EEN GEBREK AAN OUDE PROJECTGEGEVENS. WAAROM?

Ondanks de vele voordelen, vindt er in de praktijk slechts op bescheiden schaal registratie van oude projectgegevens plaats. Uit de resultaten van de in hoofdstuk 2 behandelde enquete blijkt dat slechts 50% van automatiserend Nederland projectgegevens vastlegt. Hiervoor is een aantal redenen aan te geven.

1. De grote werkdruk,
2. Vereiste registratie sluit niet aan bij dagelijkse werkpatroon,
3. Een te grote nadruk op urenregistratie,
4. Een gebrek aan uniformiteit,
5. Geen inzicht welke gegevens geregistreerd moeten worden,
6. Een onterecht gebruik van "gekochte" projectgegevens,
7. Het feit dat software veelal projectmatig wordt ontwikkeld.

Deze redenen zullen kort worden toegelicht.

ad 1. ***grote werkdruk***

Programmatuur moet in veel organisaties onder hoge tijdsdruk worden ontwikkeld. De vraag naar nieuwe applicaties is veel groter dan met het beschikbare personeel gerealiseerd kan worden. Het gevolg hiervan is dat tal van organisaties kampen met een kloof tussen de vraag naar en aanbod van toepassingen. Deze kloof wordt snel groter als men de literatuur mag geloven. Door verschillende onderzoekers zoals

Martin en McClure (1983) en Rosenberger (1981) wordt gesignaleerd dat achterstanden wat betreft het ontwikkelen van nieuwe applicaties van meer dan drie jaar zijn geen uitzondering. Het gevolg van dit alles is dat de werkdruk op ontwikkelaars groot is. Ruimte voor het registreren van projectgegevens wordt daardoor niet of nauwelijks genomen.

ad 2. *ander werkpatroon*

Veelal strookt het niet met de geaardheid van de ontwikkelaar om nauwkeurig en in detail zijn activiteiten op papier vast te leggen. Een veel gehoorde argumentatie onder analisten, ontwerpers en programmeurs in deze is dat zij hun kostbare tijd beter kunnen besteden aan het bouwen van software dan aan het invullen van de meest uiteenlopende formulieren ten behoeve van projectdocumentatie (Dawes 1985). Vaak wordt een dergelijk registratiesysteem door de betrokkenen gezien als een controle-instrument voor alleen het projectmanagement waarvan zij eerder nadeel dan voordeel zullen ondervinden. Bestaat binnen een organisatie een dergelijke houding dan is succesvolle registratie bij voorbaat uitgesloten. De direct betrokkenen moeten overtuigd zijn van de noodzaak van een dergelijk systeem en van het profijt dat zo'n systeem voor hen zelf betekent. Bemelmans (1987) spreekt in dit verband van gesloten lusautomatisering. Pas als aan deze voorwaarde is voldaan, kan men rekenen op medewerking en loopt men niet het gevaar dat het systeem wordt geboycot door het aanleveren van onbetrouwbare en onvolledige gegevens. Een en ander betekent dat een mentaliteitsverandering nodig is en dat de noodzaak van een goede begroting van kosten, inspanning en doorlooptijd en de rol van oude projectgegevens daarin, wordt ingezien.

ad 3. *accent op urenregistratie*

Het registreren van projectgegevens vindt momenteel hoofdzakelijk plaats vanuit financieel oogpunt; namelijk het doorberekenen van ontwikkelkosten. Deze gegevens hebben onder meer betrekking op de totale kosten aan programmeren, analyse en ontwerp, reis- en verblijfkosten, apparatuurkosten en dergelijke. Voor dit doel worden geen gegevens vastgelegd zoals de mate van gebruikersparticipatie, de vereiste kwaliteit en complexiteit van het systeem, het ervaringsniveau van de ontwikkelaars enz. Voor het begroten van softwareprojecten zijn deze gegevens echter wel nodig.

ad 4. *gebrek aan uniformiteit gegevens*

Zijn er al gegevens beschikbaar dan blijken deze over meerdere

afdelingen en ontwikkelteams verspreid te liggen, waarbij er veelal weinig sprake is van uniformiteit. Zo worden niet altijd dezelfde gegevens verzameld, wordt hetzelfde begrip door verschillende personen verschillend gedefinieerd, zijn er verschillen in de wijze waarop gegevens worden verzameld en vastgelegd enz. Wil men komen tot een omvangrijke verzameling consistente projectgegevens, die als basis moeten dienen voor begrotingsmethoden dan zijn afspraken en richtlijnen in deze een voorwaarde.

ad 5. ***geen inzicht in welke gegevens nodig zijn***

Het is vooralsnog niet duidelijk welke gegevens precies geregistreerd moeten worden. Wat opvalt in de bestaande projectdatabanken is dat in de ene databank een ander soort gegevens wordt geregistreerd dan in een andere. Voor een deel zijn deze verschillen te verklaren uit de verscheidenheid van ontwikkelomgevingen die als basis hebben gediend bij de opbouw van de databanken. Bij de goedwillende projectmanager schept dit onduidelijkheid en verwarring. Op het antwoord van de vraag - welke gegevens zijn relevant om geregistreerd te worden - zal hij vanuit de literatuur geen bevredigend antwoord krijgen.

ad 6. ***achterwege laten van calibratie***

Het belang van een goede prognose van kosten, inspanning en doorlooptijd wordt algemeen geaccepteerd. Dat een eigen registratiesysteem van oude projecten hiervoor noodzakelijk is, wordt nog niet alom erkend. Met hoge verwachtingen worden vaak begrotingsmodellen in huis gehaald. Het gemis aan eigen oude projectgegevens wordt, naar men dan verwacht, gecompenseerd door de uitgebreide databank die de basis vormt van het model. In de meeste gevallen worden de begrotingen volgens deze aanpak eerder slechter dan beter; het model is immers niet gecalibreerd. Het model krijgt dan ten onrechte de schuld.

ad 7. ***negatieve effecten van een projectaanpak***

Het feit dat de ontwikkeling van software plaatsvindt in een project organisatievorm, bevordert niet een adequate registratie van projecten. Hiervoor kunnen een aantal verklaringen worden gegeven. De samenstelling van een ontwikkelteam is vaak van project tot project verschillend. Men werkt niet steeds met dezelfde bemanning. Bovendien verandert de samenstelling van het projectteam gedurende de looptijd en wordt het team aan het einde van het project ontbonden. In een dergelijke organisatievorm met een wisselende samenstelling en een eenmalige karakter is het moeilijker een registratie te realiseren

dan bij een vast team en een procesmatige aanpak. Men kan zich niet veroorloven met het vastleggen van projectgegevens te wachten tot aan het einde van het project. Een groot gedeelte van het oorspronkelijke team is reeds werkzaam in andere projecten en voelt er weinig voor om alsnog een stuk registratie te verzorgen. Het is moeilijk in een dergelijke omgeving die zich bovendien kenmerkt door werken onder hoge tijdsdruk, een hoge mate van specialisatie en een grote verscheidenheid van typen projecten een gestructureerde verzameling relevante projectgegevens te realiseren.

8.4. DATABANKEN MET GEGEVENS OVER VOLTOOIDE SOFTWARE-PROJECTEN

De voordelen die behaald kunnen worden met een databank van voltooide projecten zijn evident. Ondanks deze voordelen is er een groot gebrek aan dit soort gegevensverzamelingen. In de literatuur over begroten van software worden slechts een beperkt aantal databanken in dit kader genoemd. Veelal is de inhoud dusdanig vertrouwelijk dat uitgebreide informatie niet voorhanden is. Een (niet uitputtend) overzicht van de literatuur laat de volgende databanken zien:

- Wolverton (1974) beschrijft in algemene termen een databank waarin projecten zijn geregistreerd die door TRW Systems Inc. ten behoeve van eigen gebruik zijn ontwikkeld.

- Walston en Felix (1977) hebben in de jaren zeventig een omvangrijke databank van oude projecten opgebouwd. Alle projecten zijn uitgevoerd bij IBM Federal Systems (IBM-FSD) en zijn zeer uiteenlopend van aard. Helaas is de opbouw en inhoud van deze databank nooit openbaar gemaakt.

- De meest bekende en meest toegankelijke databank is de al in hoofdstuk 3 genoemde COCOMO databank. Hierin zijn 63 projecten opgenomen die tussen de jaren 1964 en 1979 werden uitgevoerd door TRW Systems Inc. De projecten verschillen onderling sterk. Wat betreft omvang lopen ze uiteen van 2.000 tot 1.000.000 regels code. De programmeertalen zijn ondermeer assembler, COBOL, FORTRAN en PL/1. Een breed spectrum van soorten applicaties wordt afgedekt: administratieve en wetenschappelijke toepassingen, maar ook applicaties voor onder andere procescontrole. Juist die grote verscheidenheid maakt de toepassingsmogelijkheden van deze

databank voor een schattingsmodel als COCOMO twijfelachtig, ondanks het grote aantal projecten dat geïnventariseerd is. Gesteld mag worden dat de databank te klein is en teveel uiteenlopende projecten beschrijft om betrouwbare en algemene uitspraken te kunnen doen over de invloed van de 15 COCOMO cost drivers op kosten, inspanning en doorlooptijd. Zoals al eerder aangegeven, is het aan te bevelen het aantal cost drivers beperkt te houden tot uitsluitend de meest relevante en in de databank de gegevens van een homogenere groep projecten vast te leggen.

- Het Rome Air Development Center beschikt over een uitgebreide databank van voltooide projecten uitgevoerd bij het Amerikaanse ministerie van defensie. Een aantal modellen is hiermee getoetst. Zo beweert Putnam (1978) dat zijn model voor deze projecten goede begrotingen afgeeft. Helaas is deze databank voor buitenstaanders niet toegankelijk.

- Baily en Basili (1981) geven een beperkt overzicht van de gegevens van 18 NASA projecten die zijn opgenomen in een databank. Een veel uitgebreidere databank is de SDC (Systems Development Corporation) databank. Hierin zijn de gegevens van 169 projecten vastgelegd. Alle projecten zijn echter uitgevoerd vóór 1966 en kunnen derhalve moeilijk representatief geacht worden voor 1988.

- Een groot aantal projecten uitgevoerd voor het ministerie van defensie in Groot-Brittannië is vastgelegd in een databank (Stanley 1984). Hierin wordt echter geen duidelijk onderscheid gemaakt in software- en hardwareprojecten.

- Bij de ontwikkeling van het PRICE model (Freeman 1979) is gebruik gemaakt van gegevens van honderden projecten. De algoritmen en veronderstellingen waarop het model is gebaseerd, zijn niet gepubliceerd. Hetzelfde geldt voor de gegevens van de voltooide projecten.

De laatste jaren wordt er op beperkte schaal onderzoek uitgevoerd dat zich expliciet richt op het registreren van gegevens van voltooide softwareprojecten. Het centrale thema in al deze onderzoeken is de inhoud van een projectendatabank; met ander woorden: welke gegevens dienen in een dergelijke databank te worden opgenomen.

- In Nederland hebben Dekker en van den Bosch (1983) getracht een databank van voltooide projectgegevens op te bouwen. Volgens hen is het mogelijk een softwareproject met behulp van 47 variabelen te

karakteriseren. Zij maken daarbij, geïnspireerd door het model COCOMO, onderscheid in acht categorieën.
Variabelen die betrekking hebben op:
* de projectomvang,
* de moeilijkheidsgraad van het project,
* de vereiste betrouwbaarheid van de te ontwikkelen software,
* beperkingen wat betreft geheugen en doorlooptijd,
* mate van hergebruik,
* typering van de ontwikkelomgeving,
* typering van het toepassingsgebied,
* typering van de te gebruiken middelen.

Van elke variabele geven zij aan na welke fase(n) registratie ervan moet plaatsvinden. Bijvoorbeeld: ervaring met randapparatuur moet vastgelegd worden na de fasen ontwerp en implementatie; het aantal statements na de fase testen. Het onderzoek gaat niet verder dan een opsomming van relevante variabelen en geeft niet aan hoe deze variabelen gedefinieerd, gekwantificeerd, verzameld en geregistreerd moeten worden. Tot een daadwerkelijke databank van voltooide projecten is het nooit gekomen.

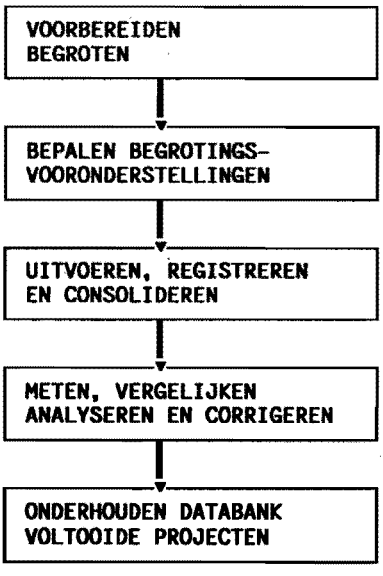
- De Purdue Universiteit van West Lafayette in Indiana (V.S.) heeft een zeer uitgebreide databank van voltooide softwareprojecten opgebouwd (Yu, Nejmek, Dunsmore en Shen 1988). De gegevens zijn afkomstig van een zeer groot aantal verschillende projecten. Zo zijn er projecten beschreven die uitgevoerd zijn in opdracht van het Amerikaanse Ministerie van Defensie, zijn de eerder genoemde COCOMO-gegevens opgenomen, heeft DeMarco een groot aantal projectgegevens aangeleverd, evenals Baily en Basili. Het belangrijke voordeel van deze databank is dat zij ter beschikking staat voor externe gebruikers en dat er faciliteiten worden geboden om statistische manipulaties met de gegevens uit te voeren. Een nadeel is dat de projecten niet alle op dezelfde wijze beschreven zijn; verschillende definities worden gehanteerd en verschillende maatstaven toegepast.

- Het onderzoek van Kitchenham en Smith (1985), uitgevoerd als onderdeel van het Alvey Software Library project, concentreert zich niet uitsluitend op een inventarisatie van projectgegevens die verzameld moeten worden, maar levert ook zogenaamde projectregistratie-formulieren op. Elk project wordt na afronding door het beantwoorden van een uitgebreide lijst van vragen op dit formulier beschreven. Doordat de meeste vragen een gesloten karakter hebben, is er impliciet sprake van een (beperkte) definiëring en kwalificering. Bij deze vragen wordt onderscheid gemaakt in vragen die betrekking hebben op het registreren van projectgegevens, produkt-

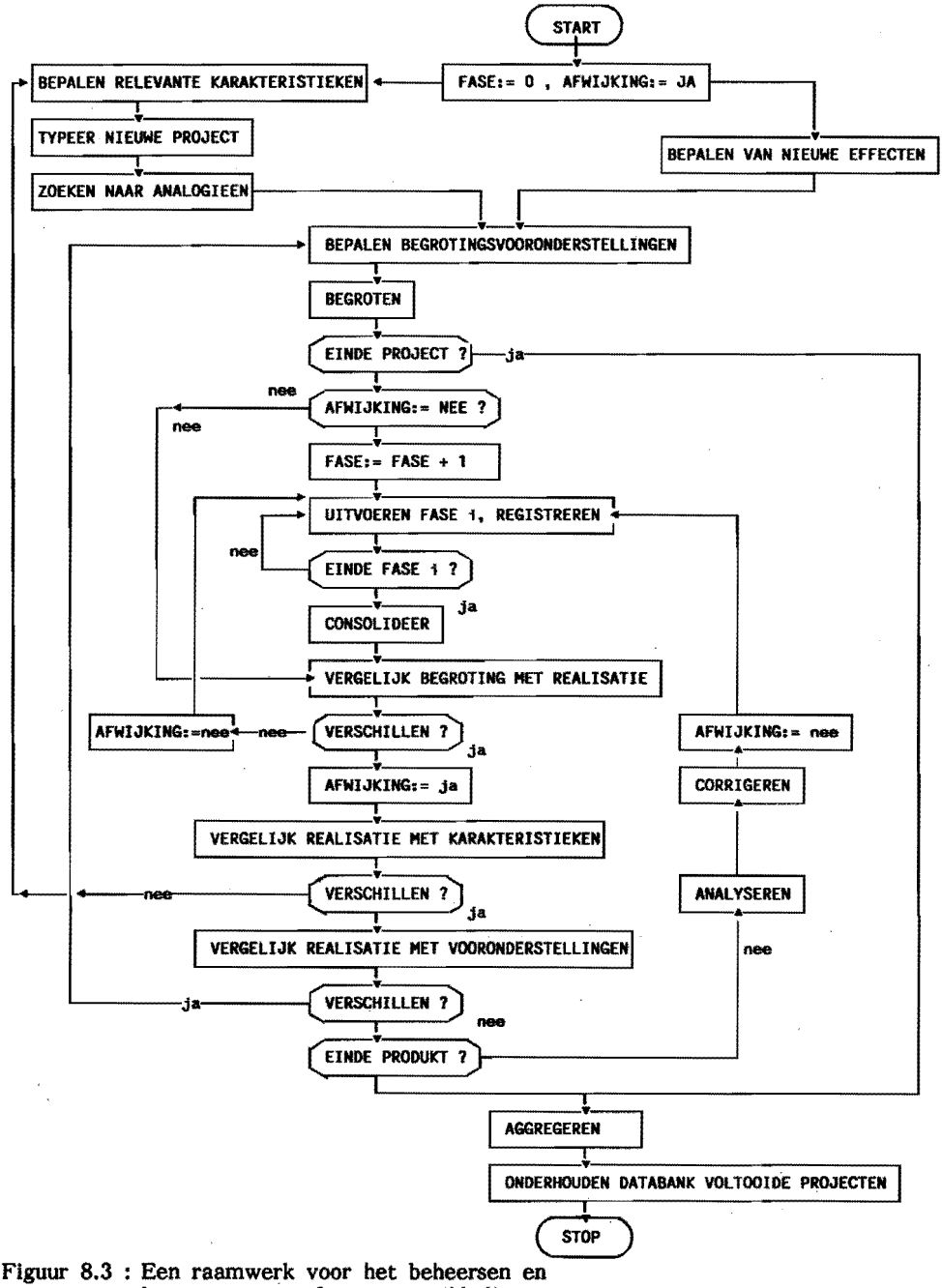
gegevens en procesgegevens. Projectgegevens hebben primair betrekking op de omgeving waarin het produkt is ontwikkeld. Produktgegevens beschrijven de software in termen van bijvoorbeeld complexiteit, omvang en kwaliteit. Procesgegevens geven aan met behulp van welke methoden en technieken het produkt ontwikkeld is. Hoewel het onderzoek niet aangeeft wat verstaan moet worden onder bijvoorbeeld erg hoge complexiteit of lage betrokkenheid van de gebruiker, geeft het een goede aanzet om te komen tot een registratief systeem van voltooide projecten.

