DEV

Gelaagde architecturen

Deel 3: Ontwerpen van een fysiek lagenmodel

Een lagenmodel maakt deel uit van de meeste software architecturen. Het gekozen lagenmodel kan het succes van een applicatie maken of breken. Een goed lagenmodel zorgt ervoor dat bepaalde kwaliteitsdoelen gerealiseerd kunnen worden met acceptabele bouw en beheerkosten. Het vorige artikel (1) beschreef het ontwerpen van een logisch lagenmodel. Dit artikel beschrijft hoe dit logische model vertaald moet worden naar een fysiek lagenmodel.

Het bepalen van het fysieke lagenmodel is een soort van vuurproef voor het logisch ontwerp. Want in PowerPoint werkt elk model, maar de fysieke implementatie is vaak lastiger dan gedacht. Gebruik daarom een iteratief proces bij het ontwerpen van het lagenmodel, waarbij de logische en fysieke bril elkaar regelmatig afwisselen. Zodoende komen technische beperkingen eerder naar voren, waarna kwaliteitsdoelen afgezwakt kunnen worden of voor een andere implementatietechniek gekozen kan worden.

Het fysiek lagenmodel

Het fysieke lagenmodel beschrijft de fysieke lagen en bijbehorende communicatieregels waaraan de applicatie en infrastructuur specialisten zich dienen te houden. Het ontwerp van het fysieke lagenmodel richt zich vooral op de vraag: Welke lagen moeten er, rekening houdend met de techniek, onderscheiden worden en hoe moeten die lagen gerealiseerd worden om aan de niet-functionele eisen te voldoen?

Kwaliteitsdoelen
1. Alle gegevens die worden vastgelegd, moeten gecontroleerd zijn.
2. Goede analyseerbaarheid van het systeem.
3. Domeingenerieke logica wordt slechts op één plek vastgelegd en van daaruit hergebruikt.
4. De applicaties zijn zoveel mogelijk onafhankelijk van wijzigingen in de databasestructuur.
5. DBMS, applicatieserver, randapparatuur, et cetera moeten eenvoudig vervangen kunnen worden.
6. Het systeem moet op alle toegangsniveaus beveiligd zijn.
7. Goede schaalbaarheid.
8. Responsiesnelheid gemiddeld maximaal 1 sec (rapportages uitgezonderd).

Tabel 1: De kwaliteitsdoelen van het AanwezigheidsRegistratieSysteem.

Heuristiek fysiek lagenmodel

We continueren het ontwerpproces uit artikel 2 (Optimize 2, maart 2011) met vijf overeenkomstige stappen die nu met een technische gerichtheid doorlopen worden. Ook hier worden de ontwerpstappen voor de duidelijkheid sequentieel beschreven, terwijl ze in de praktijk vaak parallel en/of iteratief uitgevoerd worden.

Stap 6: Evalueer de kwaliteitsdoelen en kies het platform

De resultaten van deze stap zijn (eventueel bijgestelde) kwaliteitsdoelen die met de gekozen technologie te realiseren zijn. Er moet daarom worden bepaald of de kwaliteitsdoelen vanuit het logisch lagenmodel moeten worden bijgesteld tot een realistisch niveau voor de gekozen technologie of dat een andere technologie nodig is.
Om te beginnen moeten de geselecteerde kwaliteitsdoelen voor het project weer opgezocht worden. In dit artikel gebruiken we als voorbeeld een webgebaseerd AanwezigheidsRegistratieSysteem voor een onderwijsinstelling. Een bijlage met een compleet uitgewerkt voorbeeld hiervan, met alle stappen van het logische en fysieke lagenmodel, is te vinden op de publicatiesite van Hogeschool Utrecht (2).
In tabel 1 zijn de bijbehorende kwaliteitsdoelen beschreven. Tijdens het opstellen van het logisch lagenmodel is vastgesteld dat alleen de doelen 1-6 met een lagenmodel kunnen worden geadresseerd.

