Handelshögskolan
VID GÖTEBORGS UNIVERSITET
Institutionen för informatik

2005-01-21

Ettor och nollor byter bostad

-Datamigration från arvssystem till Data Warehouse-

Abstrakt

Dagens organisationer är i ständigt behov av rapporter, sammanställningar och analyser för att underlätta för organisationsledningen att fatta strategiska och taktiska beslut. I allmänhet är den data som finns i organisationer väldigt rörig och oorganiserad. Det är därför svårt att kunna få fram information som är tillförlitlig och korrekt. Ett sätt att underlätta för organisationer att kunna få fram dessa rapporter är att bygga ett Data Warehouse, en samlingsplats för företagsdata specialanpassad för företaget. När man skall bygga ett Data Warehouse är det ofta nödvändigt att migrera data från ett system till ett annat vilket ofta är en allt annat än smärtfri process. Denna uppsats kommer att beskriva problematik som kan uppkomma vid migration av data från ett arvssystem till ett Data Warehouse. För att belysa dessa problem har en tre månader lång etnografisk undersökning genomförts på ett företag som haft detta behov. Data har hämtats hem från ett arvssystem, och ett Data Warehouse har med denna data byggts upp som grund för en rapportgenerator för försäljningsstatistik. Problematik, lösningar och moment som man bör tänka på eller som borde ha gjorts annorlunda diskuteras i uppsatsen. De viktigaste slutsatser som dragits av de fynd som gjorts i undersökningen är att det initialt bör läggas mycket tid på att förstå vad de olika databaserna har för struktur, samt få förståelse för de rent tekniska skillnaderna i systemen. Stor vikt bör även läggas på att i ett tidigt skede involvera organisationens affärsspecialister för att underlätta urvalet av data för migrationen.

Nyckelord: Datamigration, Data Warehouse, Databaser, AS/400, Arvssystem

Författare: Fredrik Gisslin, Magnus Helmvee, Clas Peterson
Handledare: Johan Magnusson
Magisteruppsats, 20 poäng

Förord

Det är många personer som har bidragit till att vi kunnat genomföra denna uppsats. Det är dock framför allt tre personer vi vill tacka. Först och främst vår handledare *Johan Magnusson* på Institutionen för Informatik vid Göteborgs Universitet vars entusiasm, engagemang och råd varit ovärderliga för oss. *Patrik Björkborg* på Yamaha Scandinavia AB för hans stöd under utvecklingen av det system som utgjort vårt laboratorium under studien. Slutligen ett tack till *Johan Ekeroth* på Yamaha Scandinavia AB, HiFi som möjliggjorde att projektet över huvudtaget kunde utföras.

Innehållsförteckning

Introduktion

I takt med det ökande användandet av datorer och databehandling så ökar även behovet av att effektivt lagra data (Finnegan, 1999). Traditionellt sett har data lagrats i så kallade databaser, där data på ett mer eller mindre strukturerat sätt placerats för att i senare skede hämta eller manipulera den. De första databaserna var relativt snabba men inte så effektivt strukturerade som de skulle kunna vara (Connolly & Begg, 1998). Samma data kan till exempel vara nödvändigt att placera på ett flertal ställen, så kallad redundans. Strukturen i denna typ av databaser blev ofta mycket komplex och svåröverskådlig vilket ställde till med problem vid behov av att snabbt få fram viss typ av information (Bontempo 1998). En ny typ av databas eftersträvades där man snabbare kunde få fram önskad information och effektivare administrera informationen i databaserna.

Tanken om *relationsdatabaser* föddes här. I en relationsdatabas ges möjlighet att koppla ihop information placerad i olika tabeller på ett helt annat sätt än vad som tidigare varit möjligt. Genom att uppdatera en tabell i en databas uppdaterades flera andra tabeller samtidigt. På detta sätt skulle redundansen minska och dessutom ge en helt annan överskådlighet över databasen. Nackdelen med denna typ av databashanteringssystem är att den kräver mer processorkraft än motsvarande arvssystem för att få fram samma mängd information (Date, 2000). Detta har lett till att de tidiga databaserna konkurrenskraftigt kunnat leva kvar en bra tid efter de första relationsdatabasernas tillkomst. Först på senare år, från mitten av 1990-talet, har processorerna blivit så snabba att implementation av relationsdatabaser blivit ett reellt alternativ.

Relationsdatabaserna var effektiva och ett klart steg i rätt riktning, men behov fanns av än mer effektiva sätt att få fram information och analysera olika affärsdata (Bontempo, 1998; Taha, 1997). Detta ledde fram till ytterligare ett sätt att organisera data, så kallade Data Warehouses. I ett Data Warehouse är data strukturerad specifikt för att snabbt och effektivt kunna leverera önskad information till användarna. Vanligtvis sker detta genom att frågor ställs genom det vedertagna databasspråket *Structured Query Language* (SQL), även om detta ofta är dolt för användaren som istället ser någon typ av frågeformulär.

Rapporter presenteras sedan i önskad, snygg och prydlig form. Data Warehouses är alltid, om de är utformade optimalt, specialanpassade efter den organisation där systemet skall användas (Ladaga, 1995; Myers, 1995). Det finns däremot ingenting som begränsar strukturen i ett Data Warehouse till en specifik typ av databas som exempelvis relationsdatabas eller arvssystem, eller hur datan i ett Data Warehouse skall lagras. Det är dock viktigt att strukturen är homogen, det vill säga att flera olika strukturer inte blandas (Marakas, 2003). Tanken med Data Warehouses är att de endast skall innehålla information som man tittar på. Det är inte tänkt att denna information skall uppdateras, data i ett Data Warehouse är alltså framför allt historisk data. En stor anledning till att Data Warehouses utvecklas är för att öka möjligheten för ledningen i

ett företag att fatta bättre beslut. Det skall alltså innehålla information som på ett bra sätt kan presenteras för användaren så att han kan dra slutsatser utifrån vad som bör göras inom de områden som Data Warehouse behandlar.

Ett centralt begrepp i denna uppsats är *arvssystem*. Arvssystem används ofta som ett samlingsbegepp för att beskriva gamla, otidsenliga datasystem som ”hängt kvar” i organisationen sedan tidigare, organisationen har *ärvt* systemet. Arvssystem innehåller följaktligen ofta de första varianterna av databassystem. Att flytta över, *migrera,* data från en databas implementerad i ett arvssystem till ett Data Warehouse är en process som ofta är allt annat än smärtfri. Ett flertal avhandlingar vittnar om stora problem vid migration av data i allmänhet och migration från arvssystem i synnerhet (Mullins, 1997; Zagelow, 1998). Ett av huvudproblemet är de fundamentalt olika strukturer de olika systemen är organiserade med.

Problemområde

För att undersöka vilka typer av problem som kan uppstå vid migration av data från arvssystem till Data Warehouses har vi för ett företags räkning byggt ett Data Warehouse med ett tillhörande rapportsystem. Företaget vi valt som bas att utföra vår undersökning på ingår i en stor elektronik- och musikinstrumentkoncern som har sitt huvudkontor i Japan. En mängd dotterbolag runtom i världen har i dagsläget all respektive information lagrad lokalt på varje kontor. Företaget skall nu genomgå en omfattande omorganisation och all information kommer att flyttas till huvudkontoret i Japan. Detta innebär i sin tur att alla dotterbolags data kommer att placeras på en central server på huvudkontoret.

Begränsad bandbredd på den förbindelse som finns till Japanservern tillsammans med det faktum att ett antal europakontor kommer att dela på samma server och processorkraft medför ett behov av att mer effektivt sätt att organisera företagets och de olika dotterbolagens data.

En lämplig lösning i ett fall som detta kan vara just att bygga ett Data Warehouse, vilket - om det är väl utfört - reducerar flera av de negativa konsekvenser en sådan flytt kan medföra (Love, 1996; Park, 1997). Detta gav oss ett ypperligt tillfälle att kunna undersöka och ge svar på det som definieras i syftet.

Eftersom flytten till Japan kommer att påverka prestandan i företagets system är det nödvändigt att åter kopiera så stor mängd data som möjligt från Japan till de olika dotterbolagens servrar. Samtidigt är det dock lika viktigt att inte hämta mer än nödvändigt med tanke på den begränsade bandbredden till Japanservern. Flera europakontor arbetar ofta vid samma tillfälle mot servern vilket givetvis påverkar den totala prestandan.

Företagets huvuddatabas i Japan är IBM AS/400. Lokalt används i Göteborg sedan tidigare en Microsoft SQL Server 2000-databas och om möjligt ville man fortsätta

använda denna även för det nya Data Warehouse vi fick i uppdrag att utveckla. Inspiration, kunskap och hjälp har vi fått från flera källor. Vad gäller problem som uppstått vid utvecklandet av Data Warehouset och rapportgränssnittet har vi hämtat kunskap hos personalen på företaget, deras externa konsulter, litteratur som berör Data Warehouse, SQL-, och VB-programmering, supportavdelningen på IBM samt Internet.

För att beskriva Data Warehouse och beskriva och jämföra olika teorier och synsätt har huvudsakligen litteratur i ämnet använts. Även Internet och då framför allt vetenskapliga databaser som ACM och Science Direct har varit till stor hjälp för oss.

Syfte

Syftet med denna uppsats är att undersöka och beskriva vilka problem som kan uppstå vid migration av data från ett arvssystem till ett Data Warehouse.

Avgränsning

Begreppet migration inom IT täcker ett brett område och avgränsningar har därför av nödvändighet satts för hur mycket denna uppsats tar upp vad gäller det undersökta området. Genom att bygga ett Data Warehouse och rapportsystem har vi definierat det laboratorium vi använder för våra empiriska undersökningar. Dessa undersökningar ligger till grund för att uppnå det definierade syftet i denna uppsats. Uppsatsen beskriver endast de problem som uppstått under undersökningen och går inte in på exempelvis olika algoritmer som används inom området.

Målgruppen för denna uppsats är personer med IT-relaterat intresse i allmänhet och personer som har för avsikt att tillämpa en migration från ett arvssystem till ett Data Warehouse i synnerhet.

Definitioner

Ad-hoc: - *spontanitet* eller *slumpvis*

Arvssystem: Det finns ett flertal definitioner gällande arvssystem eller *Legacy Systems* som är den engelska termen för denna typ system. Gemensamt för dem är att de beskriver någon typ av äldre system. Den definition som valts som bäst beskriver arvssystem i denna uppsats kontext är den som Juric et al (2000) ger: *Any system that, regardless of age or architecture, conforms to the following conditions:*
• has existing code,
• is useful today,
• is in use today,
• the architecture of the system is not distributed and object oriented.
Accordant to this definition there are two big sets of legacy systems:

• traditional mostly mainframe based systems and
• systems written in modem languages (C, C++, Smalltalk, etc.), PC-based systems, client/server systems and personal productivity tools.

CSCW (Computer Supported Cooperative Work): - Datorstött samarbete.

Databas: - *Mängd av data, ordnade i ett eller flera dataregister, som är tillräcklig för ett visst ändamål eller för ett visst databehandlingssystem* (Nationalencyklopedin.se).

Data Warehouse: - *A data warehouse is a copy of transaction data specifically structured for querying and reporting*" (Kimball, 1998).

Denormalisering - Innebär att normaliserade tabeller slås ihop till färre tabeller för att öka prestanda vid sökning i tabellerna.

Distribuerad databas: - *En samling data som hör samman på något sätt, och där dessa data är fysiskt utspridda på flera olika datorer som är sammankopplade med ett datornät* (lida.liu.se)

DTS: - *Microsoft® SQL Server™ 2000 Data Transformation Services (DTS) är en uppsättning grafiska verktyg och programmerbara objekt som medger extrahering, transformering och konsolidering av data från vitt skilda källor som exempelvis ett arvssystem till ett (eller flera) destinationer* (Microsoft.com).

Empiri: - *Något som är grundat på erfarenhet* (Nationalencyklopedin.se).

ERP (Enterprise Resource Planning): - Ett affärssystem som integrerar alla delar i ett företag som till exempel planering, marknadsföring, försäljning med mera.

ETL-verktyg: - *Ett ETL-verktyg är ett program som överför och omvandlar stora mängder data från olika källor som databaser, finans- och affärssystem till användbar samlad information som kan presenteras på ett ställe* (Computer Sweden.se).

Metadata: - Data som innehåller beskrivning om annan data.

Migration: - *En process som innebär överförning av data från ett format till ett annat. Datamigration är nödvändigt när en organisation beslutar sig för att byta till ett nytt data- eller databashanteringssystem som inte kompatibelt med det tidigare systemet. Vanligtvis utförs datamigrationen genom en uppsättning av anpassade program eller script som automatiskt överför datan* (Webopedia.com).

Mikrokod: - *I datorsammanhang detsamma som mikroprogram, det vill säga grundläggande programnivå hos CISC-datorer (Complex Instruction Set Computers) där maskininstruktionerna tolkas i elementära operationer som kan utföras av elektroniken* (Nationalencyklopedin.se).

Normalisering: - *En teknik för att producera en samling av relationer med önskade egenskaper i enlighet med kraven från en organisation* (Connolly & Begg, 1998).

ODBC (Open Data Base Connectivity): - *En standardiserad metod utvecklad av SQL Access Group 1992 för att komma åt information i en databas. Målet med ODBC är att möjliggöra åtkomst av vilken typ av data som helst från vilken applikation som helst oavsett vilket databasprogram som används. ODBC sköter detta genom att agera mellanlager, en så kallad databasdrivrutin, mellan en applikation och databasprogrammet. Syftet med detta är att översätta de frågor som sker från applikationen till kommandon som databasapplikationen förstår* (Webopedia.com).

OLTP (Online Transactional Processing): - *Ett databashanteringssystem som hanterar organisationernas dagliga transaktioner (Connolly & Begg, 1999).*

Positivism: - Vetenskapligt synsätt där den absoluta sanningen kan fås genom logiska härledningar. Vanligtvis använt inom naturvetenskap och teknologi.

Reverse Engineering: - *Innebär att bryta isär något för att se hur det är uppbyggt och för att se hur det fungerar, för att senare kunna kopiera eller förbättra objektet. Detta är en gammal teknik som länge använts inom industrin. Till exempel en bilfabrikant som köper en konkurrents bil för att kontrollera svetsningar, hopmontering och andra lösningar för att kunna bygga en ännu bättre bil men fortfarande byggd med liknande delar. Denna teknik har anammats av informationsteknologin och används ofta numera inom datorhårdvaru- och mjukvaruproduktion.* (searchsmallbizit.techtarget.com).

Stream Files - *En Stream File är en slumpmässigt åtkomlig sekvens av bytes, utan vidare inverkan från systemet vad gäller struktur. Stream Files och databaspostfiler är olika strukturerade. Denna skillnad på struktur påverkar hur dessa filer används. Strukturen påverkar till exempel hur applikationer skrivs för att interagera med filerna och hur de olika typerna av filer används bäst i en applikation* (IBM.com).

Disposition

Denna uppsats är indelad i sex huvudkapitel (figur 1.1). Initialt beskrivs i *Introduktion*s-kapitlet (sid. 3) de problem som kommer att diskuteras och i möjligaste mån kommer att lösas. En bakgrund ges även till det företag som är vår uppdragsgivare vad gäller att konstruera rapportsystemet och Data Warehouset. Därefter beskrivs i *Metod*-kapitlet (sid. 10) vilka metoder som användes och varför vi valt att använda just dessa. I efterföljande avsnitt *Teori* (sid. 18) tas de teorier som ligger till grund för uppsatsen upp och det redogörs även här för problem inom liknande område som andra undersökningar beskrivit. I *Resultat*-kapitlet (sid. 35) presenterars de fynd som gjorts under den tre månader långa undersökningen. I denna del anknyts även redovisade fynd till de teorier som ligger till grund för uppsatsen. Därefter följer ett *Diskussions*-kapitel (sid. 49) där det mer i detalj diskuteras implikationer, alternativa upplägg och möjliga tolkningar av teorier och undersökningsresultat. Till sist presenteras de *slutsatser* (sid. 52) som dragits och de lärdomar som vi fått av denna undersökning sammanfattas.

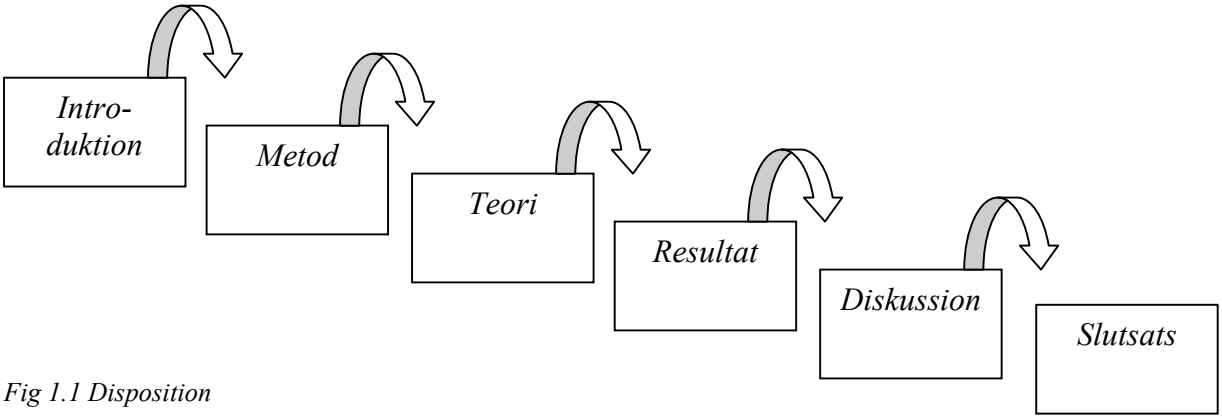


Fig 1.1 Disposition

Metod

Vid varje undersökning eller forskning ligger oftast ett problem till grund för varför man avser att undersöka just det specifika området. Problemen kan vara av många olika karaktärer och det krävs därför en bra undersökningsmetod som kan hjälpa forskaren att behandla problemen på bästa tänkbara sätt. Metoden skall inte väljas efter vilket resultat man önskar att få av undersökningen utan efter problemets karaktär och att just den valda metoden kan ge det mest objektiva resultatet av det undersökta problemet. Ett bra val av metod kännetecknas av att om någon annan avser att studera samma fenomen skall denne kunna göra detta med samma metoder och få samma resultat som den tidigare studien. Vid valet av metod och för upplägg av uppsatsen har vi använt oss av boken *Rapporter och uppsatser* (Backman, 1998) och *Management Research* (Easterby-Smith m. fl., 1991).

Förstudie

Innan vi påbörjade vår studie genomförde vi en förstudie för att utvärdera vilken undersökningsmetod och vilka eventuella förkunskaper vi skulle behöva för att genomföra studien. I vår förstudie hade vi två möten med beställarna av projektet som vi skulle utföra studien på för att försöka skaffa oss en klarare bild av vad projektet verkligen innebar. Efter avslutad förstudie ansåg vi oss fått en bra bild av hur vi skulle strukturera undersökningen och vilken metod vi skulle välja och vilka teoretiska förkunskaper som vi skulle behöva.

Vi kom fram till att vi skulle använda oss av en etnografisk undersökningsmetod som var anpassad för studier av objekt inom informatik som forskningsdisciplin. Vi valde att använda oss av *Concurrent Ethnography* som är en av de vanligaste metoderna som används vid en etnografisk studie av design av ett system (Burke & Kirk, 2000). Vår studie skulle utföras vid ett experimentellt designarbete som innebar migrering av data och utvecklandet av en specialiserad rapportgenerator. Den etnografiska studien syftade till att ge oss en mycket bred och omfattande empirisk plattform som grund till en omfattande analys. Vi skulle erhålla våra empiriska fynd genom fältanteckningar och vid interaktion med beställare och tilltänkta användare av det utvecklingsprojekt vi utförde vår studie på.

