Beheersbaarheid

We kunnen in onze omgeving leven en er beheerst mee omgaan omdat er in deze omgeving duidelijk herkenbare wetmatigheden bestaan. Er zijn wetmatigheden die een zuiver logische of wiskundige oorsprong hebben en er zijn wetmatigheden die een meer natuurkundige oorsprong hebben. Eenvoudige wetten die we vaak tegenkomen leren we snel benutten en na korte tijd beseffen we niet eens meer hoe en waarom we dat doen. Bij sommige wetmatigheden heeft het de mensheid eeuwen gekost om hun bestaan te ontdekken en doorgronden. Dat hoeft helemaal niet te betekenen dat een dergelijke wet ingewikkeld is. Een voorbeeld is de wet van Newton "kracht = massa maal versnelling". Zonder kennis van deze wet is veel van de huidige technologie onmogelijk. Het is typisch een natuurwet. Hij kan niet ontdekt worden door puur logisch of wiskundig redeneren. Hij is ontdekt door verschijnselen in de natuur nauwkeurig te analyseren. Door de wet van Newton hebben we veel van onze omgeving leren begrijpen. Zelfs de niet zo grijpbare omgeving zoals de bewegingen van planeten en sterren is door de ontdekking van de wet van Newton een stuk inzichtelijker geworden.

Een duidelijk minder bekende wet die even eenvoudig is als de wet van Newton en die een minstens zo grote invloed op ons leven heeft is de relatiewet die het aantal potentiële relaties aangeeft dat tussen een bepaald aantal dingen bestaat. Deze wet kan juist wel met behulp van puur logische en wiskundige redenering bepaald worden. Hij heeft grote invloed op de beheersbaarheid van onze activiteiten. Het is dan ook verwonderlijk dat hij zo weinig bekendheid geniet. Het gevolg van deze onbekendheid is wel dat van deze wet weinig bewust gebruik gemaakt wordt. De relatiewet zegt dat er tussen n dingen precies n maal n-1 potentiële relaties bestaan. Het contact tussen dingen en hun onderlinge beïnvloeding loopt via de relevante onderlinge relaties. Om te begrijpen wat er gebeurt, moet je dus de relevante onderlinge relaties kennen. Dat is vooral van belang als deze dingen bestuurd moeten worden. Niet alle potentiële relaties zijn relevant. Om te weten of een relatie relevant is, is extra kennis over deze relaties nodig. Dit vergt tijd, ervaring en inspanning. Ervaren bestuurders en ontwerpers kennen de relaties en hebben geleerd om ze in groepen te rubriceren. Dat kunnen groepen zijn waarin de relaties allen hetzelfde karakter hebben maar het kunnen ook groepen zijn van relaties die regelmatig gezamenlijk optreden. Als er geen tegenmaatregelen genomen worden dan vermindert de beheersbaarheid volgens de relatiewet nagenoeg kwadratisch met het aantal dingen dat bestuurd wordt. Tien maal meer dingen betekent 90 maal hogere complexiteit. Honderd maal meer dingen betekent 9900 maal hogere complexiteit. Duizend dingen hebben 999000 potentiële relaties. Als hij het niet slim aanpakt dan verbruikt de ontwerper al snel zijn beschikbare capaciteit en wordt de zaak onbeheersbaar. De kosten zullen om die reden nog sterker dan kwadratisch met het aantal potentieel samenhangende dingen klimmen.

Er bestaat een krachtige remedie tegen de negatieve gevolgen van de relatiewet. De natuur en in het bijzonder de levende natuur maakt intensief gebruik van deze remedie. Complexe levende organismen bestaan uit een aantal grotendeels onafhankelijke onderdelen, organen die zelf ook weer uit onderdelen opgebouwd zijn. Zelfs de cellen bevatten nog duidelijk aanwijsbare onderdelen. De organen en hun onderdelen zijn via

specifieke verbindingskanalen met elkaar verbonden. Deze verbindingen laten alleen de noodzakelijke communicatie door. Het gevolg is, dat het aantal relevante relaties drastisch vermindert. Ook de mens heeft deze remedie ontdekt. Grote organisaties worden gebruikelijk hiërarchisch opgezet terwijl de afdelingen zoveel mogelijk als zelfstandige eenheden opereren. Vrijwel alle complexe producten worden uit onderdelen opgebouwd. Deze onderdelen passen via uitgekiende verbindingen op elkaar. Auto's, radio's, computers, fietsen en vliegtuigen zouden zonder deze aanpak onbetaalbaar zijn en nog steeds een veel mindere bruikbaarheid bieden.

