## Architectural Knowledge Management

de Boer, R.C.

2009

# Samenvatting

Het is onmogelijk ons nog een wereld zonder software voor te stellen. Software is een onlosmakelijk deel gaan uitmaken van ons dagelijks leven, soms zonder dat we ons er al te zeer van bewust zijn. In de supermarkt worden onze boodschappen aan de hand van barcodes 'gescand'. We betalen met onze pinpas, of halen met diezelfde pas geld 'uit de muur'. Bij autopech wordt eerst de boordcomputer uitgelezen voordat de motorkap open gaat. Steeds meer mensen bellen via 'Voice over IP' (VoIP), waarbij telefoongesprekken feitelijk over computernetwerken lopen. Aandelenbeurzen, nieuwsdiensten, treinen, vliegtuigen, ziekenhuizen, mobiele telefoons, televisies en hard disk recorders; overal waar we kijken draait de wereld op software.

Het bouwen van die software, en met name het bouwen van zeer grote softwaresystemen, staat ook wel bekend als 'software engineering'. Binnen de software engineering is men continu op zoek naar betere technieken om betere software te bouwen. Hierbij dient opgemerkt te worden dat software engineering als discipline eigenlijk nog maar betrekkelijk jong is. De term 'software engineering' ontstond weliswaar als analogie naar andere ingenieursdisciplines – met name bouwkunde was een bron van inspiratie – maar de 40 jaar die verstreken zijn sinds de term in gebruik kwam staan in geen verhouding tot de duizenden jaren bouwkundige historie.

De bouwkundemetafoor is ook terug te vinden in het deelgebied dat bekend is komen te staan als 'software-architectuur'. Software-architectuur behelst het structureren en modelleren van software in modulaire eenheden op een hoger abstractieniveau dan dat van de implementatie (de uiteindelijke 'code'). Door code op een logische manier te groeperen in modules en de interne werking van een module als het ware af te schermen voor de buitenwereld (waaronder andere modules) wordt het mogelijk op een andere manier naar de werking van software te kijken dan wanneer we de details van de software regel voor regel zouden bestuderen. En door vooraf over de architectuur na te denken kan deze een leidraad vormen voor de latere implementatie van de software. Van de software-architectuur wordt dan ook wel gezegd dat deze de manifestatie is van de vroege ontwerpbeslissingen.

Sinds begin jaren zeventig, toen het begrip software-architectuur in zwang raakte, is er in het vakgebied veel vooruitgang geboekt met het beschrijven en analyseren van software-architecturen, vaak in termen van 'componenten' (de bouwblokken) en 'connectoren' (de verbindingen tussen die bouwblokken). Het beschouwen van architectuur als samenstelling van componenten en connectoren betekent echter dat we architectuur altijd zien als het eindresultaat, en nooit als de weg naar dat resultaat toe. De componenten en connectoren *per se* vertellen ons immers niets over de afwegingen die de software-architect heeft gemaakt, de alternatieven die hij heeft overwogen (en verworpen), de bij deze overwegingen betrokken belangen, et cetera. Kortom, door alleen te

kijken naar het eindresultaat wordt belangrijke kennis omtrent de architectuur buiten beschouwing gelaten. Wanneer architectuurkennis slechts gedeeltelijk wordt overgedragen, dan leidt dat tot additionele, moeilijk te kwantificeren kosten. Bovendien kan deze onvolledige kennisoverdracht leiden tot zogenaamde 'ontwerp-erosie'.

Het begrip 'architectuurkennis' krijgt de laatste jaren meer en meer nadruk. Er zijn vier belangrijke perspectieven op wat zulke architectuurkennis precies inhoudt, gedreven door vier dominante toepassingen van die kennis. Ten eerste is er de min of meer klassieke benadering van architectuurkennis als (geformaliseerde) kennis over de componenten en connectoren. Dit perspectief komt met name voort uit de toepassing van dynamische architecturen voor bijvoorbeeld 'zelfhelende' systemen: systemen die hun eigen software kunnen analyseren en de architectuur van die software zelf aan kunnen passen om bijvoorbeeld defecte componenten uit te schakelen en te vervangen door alternatieven.