Kies de technologie die bij de doelen past

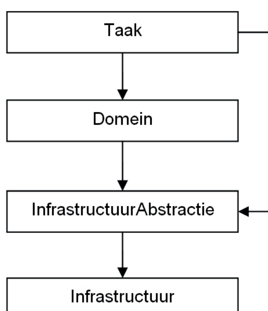
Vervolgens is het nodig om een keuze te maken voor het type platform waarmee de applicatie gerealiseerd wordt, bijv. Java,

Ruby on Rails, Microsoft.Net etc. Bij die keuze moeten de gestelde kwaliteitsdoelen worden meegenomen. Het platform wordt meestal voorgeschreven door de organisatie die deze applicatie in beheer gaat nemen. Een beheerorganisatie die gespecialiseerd is in Java, zal voor nieuwbouw ook een voorkeur voor Java hebben. Vaak worden dit soort eisen met een Project Start Architectuur (PSA) voorgeschreven.
In het voorbeeld voor dit artikel is voor het Aanwezigheids-RegistratieSysteem (ARS) de keuze gevallen op Ruby on Rails (RoR). RoR is een open-source framework dat is bedoeld om webapplicaties in korte tijd te bouwen met behoud van goed onderhoudbare applicatie logica. RoR gebruikt de object georiënteerde programmeertaal Ruby, kan op zowel Windows als Unix ontwikkeld worden, echter het hosten van RoR is alleen mogelijk op een Unix variant.

Tenslotte kan met de kennis van de techniek gekeken worden of alle kwaliteitsdoelen te realiseren en betaalbaar zijn. Bijvoorbeeld de performance eis is discutabel voor een webgebaseerde applicatie. En de schaalbaarheidseis is te vaag. Voor beide eisen is nadere afstemming met de opdrachtgever nodig. Ook kan het zijn dat er kwaliteitsdoelen bijkomen waar, vanwege zwakten in de techniek, extra aandacht aan besteed moet worden, bijvoorbeeld met betrekking tot security of continuïteit. Zo is er in het ARS-voorbeeld de eis bijgekomen dat meerdere browsers en browserversies ondersteund moeten worden.

Stap 7: Onderken en ontwerp de fysieke lagen

Na de keuze voor het platform kan gekeken worden hoe de logische lagen worden vertaald in fysieke lagen. Daarbij moet onder andere besloten worden of er lagen bijkomen, met welke technologie iedere laag precies gerealiseerd gaat worden en hoe de lagen op de tiers gemapt moeten worden. Uitgangspunt bij dat alles is het logisch lagenmodel, waarvan die van het ARS-voorbeeld in figuur 1 is weergegeven. De Taak-laag bevat de presentatielogica en taakspecifieke logica en de Domein-laag de domeingenerieke logica, conform de terminologie van het Logica In Lagen referentiemodel(3).



Figuur 1: Logische lagen ARS-voorbeeld.

7.1 Onderken en ontwerp de fysieke lagen

Het uiteindelijke lagenmodel moet de ontwikkelaars duidelijkheid geven welke functionaliteit waar en met welke technologie geïmplementeerd moet worden. Een lijstje met geboden en verboden dus. Dit is nodig omdat de techniek vaak meerdere mogelijkheden biedt en door net de verkeerde ontwerpkeuze te maken kan een belangrijk kwaliteitsdoel gemist worden.

Het fysieke lagen model ontstaat door de volgende stappen uit te voeren voor iedere logische laag:

1. Bepaal wat de implementatie alternatieven zijn binnen het gekozen platform voor de verantwoordelijkheden van de logische laag. Vaak zijn er meerdere frameworks en technieken beschikbaar.
2. Toets de alternatieven tegen de kwaliteitscriteria en kies het alternatief dat het beste past.
3. Denk na over het opsplitsen van logische lagen over meerdere fysieke lagen. Zo wordt in het ARS-voorbeeld de logische Taak-laag in meerdere fysieke lagen gesplitst.
4. Evalueer of de kwaliteitsdoelen nog steeds afdoende gehaald worden en pas waar nodig het fysieke lagenmodel of de kwaliteitsdoelen aan totdat er een stabiel lagenmodel ontstaat.
5. Ga door tot alle logische lagen ingevuld zijn.

