

Nieuwe technologie roept nieuwe vragen op

**Architectuur,
configuratiebeheer,
alignment en
kwaliteitsmetingen
in onderzoek**

De gastredacteur belicht het belang van *software-engineering* en gaat in op de consequenties van het werken met nieuwe technologieën zoals componenten. Kwaliteitsbeoordeling komt hierbij in een nieuw licht te staan.

Hans van Vliet

Software zit bijna overal in tegenwoordig. En we krijgen er steeds meer van. Inmiddels wordt 25% van de prijs van een auto bepaald door de software die in die auto zit. Bij 500.000 verkochte auto's per jaar in Nederland, à €20.000, is dat toch zo'n €10 miljard aan software per jaar. Met (in Nederland ontwikkelde) software kunnen we bepalen welk schilderij van Herman Brood echt is, en welk vervalst. De sector is wat kleiner dan de automarkt, maar toch ook een aardige niche. Zo zijn er zeer vele voorbeelden te geven. We kunnen rustig stellen dat 'software makes the world go round'. Het vakgebied waarin men zich bezighoudt met het ontwikkelen van betere methoden en technieken om die software te ontwikkelen is de software-engineering. Daar moet dus ook wel toekomst inzitten.

Vroeger werd het vak software-engineering gedefinieerd en gemotiveerd vanuit een nogal negatief wereldbeeld: software wordt te laat opgeleverd, doet niet wat het moet doen, kost teveel enzovoort. In de volksmond: de software-crisis. Die crisis zou de software-engineering wel even oplossen. Zo eenvoudig was dat natuurlijk niet. En nog steeds wordt lang niet alle software op tijd opgeleverd, doet het niet altijd wat het zou moeten doen en lopen de kosten wel eens uit de hand. Dat lijkt meer op een ongeneeslijke ziekte dan op een (tijdelijke) crisis.

De vraag is natuurlijk of dit aan dat vakgebied te wijten is. Want er zijn gigantische vorderingen gemaakt. Die enorme bergen software waar ons dagelijks leven door gestuurd en bestuurd wordt zijn – mede door de enorme vorderingen op het gebied van de software-engineering (steeds betere ontwerpmethoden, testtechnieken, ontwikkeltools) – mogelijk geworden. En door nieuwe technologische ontwikkelingen staat het vakgebied ook steeds weer voor nieuwe uitdagingen.

Een simpel voorbeeld om dit laatste te illustreren: betrouwbaarheid (*reliability*). In de standaardliteratuur over software-engineering wordt dit gedefinieerd als 'de kans dat een systeem foutvrij zal werken gedurende een bepaald tijdsinterval in een bepaalde omgeving' (Van Vliet, 2000; Lawrence Pfleeger, 2001). De omgeving wordt dus genoemd in de definitie van betrouwbaarheid. Een systeem kan in de ene omgeving betrouwbaarder zijn dan in een andere. Die omgeving betreft niet alleen de hardware waarop het systeem draait, maar ook de mensen en de wijze waarop zij met het systeem omgaan. Stel, ik heb een reisplanner waarin gegevens van zowel treinen, bussen als veren zijn opgenomen, maar de gegevens over veerdiensten worden niet correct verwerkt. Dan is er een behoorlijke kans dat gebruikers in

Limburg een hogere betrouwbaarheid van dit systeem waarnemen dan gebruikers in het noorden van het land. Dit is natuurlijk bekende kost, en is mede de reden dat we heel voorzichtig moeten zijn met generalisatie van resultaten behaald in een testomgeving naar een operationele omgeving.

Maar hoe nu om te gaan met het begrip betrouwbaarheid in de huidige, componentgebaseerde ontwikkelomgevingen? Hoe moet ik de betrouwbaarheid van een component weergeven? Kan ik dat begrip überhaupt wel gebruiken als ik geen idee heb waar en hoe die component gebruikt wordt? En kan ik de betrouwbaarheid van verschillende componenten bij elkaar 'optellen' om zo de betrouwbaarheid van een componentgebaseerd systeem te bepalen?

»Nederland heeft eigen software-engineering-onderzoek nodig«

Als we op basis van componenten die we bijvoorbeeld via het web aanschaffen een systeem willen samenstellen dat aan de vereiste kwaliteitseisen voldoet, moeten de interfaces van die componenten ook iets over kwaliteit uitdrukken. Veel aanbieders van componenten zullen niet graag al te veel informatie over hun componenten prijsgeven. Mary Shaw heeft in dit verband voorgesteld 'geloofsbrieven' bij elke component op te nemen (Shaw, 1996). Die kunnen bijvoorbeeld van de vorm <attribuut, waarde, kennis> zijn, waarbij 'kennis' weergeeft hoe je aan de waarde van het betreffende kwaliteitsattribuut bent gekomen (van horen zeggen, zelf getest, informatie van een ander bedrijf of de componentenconsumentenbond). Dat kan iets zeggen over de betrouwbaarheid van, of het vertrouwen in de informatie. Deze kennis kan in de loop der tijd veranderen – bijvoorbeeld bij goede ervaringen met het gebruik van die component. Zij kan ook voor verschillende gebruikers iets anders betekenen.

