Informationssystemarkitektur

Problemet med att realisera den önskade arkitekturen

Maria Hansson *c98mha@cs.umu.se*
Elin Kröger Nygren *c98ekn@cs.umu.se*

Institutionen för Datavetenskap
Umeå Universitet
27 april 2003

Abstract

Organizations where business depends on data demands an advanced and well adjusted software system. These systems are called enterprise systems and include all data and business rules needed. Developing and maintaining these systems demands an organized and structured enterprise system architecture. The enterprise system architecture represents the fundamental choices of design that the entire system is based on. The topic of enterprise system architecture is very complex since there are many aspects to consider and often several stakeholders to please. This thesis examines the topic enterprise system architecture on a general and comprehensive level. The thesis contains two parts, part one is a literary study and part two is the documentation of a case study at CSN (Centrala Studiestödsnämnden).

Innehåll

Figurer

1 Inledning

Stora organisationer där verksamheten är beroende av data kräver ett avancerat och väl anpassat datasystem. Dessa datasystem kallas för informationssystem eftersom de innehåller all verksamhetsanknuten data, men även det regelverk som tillämpas i verksamheten. Målen med dessa informationssystem är att avlasta användarna från rutinmässiga processer genom att låta systemet utföra dessa och på så vis öka effektiviteten i verksamheten.

Ett stort problem kring införande och förändring av informationssystem är dess komplexitet. Att utveckla och underhålla dessa system kräver en organiserad och strukturerad informationssystemarkitektur, vilken reprensenterar de grundläggande designvalen som hela systemets uppbyggnad baseras på. De arkitekturella[1] valen kommer därmed påverka hela utvecklingsprocessen och systemets fortsatta livscykel.

Denna uppsats undersöker ämnet informationssystemarkitektur på en generell och övergripande nivå. Arkitekturellt arbete omfattar allt från verksamhetsnära frågeställningar, så som budget och resursfördelning, till rena tekniska avvägningar, så som val av arkitekturell stil. Denna uppsats kommer därmed ta upp både de verksamhetsmässiga och de mer tekniskt inriktade aspekterna.

Uppsatsen riktar sig till personer med datavetenskapligt intresse och kunskap, men bör till stor del även kunna intressera mindre datavetenskapligt insatta. Trots skillnaden i kunskap bör båda grupperna kunna ta till sig innehållet i denna uppsats. Vår förhoppning är att den för de med mindre kunskap inom detta område ska ge en inblick i ämnet och förståelse för verksamheters beroende av informationssystem. De mer insatta kan ta till sig uppsatsen som en tankeställare kring vikten av ett gediget arkitekturarbete och inspireras till att fördjupa sig inom något avsnitt.

1.1 Problembeskrivning

Målet med uppsatsen är att undersöka ämnet arkitektur bakom informationssystem och vilka frågeställningar man ställs inför vid arkitekturellt arbete. Ämnet ska undersökas genom en generell litteraturstudie och kompletteras med en praktisk fallstudie där vi undersöker problemet med att realisera

[1] Enligt svensk ordlista är *arkitektonisk* det korrekta adjektivet vid arkitektursammanhang. Inom systemarkitektur är dock ordet *arkitekturell* så pass vedertaget att vi istället väljer att använda det i denna uppsats.

den önskade arkitekturen. Den generella delen ska ligga grund för ett lyckat genomförande av den praktiska delen. Den praktiska delen inleds med utformning av en generell metod för utvärdering av planerade och realiserade informationssystem med fokus på dess underliggande systemarkitektur. Denna metod ska därefter användas och testas vid fallstudien.

1.2 Metod

Uppsatsen är uppdelad i två delar. Del ett är en teoretisk studie som behandlar generella begrepp och aspekter på informationssystemarkitektur och är utförd i form av en litteraturstudie. Del två är utförd på CSN i Sundsvall och är en fallstudie av det nyligen införda informationssystemet Stis2000. Denna studie är utförd i form av en diskussion med åtta personer som varit inblandade i projektet. Ingående metodbeskivning för denna fallstudie finns redovisad i del två.

Del I
Litteraturstudie

Denna del av uppsatsen är dokumentering av utförd litteraturstudie. Utifrån det material vi har studerat presenteras vår syn på ämnet informationssystemarkitektur.

Uppsatsen inleds med förklaringar av begreppet informationssystem och andra begrepp som är vikiga för den fortsatta förståelsen av uppsatsen. Därefter kommer ämnet informationssystemarkitektur beskrivas med början på en verksamhetsnära nivå följt av en mer teknisk nivå. Delen avslutas med problematiken kring realisering av den önskade systemarkitekturen.

2 Informationssystem

Ett informationssystem är en datoriserad hantering av en verksamhets information och till för att hjälpa och avlasta användaren att utföra sina uppgifter. Rutinartad manuell hantering av information kan dessutom effektiviseras betydligt med hjälp av informationssystem. Utan informationssystem skulle t.ex. en återförsäljare måsta hålla all sin information om kunder, varor, leverantörer m.m. i pappersformat, vilket inte är speciellt effektivt.

Det existerar många olika definitioner av vad ett informationssystem är och vad det bör innehålla. Enligt Karin Axelsson bygger många av dagens definitioner av informationssystem på en tidig definition av Börje Langefors som med informationssystem avser *ett system som har till uppgift att samla in, lagra, behandla och distribuera informationsmängder* [Axe98], [Lan78].

European Software Institute ligger ungefär i linje med detta och säger att ett informationssystem kan definieras som de delar av verksamheten som tillhandahåller, använder och förmedlar information tillsammans med tillhörande resurser såsom personal, teknik och pengar [Eur02].

Även Anders Eriksson har en liknande övertygelse och anser att det finns tre grundläggande funktioner i ett informationssystem, bearbetning av data, lagring av data och kommunikation. Dessa tre funktioner är alla beroende av varandra för att kunna vara till någon nytta [Eri89].

Karin Axelsson menar att teorierna *socialfenomenologi* och *talaktsteori* kan

användas för att definiera informationssystem och att många gjort detta.

Inom *socialfenomenologin* beskrivs hur en handling, beroende på hur ofta den utförs, kan gå genom flera olika stadier och tillslut bli helt vanemässig. En handling som från början är en vana hos en människa kan dessutom institutionaliseras om den börjar utföras på ett likartat sätt av många människor i ett samhälle. Aktören eller aktörerna slipper då fundera på handlingen varje gång den ska utföras och arbetet underlättas betydligt. Om en socialfenomenologisk syn på informationssystem används kan man säga att informationssystemen utvecklas av människor, för människor, och att de därför är mänskliga produkter [Axe98].

Hans-Erik Nissen har en socialfenomenologisk syn på informationssystem och menar att det är viktigt att inte glömma bort att informationssystem är mänskliga produkter. Han menar att ett informationssystem antingen kan vara [Nis82]:

- ett system som en person eller en grupp av människor använder för att finna den information de behöver för att sköta sitt arbete
- eller ett system som en person eller en grupp av människor använder för att sända information till andra personer eller grupp av människor

Det är alltså viktigt att inte tillskriva informationssystem egenskaper som det inte har, som t.ex. att det är systemets fel när ett visst beslut fattats.

Inom *talaktteorin* är själva talet en handling. Dessa kommunikativa handlingar utförs av människor i relation till varandra under vissa givna regler. Om någon av aktörerna bryter mot någon av de institutionaliserade reglerna uppstår förvirring hos de andra. Människor påverkar alltså sin omvärld samtidigt som omvärlden påverkar den enskilda människan. Detta synsätt kan hjälpa till vid skapande av informationssystem genom att visa på vikten att se att en verksamhet till stor del består av kommunikation. Att utforma en informationssystemarkitektur innefattar då design av en kommunikativ struktur [Axe98b].

Göran Goldkuhl ansluter sig till den talaktteoretiska synen på informationssystem och menar att ett informationssystem är ett språkligt objekt utvecklat av människor för att underlätta människors kommunikation, förståelse och handlingar [Axe98b]. Han menar även att informationssystem innehåller meddelanden, processer och regler. Informationssystem styrs av dessa regler som i sin tur är uttryck av människors avsikt med systemet [Gol82].

Bengt Andersson avser med informationssystem att en viss informationshantering i en organisation i ett visst avseende har blivit strukturerad, institutionaliserad och innehåller ett formaliserat verksamhetsspråk. Även han ansluter sig till talaktsteorin och menar att informationssystem är socialt inbäddade i en verksamhet och är till för kommunikation mellan olika aktörer [And98].

De två teorierna står inte i motsatsförhållande till varandra, men de betonar olika aspekter när det gäller informationssystem. Axelsson hävdar då att en definition hämtad från båda teorierna skulle vara en mer heltäckande bild av informationssystem och använder sig av följande punkter för definitionen [Axe98]:

- Ett informationssystem är en intentionellt skapad artefakt[2].
- Ett informationssystem är en datorbaserad artefakt som är socialt inbäddad.
- Ett informationssystem styrs av verksamhetsregler av språkligt och annat slag.
- Verksamhetsreglerna är formulerade av människor och är uttryck för deras intentioner med informationssystemet.
- Ett informationssystem ska genom kommunikation informera och påverka människor.
- Människor tolkar meddelande från informationssystemet utifrån sin förståelse och motivation.

Dessa punkter visar tydligt på att informationssystem är gjort av människor och för människor. Det är då viktigt att inte glömma bort det sociala sammanhanget och vikten av att meddelanden utformas så att alla användare, oberoende av bakgrund, kan förstå dessa.

I denna uppsats kommer begreppet informationssystem överensstämma med ovanstående definition.

[2]Ordet finns förklarat i ordlistan.

2.1 Bakgrund

Sedan 1950-talet då persondatorerna gjorde sitt intåg i Sverige har många informationssystem utvecklats. Dessa första informationssystem utvecklades i stort sett inte efter någon generell metod och utvecklingsarbetet var dessutom till största delen fokuserat på den tekniska delen av systemen. Under 1960-talet började dock intresset för mer helomfattande och generella metoder växa fram [Fle85].

Under 1970-talet var det dataavdelningarna som styrde vilka system som skulle installeras och användarnas påverkan på dessa system var minimal. Detta har gjort att det fanns och fortfarande i viss mån existerar system där användarvänligheten i princip är obefintlig. Under slutet av 1970-talet började fokusering dock läggas mer på de som skulle använda systemen och verksamheten inom databehandling organiserades mer effektivt. Under 1980-talet började dessutom övergången från fokusering på behandling av data till fokusering på att behandla data som information [Gor83]. Det är under den här tiden informationssystem börjar få någon väsentlig mening.

De tidiga informationssystemen var främst till för att automatisera och effektivisera rutinartade arbetsprocesser och var aldrig synliga för kunden. Informationssystemen har sedan utvecklats till att vara så kallade "front-office-system". Sådana system är, förutom att hantera själva informationsbehandlingen, ett verktyg och hjälp för de personer inom verksamheten som är i kontakt med kunden.

De senaste åren har utvecklingen av informationsystem dessutom gått mot att inkludera användning av Internet för att förmedla information till kunder, anställda och underleverantörer [Dav00].

2.2 Olika typer av informationssystem

Det finns flera olika typer av informationsystem. Man kan dela in dessa i två huvudgrenar, skräddarsydda system och standardsystem. Det existerar naturligtvis även system som är en blandning av dessa två huvudgrenar där t.ex. färdiga moduler köps som sedan samman till ett slags skräddarsytt system. I realiteten är det dessutom sällan ett system kan byggas utan inblandning av andra system. Det finns oftast gamla system, s.k. legacysystem, att ta hänsyn till. Legacysystem beskrivs på följande sida.

2.2.1 Standardsystem

Ett standardsystem är ett standardiserat informationssystem som efter viss anpassning ska kunna användas av flera olika typer av organisationer. Dagens standardsystem grundar sig på forskning och erfarenhet från olika typer av företag och organisationer vilket medför att dessa kunskaper och erfarenheter finns inbyggda i systemen [Nil00].

Ett standardsystem kan vara aktuellt att införa för en organisation om dess verksamhet är så pass standardiserad att det existerar passande system på marknaden som kan införas utan krav på för stora förändringar inom verksamheten.

Det finns många olika typer av standardsystem, t.ex. ekonomisystem, produktionssystem och faktureringssystem.

2.2.2 Skräddarsydda systems

Ett skräddarsytt system är precis som det låter på namnet speciellt utvecklat för en viss verksamhet. En fördel med dessa är att de är anpassade till den aktuella verksamhetens krav och behov, medan nackdelen ligger i att de ofta tar lång tid att utveckla. En annan nackdel är att viktiga affärsmöjligheter kan förbises genom att den långa erfarenheten, så kallad ”best-of-practise”, som finns inbyggd i standardsystemen går förlorad.

Ett skräddarsytt system kan vara aktuellt att införa för en organisation vars verksamhet är väldigt specialiserad. Det kan då vara svårt eller omöjligt att anpassa ett standardsystem så att det kan vara till nytta för verksamheten.

2.2.3 Legacysystem

Ett legacysystem[3] är ett äldre, oftast 15 till 25 år gammalt, informationssystem. I dessa system fungerar persondatorerna oftast som smarta terminaler medan en stordator sköter all datahantering.

Legacysystemen utvecklades från 1950-talet fram till slutet på 1980-talet innan internetbaserade applikationer fick sitt genombrott och har för många företag och organisationer blivit väldigt viktiga. Detta gör att de kan vara svåra att byta ut även om de kan vara byggda med en föråldrad teknik. Att

[3]Ordet kommer av engelskans *legacy system*, men har i dessa sammanhang blivit ett vedertaget begrepp även i svenskan.

bygga helt nya system är dessutom ofta en dyr och riskfylld process som kanske inte kan motiveras ekonomiskt om legacysystemet fungerar. Det kan alltså vara ekonomiskt försvarbart att behålla legacysystemen och förändra dem till att fungera tillfredställande eller att behålla dem och att bygga ett nytt system som ska interagera med det gamla.

Reverse engineering är en process där ett legacysystem undersöks för att bestämma dess status och för att identifiera om systemet går att behålla och om så är fallet vilka förändringar som då måste göras [Tur99].

Om det beslutas att legacysystemet ska få vara kvar och köras tillsammans med det nya systemet måste dessa två kopplas ihop på något sätt. Det existerar en hel del vertyg där tillverkarna säger sig skapa kommunikation mellan gamla stordatorsystem och nya system. Denna typ av kommunikation brukar dock innebära att någon måste gå in och modifiera de system och gränssnitt som ska kommunicera genom så kallad hårdkodning. Ett sätt att slippa denna hårdkodning är att använda *Enterprise Application Integration (EAI)*. Det består bland annat av metoder och standardprogram för att bygga samman olika system inom en organisation utan att någon ska behöva göra förändringar i koden [Lid99].

3 Arkitekturen och dess begrepp

Med anledning av dagens informationssystems komplexitet och den höga graden av integrering är det oerhört viktigt att ha en genomtänkt underliggande arkitektur till grund för konstruktionen av systemet. I mindre system ligger huvudproblemet kring designen av systemet, där arkitekturen är mer eller mindre självklar, medan i de större komplexa systemen måste betydande analyser utföras inom arkitekturen. Malan och Bredemeyer hänvisar till en framstående forskare inom programvaruarkitektur, David Garlan, som 1992 skrev följande [Bre02]:

> "as the size and complexity of software increases, the design problem goes beyond the algorithms and data structures of the computation: designing and specifying the overall system structure emerges as a new kind of problem... This is the software architecture level of design."

