Methodology

Het ontwikkelen van software gaat niet zonder fouten. Wie denkt ooit foutloze software te ontwikkelen, komt er snel achter dat dit een utopie is. Software systemen van enige omvang laten zien dat het gedrag dusdanig complexe vormen aanneemt, dat die door mensen niet overzien kan worden. Om toch enige grip uit te oefenen op de correctheid, worden vaak testen en simulaties uitgevoerd om zoveel mogelijk fouten op te sporen.

Grip op ontwikkelen van correcte software

Formele methoden zorgen voor overzicht

Jan Friso Groote
(j.f.groote@tue.nl)

Frank Stappers
(f.p.m.stappers@tue.nl)

Michel Reniers
(m.a.reniers@tue.nl)

Na een reis van 286 dagen in 1998, start de Mars Climate Orbiter zijn motoren om in een baan rond Mars te komen. Maar zodra de motoren zijn gestart, raast het ruimtevaartuig door de atmosfeer van de planeet, waarna het neerstort. De oorzaak is een fout in het metrieke stelsel van de software, waardoor een project van $125.000.000 ten einde komt. In 2000, vinden acht mensen de dood en raken 20 personen in een kritieke toestand, als gevolg van blootstelling aan een te hoge straling tijdens radiotherapie. De oorzaak hier was een dubbele dosis straling, als gevolg van de volgorde waarop data was ingevoerd, terwijl dit door software voorkomen had moeten worden. Zondagavond, 31 mei 2009, stort een Airbus 330-200 neer in de Atlantische Oceaan vlak bij de Braziliaans kust tijdens zwaar weer. Volgens deskundigen zou een onjuiste indicatie van de snelheidsmeters, in wisselwerking met de besturingssoftware, geleid hebben tot 228 doden.

Dit zijn slechts enkele voorbeelden, waarin door software miljoenen dollars, en zelfs mensenlevens verloren zijn gegaan. Een terechte vraag die we onszelf mogen stellen is: Is het mogelijk systemen foutvrij te ontwikkelen? Software systemen van enige omvang laten zien dat het gedrag dusdanig complexe vormen aanneemt, dat die door mensen niet overzien kan worden. Om toch enige grip uit te oefenen op de correctheid, worden vaak testen en simulaties uitgevoerd om zoveel mogelijk fouten op te sporen. Als we een analogie proberen te trekken met andere vakgebieden, bijvoorbeeld bouwkunde, dan zien we dat het testen van software overeenkomt met de inspectie van de bouw bij oplevering. Wat er in de bouw ook gebeurt, is dat een architect een model maakt en een constructeur dat model doorrekent op constructiefouten.

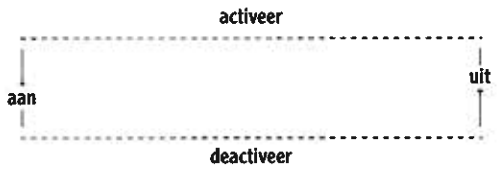
Bij de constructie van software is die eerste fase slecht ontwikkeld, met name het doorrekenen op constructiefouten. Daar betalen we gezamenlijk een hoge prijs voor, want het missen van fouten in de ontwerpfase is een kostbare aangelegenheid. Door herstel van laat ontdekte softwarefouten worden budgetten overschreden en is software onderhevig aan patches om "onvoorzien" gedrag te maskeren. Dit laatste tast de onderhoudbaarheid aan. We kennen al jaren methoden om software modellen te verkrijgen en te noteren. Bijvoorbeeld Yourdon's "structured design method", Hatley-Pirbhai modelleren en in toenemende mate UML. Het helpt om de software te structureren, maar helaas zijn deze methoden nog onvoldoende precies waardoor de interpretatiefouten blijven. Belangrijker nog is dat het niet mogelijk is om systematisch het ontwerp door te rekenen op correctheid. `Constructiefouten' worden op deze manier nog steeds niet uitgesloten.

Er zijn verschillende academische onderzoeksgroepen in de wereld die werken aan doorrekenbare specificatiemethoden voor software. Het vakgebied waarin zij werkzaam zijn wordt aangeduid als de `formele methoden'. Een aantal van deze onderzoeksgroepen hebben hun methoden omgezet in talen en tools die meestal vrij beschikbaar zijn. Tools waar nu nog actief aan gewerkt wordt zijn

FDR, mCRL2, Uppaal, Caesar/aldebaran, muSMV en SPIN. Langzamerhand vinden de methoden ook hun weg naar commerciële aanbieders. Recente producten van Nederlandse bodem, zijn ASD van Verum en Trustware ontwikkeld door Imtech.

