En metod för reengineering av databassystem

Maria Premmert

Åsa Tilly

Examensarbete I, 10 poäng

Vårterminen 1999

Göteborgs universitet

Institutionen för informatik

Handledare: Jan Ljungberg

Abstract. Vårt mål är att, utifrån befintlig litteratur inom områdena reengineering, databasutveckling, verksamhetsanalys och systemutvärdering, sammanställa en metod för reengineering av relativt små databassystem. Utifrån studien av litteratur inom ovan nämnda områden presenterar vi problematiken med systemarvet och föråldrade system (legacy systems), samt sammanställer ett första förslag till en metod. Denna metod testas sedan i en fallstudie, där ett mindre system med egenskaper liknande dem hos legacy systems omarbetas enligt metoden. En utvärdering av hur väl metoden fungerade görs och vissa förändringar av metoden föreslås i enlighet med problem som uppstod i fallstudien. Ett mer komplett förslag till metod där hänsyn har tagits till dessa förändringar presenteras slutligen. Sammanfattningsvis dras också slutsatsen att det ofta är relevant att använda sig av reengineering när man skall göra om gamla databassystem. Det konstateras också att det är viktigt att reengineeringsarbetet utförs strukturerat, och att vår metod i detta sammanhang fungerar utmärkt att arbeta efter.

1 Inledning

Det finns idag i många organisationer system som byggts för länge sedan men som fortfarande används i verksamheten. Många av dessa system konstruerades innan det fanns några strukturerade mjukvaruutvecklingsmetoder. Systemen har även under de år de har varit i bruk genomgått stora förändringar och utvecklats en hel del. Alla dessa ändringar och det faktum att systemen byggts utan att någon formell metod har följts, gör att systemen idag ofta är dåligt strukturerade och att dokumentationen kan vara bristfällig och föråldrad. De är svåra att underhålla och det är ofta svårt att göra ytterligare vidareutveckling av dem. Eftersom systemen fortfarande används i verksamheten och ofta även är verksamhetskritiska, går det inte bara att byta ut dem mot nya (Somerville, 1995; Gustafsson & Johansson, 1994). Dessa system kallas ofta *legacy systems*[1] (Somerville, 1995). Det samlade problemet som de utgör benämns med termer som *systemarv* (Gustafsson & Johansson, 1994) eller *software legacy* (Somerville, 1995).

En möjlig lösning på problemet med legacy systems är att använda *reengineering*. Detta innebär att man utvecklar ett nytt system utifrån det existerande systemet, vilket används som bas i den nya utvecklingsprocessen. Som en del i reengineering ingår ofta *reverse engineering*, vilket i princip innebär att man sätter sig in i det gamla systemet och försöker förstå hur det fungerar och vad det gör och sedan återskapar dokumentation för systemet utifrån dess källkod. Dokumentationen och den kunskap man fått fram om det gamla systemet används sedan för att gå vidare i reengineeringsprocessen. (Tilley, 1995)

Traditionellt har software engineering mest handlat om nyutveckling av system, som vid en viss punkt har ansetts vara ”färdiga” och som sedan tagits i bruk och därefter underhållits. Under nittiotalet har det dock skett en skiftning som innebär att man koncentrerar sig mer på vidareutveckling av befintliga system. Många system som nyutvecklas, liksom system som gjorts om med reengineering, anses egentligen aldrig vara färdigutvecklade – de förväntas vara operationella under en lång tid och vidareutvecklas kontinuerligt. Denna inriktning kan kallas *software evolution* och syftar till att utveckla *evolutionary systems*. (Müller et al., 1993)

Denna uppsats tar upp problemet med föråldrade system i ett specifikt sammanhang. Ett tillvägagångssätt för reengineering av databassystem sammanställs utifrån några befintliga modeller och metoder för systemutveckling. Detta tillvägagångssätt utvärderas sedan i en fallstudie där en gammal databas omformaliseras.

Systemet i fallstudien är relativt litet[2] och uppsatsen syftar därför inte till att försöka komma fram till någon generellt applicerbar metod för stora legacy systems. Trots detta kan studien anses vara relevant då det idag finns många mindre databassystem[3] i användning som, liksom det i fallstudien, lider av systemarvsliknande problem som exempelvis föråldrad struktur och bristfällig design.

---

[1] ungefär nedärvt system

[2] Det innehåller drygt 2800 poster

[3] Vi definierar ett mindre databassystem som ett system innehållande upp till ungefär 10 000 poster. Storleken beror naturligtvis också på systemets komplexitet och omfattning. Exempel på mindre system kan t ex vara kundregistret för ett litet företag med få anställda, jämfört med ett kundregister för Volvo.

1.1 Uppsatsens upplägg

Uppsatsen är indelad i följande stycken:

- **Syfte**
- **Bakgrund** – områden som systemarv, reengineering och databasutveckling presenteras. Inom det sistnämnda ges stor plats åt modelleringsmetoden KRB (Brown, 1997), då denna utgör en viktig del i vår reengineeringsmetod. Även verksamhetsanalys och systemutvärdering presenteras kortfattat.
- **Metod** – här beskrivs hur vi använder oss av fallstudien och hur vi evaluerar vår reengineeringsmetod.
- **Utformning av en reengineeringsmetod för databaser** – här beskriver vi vår reengineeringsmetod.
- **Tillämpning av reengineeringsmetoden** – här beskrivs hur metoden används i fallstudien.
- **Utvärdering av reengineeringsmetoden** - här diskuteras hur metoden fungerade. En ny version av metoden presenteras utifrån denna utvärdering.
- **Diskussion** – här tas några relevanta frågor om reengineering upp.
- **Slutsatser** – våra sammanfattande slutsatser presenteras.

2 Syfte

Denna uppsats syftar till att ta fram ett förslag till en metod för reengineering av relativt små databassystem med egenskaper liknande dem hos legacy systems. Genom att kombinera delar av befintliga tekniker och metoder för reengineering och databasutveckling, som t ex ER-modellen (Laplante, 1996) och KRB-metoden (Brown, 1997), försöker vi sammanställa ett tillvägagångssätt för att omformalisera och uppdatera sådana databassystem. Detta tillämpas sedan i en fallstudie där vi gör om en gammal databas. Utifrån denna studie vill vi utvärdera hur väl metoden fungerade och föreslå eventuella ändringar, och slutligen presentera en mer genomarbetad metod. Genom resultatet av studien hoppas vi också kunna diskutera sådant som är bra att tänka på vid reengineering och om det är bra att utgå ifrån den gamla databasen eller om man vinner mer på att börja från början.

Då vår fallstudie berör ett system av begränsad storlek, kommer vi inte att försöka göra någon generell metod för reengineering. Vi håller oss till att diskutera hur man kan göra om relativt små databassystem.

3 Bakgrund

I bakgrundsdelen redogörs för reengineering och olika metoder för att bedriva ett reengineeringsprojekt. Inom databasutveckling beskrivs Entity-Relationshipmodellen (Laplante, 1996) och KRB-metoden (Brown,1997). Avslutningsvis presenteras kortfattat reengineering ur ett databasperspektiv samt verksamhetsanalys och systemutvärdering.

Många organisationer tvingas idag underhålla större eller mindre mjukvarusystem som skapades för länge sedan. De är ofta programmerade utan någon strukturerad metod i språk som inte längre används, och den övergripande designen kan redan från början ha varit mycket bristfällig. Systemen har dessutom blivit än mer oorganiserade och svårbegripliga genom många års underhåll, ofta framkallat av kriser av olika slag. Det fortsatta underhållet av sådana system blir svårare och dyrare ju äldre systemen blir. (Tilley, 1998) Systemen är ofta mycket viktiga för verksamheten de används i, ibland kritiska, varför man inte bara kan lägga ned dem eller byta ut dem. Det ligger också mycket verksamhetskunskap inbäddad i systemen som man gärna vill ta tillvara. (Müller et al., 1993)

Problemet med föråldrade system uppmärksammas mer och mer på olika håll. Det har inom software engineering under nittiotalet skett en skiftning från den traditionella fokuseringen på konstruktion och nyutveckling till en ökad satsning på underhåll och vidareutveckling (Tilley et al., 1993; Müller et al., 1993). Software evolution framhålls som en bättre modell än den traditionella distinktionen mellan utveckling och underhåll och innebär att man fokuserar på fortlöpande förbättring av existerande system, vilket involverar både utveckling och underhåll (Tilley et al., 1993). Systemutveckling ses inte längre som ”utveckla först och underhåll sedan”, utan snarare som att ett system aldrig är färdigutvecklat. Utvecklingen av det är aldrig tänkt att avslutas. Evolutionary systems kan definieras som ”*…systems that are capable of accomodating changes over an extended operational life*”. (Tilley, 1995)

3.1 Reengineering

En definition på reengineering är ”*…the systematic transformation of an existing system into a new form to realize quality improvements in operation, system capability, functionality, performance or evolvability at a lower cost, schedule, or risk to the customer.*” (Tilley, 1995)

Reengineering innebär att man utifrån ett existerande system skapar ett nytt och bättre system (Tilley, 1995; Bernus et al., 1998). Man arbetar med mer eller mindre formella metoder och verktyg för att försöka förstå hur det gamla systemet fungerar och vad det gör. Man försöker också ta tillvara den inbäddade kunskap om verksamheten som finns i systemet, liksom gamla kravspecifikationer och designbeslut då dessa innehåller viktig information om verksamheten och systemet. Denna information används sedan som grund när man designar och utvecklar det nya systemet. (Tilley, 1995) Reengineering är ett sätt att uppdatera gamla system till att möta organisationsförändringar och nya krav från organisationen (Bernus et al., 1998). Efter reengineering av systemet får man ett system som är lättare att underhålla, kräver mindre underhåll, är mer strukturerat, lättare att förstå och har en bättre dokumentation (Somerville, 1995).

Reengineering av data, d v s av den information som lagras, innebär att analysera och omorganisera datastrukturen i ett system för att göra den mer lättförståelig. Exempelvis så bygger många gamla databaser på hierarki- eller nätverksmodellen och om de idag behöver göras om vill man ofta använda en relationsorienterad eller objektorienterad modell. Målet är ofta att lyckas strukturera data på ett mer lättförståeligt och logiskt sätt. (Somerville, 1995)

Reengineering handlar om att förbättra befintliga system och få en större lönsamhet än vad som skulle vara möjligt genom nyutveckling. Det är aktuellt att använda sig av reengineering när ett system inte längre är vidareutvecklingsbart till rimlig kostnad, men när det finns funktioner i systemet som är värda att bevara (Tilley, 1995). Det är också bra att använda sig av reengineering när det handlar om ett system som organisationen är beroende av och som ofta måste underhållas (Somerville, 1995). Det kan t ex vara så att man behöver byta plattform och programvara för ett system som i övrigt fortfarande fungerar och används. Man kan också behöva lägga till ny funktionalitet som inte är möjlig, eller är mycket svår och kostsam, att implementera i det gamla systemet.(Tilley, 1995)

Det finns gränser för hur mycket man kan förändra och förbättra ett system m h a reengineering. Om man t ex skall göra om ett system som har skapats utifrån funktionellt tänkande till ett som bygger på objektorientering, så blir förändringarna troligtvis så stora och kräver så stora strukturella förändringar att det kanske är bättre att bygga ett helt nytt system utifrån en ny kravspecifikation. (Somerville, 1995) Om reengineering inte tros vara mer kostnadseffektivt, mindre riskfyllt eller gå snabbare, så skall man överväga att i stället använda sig av nyutveckling (Tilley, 1995).

Det är alltid bra om man kan reducera omfattningen och komplexiteten av reengineeringsuppgiften. I de flesta system finns t ex delar som aldrig används och dessa finns det naturligtvis ingen orsak att behålla. Även delar som mycket sällan används eller duplicerade delar kan man kanske avvara. (Gustafsson & Johansson, 1994)

Det finns både fördelar och nackdelar med att utgå från det gamla systemet vid skapandet av ett nytt. Som fördelar kan nämnas att man tar tillvara mycket kunskap på ett relativt enkelt sätt, exempelvis åtminstone delvis vilka funktioner det nya systemet behöver ha och vilken data man har behov av att lagra (Tilley, 1995). Nackdelar som vi ser kan vara att man i och med att man utgår från det gamla systemet riskerar att låsa fast sig vid gamla vanor och därmed vid gamla arbetssätt. Man kan missa nya behov och framför allt nya infallsvinklar.

Det är viktigt att en reengineeringsmetod inbegriper en verksamhetsanalys i vilken man identifierar verksamhetens mål. Att ta reda på vad man egentligen ska göra i verksamheten är mycket viktigt eftersom detta kan påverka hur man utför reengineeringen. Eventuellt kan man välja en metod för verksamhetsanalys ur en metodik för nyutveckling (se exempelvis Andersen, 1994 eller Lewis, 1994) och anpassa denna för reengineering. (Gustafsson & Johansson, 1994)

3.1.1 Reengineeringsstrategier

Det första man bör göra i en reengineeringsprocess är att bestämma avsikten med det nya system som skall utvecklas, alltså att fastställa någon form av målformulering för det nya systemet. Detta är något som systemutvecklaren och beställaren gemensamt bör komma fram till. Det är inte säkert att beställaren inser alla de möjligheter som det nya systemet skulle kunna erbjuda, och det är därför viktigt att systemutvecklaren dels är insatt i det gamla systemet och dels har förmåga att presentera nya möjligheter för beställaren.(Gustafsson & Johansson, 1994) När målet för det nya systemet väl har blivit fastställt finns det några olika vägar att gå i utvecklingen från det gamla systemet till det nya (även kombinationer av dessa är möjliga):

Nytt system

I den här ansatsen så bortser man till stor del från det befintliga systemet och bygger det nya fristående från det gamla. Man återanvänder dock vissa delar som exempelvis designidéer och bakomliggande information från det gamla systemet. När det nya systemet är färdigt och taget i bruk så stängs det gamla systemet ner. (Tilley, 1995) Eftersom denna strategi går ut på att göra om ett gammalt system från grunden så kan den ses som en högriskstrategi, bl a av följande orsaker:

- Det nya systemet måste bli bättre än det gamla i det att det innehåller ny och utökad funktionalitet, det kan inte bara vara underhållsvänligare.
- Det finns ofta beroenden i ett system som inte finns dokumenterade och som alltså kan vara svåra att få med i ett nytt system.
- Det finns sällan bra specifikationer att tillgå. Dokumentationen kan vara bristfällig från början och eftersom systemen ofta har genomgått många förändringar genom åren som troligtvis inte alltid dokumenterats så kan situationen ha blivit än värre.
- Stora projekt tenderar att svälla ytterligare. Det kan vara svårt att i början specificera exakt vad systemet skall göra och därmed är det svårt att göra några säkra tidsramar.
- Förseningar tolereras sällan. Eftersom verksamheten ofta är beroende av systemet är det viktigt att det så snabbt som möjligt finns tillgängligt med alla nya funktioner och blir färdigt inom den givna tidsramen. (Gustafsson & Johansson, 1994)

Tilläggssystem

Man skapar det nya systemet bitvis genom att utveckla nya små delar för sig. Man kan bygga upp ett ramverk över den önskade arkitekturen i det nya systemet och sedan allt eftersom man utvecklar nya delar implementera dem i denna. Man använder och testar de nya delarna parallellt med det gamla systemet, genom ett automatiskt ”filter” som gör det möjligt att använda samma data i båda systemen. När man har nyimplementerat både relevanta delar från det gamla systemet och de nya funktioner som efterfrågas går man över till att använda bara det nya. (Tilley, 1995) Denna strategi kan anses som mindre riskfylld än den ovanstående eftersom man hela tiden parallellt har kvar det gamla systemet i användning. Det nya systemet börjar inte användas på allvar innan man har sett att allt fungerar. Man får en mer djupgående testning än om man utvecklar ett nytt system.

