Examensarbete I i Informatik
vid institutionen för Informatik vid Göteborgs universitet
VT 1999

Jämförelse av CASE*line* och Objektorienterad analys

Maria Hansson
Jeanette Nilsson

Handledare: Kjell Engberg

Uppsatsen belyser de likheter och skillnader som karakteriserar objektorienterad analys (fortsättningsvis OOA) och CASE*line* i syfte att skapa en vettigt underlag för hur dessa två metoder kan komplettera varandra. Vi har läst diverse litteratur om OOA dels för att se hur de grundläggande begreppen definieras och dels för att komma fram till ett tillvägagångssätt som visar hur OOA kan användas som analysmetod. För att få insikt i hur CASE*line* används har vi intervjuat ett flertal personer på IBS samt studerat ToolBoxen som är ett hjälpmedel i CASE*line*. Metoderna har en liknande struktur där de olika stegen och modellerna följs åt. Vi tycker att OOA är en bra analysmetod på grund av att den är lättförståelig och engagerar användarna. Att använda sig av en egenutvecklad metod såsom IBS CASE*line* ger fördelar i att den anpassas till den egna verksamheten. Vi anser att CASE*line* är en väl genomarbetad metod som de anställda använder och uppskattar. Den ger ett likartat arbetssätt för de anställda att jobba utefter. Tillvägagångssättet och dokumentationen följer samma mönster i de olika projekten. Enligt vår uppfattning har OOA vissa delar som kan komplettera CASE*line* och därmed göra denna metod mera attraktiv och användarvänlig.

INLEDNING

IBS Konsult önskan var att se om OOA kunde förbättra CASE*line*, att finna likheter och skillnader, visa för- och nackdelar med respektive metod. Bland de anställda på IBS finns olika kunskaper om vad OOA innebär. Beskrivningen av begrepp och tillvägagångssätt har därför gjorts på en grundläggande nivå. Vi vill särskilja begreppen och komma fram till en så allmängiltig definition som möjligt.

Vår uppsats är skriven för personer som arbetar med informationsteknologi och vet en del om systemutveckling. Vi vill förmedla grunderna i OOA. I och med att OOA jämförs med en annan metod kan läsaren se för- och nackdelar med att använda denna

Vi valde att skriva om OOA och CASE*line* då Jeanette Nilsson praktiserat på IBS Konsult där hon använt deras analysmetod CASE*line* för att designa ett bibliotekssystem. Jeanette gjorde sedan samma uppgift med hjälp av OOA. Efter detta visade IBS Konsult sitt intresse för att en djupare utredning skulle utföras för att se likheter och skillnader mellan metoderna.

Att utreda OOA tyckte vi verkade mycket intressant då det ingått i kurser vi läst på Informatik. Jämförelsen med CASE*line* gjorde att vi fick studera en metod som används i verkligheten och som är en egenutvecklad metod.

Systemutvecklingen strävar hela tiden framåt, mot nya och bättre lösningar. Utvecklarna vill skapa system som svarar mot kundens önskemål till minsta möjliga kostnad. Om systemen skapas på rätt sätt från början kan minskat underhåll bidra till minskade kostnader. Det är viktigt att kunden känner att systemet är deras och inte utvecklarnas. För att visa på hur systemutvecklingen har förändrats under åren och i vilken riktning den är på väg, beskrivs dess historia här nedan.

TACK TILL...

Vi skulle vilja tacka våra handledare Kjell Engberg på Institutionen vid Informatik och Stefan Milenkovic på IBS Konsult AB för hjälp och vägledning. Vill även framföra ett tack till de anställda på IBS som ställde upp på intervjuer, speciellt till Inger Björner som har varit mycket tålmodig.

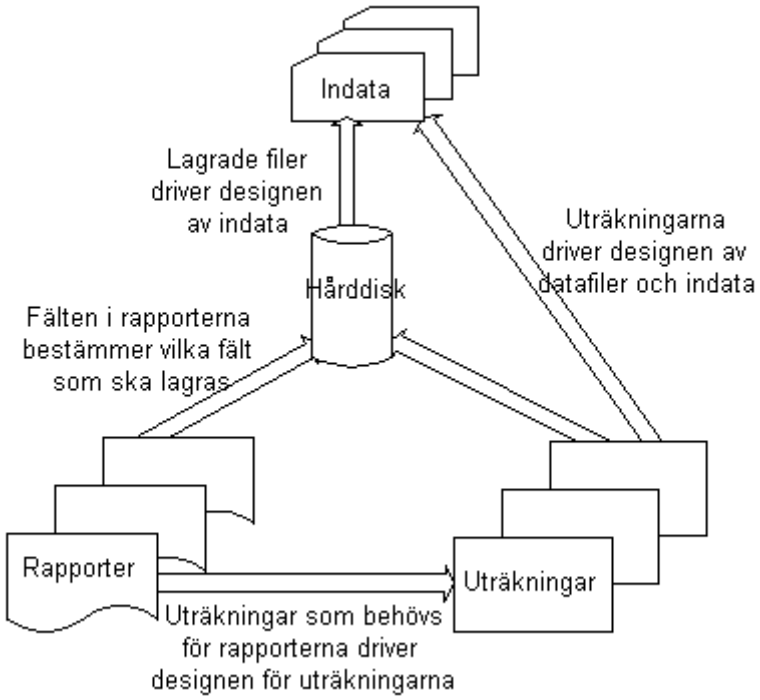
HISTORIA

Systemanalysens historia börjar någon gång på 1950-talet. Begreppen systemanalys och analytiker myntades men betydelsen av dessa var vaga. Det fanns inget strukturerat tillvägagångssätt för systemutvecklingen. Systemutvecklarna fokuserade på att finna en *lösning* på användarnas problem inte på det faktiska *problemet*.

Utdataorienterade analysmetoder

De första strukturerade analysmetoderna var utdataorienterade vilka introducerades i slutet av 1960-talet. Det inledande momentet var att ta reda på vilka rapporter och skärmar som användarna ville att systemet skulle producera. Varken användarna eller systemanalytikerna hade något problem med att se och förstå vad systemet skulle producera för utdata. Efter att ha kommit fram till vad systemet skulle producera arbetade utvecklarna sig bakåt för att så småningom få reda på vad systemet behövde för indata.

Figur 1-1



Överblick över en utdataorienterad metodologi

De utdataorienterade analysmetoderna gav stora fördelar i att systemet blev komplett. Ingen indata saknades till den utdata som skulle producerades eftersom vägen kartlagts från utdata till indata. Metoden hjälpte även till att strukturera och organisera systemutvecklingen. Ett antal steg skulle utföras på ett specifikt sätt. Problemen med

metoden upptäcktes inte förrän systemen hade varit i bruk en tid. Det visade sig att systemen var mycket svåra att underhålla. När företag förändras ändras även kraven på vad datasystemet ska producera. Eftersom systemet var uppbyggt på vad som skulle produceras behövdes väldigt stora förändringar.

Funktionell nedbrytning

I början av 1970-talet hade en standard för funktionell nedbrytning utvecklats. Vid funktionell nedbrytning så bryts affärsprocesser/funktioner ner i mindre och mindre delar tills de är av en storlek och komplexitet så enkla att de är lätta att förstå. På den lägsta nivån kallas de för enhetsfunktioner där varje representerar en ensam operation eller ett steg. I funktionell nedbrytning intresserar inte de anställda som sådana utan endast för den data och behandling av data som de ger upphov till. Den här datan organiseras sedan i funktioner och subfunktioner. Som analysverktyg är funktionell nedbrytning av viss assistans men ändå ganska begränsat. Funktionell nedbrytning har varit mycket användbar vid utveckling av *dataflödesdiagrammet* (DFD). Mer om dataflödesdiagrammen i nästa del.

Processorienterade metoder

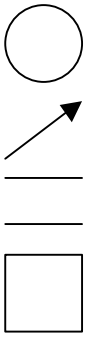
Funktionell nedbrytning är mer strukturerad och bättre än tidigare analysmetoder men kommer inte i närheten av att visa så mycket information som exempelvis den processorienterade modellen dataflödesdiagram gör.

Dataflödesdiagram

Dataflödesdiagram visar affärsprocesser och dataflöden dem emellan, externa entiteter och dataförvaring, exempelvis databaser. Det viktiga är relationerna mellan processerna och inte själva nedbrytningen i mindre processer. Relationen består av den data som skickas från en process till en annan. Vad utvecklaren behöver veta är vilken data som behövs för att utföra en viss process och var den kommer ifrån. Vilken data genererar eller skapar en process och vad händer med den här datan därefter. DFD är fortfarande ett av de bästa sätten att visa de aktiviteter, funktioner och processer som utgör användarnas omvärld anser *Brown*. När dataflödesdiagrammen är gjorda så är steget inte långt till programdesignen.

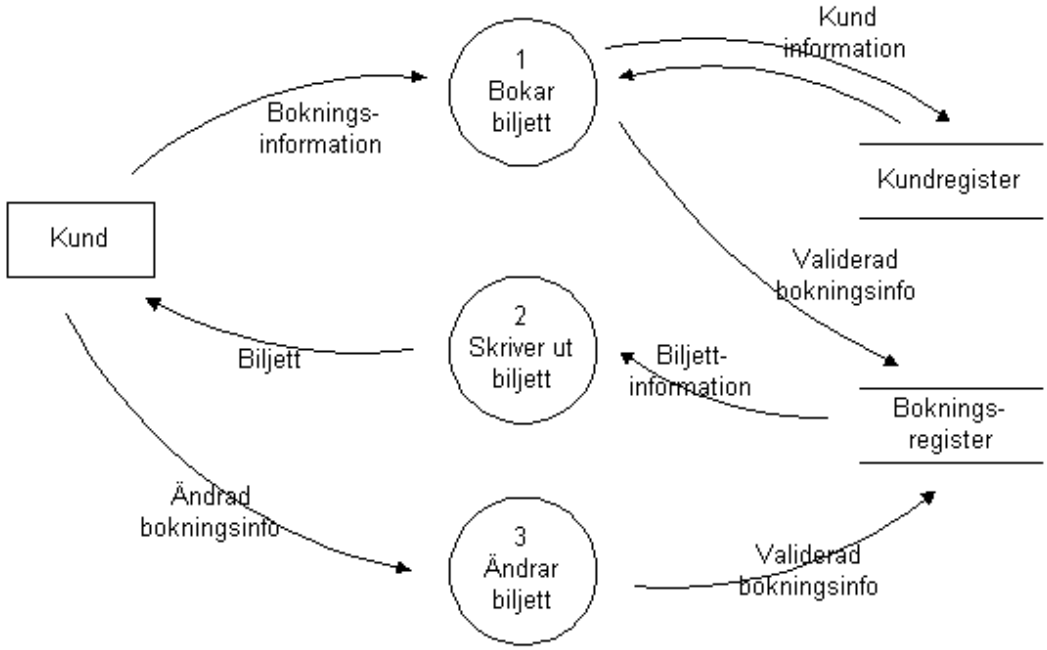
Vid uppritandet av dataflödesdiagram används fyra olika symboler. De representerar process, dataflöden, dataförvaring och externa entiteter. Symbolerna kan ritas på olika sätt. Det sätt som använts i figur 1-2 är *Demarco* och *Yordons* symboler.

Figur 1-2



Dataflödesdiagram kan konstrueras på olika nivåer där detaljerna ökar allteftersom. Det här ger möjligheten att från början göra en översikt över hela projektet vilket underlättar när en uppfattning av användarens omgivning ska skapas. När det här är klart kan utvecklaren gå in i mer detalj i de olika delarna av projektet. I figur 1-3 visas hur ett dataflödesdiagram kan se ut.

Figur 1-3



Ett DFD över ett biljettbokningssystem

Strukturerad systemanalys

Strukturerad systemanalys var populär under 1970- och 80-talet och ersattes sedan med dataorienterade metoder och nu med objekt. För strukturerad systemanalys finns det fyra huvudsteg:

- Fysiskt DFD för det existerande systemet.
- Logiskt DFD för det existerande systemet.
- Logiskt DFD för det nya systemet.
- Fysiskt DFD för det nya systemet.

Dataorienterade analysmetoder

Omkring 1980 kompletterades de tidigare analysmetoderna med metoder inspirerade av databasdesign. Önskan var att datamodellen skulle kunna användas som mall för databasdesignen. Systemet definieras genom de informationsmängder som används i systemet och de relationer som existerar mellan olika informationsmängder. De här metoderna kan kallas dataorienterade.

Entitetsrelationsmodell

Entitetsrelationsmodellen är en dataorienterad modell som använder sig av begrepp såsom entitet, attribut och relation. *Entiteter* är saker som användarna behöver *veta något om*. En dataentitet är något som har separat och distinkt existens i användarnas värld och som är av intresse för användarna på det sättet att de behöver lagra data om den.

Exempel på en entitet är Person som har attributen Personnummer, Namn, Adress, Telefon, Längd och Vikt eller entiteten Bil som har attributen Modell, Märke, Färg, Pris och Registreringsnummer.

För att kunna skilja på de olika exemplaren av en entitet används attributen. Det eller de attribut som särskiljer exemplaren av entiteten från varandra kallas *primärnyckel*. Primärnyckeln gör exemplaret unikt. Primärnyckel hos entiteten Person kan vara Personnummer och hos Bil kan den vara Registreringsnummer.

Slutligen behöver utvecklaren veta hur entiteterna interagerar. Detta beskrivs med ett verb som visar hur entiteterna är relaterade. En relation är interaktionen mellan två entiteter som representeras av ett verb.

En av de stora fördelarna med dataorienterade modeller som entitetsrelationsmodellen är att dataentiteter är stabila över tiden. Eftersom data organiserats kring saker i användarens värld så är de stabila så länge användarna håller sig inom samma bransch. Den här stabiliteten medför kostnadsreducering och att underhållet skjuts längre in i framtiden.

Objektorienterad analysmetoder

I början av 1990-talet kom de första objektorienterade metoderna. De växte fram ur modelleringen med entiteter. Sammanfattningsvis kan sägas att OOA är en förlängning av traditionella utvecklingsmetoder. De objektorienterade metoderna ger genom objektbegreppet ett nytänkande som gör det möjligt att överskrida begränsningar i de tidigare metoderna.

OBJEKTORIENTERAD SYSTEMUTVECKLING

Systemutveckling är en aktivitet som består i att anpassa datasystem till människors behov. Under systemutvecklingens trettioåriga historia har en tradition skapats som innebär att systemutvecklingen delas upp i ett antal bestämda aktiviteter eller faser. En objektorienterad metod organiseras vanligtvis i faser innehållande ett antal arbetssteg som skall utföras. De faser som normalt ingår är analys, design, konstruktion, testning, implementation och underhåll.