8.5. EEN RAAMWERK VOOR HET BEHEERSEN EN BEGROTEN VAN SOFTWAREPROJECTEN.

In hoofdstuk 6 is op basis van het besturingsparadigma een beschrijving gegeven van het primaire proces van software-ontwikkeling en van mogelijke besturingsvormen. In deze paragraaf wordt een nadere invulling gegeven van het Bestuurlijke informatiesysteem. Het accent zal hierbij worden gelegd op het verzamelen, vastleggen,



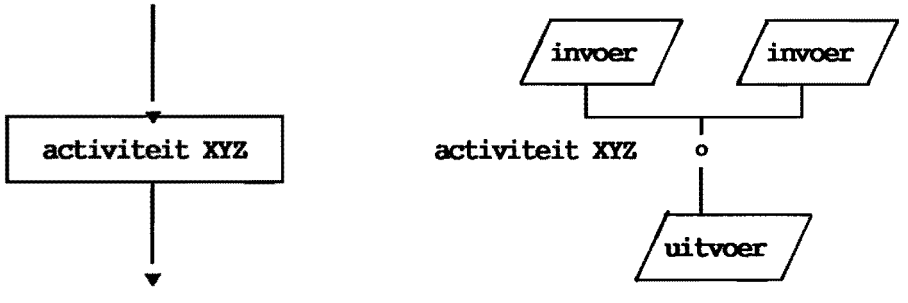
Figuur 8.2: De basisstappen in het conceptueel raamwerk.



Figuur 8.3 : Een raamwerk voor het beheersen en begroten van software-ontwikkeling.

analyseren en gebruiken van gegevens van voltooide softwareprojecten ten behoeve van het opstellen van een begroting die kan dienen als beheersinstrument. Een en ander zal worden uitgewerkt in de vorm van een raamwerk voor het beheersen en begroten van softwareprojecten. Dit raamwerk is gebaseerd op twee belangrijke uitgangspunten, namelijk op ***het registreren van projectgegevens*** en op het principe van ***evolutionair begroten.*** Dit raamwerk kan gebruikt worden als een referentiemodel voor het beheersen en in het bijzonder begroten van software-projecten. In figuur 8.2 worden de basisstappen van het raamwerk gegeven. Een verdere detaillering hiervan is gegeven in figuur 8.3 in de vorm van een stroomschema gegeven Hierin zijn alle relevante beheersactiviteiten opgenomen, alsmede de onderlinge relaties en de belangrijkste beslissingspunten.

In het stroomschema van figuur 8.3 komt duidelijk het iteratief karakter van begroten naar voren. Met behulp van de teller "fase" wordt bijgehouden in welke fase c.q. activiteit het project zich bevindt.Overgang naar een volgende fase (ophogen van de teller) is pas mogelijk als er geen afwijkingen zijn tussen realisatie en begroting of als de afwijkingen binnen afgesproken acceptabele grenzen liggen. Met behulp van de variabele "afwijking" wordt aangegeven of er wel (afwijking=ja) of geen (afwijking=nee) verschil is. In de subparagrafen 8.5.1. tot en met 8.5.10 worden de activiteiten uit het stroomschema nader toegelicht. Bovendien zal per activiteit worden aangegeven welke soort invoergegevens noodzakelijk zijn en welke soort informatie geproduceerd wordt. Hierbij is gebruik gemaakt van de techniek voor het weergeven van activiteitenschema's uit de ISAC methode (Lundeberg, Goldkuhl en Nilsson 1982). Voor de duidelijkheid is in figuur 8.4 de samenhang tussen de beide beschrijvingswijzen weergegeven.



Figuur 8.4 : Samenhang tussen een activiteit in het stroomschema en een activiteit in de precedentie analyse.

8.5.1. BEPALEN VAN RELEVANTE KARAKTERISTIEKEN

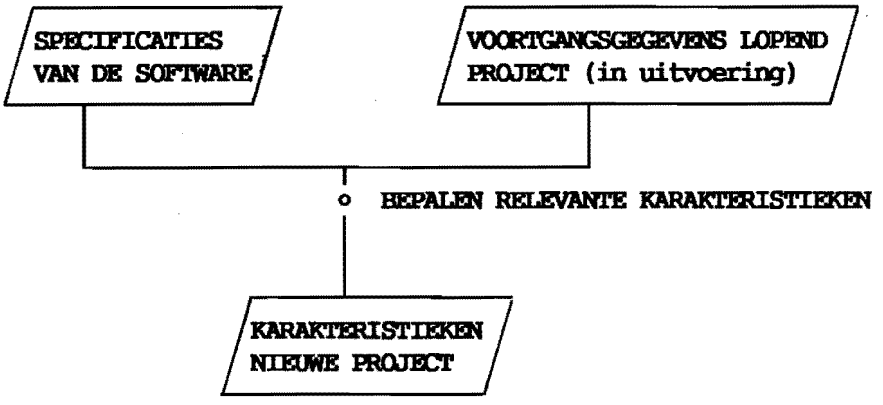
Tot de karakteristieken van een project behoort de lijst van eisen waaraan de software moet voldoen, zowel wat betreft functionaliteit als prestatieniveau. De karakteristieken van een nieuw project zijn nodig voor verschillende doeleinden.

- Een dergelijke lijst is noodzakelijk om een zo volledig en formeel mogelijke beschrijving van het produkt te krijgen die gedurende de rest van het project als doelstelling c.q. referentiemateriaal kan dienen.

- Een tweede doel van een dergelijke lijst is het dienen als basis voor het begroten van het project. In hoofdstuk 4 is aangegeven op welke wijze zo'n lijst van een softwareproject gebruikt kan worden bij het begroten. Tevens is naar voren gekomen dat bij het gebruik van begrotingsmodellen de lijst telkenmale zal verschillen afhankelijk van het model dat gebruikt wordt.

- Om een project te kunnen typeren overeenkomstig een typologie van projecten, is het van belang te beschikken over relevante gegevens van een project om snel en doeltreffend te kunnen zoeken naar analoge projecten. Jones (1986) merkt overigens terecht op dat er een gebrek is aan dergelijke typologieën. Verder onderzoek in deze richting is zeker gewenst.

Het voorgaande moge duidelijk maken dat een lijst van eisen van een project noodzakelijk is alsmede een lijst van kostenbepalende factoren, waarvan men inschat dat ze voor het betreffende project van belang zijn. Naarmate de doelstellingen van het project duidelijker zijn, is men beter in staat het project te typeren. Tijdens de uitvoering van een project kan het nodig zijn deze typering aan te passen. Gaande het project wijzigen de informatiebehoeften van de gebruikers en moeten de specificaties aangepast worden. Ook kan het gebeuren dat het gewenste projectresultaat, gezien de budgettaire beperkingen, niet haalbaar blijkt te zijn en de specificaties bijgesteld moeten worden. Veelal hebben dit soort aanpassingen directe gevolgen voor de typering van het project. In figuur 8.5 is, resumerend, aangegeven wat de vereiste invoer en de uitvoer is van de activiteit "bepalen relevante karakteristieken".



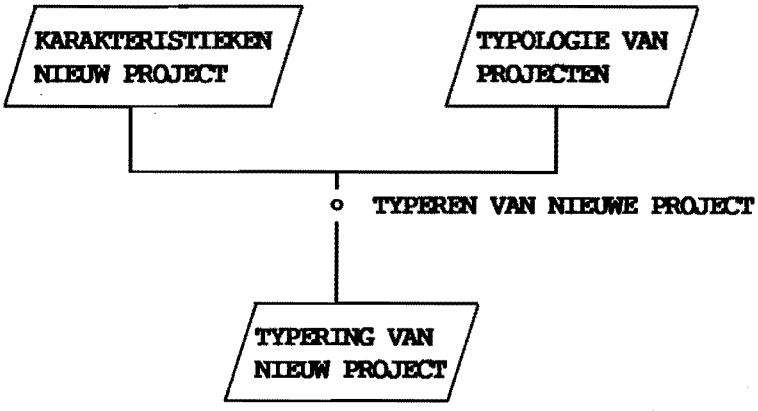
Figuur 8.5 : Bepalen relevante karakteristieken.

8.5.2. TYPEREN VAN HET NIEUWE PROJECT

Uit het raamwerk (figuur 8.3) valt af te leiden dat het typeren van een nieuw project de activiteit is die volgt op het bepalen van de relevante karakteristieken. Voor het typeren van een nieuw project moet men beschikken over (zie ook figuur 8.6):

- een karakterisering van het nieuwe project en
- een typologie van softwareprojecten. Kenmerkende eigenschappen van softwareprojecten kunnen bijvoorbeeld zijn: opdrachtgever is nauwelijks vertrouwd met automatisering, dezelfde applicatie moet door veel en verschillende gebruikers worden gebruikt, specificaties wijzigen vaak, hoge eisen aan documentatie enz.

De mate van verfijning en de relevantie van een dergelijke typologie wordt in belangrijke mate bepaald door het aantal voltooide projecten dat als basis dient voor deze typologie. Naarmate het aantal afgesloten projecten toeneemt, is men beter in staat de verschillende typen nauwkeurig en eenduidig te omschrijven.
Zoals uit figuur 8.6 valt af te leiden zijn relevante karakteristieken van een nieuw project een van de noodzakelijke invoergegevens van de activiteit "typeren". Er is reeds opgemerkt dat deze karakteristieken veelal niet stabiel zijn gedurende de uitvoering van het project.

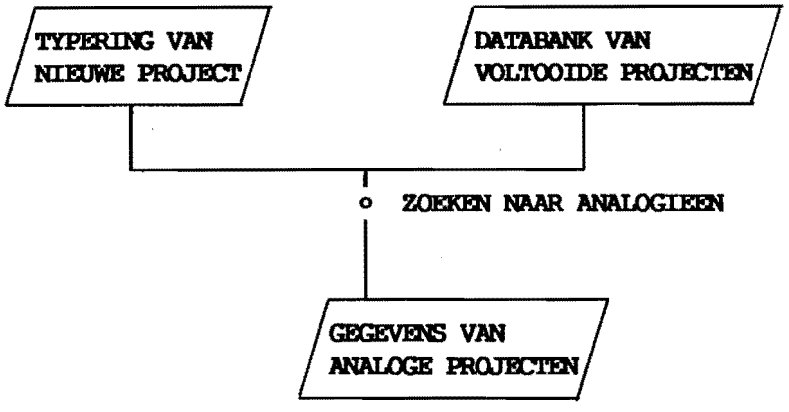


Figuur 8.6 : Typeren van het nieuwe project.

Het gevolg hiervan is dat de activiteit "typeren" meerdere malen herhaald moet worden. Een dergelijke constatering geldt uiteraard voor het gehele raamwerk. Elke keer als de invoerwaarde van een activiteit fors verandert, zal deze activiteit opnieuw uitgevoerd moeten worden. Dit onderstreept nogmaals dat een iteratieve aanpak bij begroten noodzakelijk is. Vooral bij aanvang van het project zal bij het iteratief begroten het typeren en het bijstellen van die typering een belangrijk onderdeel vormen. Het is voor een organisatie verstandig een eigen typologie van projecten te maken en niet zonder meer te vertrouwen op elders ontwikkelde typologieën.Dit geldt in het bijzonder in die situaties waarin een ontwikkelafdeling gespecialiseerd is in een beperkt gebied van de automatisering. Voor het maken van een eigen typologie zal een dergelijke afdeling een eigen databank van voltooide projecten moeten opbouwen. Deze databank kan eventueel aangevuld worden met gegevens van externe, bestaande typologieën. Gecontroleerd moet daarbij worden of deze bestaande typologieën aansluiten bij de eigen soort projecten en de eigen wijze van informatiesysteemontwikkeling. Vanzelfsprekend is het noodzakelijk om periodiek de validiteit van de typologie te controleren en een typologie zo nodig aan te passen.

8.5.3. ZOEKEN NAAR ANALOGIEEN.

De volgende activiteit in het conceptueel raamwerk is het zoeken in een databank van voltooide projecten naar analoge, vergelijkbare projecten op basis van de gemaakte typering van het nieuwe project (figuur 8.7).



Figuur 8.7 : Zoeken naar analogieën.

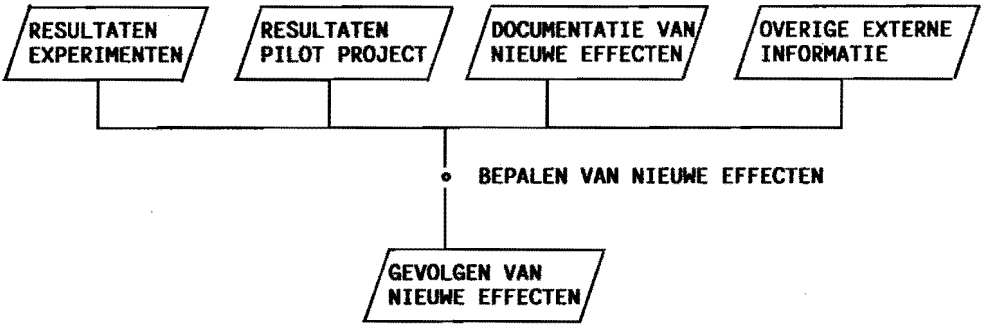
In deze fase van het opstellen van een projectbegroting willen we nagaan welke delen overeenkomsten hebben met eerder uitgevoerde projecten en welke delen van het project nieuw zijn c.q. geen analogieën hebben in de database. Bij het schatten van kosten en doorlooptijd is het aldus mogelijk kennis, feiten en ervaringen betreffende analoge delen van afgeronde projecten te gebruiken. Voor dergelijke gevallen is begroten op basis van analogieën een geschikte methode ."Echte schattingen" zijn nu alleen nog maar noodzakelijk voor nieuwe projectdelen (van Genuchten en van Gils 1988). Veronderstel, bijvoorbeeld, dat het nieuwe project betrekking heeft op de ontwikkeling van een administratieve toepassing op financieel terrein. Een verdere specificatie is dat de opdrachtgever een onderwijsorganisatie is. Met behulp van deze (weliswaar beperkte) specificaties zou het nu mogelijk moeten zijn het project te typeren en in een databank van voltooide projecten te zoeken naar analogieën.

Feiten en ervaringen van vergelijkbare projecten zijn dan beschikbaar en kunnen gebruikt worden bij het begroten van het nieuwe project. Zoeken naar analogieën is pas mogelijk als de lijst van eisen waaraan de software moet voldoen voldoende aanknopingspunten geeft om het project te karakteriseren. Een en ander betekent dat de analogiebenadering niet van toepassing is op volledige nieuwe, unieke projecten. Ditzelfde geldt voor projecten waarvan de doelstellingen vaag en onvolledig zijn.

8.5.4. BEPALEN VAN NIEUWE EFFECTEN

Voorbeelden van nieuwe effecten bij het ontwikkelen van software zijn het gebruik van vierde en vijfde generatie hulpmiddelen, prototyping, end user computing en workbenches. Als er binnen de betreffende organisatie weinig of geen ervaring bestaat met dergelijke hulpmiddelen en methoden, dan kan men weinig terugvallen op gegevens van voltooide projecten. Men heeft dus ook geen inzicht in het effect van bepaalde stuurmaatregelen, in casu het inzetten van nieuwe methoden, technieken en hulpmiddelen. Ondanks het gebrek aan ervaringsgegevens dient men toch het effect van deze nieuwe ontwikkelingen in te schatten. Dit levert uiteraard tal van problemen op. Zo is men voor het achterhalen van de gevolgen veelal aangewezen op de documentatie van deze nieuwe hulpmiddelen, op toezeggingen van leveranciers, wetenschappelijke onderzoeks-resultaten of ervaringen van andere, externe ontwikkelaars (in figuur 8.8: overige externe informatie).

Of dezelfde gevolgen ook van toepassing zijn voor de eigen organisatie is nog maar de vraag. Daarbij komt dat nieuwe ontwikkelingen, met name in de vorm van meer geavanceerde tools, elkaar in een steeds hoger tempo opvolgen. Ook de voortschrijdende ontwikkelingen op hardware gebied hebben een niet te onderschatten gevolg op de ontwikkelkosten. De ontwikkelorganisatie zal bij het gebruik van nieuwe methoden, technieken en gereedschappen hiermee moeten experimenteren en in een pilot project het effect op kosten, inspanning en doorlooptijd moeten bepalen. Bovendien zal men, zolang er nog geen zekerheid is over de gevolgen, voldoende speling in plannen en begrotingen moeten inbouwen. De invoer voor de activiteit "bepalen van nieuwe effecten" is als gevolg van bovenstaande minder geformaliseerd als bij de andere activiteiten uit het raamwerk (figuur 8.8).



Figuur 8.8 : Bepalen van nieuwe effecten.

8.5.5. BEPALEN VAN BEGROTINGSVOORONDERSTELLINGEN.

Op basis van de voorgaande vier activiteiten is het nu mogelijk een overzicht te maken van alle vooronderstellingen die tot nu toe zijn gemaakt en die als basis zullen dienen voor de activiteit "begroten". Alle resultaten van de vorige activiteiten worden samengevoegd om te fungeren als een expliciet geformuleerd overzicht van aannames, uitgangspunten en afspraken over de waarden van de factoren die van invloed zijn op kosten en ontwikkeltijd. Dit betekent dat, conform de indeling in hoofdstuk 5, wat vastgelegd wordt over zaken als:

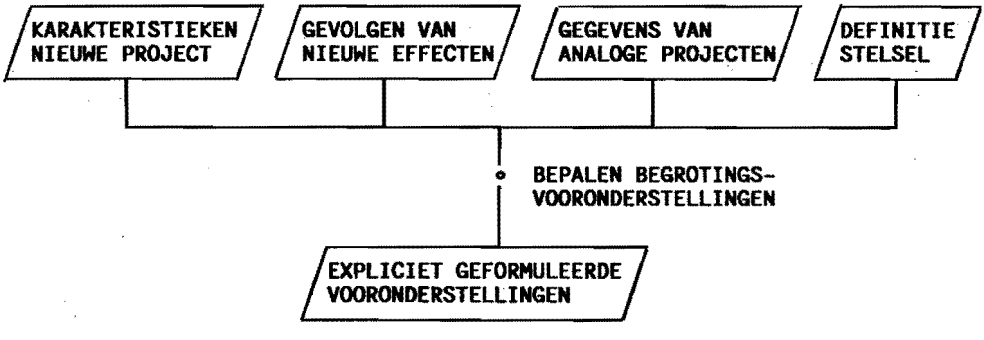
- het projectresultaat
- de middelen die worden gebruikt
- het personeel dat het project uitvoert
- de wijze waarop het project wordt uitgevoerd en
- de gebruikersorganisatie waarvoor het project wordt uitgevoerd.

Bovendien moeten zaken worden vastgelegd als het veranderingsklimaat en de mate van acceptatie door gebruikers, de bekendheid van gebruikers met automatisering, de soort organisatie, enz.

Voor een bepaald project kunnen voorbeelden van dergelijke afspraken en aannames zijn: de omvang wordt aangegeven in aantal regels code (exclusief commentaar- en blancoregels); ervaring wordt gedefinieerd als aantal jaren werkzaam in de automatisering; veel ervaring komt overeen met meer dan 10 jaar werkzaam in de automa-

tisering op het niveau van ontwerper; documentatie is inclusief gebruikershandleidingen; in de begroting hoeft geen rekening gehouden te worden met opleiding; er wordt niet gebruik gemaakt van nieuwe technieken enz.

