



# Broncode toont risico's onderhoud

## Ondersteuning strategie met directe en indirecte feiten

### Een beslissing met kostbare gevolgen

Het gebeurt vaker. Het relatie-beheersysteem dat speciaal voor een organisatie is ontwikkeld blijkt toch niet zo makkelijk aan het internet te koppelen te zijn. De organisatie overweegt het te vervangen door een standaardpakket, maar beschikt niet over de informatie om de implicaties voor de overige systemen in te schatten.

Of neem de wijzigingen in een hypotheekstelsel. Waarom zijn deze toch altijd net weer kostbaarder dan oorspronkelijk begroot?

Moet hier iets aan gebeuren? En waarom verbruikt dit relatief eenvoudige systeem het meeste geheugen en cpu-cycles van alle systemen?

De overeenkomst in deze gevallen is dat de organisatie een beslissing moet nemen over een maatwerksysteem – een uniek systeem met zijn eigen verzameling nuttige features en vervelende problemen.

Deze beslissing kan kostbare gevolgen hebben.

Een beslissing over een maatwerksysteem kan kostbare gevolgen hebben. Wijzigingen zijn prijziger dan oorspronkelijk begroot, en de interne werking van een systeem is niet altijd bekend, zodat overgang naar standaard software voor onverwachte problemen kan zorgen. De auteurs beschrijven een software risk assessment om risico's systematisch in kaart te brengen.

Arie van Deursen en Tobias Kuipers

Een software risk assessment is een onafhankelijke beoordeling van de risico's bij het bouwen, installeren, testen, beheren en onderhouden van een software systeem. Een assessment wordt uitgevoerd om een specifieke reden op een specifiek systeem, en dient in een zeer beperkte doorlooptijd uitgevoerd te worden (doorgaans binnen drie weken). De afgelopen jaren voerde de Software Improvement Group diverse van dergelijke assessments uit, zowel in commerciële als in academische omgevingen. Hieruit ontstond een methode die zich onderscheidt door het betrekken van de broncode bij de risico-

analyses. Doorgaans gaat het hierbij om honderdduizenden zo niet miljoenen regels Cobol, Java, C enzovoort. De broncode is het directe object voor analyse. Informatie die we direct uit de broncode kunnen afleiden geeft het betrouwbaarste beeld over wat er daadwerkelijk aan de hand is met het onderliggende softwaresysteem. Dit is informatie over afhankelijkheden tussen componenten, de scheidingslijn tussen klantconfiguraties en pakketmodules of de organisatie van de gegevens in het datamodel. Daarnaast maken we van allerlei nuttige indirecte bronnen gebruik. Deze omvatten documentatie over

## Samenvatting

De auteurs beschrijven een *software risk assessment* voor risico's van softwareonderhoud. Automatische analyse van de broncode levert directe feiten op. Analyse van diverse documenten en workshops met de belanghebbenden leveren indirecte feiten op. Deze worden geïnterpreteerd en vertaald naar risico's van het oorspronkelijke probleem.

