Wij zijn ontwerpers

Wij zijn ontwerpers

REDE

uitgesproken bij de aanvaarding
van het ambt van hoogleraar
in de informatica
aan de Rijksuniversiteit te Groningen
op dinsdag 17 december 1985

door

Jan L. A. van de Snepscheut

Mijnheer de rector magnificus,
dames en heren,

Programmeren is een jong vak niettegenstaande de onmiskenbare Griekse wortels die het met menig ander respectabel vak gemeen heeft. Die Griekse programmeerwortels bestaan uit de algoritme van Eratosthenes om priemgetallen te berekenen, de algoritme van Euclides om de grootste gemene deler van twee getallen te berekenen, en enkele meetkundige algoritmen, bijvoorbeeld om een bissectrice te bepalen. Toch is programmeren een jong vak want onder programmeren verstaan we het op systematische wijze opstellen van een aantoonbaar correct algoritme. Dat de Grieken zich hiermee niet hebben bezig-gehouden is niet verwonderlijk want hun algoritmen waren van heel bescheiden omvang en dan is er nauwelijks reden tot systematisch ontwerpen. De aantoonbare correctheid was al wel belangrijk, het waren immers Grieken. Overigens, om over correctheid te kunnen praten is het nodig om alle verlangde eigenschappen ondubbelzinnig vast te leggen in wat we de specificatie van het programma noemen. In de informatica spelen programmaspecificaties een heel belangrijke rol. De specificatie geeft enerzijds aan wat de gebruiker van het programma mag verwachten, en geeft anderzijds aan wat de programmeur dient te implementeren. Een formele beschrijving van de specificatie in een welgedefinieerd formalisme is onontbeerlijk om een eventueel geschil tussen partijen te kunnen beslechten. Minstens zo belangrijk is dat van plaatsen waar de formele beschrijving ingewikkelder is dan verwacht een waarschuwende werking kan uitgaan. Naar de gebruiker, dat het betreffende aspect van het programma wel eens lastig te gebruiken zou kunnen zijn. (De praktijk leert dat die last zich dan vaak tot het gebruik van het hele programma uitstrekt.) Naar de programmeur, dat het betreffende aspect van het programma wel eens lastig te implementeren zou kunnen zijn. (De praktijk leert dat die last zich dan vaak tot het ontwerp van het hele programma uitstrekt.) En naar de opsteller van de specificatie, ofwel dat de specificatie inherent te ingewikkeld is, een ongewenst product specificeert, en zo mogelijk beter door een andere vervangen kan worden, ofwel dat niet het meest adequate formalisme is gebruikt. Die laatste mogelijkheid moet niet worden onderschat: als het om een echt nieuw soort programma gaat dan is het bijna altijd noodzakelijk om een aantal essentiële begrippen en passende namen

uit te vinden om de specificatie op heldere wijze te kunnen beschrijven. Het bedenken van die begrippen is vaak een kwestie van hard werken plus een voldoende dosis gezond verstand. (Wat de lijfspreuk van het Wiskundig Genootschap ook moge suggereren: hard werken alleen is niet voldoende!) De juiste keuze van begrippen betekent in elke wetenschappelijke discipline een doorbraak. Het leren opstellen van programmaspecificaties verdient dan ook ten volle onze aandacht. Voor vandaag wil ik het echter bij de constatering daarvan laten.
Over het verantwoord gebruik van programma's, en dus van computers, is heel veel maar ook heel weinig te zeggen. Heel veel omdat verantwoord gebruik lang niet zo eenvoudig is als vaak klakkeloos wordt aangenomen; heel weinig omdat computer-gebruik geen onderdeel van de informatica is terwijl ik mij vandaag tot de informatica wens te beperken. Ik volsta daarom met twee citaten. Het eerste citaat is ontleend aan Computer Worship van Ivor Catt.

