Is programmatuurontwikkeling met formele methoden zinvol?

C.A. Middelburg
PTT Research, Tele-informatica
Leidschendam

Dit artikel geeft geen rechtstreeks antwoord op deze vraag. Het geeft aan op welke manieren formele methoden tegenwoordig proberen bij te dragen tot oplossing van de problemen die zich voordoen bij ontwikkeling van programmatuur. Het één en ander wordt toegelicht aan de hand van VDM, een formele methode voor programmatuurontwikkeling die op dit moment waarschijnlijk tot de meest geslaagde formele methoden in de praktijk behoort. Computerondersteuning bij het gebruik van formele methoden voor programmatuurontwikkeling alsmede recente ontwikkelingen op het gebied van formele methoden komen ook aan de orde. Doel van dit artikel is de lezer enig inzicht te verschaffen in wat formele methoden voor ontwikkeling van programmatuur tegenwoordig te bieden hebben en wat er in de nabije toekomst nog meer van kan worden verwacht.

1 Inleiding

De ontwikkelingen die de laatste jaren zijn opgetreden in de wiskundige onderbouwing en computerondersteuning van formele methoden voor programmatuurontwikkeling wekken hoge verwachtingen ten aanzien van wat die methoden te bieden hebben voor specificatie, ontwerp en verificatie. De manieren waarop formele methoden tegenwoordig proberen bij te dragen tot oplossing van de problemen bij programmatuurontwikkeling zouden een nuttige aanvulling kunnen zijn op wat bestaande informele methoden te bieden hebben. In dit artikel wordt een beeld geschetst van wat formele methoden op dit moment te bieden hebben.

Dit artikel is opgebouwd rondom VDM, een model-georiënteerde formele methode voor programmatuurontwikkeling. Gezien het relatief veelvuldige gebruik dat tegenwoordig van VDM wordt gemaakt bij de ontwikkeling van programmatuur in de industrie, behoort deze methode op dit moment waarschijnlijk tot de meest geslaagde formele methoden in de praktijk. Naast VDM staat Z momenteel nogal in de belangstelling. De notaties van VDM en Z hebben veel gemeen en zijn in sommige opzichten complementair. Z zal summier ter sprake komen, evenals de eigenschap-georiënteerde algebraïsche aanpak en

COLD-K — verwant aan zowel VDM als de algebraische aanpak. Dit artikel is echter geen overzicht van formele methoden voor programmatuurontwikkeling.

Een opsomming van verwachtingen c.q. beloften aangaande het nut van formele methoden wordt evenmin gegeven. Ik wil wat dat betreft volstaan met het aanhalen van twee uitspraken van C.B. Jones — één van de sleutelfiguren achter VDM. Over formele specificatie zegt hij onder meer het volgende: 'The mathematical notation can, when used with care, achieve conciseness of expression as well as precision. I believe that these ideas are important. But a major issue relating to specifications is whether they match the user's requirements.' Zijn volgende relativerende uitspraak over geverifieerd ontwerp brengt volgens mij tevens het belangrijkste argument voor (het overwegen van) het gebruik van formele methoden naar voren: 'The material relating to design aims to provide developers with ways to increase their confidence that the systems they create satisfy the specifications. ..., but it must be understood that nothing can ever provide absolute certainty of correctness.'.

In Bergstra en Renardel (1989) wordt omschreven in welke omstandigheden men iets van formele specificaties kan verwachten en in Bergstra en Renardel (1990) wordt beschreven welke resultaten van onderzoek en ontwikkeling op dit gebied kunnen worden verwacht. In dit artikel ga ik in zekere zin door op hetzelfde thema: er wordt aan de hand van VDM toegelicht wat formele specificaties zijn en wat men er verder mee kan doen. Het had ook kunnen worden opgebouwd rondom een andere formele methode die in de praktijk wordt gebruikt, zoals bijvoorbeeld RAISE (nauw verwant aan VDM), zie Nielsen et al. (1989), of de formele methode die wordt ontwikkeld rondom de specificatietaal COLD-K, zie Jonkers (1989b) of Feijs en Jonkers (1992). Bij de keuze voor VDM heeft een belangrijke rol gespeeld dat VDM in vele opzichten verder is ontwikkeld dan andere formele methoden. Maar persoonlijke voorkeur heeft ook een rol gespeeld.

De rest van dit artikel is als volgt ingedeeld. In par. 2 wordt een algemeen beeld geschetst van wat formele methoden momenteel inhouden. Nadat in par. 3 VDM in een context is geplaatst, wordt in par. 4 en par. 5 dit beeld toegelicht aan de hand van VDM. Par. 6 is gewijd aan computerondersteuning voor VDM. In par. 7 wordt VDM in verband gebracht met Z, de algebraïsche aanpak van programmatuurontwikkeling en COLD-K. In par. 8 worden enige belangrijke uitbreidingen van VDM behandeld. Par. 9 tenslotte bevat enige persoonlijke slotopmerkingen.

2 Formele methoden

De behoefte aan nauwkeurige specificaties wordt geaccepteerd in de meeste technische disciplines. Een nauwkeurige specificatie van wat er wordt verwacht van een te ontwikkelen systeem verschaft een uitgangspunt om vast te stellen of het uiteindelijke systeem aan de gestelde verwachtingen voldoet en hij dient tevens tot leidraad voor de ontwikkeling ervan. Zo kunnen er bijvoorbeeld nauwkeurige argumenten worden gegeven voor ontwerpbeslissingen. Deze aspecten van nauwkeurige specificatie worden door professionele ontwikkelaars als de belangrijkste aspecten beschouwd.

Overigens zijn er andere aspecten van nauwkeurige specificatie die zeer nuttig kunnen zijn. Een nauwkeurige specificatie maakt het bijvoorbeeld ook mogelijk om een systeem

te analyseren voordat zijn ontwikkeling wordt ondernomen. Dit opent mogelijkheden om het vertrouwen te verhogen dat het gespecificeerde systeem overeenstemt met de eisen die er aan worden gesteld. Als een verandering van een bestaand systeem wordt overwogen — hetgeen eerder regel dan uitzondering is — dan verdient het aanbeveling de gevolgen van de verandering in ogenschouw te nemen voor de verandering daadwerkelijk wordt uitgevoerd. Zonder een nauwkeurige specificatie is het vaak moeilijk die gevolgen te bevatten.

Om voldoende nauwkeurigheid te bereiken moeten specificaties worden geschreven in een notatie met volstrekt nauwkeurig gedefinieerde syntax en semantiek. Immers, als er bijvoorbeeld al onenigheid kan zijn over de bedoelde betekenis van uitdrukkingen in de gebruikte notatie, dan kan er geen sprake zijn van voldoende nauwkeurige specificatie.

Formele methoden betrekken wiskundige nauwkeurigheid bij de ontwikkeling van programmatuur. Dat lijkt in elk geval nuttig in die gevallen waar fouten niet licht kunnen worden opgevat. Bij de huidige stand van ontwikkeling zijn de belangrijkste aspecten van formele methoden *formele specificatie* en *geverifieerd ontwerp*. In het verleden lag de nadruk nogal op formele specificatie en ondersteunende gereedschappen voor formele specificatie. Dat verschuift de laatste tijd naar geverifieerd ontwerp en ondersteunende gereedschappen daarvoor. Formele specificatietalen zonder bijbehorende ontwerp- en verificatietechnieken worden niet meer beschouwd als formele methoden.

