De Spiegel van de Software Architect

Erik Philippus
Improvemen**T**
erik.philippus@improvement-services.nl

Uiteraard zal een architect oog moeten hebben voor (on-)gewenste gedragspatronen 'op de ontwikkelvloer', maar dat laat onverlet dat een architect ook in staat moet zijn hand in eigen boezem te steken. Het kunnen reflecteren op de invloed van het eigen handelen, is kenmerkend voor het volwassen worden van de software architect. Ter kennismaking en aanmoediging van dit proces volgen hier een aantal gedragspatronen 'ter leering ende vermaek'.

De karakterisering 'Architectus Reloadus' wordt genoemd door Martin Fowler in zijn artikel "Who Needs an Architect?", en is gebaseerd op een fragment uit de film "Matrix Reloaded". De Architect tracht Neo te bewegen tot een complete reset ('reloading') van de Matrix, maar uiteindelijk volgt Neo zijn liefde voor Trinity, en bewijst daarmee over een eigen wil te beschikken.

Deze houding van de Architect is een typisch gevolg van het aloude denken in hiërarchieën, dat we in één of andere vorm vaak zien terugkomen in allerlei methodieken op het gebied van software ontwikkeling. Hoewel er geen algemeen geldend recept is voor systeemontwikkeling, is er meestal wel overeenstemming over een zekere rangorde, min of meer gekoppeld aan abstractie niveau's. Gewoonlijk wordt een systeem architect bovenaan de hiërarchie geplaatst, gevolgd door architecten die zich richten op deelaspecten, dan komen de specialisten en domein experts die zich modelleren bezighouden, en ten slotte programmeurs die zich uitsluitend richten op de implementatie.

Als de ontwerper in deze hiërarchie zich bezighoudt met meta-communicatie over het te bouwen systeem, dan zal de systeem architect zich moeten bedienen van 'meta-meta-communicatie' – hetgeen aardig overeenkomt met de hierboven beschreven basishouding van de Architect uit de Matrix.

Wanneer een architect het gedragspatroon vertoont dat lijkt op dat van de 'Architectus Reloadus', zal dat verstrekkende gevolgen hebben voor de samenwerking en werksfeer in het ontwikkelteam. Zo zal een spontane, creatieve bijdrage aan het tot stand komen van het product van teamleden een uitzondering zijn.

De uiteindelijke vormgeving en functionaliteit van het product hangt grotendeels af van de denkbeelden van de dominante architect in zijn ivoren toren.

Daarbij is het maar zeer de vraag in hoeverre een dergelijke architect openstaat voor geluiden uit de hoek van toekomstige gebruikers. Ten slotte heeft de 'Architectus Reloadus' een tamelijk neerbuigende visie op de relatie tussen architectuur en maatschappij - de eerste versie van de Matrix was volkomen perfect, en om die reden niet geschikt voor de mensheid:

"The first matrix I designed was quite naturally perfect, it was a work of art, flawless, sublime. A triumph equaled only by its monumental failure. The inevitability of its doom is as apparent to me now as a consequence of the imperfection inherent in every human being, thus I redesigned it based on your history to more accurately reflect the varying grotesqueries of your nature. However, I was again frustrated by failure. I have since come to understand that the answer eluded me because it required a lesser mind, or perhaps a mind less bound by the parameters of perfection."

In zekere zin een bijzonder expliciete omschrijving van 'de menselijke maat' – maar tegelijk een uiting van hoogmoed die geen recht doet aan de bedoeling van dit artikel. Gelukkig geeft Martin Fowler ook een indicatie van de gewenste tegenhanger; de 'Architectus Oryzus', gemodelleerd naar zijn collega architect Dave Rice (Oryza is de plantensoort waartoe rijst behoort).

Kenmerkend voor architecten met deze signatuur is dat zij zich niet verschuilen in een ivoren toren, maar zich zeer bewust zijn van wat er werkelijk omgaat in een project. Zij bemoeien zich actief en oplossingsgericht met belangrijke aandachtspunten, daarmee voorkomend dat ze uitgroeien tot grote problemen. Een dergelijke architect is gericht op intensieve samenwerking, en treedt op als een mentor die het ontwikkelteam inspireert tot verdere ontplooiing. Er zal een sfeer van openheid en creativiteit ontstaan, waarin aandacht voor de kwaliteit van het ontwikkelproces èn het geleverde product een vanzelfsprekendheid wordt.

In vergelijking met de aanpak van de eenzame en dictatoriale beslissingsnemer, verschaft een dergelijke opstelling de architect veel meer krediet en armslag. Martin Fowler komt daarop tot de opmerkelijke vuistregel dat 'de waarde van een architect omgekeerd evenredig is met het aantal beslissingen dat hij of zij neemt'.

De essentie ligt waarschijnlijk echter in de wijze waarop belangrijke architecturele beslissingen tot stand komen. Is er sprake van betrokkenheid bij de gebruikers van de architectuur, en een gevoel van gezamenlijk gedragen verantwoordelijkheid voor het product? Een dergelijke bezieling is in menig ontwikkelteam soms ver te zoeken…

Een in technisch opzicht uitmuntende architectuur is geenszins een waterdichte garantie voor een moeiteloos en voorspelbaar ontwerp- en implementatie traject. Samenwerking in een ontwikkelteam gaat niet altijd van een leien dakje, zoals het gedragpatroon "Following Orders" ons laat zien.