> 'Computers can send people barmy if they have no religion and are desperate for one, and also if they are desperate for recognition. The computer is today's philosophers' stone, the supreme object of alchemy. But whereas the philosophers' stone offered only wealth, the computer seems to offer spiritual aid as well. It generates a religious fervour in its adherents, and they show their fervour and faith by muttering their incantations, streams of computer-like terminology, in much the same way as the religious will go through ritual prayer to propitiate their gods.'

Het tweede citaat is ontleend aan Van Dale's Groot Woordenboek der Nederlandse Taal (tiende druk).

> *'geprogrammeerd onderwijs,* onderwijs waarbij de leerstof in mootjes wordt gehakt en aan de patiënt toegediend alsof hij een computer was.'

Ik ga nu, na de opsteller en de gebruiker, over naar de derde en laatste bij een programmaspecificatie betrokkene, de programmeur. De programmeur is degene die een programma moet maken dat voldoet aan alle eisen waaruit de specificatie bestaat. Om enig gevoel te krijgen voor de problemen waarmee de programmeur geconfronteerd wordt begeef ik mij eventjes op het gladde ijs van een analogie. Veronderstel eens dat u een vliegtuig gaat ontwerpen. De specificatie bestaat uit een pakket van eisen zoals de gewenste vervoerscapaciteit, snelheid, vliegbereik, wendbaarheid, brandstofverbruik, enz. Als zowel een vervoers-

capaciteit van 400 personen als een slechts voor luchtacrobatiek nodige wendbaarheid worden verlangd kunt u de opdracht beter teruggeven. In de praktijk zijn zulke tegenstrijdigheden helaas vaak aanwezig, zij het wat minder in het oog springend, maar laten we, om enige voortgang te boeken, doen alsof ze dit keer afwezig zijn. U gaat daarom achter de tekentafel zitten en u gaat, ja, wat gaat u doen? Het ontwerpen van een vliegtuig is geen kleinigheid maar gelukkig kunt u terugvallen op een heleboel ervaring, kennis en theorie. U weet dat een vliegtuigje voor luchtacrobatiek bij voorkeur geen straalmotor maar een propeller heeft, uw weet dat een vliegtuig vleugels heeft, hoe daarvan de stijfheid en draagkracht berekend kunnen worden, u weet waar kunststoffen toegepast kunnen worden om de massa klein te houden en waar titanium nodig is om zijn hoge smeltpunt. Hoe zit het met de programmeur die een vergelijkbare specificatie voorgeschoteld krijgt? Hét grote verschil tussen de vliegtuigontwerper en de programmeur is de mate waarin hun ontwerpen uiteenlopen. De vliegtuigontwerper ontwerpt weliswaar altijd vliegtuigen en de programmeur altijd programma's, maar het verschil tussen het ene en het andere vliegtuig is vele malen kleiner dan het verschil tussen het ene en het andere programma. Dat is niet een verwijt aan gebrek aan fantasie van de vliegtuigontwerper of gebrek aan standaardisatie door de programmeur maar een direct gevolg van het feit dat de apparatuur waarover we beschikken de aanduiding general purpose equipment ten volle verdient. De toepassingen lopen dermate uiteen en zijn vaak zo nieuw dat de programmeur in de meeste gevallen zal moeten constateren dat er geen ervaring, kennis of theorie is om op terug te vallen, om te voorzien in de broodnodige heuristische sturing waar de vliegtuigontwerper wel over beschikt. En hiermee zijn we dan terecht gekomen bij een van de wezenlijke vragen van de informatica: hoe kan iets ontworpen worden dat aan een of andere gegeven specificatie voldoet zonder dat de in andere vakken gebruikelijke ervaring, kennis of theorie voor handen is? Je kunt natuurlijk zeggen: dan moet je die ervaring, kennis en theorie eerst opbouwen, maar dat vind ik geen bevredigend antwoord. Om te begrijpen wat je dan wel kunt doen, doen we even een stapje terug. We hebben al geconstateerd dat het nodig is om de programmeur een ondubbelzinnige specificatie te geven. Die specificatie is opgesteld in een formele notatie en vertrouwdheid met het formalisme is een absolute noodzaak om de specificatie vaardig en betrouwbaar te kunnen hanteren. Aan die vertrouwdheid met het formalisme ontleent de programmeur aanwijzingen