Vetenskapsteori

Vetenskapsteorier är filosofiska synsätt på vad som kan uppfattas som sanning. Anhängare av olika vetenskapsteorier har olika syn på vad de menar med absolut sanning. Just valet av vetenskapsteori kan ställa till det för en forskare från forskningsdisciplinen för informatik eftersom informatik som ämne inte har någon riktigt entydig och samstämmig definition. Det enda som de flesta definitioner av ämnet har gemensamt är att de hävdar att informatik handlar om användandet av

informationsteknologi. Användandet betecknas som något humanistiskt medan teknologi är mycket positivistiskt (se definitioner). Dessa två olika teoretiska synsätt betraktas oftast som motsatser till varandra (Thurén, 1991). Detta dilemma kan samtidigt betyda en stor möjlighet för forskaren inom denna disciplin, ämnet handlar om både användande och teknologi. En forskare inom informatik får den svåra men behagliga uppgiften att kombinera de båda och utifrån detta skapa ett resultat som kan anses som objektivt.

Vi har valt att använda oss av ett fenomenologiskt hermeneutiskt vetenskapsteoretiskt synsätt, vilket beskrivs senare i texten. Detta för att vår undersökning grundar sig på empiri som vi har fått genom iakttagelser och experiment. Den är fenomenologisk eftersom vi inte kan utesluta våra egna värderingar och åsikter, då vi anser att omvärlden är en social konstruktion och vi som ska undersöka den är en del av den. Detta val styrks av att vi har fått vår empiri genom iakttagelser och genom olika experiment som vi har granskat med våra sinnen. Den är även hermeneutisk eftersom vi inte kan utelämna tidigare kunskaper eller förförståelse för det objekt vi avser att studera.

Hermeneutik

Hermeneutik är en humanistisk tolkningsmetod. Denna metod innebär att sanning inte fås endast genom de fem sinnena och vår logiska intelligens. Forskaren tillåts även att använda sina egna värderingar och erfarenheter för att få en förståelse som denne senare kan beteckna som sanning. Alltså måste en hermeneutisk forskare ha förförståelse för det fenomen som denne avser att studera. Med förförståelse menas att denne har en uppfattning eller erfarenheter om de levande och handlade individerna som ingår i det fenomen som forskaren avser att studera. Forskaren kan identifiera sig med de situationer som finns inom det fenomen som denne avser att undersöka. Eftersom hermeneutiken är en utpräglad humanistisk vetenskapsfilosofisk inriktning har den oftast mer förståelse för relativistiska synsätt än positivismen. En positivistisk forskare har mycket svårt att godkänna en undersökning gjord efter hermeneutisk vetenskapsfilosofi eftersom den positivistiska forskaren anser att det bara finns absoluta sanningar. Dessa sanningar kan endast fås genom iakttagelser genom våra fem sinnen vilka sedan tolkas genom vår logiska intelligens. Hermeneutiken däremot menar att det inte finns några absoluta sanningar (Thurén, 1991).

Eftersom hermeneutiken handlar om tolkning av innebörden i symboler, handlingar, texter (dokument) och upplevelser, är detta synsätt det rätta för en studie som skall göras i enlighet med den etnografiska forskningsmetoden.

Fenomenologi

Fenomenologin introducerades vid början av 1900-talet av den tyska matematikern Edmund Husserl (1859-1938) som lämnade matematiken för kunskapsfilosofin. Den

fenomenologiska läran blev snabbt känd under slagordet ”till själva saker”. *Fenomen* betyder ”saken så som den visar sig”. Gunnarsson (2002) menar att Husserls bidrag till kunskapsfilosofin var att säga att ”människan erfar världen genom att erfara ett fenomen”. Vill vi veta något måste vi gå till fenomenet.”

Den fenomenologiska läran innebär att forskaren inte kan utelämna sin egen påverkan på det studerade objektet eftersom forskaren i enlighet med läran är en del av det fenomen som denne avser att studera. Världen är en social konstruktion som skapats av mänskligheten och forskaren är därmed en del av den. Målet med fenomenologin är att man skall eftersträva att utan förvrängning presentera upplevelsen av fenomenet så som det visar sig i världen. Eftersom den som utför studien av fenomenet är en del av fenomenets livsvärld kan detta dock ha viss påverkan på studiens resultat. I enlighet med fenomenologin skall forskaren sträva efter att utelämna sin egen förförståelse för det studerade objektet. Med detta synsätt är dolda budskap inte intressanta medan de i hermeneutiken i högsta grad är intressanta. När vi valde att kombinera dessa vetenskapliga filosofier var det för att vi inte kunde utesluta tankar ifrån de båda utan endast kunde välja ett synsätt som kombinerade dem.

Induktion – Deduktion

När man skall dra slutsatser finns det två huvudsakliga angreppssätt, *induktion* och *deduktion*, och en korsning mellan dem, den *hypotetiskt deduktiva metoden*. Induktion bygger på empiri vilket innebär att allmänna, generella, slutsatser dras utifrån empiriska data. Induktion förutsätter alltså kvantifiering. Det vill säga när man drar en induktiv slutsats, bygger denna på att det finns en premiss som säger att massan har en viss egenskap och därför blir slutsatsen att alla som ingår i massan har denna egenskap (Thurén, 1991). När induktion används som slutledningsmetod är det viktigt att tänka på två faktorer, validitet och reliabilitet. Reliabilitet eller tillförlitlighet innebär att mätningarna är korrekt utförda. Detta kan innebära att exempelvis en opinionsundersökning måste bygga på ett representativt urval av personer så att inte tillfälligheter påverkar resultatet. Om flera liknande undersökningar utförs av olika undersökare och kommer fram till samma resultat, betyder detta att undersökningen har hög reliabilitet. Validitet innebär att det som önskades undersöka verkligen har undersökts och inget annat. Detta betyder då att om det som skall undersökas är vad folk ser på tv skall man inte undersöka vilken tv de tittar på. Alltså kan validitet betecknas som relevans, vad som kan vara relevant för undersökningen.

Exempel på induktion:
Premiss: Vid en marknadsundersökning använde majoriteten Microsoftprodukter.
Slutsats: Alltså är Microsoft marknadsledande.

Det är dock viktigt att beakta att en induktiv slutledning aldrig kan vara hundraprocentig eftersom den bygger på empiriskt material som sällan utgör en fullständig uppräkning. Induktiva slutledningar grundade även på enorma mängder material kan visa sig vara falska (Thurén, 1991).

Deduktion grundar sig däremot på logik. Alltså innebär deduktion att en logisk slutsats dras som betraktas som giltig om den är logiskt sammanhängande. Slutsatsen behöver dock inte vara sann i den meningen att den överrensstämmer med verkligheten. En deduktiv slutledning kan se ut enligt följande:

Exempel deduktion:
Premiss: Björn stannar hemma när det regnar
Premiss: Det regnar
Slutsats: Alltså är Björn hemma

Man bör dock beakta att man vid deduktion kan få en logisk sanning som inte behöver vara en verklig sanning eftersom den logiska sanningen enbart grundar sig på de uppställda premisserna och verkligheten kan se annorlunda ut. Ett exempel på detta kan vara:

Premiss: Alla bilar är röda.
Premiss: Jag har en bil.
Slutsats: Alltså är min bil röd

Enligt premisserna skulle denna deduktiva slutsats vara en sanning eftersom en deduktions giltighet inte har något att göra med om premisserna är sanna eller inte, medan verkligheten ser annorlunda ut. Detta beror på att det inte har undersökts om premisserna är sanna utan man har endast antagit dem vara sanna. Slutsatser har sedan dragits utifrån dessa premisser. Därför bör man vara lite misstänksam mot alltför logiska resonemang, de kan få en att glömma att undersöka om premisserna stämmer med verkligheten (Thurén, 1991).

Den hypotetiskt deduktiva metoden bygger på att hypoteser och observationer ställs upp som premisser och utifrån dessa premisser görs sedan en deduktiv slutledning. Slutligen jämförs premisserna med verkligheten. I och med att denna metod kombinerar de två olika metoderna att dra slutsats kommer både empiri och logik att användas. Då detta angreppssätt kombinerar de båda har vi valt att använda oss av det för vår undersökning. Ett exempel på hypotetisk deduktion kan vara när vi undersökte ett databassystem och försökte finna vad problemet med prestandan berodde på. Vi ställde premisserna:

1. (Hypotes) För många användare på servern
2. (Observation) Servern har för litet primärminne
3. (Hypotes) SQL frågorna är för omfattade för servern
4. (Observation) Databastabellerna har för många rader
5. (Observation) Databasen är inte optimerad
6. (Hypotes) Servern körs på en vanlig PC
7. (Hypotes) Applikationer körs samtidigt på servern

Utifrån dessa premisser drogs slutsatsen att prestandaproblemen berodde på att ett omfattande databassystem inte bör köras på en vanlig PC där någon samtidigt arbetar med andra applikationer. Efter att vi dragit slutsatsen, gick vi tillbaka och testade varje premiss för att se om de stämde med verkligheten. Vi kunde då inte falsifiera några premisser, men vi upptäckte att flera premisser var beroende av varandra och stärkte deras sanningsvärde. I och med detta kunde vi säkerställa att slutsatsen var korrekt ur ett hypotetiskt deduktivt synsätt.

Etnografi

Den etnografiska metoden är en traditionell forskningsmetod som ursprungligen användes i socioantropologiska fältstudier, där forskarna reste till avlägset isolerade infödingsstammar och levde med dem under långa perioder för att studera deras liv. En etnografisk studie syftar till att ge en förståelse för en grupps eller organisations beteende. Denna metod har senare även visat sig vara effektiv inom andra forskningsdiscipliner. Etnografin är framför allt en kvalitativ undersökningsmetod. Eftersom konventionell etnografi säger att studien bör utföras under flera år för att kunna kartlägga och samla in material om en social konstruktion lämpar sig den dock mindre bra för forskning inom informatikdisciplinen (Bergqvist, 2002). En studie på flera år skulle vara mycket svårt att applicera inom denna disciplin eftersom det sker stora förändringar på mycket kort tid inom detta område. Nya behov, nya användningsområden och ny teknologi kan innebära att en så lång undersökning inte blir rättvisande.

Vissa etnografiska metoder har dock visat sig vara effektiva inom forskningsdisciplinen. Framför allt de där den etnografiska undersökningen löper parallellt med utvecklingen av en ny systemdesign. De fynd som görs under denna form av etnografisk studie måste dock tolkas och översättas till en mer lämplig form för att kunna ligga till grund för designspecifikationen (Bergqvist, 2002). Även annan användning av denna metod har visat sig effektiv inom informatik, till exempel där etnografiska studier görs för att utvärdera hur användarna kan tänkas reagera på ett framtida system, eller utvärdera hur användarna använder ett befintligt system idag (Harper, 1999).

En etnografisk studie kan variera stort beroende på vilken vinkling av metoden som används. När studien ska genomföras kan man välja mellan *deltagande observation* eller *observation* och alla varianter däremellan. En deltagande observation innebär att man som forskare ingår i det studerade objektet och utför samma uppgifter som de som man avser att undersöka. När denna form av studie utförs är det lättare att upptäcka annars dolda fenomen, exempelvis riter, historier, skrönor med mera som skulle kunna vara väsentliga för att förstå varför det studerade objektet fungerar på ett visst sätt. Nackdelen med denna inblandning i studien är dock att studien inte kan klassas som 100% objektiv eftersom det inte går att utesluta påverkan från forskaren.

En renodlad observation i enlighet med en etnografisk studie innebär däremot att endast objektet observeras och all inblandning i det studerade objektet undviks. Med andra ord skall objektet observeras som en fluga på väggen. Med denna form av studie kan saker som annars inte skulle kunna upptäckas faktiskt upptäckas eftersom det studerade objektet skulle kunna bete sig annorlunda mot vad det gör i normala situationer.

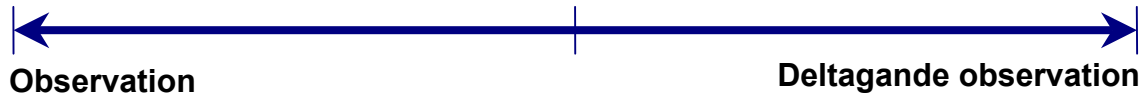


fig. 2.1 Etnografisk undersöknings två avgörande inriktningar

Exempelvis kan en etnografisk studie inom CSCW (Computer Supported Cooperative Work) utföras enligt följande principer:

Concurrent Etnography – Är förmodligen den vanligaste etnografiska undersökningsmetoden vilken utförs parallellt med designarbete av ett system. Denna metod är mycket tidsintensiv, undersökningen kan omfatta flera månader till års studier av det utvalda objektet. Om denna metod utförs med endast observerande synvinkel kan barriärer uppstå mellan observatörerna, utvecklarna och inte minst de tilltänkta användarna. Vi valde denna metod för att utföra studien ur ett helt deltagande observerande perspektiv. Genom detta slipper vi barriären mellan utvecklarna och observatörerna.

Quick and Dirty – Syftar till att en kort studie utförs som omfattar några dagar till max någon vecka. Bara för att studien inte är tidsomfattande innebär det inte att man inte kan samla in stora datamängder. En ”quick and dirty”-undersökning kan vara mycket effektiv, men med denna form av etnografisk studie kan det dock vara svårt att upptäcka djupare fenomen som endast kan upptäckas under en omfattande undersökning av objektet.

Evaluative Ethnography- Denna metod bör väljas när en redan existerande design skall studeras eller testas. Metoden är effektiv för att pröva en ny design eller teori. För snävt fokus under undersökningen kan dock förblinda forskaren ifrån att viktig information som kan finnas utanför studiens domän inte uppmärksammas.

Grounded Theory

Grounded Theory var från början en explorativ metod för utveckling av teorier som grundades på hypoteser och empiri. Denna metod introducerades i boken *The Discovery of Grounded Theory: Strategies for Qualitative Research skriven av* sociologerna Barney Glaser och Anselm Strauss 1967. Grundkonceptet i *Grounded Theory* är att skapa en så integrerad och meningsfull struktur av data som möjligt. Detta åstadkoms genom *kategorisering, sortering* och *tolkning* (”*memoing*”). Ju mer bearbetning, desto mer *integrerad, förtätad* och *mättad* teori, beskrivning eller slutsats kan man komma fram till (Langemar, 2004). Langemar menar även att detta kan ge upphov till en abstrakt och generaliserbar teori som samtidigt är empiriskt grundad (2004). Den del av denna teoriskapande metod som vi valde är där man tolkar och upptäcker mönster. I vår undersökning fann vi fem olika kategorier som de flesta fynd som rörde kärnämnet kunde kategoriseras under.

Utförandet

Vi utförde en tidsbegränsad studie på ett systemutvecklingsprojekt. Eftersom vi hade valt att följa Concurrent Ethnography med deltagande observation som forskningsmetod var vi delaktiga i projektet som vi samtidigt skulle dokumentera och undersöka. Våra empiriska fynd bestod av fältanteckningar som vi tog när vi upptäckte ett avvikande mönster, något problem eller fann en lösning på ett problem. Vi förde även övergripande anteckningar vid all interaktion med beställarna och de tilltänkta användarna. Användarna var samtliga, både säljare och ledning, som arbetade med de befintliga systemen. Beställarna var ledningsgruppen som önskade ett rapportsystem för att stödja organisationens försäljning och strategiska beslut. Den etnografiska studien begränsade vi till att omfatta tre månader av utvecklingsprojektet. Materialet skulle annars bli för omfattande för att kunna analyseras inom ramen av en magisteruppsats.

Vid själva utförandet av studien ingick vi i ett utvecklingsprojekt där data i ett arvssystem skulle migreras och en specialiserad rapportgenerator skulle utvecklas. Eftersom vi valde att utföra studien utifrån ett helt deltagande perspektiv fick vi tillgång till material som vi förmodligen annars inte skulle kunna tillgodogöra oss. Detta innebar även att vi var tvungna att se oss själva som objekt i studien. Med dethär perspektivet är det viktigt att tänka på att våra egna värderingar och erfarenheter kan ha haft viss inverkan på hur systemet utformades och problemen som uppstod. Vi försökte med detta i åtanke utföra studien på ett så objektivt sätt som möjligt.

Sammanställning av empiri

Efter den etnografiska studien sammanställde vi alla våra fynd och grupperade dem efter kategori. I enlighet med *Grounded Theory* filtrerade vi ut de huvudkategorier som vi enligt metoden kom fram till att våra empiriska fynd skulle grupperas under.

En kategori kunde innehålla flera olika fristående problem som vi var tvungna att separera. När vi väl separerat alla problem, försökte vi att hitta uppslag till våra fynd genom litteraturstudier. De uppslag som antingen styrkte eller förkastade våra fynd förde vi in i analysmaterialet för att analysen skulle kunna ge en så objektiv bild som möjligt av undersökningen.

Analys av empiri

De grupperade och sammanställda fynden analyserade vi i enlighet med ett hypotetiskt deduktivt angreppssätt. Våra empiriska fynd blev observerade premisser och till dessa förde vi även in hypoteser för att testa resultatets giltighet. Hypoteserna kunde vara teoretiskt grundade eller rena antaganden som vi hade gjort.

Om resultatet sedan visade sig vara giltigt gick vi tillbaka och testade hypoteserna för att verifiera eller falsifiera dem. Om resultatet kvarstod efter detta steg, betraktade vi resultatet som giltigt och gick vidare till nästa problem. Genom detta arbetssätt för att analysera våra empiriska data kunde vi direkt testa dem mot teorierna för att se om de stämde. Om teorierna inte stämde med våra fynd blev vi tvungna att analysera resultatet noggrannare för att avgöra om det var våra fynd som inte stämde eller om det var teorierna som var felaktiga.

Teori

När vi har funnit våra teorier har vi varit tvungna att bedöma vilken relevans de kan tänkas ha för vår undersökning. Med relevans menar vi inte endast innehållsenlig relevans utan även tidsrelevans, det vill säga att de teorier vi använder oss av skall kunna betraktas som gällande och inte föråldrade. Detta är ett problem vid forskning inom informatik eftersom ämnet ständigt förändrar sig med utvecklingen av nya IT-artefakter.

Arvssystem:

I det projekt vi har arbetat med så består arvssystemet av ett AS/400-system. Det är organisationens gamla system från vilket migrationen till den nya databasen skall utföras.

AS/400 – Teknisk specifikation

Vid beskrivning av hur AS/400 fungerar och är uppbyggt har vi framför allt använt Internetsidor och i första hand sidor från IBM. Den mest korrekta informationen borde komma direkt ifrån tillverkarna.