Op zich is het natuurlijk een goede zaak wanneer ieder teamlid in grote lijnen weet wat zijn taken en verantwoordelijkheden zijn. De focus op eigen werk kan echter zover doorschieten, dat het contra-productief wordt. Dit anti-pattern ligt op de loer wanneer software ontwikkelaars zich niet meer beschouwen als leden van een team, en vrijwel alle energie exclusief besteden aan het succesvol volbrengen van de opgedragen werkzaamheden.

Dit patroon kan worden versterkt door het invoeren van strikt eigenaarschap van broncode, waarbij gedeeltes van de code langdurig worden beheerd door slechts één ontwikkelaar – die dus op den duur de samenhang met de overige software onderdelen uit het oog gaat verliezen. Verzoeken van andere teamleden voor hulp bij problemen in hun code worden afgewimpeld, of hoogstens wordt het probleem van de ander op een 'quick and dirty' wijze opgelost.

Niet alleen de kwaliteit van het gezamenlijke product heeft hier uiteindelijk onder te leiden, ook de sfeer van samenwerking zal achteruit gaan door het wegvallen van een gemeenschappelijk doel.

Een sprekend voorbeeld van dit patroon is het z.g. 'Schedule Chicken' gedrag dat de kop kan opsteken in teams die in subteams afzonderlijke software componenten ontwikkelen. Ook hierbij gaat de exclusieve focus op het voltooien van de eigen component ten koste van het gemeenschappelijke doel. In plaats van ontwikkelaars van een component in onverwachte moeilijkheden de helpende hand te bieden, zijn de overige leden van het ontwikkelteam opgelucht dat ze niet op het kritische pad zitten met hun eigen component.

Een variant van dit gedragspatroon kan optreden wanneer er niet één, maar verschillende subteams achterlopen op schema, maar dat geen enkel subteam de problemen rapporteert - in de verwachting dat het andere subteam vroeg of laat tegen de lamp loopt en de schuld krijgt van de vertraging.

Een ontwikkelteam dat zich niet meer verantwoordelijk voelt voor het eindresultaat, kan op deze wijze de voordelen van component-based development teniet doen. Eigenlijk zien we iets dergelijks ook in breder verband terugkomen, als we kijken naar de moeite die het in de software industrieklaarblijkelijk kost om te komen tot een professionele bibliotheek van hoogwaardige software componenten.

Het delen van impliciete kennis wordt bemoeilijkt door de neiging om alles en iedereen buiten te sluiten, met uitzondering van de meest zichtbare baas of klant. De belangrijkste drijfveer hiervoor is het verkrijgen van beloning van die baas of klant voor het succesvol afronden van de opgedragen taak. Deze drijfveer wint nog aan kracht wanneer het verder kijken dan alleen de oplossing van de eigen directe problemen minder waardering krijgt (of lijkt te krijgen). Zeker als er sprake is van geografische of 'politieke' opdeling in een organisatie, kan dit gedragspatroon de kop opsteken.

Hoewel er verschillende oplossingsrichtingen te bedenken zijn, is de basis steeds het veranderen van het verwachtingspatroon met betrekking tot het delen van kennis en ervaring. Een doorgewinterde Schedule Chicken denkt misschien slim te zijn, maar gaat voorbij aan het grote belang van het delen van kennis en ervaring: het zonder verborgen agenda anderen helpen bij het oplossen van problemen herstelt de natuurlijke balans tussen geven en nemen.

Dit is in een notendop het geheim van teambuilding. Het kan even duren voordat het bewustzijn doorbreekt, maar uiteindelijk vindt de mens meer voldoening en rijkdom in het geven dan in het nemen.

De training en vorming van software ontwikkelaars speelt hierbij een rol, maar ook visie en werkwijze van de organisatie ten aanzien van beloning en promotie heeft een belangrijke invloed. Een architect die beducht is voor de hier geschetste gedragspatronen ten slotte, kan een hoop onheil voorkomen!

Concrete maatregelen die de architect kan nemen om dit gedragspatroon om te buigen of te voorkomen, zijn:

- Gebruik de architectuur als duidelijk aanwezige kapstok voor het positioneren van afzonderlijke bouwstenen, en relateer individuele werkzaamheden steeds aan de overkoepelende architectuur;
- Verschaf alle teamleden voldoende mogelijkheden om kennis te nemen van de principes en mechanismen waarop de architectuur is gebaseerd. Betrek ontwikkelaars actief bij het onderhouden van de architectuur;
- Review van ontwerp en code maakt structureel deel uit van het ontwikkelproces. Vooral peer-reviews waarbij ook ontwikkelaars van buiten het eigen team worden betrokken, zijn zinvol.
- Beloon teamleden voor samenwerking, en het inzetten van hun kennis en ervaring voor het behalen van een gezamenlijk resultaat.
- Het vermogen tot samenwerking en communicatie van kandidaten dient een duidelijk zichtbare rol te spelen bij aannames en promoties. Betrek het team hierbij, en zorg dat teamleden op sleutelposities aanwezig zijn bij interviews.