hoe een programma geconstrueerd kan worden dat aan de gegeven specificatie voldoet. Ik zal u hiervan een voorbeeld geven. Veronderstel dat een programma gevraagd wordt dat de volgorde van gebeurtenissen regelt. Laat de specificatie gegeven zijn door een Boolese expressie P hetgeen betekent dat P slechts true is indien voor de enige erin voorkomende vrije variabele een toelaatbare volgorde der gebeurtenissen wordt gesubstitueerd. Wanneer P van de gedaante P0 $\wedge$ P1 is, dan is een van de programmeertechnieken:

— maak een programma met specificatie P0;
— maak een programma met specificatie P1;
— verbind deze twee programma's met de verwevingsoperator;
— het samenstel is een programma met specificatie P.

Een essentieel kenmerk van zo'n techniek is dat toepassing ervan leidt tot een stukje programmatekst plus een aantal programmeeropgaven die op dezelfde wijze geformuleerd zijn als de oorspronkelijke opgave, maar die hopelijk eenvoudiger zijn. Een tweede kenmerk is de onafhankelijkheid van wat P voorstelt en de afhankelijkheid van de syntactische gedaante van P. Nog duidelijker wordt dit bij technieken die van toepassing zijn op gedurige operaties, zoals sommatie of universele quantificatie (over een eindig domein). De techniek waar ik aan denk levert twee eenvoudiger programmeeropgaven op, een gerelateerd aan het eenheidselement en een gerelateerd aan een enkele toepassing van de operatie, plus verbindende programmatekst die slechts afhangt van het domein en niet van de operatie. Dit is een van de redenen waarom het plezierig is om van alle gedurige operaties ten minste het domein op gelijke wijze te schrijven.

Vaak zijn specificaties niet precies van zodanige vorm dat een van de programmeertechnieken van toepassing is en dan zal de specificatie herschreven moeten worden. U mag bij die herschrijvingen denken aan het toepassen van distributieve en aanverwante regels zoals we die uit de algebra kennen. Om het aantal regels te beperken is het weer plezierig dat constructies met gelijksoortige eigenschappen ook gelijksoortig worden genoteerd. Het herschrijven van specificaties is meestal zodanig beperkt dat de oorspronkelijke en de nieuwe vorm gelijke betekenis hebben. In het geval van Boolese expressies spitst onze aandacht zich dan ook meer toe op de equivalentie dan op de implicatie die van oudsher in de logica meer aandacht krijgt.

Het moge hiermee duidelijk zijn dat de studie van eigenschappen van de notaties waarin specificaties worden geformuleerd, van eigen-

schappen van de notaties waarin programma's worden geformuleerd, en van het verband daartussen een essentieel onderdeel van de informatica vormt. Regelmatig blijkt daarbij dat speciaal ontworpen notaties zich beter voor dit spel lenen dan de gangbare notaties. Dat is niet zo verwonderlijk want het formalisme speelt in veel andere vakken de wat meer ondergeschikte rol van het precies formuleren van eigenschappen en daarbij doet de gedaante van de formule nauwelijks ter zake. Die gedaante wordt pas van belang als er met de formules zelf gewerkt, gerekend gaat worden. Mocht u nog steeds aan het nut van goede notaties twijfelen, probeert u dan eens om aritmetiek te bedrijven met getallen die genoteerd zijn met Romeinse cijfers. Dat loopt niet van een leien dakje. In de Middeleeuwen stond dan ook als een paal boven water dat voor het bedrijven van aritmetiek een niet geringe hoeveelheid (natuurlijke) intelligentie vereist is. Al heel lang weten we echter dat het, vooral dankzij een betere notatie voor getallen, een mechanisch uitvoerbaar procédé van symbolenmanipulatie is. Achteraf lijkt het misschien een vanzelfsprekende kleinigheid maar ik beschouw het liever als een indrukwekkend staaltje van échte automatisering, als een voorbeeld waarvan veel voorstanders van kunstmatige intelligentie zouden kunnen leren wat het vak, ontdaan van charlatanerie, werkelijk behelst.