Arkitektur inom området programvarukonstruktion kan enkelt liknas vi byggandet av hus. Konstruktion av stora hus kräver ett omfattande arkitekturarbete för att resultera i ett hus med önskad funktionalitet, god kvalitet och hållbarhet. Att bygga ett hus adhoc[4] kan vi alla räkna ut att det kommer resultera i misslyckande. Detsamma gäller för datasystem. Kraven på informationssystemet måste noggrant analyseras vartefter arkitekturen utvecklas enligt dessa krav. Även kontrasterna av kraven måste analyseras, dvs. vad får inte hända och hur ska systemet byggas för att motverka detta. Kvaliteten på arkitekturen underlättar även skötseln och vidareutvecklingen av systemet avsevärt.

Bass, Clementz och Kazman har beskrivit tre huvudsakliga anledningar till varför arkitekturarbete är viktigt [Bas98].

Kommunikation mellan inblandade parter. Eftersom arkitekturen presenterar systemet på en hög abstraktionsnivå kan i stort sätt alla inblandade personer och organisationer förstå systemet och på så sätt använda dessa abstraktioner som en bas i diskussioner. Arkitekturen blir med andra ord ett gemensamt språk. Det blir då lättare att få förståelse för varandra och nå gemensamma beslut.

Tidiga designbeslut. Arkitekturen är en produkt av de tidiga designbesluten och därmed basen för hela utvecklingen, driften och underhåll av systemet. Det är svårt att fatta de korrekta besluten samtidigt som det ju

[4]Ordet finns förklarat i ordlistan.

längre fram i utvecklingen av projektet kommer blir svårare att förändra dessa eftersom effekten av besluten kan genomsyra hela systemlösningen.

Återanvändbar abstraktion av systemet. Ju tidigare återanvändning kan tillämpas i utvecklingen desto större blir vinsten. Återanvändning på arkitekturell nivå omfattar alla de tidiga arkitekturella designbesluten. Arkitekturell återanvändning innebär att flera system med liknande krav kan använda samma uppsättning arkitekturella beslut. Exempel på återanvändning av arkitekturella beslut är product-line arkitekur och komponentbaserad systemarkitektur. Dessa begrepp kommer vi redogöra för senare i rapporten.

Arkitekturarbete omfattar allt från verksamhetsanalys till riktlinjer för konstruktionen av systemet och vidare till vidareutvecklingen och förnyelsen av systemet. I denna arbetsgång ingår många komplexa och något "luddiga" begrepp. I detta avsnitt ska vi beskriva de begrepp vi valt att använda. Då innebörden av begreppen har skilda betydelser i olika sammanhang bland olika forskare och forskargrupper kommer vi att försöka förmedla vår syn på dessa begrepp. Detta bör då kunna ligga till grund för förståelsen av hela denna uppsats.

3.1 Informationsystemarkitektur

Det första problemet man stöter på inom detta område är något så grundläggande som själva ordets betydelse. Vad betyder egentligen informationssystemarkitektur? Det finns ingen gemensam vedertagen definition på detta. De flesta forskare har en egen definition på begreppet informationssystemarkitektur. Det finns vissa delar av området där man nått en enhetlig uppfattning men fortfarande finns motstridigheter om vissa delar av området. För att beskriva begreppet informationssystemarkitekturs tvetydighet hänvisar Magoulas och Pessi till ett citat skrivet av E.A. Ben-Nathan 1980 i boken *The information systems architect and systems development* [Mag91]:

> "Information systems architecture is one of these simple-sounding words with a very fuzzy meaning".

Enligt Magoulas och Pessi finns det dock ett antal påståenden som alltid stämmer in på begreppet informationssystemarkitektur, även kallad IS-arkitektur, oavsett hur den aktuella definitionen lyder [Mag91].

IS-arkitekturen:

- har en bestämd integritet. Arkitekturens integritet bestäms av dess bakomliggande arkitekturfilosofi och är därför alltid permanent.

- avser först och främst att reflektera organisationens informationsförsörjnings- och informationsbehandlingsuppgifter.
- avser både existerande och planerade delar av informationsförsörjningen och informationsbehandlingen.
- bör vara relaterad till ansvarsförhållanden.
- omfattar tidsberoende delar. Men varje förändring som sker bör respektera de principer och regler som ligger under IS-arkitekturens utformning och förvaltning.
- bör vara i harmoni dels med det syfte som skall uppfyllas dels med vissa bestämda principer eller regler som kan användas för att avgöra arkitekturens lämplighet och konsistens. I annat fall finns det risk för att arkitekturens integritet förloras.

Axelsson och Goldkuhl har följande uppfattning om IS-arkitektur [Axe98b]:

> ”Med IS-arkitektur menas hur man inom en organisation fördelar information och informationshantering i olika informationssystem och därmed avgränsar dessa, samt även hur ansvar för detta fördelas.”

Informationssystemarkitekturen är därmed den övergripande beskrivningen över hur en verksamhets informationsförsörjning och informationsbehandling ska struktureras utan att gå in på de tekniska delarna. Relationer beskrivs på en verksamhetsmässig nivå, informationssystem emellan, människa-system interaktionen men även olika organisationers samverkan. Informationssystemarkitekturen blir en inledande mall för hur verksamhetens enheter ska representeras i systemet i form av ett antal delsystem. Delsystemens olika roller och ansvarsområden specificeras. Även delsystemens informationsbehov och vilka dataflöden de genererar framgår i informationssystemarkitekturen. Det arkitekturella arbetet inleds ofta med en verksamhetsanalys och en verksamhetsmodellering. Dessa beskrivs i ett senare avsnitt.

För att vidare förklara begreppet arkitektur i dessa sammanhang introduceras en rad nya begrepp, systemarkitektur, databasarkitektur, mjukvaruarkitektur och IT-infrastruktur. Det gemensamma bland dessa är att de alla används för att omnämna den struktur och ordning som råder utifrån ett visst perspektiv. Vi anser att informationssystemarkitektur är ett övergripande begrepp i förhållande till de övriga.

3.2 Systemarkitektur

Axelsson och Goldkuhl [Axe98b] anser att systemarkitektur är en synonym till informationssystemarkitektur. De har valt att använda begreppet systemarkitektur helt utifrån dess verksamhets- och informationsmässiga aspekter och tar avstånd från de tekniska aspekterna i begreppet. I denna uppsats kommer dock informationssystemarkitektur och systemarkitektur vara skilda begrepp. Vi anser att begreppet systemarkitektur har en så pass teknisk innebörd att det är olämpligt att använda det enbart ur ett verksamhetsmässigt perspektiv.

Systemarkitekturen avser den tekniska och mer ingående beskrivningen av delsystemen och relationerna beskrivna i den övergripande informationssystemarkitekturen. Delsystemen delas upp i komponenter och mer detaljerade samband beskrivs. Även inom systemarkitektur finns en mängd definitioner. En vanlig benämning på systemarkitektur är IT-arkitektur. Vi har dock valt att använda begreppet systemarkitektur då vi anser att IT-arkitektur används i alltför skilda sammanhang. Systemarkitekturens primära syfte är att specificera systemet så att man får ett så stabilt och effektivt system som möjligt, utan att för den delen bortse från några av verksamhetens krav på systemet.

I vår mening omfattar systemarkitekturen mjukvaruarkitekturen och databasarkitekturer och kopplingar däremellan. Ofta används endast begreppet mjukvaruarkitektur när ett systems tekniska beskrivning omnämns. Vi anser att mjukvaruarkitektur är ett för smalt begrepp för att omfatta både mjukvara och datalager men även viss relation till hårdvaran, därav denna uppdelning.

3.2.1 Mjukvaruarkitektur

Mjukvaruarkitekturen avser att beskriva den rena programvarans inre arkitekturella uppbyggnad. Denna arkitektur är helt friställd från hårdvara och yttre komponenter. Mjukvaruarkitekturen beskriver programvarans moduler och hur dessa moduler ska utformas. Även relationerna, som leder till den önskade funktionaliteten, emellan dessa anges [Kin02]. Bass, Clements och Kazman förklarar mjukvaruarkitekturen som strukturen eller strukturerna av ett system, dvs. dess mjukvarukomponenter, de externt synliga egenskaperna hos dessa komponenter och relationerna mellan dem. Med externt synliga egenskaper menas de antaganden andra komponenter kan göra om en komponent, så som dess tjänster, felhantering, dess användning av delade resurser

osv. Tanken med deras definition av mjukvaruarkitekturen är att man måste abstrahera bort viss information om systemet men samtidigt tillhandahålla tillräckligt med information för att forma en bas för analys, beslutsfattande och minskade risker i systemet [Bas98].

3.2.2 Databasarkitektur

Då dagens informationssystem alltjämt lagrar information i databaser är det oerhört viktigt med struktur i datalagring och datahantering. Valet av lämplig databasarkitektur har stor betydelse för systemets kvalitet, såsom prestanda och säkerhet. Databasarkitekturen måste stödja de arkitekturella besluten i den övriga utformningen av systemarkitekturen. Olika typer av arkitekturella avvägningar inom databasarkitektur är valet av datamodell och databashanterare (DBMS), principer för den fysiska lagringen, t.ex. partitionering eller spegling, hanteringsregler av korrupta databaser etc. [DFK02b]. Då denna uppsats är begränsad till att inte omfatta databasarkitektur kommer vi inte gå in närmare på detta i denna uppsats. Därmed inte sagt att databasarkitektur är oviktigt i systemarkitekturella sammanhang.

3.2.3 IT-infrastruktur

IT-infrastrukturen avser hårdvara i form av datorer, nätverk och annan utrustning. Även basprogramvara ingår här. Med basprogramvara menas operativsystem och annan grundläggande programvara, t.ex. Office-paketet. Det är viktigt att skilja mellan IT-infrastrukturen och den övriga systemarkitekturen. IT-infrastrukturen möjliggör realiseringen av olika informationssystem genom att tillhandahålla olika resurser [Axe98b]. Även IT-infrastruktur ligger utanför ämnet i denna uppsats och kommer därför inte beskrivas närmare.

3.3 Arkitekten

Den som arbetar med arkitekturens utformning har ofta rollen som arkitekt eller IT-arkitekt. Förutom att ansvara för arkitekturen har arkitekten även i uppgift att övervaka konstruktionen av arbetet för att se till att fastställda regler och principer efterföljs. Eftersom arbetet med arkitekturen är så omfattande och till stor del avgörande för systemets lyckade realisering fortsätter arkitekturarbetet genom hela projektet. När man väl lämnar över arkitekturen till designen och implementationen upphör inte arbetet för arkitekten. Denne ska då leda de tekniska frågeställningar som uppkommer under arbetets gång.

Arkitekten har även ansvar gentemot verksamheten. Denne ska fungera som länken mellan verksamhetens krav och utformningen av systemet, dvs. omforma behoven till en realiserbar arkitektur så effektivt som möjligt och till en rimlig kostnad. Arkitekten har med andra ord ett omfattande arbete att sköta. Det krävs därför att denne är insatt i så väl verksamheten som programvaruutveckling. Enligt Hofmeister, Nord och Soni måste arkitekten vara en visionär för att lyckas. Detta innebär att arkitekten måste ha en vision av hur systemet kommer att se ut i slutskedet, vilka krav det kommer att uppfylla och hur det kommer att passa in i verksamheten [Hof00].

IT-arkitekt som yrkesroll är en relativt ny företeelse. Konstruktion av system har gjorts länge och systemstrukturen har länge ansetts betydelsefull men i och med den ökade produktionen av komplexa, integrerade system med avancerade strukturer har behovet av kvalificerade IT-arkitekter ökat. Yrkesrollen som arkitekt skiljer sig från många andra roller inom IT. De flesta yrkesroller kräver en viss specialisering inom något område, tex. databaser, datakommunikation eller någon speciell programmeringsmetod, medan rollen som IT-arkitekt kräver ett övergripande kunskapsområde. Denne måste därmed vara mer av en generalist än specialist. Att klara av rollen som IT-arkitekt kräver ofta mångårig erfarenhet av programvaruutveckling [DFK02]. Det finns idag möjlighet att certifiera sig som IT-arkitekt.

Erik Dyrelius har beskrivit ett antal viktiga egenskaper hos en IT-arkitekt [Com01]:

Stark och okuvlig. Arkitekten måste har styrka nog för att kunna försvara sina arkitekturella idéer och får inte vika sig för exempelvis konstruktörers motvilja. Kompromisser kan vara lämpliga så länge arkitekturen inte förlorar sin konsistens och stabilitet senare i konstruktionen.

Lyhörd och flexibel. Då arkitekturarbete till stor del består av ett övergripande skissande över den vision arkitekten har är det viktigt att arkitekten känner av när dessa, ofta abstrakta, skisser måste förfinas, konkretiseras och exemplifieras. Arkitekten måste även våga erkänna misstag, vilket kräver ett flexibelt tankesätt.

Skicklig. Till skillnad från systemutvecklingsmodeller som går ut på att iterera fram den bästa lösningen kan arkitekten inte ta hjälp av iteration för att nå den lämpliga arkitekturen. Iterering i arkitekturella sammanhang kan innebära att arkitekturen mister sin stabilitet mer och mer för varje iteration. Det är därför viktigt att arkitekten redan i ett inledande skede kan avgöra

vilken arkitektur som är den bästa. Detta kräver stor skicklighet hos arkitekten. Självklart kan arkitekten senare i utvecklingen förfina arkitekturen och på så vis för varje iteration förbättra denna.

Vidsynt. För att lyckas måste arkitekten se systemet ur olika perspektiv. Att se systemet ur många olika håll och kunna se kopplingarna mellan olika vyer är svårt men nödvändigt.

Stålmannen. Dyrelius förklarar slutligen att den perfekte arkitekten är svår att finna. Han menar att antagligen måste egenskaper lånas av stålmannen för att en arkitekt ska vara perfekt. Med andra ord existerar endast den perfekte arkitekten i form av ett ideal som bör eftersträvas.

4 Verksamheten

Gapet mellan verksamhetens behov och systemets utformning är en svår fas i systemutvecklingen eftersom det ofta handlar om skilda avdelningar inom en organisation, så kallat verksamhetsfolk och IT-folk. Det krävs förståelse för den andra parten och god samverkan för att arbetet med utformningen av informationssystemarkitketuren ska resultera i en lyckad sådan. När man ska planera och bygga ett informationssystem är forskare inom området eniga om att arbetet ska inledas med en fas där verksamhetens behov och krav på systemet överförs till en plan eller struktur för systemets uppbyggnad. Systemet ska inledningsvis specificeras utifrån ett verksamhetsperspektiv. I denna specifikation anges systemets delsystem beroende på hur verksamhetens enheter ser ut, utan att gå in på detaljer för hur enheterna ska se ut. Att specificera de tekniska detaljerna sker därmed senare i arbetet, då systemarkitekturen utformas.

Informationssystem ska tillhanda hålla stora mängder av information och informationsbehandlingen är oftast verksamhetens primära uppgift. Detta ställer omfattande krav på systemet. Systemet måste tillhandahålla informationen så att denna kan insamlas, lagras, återsökas, överföras och presenteras så effektivt som möjligt [Axe98b]. Systemets övergripande uppbyggnad måste alltså noga planeras och analyseras. Det finns ingen given eller någon bevisat bästa arkitektur för en verksamhets informationssystem. Informationshanteringen kan struktureras på oerhört många olika sätt beroende på de verksamhetens funktionella egenskaper men även på de tekniska begränsningarna som finns. Strukturingen måste alltså utgå från båda dessa perspektiv.