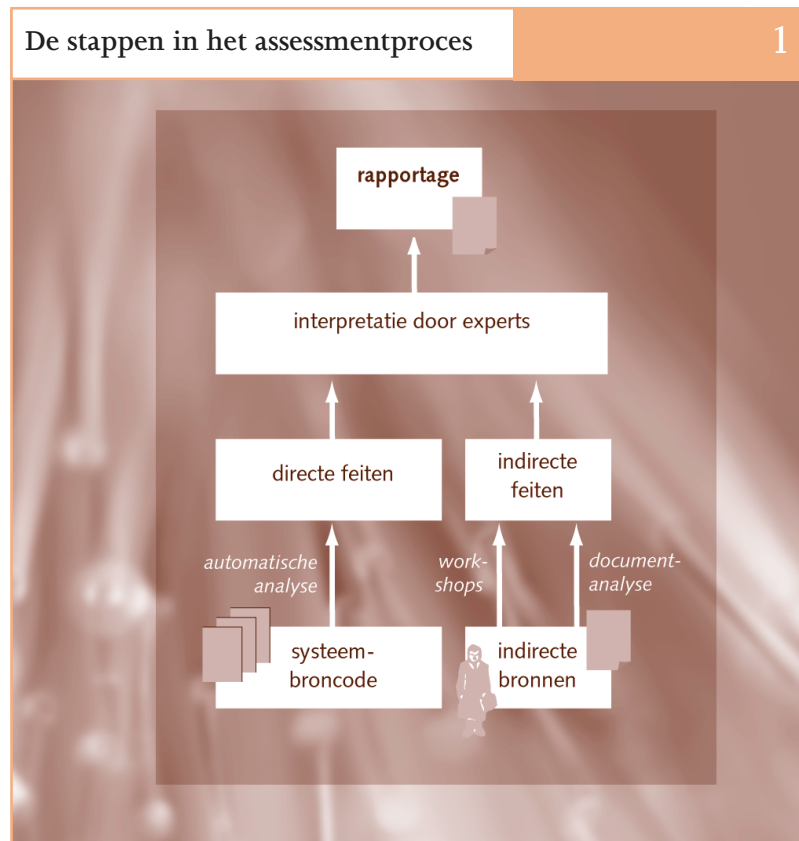
de architectuur, het gebruikte ontwikkelingsproces, en kennis bij betrokken medewerkers. De informatie uit deze indirecte bronnen heeft het hogere abstractieniveau als voordeel. Een groot probleem is echter de betrouwbaarheid: ontwerpdocumentatie is vaak verouderd en kennis van medewerkers is subjectief en vaak gericht op specifieke aspecten van het systeem. Een accuraat totaaloverzicht ontbreekt. Een software risk assessment bestaat dus niet alleen uit het naar boven halen van informatie uit de directe en indirecte bronnen, maar juist ook uit het relateren van de subjectieve, mogelijk incorrecte indirecte informatie aan de objectieve werkelijkheid van de broncode. Beide vormen van informatie worden geïnterpreteerd en vertaald naar risico's van het oorspronkelijke probleem. Deze stappen worden geïllustreerd in figuur 1.

De methode is nog in ontwikkeling. Software risk assessments worden voor zeer uiteenlopende doeleinden gebruikt, wat leidt tot een steeds verdere verbetering van de methode. Na elk assessment wordt een *debriefing* uitgevoerd, waarin de methode en zijn toepassing kritisch worden geanalyseerd en waar nodig aangepast.

### De praktijk

Voorbeelden van assessments zoals we die in de praktijk hebben uitgevoerd omvatten onder meer de volgende situaties.

- Een bedrijf koopt een standaardpakket. Dit pakket voldoet niet aan



specifieke extra wensen van het bedrijf. De maker van het pakket wordt gevraagd om het aan deze extra wensen aan te passen. Na levering van het gewijzigde pakket ondervindt het bedrijf grote problemen bij de invoering, het vraagt zich af wat de risico's van het 'live gaan' met dit systeem inhouden, in ogenschouw nemende dat het pakket data voor miljoenen gebruikers moet verwerken.

- Een bedrijf koopt een standaard e-businesspakket voor de implementatie van een virtuele marktplaats in de energiehandel. Het bedrijf doet zelf het onderhoud: de leverancier bouwt versie 1.0 en draagt daarna de broncode over. Om inzicht te krijgen in de onderhoudsrisico's

wil het bedrijf een beoordeling van het pakket.

### Informatiebronnen

Elk software risk assessment begint met de organisatie van een workshop waar de diverse belanghebbenden aan deelnemen. Zo'n workshop heeft een aantal doelen. Ten eerste maakt het iedereen binnen een project bewust van het feit dat er een assessment wordt uitgevoerd. Tijdens de workshop worden het doel van het assessment en de hiervoor gebruikte methode uitgelegd. Afhankelijk van het doel van het assessment kan het zinvol zijn twee workshops te organiseren. Een voorbeeld betreft een softwareontwikkelingsteam voor bedrijf A



en een implementatieteam voor bedrijf B; A en B zijn niet even gelukkig over hun relatie. Een belangrijk doel van de workshop is het wegnemen van angst en onzekerheid. Tijdens de workshop leggen we expliciet uit dat het assessment beoogt risico's te identificeren en tot aanbevelingen te komen over hoe het systeem verbeterd kan worden, en dat het vinden van zondebokken niet tot de doelstellingen behoort.