De genoemde activiteit van herschrijven van specificaties lijkt op het opstellen van de oorspronkelijke specificatie van het hele programma in die zin dat het wederom het introduceren van passende begrippen en notaties vergt om de specificatie in de gewenste vorm te krijgen, maar verschilt ervan in die zin dat er een aantoonbaar verband bestaat tussen de specificaties van de delen en van het geheel. Voor de volledigheid wil ik er op wijzen dat vertrouwdheid met het formalisme een machtig heuristisch hulpmiddel is, maar dat er daarnaast een zekere hoeveelheid kunde, inzicht, inventiviteit en intelligentie nodig is om uit alle alternatieven de meest geschikte herschrijving te kiezen, namelijk het alternatief dat tot de efficiëntste oplossing leidt, en om het arsenaal technieken uit te breiden. Terzijde een opmerking voor de sceptici onder u die zeggen: alles mooi en wel, maar het is vooral een kwestie van geluk om de juiste splitsing van problemen in deelproblemen te vinden. Tot u, en tot u alleen, zou ik willen zeggen: u heeft gelijk, maar u zult merken dat hoe meer u oefent, hoe meer u bewust oefent, hoe vaker u geluk hebt.

Ik wil nog even met u teruggaan naar de vergelijking tussen vliegtuig-

ontwerper en programmeur, en even stilstaan bij de kwaliteit, de betrouwbaarheid van beider producten. De vlietuigontwerper levert een voorschrift dat, wanneer het door een vliegtuigbouwer wordt uitgevoerd, tot een vliegtuig leidt. Het voorschrift bestaat uit een collectie tekeningen, materiaalbeschrijvingen, aanwijzingen over de tolerantie waarmee en volgorde waarin onderdelen gemaakt en geassembleerd worden. Als de fabricage ter hand wordt genomen dan kan het gebeuren, en in de praktijk gebeurt dat ook vaak, dat zekere onnauwkeurigheden in het ontwerp door de uitvoerenden worden opgemerkt. Meestal kunnen ze dan tijdig worden gecorrigeerd. De betrouwbaarheid van een vliegtuig wordt dus voor een deel bepaald door opmerkzaamheid tijdens de fabricage. Het geesteskind van de programmeur is een heel ander lot beschoren. Een programma is in een zo precieze notatie geformuleerd dat het mechanisch uitvoerbaar is, en als dat dan ook gebeurt kan van de zijde van de uitvoerende instantie geen intelligente opmerkzaamheid worden verwacht. Na eventuele fouten wordt, in de regel, doorgewerkt met onjuiste tussenresultaten die we echter niet onder ogen krijgen. Slechts met een flinke dosis geluk is het eindresultaat zo onwaarschijnlijk dat, bij het zien ervan, de gedachte opkomt het programma te verdenken. Overigens is dit fenomeen geen straf maar ontleent de apparatuur er juist zijn bruikbaarheid aan. Dit neemt niet weg dat bij het onderzoek naar notaties om programma's in te formuleren het uitsluiten van zekere fouten een relevant onderwerp is. Nauwkeurigheid bij het programmeren blijft echter een absolute eis.
Nog één opmerking tot besluit van de analogie. Alvorens een vliegtuig aan de klant wordt overgedragen wordt het aan een testprocedure onderworpen om na te gaan of het inderdaad aan de gestelde eisen voldoet. Het wordt onder enkele extreme condities beproefd en dat geeft meestal voldoende vertrouwen in het goed functioneren. Dat komt door het continue karakter van de diverse afhankelijkheden: als de draagkracht voor zowel 2 als 400 personen voldoende is, dan is hij ook wel voldoende voor elk tussenliggend aantal. Bij computerprogramma's is de situatie volstrekt anders door het discrete karakter van de gemanipuleerde objecten. Een sorteerprogramma dat zowel rijtjes met lengte 2 als rijtjes met lengte 400 correct sorteert kan voor, bijvoorbeeld, rijtjes van lengte 256 een geheel onverwacht resultaat produceren. Dit soort discontinuïteiten komt regelmatig voor. Omdat, zoals al gezegd, vaak nog een tijdje met verkeerde tussenresultaten wordt doorgewerkt is het vaststellen, en zeker het lokaliseren, van fouten geen sinecure. Wie op