Med beteckningen objektorienterad menas att objekt används som grundläggande element i beskrivningen. Objektbegreppet hjälper oss att förstå sambanden mellan datasystemet och dess omgivning, och begreppet kan användas under hela utvecklingsförloppet. Under analysen används objektbegreppet för att fastslå kraven på datasystemet. I designen används det för att beskriva datasystemets interna struktur. Vid design av användargränssnittet kan det användas för att beskriva hur datasystemet framstår utifrån. Slutligen kan det användas som det centrala begreppet i objektorienterad programmering. Det är en stor fördel att samma grundläggande begrepp och tankesätt används i alla faser i systemutvecklingen.

Objektbegreppet skapar med andra ord sammanhang både mentalt i vårt tänkande och materiellt i datasystemets struktur. Det blir lättare att se det som är viktigt och bortse från det oviktiga. Att använda objektbegreppet gör även att användare/kund inte påtvingas en ensidigt teknisk synvinkel.

PROBLEMBESKRIVNING

För att göra en jämförelse mellan OOA och CASE*line* vill vi ta fasta på tre punkter: begrepp, arbetssätt och dokumentationsätt. För att kunna särskilja begreppen inom objektorientering vi vill undersöka terminologin hos olika förespråkare, för att komma fram till en så allmängiltig definition som möjligt. En allmängiltig definition gör det lättare att jämföra begreppen för de båda metoderna.

För att skapa ett sammanhang kring begreppen vill vi finna ett bra och lättförståeligt tillvägagångssätt för att utföra analysen i OOA. Genom att finna ett tillvägagångssätt underlättar det att se hur OOA kan användas och möjliggör en jämförelse med CASE*lines* utförande.

Genom denna problematik kommer vi fram till följande frågor:

- Hur väl är metoderna dokumenterade?
- Var ligger skillnaderna mellan metoderna?

Vi vill diskutera för- och nackdelar med CASE*line* respektive OOA som analysmetoder och även ställa dem mot varandra. Läsaren ska kunna bilda sig en egen uppfattning om vad som är bra och dåligt med de båda metoderna genom uppsatsen. Genom att diskutera för- och nackdelar vill vi se om delar av OOA kan komplettera och förbättra CASE*line* eller om en ensidig lösning är det bästa alternativet.

Genom denna problematik kommer vi fram till följande fråga:

- Vilka effekter förväntas uppnås vid kompletteringar av CASE*line* med OOA-metodens delar?

Avgränsning

Vi har avgränsat vår uppsats genom att endast jämföra två metoder samt endast beskriva analysfasen inom systemutvecklingen. I jämförelsen blir inte Underhållsfrågan aktuell utan endast Behovsanalys, Konceptuell design, Logisk och Fysisk design. Vi har inte intervjuat användarna beträffande deras synpunkter om metoderna utan använt oss av litteratur och IBS anställda.

METOD

För att ta reda på svaren på frågeställningarna i problembeskrivningen har litteratur sökts efter i biblioteket och på nätet. Vi fann fem intressanta böcker inom objektorientering, skrivna av olika författare. Vi använde oss av de här böckerna för att undersöka terminologin och särskilja begreppen inom objektorienteringen. Böckerna beskrev olika tillvägagångssätt i OOA. Vi jämförde de olika tillvägagångssätten och valde en metod som var enkel och väl uppbyggd för att sedan ge exempel på utförandet.

Jeanette Nilsson gick en endagskurs i CASE*line* för att lära sig det grundläggande om metoden. Hon fick sedan praktisera metoden genom att utforma ett internt bibliotekssystem för IBS, vilket gav stor kunskap i användandet av CASE*line*. För att få en djupare förståelse för CASE*line* och ToolBoxen intervjuade vi Inger Björner som är metodansvarig på IBS konsult. Hon berättade hur de är tänkta att användas. Vi intervjuade även andra anställda på IBS för att ta reda på hur de använder sig av metoden i praktiken och hur de ser på den.

I jämförelsen har vi använt oss av Vattenfallsmodellen för att visa analysens olika steg på ett strukturerat sätt i de båda metoderna.

GRUNDLÄGGANDE BEGREPP

Inledningsvis beskrivs de grundläggande begreppen inom OOA för att visa hur de olika begreppen definieras. De flesta analytiker inom OOA definierar begreppen på samma sätt, men vid tillämpning finns det olika sätt att göra det här på.

OBJEKT

Ett objekt är något man vill veta något om som samtidigt är intressant för verksamheten, exempelvis en cykel, bil, kund eller faktura. Det har en unik identitet, ett tillstånd och beteende. Objektet är något man kan ta på, antingen fysiskt (ex bil) eller intellektuellt i någon mening (ex bankkonto).

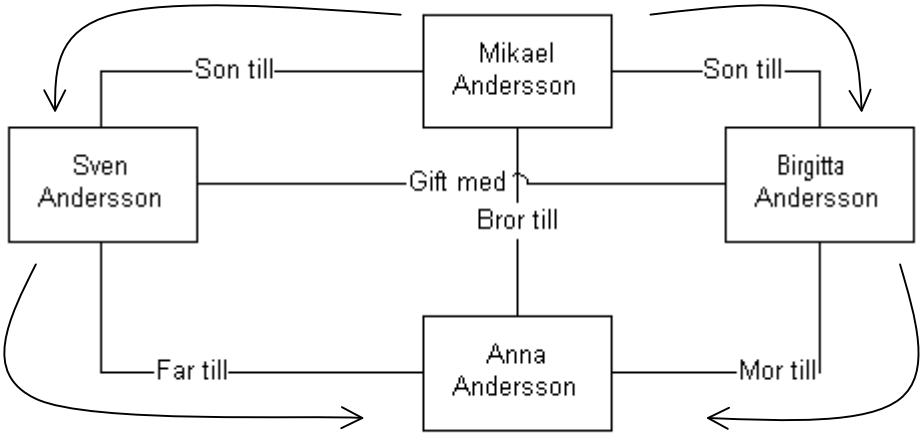
Ett objekt är en instans av en klass. Det vill säga klassbeskrivningen ses som en mall och när värden stoppas in i mallen så skapas en ny instans, ett objekt.

Ett objekt skall kunna existera fristående, men ett ensamt objekt utgör inget system. Flera objekt måste samarbeta vilket görs genom att de skickar meddelande till varandra. Ett meddelande är en begäran om att ett objekt ska utföra en tjänst åt beställaren. Meddelandet är en specifikation av vad som ska utföras, inte hur. Det är mottagaren som avgör hur en begäran hanteras. Ett objekt erbjuder vissa tjänster gentemot omvärlden, det har ett beteende. Beteende är en operation, ett stycke kod, som lagras tillsammans med objektet i databasen.

Ett exempel på objekt är fakturan 44-567-441, objektet har ett tillstånd som bestäms av utställningsdatum, avsändare, mottagare, belopp, såld vara/tjänst samt om den är betald eller ej. En tillståndsförändring av en faktura skulle kunna vara att den ändras från obetald till betald. Beteendet för den här tillståndsförändring är att ändra objektets status från obetald till betald.

Ett annat exempel på ett objekt är Mikael Andersson med personnummer 760120-4807, som är en instans av klassen Man. Mikaels tillstånd beskrivs av alla hans attribut och relationer till övriga objekt i världen. Exempel på attribut är yrke, personnummer, namn, längd, hårfärg, utbildning m m. Exempel på relationer till andra objekt är bror till Anna, son till Birgitta och son till Sven. Alla objekt har en unik identifierare som används för att adressera objektet, Mikaels identifierare skulle kunna vara personnumret. En identifierare kan endast referera till ett och endast ett objekt.

Figur 2-1



Läses i pilarnas riktning.

Attribut är all den information som behöver finnas tillgänglig om objektet. Varje attribut har ett namn, ett värde och eventuellt en typ. Exempel på attribut är färg, storlek, namn och fakturanummer.

Det finns tre olika typer av objekt.

- Entitetsobjekt
- Gränssnittsobjekt
- Kontrollobjekt

Entitetsobjekt

Entitetsobjekt är ”vanliga” objekt som hittas genom att gå igenom användarens verksamhet. De finns i användarens verkliga värld och bär på data som användaren är intresserad av. Det är ofta data som skall lagras bestående.

Gränssnittsobjekt

Gränssnittsobjekt används för att ta hand om kommunikationen mellan systemet och externa entiteter så som användare och andra system, med andra ord all kommunikation med alla externa entiteter som ses i verksamhetsdiagrammet (se sid 23).

Kontrollobjekt

Kontrollobjekt skapas under analysfasen för att ha någonstans att placera beteende (metoder) som inte passar till entitetsobjekt eller gränssnittsobjekt.

KLASSER OCH ARV

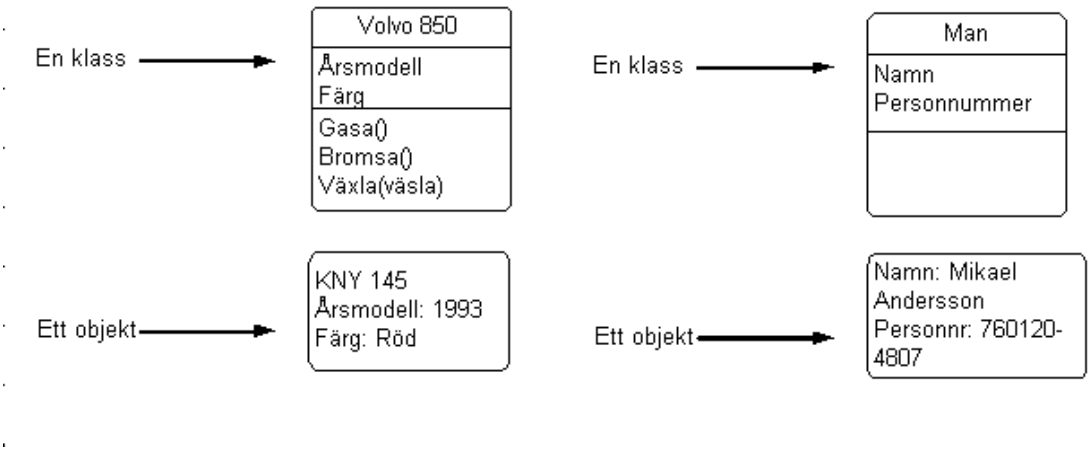
En klass är en mall för hur ett objekt skall skapas. Objekt med liknande attribut (egenskaper), beteende (operationer), relationer till andra objekt och liknande semantik grupperas ihop till en klass.

Vad innebär liknande semantik? Om ett system ska utvecklas för ett lantbruk ska då en ko och en lada ses som samma klass? Nej, det verkar inte så bra. Däremot om det är ett redovisningssystem som ska byggas för lantbruket ska de troligtvis höra till samma klass. I det här fallet ska båda räknas som tillgångar. De ska ha attribut som inköpsdatum, inköpspris m m.

Varje klass har ett namn, attribut och metoder. En klass kan skapa objekt som innehåller de attribut som beskrivs i klassen, samtidigt reserveras minne för objektet. Ett objekt är alltid av exakt en klass. Varje objekt får en egen identitet och en egen uppsättning av attribut men utnyttjar klassens uppsättning av metoder. Klassen kan ses som en egendefinierad typ och objekten som variabler av den egendefinierade typen.

Ett exempel på ett objekt är en Volvo 850 med registreringsnummer KNY 145, som är en instans av klassen Volvo 850. Ytterligare ett exempel är Mikael Andersson som är ett objekt av klassen Man (se figur 2-2).

Figur 2-2



Subklasser

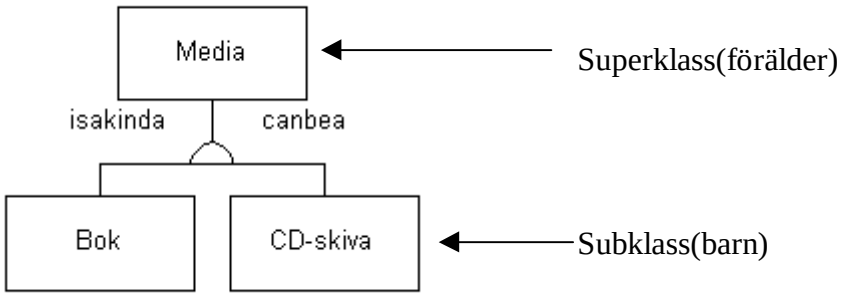
En subklass används exempelvis när det finns flera olika typer av media såsom cd-skiva, böcker, talböcker och tidningar. Olika information behöver sparas för de olika medierna. Samtidigt har de en del gemensamma attribut, exempelvis namn och upplaga. Media bildar då en superklass där cd-skiva, böcker, talböcker och tidningar blir subklasser till media. Det är vanligt att superklassen kallas förälder och subklasserna för barn. Genom att använda subklasser fås följande fördelar:

- Subklassen ärver alla attribut.
- Ärver alla metoder.
- Metoden kan definieras om.
- Andra relationer som superklasser har, ärvs till subklassen (till exempel association och aggregat).

Arv används för att dela upp det specifika för ett objekt på ett naturligt sätt, för att skapa väldefinierade och välstrukturerade klasser. Ett exempel är klassen flygplan där egenskaper fångars upp som är gemensamma för alla typer av flygplan, medan det som specifikt för passagerarplan finns i en speciell klass.

Ett verb beskriver relationen mellan superklass och subklass se figur 2-3. Klassdiagram läses alltid medsols.

Figur 2-3

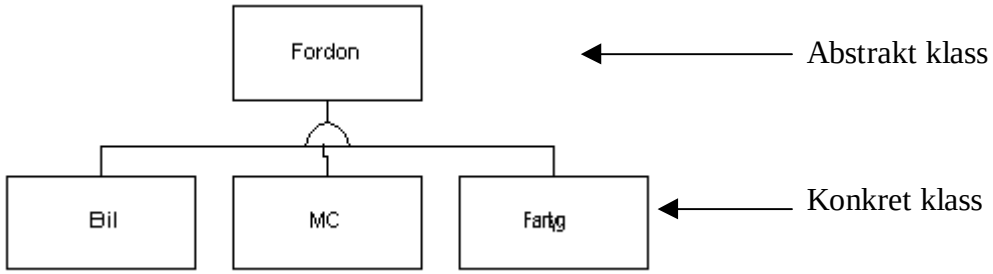


Media → *canbea* → *cd-skiva*
Cd-skiva → *isakinda* → *media*
Media → *canbea* → *bok*
Bok → *isakinda* → *media*

Abstrakt klass

En abstrakt klass är en klass som inte har några egna instanser. Den utgör endast en mall att ärva ifrån. Den används för att beskriva gemensamma egenskaper för andra klasser som ärver från den abstrakta klassen. Motsatsen till en abstrakt klass är en konkret klass. Se figur 2-4.