Daarnaast is een workshop veruit de snelste manier om een grote hoeveelheid informatie over het systeem te verzamelen. Doorgaans is er uitgebreide kennis over diverse gespecialiseerde onderdelen van het systeem aanwezig. Een voorbeeld uit de praktijk betreft een dba die van alles wist over goede en slechte manieren om de database te benaderen. Ook had hij allerlei verhalen over de hoe hij de databaseparameters moest zetten op waardes die 'ver boven het gemiddelde voor dit soort systemen' lagen. Deze databasekennis wordt gecombineerd met informatie uit andere bronnen, zoals de sources, om een accurate risicoanalyse uit te kunnen voeren. Afhankelijk van de aard van de organisatie die het assessment laat uitvoeren, is zulke informatie al dan niet bekend bij het projectmanagement. Doorgaans kunnen technische specialisten al in een vroeg stadium van een project zien dat bepaalde zaken afwijken van het gangbare. De communicatie- en beoordelingsstructuur moet zo zijn dat de specialisten hun (technische) bevindingen daadwerkelijk aan het

projectmanagement rapporteren, en het management moet in staat zijn de implicaties van deze bevindingen en afwijkingen te interpreteren. Ten derde helpt de workshop bij het vaststellen van wat de werkelijk relevante vragen zijn. Dit hoeft niet overeen te komen met de perceptie van het management van deze problemen. Weer een praktijkvoorbeeld: in sommige gevallen kan performancetuning van een database bereikt worden door indices aan de juiste kolommen toe te voegen. Als deze indices ontbreken, zal de performance van het systeem ver beneden de maat zijn. Voor het management kan dit een groot probleem lijken, terwijl de database-specialist weet dat hij dit in enkele uren voor elkaar kan krijgen, meteen na het afronden van zijn huidige project, dat klaarblijkelijk een hogere prioriteit had. De workshop vergelijkt allerlei, soms tegenovergestelde meningen over de risico's (en sterke punten) van het systeem. De workshop beoogt primair deze meningen te inventariseren, een consensus te vinden en op grond van deze meningen de belangrijkste risico's zoals ze gezien worden in kaart te brengen en te prioriteren.

### Vorbereiding

Voordat de workshop wordt georganiseerd voeren we een analyse uit op de broncode. Dit geeft een algemeen idee over de omvang en de architectuur van het systeem. Een belangrijke stap is het identificeren van uitzonderingsgevallen: modules of subsystemen die afwijken van het gemiddelde. Dit kan gebeuren met volumetrie (zoals het aantal regels code), maar ook door het aantal databasebenaderingen, de cyclomatische complexiteit, het aantal sockets dat wordt geopend voor lezen, enzovoorts. Om de juiste metriek te bepalen

gebruiken we onze intuïtie: we hebben een algemeen idee over wat het systeem moet doen. We proberen ons voor te stellen hoe zo'n systeem ontworpen en gebouwd zou worden, en proberen wat we zien en meten te relateren aan wat we ons hebben voorgesteld. Het imaginaire systeem dient als eerste mentale model van het systeem. Tijdens de analyse, maar ook tijdens de workshop wordt dit mentale model vele malen herzien zodat het steeds meer op het werkelijke systeem gaat lijken.

Gebaseerd op onze ervaringen en op de literatuur over het onderwerp hebben we een vragenlijst opgesteld die een groot aantal onderwerpen op het terrein van software-systeemontwerp, -bouw, -implementatie, -testen en meer behandelt. Deze vragenlijst wordt gebruikt om de workshop te structureren. Afhankelijk van het zwaartepunt van het software risk assessment worden specifieke vragen al dan niet aan de deelnemers van de workshop gesteld. Na elk software risk assessment wordt de vragenlijst geëvalueerd en verbeterd. In de geest van de Architectural Tradeoff Analysis Method (Atam) (Kazman e.a., 1998) van het Software Engineering Instituut aan de Carnegie Mellon-universiteit, hebben we vragen opgenomen over de kwaliteitsafwegingen en (toekomstige) scenario's die de verschillende onderdelen van het systeem beïnvloeden.