door computers geproduceerde resultaten wenst te vertrouwen dient zich hierover meer zorgen te maken dan gewoonlijk gedaan wordt, ook wanneer de apparatuur in de ruimte is gestationeerd en hoogstens eenmalig 1 à 1½ uur actief hoeft te zijn.
Tot nu toe heb ik het met u gehad over de moeilijkheden van het ontwerpen van programma's zonder in te gaan op de vraag hoe die programma's en de objecten waarop zij opereren er uit zien. In principe is die vraag heel eenvoudig te beantwoorden: programma's en objecten vormen te zamen één lange, structuurloze sliert van nullen en enen. Het is juist de structuurloosheid die het mogelijk maakt programma's met behulp van betrekkelijk eenvoudige apparatuur te verwerken. Daarentegen levert de eerder beschreven wijze van programmeren juist programma's op die in hoge mate structuur bevatten. Die structuur is dan in het geheel niet in de programmatekst terug te vinden, maar slechts in het hoofd van de programmeur en in de begeleidende documentatie. Het kan daarom aantrekkelijk zijn om te zoeken naar notaties die enerzijds de mogelijkheid bieden programma's inclusief hun structuur te noteren en anderzijds de eigenschap hebben dat elk er in geformuleerd programma vertaald kan worden in een structuurloze tekst met dezelfde betekenis. Wanneer die vertaling mechanisch uitvoerbaar is wordt van automatisch programmeren gesproken omdat een deel van het werk van de programmeur wordt geautomatiseerd. Naarmate er meer wordt geautomatiseerd, wordt de programmanotatie een hoger niveau programmeertaal genoemd. De eigenschappen die een programmeertaal moet hebben om effectief te zijn worden niet alleen bepaald door efficiënte vertaalbaarheid maar vooral door de bij het programmeren toegepaste technieken. Mijn indruk is dat enkele van de eenvoudige, voor handen zijnde programmeertalen heel redelijk aansluiten bij onze programmeervaardigheden en dat het, daarom, voorlopig zinvoller is om het arsenaal technieken te herzien en uit te breiden dan om weer nieuwe programmeertalen te propageren. Een voorbeeld van een terrein waarop onze vaardigheden dienen te worden vergroot wordt gevormd door de objecten waarop programma's opereren. We zijn maar amper in staat die objecten, eventueel te zamen met een handjevol operaties er op, fatsoenlijk en hanteerbaar te beschrijven. De omgekeerde weg, het afleiden van de structuur van objecten uit hun specificatie, is nog goeddeels onbetreden. Een tweede voorbeeld van een terrein waarop nog veel werk verzet moet worden is dat van de gedistribueerde programma's. Hiervan is sprake indien het programma en de objecten waarop het opereert