I juni 1988 introducerade IBM Application System/400 (AS/400), en ny typ av minidatorer som var enkla att använda och utformade för små och medelstora företag. AS/400 ersatte de gamla System/36 och System/38 och prestandan var avsevärt förbättrad. AS/400 har sedan introduktionen utvecklats på många plan och innefattar idag även ett eget operativsystem samt en Serverserie (IBM iSeries). ISeries serverlösning är lite annorlunda än andra serverlösningar. Som figur 3.1 visar så är integreringen mellan hårdvara och mjukvara väldigt stark. I andra miljöer, tex UNIX måste kunden välja mjukvara från olika leverantörer. I iSeries servrar binds hårdvara, mikrokod, operativsystem och IBMs mellanlager-komponenter samman vilket gör att alla dataresurser utnyttjas maximalt. (www.krypton.mnsu.edu)

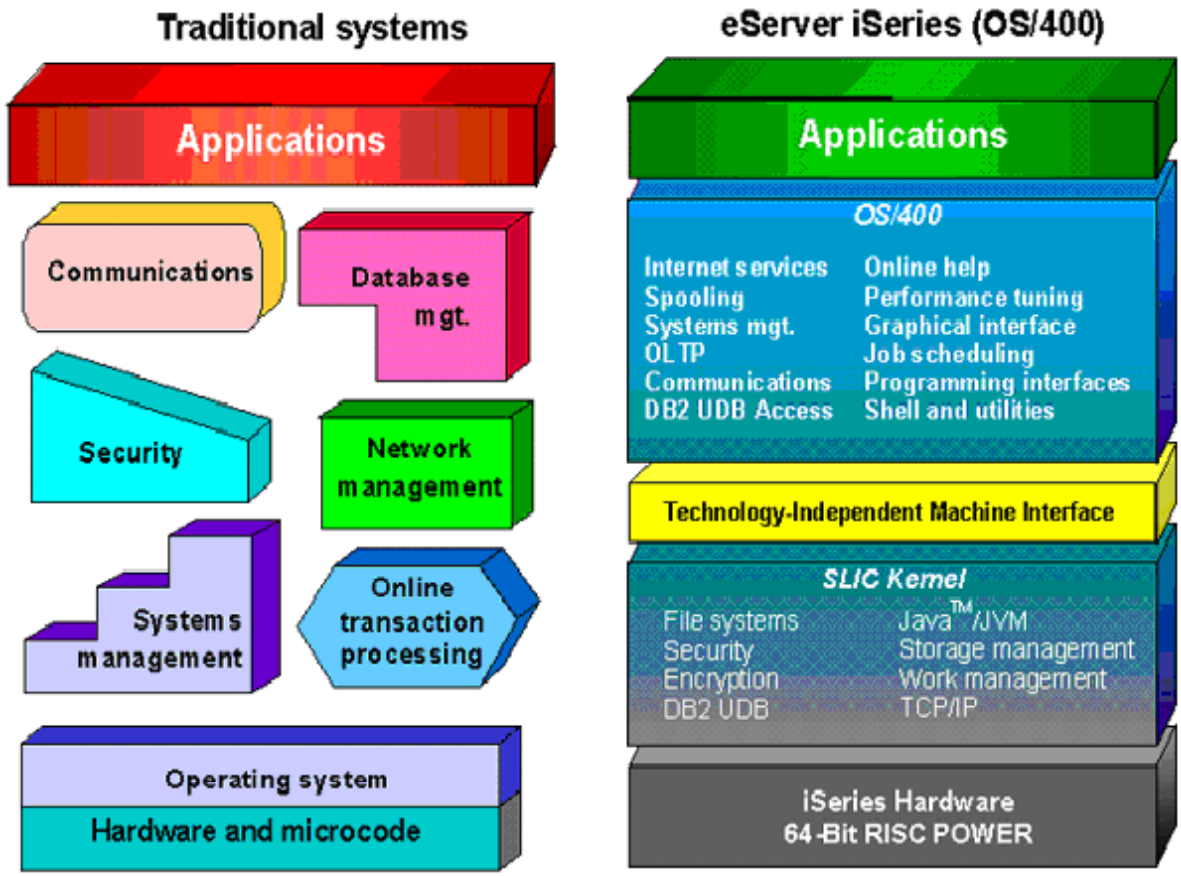


fig 3.1 Strukturen över AS/400 (www.IBM.com)

iSeries har en skiktad arkitektur med följande uppdelning:
* Användargränssnitt (OS/400 operativsystem)
* Ett teknologiskt oberoende maskingränssnitt (Hur hårdvaran kommunicerar)
* SLIC Kernel

SLIC Kernel är en ganska svårförstådd del av systemet som vi här tänkte förklara lite närmare. UDB/400 är ett integrerat databassystem i OS/400. Allting i OS/400 är organiserat som objekt och arbetar inte direkt mot de fysiska diskarna utan mot en form av mjukvara som heter ”System Licensed Internal Code” (SLIC). Denna mjukvara ”gömmer” diskarna och hanterar objektens positioner på diskarna. Ett virtuellt adressfält läggs över diskarna och används för att adressera objekten istället för de olika diskarna. De objekt som används kopieras från detta adressfält på disken in i adressfältet i RAM-minnet. OS/400 lagrar alltså data och komponenter på diskar och laddar in dem i minnet när de behövs, precis som i andra miljöer. Skillnaden är dock att OS/400 inte låter dig arbeta direkt mot data på diskar eller i minnet. Istället måste du använda specifika kommandon eller systemgränssnitt som är tillåtna för de olika sorts objekt som finns sparade. (www.ibm.com)

Databasfiler består till exempel av två typer av filer (eller objekt som kanske är ett bättre ord när det gäller AS/400).

* *Physical files*
* *Logical files.*

Physical files innehåller själva datan som är lagrad och består i sin tur av två olika sorts filer:

* *Data physical files*
* *Source physical files*

En data physical file är en fil som ej går att kompilera, till exempel en infil till ett program. En source physical file innehåller inte data utan källinformation som visar vilken typ av fil det är, till exempel Pascal- eller COBOL-filer. Filerna har en eller flera *medlemmar* som i sig själva inte är objekt utan delar av samma objekt och ärver då detta objektets egenskaper. Det är denna uppbyggnad som gör att kommandon inte kan utföras mot ett objekt som egentligen är till för andra sorters objekt och istället kan skada filen. Om källfilen visar att det är en medlem med CBL-attribut (COBOL) så kommer AS/400-editorn att hantera programmet just som ett COBOL-program och när det skall kompileras så vet den att det är en COBOL-kompilator som skall användas. (www.ibm.com)

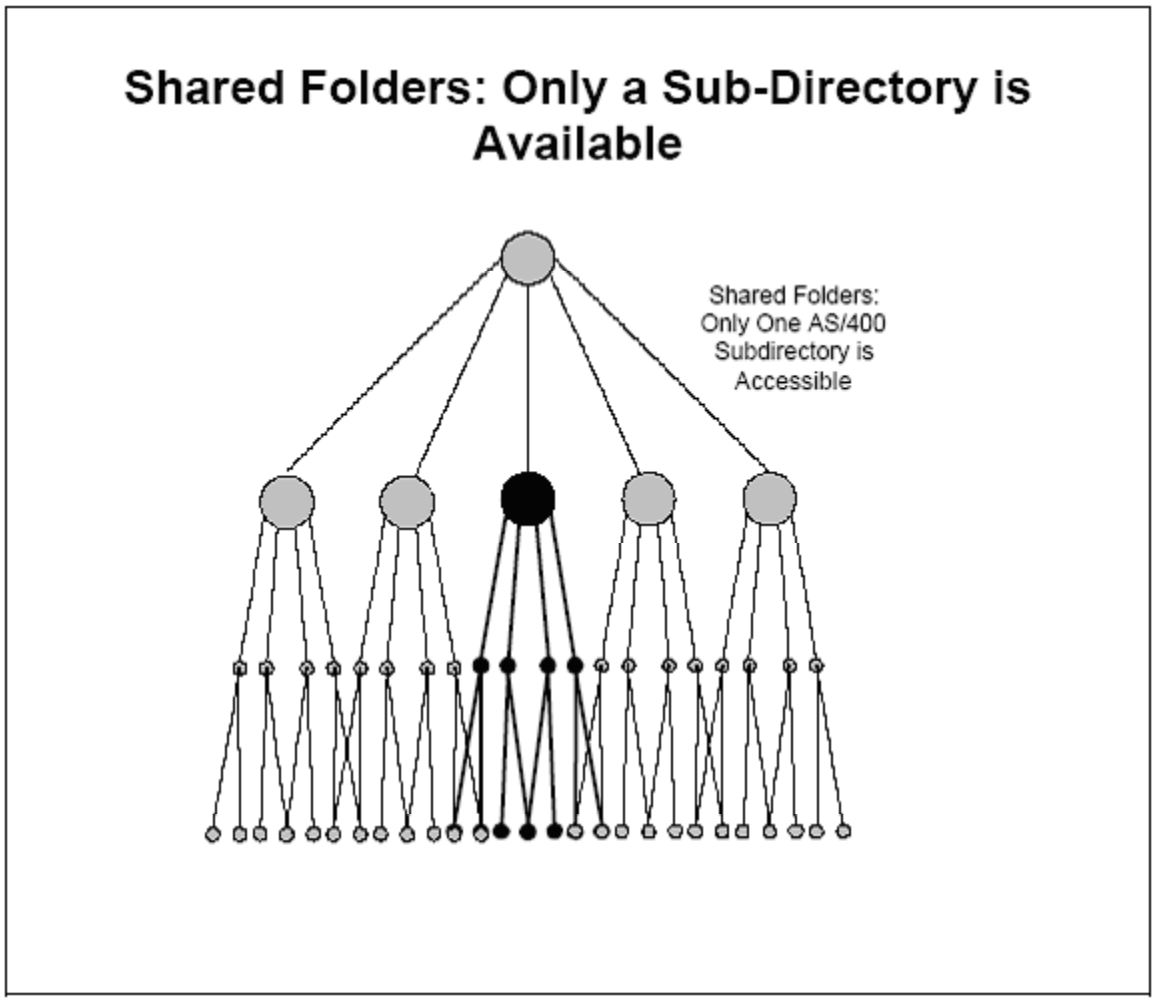
Logical files är filer som inte innehåller någon direkt data utan tillåter en annan metod att se datan för tillhörande physical file. Konceptet är liknande det ”View-koncept” som finns i SQL. Den logiska filen beskrivs för systemet genom DDS (Data Description Specifications) som är ett språk för att förklara databasfiler för systemet. När DDS:en kompileras skapas ett logical files-objekt.

Datastruktur i AS/400

Datastrukturen i AS/400 är uppbyggd som en arvstruktur. När strukturen skall förklaras finns det två viktiga delar: ”Shared Folders” och ”IFS” (Integrated File System).

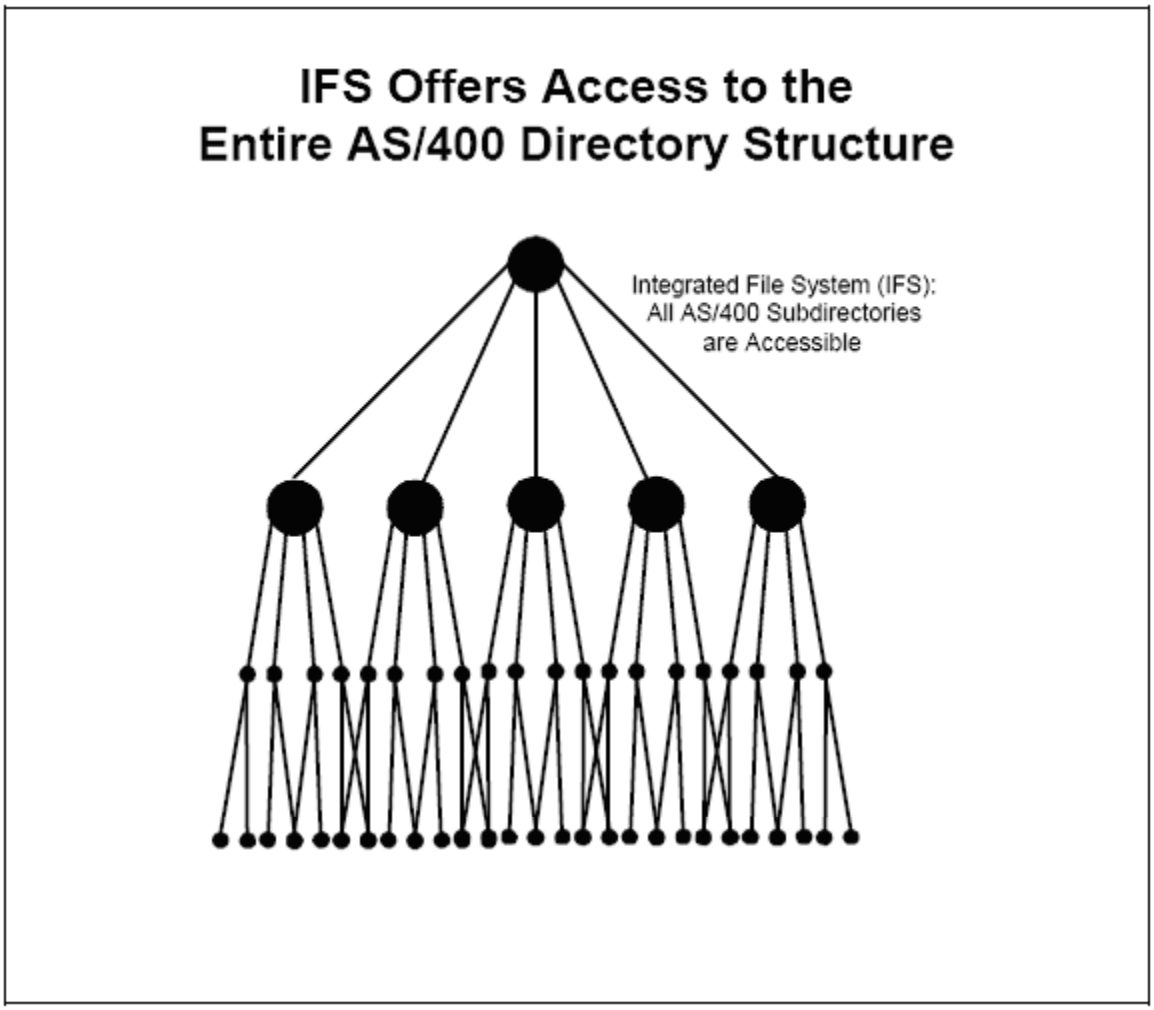
Shared Folders

En av de viktigaste mjukvarorna tillhörande AS/400 är “Shared Folders (SF)”. Detta är en funktion som tillåter PC-klienter att se information på AS/400-servern som en egen enhet på sin dator, liknande funktionaliteten i Windows NT Server.



Figur 3.2 Shared folders (www.krypton.msnu.edu)

SF ger alltså användaren möjlighet att se information på AS/400. Tillgången till AS/400s data är dock begränsad till PC:ns shared folders mapp. Klienten kan alltså inte se all information på AS/400-servern utan enbart den enhet som ges tillgång till genom SF (se figur 3.2).



Figur 3.3 Integrated File System (www.krypton.msnu.edu)

Genom att använda sig av AS/400s Integrated File System (IFS) kan servern på motsvarande sätt som på en NT-server dela ut *alla* bibliotek och enheter. AS/400-klienten måste naturligtvis ha stöd för IFS (se figur 3.3). (www.bitpipe.com)

Integrated File System:

IFS är en del av OS/400 som låter användaren interagera med systemet i realtid med *Stream Files* (se definition för förklaring) istället för med buffrad data. Hantering av lagrad data sker på liknande sätt som i PC-miljö och andra operativsystem, till exempel UNIX. Detta ger användaren en strukturerad tillgång till all lagrad information i AS/400.

Viktiga funktioner i IFS är:

- Möjlighet att lagra information i Stream Files.
- En hierarkisk mappstruktur
- Ett gemensamt gränssnitt som tillåter användare och applikationer att få tillgång till inte enbart Stream Files utan även databasfiler, dokument och andra objekt sparade i AS/400. (www.IBM.com)

Databashantering i AS/400

Som vi beskriver i den tekniska specifikationen kring AS/400 så är databassystemet UDB/400 integrerat med det övriga systemet. Det finns visst stöd för arbete med frågeverktyg mot denna databas. Här följer lite information om vad artiklar och forskning anser om detta.

Om frågor skall ställas direkt mot det inbyggda databassystemet finns det två mycket användbara applikationer, SQL/400 och Query/400. SQL-applikationen tillhandahåller användaren ett interaktivt SQL-gränssnitt samt alla rutiner som behövs för att exekvera frågorna.

Query/400 är framför allt till för att skapa rapporter av existerande data. Många funktioner här påminner om funktioner tillgängliga i SQL Server, bland annat kan frågorna sparas och köras vid ett senare tillfälle. Nackdelen med AS/400s objekthantering av information är att det tenderar att använda mer utrymme för att spara samma information jämfört med andra system. Data för ett objekt kan sparas över flera diskar. Detta kan även leda till omfattande ominstallationer även om enbart en disk kraschar då den kan innehålla delar av väldigt många program (Scholerman m.fl., 1993).

Den första versionen av IBMs Data Warehouse-arkitektur som publicerades 1993 uppkom då många användare hade problem med att få fram användbar information för beslutsfattande från grunddata (www.ibm.com). Till skillnad från relationsdata-basernas fördelaktiga uppbyggnad ur ett SQL-perspektiv är arvssystemsdatabasernas uppbyggnad i större behov av Data Warehouse då deras hierarkiska uppbyggnad gör frågor mot databasen mer komplicerade.

Datamigration

Definition av datamigration enligt www.Webopedia.com:

En process som innebär överföring av data från ett format till ett annat. Datamigration är nödvändigt när en organisation beslutar sig för att byta till ett nytt data- eller databashanteringssystem som inte kompatibelt med det tidigare systemet. Vanligtvis utförs datamigrationen genom en uppsättning av anpassade program eller script som automatiskt överför datan.

På grund av att dagens affärsprocesser blivit mer avancerade har system för affärsintegration, ETL (Extract Transform Load), processer för Data Warehousing, aktiviteter för datarensning och migrering av data från arvssystem blivit en allt viktigare fråga för ett ökat antal organisationer (Carreira & Galhardas, 2004).

Datamigration med verktyg

Vanliga datamigrationsverktyg är utvecklade för att till exempel transformera arvsdata som lagrats i källor med förbestämd struktur till målsystem med en annan fördefinierad struktur. Verktygen har med andra ord utvecklats för att hantera migration mellan endast två specifika system. När organisationer köper nya affärssystemspaket (exempelvis SAP) som skall ersätta de befintliga systemen blir ett datamigrationsprojekt där datan måste behandlas för att passa det nya applikationspaketet nödvändigt (Carreira & Galhardas, 2004). Detta kompliceras genom att traditionella migrationsverktyg oftast inte klarar att migrera koden problemfritt.

Flera faktorer blir avgörande när datamigrationstransformeringarna skall specificeras. Den första av faktorerna som Carrerira & Galhardas (2004) nämner är att de ETL-verktyg för migrering av data som i dag finns på marknaden inte är tillräckligt omfattande och inte kan hantera tillräckligt kraftfulla språk för att behandla de ingående transformeringsreglerna. För komplexa transformeringar är det vanligt att dessa hanteras av *ad-hoc*- (se definitioner) program som kodats utanför verktygen. Med detta menas ett program som slumpvis väljer en mängd data för transformering.

Den andra faktorn är att det för att skapa datamigrationsprogram behövs mer än vanliga programmerare. De som skapar migrationskod är oftast även affärsspecialister. Enligt Carrerira & Galhardas (2004) föredrar dessa utvecklare att använda sig av högnivåprogram som enkelt kan skapas med hjälp av högnivåspråk.

Den tredje faktorn är att kostnaden för utvecklingen och underhållet av migrationsprogrammen måste minimeras och koden måste vara kort och exakt för att enkelt kunna modifieras (Carrerira & Galhardas, 2004).

Slutligen, data som migreras vid införandet av ett nytt system är av mycket kritisk natur för organisationen. Detta innebär att organisationerna ställer högre krav på

migrationsprojekten och datan som migreras måste vara korrekt och exakt. Därför måste transformeringsverktygen kunna prestera exakta resultat. Carrerira & Galhardas (200b) och Dayal & Chaudhuri (1997) uttrycker också att det blir viktigare med verktyg som ”tvättar” och kontrollerar datan som skall migreras.

Problematik med datamigration

Att migrera data är ingen enkel operation, det kan innebära en rad utmaningar som måste antas. Exempelvis måste datan som överförs från det gamla systemet transformeras om till passande format för den nya miljön. Katherine Hammer (1997) menar att det finns en rad olika orsaker till att migrationsarbetet kan bli komplicerat. De vanligast förekommande orsakerna enligt Hammer är:

1. Samma form av data kan vara annorlunda i olika system. Ett specifikt fält i ett system kan vara representerat av två siffror medan det i andra systemet presenteras som två bokstäver.

2. Innehållet i en tabell kan förändras under migrationsarbete. Om inte det gamla systemet stoppas under migrationsarbetet kan innehållet i tabellerna ändras under tiden migrationsarbetet pågår och data går förlorat.

3. Förändringar och utveckling av tabellerna i det gamla systemet kan ha pågått under flera år och ingen enskild person har kontroll över vilka historiska ändringar som gjorts.

4. Datan kan vara dålig, det kan finnas flera poster med samma innehåll fast med små skillnader, exempelvis att en förkortning skrivs ut på en post och inte på den andra.

5. Datan kan vara presenterad på ett sätt som inte är användbart för slut-användaren, exempelvis blir inte användare klokare av att se att 3000 stycken av produkt 11 har sålts till kund 9908. En användare skulle vilja se vad produkt 11 är för något och vem kund 9908 är. Detta problem uppkommer oftast på grund av att databasutvecklaren till det gamla systemet har strävat efter att spara minnesutrymme.

6. Tolkning och migrering inom heterogena miljöer involverar oftast data ifrån inkompatibla databashanteringssystem (DBMS) och hårdvaruplattformar.

7. Applikationer som körs på de äldre systemen är oftast mycket specialiserade för de gamla systemen och kan vara svåra att migrera till en nyare systemplattform. Detta kan orsaka att samtliga datorsystem inom en organisation störs vid en migrering.