Formele specificatie

Formele specificaties zijn beschrijvingen van programma's in een notatie die aan de wiskunde is ontleend. Meestal is het een wiskundige notatie die is uitgebreid om de specificatie van programma's te vereenvoudigen. Aan deze uitbreidingen worden dan een nauwkeurig gedefinieerde betekenis toegekend in termen van algemeen aanvaarde begrippen uit de wiskunde. Onenigheid door verschillende interpretatie van een specificatie is hiermee uitgesloten. Maar de kosten van formeel specificeren zijn misschien te hoog om het hiermee te rechtvaardigen. Belangrijk is dat bij gebruik van een dergelijke notatie wiskundig kan worden geredeneerd over specificaties. Zo kan bijvoorbeeld de bewering dat een implementatie aan een specificatie voldoet worden geformuleerd als een wiskundige stelling.

Formele methoden gaan er van uit dat er eerst een formele specificatie kan worden gemaakt van wat er wordt verwacht van een te ontwikkelen systeem en dat daarna het ontwerp kan worden opgedeeld in stappen waarvan de rechtvaardiging te overzien is.

Geverifieerd onwerp

Bij geverifieerd ontwerp zijn er een aantal formele bewijzen te leveren om een ontwerpstap te rechtvaardigen. Men spreekt hier in het algemeen over het vervullen van de bewijsverplichtingen die een ontwerpstap met zich mee brengt. Binnen de formele aanpak bestaan er verschillende paradigma's voor ontwerp, zoals 'iteratieve specificatie, ontwerp en verificatie' en 'programma transformatie'. VDM is een belangrijk voorbeeld van het eerste paradigma. CIP, zie CIP (1985), is gebaseerd op programma transformatie. Bij elke

paradigma brengt een ontwerpstap in het algemeen bewijsverplichtingen met zich mee. Zo zijn aan transformatieregels toepasbaarheidsvoorwaarden gekoppeld, die bij gebruik van de regels bewijsverplichtingen worden.

In de praktijk betekent geverifieerd ontwerp dat computerondersteuning bij formele ontwikkeling van programmatuur zeer gewenst is. Met name is assistentie nodig bij het vinden van de bewijzen die nodig zijn voor het vervullen van de bewijsverplichtingen. De betreffende bewijzen hebben veelal weinig weg van bewijzen zoals men die vaak in de wiskunde tegenkomt. De bewijzen bij programmatuurontwikkeling zijn hoofdzakelijk bedoeld om op een nauwkeurige manier na te gaan of er geen details in een ontwerp over het hoofd zijn gezien, maar zij zijn meestal lang en nogal saai. Juist daarom zijn zowel formele bewijzen als computerondersteuning daarbij van groot belang. Computerondersteuning bij geverifieerd ontwerp van programmatuur betekent 'computer-aided design' van programmatuur. Dat is al lang gebruikelijk in andere technische disciplines.

Aan de formele specificatie van een programma gaat onder meer een fase vooraf waarin de eisen aan het uiteindelijke programma worden geanalyseerd. Formele methoden houden zich daar niet mee bezig.

Formele methoden en andere ontwikkelingsmethoden

Bestaande ontwikkelingsmethoden zoals bijvoorbeeld SDM (een afkorting van 'System Development Methodology'), zie SDM (1988), houden zich voornamelijk bezig met de planning en organisatie van de ontwikkeling van geautomatiseerde systemen. Zij stellen een verdeling van het ontwikkelingsproces in fasen voor, opperen een structurering van elke fase in termen van onderling samenhangende activiteiten en geven aan hoe die activiteiten moeten worden aangepakt en ondernomen. Een sterke nadruk wordt gelegd op de management aspecten van het ontwikkelingsproces en de kwaliteitsbewaking.

Er worden geen manieren aan de hand gedaan om de benodigde programma's voldoende nauwkeurig en ondubbelzinnig te specificeren en om de betreffende programma's geheel overeenkomstig de specificaties te vervaardigen. Formele methoden houden zich daar nu juist mee bezig. Omdat ideeën en heuristieken van andere methoden niet worden verworpen, vullen zij ontwikkelingsmethoden zoals SDM aan. Zij maken het mogelijk dat met details van specificatie en vervaardiging van programma's wordt omgegaan op een wiskundig nauwkeurige wijze. Dit kan nuttig zijn voor programma's die omvangrijk, complex of kritisch zijn. Zulke programma's kunnen voorkomen in systemen van zeer uiteenlopende aard. Hieronder vallen ook administratieve systemen — het soort systemen waarop SDM is ingesteld.

3 VDM

VDM staat voor 'Vienna Development Method'. Het is een formele methode voor de beschrijving en ontwikkeling van programmatuur, die ontstaan is uit een aanpak van programmatuur ontwikkeling die voor het eerst werd gebruikt in 1972. Het houdt zich bezig met alle stadia van ontwikkeling van specificatie tot en met codering. VDM en de

bijbehorende notatie voor het schrijven van formele specificaties hebben een hoge mate van acceptatie bereikt. De VDM-notatie is waarschijnlijk de meest gebruikte formele notaties voor de beschrijving van complexe systemen. De methode werd ontwikkeld in een industriële omgeving en is gebruikt voor de beschrijving en ontwikkeling van een grote verscheidenheid van programmatuur. Het is onder meer gebruikt voor 'compilers' (Pascal, CHILL, Ada, occam) en 'interpreters' (Prolog), 'database management systems' (PRTV, IMS, System 2000, System R), 'operating systems' (IBM OS/360, OS van System X gedeeltelijk), 'graphics systems' (GKS), een 'formal development support system' (*mural*), en de architectuur van 'hypertext systems'.

VDM wordt duidelijk uiteengezet in Jones (1990) dat VDM voor een grote groep 'software engineers' toegankelijk maakt. Het is vooral geschikt als basis voor praktijkgerichte cursussen en zelfstudie. Realistische toepassingen zijn te vinden in Jones en Shaw (1990).

Er is een 'Draft ISO Standard' voor de VDM-notatie (aangeduid met VDM-SL) is op 't ogenblik ter beschikking voor commentaar. Een belangrijke ontwikkeling op het gebied van ondersteunende gereedschappen is de beschikbaarheid van het 'formal development support system' *mural*, zie Jones et al. (1991), dat geschikt is voor VDM. *mural* zal later in dit artikel nog ter sprake komen. In onderzoek op het gebied van VDM wordt momenteel volop aandacht besteed aan aspecten van programmatuurontwikkeling die nog niet of onvoldoende worden gedekt door VDM (en andere formele methoden), zoals specificatie en ontwerp van parallelle programma's, specificatie in een object-georiënteerde stijl en modulaire structurering van specificaties.