Vanuit een tweede perspectief wordt architectuurkennis gedocumenteerd in een groeiende collectie van zogenaamde 'ontwerppatronen' die kennis over herhaald optredende problemen koppelt aan bewezen oplossingen. Deze patronen (of 'patterns') worden benoemd en besproken in boeken en op conferenties, en zijn daarmee zowel een vorm van kennisoverdracht als een verrijking van het vocabulaire dat de softwarearchitect ter beschikking staat.

Een derde perspectief op architectuurkennis behelst de traceerbaarheid van de architectuur naar het onderliggende pakket van eisen en wensen. Slechts wanneer deze traceerbaarheid in orde is kunnen wijzigingen in de eisen (bijvoorbeeld door een veranderende klantvraag, maar ook door veranderingen in wet- en regelgeving of nieuwe technologische doorbraken) vertaald worden naar wijzigingen in de architectuur. Omgekeerd kan een bestaande architectuur invloed hebben op de (on)mogelijkheid om nieuwe of veranderende eisen te realiseren.

Ten slotte is er een vrij recente benadering van architectuurkennis als de ontwerpbeslissingen die leiden tot het architectuurontwerp. Vanuit dit perspectief dienen de ontwerpbeslissingen als concrete entiteiten te worden onderkend. In tegenstelling tot het uiteindelijke architectuurontwerp hebben de ontwerpbeslissingen intrinsieke relaties met de overwogen alternatieven, de uiteindelijk gekozen oplossing, en de verschillende belangen – waaronder economische, politieke, en technologische randvoorwaarden – die een rol gespeeld hebben in het beslissingsproces. In een groter verband kan een focus op ontwerpbeslissingen dan ook gebruikt worden om de verschillende 'vormen' van architectuurkennis op een lijn te brengen en aan elkaar te relateren.

Dit proefschrift is een resultaat van het GRIFFIN onderzoeksproject waarin industriële en academische partners samengewerkt hebben aan betere manieren voor architectuurkennismanagement. Ons onderzoek is gestoeld op een beslissingsgeoriënteerd perspectief op architectuurkennis. Wij beschouwen het beslissingsproces als een lus,

waarin afgewogen wensen en eisen leiden tot ontwerpbeslissingen die vervolgens weer leiden tot nieuwe wensen en eisen die in vervolgbeslissingen meegewogen worden, et cetera, et cetera. Een analyse van vier verschillende organisaties aan de hand van een model waarin deze 'beslislus' centraal staat heeft geleid tot de identificatie van vier probleemgebieden met betrekking tot architectuurkennismanagement: het *delen* van architectuurkennis binnen en tussen organisaties, het *ontdekken* van relevante architectuurkennis in bestaande documenten en informatiesystemen, het *voldoen aan* opgestelde architectuurregels en richtlijnen, en het *traceren* van de verbanden tussen ontwerpbeslissingen en andere kennisentiteiten. Hieruit zijn binnen het GRIFFIN project vier overeenkomstige onderzoekslijnen ontstaan. In dit proefschrift gaan we in op twee van de geïdentificeerde problemen: het delen van architectuurkennis, vanuit het perspectief van de software-architect die verantwoordelijk is voor het ontwerpen van de software; en het ontdekken van architectuurkennis, vanuit het perspectief van de auditor die een kwaliteitsanalyse van de ontwikkelde software uitvoert.

Het delen van architectuurkennis is belangrijk omdat architecten typische kenniswerkers zijn. Zij spelen vaak een centrale rol in het softwareontwikkelproces en zijn verantwoordelijk voor het nemen van belangrijke ontwerpbeslissingen. Hiervoor communiceren ze met allerlei belanghebbenden over technische en niet-technische zaken, en proberen ze de beste afwegingen te maken rekening houdend met de wensen en eisen van deze belanghebbenden. Effectieve methoden en technieken om architecten te helpen kennis te delen zijn zeer gewenst. In dit proefschrift presenteren we verschillende van deze methoden en technieken.