Kelly & Nelms (2003) menar också att det finns några orsaker till problematik eller som de utrycker sig, huvudsakliga utmaningar vid ett datamigrationsarbete. De har listat fyra faktorer som stämmer med Hammers (1997) problemorsaker och dessa är:

1. Fånga data ur det gamla och det nya systemet för jämförelse
2. Skillnader på struktur hur data lagras i det nya och gamla systemet
3. Ändringar av realtidsdata under migrationsarbete, exempelvis att någon lägger in nya transaktioner i det gamla systemet under pågående migrationsarbete.
4. Noggrannheten av data som valts för transformering i migrationen.

I stora drag överensstämmer Kelly & Nelms (2003) och Hammers (1997) punkter gällande orsaker till problem vid datamigration. Ett exempel på detta är punkt 3 i Kelly & Nelms lista som hävdar samma orsak som i Hammers punkt 2.

Kontroll av migrerad data

De data som en organisationsledning avser att migrera kan vara allmän affärsdata, bokföringsdata och data för historiska transaktioner. Dessa olika former av data kan vara mycket affärskritiska och det är därför mycket viktigt att inget går förlorat vid migrationen. Det är normalt att organisationsledningen vill ha någon form av försäkring mot att inget går förlorat vid migrationen (Kelly & Nelms, 2003). Som tidigare nämnts, uttrycker även Carrerira & Galhardas (2004) detta, men de observerar även att datan måste transformeras på rätt sätt för att tillgodose ledningens krav på noggrannhet, exempelvis hur många decimaler datan skall presenteras med i det nya systemet. Dayal & Chaudhuri (1997) menar att det kan behövas speciella verktyg för att ”tvätta” datan som avses att migreras för att den skall kunna användas korrekt i de nya systemen. Exempelvis kan ej överensstämmande fältlängder behöva korrigeras för att datan skall kunna användas. Ytterligare exempel är att variabeltypen är annorlunda i de nya systemen och måste därför transformeras till passande typ (Dayal & Chaudhuri, 1997; Carrerira & Galhardas, 2004).

Kelly & Nelms (2003) föreslår fyra manuella metoder för att försäkra sig om att data inte har gått eller går förlorad vid migrationen:

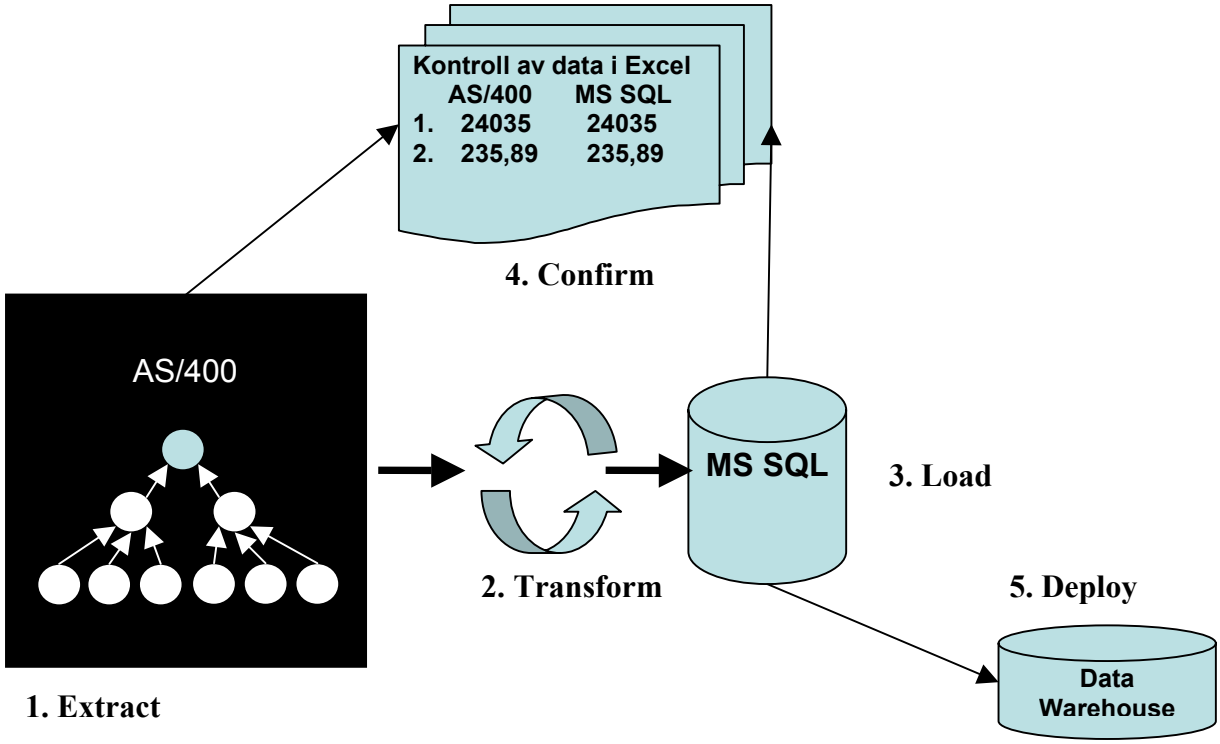
1. Testa och kontrollera data efter det att den har migrerats till det nya systemet
2. Testa och kontrollera data under migrationsprocessen
3. Kontrollera med ledningen om de har någon befintlig metodik för migration av data.
4. Gör ingenting och hoppas på det bästa.

Den första metoden kan ge bra försäkran om att allting stämmer, men om det nya systemet redan satts i drift kan skadan redan ha inträffat. Den andra metoden kan ge en 100% risktäckning medan den försvårar migrationsarbetet genom att ytterligare en faktor adderas till det redan komplexa arbete som datamigrationen innebär. Metod tre kan vara en fördel eftersom delar av ansvaret överförs på ledningen vilket gör den mer delaktig i migrationsprojektet. Det fjärde alternativet kan betecknas som mycket riskabelt och ger inte den försäkran som organisationsledningen kan kräva (Kelly & Nelms, 2003).

Strukturella dataskillnader

Skillnader på hur den lagrade datan strukturerats i det gamla respektive nya systemet kan innebära en mycket komplicerande faktor. De lagringsstrukturella problemen kan vara svåra att uppfatta vid presentationen av datan eftersom de kan vara på tabellnivå, exempelvis hur tabellstrukturen i de olika databaserna är uppbyggd eller hur en variabel presenteras i tabellerna (Hammer, 1997). Databaserna kan vara hierarkiska-, objektorienterade- eller relationsdatabaser. För att säkerställa att alla data har extraherats korrekt, rekommenderar Kelly & Nelms (2003) att tabell mot tabell jämförs.

Kelly & Nelms (2003) beskriver vidare att i deras projekt använde de Microsoft Excel för att jämföra data och de då upptäckte att äldre versioner av Excel hade en begränsning på 16384 rader medan nya versioner har stöd för upp till 65536 rader. Deras version var av den äldre varianten vilket orsakade problem för dem eftersom datan som de önskade verifiera översteg 16384 rader, vilket innebar att många rader förlorades. Detta löste de genom att exportera datan i ett dBase-format som de senare öppnade med Excel. Denna form av kontroll kan se ut enligt figur nedan.



Figur 3.4 Datamigrationsarbete ifrån AS/400. 1: Data extraheras från arvssystem. 2: Transformeras till passande format. 3: Laddas upp i det nya systemet. 4: Data från de olika systemen jämförs för att verifiera dom. 5: Data laddas in i Data Warehouse

Problem med föränderliga system

Med föränderliga system menas att dess data inte är statiska utan förändras hela tiden som systemen är i drift. Detta kan bli ett problem när data från ett arvssystem till ett nytt system skall migreras. Ny data kan ha matats in i det gamla systemet under migrationsarbetet och detta leder då till att data går förlorad (Hammer , 1997; Kelly & Nelms, 2003).

Kelly & Nelms (2003) menar att en lösning på detta problem kan vara att det gamla systemet tas ur bruk vid själva migrationstillfället. I en aktiv organisation där ett avbrott i systemet kan innebära stora förluster är detta dock inget alternativ. Carrerira & Galhardas (2004) rekommenderar att datamigrationen skall utföras under en helg då systemanvändningen är låg. Hammer (1997) menar att man måste undersöka systemet och synkronisera processen för att göra den så kort som möjligt. Detta innebär dock att när processen sker på kortare tid riskerar migrationsarbetets komplexitet att öka och risken att fel uppstår ökar (Hammer,1997). Denna form av optimering av migrationsarbetet föreslår även Kelly & Nelms (2003) som lösning på problemet gällande föränderlig data, men de hävdar att migrationsarbetet skall trimmas ner till någon minuts marginal från att det gamla systemet tas ur bruk. Alltså måste migrationen planeras så den utförs så kort tid som möjligt innan det gamla systemet tas ur bruk. Eventuella fel som kan uppstå under detta moment skall lösas i efterhand

menar de, detta för att inte förlänga övergångstiden och riskera att data går förlorad (Kelly & Nelms, 2003).

Val av data och hur den skall presenteras

Kelly & Nelms (2003) nämner att ett av de mest svårlösta problemen vid migrations-arbete kan vara hur man ska bedöma vilka data som måste överföras till det nya systemet och vilken noggrannhet som krävs för att dessa data skall betraktas som giltiga. Detta innebär en noga avvägning med organisationen för att klargöra hur långt tillbaka i tiden som datan fortfarande betraktas som affärsmässigt relevant.

För att säkerställa att relevant data migreras måste man först extrahera eventuell beskrivande data ur det gamla systemet, så kallad *metadata* (se definitioner) (Hammer, 1997). Metadatan kan i detta fall vara beskrivningar på affärslogik om hur vissa typer av data används i systemet och hur den klassificeras. Metadatan kan exempelvis beskriva hur man skall betrakta transaktioner och hur dess livslängd påverkar datan i tabellerna (Kelly & Nelms, 2003).

Data Warehouse

Vi kommer i detta avsnitt att berätta lite om Data Warehouse, framför allt vad det är, var det finns och hur forskare anser att det bör vara uppbyggt. Data Warehouse kommer genomgående att skrivas ”DW”

Definition av Data Warehouse

Det finns många definitioner av vad ett DW är. Vi har valt att presentera tre stycken. När DW började dyka upp på marknaden menade Gupta (1997) att det skulle innehålla statisk historisk data samlad under en lång tid och gav följande långa definition, fritt översatt:

”*Ett DW är en strukturerad utbyggbar miljö avsedd för analyser av ickeföränderlig data, logiskt och fysiskt transformerad från flera källor för att passa företagsstrukturen, uppdaterad och underhållen under en lång tid, uttryckt i enkla företagstermer och summerad för analyser* .”

En mer kort och koncis definition ges av Ralph Kimball (1998):

”Ett DW är en kopia av transaktionsdata strukturerad för frågor och analyser”.

Sist skall vi även visa Inmons definition då han har fått epitetet ”The father of data warehousing” (Connolly & Begg, 1998):

”En subjektorienterad, integrerad, tidsvarierande och statisk samling av data utformad för företags beslutsfattarprocess” (Inmon, 1993).

Modell och historisk kontext

Agosta (1999) ger en bra och kort förklaring av vad ett DW är: *”Ett DW är ingen mjukvara eller applikation, det är en systemarkitektur”*. När vi undersöker strukturen i ett DW är det viktigt att ha förståelse för uppbyggnaden av en databas. Den vanligaste databasen idag är relationsdatabasen vilken presenterades första gången år 1970 av E.F Codd i en uppsats som heter: *”A relational of data for large shared data banks”*. Denna uppsats klassas idag som en milstolpe inom databassystem. (Connolly & Begg, 1998)

Enligt Codd så finns det 13 regler för en relationsdatabas, numrerade från 0 till 12, som kallas Codds 12 lagar (Plew & Stephens, 2000):

0. Ett relationsdatabassystem kan hantera databaser enbart med hjälp av sina relationella egenskaper.

1. **Informationslagen:** All information i en relationsdatabas (inklusive tabell- och kolumnnamnen) representeras uttryckligen som värden i tabeller.

2. **Garanterad tillgång:** Varje värde i en relationsdatabas är garanterat tillgänglig med hjälp av en kombination av tabellnamnet, det primära nyckelvärdet och kolumnens namn.

3. **Systematiskt stöd för nullvärden:** DBMS-systemet skiljer systematiskt nullvärden (det vill säga icke tillämpliga eller okända data) från standardvärden, oberoende av domän.

4. **Aktiv och direkt tillgänglig relationsbaserad katalog:** Beskrivningen av databasen och dess innehåll representeras på logisk nivå som tabeller och kan därför avfrågas med databasspråket.

5. **Ett omfattande underspråk för datahanteringen:** Minst ett språk måste ha en väl definierad syntax och vara omfattande. Det ska stödja datadefinition, data-manipulation, integritetsregler, auktorisering och transaktioner.

6. **Lagen om uppdatering av vyer:** Alla vyer som teoretiskt sett kan uppdateras ska kunna uppdateras via systemet.

7. **Infogande, uppdatering och radering:** DBMS-systemet stödjer inte bara återhämtning på mängdnivå utan även infogande, uppdatering och radering.

8. **Fysiskt oberoende data:** Applikationsprogram och tillfälliga program påverkas inte logiskt när de fysiska återkomstmetoderna eller lagringsstruktuerna ändras.

9. **Logiskt oberoende data:** Applikationsprogram och tillfälliga program förblir i största möjliga mån logiskt opåverkade när tabellstrukturerna ändras.

10. **Integritetsoberoende:** Databasspråket måste kunna definiera integritetsregler. Dessa regler ska lagras i den direktanslutna katalogen och inte kunna kringgås.

11. **Distributionsoberoende:** Applikationsprogram och tillfälliga frågor förblir logiskt sett opåverkade när data distribueras första gången och när de distribueras på nytt.

12. **Oåtkomlighet:** Det får inte vara möjligt att kringgå de integritetsregler som definierats i databasspråket genom att använda programspråk på lägre nivå.

Det utmärkande med idén om relationsdatabaser var konceptet med relationer som normaliseras. Normalisering av databaser innebär att informationen som finns lagrad i många tabeller delas upp på flera tabeller istället för att lagra allting i en stor tabell (Elmasri, 2000). De tabeller som innehåller relaterad data kopplas sedan ihop så att man kan få ut informationen som behövs och som är relaterad men ligger i olika tabeller. Det finns många anledningar och definitioner för normalisering, Connolly & Begg (1998) anser att normalisering är:

”En teknik för att producera en samling av relationer med egenskaper i enlighet med kraven från en organisation” (Connolly & Begg, 1998).

Som Gupta antyder i sin definition av ett DW ansågs tidigare att data är statisk i ett DW och att datan inte skulle ändras när den väl var där (Gupta, 1997). Detta synsätt har på senare år frångåtts och bland annat Kimball menar att ett DW även skall innehålla ny information eller att befintlig information skall kunna uppdateras (Kimball, 1998). Connolly & Begg (1998) visar skillnaderna på ett transaktionssystem, OLTP (Online Transaction Processing) och ett DW på följande sätt:

OLTP Systems	**Data Warehousing Systems**
Innehåller aktuell data	*Innehåller historisk data*
Innehåller detaljerad data	*Innehåller summerad data*
Data är dynamisk	*Data är statisk*
Transaktionsdriven	*Analysdriven*
Applikationsorienterad	*Ämnesorienterad*
Underlag för dagliga beslut	*Underlag för strategiska beslut*
Många transaktioner	*Få transaktioner*
Återkommande processer	*Ostrukturerade processer*
Förutsägbara anv.områden	*Oförutsägbara anv.områden*
Hanterar många vanliga användare	*Hanterar få övervakande användare*

Forskning kring DW

Zagelow (1997) anser att det första som bör göras när man skall bygga ett DW är att avgränsa sig. Enligt honom är företagen ofta inte medvetna om komplexiteten med att utveckla ett effektivt DW. Ansvariga vill att det skall innehålla information för att kunna besvara *alla* intressanta frågor om *alla* intressanta ämnen och det kan ju knappast kallas en avgränsning. Storleken på DW bör alltså bestämmas tidigt.

För att kunna bygga ett DW som går att använda är det nödvändigt att den underliggande databasen studeras och att det fastställs om informationen för att skapa sitt DW verkligen är tillgänglig och korrekt. Med detta menas att det bör finnas någon form av beskrivning eller schema för databasen. Två metoder som fungerar bra för detta är *”View Integration”* och *”Reverse Engineering”*. För mer detaljer kring detta så är (Batini & Lenzerini, 1984) och (Francalanci & Fuggetta, 1997) två bra exempel. (Bonifati m.fl., 2001)

Eftersom DWs uppgift är att förse användaren med data för rapporter och beslut är svarstiderna på frågorna av stor vikt. Är det frågor som ställs under det dagliga arbetet så är det viktigt att det är snabba svarstider. Skall de köras under natten och vara klara när användaren kommer till jobbet på morgonen är inte svarstiderna lika prioriterat. Gupta (1997) använder sig av uttrycket ”*Query Respons Time Constraint*” och förklarar det med att varje fråga som ställs inte får ha en svarstid som överstiger en viss uppsatt tid. Denna tid bestäms av användaren med utgångspunkt från kravet på svarstider för den specifika frågan.

Förekomster och marknad

Vilka som kan ha användning för ett DW är kanske inte det lättaste att svara på men företag är i ständigt behov av olika rapporter, sammanställningar och analyser, allt från lönsamhetsrapporter till kundbeteende. DW är en slags samlingsplats för företagsdata, från vilken analyser av data kan göras på ett enkelt sätt (Gupta, 1997). Agosta (1999) sammanställer olika anledningar för att vilja bygga ett DW med en bra frågeställning:

”Vem köper vad – och när och var gör dom det?”

”Köper” behöver i detta fall inte vara just köper utan kan ha fler definitioner, ”använder”, ”beställer”, ”levererar” med mera. Denna frågeställning visar en stor skillnad på arvssystem, som innehåller vertikala relationer och inte ger någon *bred* bild av beteenden, och system utformade för rapporter och beslut. Data i ett DW är nedbruten data från arvssystem/ERP (Enterprise Resource Planning) och/eller e-handelssystem (Agosta, 1999). Glassey-Edholm ger en mer praktisk lättförståelig anledning till DW: *”DW behövs för att ge användaren bättre underlag för beslut” (Glassey-Edholm, 1997)*. De anser därför också att det är mycket viktigt att utgå ifrån användarens perspektiv under hela utvecklingsprocessen, från design till implementation.

Avslutningsvis skall vi nämna de stora fördelar som enligt Connolly & Begg ett byggande av DW kan medföra (Connolly & Begg, 1998):

* Möjlighet till hög avkastning på investerat kapital
* Konkurrensfördelar tack vare bättre beslutsfattande (Kan ta bättre beslut)
* Högre produktivitet hos företagens beslutsfattare (Kan ta beslut inom fler områden)

Detta går egentligen lite utanför vårt område och för mer information om detta rekommenderas boken: *"Improving data warehouse and business information quality: Methods for educing and increasing profits"* av Larry P. English (1999).

Datastruktur i DW

Hur data är lagrad har ingenting med att göra om det är ett DW eller inte anser Ralph Kimball (1998) som säger följande om strukturen på den lagrade datan:

"Ett DW kan vara normaliserat eller denormaliserat. Det kan vara en relationsdatabas, multidimensionell databas, vanlig enkel fil, hierarkisk databas, objektdatabas etc. Data i ett DW ändras ofta. DW fokuserar ofta på en specifik aktivitet eller entitet".

Eftersom uppgiften för ett DW är att ge rapporter och statistik för att hjälpa användaren att fatta beslut så det är viktigt att datan är korrekt och strukturen på datan är välordnad (Mullins, 1996). Mullins et al anser att detta ofta är ett problem och att det i många fall borde kallas Data Outhouses istället för Data Warehouses på grund av att strukturen och innehållet i databasen håller låg kvalitet. När man skall få förståelse för datans uppbyggnad och göra om outhouset till ett warehouse så underlättar det mycket om företaget har en modell för hur datan är strukturerad (Mullins, 1996).

