Les Hatton gelooft niet in C++ en Java, maar des te meer in Linux en C. Het gaat hem om de betrouwbaarheid. Ook pleit Hatton voor een meer wetenschappelijke aanpak van software-engineering. Het moet een echte ingenieursdiscipline worden en dat kan alleen door feiten te verzamelen en te leren van fouten uit het verleden. Volgens Hatton gebeurt dat veel te weinig.

HANS VAN THIEL

'C++ is te groot en veel te ingewikkeld'

PT Embedded Systems:
Uw voor ons zeer bekende boek 'Safer C' is verschenen in 1995. Is de betrouwbaarheid van software toegenomen de laatste vijf jaar?

Les Hatton:
'Nee, er is geen verbetering. Elke winst is verloren gegaan aan de toegenomen complexiteit van software. Het is al een hele worsteling om bij te blijven en de gevolgen van fouten worden steeds ernstiger. Programmatuur is chaotisch, in die zin dat kleine fouten onverwachte en grote consequenties kunnen hebben.'

PT Embedded Systems:
Is de betrouwbaarheid van software te verbeteren door methodieken te gebruiken zoals UML (unified modeling language, red.)?

Les Hatton:
'Ik weet het niet. Het is zeker goed om na te denken voor je begint met programmeren maar er zijn geen gegevens beschikbaar om een keuze in methode op te baseren. Bovendien is er meer dan UML. Zo is prototyping van software belangrijker voor de betrouwbaarheid.
De huidige software-engineering is geen ingenieurs-wetenschap. Ingenieurs maken een ontwerp met de gedachte dat het kan mislukken. Een onderdeel van een brug kan breken en de ontwerpers weten dat en houden er rekening mee. Het ontwerp is zo gemaakt dat als er op een bepaald punt iets misgaat de gevolgen beperkt blijven. Software-engineers zouden iets dergelijks moeten doen en hun systemen moeten ontwerpen op het opvangen en herstellen van fouten (*recovery*). In de Mars Mariner trad bijvoorbeeld een software-fout op. Toevallig zat er in het OS een C-interpreter waarmee op afstand een aantal globale variabelen kon worden aangepast. Deze fout kon dus worden hersteld maar dat was puur toeval. In het ontwerp was er geen rekening mee gehouden dat zoiets zou kunnen gebeuren.'

PT Embedded Systems:
In het boek 'Safer C' maakt u een duidelijk onderscheid tussen software als product en het proces van ontwikkeling.