Figur 2-4

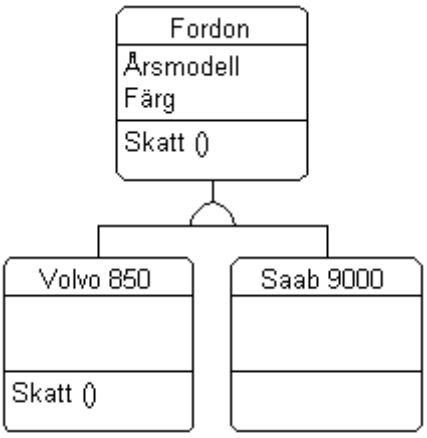


Polymorphism

Vid användning av arv kan det hända att en metod som en subklass ärver inte passar. Då kan metoden definieras om i subklassen fast den fortfarande har samma namn, detta kallas polymorphism. En subklass kan ha olika implementationer av sin superklass metoder. Det litas då helt på att den objektorienterade miljön ser till att rätt metod exekveras beroende på vilken klass anropet sker ifrån.

Ett exempel är klassen Volvo 850 som är en subklass till fordon. Där finns också subklassen Saab 9000. I klassen fordon finns metoden Skatt( ). Då skatten är olika för olika fordon definieras metoden Skatt( ) om för de fordon som det behövs, här i klassen Volvo 850. Volvo får då sin egen Skatt()-metod medan Saab 9000 ärver sin från superklassen Fordon. När en klass anropar en metod söker den först inom den egna klassen. Hittas inte metoden söks metoden uppåt i arvshierarkin.

Figur 2-5

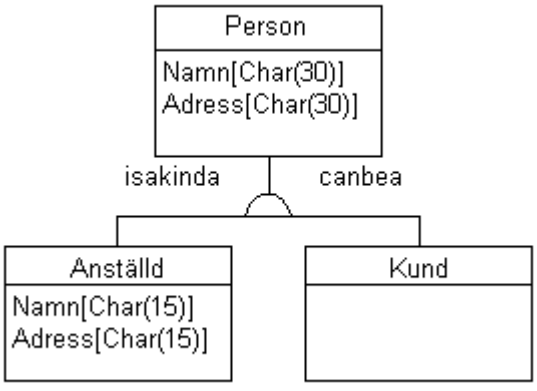


Override

Det kan finnas attribut som vid arv behöver definieras om. En subklass kan ärva attribut exakt som de är eller modifiera dem på något sätt, det här kallas override. Det uppstår om ett attribut har samma namn i båda klasserna och dess domän av tillåtna värden, storlek(längd och bredd) eller datatyp är olika. Det menas att subklassen *override* den svarande superklassen, subklassen ersätter alltså det ärvda attributet med sitt egendefinerade.

Ett exempel är klassen Person som har subklassen Anställd. Klassen Person har attributen Namn och Adress med 30 tecken. I subklassen Anställd behöver attributen bara vara 15 tecken. Då modifieras definitionen av attributen i subklassen så att de endast är 15 tecken. Se figur 2-6.

Figur 2-6

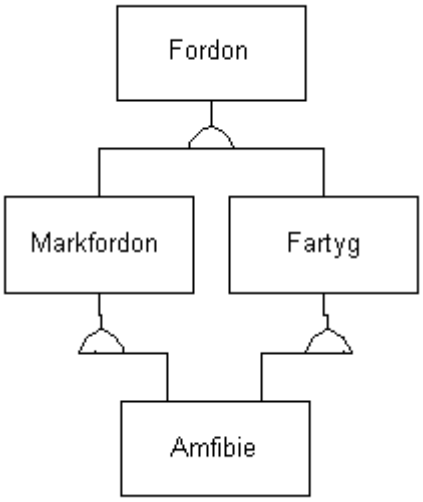


Multipelt arv

Multipelt arv kan ge upphov till namnkonflikter. Som figur 2-7 visar ärver klasserna Markfordon och Fartyg från Fordon. Klassen Amfibie ärver från Markfordon och Fartyg. När en metod existerar i flera av de ärvda klasserna kan en namnkonflikt uppstå och då blir det oklart vilken implementation av metoden som ska användas.

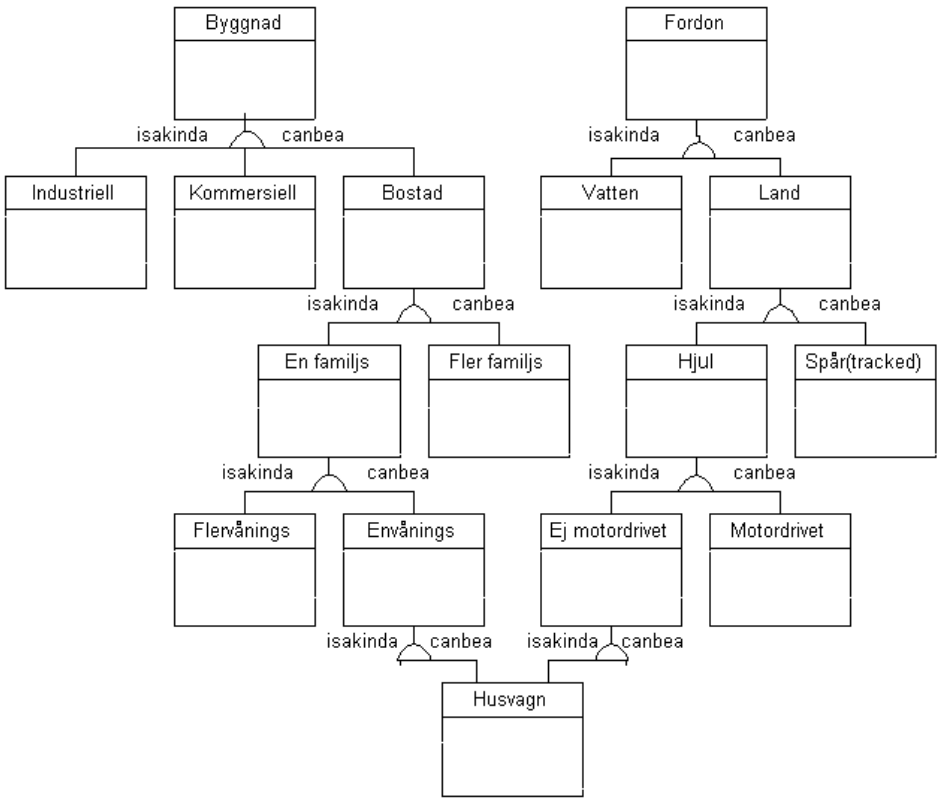
Det finns två olika typer av multipelt arv *mångfaldigt multipelt arv* och *enkelt multipelt arv*.

Figur 2-7



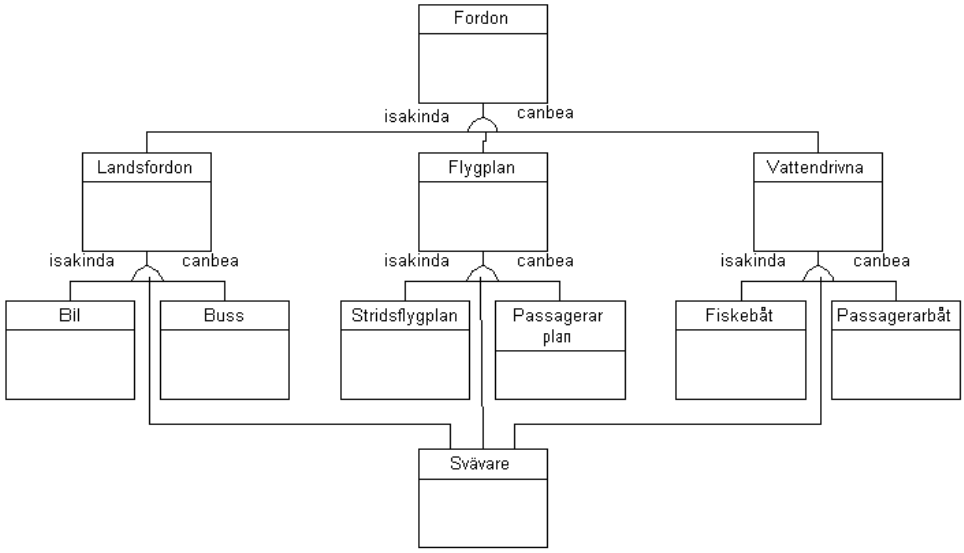
Mångfaldigt Multipelt arv innebär att en klass kan ärva från två eller flera oberoende klasser (se figur 2-8). Subklassen ärver attributen och metoderna från bägge superklasserna vilka inte är sammanlänkade i toppen

Figur 2-8



Figuren visar ett mångfaldigt multipelt arv där Husvagn ärver alla metoder och attribut från klasshierarkierna Byggnad och Fordon.

Figur 2-9



Figuren visar ett enkelt multipelt arv, där klassen Svävare ärver alla metoder och attribut från klasserna Landfordon, Flygplan och Vattendrivna.

Enkelt multipelt arv är när två eller flera grenar av en klasshierarki sluter samman efter att ha delat sig från en och samma klass (se figur 2-9).

RELATIONER

En relation är ett samband som finns mellan klasser och avbildas på objekten av de relaterade klasserna. Alla typer av relationer mellan klasser ger effekter på klassens objekt. En association eller aggregat mellan två klasser betyder att objekten av dessa klasser är sammankopplade. Om en superklass har aggregat eller association ärvs de alltid till subklasserna.

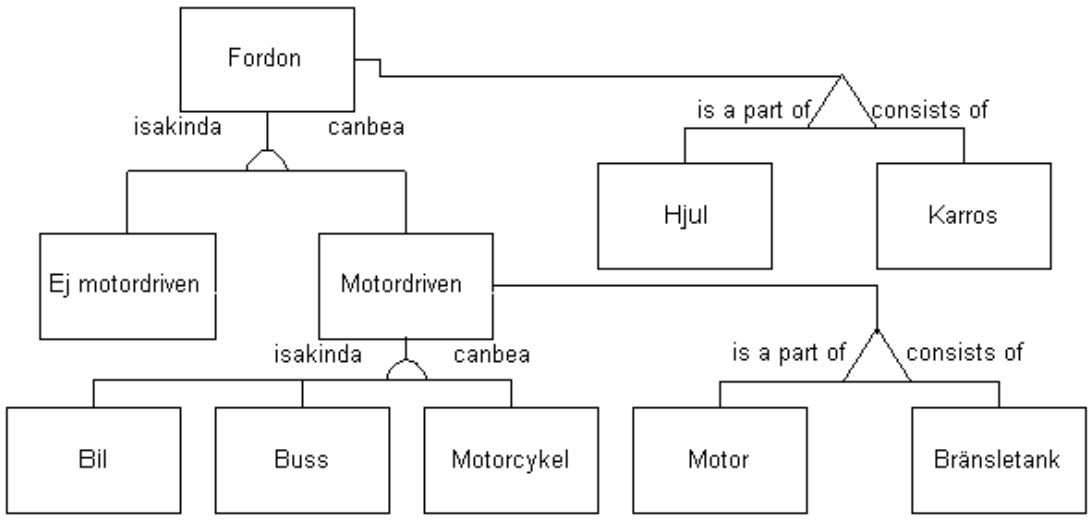
Här beskrivs tre olika typer av relationen mellan klasser/objekt. De är *aggregat*, *association* och *använder*.

Aggregat

Aggregat är en relation mellan objekt som innebär att ett objekt innehåller andra objekt. Exempelvis består en bil av delar som ratt, pedaler, hjul och motor. Var för sig är de delar men tillsammans bildar de en helhet. Här används verben ”Consists of” och ”isapartof”.

Subklass och aggregat är väldigt lika varandra och kan vara svåra att skilja på. För att skilja dem används verbet som beskriver relationen. En motor är inte en bil (isakinda) men den är en del av en bil (isapartof). En bil är en typ av (isakinda) fordon, den är inte en del av (isapartof) ett fordon, den är ett fordon.

Figur 2-10



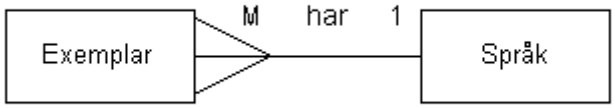
Association

En association mellan två klasser eller objekt innebär att det finns någon slags statisk koppling dem emellan. Det kan ses som om objekten/klasserna känner till varandra. Associationer används dels för att spegla en verklighet och dels för att möjliggöra att objekten kan sända meddelanden till varandra. Association mellan klasser har en kardinalitet som visar hur många instanser som är inblandade. Följande kardinaliteter finns:

- Ett till ett, en försäljning har en kund (1:1).
- Ett till många, en försäljning innehåller flera Produkter (1:M).
- Många till många, en Kund köper av flera Säljare och en Säljare säljer till många Kunder (M:M).

En association kan vara dubbelriktad eller enkelriktad. Om en association är dubbelriktad, känner klasserna som är associerade till varandra, vilket innebär att bägge klasserna kan skicka meddelanden till varandra. En association är oftast dubbelriktad. Om en association är enkelriktad kan detta markeras med en pil som anger riktningen. En association har oftast ett namn som talar om vad den representerar. Se figur 2-11.

Figur 2-11



Detta är en association med kardinaliteten ett till många. Ett språk kan ha många olika exemplar av en bok, men ett exemplar av en bok har inte många språk.

Använder

En användningsrelation avser att en klass använder sig av metoder hos en annan klass, det vill säga att ett beroende existerar mellan klasserna.

INKAPSLING

Inkapsling innebär att attribut och metoder kapslas in i en klass så att det endast är åtkomliga genom metoden i den egna klassen. Objektets beteende kan då garanteras och förutsägas.

OBJEKTORIENTERAD ANALYS

Nu är de grundläggande begreppen i objektorientering genomgångna. Nästa steg är att använda den här kunskapen för att finna objekt, klasser och vilka relationer de skall ha. De steg som visas nedan är till för finna det här enligt Brown (1997). Det är en modell som skall underlätta och hjälpa till vid analysen.

- Kravmodellen
  - Projektbeskrivning
  - Verksamhetsdiagram
  - Användarmodeller
  - Gränssnittsbeskrivningar
- Objektmodell
  - KRB-metoden
    - Kandidatklasser
    - Definiera klasser
    - Etablera relationer
    - Expandera många till många relationer
    - Attribut
    - Normalisera
    - Operationer (beteende)
- Funktionsmodell (DFD)

Analysfasen är den del av systemutvecklingen som syftar till att skapa överblick över kraven på systemet och att fastlägga grundvalen för realiseringen av det. I OOA används objekt för att beskriva fenomen i datasystemets omgivning. Till att börja med ses till det som datasystemet ska handla om. Användarnas krav ska visa vad systemet måste utföra.

KRAVMODELLEN

Syftet med kravmodellen är att dokumentera användarnas mål med systemet på ett sådant sätt att det förstås av användarna och blir användbart för utvecklarna. Den används som utgångspunkt för utveckling av systemet. Eftersom kravmodellen förstås av användarna samtidigt som den beskriver kraven på systemet kan det vara en bra idé att den blir en del av det formella kontraktet mellan båda parter. Det för att det inte ska bli meningsskiljaktigheter i vad som ska ingå och inte.