Om informationen i ett DW bör vara normaliserad eller denormaliserad är en aspekt där åsikterna går isär något. Mullins (1996) säger följande om detta:

"Många anser att informationen i ett DW bör denormaliseras, men var försiktig med detta. Eftersom denormaliserad data är optimerad för dataåtkomst och ett DW är en "enbart läs-databas" så kan man tycka att denormalisering är det naturliga att göra. Datan måste dock flyttas in i DW vid någon tidpunkt. Denormaliserad data är svår att underhålla och bör undvikas om prestandan är acceptabel."

Enligt Inmons (1993) definition av ett DW skall data i ett DW vara:

Objektorienterad – Eftersom datan i ett DW är organiserad runt de områden företaget sysslar med (kunder, produkter och försäljning) snarare än de applikationer företaget använder sig av (hantering av kundfakturering, lagerkontroll, produktförsäljning). Det

är detta man skall koncentrera sig på när man hanterar data för beslutsfattande och inte applikationsorienterad data.

Integrerad – Eftersom data i ett DW kan komma från många olika källor är det troligt att data som egentligen skall visa samma sak ser annorlunda ut i format och beskrivning. Det är av yttersta vikt att denna data blir integrerad med varandra så att alla data som skall visa liknande information också finns sparad på samma sätt.

Tidsvarierande – Datan i ett DW är endast exakt och giltig inom ett visst tidsintervall. För att datan skall vara giltig måste den uppdateras kontinuerligt.

Statisk – Data uppdateras inte i realtid utan uppdateras med jämna mellanrum av annan data. Ny data läggs alltså till den tidigare datan, den ersätter den inte. DW får alltså fortlöpande ny information som integrerar med den gamla datan

Resultat

Efter tre månaders etnografiska studier har vi sammanställt vårt insamlade material och grupperat fynden under fem huvudsakliga referensområden. Innan vi presenterar de fynd som gjorts vill vi kort beskriva det Data Warehouse, ETL-verktyg och rapportverktyg som utgjort vår undersökningsmiljö.

Laboratoriet för den etnografiska undersökningen

Företagets huvuddatabas i Japan var som tidigare nämnts IBM AS/400. Lokalt användes i Göteborg sedan tidigare en Microsoft SQL Server 2000-databas och om möjligt ville man fortsätta använda denna även för det nya Data Warehouse som vi fick i uppdrag att utveckla. Strategin var att data från Japan tankades ner varje natt för att reducera belastningen i möjligaste mån på AS/400-servern. Eftersom personalen använder företagets AS/400 löpande för bland annat orderläggning, fakturering och annan daglig administration var det av stor vikt att övertankningen skedde under lågtrafik. För att få en säkerhetsmarginal för oväntade avbrott i systemet hämtades data från närmaste dag innan körningen upp till en vecka tillbaka i tiden. Detta medför möjlighet att gå tillbaka i tiden och återskapa eventuell förlorad data.

Datan som matades in i vårt Data Warehouse strukturerades på ett sätt som specialanpassats för de rapporter företaget ville få presenterade. Ca 40 tabeller från AS/400 hämtades hem varje natt, vilka omstrukturerades till totalt sex tabeller. På detta sätt optimerades rapportsystemet samtidigt som, genom att hämta fler tabeller än vad som senare användes i vårt specifika Data Warehouse, stora möjligheter lämnas att expandera systemet med ytterligare framtida rapportalternativ och tabeller.

För att få fram önskade rapporter tidigare hade en person på företaget formulerat egna frågor via IBMs Client Access. Client Access är ett program som transformerar AS/400/Unixdata till Microsoft Windows-miljö. Resultatet av dessa frågor kan presenteras i bland annat rena kommaseparerade textfiler eller Microsoft Excel-filer. Istället för denna tidskrävande och prestandaineffektiva metod använde vi DTS, *Data Transformation Services*, som är ett verktyg i SQL Server för vår transformering av AS/400-data till Data Warehouset. Vi lyfte ut kod från SQL Server och gjorde en egen ETL-applikation för övertankningen baserad på DTS-koden. Detta gjordes eftersom viss specialanpassning behövdes, vilket beskrivs närmre senare.

För att presentera rapporterna för användarna utvecklade vi från grunden ett rapportverktyg som vi döpte till YSR 2005. Eftersom alla tre uppsatsskrivare har störst erfarenhet av systemutveckling i Java började vi utvecklingen av rapportverktyget med detta språk. Vi märkte dock snart att det föll sig naturligt att använda Microsofts *VB.NET* som utvecklingsmiljö med tanke på att SQL Server användes som databas och att uppdragsgivaren ville ha rapporterna i Excelformat. Samtliga är Microsoftprodukter och anpassade efter att fungera med varandra. Resultatet av det färdiga

rapportverktyget visas i bilaga 1. Systemet kommer att användas av säljare och ledningspersonal inom organisationen för att ge stöd för strategiska och taktiska beslut inom försäljning. För att ge en uppfattning av omfattningen av utvecklingsarbetet kan nämnas att ca 80000 rader kod användes för att programmera ETL-applikationen och ca 5000 rader kod behövdes för rapportverktyget. Vi var placerade i ett eget rum på det svenska företagets huvudkontor vilket möjliggjorde täta kontakter med berörda parter inom organisationen.

Presentation av empiriska fynd

De fem huvudsakliga referensområden som vi har grupperat våra insamlade fynd under har, som tidigare nämnts, tagits fram i enlighet med Grounded Theory-metoden. Genom att använda denna metod blir resultatet en tydlig och överblickbar grund för analysen av de insamlade fynden.

1: Empiriska fynd rörande problematiken med AS/400 och den befintliga databasen
2: Empiriska fynd rörande problematik med användare/beställare i migrationsprojektet
3: Empiriska fynd rörande de tekniska aspekterna med migration.
4: Empiriska fynd rörande problematiken med Data Warehouse
5: Empiriska fynd rörande prestandaproblem under utvecklandet av DW

Följande avsnitt kommer att beskriva respektive problemgrupp och de fynd vi gjorde i dem, samt vilka slutsatser vi har dragit med hjälp av fynden och den teoretiska referensramen. Problemområde 5 är inte direkt kopplat till vårt syfte och vi har därför inte inriktat vår teori kring det. Dock medförde det stora problem vilka kan uppkomma i alla situationer där flera klienter arbetar mot en server och framför allt när det är stora datamängder som berörs. Det är därför sannolikt intressant läsning för den som skall utföra ett experiment likt vårt och det kommer därför att beskrivas.

1: Empiriska fynd rörande problematiken med AS/400 och den befintliga databasen

I ett tidigt skede av projektet undersöktes strukturen på den befintliga databasen och det upptäcktes ganska snart att detta skulle bli en mycket komplex situation. Antalet tabeller försvårade arbetet med att kartlägga strukturen. Men den första problemfaktorn låg i att antalet kolumner var mycket stort **(fynd 1.1)**. Antalet kolumner kunde i vissa tabeller vara upp mot 200 och dessa tabeller kunde även innehålla ca 700 000 rader. En tabell på 700 000 rader och 200 kolumner med helt oförståeliga kolumnnamn blir helt omöjligt att överblicka för en enskild person. Kolumnnamnens problematik noterade vi som **(fynd 1.2)**. För att lösa dessa problem sökte vi dokument över den befintliga databasen. Vi behövde inte leta länge innan vi fann ganska omfattande dokumentation om tabellerna och dess innehåll, medan dokumentation om strukturen på databasen inte gick att finna.

Bonifati (2001) anser att det är nödvändigt att den underliggande databasen studeras när ett DW skall utvecklas. Detta är något som vi också märkte snabbt och verkligen håller med om av två anledningar; dels för att förstå databasens uppbyggnad men också som Bonifati säger att se att datan verkligen är korrekt. Om vi hade varit tvungna att bygga upp den lokala databasen/DW utan något schema eller beskrivning över den underliggande datan så hade det tagit mycket längre tid. Tillvägagångssättet som vi använde var reverse engineering som Bonifati (2001) rekommenderar. Denna metod innebär att något bryts isär för att se hur det är uppbyggt och för att se hur det fungerar, för att senare kunna kopiera eller förbättra objektet. Då det var en mycket stor datamängd från början underlättade det mycket att bryta ner den i mindre delar för att få förståelse för dess uppbyggnad.

Databasens dokumentation löste en del av våra problem och vi kunde snart identifiera vilka tabeller som vi skulle behöva migrera till den nya databasen. Genom att bara läsa dokumentationen såg detta mycket enkelt ut, men mer problem fanns eftersom vi inte hade klart för oss vilka kopplingar dessa tabeller hade till varandra. Det fanns inte heller någon klar markering av vad som skulle kunna vara primärnycklar **(fynd 1.3)** i tabellerna. Att kunna identifiera primärnycklar för de olika tabellerna var av yttersta vikt för det nya systemet. Detta behövdes för att vi skulle kunna hämta uppdaterade data och föra in denna på rätt plats i det nya systemet utan att konflikt med tidigare data uppstår. I vissa tabeller gick det inte att hitta primärnycklar trots att vi i stort sett använda varenda kolumn som del av nyckeln. Detta berodde till stor del på att raderade poster sparades i databasen men pekades ut som raderade genom att varje tabell innehåller en kolumn som heter ”Delete Flag” och innehåller ett ”D” om posten är raderad **(fynd 1.4)**. En ytterligare orsak till det tidigare problem som vi betecknar som ett eget problem **(fynd 1.5)** var att det fanns skräptext som orsakade att det inte var möjligt att identifiera en unik post med hjälp av en primärnyckel.

Mullins (1996) nämner vikten av att datan är korrekt. Denna synpunkt ställde till det ordentligt för oss då grunddatan innehöll så mycket skräp att det inte gick att identifiera unika poster där vi behövde göra det. Vi var i många fall tvungna att manuellt gå in och rensa bort data för att kunna bygga upp vårt DW. Även strukturen på data var annorlunda än vi hade väntat oss. Poster som var självklara att använda som unika poster gick inte att använda på grund av bristande underhåll av databasen.

Vi antog att *skräpposterna* orsakats av eftersatt underhåll av den befintliga databasen. Underhållsbristen har vi betecknat som **(fynd 1.6)**. Vi antar att detta även beror på att skaparen av det befintliga databassystemet har tillåtit för många olika ändringar efter önskemål av användarna utan att se till vilka konsekvenser som detta har för systemet. Detta antagande bekräftas i våra observationer genom att det är tydligt att flera användare har fått sina ”småfixar” genomförda. Dessa fixar har orsakat att det har blivit möjligt att mata in egentligen icke tillåtna värden.

En ytterligare problematisk faktor som vi observerade i vår studie som rörde AS/400 var att systemet är uppbyggt som ett arvssystem **(fynd 1.7)**. Detta skulle egentligen

inte behöva vara ett problem men i och med att våra kunskaper om arvssystemet var mycket begränsade och ett AS/400-system är mycket slutet medförde detta komplikationer. Dessutom skiljer sig plattformen väsentligt ifrån den för det nya systemet.

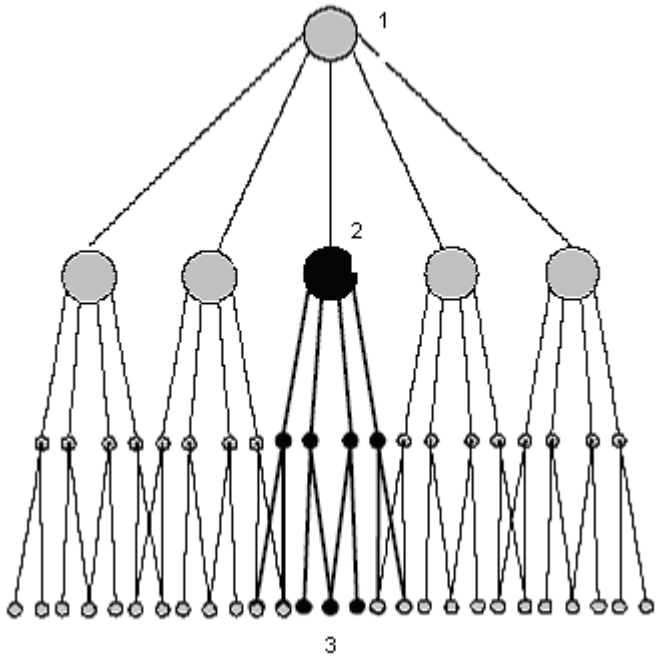


fig 4.1 Databasens uppbyggnad i AS/400. (www.krypton.msnu.edu)

Figur 4.1 visar ett exempel på hur arvsstrukturen i det gamla systemet kunde se ut. Nivå (1) består av masterfiler som innehåller grunddata om till exempel kunder, produkter, lager med mera. I nivå (2), finner man headerfiler som kan beskrivas som exempelvis fakturahuvuden. Ytterligare en nivå (3) ner finns detaljfilerna som hör till varje headerfil. I detaljfilen finns större delen av den föränderliga datan. I AS/400 sköter gränssnittet kopplingen mellan de olika nivåerna medan man i det nya systemet själv måste identifiera relationerna. Detta försvåras ytterligare av att det finns detaljfiler vars tillhörande header har raderats. Lösningen på detta var att kontrollera och radera de poster som var lösa i systemet. Vi såg även till att det i det nya systemet inte skulle vara möjligt att radera ett fakturahuvud utan att även radera detaljinnehållet **(fynd 1.8)**.

Empiriska fynd gällande problematiken med AS/400 och den befintliga databasen:

- Stort antal kolumner **(fynd 1.1)**
- Oförståeliga kolumn- och tabellnamn **(fynd 1.2)**
- Går inte att identifiera primärnycklar **(fynd 1.3)**
- Redundans i AS/400, borttagna poster markeras med ”D” **(fynd 1.4)**
- AS/400 innehåller skräpinformation som måste rensas manuellt **(fynd 1.5)**

- Bristande underhåll av databas (**fynd 1.6)**
- Arvssystem medför problem pga dålig kunskap **(fynd 1.7)**
- Raderas en huvudfil är det omöjligt att hitta tillhörande detaljfil **(fynd 1.8)**

Erfarenheter och rekommendationer kring problematik med AS/400:

Det är framför allt två aspekter som bör tas i beaktande gällande databasens uppbyggnad när information skall flyttas från ett system till ett annat, inte minst om de är byggda på olika plattformar.

1: Lägg tid på att förstå uppbyggnaden av den gamla databasen
2: Utgå inte ifrån att datan i databasen är strukturerad och lagrad enligt skolboken, det är den med stor sannolikhet inte.

Vi gjorde misstaget att försöka gå direkt på migrationen. Om vi hade lagt en vecka på att enbart få kontroll på strukturen i AS/400 och hur de tabeller vi skulle överföra och arbeta med så hade arbetet gått klart fortare. Vi trodde inte att sättet som de olika databaserna var uppbyggda på skulle skilja sig så mycket och framför allt inte att det skulle ställa till så mycket problem som det gjorde.

Databaser är ofta inte heller uppbyggda som de är i skolböckerna; att det är en tabell som heter hund och en som heter ägare och att en hund kan enbart ha en ägare medan en ägare kan ha många hundar. Precis som Mullins (1996) säger är underhållet av databaser ofta väldigt bristfälligt och inte något som vi tror är av hög prioritet för arbetsgivaren. Den kompetens som behövs för detta saknas dessutom ofta på företagen. När vi tittar tillbaka på den grunddata som vi har arbetat med så är vissa problem som vi stötte på helt ologiska. Varför lagras datum som ett tal och inte som ett datum? Varför kan inte en artikels nummer vara unikt i en tabell som enbart har som uppgift att lista de olika artiklarna och information om dessa? Det är aspekter som får accepteras men man bör vara medveten om detta när utformandet av den nya databasen börjar. Databaser är ofta väldigt ”skräpiga” och den ena är inte den andra lik. Läggs god tid på att förstå uppbyggnaden av databasen kommer man ha igen det med hästlängder längre fram i projektet. Att försöka få fram en beskrivning över uppbyggnaden av databasen är inte bara till stor hjälp utan i många fall ett krav för att arbetet skall kunna utföras på en acceptabel tid. Även om det inte finns något nedskrivet så vet ofta användarna mer om det än de själva tror. Att använda deras kunskap om uppbyggnaden kan i många fall vara ovärderligt.

2: Empiriska fynd rörande problematik med användare/beställare i migrationsprojektet

Vid ett tidigt skede i migrationsprojektet fanns många oklarheter om vilka data som skulle migreras. Olika tilltänkta användare av det nya systemet hade olika uppfattning vilken data som behövdes. Denna situation komplicerades genom att det nya systemet

främst skulle användas av personer som arbetade med försäljning inom två helt olika produktområden och som tidigare hade strukturerat datan helt olika (**fynd 2.1**).

Kelly & Nelms (2003) beskriver att detta problem hör till de vanliga problem-faktorerna vid val av data som skall migreras.

När vi fick klara besked om vilka data som skulle migreras upptäcktes ett nytt och lika komplicerat problem. Vi blev tvungna att bestämma med vilken noggrannhet datan skulle vara av i det nya systemet jämfört med det gamla (**fynd 2.2**). Med noggrannhet menar vi exempelvis hur många kolumner en tabell behöver ha för att informationen den innehåller skall vara tillräckligt detaljerad för att täcka organisationens behov. Vi fick direktiv om att försöka spegla det gamla systemet så långt som det var möjligt., det vill säga i det nya systemet efterlikna den noggrannhet som fanns i det gamla.

Noggrannheten på hur datan skall presenteras i det nya systemet beskriver både Hammer (1997) och Kelly & Nelms (2003). Kelly & Nelms menar att noggrannheten av den data som avses att migreras är mycket viktig eftersom det kan vara avgörande vilken data som betraktas gällande. Hammer menar att metadata ur det gamla systemet måste extraheras för att kunna avgöra med vilken noggrannhet data skall migreras.

Detta ansåg vi vara klara direktiv och mycket enkelt att följa men det komplicerades genom att direktiven om noggrannhet ändrades under migrationsarbetet (**fynd 2.3**). Nya direktiv efter det att vi hade påbörjat arbetet med att migrera data ifrån det gamla systemet blev en komplicerande faktor eftersom vi blev tvungna att förändra vårt ETL-verktyg varje gång detta inträffade. Detta kunde i värsta fall innebära att vi var tvungna att förändra hela verktyget för att tillgodose de nya direktiven och verktyget utgjordes av 80 000 rader skript.

Carrerira & Galhardas (2004) menar att en utvecklare av ETL-verktyg inte bara skall vara programmerare, han eller hon bör även vara affärsspecialist. Eftersom vi inte är affärsspecialister hade vi sannolikt svårare att förutse organisationens behov av olika data. Detta kan vara en av de orsaker till att vi fick nya direktiv efter det att arbetet med migrationen hade påbörjats.

Att migrera data ifrån ett arvssystem är ett mycket mer omfattande arbete än vad vi trodde det skulle vara från början. Det kan låta som ett okomplicerat ingrepp att flytta data från en plattform till en annan men så är inte fallet. Denna bild hade också beställarna/användarna av det nya systemet (**fynd 2.4**). I och med detta involverade vi dem mer i projektet under migrationsskedet för att de skulle få förståelse om att migrationen innebar ett mycket tids- och resurskrävande arbete.

När de involverades i projektet försökte vi även använda dem till att ta reda på den bakomliggande datastrukturen i det gamla systemet. Vi upptäckte att de hade viss kunskap om hur data var strukturerad men denna kunskap var begränsad till de data de normalt brukade använda (**fynd 2.5**). Den som hade den genomgripande kunskapen

om det gamla systemet var skaparen av det, vilket resulterade i att vi blev tvungna att fråga beställarna om de hade någon form av dokumentation av systemet. Det fanns en specifikation för varje fil eller tabell, (ca 630 stycken), i det gamla systemet, men att granska alla dessa skulle innebära att migrationsarbetet skulle ta betydligt längre tid än vad det gjorde.

Hammer (1997) anser att det kan vara svårt att finna dokumentation över det gamla systemet eftersom förändringar och utveckling av det gamla systemet kan ha pågått under flera år och ingen enskild person har förmodligen absolut kunskap om hur strukturen ser ut idag.

Empiriska fynd rörande problematik med användare/beställare i migrations-projektet

- Olika användare har olika önskemål om datastruktur och val (**fynd 2.1**)
- Olika användare hade behov av olika noggrannhet av data (**fynd 2.2**)
- Nya direktiv mitt under projektet (**fynd 2.3**)
- Användarna/beställarnas förförståelse av migrationsarbete stämde inte med verklig situation (**fynd 2.4**).
- Användarna saknar insikt hur datastruktur ser ut i nytt och gammalt system (**fynd 2.5**).