In principe heeft iedereen met kennis van programmeren en enige bekendheid met logica en verzamelingenleer voldoende vooropleiding om zonder veel moeite VDM te leren gebruiken. Natuurlijk is ervaring nodig om omvangrijke en/of complexe problemen aan te kunnen, maar de gebruikelijke mythe over de hoge vooropleiding die nodig zou zijn voor het gebruik van formele methoden berust nergens op.

4 Formele specificatie in VDM

In VDM beschrijft men wat er wordt verwacht van een te ontwikkelen systeem in termen van operaties die het systeem moet kunnen uitvoeren. Het begrip toestand staat hierbij centraal. Operaties kunnen resultaten opleveren die afhangen van een *toestand* en die toestand veranderen. Operaties komen in het algemeen overeen met deelprogramma's (zoals subroutines, procedures, etc.) van het uiteindelijke systeem. Met een *pre-conditie* worden de omstandigheden waaronder het systeem een operatie met succes moet uitvoeren begrensd en met een *post-conditie* worden de mogelijke effecten van de uitvoering afgebakend. Pre- en post-condities zijn logische uitdrukkingen. De soorten gegevens(objecten) die worden gemanipuleerd door de operaties, worden beschreven in termen van abstracte wiskundige begrippen, zoals (eindige) verzamelingen en rijen.

De belangrijkste onderdelen van de VDM-notatie zijn de logische notatie (van een speciale logica voor partiële functies (LPF), zie Barringer, Cheng en Jones (1984)), notaties voor eindige verzamelingen, afbeeldingen, rijen en 'samengestelde objecten', de schrijfwijze van impliciete specificaties van functies en operaties — met pre- en post-condities — en de schrijfwijze van directe definities van functies en operaties. Voor elk onderdeel van de

VDM-notatie zijn er bijbehorende bewijsregels.

In VDM brengen ook specificaties bewijsverplichtingen met zich mee. Pre- en post-condities kunnen worden gebruikt om functies en operaties te specificeren die niet geimplementeerd kunnen worden. Daarom moet van elke functie- en operatie-specificatie worden aangetoond dat er een implementatie van bestaat.

Voorbeeld: een uitzendbureau

Dit voorbeeld gaat over het bijhouden van de actuele toestand van een uitzendbureau met betrekking tot de ingeschreven werkzoekenden en de aangeboden vacatures alsmede het beantwoorden van vragen zoals 'wat zijn geschikte kandidaten voor deze vacature'. Dit houdt in dat zowel de deskundigheden waarover een werkzoekende beschikt als de deskundigheden die noodzakelijk zijn voor het vervullen van een vacature moeten worden vastgelegd. Omdat een bedrijf meerdere vacante plaatsen kan hebben waarvoor dezelfde deskundigheden nodig zijn, worden vacatures geïdentificeerd door vacature-nummers. De toestanden van dit systeem kunnen als volgt worden gedefinieerd:

$$
\begin{array}{lll}
\mathit{Agency} :: & \mathit{cands}: & \mathit{Person} \xrightarrow{m} \mathit{Skills} \\
& \mathit{vacs}: & \mathit{Vacno} \xrightarrow{m} \mathit{Vacdata} \\
\mathit{Skills} = \mathit{Skill}\textbf{-set} & & \\
\mathit{Vacno} = \mathbb{N} & & \\
\mathit{Vacdata} :: & \mathit{comp}: & \mathit{Company} \\
& \mathit{skills}: & \mathit{Skills}
\end{array}
$$

De toestanden van dit systeem zijn samengesteld uit twee componenten, *cands* en *vacs*. De eerste component wordt gebruikt om gegevens over de ingeschreven werkzoekenden bij te houden en de tweede component wordt gebruikt om gegevens over de geplaatste vacatures bij te houden. Beide componenten zijn gedefinieerd als eindige afbeeldingen. Het bereik van de eerste component bestaat uit eindige verzamelingen en het bereik van de tweede component bestaat uit objecten die weer zijn samengesteld uit twee componenten. *Person*, *Skill* en *Company* behoeven op dit punt niet verder te worden gedefinieerd. Zij kunnen als gegeven worden beschouwd.

De operatie om een nieuwe werkzoekende in te schrijven kan nu worden gespecificeerd:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathit{APPLY}(p\colon \mathit{Person}, s\colon \mathit{Skills})} \\
\textbf{ext} & \textbf{wr}\ \mathit{cands}\colon \mathit{Person} \xrightarrow{m} \mathit{Skills} \\
\textbf{pre} & p \notin \textbf{dom}\ \mathit{cands} \\
\textbf{post} & \mathit{cands} = \overleftarrow{\mathit{cands}} \cup \{p \mapsto s\}
\end{array}
$$

In de pre-conditie wordt uitgedrukt dat $\mathit{APPLY}(p, s)$ met succes moet worden uitgevoerd als persoon p nog niet is ingeschreven als werkzoekende. In de post-conditie wordt uitgedrukt dat in de eindtoestand p is toegevoegd aan de ingeschrevene werkzoekenden (met de deskundigheden s). In de post-conditie verwijzen de namen van toestandscomponenten

naar de begintoestand als zij zijn 'versierd' met een terugwijzende haak ($\leftharpoonup$) en anders verwijzen ze naar de eindtoestand. Deze conventie behoort bij de VDM-notatie.

De post-conditie lijkt in dit geval nogal op een assignment statement. Het is belangrijk te bedenken dat het een logische uitdrukking is waarmee een bewering wordt gedaan over het verband tussen enerzijds begintoestanden en eventuele argumenten en anderzijds eindtoestanden en eventuele resultaten.

De specificatie van een operatie bestaat niet alleen uit een pre-conditie en een post-conditie. Er is ook nog een 'external clause' (beginnend met **ext**) waarin wordt opgesomd welke toestandscomponenten mogen worden geraadpleegd en/of gewijzigd. Met **wr** wordt aangegeven dat de betreffende toestandscomponent mag worden geraadpleegd en gewijzigd en met **rd** wordt aangegeven dat het alleen mag worden geraadpleegd. In de specificatie van de operatie *APPLY* drukt de external clause dus uit dat alleen de toestandscomponent *cands* mag worden geraadpleegd en dat deze toestandscomponent ook mag worden gewijzigd.

Bovenstaande operatie-specificatie brengt de volgende bewijsverplichting voor implementeerbaarheid met zich mee:

$$\forall ag \in Agency, p \in Person, s \in Skills \cdot$$
$$pre\text{-}APPLY(ag, p, s) \;\Rightarrow\; \exists ag' \in Agency \cdot post\text{-}APPLY(ag, p, s, ag')$$

Met andere woorden, voor elke combinatie van begintoestand en argumenten moet er tenminste één mogelijke eindtoestand zijn.

Merk op dat de pre- en post-conditie van de operatie *APPLY* hier worden aangehaald. Deze conventie behoort bij de VDM-notatie. Het bewijs van het bovenstaande volgt onmiddelijk uit de definitie van de operatie $\cup$ op afbeeldingen. Het is vaak het geval dat bewijsverplichtingen van dit soort een minimum aan werk vereisen.

