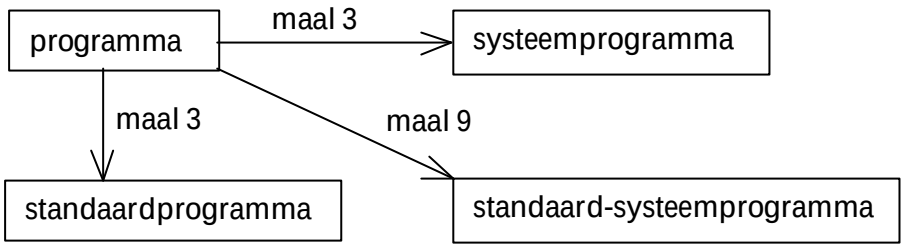
DE MYTHE VAN DE MAN –MAAND

Oorspronkelijk : « The Mythical Man-Month », Frederick P. Brooks, 1975
Samenvatting : Andy Verkeyn, 1997

De Teerkuil

Af en toe verschijnen er in de kranten artikelen waarin wordt verhaald hoe twee programmeurs in een verbouwde garage een belangrijk programma hebben gemaakt, dat de meest geslaagde pogingen van grote teams overtreft. Maar waarom worden deze industriële teams dan niet allemaal door enthousiaste teams in verbouwde garages vervangen ? Heel eenvoudig : omdat je moet kijken naar wat er wordt gemaakt.



- programma : Op zichzelf volledig, klaar om door de maker te worden gebruikt. Dit is wat er in de garage vervaardigd wordt.
- standaardprogramma : Kan door iedereen gebruikt worden, gegeneraliseerd, gedocumenteerd, grondig getest, alle mogelijke controles op correcte invoer...
- systeemprogramma : Maakt deel uit van een geintegreerd programma, gelijke user-interface, overeenkomstige hard- en software eisen, orthogonaal gebruik in heel het systeem.
- standaard-systeemprogramma : traditioneel gemaakt door industriële teams

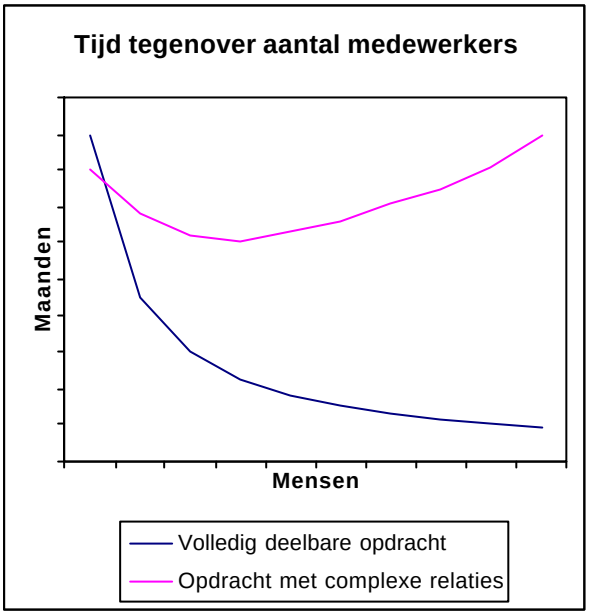
De Mythe van de Man-Maand

Computerprogrammeurs zijn geboren optimisten, die altijd hetzelfde zeggen : “Deze keer gaat het zeker goed”, “Ik heb er net het laatste foutje uitgehaald”.

De man-maand

Het hanteren van de man-maand al eenheid voor het meten van de omvang van een taak is een gevaarlijke en misleidende mythe. Was die mythe een feit, dan zou dit inhouden dat de factor man en de factor maand onderling verwisselbaar zijn.
Mensen en maanden zijn alleen onderling verwisselbare goederen als een taak kan worden opgedeeld aan een groot aantal werknemers die onderling niet behoeven te communiceren (bv. plukken van katoen).
Als een taak niet kan worden verdeeld omdat er sprake is van volgtijdelijke eisen waaraan moet worden voldaan (bv. het opsporen en corrigeren van fouten), dan heeft het aanwenden van meer inspanning geen effect op het tijdschema.
Bij taken die verdeeld kunnen worden maar ook communicatie (dit omvat zowel de training van de nieuwe werkkracht als de onderlinge communicatie, die exponentieel stijgt met het aantal teamleden) tussen de verschillende deelopgaven vereisen (bv. de bouw van software)

kan de toegevoegde inspanning de verdeling van de oorspronkelijke taak volledig teniet doen (zie grafiek).