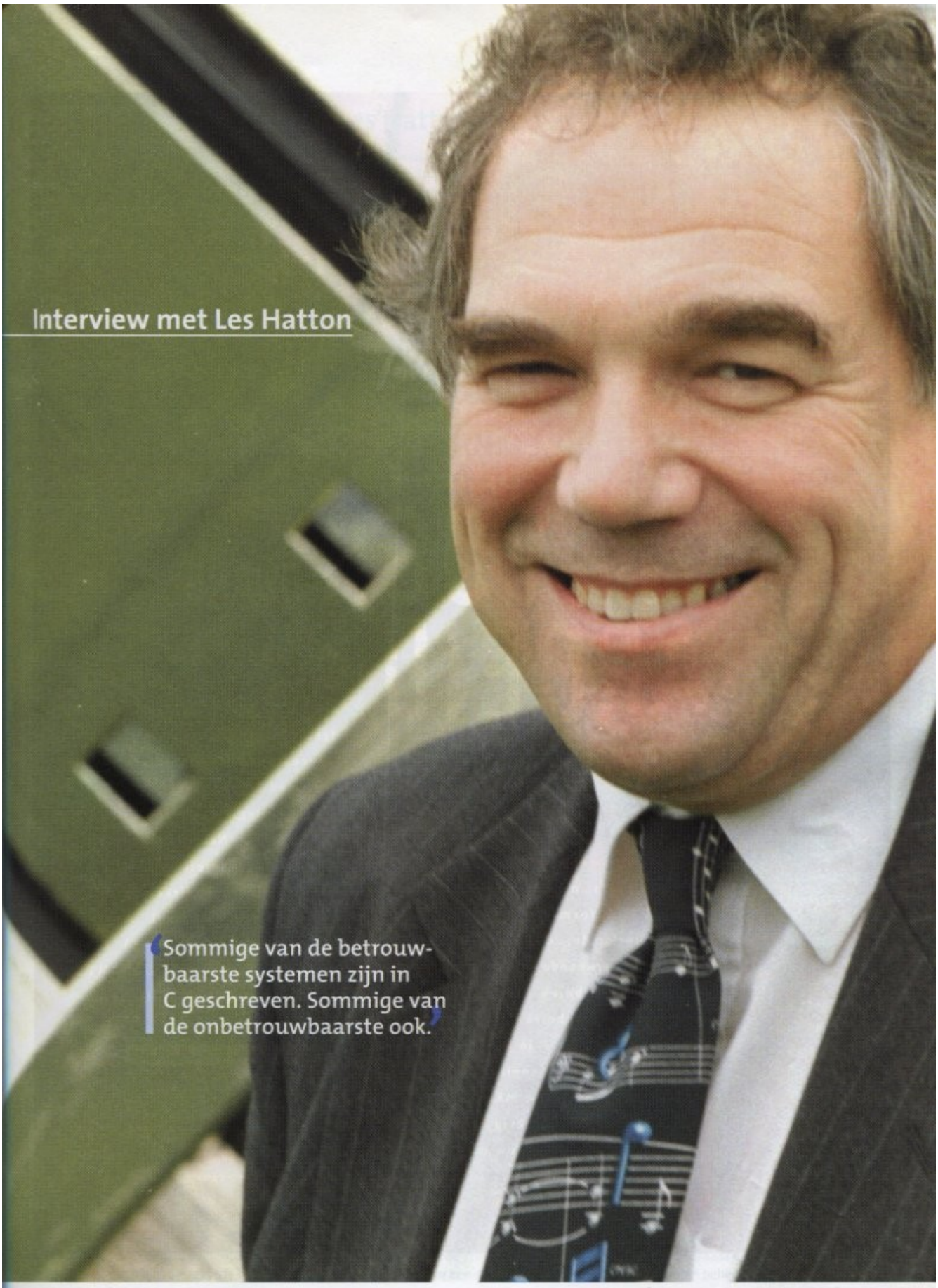
Les Hatton:
"Safer C' is uit het hart geschreven. Het is eenvoudigweg gebaseerd op twintig jaar waarneming. Je ziet keer op keer dezelfde fouten terugkomen en dat is ook al een aanwijzing dat software-ontwikkeling nog geen ingenieurs-discipline is. Zo'n 40% van alle fouten in softwaresystemen is te voorkomen maar er wordt nauwelijks geleerd van het verleden. Iedereen heeft een mening maar er zijn veel te weinig feiten bekend. De gegevens die er wel zijn worden vaak ook nog genegeerd. Goeroes bestaan dankzij de afwezigheid van meetresultaten.
Het is een probleem dat het gedrag van software al gauw een enorm aantal vrijheidsgraden heeft. Veel meer dan je bijvoorbeeld in de bouwkunde tegenkomt. Als je niet aan een product kunt meten is het ook heel moeilijk om het te verbeteren. Aan de andere kant hoeft het meten ook niet veel beter te zijn dan het proces. Zo konden de Egyptenaren de piramiden heel goed bouwen zonder over een precieze lengtemaat te beschikken. In elk geval moet je aan het product meten om het proces te kunnen verbeteren. Het heeft geen zin om aan het proces te meten om het proces te verbeteren.'

PT Embedded Systems:
In uw boek concludeert u dat C geschikt is voor kritieke programmatuur.

Les Hatton:
'C injecteert defecten bij de compilatie, maar andere talen zijn niet beter. Programmeertalen worden gestandaardiseerd in commissies. Daar komt veel politiek aan te pas en er worden achterdeurtjes ingebouwd zodat de programmeur kan verdwalen. Sommige van de betrouwbaarste systemen zijn geschreven in C. Sommige van de onbetrouwbaarste ook.'

