Voor veel bedrijven is het ontwikkelen van nieuwe systemen, met een op flexibiliteit gerichte architectuur, noodzakelijk om in de toekomst tijdig in te kunnen springen op wijzigende omstandigheden. Maar wat te doen met bestaande, vaak nog volop in gebruik zijnde systemen? De gedane investeringen en de impliciet in de systemen aanwezige kennis maken het volledig opnieuw beginnen een onaantrekkelijke optie. Bij de Westland/Utrecht Hypotheekbank (WUH), een kredietinstelling gespecialiseerd in de financiering van onroerende zaken, is daarom onderzoek gedaan naar de mogelijkheden om een bestaand systeem te migreren naar een nieuwe architectuur. Dit artikel is het laatste deel van een tweedelig artikel over de opgedane ervaringen in dit onderzoek. Het eerste deel is in het vorige nummer van Software Release Magazine verschenen.

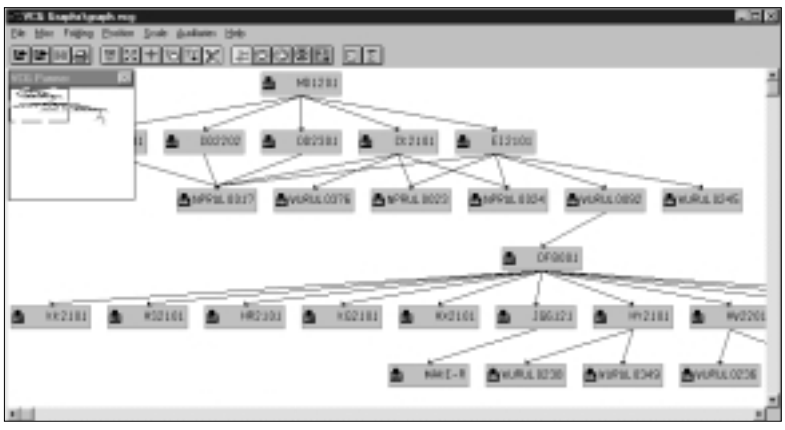
achtergrond

Migratie bij de Westland/Utrecht Hypotheekbank (2 en slot)

Noodzaak en risico's van nieuwe documentatie

Het eerste deel is geëindigd met de omschrijving van de IKV-browser, die de gegevens ontsluit uit de documentatiebase. Naast de browser is ook een aantal prototypes ontwikkeld om het abstractieproces te ondersteunen, te weten graafvisualisatie, documentatiecompilatie, dataflowanalyse en dotplot analyse. Hierna gaan we hierop dieper in.

GRAAFVISUALISATIE Een van de informatiebronnen voor het destilleren van abstracties uit de programmatuur is de call-graph, dat wil zeggen, de gegevens over aanroepen tussen de componenten onderling.



AFBEELDING 1: Visualisatie van een call-graph

De RIS repository, en dus ook de documentatie-database, bevat de basale gegevens. Voor elke component kan dus worden opgezocht welke andere componenten deze aanroept. Mede door de omvang van de IKV-programmatuur is het echter niet praktisch om deze informatie alleen tekstueel of via de IKV-browser te beschouwen. Voor een globaal overzicht zijn plaatjes handiger en de documentatieomgeving biedt de mogelijkheid deze plaatjes te genereren vanuit de IKV-browser (zie afbeelding 1).

Met de aantallen componenten in het IKV-systeem worden ook deze plaatjes echter al snel onoverzichtelijk. Om bijvoorbeeld vanuit een top-level transactie te kunnen zien welke stukken van de database worden benaderd (om bijvoorbeeld te proberen langs die weg subsystemen af te bakenen) is het nuttig om tussenliggende niveaus weg te kunnen snijden. Wat overblijft is een gecomprimeerde call-graph. Deze (gecomprimeerde) visualisatie is gerealiseerd met een (vrij beschikbaar) visualisatie-tool, genaamd VCG. Vanuit de IKV-browser wordt een (gecomprimeerde) call-graph geproduceerd als een tekstfile die door dit tool vervolgens in een daadwerkelijke graaf wordt gepresenteerd (zie afbeelding 2).