Systeemtest

Vuistregel voor de planning van een softwareproject :

- 1/3 planning
- 1/6 uitschrijven van de code
- 1/4 testen van de onderdelen
- 1/4 testen van het hele systeem

Wet van Brooks

Het vergroten van de mankracht van een vertraagd softwareproject maakt dit nog trager.

Het operatieteam

Het voorstel van Harlan Mills

Ieder segment van een grote taak moet aan een apart team worden toegewezen. Per team is er slechts één meester programmeur (auctor intellectualis) dit de functionele specificaties vastlegt en de volledige verantwoordelijk draagt.

Schaalvergroting

Door elk onderdeel door zo'n klein team te laten ontwikkelen moeten voor het grote systeem uiteindelijk slechts een 20-tal meester programmeurs rond de tafel gaan zitten.

Aristocratie, democratie en systeemontwerp

Getrouwheid aan het oorspronkelijk ontwerp

Getrouwheid aan het oorspronkelijk ontwerp bij het ontwerpen van een systeem zou de meest op de voorgrond tredende overweging moeten zijn. Een systeem waarbij een aantal afwijkende kenmerken en verbeteringen met opzet achterwege is gelaten, maar dat een gerichte strategie weerspiegelt, verdient verre de voorkeur boven een systeem dat tal van goede of zelfs uitmuntende ideeën bevat, maar waaraan interdependentie en coördinatie ontbreken.

Ontwerpgetrouwheid waarborgen

Omdat gebruikersvriendelijkheid het sleutelwoord is geworden, is deze verhouding van functie tot ingewikkeldheid van het ontwerp de ultieme proef op de som geworden. Noch alleen de functie, noch alleen het gemak van de gebruiker bepalen de kwaliteit van het systeemontwerp.
Eenvoud en directheid komen voort uit getrouwheid aan het oorspronkelijk ontwerp. Ieder onderdeel moet een afspiegeling geven van dezelfde onderliggende filosofieën en van dezelfde onderlinge afweging van desiderata.

Aristocratie en democratie

Getrouwheid aan het oorspronkelijk ontwerp schrijft op haar beurt voor dat het ontwerp door één auctor intellectualis moet zijn bedacht, of door een gering aantal deskundigen wier denkwijzen met elkaar overeenstemmen.
De tijdsdruk schrijft echter voor dat er bij het bouwen van een systeem een groot aantal mensen wordt betrokken. Er zijn twee manieren om dit dilemma te ontwijken :

- zorgvuldige taakverdeling tussen architectuur (wat er moet gebeuren) en constructie (hoe het moet gebeuren)
- samenstelling van het uitvoerend team

Zijn de architecten nu op weg om een nieuwe eigen aristocratie te gaan vormen, een intellectuele elite, die de domme massa van de uitvoerders wel eens zal voorschrijven wat zij moeten doen ? Het antwoord is ja en nee.

- Ja in die zin dat slechts een klein aantal mensen architect kan zijn. Wil een systeem een basis van integriteit bezitten, dan moet iemand de grondslagen beheersen.
- Nee omdat het vastleggen van de externe specificaties niet meer creativiteit vereist dan het ontwerpen van de tenuitvoerlegging. Alleen heeft deze creatieve inbreng een ander karakter. Het ontwerpen van de uitvoering aan de hand van een gegeven architectuur vereist en veroorlooft even veel creativiteit, even veel nieuwe ideeën en even veel technisch vernuft van het hoogste niveau als het ontwerpen van de externe specificaties.