Tijdens de workshop (en überhaupt in deze fase van het software risk assessment) proberen we voornamelijk de problemen zoals de deelnemers die zien in kaart te brengen. In de hierop volgende fase worden deze problemen vergeleken met de werkelijke toestand van de broncode.

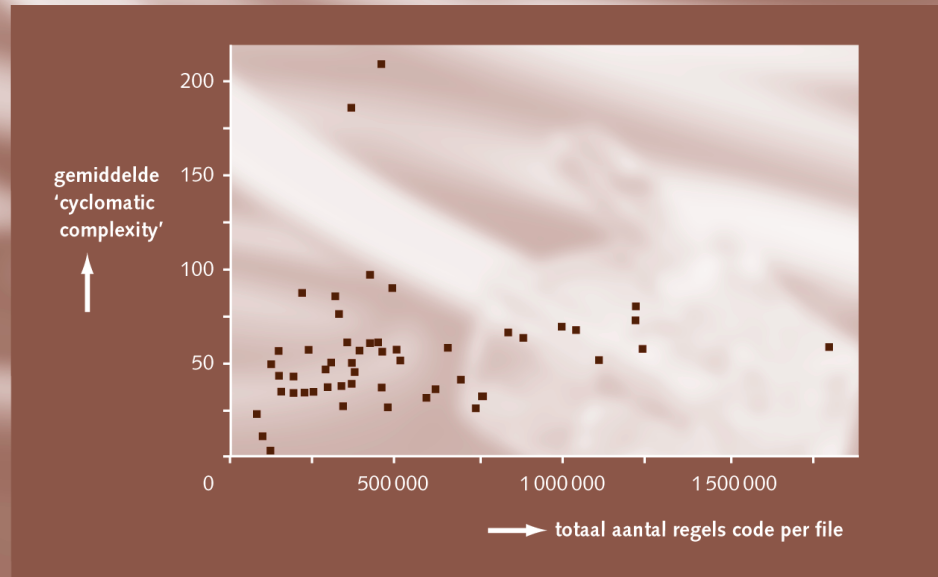
### Uitnodigen

Het uitnodigen van deelnemers aan

de workshop kan een sterk politiek proces zijn. Wie er wordt uitgenodigd hangt sterk af van wie het software risk assessment initieert en waarom. Een aantal belanghebbenden moet hoe dan ook altijd worden uitgenodigd. Afhankelijk van de politieke situatie worden deze mensen wel of niet uitgenodigd, en als ze worden uitgenodigd kan het gebeuren dat ze geen toestemming krijgen om mee te doen. In sommige gevallen is het door dit soort omstandigheden effectiever (en efficiënter) om twee workshops te organiseren. Uit onze ervaringen blijkt dat het niet werkbaar is om meer dan acht mensen voor de workshop uit te nodigen. Workshops met meer dan acht deelnemers zijn moeilijker te sturen, en zorgen ervoor dat mensen eenvoudiger inactief kunnen blijven. Het is aan te raden om een workshop in tweeën te splitsen in een inleidend deel waarbij het management aanwezig is, en een meer technisch gedeelte waarvoor het management de kamer verlaat zodat de overige deelnemers vrijer over de verschillende aspecten van het systeem kunnen discussiëren. Een aantal mensen moet altijd worden uitgenodigd. *Projectmanager*. Degene die verant-

