Studium 2 (2008) 128-144

Stijlen van programmeren 1952-1972

ADRIENNE VAN DEN BOGAARD*

In 1950 toog Willem Louis van der Poel (geb. 1926) naar het wiskundig instituut van de Universiteit van Cambridge om daar de beginselen van programmeren te leren van Maurice Wilkes. In 1951 deed Edsger Wybe Dijkstra (1930-2002) hetzelfde op instigatie van zijn vader. Maurice Wilkes had één van de eerste elektronische stored program rekenapparaten gebouwd, de zogenoemde *Electronic Delay Storage Automatic Calculator* (EDSAC). Het concept 'stored program' kwam voort uit de gedachte dat zowel de data als het programma ingevoerd werden in het geheugen van de rekenmachine. Dat was niet het geval geweest met de ponskaartenmachines. Daar bestonden programma's uit bekabelde eenheden die in de ponskaartenmachine werden geplugd: vervolgens werden de data – gerepresenteerd door ponskaarten – aan de machine aangeboden om te verwerken. In het geval van de stored program rekenmachine stonden zowel het programma als de data ergens in het geheugen en zorgde de besturing er bijvoorbeeld voor dat de juiste data uit het geheugen werden gehaald bij een bepaalde instructie.

Het concept van de stored program rekenmachine vereiste niet alleen de bouw ervan, maar ook programma's om uitgevoerd te worden, als die bouw eenmaal klaar was. Zodoende ontstond rond de eerste kamergrote rekenmachines, die nog niet over beeldschermen beschikten, het eerste programmeren.[1] Zoals in alle wetenschappen was dat iets nieuws wat niet zomaar op de plank klaar lag: programmeren moest tot ontwikkeling worden gebracht. Programmeren stond in de vroege jaren vijftig tussen aanhalingstekens, om aan te geven dat men nog niet goed wist wat het was. Een baken in de jaren vijftig vormde het eerste handboek dat over programmeren handelde, het boek dat Wilkes in 1951 publiceerde samen met zijn medewerkers David Wheeler en Stanley Gill over de grondslagen van programmeren *The preparation of programs for an electronic digital computer.*[2] Tevens gaven deze pioniers colleges programmeren.

Eenmaal terug in Nederland, ontwikkelden de oud-cursisten Van der Poel en Dijkstra, ieder hun eigen stijl van programmeren. In de periode 1952-1970 waren zij in Nederland belangrijke pioniers met betrekking tot de vorming van zoiets als programmeren en software, ofwel 'informatica'. Dijkstra ontving in 1972 zelfs de Turing Award, uitgereikt door de 'Association for Computing Machinery' (ACM).[3] Van der Poel en Dijkstra waren niet alleen in Nederland belangrijke pioniers, maar behoorden ook tot de internationale gemeenschap van computerpioniers. Zo waren ze beiden actief in de commissie van de 'International Federation for Information Processing' (IFIP) die de algoritmische hogere

*Tot 1 augustus 2008 verbonden aan de Technische Universiteit Delft. Vanaf die datum docent wiskunde aan het Hermann Wesselinkcollege te Amstelveen en post-doc in het Europese onderzoeksproject *Software for Europe* (UvA). E-mail: a.a.vandenbogaard@hotmail.com.

1 Zie de bijdrage in dit themanummer van G. Alberts en H.T. de Beer.

2 M. Wilkes, D. Wheeler, en S. Gill, *The Preparation of Programs for an Electronic Computer* (Reading Massachusetts 1951, 1957[2]).

3 Tegenwoordig definieert de 'Association for Computing Machinery' (ACM) zich veel breder als vertegenwoordiger van de mensen die werkzaam zijn in het hele gebied van de ICT.

programmeertaal ALGOL 68 ontwikkelde. Hierin speelde Adriaan van Wijngaarden zelfs een leidende rol.

De wetenschapshistoricus Chunglin Kwa past het stijlbegrip toe op de geschiedenis van de wetenschap. Hij stelt dat stijlen van wetenschapsbeoefening hun eigen criterium hebben voor wat goede wetenschap is. Stijlen zijn hun eigen rechtvaardiging. De zes stijlen die Kwa – in navolging van de wetenschapshistoricus Alistair Crombie – onderscheidt, zijn na hun ontstaan zichtbaar gebleven in de wetenschap. Kwa's stijlen zijn veelomvattende stijlen: ze spannen de wetenschapsgeschiedenis op.[4] Dit artikel is geen poging om Kwa's stijlen aan te vullen met een nieuwe stijl. Het neemt het idee van een stijl over, maar past het toe op kleinere schaal. Van der Poel en Dijkstra hadden ieder een eigen opvatting hadden over wat goed programmeren was. Deze verschillende stijlen zijn herkenbaar gebleven binnen de informatica. Elementen uit Van der Poel's stijl waren herkenbaar in de opkomst van de *Reduced Instruction-Set Computer* (RISC – geïnitieerd door IBM) – een poging de processor minder ingewikkeld te maken. Dijkstra's stijl is nog steeds herkenbaar in wat we theoretische informatica noemen. In de kern gaat het om de vraag: wat is een computer? Of beter gezegd, wat definieert een computer?

De opkomst van programmeren, 1952-1962

1. Van der Poel: de automaat als ondeelbaar apparaat

Van der Poel begon in 1947 met zijn studie natuurkunde aan de Technische Hogeschool in Delft. Naar eigen zeggen was hij altijd al geïnteresseerd geweest in hulpmiddelen bij het rekenen, omdat hij zelf slecht was in hoofdrekenen.[5] In 1947 ontmoette hij Adriaan van Wijngaarden, die toen hoofd was van de kersverse rekenafdeling van het Mathematisch Centrum te Amsterdam. Hij ontmoette hem bij een lezing van de Britse wiskundige en computerpionier Douglas Rayner Hartree (1897-1958). Deze had numerieke methoden toegepast op de toen nog maar net voor het Amerikaanse leger ontwikkelde *Electronic Numerical Integrator And Computer* (ENIAC). Hij was ook betrokken bij de ontwikkeling van de eerder genoemde EDSAC. Van Wijngaarden nodigde hem in 1949 uit om een lezing te houden op het 'Symposium over Moderne Rekenmachines', dat was georganiseerd door het Natuurkundig Genootschap.

Terwijl Van Wijngaarden inging op het fenomeen van een 'tafelrekenmachine, waaronder ook laboratoriumopstellingen vielen die niet als zodanig herkenbaar waren', legde Van der Poel zijn gehoor iets uit over de elektronica van automatische cijfermachines.[6] In zijn allereerste lezing legde Van der Poel uit wat somvormende schakelingen waren. Het volgende citaat toont hoe nieuw en belangrijk in die vroege periode de materialisering was:

> Eerst zullen we nu beschouwen van welke aard de somvormige verbindingselementen zullen moeten zijn en welke uitvoeringsvormen hiervoor bedacht kunnen worden. […] De logica leert, dat men alle verbindingen kan uitdrukken met behulp van twee grondverbindingen waarvoor men bijvoorbeeld de conjunctie en de negatie kan nemen. [...] We hebben nu nog een physische realisering nodig van deze grondverbindingen. Een van de meest veelzijdige elementen vindt men in het elektromagnetisch

4 C. Kwa, *De ontdekking van het weten* (Amsterdam 2005) 11-12.

5 W. L. van der Poel, *Een leven met computers* [uittree-rede] (Delft 1988).

6 W.L. van der Poel, 'Schakelingen van automatische cijfermachines', *Nederlandsch Tijdschrift voor Natuurkunde* 15 (1949) 255-264.