Erfarenheter och rekommendationer rörande problematik med användare/beställare i migrationsprojekt

Det är viktigt att involvera användarna på ett tidigt stadium i ett migrationsprojekt för att kunna erhålla värdefull kunskap om det gamla systemet. Även att tidigt kunna erhålla information om vilken data som skall migreras och vilken noggrannhet som krävs för att datan skall kunna tillfredställa organisationens behov är av stor vikt.

3. Empiriska fynd rörande de tekniska aspekterna med migrationen

Det första problemet med migrationen av datan upptäckte vi direkt när vi fick specifikationen av vad som skulle göras. Bandbredden skulle vara mycket begränsad (**fynd 3.1**) vilket skulle innebära att vi blev tvungna att utveckla ett ETL-verktyg som genomförde en stor överföring av data och sedan uppdaterade denna med många små körningar allt eftersom datan ändrades i det gamla systemet. Detta problem omfattande många delproblem som komplicerade vårt arbete. Exempelvis blev vi tvungna att ta reda på hur vi skulle kunna avgöra vilken data som var uppdaterad och inte.

Eftersom migrationen brutits upp i många små delar innebar det att istället för en stor körning blev vi tvungna att automatisera flera små komplicerade databaskörningar för att genomföra migrationen (**fynd 3.2**). Förutsättningen för att göra dessa körningar var att ingen samtidigt arbetade mot det gamla systemet. Detta skulle kunna innebära en

mycket stor tidsförlust samt att datan i det gamla systemet skulle förändras under migrationsarbetet. Därför förlades körningarna till natten.

Hammer (1997), Kelly & Nelms (2003), Carrerira & Galhardas (2004) menar i sina rapporter att det är extra viktigt att fånga upp de data som förändras under migrationsarbetet. Därför menar de att migrationen bör utföras vid ett tillfälle och att man om möjligt stoppar det gamla systemet. Om det inte är möjligt att stoppa systemet rekommenderar de att migrationsarbetet utförs då belastningen är minimal.

Hur vi skulle göra överföringen var något vi funderade en hel del på. Vi bestämde oss för att göra detta genom automatiserade DTS-paket som vi skapade i SQL Server.

Här fanns det två val:

1: Kopiera tabeller och views från källan.
2: Använd en fråga för att specificera vilken data som skall hämtas.

Om vi valt det första alternativet skulle vi helt enkelt hämta hem samtliga tabeller från biblioteket vilket rimligen borde vara väldigt okomplicerat. Detta fungerade dock inte av kompabilitetsskäl som vi förklarar nedan. Vi blev då tvungna att välja det andra alternativet och gå in specifikt för varje tabell och extrahera datan genom en SQL-fråga. Eftersom tabellerna var stora tog det relativt lång tid att överföra dem. Om de istället hade varit möjligt att välja alternativ nr. 1 hade vi enkelt kunna automatisera överförningen och tidsinställa den till att köras nattetid, då minsta möjliga realtidförändringar görs systemet (**fynd 3.3**).

I avsnittet om AS/400 visar vi dess uppbyggnad och förklarar där att den skiljer sig en hel del från andra system. Detta blev väldigt tydligt för oss när vi skulle lösa ovanstående problem.

När data skall överföras från AS/400 till PC-miljö krävs att data extraheras via SQL select-satser över en specifik ODBC (Open Data Base Connectivity) drivrutin. När alternativ 2 används så extraheras datan som skall överföras med hjälp av Client Access, en ODBC-applikation som möjliggör överföring av data ifrån AS/400 till PC-miljön. Nästa problem rör importen av data som skall göras varje natt. Vår plan för att få ett så optimerat bandbreddsutnyttjande som möjligt var att i ett första skede importera all information från servern och därefter varje natt hämta hem information som ändrats de senaste sju dagarna. Denna lösning var ganska självklar då varje tabell innehåller en kolumn som heter ”Registration date” och en kolumn som heter ”Update date”. Vi kan helt enkelt jämföra dagens datum med dessa datum och kolla om en post har blivit ändrad. Detta drar ner överföringstiden ifrån den globalt distanserade servern från flera timmar till ca 15 minuter. I och med att vi uppdaterar datan varje natt med de senaste sju dagarnas förändringar får vi en stabil säkerhetsmarginal, vilket innebär att om något skulle gå fel under en hämtning så repareras detta direkt vid nästa uppdatering. För att lyckas med detta var vi tvungna att ha ett bra och fungerande

datumformat *Här stötte vi dock på liknande problem som Mullins (1996) beskriver.* I AS 400 var datumen inte lagrade i ett datumformat utan som en tal (**fynd 3.4**).

Precis som Mullins (1996) nämner beträffande att befintlig grunddata ofta är av dålig kvalitet, vilket tidigare beskrivits, blev det också påtagligt för oss att även strukturen på datan kan ställa till problem. Även här var den modell över databasen vi fick tillgång till ovärderlig. Från början när vi inte hade någon beskrivning över vad de olika tabellerna stod för var det omöjligt att hitta någon kolumn som skulle kunna vara ett datum. Det hade varit nog svårt även om de hade varit lagrade i datumformat, nu var det näst intill omöjligt. Att titta på en tabell med 180 kolumner och förstå att värdet ”1010” (001010) skulle beteckna ett datum var inte direkt logiskt.

Vi förstod ganska snart att de måste ha haft vissa svårigheter vid millennieskiftet eftersom exempelvis datumet 2004-10-10 skrivs 41010 i det gamla AS/400-systemet. På alla tal som börjar med noll tas alltså nollan bort **(fynd 3.5)**. För att lösa detta problem i övergången mellan systemen skapade vi en algoritm som tolkade datumet och ersatte detta med ett nytt korrekt format.

Hammer (1997) menar att data kan presenteras olika i olika system. I detta fall klipps till exempel alla nollor av om de kommer före en värdesiffra medan de i nya systemet behålls före. Stycket ovan visar en tydlig indikation på detta fenomen. Talet 41010 tolkas normalt som ett tal och inte ett datum medan 2004-10-10 klart och tydligt indikerar ett datum efter gällande standard för datumformat i Sverige.

För att genomföra migrationen av data blev vi tvungna att finna ett sätt för att det gamla systemet skulle kunna kommunicera med det nya. Eftersom det nya systemet skulle baseras på Microsoft SQL Server behövde vi en utvecklingsplattform som skulle kunna kommunicera med de båda systemen. Vi fann därför ganska snart att det var stor fördel att använda ett högnivåspråk ifrån Microsoft eftersom de stod bakom även SQL Server. Vi valde VB (Visual Basic), som är ett högnivåspråk vilket innebär att koden måste kompileras av en körningsmiljö varje gång den skall exekveras. Detta är visserligen ineffektivt i jämförelse med ett lågnivåspråk (**fynd 3.6**) men innebar fördelar i form av till exempel tidsvinster vid utveckling som ändå gjorde detta till ett bra val.

Carrerira & Galhardas (2004) nämner även att det hör till vanligheten att ett högnivåspråk används när ett ETL-verktyg skall utvecklas, eftersom de som vanligtvis skriver migrationskod inte är renodlade programmerare utan även affärsspecialister. Trots att vi är mer programmerare än affärsspecialister valde vi ändå att använda ett högnivåspråk av de skäl vi beskriver ovan.

Som tidigare nämnts behövde vi programmera om vårt ETL-verktyg till att kunna bryta ned tabellerna i mindre delar innan det skulle extrahera och transformera datan. Denna metod resulterade i att det blev extra viktigt att kontrollera att datan som migrerats var korrekt och inte hade förändrats under migrationen. Det enklaste sättet att kontrollera detta var att skapa utdrag i Microsoft Excel för att jämföra poster ifrån

de båda systemen. Återigen blev begränsningen på Microsofts produkter ett faktum; MS Excel klarar endast att hantera 65000 rader per bok (**fynd 3.7**). Här blev vi tvungna att jämföra det som gick och hoppas på det bästa. Det visade sig dock senare att allt det vi hade extraherat stämde med källan.

Kelly & Nelms (2003) hade samma problem när de avsåg att kontrollera den migrerade datan. De försökte även verifiera den migrerade datan med tidigare versioner av Excel och upptäckte då att antalet tillåtna rader var ännu mer begränsat. De nämnde även metoder för att kontrollera datan. Vi blev tvungna att testa alla dessa metoder eftersom tabellerna i det gamla systemet varierade till storlek och struktur vilket gjorde det omöjligt att följa en enstaka strategi som skulle kunna tillgodose kraven.

Empiriska fynd rörande de tekniska aspekterna med migrationen

- Bandbredd begränsade tillgången till servern **(fynd 3.1)**
- Många komplicerade DB-körningar **(fynd 3.2)**
- Går ej att migrera tabellerna direkt till SQL server, SQL-frågor mot AS/400 måste ställas för att kunna extrahera data **(fynd 3.3)**
- Datum lagrat som tal istället för ett datum**. (fynd 3.4)**
- Börjar något med en nolla är den avkortad **(fynd 3.5)**
- Var tvungna att byta utvecklingsmiljö p.g.a. kompabilitetsproblem **(fynd 3.6)**
- Begränsning i MS Excel på 65000 rader **(fynd 3.7).**

Erfarenheter och rekommendationer kring problematik med de tekniska aspekterna i migrationsarbetet

För att minimera problematiken med de tekniska aspekterna i ett migrationsarbete bör man först studera de båda systemens möjligheter och begränsningar och försöka att hitta information ifrån andra som genomfört detta moment tidigare. Vid utvecklandet av ett eget ETL-verktyg rekommenderar vi användandet av ett högnivåspråk eftersom det är användarvänligt och tidsbesparande.

4: Empiriska fynd rörande problematiken med Data Warehouse

Då hämtningen av data från AS/400 var klar började arbetet med att bygga upp vårt DW som rapportgeneratorn skulle arbeta med. Tanken från början var att arbeta direkt mot de tabeller som vi hämtat hem. Detta visade sig dock snabbt vara helt omöjligt, då tabellerna var många och mycket stora **(fynd 4.1)** vilket gjorde att svarstiderna var långt ifrån acceptabla. Vi skapade därför nya tabeller vilka var specialanpassade för DW och de rapporter som skulle genereras.

Definitionerna av ett DW visar att det var ett bra val i vårt fall. Kimball (1998) säger till exempel: ”Ett DW är en kopia av transaktionsdata strukturerad för frågor och analyser”. Detta var ju precis det vi skulle göra.

Vi hade lite olika idéer om dels hur vi skulle bygga upp DW men även vilken information det skulle kunna presentera.

Precis som Zagelow (1997) säger så är det viktigt att man avgränsar sig. Användarna vill ju naturligtvis att DW, på ett överskådligt sätt skall kunna visa all information om allt. Detta är som sagt ingen avgränsning och då kan man hålla på i evigheter med sitt DW. Det vi till slut kom fram till var att bygga upp två olika sorts grundrapporter som beroende på hur frågan filtrerades kunde visa försäljningsstatistik som täckte de flesta av företagets behov. Att vi enbart inriktade oss på försäljning och inte tog med någonting om till exempel lagervärde med mera gjorde att antalet inblandade tabeller inte var så stort. Vi hämtade dock fortfarande hem samma mängd tabeller så att en eventuell utbyggnad av systemet skulle gå så smärtfritt som möjligt.

Resultatet av detta blev till slut sex stycken olika tabeller som hade lite olika funktioner **(Fynd 4.2).** Två tabeller har enbart till uppgift att bygga upp de övriga fyra tabellerna. Det är dessa fyra tabeller som rapportgeneratorn arbetar mot för att få fram rapporter och försäljningsstatistik. Den stora skillnaden i svarstider vid arbete mot flera tabeller istället för ett denormaliserat DW beror på logiken som skall utföras. Med den lösning som nu används så är det i stort sett ingen logik alls under körning av rapporter. I vissa fall görs lite summeringar men oftast är det bara en select-sats som exekveras och datan ligger summerad och klar att hämtas i tabellerna. Svarstiden mot en fråga i rapportgeneratorn är nu bara några sekunder även om det är rapporter som visar väldigt mycket data. All logik som bygger upp tabellerna körs under natten och tar ca två timmar. Att lägga denna logik så att den körs vid generering av en rapport är då av naturliga skäl inget alternativ. Att denormalisera eller inte var alltså för oss inget val.

Mullins anser att denormalisering bör undvikas om prestandan är acceptabel. För oss var det som sagt inget alternativ alls att inte denormalisera. Vår tanke var att göra som Gupta rekommenderar, en ”Query Respons Time Constraint” för varje fråga, och på så sätt få koll på om vi behöver denormalisera vår databas. Detta visade sig dock inte nödvändigt att göra. Vi visste att alla frågor skulle köras under dagtid när användaren satt framför skärmen och att svarstider på närmare en timme inte var acceptabla. Detta var av förståeliga skäl inget som vi ens behövde fråga användarna om. Att denormaliserad data skulle vara svår att underhålla som Mullins (1996) säger håller vi inte riktigt med om. En normaliserad databas innehåller ofta fler och mindre tabeller där varje rad är lättare att identifiera. En tabell i ett Data Warehouse är visserligen ofta mycket större och det krävs ofta fler antal kolumner för att identifiera en unik post. Dokumenteras strukturen och uppbyggnaden av Data Warehouset så anser vi att den är lika lätt att underhålla som en normaliserad databas.

För att kunna börja bygga upp dessa tabeller var det första vi behövde göra att översätta arvsstrukturen till relationsliknande struktur för att få förståelse för uppbyggnaden **(Fynd 4.3)**. Vi gick då tillbaka till det schema som vi hade fått över hela grunddatabasen. Denna gång var inte syftet att förstå strukturen i arvsmiljön. Nu var det viktiga att hitta allt som kunde liknas vid relationer eller möjliga unika nycklar för varje tabell **(Fynd 4.4)** för att göra uppbyggnaden av våra tabeller samt uppdatering av de tabeller som de utgår ifrån möjlig. Byggandet av våra tabeller som rapportgeneratorn arbetar mot delades upp i tre steg, vilka samtliga skedde nattetid:

1: Hämta hem data som skapats eller uppdaterats den senaste veckan.
2: Uppdatera de skarpa tabeller som ligger i databasen med den senaste veckans ändringar (det blir bara en dag som uppdaterats då vi varje dag hämtar en veckas information som säkerhet).
3: Skapa de tabeller som rapportgeneratorn arbetar mot grundade på de skarpa tabeller som nu innehåller all information vi behöver.

För att veta vilka rader som skulle uppdateras varje natt var det alltså nödvändigt att varje rad kunde särskiljas. Detta låter väldigt naturligt och enkelt men som vi tidigare sagt; räkna inte med att en databas ser ut som i skolboken för det gör den inte. Vi använde oss återigen av ”Reverse Engineering” **(Fynd 4.5)** för att få förståelse för hur alla tabeller var relaterade med varandra eller kunde utformas för att kunna få ut exakt den information som skulle behövas. Även när vi gjort detta och tyckte att vi hade databasstrukturen så klar att vi kunde börja bygga upp våra egna tabeller stötte vi på problem. Systemet som skapas genom att en del av den gamla databasen migreras ska hantera försäljningssiffror och därför endast transaktionsdata. Denna transaktionsdata består främst av fakturafiler. Eftersom det endast var fakturafiler antog vi att det inte skulle vara speciellt svårt att identifiera unika poster då det borde finnas unika fakturanummer. Våra observationer motbevisade detta genom att organisationen återanvände fakturanumren **(Fynd 4.6)** för internfakturorna inom koncernen. Vi blev tvungna att bryta ned filerna till en fil för varje år, fem år tillbaka i tiden. Eftersom samma fakturanummer inte användes under samma bokslutsår löste detta problemet.

Vi hade under vårt projekt kontinuerlig kontakt med en anställd på företaget som följde vårt arbete, gav instruktioner och ”knuffade oss i rätt riktning”.

Precis som Glassey-Edholm (1997) säger anser vi också att det är väldigt viktigt att hela tiden ha användarens perspektiv i åtanke. Detta för att försäkra sig om att utvecklare och uppdragsgivare hela tiden arbetar mot samma mål.

Detta gjorde att vi i slutfasen av projektet, när systemet demonstrerades, verkligen hade utvecklat ett system som motsvarade uppdragsgivaren krav och önskemål. När demonstrationen var klar var programmet också helt klart, vi slapp massa merarbete tack vare att vi hela tiden hade kontakt med uppdragsgivaren.

Empiriska fynd rörande problematiken med Data Warehouse

- Stora tabeller framtvingade utvecklandet av Data Warehouse **(fynd 4.1)**
- Fick bygga upp sex tabeller för att kunna lösa alla problem **(fynd 4.2)**
- Arvsstrukturen var tvungen att översättas till relationsstruktur **(fynd 4.3)**
- Tvungen att hitta primärnycklar för att kunna uppdatera tabellerna **(fynd 4.4)**
- Använde reverse engineering för att få koll på strukturen **(fynd 4.5)**

Erfarenheter och rekommendationer rörande problematik med Data Warehouse:

De viktigaste slutsatserna som vi dragit av de fynd vi gjort under vårt experiment kan sammanfattas i tre punkter:

1: Denormalisering kan korta ner svarstider väldigt mycket.
2: Förståelse för grunddata och att den är korrekt är mycket viktigt.
3: Att ha en kontinuerlig kontakt med uppdragsgivaren sparar mycket tid i slutet.

Att denormalisera är någonting som viss forskning anser bör undvikas, dock med reservationen för att prestanda skall vara godtagbar. För oss var det som beskrivits tidigare inget alternativ att inte denormalisera. Våra svarstider kortades ner något otroligt och därför är detta någonting som vi verkligen anser bör göras vid prestandaproblem. Vi håller inte heller med om att denna typ av data skulle vara svår att underhålla. *Denormalisera mera* anser vi.

Även i denna del av arbetet upptäckte vi vikten av att ha förståelse för vilken typ av data man arbetar med. I stora databaser är det otroligt tidskrävande att kontinuerligt sitta och försöka förstå den aktuella datan. Att få tillgång till förklaring över data man arbetar med är av yttersta vikt, antingen om det är nedskrivet eller att man tar tid på sig att diskutera med personal på arbetsplatsen. Sammanställs detta sedan i ett strukturerat dokument, som kan tas fram vid behov, så blir arbetet mindre tidskrävande.

Kontinuerlig kontakt med uppdragsgivaren under arbetet kan undvika onödigt extraarbete i slutfasen av projektet.

5: Empiriska fynd rörande prestandaproblem under utvecklandet av DW

Vi fick stora prestandaproblem under utvecklandet av DW och framför allt under testfasen. Detta kom vi under tiden och efteråt fram till berodde på följande faktorer:

Under utvecklingen hade vi tillgång till tre stycken helt nya Windows-bestyckade datorer. En av dessa fungerade som server och körde även Windows 2000 Server operativsystem. På denna dator låg således även SQL Server-programvaran. De andra två, vilka hade Windows XP installerade, användes som klienter och arbetade mot

databasen på servern. Under utvecklingsarbetet användes dessutom även den dator där serverprogramvaran var installerad till att ställa SQL-frågor **(fynd 5.1).**

På serverdatorn låg bland annat utvecklingsmiljön VB.NET installerad eftersom rapportapplikationen utvecklades bland annat på denna dator. Eftersom denna utvecklingsmiljö är relativt prestandakrävande påverkades hela systemet kraftigt när SQL-frågor kördes mot servern samtidigt som utvecklingsarbete skedde på VB.NET, jämfört med när inga andra applikationer kördes på den **(fynd 5.2).** Även mindre program som MS Word och MS Internet Explorer påverkade totalprestandan. Eftersom vi av nödvändighet behövde använda samtliga datorer vid utvecklingsarbetet blev detta alltså en faktor som gjorde arbetet mer tidskrävande för oss. Detta var visserligen en irriterande faktor under utvecklingen men eftersom Data Warehouset skall ligga på en dedikerad server för detta ändamål är det dock inget som kommer att påverka systemet när det är implementerat hos företaget **(fynd 5.3).**

Det faktum att alla tre datorer, alltså även den där serverprogramvaran var installerad, ställde SQL-frågor mot servern, var ytterligare en faktor som bidrog till att dra ner prestandan oerhört mycket. Eftersom sannolikt flera klienter kommer att arbeta mot samma server samtidigt när programmet implementeras på företaget gav detta oss betänkligheter. Detta tillsammans med ”applikationsbelastningen”, det vill säga att vi behövde köra tunga program på serverdatorn, bidrog till att även klienterna som arbetade mot databasen påverkades kraftigt. De frågor vi ställde i detta skede var dessutom ofta mycket komplicerade eftersom de innehöll både mycket data samtidigt som datan hämtades från ett flertal olika tabeller **(fynd 5.4).**

När vi hade färdigställt de slutgiltiga sex tabeller som vi byggde för vårt Data Warehouse reducerades prestandaproblemen dock kraftigt. Det vi kunde påverka i hårdvaruväg med tanke på att datorerna var nyinköpta var att expandera RAM **(fynd 5.5).** Vår uppdragsgivare ställde gärna upp med detta och genom att köra serverdatorn med 1 GB RAM påverkades prestanda ytterligare i positiv riktning. Byte av hårddiskar till SCSI-varianter diskuterades också men med tanke på att Data Warehouset ändå skulle köras på en dedikerad server senare beslöt vi att inte genomföra ett hårddiskbyte. Problemet med att utvecklingsmiljön fortfarande måste köras samtidigt med SQL-frågor kvarstod till viss del givetvis men genom ovanstående nämnda åtgärder tillsammans med att vi försökte samordna våra olika queries förbättrades prestanda avsevärt i utvecklingsskedet.

