

Intelligent Design?

Vorig jaar was het jaar van Darwin. Op tv probeerden Darwinisten hun jaar op te eisen. Creationisten en Intelligent Design-aanhangers vochten hard terug. Zo hard, dat ik in dat Darwinjaar vooral de indruk kreeg dat Nederland een land van creationisten en andere religieuze hardliners was – meer nog dan in andere jaren. De veelbesproken uitspraken van Andries Knevel en de hartstochtelijke pleidooien van Midas Dekkers en Bas Haring werden allemaal fel van replek voorzien. Arme biologen. Ze zullen wel blij zijn dat het Darwinjaar voorbij is. Eindelijk hebben ze rust. Nu kunnen ze weer in alle rust genieten van Darwins gedachtegoed zonder continu lastiggevallen te worden.

In ons vakgebied zijn we gelukkig gevrijwaard van dit soort discussies en stammenstrijden. Onze producten (software) zijn overduidelijk het product van design. Of dat design ook 'intelligent' is, valt alleen maar te hopen. We hebben het alleen over een heel ander soort design. Geen goddelijk design, maar menselijk design. Wat religie ook mag betekenen in uw (of mijn) leven, onze producten hebben geen directe religieuze achtergrond. In ons vakgebied werken gelovige en ongelovige IT'ers dan ook in goede vrede naast – en met – elkaar. Gelukkig maar.

Religie heeft dus geen directe invloed op ons werk. Maar wel degelijk een indirecte invloed. Mijn stelling is dat onze culturele achtergrond – en daarmee in grote mate onze religieuze achtergrond – overal terug te vinden is in de keuzes die wij maken voor onze systemen en architecturen. Onze (westerse) religieuze achtergrond is er een van één God, één waarheid, één entiteit die alles overziet. Die achtergrond leidt ertoe dat wij – zodra



Zodra systemen een beetje complex worden, grijpen we direct naar centrale orkestratie

onze systemen ook maar een beetje complex worden – direct naar een centrale orkestratie grijpen. De ultieme IT-visie is die van 'de business' die van boven af alle processen in de organisatie overziet, alle cijfers onder de vingers heeft en 'aan de knoppen kan draaien' om processen direct te veranderen – als een God in de eigen organisatie.

Ik ben van mening dat die centralistische, orkestrerende aanpak ons beperkt. Wanneer een systeem (of een verzameling van systemen) echt complex gaat worden, dan is het op een gegeven moment niet meer mogelijk om uit te gaan van één waarheid, één set overkoepelende processen en één orkestrerende eenheid. Er is een punt waarop we over moeten stappen op werkelijk gedistribueerde systemen. Systemen zonder (of met zeer beperkte) orkestratie. Systemen waarin processen 'van onderen uit' ontstaan in plaats van deterministisch ontworpen te worden. Systemen die zo groot zijn dat ze niet meer te 'designen' zijn maar evolutionair moeten ontstaan. Systemen die steeds meer gaan lijken op het meest complexe (en mooiste) systeem ooit: het leven zelf.

En daarmee zijn we weer terug bij Darwin. Zijn evolutietheorie is niet letterlijk bruikbaar in ons vakgebied. Evolutionair ontwikkelen heeft niet zoveel te maken met het ontstaan van soorten. Maar de evolutietheorie is wel een realistisch niet-deterministisch model voor een overweldigend complex systeem. Dit model zou ons vertrouwen moeten geven om onze eigen centralistische en deterministische gewoonten te kunnen laten vieren – ook voor de relatief simpele systemen die wij zelf maken. Of u een religieuze argwaan heeft tegen de evolutietheorie, maakt daarbij niet uit. We hebben het tenslotte over volledig 'man-made' systemen.

Daan Kalmeijer is senior adviseur/docent bij DNV-CIBIT.

Meetresultaten softwarekwaliteit van verschillende tools vertonen onaanvaardbaar grote verschillen

Metriekentools niet betrouwbaar

De bestaande metriekentools, zeggen **Jacob Brunekreef** en **Dennis Breuker**, zijn ongeschikt om de kwaliteit van software te meten. Managementinformatie op basis van deze tools is vaak niet betrouwbaar. Er moeten operationele definities van metrieken komen die bij voorkeur worden opgenomen in een internationaal erkende standaard.

van het in ontwikkeling of in beheer zijnde product is gesteld. In het verleden zijn tal van dergelijke kwaliteitsmetrieken gedefinieerd en zijn tools ontwikkeld waarmee de betreffende metrieken automatisch gemeten kunnen worden. Daarmee hebben ontwikkelaars en beheerders een belangrijk instrument in handen om de kwaliteit van hun software te toetsen en te verbeteren. Een kwaliteitsraamwerk als CMMi kent een verplicht in te richten Measurement & Analysis-proces, waarin het meten en verbeteren van eigenschappen van product en proces tot een basisactiviteit worden benoemd. Managementdashboards zijn vaak (direct of indirect) gebaseerd op een geautomatiseerde analyse van metrieken afkomstig uit programmacode. Een belangrijke vooronderstelling is dat de gebruikte metriekentools betrouwbaar zijn: dat ze meten wat ze zeggen te meten, en dat de uitkomsten een correcte afspiegeling zijn van de werkelijkheid. En dat blijkt helaas nog lang niet altijd het geval te zijn. In het kader van een meetprogramma rond softwarekwaliteit hebben onderzoekers van de Hogeschool van Amsterdam de betrouwbaarheid van een aantal populaire metriekentools onderzocht. Met niet altijd even positieve resultaten (zie kader).