Enigszins gerelateerd aan het gedragspatroon van de hiervoor omschreven ‘Architectus Reloadus’, betreft de architect die zich langere tijd op een zodanig abstractie niveau beweegt, dat het contact met de realiteit geleidelijk verloren gaat.

In een ernstige vorm kan dit leiden tot het gedragspatroon van de ‘Wereldvreemde Componist’. Deze typering is gebaseerd op een ervaring van een cellist van het Concertgebouw Orkest bij het spelen van een modern symfonisch stuk, waarbij bleek dat de cellopartij onmogelijk correct gespeeld kan worden op dit instrument.

De tegelijkertijd te spelen noten lagen onmogelijk ver uit elkaar op de hals van de cello - althans bij normale afmetingen van de hand van de cellist. Met zijn partituur gaf de componist in feite blijk van onvoldoende kennis van de technische mogelijkheden van het instrument.

De cellist in kwestie maakte overigens nog wel de relativerende opmerking dat het een dusdanige kakefonie was, dat het totaal niet opviel dat hij maar een deel van de noten 'raak' had.

Een architect zal voldoende voeling moeten hebben en houden met de praktijk. De meerwaarde van een architect heeft zijn oorsprong in de eigen ervaring. Alleen door zelf met beide benen in grote projecten gestaan te hebben en alle aspecten van software ontwikkeling aan den lijve te hebben meegemaakt, kan iemand met recht aanspraak maken op de titel van architect. Juist de praktijkervaring leert de architect onderscheid maken tussen het probleemdomein en het oplossingsdomein, en de juiste 'vertalingen' of transformaties tussen beide domeinen te maken. Soms wordt het gebrek aan praktijkervaring gemaskeerd door een arrogante houding, zoals blijkt uit een ervaring van Grady Booch:

"I was called in to a project in crisis, and I began my investigation by spending some time with the architect in order to diagnose the problem. We spent a few hours talking about certain key abstractions in the domain, and then I asked him if he would take one of these abstractions, that he had outlined, and capture it in a C++ header file. He refused, with words to the effect "No! Damn it, I'm an architect, not a coder!" It took me about three picoseconds to realize why this project was in crisis. When developers say such things, it is usually a sign of arrogance or ineptness. Unfortunately, this 'architect' was both."

De remedie voor het gedragspatroon 'De Wereldvreemde Componist' wordt gegeven door Gerrit Muller, die in een stelling behorende bij zijn proefschrift duidelijk aangeeft dat het voor het functioneren van een systeemarchitect essentieel is dat deze beschikt over voldoende diepgang in het vakgebied, maar dat het minstens zo belangrijk is dat het vakmanschap ook actief wordt onderhouden.

Naast de hierboven aangegeven relatie tussen rangordes en (mis-)communicatie, heeft een dergelijke hiërarchische of zelfs arrogante zienswijze nog een ander belangrijk nadeel. Wanneer architectuur wordt gedefinieerd als het 'hoogste abstractie niveau' van een systeem, dan wordt hiermee meestal gedoeld op de wijze waarop ontwerpers en ontwikkelaars tegen het systeem aankijken. Dat er ook nog andere belanghebbenden zijn, ieder met hun eigen zienswijze op software architectuur, blijft dan secundair.

Meestal is een software architect in dat geval voornamelijk bezig met het in een technische context vastleggen van het gedeelde begrip over het te bouwen systeem, ten behoeve van de mensen lager in de hiërarchie. Software architectuur versmalt dan al snel tot een document dat slechts de consensus onder ontwikkelaars systeem weergeeft.

Wanneer een ontwikkelteam werkt op basis van een conservatieve architectuur van een 'Wereldvreemde Componist', dan bestaat de mogelijkheid dat de huidige stand van de techniek het afgeleverde product bij de introductie al heeft ingehaald. Hierdoor worden mogelijkheden van (sommige) gebruikers ingeperkt, en zal het product niet aan bepaalde (toekomstige) wensen tegemoet kunnen komen.

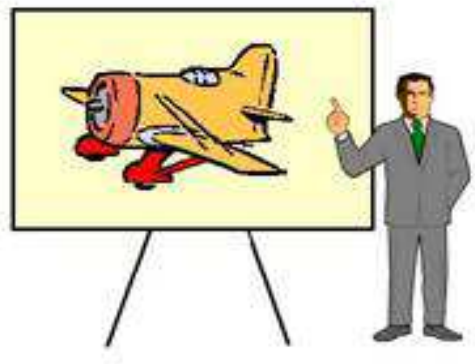
Als tegenhanger van de 'Wereldvreemde Componist' kan overigens de 'Trendsurfer' genoemd worden. Een architect die behept is met dit gedragspatroon sluit zich niet af, maar laat zich volledig leiden door de laatste stand van de techniek. Uiteraard is het volgen van relevante marktontwikkelingen noodzakelijk om ook op lange termijn succes te hebben, maar een organisatie betaalt een hoge prijs voor een steeds van richting veranderende architectuur – vooral als deze veranderingen plaatsvinden aan het eind van de release cyclus.

Meestal is het optreden van dit gedragspatroon een teken dat er geen breed gedragen visie en fundamentele overeenstemming is bij alle onderdelen van een ontwikkelorganisatie over de roadmap voor het product, daarmee voedingsbodem verschaffend aan paniek reacties en korte termijn handelen.