Edsger Wybe Dijkstra. Geboren in 1930, studeerde hij natuurkunde in Leiden. Zijn vader attendeerde hem op een zomercursus leren programmeren bij Maurice Wilkes in Cambridge. Dat werd het begin van een carrière in wat zou uitgroeien tot informatica. Hij promoveerde in 1959 aan de Universiteit van Amsterdam bij Adriaan van Wijngaarden. In 1962 werd hij hoogleraar in de numerieke wiskunde aan de Technische Universiteit Eindhoven. In 1972 ontving hij de Turing Award. In 1984 vertrok hij naar de Universiteit van Texas. Hij overleed in 2002.

> relais. […] Ook op elektronisch gebied zijn vele mogelijkheden te bedenken om de logische verbindingen te verwerkelijken.[7]

Van der Poel had ervaring met de problematiek rond materialisering. Hij ontwierp voor zijn afstuderen een apparaat, dat *Automatische Relais Calculator voor Optische berekeningen* (ARCO) heette en later de spotnaam TESTUDO (schildpad) zou krijgen omdat het apparaat zo traag was. Deze rekenmachine was ontworpen om de breking van stralen door lenzencombinaties uit te kunnen rekenen: dit was namelijk rekenintensieve arbeid.[8] Er was weinig geld waardoor het bouwen lange tijd duurde en een apparaat dat internationaal kon concurreren niet viel te realiseren. Het apparaat bleek echter erg betrouwbaar en zou tot 1964 in gebruik blijven.

In 1950 kreeg Van der Poel zijn eerste baan bij de Mathematische Afdeling van het Centraal Laboratorium van het Staatsbedrijf voor Posterijen, Telegrafie en Telefonie (PTT), op uitnodiging van de ingenieur en latere hoogleraar Leen Kosten, die directeur van de Mathematische Afdeling was. Het Centraal Laboratorium was in 1946 opgericht door dr. Lambertus Neher, directeur-generaal van de PTT, om aandacht te schenken aan 'gebieden der wiskunde, natuurkunde, scheikunde, schakeltechniek, digitale technieken, rekentuigen, vraagstukken postale automatisering en mechanisering en van automatisering van het girobedrijf'. In 1947 werd een mathematische afdeling gevormd die later 'informatica laboratorium' zou gaan heten.[9]

7 Van der Poel (n. 6) 258-259.
8 http://www.st.ewi.tudelft.nl/~poel/packages/curricul.44q, 1 (geraadpleegd 11-08-2008: deze link is ook via Wikipedia te vinden, op trefwoord Willem van der Poel).
9 H. L. Lommel, 'Een kwart eeuw Dr. Neher Laboratorium', *Het PTT-Bedrijf* (1972) 1-11.

Dit laboratorium ontwierp en bouwde de *PTT Electronische Reken-Automaat* (PTERA). Van der Poel benaderde de rekenapparatuur als een geheel van elektronica en programmering. Het één (het ontwerp van de elektronica) en het ander (de programmering) waren intrinsiek met elkaar verbonden en vormden in onderlinge samenhang zijn onderzoeksobject. Hij beschreef dan ook voor de PTERA zowel de 'werking' als de 'programmering'.[10] In zijn artikel over de werking nam hij letterlijk de elektronische circuits op van de logische bewerkingen die de rekenautomaat moest kunnen uitvoeren. In zijn artikel over de programmering (waarvan de eerste versie in de vorm van een uitgebreid rapport al in november 1952 binnen de PTT was verspreid) zette hij de methode van programmeren uiteen.

Een probleem moest – volgens de conventies van die tijd – eerst in een stroomdiagram worden vertaald. Het stroomdiagram was de stap tussen de wiskundige of verbale beschrijving van een berekening en de code. Code verwees in de jaren vijftig naar de representatie van een programmaregel. Code kon binair zijn, maar dat hoefde niet: begin jaren vijftig raakten al assembleercodes in zwang. Assembleercodes waren herkenbaarder codes dan de binaire codes. In plaats van een rijtje nullen en enen kon dan bijvoorbeeld 'Add' geschreven worden voor 'optellen'. Om de assembleercode te vertalen in de binaire code die de rekenautomaat kon uitvoeren waren dan subprogramma's nodig.[11] Volgens Van der Poel was de volgende stap na het stroomdiagram:

> het opstellen van de instructiereeksen die nodig zijn om de handelingen in de doosjes van het stroomdiagram uit te voeren. Het is hiervoor nodig op de hoogte te zijn van de mogelijkheden van de machine, d.w.z. van het effect dat de verschillende instructies hebben.[12]

Voor Van der Poel bestond er een intrinsieke relatie tussen de elektronica en de inhoud van de instructie. De gebruiker diende die elektronica te kennen. Van der Poel die zijn programmeerkennis en technieken had opgedaan bij de groep van Wilkes in Cambridge, verwees aan het eind van zijn artikel dan ook naar Wilkes boek als een 'uitgebreid leerboek over programmeren'. Van der Poel paste bijvoorbeeld 'subprograms' toe, stukjes programma's die als het ware universeel konden worden gebruikt, zoals worteltrekken. Elke keer als in een hoofdprogramma een wortel moest worden getrokken kon hetzelfde subprogramma worden aangeroepen. Ook paste hij interpretatief programmeren toe, eveneens door Wilkes als eerste beschreven. Interpretatief programmeren betekende dat dezelfde instructie verschillend kon worden geïnterpreteerd. Een voorbeeld is dat een vermenigvuldiging van twee reële getallen in een andere reeks instructies uiteenvalt, dan een vermenigvuldiging van twee complexe getallen. Als de automaat wist hoe die een vermenigvuldiging moest interpreteren, dan paste hij of de eerste, of de tweede variant toe.

10 W. L. van der Poel, 'PTERA II. De werking van de PTERA' en 'PTERA III. Het programmeren van de PTERA', *Het PTT-Bedrijf* 5:4 (december 1953) 124-134; 135-147.

11 Herman Goldstine kent het gebruik van het stroomdiagram toe aan John von Neumann: 'To help bridge this gap between the mathematician's description of the desired computation in mathematical language and the corresponding program in program code, Von Neumann invented the flow diagram'. John von Neumann introduceerde het stroomdiagram in een artikel uit 1946: 'Planning and Coding of Problems for an Electronic Computing Instrument'. Zie: H. Goldstine, 'Introduction', in: W. Aspray & A. Burks (eds.), *Papers of John von Neumann on Computing and Computer Theory.* Charles Babbage Institute Reprint Series for the History of Computing 12 (Cambridge & Londen 1987) 148.