Merk op dat de gespecificeerde operatie *APPLY* deterministisch is. Bij elke combinatie van begintoestand en argumenten bestaat er altijd slechts één mogelijke eindtoestand.

Dit is niet het geval bij de volgende operatie om een nieuwe vacature te plaatsen:

$$SUBSCR(c\colon Company, s\colon Skills)\ n\colon Vacno$$
ext **wr** $vacs\colon\ Vacno \xrightarrow{m} Vacdata$
pre **true**
post $n \notin \mathbf{dom}\ \overleftarrow{vacs} \wedge vacs = \overleftarrow{vacs} \cup \{n \mapsto mk\text{-}Vacdata(c, s)\}$

Deze operatie moet een vacature-nummer toekennen aan de geplaatste vacature. Met de post-conditie wordt met betrekking tot dit nummer alleen beweerd dat het niet al in gebruik mag zijn voor één van de andere vacatures. In de post-conditie wordt overigens gebruik gemaakt van een speciale eigenschap van LPF, de onderliggende logica van VDM. $\overleftarrow{vacs} \cup \{n \mapsto mk\text{-}Vacdata(c, s)\}$ is ongedefinieerd als $n \notin \mathbf{dom}\ \overleftarrow{vacs}$ onwaar is. De vraag is nu wat in dat geval de waarheidswaarde van $vacs = \overleftarrow{vacs} \cup \{n \mapsto mk\text{-}Vacdata(c, s)\}$ is. Dit levert echter geen probleem op omdat de waarheid van een formule van de vorm $A \wedge B$ in LPF altijd onwaar is als één van de formules A en B onwaar is (evenzo is het altijd waar als beide formules waar zijn en anders ongedefinieerd). Met de pre-conditie '**true**'

wordt uitgedrukt dat onder alle omstandigheden deze operatie met succes moet worden uitgevoerd.

De gespecificeerde operaties *APPLY* en *SUBSCR* zijn allebei voor het bijhouden van de actuele toestand van een arbeidsbureau. *APPLY* levert daarbij geen resultaat op en *SUBSCR* levert het nummer op dat aan de geplaatste vacature is toegekend.

De volgende operatie raadpleegt alleen de actuele toestand om antwoord te kunnen geven op de vraag wat geschikte kandidaten voor een bepaalde vacature zijn, maar wijzigt hem niet:

$SUITCAND(n\colon Vacno)\ ps\colon Person$-**set**
ext **rd** $cands\colon Person \xrightarrow{m} Skills$
rd $vacs\colon Vacno \xrightarrow{m} Vacdata$
pre $n \in$ **dom** $vacs$
post $ps = \{p \in Person \mid skills(vacs(n)) \subseteq cands(p)\}$

Men kan zich nog meer nuttige operaties voor stellen, zoals een operatie *AVAILVAC* om de nummers op te vragen van de vacatures die beschikbaar zijn voor een bepaalde werkzoekende, een operatie *VACDATA* om de gegevens over de vacature met een bepaald nummer op te vragen, en een operatie *ASSIGN* om een bepaalde vacante plaats aan een bepaalde werkzoekende toe te wijzen. Deze operaties worden hier alleen genoemd. Later zal er op enkele van deze operaties worden teruggekomen.

5 Geverifieerd ontwerp in VDM

VDM onderscheidt twee soorten stappen in geverifieerd ontwerp, namelijk 'data reification' en operatie-decompositie. Ontwerpstappen van beide soorten brengen bewijsverplichtingen met zich mee. De bewijsverplichtingen die data reification met zich brengt, hebben te maken met de verschillende aspecten ervan, zoals data-representatie en het modelleren van functies en operaties. Bij operatie-decompositie hangt de bewijsverplichting af van de aard van de decompositie: sequentieel, conditioneel, iteratief.

Gegeven een specificatie van een systeem, kan een ontwerpstap bestaan uit de keuze voor een representatie van de toestanden (die op implementatie overwegingen gebaseerd dient te zijn) en het modelleren van de operaties op de representatie. Dit wordt in VDM data reification genoemd. Het verband tussen de representatie en de abstractie wordt in VDM uitgedrukt door een '*retrieve-functie*'.

Nemen we het voorbeeld van de vorige paragraaf, dan kunnen we bijvoorbeeld kiezen voor de volgende representatie van de toestanden:

$Agency_c :: cands\colon Person \xrightarrow{m} Skills$
$vacs\colon Vacdata^*$
$vacnos\colon Vacno$-**set**

Een implementatieoverweging voor deze keuze zou kunnen zijn dat de gegevens over geplaatste vacatures meestal allemaal, maar wel één voor één, zullen worden geraad-

pleegd. De component *vacnos* is bedoeld om bij te houden welke vacature-nummers (die nu ook indexen in de component *vacs* zijn) in gebruik zijn.

De bijbehorende retrieve-functie wordt als volgt gedefinieerd:

$$
\begin{array}{l}
retr\colon Agency_c \to Agency \\
retr(ag_c) \triangleq mk\text{-}Agency(cands, \{n \mapsto vacs(n) \mid n \in vacnos\})
\end{array}
$$

Een bewijsverplichting die deze ontwerpstap met zich mee brengt is er één voor *adequaatheid*:

$$\forall ag \in Agency \cdot \exists ag_c \in Agency_c \cdot retr(ag_c) = ag$$

Met andere woorden, voor elke abstracte toestand moet er tenminste één representatie zijn.

Het bewijs van het bovenstaande volgt, met behulp van de inductieregel van VDM voor eindige afbeeldingen, betrekkelijk eenvoudig uit enige elementaire eigenschappen van eindige rijen en afbeeldingen. Bewijsverplichtingen van dit soort vereisen vaak inductieve bewijzen.

Modellering van de operatie *SUBSCR* leidt tot het volgende:

$$
\begin{array}{ll}
\multicolumn{2}{l}{SUBSCR_c(c\colon Company, s\colon Skills)\ n\colon Vacno} \\
\textbf{ext} & \textbf{wr}\ vacs\colon\ Vacdata^* \\
 & \textbf{wr}\ vacnos\colon\ Vacno\textbf{-set} \\
\textbf{pre} & \textbf{true} \\
\textbf{post} & n \notin \overleftarrow{vacnos} \wedge vacnos = \overleftarrow{vacnos} \cup \{n\} \wedge \\
 & \forall n' \in \overleftarrow{vacnos} \cdot vacs(n') = \overleftarrow{vacs}(n') \wedge \\
 & vacs(n) = mk\text{-}Vacdata(c, s) \wedge \\
 & \forall m \notin \overleftarrow{vacnos} \cdot n \leq m
\end{array}
$$

Hier wordt de kleinste niet in gebruik zijnde index toegekend aan de nieuw geplaatste vacature. Voor een juiste modellering had ook een willekeurige niet in gebruik zijnde index toegekend kunnen worden.