Då systemen ofta består av ett flertal sammankopplade delsystem är det viktigt att ha en plan över hur verksamhetens samlade informationsbehandling ska ske. De olika delsystemens samverkan måste noggrant specificeras. Detta sker ibland relativt oplanerat och adhoc-mässigt. Det resulterande systemet blir då ett resultat av många olika beslutsfattare med var sina principer. Framväxten av systemets delsystem sker oplanerat och de enskilda besluten kan inte ses som strategiska beslut. Detta kan leda till en bristande struktur på systemet, s.k. spaghettistruktur [Mag91]. Informationssystemarkitekturen blir då en konsekvens av systemutvecklingen och inte tvärt om. Axelsson och Goldkuhl har förklarat spaghetti-syndromet med följande punkter [Axe98b]:

- illa strukturerade
- svåröverblickbara

- intrasslade (med oklara gränser och överlappningar)
- svårföränderliga
- oförutsägbara vad gäller följdeffekter vid drift och förvaltning

Oklara gränser mellan de olika delsystemen leder ofta till ett överintegrerat system. Man har alltså integrerat delsystemen med varandra i för hög grad. Det finns även ett motsatt fall, system som är s.k. isolerade öar, där delsystemen är för lite integrerade och inte kan eller har svårt att kommunicera med varandra. Även i detta fall har man misslyckats med systemstruktureringen [Axe98b].

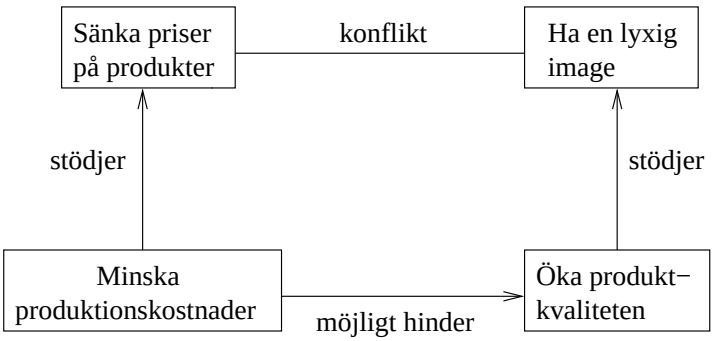
Den övergripande informationssystemarkitekturen kan kallas för en verksamhetsmodell. Denna är en naturlig följd av en genomförd verksamhetsanalys och är därmed den mest verksamhetsnära fasen i arbetet med informationssystemarkitekturen.

4.1 Verksamhetsanalys

Eftersom ett informationssystem utformas med syftet att stödja en viss verksamhet är det viktigt att utgå från verksamheten och dess uppgifter. Enligt Axelsson och Goldkuhl är verksamhetsanalysen huvudsakliga syfte att ta reda på hur den befintliga verksamheten fungerar genom att klargöra verksamhetsrutiner, arbetssätt, handlingar, aktörer och ansvarsgränser [Axe98b]. Det krävs alltså omfattande förståelse för verksamhetens alla moment. Verksamhetens handlingsprocesser måste vara identifierade och analyserade för att genom informationssystemet kunna effektiviseras. Genom att analysera verksamhetens processer ökar även förståelsen för vilka krav verksamheten har på systemet. Verksamhetsanalysen bör göras i samarbete mellan systempersonal och verksamhetspersonal för att få en så äkta bild av verksamheten som möjligt. Det är därför viktigt att låta verksamhetsanalysen vara en öppen diskussion så att det skapas ett gemensamt språk och att alla dolda antaganden och behov lyfts fram [Are01].

Verksamhetsanalysen bör ske i enlighet med organisationens mål för att kunna kartlägga förändringsbehoven. Det kan vara lämpligt att identifiera målen som organisationen har med sin verksamhet genom att utföra en målanalys. Målen relateras då till varandra och ordnas i någon form av prioritetsordning. Många mål kan delas in i delmål som syftar till att uppfylla huvudmålet. Samband mellan huvudmål och delmål kan lämpligen beskrivas i en målmodell

eller målgraf. I denna kan även konfliktande mål identifieras [Axe98b]. Figur 1 visar ett exempel på en enkel målgraf.



Figur 1: Exempel på en målgraf

4.2 Verksamhetsmodellering

För att dokumentera och få struktur på verksamhetsanalysen är det lämpligt att utföra en verksamhetsmodellering, även kallad konceptuell modellering. Genom att verksamhetsmodellen beskriver verksamheten i olika aspekter kan de inblandade få en klar bild över verksamheten [Age02]. En verksamhetsmodellering ska resultera i en definierad och avstämd problemdefinition.

Jens Allwood, professor i lingvistik, förklarar att verksamhetsmodellen kan ha två syften, ett deskriptivt syfte och ett preskriptivt syfte. En deskriptiv modell är en modell **av** verkligheten och används för att inge förståelse för den aktuella verksamheten. En preskriptiv modell är en modell **för** verksamheten och används för att beskriva önskad utformning av verksamheten, dvs. hur det borde se ut. I sammanhang som informationssystemutveckling används ofta en kombination av deskriptiva och preskriptiva modeller, eftersom det är intressant ”var verksamheten står” men även ”var verksamheten går” [Alw89].

Att modellera en verksamhet innebär att med hjälp av bilder, diagram, artificiellt och naturligt språk avbilda och beskriva de väsentliga delarna av en verksamhet. Verksamhetsmodellen ska beskriva de viktigaste aspekterna i verksamheten, så som t.ex. funktion, resurser, omgivning, aktivitet. Det kan därför vara lämpligt att dela upp modelleringen i olika perspektiv och på så vis skapa olika typer av delmodeller. Det krävs då även modeller som integrerar de olika delmodellerna.

Då verksamhetsmodellen ska ligga till grund för det fortsatta arbetet med utvecklingen av systemet är det oerhört viktigt att alla inblandade är överens om utformningen av den färdiga modellen och att det inte råder några tvivel om hur denna ska tolkas. Det är även viktigt att modellen inte innehåller några felaktigheter då detta kan få omfattande följder senare i utvecklingsarbetet.

4.3 Analys- och modelleringsmetoder

Det finns ett antal metoder för att genomföra verksamhetsanalys och verksamhetsmodellering. Då denna uppsats fokuserar på systemarkitekturens tekniska utformning går vi inte närmare in på dessa modeller. Metodernas gemensamma syfte är att stödja och vägleda kartläggningen av verksamhetens delar och uppgifter.

4.3.1 RUP/The Business Modelling Workflow

RUP, Rational Unified Process, är en mjukvaruutvecklingsmetod med syfte att göra projektet förutsägbart och repeterbart. RUP innehåller riktlinjer för arbetet och definierar vem som ska göra vad, när det ska göras och hur det ska göras. RUP består av ett antal olika arbetsflöden, dvs. sekvenser av aktiviteter, med ett resultat i form av ett observerbart värde. Ett av dessa aktivitetsmönster är affärsmodellering, ”The Business Modelling Workflow”. Målet med affärsmodelleringen är att förstå strukturen i den organisation som systemet ska utvecklas till. De nuvarande problemen i organisationen ska identifieras och möjliga förändringar ska föreslås. Kunder, användare och systemutvecklare ska i och med denna affärsmodellering få gemensam förståelse för verksamheten i organisationen. Utifrån detta ska slutligen systemkraven för att stödja verksamheten kunna identifieras [Kru01].

4.3.2 MBI

Mats-Åke Hugosson har utvecklat en metod för verksamhetsanalys, MBI. MBI står för Mål, Beslut, Information. Denna metod vägleder genomförandet av en verksamhetsanalys. MBI är en generell metod som är tänkt att fungerar i alla typer av situationer där informationsbehandlingen ska förändras eller förnyas. Metoden syftar till att praktiskt vägleda genomförandet av en verksamhetsanalys genom att lägga fokus på organisationens aktiviteter och på så sätt engagera användaren till stor del [Ber99].

4.3.3 VIBA/SIMM

Forskningsgruppen VITS[5] har utvecklat ett antal metoder för verksamhets- och systemutveckling. Metoderna tillhör en gemensam "metodfamilj", SIMM. SIMM står för Samverkan och Situationsanpassning, Ifrågasättande och Idéutveckling, Meningsskapande och Målstyrning, Modellering och Metod. En metod i SIMM är verksamhets- och informationsbehovsanalys, VIBA. VIBA är en metod för att specificera informationssystem och dess verksamhetsmässiga kontext [Axe98b].

4.4 Verksamhetens funktionella egenskaper

Utöver verksamhetsmodellen kan det behövas ingående funderingar och ställningstaganden kring ett antal funktionella egenskaper. Det är då viktigt att känna till verksamhetens struktur och dess handlingar eftersom informationssystemet i stor utsträckning ska avspegla verksamheten och dess uppgifter. Utifrån verksamhetens uppgifter kan man formulera ett antal verksamhetsspecifika krav som i sin tur leder till informationssystemarkitekturens funktionalitet.

Enligt Magoulas och Pessi finns det i huvudsak fyra övergripande verksamhetsmässiga egenskaper som måste diskuteras innan kraven på funktionalitet kan formuleras [Mag91].

- **Specialisering** - identifiera och gruppera verksamhetens handlingar i olika verksamhetsområden.
- **Standardisering** - bestämma regler, normer och principer som ska gälla inom verksamheten vid likartade händelser.
- **Formalisering** - informationens egenskaper måste preciseras och den relevanta information som krävs i utbyte av information mellan verksamhetsdelar måste definieras.
- **Centralisering/Decentralisering** - de olika verksamhetsdelarnas handlingsfrihet ska bestämmas.

Dessa fyra verksamhetsmässiga egenskaper är en bra grund till identifiering och formulering av verksamhetens funktionella egenskaper. Magoulas och Pessi har delat in de funktionella egenskaperna i ett antal områden. Dessa

[5]VITS står för Verksamhetsutveckling, Informationsteknologi, Styrning och Samarbetsformer och är en forskningsgrupp som startades 1990/1991 vid Institutionen för datavetenskap, Linköping Universitet.

egenskaper ska arbetas fram med hänsyn till verksamhetens beslutssituationer och handlingar [Mag91].

4.4.1 Informationsbasens omfattning

Informationen, dvs. datat, kan organiseras utifrån olika krav, lokala, translokala eller globala krav. Lokala krav innebär att informationen endast behövs i ett specifikt, specialiserat område av verksamheten. Translokala krav omfattar händelser där två eller ett flertal verksamhetsområden är inblandade. När hela verksamheten påverkas av en händelse handlar det om globala krav.

4.4.2 Formaliseringsgrad

Hur informationsutbytet sker mellan de olika delsystemen påverkas av graden av formalisering i språket. Formaliseringen i språket påverkas av begreppsinformationen och hur denna uttrycks. Ökad formalisering minskar redundant information, dvs. exakt formulering av det efterfrågade med den korrekta benämningen resulterar i leverans av korrekt information och inget informationsutbyte har varit överflödigt. En alltför formell förfrågan om information kan innebära en svår tolkningsprocess. Det är således inte lätt att finna en lämplig formaliseringsgrad. Formalisering innebär även att irrelevant information ska filtreras bort när beslut ska fattas.

4.4.3 Användningsfrekvens

Användningstillfälle av information bestäms när beslut och handlingar initieras. Det finns dock inga allmänna principer för detta eftersom användning av information är en situationsspecifik faktor. Vissa situationer kräver att information om en händelse omedelbart måste förmedlas till respektive ansvarige. Andra situationer kräver att informationen förvaltas på ett lämpligt sätt innan den distribueras till de olika ansvariga vid bestämda tidsintervall.

4.4.4 Behandlingsregler

Informationsbehandlingen sker i enlighet med valideringsregler, integrationsregler och härledningsregler. Dessa regler, liksom informationen, kan ha lokal, translokal eller global karaktär beroende på var i verksamheten dessa ska råda. Detta bör fastställas genom förhandlingar mellan olika verksamhetsdelar eftersom reglernas karaktär kommer ha stor påverkan på verksamhetsdelarnas handlingsfrihet.

4.4.5 Redundans och tillgänglighet

Redundant, dubbellagrad information är oftast en oönskad egenskap vid utformning av informationssystem. Detta anses skapa inkonsistens och försvårar effektiv hantering av information. Inom vissa arkitekturfilosofier motiveras dock redundans[6]. Redundant information möjliggör att information om andra verksamhetsdelar kan finnas tillgängligt inom en verksamhets egen informationsbas. Denna verksamhetsdel blir då oberoende av direkt tidskritisk kommunikation med övriga verksamhetsdelar just när informationen krävs.

4.4.6 Krosskonsistens

Om man baserar systemet på redundant information krävs en speciell förvaltningsstrategi för att säkerställa dess krosskonsistens (eng. cross consistence), med andra ord att kunna garantera att en viss information i ett delsystems informationsbas är densamma som i ett annat delsystems informationsbas. Samma fråga till olika delsystem måste alltså resultera i samma svar. Det finns dock situationer där krosskonsistens inte är så viktigt. Det kan ibland inträffa att olika information i de olika delsystemen är korrekt ur en verksamhetssynpunkt.

För att underlätta diskussioner och ställningstaganden angående dessa funktionella egenskaper kan det vara lämpligt att basera informationssystemarkitekturen på en viss arkitekturfilosofi. Arkitekturfilosofier kan vägleda och underlätta förståelsen varför informationssystemarkitekturen ska utformas och förvaltas på ett visst sätt. De grundläggande arkitekturfilosofierna beskrivs i följande avsnitt.

[6]Ordet finns förklarat i ordlistan.

5 Arkitekturfilosofier

Eftersom det ofta saknas en plan över hur en organisations samlade informationsbehandling ska struktureras i system och hur dessa system ska samverka och kommunicera, vilket kan leda till bristande systemstrukturer, är det lämpligt att basera informationssystemarkitekturen på en arkitektursfilosofi. Det existerar ett antal olika filosofier. *Information Resource Management (IRM)* och *Verksamhetsbaserad Systemstrukturering (VBS)* är två filosofier som kan ses som varandras motsatser [Axe98b]. Vi kommer här att ta upp dessa två och även kortfattat några andra filosofier som bygger på dem.

Enligt Magoulas et al består ett informationssystem av tre komponenter, en *informationsprocessor*, ett *konceptuellt schema* och en *informationsbas*. Dessa komponenter struktureras lite olika beroende på vilken filosofi som tillämpas [Mag91].

5.1 Information Resource Management (IRM)

IRM är en datadriven filosofi, vilket innebär att informationen som finns i verksamheten ses som en resurs precis som maskiner, arbetskraft eller kapital. Inom IRM anses data och datastrukturer vara grunden för informationssystemet. Datastrukturen anses dessutom vara stabil över tiden och förutsatt att verksamheten inte helt ändrats ska alltså, vad som än händer, vissa objekt och begrepp finnas kvar. Stabila objekt kan t.ex. vara kund, faktura och produkter [Axe98b].

Enligt Axelsson innebär IRM-filosofin att informationen ska [Axe98]:

- planeras med hjälp av datamodellering.
- anskaffas endast en gång och då vid källan, dvs där informationen ursprungligen finns. Till exempel vid mötet med kunden.
- lagras så att alla inom verksamheten kan få tillgång till informationen.