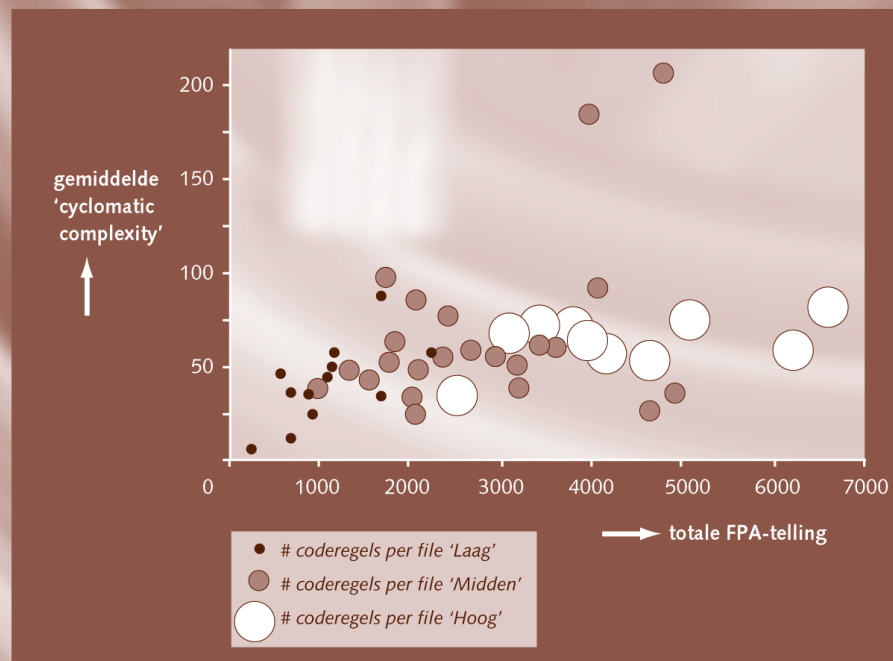
## Omvang in regels code versus complexiteit per module

2



## Functiepunten versus complexiteit per module en regels code

3



woordelijk is voor de dagelijkse gang van zaken in het project en die uiteindelijk verantwoordelijk is voor het succes van het project. *Klant*. Degene die het project betaalt. Als deze persoon geen of weinig directe bemoeienis met het project heeft, dan degene uit de klantorga-

nisatie die het meest betrokken is bij het project. In sommige gevallen is het zinvol ze beiden uit te nodigen. *Architect*. De architect van het systeem, voor zover aanwezig. Dit is de persoon die het grootste aandeel heeft in het ontwerp van het systeem.



**Hoofdontwikkelaar.** Degene die de dagelijkse leiding heeft over de programmeeractiviteit, en bij voorkeur zelf meeprogrammeert.

Iemand uit het middelmanagement die geen werkelijke kennis van het ontwikkelproces heeft is hier onbruikbaar.

**Hoofdtester.** Degene die de testactiviteit leidt.

**Diverse specialisten.** Afhankelijk van het software risk assessment kunnen diverse specialisten worden uitgenodigd. Voorbeelden zijn: de databasespecialist, de performance-tuningspecialist, de implementatiespecialist, de netwerkspecialist et cetera.

## Analyse

We hebben een raamwerk ontwikkeld om (zeer) grote hoeveelheden broncode, in verschillende programmeertalen, te analyseren. De meest prominente exponent van dit raamwerk is DocGen, een generator van technische documentatie, die de gebruiker in staat stelt op verschillende abstractieniveaus door een systeem te bladeren (Van Deursen & Kuipers, 1999).

Ook andere tools kunnen worden gebruikt om grote softwaresystemen te analyseren. In de *related-work* sectie van Moonen (2002) staat een uitgebreid overzicht van deze tools.

Het raamwerk bestaat uit een generieke parser (Van den Brand e.a., 2002), die wordt geconcretiseerd met een grammatica of taalmodel voor een specifieke programmeertaal. Met deze parser wordt broncode ontleed, en met JJForester

vertaald naar Java-objecten (Kuipers & Visser, 2002). JJForester is een tool die Java-klassen genereert voor boomstructuren, zoals bijvoorbeeld ontleedbomen. Het resultaat van de vertaling is een aantal objecthiërarchieën die ontleedbomen modelleren. Deze bomen kunnen worden verwerkt met behulp van JJTraveler (Visser, 2001). Met JJTraveler worden zeer ingewikkelde programma-analyses op een zeer beknopte wijze geformuleerd en in Java geprogrammeerd. Het grote voordeel van het gebruik van de taal Java (in plaats van meer in de academische wereld gebruikelijke talen voor programma-analyse) voor dit soort analyses is dat er een grote hoeveelheid Java-programmeurs beschikbaar is.