12 Van der Poel (n. 10) 135.

Binnen het Centraal Laboratorium van de PTT hield de Mathematische Afdeling zich in de jaren vijftig bezig met de ontwikkeling van een 'universele computer'. Het hoofd van de afdeling, Leen Kosten (staand), trok hiervoor Willem van der Poel (zittend aan de ponser) aan. Dit leidde tot de 'PTT electronische reken automaat' (PTERA). De schakeltechniek bestond voor het merendeel uit radiobuizen. Een typemachine diende als uitvoerapparaat, en de ponsbandlezer (rechts op de tafel) als invoerapparaat. Een dergelijk kamergroot apparaat had geen beeldscherm. Vroege computerpioniers luisterden naar hun machine hoe het ermee ging tijdens het draaien van een programma.

Wilkes beschrijft in zijn memoires hoe hij in juni 1949 een duizelingwekkend gevoel kreeg door het besef dat hij de rest van zijn leven bezig zou zijn om zijn eigen programma's foutloos te maken.[13] De techniek van de subroutine (door Van der Poel en Kosten subprograms genoemd) was hier onder andere uit voortgekomen: een subroutine die getest was op zijn juiste werking, hoefde daarna nooit meer opnieuw te worden getest.[14] Ook Kosten en Van der Poel besteedden aandacht aan dit aspect van fouten in programma's: 'In de regel zal blijken dat de machine wat anders doet dan men had verwacht'.[15] Als oplossing hadden Van der Poel en Kosten een 'geheugen-uittypprogramma' schreven dat de inhoud van registers achter elkaar uittypte. Op die manier konden programmeerfouten worden opgespoord. In hun ogen konden programmeerfouten alleen worden geëlimineerd door het programma in de machine in- en daarna weer uit te voeren.

13 M.V. Wilkes, *Memoirs of a Computer Pioneer* (Cambridge & London 1985) 145.
14 Subroutines scheelden natuurlijk ook tijd. Ze voorkwamen ook de sleur in het programmeren, zoals Dijkstra en Van Wijngaarden het in hun programmeerdictaat noemden.
15 Leen Kosten, 'PTERA IV: "A7"', *Het PTT-bedrijf* 5:4 (december 1953) 148-163. Citaat 154.

De opkomst van programmeren, 1952-1962
2. Dijkstra: de geboorte van de programmeur

Edsger Dijkstra studeerde natuurkunde in Leiden, toen zijn vader hem een advertentie uit *Nature* onder de neus duwde over een cursus programmeren in Cambridge. En Dijkstra ging.[16] Bij terugkomst hoorde Van Wijngaarden daarvan en hij nodigde Dijkstra uit om bij het Mathematisch Centrum te komen werken. Mensen met ervaring met computers, laat staan met een cursus programmeren bij Wilkes op zak, waren schaars. Dijkstra was bovendien gymnasiast en daar hield Van Wijngaarden van. Dijkstra startte bij het Mathematisch Centrum in 1952 en verliet het tien jaar later toen hij in Eindhoven hoogleraar werd in de numerieke analyse. In zijn eerste jaar bij het Mathematisch Centrum maakte Dijkstra een functionele beschrijving van de tweede versie van de *Automatische Relais Rekenmachine Amsterdam* (ARRA), een 'Automatische Digitale Machine' zoals hij die omschreef (zie de vorige bijdrage in dit themanummer), en met dit werkstuk uit 1953 zette hij direct de toon:

> In het volgende wordt de machine beschreven, voor zover dit van belang is voor degene, die hem *gebruikt*: er zal beschreven worden *wat* de machine doet, niet *hoe* die werkt.[17]

In Dijkstra's beschrijving stonden geen elektronische circuits – die hielden hem niet bezig. Die circuits hoefden hem ook niet bezig te houden, want hij hoefde zich binnen de rekenafdeling van het Mathematisch Centrum niet met de bouw te bemoeien. Dijkstra schreef deze handleiding ruimschoots vóórdat de ARRA-II klaar was: die kwam pas in 1954 gereed. Hij richtte zich allereerst tot de gebruiker van de machine, die alle instructies moest kennen. Maar daarna ging hij een stap verder en nam ook een hoofdstuk 'De taak van de programmeur' op. Daarmee was hij in Nederland waarschijnlijk de eerste die de programmeur een naam en een taak gaf. Dijkstra benoemde de taak van een programmeur als volgt:

1. Mathematische formulering van het probleem,
2. Mathematische oplossing van het probleem,
3. Keuze of constructie van numerieke processen die tot het gewenste antwoord leiden,
4. Programmering: gedetailleerde opbouw van de onder 3 genoemde processen uit de elementaire bewerkingen, waartoe de machine direct in staat is,
5. Codering uitschrijven van het programma in de code der machine zodat hierna de band onmiddellijk geponst kan worden.[18]

Dijkstra haalde programmering en codering uit elkaar: het niveau van de programmering achtte hij nog dermate algemeen, dat hierover 'dingen gezegd zouden kunnen worden die niet alleen voor de ARRA van toepassing zijn'.[19] Dat is een belangrijke stap: Dijkstra

16 Edsger Dijkstra, 'From my Life', 7 pp, getypt (1993). Downloaden via het digitale 'E.W. Dijkstra Archive', op: www.cs.utexas.edu/users/EWD/ewd11xx/EWD1166.pdf (geraadpleegd op 11-08-2008).
17 E.W. Dijkstra, 'Functionele beschrijving van de ARRA', *Mathematisch Centrum Rapport-MR-12* (Amsterdam 1953) 1.
18 Ibidem, 33-35.
19 Ibidem, 33.

probeerde uitspraken over programmeren te doen, die computeronafhankelijk waren. Dat toont al in 1953 zijn manier van denken en werken.

In het hoofdstuk over de programmeur gaf Dijkstra voorbeelden van wat dat soort algemeenheden waren: 'het streven naar *veiligheid* uit zich in de voorzorgen, dat de machine geen fout antwoord aflevert'.[20] Hoe kon de programmeur weten dat het antwoord van de Automatische Digitale Machine klopte?

> Door een *mathematische check* in te voeren, d.w.z. men laat de machine naast de eigenlijke berekening een accessoir resultaat afleveren waarvan men weet dat het antwoord klopt. […] Iets dergelijks is echter niet in alle gevallen elegant mogelijk.[21]

In dit afsluitend zinnetje liet Dijkstra het woord 'elegant' terloops vallen. Het streven naar veiligheid – de zekerheid dat de antwoorden juist waren – moest liefst elegant worden opgelost. Er moest een controle worden ontwikkeld die voorkwam dat er fouten in het programma zaten.

In context van de vraag van dit artikel – wat is goed programmeren? – zien we dat in die allervroegste periode programmeren als activiteit tot ontwikkeling werd gebracht rondom de eerste generatie – éénstuks rekenautomaten, en dat er over geschreven werd, wat men deed, en hoe de gebruiker de machine voor zich kon laten werken. Er werden problemen beschreven en hoe die zouden moeten worden opgelost. Het voorbeeld van de omgang met betrouwbaarheid toont dat Van der Poel en Dijkstra verschillende oplossingen voorstonden. In die vroege tijd werd betrouwbaarheid gezien als een probleem van de elektromechanische en elektronische onderdelen van het apparaat. Relais weigerden regelmatig en vacuümbuizen gingen kapot. In de jaren zestig stonden 'computers', zoals ze toen zouden worden genoemd, nog steeds tien tot vijftien procent van de tijd stil ten gevolge van 'hardware' falen.[22]

In het kader van de betrouwbaarheid kwamen Van der Poel en Kosten met een instrument, een speciaal programma – het geheugen-uittypprogramma – om aan de hand van tussentijdse output te kunnen controleren of het programma deed wat het geacht werd te doen. We zouden kunnen zeggen dat ze een ingenieursoplossing kozen: ze ontwikkelden een testinstrument. Aan de hand van outputresultaten konden ze reconstrueren wat de machine precies deed met elke instructie. Als die dan iets anders deed, dan wat ze hadden verwacht, dan konden ze het programma aanpassen. We zouden het een typisch voorbeeld van *trial-and-error* kunnen noemen.