Denna syn på information kräver att informationen styrs och administreras på samma sätt som alla andra verksamhetsresurser styrs och administreras [Mag91].

5.1.1 Datamodellering

Vid skapande av informationssystem enligt IRM-filosofin identifieras de stabila objekt som existerar i verksamheten och relateras sedan till varandra

med hjälp av datamodellering. En modell av verkligheten skapas alltså där olika objekt i datamodellen motsvarar verkliga objekt. Då dessa objekt ska vara stabila är det bara värdet på dem som får förändras [Axe98b]. Till exempel är kund ett stabilt objekt som inte får försvinna men som får anta olika värden då kunder tillkommer eller försvinner.

Då informationssystem i vissa avseenden ska ersätta muntlig kommunikation är det viktigt att begreppen som existerar inte är för formella för att kunna förstås av användarna, men heller inte tvivelaktiga. IRM-filosofin eftersträvar därför minimering av antalet begrepp som finns i verksamheten genom t.ex. klargöra förekomsten av synonymer [Axe98b].

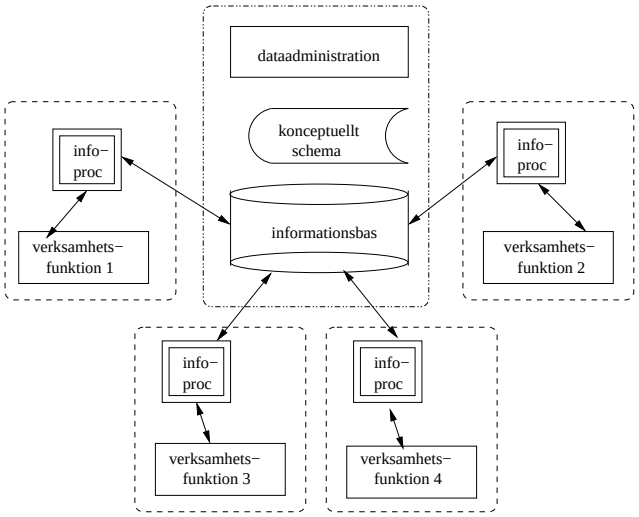
5.1.2 Anskaffning av data

För att undvika redundans tillåts information endast anskaffas en gång och då i den del av verksamheten som har ansvar för denna information [Axe98b]. Förekomst av redundans medför ökade kostnader och är dessutom tidsödande vid förändring av data, eftersom datat då måste förändras på flera ställen i databasen. Det kan även medföra att datat blir inkonsistent om förändring inte skett vid alla förekomster av en viss data. Detta fenomen gör att kvaliteten på data blir dålig. Redundans kan dessutom leda till brist på lagringsutrymme eftersom mängden data blir större [Elm00].

5.1.3 Datalagringen

Datamodellen realiseras i *en* databas som är skild från de applikationer som används för presentation av informationen. Datat i databasen struktureras efter de stabila objekten och inte efter användarnas informationsbehov, eftersom informationsbehovet anses förändras med tiden medan objekten är stabila. Databasen och applikationerna ska vara skilda från varandra för att skapa så stort programoberoende som möjligt. Detta synsätt gör att databasen blir tillräckligt robust för att vara oberoende av hur informationen ska användas och för förändringar i verksamheten. Alla eventuella framtida förändringar bör endast behöva göras på applikationerna [Axe98b].

Idealet inom IRM-filosofin är att det endast ska existera *en* datakälla att presentera information från och att all information dessutom ska vara tillgänglig för alla inom verksamheten. Ansvaret för de applikationer som presenterar informationen kan fördelas över de organisatoriska enheterna och varje enhet kan därmed själv bestämma vilken information som ska presenteras från databasen [Mag91].



Figur 2: IRM [Mag91]

Figur 2 visar tydligt IRM-filosofins grundläggande koncept med en enda central datakälla varifrån alla fyra verksamhetsfunktioner hämtar information via egna informationsprocesser. Även det konceptuella schemat ligger centralt.

5.1.4 Fördelar med IRM

En av de viktigaste målen med IRM är att skapa en stabil datalagring så att den inte behöver struktureras om vid omvärldsförändringar. En annan viktig fråga är globalisering av den lagrade informationen. Globalisering av information innebär att man gör informationen tillgänglig för alla, konsistent och fri från logiska motsägelser. Informationen får naturligtvis inte vara motstridig så att t.ex. en person med en jämn siffra näst sist i personnumret är en man eller att det finns två olika namn kopplade till ett personnummer. Genom denna globalisering av data kan kostnader minska på grund av minskade underhållskostnader, längre livslängd och ökad pålitlighet.

Den gemensamma datastrukturen leder till mindre antal anskaffningar av information och reduceringen av begrepp leder till snabbare och effektivare utvecklingsarbete.

Förändringar som rör t.ex. användargränsnitt eller listors utseende genomförs lätt eftersom applikationer och data ska vara väl avgränsade från varandra.

Genom en datadriven ansats kan förekomst av duplicerad data undvikas, vilket medför effektivare anskaffning, lagring och överföring av data. IRM-filosofin, med sin enda datakälla, underlättar även administrationen vilket kan motverka att information hålls lagrad längre än nödvändigt [Axe98b].

5.1.5 Nackdelar med IRM

Trots att reduceringen av begrepp kan effektivisera arbetet kan de kvarvarande begreppen upplevas som onaturliga och svårförståeliga av användarna. Detta kan bero på att det inte finns ett gemensamt språkbruk för verksamhetens manuella rutiner och begreppen kan då ha olika betydelse för olika personer.

En ytterligare nackdel är att då målet är en generell datastruktur för att undvika framtida förändringar kan en alltför generell datastruktur medföra problem gällande att alla informationsbehov inte kan tillgodoses. Användarna kan då tvingas utföra vissa arbetsuppgifter vid sidan av systemet och därmed skapas dubbelarbete och dubbellagring av information [Axe98b].

5.2 Verksamhetsbaserad Systemstrukturering (VBS)

VBS är en annan typ av informationssystemstrukturering som innebär att man avgränsar verksamhetsbaserade informationssystem med decentraliserat ansvarstagande [Mag91].

En av de viktigaste frågorna inom VBS är ansvaret för informationssystemen och för att uppnå decentraliserat ansvarstagande lokaliseras informationssystemen ansvarsmässigt till en specifik organisatorisk enhet. Uppdelningen överensstämmer inte alltid med de existerande organisatoriska gränserna mellan avdelningar utan grundas på hur ansvarsstrukturen ser ut. En *verksamhetsfunktion* är i detta fall en väl avgränsad del av verksamheten där det finns en bestämd uppgift att fylla och även ett ansvar för att uppgifter verkligen genomförs. För att klara av detta förfogar verksamhetsfunktionen över vissa resurser och hit hör då även det egna informationssystemet [Axe98b].

Informationen inom VBS kan delas upp i två kategorier, dels den lokala informationen som går mellan informationssystemet och lokala användare och dels sambandinformationen som går mellan informationssystemen [Mag91].

5.2.1 Oberoende

VBS strävar efter att undvika stordatordrift, centralisering och globalisering av informationsresurser. Detta synsätt medför flexibla system och situationer där en verksamhetsfunktion är beroende av att andra verksamhetsfunktioners informationssystem fungerar som de ska kan undvikas [Axe98]. VBS-filosofin fokuserar på flera olika typer av oberoende. Oberoendet kan, enligt Axelsson vara [Axe98]:

- **Funktionellt** - verksamhetsfunktionen kan själv avgöra hur informationen ska insamlas, lagras, bearbetas och presenteras. Hög grad av funktionellt oberoende har uppnåtts om verksamhetsfunktionen kan förändra den lokala informationen utan att någon annan verksamhetsfunktion berörs av detta.
- **Tidsmässigt** - varje informationssystem kan fungera utan andra informationssystem.
- **Tekniskt** - informationssystemen kan frikopplas logiskt och utrustning kan bytas ut utan att andra informationssystem påverkas av detta.

5.2.2 Meddelandesamverkan

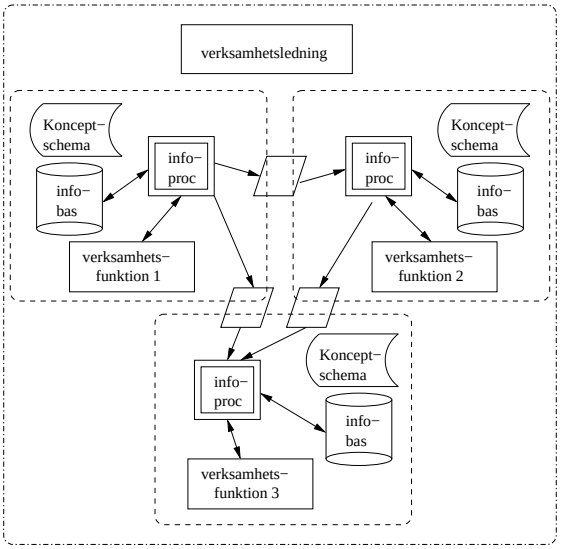
Varje informationssystem har sin lokala data samlad i en lokal databas. Trots att de olika informationssystemen ska vara så oberoende som möjligt ska de även vara samverkande och kommunicera med varandra vid behov. Denna form av kommunikation kallas för meddelandesamverkan.

Ett meddelande kan inom VBS inte uppdateras och lagras heller inte i någon databas under överföringen, men det kan buffras i en kö. Varje meddelande överförs bara en gång och har alltid ett mottagande informationssystem. Ett meddelande kan både begäras av det mottagande informationssystemet och initieras av det avsändande informationssystemet. Överföringen kan skötas av det mottagande informationssystemet, av det avsändande informationssystemet eller av ett speciellt meddelandehanteringssystem [Axe98b].

Figur 3 visar VBS-filosofins grundläggande koncept med lokala informationsbaser och konceptuella scheman.

5.2.3 Fördelar med VBS?

Målet med VBS är att undvika stordatordrift, centralisering och globalisering av informationsresurser. Det är viktigt att uppnå oberoende mellan de



Figur 3: Ideal VBS [Mag91]

olika informationssystemen för att kunna skapa en mer flexibel verksamhetsstruktur. Detta medför att anpassning och förändring blir relativt enkel och effektiv. Fler positiva effekter är att systemen blir lätta att överblicka och att de negativa konsekvenserna vid störning inom en del av de samlade informationssystemen minimeras. Tunga förändrings- och förvaltningsprocesser kan också undvikas.

Då varje verksamhetsfunktion har egna informationssystem undviks problemet med att begreppen kan uppfattas onaturliga och svårförståeliga. Verksamhetsfunktionen språkbruk kan i regel bevaras och användas i systemet [Axe98b].

5.2.4 Nackdelar med VBS

En grundsten i VBS är att strukturering tar utgångspunkt i ansvarsfrågan. Detta kan dock vara olämpligt. Ansvarsfrågor är viktiga men kanske inte lämpliga att styra indelningen och avgränsningen av informationssystem.

Inom VBS sker kommunikationen mellan informationssystemen med meddelandesamverkan. Detta kan dock upplevas om komplicerat av användarna och det kan finnas en stor osäkerhet hos användarna kring hur kommunikationen sker och huruvida ett meddelande kommit fram eller inte.

Trots att det oftast upplevs som positivt för användarna att språkbruket bevaras finns det en stor svaghet i denna filosofi. De olika delsystemen måste förstå varandra och om en term har olika betydelse i olika system uppstår lätt problem och missförstånd. Användarna kan uppleva det som att informationen som sänds till dem är felaktig när det egentligen är språkbruket som skiljer sig mellan systemen [Axe98b].

5.3 Andra arkitekturfilosofier

Ett exempel på en komponentbaserad arkitekturfilosofi är Process-, Aktivitets-, och Komponentbaserad Systemstrukturering (PAKS) som har formulerats under ett forskningsprojekt vid namn STRIKE (Strukturering av informationssystem och verksamheter - utvärdering och förändring). STRIKE startade 1994 vid Linköpings Universitet och har bland annat utvecklat teori om informationssystemarkitekturer och integration i samspel med verksamheter. En annan arkitekturfilosofi är den inter-organisatoriska arkitekturfilosofin som bygger på VBS. Båda filosofierna presenteras kortfattat nedan.

5.3.1 Process-, Aktivitets-, och Komponentbaserad Systemstrukturering (PAKS)

PAKS är en filosofi som har formulerats på grund av de problem som uppmärksammats hos IRM och VBS. IRM och VBS kan ses som två rivaliserande filosofier och PAKS som en kompromiss eller syntes av dessa. När PAKS formulerades försökte man inom STRIKE förena de positiva egenskaperna hos IRM och VBS och undvika de negativa egenskaperna.

Koppling till IRM och VBS
PAKS har från IRM influerats av främst konceptet datamodellering. Definitioner och avgränsningar av verksamhetsanknutna begrepp är en viktig del när det gäller strukturering av informationssystem. Genom att IRM är en datadriven filosofi finns möjligheten att göra just detta genom datamodellering. Datamodelleringen kan även medföra en genomtänkt strukturering av data [Axe98b].

En ansats inom IRM som man velat undvika vid formulering av PAKS är konceptet med integrerade system och databaser. Global datalagring medför för stora problem när det gäller förändringar och ansvar för datat. Axelsson och Goldkuhl menar att det kan vara lämpligt att ha central information lagrad i en global databas men att det även bör finnas lokala databaser för

data som endast är viktig för vissa delar av verksamheten [Axe98b].

En av VBS grundprinciper är att strukturera systemet i mindre autonoma, samverkande system. Denna princip togs till vara av STRIKE och även med i PAKS. VBS syn på ansvarsallokering, där ansvaret för informationssystemet ska vara klarlagt och förlagt till verksamhetssidan, är också en princip som används i PAKS [Axe98b].

Förutom influenserna från IRM och VBS består PAKS av tre grundtankar [Mag91]:

Processtänkande. Vid procestänkande läggs fokus på "vad som görs" istället för "vem som bestämmer" som fallet är vid funktionellt tänkande. Processer kan liknas vid flöden av aktiviteter och skär ofta organisatoriska gränser vilket kan leda till att en process saknar en ansvarig. En process delas ofta upp i ett antal sekvensiella delprocesser. PAKS strävar efter en processorienterad syn på verksamheten och låter arbetsuppgifter och önskade resultat styra systemstruktureringen istället för att låta ansvarsförhållanden styra struktureringen vilket fallet är inom VBS.

Aktivitets och kommunikativa tänkandet. Här kan den tidigare nämnda talaktsteorin användas, d.v.s. synen på informationssystem där själva talet (kommunikationen) är en handling och där kommunikation är en väldigt viktig aspekt att ha i åtanke. Informationssystem innebär då att olika kommunikativa handlingar utförs.

Komponent och objektorientering. Vikt läggs på inkapsling av data och att hålla samman data och dess funktionalitet (objekt).

5.3.2 Inter-organisatoriska informationssystem

Med inter-organisatoriska informationssystem menas att informationssystem i olika organisationer samverkar med varandra. Att organisationer samverkar med varandra är inte en ny företeelse, däremot är det nytt att denna samverkan grundas på organisationernas informationssystem [Mag91].