Per definitie zal een architect die dwangmatig de nieuwste technische snufjes wil verwerken in het ontwerp, gebruik moeten maken van nog niet bewezen technologie, daarmee de kans lopend dat het product allerlei kinderziektes zal vertonen. Iets anders waar gebruikers mee geconfronteerd kunnen worden is tekortschietend lange termijn onderhoud op dergelijke software producten.

Naast te weinig oog of juist overdreven aandacht voor de technische (on)mogelijkheden, kan een architect zich ook verschuilen achter de techniek, in lijn met het hierna besproken gedragspatroon, genaamd 'Anti Gravity Module'.

Een vliegtuigfabrikant vertelt de aandeelhouders dat het nieuwe model voor 99% klaar is – het enige wat nog moet worden gebouwd is de 'Anti-Gravity Module', en dan kan de eerste testvlucht gemaakt worden.



Dit gedragspatroon treedt op als de ontwikkeling van risicovolle onderdelen van het product systematisch in de tijd naar achteren worden verschoven.

Ogenschijnlijk volgt het project redelijk de planning, maar als de deadline in zicht komt is de 'Anti-Gravity Module' nog steeds niet opgeleverd. Op papier lijkt de architectuur er adequaat uit te zien, maar de Anti-Gravity Module zal helaas nooit zo werken als gehoopt...

Een sprekend voorbeeld van dit anti-pattern is wat er gebeurde met een Request For Proposal (RFP) uitgegeven door het Amerikaanse leger in de jaren 90. Geïnteresseerde partijen moesten aangeven of ze een gedistribueerde database zouden toepassen in hun ontwerp, of gebruik zouden maken van een locale, relationele database. De opdracht wordt gegund aan een partij die geen keuze maakte, maar beide oplossingen beloofde: een magische 'software switch' zou schakelen tussen beide databases. Uiteindelijk bleek deze 'oplossing' onmogelijk te implementeren, en het project mislukte jammerlijk.

Wat hierbij vaak speelt, is een aanzienlijke druk op het ontwikkelteam om nooit eerder vertoonde opties of nieuwe features op te nemen in de software, zonder dat een deugdelijke evaluatie van de noodzakelijke technologie, kosten en tijdsinspanning (b.v. voor het testen van nieuwe combinaties van features) is gemaakt. Soms worden door verkopers ook beloftes aan klanten gedaan, die onmogelijk binnen de daarvoor beschikbare tijd door de ontwikkelaars kunnen worden geïmplementeerd. In dat geval is er sprake van de beruchte (late) Killer Feature, een onder druk van de marketing-afdeling op het laatste moment ingebouwde onvoorziene functionaliteit, die het project plan en het ritme van het ontwikkelteam grondig overhoop haalt.

Aan de basis van dit gedragspatroon ligt de op zich begrijpelijke neiging om zichzelf rijk te rekenen bij het binnenhalen van een belangrijke opdracht, of bij een dreigende overschrijding van de deadline. Hoe onbetrouwbaarder of onwaarschijnlijker de planning, des te omvangrijker de beloofde functionaliteit die wordt opgeleverd in de laatste taken.

Een variant van dit patroon staat bekend onder de naam 'Near Everest'. De top lijkt niet ver weg meer, maar dit is ogenschijnlijk zo vanwege factoren zoals oplopend zuurstoftekort, toenemende vermoeidheid, optisch bedrog, etc. Zoals elke ervaren klimmer weet, wegen ook hier de laatste loodjes ook hier het zwaarst.

Niet zelden wordt er in een dergelijke situatie druk uitgeoefend op de architect om zich te verplichten aan het laatste onderdeel van een onrealistische planning. In plaats van zich in die omstandigheden te gedragen als de spreekwoordelijke struisvogel met de kop in het zand, zal een architect die sterk in z'n schoenen staat de volgende stappen moeten nemen:

1. Maak een grondige analyse van de Anti-Gravity Module, en breng dit - voorzien van een deugdelijke, kwantitatieve onderbouwing - duidelijk en eerlijk onder ogen bij het management;

2. Haal alle veronderstellingen ten aanzien van de Anti-Gravity Module boven tafel, en maak ze concreet zodat ze realistisch getest en geëvalueerd kunnen worden;

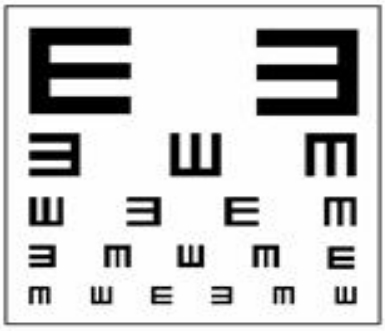
3. Stel het management voor een duidelijke keuze: als het nu duidelijk geformuleerde risico niet wordt geaccepteerd, zal het commitment van het ontwikkelteam ter discussie komen te staan:

4. En ten slotte een tip voor de sales-manager: neem een stap terug uit de sfeer van opwinding rond het binnenhalen van de grote opdracht, en stel eerlijke vragen aan de architect en het ontwikkelteam - en accepteer ook eerlijke antwoorden!