Dijkstra haalde de kwestie van betrouwbaarheid uit de context van de hardware, en definieerde het als een algemeen probleem, waarvan de oplossing tot het domein van de programmeur behoorde. De programmeur moest oplossingen bedenken om te weten dat zijn programma klopte. Daarmee creëerde hij een nieuw beroep, met eigen vraagstukken, onafhankelijk van de elektronica. Bovendien hield hij ervan als de oplossingen 'elegant' waren. De mathematische check had nog niet zijn volledige instemming – immers, stel dat de check uitwees dat het programma niet klopte, wat dan? Hoe kwam je er dan achter wat er niet klopte?

20 Ibidem, 34.
21 Ibidem.
22 Gert. C. Nielen, 'De toekomst van de elektronische rekenapparatuur', *Mens en computer* (Utrecht 1963) 109-124.

Overigens was betrouwbaarheid niet het enige onderwerp waar de gebruikers mee worstelden: gebrek aan geheugenruimte was een ander probleem. Dat gebrek aan geheugenruimte vormde – zou je kunnen zeggen – een extern motief voor Van der Poels stijl. Gebrek aan geheugenruimte kon soms met 'kunstgrepen' worden opgelost. Van der Poel en Kosten lieten bijvoorbeeld toe dat de betekenis van een 'bit' op een specifieke plaats in het woord, afhankelijk was van de waarde van het bit ervoor.[23] Daardoor kon je optimaliseren: als je ergens maar 12 opties voor nodig had, maar je had 16 mogelijkheden, dan kon je die vier andere opties voor andere functies gebruiken. Het spreekt vanzelf dat elektronica en programmeren zodoende naadloos in elkaar overliepen.

Optimum-programmeren garandeerde dat geen bit betekenisloos bleef. Optimum-programmeren – dat was iets wat de pioniers van het Mathematisch Centrum echter liever vermeden. Een programmeur moest zich niet bezighouden met dit soort elektronische (materiële) overwegingen, maar met de schoonheid van de abstractie.

Elektronisch rekenen wordt wetenschap, 1956-1962

1. Van der Poel en zijn trucologie: programmeren met computers

In 1956 hield Van der Poel voor het Colloquium Rekenmachines een van zijn eerste lezingen over het *Zeer Eenvoudig Binair Reken Apparaat*, de ZEBRA. Wat lezen we plotseling als ontwerpcriterium bij de ZEBRA?

> The coding had to be extremely flexible. Indeed, for attaining the greatest simplification, most of the complicated operations are programmed instead of being built in. For example, even multiplication and division are not built in.[24]

Dit citaat toont dat de relatie tussen elektronica en programmering object van studie geworden was. Van der Poel had er in zijn ontwerp van de ZEBRA voor gekozen dat vermenigvuldigen tot het domein van het programma behoorde en niet als elektronisch circuit in de machine aanwezig hoefde te zijn. Van der Poel had de weg ingeslagen om de elektronica zo eenvoudig mogelijk te houden (elektronica was immers nog steeds een bron van problemen). Dat was ook het thema van zijn proefschrift waarop hij in 1956 bij Van Wijngaarden promoveerde. De titel van zijn proefschrift verraadt het al: *The Logical Principles of Some Simple Computers* stelde als vraag hoe eenvoudig een 'automatic computing machine' kon zijn, zonder functionaliteit als zodanig in te leveren.[25] Van der Poel koos voor een zo eenvoudig mogelijke uitrusting van de elektronica. In 1959 zou hij daarover zeggen dat hoe eenvoudiger de constructie is, hoe betrouwbaarder de machine immers is, en hoe minder onderhoud hij dus nodig heeft.[26] Dit idee lag aan de basis van een geheel eigen stijl van programmeren: de trucologie.

23 Van der Poel (n. 10).

24 Willem van der Poel, 'Logical structure of the ZEBRA' [Lezing 27 februari 1956], herdrukt in: *Nederlands Rekenmachine Genootschap 1959-1969: Colloquium moderne rekenmachines. Uitgegeven ter gelegenheid van het tweede lustrum van het Nederlands Rekenmachine Genootschap in samenwerking met het Mathematisch Centrum* (Amsterdam 1969) deel 2, 48-53, i.h.b. 48.

25 W. van der Poel, *The Logical Principles of Some Simple Computers* (Amsterdam 1956). Proefschrift Universiteit van Amsterdam.

26 W.L. van der Poel, 'The simple code for ZEBRA', *Het PTT-bedrijf* 9:2 (augustus 1959) 31-66.

In 1952 had hij al een voorontwerp gemaakt van het *Zeer Eenvoudig Reken Orgaan* (ZERO) en deze zelf gebouwd. Hij bouwde deze machine ten tijde van de constructie van de PTERA (zie vorige paragraaf). De ZERO werd weer afgebroken toen de PTERA definitief in elkaar werd gezet. Het ontwerp van de ZERO week af van wat op dat moment de standaard was. De eerste elektronische rekenmachines bestonden achtereenvolgens uit een besturingsorgaan, een geheugenorgaan, een rekenorgaan en enige input- en output organen. Ook had een machine een bedieningspaneel, ook wel 'console' genoemd. Een rekenorgaan bestond doorgaans uit twee of drie registers: één voor het getal waar iets mee moest gebeuren, één voor het resultaat, en één voor (bijvoorbeeld) het getal waarmee vermenigvuldigd moest worden. Besturingsorgaan en rekenorgaan waren strikt van elkaar gescheiden. Van der Poel bedacht echter iets nieuws: hij integreerde de het besturingsorgaan en het rekenorgaan. Soms diende de accumulator letterlijk als rekenorgaan voor de uit te voeren berekening; maar de accumulator werd tevens ingezet om de opdracht 'tel één op bij de waarde van de geheugenplaats van de instructie' uit te kunnen voeren. Tegelijkertijd moest dit geïntegreerde orgaan zowel in staat zijn om een instructie uit te voeren, als om een instructie uit het geheugen op te halen. De kern was het dubbel gebruik van de 'opteller': die was nodig, zowel om de volgende instructie te kunnen ophalen, als om de rekenkundige bewerking uit te kunnen voeren.