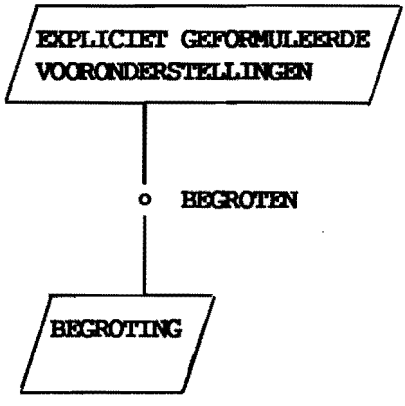
Daarnaast moeten afspraken gemaakt worden over een definitiestelsel voor begrotingsresultaten. Voorbeelden hiervan zijn: inspanning wordt weergegeven in mensweken, waarvoor geldt dat een week bestaat uit 38 uren; definiëring van begrippen als doorlooptijd, ontwikkeltijd; aantal malen dat tijdens een project wordt begroot en momenten waarop dit gebeurt; definiëring van het begrip kosten enz. Door al deze vooronderstellingen expliciet te formuleren ontstaat er een consistente basis voor de activiteit "begroten". Bovendien wordt het eenvoudiger om na te gaan of tijdens de uitvoering van het project afgeweken wordt van deze vooronderstellingen. Hierdoor ontstaat de mogelijkheid gerichter actie te ondernemen door hetzij de begroting aan te passen, hetzij door bij te sturen in de projectuitvoering of door het projectresultaat wat betreft functionaliteit en/of prestatieniveau bij te stellen.De grotere gerichtheid bij het bijsturen is het gevolg van de beschikbaarheid van een norm in de vorm van de vooronderstellingen. Het formuleren en werken met een verzameling vooronderstellingen heeft dus directe gevolgen voor de mate van besturingsvariëteit. De invoer en uitvoer voor de activiteit "bepalen vooronderstellingen" is in figuur 8.9 in beeld gebracht.



Figuur 8.9 : Bepalen begrotingsvooronderstellingen.

8.5.6. BEGROTEN

Elke begroting dient gebaseerd te zijn op expliciet geformuleerde vooronderstellingen (figuur 8.10).



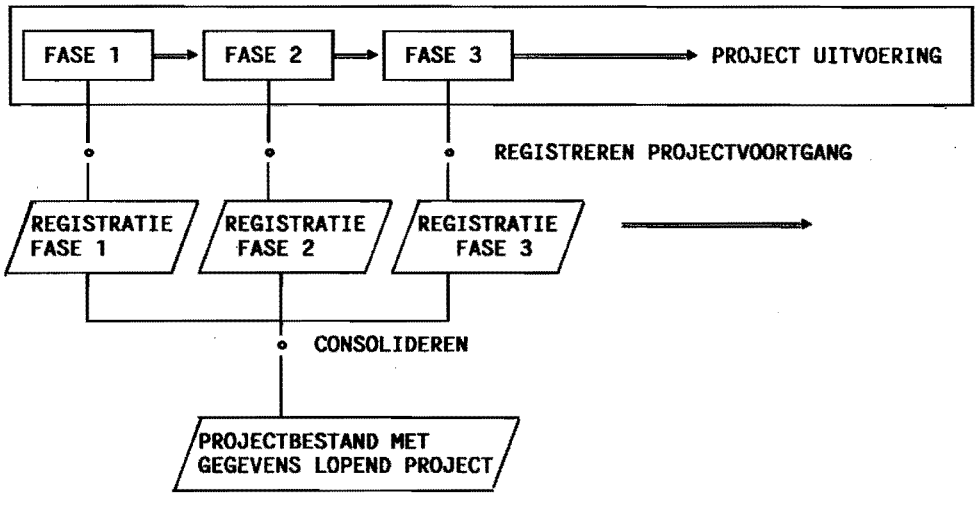
Figuur 8.10: Begroten

Dit geldt ongeacht welke begrotingsmethode wordt gebruikt. Het resultaat van deze activiteit zal uiteraard een begroting zijn van kosten, ontwikkeltijd, benodigde capaciteit enz. Omdat de activiteit "begroten" in de vorige hoofdstukken aan de orde is geweest, zal in deze paragraaf hierop niet verder worden ingegaan.

8.5.7. REGISTREREN EN CONSOLIDEREN

Na het opstellen van een projectbegroting en de goedkeuring van deze begroting kan gestart worden met de uitvoering van het project. Tegelijkertijd hiermee moet begonnen worden met het vastleggen van voortgangsgegevens van het project. Zoals al eerder is opgemerkt, is projectadministratie en -registratie een essentieel onderdeel van de beheerscyclus. Registreren is niet een eenmalige activiteit waarmee gewacht wordt tot het einde van een project. Projectregistratie moet

voortdurend tijdens de uitvoering van het project worden uitgevoerd. In het bijzonder bij het beheersen van een lopend project is het belangrijk doorlopend de begrote en gerealiseerde uitgaven onderling te vergelijken. Zo'n vergelijking is slechts op effectieve wijze mogelijk als voor elke fase, activiteit en taak vastgelegd wordt hoeveel geld, inspanning, middelen en tijd zijn besteed. Hierbij moeten uiteraard over dezelfde soort zaken en met het zelfde stelsel definities, gegevens worden vastgelegd over die zaken waarop de aannames en afspraken in de vorige paragraaf betrekking op hebben. Behalve voor het beheersen van een lopend project is projectregistratie noodzakelijk voor het in de toekomst begroten van nieuwe projecten. Na elk meetpunt in het project zal de betreffende registratie worden opgenomen en worden geconsolideerd in een zogenaamde lopend projectbestand.Uit het bovenstaande moge duidelijk zijn dat de omvang van dit lopende projectbestand tijdens het project zal groeien. In figuur 8.11 zijn de twee activiteiten "registreren" en "consolideren" met de noodzakelijke invoer en uitvoer in beeld gebracht.



Figuur 8.11 : Registreren en consolideren van projectgegevens.

8.5.8. VERGELIJKEN, ANALYSEREN EN CORRIGEREN

Wellicht een van de belangrijkste onderdelen van het raamwerk wordt gevormd door de activiteiten:

- vergelijk de realisatie met de begrotingen,
- analyseer mogelijke verschillen en
- vertaal de resultaten van deze analyses in eventuele correcties.

In figuur 8.3 is aangegeven dat de activiteit "vergelijken" en "analyseren" per fase c.q. meetpunt in een aantal stappen zal verlopen.

Stap 1.
Allereerst moet nagegaan worden of datgene wat gerealiseerd is en waar de voortgang van gemeten wordt, gelijk is aan datgene dat volgens de begroting gerealiseerd had moeten worden. Het is niet reëel een vergelijking te maken tussen werkelijke uitgaven en begrote uitgaven als het gerealiseerde eindresultaat c.q. tussenprodukt onacceptabel afwijkt van de specificaties waarop de begroting is gebaseerd.

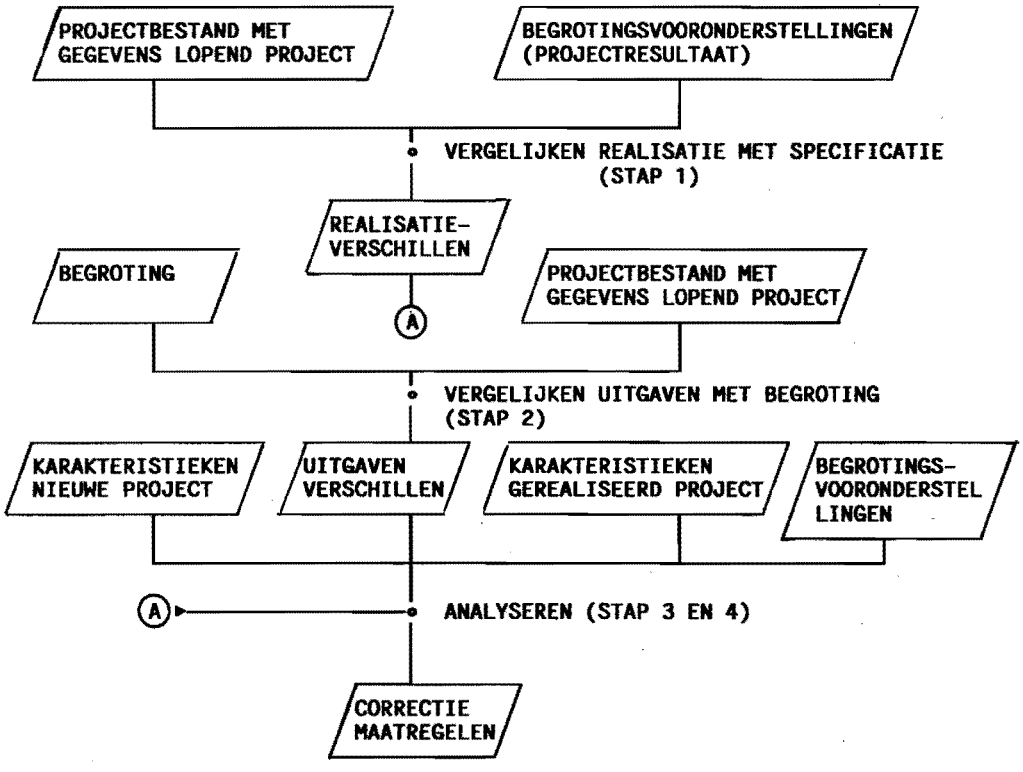
Stap 2.
Vervolgens worden de werkelijke uitgaven in termen van geld, tijd en middelen vergeleken met de begrote uitgaven. De volgende situaties kunnen optreden:

- Het kan zijn dat het project volgens plan verloopt. Dit wil zeggen dat datgene wat op een bepaald meetpunt gereed moet zijn, gerealiseerd is met de geschatte uitgaven.
- Het kan ook zijn dat er geen verschil is tussen werkelijke en geschatte uitgaven, maar dat een nadere analyse leert dat de begroting te ruim is opgesteld. De wet van Parkinson treedt dan in werking. De te ruime begroting wordt volledig benut, wat zich laat vertellen in een "te ruime" realisatie.
- Er kan sprake zijn van een significant overschrijding van de begroting.

In de tweede en derde situatie moet onderzocht worden wat de oorzaak hiervan is om gerichte bijsturing uit te oefenen alvorens verder gegaan kan worden met de volgende fase c.q. activiteit of taak van het project. Om dit te kunnen doen zijn de volgende twee vergelijkingsstappen noodzakelijk.

Stap 3.
Vergelijk de karakteristieken van het project, zoals deze destijds bepaald zijn, met de karakteristieken van het gerealiseerde project c.q. projectdeel.

Stap 4.
Vergelijk de begrotingsvooronderstellingen van het project met de werkelijkheid van het project.

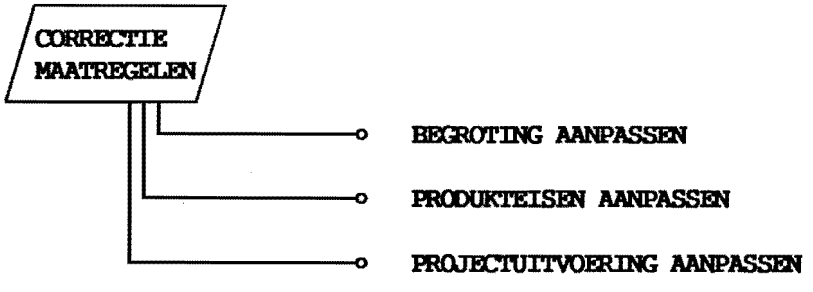


Figuur 8.12 : Vergelijken en analyseren.

Treden er verschillen aan het licht bij de hierboven genoemde stap 3 dan zijn de uitgangspunten bij het opstellen van de begroting verkeerd geweest en moeten deze eerst gecorrigeerd worden. Dit bete-

kent opnieuw karakteriseren van het project en nogmaals begroten maar dan op basis van de bijgestelde karakterisering.Zijn de verschillen terug te voeren tot afwijkingen van gemaakte vooronderstellingen dan zal nogmaals begroot moeten worden, maar nu met naleving van de eerder opgestelde aannames en afspraken. Ook deze begrotingen moeten weer vergeleken worden met de realisaties. In figuur 8.12 zijn de twee activiteiten "vergelijken" en "analyseren" inclusief de benodigde invoer en uitvoer in beeld gebracht.

Het zal duidelijk zijn dat bovengenoemde vergelijkingen slechts mogelijk zijn als de begrotingen, de projectkarakteristieken, de projectvooronderstellingen en de projectregistratie op zo'n wijze vastgelegd worden dat een vergelijking ertussen mogelijk is. Nadat deze bovengenoemde vier vergelijkingsstappen zijn uitgevoerd en mogelijke verschillen zijn geanalyseerd, kan, afhankelijk van het resultaat van deze analyses, gekozen worden uit de volgende correctie-maatregelen:(figuur 8.13)



Figuur 8.13 : Verschillende correctie maatregelen.

- *Het aanpassen van de begroting.*
  Het is zeer wel mogelijk dat de oorspronkelijke begroting niet realistisch blijkt te zijn. Ook in situaties waar ogenschijnlijk op rationele gronden een begroting is opgesteld, treft men deze aanpassing aan. Bijvoorbeeld als met behulp van een begrotingsmodel in combinatie met de expertmethode bij aanvang van een project op basis van vage doelstellingen een begroting voor het totale project wordt opgesteld. Tijdens de uitvoering krijgt de ontwikkelaar een steeds duidelijker beeld van de eisen van de gebruiker. Als gevolg van dit voortschrijdend inzicht kan hij na verloop van tijd con-

cluderen dat het systeem complexer is en dus veel meer tijd en kosten vereist dan oorspronkelijk begroot.

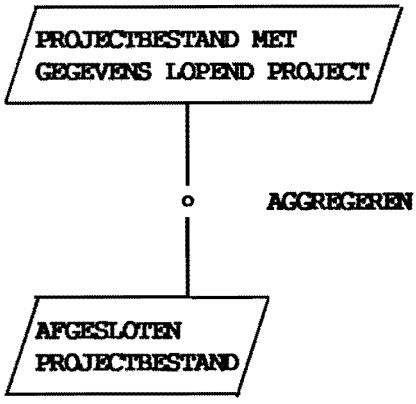
- *Het aanpassen van de projectuitvoering.*
  Hierbij kan men denken aan het inzetten van extra en hoger gekwalificeerd personeel, overwerk, uitbesteden van werk aan derden en inzetten van meer tools. Aanpassingen in de projectuitvoering hebben veelal directe gevolgen voor de begroting.

- *Het aanpassen van de produkteisen.*
  In deze situatie is het niet toegestaan veranderingen aan te brengen in het budget en de afgesproken levertijd. De enige mogelijkheid die overblijft is een aanpassing van het eindresultaat. Gebeurt dit in overleg met de opdrachtgever dan is deze maatregel, hoewel vervelend, verdedigbaar. Er kan afgesproken worden dat enkele, minder kritische modules komen te vervallen of dat het systeem wat betreft gebruiksvriendelijkheid soberder wordt.

In de praktijk zal men veelal een combinatie aantreffen van deze drie uiterste vormen van correctiemaatregelen. Voor welke maatregel gekozen wordt is deels afhankelijk van de analyses van de geconstateerde afwijkingen en deels van prioriteitsoverwegingen van de opdrachtgever c.q. ontwikkelaar.

8.5.9. AGGREGEREN

Na beëindiging van een project is er, behalve een programma, een grote hoeveelheid gegevens geregistreerd en geconsolideerd in een lopend projectbestand. Deze gegevensverzameling die primair bedoeld was om het project in uitvoering te beheersen, heeft na afloop van het project zijn diensten bewezen. De tweede doelstelling van projectregistratie wordt nu actueel, namelijk registreren ten behoeve van begroten in de toekomst. Voor dit doel is het niet noodzakelijk te beschikken over projectgegevens op eenzelfde detailniveau als in het lopend projectbestand. Hierin treft men ondermeer aan de weekmeldingen van elke individuele medewerker. Een noodzakelijke activiteit is nu het indikken/aggregeren van deze gegevens (figuur 8.14). Op deze wijze ontstaat een zogenaamd "afgesloten projectbestand". Hierin zijn gegevens opgenomen die ondermeer betrekking hebben op de belangrijkste kostenbepalende factoren in de betreffende ontwikkelomgeving.

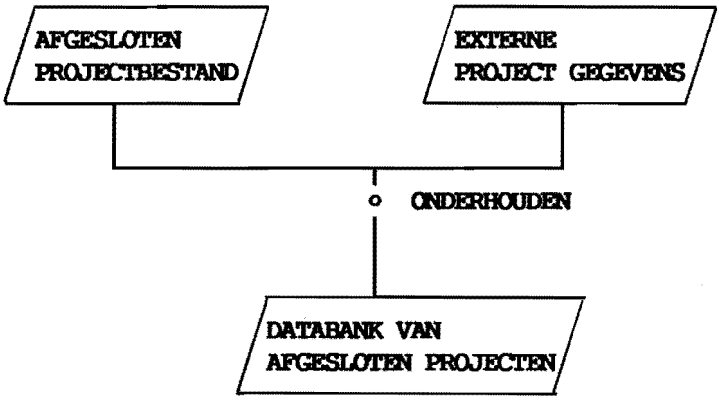


Figuur 8.14 : Aggregeren van projectdocumentatie tot gegevens over een voltooid project.

Bijvoorbeeld: type applicatie, ervaring personeel, gebruikte hulpmiddelen, complexiteit software, enz. In het afgesloten projectbestand worden ook gegevens opgenomen als de projectduur (totaal, per fase, per activiteit enz.) en aantal regels code (totaal, per module, enz.). Verder vormen gegevens die betrekking hebben op het analyseren van begrotingen versus realisaties en op het uitvoeren van aanpassingen een bron van ervaring voor het beheersen en begroten van toekomstige projecten. Bijvoorbeeld: wat waren de oorzaken voor de overschrijdingen, op welke punten was de karakterisering van het project verkeerd en waarom, van welke vooronderstellingen is afgeweken en waarom, hoe groot was de afwijking voor en na aanpassing enz. Bij het opbouwen van het afgesloten projectbestand is het belangrijk dergelijk soort "ervaringsgegevens" te verzamelen en vast te leggen. Door middel van de activiteit "aggregeren" moeten de gegevens ten behoeve van projectbeheersing gecondenseerd en vastgelegd worden in een afgesloten projectbestand. Welke gegevens vastgelegd moeten worden en welke gegevens bij welke beheersactiviteit gebruikt moeten worden, zijn vragen die nog niet beantwoord zijn. Het onderzoek van Noth (1987) houdt zich met deze vraagstelling bezig. Noth brengt de informatiebehoefte van een projectleider voor het beheersen van een automatiseringsproject in kaart. Het onderzoek van Kitchenham en Smith (1985) in het kader van het Alvey project in Groot Brittannië, genoemd in paragraaf 8.4, geeft een aanzet welke feiten en ervaringen het belangrijk zijn om in zo'n afgesloten projectbestand te worden opgenomen.

8.5.10. HET ONDERHOUDEN VAN EEN DATABANK VAN VOLTOOIDE PROJECTEN.

De databank van voltooide projecten wordt bijgewerkt met behulp van de afgesloten projectbestanden en externe projectgegevens (figuur 8.15). De afgesloten projectbestanden hebben betrekking op gegevens vanuit de eigen organisatie, terwijl de externe projectgegevens afkomstig zijn van een andere organisatie of afdeling.



Figuur 8.15 : Het onderhouden van een databank van voltooide projecten.

Veel organisaties trachten het gemis aan dit soort gegevens te compenseren door de aanschaf van software begrotingsmodellen. Deze modellen zijn immers gebaseerd op een uitgebreide verzameling oude projecten. Gaat men bij het opstellen van een begroting uitsluitend af op externe gegevens dan bestaat het gevaar dat de begroting eerder slechter dan beter wordt. Naast een verstandig gebruik van externe gegevens (bijvoorbeeld: onderzoeksresultaten naar de invloed van een bepaald vierde generatiehulpmiddel of produktiviteitsmetrieken afkomstig uit soortgelijke organisaties, verdelingen van kosten en tijd over de verschillende fasen van ontwikkeling enz.) dient iedere organisatie een eigen databank van afgesloten automatiseringsprojecten bij te houden.