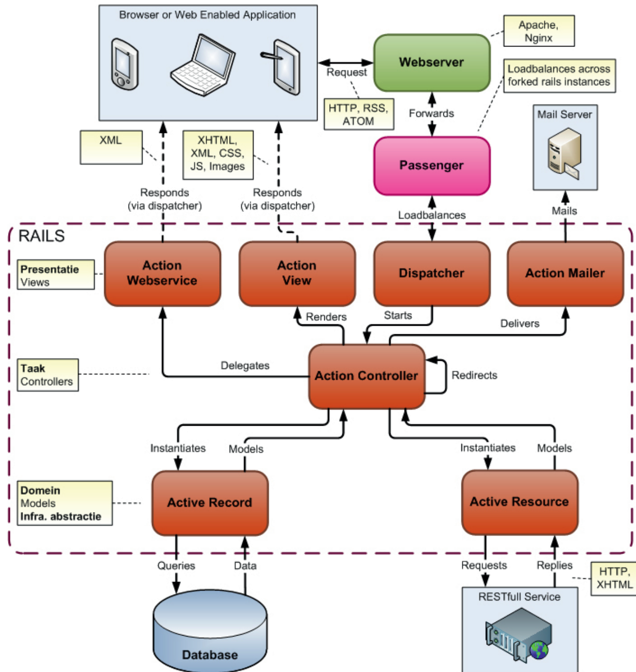
Neem hierbij de volgende ontwerpregels in acht:

- De doelen van het logische lagenmodel moeten gehaald worden.
- Als je de gekozen technologie niet goed kent, begin dan met de architectuur van de leverancier en wijk hiervan af als deze niet past bij de eisen. Het grote voordeel van het gebruiken van de standaard architectuur is dat hier vele voorbeelden van te vinden zijn en dat de sterke en zwakke punten bekend zijn.
- Vertrouw de standaard architectuur van een leverancier nooit helemaal. Elke architectuur heeft sterke en zwakke plekken, zorg ervoor dat je snapt welke zwakke plekken een bedreiging zijn voor de kwaliteitsdoelen.
- Vermijd overbodige keuzes waar geen kwaliteitsdoel voor aanwezig is. Dit voorkomt “goudgerande” oplossingen met teveel lagen of functionaliteit die niet gebruikt worden

Ruby on Rails technologie architectuur

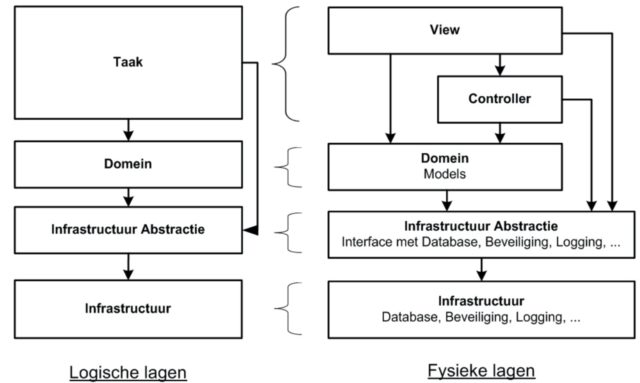
In het kader van het AanwezigheidsRegistratieSysteem is de Ruby on Rails technologie architectuur bestudeerd. Figuur 2 toont de standaard architectuur ('leveranciers architectuur') van RoR die gebaseerd is op het architectuurpattern Model-View-Controller. Rails biedt standaard drie soorten views: Webpagina’s (ActionView), E-Mail (ActionMailer) en Webservices (ActionWebservices). Eén soort controller (ActionController) en twee soorten van Models gecombineerd met

infrastructuur abstractie, namelijk een Model dat interacteert met een database (ActiveRecord) en een Model dat interacteert met webservices (ActiveResource). Alle Ruby classes erven van abstracte frameworkclasses zoals ActionController of ActiveRecord.



Figuur 2: Ruby on Rails detail fysieke software architectuur.

De match tussen de logische lagen en de fysieke lagen staat in figuur 3. Er zijn enkele verschillen waar ontwerpkeuzen over gemaakt moeten worden.
Ten eerste komt de Logische Taak-laag met meerdere fysieke RoR-klassetypen overeen. Dat is ook bij andere platformen gebruikelijk. Maar zijn het ook aparte lagen? Vanwege het kwaliteitsdoel 2, analyseerbaarheid, is het goed om het MVC-pattern toe te passen en View en Controller functionaliteit te scheiden over verschillende klassen en desnoods packages. Maar als View en Controller ook als lagen gescheiden gaan worden, mag volgens de standaard communicatieregels View alleen nog maar diensten vragen aan Controller en niet andersom. De pijlen in figuur 2 tussen Action Controller en Action View suggereren wat anders. Meer hierover volgt in stap 8.
Ten tweede is er in stap 6 de eis bij gekomen dat meerdere browsers ondersteund moeten worden, wat pleit voor het scheiden van View en Controller.
Ten derde blijken de Views de Models te benaderen om gegevens te tonen. En alle klassen maken van een aantal infrastructurele zaken zoals logging gebruik. Ook dit gaat in tegen de standaard communicatieregels. In stap 8 wordt verder ingegaan op dit aandachtspunt.