Den inter-organisatoriska arkitekturfilosofin kan ses som en utvidgning av den verksamhetsbaserade då inblandade organisationer har stor handlingsfrihet och är relativt oberoende av de andra organisationernas informationssystem. Det existerar dock en del nya problem. Till exempel kan ansvarsförhållanden inte skötas efter samma principer som vid inom-organisatoriska arkitektur-

filosofier eftersom informationssystemen rör fler än en organistation. Detta gör att det inte existerar någon egentlig ledningsfunktion och det blir väldigt svårt att tala om systemansvar och systemägarskap. Det existerar i regel heller ingen gemensam målsättning att arbeta mot utan istället ett gemensamt problem som de berörda organisationerna är intresserade av att lösa [Mag91].

6 Systemarkitektur

Varje system har en systemarkitektur, implicit eller explicit. I små system kan systemarkitekturen vara självklar och inga avvägningar har behövts göras i utformningen av systemarkitekturen. Skillnad är om systemet är stort och komplext, som ett informationssystem, då det inte finns någon given bästa lösning av systemarkitekturen. De som utformar systemarkitekturen måste då försöka utforma en systemarkitektur som så bra som möjligt kan uppfylla verksamhetens krav på systemet. Systemarkitekturen ska, som vi tidigare förklarat, vara en teknisk realisering av verksamhetsmodellen. Systemarkitekturen ska enligt Krutchen presentera följande [Kru01]:

- komponenter som motsvarar de komponenter som presenterats i verksamhetsmodellen
- arkitekturella mekanismer som ger komponenterna dess önskade beteenden, så som mekanismer för kommunikation
- arkitekturmönster och ramverk som använts
- olika lager som systemarkitekturen består av
- olika delsystem som systemet består av
- gränssnitt som finns i lösningen, mellan komponenter och mellan delsystemen
- huvudprocesser och kontrollflöden som systemarkitekturen består av

I utformningen av systemarkitekturen krävs det ofta att man även beaktar faktorer så som ledtid (time-to-market) eller andra ekonomiska faktorer, utöver de funktionella kraven. Att leverera ett färdigt system inom en viss tid kan ibland vara viktigare än att nå den optimala systemlösningen. Utformningen av den bäst lämpade systemarkitekturen kan med andra ord begränsas av ett antal icke tekniska faktorer. En olämplig systemarkitektur kan dock, i systemutvecklingssammanhang, få katastrofala följder [Com01]. Det är med andra ord oerhört viktigt att göra korrekta avvägningar.

För att hitta lämpliga lösningar krävs eftertanke och erfarenhet. Man måste inventera och undersöka olika lösningsalternativ. De fysiska begränsningar som råder, t.ex. processorkapacitet, måste spela in i avgörandet. De motstridiga krav som finns på systemet måste analyseras, det kan vara motstridigheter som pris och utvecklingstid kontra den önskade funktionaliteten eller nödvändig prestanda kontra strömförbrukning och pris [Com01]. En

vald lösning kanske satisfierar vissa krav men på bekostnad av andra. Det är därför viktigt att ha någon form av prioritet och balans på kraven.

Enligt Garlan och Perry kan en välformulerad och passande systemarkitektur, förutom att möjliggöra en lyckad realisering av systemet, ha positivt inflytande på andra aspekter inom systemutvecklingen, tex. inom följande [Com01]:

Förståelse. Systemarkitektur gör det lättare för alla inblandade att förstå stora system genom att presentera dessa på en abstraktionsnivå så att systemets högnivåkonstruktion blir begriplig.

Återanvändning. Systemarkitekturen kan stödja återanvändning av så väl komponenter som ramverk. Referensarkitekturer, domänspecifika arkitekturer och product-line arkitekturer är bevis på detta. Dessa beskrivs senare i detta avsnitt.

Systemevolution. Systemarkitekturen kan ge indikationer eller förmaningar om hur systemet bör vidareutvecklas i framtiden. Genom att ge ett system tydliga riktlinjer och avgränsningar kan de som underhåller systemet lättare förstå vilka följder en ändring i systemet skulle innebära. Detta i sin tur leder till att kostnader i och med förändringar i systemet lättare kan uppskattas.

Analys. Systemarkitekturen skapar möjlighet till olika typer av analyser, så som systemarkitekturens överensstämmelse med den valda arkitekturella stilen eller hur den valda systemarkitekturen överensstämmer med de önskade kvalitetsattributen.

Systemarkitekturen ger enligt Philippe Krutchen även en bra grund för vidare planering av konstruktionen av systemet. Tidsplaneringen och resursplanering i form av personal kan organiseras kring de komponenter som presenteras i systemarkitekturen. De fundamentala arkitekturella besluten tas av en liten arkitektgrupp och den vidare konstruktionen av systemet kan därefter fördelas på ett antal utvecklingsgrupper, som var och en ansvarar för utvecklingen av en eller flera delar av systemet. Dessa grupper ska då förstå vilka gränssnitt de ska arbeta mot [Kru01].

6.1 Mjukvaruarkitektur

Mjukvaruarkitekturen är enligt Bass, Clements och Kazman strukturen eller strukturerna av ett system, bestående av mjukvarukomponenter, deras ex-

ternt synliga egenskaper och relationerna mellan dem. Avsikten med denna definition är att mjukvaruarkitekturen måste abstrahera bort viss information om systemet men ändå tillhandahålla tillräckligt mycket information om systemet så att arkitekturen kan vara en bas för vidare analyser, beslutsfattande och minimering av risker [Bas98].

Den färdiga mjukvaruarkitekturen kan enligt, Hofmeister, Nord och Soni ses som den designplan som ska styra implementationen av systemet och dess olika faser, såsom design, kodning, integrering och testning. Precis som Bass et al anser de att mjukvaruarkitekturen är en abstraktion av systemet. De enskilda detaljer som behövs för att implementera systemet abstraheras bort och inkapslas i ett visst element i arkitekturen. Hofmeister et al förklarar vidare att mjukvaruarkitekturen ska definiera och beskriva elementen i systemet på relativt hög nivå. Den ska beskriva hur elementen satisfierar systemkraven, vilket element som ansvarar för vilken funktionalitet, hur elementen interagerar med varandra och externa enheter och deras beroende av den underliggande exekveringsplatformen. Många icke-funktionella krav måste övervägas vid utformningen och förfiningen av mjukvaruarkitekturen. Som exempel måste hänsyn tas till krav på systemets portabilitet[7] och modifierbarhet[8] eller själva byggandet av systemet med dess tillhörande tester.

Det råder dock tvister om innebörden av begreppet mjukvaruarkitektur och vilka enskilda designdetaljer som tillhör eller inte tillhör den. Hofmeister et al anser att det finns en mängd detaljer som kan vara viktiga för både arkitekturen och för implementationen, t.ex. vissa klasstrukturer och algoritmer. Dessa är viktiga för arkitekturen om de begränsar de arkitekturella elementen eller om de beskriver hur elementen interagerar med varandra, externa enheter eller den underliggande exekveringsplatformen. Algoritmer och klasstrukturer som finns inom ett element anser de inte är viktiga för mjukvaruarkitekturen. Dessa bör utvecklas och beskrivas under designfasen [Hof00].

6.2 Principer och riktlinjer

I inledningen av arbetet med systemarkitekturen är det lämpligt att formulera ett antal principer eller riktlinjer som systemarkitekturen ska bygga på. Dessa principer ska tillgodose den verksamhetmodell och de verksamhetsmässiga krav som tidigare arbetats fram i verksamhetsanalysen. Principerna ska fungera som vägledning i den vidare utformningen av systemarki-

[7] Ordet finns förklarat i ordlistan.
[8] Ordet finns förklarat i ordlistan.

tekturen och dess delarkitekturer.

6.2.1 Formulering

En bra uppsättning principer är en bra grund för de arkitekturella beslut och avvägningar som kommer att behöva göras. Det finns enligt The Open Group, fem kriterier som karaktäriserar en mängd bra principer [TOG03]:

Begriplighet. Den underliggande tanken kan snabbt förstås. Avsikten med principerna är tydlig och otvetydig så att de ej kan missförstås, avsiktligt eller ej.

Kraftfull. Principerna möjliggör beslut av hög kvalitet i arkitekturella frågor. Varje princip ska vara tillräckligt definitiv och noggrann för att medföra konsekventa beslut i komplicerade situationer.

Fullständighet. Varje möjlig princip, som kan vägleda hantering av information och teknologi i organisationen, ska vara definierad. Principerna ska omfatta alla möjliga situationer.

Konsistens. De olika principerna måste vara formulerade på ett sätt som medför en bra balans av tolkningar av dessa. Principerna ska vara inte motstridiga så att uppfyllnad (eng. adherence) av en princip bryter mot innebörden av en annan.

Stabilitet. Principerna ska vara stabila men ändå kunna anpassas till förändringar. En förändringsprocess av principerna ska lägga till, ta bort eller byta ut principerna efter att dessa är färdigställda.

6.2.2 Format

Det kan vara lämpligt att skapa ett standardformulär för att beskriva en princip. TOGAF[9] innehåller en mall för formulering av arkitekturella principer. De anser att varje princip ska innehålla följande [TOG03]:

- **Name** (Namn) - ska både beskriva innebörden av principen men även vara lätt att komma ihåg
- **Statement** (Förklaring) - en entydig förklaring

[9] *The Open Goup Architecture Framework*, beskrivs i ett senare avsnitt i denna rapport.

- **Rationale** (Logisk grund) - verksamhetsmässig beskrivning av vad verksamheten tjänar genom att upprätthålla aktuell princip

- **Implications** (Innebörd) - ska beskriva de tekniska och verksamhetsmässiga kraven för att denna princip ska uppfyllas med avseende på kostnad, resurser och aktiviteter

6.3 Metaarkitektur

Metaarkitektur är ett samlingsnamn för de övergripande inledande arkitekturella besluten som ska fungera som ett ramverk för det fortsatta arkitekturella arbetet och den följande systemutvecklingen. Metaarkitekturen beskriver den arkitekturella visionen av systemet i form av principer, riktlinjer, huvudkoncept och övergripande arkitekturella stilar [Bre02]. Dessa högnivåbeslut kommer att ha stark inverkan på det mer ingående strukturarbetet. Metaarkitekturen kan t.ex. innehålla en beskrivning av egenskaper för varje typ av komponent som senare ska implementeras, regler för hur varje typ av komponent ska interagera med andra typer av komponenter [Hpi00].

En metaarkitektur kan arbetas fram i början av systemutvecklingsprojektet eller så finns ett redan utarbetat arkitekturellt ramverk som ska efterföljas. Vi kommer nedan ta upp tre olika sådana typer av arkitekturella ramverk.

6.3.1 Domänspecifik arkitektur

En domänspecifik arkitektur definieras, enligt Gomaa och Farrukh, som en arkitektur för en familj av system. Arkitekturen beskriver sammansättningen av komponenter och deras inbördes förhållanden. Eftersom arkitekturen måste kunna appliceras på en mängd system inom den valda domänen bör den vara både generell och flexibel [Gom99].

En *domänmodell* definierar koncept och principer för en domän. Domänmodellen tas fram vid en *domänanalys* och står sedan som grund vid skapandet av den domänspecifika mjukvaruarkitekturen [Gom99]. Vid en domänanalys identifieras komponenter och hur de interagerar [Hof00].

6.3.2 Product-line-arkitektur

För en organisation är en arkitektur en stor investering både tidsmässigt och kostnadsmässigt. Ett sätt att maximera nyttan och att minska kostnaderna för denna investering är att producera flera liknande system och återanvända

arkitekturen för dessa [Bas98]. Detta är en product-line och med detta menas en arkitektur som appliceras på mängd produkter inom en organisation. Fördelen med en procuct-line-arkitektur är att en mängd liknande produkter kan utvecklas mer kostnadseffektivt eftersom de har liknande design och till och med använder samma moduler [Hof00].

En product-line-arkitektur definierar, precis om domänspecifik arkitektur, komponenter, hur de interagerar och deras funktionalitet. I vissa fall definieras vissa instanser av arkitekturen, t.ex. felmeddelanden och andra moduler som kan delas av produkterna [Hof00].

När en organisation ska börja utveckla produkter enligt en product-line-arkitektur måste först en mängd egenskaper, kärnegenskaper, som alla produkter ska ha utvecklas. Ett problem är när ny funktionalitet måste adderas till en produkt. Om denna funktionalitet ligger inom product-line-området kan produkten enkelt förändras, men om den ligger utanför måste val göras. Antingen måste produkten plockas bort från product-line-arkitekturen eller så måste kärnegenskaperna uppdateras att innehålla denna nya funktion. Ett annat problem är vad som ska göras med gamla produkter när kärnegenskaperna utvecklas. Antingen måste de gamla produkterna också utvecklas att innehålla ny funktionalitet, vilket är dyrt och tidskrävande, eller så kan dessa lämnas som de är, med risken att de saknar någon ny viktig funktion [Bas98].

Product-line-arkitekturen är bra ur återanvändbarhetssynpunkt. Några exempel där återanvändbarheten ökar med procuct-line-arkitektur är att komponenter kan användas i flera olika produkter, att bra designval kan tas till vara och att dåliga val kan undvikas. Det är dessutom lättare att förflytta personal mellan utvecklingsprojekt eftersom applikationerna liknar varandra. Även projektplaneringen blir lättare eftersom erfarenhet från tidigare projekt lätt kan appliceras på senare liknande projekt [Bas98].

6.3.3 Referensarkitektur

Med en product-line-arkitektur kan arkitektur återanvändas inom en organisation. Referensarkitektur medför att arkitektur kan delas och återanvändas mellan olika organisationer. Det är en riktlinje för att utveckla arkitekturen för ett system inom en viss domän och alltså inte en arkitektur i sig utan ett användbart steg mot en bra arkitektur. En referensarkitektur kan även vara en domänspecifik arkitektur.

En referensarkitektur ska, enligt Krzysztof Czarnecki, innehålla följande [Cza97]:

- En arkitekturell stil som definierar typ av komponenter (t.ex. filter eller objekt) och vad som ska koppla ihop komponenterna (till exempel pipor eller procedursanrop).
- Typiska komponenter och deras typiska varianter.
- Specifikation av gränssnitt mot externa enheter och system.

Arkitekturella stilar förklaras i följande avsnitt.

7 Arkitekturella stilar och mönster

I den traditionella industrin har arkitekturella stilar använts länge. Det är viktigt att det existerar en väl etablerad förståelse för vad alla figurer och termer betyder och hur dessa får användas när någonting ska designas [Sha96b]. En arkitekturell stil inom mjukvaruindustrin är på många vis jämförbar med arkitekturella stilar hos t.ex. byggnader. Den består av några grunddrag och regler för att kombinera dessa grunddrag. En stil är alltså inte en arkitektur mer än termen *gotisk* bestämmer hur en byggnad ska se ut [Bas98].

Inom mjukvaruindustrin existerar både arkitekturella stilar och arkitekturella mönster som två namn på samma sak. Många (Hofmeister et al [Hof00], Bass et al [Bas98] och Shaw et al [Sha96]) har valt att använda namnet stilar istället för mönster för att särskilja det från designmönster och kodmönster.

Enligt Shaw och Clements är en arkitekturell stil en mängd designregler som identifierar typen av *komponenter* och *kopplare* (eng. connector) som används för att bygga ett system eller delsystem [Sha96]. En komponent är en icke trivial, nästan oberoende, och utbytbar del av ett system som uppfyller en funktion i en arkitektur. En kopplare är en mekanism som förmedlar meddelanden mellan komponenter [Kru01]. Att använda komponenter tillsammans med kopplare är ett vanligt sätt att beskriva arkitekturer [Sha96b].