De juiste werking van het elektronische schema werd mogelijk gemaakt door een nieuwe manier van programmeren, namelijk het gebruik van 'functionele bits'. Een eerste bit gaf bijvoorbeeld aan of de instructie 'het ophalen van de volgende opdracht uit het geheugen' inhield, of dat het een andersoortige instructie betrof zoals 'optellen' of 'opslaan' (een bewerking van een getal). De functionele bits bepaalden dus het elektronische schema, en zodoende bepaalden ze wat met het getal in de instructie gebeurde.[27] Met de invoering van deze functionele bits plaatste hij zichzelf in de traditie die door Konrad Zuse was ingezet. Zuse was een Duitse computerpionier die zowel computers bouwde als ook bijdragen leverde aan programmering. Hij werkte onder andere aan een 'minimale instructieset'.[28] Van der Poel en Zuse hadden een goede verstandhouding: Zuse zocht hem een keer op bij de PTT.[29]

Van der Poel ontwierp de ZEBRA op basis van de ervaring die hij met de ZERO had opgedaan – het minimaliseren van het aantal elektronische componenten om het onderhoud zo eenvoudig mogelijk te houden en de betrouwbaarheid te vergroten. Vervolgens moest hij op zoek naar een producent. De PTT wilde zelf niet als rekenmachineproducent optreden. De PTT wilde Van der Poel en Kosten ook niet kwijt in een te stichten commercieel bedrijf. Philips kon niet op de productie ingaan vanwege de afspraak met IBM dat zij geen computers zouden gaan produceren. Uiteindelijk zou een Engels telefoniebedrijf, 'Standard Telephones and Cables' (Stantec) genaamd, de computer in 1958 op de markt brengen. De eerste versies waren nog op basis van elektronenbuizen, de latere

27 W.L. van der Poel, 'A Simple Electronic Computer', *Applied Scientific Research* B2 (1952) 367-400.

28 Zie bijvoorbeeld: Raúl Rojas en Ulf Hashagen (eds.), *The First Computers. History and Architectures* (Cambridge Massachusetts 2000) Dit boek bevat een deel over de geschiedenis van pionierscomputers in Duitsland, waaronder die van Konrad Zuse.

29 Voor dit onderdeel is gebruik gemaakt van een ongepubliceerd werkdocument uit het door de Stichting PAOi gesteunde UvA-project 'Computerpioniers, pionierscomputers': H. T. De Beer 'Bouw en gebruik van rekenapparaten bij de Mathematische Afdeling van het Centrale Laboratorium der PTT', Werkdocument d.d. 3 september 2007. Contact dr. Gerard Alberts: g.alberts@uva.nl.

op basis van transistoren. Het Centraal Laboratorium kreeg al een eerste exemplaar in 1957, dat daar zou worden gebruikt tot 1967.[30] Overigens promoveerde Van der Poel in 1956 op het ontwerp van (onder andere) de ZEBRA, met als leidende vraag hoe eenvoudig de elektronica van een rekenmachine kon worden uitgerust teneinde toch alle gewenste berekeningen via programmering te kunnen maken.

In 1961 gaf Van der Poel zijn programmeerstijl een naam en noemde het 'trickology'.[31] De kern ervan was dat je precies moest weten hoe je als programmeur de elektronica kon bespelen. Hoe beter je die in de greep had, hoe handiger je was met de ZEBRA. Van der Poel en de zijnen verhieven de 'truc' tot iets vernuftigs, een typische ingenieursstijl. Hoe krijgt de zo slim mogelijk iets gedaan van die computer? Het idee kwam dus voort uit de beperkingen die aan de machine waren opgelegd in het kader van het vergroten van de betrouwbaarheid.[32]

> The concept of micro-programming and the practice of devising tricks to do the more complicated composite actions is so interwoven in ZEBRA that the volume of knowledge of these tricks has been given a special name: *trickology.* Without this knowledge of trickology and the standard programs based on it, ZEBRA would be a useless machine.[33]

Voor Van der Poel waren zijn ZEBRA en zijn trucologie intrinsiek met elkaar verbonden: je zou kunnen zeggen dat de computer bestond uit zijn elektronica gecombineerd met de wijze van programmeren – die waren ondeelbaar. In de publicatie waarin Van der Poel de trucologie uiteen zette, verontschuldigde hij zich dan ook tegenover zijn lezers dat zij eerst de werking van de ZEBRA moesten doorgronden.[34] Op diverse universiteiten in Nederland werd deze stijl van programmeren tot ontwikkeling gebracht, waaronder in Groningen op het Wiskundig Instituut en in Delft.

Elektronisch rekenen wordt wetenschap, 1956-1962:
2. Dijkstra: programmeren zonder computers

Van Wijngaarden en Dijkstra gaven zelf programmeercursussen vanaf 1951 en in 1955 verscheen het eerste dictaat waarin ze in de titel het programmeren ontkoppelden van het apparaat: *Programmeren voor Automatische Rekenmachines.* Het was één van de belangrijke drijfveren achter het werk op het Mathematisch Centrum, om programmeren onafhankelijk te maken van het elektronische apparaat. In dit programmeerdictaat schreven zij:

> Het belangrijkste is echter wel, dat in het programma *wel* staat, wat er *gebeurt*, maar *niet* wat dit nu allemaal *behelst.* … Kortom, er is een behoefte aan een overzichtelijke weergave van programma's, een notatie waarin, dankzij verzwijging van de bijkomstigheden, de essentialia niet verdrinken, en waar

30 Van de ZEBRA-computers zijn zo'n 50 exemplaren verkocht, een goed resultaat voor een Europese computer.
31 W. van der Poel, 'Micro-programming and Trickology' in: W, Hoffman ed., *Digitale Informations Wandler* (Braunschweig 1961) 269-311.
32 Ibidem. Dit idee om te streven naar een 'eenvoudige machine' was een internationale discussie, waarin onder andere Zuse participeerde.
33 Van der Poel (n. 31) 274.
34 Ibidem, 272.

> tevens de *functie* van de stukken programma in aangegeven is. […] De notatie der zogenaamde *blokschema's* (= flow diagrams) voorziet in deze behoefte. Dat deze notatie geen direct gebruik maakt van de opdrachtencode van de betroffen machine, verzekert een niet te verwerpen algemeenheid van de blokschema's. Anderzijds is de notatie iets minder efficiënt bij uitgekookte, 'getructe' programma's: bij standaardprogramma's … bij welker opstelling men het onderste uit de kan wil halen, zal men immers niet schromen, geraffineerd van de – vaak onbedoelde! – speciale eigenschappen van de machine gebruik te maken.[35]

Uit dit citaat kan worden afgeleid dat Dijkstra en Van Wijngaarden streefden naar een wijze van communiceren over programma's waarbij de machine zelf er niet toe deed. Sterker nog, die specifieke eigenschappen van de machine werkten alleen maar verwarrend. Alle regels code die betrekking hadden op het ophalen van data uit het geheugen, of juist het laden in het geheugen, leidden af van waar het werkelijk om ging – het uiteindelijke algoritme dat aan het programma ten grondslag lag. Zij plaatsten blokschema's in een nieuw perspectief – het was een eerste poging machineonafhankelijk over een programma te kunnen communiceren.