Empiriska fynd rörande prestandaproblem under utvecklandet av DW

- Servern används som utvecklingsklient **(fynd 5.1)**
- VB.NET används som utvecklingsmiljö **(fynd 5.2)**
- Server kommer ligga separat när det är färdigt **(fynd 5.3)**
- Komplicerade DB frågor **(fynd 5.4)**
- Mer RAM-minne **(fynd 5.5)**

Erfarenheter och rekommendationer kring problematik med prestanda under utvecklandet av DW

Försök att dedikera en specifik dator att endast agera server och inte köra flera tunga applikationer samtidigt på denna dator. Maximera RAM på serverdatorn och försök att samordna de tunga SQL-frågor som kan behöva köras under utvecklingsfasen.

Diskussion

Vid första mötet med begreppet datamigration låter detta som en mycket okomplicerad operation, vi har dock under det här arbete erfarit att så inte är fallet. Att flytta data mellan två olika typer av system kan till och med bli ett mycket komplext moment. I vår undersökning hade vi dock turen att existerande drivrutiner fanns för att extrahera data ur AS/400. Detta medförde att vi inte behövde undersöka algoritmerna bakom själva dataformateringen utan kunde koncentrera oss på övriga problem som kan uppstå vid en datamigration. I denna uppsats har vi beskrivit de fynd vi gjort vid utvecklandet av det rapportsystem och Data Warehouse som vi utvecklat för att undersöka problematik vid datamigration. Det finns självklart saker man funderat över under resans gång. I detta kapitel diskuteras moment som eventuellt kunde ha genomförts annorlunda samt alternativa upplägg om förutsättningarna varit andra än de som nu var fallet.

När vi sökte på tidigare forskning som gjorts inom detta område fann vi mycket få artiklar. De flesta artiklar som beskrev datamigration handlade oftast om vilka algoritmer som krävdes för att data skulle kunna migreras mellan system. Denna nivå ansåg vi vara för teknisk inriktad för att passa vår undersökning och dessutom fanns det redan bra verktyg för att extrahera data. Vi fann dock några artiklar som beskrev olika former av problem eller utmaningar som kan uppkomma när man vill migrera data till nya system eller Data Warehouses. Vi saknade dock fortfarande *modeller* för hur en datamigration borde genomföras. Det finns många bra metoder och modeller på hur man bör leda projekt, utveckla applikationer eller analysera och designa system. Tyvärr verkar det dock inte finnas någon modell eller metod som beskriver tillvägagångssättet vid migration av data från ett arvssystem till nya system. Det faktum att datamigationsprojekt ofta är olika från fall till fall är troligen en bidragande orsak till bristen på standardiserade modeller och metoder. I vårt fall fick vi använda våra förkunskaper och kreativitet för att skapa en egen modell för hur vi skulle utföra vårt experiment.

Allteftersom studien fortskred och våra kunskaper om ETL-verktyg och användandet av systemens befintliga funktioner för extraktion av data ökade, kunde vi konstruera ett effektivt och automatiserat ETL-verktyg. Den information som fanns om konstruktion av ETL-verktyg var mycket knapphändig och oftast mer förvirrande än till hjälp, det borde forskas mer om utveckling av ETL-verktyg eftersom organisationernas affärsdata idag, 2005, har blivit mer komplicerad. Detta har medfört att fler organisationer börjat se sig om efter nyare affärssystem med stöd för denna mer komplicerade affärslogik. Resultatet av detta blir att en komplicerad datamigration då blir en oundviklig punkt på schemat. I berörande artiklar har vi funnit stöd för att ETL-verktyg underlättar migrationsarbetet väsentligt.

Teorierna anser också att för att detta arbete skall lyckas krävs det att den eller de som skapar dessa verktyg bör vara både programmerare och affärsspecialister. Eftersom vi är mer programmerare än affärsspecialister så är våra kunskaper om affärslogik

begränsade medan våra kunskaper om systemdesign och programmering är bättre. För att förstå de affärslogiska faktorerna använde vi oss av en person inifrån organisationen som besatt kunskaper om detta område. Detta hjälpte oss att identifiera vilken data som skulle migreras. Precis som Kelly & Nelms (2003) säger var detta ett av de svåraste momenten vid migrationsarbetet.

Det svåraste problemet bestod dock i att identifiera strukturen på datan, både före och efter vi hade migrerat den. Med struktur menar vi i detta fall hur olika data hörde ihop sinsemellan. Framför allt var det svårt att identifiera de olika relationerna i de transaktionsfiler vi hade extraherat. För att kunna lösa detta problem blev vi tvungna att söka efter metadata som beskrev de olika filerna. Detta borde ha gjorts i ett mycket tidigare stadium i experimentet. När vi väl fått metadatan blev många samband mycket klarare och vi kunde förstå vilken data som vi behövde migrera för att det nya systemet skulle kunna innehålla tillräcklig information för skapandet av ett fungerande Data Warehouse. Vi ansåg att det här fanns två strategier för att identifiera vilken data vi skulle behöva. Den första strategin bestod i att *studera befintliga rapporter* som användes inom företaget. Genom att se vilken typ av information som användes valdes vilken data som skall migreras.

Den andra strategin bestod i att vi istället *studerade det gamla systemets data* och därifrån valde ut vilken data som rapporterna skulle bestå av, det vill säga att rapporterna struktureras utifrån datan. Vi valde den andra strategin eftersom den passade vårt system och våra förutsättningar bättre. En av anledningarna till detta var att det inte var givet att våra rapporter skulle presentera samma information som de redan befintliga rapporterna. I detta skede var det inte riktigt klart hos de tilltänkta användarna vilken data som skulle användas eller vilka rapporter de ville ha. Hade vi valt den första strategin så hade systemet blivit mer statiskt och inte lika utbyggbart som vårt system blev. Det visade sig dessutom i efterhand att de resulterande rapporterna skilde sig mycket från de tidigare rapporter vi tittade på. Detta hade medfört att vi hade migrerat fel sorts data med merarbete till följd för alla berörda parter om vi valt den första strategitypen.

Forskningen rörande datamigration till Data Warehouses och hur migrerad data bör lagras diskuterar denormalisering en hel del. Mullins (1996) anser att man bör akta sig för detta på grund av att denormaliserad data är svårare att underhålla. Som vi förklarade i resultatkapitlet håller vi inte med om detta. Anledningen till denormalisering är att få upp prestanda, om prestanda är acceptabel håller vi med om att denormalisering inte är nödvändigt. Dock anser vi att anledningen i så fall bör vara för att slippa det extra arbetsmoment som det innebär, inte för att slippa data som är svår att underhålla eftersom vi anser att den inte är det. Det var just denormaliseringen som i vårt fall ökade prestandan oerhört och vi anser verkligen att det är något man bör ha i åtanke vid utvecklandet av ett Data Warehouse.

Problemet med föränderlig data löste vi genom att hämta data nattetid då inga övriga transaktioner utförs och data därmed inte förändras. Detta medför en hög säkerhet eftersom till exempel en faktura aldrig kan uppdateras under tiden dess data migreras.

Om systemet tvunget skulle vara ett realtidsystem skulle vi bli tvungna att spegla datalagringen mellan de två systemen och detta skulle då leda till betydligt minskad prestanda i de båda systemen samt att data inte skulle vara lika tillförlitlig.

Vad gäller noggrannheten på migrerad data fick vi i början olika direktiv. Då datan kommer att ligga till grund för olika rapporter för olika avdelningar inom företaget skiljde det sig mycket på hur de ville ha datan presenterad. Data som exempelvis skulle visas för en kund skulle presenteras med 1000-kronorsavrundning medan annan data skulle ha en noggrannhet på öret. Vi gjorde därför inga avrundningar när vi hämtade den ursprungliga datan utan detta sker så sent som vid visning av respektive rapport.

Ser vi tillbaka på vår undersökning skulle vi kunna ha gjort flera saker annorlunda. Vi anser att undersökningen skulle ha underlättats om vi hade lagt mer tid på förstudien och insamlingen av metadata för det gamla systemet. Om detta hade gjorts vid ett tidigare skede skulle vi kunnat spara mycket tid som vi nu istället spenderade på att försöka att analysera strukturer och samband. För andra som avser att genomföra en liknande studie rekommenderar vi dem att i ett tidigt stadium göra efterforskningar om de olika systemen som man avser att migrera data mellan. Vi anser även att den undersökningsmetod som vi valde att använda för denna studie passade mycket bra, det finns dock andra undersökningsmetoder som skulle kunna appliceras på vår studie. Om exempelvis en ännu mera experimentell inriktning hade valts skulle nog inriktningen på undersökningen bli mer teknikfokuserad och omfatta mer effektivitet av verktyg, medan vårt fokus låg på problematiken med själva operationen.

Slutligen vill vi påpeka att de problem och råd vi analyserat och diskuterat bygger på att systemet är inaktivt under någon del av dygnet. Många prestandaproblem och säkerställande av datas tillförlitlighet löste vi genom att mycket arbete schemalades till natten. Om ett liknande projekt skall utföras i ett system som aldrig är inaktivt och data måste migreras under tiden som övriga transaktioner utförs så tror vi ändå att denna uppsats innehåller användbar information. Sannolikt kommer andra typer av problem att uppstå vid sådana typer av migrationsarbete som denna uppsats inte behandlar.

Slutsats

Ett datamigrationsprojekt är beroende av många olika föränderliga faktorer och det kan därför vara svårt att göra en generell beskrivning över de problem som kan tänkas uppkomma i ett liknande projekt. Trots detta och i enlighet med uppsatsens syfte går det att sammanfatta en beskrivning av de problem som kan anses som universella vid ett migrationsarbete, dessa är följande:

1: *Den strukturella skillnaden i de olika databaserna, hur data lagrats*

Våra förkunskaper inom databasdesign begränsade sig i princip till att omfatta endast relationsdatabaser. Att initialt lägga mycket tid för förståelse om hur den lagrade datan struktureras i ett arvssystem kommer att medföra att migrationsarbete underlättas väsentligt.

2: *Den specifika databasens uppbyggnad.*

Att förstå uppbyggnaden av en stor databas som man aldrig tidigare sett utan att ha någon form av beskrivning eller schema är mycket svårt. Även om två databaser är av samma typ bygger organisationer upp databaser olika beroende på deras affärslogik. Detta medför att även om man har förståelse för den fundamentala strukturen i det aktuella databassystemet, enligt punkt 1 ovan, så kan den affärslogiska strukturen i den specifika databasen vara helt oförståelig. Att hitta personer som har kunskap om databasen eller att få tag i någon beskrivning över den är en mycket viktig del vid val av vilken data som skall migreras.

3: *Databasernas tekniska skillnader vad gäller exempelvis format*

Ett återkommande problem under vårt migrationsarbete har varit skillnaden på format i de olika databaserna samt bristande underhåll i grunddatabasen. Efter att man har fått klart för sig vilken data som skall migreras är det av stor vikt att det fastställs att denna data har samma uppbyggnad i det gamla systemet som den skall ha i det nya, samt att det inte innehåller skräp på grund av dåligt underhåll då detta kommer att ställa till stora problem när tabellerna i Data Warehouse skall utformas. Vi stötte i vår undersökning bland annat på att ett datum i den gamla databasen lagrades som ett tal med bortskalade nollor i början, vilket gjorde att fel värden fördes in i vårt Data Warehouse. Datumet 2004-10-10 skrevs till exempel i arvssystemet som ”41010”.

4: *Problematiken med kontinuerlig kontakt med uppdragsgivare*

Ett stort problem inom all utveckling som rör mer än en part är att utvecklare och användare inte har samma syn på vad som skall utföras. Dessutom ändras ofta organisationens önskemål under arbetets gång samtidigt som ett flertal personer inom organisationen kommer med olika önskemål. Att ha en kontinuerlig kontakt med uppdragsgivaren reducerar risken att man i slutet av projektet får ändra redan utförda moment på grund av att utvecklare och användare missförstått varandra. I vårt fall hade vi kontinuerlig kontakt med de olika berörda parterna inom organisationen för att säkerställa att tillräcklig mängd, och korrekt, data migrerades för att tillfredsställa alla parters behov.

5: *Problematiken med kompatibla programvaror*

Vår tanke från början var att arbeta med de utvecklingsmiljöer som vi ansåg oss ha bäst kunskap om. Vi reviderade dock denna tanke då vi stötte på kompabilitetsproblem. Att använda mjukvara från samma tillverkare kan i många fall innebära att man tjänar igen den tid de tar att lära sig den nya mjukvaran.

6: *Otillräcklig forskning och dokumentation över migration*

Det första vi gjorde när själva datamigrationen skulle utföras var att leta efter andra teorier och metoder för detta. Det finns gott om teorier om datamigration men dessa är ofta väldigt tekniskt inriktade och beskriver olika algoritmer på binärskodsnivå för transformering av data. Att hitta metoder och modeller som beskriver själva tillvägagångssättet var svårt. Detta är något vi saknar och anser att det finns ett behov av.

Genom att ta ovanstående punkter i beaktande kan problem vid migration av data från ett arvssystem till ett Data Warehouse reduceras väsentligt. Vi hoppas med denna uppsats kunna bidra till att effektivisera migrationsprojekt av den typ vi beskriver. Flera av punkterna kan vid genomläsning och i efterhand verka uppenbara men när vi stod inför dem var fallet inte så. Genom att i förhand veta var störst resurser bör läggas ökar förutsättningarna att kunna genomföra ett så lyckat byte av bostad för ettor och nollor som möjligt.

Figurförteckning

Referenser

Agosta, Louis, (1999), *Data warehousing*, Prentice Hall, augusti 1999

Backman, Jarl (1998), Rapporter och Uppsatser, studentlitteratur, Lund, Sverige, 1998

Batini, C. & Lenzerini, M, (1984), A methodology for data schema integration in the entity-relationship model. *IEEE Trans. Softw. Eng. 10*, 6 (Nov.), 650–664. 1984

Bergqvist, Jens, (2002), *Designing for local mobility*, Papers in informatics, paper 10, November 2002, ISSN 1400-7428, Department of Informatics, Gothenburg University, Sweden

Bonifati, Angela & Cattaneo, Fabiano & Fuggetta, Alfonso & ParaBoschi, Stefano (2001), *ACM Transactions on software engineering and methodology, vol 10, Nr 4, October 2001*

Bontempo, C & Zagelow, G. (1998). *" The IBM data warehouse architecture"* Communications of the ACM , Volume 41, Issue 9, pp. 38-48.

Carreira, P. & Galhardas, H., (2004a), *Data transformation and duplicate detection: Execution of data mappers*, Proceedings of the 2004 international workshop on Information quality in information systems, Pages: 2 – 9, Paris, France ISBN:1-58113-902-0

Carreira, P. & Galhardas, H., (2004), *Efficient development of data migration transformations*, Proceedings of the 2004 ACM SIGMOD international conference on Management of data, Pages: 915 - 916, Paris, France *June 13 - 18, 2004* ISBN:1-58113-859-8

Connolly, Thomas & Begg, Carolyn, (1999), *Database systems: A practical approach to design, implementation and management, second edition:* Addison Wesley Longman limited.

Dayal, U. & Chaudhuri, S., (1997), *An overview of data warehousing and OLAP technology,* ACM SIGMOD Record, Volume 26 , Issue 1 (March 1997), Pages: 65 – 74, ISSN:0163-5808, ACM Press New York, NY, USA, 1997

Date, CJ., (2000), *An introduction to database systems,* Addison Wesley Longman Inc., 2000

Easterby-Smith, M & Thorpe, R & Lowe, (1991), A; *Management Research: An introduction* SAGE publications Ltd

Elmasri, Ramez & Navathe, Shamkant B, (2000). *Fundamentals of Database Systems.* (3rd ed) Addison-Wesley

English, Larry P., (1999). *Improving data warehouse and business information quality: Methods for educing and increasing profits.* John Wiley & Sons Inc.

Finnegan. P. & Murphy.C. & O'Riordan, J. (1999). *"Challenging the Hierarchical Perspective on Information Systems: Implications from External Information Analysis,"* Journal of Information Technology

Francalanci, C. & Fuggetta, (1997), *Integrating conflicting requirements in process modelling,* a survey and research directions, inf software technology 39, 3 205-216

Gupta, H (1997), *Selection of views to materialize in a data warehouse.* In proc. Of the 6:th intl. conference on data base theory, 1997

Glassey-Edholm, Katherine (1997), *Bringing user perspective to data warehouses: Twenty-one points of consideration*:, Building, Using and Managing Data Warehouse, Prentice Hall

Hammer, Katherine (1997), *Migrating Data from legacy systems: challenges and solutions,* Building, Using and Managing Data Warehouse, Prentice Hall

Harper , R. H. R. (1999), *The organisation in ethnography- a discussion of Ethnographic Fieldwork Programs in CSCW*, Computer supported Cooperative Work 9: 239-264, 2000, Kluwer Academic Publishers, Nederländerna.

Inmon, V.H, (1993), *Building the data warehouse,* New York: John Wiley & Sons, 1993

Juric , Matjaz B. & Rozman, Ivan & Hericko, Marjan & Domajnko, Tomaz (2000) *Integrating legacy systems in distributed object architecture,* ACM SIGSOFT Software Engineering Notes ,Volume 25 , Issue 2 (March 2000) Pages: 35 - 39 ISSN:0163-5948

Kelly, C. & Nelms, C. (2003), *Roadmap to checking data migration,* 0167-4048/03 Elsevier Ltd 2003

Kimball, Ralph & Reeves, Laura & Ross, Margy & Thornthwaite, Warren. (1998) *The Data Warehouse Lifecycle Toolkit.* John Wiley & Sons Inc.

Ladaga, J. (1995). *"Let Business Goals Drive Your Data Warehouse Effort,"* Health Management Technology, Vol. 16, No. 11, pp. 26-28.

Love, B. (1996). *"Strategic DSS/Data Warehouse: A Case Study in Failure,"* Journal of Data Warehousing, Vol. 1, No. 1, pp. 36-40.

Mullins, Craig.S., (1996), *Dealing with data outhouses,* Craig S. Mullins & Associates, Inc 1996

Myers, M. (1995). *"Do You Really Need a Data Warehouse?,"* Network World, Vol. 12, No. 51, p. 26.

Park, Y.T. (1997). *"Strategic Uses of Data Warehouses: An Organizations Suitability for Data Warehousing,"* Journal of Data Warehousing, Vol. 2, No. 1, pp. 13-22.

Pierce, Elisabeth M, (1999), *Developing and delivering a data warehouse and mining course,* Indiana university of Pennsylvania, 1999

Plew. Ronald & Stephens. Ryan, (2000), *Lär dig SQL på 3 veckor, 3:e uppl*, Sundbyberg: Pagina Förlag

Scholerman, S & Miller, L & Tenner, J & Tomanek, S & Zolliker, M (1993), *Relational database integration in the IBM AS/400: Dept of computer science, Iowa state university*

Taha, Y. Helal, A., & Ahmed, K.M. (1997). *"Data Warehousing: Usage, Architecture, and Research Issues,"* Microcomputer Applications, Vol. 16, No. 2, pp. 70-80.