Om na te gaan wat voor methoden en technieken architecten kunnen helpen in het delen van architectuurkennis hebben we vier praktijkstudies uitgevoerd bij een softwareontwikkelorganisatie. Om de verkregen resultaten nader te valideren hebben we aansluitend een vijfde studie uitgevoerd: een vragenlijstonderzoek waaraan architecten van vier organisaties in Nederland hebben deelgenomen. Deze vijf studies passen goed binnen een zogehete 'action research' cyclus die bestaat uit een diagnostische fase, gevolgd door enkele ontwikkelfases en een evaluatie- en reflectiefase. Deze kwalitatieve onderzoeksmethodiek stelde ons in staat om in nauwe samenwerking met de architecten de huidige problematiek met betrekking tot het delen van architectuurkennis boven water te krijgen, en tot bruikbare oplossingen en verbeteringen te komen die goed aansluiten bij hun activiteiten en kennisbehoeften.

Tijdens een eerste diagnose hebben we het gebruik van bestaande hulpmiddelen voor het delen van architectuurkennis onderzocht. Wat hierbij opviel, was dat architecten deze hulpmiddelen grotendeels naast zich neer legden. Tijdens interviews werden hier verschillende verklaringen voor gegeven, waaronder de steile leercurve van deze hulpmiddelen, de tijd die het kost om deze hulpmiddelen te gebruiken en de beperkte toegevoegde waarde ten opzichte van meer traditionele communicatievormen waaron-

der telefoon of e-mail. Architectuurkennis bleek in deze organisatie veelal gedeeld te worden tijdens informele bijeenkomsten of op de gang bij de koffieautomaat. Het bleek dat kennisdeling middels specialistische hulpmiddelen iets is waar architecten gemotiveerd voor moeten zijn. In ons onderzoek hebben we een aantal voorwaarden gedefinieerd die noodzakelijk zijn om een omgeving te creëren waarbinnen architecten aangemoedigd worden om hun kennis te verspreiden.

De volgende fase van ons onderzoek behelste het ontwerpen van efficiëntere hulpmiddelen voor het delen van architectuurkennis. Na een uitgebreide inventarisatie van de wensen van architecten zelf en door gebruik te maken van inzichten bekend uit de literatuur, hebben we een zevental eigenschappen opgesteld waaraan hulpmiddelen dienen te voldoen. Op basis van deze eigenschappen hebben we vervolgens in twee opeenvolgende praktijkstudies multifunctionele, webgebaseerde IT-omgevingen geïmplementeerd die het mogelijk maken om verschillende typen architectuurkennis efficiënt en op het juiste moment te delen. De evaluaties van deze omgevingen door architecten laten zien dat dit soort 'all-round'-oplossingen uitermate geschikt zijn om het delen van architectuurkennis binnen een organisatie te bevorderen.

Ons onderzoek naar efficiënte ondersteuning voor het delen van architectuurkennis heeft een aantal belangrijke inzichten opgeleverd. Een van deze inzichten is dat architecten naast het maken van architectuuroplossingen zich met veel andere kennisintensieve activiteiten bezig blijken te houden. Dit inzicht is op zichzelf niet heel verrassend, maar dit betekent wel dat kennismanagementondersteuning voor architecten zich moet richten op al deze activiteiten en niet beperkt moet blijven tot bijvoorbeeld een hulpmiddel om architectuurmodellen te maken.