IKV

De WUH biedt een uitgebreid pakket financiële diensten aan zowel particuliere als zakelijke klanten. Eén van de informatiesystemen die deze dienstverlening ondersteunen is het IKV-systeem. IKV, wat staat voor Integratie Kredietverlening/Verzekeringen, is in de periode van 1991 tot 1994 door WUH ontwikkeld om tegemoet te komen aan de behoefte aan een verregaande flexibilisering van het toenmalige hypotheeksysteem en de integratie van kredietverlening met verzekeringen. Speerpunt bij het ontwerpen van IKV was, naast de integratie van hypotheken met verzekeringen, de introductie van een produktmodel. Dit model stelt de gebruiker van het systeem in staat on-line nieuwe produkten samen te stellen uit bestaande produktcomponenten. Het IKV-systeem is geïmplementeerd met het ontwikkeltool RIS van het bedrijf Infra Design. RIS biedt een 4GL-omgeving gebaseerd op COBOL; ontwikkelaars kunnen in een COBOL-achtige scripttaal functionaliteit beschrijven die door het ontwikkeltool wordt gecompileerd naar COBOL. Belangrijke componenten in de RIS-omgeving zijn FCT's (Functional Composition Transactions), modules waarin functionaliteit wordt beschreven in de RIS-scripttaal, en DIT's (Database Interaction Transactions), door RIS gegenereerde databasefuncties aangevuld met in RIS-script geschreven business rules. IKV bestaat uit respectievelijk 1200 en 1900 van deze FCT's en DIT's, terwijl de IKV-database ongeveer 180 entiteiten omvat.

DOCUMENTATIECOMPILATIE Elke logische documentatiecomponent heeft een tweetal omschrijvingen. Een korte omschrijving waarin in één of twee zinnen de essentie van dat component wordt verwoord, en een uitgebreide omschrijving, de zogenaamde compilatiebeschrijving. Dit is een omschrijving die is opgebouwd uit de korte omschrijvingen van alle logische (dynamische) componenten waaruit een component is opgebouwd. Dit betekent bijvoorbeeld dat de uitgebreide omschrijving van een logische functie is opgebouwd uit de korte omschrijvingen van de (abstracte) deelfuncties die binnen die logische functie worden uitgevoerd. In deze omschrijving is dus niet meer te zien welke logische functies of andere logische (dynamische) componenten verder nog worden aangeroepen (en welke functies die weer aanroepen etc.). Alle dynamische componenten van alle aangeroepen logische functies staan logisch gerangschikt zonder doublures in de omschrijving van de functie die op dat moment wordt bekeken. In afbeelding 3 wordt dit principe geïllustreerd.

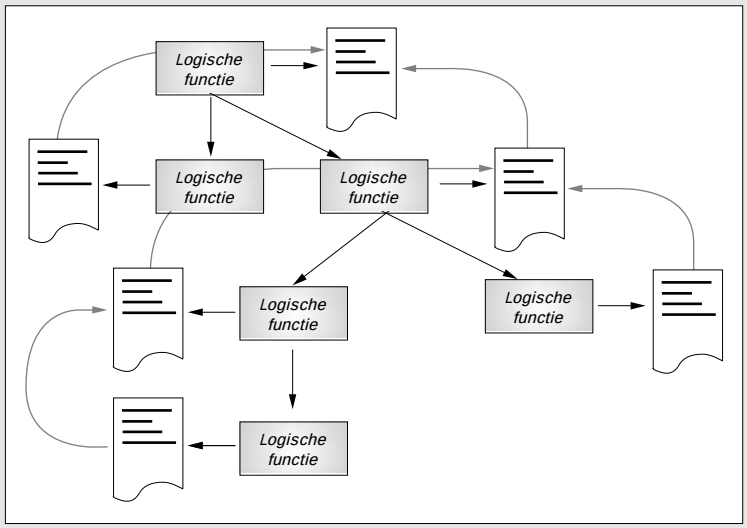
AFBEELDING 2: Visualisatie van een gecomprimeerde call-graph

Een dergelijke opbouw van de beschrijving van componenten vormt een belangrijke informatiebron voor het onderkennen van meer conceptuele componenten. Door componenten op deze manier te beschrijven wordt het ook mogelijk te onderkennen hoe en wanneer bepaalde aanvullende (business-?) functionaliteit wordt uitgevoerd ten opzichte van 'pure' datamanipulatie. Bij elke slag 'groeit' de database-interactie; door elke groeistuip afzonderlijk vast te leggen is het mogelijk precies vast te stellen welke functionaliteit altijd en welke functionaliteit afhankelijk van een bepaald proces wordt uitgevoerd.