Een simpel rekenvoorbeeld leert wat de voordelen van modularisatie zijn. Een monoliet bestaande uit duizend dingen bevat bijna een miljoen potentiële relaties. Worden de duizend dingen over tien modules verdeeld, dan zal elk module ongeveer honderd dingen bevatten. De modules bevatten dus elk ongeveer tienduizend potentiële relaties. Dat zijn voornamelijk relaties die binnen de module bestaan, maar er zijn ook relaties tussen de modulen. Tussen de modules komen verbindingen, maar dat zijn er niet zo veel. De hoeveelheid potentiële relaties is drastisch lager dan bij de monoliet. Daar komt nog bij dat de ontwikkeling en bouw van de module uitbesteed kan worden. De ontwikkelaar en bouwer van de module ondervindt een duidelijk lagere complexiteit dan de bouwer van de monoliet. Bovendien ondervindt de systeembouwer die het systeem uit de onderdelen opbouwt een nog veel lagere complexiteit. Het grootste succes boekt de modulaire aanpak wanneer modules uit eenvoudigere modules opgebouwd kunnen worden. Naarmate het aantal deelnemende functie-elementen stijgt, neemt de positieve invloed van modularisatie toe. Door deze voordelen kan een bloeiende onderdelenmarkt ontstaan. Daardoor zullen de prijzen van de onderdelen dalen terwijl de kwaliteit tegelijkertijd zal toenemen. Bovendien stijgt de diversiteit en de beschikbaarheid van onderdelen. Dat is precies wat er – in de hardwarewereld- gebeurd is.

Toch is niet op elk gebied verstandig met de relatiewet omgegaan. Fusies tussen bedrijven gaan lijnrecht in tegen de tendens tot modularisatie die door de relatiewet aangeraden wordt. Het is dan ook niet verwonderlijk dat fusies nog al eens fout lopen of meer nadelen dan voordelen brengen. Centraal bestuurde organisaties gaan door zelfoverschatting van de bestuurders ten onder zodra de complexiteit van de organisatie een kritische grens overschrijdt.

Er is een tak van technologie waaraan de zegen van modularisatie nagenoeg volledig voorbijgegaan is. De software-industrie heeft zich door zijn historische ontwikkeling van de modulaire aanpak afgekeerd. Software werd in de beginfase direct in machinecode geschreven. Kort daarna ontstonden de assembleertalen die het schrijven van programma's vergemakkelijkten door opdeling in routines toe te laten. Bij de derde generatie programmeertalen ging dit een stap verder door de aanroep van bepaalde, vaak gebruikte routines als onderdeel van de taal op te nemen. Kort daarna werden de programma's zo uitgebreid dat er behoefte ontstond om meer inzicht te krijgen in de structuur van de programma's. In de zeventiger jaren van de vorige eeuw ontstonden daarbij twee stromingen. Een van de stromingen koos voor een modulaire opbouw van programma's, waarbij de inhoud van de modules duidelijk en effectief afgeschermd werd van de omgeving. In de andere stroming werden de gegevens gescheiden van de

processen die er op werkten. Op deze wijze konden dus geen modules ontstaan. De tweede trend won snel de meeste aanhang. Er ontstonden grote bibliotheken van routines die de gekozen werkwijze ondersteunden. De aparte gegevensopslag gaf aanleiding tot het ontstaan van databases. De verbanden tussen gegevens werden ook in de databases opgeslagen. Ook hier had men de keuze tussen twee mogelijkheden. In de hiërarchische databases werden de gegevens bij elkaar gehouden in een hiërarchisch netwerk. In de relationele databases werden de gegevens in tabellen geplaatst, terwijl de relaties via sleuteltabellen vastgelegd werden. De relationele databases overheersten uiteindelijk de moeilijker te beheren hiërarchische databases. Het gevolg was dat er een steeds grotere afstand ontstond tussen de gegevens en de routines die er op werken. Na enkele jaren bleek het uit elkaar rafelen van dingen die essentieel bij elkaar horen toch averechts te werken.