Evolutionssystem

I denna ansats så förändras det gamla systemet gradvis till att slutligen övergå i det nya systemet, genom att ny funktionalitet läggs till efter hand i det gamla systemet. Den gamla koden kan omstruktureras till att innefatta nya funktioner och egenskaper. Önskvärda systemegenskaper introduceras bitvis i det existerande systemet som under processen genomgår förändringar både i arkitektur och funktionalitet. (Tilley, 1995) Liksom den ovanstående strategin kan denna ansats anses som mindre riskfylld än den första. Man arbetar hela tiden utifrån det gamla systemet och gör bara små förändringar åt gången. Om en förändring skulle misslyckas så har man möjlighet att gå tillbaka till den version man hade innan ändringen. (Gustafsson & Johansson, 1994)

3.1.2 Reverse engineering

För att kunna underhålla, omarbeta och dokumentera ett system måste man ha en bra förståelse av dess funktion, design och struktur. I gamla system som har ändrats och uppdaterats flera gånger genom åren och där allt inte dokumenterats fullständigt är det ofta svårt att få denna förståelse. (Tilley, 1998) Det kan då vara nödvändigt att försöka återskapa en beskrivning av systemets underliggande struktur och design och även av den kunskap som ligger dold i systemet i form av bl a verksamhetskännedom. För att göra detta kan man använda sig av *reverse engineering*. (Müller et al., 1993) Man kan exempelvis återskapa ett gammalt systems kravspecifikation som sedan kan användas på olika sätt, t ex som grund för en kravspecifikation till ett nytt system eller för att underlätta underhållet av det befintliga system. (Somerville, 1995) Reverse engineering kan användas som ett delsteg i en reengineeringsprocess.

I reverse engineering är målet att få fram information från befintliga mjukvarusystem (Müller et al., 1993). Det görs inga ändringar av systemet utan det handlar bara om att undersöka och förstå (Tilley, 1995). Beroende på vilken typ av information man vill få ut av det gamla systemet kan man välja att utföra olika steg, t ex:

- **Programanalys** – man analyserar källkoden för att förstå syntaxen och t ex upptäcka beroendeförhållande mellan olika programdelar.
- **Planigenkänning** – man försöker hitta mönster i koden och se hur kod har återanvänts. Man inriktar sig mot semantiken bakom ett program.
- **Konceptilldelning** – man försöker upptäcka koncept i källkoden som kan användas för att försöka beskriva hur programmeraren har tänkt. På så sätt kan man få en förståelse för hur han har arbetat och därmed lättare förstå och förutsäga vad som händer i koden.
- **Omdokumentering** – man skapar dokumentation för programdelar som är bristfälligt eller inte alls dokumenterade. Det sker genom att man skapar pseudokod utifrån källkoden och sedan utifrån pseudokoden skapar text som slutligen blir den nya dokumentationen.
- **Arkitekturåterskapande** – man skapar en bild av den övergripande designen av ett system för att på så sätt förstå helheten och inte bara enstaka delar. (Tilley, 1998)

3.2 Databasutveckling

Det finns många olika teorier och metoder som kan användas inom databasutveckling. En av dessa är modelleringsmetoden *KRB* (Kapur, Ravinda & Brown-metoden). Denna metod kan egentligen inte sägas vara en ”riktig” metod utan den utgör snarare ett ramverk som sammanfattar idéer från flera olika metoder. (Brown, 1997) Vissa delar av den kan t ex sägas grundas i Peter Chens artikel *The Entity-Relationship Model – Toward a Unified View of Data*, där han presenterar *Entity-Relationshipmodellen* (ER-modellen[4]). (Laplante, 1996)

Chen menar i sin artikel att man bör gå igenom fyra steg för att skapa en bra datamodell enligt ER-modellen:

- **Identifiera entiteter och relationer som är av intresse**
- **Identifiera semantisk information i relationerna, t ex kardinalitet**
- **Definiera attribut och värdeförråd[5]**
- **Organisera data i entiteter och relationer och bestäm primärnycklar**

När Chen presenterade sin ER-modell 1975 var den ett första försök att skapa en enhetlig syn på hur data representerades i databasmodeller. (Laplante, 1996) ER-modellen är fortfarande aktuell och i användning. Mycket har förändrats genom åren, men som beskrivs nedan i stycke 3.2.1 så är grunderna de samma än idag.

3.2.1 Entity-Relationship Modellen

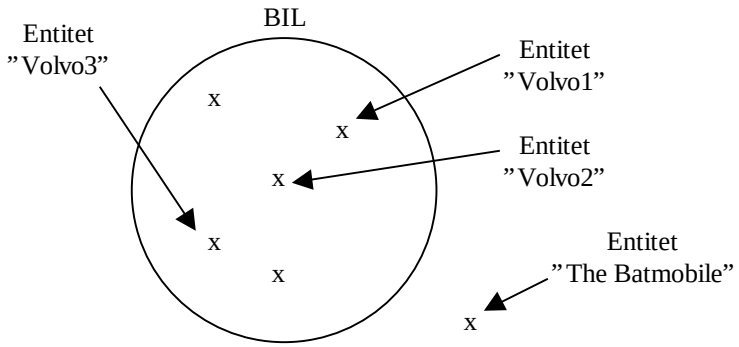
ER-modellen utgör ett sätt att grafiskt modellera verksamheten som man vill implementera i ett system. För att kunna använda sig av ER-modellen vid datamodellering är det väsentligt att förstå de grundläggande begreppen *entitet*, *attribut* och *relation*. (Andersen, 1994) Även begrepp som *kardinalitet* och *partialitet/totalitet* är viktiga.

Entitet

En entitet representerar något i världen utanför systemet, t ex en viss person, ett visst ställe, en viss sak, ett visst tillstånd, en viss tilldragelse eller en viss begreppsmässig konstruktion. (Andersen, 1994; Conolly et al., 1998) Det som krävs av en entitet är att man kan knyta viss information till den. För att kunna ordna entiteter på ett förståeligt sätt väljer man att gruppera dem i *entitetstyper* eller *klasser*. Varje entitetstyp är en samling av entiteter som uppfyller de kriterier som definierar entitetstypen. (Andersen, 1994) Exempelvis så skulle en beskrivning av entitetstypen BIL kunna vara ”alla bilar som existerar i verkligheten”. ”Volvo1” i figur 1 skulle då vara en entitet eller instans i denna entitetstyp. ”The Batmobile” är också en entitet, men enligt definitionen av typen BIL passar den inte in eftersom det är en påhittad bil.

[4] Kallas ibland även för Entity-Attribute-Relationship-modellen (EAR-modellen).

[5] Värdeförrådet utgörs av den domän där attributet kan hämta sina värden. T ex kan attributet *veckodag* hämta värden från domänen innehållande {måndag, tisdag,…,söndag}. (se exempelvis Sundgren, 1992)



Figur 1 – Exempel på entitet

Entitetstyper kan bestå av delmängder. (Andersen, 1994) Exempelvis så skulle BIL kunna vara uppdelad i KOMBIBILAR och SEDANBILAR, som då båda skulle uppfylla kriterierna för BIL, men även vara egna entitetstyper.

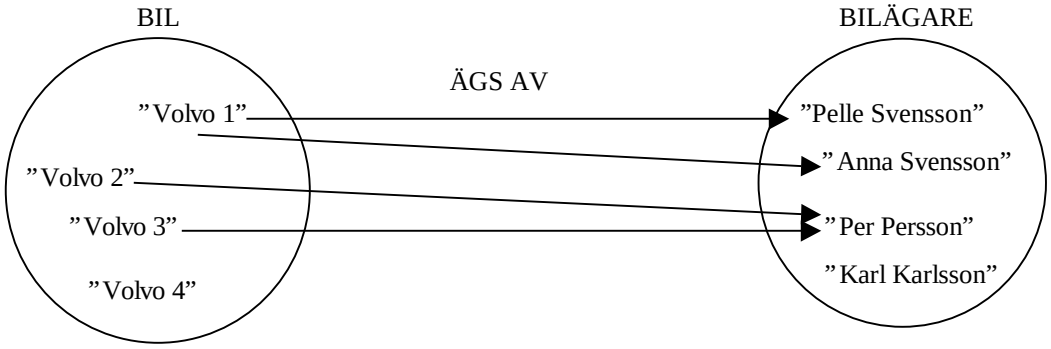
Attribut

Ett attribut är en egenskap hos en entitet och till varje entitetstyp hör ett antal attribut. Ett attribut karakteriserar alla entiteter som hör till entitetstypen. (Andersen, 1994; Conolly et al., 1998) Entitetstypen BIL kan exempelvis ha attribut såsom BILNUMMER och ÅRSMODELL och typen BILÄGARE attribut såsom PERSONNUMMER, ÅLDER och KÖN.

Eftersom det ibland är oklart vad som menas med ett visst attribut så måste alla attribut ha en förklarande beskrivning. En sådan beskrivning för BILNUMMER skulle då bli t ex ”ett uttryck bestående av tre bokstäver följt av tre siffror som identifierar en bil”. Eftersom BILNUMMER är olika för alla bilar och ingen bil kan ha samma bilnummer som någon annan så säger man att BILNUMMER är ett identifierande attribut eller ett nyckelattribut. Det betyder att BILNUMMER kan användas för att specificera exakt vilken bil man talar om. (Andersen, 1994) Man måste för varje klass definiera ett nyckelvärde (detta representeras i uppsatsen med *). För vissa klasser har man sammansatta nycklar, d v s två värden som hämtas från olika klasser och tillsammans utgör klassens nyckel (Brown, 1997) (se nedan KRB-metodens steg 4).

Relation

En relation är ett samband mellan entiteter inom olika klasser. Exempelvis så finns det en relation mellan ”Volvo1” i entitetstypen BIL och ”Pelle Svensson” i entitetstypen BILÄGARE. (se figur 2) Denna relation kan döpas till ÄGS AV, d v s ”Volvo1 ägs av Pelle Svensson”. Alla relationer som går mellan BIL och BILÄGARE är av typen ÄGS AV. (Andersen, 1994; Conolly et al., 1998)

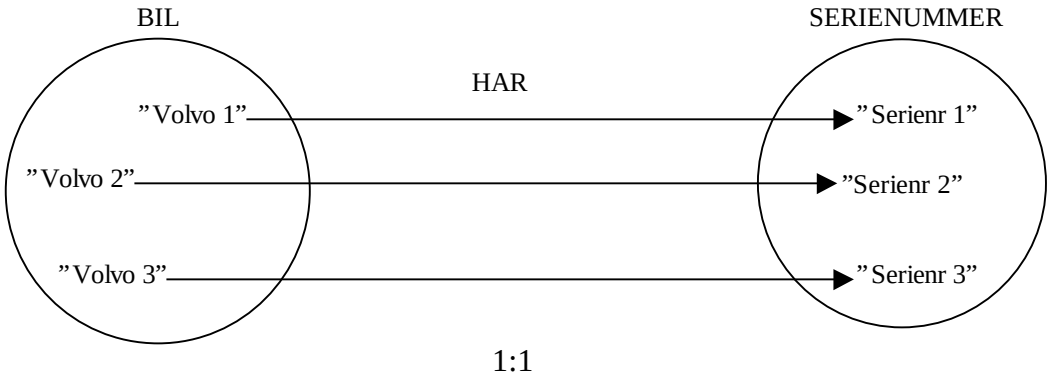


Figur 2 – Exempel på relation

Kardinalitet

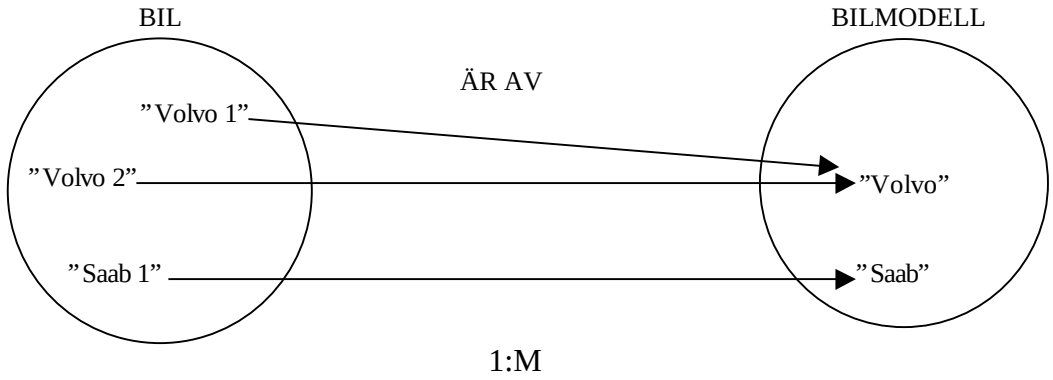
Varje relation har en kardinalitet. Kardinaliteten innebär hur många instanser eller entiteter i de berörda klasserna som kan delta i relationen (Andersen, 1994; Brown, 1997; Conolly et al., 1998). För binära relationer, alltså relationer mellan två klasser, finns förhållandena **1:1** (ett till ett), **1:M** (ett till många) och **M:M** (många till många). (Brown, 1997)

Ett exempel på ett 1:1 förhållande är relationen BIL HAR SERIENUMMER. Varje bil har bara ett serienummer och varje serienummer hör bara till en bil. (se figur 3)



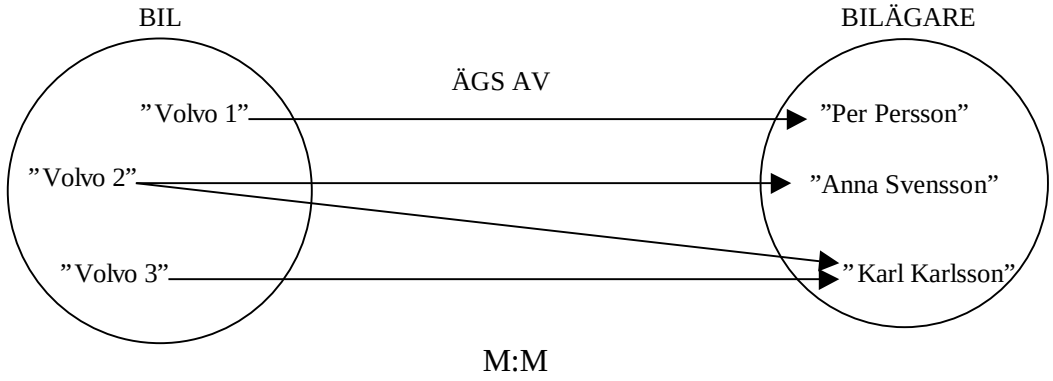
Figur 3 – Exempel på ett 1:1 förhållande

En 1:M relation kan illustreras med BIL ÄR AV BILMODELL – en bil kan bara vara av en modell, men för varje modell finns det många bilar. (se figur 4)



Figur 4 – Exempel på ett 1:M förhållande

BIL ÄGS AV BILÄGARE kan illustrera ett M:M förhållande. En bil kan kanske i ett system ägas tillsammans av flera ägare, och en ägare kan äga flera bilar. (se figur 5)



Figur 5 – Exempel på ett M:M förhållande

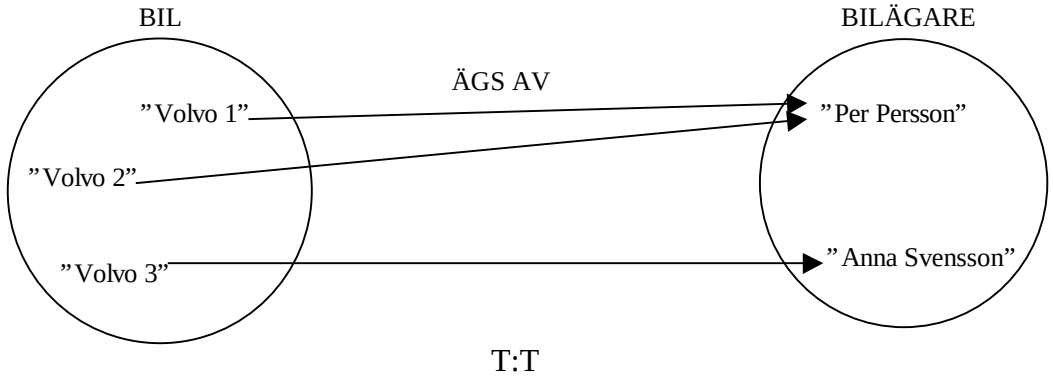
Ett M:M förhållande är svårt och opraktiskt att implementera i en databas och man bör alltså omforma en sådan relation till en 1:M relation (Brown, 1997) (se nedan KRB-metodens steg 4)

Det är viktigt att komma ihåg att kardinaliteten beror på systemet, eller snarare på den verksamhet systemet fungerar i. I M:M exemplet ovan kan flera personer stå som ägare till samma bil, men i ett annat system kanske man bara registrerar en ägare per bil och då blir relationen 1:M. Det beror också på hur man definierar en klass – en produkt kan ju t ex vara en speciell burk med majonnäs, eller konceptet ”burk med majonnäs” – det är frågan om i det första fallet *identifierbara* och i det andra fallet *kvantifierbara* produkter. Vilket man menar eller vill avse måste man göra klart för sig, eftersom detta påverkar kardinaliteten – ser man det som en speciell burk majonnäs så kan det bara finnas en köpare, annars flera. (Brown, 1997)

Partialitet/Totalitet

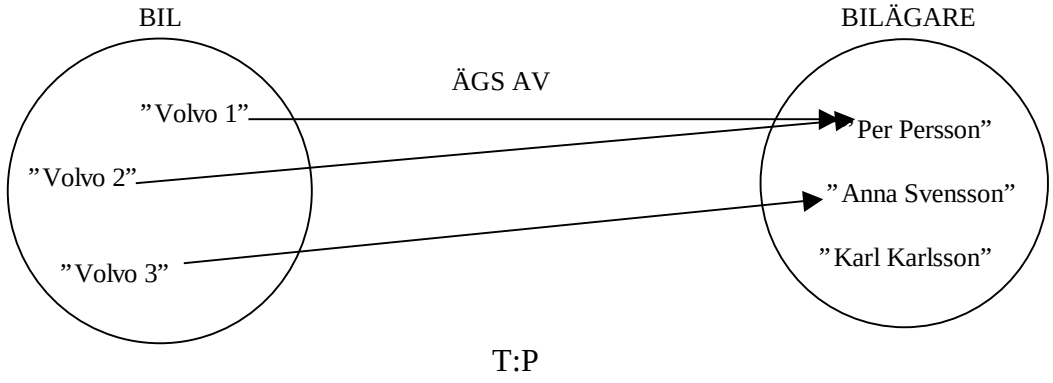
Förutom kardinalitet kan man för relationer bestämma om de är totala eller partiella (Conolly et al., 1998). Man avgör hur många instanser av en klass som ingår i en relation med en annan klass (Sundgren, 1992). En relation är partiell om inte alla entiteter i de berörda klasserna måste vara sammankopplade med någon annan entitet och alltså delta i relationen. Om samtliga entiteter i en klass måste vara sammankopplade med en entitet i den relaterade klassen och det alltså inte finns några ”lösa” entiteter, är relationen total. (Conolly et al., 1998; Sundgren, 1992) Ett annat sätt att uttrycka det är att en relation är total om en entitets existens kräver existens av en entitet i den relaterade klassen, annars inte. (Conolly et al., 1998) En relation kan vara **T:T** (total till total), **T:P** (total till partiell) eller **P:P** (partiell till partiell). Relationen BIL ÄGS AV BILÄGARE kan illustrera samtliga dessa.