Veel metrieken zijn in een meer of minder ver verleden gedefinieerd en zijn beschreven in wetenschappelijke publicaties. Bekende metrieken zoals de McCabe Cyclometrische Complexiteit en de Chidamber-Kemerer-set zijn gedefinieerd in artikelen van jaren her. Deze artikelen vormen als het ware de specificaties voor toolbouwers. Maar die artikelen zijn nooit als specificatiedocument geschreven. Dat betekent dat definities in deze artikelen niet direct een operationeel karakter hebben. En dat is wel wat toolbouwers nodig hebben. Het gevolg is dat men moet gaan bouwen op basis van een afgeleide definitie. Met het gevaar dat iedere toolbouwer zijn eigen afgeleide definitie hanteert, en daarmee een meetresultaat produceert dat niet vergelijkbaar is met de uitkomsten van andere tools. Een duidelijk voorbeeld is de definitie van de eerdergenoemde cyclometrische complexiteit (zie kader). Operationele softwaremetrieken zijn,

wat betreft producteigenschappen, vrijwel altijd gebaseerd op tellen: het aantal regels code in een programma, het aantal keuzestatements in een methode, het aantal relaties tussen klassen et cetera. Om objectief te kunnen tellen, is een eenduidige definitie van de te tellen grootte nodig. Soms is dat niet moeilijk (bijvoorbeeld bij een regel code). Soms, zoals bij keuzestatements, is de definitie noodzakelijkerwijs taalspecifiek – iedere taal kent haar eigen varianten. Om als definitie voor een betrouwbare metriek te kunnen dienen, moeten de verschillende varianten expliciet benoemd worden. Een lastig karwei in een wereld met een veelvoud aan programmeertalen die zich deels nog steeds verder ontwikkelen en waar om de zoveel tijd nieuw talen ontstaan (denk aan Scala). Bij sommige metrieken worden berekeningen uitgevoerd met de telresultaten. Ook die berekeningen moeten deel uitmaken van de operationele definitie van de betreffende metriek.

Het gevaar bestaat dat elke toolbouwer zijn eigen afgeleide definitie hanteert

Onderzoek laat zien dat de huidige generatie metriekentools onvoldoende betrouwbaar is. Meetresultaten van een en dezelfde eigenschap, verkregen met verschillende tools, tonen te vaak een onaanvaardbaar grote spreiding. Hiervoor zijn verschillende oorzaken aan te geven. Aan de ene kant zijn er problemen met de formuleringen in de literatuur: onduidelijke of niet-operationele definities van metrieken en/of het wijzigen van definities in de loop van de tijd maken het toolbouwers lastig om een goed product te leveren. Aan de andere kant zijn er problemen bij de toolbouwers: ontbrekende of onzorgvuldige specificaties van wat er gemeten wordt, soms gebrekkige implementaties. De instrumenten die worden ingezet ter bestrijding van de softwarecrisis zijn daarmee zelf slachtoffer van die crisis en krijgen zo het karakter van een Baron van Münchhausen die zich aan zijn



ILLUSTRATIE: JOS THOMMASEN

eigen haren uit het moeras moet trekken. De huidige generatie metriekentools is ongeschikt om in te zetten als absolute graadmeter voor de kwaliteit van software. Je weet niet echt wat je meet. Managementinformatie verkregen op basis van metingen met deze tools is niet altijd te vertrouwen. Dat betekent dat mogelijk verkeerde beslissingen genomen worden, met alle gevolgen van dien. Daarmee willen we niet betogen dat het inzetten van de huidige generatie metriekentools in alle gevallen zinloos of alleen maar gevaarlijk is. De tools kunnen

bijvoorbeeld prima ingezet worden om zwakke plekken in programmatuur te identificeren. Of voor één specifieke methode de exacte waarde van de cyclometrische complexiteit nu 100 of 150 is, maakt niet zo veel uit, in beide gevallen zijn aanpassingen gewenst. Ook kan een en hetzelfde tool prima gebruikt worden om tijdens de ontwikkeling van code de kwaliteit te bewaken. Het stijgen (of dalen) van een meetwaarde is dan van groter belang dan de correctheid.

Om tot een serieuze verbetering van de gescheste situatie te komen, is een aantal stappen noodzakelijk. Allereerst

Tools wel geschikt voor identificeren zwakke plekken in programmatuur moeten er operationele definities worden opgesteld van metrieken die met tools gemeten worden, gebaseerd op het tellen van duidelijk omschreven grootheden.