8.6. EEN DATABANK VAN AFGESLOTEN PROJECTEN

Wil men bij het begroten van softwareprojecten met succes gebruik maken van een databank van voltooide projecten, dan zal aan een aantal dwingende voorwaarden voldaan moeten worden. Zo zal men ondermeer afspraken moet maken in welke stadia van een project een begroting wordt opgesteld, welke functionarissen hierbij betrokken moeten worden, hoe vaak een begroting wordt opgesteld tijdens een project, wat de status is van een begroting, welke marges bij de verschillende begrotingen worden ingebouwd, welke zaken een begroting worden meegenomen enz. Verder zal men afspraken moeten maken over de omschrijving van begrippen als omvang, kwaliteit en complexiteit van de software, over kwaliteit en ervaring van personeel enz. Met andere woorden, men zal moeten komen tot een definitie van kostenbepalende factoren. Ook zal men moeten aangeven hoe men deze factoren meet. In paragraaf 4.10 zijn hiervan een aantal voorbeelden genoemd. Het is belangrijk dat eenmaal gekozen definities en maatvoeringen konsekwent wordt toegepast. Hierdoor onstaat op den duur een verzameling voltooide projectgegevens die alle op een uniforme wijze zijn beschreven en zodoende onder bepaalde voorwaarden onderling vergelijkbaar zijn. Het voorbehoud "onder bepaalde voorwaarden" wordt hier gemaakt omdat ondanks een gelijke beschrijvingswijzen onderlinge vergelijking van projecten, die weinig overeenkomsten met elkaar hebben, niet zinvol is. Denk hierbij aan batch-applicaties in de administratieve sfeer en on-line toepassingen voor technisch getinte applicaties. Om er voor te zorgen dat de afspraken en richtlijnen binnen een ontwikkelorganisatie worden nageleefd, is het aan te bevelen hiervoor een beheersfunktie te creëren. Een belangrijk aspect van deze funktie zal zijn het controleren of de beschrijving van elk voltooid project op dezelfde standaardwijze heeft plaatsgevonden, of de beschrijvingen volledig zijn er geen verwarring bestaat over te hanteren definities, maatvoeringen e.d. Een ander aspect van deze funktie is het onderhouden van de databank.

In deze paragraaf wordt een aanzet gegeven voor de beantwoording van de vraag "welke soort gegevens van voltooide automatiseringsprojecten moeten worden opgenomen in een databank om het begroten van toekomstige projecten te ondersteunen".

Bij het registreren van gegevens over voltooide projecten is het voor de overzichtelijkheid raadzaam een onderscheid te maken in:

- *Realisatiegegevens.*
  Dit soort gegevens heeft betrekking op de werkelijk bestede middelen. Deze gegevens zijn bekend na afronding van het project, van een fase c.q. activiteit. Zover als mogelijk tracht men ook vast te leggen welke factoren in dit project in welke mate van invloed zijn geweest op de kosten.
- *Begrotingsgegevens.*
  Deze gegevens hebben betrekking op de schattingen van ontwikkelkosten, doorlooptijd, benodigde menskracht en middelen. De schattingen van de waarden van de cost drivers moeten eveneens vastgelegd worden.
- *Analysegegevens.*
  De analyseresultaten van de afwijkingen tussen realisatie en begroting zijn een belangrijk onderdeel van de databank van voltooide projecten. Het zijn juist dit soort gegevens die een bron van ervaring vormen bij het begroten van nieuwe projecten (wat zijn de oorzaken van miscalculaties, welke zaken zijn over het hoofd gezien, e.d.). Uit analyses van oude projecten moet duidelijk worden wat de belangrijkste oorzaken van miscalculaties zijn. Van Genuchten (1988) heeft van een groot aantal projecten onderzocht wat de oorzaken van de verschillen tussen plan/begroting en realiteit waren. Het bleek dat het merendeel van de afwijkingen terug te voeren is tot een beperkt aantal oorzaken. In paragraaf 1.3 is een voorbeeld gegeven van een opsomming van mogelijke oorzaken.

Bij de toelichting op begrotings- en realisatiegegevens is impliciet een onderscheid gemaakt in gegevens die betrekking hebben op de uitvoer en gegevens die betrekking hebben op de invoer van een begroting. Vanuit begrotingsoptiek zijn voorbeelden van uitvoergegevens: de projectduur in maanden, de doorlooptijd in mensmaanden en de kosten in guldens. Dit alles uitgesplitst in fasen c.q. activiteiten. Een voorbeeld van invoergegevens zijn de kostenbepalende factoren, genoemd in hoofdstuk 5.

De gegevens in de databank van afgesloten projecten kunnen ook worden ingedeeld naar beschrijvende gegevens en voortgangsgegevens.

- *Voortgangsgegevens.*
  Hiermee wordt bedoeld dat de waarde van dit soort gegevens bij aanvang van een project, fase, activiteit of taak geschat moet worden of pas na afronding ervan bepaald kan worden. Alleen voor dit soort gegevens is de indeling in realisatie-, begrotings- en analysegegevens van toepassing.

Tabel 8.2: Samenhang tussen soorten gegevens in een databank van voltooide projecten.

BESCHRIJVENDE GEGEVENS				
projectnaam : voorraadbeheer-86 projectleider : heemstra				
VOORTGANGSGEGEVENS				
INVOERGEGEVENS		BEGROTING	REALISATIE	ANALYSE
PRODUKT:	omvang	124.000	159.756	22
	hergebruik code	30%	32%	-
MIDDELEN	geheugenbeperking	10%	35%	12
PERSONEEL	verloop	0%	5%	-
	ervaring team (jaren)	7	3	7
PROJECT	beperkingen projectduur	10%	20%	18
GEBRUIKER	participatie	35%	10%	3, 4
	aantal wijzigingen spec's	1	6	1, 3
UITVOERGEGEVENS		BEGROTING	REALISATIE	ANALYSE
KOSTEN: (guldens * 1.000)	totaal	1.200	1.750	1, 4
	fase analyse	100	500	1
	reis en verblijf	20	22	-
	opleiding	12	12	-
DOORLOOPTIJD: (maanden)	totaal	37	48	1, 4, 6
	fase analyse	3	5	1
INSPANNING: (mensmaanden)	totaal (per fase)	420	600	7

- *Beschrijvende gegevens.*
  Een deel van de gegevens die opgenomen worden in de databank heeft een beschrijvend karakter. Dat wil zeggen dat de waarde van dit soort gegevens niet geschat hoeven of kunnen worden. De waarden liggen bij aanvang van het project vast of kunnen pas na afloop van het project ingevuld worden. Voorbeelden hiervan zijn:

- naam/code van het project,
- namen gebruikte hulpmiddelen,
- gebruikte programmeertaal,
- naam projectleider,
- type applicatie,
- beschrijving van wijzigingsvoorstellen,
- beschrijving van verstoringen opgetreden tijdens de uitvoering van het project.

De samenhang tussen de verschillende indelingen in gegevenssoorten is in Tabel 8.2 in beeld gebracht.

Om het geheel te verduidelijken zijn in het overzicht een aantal "dummy-gegevens" opgenomen. Nu bekend is welk soorten gegevens geregistreerd moeten worden, zal worden aangegeven om welke gegevens het daadwerkelijk gaat. Het is zaak alleen de meest relevante gegevens vast te leggen. Het kost absoluut geen moeite een enorme hoeveelheid gegevens van elk project te registreren. Een
Een heel simpele vorm van registratie, zoals weergegeven in tabel 8.3, kan voor een ontwikkelorganisatie al bijzonder verhelderend werken (Pressman 1987).

Tabel 8.3 : Een voorbeeld van een beperkte vorm van project-registratie. Inspanning wordt uitgedrukt in mens-maanden. De waarden van de getallen in de kolommen regels code en kosten moeten met 1000 worden vermenigvuldigd. De letters B en W in de kop van de tabel staan voor respectievelijk Begroot en Werkelijk.

project-code	type applic.	Inspan. B	Inspan. W	kosten B	kosten W	reg.code B	reg.code W	pag.doc. B	pag.doc. W	medew. B	medew. W	oorzaak	fout
aaa-01	adm.	14	24	132	168	8.0	12.1	320	365	3	3	1,3 + 5	29
ccc-04	AI	38	62	400	440	25	27.2	950	1224	6	5	1	86
fff-03	adm	28	43	300	314	15.5	20.2	825	1050	6	6	1 + 6	64
.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.

Zo valt uit de tabel af te leiden dat project met code aaa-01 gerekend moet worden tot de categorie "administratieve software" en dat de ontwikkelinspanning geschat is op 14 mensmaanden maar in

werkelijkheid 24 mensmaanden bedraagt.
De totale ontwikkelkosten zijn begroot op 132.000 gulden en blijken in werkelijk 168.000 gulden te zijn. Verder valt uit de projectregistratie af te lezen, de geschatte en werkelijke hoeveelheid regels code, het aantal pagina's documentatie en het aantal projectmedewerkers. Verder is (via een code) vastgelegd wat de oorzaken zijn van de afwijkingen tussen begroting en werkelijkheid. In de laatste kolom van tabel 8.3 (fout) is aangegeven dat het aantal fouten dat binnen een jaar na oplevering van het systeem optrad, 29 is.
Zelfs met behulp van deze elementaire gegevens kunnen al een aantal belangrijke analyses worden uitgevoerd. Zo kan de gemiddelde produktiviteit over alle projecten worden berekend:

Produktiviteit = regels code / mensmaanden

Ook is het mogelijk de produktiviteit per type applicatie te berekenen. Een dergelijke maat is een waardevolle ondersteuning bij het begroten van toekomstige soortgelijke projecten. Een mogelijke verfijning zou kunnen zijn om de begrotingen uit te splitsen naar de verschillende fasen van een project.
Uit de gegevens kan ook geconcludeerd worden dat er een systematische onderschatting is van de benodigde inspanning met circa 40%. Ook dit levert weer zinvolle informatie op voor toekomstige begrotingen. Eenzelfde verschijnsel treedt op bij aantal pagina's documentatie. Ondanks deze onderschattingen blijken er geen noemenswaardige verschillen op te treden tussen werkelijke en begrote kosten. Voor het projectmanagement een reden hier nader onderzoek na te doen. Verder valt op dat code 1 vaak genoemd wordt als oorzaak van de afwijking tussen begroting en realiteit. Voor het projectmanagement is dit een belangrijk signaal. Bijsturing op dit punt kan een positief effect hebben op het in de hand houden van toekomstige projecten. Met behulp van de geregistreerde gegevens is het mogelijk ook nog andere kengetallen te bepalen, zoals:

- kosten / regels code (kosten per regel code),
- pagina's documentatie / regels code (de verhouding tussen omvang en hoeveelheid documentatie).
- fouten / regels code (een maat voor de kwaliteit van het eindresultaat).

Het voorbeeld in tabel 8.3 laat zien dat zelfs zo'n beperkte vorm van registratie een waardevolle ondersteuning is bij het begroten. Het verzamelen en vastleggen van dergelijke gegevens kost nauwelijks tijd en kan gemakkelijk handmatig worden uitgevoerd. Dit ligt anders als het gaat om een uitgebreide vorm van registratie. Al eerder is gewezen op de noodzaak van een eenduidig begrippenstelsel, van het spreken van eenzelfde taal en van het naleven van gemaakte afspra-

ken en richtlijnen. Dergelijke maatregelen snijden diep in een organisatie en vergen de nodige inspanning en discipline. Van een handmatig systeem kan hier geen sprake meer zijn. Het statistisch manipuleren met de gegevens, het afleiden en actualiseren van kengetallen en het snel zoeken in de verzameling projectgegevens maken een computerondersteuning noodzakelijk.

8.7. CONCLUSIES

In dit hoofdstuk is de betekenis van bestuurlijke informatie bij het begroten en beheersen van softwareprojecten toegelicht. Er is een onderscheid gemaakt in twee informatiebronnen bij het begroten c.q. herbegroten. Deze zijn produktgegevens - de eisen waaraan de software moet voldoen - en procesgegevens - gegevens van afgesloten projecten en voortgangsgegevens van het project in kwestie.
In dit hoofdstuk zijn grofweg drie delen te onderscheiden. In het eerste deel wordt nader ingegaan op het belang, de rol en de structuur van een registratief systeem van voltooide projectgegevens. In het tweede deel wordt een raamwerk voor het begroten en beheersen van softwareprojecten beschreven. In het derde deel wordt op een onderdeel van dit raamwerk, de databank van voltooide projectgegevens, nader ingegaan.
Ondanks het belang van gegevens van voltooide projecten voor een organisatie, met name bij het begroten en bewaken van softwareprojecten, treft men in de praktijk weinig databanken met dergelijk soort gegevens. In het eerste deel van dit hoofdstuk worden de belangrijkste redenen voor dit manco aangegeven.
In het tweede deel van dit hoofdstuk is verder een raamwerk ontwikkeld voor het begroten en beheersen van softwareprojecten. Duidelijk is geworden dat begroten onlosmakelijk verbonden is met de overige beheersactiviteiten, te weten het formuleren van het projectresultaat, het plannen van de noodzakelijke activiteiten, het controleren (meten) van de projectvoortgang en het bijsturen op basis van eventueel geconstateerde verschillen. Verder is naar voren gekomen dat begroten niet een eenmalige activiteit is maar meerdere malen tijdens de uitvoering moet plaatsvinden. In het raamwerk is dit iteratief aspect van begroten duidelijk aangegeven. Registreren loopt als een rode draad door het hele raamwerk. Projectresultaat en projectweg dienen vastgelegd te worden, de karakterisering van het project, de gemaakte vooronderstellingen en alle begrotingsresultaten moeten geregistreerd worden. Tijdens de uitvoering van het project moet voortdurend

verslag gelegd worden van de projectvoortgang in termen van bestede tijd, geld, middelen en geleverde kwaliteit. Verschillen tussen begroting en werkelijkheid dienen vastgelegd te worden. Het gepresenteerde raamwerk kan gebruikt worden als een referentiemodel voor het beheersen van automatiseringsprojecten. Alle belangrijke activiteiten, hun onderlinge afhankelijkheden en de vereiste gegevensverzamelingen zijn erin opgenomen. Het raamwerk kan ook gebruikt worden om als norm te dienen bij het beoordelen van bestaande projectorganisaties. Tevens kan het gebruikt worden als hulpmiddel bij het selecteren van tools voor het beheersen c.q. begroten van softwareprojecten.
Tenslotte is in het derde deel van dit hoofdstuk in grote lijnen aangegeven welk soort gegevens in een databank van voltooide projectgegevens opgenomen moet worden.

9. NABESCHOUWING

9.1. SAMENVATTING

In deze studie heeft het onderwerp beheersen en begroten van software-ontwikkeling centraal gestaan. De aanleiding vormden de signalen over forse overschrijdingen van budgetten en levertijden bij het ontwikkelen van software. Afgaande op berichten uit de praktijk en bevindingen in de literatuur komt het geregeld voor dat softwareprojecten uit de hand lopen.
In deze studie kunnen drie delen worden onderscheiden.

In het eerste deel van het onderzoek ligt het accent op een verkenning van het onderzoeksobject - het begroten van software-ontwikkeling. Door middel van een veldonderzoek en een inventarisatie van bestaande theorieën over en methoden voor begroten, is een overzicht gegeven van de huidige stand van zaken. Het beeld dat hierbij ontstaat laat vooralsnog een aantal witte vlekken op de "begrotingslandkaart" zien. Het onderwerp staat nog in de "kinderschoenen", hoewel er voldoende redenen zijn om het binnen het vakgebied softwareontwikkeling een volwassen plaats te laten innemen. Hiervoor is uitgebreid onderzoek op dit gebied noodzakelijk.

Het tweede deel van het onderzoek heeft een ordenend karakter. Zo worden de belangrijkste begrotingsmodellen beschreven en geëvalueerd. Deze evaluatie is uitgevoerd met behulp van een beoordelingsmethode, die door de onderzoeksgroep "Begroten van automatiseringsprojecten" van de Technische Universiteit Eindhoven is ontwikkeld. Belangrijke conclusies uit de uitgevoerde evaluaties zijn:

- er is vooralsnog een gebrek aan empirisch materiaal om uitspraken te kunnen doen over de nauwkeurigheid van de modellen;
- de meeste modellen bieden weinig mogelijkheden voor calibratie met gegevens van voltooide projecten;

- in begrotingsmodellen is er een duidelijke tendens om de omvang van de software met behulp van een op Funktie Punt Analyse gebaseerd aanpak te schatten;
- dat de meest recent ontwikkelde modellen worden geïmplementeerd in een geautomatiseerde versie. Deze zogenaamde begrotingspakketten zijn doorgaans eenvoudig in gebruik, snel toe te passen en bieden mogelijkheden om gevoeligheids- en risico-analyses uit te voeren.

Verder is er een ordening gemaakt van belangrijke factoren die van invloed zijn op de kosten, inspanning en doorlooptijd van software-ontwikkeling. Een analyse van deze factoren laat zien dat ze betrekking hebben op een van de volgende vijf aspecten:

- het te ontwikkelen produkt,
- het ontwikkelpersoneel,
- de hulpmiddelen,
- de projectorganisatie,
- de gebruiker.

Bij het begroten en beheersen van software-ontwikkeling is het belangrijk dat het projectmanagement inzicht heeft in welke factoren in welke mate de kosten bepalen. Hoe groter dit inzicht, des te beter men in staat is de ontwikkeling te beheersen en de kosten te begroten. Een belangrijke conclusie uit de analyse van de kostenbepalende factoren is, dat het doorgaans niet eenvoudig is deze factoren te operationaliseren en de invloed ervan op kosten en doorlooptijd aan te geven.
Verder komt uit de analyse naar voren, dat er geen sprake kan zijn van "de belangrijkste kostenbepalende factoren". Welke factoren belangrijk zijn en de invloed van die factoren op de kosten, wordt bepaald door kenmerken van de betreffende ontwikkelsituatie.

Vanuit de inventarisatie en ordening zijn ideeën en theorieën ontwikkeld, die een basis vormen voor een betere beheersing en begroting van software-ontwikkeling. Deze zijn beschreven in het derde deel van de studie.
Er is een typologie ontwikkeld voor het beheersen en begroten van software. Deze typologie is voor een belangrijk deel gebaseerd op het P-B-I model (Bemelmans 1987). In de typologie worden vier verschillende beheerssituaties onderscheiden. Welke beheerssituatie van toepassing is op een specifieke software-ontwikkeling wordt bepaald door de kenmerken van de software die moet worden ontwikkeld, het proces dat moet worden uitgevoerd en de middelen die nodig zijn. Het kenmerkend verschil tussen de vier beheerssituaties wordt ge-

vormd door de mate van zekerheid die men heeft over het produkt, het proces en de middelen.
Is die zekerheid groot dan heeft beheersen het karakter van een routinematig karwei. Het accent ligt op de realisatie. Gestreefd wordt naar het maximaliseren van de efficiëncy en het minimaliseren van de doorlooptijd. Omdat er zekerheid is over het te realiseren produkt zal men ernaar streven dit te realiseren met een optimaal mogelijk gebruik van middelen. Een begroting heeft in een dergelijke situatie een taakstellende en bewakende funktie.
Totaal anders ligt het, als de onzekerheid groot is; beheersen heeft dan het karakter van een ontwerpkarwei. Omdat het produkt, proces en de middelen een onzekere factor zijn, zal men ernaar streven de risico's te verlagen. Een mogelijkheid is de middelen vast te leggen en met deze gegeven middelen het eindresultaat te maximaliseren. Risico's kunnen tevens verkleind worden door de software volgens een incrementele aanpak te ontwikkelen en veel beslispunten in het ontwikkeltraject in te bouwen. Beheersen en begroten is in zo'n situatie gebaseerd op mutual adjustment en committering. Een begroting kan bij grote onzekerheid geen taakstellende funktie hebben, maar heeft een voorwaardenscheppende funktie. Bij het begroten zal men met name behoefte hebben aan risico-analyses.