De mest kända arkitekturella stilarna kan, enligt Shaw och Garlan, kategoriseras enligt nedan [Sha96b]:

- **Dataflöden**
  Pipe and filter
  Batchsekvens
- **Oberoende komponenter**
  Kommunicerande processer
  Händelsebaserade system
- **Call-and-return system**
  Objektorienterade system
  Hierarkiska lager
  Huvudprogram och subrutiner
- **Datacentrerade arkitekturer**
  Databaser
  Blackboards
  Hypertextsystem

- **Virtuella maskiner**
  Interpreters
  Regelbaserade system

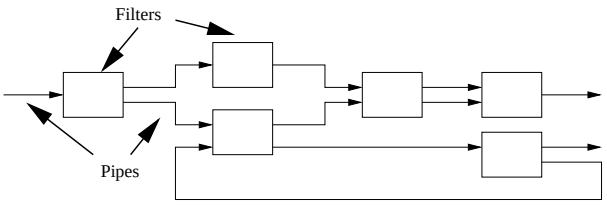
7.1 Dataflöden

Ett system som byggts med denna arkitekturella stil kan ses som en serie av omvandlingar av data. Data kommer in i systemet, flödar igenom komponenterna tills det omvandlas till utdata eller lagras. Komponenterna opererar på stora kontinuerligt tillgängliga dataflöden. Målet med dataflödesstilen är hög kvalitet genom återanvändning och modifierbarhet. Modifiering kan dock vara svårt eftersom komponenterna styr sig själva och interagerar med varandra utan begränsningar från systemet [Bas98].

7.1.1 Pipe and filter

Varje komponent har en mängd indata och en mängd utdata. En komponent läser dataströmmar på sin ”input” och producerar dataströmmar på sin ”output”. Eftersom en lokal omvandlare utför beräkningar på ”inputströmmen” och genererar utdata innan all indata är avläst kallas komponenterna för *filters*. Kopplarna fungerar som ledningar eller kanaler för dataströmmarna och leder dem från ”output” på en komponent till ”input” på en annan och kallas för *pipes*.

Filterna delar inte tillstånd med något annat filter utan är helt oberoende och vet heller inte om vilket filter som tar emot dess utdata eller vilket filter som producerar dess indata [Sha96b]. Denna stil illustreras med figur 4.



Figur 4: Exempel på stilen pipe and filter [Sha96b]

7.1.2 Batchsekvens

I batchsekvensstilen är alla komponenter oberoende program och varje komponent ska ha kört klart innan data skickas till nästa komponent [Bas98]. I denna stil är alla filter ordnade i en linjär sekvens hopkopplade av typade pipor, vilket innebär att all data som skickas mellan filterna måste vara av en väl definierad datatyp [Sha96b].

7.2 Oberoende komponenter

Ett system baserat på denna arkitekturella stil består av ett antal oberoende komponenter som kommunicerar med hjälp av meddelanden. Målet med denna stil är hög modifierbarhet genom att göra delar av beräkningarna oberoende av varandra. Komponenterna skickar alltså meddelanden men har ingen form av kontroll över varandra. Meddelanden kan skickas till en speciell komponent eller till alla samtidigt [Bas98].

7.2.1 Kommunicerande processer

Ett exempel på kommunicerande processer är ett klient-server-system. I ett klient-server-system representerar en server processen som tillhandahåller tjänster till andra processer (klienter). Servern och klienterna är oftast oberoende processer som körs parallellt och ofta distribuerade. En server vet oftast inte om identiteten på klienterna eller hur många klienter som kommer att kommunicera med servern eftersom det är klienterna som söker upp en server och försöker få tillgång till serverns data [Bas98].

7.2.2 Händelsebaserade system

Idén bakom händelsebaserade system är att en komponent kan annonsera en eller flera händelser. En komponent i ett händelsebaserat system är en modul med ett interface som tillhandahåller procedurer och händelser. Procedurer kan anropas på vanligt vis men komponenter kan även registrera intresse i en viss händelse genom att associera en eller flera procedurer med den händelsen. Detta medför att de procedurerna körs när händelsen inträffar. En händelse kan alltså implicit invokera procedurer i andra moduler [Sha96b].

7.3 Call-and-return system

Call-and-return-arkitekturen är idag den mest vanliga arkitekturella stilen. Den innebär att en procedur anropar en annan och ligger och väntar tills den får svar av den anropade. När svar kommit kan den anropande proceduren

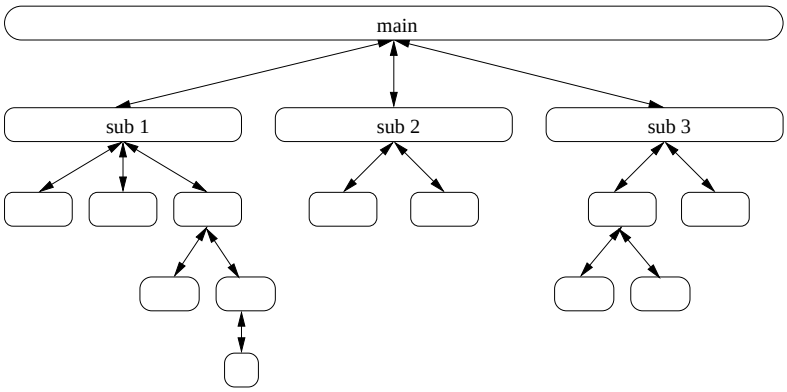
fortsätta. Målet med denna stil är att nå hög modifierbarhet och skalbarhet[10] [Bas98].

7.3.1 Objektorienterade system

I system baserade på den objektorienterade stilen inkapslas datarepsentationen och motsvarande operationer i abstrakta datatyper kallade objekt. Komponenterna i denna stil är objekten och är, enligt Shaw et al, av typen *manager* eftersom de är ansvariga för att bibehålla integriteten hos datarepresentationen. De viktigaste aspekterna inom denna stil är just bibehållandet av integritet och att datarepresentationen ska hållas dold för andra objekt [Sha96b].

7.3.2 Hierarkiska lager

I ett system bestående av hierarkiska lager tillhandahåller varje lager tjänster till lagret ovanför och fungerar som en klient för lagret nedanför. I vissa hierarkiska system är alla inre lager, förutom vissa valda funktioner, helt gömda för alla lager utom det närmast yttre lagret [Sha96b].



Figur 5: Huvudprogram och subrutin-stilen [Bas98]

7.3.3 Huvudprogram och subrutiner

Huvudprogram och subrutiner är den klassiska programmeringsparadigmen. Målet är att dela upp ett program i mindre delar för att öka modifierbarheten. Programmen delar upp hierarkiskt där den komponent som ligger över i

[10] Ordet finns förklarat i ordlistan.

hierakin är ”förälder” till de som ligger nedanför. Det är oftast bara en komponent i taget som kan exekvera och ”exekveringsrätten” får en komponent av sin förälder [Bas98]. Denna stil illustreras med figur 5.

7.4 Datacentrerade arkitekturer

Den datacentrerade arkitekturella stilen innebär att bygga system med fullt åtkomlig data och möjlighet till uppdatering av denna för alla komponenter. Alla komponenter delar samma data. Målet med denna stil är hög kvalitet vid integration av data [Bas98].

7.4.1 Databas

En databas är en samling information som organiserats så att ett dataprogram snabbt kan hitta önskade data. En databas kan närmast liknas med icke-datoriserade kortkataloger och är ett passivt datalager. För att hantera informationen i en databas används ett databashanteringssystem (Database Management System, DBMS). Detta är en samling program för att lägga in, organisera och hämta ut data i en databas. Uttrycket databas används numera oftast för att beteckna ett helt databashanteringssystem [Pag03].

7.4.2 Blackboards

En blackboard skiljer sig från en vanlig databas i det att den inte är passiv utan kommunicerar med ett antal klienter genom att skicka meddelanden när data förändrats till klienter som anmält intresse av det ändrade datat [Bas98].

7.4.3 Hypertextsystem

Ordet hypertext betyder *icke-sekvensiellt skrivande* och ett hypertextsystem är ett system där data förvaras i noder som är ihopkopplade av länkar (hyperlinks). Noderna kan innehålla text, bilder, ljud, kod eller någon annan form av data. Länkarna används för att traversera hypertextstrukturen för att flytta från en nod till en annan [Cbl94].

7.5 Virtuella maskiner

En virtuell maskin är mjukvara som simulerar någon form av funktionalitet som inte finns naturligt i den hårdvara/mjukvara där den är implementerad.

Den kan till exempel simulera plattformar som inte är byggda än eller katastrofhändelser för till exempel flygsimulatorer som skulle vara för dyra eller för komplexa att testa med riktiga system [Bas98].

7.5.1 Interpreters

En interpreter är ett program som tolkar och översätter kod. En interpreter har vanligen fyra beståndsdelar; en översättare, ett minne innehållande koden som ska översättas, en komponent som visar översättarens status och en komponent som visar status på programmet som exekveras. Interpreters används ofta för att bygga virtuella maskiner [Sha96b].

7.5.2 Regelbaserade system

Regelbaserade system presenterar information som en mängd regler som säger vilka slutsatser som kan dras vid olika situationer. Det består av en mängd "if-then-regler", en mängd fakta och någon form av interpreter som kontrollerar användandet av reglerna på faktat. Det finns två typer av regelbaserade system: "forward chaining" och "backward chaining". Ett forward-chaining-system startar med initiala fakta och använder reglerna för att dra ytterligare slutsatser. Ett backward-chaining-system startar med en hypotes som ska bevisas och letar efter regler som kan bevisa hypotesen [Cee02]. Ett system byggs sällan genom att använda bara en av de ovanstående stilarna, utan flera kombineras oftast på olika sätt. Det kan dock vara svårt att välja stilar i och med att kombinationerna skulle kunna göras på många olika sätt. En god tumregel är att börja med den arkitekturella strukturen som har störst påverkan på de kvalitetsaspekter som är viktigast för det aktuella systemet. Utifrån detta väljs en stil som kan komma att kompletteras med andra stilar för att lösa sekundära kvalitetsfrågor.

8 Komponentbaserade informationssystem

En strategi för att nå konkurrensfördelar med avseende på en verksamhets informationssystem är att utveckla ett komponentbaserat informationssystem. Denna strategi har ännu inte revolutionerat mjukvaruindustrin men blir dock allt mer populär. Komponentbaserade informationssystem baseras på en komponentbaserad systemutveckling (Component-Based System Development, CBSD eller Component-Based System Engineering, CBSE) vilket innebär att genom integration av färdiga mjukvarukomponenter skapa en unik systemlösning. Systemutveckling genom att skriva kod ersätts av sammanfogning och integrering av existerande mjukvarukomponenter. Detta är dock inget nytt tankesätt inom systemutveckling utan presenterades redan 1968 av Douglas Mcllroy på "the NATO Conference on Software Engineering" [Chr99]. Hans tanke var att massproducera mjukvarukomponenter som sedan kunde användas som byggblock i olika mjukvaruproduktioner. Kunden, dvs. programmeraren skulle kunna skräddarsy systemet genom att med hjälp av en produktkatalog välja ut de komponenter som behövdes och därefter foga samman dessa till ett färdigt system [Chr98].

8.1 Fördelar med komponentbaserad systemutveckling

Traditionell systemutveckling kan enligt Christiansson och Jakobsson beskrivas av två extrema huvudstrategier. Den ena strategin är att utveckla system för att fullkomligt uppfylla kundens krav på systemet, s.k. skräddarsydda informationssystem. Den andra strategin är att utveckla system med en viss typ av kundgrupp i fokus, s.k. standardsystem.

Fördelen med den första, kundorienterade strategin är att kundens unika vinstgivande verksamhet kan stödjas av systemet och på så vis ge kunden en fördelaktig position i förhållande till konkurrenter. Nackdelen är dock att sådana system ofta är oerhört kostsamma att utveckla och kan resultera i katastrofala kostnadsförluster om systemet ej medför de förväntade vinsterna.

Den andra strategin baseras på att utvecklingskostnaden fördelas på flera kunder och därmed blir lägre för varje enskild kund. En nackdel med denna strategi är att en anpassning till färdigutvecklade system kan kräva en omfattande omorganisering av verksamheten. En annan nackdel med denna strategi är svårigheten att förändra systemet när verksamheten förändrar sitt arbetssätt.

Dessa två strategier ska dock ses som extrema då systemutvecklingsprojekt

i verkligheten snarare är blandningar av dessa. Christiansson och Jakobsson menar därför att system bör kunna byggas enligt en mellanstrategi där de aktuella unika situationerna styr systemutvecklingen. Detta skulle kunna uppnås genom att använda ett komponentbaserat synsätt på systemutvecklingen med användning av standardiserade mjukvarukomponenter eller unika specialutformade komponenter. En komponentbaserad systemutveckling skulle då innehålla fördelarna från de båda extrema traditionella systemutvecklingstrategierna och utelämna dess nackdelar [Chr99].

Att utveckla komponentbaserade informationssystem underlättas i dagsläget av att kvaliteten på färdiga standardkomponenter, så kallade *commercial off-the-shelf-komponenter* (COTS) ökar men även att olika teknologier för att integrera dessa utvecklats och förbättrats. En annan faktor som möjliggör och inbjuder till komponentbaserad systemutveckling är all den mjukvara som redan existerar inom organisationer, mjukvara som kan återanvändas i nya system. Den ursprungliga tanken var att de delar av system som ständigt återkommer vid skapande av nya system skulle kunna utvecklas endast en gång och därefter återanvändas istället för att utvecklas på nytt varje gång [SEI02]. Principen är alltså att köpa eller att återanvända istället för att göra.

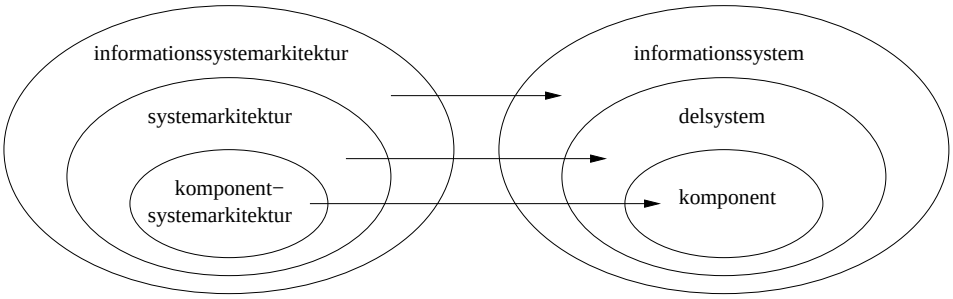
Ur ekonomisk synvinkel är möjligheten att reducera utvecklings- och förvaltningskostnader motivationen till att övergå till en komponentbaserad systemutveckling. Tanken att kostnaderna **endast** minskar genom att implementationsfasen uteblir är dock felaktig. Vid en komponentbaserad systemutveckling tillkommer faser såsom utvärdering av befintliga komponenter, anskaffningsfas och integration av utvalda komponenter.

8.2 Innebörden av komponentbaserat

Ett komponentbaserat informationssystem består av en uppsättning mjukvarukomponenter. Dessa komponenter kan antingen vara standardkomponenter (COTS) eller egenutvecklade komponenter. Med standardkomponenter menas komponenter som producerats i syfte att användas i många olika system. En standardkomponent kan köpas och användas av många olika kunder men utvecklas av endast en leverantör [Chr99]. Mjukvarukomponenterna sammanfogas enligt en *komponentsystemarkitektur*. Komponentsystemarkitekturen är den uppsättning beslut som styr hur mjukvarukomponenterna ska integreras och kommunicera med varandra. Komponentsystemarkitekturen styrs i sin tur av de beslut som omfattas av informationssystemets övergripande systemarkitektur. Även komponentsystemarkitekturen kan vara standardiserad eller egenutvecklad [Chr98].

