Bericht uit 

Er zijn van die momenten dat je aan jezelf gaat twijfelen. Je denkt een gewone, voor de hand liggende mening te hebben maar iedereen blijkt er anders tegenaan te kijken. De 'ben ik nou gek?'- momenten. Ook is er de collectieve variant, de 'zijn wij nou gek?' - variant. Die is nog vervelender, want het gemeenschappelijk gedachtegoed is onderling versterkt; het kan toch niet zijn dat we er allemáál naast zaten? Ons recente 'zijn wij nou gek?' - moment kregen we na vele audits aan service oriented systemen, gebouwd in Java of .NET. (durft u nog tegen de stroom in? – een niet SOA-systeem?).

Zijn wij nou gek?

Het begon de laatste tijd al op te vallen dat we vrijwel altijd tot dezelfde conclusies kwamen. Wij komen heel veel lelijke code tegen. De code is keurig opgemaakt, aan commentaar is ook wel gewerkt en toch is het hele lelijke code. Wanneer je dat een paar keer achter elkaar hebt ondervonden, dan haal je er een collega bij. Na een paar diagonaal doorgelezen classes zegt die collega ook "bah, lelijk!". Je voelt jezelf gesterkt, er is nog niks aan de hand. Maar dan begint het om te slaan; je komt projecten tegen waar de lelijkheid verplicht is gesteld in richtlijnen en architecturen. Projectleden blijken trots te zijn op de lelijke code; ze worden boos om onze conclusies. Zijn wij nou gek? Tijd om de koppen bij elkaar te steken, tijd om te kijken wat er aan de hand is.

Mijn persoonlijke conclusie is de volgende: we jubelen wel allemaal om SOA, maar we hebben geen clou van hoe je dat bouwt. Een fictief maar realistisch voorbeeld: de 'persoonsservice'. De persoonsservice is de service die verantwoordelijk is voor alle persoonsgegevens. Er wordt een (data-) model gemaakt en er wordt beoordeeld welke gegevens allemaal rondom personen gebruikt worden. Voor het beheer van al die gegevens worden methodes gedefinieerd. Een typische methode zou 'update adres' kunnen zijn. De code van deze service zou ongeveer de volgende dingen doen: 1) kijk of de service-call correct is (geen leeg XML-bericht?, alle verplichte gegevens aanwezig?) 2) haal het persoon-ID en het nieuwe adres uit het bericht 3) zoek de betreffende persoon op en 4) wijzig het adres 5) stel een retourbericht op. Herkenbaar? Helaas wel; ik zie dezelfde treurige volgorde methode voor methode al voor me.

Al jaren geleden hebben we geleerd dat systeemintegratie niet op databaseniveau geregeld moet zijn. Zodra je daar aan begint, gebruik je data zonder de bijbehorende business logica en moet je die logica dus nog een keertje programmeren in het te koppelen systeem. Integratie op database-niveau is niet beheerbaar en foutgevoelig. We moeten integreren op applicatieniveau. Nu maken we services (want dat is zo goed te hergebruiken) maar vergeten nog steeds de business logica. Bovenstaande service-methode had 'verhuis' moeten heten. Zodra een service 'verhuis' als methode aanbiedt, in plaats van 'update adres', is er plaats voor business logica. Wat moet er allemaal gebeuren wanneer iemand verhuist? Moeten we een welkomstkaartje sturen? Moet hij / zij door een ander filiaal bediend worden? Moet de verzekeringspremie aangepast worden omdat de persoon in een andere buurt is gaan wonen? Dáár moet de code achter een service over gaan! Dát is de enige manier om services herbruikbaar te maken. Als je al per se services wilt bouwen, bouw dan business-services; niemand in een bedrijf is verantwoordelijk voor het 'updaten van een adres', er moet iemand verantwoordelijk zijn voor het correct afhandelen van verhuisberichten. Het lijkt misschien een woordspelletje maar 'update adres' levert vanzelf foeilelijke code, hoe goed de programmeurs ook zijn. Het probleem ontstaat bij ontwerpers en architecten. De mindset klopt niet.

Nee, wij zijn niet gek!

Daan Kalmeijer is docent consultant bij CIBIT-adviseurs | opleiders (e-mail: daan@cibit.nl).