De studie heeft het inzicht bevestigd, dat gegevens van voltooide softwareprojecten nodig zijn voor het beheersen en begroten. Verder is beargumenteerd dat dergelijke gegevens maar in een beperkt aantal situaties bruikbaar is, namelijk in dié situaties waarin het mogelijk is het eindresultaat precies te omschrijven.
De bestaande begrotingsmethoden en -modellen zijn gebaseerd op een dergelijke verzameling gegevens. Hoewel het belang van historische projectgegevens alom wordt erkend, is het opvallend hoe weinig binnen organisaties projectgegevens worden verzameld en op een systematische wijze worden vastgelegd. In deze studie is voor dié beheerssituaties, waarin sprake is van een hoge produktzekerheid, een eerste aanzet gegeven welke soort gegevens van voltooide projecten vastgelegd moeten worden en op welke wijze deze gegevens voor beheersen en begroten gebruikt kunnen worden.

We sluiten deze studie af met het geven van een aantal concrete richtlijnen voor het beheersen en begroten van software-ontwikkeling. De meeste aanbevelingen zijn in de vorige hoofdstukken reeds aan de orde geweest. Ze zijn echter van zo'n belang dat elk aanbeveling als een onderwerp voor nadere studie gezien kan worden.

9.2. AANBEVELINGEN

aanbeveling 1: typeer beheerssituatie.

Zoals in hoofdstuk 6 is aangegeven, is de wijze waarop software-ontwikkeling beheerst en begroot dient te worden, afhankelijk van de kenmerken van de software die ontwikkeld moet worden, van het ontwikkelproces en van de benodigde en beschikbare middelen i.c. ontwikkelpersoneel. Voor de keuze van de juiste beheersstrategie en begrotingsaanpak is het belangrijk inzicht te hebben in deze kenmerken. Een belangrijke stap is daarom het karakteriseren en typeren van de software-ontwikkeling in kwestie.

In deze studie is een aanpak beschreven om tot zo'n typologie van beheerssituaties te komen. Het verdient aanbeveling om in een verder onderzoek de gepresenteerde typologie te verfijnen en de praktische waarde ervan in de praktijk te toetsen.

aanbeveling 2: beheers en begroot evolutionair.

Naarmate men duidelijker het eindresultaat van de ontwikkeling kan omschrijven, beter in staat is aan te geven op welke wijze men dit resultaat kan realiseren en welke middelen men hiervoor nodig heeft, zijn de voorwaarden om de ontwikkeling te beheersen en begroten beter. In hoofdstuk 7 is in dit verband gesproken over een geleidelijke overgang van de ene beheerssituatie naar de andere. De kenmerken van een bepaalde software-ontwikkeling zijn niet gedurende het hele ontwikkeltraject stabiel. Aanvankelijk zal men moeite hebben met het bepalen van produkt-, proces- en middelenkenmerken. Naarmate de ontwikkeling vordert, is men beter in staat de software-ontwikkeling te karakteriseren. Dit betekent dat men tijdens de ontwikkeling van software voortdurend alert moet zijn welke beheerssituatie op een bepaald moment actueel is. Verandert de beheerssituatie voor een bepaalde software-ontwikkeling, dan zal men de beheersstrategie en begrotingsaanpak moeten aanpassen. Het typeren van de beheerssituatie is dus een activiteit, die meerdere malen in het ontwikkeltraject moet worden uitgevoerd. Het moment van typeren kan samenvallen met de beslispunten tijdens de ontwikkeling.

Er is nader onderzoek gewenst naar het optreden van verschillende beheerssituaties tijdens de uitvoering van een softwareproject. Is een bepaalde beheerssituatie dominant aanwezig in een bepaalde fase van ontwikkeling, en zoja van welke factoren is dat afhankelijk? Is er sprake van een zekere volgorde waarin de verschillende be-

heerssituaties elkaar opvolgen? Zijn voor verschillende type applicaties andere beheerssituaties dominant? Om een antwoord te kunnen geven op dergelijk soort vragen, is meer empirisch onderzoek naar het beheersen van software-ontwikkeling noodzakelijk.

aanbeveling 3: bouw beslispunten in.

Het inbouwen van beslispunten in een software-ontwikkeltraject wordt algemeen geaccepteerd. Alle bestaande projectbeheersingsmethoden zijn gebaseerd op dit principe. Het aantal beslispunten en de frequentie van deze punten zal ondermeer bepaald moeten worden door de betreffende beheerssituatie. De beheersbaarheid van een project en de nauwkeurigheid van een begroting worden groter als men meer zekerheid heeft over de te ontwikkelen software, het ontwikkelproces en de middelen. Is de onzekerheid groot dan zal men vanuit beheersoptiek zeer frequent moeten meten of de juiste software, op de juiste wijze, met de juiste middelen wordt gemaakt.

De huidige stand van het onderzoek naar beheersen van software-ontwikkeling geeft geen antwoord op de vraag welk aantal meetpunten het meest optimale is bij een bepaalde mate van onzekerheid. Te weinig meetpunten heeft tot gevolg dat het projectmanagement te traag reageert op afwijkingen tussen plan/begroting en werkelijkheid. Te veel meetpunten leidt tot "oversturing". Nader onderzoek op dit punt is gewenst.

aanbeveling 4: registreer : meten is weten.

In deze studie is een pleidooi gehouden voor het aanleggen van een databank met gegevens van voltooide softwareprojecten. In situaties waarin men duidelijkheid heeft over de te ontwikkelen software, kunnen dergelijke gegevens een waardevolle ondersteuning zijn bij het beheersen en begroten. Op basis van produktkenmerken kan in een databank worden gezocht naar overeenkomstige, in het verleden uitgevoerde projecten. Deze analoge produktgegevens kunnen het projectmanagement ondermeer helpen om eventuele onzekerheden over het uit te voeren proces en vereiste middelen te verminderen. In deze studie is een aanzet gegeven voor een registratief systeem van voltooide projecten. Er is aangegeven welk soort gegevens vastgelegd kunnen worden en op welke wijze deze gegevens tijdens de software-ontwikkeling gebruikt kunnen worden ten behoeve van beheersen en begroten.

De huidige stand van het onderzoek naar het beheersen en begroten van software-ontwikkeling geeft vooralsnog geen duidelijk

antwoord op de vraag welke gegevens van voltooide projecten in een databank opgeslagen moeten worden ten behoeve van het beheersen van lopende projecten en het begroten van nieuwe projecten. Door de onderzoeksgroep "Begroten van Automatiseringsprojecten" van de Technische Universiteit Eindhoven wordt momenteel uitgebreid onderzoek uitgevoerd op dit onderwerp (van Genuchten 1989, van Lierop en Volkers 1989).
Om een aantal redenen is het moeilijk aan te geven welke gegevens relevant zijn om opgenomen te worden in een databank van afgesloten projecten. De belangrijkste hinderpaal is dat deze gegevens sterk situatiegebonden zijn. In de ene ontwikkelomgeving zullen andere gegevens relevant zijn dan in de andere; de gegevens zullen per type applicatie verschillen. Een en ander betekent dat elke organisatie afzonderlijk de inhoud van een dergelijke databank zal moeten bepalen. Om organisaties hierbij te helpen is het aan te bevelen onderzoek te verrichten naar een referentiemodel voor projectbeheersing.

aanbeveling 5: gebruik meerdere begrotingsmethoden.

Het accent in deze studie heeft voornamelijk gelegen op het beheersen en begroten van omvangrijke softwareprojecten. De sociale, organisatorische, technische en financiële risico's zijn bij dergelijke projecten doorgaans groot. Gezien die grote risico's is het voor organisaties belangrijk dat zij in staat zijn deze risico's in te schatten, begrotingen te maken van ontwikkeltijd, -geld en -kosten en aan te geven wat de betrouwbaarheid is van dergelijke begrotingen. Deze bewering ligt zo voor de hand, dat ze bijna triviaal is. Desondanks is het opmerkelijk, dat volgens de enqueteresultaten van hoofdstuk 2 door slechts 65% van de responderende organisaties een begroting wordt opgesteld. Slechts 27% houdt zich serieus bezig met kostenbeheersing van software. Dat wil zeggen, er wordt een begroting opgesteld, er vindt nacalculatie plaats en gegevens over voltooide projecten worden geregistreerd.
In de hoofdstukken 3 en 4 is gesignaleerd dat er een groot aantal begrotingsmethoden en -modellen beschikbaar zijn. Verder is in deze studie naar voren gekomen dat de kwaliteit van de huidige methoden en modellen vooralsnog gering is. Deze geringe kwaliteit en de eerder genoemde risico's maken het noodzakelijk om frequent te begroten en hierbij gebruik te maken van meerdere methoden. Een mogelijke begrotingsaanpak kan er als volgt uitzien: leg de omschrijving van de te ontwikkelen software voor aan meerdere experts; laat deze experts onafhankelijk van elkaar een begroting opstellen; zoek in een databank van voltooide projecten naar analogieën; maak een of meerdere begrotingen met behulp van een begrotingsmodel en maak hierbij

gebruik van de gegeven produktomschrijving en de gegevens van analoge projecten; confronteer de experts vervolgens met elkaars begrotingen; probeer in overleg tot consensus te komen; vraag tenslotte iedere expert afzonderlijk hoe hard de voorspellingen zijn. Op deze wijze krijgt men van een project een aantal begrotingen. De mate waarin de verschillende begrotingsresultaten onderling verschillen, kan worden beschouwd als een maat voor de onzekerheid en het risico van het project. Liggen de resultaten ver uiteen dan is dit een signaal voor het projectmanagement dit project extra te bewaken. Ook kan het een waarschuwing zijn dat de projectomschrijving te onduidelijk is. In ieder geval zal onderzocht moeten worden waarom er verschillen zijn.

Er is nader onderzoek nodig welke combinatie van methoden en modellen in een bepaalde situatie de voorkeur geniet en welke criteria die keuze bepalen. De huidige stand van het onderzoek biedt geen oplossing voor dit probleem. Ook verdient het aanbeveling een begrotingsaanpak zoals hiervoor beschreven nader uit te werken en in de praktijk te toetsen op bruikbaarheid.

aanbeveling 6: calibreer begrotingsmodellen.

In deze studie is meer dan eens gesignaleerd dat de kwaliteit van begrotingsresultaten, verkregen met behulp van een model, voor een belangrijk deel wordt bepaald door de calibratie van een model. Commercieel verkrijgbare begrotingsmodellen zijn over het algemeen gebaseerd op projectgegevens en heuristieken, die specifiek zijn voor de omgeving waarin het model is ontwikkeld. De kenmerken van de omgevingen waarin het model wordt toegepast, wijken hier in de meeste gevallen vanaf. Het gevolg hiervan is, dat begrotingsresultaten verkregen met modellen, die niet gecalibreerd zijn met eigen projectgegevens, grote verschillen laten zien met realisatiegegevens. In deze studie zijn een aantal onderzoeken genoemd, die dit bevestigen.
Hoewel calibratie onmiskenbaar een voorwaarde is voor het met succes toepassen van begrotingsmodellen, blijkt uit de beoordeling van de belangrijkste modellen in hoofdstuk 4, dat slechts weinige modellen mogelijkheden tot calibratie ondersteunen. Voor organisaties die gebruik willen maken van begrotingsmodellen wil dit zeggen dat bij modelselectie de mogelijkheid tot calibratie een belangrijke factor moet zijn. Een andere mogelijkheid is, dat een organisatie een eigen begrotingsmodel ontwikkelt. Het model COCOMO, dat door Boehm (1981) is beschreven, kan model staan voor de wijze waarop een begrotingsmodel kan worden ontwikkeld.

Gezien de beperkte kwaliteit van de bestaande begrotingsmodel-

len is nader onderzoek zeker gewenst. Begrotingsmodellen voldoen slechts in zeer beperkte mate aan de eisen die genoemd worden in de beoordelingsmethode, beschreven in hoofdstuk 3 van dit onderzoek. De researchgroep "Begroten van Automatiseringsprojecten" heeft in haar onderzoeksplan opgenomen het maken van een functioneel ontwerp voor een begrotingsmodel. Dit ontwerp dient zodanig te zijn dat voldaan wordt aan de eisen uit de beoordelingsmethode.

aanbeveling 7: zorg voor commitment.

Naarmate er meer onzekere factoren spelen bij het ontwikkelen van software, wordt het belangrijker projectmedewerkers te betrekken bij het opstellen van een begroting. Wil een organisatie onder hoge onzekerheid software ontwikkelen, dan zal zij daarbij gebruik moeten maken van professioneel, hoog gekwalificeerd personeel. Voor professionele ontwikkelaars gelden doorgaans de volgende zaken (Grinwis 1989):

- individualisme in de wijze van beroepsuitoefening wordt belangrijk gevonden;
- een goed professioneel resultaat vindt men belangrijk;
- men wenst door het management gekend te worden in beslissingen over de wijze van werken en het gewenste eindresultaat;
- bij de uitoefening van het werk wenst men zo weinig mogelijk door management "voor de voeten gelopen" te worden.

Ontwikkelaars, aldus gekarakteriseerd, moeten niet worden geconfronteerd met een van hogerhand opgelegde begroting en plan. Een hierarchiesch leiderschap past niet in deze situatie. Plan en begroting moet in overleg tot stand komen. Behalve dat op deze wijze het oordeel van een specialist/expert wordt ingewonnen, worden ook in een vroeg stadium de uiteindelijke uitvoerders gekend in de begrotingsproblematiek. De betrokkenheid wordt daardoor vergroot en men krijgt op deze wijze het commitment dat zo noodzakelijk is voor het welslagen van een project.

Het is belangrijk dat onderzoek wordt uitgevoerd naar het effect van commitment van de ontwikkelaar bij het begroten ten opzichte van zowel het ontwikkelproces als het te ontwikkelen produkt. Onderzoek naar dit effect is nog onvoldoende uitgevoerd.

aanbeveling 8: beperk het aantal cost drivers.

In dit onderzoek is gesignaleerd dat een groot aantal factoren van invloed kan zijn op de ontwikkelkosten van software. Verder is er

gewezen op de problematiek rond deze factoren. Het merendeel van de kostenbepalende factoren is moeilijk te definiëren, te meten en te kwantificeren. Het is vaak lastig aan te geven wat de relatie is tussen een cost driver en de ontwikkelkosten, doorlooptijd en inspanning. In veel gevallen speelt subjectiviteit hierbij een grote rol. Bij het opstellen van een begroting wegen deze problemen zwaarder omdat men vooraf een schatting moet maken van de waarde van een kostenbepalende factor.
Een deel van deze problematiek kan verminderd worden door niet alle factoren in de begroting te betrekken, maar dit aantal te beperken tot de meest dominante. Naar alle waarschijnlijk zal bij de ontwikkeling van software gelden, dat het grootste deel van de kosten veroorzaakt wordt door een beperkt aantal factoren. Door middel van het analyseren van gegevens van afgesloten projecten, kan een organisatie achterhalen welk beperkt aantal factoren dit is. Het reduceren van het aantal factoren heeft tal van voordelen. Zo hoeft slechts over een beperkt aantal kostenbepalende factoren afspraken te worden gemaakt over definiëring, maatvoering e.d. Ook het registreren van voltooide projecten wordt vereenvoudigd. In plaats van registratie over een groot aantal cost drivers, kan deze registratie beperkt blijven tot de meest dominante. Een volgend voordeel heeft betrekking op de beheersing van het project. Deze kan gerichter zijn. Het is immers eenvoudiger voor bijvoorbeeld vijf factoren schattingen te maken over de invloed op kosten, inspanning en doorlooptijd, het effect ervan te meten en zonodig bij te sturen. Doordat bekend is welke factoren echt kostendominant zijn, weet men beter waar de aandacht bij beheersing op gericht moet zijn.

In dit onderzoek is een overzicht gegeven welke factoren een belangrijke invloed hebben op ontwikkelkosten, -tijd en -inspanning. Er is echter geen antwoord gegeven op vragen als: op welke wijze kan een organisatie de meest dominante cost drivers achterhalen; verschilt deze verzameling cost drivers per fase; hoe betrouwbaar zijn begrotingen waarbij rekening is gehouden met alleen de meest dominante cost drivers ten opzichte van begrotingen waarbij deze aanpak niet is gevolgd. En als logische volgende vraag: wegen de voordelen van het verzamelen en registreren van een beperkte verzameling gegevens op tegen waarschijnlijk minder nauwkeurige begrotingen.

aanbeveling 9: spreek één taal.

Om met succes software-ontwikkeling te kunnen begroten en beheersen is het belangrijk binnen een organisatie dezelfde taal te spreken, hetzelfde begrippenstelsel te hanteren en op een uniforme wijze te begroten, te registreren en begrotingsmodellen toe te passen. Er

dienen ondermeer afspraken gemaakt te worden over wat wordt begroot, welke zaken in een begroting worden betrokken, welke kostenbepalende factoren in de begroting meegenomen worden, hoe deze worden geoperationaliseerd, hoe vaak en op welke mate van detail wordt begroot, enz. In deze studie is meerdere malen gewezen op de noodzaak hiervan en is in hoofdstuk 7 een eerste stap in die richting gezet.
Het voordeel van het gebruik van een begrotingsmodel is dat het de gebruiker dwingt zich te conformeren aan de modeldefinities en de in het model gehanteerde maatvoering. Het is echter jammer dat begrotingsmodellen zich daarbij niet richten naar bestaande projectbeheersingsmethoden. Gezien de onlosmakelijke relatie tussen begroten en beheersen, zou men verwachten dat een begrotingsmodel een integraal onderdeel zou zijn van beheersingsmethode. Dit blijkt nauwelijks het geval te zijn.

Het onderwerp begroten neemt vooralsnog een ondergeschikte plaats in bij het beheersen van software-ontwikkeling. Dit blijkt ondermeer uit het beperkt aantal publicaties over dit onderwerp, uit de beperkte ruimte die in projectbeheersingsmethoden als SDM, PRODOSTA, PARAET en dergelijke voor begroten is ingeruimd en uit het nagenoeg ontbreken van een begrotingsmodule in de bestaande Projectmanagers-workbenches. Nader onderzoek naar een integratie van begroten en beheersen is noodzakelijk. In dit onderzoek, met name in hoofdstuk 8, is het belang van deze samenhang onderstreept. Een nadere uitwerking is evenwel gewenst.

aanbeveling 10: keep it small.