Wat doet de constructeur terwijl hij zit te wachten

Wordt het voorstel naar voor gebracht om een klein team van architecten het hele eisenpakket voor een computer- of een programmeersysteem te laten schrijven, dan worden er van de kant van de constructoren drie bezwaren geopperd :

- Het eisenpakket zal te veel functies bieden en zal geen afspiegeling vormen van de praktische overwegingen van “wat het allemaal zal kosten”.
- De architect gaat met de eer strijken, omdat hij het creatieve deel van het werk verricht en de inventiviteit van de implementoren wordt onbenut gelaten. Dit is een puur

hersenspinsel want zoals eerder gezegd, worden de mogelijkheden om binnen het implementeren creatief en inventief te werk te gaan niet noemenswaardig verminderd als binnen het kader van een gegeven externe eisenpakket moet worden gewerkt.
- De implementoren, groot in aantal, zitten werkeloos te wachten totdat het eisenpakket door de nauwe sluis van het architectenteam wordt geloodst. Ook dit is niet realistisch : het inschakelen van uitvoerders kan uitgesteld worden totdat het eisenpakket geheel voltooid is. Bovendien kunnen ontwerp en uitvoering elkaar gedeeltelijk overlappen.

Beter is de grootste vijand van goed

Samenspel en discipline

In de praktijk is het zo dat een permanent contact tussen architect en uitvoerder, mits in een vroeg stasium tot stand gebracht, de architect een juist inzicht in de kosten van uitvoering kan verschaffen en de uitvoerder vertrouwen in de kwaliteit van het ontwerp kan schenken, zonder dat een duidelijke verdeling van de verantwoordelijkheidhierbij uit het zicht wordt verloren. Wordt de architect met een te hoge schatting van de kosten geconfronteerd, dan zijn er twee reacties mogelijk :
- hij verschraalt het ontwerp
- hij probeert aan de kostenschatting te tornen door een goedkopere uitvoering voor te stellen. Hierbij moet hij echter rekening houden met het feit dat de uitvoerder zijn eigen verantwoordelijkheid draagt in inventiviteit en creativiteit. De architect mag mogelijke implementaties voorstellen maar hij mag niet proberen de werkelijke implementatie te dicteren. In de regel zal de uitvoerder als tegenzet voorstellen tot wijzigingen in het architectonisch ontwerp doen. Hij zal daarbij niet zelden het gelijk aan zijn kant hebben.

Zelfdiscipline

Het ontwerpen van zijn tweede systeem is de meest riskante opdracht die de architect in zijn hele loopbaan kan krijgen. Iedere ontwerper vertoont de neiging zijn tweede systeem mooier te maken dan nodig is, en het met alle toeters en bellen te versieren die bij het eerste systeem even in de kast waren gezet.

Waarom is de toren van Babel nooit afgebouwd

Communicatie binnen een groot programmeerproject

Hoe zou de interne communicatie tussen de verschillende teams dan idealiter moeten verlopen ? Het antwoord luidt : zo vaak en zo ruim als dit mogelijk is.
- Informeel : Het is van essentieel belang dat de implementator, die zich voor een raadsel gesteld ziet, weet dat het hem ten alle tijde vrij staat zijn vraag per telefoon aan de architect voor te leggen – dit verdient de voorkeur boven gissen en raden van zijn kant.
- Formeel mondeling overleg : Regelmatige vergaderingen
- Logboek : Alle documenten van het project dienen in deze structuur te worden opgenomen.

De tijdrit

De gegevens van Nanus en Farr

inspanning = constante x aantal_instructies^{1,5}

De gegevens van Portman

Enkel de helft van de werktijd van programmeurs wordt effectief besteed aan het programmeren en het testen. Vergaderingen, de onvermijdelijke papierwinkel, zakelijke contacten binnen het bedrijf, ziekte,... neemt de andere helft in beslag.

De gegevens van Aron en Harr

De produktiviteit bij het ontwikkelen van compilers (600 à 800 geteste instructies per man-jaar) ligt driemaal zo laag als deze voor het maken van applicatieprogramma's (2000 à 3000 geteste instructies per man-jaar). Het ontwikkelen van besturingssystemen is op zijn beurt ook driemaal zo traag als deze voor compilers.

De gegevens van Corbato

De produktiviteit van de programmeur kan tot vijf keer toe worden verhoogd als een passende taal van hoog niveau wordt gebruikt.