Afgezien van de sluimerende onrust die de Anti-Gravity Module in het ontwikkelteam veroorzaakt, zullen ook andere belanghebbenden het effect kunnen waarnemen: het product wordt te laat (of nooit) opgeleverd. Mogelijk is er een work-around voor de Anti-Gravity Module gevonden, of is er noodgedwongen in de functionaliteit geschrapt. Daarnaast zullen gebruikers van het software product ongetwijfeld merken dat er door alle onrust in de slotfase van de ontwikkelcyclus is bezuinigd op de kwaliteit van het software product.

Stephen Covey geeft de volgende anekdotische beschrijving van het gedragspatroon van ' De Betweter", dat we wellicht bij ongeduldige opticiens, maar ook bij sommige architecten kunnen aantreffen:

"Veronderstel eens dat u problemen heeft met uw ogen en dat u besluit om naar een opticien te gaan. De man luistert even naar uw klachten, neemt vervolgens zijn bril af en geeft hem aan u. 'Zet deze maar op,' zegt hij. 'Ik draag hem al tien jaar en hij voldoet nog steeds. Ik heb thuis nog een bril, dus u mag deze wel hebben.'



U zet hem op, maar nu ziet u nog minder. 'Dit is verschrikkelijk,' zegt u. 'Ik zie absoluut niets meer!' 'Hoe bedoelt u?' vraagt hij. 'Ik kan er anders uitstekend mee zien. Doet u eens wat meer moeite. En dat terwijl ik zo mijn best doe om u te helpen!' moppert hij."

Hoe groot is de kans dat u ooit nog eens teruggaat naar die opticien? Als het gaat om onze gezondheid, dan hebben de meeste van ons een gezond wantrouwen bij iemand die iets voorschrijft zonder een diagnose te stellen. Ik zekere zin is een architectuur te vergelijken met een recept, en we kunnen ons de vraag stellen hoe vaak een architect uitgaat van een deugdelijke diagnose alvorens te komen tot de voorgestelde architectuur. Het al eerder genoemde inlevingsvermogen van de architect speelt hierbij een hoofdrol, hetgeen in de terminologie van Covey terugkomt in het principe:

"Probeer eerst te begrijpen dan begrepen te worden."

Dit principe van empatische communicatie is het belangrijkste in de relatie tussen mensen in het algemeen, en tussen architecten en stakeholders in het bijzonder.

Niets voor niets bestaat er het veelzeggende gezegde dat je in een week tijd meer vrienden maakt door je te interesseren voor de ander, dan dat je een jaar lang bezig bent met pogingen om anderen voor jou te interesseren.

Genoemd gedragspatroon heeft tot gevolg dat de gemaakte software producten in geringe mate zijn gebaseerd op een diepgaande analyse van gebruikersbehoeftes. Dat gebruikers meestal niet gewend of in staat zijn nauwkeurige specificaties af te geven, maakt een dergelijke analyse des te noodzakelijker. Ander is de kans groot dat een gebruiker zich niet herkent in het product, met klantontevredenheid als logisch gevolg. Dit nadenken over wie die gebruiker nu eigenlijk is, staat geheel los van de technische structuur van het te bouwen systeem.

Zeker als we het hebben over producten waarvan de gebruiker het gevoel dient te hebben dat ze op de menselijke maat zijn gebaseerd, dan is het uiteraard een eerste vereiste dat de architect een zo compleet mogelijk beeld heeft van eisen, wensen en gedragingen van toekomstige gebruikers.

Ook uit puur enthousiasme kan een architect bepaalde gebruikerswensen over het hoofd zien. Zeker als de nieuwe architectuur een scala aan veelbelovende mogelijkheden biedt, kunnen basale zaken soms worden gemist.

Kenmerkend voor dit gedragpatroon genaamd 'The Missing Piece' is dat de architectuur weliswaar veel nieuwe functionaliteit mogelijk maakt voor gebruikers, maar dat er jammer genoeg enkele cruciale onderdelen missen waardoor de architectuur uiteindelijk niet goed bruikbaar blijkt te zijn.

De waarde van alle nieuwe mogelijkheden wordt op die manier dus teniet gedaan, terwijl de onderdelen die missen in feite overduidelijk en zelfs triviaal kunnen zijn. Op zich nog wel te herstellen, maar kostbare tijd en energie is ondertussen verloren gegaan.

Een andere vorm van dit gedragspatroon kan optreden in een architecten team, waarbij elke lid van het team veronderstelt dat één van de andere architecten een bekend doch cruciaal onderdeel heeft uitgewerkt. Pas veel later blijkt het dus niet opgepakt te zijn - omdat het zo voor de hand ligt, wordt het over het hoofd gezien. Voorbeelden hiervan zijn ook bekend uit de bouwwereld: pas bij de aanvang van de bouw van een flatgebouw kwam men er achter dat de toiletten vergeten waren….

De oorzaak voor het vergeten van fundamentele onderdelen is toch vaak gelegen in onvoldoende samenwerking met belanghebbenden. Wanneer er in betrekkelijk isolatie wordt gewerkt aan de realisatie van een geavanceerd software product, dan kan een gemeenschappelijke blinde vlek leiden tot de 'Missing Piece'. Een nuchtere opmerking van een buitenstaander of toekomstige gebruiker, die simpelweg de vraag stelt: "Waar zit eigenlijk de aan/uit knop van dit navigatiesysteem?" kan een hoop ellende voorkomen.

