Jaron Lanier is vooral bekend geworden vanwege zijn bijdrage aan *virtual reality*, een term die hij in de jaren tachtig geïntroduceerd heeft. Vermaard als componist, musicus en artiest, heeft hij aan een aantal universiteiten computerwetenschap gedoceerd. Onlangs nam hij deel aan het National Tele-Immersion Initiative, dat onder meer gewijd is aan het gebruik van computers om mensen op verschillende locaties de illusie te laten ervaren dat zij fysiek in dezelfde ruimte aanwezig zijn. Janice Heiss sprak met Lanier over zijn visie op de fundamentele veranderingen, die hij nodig acht voor software ontwikkeling.

'From scratch coderen'

Interview met virtual reality pionier Jaron Lanier

Java 3D en het project Looking Glass zijn voor een belangrijk deel gebaseerd op patenten op het gebied van virtuele realiteit die Sun Microsystems van Lanier heeft overgenomen. Momenteel werkt Lanier aan iets wat hij zelf omschrijft als *phenotropic computing*, waarin het huidige model van protocol-afhankelijke software wordt vervangen door patroonherkenning als manier om componenten van software systemen onderling te verbinden.

Wat is er eigenlijk mis met hoe we tegenwoordig software ontwikkelen?

Lanier: 'Ik vind dat de manier waarop we software schrijven en hoe we denken over software niet klopt. Kijk maar naar hoe software nu werkt: niemand, echt niemand kan software schrijven die op een betrouwbare manier met grote problemen omgaat. Als we niet anders gaan denken over het maken van software, zullen we nooit programma's kunnen schrijven die groter zijn dan pakweg 20 tot 30 miljoen regels code. Het huidige gebrek aan schaalbaarheid is een universele last. Er zijn monopolies in de industrie ontstaan omdat het zo ontzettend moeilijk is om zelfs maar op de markt te komen. Het is zo moeilijk om grote software applicaties te schrijven. Dat vind ik heel vreemd. Als je naar andere zaken kijkt die door mensen gebouwd worden, neem bijvoorbeeld olieraffinaderijen of passagiersvliegtuigen, dan zie je dat men daar veel effectiever met complexiteit kan omgaan dan zoals we dat met software doen. Het probleem met software is dat we nooit eerder geleerd hebben hoe we de neveneffecten van de keuzes die we maken (doorgaans noemen we die bugs) kunnen beheersen. We zouden daar niet zo berustend in moeten zijn. Ik geloof nog steeds dat er veel nieuwe ideeën zijn die uitgevoerd zullen worden, en dat we ooit nieuwe manieren zullen vinden om software te schrijven, waarmee we deze problemen zullen overwinnen. Dat is mijn voornaamste beroepsmatige interesse. Ik wil graag een bijdrage leveren aan het definitief afrekenen met bugs.'

Zijn bugs niet gewoon beperkingen van de menselijke geest?

Lanier: 'Nee, absoluut niet. Er is een verschil tussen een bug en een variant of imperfectie. Ga maar na: als je een kleine verandering in het programma aanbrengt, kan dit al resulteren in een enorme verandering in wat het programma doet. Als de natuur op dezelfde manier zou functioneren, zou het complete universum aan de lopende band crashen. Er zou geen enkele evolutie zijn en geen leven. Er is iets met de wijze waarop complexiteit in de natuur is opgebouwd, waardoor een kleine verandering ook aanvaardbaar kleine consequenties heeft. Incrementele evolutie is mogelijk. Je ziet een klein beetje - geen volledige, maar een beetje - lineariteit in de verbinding tussen genotype en phenotype, als je in die termen wilt spreken. In software is er echter een onduidelijk verband tussen de source code (het "genotype") en de waargenomen effecten van programma's -- wat je het "phenotype" van een programma zou kunnen noemen. Die onduidelijkheid, die chaos breekt ons op. Ik weet niet of ik ooit met een goed idee zal komen om dat op te lossen. Ik werk op dit moment aan een aantal dingen, maar wat me eigenlijk nog het meeste verontrust is het gebrek aan overtuiging onder

programmeurs dat we überhaupt iets aan dit probleem kunnen doen. De afgelopen decennia zijn we bijna in een soort zelfgenoegzaamheid verzand geraakt. Met iedere nieuwe generatie programmeurs zie je een toenemende berusting, dat het nu eenmaal zo gaat en ook altijd wel zo zal blijven. Misschien is dat waar. Misschien kunnen het we niet vermijden, maar daarmee is het nog geen gegeven. Voor mij hangt die inschikkelijke houding ten aanzien van bugs als een donkere wolk boven alle programmeerwerk.'