Deze modellering brengt een bewijsverplichting voor het resultaat met zich mee:

$$
\begin{array}{l}
\forall ag_c, ag'_c \in Agency_c, c \in Company, s \in Skills, n \in Vacno \cdot \\
\quad post\text{-}SUBSCR_c(ag_c, c, s, ag'_c, n) \ \Rightarrow\ post\text{-}SUBSCR(retr(ag_c), c, s, retr(ag'_c), n)
\end{array}
$$

Met andere woorden, de operatie op de representatie mag, wanneer er wordt geabstraheerd met de retrieve-functie, geen andere effecten hebben dan die van de oorspronkelijke abstracte operatie.

Het bewijs van het bovenstaande volgt, hoofdzakelijk door equationeel redeneren uit algemeen bekende eigenschappen van eindige rijen en afbeeldingen.

6 Computerondersteuning voor VDM

Ook bij gebruik van VDM betekent geverifieerd ontwerp dat computerondersteuning bij programmatuurontwikkeling gewenst is. De hoofdcomponenten van het 'formal development support system' *mural* zijn een 'VDM support tool' en een 'proof assistant'. Tezamen bieden zij zowel ondersteuning voor het creëren van VDM-specificaties en -ontwerpen als voor het vervullen van de bewijsverplichtingen die dat met zich mee brengt. Bij de ontwikkeling van *mural* is er van uitgegaan dat het systeem een aanlokkelijker omgeving moest bieden dan potlood en papier. Dit wordt in sterke mate bereikt doordat de gebruiker, in plaats van het systeem, leidend is bij het vinden van een bewijs.

De VDM support tool ondersteunt het creëren van VDM-specificaties en ontwerpen, die overigens in 'ontwikkelingen' kunnen worden gegroepeerd. Met de VDM support tool kunnen ook uit gecreëerde VDM-specificaties en ontwerpen de bijbehorende bewijsverplichtingen worden gegenereerd. Die bewijsverplichtingen kunnen dan met behulp van de proof assistant worden vervuld. De gegenereerde bewijsverplichtingen zijn vaak zeer verhelderend en ook nuttig in het geval er geen formele bewijzen voor worden geleverd. Al mogen we aannemen dat gemaakte onwerpstappen op inzicht berusten, de bewijsverplichtingen brengen vaak details naar voren die men over het hoofd heeft gezien.

Belangrijke aspecten van de proof assistant zijn: (1) de gebruiker stuurt het opbouwen van een bewijs uitgaande van zijn inzicht en het systeem verricht routinematig werk; (2) het systeem laat de gebruiker vrij om onderdelen van een bewijs niet in detail uit te werken.

Tijdens het opbouwen van een bewijs ziet de gebruiker voortdurend een bewijsschets op het scherm waarin is aangegeven wat nog niet formeel is gerechtvaardigd. Als hij die bewijsschets verder wil uitwerken door één van de bewijsregels op een bepaalde manier toe te passen, dan zorgt het systeem voor het bijwerken van de bewijsschets. Tot het routinematige werk bedoeld onder (1) behoort onder meer het zoeken naar mogelijke volgende stappen in een bewijs als de gebruiker die zelf niet kan vinden en het nagaan van de consequenties van elk van die stappen voor het verdere verloop van het bewijs. Dit betekent dat de gebruiker het bewijs niet met de hand behoeft voor te bereiden.

De vrijheid zoals genoemd onder (2) is belangrijk omdat het uitwerken van alle onderdelen van een bewijs in de praktijk niet altijd even nuttig is. Zo komen er, evenals in de gebruikelijke wiskunde, meestal stappen in een bewijs voor die als triviaal kunnen worden beschouwd. Het bewijs van dergelijke stappen voegt niets toe aan het vertrouwen dat er een product wordt ontwikkeld dat aan de gestelde eisen zal voldoen. Het blijft echter bekend wat niet formeel is gerechtvaardigd, zodat er bij latere twijfel altijd nog op terug kan worden gekomen.

7 VDM en andere methoden

Z

Het nauwst verwant aan VDM is het recentere Z dat werd ontwikkeld in een academische omgeving. Het is ontstaan uit een stijl van specificatie en ontwerp die voor het eerst werd

gebruikt omstreeks 1980. De wiskundige betekenis van de uitdrukkingen in de Z-notatie wordt uitgebreid behandeld in Spivey (1988). Geverifieerd ontwerp is in Z minder ver ontwikkeld dan in VDM. Evenals in VDM, bestaat de beschrijving van een systeem in Z uit een definitie van de toestanden van het systeem gevolgd door de specificatie van de operaties die het systeem moet kunnen uitvoeren.

Ter vergelijking wordt het voorbeeld uit par. 4 hieronder gedeeltelijk herhaald in de Z-notatie. De toestanden kunnen in Z als volgt worden gedefinieerd:

$[Person, Skill, Company]$

Agency

$cands : Person \pfun Skills$
$vacs : Vacno \pfun Vacdata$

$Skills \mathrel{\widehat=} \mathbb{F}\, Skill$
$Vacno \mathrel{\widehat=} \mathbb{N}$

Vacdata

$comp : Company$
$skills : Skills$

De abstracte wiskundige begrippen in termen waarvan de toestanden hier worden beschreven verschillen niet van die in par. 4. Afgezien van details op het niveau van de concrete syntax is het enige verschil dat *Person*, *Skill* en *Company* hier expliciet als gegeven worden beschouwd.

Bij het specificeren in Z van operaties zijn, in aanvulling op de gegeven formele semantiek van de Z-notatie, enige informele conventies over de overeenkomst tussen specificaties en operaties nodig: (1) argumenten hebben namen die eindigen met een vraagteken, (2) resultaten hebben namen die eindigen met een uitroepteken, (3) de namen van toestandscomponenten verwijzen naar de eindtoestand als zij worden gevolgd door $'$ en anders verwijzen ze naar de begintoestand (de namen met $'$ worden geintroduceerd met Δ). Merk op dat VDM dergelijke informele conventies niet kent.

De operatie om een nieuwe werkzoekende in te schrijven kan in Z als volgt worden gespecificeerd:

APPLY

$\Delta Agency$
$p? : Person$
$s? : Skills$

$p? \notin \mathrm{dom}\ cands$
$vacs' = vacs$
$cands' = cands \cup \{p? \mapsto s?\}$

De pre- en post-conditie zijn niet, zoals in VDM, gescheiden. Hierdoor worden de uitdrukkingsmogelijkheden van Z vergroot, maar wordt het tevens moeilijk gemaakt om geschikte bewijsverplichtingen te koppelen aan specificaties en ontwerpstappen. Ook bestaat er geen speciale notatie om aan te geven dat een bepaalde toestandscomponent alleen geraadpleegd mag worden. Daarom is hierboven ook de conditie $vacs' = vacs$ opgenomen.

Evenmin als in VDM, is de specificatie van niet-deterministische operaties een probleem in Z. De specificatie van de operatie om een nieuwe vacature te plaatsen kan bijvoorbeeld als volgt worden gespecificeerd:

SUBSCR

$\Delta Agency$
$c?: Company$
$s?: Skills$
$n!: Vacno$

$cands' = cands$
$n! \notin \text{dom } vacs$
$vacs' = vacs \cup \{n! \mapsto \mu\, Vacdata \mid comp = c? \land skills = s?\}$

Er zijn duidelijk veel overeenkomsten tussen de notaties van VDM en Z. De verschillen maken VDM geschikter voor programmatuurontwikkeling van specificatie tot en met codering. Daarentegen maken die verschillen Z misschien geschikter voor de analyse-fase die aan formele specificatie vooraf gaat.

