Het moest er weer eens keer van komen. Tijdens een recente safari door de woeste wildernis van enterprise software development ben ik onlangs in onzachte aanraking gekomen met een exemplaar van een van de meest onderschatte, maar tegelijk een van de meest gevaarlijke diersoorten in deze jungle: de Architectus Non-Programmicus.

De Architectus Non-Programmicus

Sinds enige maanden ben ik de coach van een afdeling voor enterprise software development van een grote onderneming. Tijdens een recente sessie met ontwikkelaars stuitte ik op een hiaat in hun applicatiearchitectuur. Alhoewel ik er niet veel trek in had, zat er niets anders op dan het gevonden hiaat te bespreken met de architect, die verantwoordelijk is voor het opstellen van de referentiearchitectuur van mijn klant. Hiervoor heeft de architect een gewichtig document samengesteld waarin alle verschillende lagen en typen componenten zijn gedefinieerd. Alle? Nou ja, bijna alle.

De applicaties die gebruik maken van de services van componenten zijn natuurlijk thin clients – dat snapt toch iedereen. De schermen van de applicaties zijn formulieren in InfoPath en roepen de services aan – in BizTalk. Deze services zijn echter geïmplementeerd door allerlei legacy te ontsluiten – via MQ. En daar zit 'm de crux. De berichten die heen en weer worden gestuurd vanuit de formulieren naar de legacy en terug bevatten allerlei eigenaardigheden, die de legacy met zich meebrengt. Zo wordt bijvoorbeeld een kommagescheiden string meegegeven die de bevoegdheden van een tussenpersoon kenmerkt – niet bepaald een hoogtepunt in de objectoriëntatie.

Omdat services rechtstreeks worden aangeroepen door de formulieren, bevatten de applicaties op diverse plekken code om deze eigenaardigheden om te zetten naar voor de applicatie zinvolle bedrijfsklassen – een tussenpersoon met diverse bevoegdheden. Bovendien bevatten de applicatie op al deze plekken vergelijkbare code om verbinding te maken met de achterliggende services. Ook geen kenmerk van optimaal hergebruik.

Een oplossing was snel gevonden. Creëer eenduidige toegang tot de services, en plaats een façade tussen de formulieren en deze toegang. Deze façade, en alleen deze façade, is verantwoordelijk voor de transformatie van berichten naar voor de applicatie begrijpelijke bedrijfsklassen. Hergebruik treedt nu zelfs al op in deze applicatiegerichte nieuwe bedrijfslaag.

Al programmerend ging er een lichtje bij mij branden. Al in twee eerdere projecten was ik tegen hetzelfde hiaat aangelopen. In ieder van deze projecten werd het hiaat veroorzaakt door hetzelfde fenomeen: de architectuur was opgesteld door een architect die niet programmeert. Gewapend met PowerPoint en Visio ziet deze diersoort schijnbaar een belangrijk patroon over het hoofd.

Maar ja, hoe overtuig je een Architectus Non-Programmicus – God's plaatsvervanger op aarde – ervan dat hij iets over het hoofd ziet? Niet op basis van gezonde argumenten, kan ik u verzekeren. Ik ken slechts een remedie: vakliteratuur. In het geval van enterprise software development is er maar één boek dat genoeg gewicht in de schaal legt. *Enterprise Integration Patterns* van Gregor Hohpe en Bobby Woolf. *Enterprise Integration Patterns* beschrijft heel leesbaar een boeiende collectie patronen voor het uitwisselen van berichten in enterprise software development. Sommige liggen voor de hand, andere bevestigen vooral uw werk. Mijn patroon is snel gevonden. Het is een Service Gateway. Mocht u in uw natuurlijke habitat te maken krijgen met een Architectus Non-Programmicus dan heeft u het ideale verjaardagscadeau in handen.

Sander Hoogendoorn
(www.sanderhoogendoorn.com)

Waardering ★★★★½

Auteurs: Gregor Hohpe & Bobby Woolf
Titel: *Enterprise Integration Patterns*
Uitgever: Addison-Wesley