Ett komponentbaserat informationssystem är mer komplext än ett traditionellt utvecklat sådant. Ett komponentbaserat informationssystem kan enligt Christiansson och Jakobsson betraktas i tre olika nivåer, komponentsystemarkitekturen, systemarkitekturen och informationssystemarkitekturen.

Den innersta nivån är komponentsystemarkitekturen. Den representerar mjukvarukomponenterna och den nödvändiga tekniken för att få dessa att kommunicera. Mellannivån är systemarkitekturen. Den beskriver grupperingar av komponenter i olika delsystem, applikationer. Den yttersta nivån är informationssystemarkitekturen.
Med avseende på dessa tre nivåer menar Christiansson och Jakobsson att livscykeln hos ett komponentbaserat informationssystem bör delas in i tre nivåer, en livscykelmodell för varje komponent, en för varje delsystem och en för varje informationssystem. Genom denna uppdelning ges en mer nyanserad och användbar beskrivning av hur utveckling och förvaltning av det komponentbaserade informationssystemet ska ske [Chr99]. I figur 6 presenteras de tre olika nivåerna tydligt.



Figur 6: De tre nivåerna i ett komponentbaserat informationssystem [Chr99]

Komponentbaserad systemutveckling kan ses ur två perspektiv, konsumentens perspektiv och producentens perspektiv. Konsumenten är den verksamhet som använder sig utav färdigutvecklade mjukvarukomponenter och sammanfogar dessa till ett färdigt system. Producenten är den verksamhen som producerar mjukvarukomponenterna som konsumenten köper och konsumerar. De båda parterna delar tankesättet för hur systemutvecklingen bör ske men hanterar olika problem.

8.2.1 Konsumentens perspektiv

Konsumenten måste förlita sig på kvaliteten hos komponenter som används, speciellt när komponenter endast säljs i binär form och komponenten endast kan ses som en så kallad black-box. Konsumenten har då svårt för att förändra en komponents funktionalitet utan hjälp från producenten. Detta kan i sin tur innebära att konsumenten kan bli alltför beroende av leverantörerna av systemets olika komponenter. Det är därför viktigt att välja leverantörer utifrån ett långsiktigt perspektiv och försäkra sig om att leverantören i framtiden kommer att kunna erbjuda det stöd som krävs.

Ett annat problem som konsumenten måste hantera är komponenters gränssnitt utåt. Det finns ännu ingen allmänt använd standard för hur komponenter ska kommunicera med den övriga systemarkitekturen, som det t.ex. finns i hårdvaruindustrin. Komponenternas producenter har olika antaganden om användare och hur gränssnitt bör se ut [Max01]. Detta medför att sammanfogning av komponenter ofta måste föregås av en anpassningsfas där konflikter mellan komponenter minimeras [Chr99]. Denna komponentanpassningsfas beskrivs närmare i avsnittet om systemutvecklingsprocessen.

8.2.2 Producentens perspektiv

Producentens arbete inleds med en analysfas där de behov som komponenten ska uppfylla analyseras. Därefter arbetas en design fram vartefter komponenten implementeras, testas, förvaltas, marknadsförs och slutligen färdigställs för distribution. Resultatet av denna process är en komponent som uppfyller de aktuella kraven som tidigare analyserats. Komponenten riskerar alltså att endast uppfylla de behov som beaktats i nutid inte i framtida situationer. Producenten måste därför tillhandahålla service och underhåll för sina komponenter även efter distribution eftersom konsumenten själv inte kan förändra komponentens funktionalitet, med undantag för white-box-komponenter, komponenter där koden är tillgänglig [Chr99].

Eftersom producenten i de allra flesta fall inte deltar i någon användningfas av komponenten är det enligt Christiansson och Jakobsson viktigt att denne kommunicerar med konsumenten för att på så vis får ta del av iakttagelser under exekvering. Detta för att kunna genomföra rätt förändringar och förnyelse i komponenten [Chr99].

Producenten måste även ständigt sträva efter att utveckla attraktiva komponenter som lätt kan integreras och konfigureras för att anpassas till andra

komponenter. Om en komponent inte är tillräckligt bra och alldeles för få konsumenter köper den kommer komponentens utveckling inte att ge den ekonomiska utdelning som förväntats vilket innebär att producenten riskerar att gå med förlust [Lud99].

8.3 Systemutvecklingsprocessen

Att använda ett komponentbaserat synsätt på systemutvecklingen medför att utvecklingsprocessen förändras. En komponentbaserad systemutveckling innebär att utifrån kraven på önskat system köpa eller konstruera komponenter som sedan fogas samman och tillsammans skapar det färdiga informationssystemet [Chr98]. Christiansson jämför det med bilindustrins utveckling där varje biltillverkare till en början konstruerade alla delar själv och därefter fogade samman dessa till en färdig bil. Nu för tiden har biltillverkare en uppsättning underleverantörer som förser dem med de delar som behövs för att kunna montera ihop en färdig bil.

Eftersom den komponentbaserade systemutvecklingsprocessen ännu inte etablerat sig så pass att det finns en genomgående uppfattning av dess faser finns det ingen återkommande systemutvecklingsmodell liknande den traditionella systemutvecklingsmodellen med faser som analys, design, implementation, integration osv. Den komponentbaserade systemutvecklingsprocessen karaktäriseras enligt *the Software Engineering Institute*, SEI av fyra huvudsakliga faser, komponentkvalificering, komponentanpassning, komponentintegrering och komponentförvaltning [SEI02]. De har då utelämnat arbetet kring analys och design i sin beskrivning. Christiansson och Jakobsson beskriver den komponentbaserade systemutvecklingen ur ett mer traditionellt perspektiv med de traditionella utvecklingsfaserna som bas [Chr99] [Chr98]. Dock liknar de olika beskrivningarna varandra till innehåll. Skillnaden ligger i fasindelningen och namnen på dessa.

De traditionella namnen på faserna, främst med avseende på implementering och integrering påminner mycket av dess traditionella innebörd och kan därför få en förvirrande inverkan på förståelsen. I nedanstående avsnitt används därför namnen från SEIs beskrivning. Dessa kompletteras dock med inledande beskrivning av analys och designfasen. Fasindelningen i denna uppsats utgör bara ett exempel på olika faser som kan ingå i den komponentbaserade systemutvecklingsprocessen.

8.3.1 Analys

Här identifieras de krav som finns på det kommande informationssystemet. Vid ett komponentbaserat synsätt på systemutvecklingsprocessen sker analysen av systemkraven utifrån ett komponentperspektiv. Systemet specificeras med ett antal komponentspecifikationer. Även specifikationer över hur komponenterna ska integreras skapas i analysfasen. Om en komponent beskrivs på ett standardiserat och väldefinierat sätt ökar möjligheten att finna lämpliga befintliga komponenter och på så vis kunna återanvända denna. Ett standardiserat specifikationsarbete är alltså en grundläggande del i analysfasen för att kunna nå den grad av återanvändning verksamheten eftersträvar. En förutsättning är dock att även producenterna använder standardiserade metoder för att beskriva sina komponenter.

Genom att ha ett komponentbaserat synsätt på analysfasen behövs inte detaljerade beskrivningar av systemets olika delar, såsom exakta protokoll eller detaljerade specifikationer av dataformat, eftersom de komponenter som införskaffas redan implementerat dessa detaljer. Det räcker med att beskriva den önskade tekniken med vanliga ord, t.ex. ”batchöverföring med hjälp av Internet”. Systemet kan alltså beskrivas med ett verklighetsnära språk istället för i tekniska termer. Detta i sin tur leder till att slutanvändaren kan delta mer i systemutvecklingsprocessen [Chr98].

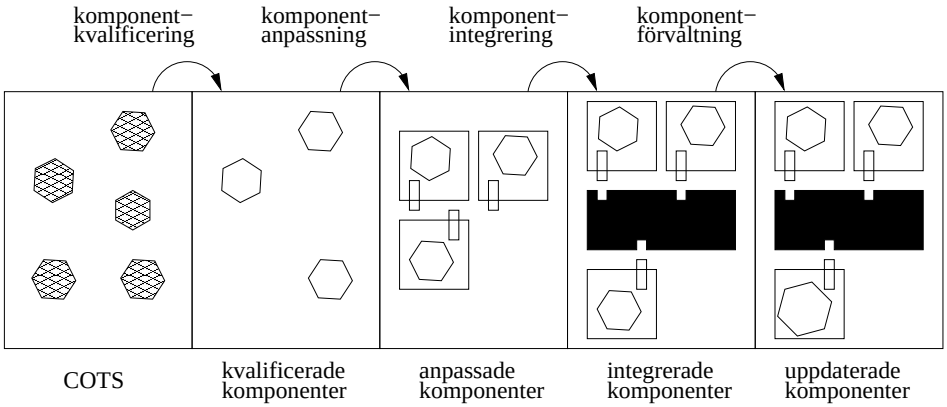
8.3.2 Design

I designfasen ska komponentsystemarkitekturen, komponentspecifikationerna och dess detaljer vidareutvecklas. Det kan även krävas att komponentsystemarkitekturen omarbetas efter en viss marknadsstandard så att samarbete med andra informationssystem, t.ex. underleverantörers eller olika kunders informationssystem, underlättas. Designarbetet bör inte bedrivas separat för varje problemsituation eftersom likhet ska eftersträvas. Att utarbeta och styra designen centralt är dock en slags motsägelse till en av de grundläggande fördelarna med det komponentbaserade synsättet, nämligen möjligheten till decentraliserad förvaltning [Chr98].

8.3.3 Komponentkvalificering

I komponentkvalificeringen identifieras befintliga komponenter, på marknaden eller inom den egna organisationen, som kan uppfylla komponentspecifikationerna framtagna i analysen och designen. Detta kan innebära att konkurrerande komponenter som uppfyller samma komponentspecifikation måste genomgå någon form av process där en väljs ut. Att välja ut rätt

komponenter kan vara oerhört svårt eftersom både funktionella och icke-funktionella krav måste undersökas samtidigt som många andra kvalitetsaspekter ska tas hänsyn till. Det finns dock ett antal utvärderingsmetoder som kan användas för att välja ut lämpliga komponenter. Olika organisationer, t.ex. ISO[11] har utformat metoder för produktutvärdering [SEI02].



Figur 7: Faserna i komponentbaserad systemutveckling enligt SEI [SEI02]

Efter att de olika komponenterna införskaffats måste utvecklarna försäkra sig om att komponenterna beter sig enligt komponentspecifikationerna. Det räcker alltså inte att förlita sig på komponenternas dokumentation, då denna kan vara felaktig och ofullständig. Om verksamheten däremot använder komponenter som redan finns i det interna komponentlagret, vilket innebär att de tidigare genomgått denna testfas, behövs inga separata tester av dessa [Chr98].

Självklart finns även möjligheten att verksamheten själv tillverkar den efterfrågade komponenten. Egentillverkning av en komponent kan även vara enda alternativet, t.ex. om komponenten är alltför unik och verksamhetsspecifik för att kunna införskaffas från någon producent [Chr98]. I dessa fall utökas komponentkvalificeringen med ett traditionellt design- och implementationsarbete, dock i mindre skala eftersom enskilda komponenter är mindre komplexa än hela system. Komponentdesignen underlättas även av att or-

[11] *International Organisation for Standardization* är ett internationellt standardiseringsorgan som tillhandahåller standarder för kvalitet, miljö o.dyl. inom diverse branscher (www.iso.org).

dentliga specifikationer av önskad funktionalitet redan utformats i analysfasen [Chr98].

I figur 7 finns processen från komponentkvalificering till komponentförvaltning beskriven.

8.3.4 Komponentanpassning

Komponentanpassningen kan ses som inledningen av monteringsarbetet, dvs. då den planerade konstruktionen realiseras. För att de utvalda komponenterna ska kunna sammanfogas måste ett visst anpassningsarbete ske. Komponenterna kan vara producerade av en mängd olika producenter, alla med olika föreställningar om hur deras komponent ska interagera med andra komponenter och i vilket sammanhang den kommer att användas [SEI02]. Komponenterna måste därför anpassas så att de passar in i den planerade systemarkitekturen och att de fungerar som väntat tillsammans med övriga komponenter.

Det finns olika tekniker för att genomföra komponentanpassningen beroende på vilken typ av komponent som ska anpassas. Öppna komponenter, så kallade *white-box-komponenter* kan förändras direkt i källkoden och på så sätt anpassas till övriga komponenter. *Wrapping*[12] är exempel på en teknik för att anpassa komponenter där källkoden inte är åtkomlig, dvs. *black-box-komponenter* eller *grey-box-komponenter* där endast ett användargränssnitt (eng. application programming interface, API) är åtkomligt [SEI02]. De komponenter som inte går att anpassa väljs bort och ersätts med någon annan [Max01]. Denna fas skulle underlättas av att producenter enades kring en standard för hur komponenters gränssnitt ska utformas. Komponentanpassningen kan ses som motsvarigheten till implementationsfasen i den traditionella systemutvecklingen.

8.3.5 Komponentintegrering

Komponenterna ska slutligen integreras i den omgivande systemmiljön. Detta arbete styrs av den övergripande systemarkitekturen och de valda arkitekturella stilarna för integrering. Exempel på integrationstekniker är databas, blackboard eller Object Request Broker (ORB)[13] [14].

[12] Ordet finns förklarat i ordlistan.

[13] *ORB* är en teknik som kan tillämpas vid implementering av system bestående av oberoende komponenter. Mer om oberoende komponenter under avsnittet Arkitekturella stilar.

[14] Ordet finns förklarat i ordlistan.

Komponentintegreringen utgör en central del i den komponentbaserade systemutvecklingen eftersom det är under denna fas som komponenterna sammanfogas till det färdiga systemet. Eftersom arbetet i denna fas styrs av systemarkitekturen är det oerhört viktigt att ha en genomtänkt sådan. Enligt förespråkarna av komponentbaserad systemarkitektur ska komponentintegreringen kunna ske i form av "plug-and-play"[15] förutsatt att det finns en grundläggande väl formulerad systemarkitektur [Chr98]. Givetvis är detta en idealiserad bild. Många komponenter, främst de av typen black-box, kan innehålla brister som inte upptäcks förrän i integrationsfasen [SEI02]. Detta medför att integrationsfasen till stor del kommer att bestå av tester.

Då risken finns att integrationsfasen tar längre tid än förväntat poängterar Lüders att det är viktigt att inte låta integrering av en inköpt komponent ta längre tid än vad egentillverkning av samma funktionalitet skulle ta [Lud99].

8.3.6 Komponentförvaltning

En fördel med ett komponentbaserad systemstruktur är möjligheten att kontinuerligt byta ut eller förändra komponenterna för att motsvara verksamhetens förväntningar av systemet [Chr98]. Felaktigheter åtgärdas genom att de felaktiga komponenterna byts ut mot nya uppdaterade komponenter. Vidareutveckling i form av utökad funktionalitet sker genom att addera en eller flera komponenter till systemet.