Om det i systemet finns minst en ägare till varje registrerad bil och minst en bil för varje registrerad bilägare, blir relationen T:T. (se figur 6)



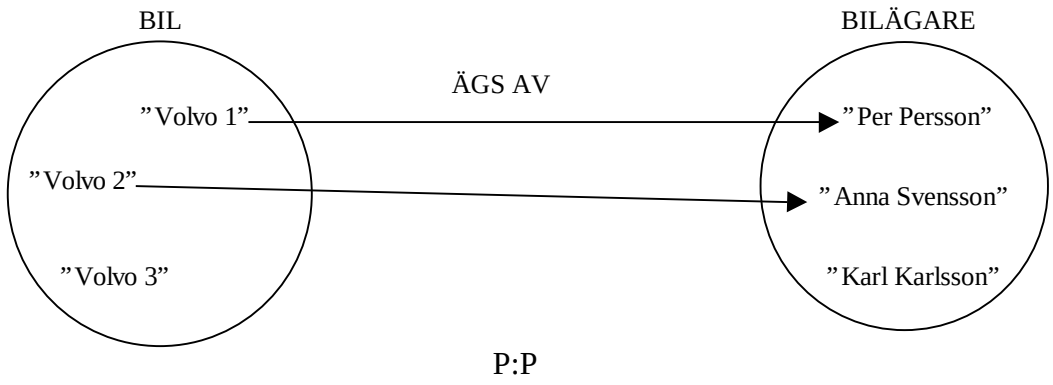
Figur 6 – Exempel på ett T:T förhållande

Om det däremot kan finnas registrerade bilägare som inte äger någon bil blir relationen T:P. (se figur 7)



Figur 7 – Exempel på ett T:P förhållande

Om man dessutom kan tänka sig att ha registrerade bilar utan någon ägare i systemet blir relationen P:P. (se figur 8)



Figur 8 – Exempel på ett P:P förhållande

3.2.2 Beskrivning av KRB-metoden

KRB-metoden är en modelleringsmetod och syftar till att utveckla ett klassdiagram eller en objektmodell samt dokumentation för denna. Detta ligger sedan till grund för implementeringen av systemet man modellerar. KRB består i korthet av följande sju steg (Brown, 1997):

1. **Identifiera kandidatklasser**
2. **Bestäm definitiva klasser**
3. **Etablera relationer mellan klasser**
4. **Utveckla relationer med kardinalitet M:M**
5. **Bestäm attribut för varje klass**
6. **Normalisera**
7. **Bestäm operationer (beteende) för relevanta klasser**

Steg 1 – Identifiera kandidatklasser

I detta steg gäller det att hitta olika klasser av objekt som *kan* vara intressanta för systemet – alltså en lista av substantiv som i nästa steg utvärderas för att avgöra om de verkligen hör hemma i systemet.

Den vanligaste klasstypen är entitetsklasser. Dessa relaterar till verkliga entiteter, eller saker, i användarens värld, t ex BIL och BILÄGARE. Sådana saker representeras av substantiv, och det finns åtminstone fyra olika sätt som ensamma eller kombinerade kan ge en lista av sådana substantiv:

- **Intervjuer med användare/klient** – dessa kan göras i grupp eller enskilt. Man kan utifrån de första intervjuerna sammanställa en lista som cirkuleras bland användarna för att nya förslag ska komma fram.
- **Utgå från kravspecifikationen och annan dokumentation** – man plockar ut alla substantiv man tror kan vara relevanta ur all den dokumentation man har.
- **Brainstorming** – man samlar en grupp användare och låter dem helt fritt kasta fram idéer, som sedan utvärderas.
- **Delphi-metoden** – samma idé som brainstorming, fast det utförs per mail.

Utöver entitetsklasser finns det även gränssnittsklasser som definierar systemets gränssnitt/kommunikationsmedel mot människor och mot andra system, och kontrollklasser som skapas när man upptäcker en operation eller händelse som anropar data eller operationer från många andra klasser.

Steg 2 – Bestäm definitiva klasser

I steg 2 testar man alla kandidatklasser för att komma fram till om de verkligen är relevanta i systemet. Om de visar sig vara det, testar man om de behöver vara egna klasser eller om de kan vara attribut till någon annan klass. Varje klass testas mot tre olika kriterier:

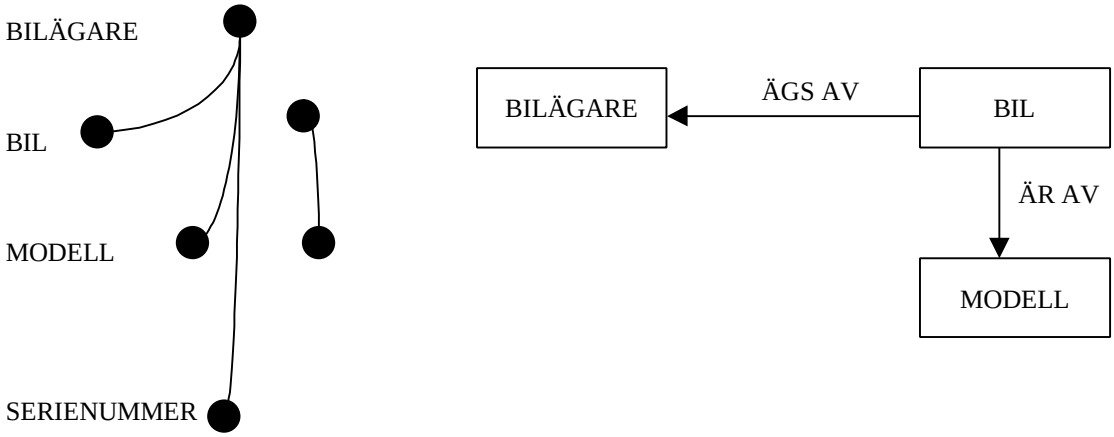
- **Identifierare** – instanserna i en klass måste i den verkliga världen vara distinkta och åtskiljbara, de måste ha en identitet. För varje klass måste ett nyckelvärde, en identifierare, hittas som skiljer de olika medlemmarna eller instanserna i klassen ifrån varandra. Ett exempel är attributet PERSONNUMMER för en tänkt klass PERSON. Det är en sifferkombination som är unik för varje människa och alltså duger som identifierare – man råkar aldrig ut för dubletter som man skulle riskera att göra om man i stället valde t ex NAMN. Om det inte finns någon naturlig identifierare så kan man skapa en genom att lägga till en räknare för varje medlem. För en klass KUND kan t ex ett attribut KUNDNUMMER skapas vilket höjs ett nummer för varje ny kund som läggs till.
- **Definition –** för varje klass skall en beskrivning ges av vad klassen egentligen är. Exempelvis beskrivs klassen BILÄGARE som ”en person som äger en bil”. Det kan vara svårt att ge en tydlig definition som verkligen klargör vad man menar med klassen, men det är viktigt att man lyckas. I senare faser kan man behöva gå tillbaka och ta hjälp av definitionen, och då kan det också hända att man inser att definitionen behöver justeras.
- **Exempelattribut och beteenden** – en klass behöver oftast inte bara en identifierare utan även andra attribut som innehåller viktigt information. Det gäller att hitta attribut hos objektet som är relevanta i systemet. För klassen PERSON kan man t ex tänka sig att man behöver uppgifter om ADRESS. Även beteenden kan vara viktiga. Man kan t ex fråga sig ”vad kan en person göra som är relevant för systemet?”. Svaret kan t ex vara ”byta adress”. Eftersom detta är relevant för systemet vet vi att vi ska inkludera klassen PERSON.

Trots att man använder dessa test kan man aldrig få ett säkert svar på om klassen är relevant eller inte, det är alltid en bedömningsfråga. Det kan dock klargöras om klassen verkligen ska vara en klass eller om den bör vara ett attribut till någon annan klass.

Steg 3 – Etablera relationer mellan klasser

I detta steg tar man reda på hur de olika klasserna interagerar med varandra, d v s vilka relationer som finns mellan dem. Flera faktorer behöver beaktas för att man ska lyckas med detta steg, bl a följande:

- **Upptäck verb** – olika klasser påverkar eller påverkas av andra klasser, och dessa relationer beskrivs med ett verb mellan klasserna. Ett exempel är klasserna BIL och BILÄGARE – en bil ÄGS AV en bilägare. De verb som bidrar till verksamheten som modelleras är de som är intressanta.
- **Fånga alla möjliga relationer** – för att vara säker på att hitta alla relationer bör man arbeta efter följande metod: man listar alla klasser och börjar med att kolla första klassen mot alla andra klasser, sedan tar man nästa klass och kollar den mot alla utom den första o s v. För varje relation man upptäcker ritar man in denna i modellen. (se figur 9)



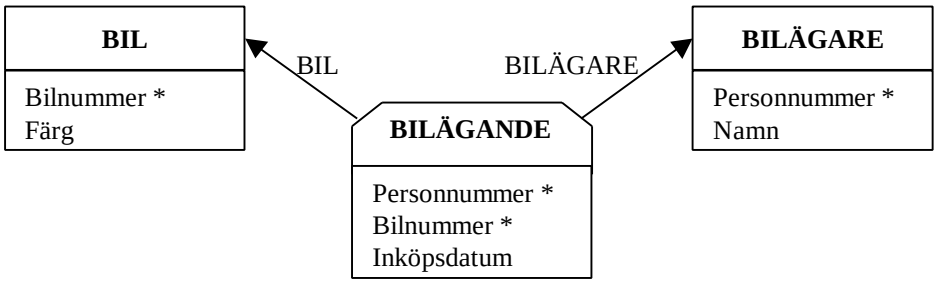
Figur 9 – Fånga alla möjliga relationer

- **Etablera kardinaliteten** – bestäm relationens kardinalitet till 1:1, 1:M eller M:M beroende på hur många instanser eller medlemmar i de berörda klasserna som kan delta i relationen. (se stycke 3.2.1, Kardinalitet)

Steg 4 – utveckla relationer med kardinalitet M:M

M:M förhållanden vållar problem när de skall implementeras i en databas. I exemplet med BIL och BILÄGARE ovan, där varje bil kan ha flera ägare och varje bilägare kan äga flera bilar, så går det inte att använda sig av en ensam nyckel för att representera detta. Man måste då göra plats för flera bilar respektive bilägare för varje post i BILÄGARE och BIL (se figur 5). Problemet blir då att avgöra hur många ägare varje bil kan ha och hur många bilar varje bilägare kan äga – detta måste hårdkodas in i systemets struktur vilket blir mycket opraktiskt och dessutom bryter mot normaliseringsreglerna (se KRB-metodens steg 6).

M:M förhållanden kan utvecklas till att bli 1:M förhållanden genom att vi inför en ny klass, BILÄGANDE, som så att säga ligger ”mitt emellan” klasserna BIL och BILÄGARE och kombinerar båda nycklarna från dessa till en sammansatt nyckel. I denna nya klass kan vi dessutom införa nya attribut som t ex inköpsdatum, som är specifika för just en bil och en bilägare. (se figur 10)



Figur 10 – Utveckla M:M förhållanden

Man måste gå igenom alla klasser i modellen och göra på samma sätt i varje situation där det finns ett M:M förhållande.

Steg 5 – Bestäm attribut för varje klass

I detta steg går man igenom varje klass och listar alla de attribut klasserna ska ha, d v s vilken information man ska lagra för varje klass. Attributen bör också definieras för att förklara vad de betyder och innebär. Nyckelvärdet i varje klass markeras, vanligen med en asterisk. För BIL t ex kan man tänka sig att ha BILNUMMER som nyckel och TILLVERKNINGSDATUM som attribut.

Steg 6 – Normalisera

Normalisering innebär att se till att varje attribut hör till rätt klass. Man kan välja att normalisera till olika nivåer, vanligast är till 3:e normalformen (3NF).

Första normalformen (1NF)

Första normalformen sägs vara uppnådd då det på varje rad i en tabell endast förekommer ett värde i varje kolumn. I figur 11 som kan sägas vara helt onormaliserad, förekommer det flera värden på samma rad i attributen BILÄGARE och PERSONNUMMER – tre personer står som ägare till samma bil. För att uppnå 1NF så gör man om modellen så att varje värde står på en egen rad, d v s det blir tre förekomster med samma värde under BILNUMMER, men endast ett värde på varje rad (se figur 12). Datan innebär fortfarande samma sak, men bilnumret upprepas för varje ägare.

BILÄGANDE

Bilnummer*	**Personnummer***	**Bilägare**	**Antal skadefria år**	**Försäkringsklass**
ABC123	551216-7890	Maria	15	B
	560917-8901	Jonny	2	B
	750212-8965	Lasse	4	D
DEF456	781123-7512	Anna	3	D

Figur 11 – Exempel på onormaliserad tabell

BILÄGANDE

Bilnummer*	Personnummer*	Bilägare	Antal skadefria år	Försäkringsklass
ABC123	551216-7890	Maria	15	B
ABC123	560917-8901	Jonny	2	B
ABC123	750212-8965	Lasse	4	D
DEF456	781123-7512	Anna	3	D

Figur 12 – Exempel på tabell i 1NF

Andra normalformen (2NF)

Andra normalformen är uppnådd när alla attribut som ej är nycklar är beroende av hela den sammansatta nyckeln. I figur 12 så är den sammansatta nyckeln PERSONNUMMER och BILNUMMER. Attributen BILÄGARE, ANTAL SKADEFRIA ÅR och FÖRSÄKRINGSKLASS är dock inte beroende av nyckeln BILNUMMER utan bara av nyckeln PERSONNUMMER. För att gå över till 2NF så måste alltså dessa brytas ut och representeras i en egen tabell med den enkla nyckeln PERSONNUMMER (se figur 14). Den ursprungliga tabellen består nu alltså endast av den sammansatta nyckeln PERSONNUMMER och BILNUMMER. (se figur 13)

BILÄGANDE

Personnummer*	Bilnummer*
551216-7890	ABC123
560917-8901	ABC123
750212-8965	ABC123
781123-7512	DEF456

Figur 13 – Exempel på tabell i 2NF

BILÄGARE

Personnummer*	Bilägare	Antal skadefria år	Försäkringsklass
551216-7890	Maria	15	B
560917-8901	Jonny	2	B
750212-8965	Lasse	4	D
781123-7512	Anna	3	D

Figur 14 – Exempel på tabell i 2NF

Tredje normalformen (3NF)

I figur 14 så är FÖRSÄKRINGSKLASS egentligen beroende av ANTAL SKADEFRIA ÅR och inte av PERSONNUMMER. För att komma till tredje normalformen är det precis sådana här förhållanden, där ett attribut är beroende av ett annat attribut som inte är en nyckel, som skall bort. Genom att flytta ut attributet FÖRSÄKRINGSKLASS till en egen tabell och sedan låta ANTAL SKADEFRIA ÅR vara nyckel i den tabellen så uppnår vi 3NF.