Figuur 3: Mapping Logische lagen en Fysieke lagen.

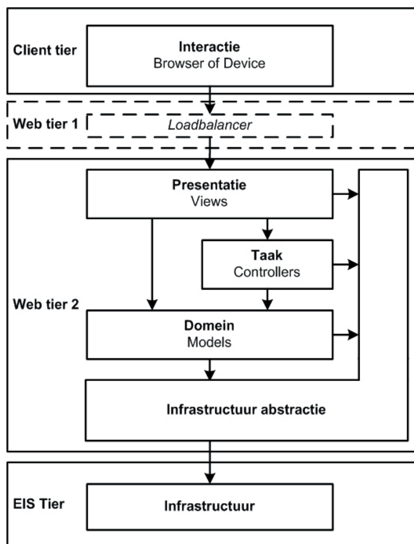
7.2 Positioneer de fysieke lagen op de tiers.
De termen lagen en tiers worden vaak door elkaar gebruikt. De meeste auteurs houden het onderscheid aan dat lagen voortkomen uit een logische, verticale ordening van de functionaliteit, terwijl tiers voortkomen uit deployment overwegingen en leiden tot een horizontale indeling. Op lagen is het Layer-pattern van Buschmann (4) compleet van toepassing en op tiers niet.
De Patterns en Practices Group van Microsoft (5) heeft een aantal hele goede ontwerpregels staan in hoofdstuk 19 van hun boek, dat gratis te downloaden is en ook voor de niet-Microsoft architect een heel goede referentie om te gebruiken bij architectuur ontwerp.

De belangrijkste ontwerpregels die zij beschrijven voor Tiers en deployment zijn:

1. Positioneer lang lopende kritieke processen op een aparte tier. In de regel verlangen kritieke processen een andere integriteit en beschikbaarheid dan reguliere applicatie onderdelen.
2. Distribueer alleen wanneer dat echt nodig is. Redenen om onderdelen van een systeem op verschillende tiers te plaatsen zijn: beveiligingsbeleid, fysieke beperkingen (bijv. bandbreedte), gedeelde business logica en schaalbaarheid.
3. Plaats presentatie en domeinlogica op verschillende tiers als het vanuit beveiliging nodig is om bijv. autorisatie of een ander beveiligingsmechanisme er tussen te plaatsen. Dit kan bijvoorbeeld van toepassing zijn als de presentatie logica op een smartphone komt en de domeinlogica op de server. In dit geval is er geen enkele garantie dat de smartphone niet gehacked of gestolen is. De beveiliging moet dan server side bewaakt worden.
4. Plaats fysieke lagen op dezelfde tier als deze elkaar synchroon aanroepen.

Fysieke lagen en tiers van ARS
Toepassing van deze ontwerpregels op het Aanwezigheids-

RegistratieSysteem kan bijvoorbeeld leiden tot het model dat in figuur 4 is weergegeven. Die toont een eenvoudig tier model, waarbij de meeste lagen op web tier 2 zijn geclusterd. De hoofdoverweging hierbij is dat deze groep van logische lagen vrijwel altijd als geheel betrokken is bij de interactie met de client. Een andere overweging is het aantal fysieke voorkomens van een bepaalde laag van het systeem. Het is het beste om lagen met ongeveer dezelfde aantallen van gebruik in dezelfde tier te plaatsen. Voor het ARS-systeem is het verwachtte aantal fysieke voorkomens van de views, taak, domein en infrastructuur abstractie lagen maximaal enkele tientallen. Terwijl het verwachtte aantal voorkomens van de infrastructuur maximaal tien is.
Boven de web tier is een loadbalancer geplaatst. Voor de software is deze loadbalancer onzichtbaar, maar bij het ontwerp van de software moet wel rekening gehouden worden met de manier waarop session state bewaard wordt en hoe locking geregeld wordt.



Figuur 4, Het fysieke model op tiers gepositioneerd.

Stap 8: Ontwerp de communicatie tussen de lagen

De fysieke lagen en de fysieke distributie zijn nu bepaald. De wijze waarop de lagen met elkaar communiceren is nu nog niet helder. In deze stap wordt de invulling van de communicatie tussen de lagen bepaald. Een goed ontwerp van de communicatie tussen de lagen is erg belangrijk voor de applicatie, zonder deze communicatie zouden lagen elkaar niet kunnen aanspreken.
Als eerste worden in stap 8.1 de logische communicatieregels tegen het licht gehouden en vervolgens wordt in stap 8.2 het ontwerp van de communicatieregels ingevuld. In stap 8.3 wordt als laatste de exceptie afhandeling bepaald.