Het hergebruik van gegevens over de ene component als onderdeel in de documentatie van een andere component moet door de documentatieomgeving ondersteund worden. De documentatieomgeving biedt een oplossing waarbij de documentatie over de gerelateerde onderdelen beschikbaar is bij het invoeren of wijzigen van de documentatie van de overkoepelende component. Via hyperlinks (die worden afgeleid uit de verbanden tussen de componenten in de database) kunnen eventueel de teksten van de onderdelen worden ingevoegd. Daarbij wordt het structurele verband (het feit dat deze tekst van een onderdeel afkomstig is) bewaard. Afbeelding 4 illustreert het mechanisme en geeft aan hoe de documentatie van een component (FCF 1) mede opgebouwd kan zijn uit de documentatie van gerelateerde componenten (FCF 2).

DATAFLOWANALYSE In de RIS repository is de callgraph tussen componenten vastgelegd. Dat betekent dat

Compilatiebeschrijving



AFBEELDING 3: In de aanroepboom wordt per logische functie een beschrijving samengesteld uit de korte omschrijvingen van de gerelateerde dynamische componenten. Deze beschrijvingen worden samengevoegd met de korte beschrijving van de in de functie zelf uitgevoerde funcionaliteit. Hierbij wordt uiteraard een logische volgorde gehanteerd en wordt onnodige redundantie vermeden. Op deze manier komt de beschrijving volledig los van de implementatie.

AFBEELDING 4: Het opbouwen van documentatie

– zonder inspectie van de programmatuur – bekend is welke FCT's en DIT's vanuit een bepaalde FCT worden aangeroepen. Wat echter niet bekend is, is welke gegevens daarbij precies heen en weer gaan.

Binnen RIS worden parameters en terugkeerwaarden gerepresenteerd door (Cobol) records. In het IKV-systeem worden dit soort records veelvuldig hergebruikt. Dat betekent dat een aanroep van een FCT weliswaar een bepaald record doorgeeft, maar dat lang niet altijd alle velden worden gezet door de aanroeper, en dat ook lang niet altijd alle velden worden gebruikt door de aangeroepene. Een speciaal geval is de situatie waarin een FCT meerdere logische functies implementeert. In dit geval zal de ene aanroep een andere set van parameters doorgeven dan een andere aanroep (en zullen ook andere resultaten worden opgeleverd).

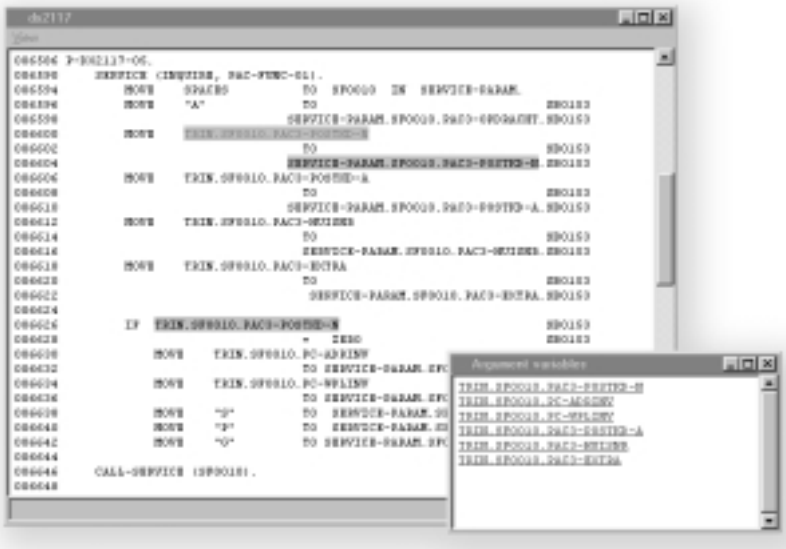
Om de data die bij aanroepen tussen componenten wordt meegegeven en opgeleverd in kaart te kunnen brengen, bevat de documentatieomgeving beperkte ondersteuning voor dataflowanalyse. De dataflowanalyse is een transitief probleem. Vanuit een aanroep naar Y door X worden weer andere componenten aangeroepen, die bijvoorbeeld effect kunnen hebben op de door Y opgeleverde resultaten. Vanuit een 'toplevel'-aanroep met bepaalde waardes kan dan een boom worden bepaald van (mogelijke) aanroepen en de stroom van gegevens daartussen.