BILÄGANDE

Personnummer*	Bilnummer*
551216-7890	ABC123
560917-8901	ABC123
750212-8965	ABC123
781123-7512	DEF456

Figur 15 – Exempel på tabell i 3NF

BILÄGARE

Personnummer*	**Bilägare**	**Antal skadefria år**
551216-7890	Maria	15
560917-8901	Jonny	2
750212-8965	Lasse	4
781123-7512	Anna	3

Figur 16 – Exempel på tabell i 3NF

FÖRSÄKRINGSKLASS

Antal skadefria år*	**Försäkringsklass**
15	B
2	D
4	D
3	D

Figur 17 – Exempel på tabell i 3NF

Innan normaliseringen hade vi klassen BILÄGANDE med attributen PERSONNUMMER, BILÄGARE, BILNUMMER, ANTAL SKADEFRIA ÅR och FÖRSÄKRINGSKLASS. Nu i 3NF så har vi klasserna BILÄGANDE, BILÄGARE och FÖRSÄKRINGSKLASS. (se figurer 15, 16 och 17)

Steg 7 – Bestäm operationer (beteende) för relevanta klasser

Steg 7 handlar om att upptäcka och specificera operationer och beteende och syftar till att kontrollera att allt är korrekt i objektmodellen. För att göra detta finns olika metoder. KRB-metoden tar upp State Transition Diagrams (STD) och Responsibility Driven Design (RDD).

Den förstnämnda metoden innebär att man uppmärksammar olika händelser som är relevanta för en objektklass och som förändrar objektets tillstånd (värdena på attributen). En slags karta upprättas över alla tillåtna tillstånd och förändringar[6] för objektklassen, tillsammans med de händelser som orsakar förändringarna och de aktioner som de resulterar i. Meningen med ett State Transition Diagram är att identifiera de nödvändiga beteenden objekten måste kunna handskas med. Inte alla klasser behöver ett STD – det är relevant att använda för klasser med en komplex livscykel, t ex klasser som skapar eller tar bort relationer eller instanser.

Den andra metoden, RDD, handlar om att undersöka alla klasser och deras attribut för att upptäcka vad vi behöver av klassen, alltså vilka operationer klassen måste klara av. Det är en form av sista kontroll av att alla klasser har alla de attribut som de behöver och att alla attribut verkligen är attribut och inte klasser. Detta undersöks genom att man studerar hur data behandlas av de olika klasserna och hur datan förändras. För varje klass skapas ett klasskort som beskriver dess attribut och hur data förändras. Fördelen med att använda sig av RDD är att man redan på ett tidigt stadium upptäcker eventuella felaktigheter i objektmodellen. Dessa fel är sådant som annars troligen skulle skapat problem vid implementeringen och då tagit mycket längre tid att åtgärda.

[6] Med tillåtna tillstånd menas att det för varje attribut finns en domän varifrån det kan hämta sina värden. Det som ligger utanför denna domän är inte intressant för systemet och inkorporeras inte i det.

De sju stegen i KRB-metoden ska resultera i en korrekt, normaliserad objektmodell. (Brown, 1997) Denna ligger till grund för det system som skall implementeras.

3.3 Orsaker till reengineering av databaser

Det finns flera anledningar till att man kan behöva göra om en gammal databas och reengineering kan i många fall vara ett bra alternativ att använda sig av. Det kan vara så att den nya databasen förväntas lagra mer data än vad som var tänkt när den ursprungligen byggdes och att detta inte är möjligt i det nuvarande systemet. En annan anledning kan vara att det har uppstått problem med hur datan lagras. Exempelvis så kanske man lagrar för- och efternamn tillsammans i ett fält och om man i stället vill lagra för- och efternamn åtskilda, så får man göra om strukturen i systemet. Det kan också vara så att namn och strukturer är kryptiska och svåra att förstå, exempelvis namn i form av olika förkortningar. Om det dessutom saknas dokumentation som förklarar dessa förkortningar, så kan en anledning till att göra om databasen och/eller omdokumentera den vara att göra dessa mer lättförståeliga. Ytterligare en anledning kan vara om det finns absoluta värden som är hårdkodade in i systemet. Eftersom dessa är svåra att uppdatera kan man välja att göra om databasen, så att dessa är åtkomliga och kan ändras utan att man behöver gå in i systemets struktur. (Somerville, 1995) Oftast kanske det inte räcker med ett av dessa problem för att man skall välja att göra om en databas, men om flera av dem förekommer i samma system, vilket ofta är fallet i legacy systems, så är det relevant att fundera över att eventuellt göra om systemet med reengineering.

3.4 Verksamhetsanalys

Enligt Soft Systems Methodology (SSM) så är det väsentligt att utveckla ett system så att det passar in i en organisation som helhet och inte bara utforma det för att passa till en speciell funktion inom organisationen. Därför är det viktigt att som ett första stadie i en systemutvecklingsprocess göra en verksamhetsanalys för att förstå organisationen som systemet ska ingå i. (Lewis, 1994)

En verksamhetsanalys syftar till att klargöra hur organisationen där ett system ska implementeras ser ut och vad som händer i den. Man analyserar informationsflödet i verksamheten och tar reda på vilka informationskanaler som är relevanta för systemet. Man identifierar också ineffektiva delar av informationshanteringen – meningen med datoriseringen är att öka effektiviteten i informationsflödet. (Allwood, 1991) Eftersom informationssystemet skall hjälpa verksamheten till bättre resultat är det en viktig del i systemutvecklingsprocessen att undersöka och diskutera med användarna på vilket sätt detta kan uppnås. (Andersen, 1994)

I verksamhetsanalysen bör man titta på utomliggande faktorer, exempelvis kunder och leverantörer, som påverkar den verksamhet man studerar. Man bör även försöka fastställa de mål verksamheten strävar mot. Det är också väsentligt att studera människorna i organisationen för att studera hur det planerade systemet kommer att påverka dem. De personer vars arbetssituation förändras bör sedan involveras i den fortsatta utvecklingen, för att öka chanserna att systemet skall accepteras i verksamheten. (Lewis, 1994)

Viktiga frågor att ställa i en verksamhetsanalys är t ex:

- **Vad ska systemet klara av?**
- **Vilka processer ska systemet stödja?**
- **Vilka kommer att använda systemet?**
- **Med vilka andra formella och informella system skall systemet samspela?**
- **Vilka informationskällor till systemet finns?**
- **Vilka ”frågor” ska systemet kunna svara på?**

Gemensamt för dessa och eventuella andra frågor man kan ställa i verksamhetsanalysen, är att de är inriktade på *vad* och *varför*. Frågor av den mer tekniska typen *hur* sparas till senare i systemutvecklingen. (Sundgren, 1992)

3.5 Utvärdering av system

Utvärdering kan beskrivas som *”a process through which information about the usability of a system is gathered in order to improve the system or to assess a completed interface.”* Det är en central del i systemutvecklingsprocessen och därmed också en central del i reengineering. Det är viktigt att utvärdera system och det är bättre att göra en kort, snabb utvärdering än ingen alls. Man bör utvärdera på flera olika stadier, både under skapandet av kravspecifikationen och den konceptuella designen och under implementationen. (Preece, 1994) Även när systemet har tagits i bruk bör det utvärderas för att se att alla krav för systemet är uppfyllda.

Utvärdering kan vara antingen *formande* eller *summerande*. Formande utvärdering av ett system görs under hela utvecklingsprocessen, från skapandet av en kravspecifikation till och med implementation och testning av systemet. Sådan utvärdering behöver inte alltid följa i förväg fastställda mallar utan är ofta ganska informell. Man kan exempelvis fråga användaren om vissa detaljer under implementationsfasen och få svar direkt på enklare designfrågor. Man kan också utföra test där man t ex jämför två olika designidéer för att se vilken som bäst uppfyller användarnas krav. Den här typen av utvärdering hjälper till att skapa ett bra och användbart system. Summerande utvärdering sker efter det att systemet är implementerat och har tagits i bruk. Det handlar om att testa det färdiga systemet och avgöra om det uppfyller kravspecifikationen och om användarna/beställaren är nöjd. (Preece, 1994) Man kan t ex ha ställt upp användbarhetsmål som systemet skall uppfylla, exempelvis att en användare efter en viss tids inlärning av systemet skall klara att utföra vissa uppgifter (Allwood, 1991). Man kan också utifrån kravspecifikationen avgöra om all den nya funktionalitet som efterfrågas i denna verkligen har implementerats och fungerar i systemet, och om andra aspekter som exempelvis säkerhetskrav är tillgodosedda.

För ett reengineeringsprojekt finns olika alternativ för att mäta framgången, exempelvis utifrån ny funktionalitet – hur många nya funktioner har tillkommit? (Tilley, 1995) Många av de metoder som används för utvärdering av nyutvecklade system är även applicerbara på ett system som utvecklats genom reengineering, eftersom det är utvecklingsmetoderna som skiljer sig åt och inte själva systemen.

4 Metod

Utifrån den litteraturstudie som presenterats i Bakgrund sammanställer vi ett förslag till en metod för reengineering av relativt små databaser, hela vägen från verksamhetsanalys till evaluering. För att testa vår metod använder vi oss av den i en designorienterad fallstudie. Efter studien utvärderar vi hur bra metoden fungerade och föreslår eventuella förändringar av den.

4.1 Fallstudien

I fallstudien testar vi metoden genom att använda oss av den under ett reengineeringsprojekt där en databas görs om. Vår fallstudie kan därför sägas vara designorienterat undersökande[7] – genom användningen av metoden i fallstudien omformar vi ett system samtidigt som vi undersöker hur väl metoden fungerar och kommer fram till eventuella förändringar som kan krävas i metodens utformning.

Under fallstudien kommer vi att uppdatera innehåll och funktioner i databasen enligt beställarens önskemål och modellera om den i enlighet med Entity-Relationshipmodellen (ER-modellen) (Laplante, 1996). Modelleringsdelen i vår anpassade metod bygger till stor del på KRB-metoden (Brown, 1997). Eftersom denna metod ursprungligen är avsedd för modellering i nyutveckling av system så kommer vi att göra vissa anpassningar som går mer i linje med reengineering.

4.2 Evaluering av metoden

Man kan utvärdera en metod för systemutveckling på olika sätt. Ett alternativ är att jämföra den med andra metoder när det gäller aspekter som exempelvis uppbyggnad och mål. Ett annat alternativ kan vara att se hur metoden har fungerat tidigare i olika systemutvecklingsprojekt (Ljungberg & Holm, 1998). I vårt fall, eftersom vi själva har satt samman metoden och den därför inte tidigare använts, har vi valt att helt enkelt använda metoden i ett projekt där vi gör om en gammal databas. Sedan utvärderar vi metoden utifrån hur väl den gick att följa i vår fallstudie.

Det första vi vill svara på när metoden har tillämpats i fallstudien är om metoden fungerade. För att ytterligare utvärdera hur väl metoden gick att följa och ta reda på eventuella förändringar som krävs så evaluerar vi metoden utifrån följande kriterier:

- **Hur väl gick metoden att följa i praktiken** – var de olika stegen relevanta? Kom de i rätt ordning? Vad behövde förändras?
- **Vad saknades i metoden** – behövdes det några ytterligare steg?
- **Vad var överflödigt i metoden** – kunde något steg tas bort eller kombineras med något annat steg?

Att vi har valt just dessa kriterier beror på att de känns relevanta för att kunna mäta hur väl metoden fungerar och för att beskriva vad som eventuellt behöver förändras. Efter att ha utvärderat vår metod utifrån dessa kriterier hoppas vi kunna sammanställa en mer komplett metod, där vi tagit hänsyn till problem som framkommit.

[7] En fallstudie kan vara beskrivande, förklarande eller undersökande (Backman, 1998).

5 Utformning av en reengineeringsmetod för databaser

I detta stycke utformar vi ett förslag till metod för reengineering av databaser. Vi går ingående, under stycke 5.1 – 5.7, igenom de moment vi anser bör ingå i en sådan metod och avslutar med en kortfattad sammanfattning i form av en punktlista i stycke 5.8.

Under hela utvecklingsprocessen bör kontinuerlig formande utvärdering genom t ex användarkontakter eller testning av vissa delar utföras. Denna utvärdering bör göras för att i möjligaste mån försöka tillgodose användarnas krav och skapa ett så bra system som möjligt. Det medför också att användarna känner sig delaktiga i utvecklingen och kanske lättare accepterar systemet. Denna typ av utvärdering anser vi vara mycket viktig och den ingår därför som en del i varje steg i metoden.

5.1 Verksamhetsanalys

Vi anser att det första man bör göra i ett reengineeringsprojekt är att utföra en verksamhetsanalys, för att lära känna organisationen och verksamheten där det omarbetade systemet ska fungera. Detta steg är också viktigt i ett nyutvecklingsprojekt, men i ett reengineeringsprojekt måste man utöver verksamhetskunskapen också skaffa sig en förståelse för hur det existerande systemet fungerar och används i organisationen. En verksamhetsanalys i ett reengineeringsprojekt måste alltså förutom de vanliga delarna även inbegripa en noggrannare undersökning av den roll systemet har i verksamheten och på vilket sätt olika användare använder det. Det är också viktigt att konstatera om systemet fungerar bra i verksamheten eller om användarna upplever att det finns problem med systemet som kan behöva beaktas i omarbetningen. Hur verksamheten fungerar både oberoende och beroende av systemet är viktigt att ta reda på eftersom detta kan ha betydelse för hur man går vidare i reengineeringsprocessen, t ex vilken reengineeringsstrategi man väljer.

Vi tar inte närmare upp hur man kan utföra verksamhetsanalysen, eftersom detta beror på förutsättningarna i det aktuella projektet och verksamheten som ska analyseras. Ett företag som t ex är ISO-certifierat har förmodligen redan lagt ner mycket tid på att kartlägga sina processer och sin verksamhet och mycket av analysen borde då kunna hämtas härifrån. I ett projekt där systemutvecklaren redan har kunskap om den aktuella verksamheten är förutsättningarna för verksamhetsanalysen annorlunda än i ett projekt med en helt okänd verksamhet. För att utföra analysen kan man t ex välja en metod ur en vanlig systemutvecklingsmetodik (se t ex Andersen, 1994 eller Lewis, 1994) och anpassa denna för just det reengineeringsprojekt den ska användas i.

Resultatet av verksamhetsanalysen blir någon form av beskrivning av verksamheten och verksamhetsprocesserna, inbegripande även det gamla systemets roll och funktion.

5.2 Reverse engineering

När man har skapat sig en bild av verksamheten och systemets roll i denna så blir nästa fråga att få en uppfattning om hur systemet i sig fungerar och är uppbyggt. Man bör alltså utföra någon form av reverse engineering för att försöka återskapa den ursprungliga objektmodellen och saknad dokumentation. Vilken typ av reverse engineering man väljer beror på systemets storlek, den befintliga dokumentationens omfattning och kvalitet och vilken typ av information man anser sig behöva för att kunna gå vidare i reengineeringen. Exempelvis kan man för att få en bild av den övergripande strukturen och systemet som helhet göra arkitekturåterskapning. I ett system som innehåller mycket kod behöver man förmodligen göra en källkodsanalys. Även koncepttilldelning kan vara av intresse eftersom det kan klargöra varför vissa programmerings- och strukturlösningar valts. (Tilley, 1998) Utifrån det gamla systemet kan man även få en grov uppfattning om vilka data som behöver lagras i det nya systemet och vilken funktionalitet som efterfrågas.