Kravmodellen kan delas upp i de fyra delarna *projektbeskrivning*, *verksamhetsdiagram*, *användarmodeller* och *gränssnittsbeskrivningar*.

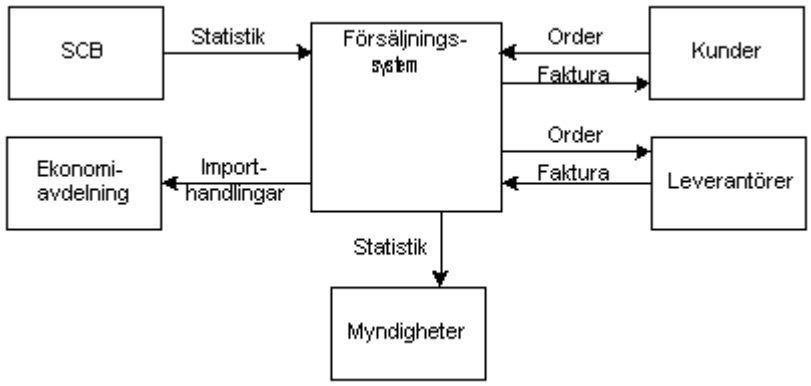
Projektbeskrivning

Detta är en kort textuell beskrivning av vad systemet ska producera. Projektbeskrivningen beskriver generellt vad systemet ska göra, vilka funktioner det ska innehålla och vilka som ska använda det. Beskrivningen ska även innehålla vad som inte ska ingå i systemet.

Verksamhetsdiagram

Verksamhetsdiagrammet visar systemets relationer till externa entiteter, exempelvis yttre system, personer, organisationer, som antingen tar eller ger data från systemet. Det egna systemet visas som en ensam box med pilar till andra boxar som representerar de externa entiteterna. Pilarna från systemet visar den data som skickas från och tas emot från de externa entiteterna. Se figur 3-1.

Figur 3-1



Användarmodeller

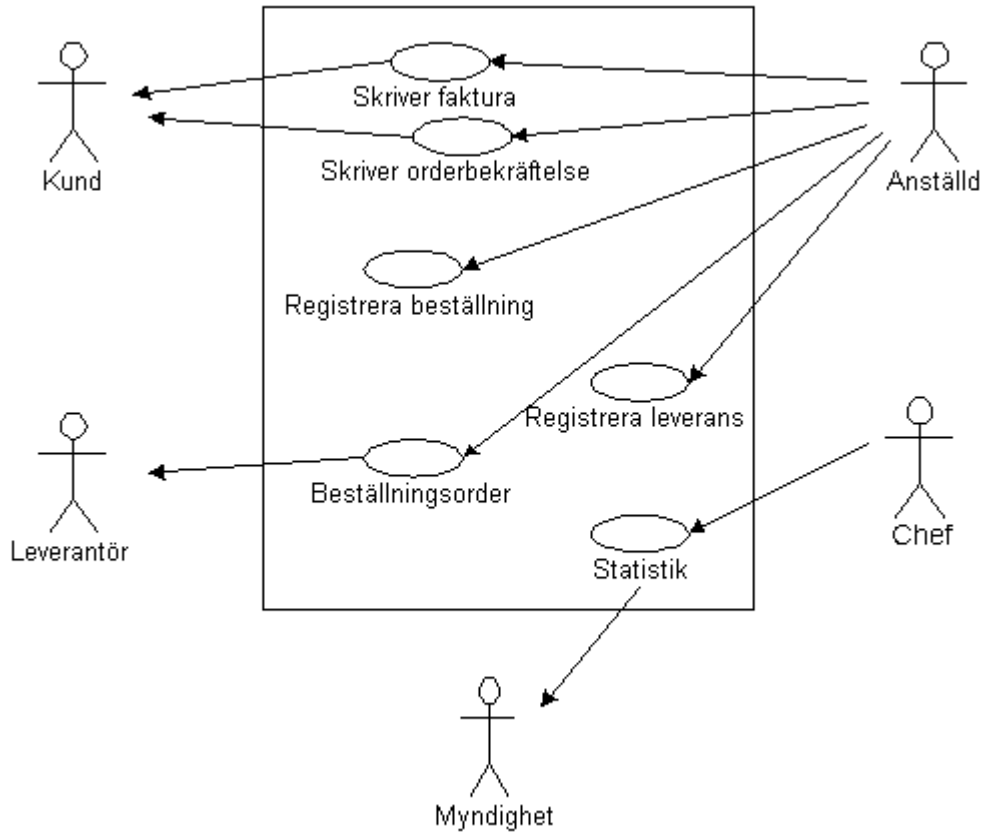
Den här modellen utvecklades av Ivar Jacobson. Det är många utvecklare, som använder sig av vitt skilda utvecklingsmetoder, som finner stora fördelar i att använda Jacobsons användarmodeller. De beskriver de olika situationer där systemet används av någon. De ska alltså visa alla tänkbara användningssätt av systemet och vilka aktörerna är.

För att finna användarscenarion/modeller för systemet ska det tas reda på vad systemet ska utföra. För att finna aktörerna är verksamhetsdiagrammet mycket användbart. Varje extern entitet i verksamhetsdiagrammet blir en aktör. Även fast de interna aktörerna inte är en del av verksamhetsdiagrammet är det vanligt att de upptäcks under skapandet av det. En aktör kan vara inblandad i flera användarscenarios eller bara i ett enda. Ett scenario kan involvera ett flertal aktörer eller bara en. Aktörerna brukar delas in i primära och sekundära aktörer. De primära aktörerna är alla som använder systemet, hämtar eller

lämnar information. De sekundära är de som underhåller systemet och därmed löser användarnas problem med det.

Aktörerna ritas vanligtvis som streckgubbar, där pilar länkar dem samman med ovaler som representerar användarens interaktion med systemet. Ovalerna finns alla inom en rektangulär box som motsvarar själva systemet. Pilarna kan vara enkel- eller dubbelriktade, som visar det huvudsakliga flödet mellan aktör och användarscenario. Är pilen dubbelriktad innebär detta att dataflödena är stora både in och ut ur systemet. Det viktigaste är inte att pilarna är fullständigt korrekta utan att alla aktörer och användarscenarion hittas.

Figur 3-2



Användarmodell över ett försäljningssytem.

Förmodligen hittas inte alla användarscenarion i det här skedet utan de resterande dyker upp under den kvarvarande delen av analys- eller designfasen. När listan av scenarion känns någorlunda komplett är det dags att textuellt beskriva dem. Gå först igenom listan av scenarion tillsammans med användarna och se till att alla scenarion har namn som är så beskrivande som möjligt. Jakobson beskriver ett användarscenario som en serie av händelser som användaren kommer att gå igenom.

Ett av målen med användarmodellerna är att visa utvecklarnas syn på vad användarna vill ha. För att utvecklarna ska veta vad användarna vill ha är det viktigt att båda parter är med vid utformandet av användarmodellerna. Andra mål med användarmodellerna är att ge grunden till att finna objektklasserna och operationerna för varje klass. När klassdiagrammen är gjorda är användarmodellerna mycket användbara för att visa vilket beteende som förväntas av varje klass när användarna interagerar med systemet. Att lägga till beteendet gör att datamodellen övergår i en objektmodell.

Gränssnittsbeskrivning

Gränssnittsbeskrivningarna för systemet delas in i två huvudkategorier, användargränssnitt och gränssnitt mot andra system.

Användargränssnitt
Varje användarmodell kommer att resultera i ett eller flera användargränssnitt. För varje användargränssnitt görs en layout med de fält, knappar, listor och dialogboxar etc som representerar information. Övriga detaljer såsom rullistor behöver inte visas i layouten. Det kan vara till hjälp att göra layouterna redan när användarmodellerna görs. Användarna får en uppfattning om vad som behöver vara med i layouterna när användarscenariona utförs.

Det är viktigt att varje skärmbild har ett skärmbildsnummer och en titel, en identifierare unik för systemet. Ordningen på skärmbilderna är också viktig, det underlättar när det är dags att göra en prototyp.

Gränssnitt mot andra system
De andra systemen som det egna systemet kommer att ha kontakt med är av olika typer. För varje typ behövs en specifikation på hur det egna systemet utbyter information med det.

Om de andra systemen eller databaserna finns inom det egna företaget hänvisas enkelt till redan befintlig dokumentation angående datans tillgänglighet. Finns det ingen dokumentation behöver en undersökning utföras och dokumenteras. Undersökningen behöver inte göras speciellt detaljrik. För system och databaser utanför det egna företaget är det bästa att inkludera publicerad dokumentation om gränssnittet. Om det är väldigt mycket dokumentation räcker det att referera till den.

OBJEKTMODELL

I objektmodellen ska klassdiagram och klasskort där klassen, dess attribut, regler och andra noteringar definieras. Då framgår vilka relationer som finns mellan klasserna. Klasskortens utformning tas inte upp då det är upp till var och en vad som ska vara med och inte.

KRB de sju stegen

Det finns en metod som kallas *KRB de sju stegen* som är utvecklad av Anal **K**apur, Ravi **R**avindra och David **B**rown, därifrån namnet (D. Brown 1997). De sju stegen är ingen formell metod men det är ett sätt att gå tillväga för att komma fram till klasser, relationer m m.

De sju stegen är:
1. Kandidatklasser
2. Definiera klasser
3. Etablera relationer
4. Expandera många till många relationer
5. Attribut
6. Normalisera
7. Operationer(beteende)

Steg ett: Kandidatklasser

Uppgiften är att finna alla entitets-, gränssnitts- och kontrollklasser. Resultatet i det här steget är en lista av substantiv som kan vara namn på klasser som är intressanta för systemet. De här klasserna är kandidatklasser. De kandiderar till att inkluderas i systemet.

Entitetsklasser relaterar till användaren, de är *saker* som användaren behöver spara data om. *Saker* representeras av ett substantiv och är verkliga entiteter, exempelvis person.

De finns fyra olika metoder som Brown förespråkar för att finna entitetsklasser. Det går givetvis att kombinera dessa metoder i olika former. Du kanske redan använder dig av en metod som du föredrar eller vill kombinera ihop med de andra.

Metod ett: Användarintervjuer
Den här metoden kan användas när användarna är mycket upptagna. De intervjuas i små grupper för att få grunderna. Sedan utvecklas definitioner och resten av modellen. Resultatet visas upp för användaren för verifikation och samtidigt ombeds varje enskild användare att skriva en lista av substantiv. Utvecklaren sätter ihop listorna till en och låter den cirkulera. De följande två metoderna som beskrivs kommer fortfarande behövas.

Metod två: Från kravmodellen och annan dokumentation
Den här metoden fungerar bra för sig själv men även i kombination med andra metoder. Metoden går ut på att finna substantiv i den dokumentation som hittills gjorts men även från andra dokument. Dokumentationen som används är exempelvis offertförfrågan, verksamhetsdiagrammet och användarmodellerna.

Metod tre: Brainstorming
Formell brainstorming sker i två eller flera faser. Den första är för generella idéer och den andra är för att utvärdera dessa. I första fasen eftersträvas ett snabbt flöde av idéer. För att uppnå det här är det viktigt att idéerna inte kritiseras eller utvärderas, vilket görs i andra

fasen. Alla ord måste vara substantiv och de ska beskriva de saker användaren/kunden vill **veta något om,** inte vad de vill **veta**.

Metod fyra: Delphi metoden
Delphi är en typ av brainstorming som används när användarna har svårt att träffas. Innan första mötet skickas ett mail till varje deltagare som ombeds skriva en lista av kandidatklasser. Utvecklaren skriver ihop listan och låter den cirkulera mellan deltagarna för att inspirera dem till nya idéer. Efter komplement kan utvecklaren låta listan cirkulera igen.

Gränssnittsklasser är människa-maskin, datakommunikation eller systemstyrt system. Gränssnitt mot användare, ett så kallat OOGUI (objektorienterat grafiskt användargränssnitt) klassbibliotek, köper eller skapar kunden. Där är alla eller nästan alla gränssnittsklasser redan definierade.

Datakommunikationsklasser representerar kommunikationsvägar. Det behövs ett par objekt som visar kommunikationsvägarna. De beskrivs vanligtvis i publicerade kommunikationsprotokoll vilka kan refereras till i dokumentationen.

Om systemet är övervakat eller kontrollerat av ett annat verkligt system är större delen av jobbet klart. De externa systemet kan vara en industriell process, ett trafikljus, en student eller en spelare av ett video spel. Det är oftast en kombination av egenutvecklad och anpassad utrustning tillsammans med delar och komponenter där tillverkaren redan publicerat ett protokoll.

Kontrollklasser visar beteende som är spridda över flera klasser. De flesta kontrollklasser hittas i designfasen. Om någon klass hittas under analysfasen så är det utvecklarens omdöme som avgör om klassen skall vara med i analysfasen eller om det är bäst att vänta tills designfasen.

Kontrollklasser kommer till när en operation anropar efter data och operationer från många andra klasser. Ofta så hör inte en sådan operation ihop med någon annan klass som existerar just nu. Styrklassen kanske bara innehåller den operationen och ett par attribut för att stödja.

Steg två: Definiera klasser
När listan med kandidatklasser är klar är det dags att undersöka varje klass och se hur viktigt den är för projektet.

Genom att följa de tre stegen nedan så får du inte ett definitivt svar på om klassen skall vara med, men utvecklaren kommer att se om klassen betyder något eller är ett attribut.

- Verkliga identifierare
- Definition
- Exempel på attribut och beteende

Om namnet är av någon betydelse så måste instansen ha en identitet. Identiteten måste vara unik så att objekten kan skiljas ifrån varandra. Om det inte finns en verklig identifierare så som id-nummer så måste en unik identifierare skapas. Om en identifierare hittas så är det första indikationen om att detta är en klass som ska vara med, vidare kontroll behövs.

Nu definieras klassen. Det är lämpligt att använda användarens ordval, det gör att utvecklaren börjar förstå användarens värld. Utvecklaren gör en ordlista genom att fråga : ”Vad är en___?”(sätt in klassen som ska definieras). Det är ofta så att olika användare har olika definitioner av ett ord och då måste en gemensam beskrivning tas fram som alla förstår och accepterar, så att inga onödiga missförstånd uppstår på grund av att olika tolkningar.

För att bli ytterligare övertygad om en klass behövs så ska du se till dess attribut och beteende. Ställ frågan: ”Säg något som du behöver veta om___?” vilket ger attributen. Frågan: ”Säg något som ___ kan göra som påverkar systemet?” ger klassens beteende.