Dit streven stond haaks op de 'getructe programma's' waartegen ze zich dan ook verzetten. Dit verzet tegen wat ook wel 'optimum-programmeren' werd genoemd – een techniek van programmeren waarbij het optimaal gebruiken van de snelheid en van het geheugen een expliciet doel was – had ook het ontwerp van de *Automatische Rekenmachine Mathematisch Centrum* (ARMAC) geïnspireerd. De ARMAC kwam in 1956 gereed:

> When we think instructions to be put on a magnetic drum, and we would like to reduce access times as much as possible without using some kind of optimum-programming, we can do that by reserving a short amount of fast memory for those instructions.[36]

We zien hier dat een probleem in het gebruik van rekenmachines – de traagheid van het geheugen – werd opgelost in 'hardware', er werd een stukje snel geheugen in de machine geplaatst. Dat was beter dan dat de programmeur zijn programma ging aanpassen aan dat langzame geheugen zodanig dat die toch kon profiteren van de snelheid van de machine.

In 1959 promoveerde Dijkstra bij Van Wijngaarden op het 'interrupt mechanisme', dat tevens was gematerialiseerd in de Electrologica X1, die in 1958 op de markt gekomen was: net als de ZEBRA was ook dit een tweede generatie commercieel geproduceerde computer, alleen van Nederlandse bodem.[37] Dijkstra's proefschrift droeg als titel: *Communication with an Automatic Computer*. Dit was een internationaal probleem waar Dijkstra zich op had geworpen: in de regel was de computer zelf veel sneller dan de input- en output apparatuur. Elke keer als een instructie betrekking had op input- of output apparatuur bestond de kans dat die apparatuur nog bezig was een vorige instructie uit te voeren, waardoor de computer zelf moest wachten. Dat was jammer van de tijd.

35 A. van Wijngaarden en E. Dijkstra, *Programmeren voor Automatische Rekenmachines* (Amsterdam 1955) 5-1, 5-2.

36 B. Loopstra (1956). Lezing opgenomen in de herdenkingsbundel: *Nederlands Rekenmachine Genootschap 1959-1969* (n. 24).

37 Vgl. de bijdrage van Alberts en De Beer in dit themanummer.

Dit probleem stond bekend als het probleem van de synchronisatie.[38] Hoe kon je er voor zorgen dat de computer instructies kon uitvoeren, terwijl tegelijkertijd de input- en output apparatuur hun opdrachten konden uitvoeren?

Het was Dijkstra's stijl van werken die hem de oplossing bracht: hij abstraheerde van de machine zelf en definieerde een algemeen probleem. Hij hield van onderstrepingen om zijn gedachten duidelijk te maken (zoals we al eerder zagen) en ook van het woord: 'algemeen'. (Van der Poel hield van het woord: 'eenvoudig'. In het Engels gebruikte hij het woord: 'simple'). Problemen die zich voordeden met het rekenen met automaten dienden algemeen onder woorden te worden gebracht:

> We prefer to direct our attention to a general problem that arises in connection with the problem of synchronisation. […] The so-called 'interruption' has been built into the machine to execute a specific communication program within a period of time, that is independent of the main program.[39]

De kern ervan was dat een input- of outputapparaat aan de computer vertelde dat het klaar was voor een volgende opdracht. Dan kon een volgende instructie voor het input- of outputapparaat worden uitgevoerd. Ondertussen had het hoofdprogramma gewoon door kunnen gaan.

Het kunnen spreken over programma's, zonder je te bemoeien met de details van de elektronica, stond ook aan de wieg van de ontwikkeling van de algoritmische hogere programmeertaal 'Algorithmic Language' (ALGOL). Dit was een taal om algoritmes te kunnen communiceren zonder dat ze door de computer behoefden te worden uitgevoerd. Van Wijngaarden en Dijkstra waren vanaf 1959 intensief betrokken bij de ontwikkeling van ALGOL. Taal kon worden geconstrueerd en dat was iets nieuws. Dat onderzoek had veel te danken aan het boek *Syntactic Structures* van Noam Chomsky, dat taal onder andere ontleedde in syntaxis en semantiek.[40]

Net als in het geval van het probleem van synchronisatie stond ook het onderzoek naar hogere programmeertalen van begin af aan in een internationaal kader. Het zoeken naar manieren om de programmering eenvoudiger te maken, was zo oud als de digitale elektronische computer zelf. Er was een ontwikkeling gaande waarin pioniers trachtten zoveel mogelijk begrippen in de computertaal toe te laten die leken op natuurlijke taal. In de assembleertaal was dat bijvoorbeeld 'Add' of 'Store'. Wilkes had aan de wieg gestaan van het idee dat een instructie als 'Add' *automatisch* vertaald kon worden in binaire taal, dat wil zeggen dat er een programma mogelijk zou moeten zijn die de assembler code zou vertalen naar nullen en enen. Dat was bijzonder modern, want aanvankelijk dacht men dat het vertalen van assembler code naar binaire code menselijke arbeid zou zijn. David Wheeler, een wiskundige en één van de auteurs van het eerder genoemde leerboek over programmeren, realiseerde inderdaad dit 'automatische' programma'.[41]

Onderzoek naar hogere programmeertalen, zoals 'Common Business Oriented Language' (COBOL – de taal voor administratieve gegevensverwerking), 'Formula Translator' (FORTRAN – de taal voor technisch-wetenschappelijk rekenen) en ALGOL stond in die

38 Zie bijvoorbeeld het STRETCH project van IBM.

39 E. Dijkstra, *Communication with an automatic computer*, Proefschrift Universiteit van Amsterdam 28 oktober 1959, 28.

40 N. Chomski, *Syntactic Structures* (Den Haag en Parijs 1957).

41 M. Campbell-Kelly & W. Aspray, *Computer. A History of the Information Machine* (New York 1996) hoofdstuk 8.

traditie, waarin gezocht werd naar manieren om, enerzijds het de programmeur mogelijk te maken te kunnen programmeren in enigszins begrijpelijke taal, terwijl anderzijds het vertaalde programma in binaire code voldoende efficiënt moest zijn. Dit onderzoek vond plaats onder de noemer 'automatic programming systems' of autocoding. Zo publiceerde het onderzoeksteam naar hogere programmeertalen van International Business Machines (IBM), dat onder leiding stond van John Backus, in 1957 het eerste rapport over FORTRAN: *The FORTRAN automatic coding system.*[42]

Bij onderzoek naar hogere programmeertalen ging het dus zowel om de definitie van de taal zelf, als ook om de automatische vertaling ervan naar binaire code; iets wat we tegenwoordig onder 'compiling' verstaan. Het was precies met betrekking tot die automatische vertaling, dat Dijkstra internationaal aanzien verwierf:

> Recursive programming' by Professor Dijkstra is an early and important contribution to the art of writing compilers. The problems involved in permitting recursive calls on subroutines are attacked and handled in a simple elegant fashion. Almost everyone involved in writing an Algol compiler has used some of the ideas developed in connection with the Algol Compiler written by professor Dijkstra and his colleagues at the Mathematical Centre in Amsterdam.[43]

En elegante oplossingen, dat wilde Dijkstra. In 1960 ontwikkelde hij samen met Jaap Zonneveld de eerste ALGOL-compiler. Dijkstra definieerde een manier om met geheugen om te gaan tijdens de vertaling van het programma naar de machinecode: de zogenaamde 'stack'. In meer gemoderniseerde vorm zou dit interruptmechanisme later onderdeel worden van wat in de jaren zestig 'besturingssystemen' zouden gaan heten. Het interruptmechanisme kwam voort uit het nadenken over het rekenproces zelf en daar op een hoger niveau over theoretiseren. In toenemende mate zou Dijkstra 'rekenen' als een mentaal proces beschouwen.