De theorie

Formele methoden zijn gebaseerd op een eenvoudig onderliggende gedachte. Het gedrag van een systeem is te zien als een automaat met toestanden en overgangen. Extreem versimpeld is een computer een apparaat met twee toestanden: `AAN' en `UIT'. Er zijn twee acties `activeer' en `deactiveer'. Het toestandsdiagram ziet er uit als:

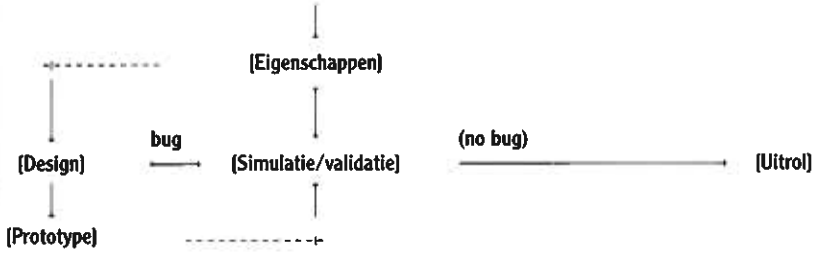


Een simpel apparaat met twee toestanden.

De werkelijkheid is veel complexer en bestaat uit heel veel van dergelijke gedragsautomaten die parallel staan, en elkaar beïnvloeden. Op een moderne computer kan de aan/uit schakelaar al beschouwd worden als een automaat die als die ingedrukt wordt nagaat of de computer feitelijk aan of uitgeschakeld is, en afhankelijk daarvan verschillende acties onderneemt. Een samenstel van dergelijke automaten is weer een gedragsautomaat. Het probleem is dat deze automaten extreem groot kunnen worden. Voor eenvoudige systemen is 10^1000 (10 tot de macht 1000) niet absurd groot.

Een moderne computer heeft veel meer dan 10^(10^10) toestanden. Dit zijn getallen die niet op te schrijven zijn. We spreken hier over informaticagetallen, om ze te contrasteren met de astronomische getallen die in verhouding verwaarloosbaar klein zijn. Het aantal atomen in het heelal is kleiner dan 10^100 (10 tot de macht 100). Lachwekkend klein gezien vanuit het perspectief van gedragsautomaten. Er zijn in essentie twee manieren om dergelijke automaten te analyseren op correctheid. De eerste is door het verifiëren dat het systeem aan zekere eigenschappen voldoet. Typische eigenschappen waar aan te denken valt zijn dat een systeem geen deadlock bevat, of dat een computer onder alle omstandigheden uit te zetten is. Als deze eisen complex worden, worden ze vaak geformuleerd in een zogenoemde modale logica. Populair zijn de logica's CTL, LTL en de modale mu-calculus. Het gaat hier te ver deze logica's in detail uit te leggen, maar belangrijk is te weten dat ze inmiddels zo krachtig zijn dat vrijwel iedere verifieerbare eigenschap kan worden uitgedrukt. Het verifiëren van een modale formule op de gedragsautomaat heet model checking. De tweede methode bestaat uit het versimpelen van de automaat door het toepassen van een gedragsreductiemethode. Een populaire methode is bijvoorbeeld branching bisimulatie reductie, maar er zijn nog vele anderen. Door eerst acties van een automaat te verbergen en die daarna te reduceren kan een automaat teruggebracht worden tot een zo'n klein aantal toestanden dat het resultaat met de hand kan worden geïnspecteerd. Op deze wijze wordt snel duidelijk of er ongewenst gedrag aanwezig is. Als eenmaal bewezen is dat de gespecificeerde gedragsautomaat alle gewenste eigenschappen heeft, kan die als uitgangspunt dienen voor implementatie van het systeem. Commerciële aanbieders hebben vrijwel altijd code generatoren waarmee automatisch code kan worden gegenereerd uit de formele specificatie. De formele specificatie kan ook worden gebruikt om automatisch testcases te genereren om te controleren of de implementatie wel voldoet aan de specificatie.

Formele methoden gaan uit van eenvoudige gedachtes.



De systeemontwikkeling, zoals deze er tegenwoordig meestal uitziet.

Het softwareontwikkelproces

Als we een schets maken van de huidige systeemontwikkeling, dan kan deze worden gekarakteriseerd als in bovenstaande afbeelding.:
Voordat men begint aan de ontwikkeling van een systeem, maken de verschillende stakeholders hun requirements bekend. Aan de hand van deze requirements worden er verschillende designs gemaakt. Deze designs bestaan vaak uit tekstuele beschrijvingen, systeem diagrammen en soms een eenvoudig programma wat het gedrag van het systeem simuleert. Om de kwaliteit van de documenten te waarborgen worden Peer-2-Peer reviews gehouden, wat vaak neerkomt op een oppervlakkige analyse van de structuur van het systeem. Om het gedrag beter te begrijpen worden deze designs vertaald naar eenvoudige opzichzelfstaande simulaties die het gedrag van het toekomstige systeem beogen te representeren. Deze vertaling komt vaak tot stand op basis van eerder opgedane ervaring en interpretatie van de ontwerper. Wanneer het design het systeem representeert, worden de simulaties uitgebreid met verschillende soorten software testen (unit tests, characterization tests, regressie tests, etc..). Parallel daaraan wordt vaak de ontwikkeling van hardware