Automatische dataflowanalyse vergt eigenlijk volledig inzicht in het executiepatroon van een programma en het gebruik van variabelen. In de context van dit project is bewust gekozen voor een beperktere aanpak, waarbij analyse van de code op basis van beperkte taalkennis plaats vindt. Het dataflowanalysetool wordt gestart op een component en toont de broncode daarvan, waarin bijvoorbeeld het gebruik van variabelen en aanroepen gemarkeerd zijn. In een apart venster worden de door de component gerefereerde parametervariabelen (in het venster 'Argument variables' in afbeelding 5) en gezette returnvariabelen getoond. Voor elke aanroep kan de gebruiker laten bepalen welke parameters in het argumentrecord worden gezet en welke updates in de code daarvóór op die parameters betrekking (kunnen) hebben. De gebruiker kan zelf aangeven welke parameters voor een aanroep echt van belang zijn, en deze gebruiken om de code van de betreffende component te analyseren door deze te openen in een nieuw venster.

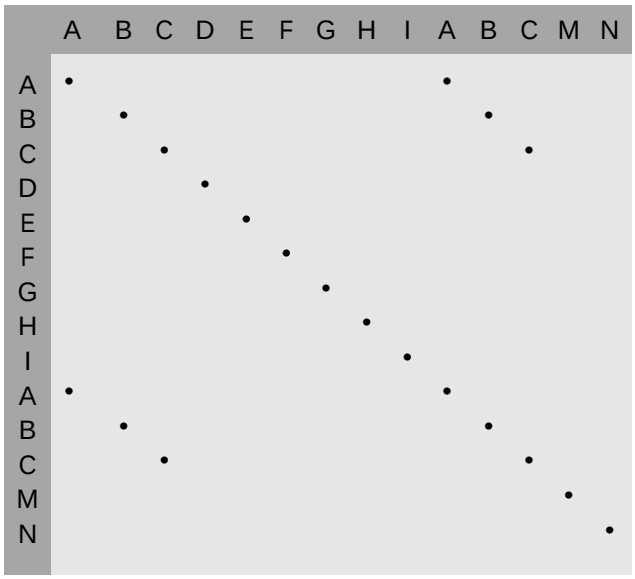
De rol van de gebruiker is cruciaal bij het gebruik van het dataflowtool voor de analyse van transitieve dataflows. Omdat onder meer conditionele executie niet wordt herkend moet de gebruiker bepalen welke van de voorgaande updates wel of niet mee moeten worden genomen bij een bepaalde aanroep. Via dit tool kan de hierboven aangegeven informatie over aanroepen en 'flow' van data tussen modules worden bepaald en vervolgens in de documentatie-database worden opgeslagen.

DOTPLOTANALYSE Het ligt voor de hand dat in een groot systeem als IKV herhaaldelijk dezelfde of vergelijkbare programmeerconstructies voorkomen. Als bijvoorbeeld dezelfde 'business rule' in verschillende componenten wordt gecontroleerd, dan is het mogelijk dat deze controles op dezelfde manier zijn gecodeerd. De documentatie-auteurs moeten dergelijke gevallen dus vinden en – op een hoger abstractieniveau – aan elkaar relateren. Voor het zoeken naar dergelijke herhaalde structuren is een prototype gemaakt van een dotplottool.

Dotplotanalyse gaat uit van twee (programma)teksten. Beide teksten vormen één dimensie in een matrix: elke regel van een tekst vormt een kolom of een rij in een matrix. Tijdens analyse worden beide programmateksten