8.1 Evalueer de communicatieregels gedefinieerd op logisch niveau.
Vanuit een technisch standpunt kan gekeken worden of alle communicatieregels die op logisch niveau verzonnen zijn geïmplementeerd kunnen worden. Het kan zijn dat er in bepaalde situaties tegen een regel gezondigd moet worden. Dat hoeft geen probleem te zijn, indien er in een dergelijke situatie een technische truc is waarmee toch het kwaliteitsdoel gehaald kan worden.
Voor het ARS-voorbeeld zijn op logisch niveau drie gebodsregels en drie uitzonderingsregels afgeleid van de kwaliteitsdoelen. Zie tabel 2.
Voor de regels 1, 3, 4 en 5 zijn voor RoR geen ontwerpbeslissingen nodig. Die kunnen zo worden overgenomen. Voor de regels 2 en 6 moet uitgezocht worden wat de consequenties zijn en welke technische oplossing het best voldoet en tot standaard verheven wordt. Design patterns kunnen hier heel goed bij helpen. Zo geeft Larman (6) aan dat het Observer pattern een oplossing biedt, wanneer toch van beneden naar boven gecommuniceerd moet worden.
Omdat er in stap 6 een kwaliteitsdoel bij is gekomen en in stap 7 een laag, zullen de communicatieregels daar nog aan moeten worden aangepast.

Gebodsregels
1. Bij verwijdering, wijziging of invoer van gegevens mag de domeinlaag niet worden overgeslagen.
2. Geen aanroepen van de Domeinlaag naar de Taaklaag die ten koste gaan van de herbruikbaarheid.
3. De infrastructuurabstractielaag mag nooit worden overgeslagen.
Uitzonderingsregels
4. Aanroep vanuit alle lagen naar de security component in de Infrastructuurlaag is toegestaan (via de infrastructuurabstractielaag).
5. Bij read-only acties mag de domeinlaag worden overgeslagen.
6. Read-only acties, waarbij de domeinlaag wordt overgeslagen moeten via een databasestructuur-abstractie mechanisme werken.

Tabel 2: Communicatieregels van het AanwezigheidsRegistratieSysteem.

8.2 Ontwerp de fysieke communicatie tussen de lagen.
Ontwerp de fysieke communicatie door voor elke fysieke laag de volgende stappen te nemen:
1. Bepaal welke communicatie er nodig is met deze fysieke laag.
2. Bepaal de mogelijkheden die het platform biedt om de communicatie te implementeren. Bijvoorbeeld, met http of method calls; open of platform specifiek, etc.
3. Kies de communicatievorm die past bij de kwaliteitsdoelen.
4. Kies zo nodig aanvullende maatregelen om specifieke

kwaliteitseisen te realiseren. Denk aan eisen als replaceability of testability.

5. Ontwerp de foutafhandelingsstrategie. Siedersleben (7) gaat uitgebreid op dit onderwerp in en stelt: "het omgaan met fouten/excepties en het verborgen houden van implementatiedetails gaat slecht samen. Daarom is het belangrijk om de wijze waarop lagen hun fouten en excepties afhandelen goed te ontwerpen." Vreemd genoeg komt foutafhandeling er vaak bekaaid vanaf, met alle toekomstige onderhoudsproblemen van dien.

Neem de volgende ontwerpregels in acht:

- Een lage koppeling en vervangbaarheid kan gerealiseerd worden door een laag als een black box te implementeren en van een duidelijke (service) interface te voorzien. Buschmann (4) raadt dit aan en Larman (6) adviseert hiervoor toepassing van het Facade pattern.
- Indien de volgordelijkheid van de communicatie gegarandeerd moet worden, overweeg dan een synchroon communicatie model. Hanteer anders bij voorkeur asynchrone communicatie tussen de lagen voor maximale performance en minimale koppeling tussen de lagen. Indien een hoger gelegen laag alleen synchroon kan werken, maak dan component die de bestaande asynchrone communicatie met een synchrone interface aan deze hogere laag aanbiedt. (5)
- Gebruik open standaarden (HTTP, XML, XHTML, JSON etc.) en een berichtgebaseerde communicatie (SOAP, REST etc.) om de lagen zo ontkoppelt mogelijk te houden en de interoperabiliteit te verhogen. En vermijd het gebruik van platform specifieke datatypes in de communicatie. Randvoorwaarde is wel dat de performance voldoende blijft.