Ook bij het vervangen van legacy software dient men beducht te zijn voor de 'Missing Piece'. Aangezien de meeste legacy systeem niet voor langere tijd kunnen worden uitgeschakeld ter vervanging, zijn dergelijke migratie trajecten verre van simpel, en vereisen een nauwkeurig ontworpen, stapsgewijze aanpak. De complexiteit van het traject wordt nog opgevoerd doordat vaak van de gelegenheid gebruik wordt gemaakt om modificaties door te voeren of functionaliteit uit te breiden. Niet zelden blijkt tijdens de migratie een paar kleine, maar essentiële functies tussen wal en schip geraakt te zijn, en meestal worden pas veel later de 'Missing Pieces' door een eindgebruiker opgemerkt.

Het is voorgekomen dat dit heeft geleid tot het noodgedwongen in bedrijf houden van zowel het nieuwe als het oude software systeem, met uiteraard torenhoge onderhoudskosten tot gevolg (die uiteindelijk door de gebruikers van het systeem moeten worden opgebracht).

Het is dus ook hier cruciaal dat de software architect alle relevante stakeholders betrekt bij belangrijke beslissingen, in dit geval om nadelige, kostbare gevolgen van het onnodig missen van cruciale functionaliteit te voorkomen.

Het gedragspatroon 'Know Thy Stakeholders' dient elke architect dus hoog in het vaandel te hebben. Dit lijkt een open deur, maar vaak blijken niet alle (belangrijke) stakeholders bekend te zijn bij de architect. Soms zijn bepaalde belanghebbenden min of meer onzichtbaar voor de architect, totdat blijkt dat ze wel degelijk een stem of zelfs beslissingsbevoegdheid bezitten. Zo kan een organisatie die zich bezighoudt met standaardisatie in een waardeketen, wel degelijk een belang hebben bij bepaalde aspecten van de architectuur. Of een marketingmanager die vanwege een onverwachte aankondiging van de concurrentie, als een donderslag bij heldere hemel beslist dat het product 4 maanden eerder op de markt moet komen, noodgedwongen in afgeslankte vorm. Een belangrijke, en soms lastige taak van de architect is om uit de soms overweldigende hoeveelheid belanghebbenden de meest relevante te selecteren. Want de oren te laten hangen naar iedere wens die de architect op z'n bord krijgt, komt de architectuur ook bepaald niet ten goede!

De werkwijze van de 'Ongeduldige Opticien' zien we op een bepaalde manier ook terug in een gedragspatroon dat veelvuldig de kop opsteekt onder software ontwerpers, namelijk het anti-pattern "The Golden Hammer".

Zonder dat er echt goed gekeken wordt naar de essentie van het probleem, en een weloverwogen keuze wordt gemaakt uit de mogelijke oplossingen, wordt er teruggegrepen op bekend gereedschap of een vertrouwde methode. Samenvattend: "Als je enige gereedschap een hamer is, dan is al het andere een spijker". Vooral als een bepaalde aanpak in het verleden goede diensten heeft bewezen, is de verleiding voor het opnieuw toepassen van dezelfde succesformule groot. Daarbij speelt ook vaak de overweging dat er al veel is geïnvesteerd in het beschikbare gereedschap en opleidingen.

Een mogelijk gevolg van de Golden Hammer voor gebruikers van software producten is dat ze worden geconfronteerd met de gevolgen van het toepassen van minder geschikte of zelfs obsolete technologie. Dergelijke nadelen kunnen aan de oppervlakte komen wanneer het gaat om zaken zoals uitbreidbaarheid en connectivity.

Soms kan er op een ontwikkelafdeling een opmerkelijke weerstand bestaan om bewezen standaarden te adopteren, software van anderen te (her-)gebruiken, of om nauw samen te werken met andere ontwikkelteams die met een vergelijkbare oplossing bezig zijn. Dit verschijnsel gaat vaak hand in hand met het onvermogen om te leren van fouten gemaakt in het verleden.