Het testen van componenten vereist ook de nodige aandacht. Je kunt namelijk niet afgaan op de tests die de ontwikkelaar van die component zelf heeft gedaan. Perfect gedrag van een component in een bepaalde omgeving is geen garantie dat die component ook in een andere context goed zal werken. Het voorbeeld van de Ariane 5-raket is hier een perfecte illustratie van. Het mislukken van de eerste Ariane 5-lancering kwam door een softwarefout in een bepaalde component. Diezelfde component kwam ook in de software van de Ariane 4 voor, en de Ariane 4 was zo'n 40 keer succesvol gelanceerd. Maar omdat de omstandigheden van de Ariane 4 anders waren, was die fout nooit aan het licht gekomen. Als de context waarin een component wordt gebruikt verandert, moet die component opnieuw getest worden. Eén mogelijke techniek hiervoor is om bij een component niet alleen aan te geven hoe hij gebruikt moet worden, maar ook hoe hij getest moet worden.

Nieuwe ontwikkelingen, zoals die rond componentgebaseerde ontwikkeling, webservices en internetapplicaties roepen nieuwe onderzoeksvragen op. Daarmee blijft het vakgebied levend en uitdagend. Hoe levend en uitdagend het vakgebied ook is, als wij als Nederland iets willen blijven betekenen op dit terrein, moet er wel in geïnvesteerd worden. Dan moeten we niet volhouden te denken dat al die resultaten wel uit de VS geïmporteerd kunnen worden, zoals onze Minister van Onderwijs op het laatste ict-kenniscongres meldde. Nederland heeft eigen software-engineeringonderzoek nodig (Automatisering Gids, 2003). Voorbeelden van goed onderzoek van eigen bodem komen in deze special aan bod.

Kwaliteit

Al die verworvenheden en inzichten uit het vakgebied moeten natuurlijk wel toegepast worden om de vruchten te plukken. En daar schort het nog wel eens aan in Nederland. Zonder ook maar enige pretentie van volledigheid te hebben, vallen mij in contacten met bedrijven verschillende zaken op.

Software-inspecties (of varianten als Fagan-inspecties en walkthroughs) worden weinig toegepast, hoewel uit onderzoek is gebleken dat dit



een uiterst kosteneffectieve testmethode is. In heel veel gevallen worden hiermee 50% van de fouten ontdekt. In Nederland gelden kennelijk andere wetten.

In veel omgevingen heeft men geen idee van de eigen prestaties. Op vragen als 'hoeveel fouten hebben jullie vorige week gevonden', 'hoeveel fouten denk je dat er nog in het product zitten', 'hoeveel regels code schrijven jullie gemiddeld per week', of 'wat is de productiviteitsverbetering nu jullie ontwikkeltool X hebben ingevoerd' moet men meestal het antwoord schuldig blijven. We meten niet, en we weten dus ook niet.

Goed configuratiebeheer is zeker niet overal aanwezig. Ik heb een simpele lakmoestest in gesprekken over Capability Maturity Modeling. Ik hoor nog wel eens dat men denkt toch zeker wel op niveau 2 of 3 te zitten. Op mijn vraag of men van het product dat men nu aan het ontwikkelen is even de versie van eergisteren voor mij kan reproduceren, is het antwoord vaak 'uh, nee, we bewaren alleen de laatste versie'. Zo'n organisatie zit op CMM niveau 1.

Los van dit soort concrete zaken valt mij de manier van omgaan met automatiseringsprojecten op. Gerald Weinberg heeft rond 1970 een leuk onderzoekje gedaan. Hij vroeg een aantal studenten één en hetzelfde programmeerprobleem op te lossen. De ene student kreeg als opdracht een snelle oplossing te zoeken, een ander moest een zo betrouwbaar mogelijk programma opleveren, een derde moest vooral op de leesbaarheid van de oplossing letten. Elk van de studenten leverde een verschillende oplossing in, geoptimaliseerd voor de specifieke kwaliteitseis die hij of zij had gekregen. Het zijn ook net mensen.

Ook software-engineeringprojecten worden door mensen uitgevoerd. Als die mensen veel regels code of veel functiepunten moeten produceren (bijvoorbeeld omdat zij daarop beoordeeld worden), dan worden er ook veel regels code of functiepunten geproduceerd. Als de projectleider beoordeeld wordt op het op tijd opleveren van een systeem, zal hij zijn uiterste best doen dat systeem op tijd op te leveren, en kan de kwaliteit van het testen daar bijvoorbeeld onder lijden.

Projecten leveren vaak geen herbruikbare componenten op omdat een projectleider daar niet op afgerekend wordt. Code is vaak niet goed onderhoudbaar omdat het team wordt afgerekend op het huidige project, dat afgesloten wordt zodra het systeem over de muur naar beheer wordt gegooid. Je krijgt wat je vraagt. Nieuwere en betere technieken om software te ontwerpen doen dit tij niet keren.

