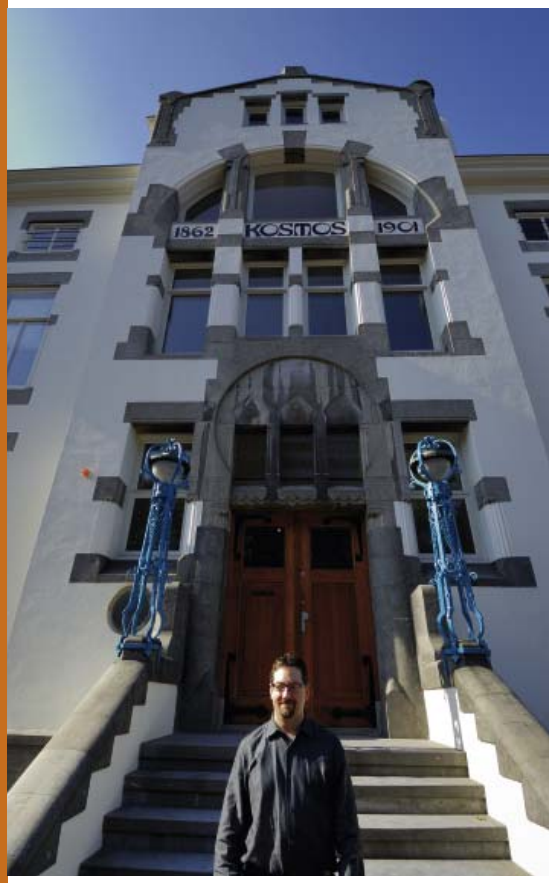
Core

Justin Gehtland is medeoprichter van het consultancy & trainingsbedrijf Relevance LLC, gevestigd in North Carolina, USA. Binnen de wereld van Java en Ruby staat Justin mede bekend om zijn onderzoek in het kader van efficiëntieverbeteringen van ontwikkeltrajecten met het framework Ruby on Rails. Hij schreef samen met Stuart Halloway het boek 'Rails for Java developers'. Bij gelegenheid van zijn workshop voor QNH sprak Java Magazine met hem.

Ruby voor Java-ontwikkelaars

Interview Justin Gehtland

Gehtland is geen Ruby-aanhanger van het eerste uur. Ruby bestond al tien jaar toen Gehtland zich ervoor begon te interesseren.



Gehtland voor het gebouw van QNH

Gehtland: "Het moet bijna drie jaar geleden zijn geweest dat Dave Thomas – de auteur van *Programming Ruby* en *Agile development with Rails* – een vriend van ons, begon te praten over Rails. Ik werkte in die tijd ook veel met Bruce Tate, ook een van de auteurs en we bouwden een Java-project voor een start up. Het was in wezen een beheersysteem voor een computerchip fabrikant. Het was webgebaseerd zodat ze het gemakkelijk zouden kunnen distribueren over de fabriek. Geen klein systeem, wel complex, maar niet veel belasting omdat er maar veertig mensen werkten. We hebben vijf maanden gewerkt aan Spring, Hibernate, de standaard lichtgewicht Java-stack. Ik heb toen besloten om dat project voor educatieve doeleinden ook in Rails te doen. Dat kostte me vier avonden/nachten. Ik heb daarover geblogd en iedereen viel over me heen. Wat ik zei was dat ik gegeven het feit dat het systeem al eens had geïmplementeerd had ik natuurlijk een vliegende start had. In Bruce's boek *Beyond Java* is hier een heel hoofdstuk aan gewijd. Maar zelfs wanneer je rekening houdt met het feit dat je niet opnieuw alle architectuur- en designstappen hoeft te zetten, is vier avonden een zeer indrukwekkende prestatie. Dat was het moment waarop we besloten dat het framework

Dré de Man
Tekst en fotografie:

iets belangrijks moest hebben en vanaf dat moment is het een belangrijk onderdeel (kern) geworden van wat we doen. Mijn bedrijf is opgericht als een Java en .Net consulting- en trainingbedrijf. We deden alleen maar Java en .Net op dat moment, maar we hadden heel verschillende achtergronden.

Vanaf het moment dat we RoR eraan toegevoegd hebben, zijn we ieder jaar in omzet verdubbeld. Het grootste deel van die groei kwam door het adopteren van RoR. Natuurlijk, het is iets nieuws en er wordt veel over gepraat, maar nog steeds: we zijn in staat geweest om meer projecten te produceren in een jaar met een kleiner team dan we ooit hadden kunnen doen met Java en .Net. Voor ons was dat het bewijs.