Een ander belangrijk inzicht is dat niet alle architectuurkennis eenvoudig expliciet te maken is. Kennismanagementonderzoekers, en zeker degenen die zich richten op het IT-domein, hebben vaak de neiging om de oplossing voor kennisdeling te zoeken in sjablonen, databanken of andere opslagmechanismen waarmee de kennis in een bepaalde vorm gegoten wordt en vervolgens eenvoudig te hergebruiken is. Dit principe heet 'codificatie' van kennis. Veel expertkennis blijkt echter heel lastig om op deze manier vast te leggen. Voor het delen van dit soort kennis is het beter om een andere kennismanagementstrategie te volgen. Dit noemen we 'personalisatie' van kennis, wat inhoudt dat niet de kennis zelf opgeslagen wordt maar informatie over de kennisbron, dat wil zeggen de persoon die die kennis bezit. Kennisdeling kan dan plaatsvinden door de persoon te benaderen en hiermee te communiceren. Op deze manier kan bepaalde expertise ook overgedragen worden aan collega's. Uit ons onderzoek blijkt dat architectuurkennis het meest efficiënt gedeeld wordt in een organisatie door gebruik te maken van zowel codificatie- als personalisatiemechanismen. We pleiten daarom voor hybride ondersteuning voor het delen van architectuurkennis en laten in dit proefschrift zien welke voordelen zo'n strategie heeft in de praktijk.

Daar waar architecten verantwoordelijk zijn voor het ontwerpen van een softwarearchitectuur, zijn auditors verantwoordelijk voor een kwaliteitsanalyse van het uiteindelijke softwareproduct. Voor deze taak is het ontdekken van relevante architectuurkennis in bestaande documenten en auditspecifieke informatiebronnen van belang. Immers, de auditor zal de bestaande, gedocumenteerde architectuur moeten vergelijken met de (kwaliteits)eisen van de klant – vaak door eerst concrete maatregelen ('kwaliteitscriteria') te definiëren die al dan niet in het softwareproduct aanwezig moeten zijn. Dit kennisintensieve proces wordt in de regel gekenmerkt door een overdaad aan zowel documentatie als potentiële kwaliteitscriteria.

De al eerder beschreven beslislus maakt duidelijk dat er geen strikt onderscheid gemaakt kan worden tussen 'eisen' en 'ontwerpbeslissingen'. In feite zijn eisen die invloed hebben op de architectuur namelijk ook ontwerpbeslissingen, in die zin dat ze – net als andere ontwerpbeslissingen – randvoorwaarden scheppen voor vervolgbeslissingen. Kwaliteitscriteria bepalen dus ook een architectuur, zij het als ideaalbeeld: ze beschrijven hoe de architectuur van het softwareproduct er uit zou *moeten* zien.

Er is sprake van een zekere mate van herbruikbaarheid van kwaliteitscriteria in verschillende audits. Zo zal voor een willekeurig systeem waarvoor veiligheid een belangrijke kwaliteitseis is vrijwel altijd een vorm van gebruikersauthenticatie (bijvoorbeeld middels een wachtwoord) noodzakelijk zijn. Het vóórkomen van gebruikersauthenticatie is dus een kwaliteitscriterium dat voor al deze systemen geldt.

In een auditorganisatie waar wij het hergebruik van kwaliteitscriteria hebben onderzocht, bleek zulk hergebruik in eerste instantie te bestaan uit het herlezen van eerdere auditrapporten om daarin kwaliteitscriteria toepasbaar binnen een nieuwe audit te identificeren. Om het hergebruik van kwaliteitscriteria beter te ondersteunen ondernam deze organisatie ook pogingen om een elektronische 'catalogus' van kwaliteitscriteria op te zetten, waarin echter geen ruimte was voor onderlinge relaties tussen de criteria. Op basis van een bestaande ontologie van ontwerpbeslissingen hebben wij een ontologie voor kwaliteitscriteria voorgesteld die gebruikt kan worden om kennis omtrent zulke criteria vast te leggen en vervolgens met deze kennis te redeneren, ondersteund door een vernieuwende visualisatie van ontwerpbeslissingen. Middels scenario's hebben we aangetoond hoe deze combinatie van ontologie en visualisatie de auditor bij aanvang van een nieuwe audit ondersteunt bij het ontdekken van relevante kwaliteitscriteria.