Bij voorkeur worden deze definities in een internationaal erkende standaard opgenomen. Taalspecifieke aspecten moeten niet overgelaten worden aan toolbouwers (met het gevaar van een verschil in aanpak, maar worden opgenomen in bijlagen van de standaard. Toolbouwers dienen zich expliciet te committeren aan deze definities, inclusief taalspecifieke invullingen. Een afwijkend meetresultaat is dan uitsluitend te herleiden tot een implementatiefout van de toolbouwer. Alleen met een dergelijke strikte benadering kunnen metriekentools daadwerkelijk bijdragen aan het bestrijden van de softwarecrisis.

Jacob Brunekreef is lector Softwarekwaliteit bij de Hogeschool van Amsterdam (j.j.brunekreef@hva.nl). Hij is tevens senior consultant bij DNV-CIBIT. Dennis Breuker is docent/onderzoeker bij de Hogeschool van Amsterdam (d.m.breuker@hva.nl). Het onderzoek waarover in het artikel wordt gesproken is mede uitgevoerd door Jan Derricks en Ahmed Nait Aicha, docenten/onderzoekers bij de HVA. Een uitgebreidere rapportage van het onderzoek ('Reliability of software metrics tools') is te vinden via http://iws-mensura-2009.nesma.nl/download/IWSM_Mensura_2009_industrial_papers.pdf.

▀ Voor reacties en nieuwe bijdragen van deskundigen: Henk Ester (h.ester@sdu.nl, (070) 378 03 97).

Onderzoek

In het kader van een meetprogramma rond softwarekwaliteit hebben onderzoekers van de Hogeschool van Amsterdam de betrouwbaarheid van een aantal populaire metriekentools onderzocht. Voor het onderzoek is een selectie gemaakt van twaalf metrieken en zes verschillende tools. Voor deze metrieken en tools is allereerst onderzoek hoe een metriek in de literatuur is gedefinieerd en hoe de metriek in de tooldocumentatie is beschreven. Vervolgens zijn deze metrieken gemeten met de

betreffende tools. De beperkte omvang van het project maakte het mogelijk om de verschillende meetresultaten te controleren via een visuele inspectie van de code. De twaalf geselecteerde metrieken zijn onder te brengen in drie categorieën: omvang, structuur en complexiteit. Het meten van de metrieken in de eerste categorie (omvang) komt meer op het tellen van relatief eenvoudige (eenduidige) eenheden als 'classes', 'methods', fysieke regels. Desalniettemin zijn de uitkomsten

verrassend: ze vertonen onderling soms grote verschillen. Slechts twee tools leveren uitkomsten die overeenkomen met de controletoelating. Eén tool heeft zeer afwijkende resultaten. Bij nadere inspectie bleek dat dit tool niet in staat was om Java-versie 1.5 en hoger te verwerken: 'classes' en 'methods' met nieuwe taalconstructies werden simpelweg genegeerd. Deze beperking was nergens in de documentatie van het betreffende tool aangegeven.

De andere tools vertonen kleine afwijkingen die waarschijnlijk zijn terug te voeren op onterechte dubbelstellingen. De definitie van de metrieken in de tweede categorie (structuur) is vaak wiskundig van aard en dwingt de toolbouwer tot een eigen operationele definitie. Daarnaast is de definitie soms complex, wat maakt dat niet te verwachten is dat iedere toolbouwer tot dezelfde operationele definitie zal komen. De uitkomsten bevestigen dit: bij bijna alle

metrieken vertonen de meetresultaten een dusdanige spreiding dat nauwelijks nog gesproken kan worden van een serieuze meting van een en dezelfde grootte. Het niet-operationele karakter van de definitie van de metriek in de derde categorie (complexiteit) zorgt voor verschillende interpretaties van de toolbouwers. Dit was terug te vinden in de verschillende meetwaarden. Visuele inspectie kon deze verschillen goed verklaren.

Definitie cyclometrische complexiteit

De metriek – cyclometrische complexiteit – meet het aantal onafhankelijke paden door een programma (routine, method). In het oorspronkelijke artikel van McCabe (1977) staat een definitie op basis van de 'control flow graph' van de code. Het construeren van zo'n graph is duidelijk een brug te ver voor toolbouwers. Die werken vanuit een operationele definitie en gaan het aantal statements tellen dat leidt tot

een vertakking in de control flow graph: if-statements, while-statements, switch-statements et cetera. Dat leidt in principe tot dezelfde resultaten, maar genereert wel vragen: Introduceert het Java-exception-statement 'try ... catch ...' nu ook een extra pad door de code? Toolbouwers beantwoorden zelden dit soort detailvragen in hun beschrijving van hoe deze metriek gemeten wordt. Een andere bron van onduidelijkheid is

het wel of niet metellen van de complexiteit van een conditie. Sommige tools tellen het statement 'if A or B then ... else ...' dubbel, omdat zowel subconditie A als subconditie B meegeteld wordt. Andere doen dat niet. Ook deze keuze wordt lang niet bij ieder tool gespecificeerd. In de praktijk blijkt dat toolbouwers verschillende keuzes maken. En dan is het te verwachten dat meetresultaten uiteen gaan lopen.