Zowel uit de literatuur als uit de enqueteresultaten in deze studie komt naar voren dat de problemen bij het beheersen en begroten van software-ontwikkeling het grootst zijn bij "omvangrijke" projecten. De berichten over forse budget- en doorlooptijdoverschrijdingen hebben doorgaans betrekking op dit soort projecten. Deze constatering in de praktijk van software-ontwikkeling leidt ertoe dat men de volgende vuistregel hanteert: als men schat dat de doorlooptijd van een project langer dan één jaar wordt, splits dan het project dan op in meerdere projecten met ieder afzonderlijk een doorlooptijd van minder dan één jaar. De achterliggende gedachte is dat kleine projecten beter beheersbaar zijn dan grote en dat het zaak is grote projecten op te splitsen in kleinere. In deze studie is gewezen op de incrementele ontwikkelstrategie als mogelijkheid software-ontwikkeling te decomponeren.

Door Pels (1988) zijn richtlijnen voor decompositie aangereikt. Onderzocht moet worden of deze richtlijnen geschikte zijn voor het

beheersen en begroten van software-ontwikkeling.
Tot zover een aantal aanbevelingen voor verder onderzoek. We hebben in deze studie een pleidooi gehouden voor meer aandacht voor het beheersen en begroten van softwareprojecten. Zowel uit de praktijk als uit de theorie komt naar voren dat de stand van zaken met betrekking tot deze onderwerpen nogal mager is, lettende op het belang van software-ontwikkeling. Deze studie is bedoeld om die stand van zaken beter in beeld te brengen en aanzetten te geven voor verbeteringen.

In hoofdstuk 1 heb ik de actualiteit van de perikelen rond de automatisering van de studiefinanciering als voorbeeld gebruikt van een uit de hand gelopen project. De overschrijding van het tijdstip van levering is in de media breed uitgemeten. Hoewel dit project het denken over beheersen en begroten van software-ontwikkeling een impuls heeft gegeven, is er nog een lange weg te gaan, getuige het onderstaande krantebericht van 28 februari 1989.

Eigenrichting per overheidscomputer

MET ENIG VERTOON presenteerde het ministerie van Volkshuisvesting en ruimtelijke ordening (VROM) gisteren in Zoetermeer het nieuwe computersysteem voor de huursubsidies. De boodschap was duidelijk: bij VROM géén rampen in de trant van studiefianciering of paspoort. Overheidsautomatisering komt ook wel eens goed uit de verf. Wel is het systeem tweeëneenhalf maal zo duur uitgevallen als was begroot. Maar in de hele automatisering is budgetoverschrijding eerder regel dan uitzondering, zo bleek vorig jaar uit een enquête.

Figuur 9.1 : Een voorbeeld van een uit de hand gelopen project (NRC Handelsblad, 28 februari 1989).

SUMMARY

THE ESTIMATION AND CONTROL OF SOFTWARE DEVELOPMENT COSTS

The central subject in this thesis is the estimation and control of software development costs. It has been prompted by the sharp overshoots of budgets and delivery times which have been signalled in the development of software. Judging by reports from everyday practice and findings in the literature, it regularly happens that software projects get out of hand. This study falls into three parts.

In the first part, the emphasis is on exploring the subject of the research, namely the estimation of software development costs. On the basis of a field study and an inventory of existing theories on estimating and the methods used for this, an overview is presented of the present state of the art. For the time being, the resulting picture shows a number of blank spaces on the "estimation map". The subject is still in its infancy, although there are sufficient reasons to allow it to take its place as a fully-fledged branch within the discipline of software development. Extensive research in this field is required for this.

The second part of the study is concerned with ranking the estimation models by describing and evaluating the most important of these. This was done with the aid of an evaluation method developed by the research group concerned with the "Estimation of Automation Projects" at Eindhoven University of Technology. Important conclusions from the evaluations carried out are as follows:

- For the time being there is a lack of empirical material which would enable statements to be made about the accuracy of the models.
- Most models offer few possibilities for calibration with data from

completed projects.
- In the models used there is a clear tendency to estimate the size of the software by using an approach based on Function Point Analysis.
- The most recently developed models are implemented in an automated version. These estimation packages are generally easy to use, can be quickly applied and offer possibilities for performing sensitivity and risk analyses.

In addition, a ranking has been made of important factors which influence the costs, effort and lead time involved in software development. An analysis of these factors shows that they relate to one of the following five aspects:

- the product to be developed
- the development staff
- the tools
- the project organisation
- the user.

In estimating and controlling the costs of software development it is important for the project management to have an insight into the factors which determine the costs and the extent of their influence. The greater this insight, the better one is able to control the development and estimate the costs. One of the most important conclusions which emerges from the analysis of the cost-determining factors is that it is generally not easy to operationalise these factors and indicate their influence on costs and lead time.
Another point which emerges from the analysis is that there can be no question of "the most important cost-determining factors". The factors which are important and their influence on the costs is determined by the characteristics of the development situation concerned.

By inventorying and ranking the models, ideas and theories have been developed which constitute a basis for improving the control and estimation of software development costs. These are described in the third part of the study.
A typology has been developed for controlling and estimating software which is largely based on the P-B-I model (Bemelmans 1987). In the typology, a distinction is made between four different control situations. Which of these applies to a specific software development situation is determined by the characteristics of the software to be developed, the process to be implemented and the resources required. The characteristic difference between the four control situations lies

in the degree of certainty which exists about the product, the process and the resources.
If there is a high degree of certainty, controlling becomes a matter of routine. The emphasis is on implementation. An effort is made to maximise efficiency and minimise lead time. Since certainty exists about the product to be developed, the aim will be to achieve this by using resources as optimally as possible. In a situation like this, an estimate has a task-setting and monitoring function.
The situation is entirely different when there is great uncertainty; then controlling assumes the nature of a design task. Since the product, process and resources are uncertain factors, the aim will to be reduce the risks. One possibility is to establish what the resources are and maximise the final result using these as givens. Risks can also be reduced by developing the software according to an incremental approach and incorporating many decision points in the development path. In this kind of situation, controlling and estimating are based on mutual adjustment and commitment. In the case of great uncertainty, an estimate cannot have a task-setting function; rather, it has a condition-creating function. In particular, there will be a need for risk analyses when making the estimate.

A conclusion of this study is the realisation that data on completed software projects are needed for controlling and estimating. The existing estimation methods and models are based on a collection of data of this kind. Although the importance of historical project data is generally recognised, it is striking to note how few project data are collected and recorded systematically within organisations. This study presents a first attempt at indicating the type of data from completed projects which must be recorded for those control situations in which high product certainty exists and the way in which these data can be used for controlling and estimating.

LITERATUURVERWIJZINGEN

Abdel-Hamid, T.K. "Understanding the '90% syndrome' in software project management: A simulation-based case study." The journal of systems and software, 8, 1988.

Abdel-Hamid, T.K., en Madnick, S.E. "On the portability of quantitative software estimation models." Information and Management, 13, 1-10, 1987.

Albrecht, A.J., en Gaffney, J.E. "Software Function, source lines of code, and development effort prediction: a software science validation." IEEE transactions on software engineering, volume SE-9, no. 6, 1983.

Albrecht, A.J. "Measuring application development productivity." The proceedings of the Joint SHARE/GUIDE/IBM application development symposium, oktober 1979.

Albrecht, A.J. "AD/M Productivity Measurement and Estimate Validation." IBM Guideline, 1984.

Alter, S. "Development patterns for decision support systems." MIS quartely, september 1978.

Aron, J.D. "Estimating resources for large scale programming systems." NATO science committee, Rome Italie, oktober 1969.

Bailey, J.W., en Basili, V.R. "A meta-model for software development resource expenditures." Proceedings of the 5th international conference on software engineering, IEEE 1981.

Basili, V.R. "Product Metrics". In: Tutorial on models and metrics for software management and Engineering, editor Basili, V.R., IEEE, 1980.

Baker, B.N. "The future Project Management Revisited." INTERNET, 1985.

Bartelink, R.J. "praktijkervaringen met 4e generatie tools en het opzetten en uitbouwen van een informatiecentrum." Proceedings van de themadag vierde generatietools, CCAA, Lelystad, oktober 1988.

Before You Leap. User's Guide, Gordon Group, 1986.

Bemelmans, T.M.A., en de Boer, J.G. "Ontwikkelingsmethoden 1: het ontwikkelen van informatiesystemen." Informatie, jaargang 23, nr. 1, februari 1981.

Bemelmans, T.M.A. "Ontwikkelingsmethoden voor informatiesystemen: onopgeloste problemen." Informatie, jrg. 26, 1984.

Bemelmans, T.M.A., van der Pool, J.A., en Zwaneveld, N.J.M. Poly automatiseringszakboekje, PBNA-uitgave, 2e druk, 1984.

Bemelmans, T.M.A. "Bedrijfskundig ontwerpen van bestuurlijke informatiesystemen." In: Automatisering met een menselijkgezicht, red. Cornelis, P.A., en van Oorschot, J.M., Kluwer, Deventer, 1986.

Bemelmans, T.M.A. Bestuurlijke informatiesystemen en automatisering,derde druk, Stenfert Kroese 1987.

Birrell, N.D., en Ould, M.A. A practical handbook for software development, Cambridge University Press, 1985.

BIS/Estimator. User Manual version 4.4, BIS applied System Ltd. 1987.

Black, R.K.D., Curnow, R.P., Katz, R., en Gray, M.D. "BCS Software Production data." Final technical report, RADC-TR-77-116, Boeing computer services Inc., maart 1977.

Blokdijk, A., en Blokdijk, P. Planning and design of information systems, Academic Press, 1987.

Boehm, B.W., Holmes, C.E., Katkus, G.R., Romanos, J.P., McHenry, R.C., en Gordon, E.K. "structured programming: A quantitative assessment." Computer, juni 1975.

Boehm, B.W. Software engineering economics, Prentice Hall 1981.

Boehm, B.W. "Seven principles of software engineering." The journal of systems and software, no. 3, 1983.

Boehm, B.W. "Software engineering economics." IEEE transactions on software engineering, volume se-10, nr. 1, januari 1984.

Boehm, B.W., Gray, T.E., en Seewaldt, T. "Prototyping versus specifying: a multiproject experiment." IEEE transactions on software engineering, Volume, SE-10, no. 3, mei 1984.

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

Boehm, B.W. "Improving Software Productivity." IEEE Computer, survey & tutorial series, september 1987.

Boehm, B.W., en Standish, T.A. "Software technology in the 1990's: using an evolutionary paradigm." IEEE computer, 1983.

Boehm, B.W. "A spiral model of software development and enhancement." IEEE Computer, mei 1987.

Boehm, B.W., en Papaccio, P.N. "Understanding and Controlling Software Costs." IEEE transactions on software engineering, volume SE-14, no. 10 oktober 1988.

Boehm, B.W. Documentatie van het Seminar: Software cost estimation using COCOMO and ADA COCOMO, SAL, London. 1988a.

Boer de, J.G., en Vonk, R. "Help ..... de informatie analist verzuipt." Informatie, jaargang 26, nr. 12, 1984.

Botter, C.H. Industrie en Organisatie, Kluwer NIVE, 1979.

Brooks, F.B. The Mythical Man-Month, Essays on software engineering, Addison Wesley Publishing Company, London, 1975.

Brooks, F.B. "No Silver Bullet, Essence and Accidents of software engineering." IEEE Computer, april 1987.

Brooks, W.D. "Software technology payoff: some statistical evidence." IBM- FSD, Bethesda, MD, april 1980.

Bruns, B. "Begroten van automatiseringsprojecten." Afstudeerverslag, Faculteit Informatica, Technische Universiteit Eindhoven, 1989.

Carriere, W.M., en Thibodeau, R. "Development of a logistic software cost estimating technique for foreign military sales." GRC Report CR-3-839, 1979.

Cheatham, T.E. "A computer-based project management assistent." IEEE Compcon Fall, 1984.

Chen, E., en Zelkowitz, M.V. "Use of cluster analysis to evaluate engineering methodologies." Proceedings fifth international conference on software engineering, IEEE, maart 1981.

Computer Associates. CA-Estimacs User Guide, Release 5.0, juli 1986.

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

Conte, S.D., Dunsmore, H.E., en Shen, V.Y. Software engineering metrics and models, Benjamin Cummins, 1986.

Cote, V., Bourque, P., Oligny, S., en Rivard, N. "Software Metrics: an overview of recent results." The journal of systems and software, 8, 1988.

Couger, J.D., en Zawacki, R.A. "What motivates DP professionals." Datamation, september 1978.

Craenen, G. "begroten van automatiseringsprojecten." Informatie, jaargang 26, nr. 3, maart 1984.

Crossman, T.D. "Taking the measure of programming productivity." Datamation, volume 25, no. 5, 1979.

Cuelenaere, A.M.E., van Genuchten, M.J.I.M., en Heemstra, F.J. "Een expert systeem voor het gebruik van een software begrotingsmodel." Informatie, jaargang 29, speciaal nummer, november 1987.

Cuelenaere, A.M.E., van Genuchten, M.J.I.M., en Heemstra, F.J. "Calibrating a software cost estimation model: why and how." Information and software technology, volume 29, no. 10, december 1987a.

Daly, E.B., "Organizing for successful software development." Datamation, december 1979.

van Dam, B., Wentink, T., en Zanders, H. Management en automatisering: kansen en bedreigingen, een opinie-onderzoek over de gevolgen van automatisering, Vifka-publicaties, Tilburg, mei 1982.

Davis, G.B. "Strategies for information requirements determination." IBM systems journal, jaargang 21, nr.1, 1982.

Davis, A.M, Bersoff, E.H., en Comer, E.R. "A strategy for comparing alternative software development life cycle models." IEEE transactions on software engineering, volume 14, nr. 10, oktober 1988.

Dawes, B. "Management techniques to control software development." Data Processing, volume 27, nr. 9, november 1985.

Dekker G.J., en Van Den Bosch F.J. "Functional requirements for the development and use of a software cost database." Information and Managament, nr. 6, 1983.

DeMarco, T. Controlling software projects: management, measurement and estimation, Yourdon Press, New York, 1982.

DeMarco, T. "An Algorithm for sizing software products." Performance Evaluation Review, 12, 13-22, 1984.

Demshki, M., Ligett, D., Linn, B., McCluskey, G. en Miller, R. Wicomo tool, Wang institute cost model tool, User's Manual. Sofware product report PUM1 release 1-1-82, 1982.

Druffel, L.E. "Strategies for a DoD Software initiative." CSS DUSD-(RAT), Washington, DC, 1982.

Duquette, J.A., en Bourbon, G.A. "ESD, A computerized model for estimating software life cycle costs." FSD-TR-235, volume 1, april 1978.

Earl, M.J., en Hopwood, A.G. "From management information to information management, in Lucas H. (editor)." The Information Systems Environment, North Holland Publishing Company, Amsterdam, 1980.

Esterling, B. "Software manpower costs: a model." Datamation, maart 1980.

Estimate/1. Documentatie Method/1: Automated Project Estimating Aid. Arther Anderson, 1987.

Faszbender, F.J. Projectbeheersing: wie schrijft, die blijft, Afstudeerverslag, Faculteit Bedrijfskunde, Technische Universiteit Eindhoven, 1988.

Fink, A., en Kosecoff, J. How to conduct surveys: a step-by-step guide, Sage publications, Beverly Hills, California, 1985.

FPA-Proceedings. Conferentie proceedings: FPA in beweging, NGI-SIC, november, 1988.

Frederic, B.C. "A professional model for estimating computer program development costs." Telecote research Inc., 1974.

Freiman, F.R. "The FAST methodology." Journal of parametrics, 1 (2), 1981.

Freiman, F.R., en Park, R.E. "The Price software cost model: RCA government systems division." IEEE, 1979.

Galbraith, J. Designing complex organizations, Addison-Wesley, 1973.

Gane, C., en Sarson, T. Structured Systems Analysis tools and techniques, improved systems technologies inc., New York, 1977.

Gecomo. "Software tools for professionals." Gec software documentation, G & C Company, London, 1985.

Genuchten van, M., en Van Gils, A.C.E. "The application of knowledge-based systems to software cost estimation." Te publiceren, 1989.

Genuchten van, M. "Een techniek voor het analyseren van softwareontwikkeling." Nog te publiceren, 1989.

Goethert, W.B. "SECOMO in: Boehm, B.W. Documentatie van het Seminar: Software cost estimation using COCOMO and ADA COCOMO, SAL, London. 1988a." ITT Research Institute, Data & Analysis Center for software.

Grinwis, P. "Besturen van onderwijsinstellingen." Lezing Studiecentrum voor bedrijf en overheid, 1989.

GUIDE, Inc. "GUIDE survey of new programming technologies." Guide Proceedings, GUIDE, Inc., Chicago, 1979.

Haarhuis, R.H.J. Software ontwikkelen: sneller produceren door beter specificeren, Afstudeerverslag, Faculteit Bedrijfskunde, Technische Universiteit Eindhoven, 1988.

Haas de, B.G.M. "Funktie Punt Analyse in kort bestek." Proceedings van de conferentie FPA in beweging: de praktijk van funktiepunt analyse in Nederland en de Verenigde Staten, NGI-SIC, 21 en 22 november, Scheveningen, 1988a.

Hall, D.L., Gibbons, J.J., en Knell, M. "Quantifying productivity: A software metric database to support realistic cost estimation." Proceedings of the international society of parametric analists, 8th annual conference, volume 5, no. 1, 1986.

Halstead, M.H. Elements of software science, North Holland, 1977.

Heemstra, F.J., van Genuchten, M., en Kusters, R. "An empirical validation of automated tools for software cost estimation." Te publiceren, 1989.

Heemstra, F.J. "Problemen bij het ontwikkelen van software: produktiviteit van programmeurs is laag." De automatiseringsgids, november 1985.

Heemstra, F.J. "Het schatten van softwarekosten: een inventarisatie." Researchrapport Technische Universiteit Eindhoven, 1986.

Heemstra, F.J. "Het begroten van automatiseringsprojecten." Conferentieverslag Stichting Nederlandse Organisatie voor Bedrijfskundige Onderzoek (NOBO), november 1987.

Heemstra, F.J. "Wat bepaalt de kosten software?." Informatie, volume 29, extra editie, pp. 586-601, 1987.

Heemstra, F.J. Het begroten van softwareprojecten: meten is weten!, Report EUT/BDK/28, ISBN 90-6757-028-1, Eindhoven 1987.

Heemstra, F.J., van Genuchten, M.I.J.M., en Kusters, R. "Examination of software cost estimation models." Report no. 1, Theoretical examination, report no. 2, design of practical evaluation, Eindhoven University of Technology, Philips Corporate ISA-TMS, 1988.

Heemstra, F.J. "Begroten van projecten: wat kost automatisering." Beheersing van automatiseringsprojecten, Studiedag, Nederlands Studie Centrum, maart 1988.

Heemstra, F.J., Siskens, W.J.A.M., van der Stelt, H. "Kostenbeheersing van automatiseringsprojecten in de praktijk." De Automatiseringsgids, Back Up, Europe Software '88, nr. 20, 18 mei 1988.

Heemstra, F.J., Siskens, W.J.A.M., van der Stelt, H. "Kostenbeheersing van automatiseringsprojecten; een empirisch onderzoek." Informatie, jaargang 31, nummer 1, 1989.

Heemstra, F.J., Kusters, R. A conceptual Framework for software cost control and estimation, Report EUT/BDK/32, ISBN 90-6757--032-X, 1988.

Heemstra, F.J. "Hoe staat het met de softwarekosten." Proceedings van de conferentie FPA in beweging: de praktijk van funktiepunt analyse in Nederland en de Verenigde Staten, NGI-SIC, 21 en 22 november, Scheveningen, 1988a.