Natuurlijk zijn we ook enthousiast over light weight development en agile methodes, *Better faster lighter Java*. We werken ook op die manier, houden van kleine teams en snelle *turn around*. Ruby in het algemeen en Rails in het bijzonder geven ons een echt krachtig gereedschap om het team klein te houden. Dat is van grote waarde voor ons en voor onze klanten geweest. Sinds die eerste ervaring doen we professioneel Rails development en genieten daar echt van. Daarvóór was Ruby een tool, maar we bouwden er geen applicaties in. Rails opende een deur. Nu hebben we verschillende applicaties voor meerdere klanten gebouwd met Ruby zonder Rails, maar Rails maakte het hen mogelijk om Ruby als een mogelijke oplossing te zien."

Dat brengt me op de vraag die ik al had, namelijk hoeveel heb je aan Ruby zonder Rails?
"Als je geen webfrontend bouwt, heb je veel onderdelen van Rails ook niet nodig. We hebben meerdere applicaties die niet rails gebruiken maar wel active record van de database management. We hebben dat er min of meer uitgehaald, dat is vrij gemakkelijk gezien de architectuur. Het meeste werk wat we doen op Ruby-gebied is Rails-werk, maar het pure Ruby-deel groeit. We werken met grote internationale ondernemingen die Ruby zonder Rails intern ook beginnen te adopteren."

Het gevaar van een framework als Rails is dat een mode over kan waaien, en dat het na verloop van tijd legacy wordt..
"Ja precies, Ruby is een taal en die heeft veel meer overlevingskracht. Als we hierover spreken dan proberen we te benadrukken dat de echte kracht van Rails het populariseren van Ruby is. We houden ervan, maar er zijn delen van Rails die niet zo geweldig zijn. De echte kracht ervan had van doen met de kracht van Ruby als taal.
Het is heel goed om de lessen van Rails niet alleen terug te zien in veel van de bekende Java frameworks, maar om ze ook door Java en C# deels opgenomen te zien, in een wat meer dynamische aanpak. Voor mij is dat heel prettig want ik wil Java niet zien verdwijnen. Er is te veel goeds gebeurd, er is te veel infrastructuur, gemeenschap, te veel geschiedenis. Java maakt ook de weg vrij om anders te denken dan voordat Java bestond. Denk aan de tijd dat enterprise development gedomineerd werd door Micrsosoft en IBM en gesloten technologie."

Gehtland: 'Ruby is een taal en die heeft veel meer overlevingskracht'

Gehtland is zeer enthousiast over JRuby:
"JRuby is een van de mooiste dingen die ik ooit gezien heb. Charles en Thomas hebben geweldig werk op dit platform verricht (zie Java Magazine 1/2007, DdM). Nu ben ik een spreker net als een aantal van mijn collega's en mensen willen graag dat je bij presentaties controversiële dingen zegt en dingen die stof doen opwaaien. Het verhaal van JRuby is het omgekeerde daarvan.

Het is het mooie van Ruby met al de kracht van de JVM. In staat zijn terug te vallen op die enorme bibliotheek van code, die geweldige gemeenschap en wat in mijn mening de beste runtime is die we hebben. Als ik die dingen kan gebruiken, waarom zou ik dat niet doen? Het probleem daarbij is dat alle Ruby programmeurs niet willen horen dat er behoefte is aan Java of aan een JVM, en dat alle Javaprogrammeurs niet willen horen dat er behoefte aan een andere taal dan Java bestaat. Wij zouden graag zien dat Ruby Java een niveau naar beneden zou drukken. Het is leuker om code in Ruby te schrijven. Als ze daarin slagen dan wordt ons leven als ontwikkelaar oneindig veel interessanter. We hebben heel hoge verwachtingen van JRuby."

Ook al gaan ze nooit in Ruby programmeren, Gehtland vindt dat Javaprogrammeurs ook dan lessen uit Ruby kunnen trekken:
"Ik ben min of meer gevormd als Javaontwikkelaar op het hoogtepunt van de EJB-ontwikkeling. In die tijd gebruikte de industrie EJB overal voor. Ik las Bruce Tates' *Better faster lighter Java*-boek en ik was een heel vroege adapter van Spring. Het idee was voor ons op

dat moment: terugvechten tegen hypercomplexiteit. Het grootste probleem was voor ons dat we meer tijd spendeerden aan het schrijven van Javacode om met het Java framework samen te werken, dan om werkelijke businessproblemen op te lossen. Als meer dan de helft van je code zich bezig houdt met je framework dan heb je een probleem. Zelfs wanneer het grootste gedeelte van je EJB-code voor je gegenereerd wordt, dan is het er nog steeds. Het zit overal en maakt je code erg onoverzichtelijk. Als je een simpele web applicatie schrijft heb je geen EJB nodig. Wij kwamen applicaties tegen met miljoenen regels code voor een paar webpagina's."