Resultatet av detta steg är en beskrivning av det gamla systemet, i den form och av den omfattning den fortsatta reengineeringen kräver. Vi rekommenderar starkt att man återskapar den gamla objektmodellen, eftersom detta är en grundläggande del i ett databassystem som man har stor nytta av i den fortsatta reengineeringen. Även övrig dokumentation kan vara mycket viktig och relevant att återskapa, men ger inte samma övergripande förståelse av det gamla systemets uppbyggnad och innehåll.

5.3 Målformulering

När man har skaffat sig en uppfattning både om verksamheten och om det gamla systemet är det dags att tillsammans med beställaren komma fram till målet med det nya systemet. Man måste bestämma vilken funktionalitet i det gamla systemet som man behöver ha kvar i någon form, vilken funktionalitet som inte längre är relevant och som man alltså inte behöver ta med, och vilken ny funktionalitet som efterfrågas i det nya systemet. Även andra krav som systemet ska uppfylla måste beaktas, t ex säkerhetskrav. Detta resulterar i någon form av kravspecifikation för det nya systemet, alltså en sammanställning av den funktionalitet systemet ska innehålla och de övriga krav som måste uppfyllas.

5.4 Modellering

När man har fastställt en kravspecifikation och alltså vet vad man vill ha ut ur systemet är det dags att börja modellera det. I denna fas har vi utgått ifrån KRB-metoden (Brown, 1997) och gjort vissa anpassningar i denna så att den bättre skall passa för ett reengineeringsprojekt.

Det som bl a skiljer ett reengineeringsprojekt från ett nyutvecklingsprojekt är att man har mer information att utgå ifrån när man har kommit till modelleringsfasen, eftersom man då förmodligen har lyckats återskapa information om det gamla systemets innehåll och uppbyggnad i någon mån. Man har också till viss del tydligare riktlinjer att rätta sig efter när man utvecklar ett system genom reengineering än om man nyutvecklar ett system: det nya systemet måste på något sätt anknyta till det gamla i innehåll, funktionalitet eller utseende. Detta gör att man i så stor utsträckning som möjligt kan utgå ifrån hur det såg ut och fungerade i det gamla systemet, naturligtvis så länge detta var något som fungerade bra och är önskvärt att beakta.

De sju stegen i KRB-metoden ser på ytan i stort sett likadana ut även efter våra anpassningar. Hur man ska uppnå målet för varje steg skiljer sig däremot en del från ursprungsmodellen och därför beskriver vi ganska ingående för varje steg hur vi har tänkt oss att det kan utföras. Hela processen bör göras med så stor användarmedverkan som möjligt. Graden av användarmedverkan och i vilken form den utförs påverkas dock av många faktorer, exempelvis tidsbrist, och därför är detta någonting som får avvägas från projekt till projekt.

De sju stegen enligt vår metod:

5.4.1 Steg 1 – Identifiera kandidatklasser

Eftersom man genom reverse engineering har återskapat en objektmodell för det befintliga systemet så kan man använda denna som grund för att identifiera klasser i det nya systemet. Man börjar med att plocka ut de substantiv eller klasser som finns i den gamla objektmodellen. Sedan jämför man listan som blir resultatet av detta med kravspecifikationen för det nya systemet. Om det finns några klasser som inte längre är relevanta plockas dessa bort ur modellen. Om det verkar som att det skulle behövas fler klasser än de som fanns i det gamla systemet, alltså om det skall tillkomma ny funktionalitet i det nya systemet, så måste de klasser som behövs för detta identifieras. Detta kan ske på samma sätt som används för att identifiera kandidatklasser i KRB-metoden, t ex genom användarintervjuer eller utifrån tillgänglig dokumentation. Man bör också kontrollera om inte något som i den gamla objektmodellen var attribut till någon annan klass nu bör flyttas ut och bli en egen klass, eller om någon gammal klass kanske kan bli ett attribut.

5.4.2 Steg 2 – Definiera klasser

Tillgången till ett gammalt system kan vara till stor hjälp i detta steg, eftersom man kan återanvända de definitioner, attribut och identifierare som fanns för relevanta klasser i det gamla systemet om de verkar stämma överens med den nya kravspecifikationen. För sådana klasser behöver man egentligen inte definiera exempelattribut eftersom man kan använda de attribut man har för klassen från det gamla systemet. Vi tror att det är bra om man i möjligaste mån låter definitioner och identifierare för klasser som även fanns i det gamla systemet vara så oförändrade som möjligt, så länge detta inte påverkar funktionaliteten i det nya systemet. Detta underlättar för användaren som har lättare för att känna igen sig i det nya systemet och därmed kanske lättare att acceptera det. För nya klasser bör man på samma sätt som i den ursprungliga KRB-metoden definiera exempelattribut, definitioner och identifierare. Det är som redan nämnts i steg 1 viktigt att vara uppmärksam på att sådant som var klasser och attribut i det gamla systemet kanske inte ska fortsätta att vara det – attribut kan i det nya systemet bli klasser och tvärtom.

5.4.3 Steg 3 – Etablera relationer

Om det fanns relationer i det gamla systemet passar kanske några av dessa in i det nya systemet också, men det finns säkert även många nya relationer som behövs. Vi rekommenderar därför att man för att inte missa någon relation använder sig av metoden för att fånga alla möjliga relationer som beskrivs i KRB-metodens steg 3. Man bör också gå igenom kardinaliteten för alla relationer i den nya objektmodellen, även de som kommer från den gamla databasen, eftersom förhållandena kan ha ändrats.

5.4.4 Steg 4 – Utveckla M:M förhållanden

När man har lagt till nya klasser och det tillkommit nya relationer är det även troligt att det har tillkommit fler M:M förhållanden. Man bör alltså gå igenom samtliga klasser och utveckla sådana, genom att bryta ut nya klasser som gör om M:M förhållandet till 1:M (se KRB-metodens steg 4).

5.4.5 Steg 5 – Definiera attribut

Även i detta steg kan man ta hjälp av den gamla objektmodellen, genom att man för varje klass som kommer från denna går igenom dess attribut och avgör om de hör hemma i det nya systemet. Man måste naturligtvis även utgå ifrån den nya kravspecifikationen, dels för att lista attribut för de nya klasser som har tillkommit och dels för att hitta eventuella ytterligare attribut för de klasser som kommer från det gamla systemet.

5.4.6 Steg 6 – Normalisering

Eftersom man har återskapat den gamla objektmodellen vet man i vilken normalform det gamla systemet befann sig i. Detta kan vara en hjälp om man har tagit in hela klasser utan att förändra dem – om det gamla systemet befann sig i 3NF så vet man att också den inflyttade klassen befinner sig i 3NF. Eftersom det förmodligen är ganska sällan man flyttar in en hel klass utan att förändra den alls så blir nog inte den reella nyttan av detta så stor och vi rekommenderar därför att man går igenom hela objektmodellen och normaliserar den till önskad nivå.

5.4.7 Steg 7 – Operationer

Eftersom detta steg är en form av sista kontroll av att allt är korrekt i objektmodellen så menar vi att det behöver göras oavsett om det gäller nyutveckling eller ett reengineeringsprojekt. Även om den gamla objektmodellen var korrekt, vilket naturligtvis inte alls är säkert, så har förmodligen så mycket förändrats att detta inte längre gäller för den nya objektmodellen. Om man väljer att utföra detta steg genom att skapa State Transition Diagrams (STD) eller genom Responsibility Driven Design (RDD) (Brown, 1997), eller om man väljer någon mer informell metod, är upp till systemutvecklaren.

5.5 Implementation

När en objektmodell har skapats är det dags för implementationen. Innan man kan implementera systemet måste man välja vilken reengineeringsstrategi man vill arbeta efter, d v s om man vill göra ett nytt system, ett tilläggssystem eller ett evolutionssystem (Tilley, 1995). Först när man har valt strategi kan man gå vidare med implementationen. Vilken strategi man väljer beror på förutsättningarna i varje projekt – om man t ex snabbt behöver ny funktionalitet i ett verksamhetskritiskt system så väljer man kanske evolutionsstrategin, eftersom man i denna kan implementera de viktigaste funktionerna först i det gamla systemet som fortfarande är i användning.

I detta moment vill vi igen påpeka att en formande utvärdering är viktig, kanske extra viktig just under implementationen eftersom det är nu det reella systemet skapas. Genom kontinuerliga kontakter med användarna i form av t ex intervjuer eller tester ökar chanserna att systemet blir användarvänligt, funktionellt och accepterat.

5.6 Dataöverföring

När det nya systemet är tillräckligt färdigt för att börja användas måste data från det gamla systemet överföras till det nya. Hur detta sker beror på vilken reengineeringsstrategi man valde i implementationen. Vid nytt system så överförs all data när systemet är färdigt att tas i bruk. I ett tilläggssystem så används via ett filter samma data i både det nya och det gamla systemet under implementationen. När det nya systemet anses färdigt stängs det gamla systemet ner och data förs över. I ett evolutionssystem används samma data hela tiden, eftersom nya delar implementeras direkt i det existerande systemet.

Dataöverföringen kan vara ett mycket problematiskt steg då den gamla informationen kan kräva stora anpassningar för att gå att använda i det nya systemet. Hur man väljer att göra överföringen beror på både det gamla och det nya systemet – man kan kanske använda import- och exportrutiner i databashanteraren om det finns sådana, eller så kan man programmera egna överföringsfunktioner. Man kan även tänka sig att programmera funktioner för eventuella anpassningar av data, t ex ett program som tar isär för- och efternamn om dessa är lagrade tillsammans i ett fält i det gamla systemet och man nu vill skilja på dem. När dataöverföringen är gjord måste man på något sätt kontrollera att den blivit korrekt utförd, antingen genom någon form av kontrollfunktion eller genom manuell kontroll om man anser att detta är tillräckligt.

I detta steg är det troligen så att användarna inte har så mycket att tillföra och därför är användarmedverkan kanske inte nödvändig i detta steg.

5.7 Utvärdering

När systemet har tagits i bruk och använts en tid bör en summerande utvärdering göras för att kontrollera att systemet uppfyller kravspecifikationen. Man kan även utvärdera systemet för att säkerställa att det möter användarnas behov. En jämförelse med det gamla systemet vad gäller t ex ökad funktionalitet kan också göras. Det kan också vara relevant att se om något har blivit sämre i jämförelse med det gamla systemet.

5.8 Vår reengineeringsmetod i korthet

Vår metod består av sju steg. Det är som sagts tidigare viktigt att användarna deltar så mycket som möjligt i samtliga steg och att formande utvärdering utförs under hela processen.

1. **Verksamhetsanalys**
2. **Reverse engineering**
3. **Målformulering**
4. **Modellering av det nya systemet**
    1. **Identifiera kandidatklasser**
    2. **Definiera klasser**
    3. **Etablera relationer**
    4. **Utveckla M:M förhållanden**
    5. **Definiera attribut**
    6. **Normalisering**
    7. **Operationer**
5. **Implementation**
6. **Dataöverföring**
7. **Utvärdering**

6 Tillämpning av reengineeringsmetoden

I detta stycke beskriver vi hur vi har tillämpat ovanstående reengineeringsmetod i en fallstudie, där ett gammalt databassystem görs om. För att sätta beskrivningen i ett sammanhang börjar vi med att redogöra för verksamheten där systemet ingår. Detta görs genom att vi redovisar delar av de tre moment som i enlighet med metoden utfördes först, d v s verksamhetsanalys, arkitekturåterskapning och omdokumentering och målformulering.

6.1 Fallstudien

Vårt uppdrag utförs åt Göteborgs Läkarförening (GLF). GLF ingår som en lokalförening i Sveriges Läkarförbund, den nationella fackföreningen för alla läkare i Sverige. Idag arbetar på GLF:s kansli tre heltidsanställda personer, varav två fungerar som fackliga ombudsmän och den tredje sköter kansliets ekonomi och administration vilket även inkluderar medlemsadministration och utskick. Man har också en styrelse som förutom kansliets ombudsmän består av 16 fackligt aktiva läkare som arbetar i nära samarbete med kansliets personal. GLF är idag den näst största lokalföreningen i Läkarförbundet med drygt 2800 medlemmar.

Vårt uppdrag innebar att för GLF:s räkning uppdatera en gammal Microsoft Access-databas. Databasen utgör ett medlemsregister över alla läkare anslutna till GLF och innehåller alltså uppgifter om samtliga drygt 2800 medlemmar. Det är ett medlemsregister med ganska omfattande information, inte bara namn- och adressuppgifter utan även uppgifter om t ex specialistkompetens, yrkesföreningstillhörighet (exempelvis distriktsläkar- eller underläkarföreningen), tidpunkt för legitimation m m.

Nämnas bör att en av författarna sedan tidigare har erfarenhet av den databas som ommodelleras efter att ha arbetat i verksamheten med bl a underhåll av systemet. De konsekvenser som detta har för fallstudien tas upp längre fram.

6.1.1 Beskrivning av nuläge

Verksamheten

GLF:s huvudsakliga uppgift är att företräda sina medlemmar i alla frågor som rör relationer mellan arbetsgivare och arbetstagare. Detta innebär exempelvis att erbjuda sina medlemmar hjälp i fackliga frågor och att förhandla med arbetsgivarna i löneförhandlingar.

Löneförhandlingar

Sveriges Läkarförbund tecknar med arbetsgivarorganisationen centrala avtal som löper i perioder om ett varierande antal år. Under en sådan period har man en eller flera lokala löneöversynsförhandlingar där GLF med arbetsgivaren löneförhandlar individuellt för varje ansluten anställd läkare i Göteborg. Det finns en mängd uppgifter som är väsentliga i dessa individuella förhandlingar, som t ex om läkaren är specialist och hur länge han/hon har varit det eller om han/hon har andra meriter som t ex en disputation bakom sig. Inför förhandlingarna får föreningen på diskett lönelistor från ADB-kontoret i Göteborgs Stad som förs in i Microsoft Excel. Dessa innehåller emellertid inte den typen av information som nämns ovan, utan bara information om aktuella anställningar och uppgifter om arbetsplats, anställningstyp[8], lön, eventuella tillägg för särskilda uppdrag m m. Annan information får man på föreningen själv lagra i sitt medlemsregister och dra ut inför förhandlingar.

Efter avslutade löneförhandlingar har man ett behov av att föra statistik över löneuppgifterna för att kunna bistå medlemmarna i lönerådgivningsfrågor. Man vill t ex kunna svara på hur mycket man bör tjäna på en viss tjänst och med en viss kompetensnivå. Statistiken kan gälla både i nuläget och över tid. Därför finns det ett behov av att för varje förhandlad läkare lagra löneuppgifter för ett antal förhandlingar bakåt i tiden.

Övrig verksamhet

I den övriga verksamheten ingår att bistå medlemmarna i andra fackliga frågor som t ex arbetsplatsproblem eller med statistik och råd inför egna löneförhandlingar, anordnande av fackliga kurser m m. Några gånger om året skickas olika typer av information ut till samtliga medlemmar m h a adressetiketter från medlemsregistret. Man utför även vissa tjänster åt de lokala yrkesföreningarna[9] som t ex utskick till deras medlemmar. Även etiketter eller listor över olika kombinationer av information dras ut ur medlemsregistret för olika ändamål, t ex alla kvinnliga medlemmar. Man säljer också annonser i den egenproducerade Läkarkatalogen och Telefonkatalogens Blå Sidor, där bl a de privatpraktiserande läkarna i Göteborg med omnejd kan annonsera.

Förutom den statistik som sammanställs utifrån löneförhandlingarna har man ibland även behov av att dra fram andra typer av statistik. Det kan exempelvis röra sig om att förutsäga pensionsavgångar i framtiden, både totalt och inom varje specialistkompetens, eller att ta reda på könsfördelningen inom olika specialistområden. Även för detta måste medlemsregistret fungera.

[8] Exempelvis vikariat, ordinarie, vilande.

[9] Det finns sju stycken yrkesföreningar: för medicinstuderande, underläkare, specialistläkare, chefsläkare, distriktsläkare, företagsläkare och militärläkare.

Databasen

Till hjälp i verksamheten har man idag ett medlemsregister som innehåller uppgifter om namn och adress, yrkesföreningstillhörighet (en eller flera), specialistkompetens[10] (en eller flera), eventuella fackliga förtroendemannauppdrag[11] (ett eller flera), årtal för examen, legitimation, specialisering, eventuell disputation och docentur samt vissa uppgifter om anställning och lön. Man sparar även uppgifter om datum för inträde och utträde både för GLF och för yrkesföreningarna eftersom man inför varje årsmöte sammanställer statistik om detta. Man kan även vara huvudsaklig medlem i någon annan lokalförening[12] än GLF men ändå vara ”dubbelansluten” till GLF av olika skäl och detta specificeras också i medlemsregistret.