De selectie van kwaliteitscriteria is echter pas een eerste stap in een audit. Daarna zal het softwareproduct zelf moeten worden geanalyseerd, om zodoende vast te stellen of en waar het product niet aan de kwaliteitscriteria voldoet. De documentatie die bij een softwareproduct wordt geleverd is voor deze analyse een belangrijke informatiebron. Zulke productdocumentatie beslaat echter meestal tientallen tot soms wel honderden documenten, die door de auditor vaak onder enige tijdsdruk moeten worden bekeken. Het spreekt voor zich dat het onmogelijk is om alle documenten tot in de-

tail te bestuderen. Omdat een leeswijzer vrijwel altijd ontbreekt zijn auditors op hun ervaring en gevoel aangewezen om een selectie te maken tussen teksten die wel tot in detail worden geanalyseerd en teksten die slechts oppervlakkig worden gelezen of ter kennisgeving worden aangenomen.

Uit ons onderzoek blijkt dat het heel goed mogelijk is auditors bij deze selectie te ondersteunen door toepassing van *latent semantic analysis* (LSA), een vorm van geautomatiseerde tekstanalyse. Met behulp van LSA waren we in staat een interactieve leeswijzer op te stellen, waarmee auditors die documenten kunnen vinden die voor hen van belang zijn. Zo willen auditors in de regel aan het begin van een audit snel inzicht krijgen in de architectuur van een softwareproduct, om daarna langzaam maar zeker in te zoomen op verschillende onderdelen van het systeem. Onze leeswijzer wees een *factsheet* van slechts twee pagina's aan als eerst te lezen document. Deze factsheet bleek, hoewel het woord 'architectuur' er niet letterlijk in voorkwam, inderdaad een goede eerste indruk van de architectuur van het betreffende product te geven. Bovendien konden we tegemoet komen aan de wens van de auditors om langzaam af te dalen naar steeds gedetailleerdere informatie door bij vervolgsuggesties rekening te houden met de 'afstand' tussen het eerder gelezen document en nog te lezen documenten.

Om de validiteit van onze leeswijzer te bepalen, hebben we de nabijheid van documenten in het LSA-model vergeleken met de nabijheid van documenten volgens twee auditors. Hiervoor hebben we de individuele mentale modellen bepaald die deze twee auditors van de documenten hadden. Na correlatieberekeningen bleek dat er weliswaar verschillen zijn tussen het mathematische documentenmodel van LSA en de mentale documentenmodellen van de twee auditors, maar dat LSA niet meer afwijkt van de auditors dan de auditors onderling. Mensen blijken dus ook ten opzichte van elkaar op verschillende manieren naar dezelfde verzameling documenten te kijken.

In een vervolgstudie hebben we gekeken naar de verschillende mentale documentenmodellen binnen één ontwikkelteam. Hieruit kwam het beeld naar voren van een bekend kinderspelletje waarin kinderen in een ketting een woord doorfluisteren. Wie dit spel weleens gespeeld heeft weet dat het woord dat het laatste kind hoort nooit hetzelfde is als het woord dat het eerste kind werd ingefluisterd. Zoals het woord in de fluisterketting steeds een beetje verandert, zo veranderde ook het mentale model van de documenten in het ontwikkelteam. Auditors maken geen deel uit van het ontwikkelteam, en staan dus volledig buiten de fluisterketting van documenten. Juist de locatie in deze ketting lijkt het begrip van de softwaredocumentatie te bepalen. Technische oplossingen om auditors te ondersteunen kunnen dan ook niet meer zijn dan ondersteuning: een waardevol hulpmiddel, maar geen eindoplossing. Auditors moeten in contact komen met de softwareleverancier, en praten met leden van het ontwikkelteam. Het ontwikkelen van kwalitatief hoogwaardige software is niet slechts een kwestie van techniek; het is evenzeer een kwestie van mensen.