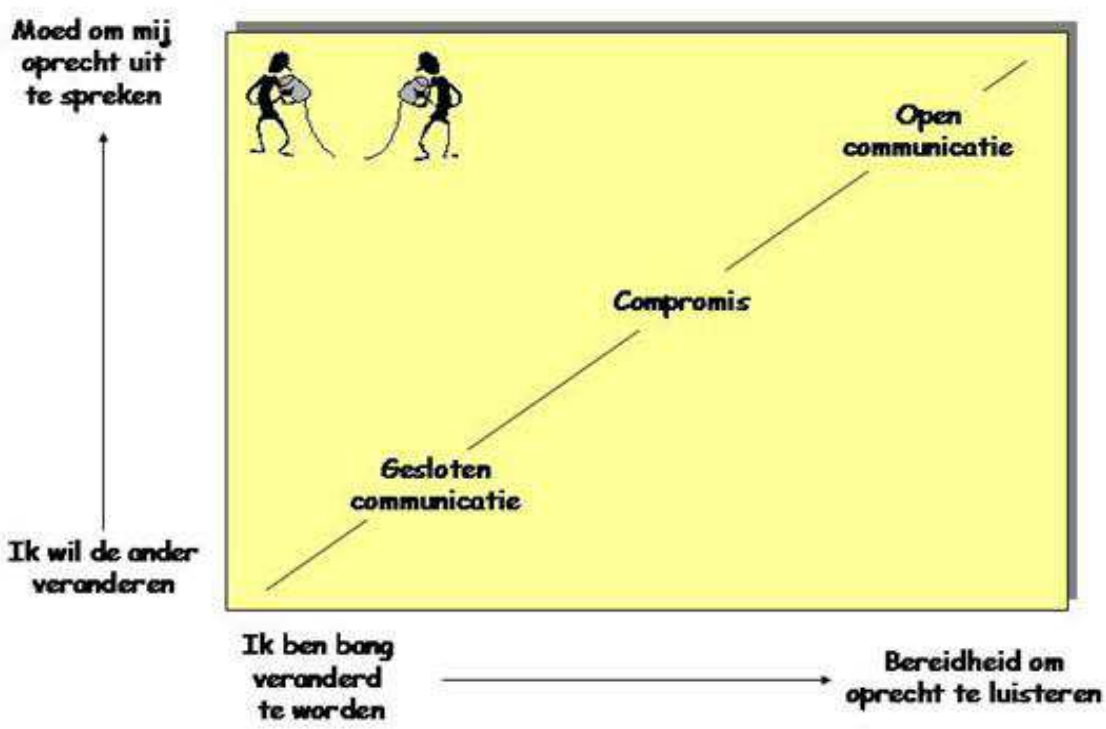
Een misplaatst gevoel van uniekheid vormt de voedingsbodem voor het gedragspatroon 'Het Stokpaard', dat wel enige gelijkenis vertoont met het 'Not Invented Here Syndrome'. Soms wordt deze houding versterkt door een gevoel van superioriteit gebaseerd op het imago van de organisatie, maar kan ook worden veroorzaakt door pijnlijke ervaringen met een door de organisatie opgedrongen samenwerking. Een vaker voorkomend voorbeeld van de laatste oorzaak is de ontwikkeling van een software framework dat door meerdere (zelfstandige) afdelingen gebruikt dient te gaan worden.

In dergelijke omstandigheden wordt de ineffectiviteit van het ontwikkelteam dramatisch versterkt wanneer de architectuur wordt gepresenteerd als het 'magnus opus' van de architect. Een open houding ten opzichte van verreikende veranderingen in de architectuur (of de huidige wijze van werken) zal ver te zoeken zijn wanneer de architect zijn/haar architectuur vanuit dit standpunt te vuur en te zwaard verdedigd.

Een belangrijk element in dit gedragspatroon is wantrouwen – niet alleen ten aanzien van de kwaliteit van het werk van anderen, maar er kan ook sprake zijn van angst om de controle over het dagelijkse werk te verliezen, de eigen identiteit kwijt te raken, of voor het verliezen van de eigen positie of afhankelijkheid. Om deze angst weg te nemen, is om te beginnen intensieve en oprechte communicatie tussen alle betrokkenen noodzakelijk. Ineffectieve communicatie is vrijwel altijd de reden dat het opzetten van een software platform veel moeite kost of zelfs mislukt.

Effectieve communicatie wordt wel eens schertsend omschreven als ‘zo dicht mogelijk langs elkaar heen praten’. Toch zit hier wel een kern van waarheid in, want hoe beter de zender en ontvanger op elkaar zijn afgestemd, des te minder (onuitgesproken) misverstanden er zullen optreden. Dit afstemmen gebeurd door het uitwisselen van verbale en non-verbale signalen. In technisch jargon: ‘encoders en decoders dienen regelmatig gekalibreerd te worden.’ Technieken gebaseerd op Neuro Linguistisch Programmeren (NLP) kunnen gebruikt worden om de invloed van verbale en nonverbale communicatie te begrijpen, te versterken en gericht te gebruiken.

Hoewel een bedrijfscultuur hierbij een invloedrijke factor kan zijn, kan de architect wel degelijk het voortouw nemen om de weg in te slaan van ‘gesloten communicatie’ naar ‘open communicatie’. Hierbij zal de architect zelf het goede voorbeeld dienen te geven, en bovendien over coachingsvaardigheden dienen te beschikken. De volgende figuur dient hierbij als leidraad:



Open communicatie roept altijd een reactie op. Het is soms even volhouden, maar open communicatie zal vrijwel altijd het onderlinge vertrouwen doen toenemen. De vaak gehoorde tegenwerping 'wat te doen als ik zelf open communiceer, en de ander volhardt in gesloten communicatie?' is op de keper beschouwd een voorbeeld van gesloten communicatie. Immers, het bij voorbaat verwachten van een bepaalde houding of reactie van de ander is geen voorbeeld van oprechte, open communicatie.

Open communicatie vraagt moed en doorzettingsvermogen van de architect, maar de beloning is een respectvolle en zinvolle samenwerking in het ontwikkelteam, dat steeds beter raakt aangesloten op de context van de eindgebruikers. In die atmosfeer vervalt voor de architect ook de reden om de architectuur ten koste van alles te verdedigen. Vanwege deze openheid zal de kwaliteit van de architectuur toenemen.