Door de keuzes die de software-industrie gemaakt heeft, is de modulaire aanpak ernstig bemoeilijkt. In een module zitten de gegevens en de dingen die er mee werken binnen dezelfde omhulling. Deze omhulling schermt de interne gegevens af van de buitenwereld van de module. Daardoor kan een module zijn integriteit zelf behouden en een gedefinieerd gedrag garanderen. Er is in de softwarewereld nog wel een poging ondernomen om software meer modulair te maken, maar dat is zeer halfslachtig gebeurd. De huidige trend van programmeren heet objectgeoriënteerd programmeren. Deze methodiek maakt gebruik van tamelijk slecht ingekapselde objecten die zowel de gegevens als de daarop werkende routines bevatten. Er is geen stimulans om de verbindingen tussen de objecten zoveel mogelijk gelijkvormig te maken zodat zoveel mogelijk objecten op elkaar zullen passen. Er is ook geen stimulans om objecten in een open markt aan te bieden zodat daaruit gemakkelijk allerlei complexe systemen opgebouwd kunnen worden. In plaats daarvan richt de objectgeoriënteerde technologie zich op het vererven van eigenschappen van eenvoudige objecten, zodat daaruit complexere objecten ontstaan. Op deze wijze ontstaan in zichzelf gekeerde hiërarchische klassenbibliotheken die niet of nauwelijks compatibel zijn met andere klassenbibliotheken.

Door deze ontwikkelingen heeft de software-industrie de boot gemist. In de hardware-industrie heeft modularisatie tot grote successen geleid. Zo zelfs dat het aantal functie-elementen in complexe IC’s al jarenlang elke achttien maanden verdubbelt terwijl de prijs nagenoeg gelijk blijft. De structuur van complexe softwaresystemen lijkt meer op een stapel lappendekens, waarin de lappen met lange draden aan elkaar hangen. Deze structuur is dusdanig onoverzichtelijk dat geen mens de werking ervan volledig kan beschrijven. Het gevolg is dat het evenmin mogelijk is om het systeem betrouwbaar en volledig te testen. De correcte werking van dergelijke systemen kan dan ook niet gegarandeerd worden. De ingewikkelde structuur veroorzaakt dat complexe softwareproducten erg fragiel zijn. Hun gebruikers moeten leren leven met hun gebreken. Een duidelijk teken is het feit dat het ontwikkelen van softwareproducten die meer functie-elementen bevatten vaak exponentieel meer ontwikkelingstijd vergen. Het is geen wonder dat de kosten van complexe software de pan uitrijzen.

Er bestaat geen theoretische reden waarom software niet op dezelfde modulaire wijze ontwikkeld en geproduceerd kan worden als hardware. Dat zou echter betekenen dat het softwaremaakproces volledig aangepast moet worden. In dat geval zou er bovendien een goed lopende open markt voor softwaremodules moeten komen. De nieuwe technologie moet hergebruik van de interfaces voor modules stimuleren zodat zoveel mogelijk modules op elkaar passen. Bovendien moeten de technologie en het marktmechanisme de ontwerpers van modules beschermen zodat de in de module geïnvesteerde kennis te gelde gemaakt kan worden. Modulaire systeembouw wordt pas echt effectief wanneer modules uit modules gebouwd kunnen worden. Die mogelijkheid moet van begin af aan in de totale aanpak ingebouwd worden.