Vijf kilo in een zak van vijf pond

Representatie is de kern van het programmeren

Achter het vakmanschap ligt de vindingrijkheid, daar staat de wieg van krappe, zuinige en snelle programma's. Deze programma's zijn bijna altijd in hogere mate aan een strategische doorbraak dan aan tactische slimheid te danken. Soms is de strategische doorbraak een nieuw algoritme, veel vaker komt die echter voort uit een omwerking van de representaties van gegevens.
De programmeur die radeloos is omdat hij gebrek aan geheugenruimte heeft, doet er dikwijls het beste aan zich uit de strikken van zijn eigen code te bevrijden, een forse sprong achterwaarts te doen en zijn gegevens helemaal opnieuw te overzien. Representatie is de kern van het programmeren.

Een plan om weg te gooien

Proeffabrieken en schaalvergroting

Bij de meeste projecten is echter het eerst-gebouwde systeem nauwelijks bruikbaar. Of het is te traag, of het is te groot, of het is te onhandig, of het is alle drie tegelijk. Er is maar één alternatief en dat is : helemaal opnieuw beginnen - terechtgewezen maar wijzer - en een herzien ontwerp te bouwen, waarin genoemde problemen tot een oplossing zijn gebracht. Het weggooien en opnieuw ontwerpen kan in één keer gebeuren, of beetje bij beetje. Maar de ervaring met alle grote systemen leert dat het onherroepelijk zal gebeuren. Zodra er een nieuw systeemontwerp of een nieuwe technologie wordt toegepast, kun je maar het beste meteen een systeem bouwen dat voor de prullenmand bestemd is, want zelfs de beste ontwerper is niet zo alwetend dat hij de eerste keer raak schiet.

De onstandvastigheid staat vast

Cosgrove heeft scherpzinnig opgemerkt dat een programmeur niet in de eerste plaats een tastbaar produkt, maar een bevrediging van de behoefte van de klant moet afleveren. En zowel de behoefte van het moment zelf als de inzichten en de reacties van de klant zullen veranderingen ondergaan gedurende de periode waarin een programma wordt gebouwd, getest en gebruikt.

Het zij verre van mij de gedachte te wekken dat alle veranderingen in de doelstellingen van de klant en alle geformuleerde eisen in het ontwerp zouden moeten of kunnen worden opgenomen. Het is duidelijk dat er een bepaalde drempel moet worden vastgesteld en deze moet, naarmate de ontwikkeling voortgang vindt, steeds hoger worden - anders zou geen enkel project het stadium van voltooiing bereiken.

De hinkstapsprong

Een programma houdt niet op te veranderen als het aan de klant wordt afgeleverd. De veranderingen die na de levering plaatsvinden heten programmaonderhoud.
De totale kosten van het onderhoud van een op grote schaal gebruikt programma bedragen in de regel 40% of meer van de kosten van de ontwikkeling van dit programma.
Merkwaardig genoeg worden deze kosten sterk door het aantal gebruikers beïnvloed. Hoe groter het aantal gebruikers, des te groter het aantal fouten dat zij constateren.
Het fundamentele probleem van het programmaonderhoud is dat je iedere keer dat je er een fout uithaalt een flinke kans (20 tot 50%) loopt dat je er een nieuwe fout inbrengt.
Het is duidelijk dat wanneer het ontwerpen van het programma zo gebeurt dat neveneffecten worden uitgeschakeld of tenminste eruitspringen, dit een enorm gunstig effect op de kosten van onderhoud zou opleveren. Dit kan gebeuren door implementatiemethodieken toe te passen, waarbij minder mensen, minder raakvlakken en dus ook minder fouten een rol spelen.

Een stap naar voor en een stap terug

Alle reparaties lijken op de vernietiging van de structuur te zijn gericht en de entropie en wanorde van het systeem te doen toenemen. Er wordt steeds minder aandacht besteed aan het repareren van zwakke plekken die uit het ontwerp stammen, er wordt steeds meer tijd besteed aan het herstellen van fouten die ten gevolge van vroegere reparaties hun intrede hebben gedaan. Met het verstrijken van de tijd boet het systeem voortdurend aan ordening in. Vroeg of laat lijken de reparaties geen enkele terreinwinst meer te geven. Er moet dan een herzien ontwerp komen, dat van de grond af steen voor steen wordt opgebouwd.
"De dingen zijn in het begin altijd op hun best" (Pascal Blaise)
Het bouwen van systeemprogrammatuur is een proces dat de entropie doet afnemen en daarom metastabiel is. Het programmaonderhoud is een proces dat de entropie doet toenemen, en zelfs de kunstigste uitvoering kan niet meer doen dan het vertragen van een systeemverzakking tot een onherroepelijke staat van aftakeling.