Gehtland vindt nu dat de ontwikkeling te ver de andere kant op is gegaan:
"Mensen denken te snel 'we schrijven te veel boiler plate Java code, we doen het nu allemaal met configuratie'. Nu zit de helft van je code in een XML-document. En om eerlijk te zijn, de problemen die daaruit voortkomen zijn bijna net zo groot als vroeger met EJB's. Java kent statische typing en statische compiling. Het is heel moeilijk om veranderingen aan te brengen in een complex framewerk als je eenmaal alles met elkaar verbonden hebt. Het framewerk verwacht bepaalde dingen en als je veranderingen aanbrengt dan kan dat vrij ver grote rimpelingen veroorzaken in zo'n applicatie. Spring lost dat probleem op door een laag tussen de verschillende stukken van je code aan te brengen. Die laag is XML. Maar ongelukkigerwijs krijg je nu het slechtste van beide werelden: de taal waarin je de meeste programmeert levert eigenlijk geen code op. Het is niet te compilen of te verifiëren, maar aan de andere kant staat een bij het compilen statisch controlerende taal.

Je kunt dus in de situatie terecht komen waarin de compiler zegt dat alles prima is, maar de applicatie in runtime toch crasht omdat de XML verkeerd geschreven is. Wat is het nut van een statisch gecompileerde taal op dat punt? Het voordeel ervan zou moeten zijn dat je van te voren verteld wordt dat systeem niet zal werken. Je vindt zeer zelden een set unittests die veelomvattend genoeg is om dit soort problemen op te vangen. Nu heb je een complete suite van infrastructuur die als taak heeft je XML te verifiëren. Dat klinkt vrij zinloos. Op conferenties vraag ik wel wie zich een Java developer noemt: 100%. Vervolgens vraag ik wie er meer tijd besteed aan het schijven van XML dan aan Java? Een grote meerderheid. Ik wil geen XML-ontwikkelaar zijn. Ik heb er nooit van gehouden. Het is geweldig voor data-uitwisseling maar het is niet bedoeld voor mensen, het is zinloos. De kracht van Rails als framework is dat het demonstreert dat je een complex framework kunt hebben zonder een regel XML."

Leer een nieuwe taal

"Een belangrijke les uit Ruby is: leer een nieuwe taal. Veel Javaontwikkelaars van nu hebben geen enkel idee hoe het was vóór Java. Java is de manier waarop ze dingen altijd gedaan hebben en daarom weten niet wat het zou betekenen om een probleem vanuit een ander perspectief te benaderen. Alleen al omdat na het leren van een andere taal kritisch leert te kijken naar je Java-architectuur heeft het zin."

Ruby is ook handig om dingen in Java te demonstreren volgens Gehtland:
"We zijn al een tijd voorstanders van AOP in Java en in AspectJ. Maar het is moeilijk om die boodschap over te brengen, mensen zien het als onnodig complex. Als je laat zien wat er mogelijk is in Ruby als deel van de syntax, heropen dan een klasse tijdens runtime en definieer een heel nieuwe set van API's op een instantie van een klasse, voor bepaalde doeleinden, security of transactionele doeleinden. Je kunt dus hele goed iets demonstreren in Ruby, om de voordelen van dingen als AOP in Java te laten zien."