Reconstrueren

Misschien moeten we teruggaan en gewoon helemaal opnieuw beginnen?

Lanier: 'Daar heb ik de laatste tijd vaak aan gedacht. Als je de geschiedenis van het programmeren in kaart brengt, kun je momenten zien waarop het fout ging, vanwege de beperkte ervaringen en metaforen waarover men toen de beschikking had. Het is mogelijk je voor te stellen hoe die geschiedenis anders had kunnen verlopen. Laten we teruggaan naar het midden van de twintigste eeuw, naar de briljante, eerste generatie serieuze hackers, waaronder mensen als Alan Turing, John von Neumann, en Claude Shannon. Hun codeerervaring betrof het coderen van informatie die via een draad werd verstuurd. Zij waren bekend met gecodeerde berichten via de telegraaf en de telefoon. Alles werd geformuleerd in termen van een boodschap die werd verstuurd van punt A naar punt B, met enige voorkennis over punt B, wat betreft het soort boodschap. Of, als dat niet het geval was, met tenminste een poging van punt B om die kennis te reconstrueren, in het geval van hacken.

Uiteraard was Turing de eerste hacker die een computer gebruikte om een geheime code te kraken. Er is de notie dat je een soort tijdsschema hebt voor het coderen van informatie. Je hebt dus een soort transportlaag die een tijdspatroon bevat, en daarnaast heb je variaties daar bovenop die het eigenlijke signaal dragen. Dat is het gedeelte dat varieert. Computerwetenschap en elementaire informatietheorie zijn beiden op die notie gebaseerd. Computer architecturen zoals we die kennen, werden aanvankelijk ontworpen rondom gesimuleerde draadverbindingen. Source code is een simulatie van signalen die opeenvolgend via een draad verstuurd worden, en dat geldt ook voor verstuurde variabelen of berichten.'

Verschrikkelijk fout Lanier: 'Nu denk ik dat deze notie heel ver kan worden opgerekt. Als je zou willen, zou je het zelfs kunnen gebruiken om de gehele natuur te beschrijven, maar het is niet het enige idee dat zo ver zou kunnen worden doorgetrokken. En het zou ook lastig worden, want het is uiteindelijk ook niet de manier waarop natuurlijke systemen werken. Bijvoorbeeld, als je de connectie tussen een steen en de grond waarop die steen rust, zou willen beschrijven met de analogie van informatie die via een draad verstuurd wordt - dat kan wel, maar het is niet de beste manier. Het is geen elegante manier om dat te doen. Als je de natuur in zijn geheel bekijkt, is er waarschijnlijk een betere manier om te beschrijven hoe zaken onderling met elkaar in verbinding staan. Er bestaat tussen elke twee objecten een oppervlak dat een patroon laat zien. Het is op elk gegeven moment mogelijk om deze patronen te herkennen.

Zo zou je ook in de informatiewetenschap kunnen denken in termen van hoe verschillende delen van het universum met elkaar in verbinding staan door elkaars patronen te herkennen, in plaats van zich aan een protocol te houden. Het verschil tussen twee verschillende onderdelen die zich aan een protocol houden en informatie via een draad versturen enerzijds, of twee onderdelen waartussen zich een ruimte bevindt, die proberen elkaars patronen te herkennen anderzijds, is er een van gradatie.

Waar beide methoden de nadruk op leggen, verschilt met name in het soort informatie over het andere onderdeel dat is opgeslagen. In het bijzonder heeft dat betrekking op tijdelijk coderen, en of je een soort van stack van het verleden nodig hebt, om te decoderen wat er zoal binnen komt.