AFBEELDING 5: Het dataflowanalyse prototype



AFBEELDING 6: Een dotplotmatrix voor een volgorde van letters

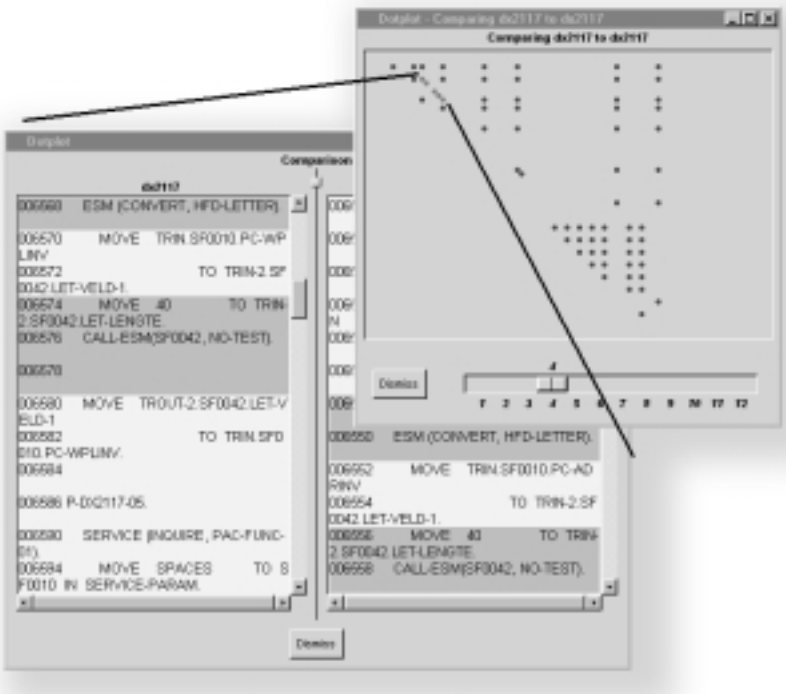
onderling vergeleken, iedere regel uit de ene tekst wordt vergeleken met iedere regel uit de andere tekst. Indien regels overeenkomen, wordt op de overeenkomstige plaats in de matrix een punt (een dot) gezet. De gehele matrix, de dotplot, geeft nu een grafisch overzicht van overeenkomstige regels in de verschillende teksten, in één oogopslag kunnen overeenkomstige regels of zelfs fragmenten worden geïdentificeerd. In afbeelding 6 is een dotplotmatrix te zien voor een vergelijking van een tekst (bestaande uit een aantal regels van ieder slechts één letter lang) met zichzelf. Doordat elke letter, en in het voorbeeld dus iedere regel, gelijk is aan zichzelf is de diagonaal van de matrix gevuld met dots.

Overeenkomstige fragmenten komen in een dotplotmatrix tot uitdrukking als patronen van punten, dotplot patterns (afbeelding 6). De diagonaal uit de dotplotmatrix voor een vergelijking van een tekst met zichzelf is hiervan een voorbeeld. Ook herhaald voorkomende tekstfragmenten vormen een patroon: gelijke codefragmenten worden in een dotplotmatrix gerepresenteerd met korte diagonale rijtjes dots, weg van de hoofddiagonaal van de matrix.

Er is slechts beperkte ervaring opgedaan met de bruikbaarheid van dotplotting. Verfijningen, waarbij bijvoorbeeld niet de tekst, maar de werkelijk uitgevoerde acties worden vergeleken, of waarbij niet een precieze 'match' maar per vergelijking een mate van overeenkomst wordt bepaald, zouden interessante aanknopingspunten kunnen bieden.

VALIDATIE VAN MODEL EN TOOL Is het inderdaad mogelijk om, met de documentatie-omgeving en het onderliggende meta-model in de hand, coding te abstraheren tot documentatie-componenten? Om antwoord te kunnen krijgen op deze vraag is geprobeerd voor een deel van het IKV-systeem het meta-model te vullen. Gedurende dit onderzoek is gebleken dat het vermogen tot abstraheren niet iedereen gegeven is, waarbij het niet vertrouwd zijn met CBD- of OO-principes verder belemmerend werkt. Uiteindelijk bleken slechts enkele projectgroepleden in staat te zijn om na vijf á zes weken oefenen een groot deel van de coding op een gelijksoortige manier te vertalen naar documentatiecomponenten. Door de grote variëteit waarmee functionaliteit is geïmplementeerd is het echter vrijwel onmogelijk om een volledig eenduidige vertaling van coding naar documentatie-componenten te realiseren. Omdat deze éénduidige vertaling van coding voor het gehele traject van wezenlijk belang is, wordt het dus noodzakelijk om het werk van de documentalisten te valideren. Hiertoe is er een tweede team samengesteld, dat zich uitsluitend bezig hield met het definiëren van meer conceptuele componenten. Het documenteren bestaat op die manier dus uit twee parallelle processen waartussen een permanente wisselwerking en informatie-uitwisseling plaats vindt.