In de periode 1956-1962 ontwikkelden Dijkstra en Van der Poel zich elk op een voor hen kenmerkende wijze. Beiden vormden zich rond een tweede generatie computer (resp. de X1 en de ZEBRA) . Hun probleemdefinities verschilden echter. Van der Poel beschouwde de computer als een intrinsiek met elkaar verbonden geheel van elektronica en programmering. Rond zijn trucologie ontstond een geheel eigen gemeenschap van mensen die de trucs met elkaar uitwisselden. Van der Poel was ook betrokken bij de ontwikkeling van Algol, en ontwikkelde een Algol-compiler voor de ZEBRA. Hij was echter geen principieel aanhanger van Algol. Als het voor bepaalde toepassingen handiger was om andere talen te gebruiken, moest de gebruiker dat vooral doen. Voor de ZEBRA bestonden ook een Fortran- en een Cobol-compiler. Voor X1 werd ook een Fortran-compiler gebouwd, maar niet door de programmeurs van het Mathematisch Centrum. Die werd gebouwd door de programmeerafdeling van Electrologica (die de X1 computer produceerde).

Wilde Van der Poel programmeren met computers, Dijkstra wilde juist het tegenovergestelde: programmeren moest tot wetenschap worden gemaakt. Dat kon door programmeren als een wiskundige activiteit te definiëren. Programmeren zonder computers, dat was zijn ideaal. Dat heeft hem overigens zelf in een depressie gebracht, want

42 J. W. Backus e.a., 'The Fortran Automatic Coding System', *Proceedings of the Western Joint Computer Conference* (1957). Reprinted in: S. Rosen, *Programming Systems and Languages* (New York 1966) 29-47.
43 S. Rosen (n. 42) 181.

als natuurkundige was hij zelf niet in logica geschoold. Liefst had hij vervolgens vanuit inzichten die uit die wiskundige arbeid voortkwamen, het computerontwerp beïnvloed:

> The solution (of the stack, AvdB) can be applied under perfectly general conditions, e.g., in the structure of an object program to be delivered by an ALGOL 60 compiler. The fact that the proposed methods tend to be rather time consuming on an average present day computer, may give a hint in which direction future design might go.[44]

Twee programmeerstijlen, 1962-1972

In 1962 werden beide pioniers Van der Poel en Dijkstra hoogleraar, respectievelijk in Delft en in Eindhoven. Twee dagen na elkaar hielden ze hun intreeredes. Beiden weigerden ze het Engelse woord 'computer' te gebruiken en schreven ze afwisselend over 'rekenautomaten', 'logische automaten' en 'automatische rekenmachines'.

In zijn inaugurele rede riep Van der Poel zijn toekomstige studenten op plezier te beleven aan het puzzelen dat zijn vak met zich mee zou brengen. Hij werd aangesteld in de faculteit Elektrotechniek als degene die studenten programmeren moest leren:

> Er is ongetwijfeld een grote gelijkenis tussen de schakeltechniek die zich bezighoudt met het ontwerpen van de hardware en de wiskunde en de logica van programmeersystemen aan de zijde van de software. De schakeltechniek mag zich aan de Technische Hogeschool op een grote belangstelling verheugen. Ik heb dus goede hoop dat er voldoende puzzelaars en pluizers onder U zullen zijn die er behagen in scheppen om zich in het vak van programmering te gaan verdiepen. Want ik kan U verzekeren dat het element van puzzle en spel bij rekenmachinetechniek een grote rol spelen. […] Ik hoop er veel toe bij te dragen U deelgenoot te maken in deze typische ingenieurswetenschap die techniek is en kunst tegelijk.[45]

Eerder had Van der Poel in zijn rede uitgelegd wat 'software' was: het samenspel van assembleer-, compileer-, en interpreteerprogramma's, talen, vertalers en operationele systemen. Het maken van deze software was evenzeer constructie als het maken van de machine zelf.[46] De term 'software' verschijnt begin jaren zestig, zowel in Nederland als internationaal. De uitdrukking verwijst naar het in toenemende mate automatiseren van handelingen rond het programmeren die aanvankelijk door de programmeur/operateur moesten worden verricht. Als voorbeeld noemde Van der Poel de constructie dat een computer automatisch overstapt op een ander programma zodra deze door een programmeerfout vastloopt in het oorspronkelijke programma.

Van der Poel benoemde de essentie van het programmeren als 'puzzelen'. Programmeren was in zijn ogen een ingenieurswetenschap. Omdat hij was aangesteld in de schakeltechniek, behoefde hij zich geen zorgen te maken over de kennis van de automaten zelf: dat leerden de studenten in elk geval. Hij moest geïnformeerde elektrotechnici enkel het programmeren leren.

44 E. Dijkstra, 'Recursive Programming', *Numerische Mathematik* 2 (1960) 312-318.

45 W. van der Poel, *Talen en kunsttalen, Rede uitgesproken bij de aanvaarding van het ambt van bijzonder hoogleraar in de toegepaste logica als grondslag van structuur en gebruik van rekenautomaten aan de Technische Hogeschool te Delft* (Delft 1962) 17.

46 Ibidem 14.

Voor de nieuwe beroepsvereniging 'Stichting Studiecentrum voor Administratieve Automatisering' (SSAA) ontwikkelde Van der Poel een hypothetische code – de SERA-code – die in 1969 werd gepubliceerd, maar die al sinds 1963 werd gebruikt in programmeercursussen die de SSAA organiseerde. De SSAA wilde objectief zijn: onafhankelijk van producenten. Deze SERA-code was een binaire code, terwijl inmiddels diverse hogere programmeertalen onderwezen hadden kunnen worden. Dit had deels als achtergrond dat cursisten niet getraind moesten worden in een specifieke computer of taal, zodat de SSAA niemand zou bevoordelen. Maar Van der Poel ontwikkelde deze binaire code ook vanwege zijn eigen programmeerstijl:

> Zoals men om wiskunde te studeren het niet zonder gedegen kennis van het rekenen kan stellen, zo kan ook de programmeur het niet stellen zonder kennis van de werking van de machine in technisch en programmatisch opzicht. […] De bouw van de SERA heeft weer een goed inzicht gegeven in de uitwisselbaarheid van hardware tegen software, van materiële realisering tegen immateriële realisering op een andere machine met behulp van programma's.[47]

Zoals we zagen was het voor Van der Poel essentieel om de elektronica en de programmering als één geheel te beschouwen. Dit aspect van zijn stijl zou op diverse momenten in de informatica opnieuw belangrijk worden. In de jaren zeventig zouden de 'Machine Oriented Higher Level Languages' een onderzoeksdomein zijn waarin opnieuw de grens tussen hardware en software object van onderzoek was.[48] Het meest recente voorbeeld zijn de 'embedded' systemen, waar programmering en elektronica opnieuw tot elkaar zijn veroordeeld. In de wereld van embedded systemen worden nu cursussen ontwikkeld om informatici (die weinig van elektrotechniek weten) de basis van elektrotechniek bij te brengen.[49] Als we moeten benoemen welke waarden voor Van der Poel sturend waren, dan waren dat 'bruikbaarheid' en 'creativiteit'.