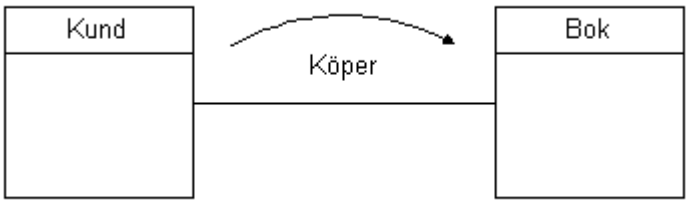
Steg tre: Etablera relationer

Här ställs frågan: ”Vad gör någon av dessa till en av dem?” för att få veta relationerna mellan klasserna. För de funna relationerna gås följande steg igenom:

- Hitta verbet
- Etablera kardinalitet
- Finna alla möjliga relationer

För att hitta verbet ställs frågan: ”Vad gör ___ för ___?” Verbet beskriver relationen mellan instanserna. En relation är en interaktion mellan två instanser av två klasser, som representeras av ett verb som beskriver interaktionen.

Figur 3-4

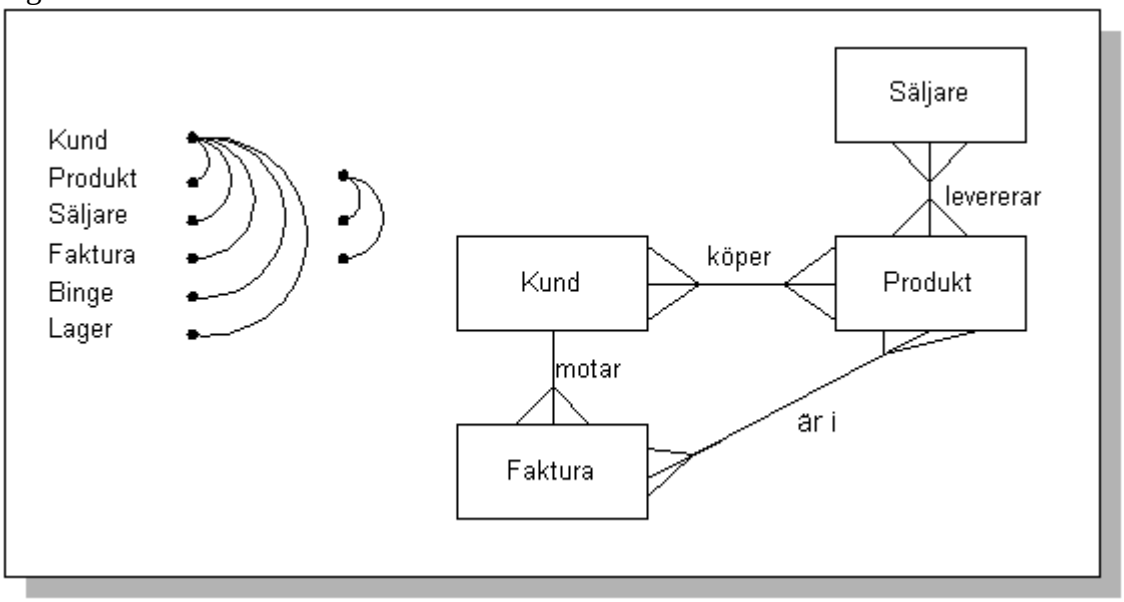


Läses alltid medsols.

Det är viktigt att samma frågor ställs till alla objekt så att inget glöms bort.

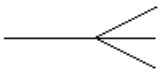
För att se mellan vilka objekt som relationer verkligen förekommer radas alla klasser upp (se figur 3-5). När relationen är kontrollerad prickas de av så att någon inte missas. Prickningen säger inte att det är en relation däremellan utan säger bara att det är kontrollerat. Samtidigt så ritas relationerna upp vid sidan om.

Figur 3-5



Etablera kardinaliteterna mellan klasserna görs genom att ställa frågan om hur många produkter kunden köper och hur många kunder köper produkter, svaret blir att kunden köper många produkter.

Figur 3-6



Symbolen står för ett till många förhållande (1:M). Kråkfötterna representerar många.

Steg fyra: Expandera många-till-många förhållanden

Många till många (M:M) förhållanden är för komplexa för att implementera eller bygga i en databas. De behöver därför expanderas till *ett till många* (1:M) förhållanden.

För att visa på problemet med M:M förhållanden kan två tabeller i en relationsdatabas användas, exempelvis en kundtabell och en produkttabell. M:M förhållandet innebär att en kund kan köpa många produkter men även att en produkt kan köpas av många kunder. För att bygga relationen mellan två tabeller behövs främmande nycklar. Om de två tabellerna har ett M:M förhållanden behövs många främmande nycklar. För att koppla en kund till flera produkter behövs kundnumret (exempel på främmande nyckel) i produktpost. Det är okej, men produkten behöver även vara kopplad till många kunder. För att lösa det här behövs det inte bara ett utan flera kundnummer i produktposten. Det skapar problem som måste lösas genom expansion till 1:M förhållanden. Om det ändå skulle skapas två tabeller med M:M förhållande skulle det snart finnas väldigt många poster med mycket dubbellagrad information i produkttabellen.

För att expandera M:M förhållandet behövs en tredje tabell skapas som kopplar ihop kund- och produkttabellen. Den här tabellen innehåller endast vilka produkter som köpts

av vilka kunder, alltså kundnummer och produktnummer. Varje gång en kund köper en produkt skapas en ny post i den nya tabellen, som kan kallas inköp. I stället för M:M förhållandet är det nu två 1:M förhållanden. De är:

- en kund gör flera inköp, ett inköp har en kund
- ett inköp har flera produkter, en produkt tillhör ett inköp

Den nya tabellen representerar en ny klass, en så kallad associativ klass. En associativ klass skapas mellan två klasser för att expandera ett M:M förhållande till två 1:M förhållanden.

Den nya klassen behöver genomgå steg två och tre för att få identifierare, attribut etc. Attributen som bildade inköpstabellen är kundnummer och produktnummer, men det kan finnas ytterligare attribut som kompletterar den. De nya attributen kan vara från kund- eller produkttabellerna, men det kan även finnas andra som behövs för att beskriva klassen. Som exempel kan nämnas inköpets pris och vilket datum det gjordes.

Vid expansion av M:M förhållande kan det ibland hända att flera av de associativa klasserna representerar samma sak. Är så fallet ska dessa klasser bli en. Det är viktigt att kontrollera att de verkligen representerar samma sak innan de slås samman till en klass.

Steg fem: Attribut

Efter att ha gått igenom relationerna ett antal gånger finns nu ett komplett klassdiagram. Det kan hända att ett par klasser till hittas i attributsteget och normaliseringssteget. Varje gång en ny klass hittas måste den här klassen gå igenom steg två till sju för att definiera klassen och finna relationer och operationer.

I steg fem är det dags för användarna att tillsammans med utvecklarna gå igenom klasserna i klassdiagrammet för att finna attributen. Vid dokumentationen är det bra att ha markerat primärnyckeln för varje klass, vanligtvis görs detta med en asterisk (*). För de attribut som det inte finns en given förståelse för är det bra att definiera dessa så allmängiltigt som möjligt.

Steg sex: Normalisering

Normalisering är den process som garanterar att varje attribut är kopplat till den klass som det verkligen beskriver. I första hand är normalisering en databasteknik men har visat sig göra nytta genom hela systemutvecklingsprocessen.

Det finns många nivåer av normalisering. De tre första normalformerna som beskrivs här är menade att utveckla en objektorienterad databas som lagrar data på ett tillgängligt sätt och som inte påverkas av förändringar. Databasen ska inte bara kunna producera den utdata som den nu klarar av utan även oförutsägbara frågor som behöver läggas till under kommande år.

Som vid uppbrytandet av M:M förhållanden är normaliseringen till för att designen ska bli smidig och att undvika dubbellagring.

Första normalformen
Första normalformen innebär att det endast får finnas ett värde för varje attribut.

Andra normalformen
Andra normalformen innebär att alla icke-nyckel attribut ska vara *funktionellt beroende* av hela nyckeln.

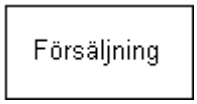
Tredje normalformen
Under utförandet av tredje normalformen söks de icke-nyckel attribut som beror på ett annat icke-nyckel attribut. Det kallas *transitivt funktionellt beroende*. För att eliminera de här beroendena behöver nya klasser skapas. När tredje normalformen har utförts är alla icke-nyckel attribut beroende av nyckeln och endast av nyckeln.

Normalisering är ett verktyg som ska användas med försiktighet. Vid designen av systemet görs en avvägningen mellan hur normaliserad designen är och hur mycket maskintid som går åt. En hårt normaliserad design ger ofta långa svarstider. Designansvarig får väga flexibilitet mot korta svarstider.

Figur 3-8

Ej normaliserat

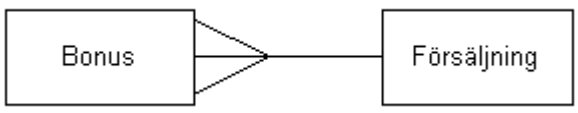
Försäljning	
Kundnummer	25
Katalognummer	255
Försäljnings datum	990302
Kundnamn	Niklas Svensson
Kundadress	Sveavägen 2
Distrikt	Norra
Kvantitet	8
Försäljare	Nisse, Eva, Gunnar
Bonus	5%, 12%, 10%



Första normalformen

Försäljning	
Kundnummer	25
Katalognummer	255
Försäljnings datum	990302
Kundnamn	Niklas Svensson
Kundadress	Sveavägen 2
Distrikt	Norra
Kvantitet	8

Bonus	
Kundnummer	25
Katalognummer	255
Försäljnings datum	990302
Försäljare	Nisse
Bonus	5%

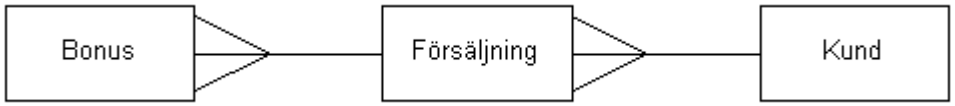


Andra normalformen

Försäljning	
Kundnummer	25
Katalognummer	255
Försäljnings datum	990302
Kvantitet	8

Bonus	
Kundnummer	25
Katalognummer	255
Försäljnings datum	990302
Försäljare	Nisse
Bonus	5%

Kund	
Kundnummer	25
Kundnamn	Niklas Svensson
Kundadress	Sveavägen 2
Distrikt	Norra



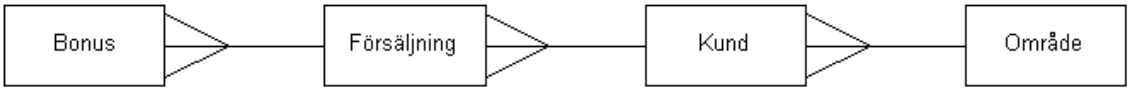
Tredje normalformen

Försäljning	
Kundnummer	25
Katalognummer	255
Försäljnings datum	990302
Kvantitet	8

Kund	
Kundnummer	25
Kundnamn	Niklas Svensson
Kundadress	Sveavägen 2

Distrikt	
Kundadress	Sveavägen 2
Distrikt	Norra

Bonus	
Kundnummer	25
Katalognummer	255
Försäljnings datum	990302
Försäljare	Nisse
Bonus	5%



I den ej normaliserade tabellen har attributen Försäljare och Bonus mer en ett värde. För att elminera de här värdena behövs en ny tabell (klass) som visas i första normalformen (se figur 3-9). Vid utförandet av andra normalformen är attributen Kundnamn, Kundadress och Distrikt inte beroende av hela nyckeln utan endast av Kundnummer. Resultatet blir ytterligare en tabell, som ses i andra normalformen (se figur 3-9). I tredje normalformen söks efter icke-nyckel attribut som är beroende av varandra. Attributet Distrikt är beroende av Kundadress i tabellen Kund. Ytterligare en tabell skapas som ses i tredje normalformen (se figur 3-9).

Steg sju: Operationer (beteende)

Det finns många olika tekniker för att finna objektens operationer/beteenden. Den teknik som beskrivs här ser till objektens tillstånd och de händelser som berör objekten, vilket kan presenteras i ett tillståndsdiagram.

Till att börja med ses det till hur ett objekt förändras över tiden. Vid varje tidpunkt existerar objektet på ett speciellt sätt, detta kallas för objektets *tillstånd*. Värdena på objektets attribut formar tillståndet. Att se till ett *objekts livscykel* är att titta på hur ett

objekt föds, lever och dör. Livscykeln består av de tillstånd som objektet skiftar mellan. Den ordning som tillstånden har följer objektets utveckling från att det skapas tills det så småningom avlägsnas. Objekten förändras från ett tillstånd till ett annat leder till att finna objektens operationer.

För att hålla reda på de tillåtna tillstånden och de förändringar som kan ske för ett objekt eller en klass kan ett *tillståndsdiagram* användas. De håller även reda på händelser som orsakar tillståndsförändringar och händelser som orsakas av de här förändringarna. Tillståndsdiagrammen hjälper till att finna de nödvändiga beteenden för objekten. Det är inte för alla klasser som ett tillståndsdiagram behöver göras. Det är de klasser som har en intressant eller komplex livscykel som det är bra att ha ett tillståndsdiagram för. Oftast är det klasser som tar bort eller lägger till instanser alternativt relationer.

Notationen för tillståndsdiagram visar tillstånden som rektanglar och pilarna mellan rektanglarna visar alla möjliga förändringar av tillstånden (se bilaga 1, sid 19).

När objektens tillstånd förändras är det i huvudsak attributen som visar detta. Det är viktigt att de attribut som är nödvändiga för att se dessa förändringar är definierade. Metoder för att uppdatera attribut måste göras för att de olika tillståndsförändringarna som objekten går igenom ska kommas ihåg. De flesta objekten har ett mönster av tillstånd som de går igenom. Mönstret beror på vilken klass som objektet tillhör, alla objekt av en klass har samma mönster.

Ett objekt lever en tid och förändras då och då. Förändringen är aktiv det vill säga de förändrar sig själva, de har ett beteende. I tillståndsdiagrammet är beteende en händelse som är associerat med varje tillstånd. En händelse inträffar varje gång ett objekt förändrar sitt tillstånd. Varje händelse kommer så småningom översättas i en eller flera metoder så det är viktigt att specificera dem så klart och tydligt som möjligt.

FUNKTIONSMODELL (DFD)

Ett exempel på funktionsmodell är ett dataflödesdiagram. Det förklaras på se sid 6f.

CASE*LINE*

CASE*line* är IBS Konsult egenutvecklade metod. Den är generell och inte beroende av vilka verktyg som kommer att användas. Det började med att IBS AB köpte upp ett företag i Belgien 1988 som utvecklat grundkonceptet till dagens CASE*line*. CASE*line* ToolBoxen utvecklades i ett projekt vid namn NEDS (New European Distibution Service) mellan SKF och IBS Konsult.

CASE*line* används i alla projekt som IBS styr. Beskrivningen blir generell och inte beroende av vilket verktyg som systemet ska utvecklas i.