Heemstra, F.J., Kusters, R., en van Genuchten, M.I.J.M. "Verslag van het experiment", Eindhoven University of Technology, Philips Corporate ISA-TMS, 1988a.

Heemstra, F.J., Siskens, W., en van der Stelt, H. "Kostenbeheersing automatiseringsprojecten." Tijdschrift Financieel Management, Kluwer, jaargang 8, april 1988.

Herd, J.R., Postak, J.N., Russell, W.E., en Stewart, K.R. "Software cost estimation study - study results." Final technical report, RADC-TR-77-220, volume 1, DOTY Associates, Inc., Rockville, MD, 1977.

Herrman, O. "Analyse der Einflußfaktoren auf die Kosten von Softwareentwicklungen." Angewandte Informatik, nr. 4, 1983.

Hofstede, G. The game of budget control, Van Gorcum and London: Tavistock, Assen 1967.

Howard, P.C. "Verslag van de Conference on Improving Productivity in EDP System Development (jan. 25-29 in Tuscon, Arizona). " System Development, Volume 8, nr.3, maart 1988.

IEEE. IEEE Guide to Software requirements specifications, published by the institute of electrical and electronics engineers, American national standard, ANSI/IEEE std 830, 1984.

Ives, B. en Olson, M.H. "User involvement and MIS success: A review of research." Management Science, volume 30, no. 5, 1984.

Jeffery, D.R. "A software development Productivity model for MIS Environments." The journal of systems and software, 7, 1987.

Jeffery, D.R., en Lawrence, M.J. "Managing Programming Productivity." The journal of systems and software, 5, 1985.

Jeffery, D.R., en Basili, V.R. "Validating the TAME resource data model." Proceedings of the 12th international conference on software engineering, Singapore, IEEE, 1988.

Jensen, R.W., en Lucas, S. "Sensitivity Analysis of the Jensen Software Model." Proceedings of the fifth ISPA Conference, st Louis, MO, 1983.

Jensen, R.W. "An improved macrolevel Software Development Resource Estimation Model." Proceedings of the fifth ISPA Conference, st. Louis, MO, 1983.

Jensen, R.W. "A comparison of the Jensen and COCOMO Schedule and Cost Estimation Models." Proceedings of the sixth ISPA Conference, San Fransisco, 1984.

Jones, C. "Reusability in programming: A survey of the state of the art." IEEE transactions on software engineering, volume SE-10, no. 5, september 1984.

Jones, C. Programming Productivity, McGraw-Hill, 1986.

de Kater, A.L. "Funktie Punt Analyse: Pruduktiviteitsmeting en budgettering van automatiseringsprojecten." Beleidsinformatica Tijdschrift, volume 11 nr. 2, tweede kwartaal 1985.

Kemerer, C.F. "An empirical validation of software cost estimation models." Communications of the ACM, volume 30, nr. 5, mei 1987.

Kitchenham, B.A., en Taylor, N.R. "Software project development cost estimation." The journal of systems and software, no. 5 1985a.

Kitchenham, B.A., en Smith, M. "Software data library Phase 1 manual software collection." SDM SIG/DP(86), 1985.

Krooshof, R.L., Thackwray, J.D., Brands, J.W., Bates, R., Boutelegier, R. Guide to Project Management - Prodosta, version 3.1, intern rapport Philips n.v. gloeilampenfabrieken, C-ISA-TMS/SM, CFT CAM-centre, Eindhoven, juli 1987.

Kustanowitz, A.L. "System life cycle estimation (SLICE): a new approach to estimating resources for application program development." IEEE first international computer software and application conference, Chicago, 1980.

Lanergan, R.G., Grasso, C.A. "Software Enineering with reusable designs and code." IEEE transactions on software engineering, volume SE-10, no. 5, 1984.

Langerhorst, R.P. "SDM vernieuwd." Informatie, jaargang 28, nr.5, 1986.

Laughery, K.R. jr., en Laughery, K.R. sr. " Human factors in software engineering: a review of the literature." The journal of systems and software, 5, 1985.

Leeuw de, A. Systeemleer en organisatiekunde, Stenfert Kroese, Leiden, 1974.

Lierop van, F., en Volkers, R. Beheersing van softwareprojecten: beter een project in de hand dan tien uit de hand, Afstudeerverslag, faculteit Bedrijfskunde, Technische Universiteit Eindhoven (nog te publiceren), 1989.

Ligett, D. The development of WICOMO, Wang Institute of graduate Studies, Tyngsboro, Massachusetts, 1985.

Londeix, B. Cost Estimation for Software Development, Addison-Wesley Publishing Company, 1987.

Lundeberg, M., Goldkuhl, G., en Nilsson, A. Systeemontwikkeling volgens ISAC, de ISAC-Methodiek, 2de editie, Samsom 1982.

Martin, J., en McClure, C. Software maintenance the problem and its solutions, Prentice-Hall inc., Englewood Cliffs, New Jersey, 1983.

Martin, J. An information manifesto, Englewood Cliffs, New Jersey, 1984.

Martin, J. Fourth-Generation Languages, volume 1 Principles. Prentice-Hall inc., Englewood Cliffs, New Jersey, 1985.

Matsumoto, E.Y. "Approaching productivity and quality in software production - How to manage a software factory." Proceedings of the information technology payoff, managing productivity and risks, Diebold research program-europe, Parijs, mei 1987.

Matsumoto, E.Y. "Some experience in promoting reusable software: presentation in higher abstract levels." IEEE Transactions on software engineering, volume SE-10, no. 5, 1984.

McCabe, T.J. "A complexity measure." IEEE transactions on software engineering, volume SE-2, no. 4, 1976.

McCracken, D.D., en Jackson, M.A. "Life cycle concept considered harmful," Software engineering notes, ACM, april 1982.

McIlroy, M.D. "Mass-produced software components." Proceeding Nato conference on software Engineering, Garmisch 1968.

McKeithen, K.B., Reitman, J.S., Rueter, H.H., en Hirtle, S.C. "Knowledge organization and skill differences in computer programmers." Cognitive Psychology, volume 13, 1981.

Mintzberg, H. Structure in fives: designing effective organizations, Prentice-Hall, inc., Englewood Cliffs, New Jersey, 1983

Miyazaki, Y., en Mori, K. "COCOMO evaluation and tailoring." Proceedings of the 8th international conference on software engineering, IEEE 1985.

Mizuno, Y. "Software quality improvement." IEEE Computer, 1983.

Mohanty, S.N. "Software cost estimation: present and future." Software - practice and experience, 1981.

Musa, J.D., Iannino, A., en Okumoto, K. Software Reliability, measurement, prediction, application, McGraw-Hill international editions, computer science series, 1987.

Musa, J.D., "Software Engineering: The future of a profession." IEEE Software, 1985.

Nelson, E.A. Management handbook for the estimation of computer programming costs, AD-A648750, Systems Development Corporation, 1966.

NGI-wergroep 1986. Werkgroep Funktie-ordening van het Nederlands Genootschap voor Informatica. Funkties in de Automatisering, 1986.

Norden, P.V. "Useful Tools for project management." Operations research in research and development, New York, John Wiley and sons, 1963.

Noth, T., and Kretzschmar, M. Aufwandschätzung von DV-Projekten, Springer Verlag, Berlijn 1984.

Noth, T. "Unterstützung des Softwareprojectmanagements durch eine Erfahrungsdatenbank." Proceedings Compas '87, Erfolgsfaktoren der integrierten Informationsverarbeitung, AMK Berlijn, Mei 1987.

Parnas, D.L. "On the criteria to be used in decomposing systems into modules." Communications of the ACM 15, no. 12, 1978.

Parnas, D.L. "Software-aspecten van strategische defensiesystemen." Informatie, jaargang 28, nr. 3, maart 1986.

Pels, H.J. Geïntegreerde informatiebanken, modulair ontwerp van het conceptuele schema, Stenfert-Kroese, 1988.

Phister, M. Data Processing Technology and Economics, 1985.

Pressman, R.S. Software Engineering; A practitioner's approach, McGraw-Hill international editions, Computer science series, 1987.

PRICE, RCA PRICE Systems. PRICE S/SP manual, 1985.

Price, B.C. Government tailored COCOMO (GTCOCOMO), US Army Electronics Research & Development Command, Forth Monmouth, New York, 1985.

Putnam, L.H., en Fitzsimmons, A. "Estimating software costs." Datamation, sept., okt., nov., 1979.

Putnam, L.H. "A general empirical solution to the macro software sizing and estimating problem." IEEE transactions on software engineering, SE-4, 4 , 1978.

Putnam, L. "Software costing estimating and life cycle control." IEEE computer society press, 1980.

Rabbers, R.K. "Projectmatige invoering van FPA bij PTT Telecommunicatie." Proceedings van de conferentie FPA in beweging: de praktijk van funktiepunt analyse in Nederland en de Verenigde Staten, NGI-SIC, 21 en 22 november, Scheveningen, 1988a.

Rampton, J. "Unique features of the JS-2 system for Software Development Estimation." Proceedings of the sixth ISPA Conference, San Fransisco, 1984.

Reddin, W.J. Managerseffectiviteit, Samsom NIVE, 1973.

Reifer, D.J. "Softcost: A user-friendly software cost estimation tool." AIAA computers in aerospace, V conference, long Beach, CA, 1985.

Reifer, D.J. "A poor man's guide to estimating software costs." Texas instruments inc. technical report RC1-TR-012, revised, november 1, 1985.

Rooks, P. GECOMO: an implementation of extended COCOMO, GEC software limited, London, 1985a.

Rooks, P. "Meeting the bottom line." Datalink, VNU Business Publications, 1985.

Royce, W.E. PCOC: a complete, user tailored, interactive cost analysis tool based on COCOMO, Electric Systems, Torrance California, 1985.

Royce, W.E. "ADA COCOMO overview." Proceedings of the second COCOMO users' group meeting, Wang institute 1988.

Rosenberger, R.B. "The Information Center." SHARE proceedings, nr. 56, maart 1981.

Rubin, H.A. "Macro and micro-estimation of maintenance effort: the estimacs maintenance models." IEEE, 1984.

Rubin, H.A. "Interactive macro-estimation of software life cycle parameters via personal computer: a technique for improving customer/developer communication." Proceedings of the symposium on application & assesment of automated tools for software development, IEEE, San Fransisco, 1983.

Rubin, H.A. "A comparison of cost estimation tools." Proceedings of the 8th international conference on software engineering, IEEE, 1985.

Rudolph, E.E. "Productivity in Computer Application Development, Department of Management studies." Working paper no 9, University of Auckland, maart 1983.

Rudolph, E.E. "Function Point Analyses, Cookbook." eigen uitgave van Rudolph, 3/1983a.

Saalfrank, R.F., Schelle, H., en Schnopp, R. "Produktivitäts-Effekte von Aufwandeinflußgroße bei der Softwareentwicklung." Angewandte Informatik, nr. 3, 1987.

Sandler, G., en Bachowitz, B. "Software cost models - Grumman experience." Quantitative software model conference IEEE, 1979.

Setzer, R. "Spacecraft software cost estimation: striving for excellence through parametric models (a review)." The proceedings of the ISPA 8th anual conference, 1986.

Sierevelt, H. Report on ISPA 1985 conference, COCOMO conference, Interne Philips publicatie, Corporate TEO, Eindhoven, 1985.

Sierevelt, H. "Observations on software models." Journal of parametrics, Volume VI, no. 4, december 1986.

Sierevelt, H. "Price-SP." Voordracht op de Technische Universiteit Eindhoven, september 1986.

Sikkel, K., en van Vliet, J.C. "Kan software langer mee? Een overzicht van hergebruik van software." Informatie, jaargang 30, nr.7/8, juli/augustus 1988.

Silverman, B.G. "Expert intuition and ill-structured problem solving." IEEE Transactions on Systems, Management and Cybernetics, no. 1, 1985.

Siskens, W.J.A.M. Kostenbeheersing van automatiseringsprojecten, afstudeerverslag, Faculteit Bedrijfskunde, Technische Universiteit Eindhoven, mei 1988.

Stanley, M. "Software cost estimating." The proceedings of the ISPA 6th annual conference, 1984.

Stehouwer, T.L. "Omgaan met Adviesfunctie." Conferentieverslag beheersing van Automatiseringsprojecten, Nederlands Studie Centrum, Amsterdam, 2-3-1988.

Van de Stelt, H. "Automatiseringskosten: beheersing vereist inzicht." Kantoor en efficiency, september 1987.

Surbock, E.K. Management von EDV-Projekten, Berlin 1978.

van Straten, R. "Functie Punt Analyse: theorie, praktijk en resultaten." Informatie, jaargang 29, extra editie begroten, 1987.

van Straten, R. "Begroten met Functie Punt Analyse." Conferentie beheersing van Automatiseringsprojecten, Nederlands studie Centrum, Amsterdam, 2 maart 1988.

Strausz, I. "De kwaliteit van software." Informatie, jaargang 25, nr. 12, december 1983.

Swanson, E.B. "Management Information Systems: Appreciation and involvement." Management Science, Volume 21, no. 2, 1974.

Symons, C.R. "Function Point Analysis: Difficulties and Improvements." IEEE transactions on software engineering, volume 14, no. 1, januari 1988.

Tait, P., en Vessey, I. "The effect of user involvement on system success: a contingency approach." MIS quarterly, maart 1988.

Tausworthe, R.C. "The work-breakdown-structure in software project management." The journal of systems and software, 1, 1980.

Tausworthe, R.C. "Deep space network software cost estimation model." Publication 81-7, Jet Propulsion laboratory, Pasadena, CA, 1981.

Thebaut, S.M., en Shen, V.Y. "An analytic resource model for large-scale software development." Information processing and management, 20, 1-2, 1984.

Theeuwes, J.A.M. "Budgettering: stuurinstrument of administratieve rompslomp ?" Bedrijfskunde, nr.1, 1987.

Thibodeau, R. "An evaluation of software cost estimating models." RADC-TR- 81-144, 1981.

Thibodeau, R., en Dodson, E.N. "Life cycle phase interrelationships." The journal of systems and software, 1981.

Turner, W.S., Langerhorst, R.P., Eilers, H.B., Hice, G.F., Castermans, R.J.M.M., Cashwell, L.F., Uijttenbroek, A.A. (editor). Een samenvatting van de System Development Methodology, voorlopige versie, Pandata, Rijswijk 1985.

Veld in 't, J. Organisatiestructuur en arbeidsplaats; de organisatie van mensen en middelen; theorie en praktijk, Elsevier, 1981.

Ven van der, H. "Fourth-generation environments give structural cost reduction." VISA, 88/1 februari 1988.

Verberne, J.B. "Vierde generatietools en de EDP-Professional." Proceedings van de themadag vierde generatietools, CCAA, Lelystad, oktober 1988.

Verner, J.M., en Tate, G. "A model for Software Sizing." The journal of systems and software, 7, 1987.

Verner, J.M., en Tate, G. "Estimating Size and effort in fourth-generation development." IEEE Software, juli 1988.

Visser, H. "Schatting van ontwikkelingskosten van software, evaluatie van een eerste toepassing van WICOMO." Informatie, jaargang 29, extra editie begroten, 1987.

Vliet van, J.C. "Wat kost dat nou, zo'n programma." Informatie, jaargang 29, extra editie, 1987.

Vliet van, J.C. Software Engineering, Stenfert Kroese, 1988.

Vliet van, J.C. Over kwaliteit gesproken, Samson uitgeverij, Open universiteit, vrije universiteit, 1987a.

Vliet van, J.C., Heemstra, F.J. Wat kost dat nu, zo'n programma ?, Informatiemanagement, Kluwer Bedrijfswetenschappen, Deventer, 1988.

Vonk, R. "Prototyping - concepten en richtlijnen." Informatie, jaargang 27, nr. 1, 1985.

Vonk, R. Prototyping van informatiesystemen, Academic Service, 1987.

Vonk, R. "Function Point Analysis, in the context of software cost estimation and development productivity measurement." UDR-TM-S/SM-85/001, Philips, 1985.

Walston, C.E., en Felix, C.P. "A method of programming measurement and estimating." IBM system journal, 16, 1977.

Wijnen, G., Storm, P., en Renes, W. Projectmatig werken, Marka paperback reeks, 1984.

Willmer, H. Systematische Software Qualitätsicherung anhand von Qualitäts- und Produktmodellen, Springer-Verlag, Informatikfachberichte, 1985.

Wingfield, C.G. "USACSC experience with SLIM." Report IAWAR 360-5, US army institute for research in management information and computer science, Atlante, GA, 1982.

Winkelhage, F., en Marock, J. "User participation in information system design." In: The information systems environment. red.: Lucas, H. e.a., North Holland Publ. Co., Amsterdam/New York, 1980.

Wolverton, R.W. "The cost of development large-scale software." IEEE transactions on computers, volume c-23, nr. 6, juni 1974.

Yourdon, E. Economics of software productivity, Yourdon inc., version 1.0, 1987.

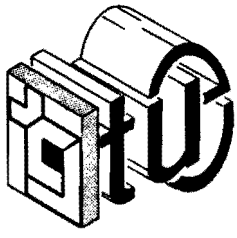
Yourdon, E., en Constantine, L.L. Structured Design: Fundamentals of a discipline of computer program and systems design, Prentice-Hall inc., Englewood Cliffs, N.Y., 1979.

Yu, T.J., Nejmeh, B.A., Dunsmore, H.E., en Shen, V.Y. "SMDC: an interactive software metrics data collection and analysis system." The journal of systems and software, 8, 1988.

Zelkowitz, M.V., Shaw, A.C., en Gannon, J.D. Principles of software engineering and design, Prentice-hall inc., Englewood Cliffs, New Jersey, 1979.

BIJLAGE

"Kostenbeheersing van automatiseringsprojecten"



ENQUETE "KOSTENBEHEERSING VAN AUTOMATISERINGSPROJECTEN"

december 1987

ENQUÊTE "KOSTENBEHEERSING VAN AUTOMATISERINGSPROJECTEN"

Toelichting

De meeste vragen zijn gesteld in een meerkeuzevorm. Dit om niet onnodig veel van uw tijd in beslag te nemen. De bedoeling is dat u slechts één antwoord aankruist, tenzij anders staat aangegeven. Tenslotte wil ik u erop wijzen dat de vragen gericht zijn op de automatiseringsprojecten die de laatste jaren binnen uw bedrijf zijn uitgevoerd of nog in uitvoering zijn.

Deze enquête bestaat uit de volgende onderdelen:

- positionering
- begroting : • algemeen
  • registreren
  • begrotingsmodellen
  • vierde generatie talen
- nacalculatie
- ervaringen

POSITIONERING

1. Op welk terrein bent u hoofdzakelijk werkzaam?
   - ☐ Controlling/Administratie
   - ☐ Beheer hardware en systeemsoftware
   - ☐ Systeemanalyse/programmering
   - ☐ Informatieanalyse
   - ☐ Projectmanagement
   - ☐ Overige, t.w .....

2. Wat is hoofdzakelijk de aard van uw werkzaamheden?
   - ☐ Leidinggevend
   - ☐ Adviserend
   - ☐ Uitvoerend
   - ☐ Overige, t.w. .....