Databasen i verksamheten

GLF ingår som nämnts i Sveriges Läkarförbund och en stor del av de uppgifter som lagras i medlemsregistret kommer till föreningen därifrån. Information om nytillkomna och avgående medlemmar i GLF liksom information om ändringar i namn och adress och byte av yrkesförening kommer på papperslistor ungefär en gång i månaden och förs in manuellt. Annan information som behöver hållas uppdaterad, som t ex uppgifter om tidpunkt för legitimation, specialistkompetens och tidpunkt för specialisering kommer dock inte automatiskt från något håll utan detta måste sökas upp i olika förteckningar och sedan föras in. Detta är naturligtvis något av ett problem, men eftersom det beror på Läkarförbundets rutiner och system så är det tyvärr inte något vi kan ändra på. Man håller dock på att utarbeta nya rutiner och snart kommer det att finnas möjlighet att via en ISDN-anslutning hämta uppgifter i Läkarförbundets centrala register.

I databasen finns ett ganska stort antal fördefinierade urval och rapporter inlagda som är åtkomliga genom en enkel knapptryckning. Det uppstår dock problem om man behöver dra ut ett urval eller en rapport som inte finns förinlagd, eftersom man då får ta in en extern konsult som ordnar detta. Överhuvudtaget är det så att om någonting förutom den rena datan behöver förändras eller läggas till i databasen så kan inte personalen göra så mycket, utan man måste då ta in en utomstående person som gör förändringarna.

Idag är det så att databasen i princip bara används av den administrativt ansvariga personen på kansliet. Hon håller databasen uppdaterad genom att registrera alla ändringar som kommer och sköter med hjälp av registret utskick och informationsutdrag. Hon är i princip nöjd med databasen och tycker att den fungerar bra och innehåller de uppgifter som är relevanta.

Trots att det alltså bara är en person som idag använder databasen är den mycket viktig för verksamheten. Även den övriga personalen använder systemet indirekt genom att fråga den administrativt ansvarige. Databasen används dagligen och en stor del av verksamheten är beroende av att den fungerar. Det är t ex nödvändigt för utskick och service till yrkesföreningarna att adressetiketter och listor kan dras fram och det är även viktigt att ha tillgång till medlemsregistret för att kunna svara på t ex telefonförfrågningar.

[10] Exempelvis kirurgi, internmedicin eller allmänmedicin.
[11] Exempelvis förtroendeläkare, fackligt ombud eller uppdrag i föreningens styrelse.
[12] Vilken lokalförening en fackligt ansluten läkare tillhör beror på var han eller hon arbetar någonstans.

Att de övriga två i personalen inte använder databasen beror på att den dels inte innehåller alla de uppgifter de har behov av och dels att många av de funktioner de ändå skulle kunna använda med den befintliga informationen inte går att utföra på ett smidigt sätt. Alla de funktioner de efterfrågar rör i princip löneförhandlingar i någon form, och det är den delen som fungerar sämst i det nuvarande registret. Man lagrar exempelvis inte löneuppgifter för mer än den senaste förhandlingen, och det finns en del uppgifter som man skulle behöva ha med som inte alls finns i systemet. Det finns inte heller plats för mer än en statlig eller kommunal tjänst per medlem, trots att det är ganska vanligt att medlemmar är anställda på flera tjänster samtidigt varav en t ex är vilande.

Ett ytterligare hinder är införseln av löneförhandlingsuppgifterna i systemet. Eftersom man löneförhandlar för ca två tredjedelar av medlemmarna vid varje löneöversynsförhandling så är det en ganska stor mängd uppgifter som behöver registreras. Lönelistorna ligger som nämnts i MS Excel och det är detta program som används vid förhandlingarna och där även resultatet av löneförhandlingen läggs in, d v s uppgifter om den nya lönen m m. Idag måste dessa uppgifter föras in i medlemsregistret manuellt vilket naturligtvis är ett mycket tidsödande och långtråkigt arbete. Risken för att uppgifterna blir felaktigt införda är dessutom stor. Samma problem förhindrar möjligheten att inför löneförhandlingar föra över relevant information om de medlemmar som ska förhandlas till lönelistorna i Excel.

Databasens struktur och uppbyggnad

Databasen är konstruerad utan att någon formell metod har följts under uppbyggnaden och den följer inte någon teori om systemutveckling eller databaser. Databasen har trots detta fungerat relativt bra under de år den har varit i användning, men den uppvisar stora brister som ur flera aspekter gör den svår att förstå och förändra.

Trots att databasen är skapad i relationsdatabashanteraren MS Access är den inte någon relationsdatabas utan all information ligger samlad i en enda stor tabell. Uppgifter som kan vara flervärdiga, t ex yrkesföreningstillhörighet eller specialistkompetens, lagras i samma fält på rad efter varandra. För att göra databasen användarvänlig och minska risken för felaktiga inmatningar, liksom risken för att samma sak ska benämnas på olika sätt, har för relevanta fält de olika alternativa värden som kan förekomma hårdkodats in i systemet. De visas i form av ett formulär med kryssrutor där man väljer det eller de värden man önskar, vilka sedan skrivs ut i fältet och lagras i databasen. Mycket i databasen är också namngivet i form av förkortningar vilka kan vara svåra att tolka. Den dokumentation som finns för systemet är bristfällig och svår att förstå.

Databasen har under åren också genomgått stora förändringar i och med att ny information har behövts och befintlig information har blivit utdaterad. Detta har naturligtvis bidragit till att systemet har blivit än mer ostrukturerat och svårförståeligt. Förändringarna har sällan följts upp i dokumentationen vilket gör att denna i dagsläget är i princip oanvändbar.

Sammantaget är databasen mycket svår att sätta sig in i för en utomstående person. Det är svårt att förstå hur olika saker hänger ihop – det finns t ex en del verifieringar[13] som görs men eftersom dessa är implementerade i form av kod i formulären eller i makron utan någon vidhängande beskrivning kan man bara förstå dem genom att förstå koden eller makrot. Ett annat problem är att det på vissa ställen saknas säkerhetsfunktioner som kontrollerar hur informationen i databasen får ser ut, t ex kan man i personnummer skriva in ett nummer som är mindre än tio siffror. Även de kryptiska namnen i form av förkortningar ställer till problem.

Detta tillsammans med den dåliga strukturen och de hårdkodade bekvämlighetsfunktionerna gör att databasen är svår att förändra och uppdatera. Databasen kan beskrivas som mycket oflexibel och svårunderhållen.

Ett annat problem är att man så fort man gör något med databasen öppnar eller söker igenom hela den samlade datamängden. I en relationsdatabas arbetar man i princip bara med de delar man för tillfället är intresserad av, resten används inte. Den stora datamängden gör att databasen blir ganska trög och det kan ta långt tid att t ex köra en fråga.

Trots att databasen inte är så gammal uppfyller den alltså de flesta kriterier på ett legacy system: den är uppbyggd utan någon formell metod på ett ostrukturerat sätt, den lider av bristfällig design och den har förändrats mycket genom åren, ibland i form av krisframkallat underhåll. Den innehåller också hårdkodade värden och kryptiska namn. Den saknar korrekt dokumentation och är numera svår att förstå, underhålla och vidareutveckla.

6.1.2 Önskemål på det nya systemet

GLF har i samband med att Västra Götalandsregionen skapades inlett ett närmare samarbete med de övriga tre lokalföreningarna i regionen. Man har kommit överens om att samtliga föreningar skall använda likadana medlemsdatabaser så att man har möjlighet att sammanställa ett medlemsregister för hela regionen. Ett krav på det nya systemet är alltså att det skall vara designat så att det passar verksamheten hos alla föreningarna – det måste alltså gentemot det gamla systemet generaliseras en del.

Ett annat krav är att lönedelen ska fungera. Ett antal nya uppgifter måste kunna sparas, bl a löneuppgifter från förhandlingar bakåt i tiden. Man måste även kunna lagra flera tjänster per medlem och se lönehistoriken för varje tjänst. En automatisk överföring från Access till Excel för att föra över relevanta medlemsuppgifter inför löneförhandlingar, liksom en liknande funktion från Excel till Access för att föra in de nya uppgifterna efter slutförda löneförhandlingar, ska implementeras. Möjlighet att föra tillbaka utvalda kombinationer av information från Access till Excel för statistiskt arbete ska finnas.

[13] Exempelvis finns en kontroll av att man inte är registrerad som student om man samtidigt har ett datum för examen.

Ett annat önskemål är någon form av funktion för att kunna skapa åtminstone enklare urval och rapporter utan att använda Access något svårarbetade funktioner för detta, och utan att behöva kalla in någon utomstående person. Även ett antal urval och rapporter som man bedömer ofta kommer att användas skall fördefinieras och sparas i databasen lätt åtkomliga för användarna.

Det finns också ett behov av att kontrollera den information som förs in i systemet på ett bättre sätt än vad som sker i det gamla systemet, t ex en kontroll av att personnumret är korrekt.

Det finns även i registret information som inte längre används eller som man har insett inte går att hålla uppdaterad eftersom man inte kan få uppgifterna någonstans ifrån på ett smidigt sätt, och som man därför kan ta bort. Ett exempel är möjligheten att specificera vilka läkare som arbetar privat och vilken den privata arbetsplatsen är.

Kortfattat är vår uppgift alltså att identifiera nya informations- och funktionalitetsbehov samt sådan information som inte längre behövs, samt att generalisera databasen och göra den mer flexibel och lättunderhållen. Det nya systemet skall också dokumenteras på ett bra sätt. (för en översiktlig kravspecifikation se 6.1.3, Målformulering)

6.1.3 Hur vi använde vår metod

I detta stycke redogör vi steg för steg för hur vi använde oss av vår metod i reengineeringen av databasen. Under alla steg hade vi fortlöpande kontakt med användarna genom telefonsamtal, mailkontakter och personliga möten.

Verksamhetsanalys

Vi menar som nämnts i genomgången av vår metod att verksamhetsstudien är en mycket viktig del i en reengineeringsprocess och att man bör börja med att titta på verksamheten innan man analyserar det befintliga systemet. Vi hade mycket hjälp i båda dessa aktiviteter av att en av författarna hade erfarenhet av både verksamheten och det gamla systemet sedan tidigare. Dock var det ändå relevant för oss att gå igenom båda faserna eftersom en av författarna inte tidigare kände till verksamheten eller systemet. Det fanns dessutom delar av verksamheten som var nya för oss båda två, t ex lönedelen som är en viktig del av det nya systemet, liksom delar som var osäkra och behövde klargöras ytterligare.

Vi började alltså med att intervjua de anställda på kansliet för att få en bild av både hur verksamheten fungerade och vilka delar av verksamheten som behövde stödjas i det nya systemet. Vi undersökte också hur det gamla systemet fungerade i verksamheten och vad personalen upplevde som problematiskt och vad som upplevdes som positivt. Resultatet av verksamhetsanalysen presenteras kortfattat under de första delarna i stycke 6.1.1.

Reverse engineering

Efter verksamhetsanalysen gjordes en noggrann genomgång av det gamla systemet för att inventera alla funktioner som fanns, vilken data som lagrades och vilka kontroller eller verifieringar som gjordes av denna data. Detta beskrivs översiktligt under den sista delen av stycke 6.1.1. För att undersöka vilka kontroller och verifieringar som fanns användes reverse engineeringssteget programanalys i en begränsad form, genom att kod och makron undersöktes. En mindre omdokumentering av dessa delar gjordes genom att förklaringar för dessa återskapades. Arkitekturen återskapades också i form av en objektmodell över det gamla systemet (se figur 18).

Medlem	
Namn	Fackligt förtroendeuppdrag
Personnummer*	Yrkesförening
Medlemskod	Indatum yrkesförening
Kön	Utdatum yrkesförening
Glfmedlem	Privatläkare
Indatum	Privat arbetsplats
Utdatum	Anteckningar
Adress	Utlandstjänst
C/O	Kommunal tjänst
Postnummer	Statlig tjänst
Ort	Huvudarbetsplats
Land	Kommunal arbetsplats
Telefonnummer	Statlig arbetsplats
Dubbelansluten	Anställninstyp kommunal
Specialistår	Anställningstyp statlig
Specialistkompetenskod	Kommunalt uppdragstillägg
Docent år	Statligt uppdragstillägg
Disputerat år	Statlig ny totallön
Student	Kommunal ny totallön
Pensionär	Yrkesföreningshistorik
Examensår	Fd Bohus
Legitimeringsår	

Figur 18 – Objektmodellen för det gamla systemet

Målformulering

En noggrann genomgång av de funktioner och egenskaper som var önskvärda i det nya systemet gjordes tillsammans med personalen på GLF under ett flertal möten och samtal. Detta beskrivs översiktligt under stycke 6.1.2. Utifrån detta fastställdes en form av målbeskrivning eller kravspecifikation som översiktligt innefattar följande punkter:

Alla funktioner som inte längre används skall tas bort, t ex privat arbetsplats.
Verksamheten hos de övriga tre föreningarna i Västra Götaland ska undersökas och databasen anpassas så att den fungerar i alla fyra föreningarna.
Ett antal urval och rapporter skall fördefinieras och läggas i databasen lätt åtkomliga för användarna, t ex alla medlemmar per valfri specialistkompetens.
Möjlighet att välja ett antal valfria uppgifter att kombinera i en fråga eller rapport utan att använda Access inbyggda funktioner skall implementeras. Detta för att underlätta för användarna när de önskar ett urval som inte finns förinlagt, t ex alla medlemmar per valfri specialistkompetens födda mellan valfria år.
Säkerhetskontroller av viss data ska implementeras, exempelvis skall det inte vara möjligt att lägga in ett felaktigt personnummer.
Fält för de löneuppgifter som idag inte lagras alls skall läggas till, exempelvis kostnadsställekod och kostnadsställe i klartext.
Möjlighet att spara löneuppgifter för varje person och tjänst under de senaste tio förhandlingsomgångarna skall läggas till.
Automatisk överföring från Excel till Access av utvalda uppgifter efter slutförd löneförhandling skall implementeras[14].
Automatisk överföring från Access till Excel av utvalda uppgifter inför löneförhandlingar skall implementeras[14].
Möjlighet att föra över valfria uppgifter från Access till Excel för statistikarbete skall implementeras[14].
Hårdkodade värden skall tas bort ur systemet och lagras i en form som är lättare att uppdatera.
Dokumentation skall finnas för alla delar i systemet.

Modellering

När kravspecifikationen var fastställd var det dags att börja modellera systemet enligt vår anpassade KRB-metod.

Identifiera kandidatklasser

Vi började med att utifrån objektmodellen för det gamla systemet (se figur 18) sammanställa en lista med kandidatklasser för det nya systemet. Eftersom det gamla såg ut som det gjorde – det bestod bara av attribut – fick vi göra klasser av många attribut. Vi utgick även från den nya kravspecifikationen för att få med klasser för de nya funktioner som skulle implementeras i systemet.

Definiera klasser

När listan över kandidatklasser var klar bestämde vi i samråd med GLF:s personal vad som var relevant att ta med i systemet, genom att definiera vad varje klass innebar och vilken information den skulle innehålla. Vi fick godkännande för vilka funktioner som inte längre behövdes, alltså vilka klasser och attribut som kunde tas bort.

[14] Detta kommer inte att ingå som en del i fallstudien för uppsatsen, men systemet skall i designen anpassas för att klara av detta.

Etablera relationer och Utveckla M:M förhållanden

För att bestämma de relationer som skulle finnas i systemet följde vi i stort sett KRB-metodens modell – vi kollade alla klasser mot varandra i samråd med personalen. I detta steg hade vi ingen större nytta av det gamla systemets objektmodell eftersom det gamla systemet inte innehöll några relationer. Samtidigt som relationerna skapades kontrollerade vi också deras kardinalitet. När vi upptäckte relationer som hade ett M:M förhållande åtgärdade vi detta direkt. Man kan alltså säga att vi bakade ihop steg 3 och steg 4 till ett steg.

Definiera attribut

I steg 5 ska alla attribut i systemet bestämmas. Detta hade vi till stor del redan gjort – vi fick mycket från det gamla systemet, och i de nya klasser som kom fram bestämde vi ofta attributen redan när klassen definierades. Vi gick dock igenom alla klasser ändå för att vara säkra på att inte missa något relevant attribut, och vi bestämde också vilka attribut som skulle vara nyckelvärden i de olika klasserna.