Scherp gereedschap

Tijdtoewijzing

Het foutvrij maken van een systeem is altijd al gedaan onder het lopen van de hondewacht, net zoals de beoefening van de astronomie. Twintig jaar geleden, raakte ik al vertrouwd met de informele atmosfeer van de produktieve uren voor zonsopgang, als alle chefs thuis nog op één oor liggen, terwijl de operators minder geneigd zijn zich aan de regeltjes te houden.

Simulatoren

We zijn er langzamerhand aan gewend geraakt dat onze computer hardware bijna op elk moment nauwkeurig functioneert. Tenzij een toepassingsprogrammeur daadwerkelijk ziet dat een systeem zich van een bepaalde run naar dezelfde inconsistent gedraagt, doet hij er goed aan eerder de fouten in zijn code, en niet in zijn computer te zoeken.

Deze ervaring vormt echter een slechte voorbereiding op het programmeren van de ondersteuning voor een nieuwe machine. Hardware die uit het laboratorium, de preproduktie komt, of de eerste exemplaren van hardware zijn niet betrouwbaar en blijven niet van dag tot dag dezelfde. Naar gelang er fouten worden gevonden, worden er in alle exemplaren van de machine, en dus ook in die van het programmeerteam, technische veranderingen aangebracht. Dat de basis zo verschuift is al erg genoeg. Maar gebreken van de hardware, die alleen maar van tijd tot tijd optreden, zijn nog veel erger. Het allerergste is de onzekerheid, want deze berooft je van de drang om ijverig in de code naar een fout te gaan zoeken, want er zou ook wel eens geen fout kunnen zijn ! Een betrouwbare simulator op een machine die zijn sporen heeft verdiend blijft dus veel langer zijn nut behouden dan je zou verwachten.

Programmabibliotheken

Aanvankelijk had iedere groep of iedere programmeur een gebied waar hij kopieën van zijn programma's, zijn proefnemingen en het steigerwerk dat hij voor het testen van onderdelen nodig had, bewaarde. In dit eigen speelhoekje had men alle vrijheid van handelen : het was een strikt eigen terrein.
Was een onderdeel zo ver gereed dat het in een groter deel kon worden geïntegreerd, dan werd een kopie hiervan overgedragen aan de manager van dat grotere systeem, die deze kopie in de onderbibliotheek voor de systeemintegratie onderbracht. Nu kon dit onderdeel niet meer door de oorspronkelijke programmeur worden veranderd, tenzij met toestemming van de integratie-manager. Terwijl het systeem groeide, was laatstgenoemde bezig met allerlei typen systeemtests, met het localiseren van fouten en met het aanbrengen van verbeteringen.
Op gezette tijden werd er een systeemversie, die ver genoeg ontwikkeld was, bevorderd naar de onderbibliotheek voor de lopende versie. Zo een kopie was hoogst allerheilig en mocht alleen worden ontwijd als er fatale fouten uit moesten worden gehaald. De kopie was beschikbaar voor de integratie en het testen van alle versies van de nieuwe modules.

Hogere programmeertaal

De verbetering op het terrein van het testen komt voort uit het feit dat de fouten geringer in aantal en bovendien gemakkelijker op te sporen zijn. Het aantal fouten is geringer omdat je een hele laag van foutgevoeligheid vermijdt op het niveau waarop fouten niet alleen syntactisch, maar ook semantisch van aard zijn, zoals het foute gebruik van registers. De fouten zijn gemakkelijker te vinden en – dit is nog van meer belang – omdat het invoegen van testsnapshots veel eenvoudiger is.