De algebraïsche aanpak

De algebraïsche aanpak behelst technieken voor specificatie, ontwerp en verificatie die er van uitgaan dat een te ontwikkelen systeem wordt beschreven in termen van zijn gewenste eigenschappen. Deze eigenschappen moeten in het algemeen worden gegeven in de vorm van (conditionele) vergelijkingen die de betreffende operaties met elkaar in verband brengen. Er zijn vele algebraïsche specificatietalen ontwikkeld, zoals bijvoorbeeld Clear (zie Burstall en Goguen (1981)), ACT ONE (zie Ehrig, Feys en Hansen (1983)), the Larch Shared Language (zie Guttag en Horning (1986)) en ASF (zie Bergstra, Heering en Klint (1989)). Hoewel er veel is gedaan aan de wiskundige grondslagen voor algebraïsche formele methoden voor programmatuurontwikkeling, is er weinig bekend over dergelijke methoden. Een goed overzicht van de algebraïsche aanpak wordt gegeven in Wirsing (1989).

In de algebraïsche aanpak worden de toestanden niet expliciet beschreven. De begintoestand van elke operatie moet worden behandeld als argument van de betreffende operatie en de eindtoestand van elke operatie die de toestand bijwerkt moet worden behandeld als (onderdeel van) het resultaat. Op die manier kunnen de toestanden impliciet worden beschreven door vergelijkingen die de operaties met elkaar in verband brengen. Een beperking van de algebraïsche aanpak is verder dat elke operatie moet worden beschouwd als functie.

Om een beeld te kunnen geven van algebraïsche specificatie aan de hand van het voorbeeld uit par. 4, veronderstellen we het volgende: (1) partiële functies (functies die niet altijd een resultaat opleveren) zijn toegestaan en (2) alle modellen die aan de gegeven vergelijkingen voldoen worden in beschouwing genomen. Deze veronderstellingen sluiten veel bestaande algebraïsche specificatietalen uit. Door partiële functie te accepteren doet zich verder het probleem voor dat één of beide zijden van een vergelijking ongedefinieerd kunnen zijn. We kiezen er hier voor in die gevallen de vergelijking altijd onwaar te beschouwen. Hierdoor komt vooral de betekenis van conditionele vergelijkingen wel eens niet overeen met de intuïtie.

Een manier om de benodigde vergelijkingen te vinden is als volgt: (1) bepaal een verzameling elementaire raadplegingsoperaties (operaties die de toestand raadplegen) waaruit de gevraagde raadplegingsoperaties kunnen worden afgeleid, (2) karakteriseer elk van de wijzigingsoperaties (operaties die de toestand wijzigen) door vergelijkingen die het effect op het resultaat van alle elementaire raadplegingsoperaties voldoende beschrijven, en (3) karakteriseer elk van de gevraagde raadplegingsoperaties door vergelijkingen die het resultaat ervan voldoende beschrijven in termen van de elementaire raadplegingsoperaties.

De operatie *apply* (de operatie-namen worden hier met kleine letters geschreven) kan nu worden gekarakteriseerd door de volgende conditionele vergelijkingen:

$$
\begin{array}{lll}
iscand(apply(a,p,s),p') = true & \textbf{if} & iscand(a,p) = false \\
iscand(apply(a,p,s),p') = iscand(a,p') & \textbf{if} & iscand(a,p) = false, \\
 & & p \neq p' \\
candata(apply(a,p,s),p) = s & \textbf{if} & iscand(a,p) = false \\
candata(apply(a,p,s),p') = candata(a,p') & \textbf{if} & iscand(a,p) = false, \\
 & & iscand(a,p') = true \\
isvac(apply(a,p,s),n)) = isvac(a,n) & \textbf{if} & iscand(a,p) = false \\
vacdata(apply(a,p,s),n)) = vacdata(a,n) & \textbf{if} & iscand(a,p) = false
\end{array}
$$

Als elementaire raadplegingsoperaties zijn gekozen: een operatie *iscand* om op te vragen of een bepaalde persoon is ingeschreven als werkzoekende, een operatie *candata* om de deskundigheden van een bepaalde werkzoekende op te vragen, een operatie *isvac* om op te vragen of een bepaald nummer in gebruik is als vacature-nummer voor één van de geplaatste vactures, en de operatie *vacdata*. Merk op dat de elementaire raadplegingsoperaties *iscand* en *candata* hier tezamen de rol van de toestandscomponent *cands* (par. 4) overnemen. Evenzo nemen *isvac* en *vacdata* tezamen de rol van de toestandscomponent *vacs* over. Dit benadrukt één van de grootste verschillen met model-georiënteerde specificatie, zoals in VDM, namelijk dat er geen expliciete beschrijving van de toestanden wordt gegeven.

Het karakteriseren van de operatie *subscr* is lastiger. Zelfs het karakteriseren van een vereenvoudigde versie waarbij het resultaat alleen bestaat uit de eindtoestand is moeilijk. De twee conditionele vergelijkingen die het moeilijkst zijn te doorgronden zijn:

$$
\begin{array}{lll}
isvac(subscr(a,c,s),n)) = true & \textbf{if} & isvac(a,n) = false, \\
 & & vacdata(subscr(a,c,s),n) = pair(c,s) \\
vacdata(subscr(a,c,s),n)) = pair(c,s) & \textbf{if} & isvac(a,n) = false, \\
 & & isvac(subscr(a,c,s),n) = true
\end{array}
$$

Zonder de beperking tot conditionele vergelijkingen zouden deze twee vergelijkingen vervangen kunnen worden door de volgende formule:

$$
\begin{array}{l}
isvac(a,n) = false \;\Rightarrow \\
\quad (isvac(subscr(a,c,s),n) = true \;\Leftrightarrow\; vacdata(subscr(a,c,s),n) = pair(c,s))
\end{array}
$$

Merk verder op dat de operatie *subscr*, in tegenstelling tot zijn tegenhanger uit par. 4, deterministisch is. Hieraan valt niet eenvoudig te ontkomen omdat operaties alleen als functie zijn te beschouwen. Er bestaat wel een mogelijkheid om er aan te ontkomen, maar die zou de specificatie van deze operatie nog verder compliceren.