## Objectmodel

Het centrale gedeelte van het raamwerk is een objectmodel dat een aantal programmeertaalconstructies modelleert. Er zijn meerdere overlappende modellen, voor bijvoorbeeld procedurele talen, vierde generatietalen en objectgeoriënteerde talen. Uit dit objectmodel wordt een datamodel gegenereerd dat gebruikt wordt om een relationele database te maken waar de objecten in worden opgeslagen (met object-relationale mapping). Uit de database kan een aantal rapporten worden gegenereerd, ofwel door een gespecialiseerd programma (zoals het al genoemde DocGen), ofwel door gebruik te maken van standaard spreadsheets. Door deze architectuur kunnen we de analyses op twee niveaus aanpassen. Op basis van de gegevens in de database kunnen we metriecken uitrekenen die relevant zijn voor het huidige software risk assessment. We kunnen deze metriecken met grafieken weergeven, om ze beter inzichtelijk te maken of om trends in de metriecken te identificeren. We kunnen zo ook de uitzon-

deringsgevallen identificeren: welke module leest het meest uit een database, welke module schrijft het meest naar bestanden, welke module roept de meeste andere modules aan. Deze analyses worden uitgevoerd op het basisdatamodel. Als het datamodel een specifieke analyse niet toestaat kunnen we de analyses op het tweede niveau aanpassen: we schrijven een nieuwe programma-analyse in Java die wordt uitgevoerd op de ontleedboom van een programma (of een systeem). Indien noodzakelijk wordt het objectmodel voor de programmeertaal aangepast om de resultaten van de nieuwe analyse in de database op te slaan.

Over het algemeen zijn veranderingen op het eerste niveau (na de database) eenvoudiger uit te voeren dan veranderingen op het tweede niveau. Voor veranderingen op het tweede niveau moeten alle ontleedbomen opnieuw worden geanalyseerd, dat kan veel tijd kosten. We voeren veranderingen uit op het tweede niveau als de resultaten van de workshop voor een specifiek software risk assessment duidelijk maken dat zo'n analyse noodzakelijk is om de risico's te identificeren. Veranderingen op het tweede niveau worden nooit vóór de workshop uitgevoerd; dan wordt een aantal standaard metriecken afgeleid, en eventueel een paar aangepaste metriecken, door nieuwe analyses te maken op het eerste niveau.

## Benchmarking

De resultaten van de diverse software risk assessments worden allemaal bewaard. Dat geeft de mogelijkheid om analyses te doen met benchmarkinggegevens. We kunnen bijvoorbeeld de resultaten van een specifiek systeem vergelijken met de resultaten van een vergelijkbaar systeem dat door een andere organisatie is gebouwd. Op dit moment bevat onze database

gegevens die zijn afgeleid van ongeveer 45 gigabyte broncode, van zowel commerciële als overheidsorganisaties.

De volgende metrieken worden standaard berekend: het aantal modules in een systeem, het aantal regels code, het aantal programmeertalen, McCabe's cyclomatische complexiteit, de onderhoudbaarheidsindex van Oman en Hagemeister (Oman & Hagemeister, 1994), het met de backfiring-methode geschatte aantal functiepunten (Jones, 1998), de fan-in en fan-out van de modules, het aantal databasetabellen dat door een module wordt gebruikt, het percentage geduplicateerde (of gekloonde) code, het aantal parameters per module of procedure, het aantal velden per databasetabel en het aantal goto-statements.