Het is gemakkelijk om een schets te geven van de situatie waarin software modulair ontwikkeld en gebouwd wordt. De complexiteit van het samenstellen van systemen uit modules is relatief laag. Dat betekent dat dit deel van het maakproces grotendeels geautomatiseerd kan worden. Dat houdt ook in dat het niet langer nodig is om een geniale systeemarchitect het maakproces van grote complexe softwareproducten te laten leiden. Een creatieve systeemontwerper kan met de nieuwe aanpak in een fractie van de tijd hetzelfde bereiken. De ontwikkelhulpmiddelen kunnen uit eenvoudige machinaal leesbare specificaties geheel automatisch skeletten van modules genereren. De skeletten kunnen door gekochte producten vervangen worden of kunnen stap voor stap in volwaardige modules omgezet worden en tussentijds als onderdeel van het doelsysteem getest worden. Het is realistisch denkbaar om het hulpmiddel waarmee het systeem samengesteld wordt een uit speciale automatisch gegenereerde modules samengesteld operating system toe te laten voegen dat precies past bij de eisen van de door de systeemontwerper gekozen toepassingsmodules. Zoiets is vooral zinvol bij software die in hightech producten ingebed is. De totaaloplossing omvat ontwikkelingshulpmiddelen, publicatiemedia en een distributie-instituut. Er zijn geen technologische belemmeringen om een dergelijke totaaloplossing neer te zetten. In feite zijn alle benodigde ingrediënten al beschikbaar. Daartoe behoren XML technologie, webservices, lokale en web based repositories en versleutelinghulpmiddelen. De grootste belemmering is van organisatorische aard.

Diversiteit is zowel een must als een gevaar. Ondersteuning van meerdere platforms en van diep doorlopende versies kan gemakkelijk tot een onbeheersbare diversiteit leiden. Hetzelfde gebeurt als er verschillende onderling concurrerende ontwikkelmethodieken ontstaan. Het onderscheiden en identificeren van nagenoeg gelijkwaardige modules voorkomt marktvervuiling. Deze problematiek is in de hardware-industrie niet anders.

De huidige deelnemers aan de software-industrie verdienen veel aan het feit dat de gebruikers van de eindproducten de huidige situatie accepteren omdat ze ermee hebben leren leven. Deze softwaregiganten zijn daarom niet geïnteresseerd in een drastische verbetering van het softwaremaakproces. Het zijn niet de makers van de software maar de gebruikers van complexe software die onder de slechte performance van de software-industrie te lijden hebben.

Overheden trachten met grote subsidies de ontwikkeling van softwaremodules op gang te brengen. Wat daarbij steeds vergeten wordt is de noodzaak om een bloeiende markt van softwaremodules op gang te brengen. Modules zijn relatief duur en hun toepassing is alleen zinvol wanneer hun diversiteit, beschikbaarheid, kwaliteit en door onderlinge concurrentie afgedwongen lage prijs het gebruik ervan zinvol maken. Deze voorwaarden worden door een open markt voor softwaremodules geboden. Het ontwikkelingsproces, het verhandelen van modules en het samenvoegen van modules in systemen moeten daarbij goed op elkaar passen.

Verbetering is alleen te verwachten wanneer een kleine groep pioniers er in slaagt om op kleine schaal te bewijzen dat modulaire software generatie en een daarop passend marktmechanisme voor softwaremodules technologisch en organisatorisch haalbaar is. De olievlekwerking die van een dergelijke ontwikkeling uitgaat, zal de hele softwaretechnologie omvormen. Niemand kan concurreren tegen een softwaremaakproces dat meer dan honderd maal sneller en evenredig goedkoper is. Het mooie is dat vrijwel iedere geïnteresseerde in dat proces mee kan doen en met de ontwikkeling van softwaremodules zijn brood kan verdienen.

De ontwikkelaars van hightech producten zijn vaak consumenten van complexe software. De kosten en de tijdsduur van de softwareontwikkeling beperkt in sterke mate het vernieuwingsproces in de hightech industrie. Softwareontwikkeling is niet hun hoofddoel. Zij zouden sterk gebaat zijn bij een modulaire software constructie. Het samenstellen van complexe systemen uit modules gaat ordegroottes sneller en gemakkelijker dan het ontwikkelen en bouwen van een gelaagde softwarelappendeken zoals dat nu met de huidige technologie gebeurt. De kosten zullen evenredig lager zijn. Het is onbegrijpelijk dat deze weg niet veel eerder ingeslagen is.