Aan de hand van de uitgevoerde pilot blijkt op basis van het meta-model vrijwel alle gewenste informatie verkregen te kunnen worden. Door steeds verder abstraheren van code richting meer conceptuele eenheden wordt het mogelijk te bepalen waar de scheiding tussen de verschillende gewenste lagen uit de doelarchitectuur ongeveer ligt en wat kandidaten zijn voor componenten binnen deze lagen. Doordat het meta-model de verbanden tussen de abstracte componenten en de huidige implementatie vastlegt vormt het (ingevulde) meta-model zo



AFBEELDING 7: Het dotplotanalyse prototype. Door te 'klikken' op een punt worden de betreffende regels code gemarkeerd in twee tekstvensters die de broncode tonen.

een basis voor hergebruik van het bestaande IKV-functionaliteit binnen een nieuwe architectuur. Aangevuld met de bestaande documentatie biedt dit op zich een uitstekend uitgangspunt voor een beheersbaar migratietraject.

Er kunnen ook een paar kanttekeningen worden geplaatst bij de verkregen resultaten. Doordat het zeer lastig is de semantiek van de documentatiecomponenten uit het meta-model uit te drukken in eigenschappen van de bestaande programmatuur, kan een groot gedeelte van de vertaling van de coding naar documentatiecomponenten nog onvoldoende geautomatiseerd worden ondersteund. De in dit project ontwikkelde prototypes bedekken slechts een deel van de technieken zoals die bekend zijn uit reverse engineering-onderzoek, maar geven wel een representatief beeld van het soort ondersteuning dat reverse engineering-technieken momenteel kunnen bieden: hulp bij het begrijpen van programma's. Om programma's te kunnen begrijpen zijn overzichten nodig zoals call-graphs, dataflowanalyses en zelfs dotplots. De documenatieomgeving kan daarmee hulp bieden voor het afleiden van informatie uit de code en het presenteren van overzichten, maar het echte abstraheren, juist het moeilijkst onderdeel van het documentatieproces, lijkt toch voornamelijk mensenwerk. Mensenwerk kost echter tijd, wat in een dynamische ontwikkelomgeving lijdt tot het introduceren van allerlei extra procesmatige maatregelen om te voorkomen dat reeds gedocumenteerde coding niet ongemerkt wordt gewijzigd.

Het automatisch kunnen abstraheren blijft een open probleem

Ook het feit dat het meta-model gebaseerd is op een ander paradigma dan binnen de ontwikkelorganisatie gebruikelijk is, leidt tot een probleem; niet elke ontwikkelaar kan dusdanig met meer componentgerichte ideeën omgaan dat het mogelijk wordt een link te leggen tussen documentatiecomponenten en code. Dat heeft gevolgen: er is ofwel een permanente vertaalslag van de lopende projecten naar de opgeleverde documentatie nodig, ofwel er vindt een paradigmswitch plaats binnen de gehele ontwikkelafdeling, ofwel er wordt een directe migratie uitgevoerd van ieder afgebakend deelsysteem. De laatste twee opties betekenen een andere manier van ontwerpen en ontwikkelen (én een verandering in de bestaande ontwikkelprocessen), waarbij de laatste optie ook nog eens de nodige ontwikkelcapaciteit vergt.

Het is duidelijk dat een aparte documentatiedatabase voor migratiedoeleinden een synchronisatieprobleem introduceert met lopende onderhoudsactiviteiten. Wijzigingen in de IKV-programmatuur leiden tot veranderingen in de RIS repository en deze moeten vervolgens weer in de documentatiedatabase worden opgenomen. Het technische aspect van deze synchronisatie is relatief eenvoudig op te lossen, door vanuit de repository wijzigingen door te sturen naar het documentatiesysteem. Echter, het functionele synchronisatieprobleem is daarmee niet opgelost. Als de programmatuur wijzigt moeten ook de hogere abstractieniveaus in de documentatie worden aangepast. In feite moet dit gebeuren tijdens de ontwikkeling van de wijziging, dus door de ontwikkelaars. Het feit dat een herdocumentatietraject niet zomaar los te koppelen valt van het reguliere onderhoud kan dan ook een serieuze hindernis vormen.