Tientallen jaren voordat Willem Barentz vertrok voor zijn noordelijke expeditie eind 16e eeuw, hadden beroemde Nederlandse kaartenmakers zoals Marcator en Plancius al getracht een waarheidsgetrouwe kaart van het noordpoolgebied te maken.

Ruim 400 jaar later is het nauwelijks voorstelbaar dat hun geografisch inzicht, gebaseerd op mythen, geruchten en waanbeelden, de basis vormde voor kostbare en risicovolle ondernemingen zoals het vinden van de noord-oost passage naar China en Japan.

Zo veronderstelde dat Mercator de noordpool een grote zwarte rots was, omgeven door vier eilanden. Om onduidelijke redenen dacht hij dat de temperatuur van het water rond deze rots altijd boven het vriespunt bleef ('mare librum'), en de noordpool dus per zeilschip gepasseerd kon worden.

Het schijnt dat Amerikaanse parachutisten die tijdens D-Day werden afgeworpen achter de vijandelijke linies, een landkaartje bij zich hadden met daarop de instructie:

"If the territory doesn't match the map, trust the territory".

Deze instructie is gebaseerd op een uitspraak van Alfred Korzybski, een pioneer op het gebied van semantiek:

"A map is not the territory it represents, but, if correct, it has a similar structure to the territory, which accounts for its usefulness…"

Mensen zijn continue bezig met het modelleren van de wereld. Het testen van de validiteit van onze modellen is in feite de basis van hoe wij leren. In ons leven construeren we zoveel modellen, dat we ons nog nauwelijks bewust zijn van dit modelleer proces.

Toch berusten communicatie problemen voor een groot deel op het opdringen van onze map aan anderen - terwijl we weten dat mensen nooit exact gelijke modellen hebben. Geen enkel model is correct - sommige zijn hoogstens bruikbaar.

Eigenlijk halen mensen bij voortduring twee typen concepten door elkaar.

Exoterische concepten zijn concepten die volledig kunnen worden overgedragen tussen mensen door gebruik te maken van nauwkeurige beschrijvingen en taalconstructies. Wiskunde is hiervan een voorbeeld. Esoterische concepten daarentegen zijn concepten die niet volledig overdraagbaar zijn anders dan door directe ervaring. Zo zal een persoon die nooit van een appel heeft geproefd nooit in staat zijn door alleen gebruik te maken van taal, begrijpen wat de smaak van een appel is. Alleen door directe ervaring – het eten van de appel – kan die ervaring volledig worden begrepen.

Dit gegeven is reeds in 1928 uitgewerkt door de surrealistisch kunstenaar René Magritte in zijn schilderij "De Misleiding van Afbeeldingen", waarbij de Franse tekst op het schilderij ons laat weten dat dit geen pijp is.

Begin van de negentiger jaren kwam het gebruik van modellen om de infrastructuur, gedrag en constructie van software te abstraheren, sterk in opkomst. Modelleertalen zoals de Unified Modelling Language (UML) zijn niet meer weg te denken, alhoewel sommigen de mening zijn toegedaan dat UML een contraproductief hulpmiddel is bij software- en systeemontwerp.

Om complexe systemen te begrijpen en te beredeneren, zijn modellen onmisbaar. Een architect dient zich echter bewust te zijn van de vergissing die op de loer ligt bij het hanteren van elk model: dat de modellen worden verwisseld met het feitelijke systeem.

Daarbij komt nog dat dergelijke modellen niet passen bij de belevingswereld van alle groepen van stakeholders – zeker de meer complexe UML modellen zijn exclusief bedoeld voor technisch georiënteerde ontwerpers en spelen nauwelijks een rol in het overleg met andere belanghebbenden. Een te sterke focus op modellen kan zodoende een onbalans aanbrengen in de feedback die de architect ontvangt van afnemers van de architectuur.

Een architect dient zich te bewegen in de wereld van alle belanghebbenden, en zal op basis van die ervaringen een rijk geschakeerde blik op de architectuur ontwikkelen. Het is de architect die vanuit de juiste 'view', de vertaalslag kan maken naar de wereld van de modellen en vice versa. Een dergelijke architect ziet modellen daarbij als een mogelijk bruikbaar hulpmiddel om tot hoogwaardige producten te kunnen komen.

In tegenstelling daarmee, verheft de cartograaf het model of de methodiek tot doel. In ieder geval wordt de verzameling modellen gebruikt om zich af te schermen voor de feedback van bepaalde groepen van stakeholders – de eindgebruikers van vlees en bloed met 'echte' (en dus zeer relevante) ervaringen dreigen in het modelleer proces een ondergeschikte rol te gaan spelen. Een architect met dit gedragspatroon gaat onwillekeurig een soort tunnelvisie ontwikkelen ten aanzien van het architectuur model. Wanneer na veel bloed, zweet en tranen een model is verkregen dat ontwerp-technisch uitstekend in elkaar steekt, dan zal verstorende feedback uit onverwachte hoek niet altijd met gejuich worden begroet.

De invloed van deze vorm van verkokering op het uiteindelijke software product laat zich gemakkelijk raden. De kans dat het product in technisch opzicht goed is geconstrueerd is groot, maar of het ook het goede product is gemaakt is nog maar de vraag. De menselijke maat laat zich niet vangen in modellen.