Het geheel en de delen

Een foutvrij ontwerpprincipe

- Foutvrij maken van de definitie : De fouten die het meeste onheil aanrichten en het moeilijkst te traceren zijn, zijn systeemfouten die voortkomen uit strijdige aannamen door de auteurs van de verschillende onderdelen. Met het streven naar getrouwheid aan het oorspronkelijk ontwerp, wordt deze problematiek bij de wortel aangepakt.
- Top-down ontwerp :
  * Een methodiek van verfijningsstappen
  * "Program development by stepwise refinement", Niklaus Wirth, april 1971
- Gestructureerd programmeren :

* “Flow diagrams, Turing machines, and languages with only two formation rules”, C. Böhm, A. Jacopini, mei 1966 (theoretische basis)
* “GOTO statement considered harmful”, Dijkstra, maart 1968
* “Structured programming”, Dahl, E.W. Dijkstra, C.A.R. Hoare, 1972 (praktische uitwerking)

Broedstoof voor een catastrofe

Als je aan iemand vertelt dat er in een project een catastrofale vertraging van het tijdschema is opgetreden, dan zal die ander direct veronderstellen dat er een hele reeks van rampen is losgebarsten. Meestal wordt de rampspoed veroorzaakt door een mug en niet door een olifant, en het tijdschema is nauwelijks merkbaar, maar onverbiddelijk uit de hand gelopen. Het is eigenlijk gemakkelijker om tegen grote rampen in het geweer te komen : er wordt dan gereageerd door een grote mlankracht in te zetten, de zaken radicaal te reorganiseren of nieuw benaderingen uit te sleutelen. Het hele team staat als één man klaar om de rampspoed te keren.

Mijlpalen of molenstenen

Hoe houd je een groot project op een krap tijdschema in de hand ? De eerste maatregel is het maken van een tijdschema. Binnen een lange reeks van gebeurtenissen wordt voor ieder scharnierpunt, beter gezegd voor iedere mijlpaal een datum vastgesteld. Het kiezen van de juiste data is een schattingsprobleem. Dit probleem is in de hoogste mate van de ervaring afhankelijk.
Voor het vaststellen van de mijlpalen is maar één relevante benadering toepasbaar. Mijlpalen moeten concrete, specifieke en meetbare gebeurtenissen zijn die op het scherp van de snede worden gedefiniëerd.
Met het scherp definiëren van mijlpalen bewijst men in feite het team een dienst, en het team heeft het recht deze dienst van de manager te verwachten. De onscherpe mijlpaal is in feite een molensteen die het moreel van de troepen vermaalt omdat hij een onjuist beeld geeft van de tijd die verloren gaat - tot aan het punt waarop herstel niet meer mogelijk is. En een chronische vertraging van het tijdschema is dodelijk voor het moreel van de troep.

Onder het tapijt

Als de manager van het kleinste team merkt dat zijn mensen een achterstand oplopen, zal hij niet meteen naar de hoogste bass hollen om zijn beklag te doen. Misschien halen ze het zelf wel in of misschien kan hij zelf wel een oplossing verzinnen. Dan heeft het ook geen zin zijn superieur ermee lastig te vallen. Wat niet weet, niet deert. Dus neemt de manager stoffer en blik, en al het ongewenste vuil wordt steevast onder het kleed geveegd.
Maar iedere baas heeft recht op twee soorten informatie : hij moet op de hoogte worden gebracht van afwijkingen op het oorspronkelijke plan die direct ingrijpen vereisen en hij moet een plaatje hebben van de stand van zaken om op de hoogte te blijven.
De belangen van het laagste baasje en van de hoogste chef komen hier intrinsiek met elkaar in conflict. De manager van het kleinste team is bang dat als hij zijn probleem aan zijn baas voorlegt, de laatstgenoemde actie zal ondernemen. Door dit ingrijpen van bovenaf wordt de bevoegdheid van de manager uitgehold, wordt zijn gezag aangetast en worden zijn andere plannen in de war gestuurd. Zolang hij meent het karwei zelf te kunnen klaren, zal hij zijn baas onkundig houden.
De hoogste chef kan nu op twee manieren onder het tapijt kijken :

- De chef moet onderscheid kunnen maken tussen informatie die om ingrijpen vraagt, en informatie die de status verduidelijkt. Hij moet zichzelf genoeg in de hand hebben om

niet in te grijpen in problemen die zijn managers zelf kunnen oplossen, en nooit in te grijpen als hij bezig is uitdrukkelijk een status te beoordelen.
- Ondanks dit alles moet er een manier te vinden zijn die een waarheidsgetrouw overzicht van de situatie geeft, of dit nu van harte gaat of niet. Het tijdschema met zijn frequente en scherpgeplaatste mijlpalen verschaft het basismateriaal voor een dergelijk overzicht. Bij een groot project lijkt het wenselijk eens per week een deel en eens per maand het geheel te overzien.