Thurén , Torsten , (1991), *Vetenskapsteori för nybörjare,* första upplagan, Liber AB

Zagelow, George, (1997), *Data warehousing – Client/Server for the rest of the decade*, Prentice Hall

Internetreferenser

Burke, Jason & Kurk, Andrea (2001), *Ethnographic Methods*, http://www.otal.umd.edu/hci-rm/ethno.html Maryland University october 2001 avläst : 2005-01-03

Langemar, Pia (2004) Grounded Theory, Stockholm University http://www.psychology.su.se/units/gu/pk/handoutsht04/kap14.pdf avläst: 2005-01-22

http://krypton.msnu.edu, *Introductory Reference to the IBM AS/400*, http://krypton.mnsu.edu/~j3gum/web/AS/400/intref.html, *Minnesota State University, Mankato 1992,1995* avläst: 2004-11-28

http://www.Webopedia.com, *Data Migration* http://www.webopedia.com/TERM/D/data_migration.html avläst: 2004-11-25

http://www.ibm.com, *AS/400 Storage Management*
http://AS/400bks.rochester.ibm.com/html/AS/400/v4r5/ic2924/index.htm?info/RZAHQSTORMGTCONCEPT.HTM avläst: 2004-11-02

http://www.bitpipe.com, *Delivering AS/400 Services to Microsoft Windows PC Clients*., http://wp.bitpipe.com/resource/org_905740864_63/140-7271.pdf avläst: 2004-11-02

Gupta, Vivek R. (1997) *An Introduction to Data Warehousing*.
http://www.sserve.com/dwintro.asp (2001-09-13) avläst: 2004-10-12

Gunnarsson, Ronny MD PhD, (2001), *Fenomenologi*,
http://infovoice.se/fou/bok/kvalmet/10000009.htm avläst: 2004-11-18

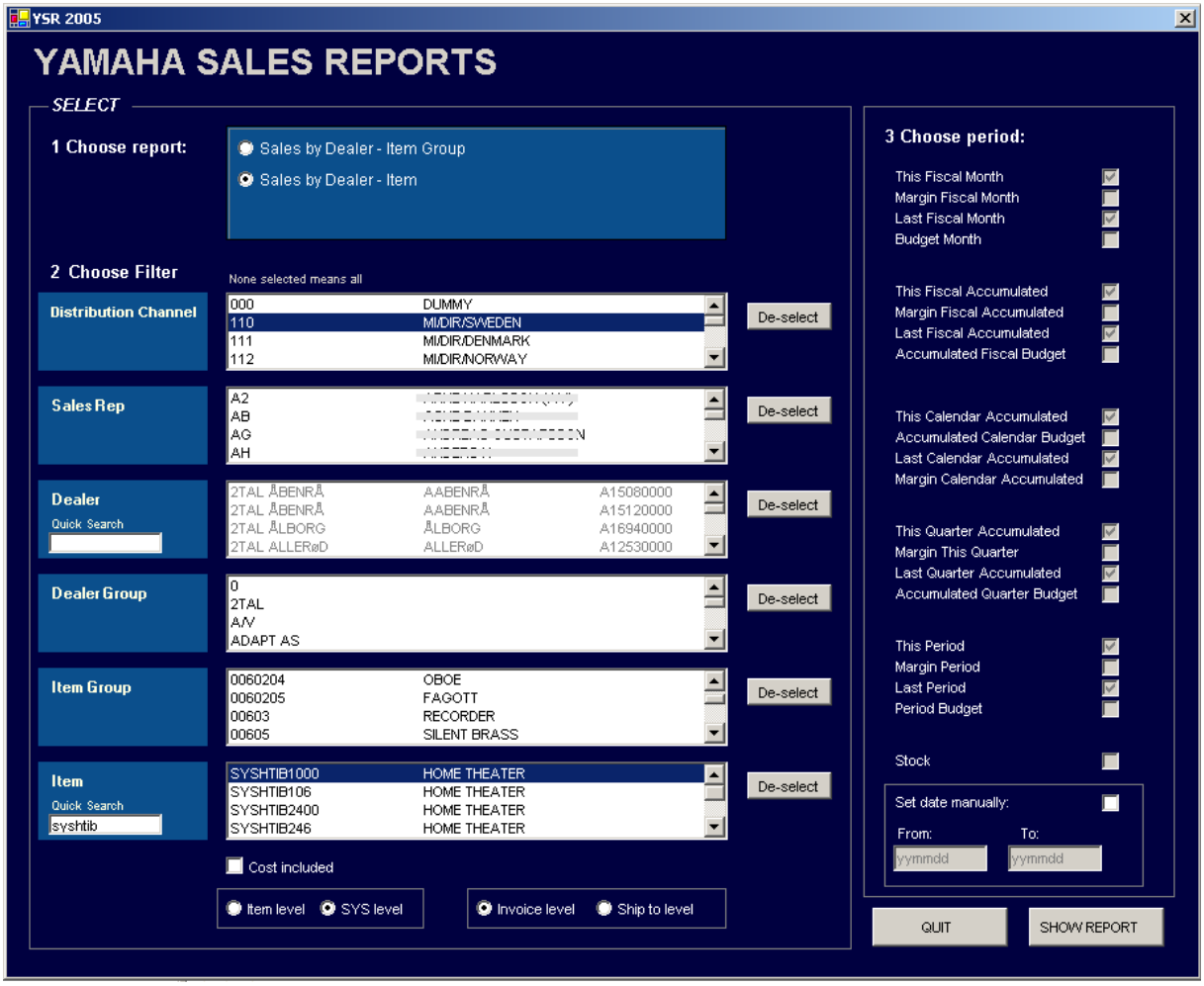
www.ibm.com, *Integrated File System Introduction*
http://publib.boulder.ibm.com/iseries/v5r1/ic2924/index.htm?info/ifs/rzaaxmststmfile.htm avläst: 2004-11-25

www.lida.liu.se *Distribuerade databaser*
http://www.ida.liu.se/~tompa/databaser/distribuerade.html avläst: 2005-01-03

idg.computersweden.se, *Tidsbesparande städhjälp i datastöket*
CS Nr 17 2000
http://domino.idg.se/cs/artikel.nsf/0/01c76a355bb422f1c125688c00454670?OpenDocument avläst: 2004-10-04

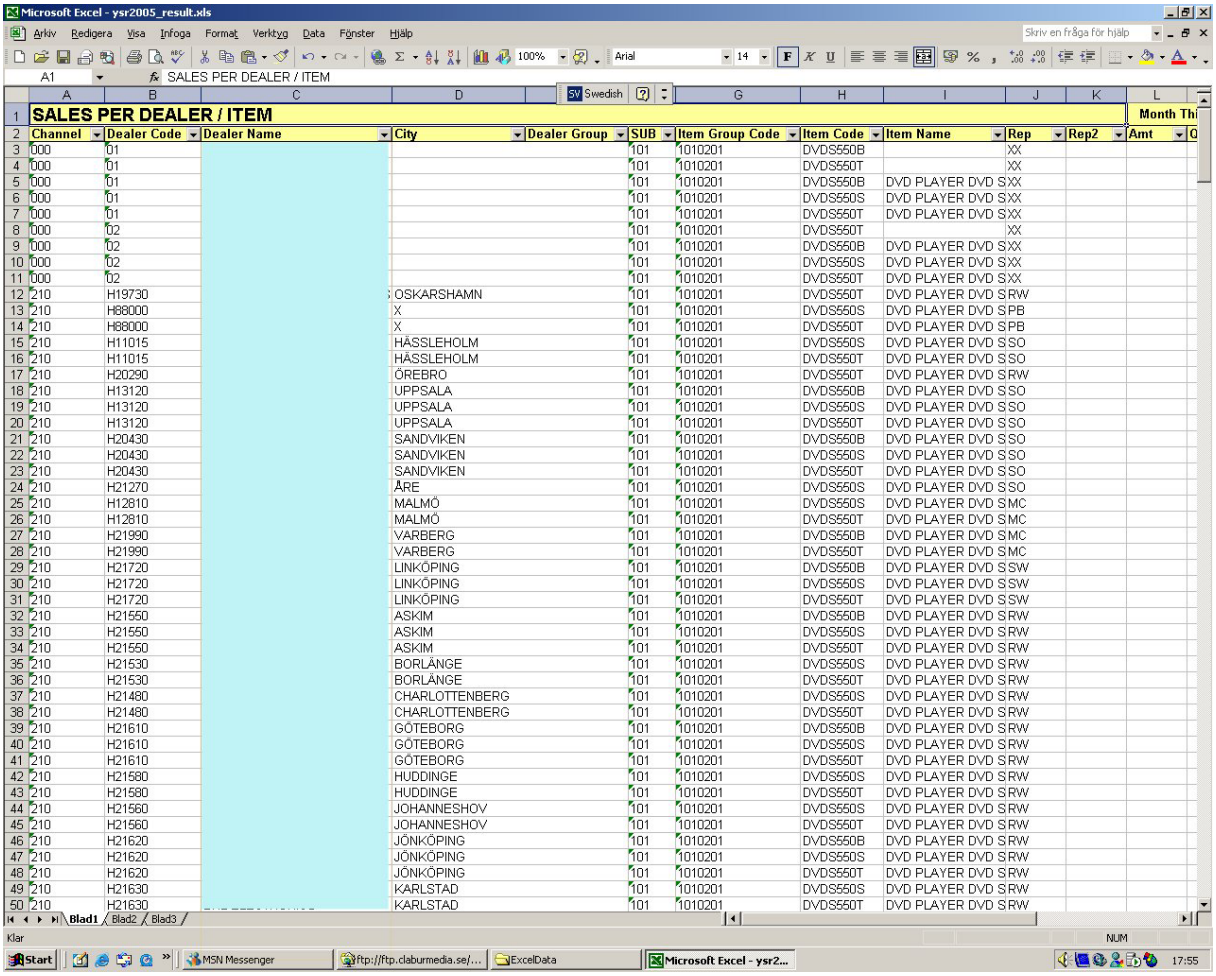
searchsmallbizit.techtarget.com, *Reverse engineering*
http://searchsmallbizit.techtarget.com/sDefinition/0,,sid44_gci507015,00.html avläst: 2004-10-18

Bilaga 1



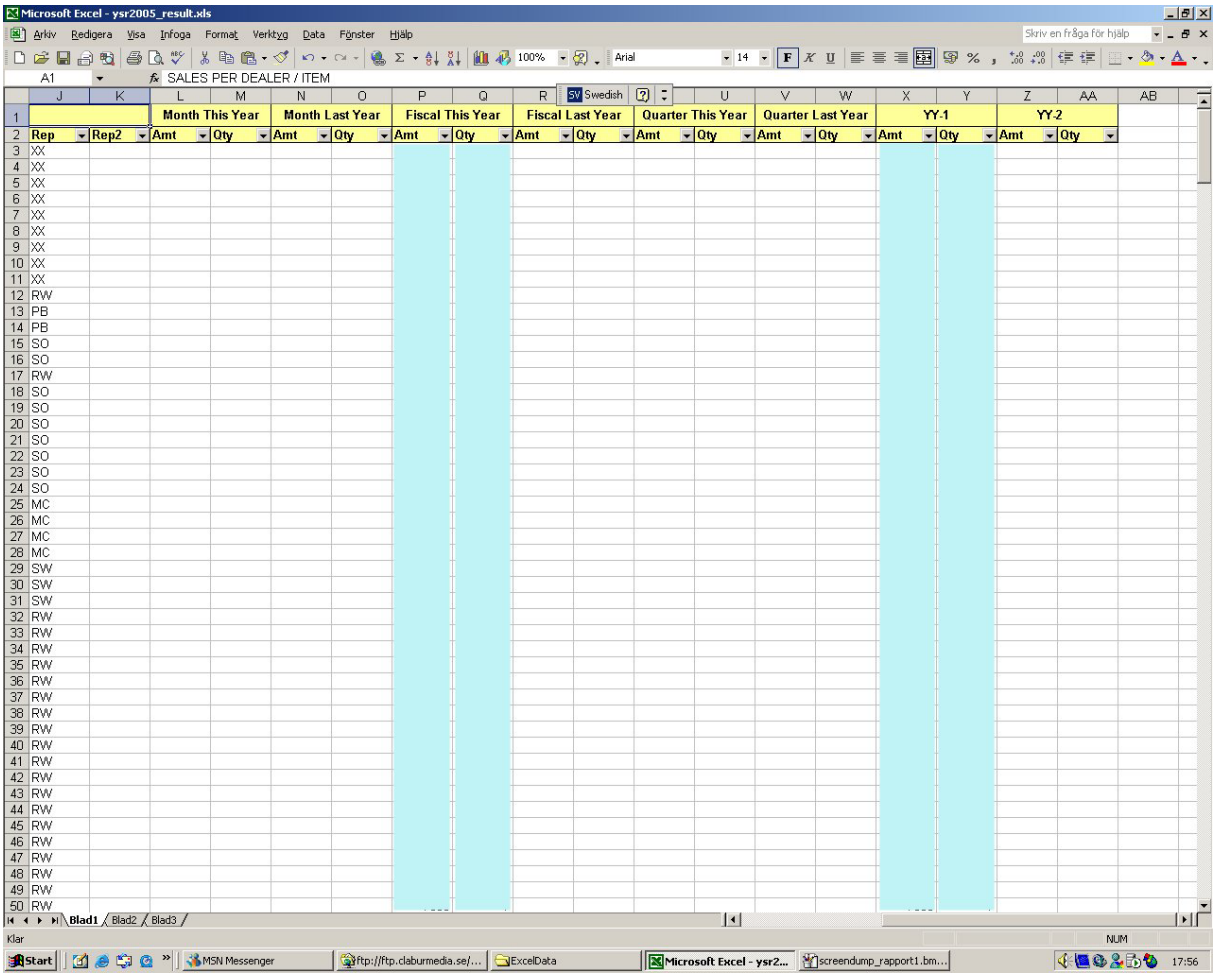
Den resulterande rapportgeneratorn –YSR 2005

En kortfattad beskrivning kommer här att ges av hur den rapportgenerator vi utvecklade används. Efter att ha loggat in via en dialogruta presenteras huvudskärmen enligt figur ovan. Användaren väljer först mellan att presentera rapporterna per *återförsäljare och produktgrupp* eller *återförsäljare och specifik produkt*. Efter detta ges möjlighet att välja ut olika sökbegrepp som *distributionskanal*, *säljare*, *återförsäljare*, *grupp av återförsäljare*, *produktgrupp* eller *produkt*. Eftersom *återförsäljare* och *produkt* innehåller en mycket stor mängd data skapades fält för dynamisk snabbsökning för dessa. Val för att presentera kostnader, paket- eller produktnivå och vilken typ av kundnivå är implementerat. Användaren väljer sedan vilken period man vill ha informationen presenterad för. Som standardläge visas fördefinierade perioder baserade på företagets affärslogik men om användaren önskar kan godtycklig period för rapporten matas in.



SALES PER DEALER / ITEM												
Channel	Dealer Code	Dealer Name	City	Dealer Group	SUB	Item Group Code	Item Code	Item Name	Rep	Rep2	Amt	Q
000	01				101	1010201	DVDS550B		XX			
000	01				101	1010201	DVDS550T		XX			
000	01				101	1010201	DVDS550B	DVD PLAYER DVD S	XX			
000	01				101	1010201	DVDS550S	DVD PLAYER DVD S	XX			
000	01				101	1010201	DVDS550T	DVD PLAYER DVD S	XX			
000	02				101	1010201	DVDS550T		XX			
000	02				101	1010201	DVDS550B	DVD PLAYER DVD S	XX			
000	02				101	1010201	DVDS550S	DVD PLAYER DVD S	XX			
000	02				101	1010201	DVDS550T	DVD PLAYER DVD S	XX			
210	H19730		OSKARSHAMN		101	1010201	DVDS550T	DVD PLAYER DVD S	RW			
210	H88000		X		101	1010201	DVDS550S	DVD PLAYER DVD S	PB			
210	H88000		X		101	1010201	DVDS550T	DVD PLAYER DVD S	PB			
210	H11015		HÄSSLEHOLM		101	1010201	DVDS550S	DVD PLAYER DVD S	SO			
210	H11015		HÄSSLEHOLM		101	1010201	DVDS550T	DVD PLAYER DVD S	SO			
210	H20290		ÖREBRO		101	1010201	DVDS550T	DVD PLAYER DVD S	RW			
210	H13120		UPPSALA		101	1010201	DVDS550B	DVD PLAYER DVD S	SO			
210	H13120		UPPSALA		101	1010201	DVDS550S	DVD PLAYER DVD S	SO			
210	H13120		UPPSALA		101	1010201	DVDS550T	DVD PLAYER DVD S	SO			
210	H20430		SANDVIKEN		101	1010201	DVDS550B	DVD PLAYER DVD S	SO			
210	H20430		SANDVIKEN		101	1010201	DVDS550S	DVD PLAYER DVD S	SO			
210	H20430		SANDVIKEN		101	1010201	DVDS550T	DVD PLAYER DVD S	SO			
210	H21270		ÅRE		101	1010201	DVDS550S	DVD PLAYER DVD S	SO			
210	H12810		MALMÖ		101	1010201	DVDS550S	DVD PLAYER DVD S	MC			
210	H12810		MALMÖ		101	1010201	DVDS550T	DVD PLAYER DVD S	MC			
210	H21990		VARBERG		101	1010201	DVDS550B	DVD PLAYER DVD S	MC			
210	H21990		VARBERG		101	1010201	DVDS550T	DVD PLAYER DVD S	MC			
210	H21720		LINKÖPING		101	1010201	DVDS550B	DVD PLAYER DVD S	SW			
210	H21720		LINKÖPING		101	1010201	DVDS550S	DVD PLAYER DVD S	SW			
210	H21720		LINKÖPING		101	1010201	DVDS550T	DVD PLAYER DVD S	SW			
210	H21550		ASKIM		101	1010201	DVDS550B	DVD PLAYER DVD S	RW			
210	H21550		ASKIM		101	1010201	DVDS550S	DVD PLAYER DVD S	RW			
210	H21550		ASKIM		101	1010201	DVDS550T	DVD PLAYER DVD S	RW			
210	H21530		BORLÄNGE		101	1010201	DVDS550S	DVD PLAYER DVD S	RW			
210	H21530		BORLÄNGE		101	1010201	DVDS550T	DVD PLAYER DVD S	RW			
210	H21480		CHARLOTTENBERG		101	1010201	DVDS550S	DVD PLAYER DVD S	RW			
210	H21480		CHARLOTTENBERG		101	1010201	DVDS550T	DVD PLAYER DVD S	RW			
210	H21610		GÖTEBORG		101	1010201	DVDS550B	DVD PLAYER DVD S	RW			
210	H21610		GÖTEBORG		101	1010201	DVDS550S	DVD PLAYER DVD S	RW			
210	H21610		GÖTEBORG		101	1010201	DVDS550T	DVD PLAYER DVD S	RW			
210	H21580		HUDDINGE		101	1010201	DVDS550S	DVD PLAYER DVD S	RW			
210	H21580		HUDDINGE		101	1010201	DVDS550T	DVD PLAYER DVD S	RW			
210	H21560		JOHANNESHOV		101	1010201	DVDS550S	DVD PLAYER DVD S	RW			
210	H21560		JOHANNESHOV		101	1010201	DVDS550T	DVD PLAYER DVD S	RW			
210	H21620		JÖNKÖPING		101	1010201	DVDS550B	DVD PLAYER DVD S	RW			
210	H21620		JÖNKÖPING		101	1010201	DVDS550S	DVD PLAYER DVD S	RW			
210	H21620		JÖNKÖPING		101	1010201	DVDS550T	DVD PLAYER DVD S	RW			
210	H21630		KARLSTAD		101	1010201	DVDS550S	DVD PLAYER DVD S	RW			
210	H21630		KARLSTAD		101	1010201	DVDS550T	DVD PLAYER DVD S	RW			

Exempelrapport från systemet. Företagskänsliga uppgifter har dolts.



Fortsättning rapport.