PT Embedded Systems:
Een programmeerstandaard geeft aan welke programma-constructies moeten worden vermeden in kritieke systemen en welke absoluut moeten worden gebruikt. U noemt bijvoorbeeld *assignments* waarvan het resultaat afhankelijk kan zijn van de evaluatievolgorde als bron van fouten. Gebruik van functieprototypen voorkomt daarentegen *mismatches* in aantal en type argumenten. Van dergelijke aanwijzingen heeft u er een groot aantal in 'Safer C'. Bestaat er ook niet zo'n standaard voor het gebruik van C in de autoindustrie?'

Les Hatton:
'Ja, dat is het Misra-document uit 1998. Misra staat voor 'Multi industry software research association' en is een we- ›



Interview met Les Hatton

'Sommige van de betrouwbaarste systemen zijn in C geschreven. Sommige van de onbetrouwbaarste ook.'

Over Les Hatton

'Safer C', zoals de meeste ontwerpers het kennen, is ongetwijfeld het bekendste boek van Les Hatton. De volledige titel luidt: 'Safer C: developing software for high-integrity and safety-critical systems'. Het gaat over het minimaliseren van het aantal fouten in C-programmatuur. Maar het werk heeft een grotere reikwijdte, zoals de volledige titel ook aangeeft. Software-engineering moet een echte ingenieursdiscipline worden en dat kan alleen door feiten te verzamelen en door te leren van fouten uit het verleden. Volgens Hatton gebeurt dat in de software-industrie veel te weinig.

Les Hatton is Professor of Software Reliability aan de University of Kent at Canterbury en als consulent verbonden aan Oakwood Computing. In 1970 studeerde hij af in de wiskunde aan het King's College in Cambridge. Hij haalde hierin nog een MSc (1971) en een PhD (1973) aan de University of Manchester. Hij heeft echter ook een diploma in klassieke gitaar van de London College of Music (1980) en een juridische graad (LL.M in IT law) aan de University of Strathclyde (1999).

In de jaren zeventig en tachtig werkte Les Hatton als geofysicus. Hij was enige tijd gasthoogleraar (Professor of Geophysics) aan de TU Delft. Zijn brede ervaring is niet alleen terug te vinden in artikelen maar ook in zijn werk over programmeren, onder meer in de voorbeelden die hij als toelichting gebruikt.

Sinds 'Safer C', dat in 1995 uitkwam, heeft hij een dertigtal artikelen gepubliceerd, de meeste over software-betrouwbaarheid. Hij werkt ook aan enkele boeken, maar kan nog geen verschijningsdatum noemen. 'Software failure: avoiding the avoidable and living with the Rest', geschreven met Shari Pfleeger en Chuck Howell is echter gereed en zal binnenkort uitkomen. Het gaat over systemen, projectmanagement, configuratiebeheer en beschrijft praktijkgevallen bij onder meer Sun en Lockheed.

Les Hatton was 16 Oktober 2000 in Nederland op uitnodiging van PTS Software uit Bussum, info@pts.nl, www.pts.nl.

reldwijd samenwerkingsverband van een aantal autofabrieken. Niets dan lof voor deze standaard die overigens ook in het Japans is vertaald. Er zijn wel een paar problemen mee maar het is zo'n enorme verbetering. Ook voor medische toepassingen en in de lucht- en ruimtevaart is Misra aanvaard.

Wereldwijd is nu de autoindustrie de drijvende kracht achter software-ontwikkeling. Er is meer software, die is kritischer geworden, er zijn strakkere deadlines, de consument bepaalt de richting en de kosten van fouten zijn het hoogst. Vijf jaar geleden zat er twintigduizend regels programmacode in een auto. Nu is dat een miljoen.

Ik ben net terug uit Japan, waar men zeer kwaliteitsbewust is, en ik heb eerder lesgegeven in Detroit. Negentig procent van mijn werk in de afgelopen vijf jaar is in de autoindustrie geweest. Toch wordt de deadline voor de ontwikkeling van een airbag nog steeds bepaald door het tijdstip van introductie van het nieuwste automodel.'

PT Embedded Systems:
Zijn er ernstige software-fouten geweest in de autoindustrie?