OPEN PROBLEEM Dit alles betekent dat een volledig documentatietraject zowel organisatorisch als procesmatig een niet geringe impact heeft op een automatiseringsafdeling. Recente ontwikkelingen binnen de ING-groep hebben bovendien de noodzaak van een IKV-migratie behoorlijk op losse schroeven gezet. Er wordt gekeken naar de mogelijkheden om één hypotheeksysteem binnen de groep te ontwikkelen waar iedere backoffice gebruik van kan maken. Binnen een dergelijk visie kan er natuurlijk geen sprake zijn van het uitvoeren van een risicovol documentatieproject voor het eigen hypotheeksysteem. Uiteindelijk waren er voor de WUH dan ook redenen genoeg om de migratie van IKV vooralsnog niet te continueren. Dat maakt de conclusie van het onderzoek echter niet minder bruikbaar: de huidige manier van ontwerpen en de manier van documenteren daarvan bieden onvoldoende informatie om structurele architectuurveranderingen mogelijk te maken. Zij zijn nog te zeer gericht op het oplossen van de huidige problematiek en het overdragen van kennis gedurende het ontwikkeltraject. Reverse engineering-technieken kunnen deze tekortkoming slechts gedeeltelijk compenseren door begrip van de programmatuur te ondersteunen. Het automatisch kunnen abstraheren (een cruciale stap in een migratietraject als adequate documentatie ontbreekt) blijft een open probleem.

George Starke
is projectmanager bij de Westland/Utrecht Hypotheekbank.

Gert Florijn en Matthijs Maat
zijn werkzaam bij het Software Engineering Research Centre te Utrecht.

Literatuur op pagina 48

Vervolg van pagina 22

Referenties

[Bakker99] G. Bakker, Conservatisme beheerst de IT-wereld, in Automatiseringsgids, 12 maart 1999.

[Brand97] M.van den Brand, P. Klint, C. Verhoef, Reverse Engineering and System Renovation - An Annotated Bibliography, in ACM SIGSOFT Software Engineering Notes 22 (1997), nummer 1, pagina 57-68.

[Brodie95] M.L. Brodie, M. Stonebraker, Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach, Morgan Kaufman Publishers Inc., San Francisco, 1995.

[Chikofsky91] E.J. Chikofsky, J.H. Cross II, Reverse Engineering and Design Recovery: a Taxonomy, in IEEE Software 7 (1990), nummer 1, pagina 13-17.

[Gall95] H.C. Gall, R.R. Klösch, R.T. Mittermeir, Architectural Transformation of Legacy Systems, International Conference on Software Engineering (1995), Technical report number CS95-418, Seattle, 1995.

[Geest91] T. van der Geest, RIS II 4GL omgeving voor Unisys A-series – Garantie voor minimum aan onderhoud, in Database Magazine, november 1991.

[Geest93] T. van der Geest, Hypotheekmarkt in beweging – Produktmodellen automatiseren met verandering als uitgangspunt, in Bank en Technology, maart 1993.

[Helfman96] J. Helfman, Dotplot Patterns: A Literal Look at Pattern Languages, in Theory and Practice of Object Systems 2 (1996), nummer 1, pagina 31-41.

[Kennedy81] K.W. Kennedy, A survey of data flow analysis techniques, in (S.S. Muchnik en.D. Jones, editors) Program Flow Analysis: Theory and Applications, Prentice-Hall, 1981, pagina 5-54.

[Krikhaar99] R.L. Krikhaar, R.P. de Jong, J.P. Medema, Architecture Comprehension Tools for a PBX System, in (P. Nesi en C. Verhoef, editors) Proceedings of the third European Conference on Software Maintenance and Reengineering, IEEE Computer Society, Los Alamitos, 1999, pagina 31-39.

[Kristen93] G. Kristen, KISS-methode voor Object Oriëntatie: van Informatie Architectuur naar Informatie Systeem. Academic Service, Schoonhoven, 1993.

[Moonen97] L. Moonen, A Generic Architecture for Data Flow Analysis to Support Reverse Engineering, in (A. Sellink, editor) Proceedings of the Second International Workshop on the Theory and Practice of Algebraic Specifications (ASF+SDF'97), Springer-Verlag, november 1997.

[Sander95] G. Sander, VCG – Visualization of Compiler Graphs, technisch rapport A01-95, Universität des Saarlandes, Saarbrücken, 1995.

[Vogt94] H.H. Vogt, P.R.H. Hendriks, Code-analyse in de Praktijk, in Informatie 36 (1994), nummer 12, pagina 764-770.

[Zaal93] R. Zaal, Op weg naar de produktmodel-interpreter, in Informatie Management, april 1993.