Het bovenstaande levert het beeld op dat algebraïsche specificatie van operaties erg lastig is en al gauw tot onvoldoende abstractie leidt. Dit is niet verwonderlijk als men bedenkt dat er wordt uitgegaan van een zeer elementaire notatie die op geen enkele manier is aangepast om de specificatie van programma's te vereenvoudigen. Zo zijn er geen speciale voorzieningen voor het beschrijven van systemen in termen van operaties die een toestand raadplegen en/of wijzigen — zoals bijvoorbeeld in VDM-SL. Wel kunnen betrekkelijk eenvoudig algebraïsche specificaties worden gegeven van de data types die in VDM-SL worden gebruikt om de toestanden van een systeem te modelleren (natuurlijke, gehele en rationale getallen, eindige verzamelingen, afbeeldingen en rijen, etc.). Dit gaat ook op voor andere data types die nuttig of nodig zijn voor een of ander soort toepassingen (b.v. eindige relaties). Zulke algebraïsche specificaties leiden in het algemeen tot vereenvoudiging van de formele bewijzen die nodig zijn voor het vervullen van bewijsverplichtingen. Algebraïsche specificatietechnieken lijken daarom het best tot zijn recht te komen in een specificatietaal waarin algebraïsche specificatie (van data types) met model-georiënteerde specificatie (van toestandsgebaseerde systemen) kan worden gecombineerd.

COLD-K

Zo'n taal is COLD-K, zie Jonkers (1989b). COLD is een afkorting van 'Common Object-oriented Language for Design' en K staat voor 'Kernel'. Deze taal kan worden beschouwd als een algebraïsche specificatietaal die is uitgebreid met speciale voorzieningen voor het beschrijven van toestandsgebaseerde systemen. COLD-K en een formele methode voor programmatuurontwikkeling rondom deze specificatietaal werden recentelijk ontwikkeld op het Philips Research Laboratorium in Eindhoven, zie Feijs en Jonkers (1992).

VDM-SL kan grofweg worden gezien als een beperkte versie van COLD-K met veel 'syntactic sugar'. Daarom is het nutteloos om het voorbeeld uit par. 4 nogmaals te herhalen in COLD-K. Voor impliciete specificatie van een operatie wordt in COLD-K een formule van de dynamische logica gebruikt. De gebruikelijke pre- en post-conditie kunnen tezamen worden beschouwd als een afkorting van een eenvoudige formule van de dynamische logica. Dit is slechts een van de punten waarop VDM-SL in uitdrukkingskracht de min-

dere is van COLD-K. Bewijsverplichtingen worden alleen gekoppeld aan specificaties en ontwerpstappen waarbij gebruik is gemaakt van een beperkte versie van COLD-K die nauw verwant is aan VDM-SL.

8 Uitbreidingen van VDM

Soms bestaat er behoefte om operaties te specificeren die gevoelig zijn voor interferentie door gelijktijdig uitgevoerde operaties. Ook komt het voor dat door de omvang of complexiteit van een te ontwikkelen systeem er behoefte bestaat aan voorzieningen voor modulaire structurering van specificaties. Het ESPRIT-project 'VDM for Interfaces of the PCTE' (VIP) werd hiermee geconfronteerd. Het had hierdoor te maken met de situatie dat er geen taal voorhanden was die voorzag in de behoeften van de belangrijkste taak van het project: het opleveren van een formele specificatie van de interfaces van de 'Portable Common Tool Environment' (PCTE). De PCTE beoogt de coördinatie en integratie van gereedschappen voor software engineering te ondersteunen. Zij biedt onder meer een object management system en een gemeenschappelijke gebruikersinterface. Het ontbreken van een geschikte specificatietaal heeft geleid tot het ontwerp van VVSL (een afkorting van 'VIP VDM Specification Language'), zie Middelburg (1989).

VVSL kan worden gezien als een combinatie van VDM-SL en de taal van een temporele logica. De belangrijkste verschillen tussen VVSL en VDM-SL zijn: (1) de toevoeging van de inter-conditie aan de gebruikelijke pre- en post-condities voor specificatie van operaties in VDM-stijl ter ondersteuning van specificatie van operaties die op elkaar inwerken via gemeenschappelijke toestandscomponenten; (2) de voorziening van modularisatie- en parameterisatie-mechanismen die geschikt zijn voor het modulair structureren van grote specificaties in VDM-stijl en een deugdelijke wiskundige grondslag hebben. De inter-conditie is een formule uit de taal van een temporele logica. Door het gebruik van de inter-conditie kunnen operaties die gevoelig zijn voor interferentie door externe toestandsveranderingen worden gespecificeerd terwijl de VDM-stijl van specificeren waar mogelijk wordt gehandhaafd. De modularisatie- en parameterisatie-mechanismen staan twee modules toe om gezamelijke toestandscomponenten te hebben met inbegrip van verborgen toestandscomponenten. Zij laten ook toe dat er eisen worden gesteld aan de modules waarop een geparameteriseerde module mag worden toegepast. Hieronder worden specificatie van interferentie en modulaire structurering in VVSL alleen ingeleid. Meer over deze uitbreidingen van VDM-SL is te vinden in Middelburg (1991a) en Middelburg (1991b).

Specificatie van interferentie

Wat er toe doet voor de gebruikers (personen, programma's of wat dan ook) van een computer-gebaseerd systeem zijn de operaties die het systeem kan uitvoeren en de waarneembare effecten van hun uitvoering. Een systeem kan zorgen voor gelijktijdige uitvoering van meerdere operaties in een omgeving met meerdere gebruikers of niet. Als het systeem er wel voor zorgt dan kan de situatie zich voordoen dat enkele van de operaties met opzet gevoelig worden gemaakt voor interferentie door gelijktijdig uitgevoerde operaties. De uitvoering van zulke operaties eindigt in een toestand en/of levert een resultaat

op dat afhangt van tussenliggende toestandsveranderingen teweeggebracht door de gelijktijdige uitvoering van andere operaties. De uitvoering ervan kan zelfs worden onderbroken om te wachten op een geschikte toestandsverandering. Het is ook mogelijk dat bepaalde tussenliggende toestansveranderingen, die niet door de uitvoering van de operatie zelf zijn teweeggebracht, er voor zorgt dat de uitvoering ervan nooit eindigt.

Als een operatie die gevoelig is voor interferentie wordt gespecificeerd met behulp van alleen een pre- en post-conditie, dan wordt er niet beschreven welke interferentie vereist is voor het voorkomen van vele eindtoestanden en/of opgeleverde resultaten. Het beschrijven hiervan kan met behulp van inter-condities worden gedaan op een manier die dicht ligt bij de manier waarop het normaal wordt besproken. Een inter-conditie bepaalt de mogelijke opeenvolgingen van toestandsveranderingen die kunnen worden gegenereerd door de betreffende operatie en zijn interfererende omgeving. Hierbij wordt onderscheid gemaakt tussen de toestandsveranderingen teweeggebracht door de operatie zelf (interne stappen) en de toestandsveranderingen teweeggebracht door de zijn interfererende omgeving (externe stappen). In Jones (1983) wordt toevoeging van rely- en guarantee-condities aan de gebruikelijke pre- en post-condities voorgesteld. Deze toegevoegde condities kunnen tezamen worden beschouwd als een afkorting van een eenvoudige inter-conditie.

Modulaire structurering

Door de omvang en complexiteit van veel systemen die tegenwoordig worden ontwikkeld is er behoefte ontstaan aan voorzieningen voor modulaire structurering van specificaties. De volgende doelen van modulaire structurering van een specificatie worden — in de context van formele methoden voor programmatuurontwikkeling — in het algemeen onderkend: (1) het begrijpelijker maken van de specificatie, (2) het verbeteren van de aanpasbaarheid van de specificatie en (3) het mogelijk maken van hergebruik van bestaande modules. Dit leidt tot het hanteren van de volgende criteria voor de keuze van de modulaire structuur van een specificatie: (1) eenvoud van de afzonderlijke modules, (2) intuïtieve duidelijkheid van de modulaire structuur en (3) geschiktheid van de afzonderlijke modules voor hergebruik.