3. Tot welke branche behoort het bedrijf waar u werkzaam bent?
   - ☐ Landbouw en visserij
   - ☐ Delfstoffenwinning
   - ☐ Industrie
   - ☐ Openbare nutsbedrijven
   - ☐ Bouwnijverheid en installatiebedrijven
   - ☐ Handel, hotel- en restaurantwezen, reparatiebedrijven voor gebruiksgoederen
   - ☐ Transport-, opslag- en communicatiebedrijven
   - ☐ Bank- en verzekeringswezen, zakelijke dienstverlening
   - ☐ Softwarehuizen/systeemhuizen
   - ☐ Overheid en semi-overheid, onderwijs en verenigingen
   - ☐ Overige dienstverlening (non profit)

4. Hoe groot is het personeelsbestand van het bedrijf waar u werkzaam ben
   - ☐ < 20 personen
   - ☐ 20 t/m 49 personen
   - ☐ 50 t/m 99 personen
   - ☐ 100 t/m 199 personen
   - ☐ 200 t/m 499 personen
   - ☐ > 500 personen

5. Hoeveel personeelsleden hebben een functie die primair met automatisering te maken heeft?
   - ☐ < 2 medewerkers
   - ☐ 2 t/m 9 medewerkers
   - ☐ 10 t/m 19 medewerkers
   - ☐ > 20 medewerkers

In het hiernavolgende enquêtegedeelte onderscheiden we 4 categorieën projecten, al naar gelang de omvang, gemeten in mensmaanden inzet

Kleine projecten	:	< 12 mensmaanden
Middelgrote projecten	:	12 t/m 48 mensmaanden
Grote projecten	:	49 t/m 200 mensmaanden
Zeer grote projecten	:	> 200 mensmaanden

	Omvang automatiseringsprojecten				
	klein	middel-groot	groot	zeer groot	totaal
6. Hoe is de inspanning in mensmaanden verdeeld over genoemde categorieën projecten?	%	%	%	%	100 %

7. Kunt u in procenten aangeven hoe de ontwikkelde software verdeeld is over de volgende toepassingen?	
Administratieve toepassingen (bijv. salarisberekening, grootboek, voorraadadministratie, urenverantwoording)	80,5 %
Technische toepassingen (bijv. procesbesturing, geautomatiseerde magazijnen)	...%
Kantoor/secretariële toepassingen (bijv. tekstverwerking, agendabeheer, archivering)	...%
Management/professionele toepassingen (bijv. planning, budgettering, financiële analyse, computer-aided design)	...%
Overige toepassingen (bijv. systeemsoftware)	...%
	100%

De hiernavolgende vragen hebben betrekking op het toepassingsgebied, waarvoor de meeste software wordt ontwikkeld

8. Voor welke doelgroepen wordt software ontwikkeld?	
Persoonlijk computergebruik	...%
Het eigen bedrijf	...%
De overheid	...%
Militaire instanties	...%
Overige organisaties	...%
	100%

9. Hoe is de inspanning in mensmaanden verdeeld over nieuwbouw en onderhoud?	
Nieuwbouw	...%
Onderhoud	...%
	100%

10. Wordt er een geldbudget voor automatiserings-projecten opgesteld?

☐ Ja
☐ Nee (ga verder met vraag 12)

11. Indien overschrijdingen van het oorspronkelijke geldbudget plaatsvinden, hoe groot zijn deze gemiddeld, verdeeld over automatiseringsprojecten van de volgende omvang?
(S.v.p. per kolom eenmaal aankruisen)

	Omvang automatiseringsprojecten			
	klein	middel-groot	groot	zeer groot
Geen overschrijding				
< 10%				
10 t/m 50%				
50 t/m 100%				
> 100%				

12. Indien overschrijdingen van de oorspronkelijk geplande doorlooptijd plaatsvinden, hoe groot zijn deze gemiddeld verdeeld over projecten van de volgende omvang?
(S.v.p. per kolom eenmaal aankruisen)

	Omvang automatiseringsprojecten			
	klein	middel-groot	groot	zeer groot
Geen overschrijding				
< 10%				
10 t/m 50%				
50 t/m 100%				
> 100%				

BEGROTEN

algemeen

13. Wordt voor een automatiseringsproject een begroting gemaakt?

☐ Ja
☐ Nee (ga verder met vraag 22)

14. Bij welke automatiseringsprojecten, van de volgende omvang, wordt een begroting gemaakt?

Omvang automatiseringsprojecten			
klein	middel-groot	groot	zeer groot

15. Welke zaken worden betrokken bij het opstellen van een begroting?
(Meer dan één antwoord mogelijk)

☐ Het benodigde budget in geld
☐ De geplande doorlooptijd
☐ De totaal benodigde inzet in mensjaren
☐ Het benodigde personeel per categorie (informatie-analisten, systeemanalisten, programmeurs, etc.)
☐ De benodigde hoeveelheid hardwaremiddelen
☐ De benodigde hoeveelheid softwarehulpmiddelen
☐ De overhead (reiskosten, huur gebouw e.d.)
☐ Overige, t.w .....
.....
.....

16. Op welk niveau van detaillering wordt voor automatiseringsprojecten van de volgende omvang een begroting opgesteld? (S.v.p. per kolom eenmaal aankruisen)

	Omvang automatiseringsprojecten			
	klein	middel-groot	groot	zeer groot
Het totale project				
De fasen van het project				
De activiteiten binnen de fasen van het project				

17. Wie zijn betrokken bij het opstellen van een begroting? (Meer dan één antwoord mogelijk)

- ☐ Het management
- ☐ De afdeling controlling/administratie
- ☐ Het ontwikkelteam
- ☐ De projectbegroter (aparte functionaris)
- ☐ De projectleider
- ☐ De klant/opdrachtgever
- ☐ Iemand anders, t.w .....
  .....
  .....

18. Wanneer wordt voor automatiserings-projecten van de volgende omvang de eerste maal een begroting opgesteld? (S.v.p. per kolom eenmaal aankruisen)

	Omvang automatiseringsprojecten			
	klein	middel-groot	groot	zeer groot
Na een eerste globale omschrijving				
Na het opstellen van de specificaties				
Na de afronding van het ontwerp				

19. Hoe vaak wordt gedurende automatiseringsprojecten van de volgende omvang een begroting opgesteld c.q. bijgesteld? (S.v.p. per kolom eenmaal aankruisen)

	Omvang automatiseringsprojecten			
	klein	middel-groot	groot	zeer groot
Nooit				
1 maal				
2 maal				
3 - 5 maal				
> 5 maal				

20. Wordt de begroting als instrument voor de voortgangscontrole gebruikt?

- ☐ Ja
- ☐ Nee

21. Welke van de volgende methoden hanteert u bij het opstellen van een begroting? (Meer dan één antwoord mogelijk)

- ☐ Inschakelen van een expert op dit gebied
- ☐ Gebruik maken van project c.q. begrotingsgegevens van soortgelijke projecten uit het verleden
- ☐ Gebruik maken van begrotingsmodellen
- ☐ Begroting (alleen) laten afhangen van commerciële motieven
- ☐ Begroten zien als een capaciteitsvraagstuk: hoeveel mensen kunnen we vrijmaken voor een project
- ☐ Intuïtie en ervaring
- ☐ Overige, t.w. .....

registreren

22. Vindt registratie van projectgegevens plaats?

- ☐ Ja
- ☐ Nee (ga verder met vraag 24)

23. Wilt u d.m.v. een letter per hokje aangeven welke gegevens van automatiseringsprojecten van genoemde omvang geregistreerd worden?
De letters geven een indicatie van de frequentie van registratie:

E = eenmalig, aan het einde van het project
M = meerdere malen, gedurende het hele project
N = nooit, deze gegevens worden niet geregistreerd

	Omvang automatiseringsprojecten			
	klein	middel-groot	groot	zeer groot
Besteed budget in geld				
Doorlooptijd				
Personeelsinzet				
Personeelskwaliteit				
Gebruikte apparatuur, tools				
Verdeling middelen naar fase				
Verdeling middelen naar activiteiten				
Omvang van de software				
Kwaliteit van de software				
Hergebruik van de software				
Oorzaken van afwijkingen in tijd/kosten				
Overige, t.w.				

begrotingsmodellen

24. Welk begrotingsmodel gebruikt u? (Meer dan één antwoord mogelijk)

- ☐ Geen (ga verder met vraag 27)
- ☐ COCOMO of een variant zoals WICOMO, GECOMO, enz.
- ☐ ESTIMACS
- ☐ Functie Punt Analyse
- ☐ PRICE-S
- ☐ SLIM
- ☐ SPQR
- ☐ Andere t.w. .....

	Omvang automatiseringsprojecten			
	klein	middel-groot	groot	zeer groot
25. Bij hoeveel procent van automatiseringsprojecten van de volgende omvang maakt u gebruik van een begrotingsmodel?	%	%	%	%

26. Welke twee problemen ervaart u als de meest belangrijke bij het opstellen van een begroting met gebruik van een model?

- ☐ Gebrek aan eenduidige definities
- ☐ Subjectiviteit bij inschalen van kostenbepalende factoren
- ☐ Keuze van een bepaald model bij een bepaald project
- ☐ Afstemming van het model op de organisatie (calibreren)
- ☐ Onvoldoende inzicht in het project om model-parameters te kunnen invullen
- ☐ Bepalen van de omvang van het softwareproject (aantal coderegels of functiepunten)
- ☐ Overige, t.w .....
  .....
  .....

Vierde generatie talen

27. Bij hoeveel procent van de projecten wordt gebruik gemaakt van vierde generatie talen voor het ontwikkelen van software?

- ☐ 0% (ga verder met vraag 29)
- ☐ 1 - 9%
- ☐ 10 - 49%
- ☐ 50 - 89%
- ☐ 90 - 100%
- ☐ 100%

28. Wat is volgens u het belangrijkste effect van de vierde generatie talen?

- ☐ Verlaging van de onderhoudskosten
- ☐ Verlaging van de ontwikkelingskosten
- ☐ Verkorting van de projectdoorlooptijd
- ☐ Overige,t.w .....
  .....
  .....

NACALCULATIE

29. Vindt er na afloop van een automatiseringsproject nacalculatie plaats?

- ☐ Ja
- ☐ Nee (ga verder met vraag 31)

30 Welke zaken worden betrokken bij de nacalculatie?
(meer dan één antwoord mogelijk)

- ☐ Besteed budget in geld
- ☐ Personeelsinzet
- ☐ Gebruikte apparatuur, tools
- ☐ Overhead (reiskosten, huur gebouw e.d.)
- ☐ Overige, t.w. .....
  .....
  .....

ERVARINGEN

31. Welke vijf factoren hebben naar uw inzicht de grootste invloed op de omvang van de kosten van een automatiseringsproject? (De meest belangrijke factor geeft u score 1, de daarna volgende factor score 2, etc., t/m score 5)

_____ De omvang van de te ontwikkelen software
_____ De kwaliteitseisen gesteld aan de software
_____ De complexiteit van de software
_____ De vereiste documentatie
_____ De mate van hergebruik
_____ Het gebruik van software-ontwikkelgereedschap (tools)
_____ De beperkingen van de aanwezige hardware
_____ De prestatie-eisen gesteld aan de nieuw aan te schaffen hardware
_____ Het gebruik van moderne programmeertechnieken
_____ De kwaliteit van het personeel
_____ De ervaring van het personeel
_____ Het verloop van personeel
_____ De kwaliteit van het management
_____ De eisen die gesteld worden aan de project-doorlooptijd
_____ De mate van gebruikersparticipatie bij de ontwikkeling van de software
_____ Het aantal verschillende gebruikers
_____ Het veranderen van de specificaties tijdens de ontwikkeling van de software
_____ De scholing en opleiding van de gebruikers

32. Hebben deze factoren eveneens de grootste invloed op de opbrengsten van een automatiseringsproject?

☐ Ja
☐ Nee, dat zijn andere t.w.
.....
.....
.....

33. Welke twee van de volgende punten zullen naar uw inzicht het grootste effect op de verbetering van de kostenbeheersing hebben en in het bijzonder op het begroten van automatiseringsprojecten?

☐ Meer discipline in relatie met klant/opdrachtgever
☐ Beter personeel
☐ Expliciete toekenning van verantwoordelijkheden en bevoegdheden aan projectmedewerkers
☐ Meer betrokkenheid management
☐ Uitvoerige registratie/bewaking van software-projecten
☐ Gebruik van begrotingsmodellen
☐ Hergebruik van bestaande ontwerpen, code e.d.
☐ Gebruik van vierde generatie hulpmiddelen
☐ Gebruik van expertsystemen en Kunstmatige Intelligentie-technieken
☐ Meer onderzoek op dit gebied
☐ Overige, t.w. .....
.....
.....

INDEX

CURRICULUM VITAE

De auteur van dit proefschrift werd op 14 februari 1950 geboren te Heerlen. In 1970 behaalde hij het diploma Gymnasium-B aan het St-Janscollege te Hoensbroek. In 1971 begon hij zijn studie aan de Technische Universiteit Eindhoven, in de faculteit Bedrijfskunde. Het ingenieursdiploma werd in 1976 behaald. Het afstudeeronderzoek lag op het gebied van de statistische kwaliteitsbeheersing. Van 1976 tot 1985 was hij verbonden aan het Instituut voor Hoger Beroepsonderwijs te Breda, afdeling Hoger Economisch en Administratief onderwijs. Voor een deel van zijn tijd was hij belast met het onderwijs in de informatica en voor het andere deel met onderwijsorganisatie (voorzitter vakgroep Automatisering, leerplancommissie, dagelijks bestuur). Vanaf 1983 tot 1986 is hij tevens mededirecteur geweest van het automatiseringsadviesbureau Infologic. Sinds 1 maart 1985 is hij als universitair docent in dienst bij de faculteit Bedrijfskunde aan de Technische Universiteit Eindhoven, in de vakgroep Bestuurlijke Informatiesystemen en Automatisering.

STELLINGEN

behorend bij het proefschrift

HOE DUUR IS PROGRAMMATUUR?

begroten en beheersen van software-ontwikkeling

van

Fred J. Heemstra

I

Een qua kosten en doorlooptijd ontspoord automatiseringsproject zou een stimulans moeten zijn voor het verantwoord begroten van software-ontwikkeling.
(Bron: hoofdstuk 7 en 8 van dit proefschrift).

II

Er bestaat geen standaardaanpak voor het begroten van software-ontwikkeling. Naarmate de onzekerheid over een te realiseren software-produkt stijgt, zal de aanpak verschuiven van het efficient benutten van produktiemiddelen naar het zo goed mogelijk opstellen van een programma van eisen en wensen bij vooraf vastgestelde middelen en doorlooptijd.
(Bron: hoofdstuk 7 van dit proefschrift).

III

In de investeringsliteratuur wordt aanbevolen marges te hanteren. De omvang van de marges is afhankelijk van de mate van onzekerheid. Deze aanbeveling wordt door software-ontwikkelaars vaak niet gevolgd.
(Bron: Verheijen, P.A.; Investeren en Risico, Stenfert Kroese, 1975)

IV

Het begroten van software-ontwikkeling is meer dan het rationeel doorrekenen van modellen. Betrokkenen bij software-ontwikkeling dienen zich ook emotioneel te binden.
(Bron: hoofdstuk 7 van dit proefschrift).

V

Niet-realistische begrotingen hebben een negatief effect op produktiviteit en motivatie van software-ontwikkelaars.
(Bron: hoofdstuk 7 van dit proefschrift).

VI

Beheersen vereist registreren en dus meten. Meten kan echter alleen als men weet wat men moet meten en waarom. Zolang men dit niet weet, kan men niet zinvol registreren en dus beheersen.
(Bronnen: 1. hoofdstuk 8 van dit proefschrift.
2. Noth, T.; Unterstutzung des Managements von Software-Projekten durch eine Erfahrungsdatenbank; Springer-Verlag, 1987).

VII

Calibreren is een essentiële voorwaarde voor een verantwoord gebruik van begrotingsmodellen.
(Bronnen: 1. hoofdstuk 4 van dit proefschrift).
2. Van Vliet, J.C.; Wat kost dat nu, zo'n programma? ; Informatie, jaargang 29, extra editie, 1987).

VIII

Het ontwikkelen van informatiesystemen omvat modelleren, balanceren, veranderen, leren en beheersen. Het eenzijdig benadrukken van één van deze aspecten leidt tot een onevenwichtig ontwikkelproces.
(Bron: Bemelmans, T.M.A.; Bestuurlijke informatiesystemen en automatisering; Stenfert Kroese, 1987.

X

Door afstudeerprojecten te integreren in lopende onderzoeksprojecten kan de faculteit Bedrijfskunde haar onderzoek verstevigen en studenten beter academisch vormen.

XI

De multi-disciplinariteit van onderwijs en onderzoek binnen de faculteit Bedrijfskunde vergt een voortdurende stimulering en sturing. Bij het achterwege blijven daarvan, vervalt men weer snel in mono-disciplines.

XI

Een promovendus moet de mogelijkheid hebben dankbetuigingen op te nemen in een proefschrift. Daarmee kan tot uitdrukking worden gebracht dat een promotie het werk is van velen.
(Bron: brief van de rector aan de hoogleraren van de Technische Universiteit Eindhoven, briefnummer CvD 159.591, de datum 5-1-1989).

XII

Gezien het fileprobleem en het belang van de transportsector voor de Nederlandse economie, verdient het aanbeveling op snelwegen uitsluitend vrachtverkeer toe te laten.
(Bron: rapport Nederland Transitoland Smit-Kroes).

XIII

Lachen is gezond. Trimmers en hun bontgekleurde kledij bevorderen in dit opzicht niet alleen hun eigen gezondheid maar ook die van passanten.

XIV

De aankomst per trein in een Nederlandse stad is doorgaans geen goede promotie voor die stad.

Eindhoven, 1 mei 1989
Fred J. Heemstra

Beheersen en begroten van software-ontwikkeling is moeilijk, gelet op de signalen over forse overschrijdingen van budgetten en levertijden. Afgaande op berichten uit de praktijk en bevindingen in de literatuur komt het geregeld voor dat softwareprojecten uit de hand lopen. In dit boek staat deze problematiek centraal.
In de eerste hoofdstukken wordt de nadruk gelegd op een inventarisatie van de problemen waarmee men wordt geconfronteerd bij het beheersen en begroten van software-ontwikkeling. Door middel van een veldonderzoek is een overzicht gegeven van de huidige stand van zaken. In de volgende hoofdstukken wordt aandacht besteed aan de bestaande theorieën en methoden voor het begroten van software. Zo worden de belangrijkste begrotingsmodellen beschreven en geëvalueerd en wordt een ordening gemaakt van belangrijke factoren die van invloed zijn op kosten, inspanning en doorlooptijd. In de laatste hoofdstukken van het boek worden ideeën en theorieën beschreven, die als basis kunnen dienen voor een betere beheersing en begroting van software-ontwikkeling. Zo worden vier verschillende beheerssituaties onderscheiden, die elk om een specifieke wijze van begroten en beheersen vragen. Welke beheerssituatie van toepassing is bij de ontwikkeling van een bepaald programma, wordt bepaald door de kenmerken van de software die moet worden ontwikkeld, het proces dat moet worden uitgevoerd en de middelen die hiervoor nodig zijn.
In deze laatste hoofdstukken wordt de nadruk gelegd op het belang van gegevens van voltooide projecten voor de beheersing van lopende projecten en het begroten van nieuwe.

Kluwer Bedrijfswetenschappen

ISBN 90 267 1371 1 NUGI 852