Het andere gezicht

Geloofwaardigheid van een programma

Grondige testgevallen betreffen drie delen van het domein van de invoergegevens :
- Normale gevallen die de belangrijkste functies van het programma voor normaal voorkomende gegevens testen.
- Nauwelijks legitieme gevallen die het randgebied van het bereik van de invoergegevens testen, garanderend dat de grootst mogelijke waarden, de kleinst mogelijke waarden en allerlei soorten van valide uitzonderingen functioneren.
- Nauwelijks illegitieme gevallen die de grens van het bereik vanaf de andere kant onderzoeken, garanderend dat ongeldige invoer de juiste diagnostische boodschappen oproept.

De vloek van het stroomdiagram

Het stroomdiagram maakt de beslissingsstructuur van een programma zichtbaar en vormt als zodanig slechts een aspect van de structuur van dat programma. De structuur van de beslissingen wordt op tamelijk elegante wijze zichtbaar gemaakt als het stroomdiagram niet meer dan een pagina beslaat, maar de overzichtelijkheid valt weg als het diagram meerdere bladzijden telt.
Het stroomdiagram, dat op één bladzijde past, wordt bij een programma van flinke afmetingen in wezen een diagram van de programmastructuur en van de fasen of stappen hiervan. Als zodanig is het erg handig.
Het gedetailleerde stap-voor-stap-stroomdiagram is echter verouderd en nodeloos lastig en alleen geschikt voor beginnelingen in het algoritmisch denken. Toen de kleine hokjes en de daarin opgesloten liggende voorschriften door Herman Goldstine en John von Neumann werden geïntroduceerd, fungeerden deze als een taal van hoog niveau, waarbij de onbegrijpelijke statements in machinetaal tot zinvolle clusters werden gegroepeerd. In een systematische taal van hoog niveau heeft de clustering al plaatsvonden en bevat ieder hokje slechts één statement.

Zelfdocumenterende programma’s

De oplossing voor het documentatieprobleem (veranderingen aan het programma worden niet snel, accuraat en correct op papier overgebracht) ligt naar mijn smaak in het samenvoegen van de bestanden en in het inlijven van de documentatie in de broncode. Dit geeft ter zelfde tijd een enorme stimulans tot correct onderhoud en een waarborg dat de documentatie door de gebruiker van het programma zonder veel moeite kan worden opgezocht. Dit soort programma’s noemen we zelfdocumenterend.
Een mogelijke benadering :
- De eerste gedachte houdt in dat onderdelen van het programma die er vanwege de programmeertaal sowieso moeten zijn, zoveel mogelijk als dragers van documentatie

worden gebruikt. Labels, declaraties en symbolische benamingen worden dus in het geweer gebracht om de lezer zoveel mogelijk informatie te bezorgen.

- Een tweede suggestie houdt in dat ruimte en opmaak zo veel mogelijk worden gebruikt om de leesbaarheid te vergroten en onderschikking en geneste structuren zichtbaar te maken.
- Het derde punt betreft de invoeging van noodzakelijke talige documentatie in het programma, als alinea's commentaar.

SAMENVATTEND OVERZICHT

Aristocratie, democratie en systeemontwerp

- Getrouwheid aan het oorspronkelijk ontwerp
- Ontwerpgetrouwheid waarborgen
- Aristocratie en democratie :
  * Taakverdeling
    - Paragraaf : Architecten en aristocratie ? Ja en nee !
    - Wat doet de constructeur terwijl hij zit te wachten ?
      - Beter is de vijand van goed
      - ...
      - ...
  * Het operatieteam