Als je kijkt naar de manier waarop we software schrijven, heeft de metafoor van de telegraaf-verbinding die signalen als Morse code verstuurt, een enorme invloed gehad op alles wat we doen. Zo is een variabele die aan een functie wordt doorgegeven een simulatie van een draadverbinding, en dat geldt ook voor een boodschap die aan een object wordt doorgegeven. Daar is niets mis mee, maar ik kan een empirische observatie maken: als je een tijdgebaseerd protocol gebruikt om codes via een draad te verzenden, is het inefficiënt om dat soort van codering fouttolerantie mee te geven. Het is veel efficiënter om het fout-intolerant te maken. Fouten worden daardoor catastrofaal. Je creëert een situatie waar een functie - wanneer bij het doorgeven van een variabele aan een functie ook maar één bit verkeerd gaat -, iets gaat uitvoeren dat niet alleen maar fout is, maar chaotisch fout, willekeurig fout, verschrikkelijk fout.'

Dat lijkt wel op een kaartenhuis dat in elkaar stort.

Lanier: 'Precies. En het resulteert in het soort fouten waar je eigenlijk niets van kunt leren. Je krijgt chaotische fouten waarvan je alleen maar kunt zeggen: "Tjonge, wat een rotzooi, ik denk dat ik het maar het beste opnieuw kan doen." Je hebt geen fouten die in verhouding staan tot de bron van de fout. En dat betekent dat je nooit enig besef van een geleidelijke ontwikkeling krijgt. Dus het echte verschil tussen het huidige idee van software, gebaseerd op protocolafhankelijkheid, en het idee dat ik voorstel, patroonherkenning, heeft te maken met het

soort fouten dat we maken. We hebben een systeem nodig waarin fouten vaker proportionele consequenties hebben in relatie tot de bron van de fout.'

Pattern Recognition Software

Hoe zou je dat kunnen doen met patroonherkenningssoftware?

Lanier: 'In de laatste vijf jaar, en in het bijzonder het afgelopen jaar, zijn significante verbeteringen gerealiseerd in sommige specifieke patroonherkenningsproblemen. Ik heb geruime tijd samengewerkt met neuroloog Christophe von der Marsburg van de University of Southern California en één van zijn studenten, Hartmut Nevern. We onderzochten het volgen van gezichtskenmerken, dat onderscheiden moet worden van gezichtsherkenning. We hebben dynamisch kenmerken in een menselijk gezicht gevolgd om *avatars* te maken die automatisch gezichten van mensen kunnen volgen. Het was heel boeiend om te zien hoe goed dat begon te werken. Er zijn eigenlijk twee factoren die daaraan hebben bijgedragen. De eerste is dat processoren veel sneller zijn geworden. We staan op het punt een drempel over te gaan waarachter veel herkenningstaken mogelijk zullen worden.

Een andere factor is, dat een aantal van de wiskundige technieken die we tot onze beschikking hebben, sterk verbeterd zijn. Een voorbeeld daarvan is *wavelets*, die een signaal kunnen opsplitsen in frequentie- en tijdgebaseerde componenten. Het is een wat flitsender versie van een meer bekende techniek, die de meeste programmeurs wel zullen kennen: de Fourier Transformation of FFT (Fast Fourier Transformation). Dat is een bekend voorbeeld van een nieuwe patroonherkenningstechniek, maar er zijn ook andere. Er is dus veel toegepaste wiskunde beschikbaar die steeds beter en beter wordt. Er zijn ook enkele empirische resultaten uit de neurowetenschap die ons nuttige ideeën geven, en dat is in het bijzonder een verheugende ontwikkeling. Sinds kort zien we betere gezichtsherkenners, betere face trackers, betere handschriftherkenners, betere gate trackers - al dat soort dingen. Dus patroonherkenning wordt zo langzamerhand echt volwassen. Helaas wordt die ontwikkeling vooral aangedreven door vereisten op het terrein van beveiliging en defensie, maar om de een of andere reden wordt het levensvatbaar. We komen nu bij het punt waarop computers overeenkomsten gaan herkennen in plaats van perfecte identiteiten, en dat is in de kern waar patroonherkenning over gaat. Als we de beweging kunnen maken van perfectie naar overeenkomst, kunnen we ook opnieuw gaan kijken naar de manier waarop we software bouwen. Dus in plaats van de vereiste van protocol-afhankelijkheid waarin iedere component tot op bitniveau perfect moet matchen met andere componenten, kunnen we beginnen met overeenkomstigheid. Dan zal een vorm van zeer gracieuze fouttolerantie mogelijk worden, met een voorspelbare overhead. De weddenschap die ik als computerwetenschapper wil aangaan is dat dit het geheime ontbrekende ingrediënt zal zijn dat we nodig hebben om een nieuw type software te kunnen maken.

