STOMME FOUTEN DOOR TIJDSDRUK

MARIEKE HUISMAN GAAT MET VICI-BEURS PROGRAMMEERFOUTEN TE LIJF

Op verjaardagen heeft ICT-hoogleraar Marieke Huisman wat uit te leggen, als ze vertelt over haar werk. “Ik ontwerp technieken waarmee programmeurs foutloos software kunnen schrijven. Dat vinden veel mensen te abstract, dus meestal vertel ik gewoon dat ik studenten leer programmeren, dat snapt iedereen.” DOOR **Marc Laan** FOTOGRAFIE **Rikkert Harink**

Een beurs van anderhalf miljoen euro sleepte Marieke Huisman (1973) onlangs in de wacht. De hoogleraar bij het Twentse onderzoeksinstituut CTIT mag deze Vici-subsidie de komende vijf jaar gebruiken om een eigen onderzoeksgroep met vijf researchers op te bouwen.

Wat gaat u doen met anderhalf miljoen?
“Wij willen software maken waarmee je computerprogramma’s kunt controleren op betrouwbaarheid. Daar is nogal eens wat mee mis. Vooral software die meer taken tegelijk uitvoert zie je soms onverwachte stappen doen. Met het geld van de Vici-beurs gaan wij gereedschappen ontwikkelen, waarmee programmeurs straks met één druk op de knop kunnen verifiëren of er fouten in hun software zitten. Ons ideaal is dat de programmeurs hiermee zelf hun eigen code kunnen controleren. Dat kost ze weliswaar extra energie, maar het scheelt enorm veel tijd wanneer je dat al tijdens het ontwerpen van een programma doet. Later repareren van fouten is kostbaarder.”

Aan welk soort fouten moeten we dan denken?
“Er worden veel stomme programmeerfouten gemaakt. Een beruchte is bijvoorbeeld dat je een antwoord van een berekening bewaart in een geheugenlocatie die daarvoor helemaal niet is gereserveerd. Of andersom: je laat je programma een geheugenplaats uitlezen waar niks naar toe geschreven is.”
De gereedschappen die Huisman voor ogen heeft, zoeken onder meer in de softwarecode naar de commentaarregels waarmee programmeurs aangeven wat een brokje code wordt geacht te doen. “In een commentaarregel zegt de programmeur bijvoorbeeld: “Hier gaat het programma een berekening doen en de uitslag zet ik op een bepaalde plek in het geheugen van de computer klaar voor verder gebruik.”
“Een bekende fout is dat je een reeks data in een buffer opslaat, en dan verderop in je programma data probeert op te halen uit een heel andere geheugenbuffer. Met onze controlesoftware vis je dit soort simpele fouten er straks zo uit. Maar er zijn ook hackers die zulke fouten kunnen opsporen, om ze vervolgens te misbruiken voor een aanval op het computersysteem. Onze controlegereedschappen dienen dus ook de veiligheid van de software.”

Het klinkt allemaal nogal abstract.
“Op verjaardagen merk ik dat mijn vak moeilijk uit te leggen is. Dan vragen ze of ik hun vastgelopen Windows kan repareren. Meestal vertel ik nu dat ik studenten leer programmeren, dat snapt iedereen. Ik gebruik vaak het voorbeeld van een stoplicht. Software zorgt ervoor dat controlesoftware bekijkt of het verspringen van kleur logisch is geprogrammeerd. Er mogen geen twee straten tegelijk op groen staan. De simpele vraag die wij ons stellen is: doet een programma wat het moet doen? Bij een vliegtuig wil je dat heel graag weten. Bij een computerspelletje hoeft dat niet. Je moet dus prioriteiten stellen.”

Kan uw slimme controlesoftware ook al automatisch fouten verbeteren?
“We kunnen steeds meer, maar helemaal automatisch lukt het nog niet. Het kost veel rekentijd om te bewijzen dat de bedoeling van het programma inderdaad klopt. Het controleren van vijf tot tien regels code op logische fouten kost nu ongeveer anderhalf uur. Een gemiddeld programma telt soms wel honderdduizend regels code. Er is dus nog een hoop tijdwinst te behalen.”
“Onze software wijst aan: hier zit een fout, hier crasht het programma straks. Onze gereedschappen helpen programmeurs beter te programmeren. Software schrijven gebeurt vaak onder zeer grote tijdsdruk. Dus je weet op voorhand dat er bugs in zitten. Die willen wij in een vroeg stadium vinden. Ideaal gesproken moeten programmeurs al naar fouten zoeken zodra zij een klein blokje code af hebben.”
“Het unieke aan onze aanpak is, dat wij een abstract logisch model maken van het programma, en vervolgens verifiëren of de programmacode overeenkomt met de eisen van dit abstracte model. Deze logicatechniek stamt al uit de jaren zestig, toen Bob Floyd en Tony Hoare konden bewijzen dat een stuk software correct werkt. Zij deden dit met pen en papier. Wij gebruiken er ICT-technieken voor.”

> “De simpele vraag die wij ons stellen is: doet de software wat hij moet doen”

Voorkomen moderne programmeertalen die fouten niet?
“Nee, was dat maar waar. Een taal als C laat veel fouten toe. Java ook, al is dat iets beter gestructureerd. De taal Rust van Mozilla is wel sterk in het checken van inconsistente code, maar die optie kun je uitzetten als je snellere software wilt maken. Ik zie het overigens niet zozeer als mijn taak om op foutenjacht te gaan. Ik doe liever het omgekeerde: laten zien dat een brok code veilig is. Dat geeft de programmeur een goed gevoel: ik bewijs dat hun software werkt.”