worden verdeeld over een aantal gelijktijdig werkende machines. Wanneer er afhankelijkheden tussen de kavels bestaan is het nodig die afhankelijkheden te overbruggen door de machines te verbinden via een netwerk van communicatiekanalen. Het vinden van geschikte verkavelingen en netwerken is een moeilijke en boeiende activiteit. Dit onderwerp sluit aan bij een nog modernistischer onderwerp uit de informatica, nl. het implementeren van programma's in de vorm van VLSI circuits. VLSI (Very Large Scale Integration) is een techniek die het mogelijk maakt om een groot aantal transistoren en verbindingen onder te brengen in een chip, een klein plakje halfgeleidermateriaal. De aansluiting bij het onderzoek naar gedistribueerde programma's wordt gesuggereerd door de potentieel gelijktijdige activiteit van veel componenten van zo'n chip. Het idee is om een programma niet te vertalen in een homogene sliert van nullen en enen maar te vertalen in een homogeen vlak van transistoren en verbindingen. De homogeniteit in opbouw en realisatie van circuits is specifiek voor VLSI. Zoals zo vaak is het realiseren van dit eenvoudige idee niet zo eenvoudig. Efficiëntieoverwegingen wijzigen zo ingrijpend dat we opnieuw moeten leren programmeren. Een heel ander probleem is dat de vertaling gepaard gaat met een fabricageslag die zo onbetrouwbaar is dat de meeste chips niet werken. Het scheiden van kaf en koren is dus noodzakelijk én een levensgroot probleem. De overgang van een 1- naar een 2-dimensionaal medium maakt de vertaling bepaald niet makkelijker. De vertaling wordt ook bemoeilijkt door de discrepantie tussen het discrete karakter van programma's en het fysische karakter van chips. De microfysica vertoont een aantal discrete fenomenen die mij de hoop en het vertrouwen geven dat hier uiterst fascinerende fundamentele problemen en mogelijkheden zijn. Ik denk daarbij onder andere aan vragen naar het minimum aantal deeltjes in enige machine die een gegeven berekening kan uitvoeren, naar de minimaal benodigde hoeveelheid energie of rekentijd. Het is niet ondenkbaar dat de inzichten in de fysica veranderen door het onderzoek naar deze vragen. Feynman heeft bij herhaling verklaard

> 'It always bothers me that, according to the laws as we understand them today, it takes a computing machine an infinite number of logical operations to figure out what goes on in no matter how tiny a region of space, and no matter how tiny a region of time. How can all that be going on in that tiny space?'

en hieraan speculaties verbonden ten aanzien van de mogelijke onjuistheid der fysische wetten.

De ons meer vertrouwde macrofysica is continuer van aard en bij de vertaling van discrete programma's naar circuits dienen we ons restricties op te leggen. Een mogelijkheid is om ons te beperken tot circuits waarvan de correcte werking niet afhangt van de vertragingen die signalen in verbindingen zouden kunnen oplopen. Terzijde zij opgemerkt dat deze beperking ook om technologische redenen aantrekkelijk is en dat op dit gebied nog volop lol te beleven is, en daar gaat het tenslotte om.

Hiermee wil ik mijn bespiegelingen over programmeren, en daarmee over de kern van het vak informatica, afsluiten. Ik wil nog één vraag te berde brengen. Programmeren heb ik beschreven als een tamelijk formele, wiskundige activiteit terwijl in de praktijk veel programmeurs heel anders te werk gaan. Waar komt dat verschil vandaan? Er zijn vele antwoorden mogelijk, waarvan ik u er een paar geef.