Communicatieregels en de RoR-architectuur

De communicatieregels van het ARS-systeem (tabel 2) zijn voor Ruby on Rails geen probleem.

Gebodsregels 1, 2 en 3 worden door RoR afgedwongen. Uitzonderingregel 4 wordt gerespecteerd, op al de lagen kan een beveiligingscomponent toegepast worden, die beveiliging compleet onzichtbaar in de applicatie inbouwt. Uitzonderingregels 5 en 6 zijn doorgaans niet nodig, want ook read-only access wordt in de models (en daarmee in het domein) ingebouwd. Alleen bij hoge performance eisen is het aan te raden om een database abstractie zoals een stored procedure of view toe passen.

De communicatie met externe systemen vindt plaats met open standaarden (zie figuur 2). De interne communicatie tussen de views, controllers en models vindt plaats door messages tussen de classes. De communicatie met de database vindt op basis van SQL met een database specifiek transportprotocol over TCP-IP plaats. De database is verbogen achter de infrastructuur abstractie logica en kan transparant vervangen worden door een andere database.

Stap 9: Toets de geschiktheid van het lagenmodel

Deze stap richt zich met op de vragen: Kan de software wel gaan werken onder de bedachte architectuur? En worden de kwaliteitsdoelen dan voldoende gerealiseerd?

In artikel 2 van deze serie (1) is deze stap al beschreven voor het logisch lagenmodel. Dezelfde aanpak kan bij het fysieke lagenmodel gekozen worden. Dus doorloop op papier (sequence diagram) de afhandeling van de significante use cases integraal over alle lagen, incl. fout scenario's. En bouw een klein prototype. Wat blijkt wel en niet te kunnen? Bijvoorbeeld, kan de database vervangen worden conform kwaliteitsdoel 5? Probeer ook een aantal kwaliteitsdoelen en regels geweld aan te doen in het prototype.

Stap 10: Documenteer het lagenmodel

Ga uit van de documentatie voor het logische lagenmodel, pas die aan en breidt die uit met de resultaten van de ontwerpstappen van het fysieke lagenmodel. Zorg ervoor dat de grafische weergaven begrijpelijk zijn voor anderen en schrijf er een toelichting bij, zodat de ontwikkelaars kunnen weten welk type functionaliteit thuishoort in welke laag en welk type klasse. En met welke consequenties van het tier model en de communicatieregels zij rekening moeten houden.

Leg ook de overwegingen bij de keuzes vast. Helemaal compleet maak je het met een confrontatie matrix "doelen x keuzes", die inzichtelijk maakt of alle doelen door technische keuzes zijn afgedekt. Dit wordt ook wel een traceability matrix genoemd.

Literatuur

(1) Meer inzicht in een gelaagde architectuur; Deel 2: Ontwerpen van een logisch lagenmodel Pruijt, Leo, Wiersema, Wiebe Release, 2011, maart
(2) Gelaagde Architecturen 4 - Voorbeelden.pdf, Pruijt, Leo, Wiersema, Wiebe www.onderzoek.hu.nl/publicaties (en zoek vervolgens op auteur)
(3) Meer inzicht in een gelaagde architectuur; Deel 1: Uitleg, terminologie en methoden Pruijt, Leo, Wiersema, Wiebe, Release, 2010, december
(4) Buschmann, Frank et al, Pattern-Oriented Software Architecture: A System of Patterns John Wiley, 1996
(5) Microsoft Patterns & Practices; Microsoft Application Architecture Guide; Microsoft Corporation 2009
(6) Larman, Craig, Applying UML and Patterns, Pearson Education, 2005
(7) Siedersleben, Johannes, Moderne Software-Architektur, Dpunkt.verlag, 2004
(8) Fowler, Martin, Patterns of Enterprise Application Architecture, Addison Wesley Publishing, 2003
(9) International Organization for Standardisation, ISO/IEC 9126-1:2001

Leo Pruijt (l.) is docent op de Hogeschool Utrecht, verbonden aan het lectoraat 'Architectuur van Digitale Informatiesystemen.

Wiebe Wiersema is Vakgroepleider Solution Architect bij Capgemini.