### **Van cijfers naar risico's**

Het resultaat van het software risk assessment is een beoordeling van de risico's die zijn geïdentificeerd in de workshop, gebaseerd op de directe feiten die uit de broncode zijn afgeleid. Om deze beoordeling bruikbaar te maken voor de opdrachtgever bevat het rapport een interpretatie van de waarnemingen in de broncode. Deze interpretatie maakt duidelijk waarom de waarnemingen relevant zijn in relatie tot de geïdentificeerde risico's, en hoe deze waarnemingen gebruikt kunnen worden om oplossingen te vinden voor de geïdentificeerde problemen.

De rapportage bestaat uit de volgende onderdelen:

- Een objectieve beschrijving van de problemen zoals de belanghebbenden ze zien. Dit is in de workshop naar voren gekomen.
- Een objectieve beschrijving van de waarnemingen die gedaan zijn in, of op basis van de broncode, gerelateerd aan de geïdentificeerde problemen.

- Een objectieve inventaris van de potentiële risico's en voordelen voor elk van de technische waarnemingen.
- Een subjectieve evaluatie – gebaseerd op onze ervaring – van de risico's en voordelen. Deze leidt tot aanbevelingen om te risico's te verkleinen en de voordelen uit te buiten.

De op de broncode gebaseerde waarnemingen vormen het 'bewijs' van het rapport. De aanbevelingen vormen de best mogelijke interpretatie van de feiten, bedoeld om niet-technische managers de informatie te bieden die ze nodig hebben om in actie te komen en het project beter te maken.

### **Arie van Deursen**

werkt bij het Centrum voor Wiskunde en Informatica, Amsterdam en aan de Technische Universiteit Delft.  
E-mail: [Arie.van.Deursen@cw.nl](mailto:Arie.van.Deursen@cw.nl).  
Website: [www.cwi.nl/~arie/](http://www.cwi.nl/~arie/).

### **Tobias Kuipers**

werkt bij de Software Improvement Group Diemen. E-mail: [tobias.kuipers@software-improvers.com](mailto:tobias.kuipers@software-improvers.com).

## Literatuur

- Brand, M.G.J. van den, J. Scheerder, J. Vinju, & E. Visser (2002). Disambiguation filters for scannerless generalized LR parsers. In N. Horspool (editor). *Compiler Construction (CC'02), Lecture Notes in Computer Science*. Springer-Verlag, 2002.
- Deursen, A. van, & T. Kuipers (1999). Building documentation generators. In *International Conference on Software Maintenance, ICSM'99* (pp. 40-49). IEEE Computer Society, 1999.
- Jones, C. (1998). *Estimating Software Costs*. McGraw-Hill, 1998.
- Kazman, R., M. Klein, M. Barbacci, T. Longstaff, H. Lipson, & J. Carriere (1998). The Architecture Tradeoff Analysis Method. In *Proceedings 4th International Conference on Engineering of Complex Computer Systems*, 1998.
- Kuipers, T. & J. Visser (2002). Object-oriented tree traversal with JForester. *Science of Computer Programming*, 47(1):59-87, November 2002.
- Moonen, L. (2002). *Exploring Software Systems*. PhD thesis, University of Amsterdam, December 2002.
- Oman, P. & J. Hagemeister (1994). Constructing and testing of polynomials predicting software maintainability. *Journal of Systems and Software*, 24(3):251-266, 1994.
- Visser, J. (2001). Visitor combination and traversal control. *ACM SIGPLAN Notices*, 36(11):270-282, November 2001. OOPSLA 2001 Conference Proceedings.

### **IEEE-conferentie onderhoud en beheer**

In de week van 22 september van dit jaar komt de grootste conferentie over onderhoud en beheer naar Amsterdam. Deze International Conference on Software Maintenance (ICSM) startte in 1983 en is daarna vrijwel elk jaar gehouden. Deze conferentie, die onder auspiciën van de IEEE Computer Society staat, heeft ook dit jaar vele toonaangevende sprekers uit binnen en buitenland, aanpalende workshops, tutorials en meer. Dit thema is een mooi voorproefje van wat er in Nederland en België op dit gebied aan hoogwaardige technologie ontwikkeld wordt.  
[www.cs.vu.nl/icsm200](http://www.cs.vu.nl/icsm200)