— Veel professionele programmeurs zijn onvoldoende opgeleid, kunnen slecht met formalismen omgaan, en hebben niet geleerd op het goede abstractieniveau te denken. Verschillen in productiviteit van programmeurs belopen reeds factoren 10 à 20 en zullen verder oplopen naarmate programmeerprojecten ambitieuzer worden omdat de complexiteit sneller toeneemt dan de omvang.
— Veel managers van programmeerprojecten denken dat de problemen geen technische maar managementproblemen zijn. Dit is een ernstige misvatting.
— Specificaties worden niet op de geeigende manier geformuleerd. Ze zijn vaak informeel, vaag, inconsistent en onvolledig, en dienen daarom te worden gecorrigeerd voor het programmeren begint. Op drijfzand is het niet goed bouwen.
— Echte vernieuwingen worden vaak niet getolereerd. In de Middeleeuwen hielden de bankiers vast aan het gebruik van Romeinse cijfers. De consequenties van het werk van Cantor werden door Kronecker als zo ongewenst ervaren dat hij, rond 1885, zijn uiterste best deed om Cantor's loopbaan aan de Duitse universiteiten te dwarsbomen. Wanneer, naar analogie van de geautomatiseerde aritmetiek, een bepaald vak op een andere wijze wordt bezien is de reactie 'maar zó doen we dat niet' onvermijdelijk. De constatering is juist, het zou echter geen verwijt moeten zijn.
— Sommigen vrezen dat een wiskundiger wijze van programmeren duf en vervelend is. Onzin, het is juist hartstikke leuk. Je kunt het misschien jammer vinden dat het vak wiskundig van aard is en wiskun-

de de reputatie heeft moeilijk te zijn. Maar er is geen andere manier.

Aldus gekomen aan het einde van mijn beschouwing wil ik Hare Majesteit Koningin Beatrix mijn eerbiedige dank betuigen voor mijn benoeming aan deze universiteit.

Dames en heren leden van de universiteit,

Ik ben u zeer erkentelijk voor het in mij gestelde vertrouwen, dat mij ook enigszins verlegen maakt wanneer ik bedenk welke rol de informatica in onze maatschappij speelt. Ik hoop en vertrouw in goede verstandhoudig met u samen te kunnen werken.

Hooggeleerde Dijkstra, Kruseman Aretz en Rem,
Beste Edsger, Frans en Martin,

Aan mijn opleiding tot informaticus hebben jullie ieder op unieke wijze bijgedragen. Ik wil de verschillen nu niet uiteenzetten maar juist wijzen op jullie eensgezinde, inspirerende houding jegens de informatica. Ik ben jullie dankbaar voor het feit dat ik in dit vak terecht gekomen ben, voor jullie talrijke uitdagingen en scherpe inzichten, voor de vele voorrechten die ik genoten heb, en voor jullie enthousiasme waarmee mijn vorming van amateur tot prof gepaard ging.

Dames en heren studenten,

Ik maak van de gelegenheid gebruik mijn betoog af te ronden met enkele opmerkingen die speciaal tot u gericht zijn en het onderwijs betreffen. Is informatica wel te leren? Ik ben er vast van overtuigd dat dit vak te onderwijzen is, en ik ben er ook van overtuigd dat sommigen het nooit zullen leren. Als u toch mee in zee durft te gaan mag u eisen dat we in ons onderwijs verder gaan dan in deze wetenschapswinkel gebruikelijk is, namelijk dat we u niet alleen tastbare resultaten, stellingen, bewijzen presenteren maar dat we uitgebreid aandacht besteden aan de overwegingen die tot de resultaten of tot dwaalsporen hebben geleid in de hoop bij volgende ontwerppogingen ons meest waardevolle en schaarse goed, ons gezond verstand, effectiever te kunnen gebruiken. Verder zult u hard moeten werken en zelf aan onderzoek moeten

doen om de bij het programmeren broodnodige ervaring op te bouwen. We wijken dan wel af van een duidelijk merkbare, maar mijns inziens verkeerde, trend in het academisch onderwijs waarbij de nadruk steeds meer komt te liggen op kennen i.p.v. kunnen, op weten i.p.v. begrijpen. Als u niet bang bent om af te wijken van deze trend, bent u van harte welkom aan boord. U zult dan merken dat informatica een vak met een wiskundig karakter is, waarin enige ingenieursmentaliteit niet misstaat, en waarin theoretisch en praktisch geen tegenovergestelde begrippen zijn.

Dames en heren, ik dank u voor uw aandacht.