Les Hatton:
'Jazeker. In 1999 heeft General Motors 3,5 miljoen auto's moeten terughalen door een software-fout in het ABS. Er waren in het jaar daarvoor 2111 botsingen met 293 gewonden. Hier heeft het Amerikaanse NTSB, de 'national transport safety board', moeten ingrijpen. In september dit jaar is Ford in Canada twee weken gesloten geweest om bij 15 duizend auto's een softwarefout in de airbags te herstellen. Die dingen gebeuren.'

PT Embedded Systems:
Embedded systems zijn dikwijls aan tijdsbeperkingen onderhevig. Zijn realtime failures belangrijk, in de praktijk?'

Les Hatton:
'Ik weet het niet. Bij embedded software is het vaak extra moeilijk om fouten vast te stellen. Om tijd en ruimte te winnen is de diagnostiek die bij het testen nog een rol speelde er meestal in de productie uitgehaald. Als er dan iets gebeurt is niet meer te achterhalen wat. Nog belangrijker is dat over fouten niet wordt gepraat. Advocaten verbieden zelfs het spreken over fouten omdat het bedrijf aansprakelijk kan worden gesteld.

Dit is ook zo'n factor die de ontplooiing van software-engineering tot een techniek tegenwerkt. Openheid over fouten en een analyse van hun oorzaken is het beste middel om dezelfde gebreken in de toekomst te voorkomen. Als er ergens ter wereld een hotel instort kan een onafhankelijk ingenieur meestal binnen een etmaal de oorzaak aangeven, maar met software lukt dat niet.'

PT Embedded Systems:
U hebt onlangs (SM/ASM 2000, red.) een presentatie gehouden onder de titel: 'Why is Linux so reliable ...'

Les Hatton:
'De betrouwbaarheid van Linux is verbazingwekkend. Het is twee ordes van grootte betrouwbaarder dan de competitie. De belangrijkste reden is dat de code aan een grootschalige inspectie is onderworpen. Een groot deel van software-fouten is statisch, dat wil zeggen te achterhalen voor de compilatie, en dat is bij Linux wereldwijd gebeurd.

> 'Ingenieurs maken een ontwerp met de gedachte dat het kan mislukken.'

Verder is het testen formidabel geweest, met continue feedback naar de ontwerpers toe. Er hebben een stuk of zes mensen aan de Linux-kern gewerkt. Daaromheen had je een vijftigtal die zich met andere functionaliteit bezig hielden, en daaromheen had je weer een wolk van programmeurs die af en toe iets bijdroegen. Ik heb zelf meegemaakt dat ik in een nieuwsgroep een vraag stelde over een multiprocessor Linux systeem en dat ik binnen een kwartier antwoord kreeg van een professor uit Palo Alto in Californië.'

PT Embedded Systems:
En embedded Linux?

Les Hatton:
'Dat is vrijwel hetzelfde. Ik denk dat we over vijf jaar in kritische systemen alleen nog Linux tegenkomen.'

PT Embedded Systems:
Er wordt veel verwacht van Real Time Java. Hoe denkt u daarover?

Les Hatton:
'Java is een hele goede taal, maar er zijn twee, nee eigenlijk drie problemen mee. Het ontwerp is moeilijk te optimaliseren, het probleem met het dynamische geheugen, de *garbage collection*, is niet opgelost en de *exceptions* worden door de JVM afgezet. Er zijn heel wat exotische sites met Java-fouten. Ik denk daarom niet dat RT Java (realtime Java, red.) een toekomst heeft. Het heeft niet veel zin om alles in een taal te stoppen. Dat is zeer ambitieus en de kans dat het lukt is kleiner dan dat het niet lukt. Maar dat is mijn persoonlijke mening.'

PT Embedded Systems:
Wat vindt u van C++?

Les Hatton:
'C++ is te groot en veel te ingewikkeld. In de embedded wereld telt alleen snelheid en ruimte; *speed and space*. Wel zul je - dat zie je al gebeuren - met talen van een hoger niveau C-code kunnen genereren. Maar embedded software zal worden beheerst door C.' ◻