Inger Björner som är metodansvarig på IBS Konsult har haft stor del i CASE*lines* utveckling. Hon ser ToolBoxen som den viktigaste hjälpmedlet i CASE*line,* då den underlättar för både användare och utvecklare. ToolBoxen innehåller mallar att utgå ifrån. De är färdiga dokument som beskriver alla steg som ingår i systemutveckling med hjälp av CASE*line*. Mallarna gör det lättare att se vad som skall vara med eller inte och det sparar tid. Tanken med ToolBoxen är också att kunna använda de färdiga mallarna som krävs vid analysen för att skapa en heltäckande och lättförstålig dokumentation. Dokumentationen för de olika projekten blir konsekventa.

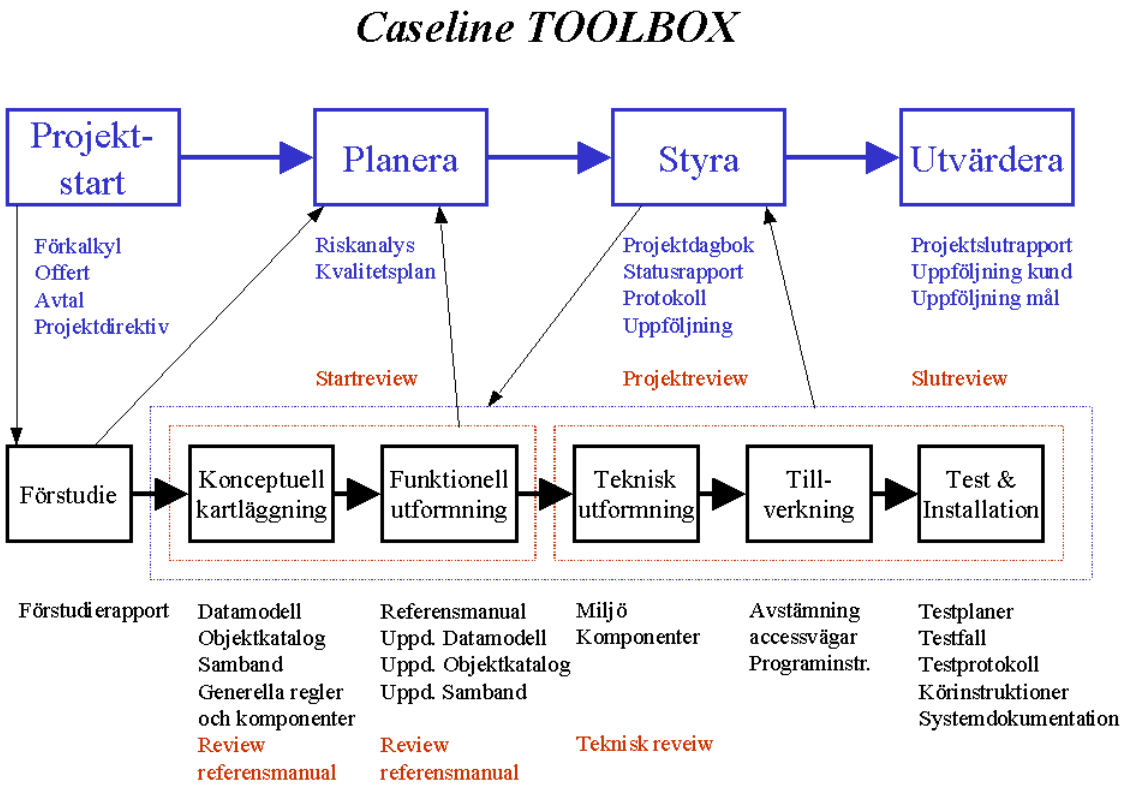
Bilaga 2 är ett exempel på hur CASE*line* används som analysmetod. Titta gärna igenom den innan du börjar läsa följande beskrivning.

SYSTEMANALYS MED CASE*LINE*

CASE*line* är uppdelat i två huvudgrenar *Projektledning* och *Systemutveckling* vilka sker parallellt. Projektledningen utförs av projektledaren. Det innebär ansvar över offerter, avtal och kalkyler. Den ansvarige gör en riskbedömning över projektet där det planerade tidsperspektivet och vad som kan äventyra projektet ingår. I ansvaret ingår också att följa upp hur projektet går och planera om, när det behövs.

I systemutvecklingsfasen ingår steg motsvarande analys, design, konstruktion, testning och implementation. I CASE*line* vävs analys och design ihop vilket gör det svårt att skilja dem åt. De delar i CASE*line* som beskrivs här är förstudie, konceptuell utformning och delar av den funktionella utformningen, se figur 4-1. Användarna medverkar ända fram till den tekniska utformningen.

Figur 4-1



Översiktsbild CASEline. Den övre raden av boxar visar Projektledning och den undre Systemutveckling.

Förstudie

En förstudierapport görs tillsammans med användaren och den består av två grenar med flera steg:

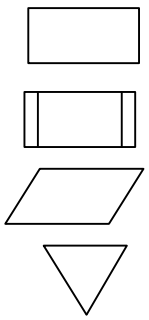
- Dagens situation
  - Verksamhetsbeskrivning
  - Rutingraf
  - Problem och möjligheter

- Föreslagna åtgärder
  - Verksamhetsbeskrivning
  - Rutingraf
  - Datamodell

Dagens situation

I *dagens situation* dokumenterar utvecklaren kundens verksamhet tillsammans med kunden i en v*erksamhetsbeskrivning*. Verksamheten beskrivs som den ser ut i dag med de processer som är inblandade för aktuellt projekt. De kopplingar som det befintliga systemet har (om det finns något tidigare system) till omvärlden i form av meddelanden, transaktioner m m beskrivs i en verksamhetsgraf.

Figur 4-2

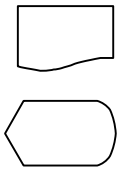


Symbolbeskrivning för verksamhetsgraf.

Nu ritas en *rutingraf* för det befintliga systemet. Den görs i grova drag för att utvecklaren skall förstå hur användarens befintliga system fungerar och kunna se vad som skall förbättras i systemet eller/och i verksamheten. Den första rutingrafen skall spegla vilka processer som finns i systemet. Därefter ritas de rutingrafer som speglar varje process. I de här fallen visar rektanglarna en del av en process och rutingrafen motsvarar processen. Antalet nivåer beror på storleken av systemet. Gäller det enbart ett litet delsystem räcker oftast en nivå.

Flödet visar i vilken ordning de olika processerna/delprocesserna i varje rutingraf sker. För varje process/delprocess ritas det upp vad den kommunicerar med för att kunna utföras. Rutingrafen visar vilken information och var den behöver hämtas, vilka utskrifter som sker, vilka skärmbilder som finns och vilka andra system delprocessen behöver kommunicera med. Till rutingrafen skrivs också en kort textuell beskrivning.

Figur 4-3



Symbolbeskrivning för rutingraf.

Problem och möjligheter används för att specificera alla problem som finns med nuvarande lösning. Vad har kunden för problem med dagens system och/eller rutiner, vad orsakas problemet av, möjliga åtgärder och vad det går att vinna på att genomföra de föreslagna åtgärderna. Det är tillämpbart när uppdragsgivaren inte redan har bestämt en lösning.

Föreslagna åtgärder

Föreslagna åtgärder börjar med att det skrivs en textuell projektbeskrivning. Den beskriver generellt vad kunden önskar att det nya systemet skall göra. Med hjälp av problem/möjligheter ritas en verksamhetsgraf upp samt en rutingraf som visar förslag på hur det nya systemet skall fungera eller ändras.

När kunden godkänt verksamhetsgrafen som ritats för de befintliga systemet, görs en ny v*erksamhetsbeskrivning*. Utvecklaren ritar upp en ny verksamhetsgraf som beskriver det nya/ändrade systemet. Efter att det beskrivits i bild så ska det även beskrivas med text. Texten skrivs punktvis där varje punkt tar upp en händelse som syns i verksamhetsgrafen. Därefter specificeras informationen från och till andra system textuellt (se bilaga 2 sid 6). Om kunden redan vet hur den nya/ändrade verksamheten ser ut så kan verksamhetsgrafen ritas direkt och ingen verksamhetsgraf för den befintliga verksamheten görs. Samma sak gäller för rutingrafen.

Rutingrafen ger en överskådlig blick över det nya systemet på ett tidigt stadium. Utvecklaren får en uppfattning om hur komplext systemet kommer att bli. Utvecklare och användare går tillsammans igenom och skapar rutingrafen.

Nu ritar utvecklaren och användaren/kunden en eller flera rutingrafer (se sid 36) som visar vilka processer som ska finnas i det nya systemet. De olika processer som ska ske inne i systemet ritas upp och utvecklaren arbetar fram ett flöde tillsammans med användaren. Rutingrafen visar vilken information och var den behöver hämtas, vilka utskrifter som sker, vilka skärmbilder som skall finnas och vilka andra system stegen behöver kommunicera med. Till rutingrafen skrivs också en kort textuell beskrivning till varje process. Ett exempel på en rutingraf som visar det nya systemet kan ni se i bilaga 2, sid 8.

En enkel *datamodell* ritas för att finna grunden till de tabeller som skall finnas i databasen och den är också ett mått på systemets komplexitet. Datamodellen förklaras längre fram.

Konceptuell kartläggning

Generella regler

Nu är det dags att beskriva generella regler för dialogprinciper som skall gälla för det specifika projektet. Här skiljs grafiska och teckenbaserade gränssnitt åt. Dessa skiljs åt då IBS använder sig av två olika utvecklingsverktyg där det ena använder grafiska gränssnitt och det andra teckenbaserade.

För teckenbaserade gränssnitt ska det finnas generella regler för vilken standard som gäller för skärmlayouter, vilket språk som ska användas (ett eller flera), hur fältnamnen ska förkortas, fältstandard, hjälptexter och fönsterteknik. Vad skall det finnas för menysystem, vilka snabbkommandon och funktionstangenter är reserverade.

För grafiska gränssnitt ska det finnas generella regler för teckensnitt, språkbruk och förkortningar. Det ska bestämmas färgkodning och hur knappar ska grupperas o s v. Vad skall panelernas storlek vara (upplösning), utformning av menyer och verktygsrader, titeltext, hur menyraderna skall fungera bl a tangentkombinationer och funktionstangenter, verktygsraden skall stämma överens med de funktioner som finns.

När de generella reglerna utformats så skall det finnas bilder på hur knappar, menyer, ikoner, listboxar, radioknappar m m ser ut. Det är en fördel för kunden om det finns exempel på hur skärmbilderna kommer att se ut för det blir enklare att förstå hur systemet kommer att se ut och fungera.

Om det finns färdiga standarder hos en kund så används dem i de flesta fall. De generella reglerna beskrivs först översiktligt och sedan varje del i detalj.

Datamodell

Nu ritas en mer detaljerad datamodell för att finna de tabeller som skall finnas i databasen. Här symboliserar varje rektangel en tabell och diamanterna är associativa tabeller. Kråkfötterna representerar 1:M eller M:1. Här måste M:M förhållanden brytas ut, precis som i OOA (se sid 29f).

Objektkatalog

Här beskrivs varje entitet med namn, ägare, underhåll, beskrivning, volymer samt nycklar och attribut. Fältet specificeras som numeriskt eller textuellt och hur långt det är. Nu avgörs om tabellerna behöver normaliseras och det här görs i så fall i datamodellen.

Det är objektkatalog, datamodell och de generella reglerna som programmerarna använder sig av när de skall sätta upp programmeringsmiljön. De ser direkt vilka tabeller som behövs, hur långa fälten skall vara och vilka generella regler som gäller som standard för hela projektet (se bilaga 2, sid 12).

Datamodellen görs i samband med objektkatalogen för då är alla nycklar och attribut funna och det går att normalisera datamodellen. När det sedan är dags att programmera kan det hända att avnormalisering görs av tabellerna för att öka prestandan. Det är, som nämnts, en avvägningsfråga som görs vid programmeringen.

Meddelandebeskrivningar/Samband

Här beskrivs varje meddelande/transaktion i sin helhet. Att beskriva ett systems alla meddelanden och transaktioner till och från andra system är viktigt då de beskiver omgivande systems krav på det aktuella systemet.

I meddelandebeskrivningar specificeras information om avsändare och mottagare. Alla fält som ingår i meddelandet/transaktionen ska beskrivas med regler för var informationen hämtas och hur den beräknas, valideras och används.

Skillnaden mellan meddelande och transaktion ligger i svaret som mottas från ett annat system. Ett meddelande är alltid en fråga som får ett svar, det vill säga programmet väntar till svar anhållits från det andra programmet/systemet. Meddelande skickar en fråga i taget. En transaktion väntar inte på svar från de andra systemen och skickar flera poster i taget. En transaktion innehåller information som ett annat system behöver för att kunna sköta sin bit av processen. Därefter beskrivs alla fält som ska ingå i transaktionen.

Figur 4-4

NAMN	
Skickas från:	Det system som skickar filen + vilken del av detta system.
Skickas till:	Det system som tar emot filen.
Syfte:	Varför skickas filen.
Villkor:	Vilka villkor måste vara uppfyllda när filen skickas.
Frekvens:	Hur ofta och när skickas filen.
Volymer:	Hur många poster skickas i medeltal i varje omgång.
Ansvarig namn på system:	Namn på ansvariga i aktuellt system
Ansvarig namn på system:	Namn på ansvariga i aktuellt system
Övrigt	Annan information.

Huvud för en transaktion.

Funktionell utformning

Referensmanual

Det utvecklarna och användarna kommit fram till i de tidigare stegen samlas upp i ett dokument, referensmanualen. All information tas inte med utan endast de delar som rör det nya systemet d v s dagens situation tas inte med. De dokument som vanligtvis ingår är verksamhetsgraf, rutingraf, generella regler, datamodell, objektkatalog, samband och procedurflöden. De här delarna ses över och uppdateras undertiden referensmanualen skapas.

Referensmanualen används som ett underlag under resterande delen av utvecklandet. Kund och utvecklare ser det som en specifikation på vad systemet ska innehålla enligt överenskommelse. Används även som ett kontrakt på vad som ingår i det nya systemet, så att utvecklaren kan hänvisa till referensmanualen vid eventuella ändringar som då skulle påverka tidsplan och budget.

I beskrivningen av funktioner/arbetsuppgifter skiljs teckenbaserade- och grafiska gränssnitt åt. De delas in i följande steg:

- Skärmlayout/listlayout
- Skärminnehåll/listinnehåll
- Kontrollregler
- Behandlingsregler

Skärmlayout/listlayout visar hur en tänkt skärmbild eller lista kommer att se ut vid exempelvis Ny post eller Registrering.

Skärminnehåll/listinnehåll visar fälten som är på skärmen och om det är ett inmatning- eller ett visarfält. Sedan följer en liten kort beskrivning av fältet. Se figur 4-5.