Jacquard

Als software zo belangrijk is voor de economie, behoort software-engineering natuurlijk ook een belangrijke plaats in het Nederlandse onderzoekslandschap in te nemen. Dit hebben NWO en de ministeries van EZ en OCenW ook onderkend, en ze hebben samen het onderzoeksprogramma Jacquard gestart. Jacquard staat voor Joint Academic and Commercial QUALity Research and Development. Joseph-Marie Jacquard (1752-1834) is de uitvinder van een weefstoel die kon worden 'geprogrammeerd' met geperforeerde kartonnen kaarten. De subsidiepot is ruim € 6 miljoen groot en het meeste daarvan is nog te verdelen. In 2003 zijn de eerste vier projecten goedgekeurd. In dit software-engineeringnummer is over drie van deze projecten een artikel opgenomen. Het vierde project, over productsoftware, is aan de orde geweest in het oktobernummer van 2003.

De agenda voor Jacquard is opgesteld door een programmacommissie met een evenredige vertegenwoordiging uit bedrijven en onderzoeksinstellingen. Deze commissie heeft vier thema's binnen het vakgebied software-engineering gekozen. Onderzoek op deze vier terreinen wordt van vitaal belang geacht voor de Nederlandse kenniseconomie, en sluit ook goed aan bij expertise in Nederland, zodat verwacht mag worden dat op deze terreinen succesvol onderzoek kan worden verricht dat voor het bedrijfsleven relevant is. Deze vier thema's zijn architectuur, configuratiebeheer, alignment en kwantificering van kwaliteit.

Architectuur

In een architectuurbeschrijving worden de vroegste ontwerpbeslissingen vastgelegd. Deze vroege beslissingen hebben verregaande gevolgen, omdat een correctie tijdens een latere fase hoge inspanningen en kosten met zich brengt. Tijdens de architectuurfase worden in 10% van de tijd, 90% van de kosten bepaald.

Configuratiebeheer

Complexe softwaresystemen zijn configuraties van deelsystemen en componenten. Elk hiervan bestaat in verschillende versies en kan op verschillende manieren geconfigureerd worden. Met de komst van componentgebaseerd ontwikkelen, gedistribueerd ontwikkelen en dergelijke zijn hier vele nieuwe onderzoeksproblemen ontstaan, maar ook interessante mogelijkheden tot significante productiviteitsverbeteringen.

Alignment

Alignment betreft de afstemming tussen verschillende entiteiten, zoals de systeembenaderingen van twee bedrijven die samenwerken, de bedrijfscontext tegenover de geautomatiseerde versie daarvan, de automatiseringsinfrastructuur tegenover de applicatieportfolio. Goede alignment betekent een betere ondersteuning van bedrijfsprocessen en betere mogelijkheden voor organisatieveranderingen.

Kwantificering van kwaliteit

We meten niet en dus weten we ook niet. Als we er in slagen de kwaliteit van een systeem te kwantificeren, meetbaar te maken, kunnen we verantwoorde beslissingen nemen over architectuurkeuzes of systeemrenovatie.

De eerste vier goedgekeurde projecten gaan allen over architectuur en configuratiebeheer. In de volgende ronden (in 2004 en 2005) zal vooral ook geprobeerd worden projecten van de grond te krijgen op de beide andere gebieden. Daartoe zullen een of meer *matchmaking* bijeenkomsten worden georganiseerd waarop onderzoekers en bedrijfsleven met elkaar discussiëren over potentiële onderzoeksideeën die in een vervolgstap tot

concrete voorstellen kunnen worden uitgewerkt. De eerste *matchmaking* dag vindt plaats op 10 maart 2004. Voor nadere informatie over deze dag: zie www.jacquard.nl.

Software-engineering is een vak met toekomst. Daarvoor is goed onderwijs en goed onderzoek nodig. Op beide terreinen doet Nederland mee, zoals blijkt uit de bijdragen in dit nummer. In de bijdragen van Ballintijn e.a., Dolstra e.a. en Tekinerdoğan e.a. wordt een indruk gegeven van lopend software-engineeringonderzoek in het kader van Jacquard. In de bijdrage van Kuipers wordt getoond hoe software-engineeringonderzoek kan leiden tot commercieel interessante en waardevolle toepassingen. De bijdragen van Van den Brand e.a. en Van der Raadt e.a. laten zien hoe state-of-the-art software-engineering aan de orde komt in het onderwijs en wat dit op kan leveren.

Literatuur

- Aksit, M. e.a. (2003). Nederland onderschat belang software-ontwikkeling. In *Automatisering Gids* 46, 14 november 2003.
- Lawrence Pfleeger, S. (2001). *Software Engineering: Theory and Practice*. Prentice Hall.
- Shaw, M. (1996). Truth vs Knowledge: The Difference Between What a Component Does and What We Know It Does. *Proceedings 8th International Workshop on Software Specification and Design*. IEEE, 1996.
- Vliet, H. van (2000). *Software Engineering: Principles and Practice*. John Wiley & Sons.

Hans van Vliet

is hoogleraar software-engineering aan de Vrije Universiteit. E-mail: hans@cs.vu.nl.