In geval van een goede modulaire structurering behoort de ontwikkeling van theorieën over de afzonderlijke modules tot de mogelijkheden. Dit kan nuttig zijn om ontwerpstappen te rechtvaardigen en te verduidelijken wat in een module wordt beschreven (bijv. om potentiële herbruikbaarheid te vergroten). Het is niet geheel duidelijk of in geval van een goede modulaire structurering ook componentsgewijs ontwerp van het gespecificeerde systeem tot de mogelijkheden behoort.

In VVSL kunnen modules worden aangepast en gecombineerd door middel van *herbenoemen*, *importeren* en *exporteren*. Decompositie en 'information hiding', basisbegrippen van modularisatie, worden mogelijk gemaakt door respectivelijk importeren en exporteren. Herbenoemen maakt controle over 'name clashes' bij het samenvoegen van modules mogelijk. De gebruikelijke ongestructureerde VDM specificaties worden gebruikt als bouwstenen. Modules kunnen worden geparameteriseerd door middel van *abstractie* en deze modules kunnen weer worden toegepast door middel van *applicatie*. Herbruikbaarheid wordt in de eerste plaats mogelijk gemaakt door abstractie en applicatie. Deze mechanismen voor modulaire structurering zijn ontleend aan die van COLD-K, zie Jonkers

(1989a).

9 Slotopmerkingen

De volgende uitspraak kan worden gezien als een gedeeltelijk antwoord van mij op de vraag die als titel is gekozen: elke 'software engineer' behoort in zijn opleiding ervaring op te hebben gedaan met het gebruik van één of meer formele methoden voor programmatuurontwikkeling. De bedoeling hiervan is niet zozeer om formele methoden altijd even strikt toe te passen, maar om de benodigde theoretische basis voor een professionele aanpak van programmatuurontwikkeling te leggen. Het is uiterst belangrijk om goed te weten wat er precies bij komt kijken als alle formele details worden uitgewerkt. Die kennis is nodig voor een goed begrip van het proces van programmatuurontwikkeling alsmede de vereiste kundigheden om het uit te voeren.

Men kan ook blijven doorgaan zich alleen bezig te houden met datgene waar managers enthousiast voor te krijgen zijn, namelijk de intuïtief eenvoudige oplossingen voor problemen bij programmatuurontwikkeling via metrieken, documentatie standaarden, management procedures, etc. Dergelijke oplossingen zullen in vele gevallen voldoen. Het is echter aannemelijk dat een professionelere aanpak van programmatuurontwikkeling, die mede is gebaseerd op het toepassen van een degelijke theorie voor programmatuurontwikkeling, voor bepaalde soorten programmatuur (bijvoorbeeld programmatuur van computer-gebaseerde systemen die omvangrijk, complex of kritisch zijn) nuttig of soms zelfs noodzakelijk is om tot goede produkten te komen.

Referenties

Barringer, H., H. Cheng en C.B. Jones (1984), 'A logic covering undefinedness in program proofs'. *Acta Informatica*, 21:251–269.

Bergstra, J.A. en G.R. Renardel de Lavalette (1989), 'De plaats van formele specificaties in de softwaretechnologie'. *Informatie*, 31:480–494.

Bergstra, J.A. en G.R. Renardel de Lavalette (1990), 'Onderzoek en ontwikkeling op het gebied van formele specificatietalen'. *Informatie*, 32:73–83.

Bergstra, J.A., J. Heering en P. Klint (eds.) (1989), 'Algebraic Specification'. ACM Press, Addison-Wesley.

Burstall, R.M. en J.A. Goguen (1981), 'An informal introduction to specifications using Clear'. In R. Boyer en J. Moore, editors, *The Correctness Problem in Computer Science*, hoofdstuk 4. Academic Press.

CIP Language Group (1985), *The Wide Spectrum Language CIP-L*. Springer Verlag, LNCS 183.

Ehrig, H., W. Feys en H. Hansen (1983), 'ACT ONE: An algebraic specification language with two levels of semantics'. Bericht Nr. 83-03, Technical University of Berlin, Department of Computer Science.

Feijs, L.M.G. en H.B.M. Jonkers (1992), *Specification and Design with COLD-K*. Cambridge University Press, Cambridge Tracts in Theoretical Computer Science.

Guttag, J.V. en J.J. Horning (1986), 'Report on the Larch shared language'. *Science of Computer Programming*, 6:103–134.

Jones, C.B. (1983), 'Specification and design of (parallel) programs'. In R.E.A. Mason, editor, *IFIP '83*, blz. 321–332. North-Holland.

Jones, C.B. (1990), *Systematic Software Development Using VDM*. Prentice-Hall, second edition.

Jones, C.B., K.D. Jones, P.A. Lindsay en R. Moore (1991), *mural — A Formal Development Support System*. Springer Verlag.

Jones, C.B. en R.C.F. Shaw (1990), *Case Studies in Systematic Software Development*. Prentice-Hall.

Jonkers, H.B.M. (1989a), 'Description algebra'. In M. Wirsing en J.A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications*, blz. 283–305. Springer Verlag, LNCS 394.

Jonkers, H.B.M. (1989b), 'An introduction to COLD-K'. In M. Wirsing en J.A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications*, blz. 139–205. Springer Verlag, LNCS 394.

Middelburg, C.A. (1989), 'VVSL: A language for structured VDM specifications'. *Formal Aspects of Computing*, 1(1):115–135.

Middelburg, C.A. (1990), *Syntax and Semantics of VVSL — A Language for Structured VDM Specifications*. Proefschrift, Universiteit van Amsterdam.

Middelburg, C.A. (1991a), 'Modular structuring of VDM specifications in VVSL'. Pub. 288/91, PTT Research. Verschijnt in *Formal Aspects of Computing*, 4(1).

Middelburg, C.A. (1991b), 'Specification of interfering programs based on inter-conditions'. Pub. 166/91, PTT Research. Verschijnt in *Software Engineering Journal*, 7(3).

Nielsen, M., K. Havelund, K.R. Wagner en C.W. George (1989), 'The RAISE language, method and tools'. *Formal Aspects of Computing*, 1(1):85–114.

Spivey, J.M. (1988), *Understanding Z*. Cambridge University Press, Cambridge Tracts in Theoretical Computer Science 3.

Wirsing, M. (1989), 'Algebraic specification'. Technical Report MIP-8914, University of Passau, Department of Mathematics and Computer Science.

Kees Middelburg is onderzoeker bij PTT Research (Hoofdafdeling Informatica) te Leidschendam. Hij is werkzaam op het gebied van methoden en technieken voor de beschrijving en ontwikkeling van programmatuur. Verder is hij betrokken bij de standaardisatie van de VDM-notatie (VDM-SL).