Figur 4-5

Fält	I/V	Beskrivning
Antal kopior	I	Anger hur många kopior som skall skrivas ut av varje xxx.(Ett original skrivs alltid ut)
Fält 1	I	Om ett värde anges här skrivs enbart denna post ut, inga andra. För att skriva ut alla hoppa över detta fält.
Starta utskrift	I	Anger om utskriften skall startas eller inte. J = starta utskrift N = gör ingenting

I betyder inmatningsfält, V betyder visarfält, I/V betyder både och.

Kontroll regler är regler som talar om vad som gäller för fälten, se figur 4-6.

Figur 4-6

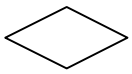
Fält	O/F	Kontroll
Antal kopior	F	Måste vara numeriskt mellan 0 och 9.
Starta utskrift	O	Måste vara J eller N.

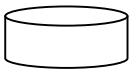
O betyder obligatorisk inmatning, F betyder frivillig inmatning.

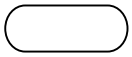
Behandlingsreglerna är en textuell beskrivning över hur en funktion fungerar. Behandlingsregler innefattar regler för varje beslut som funktionen tar samt alla formler som skall användas. De beskriver ett tillvägagångssätt exempelvis en ny post skall läggas till. I behandlingsreglerna finns procedurflöden som ritas vid komplicerade funktioner och vid batchar. De är till för att underlätta beskrivningen av behandlingsreglerna. För varje process som sker i rutingrafen ritas ett procedurflöde. En utförlig procedurbeskrivning skrivs till varje procedurflöde. Procedurflöden ritas för att kunna se till vilka tabeller proceduren hämtar och lämnar data, vilka valmöjligheter som den

erbjuder. De är framförallt till för att kunden/användaren lättare skall kunna förstå behandlingsreglerna.

Figur 4-4


Visar vilka valmöjligheter som finns i den aktuella processen.


Databaser att lagra data i.


Exekveringen slut.

Symbolbeskrivning för procedurflöden samt redan beskrivna symboler(ej lagersymbolen).

Intervjuer med IBS anställda

Vi intervjuade ett antal personer på IBS Konsult, med olika befattningar. Det här skedde under diskussionsformer där de intervjuade pratade fritt. Diskussionen utgick från några frågeställningar om för- och nackdelar med CASE*line* och ToolBoxen, hur de använder sig av dem och om de är lättanvända. Vilka modeller är bra respektive dåliga? Behöver de omstruktureras eller kompletteras? Hur uppfattas strukturen av både CASE*line* och ToolBoxen?

De flesta tycker att konceptet med en metod som ger riktlinjer är väldigt bra. Det finns en struktur att följa med specifika moment vilket gör det enkelt. Utseendet på dokumentationen blir likartat. Stora delar av dokumentationen från analysen kan användas som användarmanual.

De olika delarna i CASE*line* kan användas i en följd eller var för sig. I många projekt används inte alla modeller, utan de som behövs plockas ut. Det är därför viktigt att de är separata så att de är lätta att använda var för sig, vilket de flesta tycker att de är.

De anställda anser att projektbeskrivning, verksamhetsgraf och rutingraf är bra och lättförståliga för både kund och utvecklare. De ger en överblick över kundens verksamhet.

Procedurflödena som är en del av behandlingsreglerna råder det delade meningar om. En del tycker att de är väldigt viktiga att ha vid programmeringen, om de är bra utförda. Procedurflödena skall vara väl genomarbetade för att inte orsaka problem med presstanda vid programmeringen. De utvecklades från början för stordator och batchsystem. En del tycker att de behöver bytas ut för att flödet i stordator och batch system går från A till B vilket skiljer sig från idag där systemen är händelsestyrda. De som tycker procedurflödena är bra ser varje händelse som ett flöde. Många tycker att det är texterna till procedurflödena som ger mest stöd vid programmeringen. Procedurflödena med tillhörande texter är beskrivna på ett sådant sätt att programmerare har en god grund att

utgå ifrån, de ger också en uppfattning om hur komplex en funktion är. Idéer och andra lösningar utvecklas under konstruktionsfasen.

Referensmanualen fyller sin funktion som underlag vid utvecklandet av ett system och som kontrakt i vad som ingår. Uppdateringen av referensmanualen är väldigt dålig. Skulle uppdatering ske kontinuerligt hade fel upptäckts tidigare och kunden skulle kunna följa förändringar. Det kan vara väldigt svårt att hinna med uppdatering av referensmanualen då utvecklaren oftast har tidspress och då är det viktigare att systemet blir klart än att hålla den uppdaterad.

ToolBoxen sparar tid vid utvecklingen för att den innehåller färdiga mallar att utgå ifrån. De anställda som intervjuats använder ToolBoxen för att ta ut dokumentmallar som de vet finns där. För att se vilka dokument som kan ingå i ett projekt eller lära sig strukturen i CASE*line* används framför allt gamla projekt och inte ToolBoxen. Över lag tycker de anställda som intervjuats att det är bra med en ToolBox som visar utformning av olika dokument för projekt.

JÄMFÖRELSE MELLAN OOA OCH CASELINE

Efter genomgång av tillvägagångssätt för de båda metoderna OOA och CASE*line* vill vi nu jämföra dem. Steg för steg visas likheter och skillnader i begrepp, arbetssätt och dokumentationssätt. För att kunna göra en jämförelse strukturerar vi upp den enligt vattenfallsmodellen där behovsanalys, konceptuell design, logisk och fysisk design ingår.

Behovsanalys i OOA och metoden CASE*line*

Likheter och skillnader i begrepp

De begrepp som tas fasta på i behovsanalysen för OOA är *datasystemet*, de *externa entiteterna* och den information som löper mellan dem. En definition av *aktörer* och hur de använder sig av systemet (*användarscenarion*, se sid 44) görs.

I CASE*line* ses också till *datasystemet* och de *externa entiteterna*. *Processerna* inuti systemet tas även upp och definieras vilket inte görs i lika stor utsträckning i OOA. De användarscenarion som aktörerna utför kan liknas vid en process. Skillnaden ligger i att scenariot inte beskrivs på ett lika tekniskt sätt utan stannar vid att beskriva vad användaren gör i systemet. För aktörerna i OOA finns ingen motsvarighet i CASE*line* (Se sid 44).

Likheter och skillnader i arbetssätt

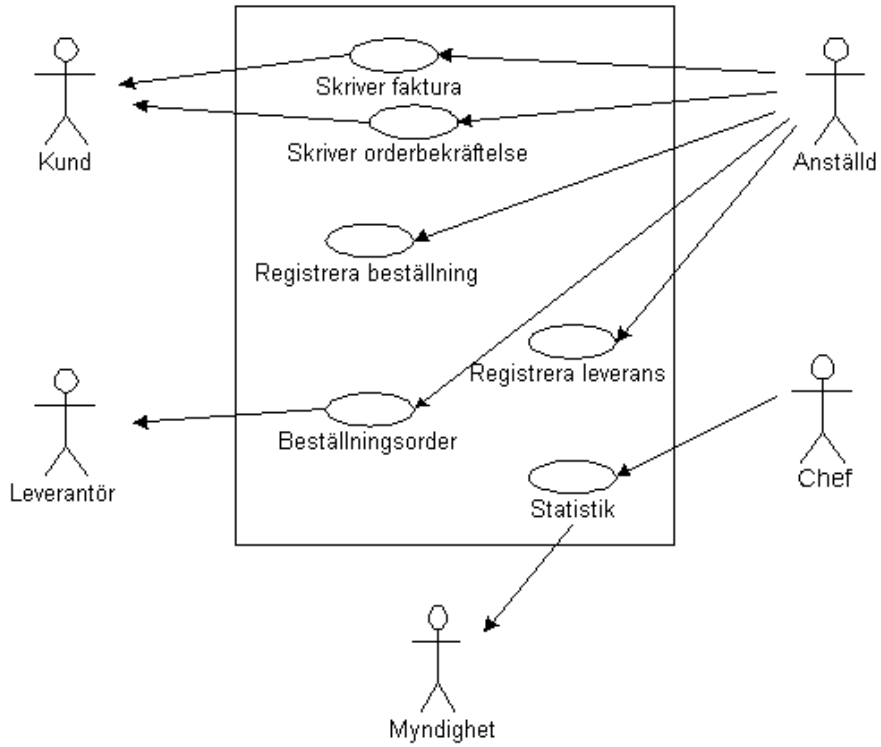
De begrepp som diskuterats används i olika arbetssätt i respektive metod. I OOA används kravmodellen där det ingår fyra steg; projektbeskrivning, verksamhetsdiagram, användarmodeller och gränssnittsbeskrivningar (se sid 22ff). Arbetssättet i CASE*line* består i att utföra en förstudie. I förstudien ingår verksamhetsbeskrivning, verksamhets-graf och rutingraf. Båda metoderna utgår från verksamheten och de informationsmässiga förhållandena till informationssystemen. Skillnaden ligger i att OOA refererar till ansvarsområden för aktörer medan CASE*line* istället refererar till processer.

Gränssnittsbeskrivningar används i de båda metoderna men i olika utformning och omfattning. De beskrivningar som finns i OOA är som sagt sista steget i kravmodellen, för CASE*line* beskrivs gränssnitten inte förrän i den konceptuella kartläggningen. Jämförelsen av gränssnitten görs ändå här i behovsanalysen.

Likheter och skillnader i dokumentationssätt

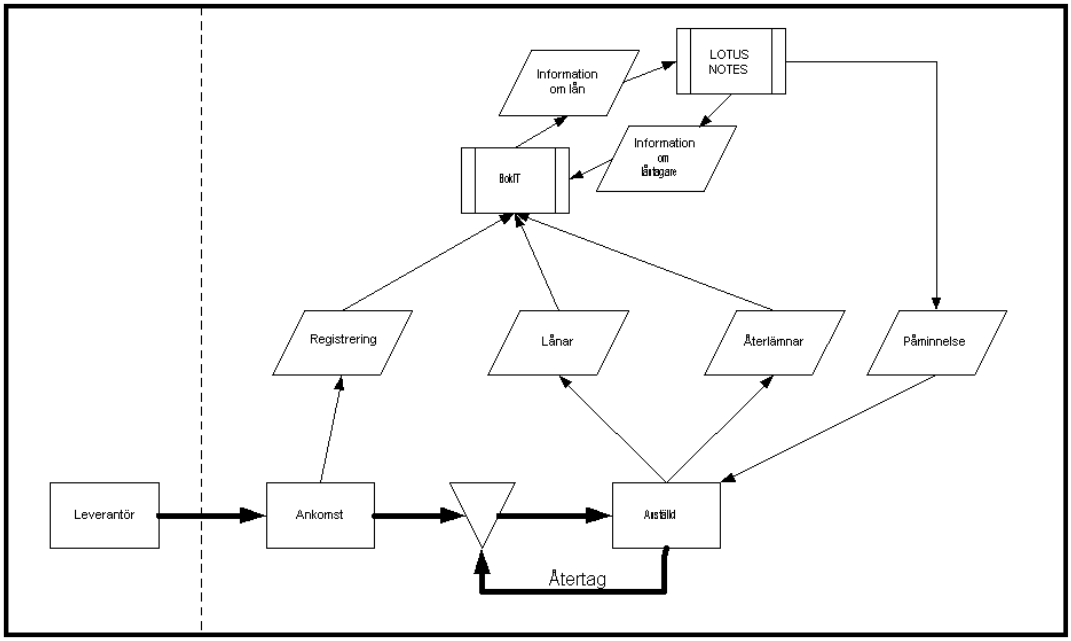
Projektbeskrivningen och verksamhetsdiagrammet i OOA påminner mycket om CASE*lines* projektbeskrivning och verksamhetsbeskrivning. Innebörden är den samma i de båda metoderna men figurerna skiljer sig åt. Den verksamhetsgraf som används i CASE*line* visar det fysiska flödet. Ett exempel på det fysiska flödet ses i verksamhetsgrafen i bilaga 2, sid 6. Där följs medias väg och ser var i flödet som de olika processerna sker. I OOAs verksamhetsdiagram visas inte det fysiska flödet utan endast interaktionen mellan de externa entiteterna och systemet.

Figur 5-1



En användarmodell av ett försäljningssystem.

Figur 5-2



Verksamhetsgraf över ett biblioteksystem.

I OOAs användarmodeller visas hur aktörer använder systemet, vilka processer och vilken information de har tillgång till. I CASE*line* används en rutingraf (bilaga 2, sid 8) som visar processerna i ett flöde. Rutingrafen ser inte till aktörerna, vad de gör och vilken information de har tillgång till. Istället visar den var informationen lagras i grova drag och vilka rapporter och skärmbilder som bör finnas.

I OOA används användarmodellerna för att se vilka gränssnitt som behöver vara med. Det görs ett första utkast på hur gränssnitten kommer att se ut, både mot användare och system. CASE*line* visar exempel på skärmdumpar liknande de i OOA. Skillnaden är att CASE*line* använder sig av generella regler som skrivs väldigt utförligt. Reglerna beskriver de standarder som ska användas för ett projekt såsom funktionstangenter, hjälptexter och teckensnitt. CASE*line* skiljer på generella regler för teckenbaserade och grafiska gränssnitt.

De dokument som erhålls av modellering och uppritande av grafer dokumenteras. I CASE*line* sätts de dokument som produceras ihop till en *referensmanual*.

Konceptuell design i OOA och metoden CASE*line*

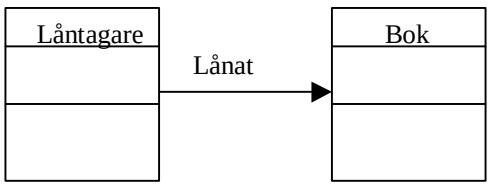
Likheter och skillnader i begrepp

I den konceptuella designen används i OOA begreppen objekt, attribut och relationer objekten emellan. Det finns diverse olika relationer; association, aggregering och arv (se sid 14ff, 20ff). I CASE*line* används entiteter, attribut och relationen association.

Attributen i de båda metoderna har samma funktion, att beskriva tillståndet för objektet/entiteten. Objekt och entitet är olika på en punkt, beteendet. Grunden i den objektorienterade läran är att information om beteenden slås samman i objekt i stället för att som i traditionell modellering behandlas separat i till exempel datamodeller och funktionsmodeller. CASE*lines* entiteter har inte information om beteendet kopplat till sig.

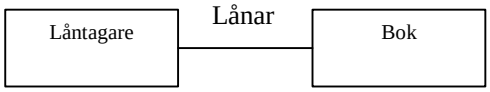
Skillnaden i relationer mellan de båda metoderna ses då OOA har ett antal relationer som beskriver olika slags förhållanden mellan objekten, såsom består av (consist of), är en del av (isapartof). Arvet är en egenskap som visar att subklasser ärver egenskaper från superklassen. Denna egenskap kan sedan på ett lätt sätt implementeras i en objektorienterad miljö. CASE*line* använder sig endast av en relation som beskriver att entiteterna interagerar med varandra.