Normalisering

Ovanstående steg resulterade i en första objektmodell för det nya systemet. Denna var till stor del redan normaliserad. Detta berodde på att vi under alla de föregående stegen varit uppmärksamma på klasser och attribut som kunde tänkas bli problematiska och normaliserat dessa när de dök upp. En annan orsak kan ha varit det faktum att vi hade ett gammalt system att utgå ifrån och att det såg ut som det gjorde: eftersom det vi började med var en lång lista med en stor del av de attribut vi ville ha i det nya systemet, var det förmodligen lättare att från början dela upp dessa i riktiga, normaliserade klasser än om man börjar med att definiera klasser utan att tänka på attribut. Man börjar så att säga med de ”minsta” delarna i systemet, attributen, snarare än de ”största”, klasserna.

Operationer

Steg sju, kontrollen av att objektmodellen är riktig och innehåller allt den behöver, gjorde vi enligt en mer informell metod än de KRB rekommenderar. Vi gick igenom objektmodellen och för varje klass tänkte vi igenom de möjliga händelser som skulle kunna påverka klassen och såg att modellen var anpassad för dessa. Vi gjorde dock inga klasskort eller diagram över detta utan kontrollerade mer informellt att allt hängde ihop. För relevanta klasser bestämdes även vilka tillstånd som är tillåtna. Exempelvis kan klassen yrkesförening bara innehålla någon av de befintliga yrkesföreningarna[15], och för klassen medlem kan det inte finnas ett tillstånd där både medlem och ett datum för utträde ur föreningen är ifyllt.

[15] Det finns sju stycken yrkesföreningar: för medicinstuderande, underläkare, specialistläkare, chefsläkare, distriktsläkare, företagsläkare och militärläkare.

Implementation

Modelleringen resulterade i en komplett objektmodell för hur GLF ville ha systemet. Innan vi kunde börja implementera denna behövde vi dock ta hänsyn till även de övriga tre föreningarnas verksamhet. Utifrån objektmodellen gjordes därför en presentation av systemet för de övriga tre föreningarna. De fick sedan komma med frågor och önskemål och under ett längre möte sammanfogades deras önskemål med GLF:s. Utifrån detta modifierades objektmodellen för att inkorporera allas önskemål. Eftersom verksamheterna i de olika föreningarna är relativt likartad så var det inte några stora diskussioner om innehållet i systemet. Det fanns dock önskemål om viss funktionalitet som var individuell för olika föreningar. Detta löstes genom att lägga till detta i systemet, med överenskommelsen att de föreningar som inte vill använda funktionerna inte behöver göra det. Det viktiga ansåg vi var att den grundläggande strukturen i databasen ser likadan ut för samtliga föreningar – detta underlättar sammanslagningen av de olika registren till ett gemensamt.

Slutresultatet var en slutgiltig objektmodell för det nya systemet (se appendix 1), som sedan kontrollerades igen för att säkerställa att den var normaliserad och inte innehöll några problematiska relationer.

För implementeringen valde vi den något mer osäkra reengineeringsstrategi 1, nytt system, (Tilley, 1995) (se stycke 3.1.1) och gjorde om systemet från början. Det kändes motiverat i detta fall eftersom det gamla systemet var så ostrukturerat och det egentligen inte fanns något annat att återvinna ur det än bakomliggande information, som t ex vilka funktioner som användes, vilken data som lagrades och vilka verifieringar av denna data som gjordes. Det hade varit svårt att göra om små bitar i taget och koppla samman dessa med det befintliga systemet som i ett evolutionssystem (Tilley, 1995). Det var inte heller praktiskt att skapa och ta i bruk små enskilda delar åt gången, som man gör med ett tilläggssystem (Tilley, 1995), eftersom systemet var så pass begränsat. Farorna med att göra ett nytt system (Gustafsson & Johansson, 1994) har också kunnat hanteras tillfredsställande. Eftersom systemet var relativt litet så var det ganska lätt att få med även odokumenterade beroenden. Vi har helt enkelt specificerat alla beroendeförhållanden[16], dokumenterade eller ej. Det har inte heller varit svårt att påvisa utökad och ny funktionalitet eftersom det var ett av de grundkrav som ställdes från GLF då vi blev ombedda att göra om systemet. Specifikationerna till det gamla systemet var klart bristfälliga, men eftersom systemet var relativt litet och eftersom en av oss hade erfarenhet av systemet sedan tidigare så orsakade inte heller detta några stora problem. Det tidigare systemet var under hela utvecklingen i drift och fungerade någorlunda bra, vilket gjorde att vårt arbete inte var tidskritiskt på samma sätt som det kunde ha varit om det gamla systemet inte längre gått att använda.

Utifrån objektmodellen började vi sedan implementera systemet i Access. Under hela implementationen utvärderade vi det framväxande systemet genom att fortlöpande kontrollera osäkra delar med personalen på GLF genom mail-, telefon- och personliga kontakter.

[16] Exempelvis så är möjligheten att vara facklig förtroendeman beroende av att man är medlem.

En del saker var vi under implementeringens gång tvungna att ändra. Vi hade t ex för relationerna specialistkompetensinnehav, yrkesföreningstillhörighet och fackliga förtroendemän i modellen inte tagit hänsyn till partialitet/totalitet (se stycke 3.2.1, Partialitet/Totalitet). Eftersom detta inte var något som togs upp i KRB-metoden tog vi inte heller upp det i vår metod. Nu upptäckte vi att partialitet i vissa fall kunde orsaka problem. För ovanstående relationer fanns det inte uppgifter för alla medlemmar (det finns t ex medlemmar som inte har någon specialistkompetens eller något fackligt förtroendeuppdrag). Vi hade först valt att helt enkelt låta dessa värden vara tomma (att t ex inte ha någon representation för ickespecialister i klassen för specialistkompetensinnehav), men nu upptäckte vi att det ställde till stora problem under utsökningen av data. Som ett exempel kan nämnas en parameterfråga[17] där man kan söka efter olika kombinationer av läkare utifrån specialistkompetens och fackligt förtroendeuppdrag. Om man med hjälp av denna fråga vill söka efter alla kirurger, oavsett om de har ett fackligt förtroendeuppdrag eller inte, går detta inte att genomföra. Det går nämligen inte att ange ett villkor som matchar både medlemmar med fackligt förtroendeuppdrag och medlemmar utan – man får bara ut de kirurger som har ett fackligt förtroendeuppdrag.

Så snart vi upptäckt detta problem återgick vi till litteraturen för att avgöra hur vi skulle lösa problemet. Vi hittade två alternativ. Antingen kan man låta de värden som man inte har uppgift om vara tomma eller NULL (NULL representerar värdet ”okänt” – att värdet inte är aktuellt för posten eller att det inte är känt). Det andra alternativet är att skapa ”falska” värden som representerar att det inte finns någon uppgift för värdet, t ex ”Nej”, ”Ingen” eller ”00”. (Conolly et al., 1998; Date, 1995) Det finns olika uppfattningar bland olika författare om vilken av dessa lösningar som är den bästa. Date t ex anser att NULL-värden inte alls hör hemma i en databas, utan att det för varje post i databasen ska finnas ett värde för varje nyckel. Han menar att NULL-värden ”... *undermines the logical foundations of the relational model.*”. (Date, 1995) Codd å andra sidan tycker att NULL-värden är en integrerad del av databasen (Conolly et al., 1998).

Vi valde att följa Dates rekommendation och skapa ”falska” värden. Vi tyckte dels att detta skulle vara tydligare för användaren och dels underlättade detta utsökningarna ytterligare. Således lade vi för klasserna Specialistkompetens, Yrkesförening och Fackligt förtroendeuppdrag in värden som ”Nej”, ”Ingen” och ”Inget”, och satte dessa som standardvärden för nya poster. I och med detta hade vi löst vårt problem och datautsökningen fungerade – nu kan man ange jokertecknet ”*” i fackligt förtroendeuppdrag och få ut alla kirurger, oavsett förekomsten av fackligt förtroendeuppdrag.

En annan sak som kan vara problematisk att implementera är kontroller av data som skrivs in, om dessa är av mer komplex natur. Vi hade t ex svårigheter med en kontroll av att personnummer är av rätt form, d v s att de nio första siffrorna i personnumret stämmer med den tionde siffran, kontrollsiffran. Denna implementerades i form av ett Visual Basic-script där en beräkning av detta görs.

[17] En fråga som frågar efter utsökningsvillkor vid varje körning

Dataöverföring

När implementationen var så pass färdig att systemet skulle börja användas var det dags för dataöverföringen från det gamla systemet till det nya. Detta innebar ganska stora problem eftersom mycket av datan i det gamla systemet lagrades på ett sätt som helt skiljer sig från det nya systemet. Exempelvis så lagrades i det gamla systemet personnummer utan bindestreck, men i det nya systemet vill man ha bindestreck för att underlätta överföringen mellan Excel och Access. Detta löstes genom ett litet c-program som i varje personnummer skrev ett bindestreck på rätt ställe.

Ett annat exempel är att det i vissa fält lagrades flera värden på rad efter varandra, separerade med kommatecken (se figur 19). För att få en normaliserad objektmodell var vi tvungna att ta isär dessa och lägga dem i enskilda poster. Ibland fanns det även andra värden som hörde ihop med dessa och som i det nya systemet fick kopplas ihop med någon av de nya posterna. Ett exempel är specialistkompetens och specialiseringsår. Här fick man välja vilken specialistkompetens man ville koppla ihop året med eftersom det inte gick att utläsa ur datan till vilken kompetens året hörde. Man måste dessutom hålla reda på personnumret för varje värde så att rätt specialistkompetens kopplas ihop med rätt person.

Personnummer*	**Specialistkompetens**	**Specialiseringsår**
111111-1111	01, 03	1980
222222-2222	08	1986

Figur 19 – Exempel på datalagring i det gamla systemet

Detta löste vi genom att köra de relevanta uppgifterna personnummer, specialistkompetens och specialiseringsår genom ett c-program, som plockade isär värdena för specialistkompetens och kopplade ihop varje värde med rätt personnummer. Året kopplades till den första kompetensen. (se figur 20) Sedan importerades detta till det nya systemet. För övriga fält med flera värden, yrkesföreningstillhörighet och facklig förtroendeman, gjordes liknande lösningar.

Personnummer*	**Specialistkompetens***	**Specialiseringsår**
111111-1111	01	1980
111111-1111	03	
222222-2222	08	1986

Figur 20 – Exempel på datalagring i det nya systemet

När all data var överförd från det gamla systemet till det nya gjordes en kontroll av att överföringen hade blivit korrekt gjord, genom att ett antal poster slumpvis valdes ut och kontrollerades manuellt mot det gamla systemet.

Utvärdering

Systemet är idag i användning på GLF och fungerar. Det befinner sig dock fortfarande under utveckling, främst vad gäller gränssnittet, men även överföringsrutinerna mellan Excel och Access återstår att implementera. På grund av detta har vi valt att inte genomföra några intervjuer med användarna ännu. Vi har inte heller gjort någon kontroll av att systemet motsvarar kravspecifikationen, eftersom alla delar i systemet ännu inte är färdigimplementerade och kravspecifikationen därför inte är helt uppfylld. Vi anser dock att båda dessa typer av summerande utvärdering är mycket viktiga och att de alltid bör utföras innan ett systemutvecklingsprojekt kan ses som avslutat.

Vi har dock gjort en utvärdering av systemet i form av en jämförelse med det gamla systemet i enlighet med följande kriterier, som vi anser vara ett bra mått på systemets förändring och förbättring:

- **Ökad funktionalitet** – vilka nya funktioner har tillkommit och vilka överflödiga funktioner har tagits bort?
- **Ökad flexibilitet** – är det nya systemet lättare att underhålla och på vilka sätt? Är det lättare att använda?
- **Ökad generalitet** – är det nya systemet mindre beroende av den specifika verksamheten och på vilka sätt?

Ökad funktionalitet
Möjlighet att lagra ett årtal för varje specialistkompetens
Möjlighet för personalen att själva göra enklare urval och rapporter
Möjlighet att spara fler uppgifter om lön, som t ex flera tjänster per person
Möjlighet att spara löneuppgifter för varje person och tjänst tio förhandlingar bakåt i tiden – ökad möjlighet till statistikföring
Överföring från Excel till Access efter löneförhandlingar*
Överföring från Access till Excel inför löneförhandlingar*
Överföring från Access till Excel för statistikarbete*
Ett antal funktioner ur det gamla systemet som inte längre används har inte implementerats i det nya systemet
Viss ny säkerhetsfunktionalitet har implementerats

**Dessa delar kommer snart att vara i funktion*

Ökad flexibilitet
Möjlighet att enkelt utforma urval och rapporter genom val i ett formulär
Hårdkodade värden har tagits bort i och med den nya strukturen, som innebär att personalen själv kan ändra dessa
Systemet är tack vare den nya strukturen lättare att förstå, vilket ökar möjligheten att underhålla och vidareutveckla det och sårbarheten minskar eftersom det nu är lättare för utomstående att sätta sig in i systemet
Systemet kräver mindre underhåll eftersom personalen nu kan göra fler saker själva

Ökad generalitet
Strukturen och funktionaliteten har anpassats till samtliga fyra föreningars verksamhet – idag skulle alla fyra föreningarna kunna använda systemet i sin dagliga verksamhet

Det finns även några saker som man kan säga har försämrats om man jämför med det gamla systemet. Adressetiketter fungerar inte längre så bra att skriva ut – de tre etiketterna i nedersta raden på sidan lämnas tomma. Detta är dock något som beror på Access97 och som vi inte har kunnat åtgärda (även på de i Access97 fördefinierade etikettsorterna uppstår samma problem). Ett annat problem är listrapporter som inte längre går att få lika kompakta som tidigare. Med den gamla strukturen med flera värden på samma rad gick det också att få ut snygga rapporter med allt samlat på en enda rad, vilket naturligtvis inte längre går eftersom värden nu hämtas från olika poster och tabeller.

Ett annat resultat av omstruktureringen är att databasen i dag tar mycket mera plats på hårddisken, på grund av att den innehåller fler tabeller och fler formulär. Den går dock snabbare att arbeta med eftersom man ofta bara använder vissa delar av informationen åt gången.

7 Utvärdering av reengineeringsmetoden

Enligt Müller, Tilley och Wong (1993) är det är viktigt att testa sina metoder på stora projekt och inte bara på små prototyper. Detta eftersom metoder som fungerar bra för reengineering av små system tenderar att ofta inte fungera så bra i samband med reengineering av stora system. Eftersom vår fallstudie grundar sig på ett relativt litet system, en databas med ca 2800 poster, är denna studie alltså enligt författarna ovan inte stor nog att ha som grund när man drar slutsatser om hur väl metoden har fungerat. Vi ser dock möjligheten att dra slutsatser om hur vår metod fungerat just i samband med relativt små system. Även system som dessa är vanliga och är därför intressanta att studera. Vi försöker oss dock alltså inte på att göra någon generell metod för reengineering utan håller oss till att diskutera hur man kan göra om relativt små databassystem utifrån reverse engineering och reengineering.

Vi börjar med att konstatera att metoden fungerade bra i vår fallstudie. Vi upplevde att den underlättade systemutvecklingen och medförde att vi arbetade på ett mer systematiskt och kontrollerat sätt. I ett reengineeringsprojekt tror vi att det är extra viktigt att arbeta strukturerat, eftersom det finns så mycket bakomliggande information och kunskap som man kan ha nytta av. En bra metod kan hjälpa systemutvecklaren att hitta all relevant information och därmed undviks att delar som skulle kunna hämtas från det gamla systemet görs om. Tack vare detta kan projektet förhoppningsvis genomföras både billigare och snabbare. Vi anser att vår metod fungerade bra i detta hänseende.

En metod kan sägas vara ”*…en uppsättning förutbestämda regler, principer och arbetsmoment för hur forskaren skall bedriva sitt arbete.*” (Larsson, 1986) Vi tycker alltså att vår metod uppfyller detta och att den var till hjälp för oss i vårt arbete.

Nedan utvärderar vi reengineeringsmetoden mer ingående utifrån de kriterier som togs upp i Metod (stycke 4):

- **Hur väl gick metoden att följa i praktiken** – var de olika stegen relevanta? Kom de i rätt ordning? Vad behövde förändras?
- **Vad saknades i metoden** – behövdes det några ytterligare steg?
- **Vad var överflödigt i metoden** – kunde något steg tas bort eller kombineras med något annat steg?