Ruby-projecten zijn echt anders dan Java-projecten:
"Ruby-projecten neigen ertoe een ander architecturale aanpak te vertonen dan Javaprojecten. Javaprojecten hebben heel grote klassen met veel kleine interfaces. Die klassen zijn vaak vervuild met een hoop redundante en soms zinloze code. Getters en setters die willekeurige dingen doen die ze niet zouden moeten doen, omdat ze ingekapseld zijn in een JAR file en vergeten worden. Ze maken zeer sterk gebruik van public en private, maar veel dingen blijven verborgen. Aan de andere kant neigen Ruby-projecten ertoe de meeste dingen te exposen, en worden ander samengesteld. Er zijn veel kleine modules geïntegreerd in grotere klassen, maar de feitelijke code zit in heel kleine samengestelde modules die gemakkelijk leesbaar zijn. Het doorlezen van Ruby-code is veel gemakkelijker, ook al is het meestal minder goed gedocumenteerd, want het is meer zelf documenterend. Dat heeft minder te doen met de syntax en meer met welke architectuur die gemeenschap voor zijn projecten kiest. Natuurlijk zijn het generalisaties, er zijn projecten met geweldige samengestelde Java projecten en verschrikkelijke Ruby-projecten. Rails is een goed voorbeeld; de manier waarop de architectuur is gekozen maakt het heel gemakkelijk om het te lezen en te zien wat het doet. Dat zijn ook de dingen die ik in mij lezing benadruk: wat zijn de voordelen van Ruby en hoe kun je die ook in Java gebruiken. Het is jammer dat ons boek zoveel over Rails gaat, want wij wilden het voor de helft over Ruby en voor de helft over Rails laten gaan, maar de uitgever besliste anders."

Gehtland wordt nogal eens met twijfels geconfronteerd met betrekking tot de mogelijkheden voor het gebruik van Ruby:
"We zitten vaak in panels en dan wordt je regelmatig met de vraag geconfronteerd: 'Ruby lijkt geweldig en krachtig, maar ik werk met een hoop idioten. Wat zou je in die situatie doen?'
Dave's antwoord is meestal: 'waarom zou je dat willen doen, met idioten werken?' Maar de realiteit is dat de meeste mensen een baan hebben en die nodig hebben.

De vraag die ik vaak krijg is: 'Ruby lijkt echt geweldig, maar het lijkt ook alsof je een soort wizzard moet zijn om het te gebruiken. Wat gebeurt er wanneer we consultants binnen halen en geweldige Rubycode schrijven en dan weer weggaan. Hoe onderhouden we die code?'
Het eerste probleem bij die vraag is dat de vragensteller de meeste ontwikkelaars onderschat. Je hoeft geen tovenaar te zijn om Ruby-code te schrijven. Hij onderschat ook de echte wizzards: er zijn mensen die code schrijven die ik nooit zal begrijpen. De geweldig infrastructuur bij IBM bijvoorbeeld, de Windows foundation code. Ik begrijp die code niet, ik

weet niet hoe die mensen het deden. Ik zou niet willen dat díe mensen binnenkwamen en code voor me schreven. Ruby is niet meer dan een syntax en het heeft een paar andere filosofieën over hoe je code schrijft, maar uiteindelijk is het gewoon OO-programmeren.

Als je dat kunt, kun je het oppikken. Of je daar de tijd voor vindt in je dagelijks werk is een ander vraag. De meeste mensen hebben een aantal taken, en meestal vormen die een te grote werkbelasting. Ze werken steeds aan de grens van hun kunnen, maken te veel uren en wanneer het project eindigt is er weer een ander project dat alweer achterloopt. Het opvolgen van die geweldige adviezen, 'leer ieder jaar een nieuwe taal', dat soort dingen is eigenlijk alleen weggelegd voor dat kwart van de ontwikkelaars dat er tijd voor heeft. Ook dat is er reden waarom we JRuby zo waarderen. In een project dat verder voor 80 of 90 % uit Java bestaat kun je een beetje Ruby-code integreren om een bepaald probleem op te lossen. Dan moet iedereen het min of meer absorberen. Je zult vaak zien dat wanneer er eenmaal een beetje Ruby in zo'n project zit, mensen heel erg hun best doen om er nog meer in te krijgen. Maar ik geef toe dat het moeilijk is om er tijd voor vrij te maken. Het beste advies wat ik mensen kan geven die in Ruby geïnteresseerd zijn maar weinig tijd hebben: ga naar conferenties en pik daar Ruby-sessies mee. Desnoods kijk je naar Groovy of naar Frails waar je dezelfde sfeer opsnuift maar je baas kunt vertellen dat je nog steeds Java doet. Zo krijg je niet alleen een idee waarom het belangrijk is, maar voel je ook het enthousiasme. Ruby heeft harten en hoofden veroverd terwijl andere talen daartoe niet in staat waren. Er zij vele mensen die Python gebruiken maar slechts weinigen die het evangeliseren. Ruby is een geweldige taal die mensen enthousiast maakt en dat moet je beleven."