"Phenotropisch" is het steekwoord waarmee ik dit nieuwe type software zou willen aanduiden. "Pheno" refereert aan "phenotype," de uiterlijke verschijning van iets. "Tropisch" betekent interactie. Het basisidee beschrijf ik in een boek, getiteld: "The Next 50 Years: Science in the First Half of the Twenty-First Century". In phenotropische computing zouden software-componenten zich op een gracieuze fout-tolerante wijze met elkaar verbinden die statistisch en soft en fuzzy is en gebaseerd op patroonherkenning zoals ik die beschreven heb.'

Advies aan ontwikkelaars

Wat zou je tegen ontwikkelaars willen zeggen?

Lanier: 'Het belangrijkste is om een optimistische mindset te behouden. In de computerwereld kom je snel in een soort hypnose terecht: omdat gedetailleerde overwegingen vereist zijn om wat dan ook met computers te kunnen doen, gaat het je volledig in beslag nemen. Computerwetenschappers zeggen wel eens dat je verliefd wordt op datgene waarmee je aan het worstelen bent. Wanneer je werkt aan een of ander groot programmeerproject, dan neemt het in je hoofd krijgen van het geheel - zodat je de juiste beslissingen neemt - je volledig in beslag, en dat geldt voor alle betrokkenen bij zo'n proces. Het is helemaal niet verkeerd voor een programmeur om door je werk in beslag genomen te zijn. Het is juist goed. Om effectief te kunnen zijn bij welk groot software project dan ook, moet je volledige toewijding opbrengen. Je moet heel veel in je hoofd opslaan. Ik droomde 's nachts vaak van code, wanneer ik middenin een of ander groot project zat.

'Ik vind dat de manier waarop we software schrijven en hoe we denken over software niet klopt'

En precies op dat punt bestaat het risico dat je het vertrouwen verliest dat bij voorgaande generaties nog bestond: dat computing op een fundamenteel niveau beter zou kunnen worden. En aangezien bijna iedereen in de gehele beroepsgroep via de academische wereld de industrie ingaat, wordt iedereen op deze manier opgeslokt. Ik ben bang dat we onze hogere ambities verloren hebben, en dat vind ik zeer verontrustend. Als je praat met mensen die een academische carrière nastreven en

willen promoveren, zou het gesprek wel eens kunnen komen op quantum computing, dat erg opwindend schijnt te zijn. Ik denk ook wel dat dat zo is, maar voor mij is veel belangrijker hoe we over software nadenken. Als we het probleem niet kunnen oplossen hoe we grote programma's schrijven, maakt het niet uit of die programma's quantum zijn of conventioneel. We zullen nog steeds tegen dezelfde complexiteitsgrens oplopen. Dit overweldigend belangrijke issue krijgt niet de aandacht die het verdient, terwijl het zo de kern raakt. Ik vermoed dat we allemaal zo emotioneel betrokken zijn bij de alledaagse, concrete problemen die we onder onze neus krijgen, dat we een lange termijn-vertrouwen ontberen dat software fundamenteel beter zou kunnen zijn. Dat lange termijn-vertrouwen stuwt andere vakgebieden nog steeds voort, neem de medische wetenschap. Ik wil ook de computerwetenschap dat vertrouwen weer teruggeven.'

Laaghangend fruit

Welk advies zou je willen geven aan ontwikkelaars die net begonnen zijn?