7.1.1 Hur väl gick metoden att följa i praktiken

Vi upplevde att metoden fungerade bra i praktiken. Alla steg kändes relevanta och kom i en lämplig ordningsföljd och några större förändringar behövde inte göras. Dock vill vi påpeka att den tid man lägger ned på de olika stegen kan variera från projekt till projekt. Vi utförde t ex en relativt sett ganska liten omdokumentering. I projekt med andra förutsättningar tror vi dock att detta steg kan och bör bli mycket mer omfattande. Målformuleringen å andra sidan lade vi ned ganska mycket koncentration på, vilket vi tror är riktigt i så gott som alla projekt. Eftersom vi dessutom skulle försöka nå fram till ett mål gemensamt för fyra olika föreningar var det naturligt att detta blev ett ganska stort steg.

I modelleringsstadiet insåg vi att det i praktiken var ganska svårt att följa modelleringsstegen steg för steg – man bakade mer eller mindre medvetet ihop olika steg så att man t ex normaliserade en klass redan vid införandet i modellen. Vi utförde dock alla stegen, och om man väljer att göra det i ordning eller inte tror vi kan bero på både person och situation. I ett större projekt med både fler användare och fler utvecklare blir det säkert lätt väldigt rörigt om man inte arbetar strukturerat, och det är lättare att missa saker. Detta är något som får avgöras från fall till fall. I princip anser vi dock att om man väljer att utföra alla steg i följd så kommer de i en lämplig ordning. Det finns också en poäng med att specificera varje delsteg tydligt eftersom man då får en kontroll på att alla stegen verkligen utförs. Vi tycker inte att man bör utelämna något delsteg.

Implementationen fungerade bra med den utvärdering vi gjorde under gång. Denna liksom användarmedverkan i verksamhetsanalysen utfördes med ganska informella metoder – när vi stötte på problem tog vi helt enkelt kontakt med en av användarna och frågade. I ett större projekt eller ett projekt med mer onåbara användare är detta naturligtvis ett större problem. Vi tror dock att det är ett mycket viktigt steg och ett sätt att lösa det kan vara att sätta upp fasta tider för möten under implementationen där sådana här frågor tas upp.

Även den summerande evalueringen ser vi som relevant. Vi har som tidigare sagts ännu endast gjort en utvärdering i form av en jämförelse med det gamla systemet. En utvärdering utifrån kravspecifikationen och användarintervjuer anser vi dock vara mycket viktig att göra. Men som sagts om utvärdering: det är bättre att göra en liten, snabb utvärdering än ingen alls (Preece, 1994). Det är alltid viktigt att involvera användarna i utvärdering och testning, trots att detta tenderar att bli kostsammare, eftersom det aldrig är möjligt för en systemutvecklare att förutse precis hur en användare kommer att reagera på och uppfatta olika fenomen. Fel och missförstånd kommer alltid att uppstå. (Helander, 1988)

7.1.2 Vad saknades i metoden

Vi fick som nämnts i stycke 6.1.3 under Implementation problem med partiella relationer. Eftersom detta inte överhuvudtaget nämns i KRB-metoden fanns det inte heller med i vår modelleringsdel. Detta ser vi som ett tillägg som bör vara ett eget steg efter steg 4 i modelleringsdelen. I detta nya steg går man igenom alla relationer i objektmodellen och undersöker om de ska vara partiella eller totala. När man hittar en partiell relation som kan ställa till svårigheter längre fram måste man avgöra hur man vill lösa problemet med att representera tomma värden (se stycke 6.1.3, Implementation).

7.1.3 Vad var överflödigt i metoden

Vi tycker att alla steg är relevanta och borde finnas med. Som vi nämnt tidigare vad gäller modelleringsstegen kan det vara lätt att i praktiken kombinera olika steg med varandra. Detta kan eventuellt även gälla för verksamhetsanalys och reverse engineering. Det kan kännas opraktiskt att skilja på dessa steg eftersom de till viss del har likartade syften, att ge verksamhets- och systemkännedom. Vi tror dock att det kan finnas en poäng i att börja med att titta på verksamheten oberoende av systemet, eftersom man annars riskerar att låsa fast sig vid det gamla systemets roll och funktionalitet och missa nya infallsvinklar. Att koncentrera sig för mycket på det gamla systemet innan man har skaffat sig en bild av verksamheten kan också ge en felaktig bild av verksamheten, eftersom det gamla systemet kan innehålla föråldrad verksamhetskunskap.

Även om vi tror att det ibland kan vara praktiskt att i praktiken baka ihop olika steg så menar vi att det är viktigt att de finns med åtskilda från varandra i metoden, eftersom man på så sätt får en större kontroll av att alla steg utförs.

7.2 Metoden i korthet

Efter att hänsyn har tagits till utvärderingen av hur metoden fungerade i fallstudien har vi kommit fram till följande, mer kompletta, metod:

1. **Verksamhetsanalys**
2. **Reverse engineering**
3. **Målformulering**
4. **Modellering av det nya systemet**
    1. **Identifiera kandidatklasser**
    2. **Definiera klasser**
    3. **Etablera relationer**
    4. **Utveckla M:M förhållanden**
    5. **Kontrollera partialitet/totalitet**
    6. **Välj relevanta attribut**
    7. **Normalisering**
    8. **Operationer**
6. **Implementation**
7. **Dataöverföring**
8. **Utvärdering**

8 Diskussion

Grunden för att använda sig av reengineering i ett systemutvecklingsprojekt ligger i att det gör projektet mer tids- och kostnadseffektivt och mindre riskfyllt (Tilley, 1995). Man bör alltså innan man påbörjar arbetet i ett reengineeringsprojekt undersöka att dessa villkor verkligen är uppfyllda, d v s att det är relevant att använda sig av reengineering snarare än nyutveckling. Något som bör undersökas i sammanhanget är hur stora förändringar av systemet som krävs för att de nya målen ska uppfyllas. Är de för stora är det troligt att det är bättre att göra ett helt nytt system än att lägga tid och pengar på att försöka utföra reengineering. (Somerville, 1995) I vår fallstudie skulle det nya systemet likna det gamla i många avseenden, t ex skulle mycket av funktionaliteten vara likadan. Vi fann det alltså relevant att använda oss av reengineering trots att det inte var så mycket av själva systemet som kunde återanvändas. Det fanns dock mycket återanvändningsbar kunskap i systemet som t ex ungefär vilken data man hade behov av att lagra och vilken typ av urval och rapporter som efterfrågades. Det fanns även en del verksamhetskunskap inbäddad i systemet som t ex vilka fackliga förtroendeuppdrag och vilka specialistkompetenser som finns.

I vilken grad det gamla systemet kan användas som grund för det nya beror naturligtvis på hur väl det gamla systemet fungerar. Om det i stort uppfyller de krav på funktionalitet som användarna ställer så är det troligen en bra utgångspunkt för vad som behövs i det nya systemet. Om det fungerar dåligt så kan systemet möjligen fungera som en utgångspunkt för vad som brister vilket kan vara till hjälp i det nya systemet, men det är förmodligen färre delar man direkt kan återvinna. I vårt fall fungerade det gamla systemet ganska bra utom vad gäller lönedelen. Vi har också kunnat återvinna väldigt mycket ur det gamla systemet utom just vad gäller löneuppgifterna.

En fara som vi ser med att använda reengineering vid systemutveckling är att man riskerar att låsa fast sig vid de lösningar och den funktionalitet som fanns i det gamla systemet. Om beställaren är ovan vid systemarbete har han/hon förmodligen svårt att ”tänka förbi” det gamla systemet, eftersom det är detta man länge har arbetat med och är van vid. Det är som systemutvecklare viktigt att hjälpa beställaren att se nya möjligheter som skapas i och med att man utvecklar ett nytt system, samtidigt som man naturligtvis själv måste hålla sig öppen för dessa. Ett sätt att undvika att låsa fast sig vid det gamla systemet är att, som metoden säger, utföra verksamhetsanalysen innan man börjar undersöka det gamla systemet. Då kan man skapa sig en bild av hur verksamheten skulle kunna stödjas innan man ser hur det gamla systemet i praktiken fungerar. Denna insikt tillsammans med en god förståelse av det gamla systemet kan hjälpa systemutvecklaren att föra fram idéer till beställaren och möjliggöra att en bra målbild för det nya systemet skapas.

Eftersom en av författarna kände till det gamla systemet i vår fallstudie sedan tidigare så verkade risken för att låsa fast sig vid detta ännu större. Dock motverkades detta av att den andra författaren varken var insatt i systemet eller verksamheten och därför kunde bidra med nya infallsvinklar. Det visade sig också att det i slutändan var mer positivt än negativt att ha verksamhetskunskapen eftersom detta i många fall underlättade systemutvecklingen, bl a genom att verksamhetsanalysen förenklades, samtidigt som det inte på något sätt som vi har kunnat upptäcka i praktiken utgjort något stort problem.

För att undvika att även det nyutvecklade systemet blir ett legacy system så tror vi att det är viktigt att man i designen av systemet tar hänsyn till att det ska vara vidareutvecklingsbart. Detta innebär att strukturen skall vara både begriplig och enkel att föra in ny funktionalitet i. Även namn skall vara tydliga och begripliga, man bör t ex undvika förkortningar. Man skall också korrekt och utförligt dokumentera alla delar i systemet. En policy för hur underhåll och vidareutveckling skall ske bör utformas. Denna policy bör klargöra hur dokumentation skall uppdateras, hur objektmodellen skall ändras och vilka typer av lösningar man bör använda när man gör om någonting i systemet.

Möjligheten till vidareutveckling är någonting vi har tänkt på när vi har utvecklat vårt system. Den nya strukturen gör det t ex möjligt att föra in ny funktionalitet utan att resten av systemet påverkas i någon stor utsträckning. Vi har varit noga med att använda tydliga namn och undvikit förkortningar. Eftersom systemet ännu inte är fullt implementerat är inte heller dokumentationen färdigställd, men detta kommer att göras i samband med att implementationen slutförs. Vi planerar även att utarbeta en policy för underhåll och vidareutveckling i samband med detta.

En stor fråga i all systemutveckling är användarmedverkan. I fallstudien fungerade det bra med användarmedverkan i form av intervjuer, diskussionsmöten, telefonkontakter och mailkontakter. I andra projekt kan det säkert vara bra att använda sig av andra metoder som t ex brainstorming – detta är också något som får avgöras från fall till fall. Det avgörs naturligtvis också av dels hur mycket tid användarna har att lägga och dels av hur mycket verksamhetskunskap systemutvecklaren har. Även projektets storlek och antalet användare är av betydelse – hur många måste delta i beslutsfattandet? Enligt vår uppfattning fungerade användarmedverkan i den form vi hade eftersom det i större delen av modellering och implementering bara var tre personer som var med och påverkade. De använder också till stor del olika delar av systemet vilket innebar att mycket bara berörde t ex den administrativt ansvariga personen och då behövde bara hon rådfrågas.

I vårt projekt, liksom säkert även i många andra, hade användarna inte obegränsat med tid att ägna åt utvecklingen av det nya systemet. Detta innebar i vissa fall problem. Ofta kunde vi dock handskas med dessa genom att sammanställa våra frågor och sedan ta upp dessa vid ett och samma tillfälle, exempelvis i samband med ett telefonsamtal. En annan lösning som skulle kunna fungera om det är flera användare inblandade vore att bestämma fasta mötestider, exempelvis en gång i veckan, och samla alla sina frågor till det tillfället. Det väsentliga är inte hur man väljer att lösa problemet utan att man är medveten om att det är ett problem som måste lösas. Enligt författarna till KRB-metoden kan otillräcklig tid från användarna förstöra framgången hos ett helt projekt (Brown, 1997), och detta är något som vi håller med om.

9 Slutsatser

En slutsats som vi drar är att det i de flesta fall där det finns ett gammalt system att utgå ifrån är relevant att använda sig av det som grund i utvecklingsarbetet. Även om det nya systemet kan tyckas skilja sig mycket från det ursprungliga så finns det troligen mycket bakomliggande information att hämta. Vi anser alltså att det väldigt ofta är bra att använda sig av reengineering snarare än nyutveckling.

För att i så stor utsträckning som möjligt utnyttja de möjligheter det gamla systemet erbjuder tror vi att man bör följa någon metod under reengineeringsarbetet. Genom att arbeta strukturerat kan man undvika att missa bakomliggande information som ligger inbäddad i det gamla systemet.

Vi upplevde att vår metod fungerade bra i fallstudien. Efter att metoden gjorts om för att hantera de problem vi stötte på, tror vi att den nu även kan appliceras på andra liknande projekt. Dock behöver metoden naturligtvis testas ytterligare. En väg att gå för att fortsätta arbetet med metoden kan vara att jämföra den med andra metoder inom området. Ett annat alternativ är att använda den i flera projekt och sedan utvärdera och förändra metoden ytterligare.

En sista slutsats man kan dra är att problemet med legacy systems troligen kommer att finnas kvar under en lång tid framöver. Eftersom dessa system är värdefulla för organisationerna de finns i kommer det troligtvis att även i fortsättningen vara relevant att studera systemarvet och metoder för att handskas med det.

10 Referenser

Allwood, C M. (1991). *Människa-Datorinteraktion – Ett psykologiskt perspektiv*. Lund: Studentlitteratur.

Andersen, E S. (1994). *Systemutveckling – principer, metoder och tekniker* (2:a uppl.). Lund: Studentlitteratur.

Backman, J. (1998). *Rapporter och Uppsatser*. Lund: Studentlitteratur.

Bernus, P, Mertins, K, Schmidt, G (Eds.). (1998) *Handbook on Architectures of Information Systems* Berlin, Tyskland: Springer-Verlag.

Brown, D. (1997). *An introduction to Object-Oriented Analysis: Objects in plain english* (1st ed.). USA: John Wiley & Sons Inc.

Conolly, T, Begg, C, Strachan, A. (1998). *Database Systems, A Practical Approach to Design, Implemenatation, and Management* (2nd ed.). England: Addison-Wesley.

Date, C J. (1995). *An Introduction to Database Systems* (6th ed). Addison-Wesley Publishing Company.

Gustafsson, M R, Johansson, L-Å. (oktober 1994). *Metodik för Reverse Engineering/Reengineering – ett eftersatt område*. Effektiv IT, Systemarvet, Rapport nr 17. Svenska Institutet för Systemutveckling (SISU).

Helander M (ed.). (1988). *Handbook of Human-Computer Interaction*: Software Evaluation Methodologies by Karat, J. Elsevier Science Publishers B. V. (North-Holland).

Laplante, P (ed.). (1996). *Great Papers In Computer Science*: The Entity Relationship Model - Toward A Unified View of Data by Peter Pin-Shan Chen, Minneapolis/St. Paul: West Publishing Company.

Larsson, S. (1986). *Kvalitativ analys – exemplet fenomenografi*. Lund: Studentlitteratur.

Lewis, P. (1994). *Information-Systems Development.* London: Pitman Publishing.

Ljungberg, J, Holm, P. (1995). *Work, Communication, and Information Technology*. IRIS95. Dept. of Informatics, Göteborg University.

Müller, H A, Tilley S R, Wong K. (1993). *Understanding Software Systems Using Reverse Engineering Technology, Perspectives from the Rigi Project*. Department of Computer Science, University of Victoria, Victoria, Canada.

Preece, J (red.). (1994). *Human Computer Interaction*. Addison-Wesley Publishing Company.

Sommerville, I. (1995). *Software Engineering*(5th ed.). Essex, England: Addison-Wesley Longman Limited.

Sundgren, B. (1992). *Databasorienterad systemutveckling, Grundläggande begrepp, Datamodellering, Systemkonstruktion.* Lund: Studentlitteratur.

Tilley, S R, Müller, H A, Whitney, M J, Wong, K. (1993). *Domain-Retargetable Reverse Engineering*. University of Victoria, Dep. of Computer Science, Canada. The 1993 Conference on Software Maintenance. www.sei.cmu.edu/reengineering/pubs.

Tilley, S R. (1995). *Perspectives on Legacy Systems Reengineering* (Draft – version 3). Reengineering Center, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA. www.sei.cmu.edu/reengineering/pubs.

Tilley, S R. (april 1998). *A Reverse-Engineering Environment Framework*. Technical Report, CMU/SEI-98-TR-005, ESC-TR-98-005. Carnegie Mellon University, Software Engineering Institute, Pittsburgh, USA. www.sei.cmu.edu/reengineering/pubs.

11 Appendix 1 – Objektmodell för det nya systemet