Dijkstra had een geheel andere opvatting over programmeren: dit was bovenal een mentale, wiskundige activiteit. Programma's definieerde hij als abstracte mechanismen. Dijkstra's oproep om van informatica een wiskundige discipline te maken, vond veel weerklank. In toenemende mate zou Dijkstra bekendheid verwerven met zijn stelling dat de correcte werking van programma's bewezen diende te worden. Met 'testen' kon men nooit bewijzen dat een programma correct was, gezien de beperkte hoeveelheid toestanden die een programmeur überhaupt in staat was te testen.

Met zijn besturingssysteem voor de computer van Electrologica zou Dijkstra opnieuw bekendheid verwerven. In het artikel 'The Structure of the 'THE'-multiprogramming System', bracht hij als wapenfeit het *a-priori* correctheidsbewijs van het systeem naar voren. Dijkstra had gezocht naar een manier om definitief fouten te voorkomen:

47 W.L. van der Poel, 'Sera, een machine voor studiedoeleinden', *Informatie* nr. 27 (juli 1963) 41-42. Zie ook: idem, 'SERA, een gesimuleerde machine', *Mededelingen Nederlands Rekenmachine Genootschap* 6:5 (1964) 6-17 en idem, *SERA 69: Definiërend rapport onder verantwoordel kheid van de Sera-Kommissie* (Amsterdam 1969).

48 Zie bijvoorbeeld W. A. Wulf, 'Issues in Higher-Level Machine Oriented Languages' in: W.L. van der Poel, L. Maarssen (eds.), *Machine Oriented Higher Level Languages* (Amsterdam 1974) 7-18. (Proceedings van een IFIP-conferentie gehouden in 1973).

49 Vriendelijke mededeling Jochen Jess (24 september 2007).

> This decision […] I regard as the group's main contribution to the art of system design. We have found that it is possible to design a refined multiprogramming system in such a way that its logical soundness can be proved a priori and its implementation can admit exhaustive testing. […] We shall have proved the correctness of the system with a rigor and explicitness that is unusual for the great majority of mathematical proofs.[50]

In 1976 publiceerde Dijkstra zijn boek *A Discipline of Programming*. Dat was een hoogtepunt in zijn intellectuele ontwikkeling tot dan toe. Exemplarisch voor Dijkstra's wijze van denken, is het volgende citaat:

> Historically speaking, the fact that programming languages could be used as a vehicle for instructing existing automatic computers, has for a long time been regarded as their most important property. The efficiency with which existing automatic computers could execute programs written in a certain language became the major quality criterion for that language! As a regrettable result, it is not unusual to find anomalies in existing machines truthfully reflected in programming languages, this at the expense of intellectual manageability of the programs expressed in such a language. […] I view a programming language primarily as a vehicle for the description of (potentially highly sophisticated) abstract mechanisms. […] We redress the balance (between efficiency and intellectuality) by regarding the fact that our algorithms could actually be carried out by a computer as a lucky accidental circumstance that need not occupy a central position in our considerations.[51]

Dijkstra's proefschrift uit 1959 was gebaseerd geweest op zijn kennis van de Electrologica X1. Het proefschrift zelf ging over een probleem dat hij gedefinieerd had als abstract probleem, geldend voor alle computers. Bovengenoemd citaat, dat zeventien jaar later werd neergeschreven, toont de continuïteit met zijn vroege werk. De computer zelf is hoogstens een toevallige bijkomstigheid. Dijkstra zocht naar algemeen geldende uitspraken en vond die door het dynamische proces van de computer zelf tot object van studie te maken. In 1972 zou hij dat 'executional abstraction' noemen. Het object van de informatica was de studie van abstracte mechanismen en correctheidbewijzen vormden daarbij zijn belangrijkste methode. 'Elegantie' was daarin een leidend principe.

Besluit

Het begrijp stijl maakt het mogelijk verschillen in programmeren niet alleen als cognitief te duiden, maar ook als sociaal-cultureel en normatief. De context van het Amsterdamse Mathematisch Centrum, waar het adagium van dienstbaarheid al snel op de achtergrond verdween ten gunste van een hoog niveau van wiskunde, was een geheel andere context dan die van de Mathematische Afdeling van het Centraal Laboratorium van de PTT. Dijkstra paste bij het Mathematisch Centrum, zoals Van der Poel hoorde bij de PTT. Beiden werden geleid door verschillende opvattingen over wat mooi was: voor Dijkstra zat schoonheid in een elegant bewijs, in abstractie. Voor Van der Poel zat de schoonheid in een 'truc' – een nog iets betere oplossing die optimaal gebruik maakte van de karakteristieke eigenschappen van een machine. Van der Poel beschouwde de weten-

50 E. Dijkstra, 'The structure of the 'THE'-multiprogramming system', Manuscript EWD 196 (zie n. 16). Gedrukt in: *Communications of the ACM* [Association of Computer Machinery] 11:5 (1968), 342-346, i.h.b. 342.

51 E. Dijkstra, *A Discipline of Programming* (Englewood Cliffs, New Jersey 1976).

schap veel meer dan Dijkstra als een collectieve onderneming: hij bedankte altijd iedereen en anno 2008 besteedt hij nog steeds tijd aan het schrijven van artikelen over de pioniers die met hem samenwerkten.

Dijkstra en Van der Poel gebruikten niet alleen verschillende programmeermethoden, ze beschouwden de computer op een andere manier; ze stonden anders in de wetenschap en werkten in verschillende contexten. In die zin staan ze symbool voor een grotere dichotomie, namelijk die tussen de ingenieurs die vuile handen maken (Van der Poel) en de wetenschappers die in abstractie het hoogste doel zien (Dijkstra). Het is dan ook niet toevallig dat Dijkstra onder hedendaagse vakgenoten een bekend man is, die bijvoorbeeld bekend is gebleven om het ontwerp van algoritmes (zoals het kortste-pad-algoritme), zijn multiprogrammeringsssyteem en zijn oproep om van een programmeur een wiskundige scholing te verwachten, terwijl Van der Poel veel minder bekend is gebleven. Het is een Nederlandse spiegel van het verschil in academisch prestige tussen de wiskundige John von Neumann en de ingenieurs Presper Eckert en John Mauchly, die gedrieën werkten aan de ENIAC in 1945.

SUMMARY

Styles of programming 1952-1972

In the field of History of Computing, the construction of the early computers has received much scholarly attention. However, these machines have not only been important because of their logical design and their engineering, but also because of the programming practices that emerged around these first machines. This article compares two styles of programming that developed around Dutch 'first computers'. The first style is represented by Edsger Wybe Dijkstra (1930-2002), who would receive the Turing Award for his work in 1972. Dijkstra developed a mathematical style of programming – a program was something you should be able to design mathematically and prove it logically. The second style is represented by Willem Louis van der Poel (born 1926). For him, programming is 'trickology'. A program is primarily a technical artefact that should work: a program is something you play with, comparable to the way one solves a puzzle.