Lanier: 'Er zijn veel dingen die ik zou willen zeggen. Als je geïnteresseerd bent in user interfaces, zijn er vandaag de dag geweldige mogelijkheden om te pushen wat een user interface kan zijn. Als een user interface een gebruiker een zekere macht geeft, probeer dan eens te bedenken hoe je de gebruiker nog meer macht zou kunnen geven, terwijl je de user interface tegelijkertijd inspirerend en gebruikersvriendelijk houdt. Kun je dat? Zou je bijvoorbeeld een zoekmachine kunnen ontwerpen, die mensen zou stimuleren om complexere zoekopdrachten te geven dan ze nu kunnen doen via een service als Google, maar die nog steeds eenvoudig in het gebruik is? Ik heb nog geen echt goede visuele interface gezien, bijvoorbeeld voor het zoeken op Google. Zou je dat kunnen maken? Zou je massa's mensen kunnen verleiden om plotseling specifieker en effectiever te zoeken, puur door een betere user interface?

Er zijn honderden uitdagingen, en welke daarvan dan ook zou de levens van miljoenen mensen echt snel kunnen verbeteren. Veel terreinen zijn dus rijp. Ik noem ze "rijp," maar ik noem ze niet "laaghangend fruit", omdat ik denk dat deze problemen over het algemeen erg gecompliceerd zijn, en het is belangrijk niet te doen alsof het eenvoudig is. Ik denk dat iedereen die met een dergelijk probleem vooruitgang boekt heel veel krediet verdient.

Ik zou willen aanbevelen dat ontwikkelaars de geschiedenis van de computerwetenschap heel sceptisch tegemoet treden. Lees maar eens de vroege geschriften van Turing, Shannon en von Neumann, en probeer te bedenken waar deze mensen er naast zaten. Als ze vandaag zouden beginnen, waar zouden ze dan nu anders over denken?'

Enorme soep Lanier: 'Dat is ook het probleem met computers: het is zo ontzettend veel werk om over programma's na te denken, dat mensen details uit de software behandelen alsof ze door God zelf zijn uitgevonden. Wanneer je voor het eerst les krijgt in programmeren, wordt er over een computerbestand gepraat alsof het een natuurwet betreft. Maar als je naar de geschiedenis kijkt, zie je dat bestanden controversieel waren. De eerste versie van de Macintosh, voordat die op de markt kwam, had zelfs helemaal geen bestanden. In plaats daarvan bouwden ze voort op het idee van een enorme soep met kleine schattige primitieven zoals letters. Er mochten niet eens bestanden komen, want dan heb je ook geen last van niet-compatibele bestandsformaten, toch? Het is belangrijk om te kijken naar hoe bestanden de standaard werden. Plotseling bleek namelijk dat UNIX ze had, vervolgens hadden de IBM mainframes ze ook, toen had DOS ze, en vervolgens Windows. En toen kwam ook Macintosh met bestanden. Dankzij de UNIX erfenis, gingen we ook met Internet denken in termen van het verplaatsen van bestanden en bestandgeoriënteerde protocollen zoals FTP. Het bestand is een universeel idee geworden, ook al was het dat oorspronkelijk helemaal niet.

Nu leer je bij computerwetenschap, dat het bestand een soort natuurelement is, zoals een photon. Dat is een gevaarlijke mentaliteit. Zelfs wanneer je er niets aan kunt veranderen, en je kunt op dit moment praktisch geen software schrijven zonder bestanden, is het nog steeds belangrijk dat je je niet laat beetnemen. Je moet goed in de gaten houden wat een menselijke uitvinding is en wat niet. Je zou bestanden eigenlijk moeten zien als winkelwagentjes: gewoon een uitvinding met positieve en negatieve kenmerken. Het is heel belangrijk om die scepsis levend te houden. Als je dat doet, zal het echt invloed hebben op de kwaliteit van de code die je vandaag gaat schrijven.'

Referenties

Homepage van Jaron Lanier
www.jaronlanier.com/
Korte biografie van Jaron Lanier
people.advanced.org/~jaron/general.html
"One Half a Manifesto"
www.edge.org/3rd_culture/lanier/lanier_index.html

Dit artikel werd oorspronkelijk gepubliceerd op de website java.sun.com en is geschreven door Janice Heiss. Vertaling: Vincent Hoogendijk.