Figur 5-3



I objektorientering definieras varje objekt i termer av:
Identitet, struktur och beteende

Figur 5-4



Vid entitetsrelationer organiseras information. Entitetens beteende behandlas separat och tillhör inte den konceptuella designen.

Som beskrevs i figur 5-3 så definieras varje objekt med identitet, struktur och beteende. Begreppet *identitet* klargör identifieringsfrågor för objekten såsom att de är unika och odelbara. För alla objekt gäller att de är unika medan endast sammansatta objekt ska vara odelbara. Sammansatta objekt kallas i viss litteratur associativa objekt (ex Brown, 1997).

Begreppet *struktur* refererar till såväl attribut som polymorphism, aggregerings- och arvsförhållande.

Beteendet refererar till operationer/metoder som tillhör en viss objektklass och som klargör under vilka förhållanden objekten ska fungera. En sak som skiljer de olika objektorienterade synsätten är hur djupt de går i specifikationen av ett objekt. Brown (1997) tar upp en del tekniska termer som specificerar beteendet såsom *override*.

Likheter och skillnader i arbetssätt

För att finna klasser av objekt och relationerna i OOA används *KRB-metoden*. Resultaten visas i klassdiagram och klasskort. I den *konceptuella kartläggningen* i CASE*line* bestäms och beskrivs generella regler, datamodell och objektkatalog. Det finns ingen specifik metod, motsvarande KRB, för att ta fram datamodellen och objektkatalogen i CASE*line*.

Likheter och skillnader i dokumentationssätt

I dokumentationen för OOA ingår klassdiagram och klasskort vilka kan jämföras med datamodell och objektkatalog i CASE*line*. Klassdiagram/datamodell visar objekten/entiteterna med de relationer som råder mellan dem.

Klassdiagram/datamodell utförs för att komma fram till de tabeller som skall finnas i databasen. De beskrivs sedan utförligt i klasskorten/objektkatalogen. I objektkatalogen visas även vilken fälttyp attributen ska ha. Klassdiagrammet visar klasserna och relationerna emellan dem. Därefter följer klasskorten som är en textuell beskrivning av klasserna.

I KRB steg sju i OOA visas objektets beteende genom de olika tillstånd som objektet genomgår under sin livscykel. CASE*line* använder sig av entiteter som inte har ett beteende knutet till sig. De använder sig av procedurflöden där systemets procedurer visas. Se bilaga 2, sid 18, där det visas vad som sker i processen Preliminär registrera.

Datamodell och objektkatalog läggs in i CASE*lines* referensmanual.

Logisk och fysisk design i OOA och metoden CASE*line*

Likheter och skillnader i begrepp

I den logiska/fysiska designen så är begreppet som används tabeller. Först specificeras de och sedan implementeras de i en databas. Detta gäller för både OOA och metoden CASE*line*.

Figur 5-5

Bok

Id-exemplar	Titel	Antal sidor

Bok

Id-exemplar	Titel	Antal sidor

I transformationen i en relationsmiljö skulle de båda metoderna resultera i samma logiska struktur som i figur 5-5. I en objektorienterad miljö skulle realiseringen omfatta alla aspekter som finns dokumenterade i klasskorten. I en entitetsrelationsmiljö skulle det behöva utredas vilka program som skall kopplas till tabellerna.

Likheter och skillnader i arbetssätt

I den logiska designen normaliseras tabellerna från klasskorten/objektkatalogen. I OOA ingår detta moment i KRB-metoden. I CASE*line* finns det inte ett uttalat tillvägagångssätt för att utföra normaliseringen.

I den fysiska designen implementeras tabellerna i en databas i de båda metoderna. Implementationen av tabellerna ingår inte i denna utredning men en väl utformad analys underlättar utförandet av den. I klasskorten ses de tabeller som ska användas i databasen.

SLUTSATSER

I det här stycket diskuteras de resultat vi kommit fram till genom den kunskap som inhämtats vid utförandet av denna uppsats. Diskussionen förs kring de frågeställningar som tas upp i problembeskrivningen.

Det finns grundläggande begrepp inom objektorienteringen som utgör stommen. Vid studerandet av dessa begrepp kom vi fram till att olika författare och förespråkare inom objektorienteringen använder likvärdiga definitioner.

- Hur väl är metoderna dokumenterade?

Vi tycker det finns bra och begriplig dokumentation av båda metoderna. I objektorienteringen anser vi att mycket av den dokumentation som skapas i och med de olika modellerna både ger en överskådlig bild och gör det lätt för användarna att förstå.

Dokumentationen i referensmanualen är utförlig och den används som specifikation på vad systemet ska innehålla. Den ger generellt en mer teknisk dokumentation än vad objektorientering ger. Det här kan tänkas vara då objektorienteringen medför ett helt tankesätt som är verklighetsnära.

- Var ligger skillnaderna mellan metoderna?

Det finns både skillnader och likheter mellan de båda metoderna vilket jämförelsen visar (se sid 43ff).

Vi tycker att OOA är en bra analysmetod för den är lättförståelig och engagerar användarna. Den har en struktur där det ena momentet följer på det andra. Genom att låta användarna medverka så mycket som möjligt utvecklas systemet närmare användarens önskemål och de får en större förståelse för hela analysfasen. Användarna känner sig delaktiga i projektet och får en känsla av att det är deras system, inte utvecklarnas.

Att använda sig av en egenutvecklad metod såsom IBS CASE*line* ger fördelar i att den anpassas till den egna verksamheten. Vi anser att CASE*line* är en väl genomarbetad metod som de anställda använder och uppskattar. Den ger ett likartat arbetssätt att jobba utefter. Tillvägagångssättet och dokumentationen följer samma mönster i de olika projekten.

I stora drag påminner CASE*line* om OOA. I jämförelsen ser vi att stegen följer varandra i de båda metoderna. Många av modellerna har liknande innebörd men skiljer sig åt i utformning. I OOAs kravmodell finns verksamhetsdiagram och i CASE*line* verksamhetsgraf. Verksamhetsgrafen visar det fysiska flödet vilket kan vara svårt att finna i alla verksamheter. Därför kan det vara enklare att använda sig av OOAs verksamhetsdiagram som bara visar interaktionen mellan systemet och de externa entiteterna. Båda är överskådliga men verksamhetsgrafen kan lätt bli rörig och svår att

följa när den innehåller för mycket information. Upplägget som CASE*line* följer på den textuella beskrivningen av verksamhetsgrafen är bra. Informationen spaltas upp på ett överskådligt sätt, som blir informationsrik.

Rutingraferna i CASE*line* visar mer information än användarmodellerna. Det gör att rutingrafen blir rörig vid stora informationsmängder. För användare/kunder med datakunskap och stor insikt i sin egen verksamhet är rutingrafen det bättre alternativet. Är användarna oerfarna är det lättare med användarmodellerna, antingen för sig självt eller som ett komplement till rutingrafen. Användarmodellerna blir lättförståeliga då aktörerna kan se och följa sin egen roll. Det är väldigt viktigt att aktörer från alla områden medverkar vid utvecklandet så att alla delar i systemet representeras av en användare. I rutingrafen ritas flödet upp mellan processerna och de kan ibland vara svåra att finna, då kan det vara enklare att ta till användarmodeller.

Att bestämma generella regler för ett projekt (se sid 37f), ser vi stora fördelar i. Exempelvis kan generella regler vara standarder för dialogboxar, färgkodning och informationshantering. Standarden ger gränssnitt med liknande utseende där kunden känner igen sig. Om kunden redan har en standard som de är nöjda med så kan det vara positivt att använda sig av den.

I CASE*line* används en datamodell för att hitta tabeller till relationsdatabasen. I OOA används klassdiagram. För att kunna använda sig av OOAs klassdiagram fullt ut måste databasen vara objektorienterad. I klassdiagrammet visas olika typer av relationer som beskrivs på sidan 20f. Fördelen med klassdiagrammet är att modellen visar var arv, association, använder och aggregat ska användas. Det här framgår inte i CASE*lines* datamodell trots att de ibland används vid programmering.

De sju stegen, KRB gör det lättare att använda sig av OOA. Tillvägagångssättet för att finna klasser och relationer är heltäckande. Vi anser att stegen är väl genomarbetade och rekommenderar att den används.

- Vilka effekter förväntas uppnås vid kompletteringar av CASE*line* med OOA-metodens delar?

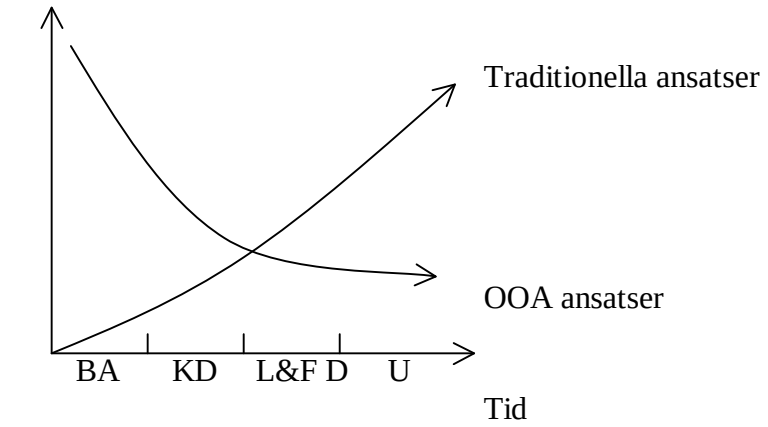
Det ”förväntas” att den tid som satsas på analysen (behovsanalys och konceptuell design) leder till mindre underhåll, kostnader och insatser.

Vid användandet av OOA tar det längre tid än de traditionella analysmetoderna. Det man förlorar i tid vinner man i mindre underhåll. Underhåll kräver mycket tid och kostar mycket pengar, därför strävar systemutvecklare efter enklare underhåll. Objektorienteringen har många fördelar som bidrar till detta. Inkapsling, arv och en väl strukturerad analys ser vi som bidragande faktorer. Inkapsling innebär att attribut och metoder kapslas in i en klass så att de endast är åtkomliga genom metoden i den egna klassen. Koden i objektet är skyddad, det vill säga koden kan inte ändras av händelser utifrån. Vid arv så ärver subklasserna både attribut och programkod från superklasserna.

Eftersom koden endast skrivs en gång reduceras antal fel och uppstår ett fel behövs korrigering endast göras på ett ställe. Det här bidrar till minskat framtida underhåll.

Vi ser att ett problem med att göra en objektorienterad analys kan vara att den tar längre tid att genomföra vilket ger högre kostnader för utvecklandet. Trots att det lönar sig i långa loppet så har kunderna svårt att inse det. De ser till utvecklingskostnaderna i första hand istället för att tänka långsiktigt. I figur 5-3 ses skillnaden i arbetsinsatser under olika utvecklingsstadier mellan traditionella och OOA ansatser.

Figur 5-3



BA = Behovsanalys
KD = Konceptuell design
L&F D = Logisk och Fysisk Design
U = Underhåll

Efter att vi jämfört CASE*line* med OOA och sett till likheter och skillnader dem emellan har vi kommit fram till en del förslag på förändringar i CASE*line* och dess ToolBox. Vi tycker att upplägget på CASE*line* är bra men det är aldrig fel att göra metoden enklare att förstå och utföra av både användare/kunder och utvecklare.

Vid ovana användare är det viktigt att tänka på att bl a modeller inte är för tekniska. Det kan exempelvis vara bra att komplettera rutingrafen med de mer lättförståeliga/målande användarscenariona. Det blir roligare och enklare att förstå med streckgubbar som motsvarar aktörerna/användarna. Varför inte lägga in lite färg i de olika modellerna. Det gör bilderna gladare och läggs det betydelse i de olika färgerna blir det lättare för användarna att hålla isär de olika begreppen. Gubbar och roliga bilder gör det lättare att lära och roligare att läsa både för kunder och utvecklare.

ToolBoxen ses som det viktigaste hjälpmedlet inom CASE*line*. Vad vi förstått av de anställda vi intervjuat på IBS så använder de inte ToolBoxen i den utsträckning vi anser möjligt. Det som används är mallarna för de dokument som de anställda vet finns där och vet vad de innebär.

För att den ska användas mer krävs en del förändringar. Vi upplever att ToolBoxen är väldigt svår att överblicka. Stilistiskt sett är den osammanhängande och alla rubriknivåer stämmer inte överens. Det borde finnas ett dokument som beskriver hur och i vilken ordning som stegen ska göras.

Det skulle förenkla att ha en översiktsbild som visar de olika stegen och hur de följer på varann. Figur 4-1 är ett exempel på hur en sådan kan se ut. Det viktigt att bildtexterna i översiktsbilden stämmer överens med de rubriktexter som finns vidare i ToolBoxen. Den information som vi fått genom utbildning och andra bilder och texter har varit svåra att direkt jämföra med de som finns i ToolBoxen. För att ytterligare underlätta förståelsen för strukturen och hur stegen skall utföras vore det bra att ha ett genomgående exempel. Det ska vara ett enkelt exempel som kan följas i alla steg och som utvecklare kan relatera till. Exemplet får inte vara ett som tas upp på utbildningen i CASE*line*, det är bra att kunna relatera till flera exempel. Utbildningen av CASE*line* borde vara något längre, i dag är den en dag, samt inkludera genomgång av ToolBoxen. Många av de anställda saknar idag utbildning inom CASE*line* men det är något som IBS planerar att ändra på.

Vårt uppsatsämne kan utvecklas genom att gå vidare och beskriva de återstående stegen inom systemutveckling och jämföra design, konstruktion, testning, implementation och underhåll i objektorientering och CASE*line*. Det vore intressant att göra en omfattande undersökning hur användarna/kunderna uppfattar metoderna. Det kan vara en idé att studera hur det fungerar på ett företag som använder sig av objektorienterad systemutveckling.

KÄLLFÖRTECKNING

Litteraturkällor

- David Brown, An introduction to Object-oriented Analysis, *John Wiley & sons, Inc.*, 1997
- Peter Coad & Edward Yourdon, Object-oriented analysis, *Yourdon Press*, 1991
- Hans-Erik Eriksson & Magnus Penker, Objekt-orientering – handbok och lexikon, *Lunds Studentlitteratur*, 1996
- Johan Fagerström, Objektorienterad analys och design – en andra generationens metod, *Lunds Studentlitteratur*, 1995
- Lars Mathiassen m fl Objektorienterad analys och design, *Lunds Studentlitteratur*, 1995

Intervjuer IBS Konsult AB

- Stefan Milenkovic, utredare, 990507
- Gerardo Machado Gandaria, programmerare, 990510
- Inger Björner, metodansvarig, 990512
- Gunilla Jarfelt, systemerare, 990512
- Clas Boström, systemerare, 990521
- Fredrik Svensson, programmerare,990521
- Francisco Rubiano, systemerare, 990521

Övriga källor

- CASE*line* ToolBox är en databas som finns på IBS intranet.