Patches Patches Patches Patches Patches Patches Patches P

Artikelen over onderwerpen als software-ontwikkeling, Java, UML, eXteme Programming en nog veel meer vindt u in het Online Archief van Array Publications. Vaktijdschriften als Software Release, Java Magazine, Database Magazine en ons Oracle vakblad Optimize hebben hun artikelenarchief online gezet. Dankzij de heldere zoekstructuur vindt u snel wat u zoekt op www.release.nl.

Bedrijfssoftware niet op waarde geschat

Hoger rendement door toekomstvaste software-investeringen en adequate taxatie Bedrijven in de hele wereld kunnen veel meer rendement halen uit software. Ze dienen dan toekomstvaste software-investeringen te doen in plaats van deze te beschouwen als een verplichte, tijdelijke uitgave. Daarnaast moeten organisaties software beoordelen op bedrijfsresultaat en niet alleen op het technisch functioneren. Hierdoor kunnen ze de waarde van hun IT-systemen vergroten. Dat althans is de conclusie van een gezamenlijk onderzoek van Micro Focus, leverancier van applicatiebeheer- en moderniseringsoplossingen en INSEAD, business school and research instituut. Professor Soumitra Dutta, voorzitter Business & Technology bij INSEAD, gaat nog een stap verder in het bijbehorende onderzoeksrapport. Hij spoort leidinggevenden aan om de directie te waarschuwen voor de verkeerde manier waarop hun bedrijf software beheert en taxeert. In zijn rapport benadrukt Dutta dat software een van de belangrijkste bedrijfseigendommen is. Daarbij is software cruciaal voor een succesvolle bedrijfsstrategie en een middel om de totale bedrijfswaarde te verhogen. Maar daar stuiten organisaties volgens Dutta op een probleem; er is namelijk een schrijnend tekort aan tools om de waarde van bedrijfssoftware adequaat te meten.

Voor het onderzoek ondervroeg INSEAD 250 CIO's en CFO's die werkzaam zijn bij bedrijven met een jaarlijkse omzet van meer dan 100 miljoen dollar, in Frankrijk, Duitsland, Italië, Groot-Brittannië en de Verenigde Staten. Het onderzoek wees ook uit dat wereldwijd toonaangevende bedrijven de omvang en waarde van IT-middelen vaak negeren in vergelijking met andere, regelmatig geëvalueerde bedrijfsmiddelen, zoals de omzet, eigendommen, kennis en het merk. Dit is volgens INSEAD opmerkelijk gezien de ruim één biljoen dollar die in 2006 wereldwijd uitgegeven is aan IT (Bron: IDC januari 2007; 1,16 biljoen dollar IT-uitgaven in 2006).

"Organisaties beschouwen software vaak niet als een middel om de bedrijfswaarde te vergroten. Integendeel, ze zien het als een bedrijfsmiddel dat zo min mogelijk moet kosten. Hier moet verandering in komen", zegt professor Dutta. "CIO's en CFO's moeten de raad van bestuur op de hoogte brengen van de waarde van software en deze geregeld meten. Software vormt namelijk een onzichtbare basis voor bedrijfsprocessen. Door de waarde van software adequaat te meten, kunnen bedrijven de juiste beslissingen nemen over veranderingen aangaande de financiële balans, fusies en overnames, onderhandelingen over joint ventures, vergunningen en franchises, en relaties met investeerders."

Volgens Dutta is 'Conjoint Analysis' dé manier om de waarde van bedrijfssoftware adequaat te meten: "Bij een traditionele conjoint analysis-applicatie worden verschillende productattributen uitgewisseld. Deze methode is ook toepasbaar op software. Hiervoor dienen mensen te handelen in verschillende bedrijfsresultaten die samenhangen met specifieke software. Wanneer organisaties de software in IT-systemen analyseren vanuit concrete bedrijfsresultaten, kunnen ze de financiële waarde van belangrijke software berekenen."

Professor Dutta komt met een duidelijk stappenplan om de waarde van IT adequaat in te schatten en niet langer alleen te beoordelen op het technisch functioneren. Daarbij biedt hij CIO's handvatten om een grotere strategische rol te spelen en de directie te overtuigen van de verborgen waarde van software", zegt Kelly. "Bedrijfssoftware is uiterst waardevol. Die waarde is meestal door de jaren heen opgebouwd in verschillende bedrijfsprocessen. Nu hebben CIO's en CFO's voor het eerst de kans om de daadwerkelijke waarde van dit laatste ondergewaardeerde bedrijfsmiddel onder de aandacht te brengen van de directie."