Att betrakta detta förvaltningsarbete som enkelt är dock alltför simpelt och optimistiskt enligt SEI. Byte av en komponent mot en annan är tidskrävande eftersom den nya komponenten först måste genomgå samma anpassningsprocess och testning som den gamla komponenten gjort. Den anpassningsteknik som använts för den gamla komponenten måste antagligen förändras för att passa den nya komponenten. Därefter, när den nya komponenten integrerats med övriga komponenter måste den testas igen för att säkerställa att den utför det förväntade [SEI02]. Ytterligare arbete krävs om den nya eller uppdaterade komponenten kräver uppdaterad funktionalitet även hos andra komponenter i systemet.

Ett annat problem med förvaltning av komponentbaserade informationssystem är det leverantörsberoende som kan uppstå när en verksamhet köper komponenter från en viss producent. Producenten kan stå för utveckling

[15] Ordet finns förklarat i ordlistan.

av nya komponenter eller vidareutveckling av befintliga komponenter. Ett sådant leverantörsberoende medför att förvaltningsarbetet får ytterligare en dimension att ta hänsyn till [Chr98].

9 Arkitekturell beskrivning

För att möjliggöra diskussioner om arkitekturen mellan olika inblandade är det nödvändigt att presentera arkitekturen förståeligt. Arkitektur och arkitekturell beskrivning skiljer sig på så sätt att en representation av arkitekturen kan utelämna irrelevanta delar för att passa betraktaren. Arkitekturen är därmed, som Philippe Krutchen säger, inte platt utan multidimensionell. Detta innebär att arkitekturen kan presenteras ur olika synvinklar [Kru01].

En arkitekturell beskrivning är enligt The Open Group en formell beskrivning av ett informationssystem och dess arkitektur med uppgift att stödja diskussioner och analyser kring strukturella egenskaper hos systemet [TOG03]. Vi kommer i detta avsnitt beskriva de huvudsakliga delarna av en arkitekturell beskrivning.

9.1 Arkitekturella ramverk och standarder

Genom att använda ett ramverk eller en standard som stöd kan utveckling av systemarkitekturer och dess tillhörande arkitekturella beskrivning förenklas och genomföras på kortare tid än utan detta stöd. Användning av ramverk kan dessutom medföra en mer heltäckande designlösning och att arkitekturen bättre klarar av en framtida organisatorisk förändring.

Ett arkitekturellt ramverk är ett verktyg som kan användas för att utveckla många olika typer av systemarkitekturer och olika ramverk kan då se väldigt olika ut. The Open Group anser dock att det bör innehålla vissa saker för att räknas som ett ramverk [TOG03]:

- En metod för att designa informationssystem bestående av några olika steg. Metoden ska även beskriva hur de olika stegen hör ihop.
- En mängd verktyg.
- En ordlista över ”tillåtna” ord för att förhindra tvetydighet.
- En lista över rekommenderade standarder och produkter som kan användas för att implementera alla stegen i metoden.

Nedan kommer vi presentera den standard som många forskare och organisationer baserar sina arkitekturella beskrivningar på, IEEE Std 1471. Vi kommer även redogöra för två kända ramverk i arkitekturella sammanhang. Dessa är TOGAF och the Zachman Framework.

9.1.1 IEEE-1471

IEEE (Eye-triple-E) står för Institute of Electrical and Electronics Engineers, Inc. Det är ett professionellt, icke vinstgivande, tekniskt samfund med medlemmar från 150 olika länder. IEEE har formulerat många olika standarder för diverse olika tekniska sammanhang. En av dessa standarder är IEEE Std 1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems* (härefter IEEE-1471). Med "software intensive systems" avses system där mjukvaran har den huvudsakliga påverkan på design, konstruktion, underhåll och vidareutveckling av systemet. Många arkitekturella ramverk och systemutvecklingsmetoder, däribland RUP, baseras på IEEE-1471 [Kru01].

IEEE-1471 är utformad för att fungera som ett vokabulärt stöd i arkitekturella sammanhang och för att underlätta dess diskussioner. Den innehåller termer och koncept som ska användas i den arkitekturella beskrivningen. De viktigaste delarna i IEEE-1471 är *stakeholders*, *views* och *viewpoints*, dvs. de inblandade parterna, vyer och synvinklar (utgångspunkt). Standarden skiljer därmed på vyer och synvinklar. En vy är den faktiska representationen av ett system utifrån en viss synvinkel. En synvinkel är ett mönster eller en mall för hur vyer ska utformas utifrån det aktuella perspektivet.

Då IEEE-1471 främst fungerar som en standard för termer och koncept innehåller den ingen rekommendation av arkitekturellt språkval (eng. architectural description language) för den arkitekturella beskrivningen. Den innehåller inte heller en uppsättning synvinklar som bör användas i det arkitekturella arbetet. Dock innehåller den en mall för hur synvinklarna ska specificeras [IEEE03].

9.1.2 TOGAF

TOGAF är ett ramverk för utveckling av systemarkitekturer. Det utvecklades 1995 av *The Open Group* och baserades då på *The Technical Architecture Framework for Information Management* som var utvecklat av *US department of Defense*. Sedan 1995 har The Open Group utvecklat en ny version av TOGAF varje år och publicerat denna på sin webbsida.

TOGAF innehåller två huvuddelar [TOG03]:

The TOGAF Foundation Architecture. En arkitektur med allmänna tjänster och funktioner som en grund på vilken specifika arkitekturer och

arkitekturella block kan byggas. The TOGAF Foundation Architecture innehåller *The TOGAF Standards Information Base (SIB)*, som är en databas med öppna industristandarder som kan användas för att definiera tjänster och organisationsspecifika komponenter.

The TOGAF Architecture Development Method (ADM). ADM beskriver steg för steg hur utvecklingen från the Foundation Architecture till en organisations.specifik arkitektur ska gå till. ADM tillhandahåller bland annat *The Business Scenario Method* som är en beprövad metod för att definiera och förstå verksamhetskrav och guidning för att utveckla arkitekturella vyer.

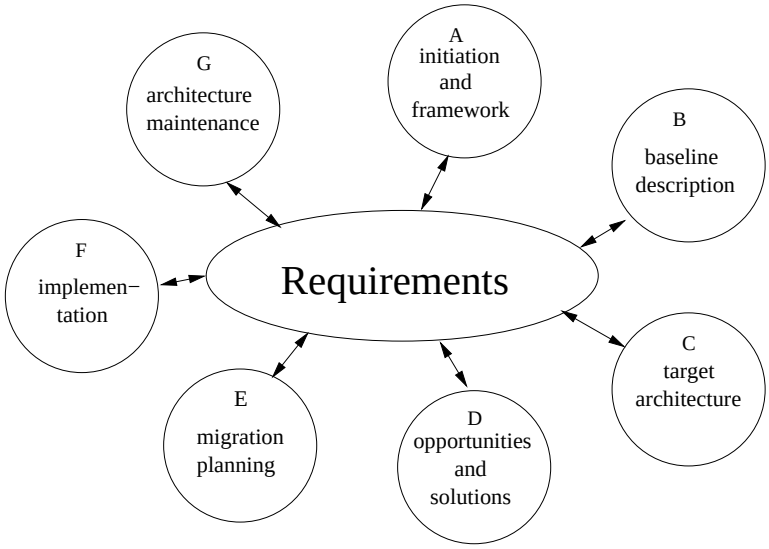
ADM
ADM är en generell metod för arkitekturell utveckling som har designats att hantera de flesta system och verksamhetskrav. Den måste dock ofta modifieras eller utökas för att passa speciella krav. Det finns ett antal generella principer som fått stå till grund vid skapandet av TOGAF ADM:

- En arkitektur ska bara specifiera de tjänster som absolut krävs.
- Ett element i en arkitektur kan specifiera en, fler än en eller bara en del av en tjänst.
- Element i en arkitektur ska definineras med standarder som är relevanta för den aktuella tjänsten.
- En arkitektur måste följas, annars är den lönlös.

ADM består av sju iterativa steg. Dessa är:

- Initiation and Framework
- Baseline Description
- Target Architecture
- Opportunities and Solutions
- Migration Planning
- Implementation
- Architecture Maintenance

Figur 8 visar de sju iterationsstegen, där alla faser är iterativa, både inom och mellan faserna.



Figur 8: TOGAFs sju iterationsfaser [TOG03]

9.1.3 Zachmans ramverk för informationssystemarkitektur

Zachmans ramverk för informationssystemarkitekturer presenterades första gången 1987 och senare som en utbyggd version 1992 och har enligt Office of Information and Technology blivit en ”de facto” standard inom utveckling av informationssystemarkitekturer [Va02]. Ramverket är ett resultat av Zachmans studier av processen när komplexa ingengörsprodukter byggs, både gällande arkitektur, konstruktion och själva tillverkningen. Resultatet av dessa studier blev ett ramverk att använda inom mjukvaruindustrin eftersom denna inte har tillgång till samma erfarenhet som den traditionella industrin har. Zachman menar att det inte är speciellt stor skillnad mellan och definitivt inte lättare att designa och förändra informationssystemarkitekturer än att designa och förändra byggnader eller flygplan och att det därför går att dra nytta av erfarenheten som existerar inom den traditionella arkitekturen [Zac02a]. Målet med ramverket är att tillhandahålla en logisk struktur för att stödja organisering, åtkomst, samverkan, tolkning, utveckling, skötsel och förändring av en verksamhets representation av dess informationssystemarkitektur [Zac99].

Zachman har, till skillnad från många andra, frångått att representera en process som ett antal steg och istället organiserat sitt ramverk efter ett antal aktörer eftersom det är viktigt att inse att ett system utvecklas av flera olika

grupper som har olika syn på och mål med systemet.

Zachmans ramverk illustreras i form av en matris, se figur 9. Raderna i matrisen representerar olika aktörers syn på utvecklingsprocessen, det vill säga olika vyer. Dessa är, enligt The Zachman Institute for Framework Advancement [Zif02]:

- **Scope** (Planerarvy) - definition av verksamhetens mål och inriktning.
- **Enterprise Model** (Ägarvy) - definition av verksamheten och dess struktur, funktion, organisation m.m.
- **System Model** (Designvy) - definition av verksamheten som i rad 2 men mer informationsinriktat.
- **Technology model** (Utvecklarvy) - definierar hur teknik kan användas för att hantera informationsprocesserna som identifierats i tidigare rader.
- **Detailed representations** (Underleverantörvy) - specificerar till exempel databasen, nätverk osv. Uttrycks i speciellt språk.
- **Functioning enterprise** - systemet är färdigutvecklat.

Kolumnerna i matrisen representerar det som ska undersökas av aktörerna och kan enligt, Zachman, beskrivas som nedan [Zac97]:

- **Data** (vad) - beskriver verksamhetens data och relationen mellan datat.
- **Function and Processes** (hur) - beskriver hur verksamhetens data används. Till exempel hur ordrar görs.
- **Network** (var) - beskriver verksamhetens nätverk. Till exempel om all data förflyttas mellan datorer i ett och samma hus eller om datorer måste vara hopkopplade över hela världen.
- **People** (vem) - beskriver ansvarshierakier och människor inom verksamheten och det arbete de utför.
- **Time** (när) - beskriver tidsplanering av processer, vilket kan vara en fixt tid som triggar en process, tiden på en händelse som triggar en annan process eller en tidssekvens som beskriver i vilken ordning processer måste köras.
- **Motivation** (why) - beskriver verksamhetens mål och drivkrafter.

	data (what)	**process** (how)	**network** (where)	**people** (who)	**time** (when)	**motivation** (why)
Scope (planner)	List of important entities	List of business processess	List of operating locations	List of important organizationals	List of significant events	List of business Goals
Enterprise Model (Owner)	Semantic Model	Business Process Model	Logistic Network	Work Flow Model	Master Schedule	Business Plan
System Model (Designer)	Logical Data Model	Application Arch.	Distributed System Arch.	Human Interface Arch.	Processing Structure	Business Rule Model
Technology Model (Builder)	Physical Data Model	System Design	System Arch.	Presentation Arch.	Control Structure	Rule Design
Detailed Representation (Subcontractor)	Data Definition	Program	Network Arch.	Security Arch.	Timing Defintion	Rule Specification
Functioning Enterprise	Data	Function	Network	Organization	Schedule	Strategy

Figur 9: Zachmans ramverk för informationssystemarkitektur [Zac97]

Kombination av alla celler på en rad ger en fullständig beskrivning av den aktuella vyn.

Användning av detta ramverk i en utvecklingsprocess innebär att beskriva verksamheten utifrån alla dessa celler. Detta görs radvis med start på rad ett. Kolumnerna har däremot ingen inbördes ordning och kan beskrivas efter valfritt system [Va02].

Zachman har med sitt ramverk inte specificerat lämpliga modeller att använda för alla celler. För vissa celler existerar många modeller på marknaden och för vissa existerar ingen väl dokumenterad modell alls. Det är alltså verksamheten som använder sig av Zachmans ramverk som måste avgöra vilka modeller som passar bäst för varje cell [Hay97].

9.2 Arkitekturella vyer

I likhet med vanlig traditionell byggnadsarkitektur behövs vyer för att beskriva systemets arkitektur. En vy är enligt Philippe Krutchen en förenklad beskrivning av systemet ur ett visst perspektiv. För att ge betraktaren en fördelaktig synvinkel utelämnas irrelevanta enheter.

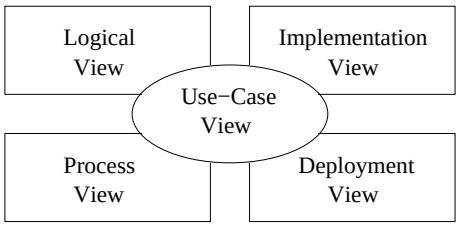
Krutchen anser att för varje vy måste följande egenskaper fastställas [Kru01]:

- vem och vad vyn är till för
- vilka element och deras relationer som finns representerade i vyn
- vilka organisatorisk principer som använts för att strukturera den aktuella vyn
- hur element i den aktuella vyn relaterar till element i andra vyer
- bästa sättet (the best process) att skapa den aktuella vyn

Det finns emellertid inget generellt sätt att dela upp mjukvaruarkitekturen i vyer. Anledningen till uppdelningen är dock, enligt Hofmeister, Nord och Soni alltid densamma; att separera olika aspekter i separata vyer för att hjälpa människor att hantera komplexiteten [Hof00].

9.2.1 RUP

RUP förslår en modell för vilka vyer som bör presenteras i systemarkitekturen. Modellen kallas ”The 4+1 View Model of Architecture”, se figur 10. Modellen bygger på fyra huvudperspektiv, *logical*, *implementation*, *process* och *deployment*. Den femte vyn innehåller de huvudsakliga användarscenarion som ska ske i systemet, denna vy kallas *the Use-Case View*.



Figur 10: ”The 4+1 View Model of Architecture” [Kru01]

The Logical View (Logisk vy)
Denna vy riktar sig till användarna av systemet och beskriver de funktionella kraven på systemet. Det är en abstraktion av systemet bestående av de olika designpaketen, delsystemen och klasserna.

The Implementation View (Implementationsvy)
Denna vy beskriver organiseringen av de statiska mjukvarumoduler som systemet byggs av, dvs. källkod, filer, komponenter, exekverbara filer och andra artefakter. Dessa moduler finns med i utvecklingsmiljön och är delar i paket och olika datalager. Frågor som rör utvecklingen av dessa moduler och återanvändning hör till denna vy.

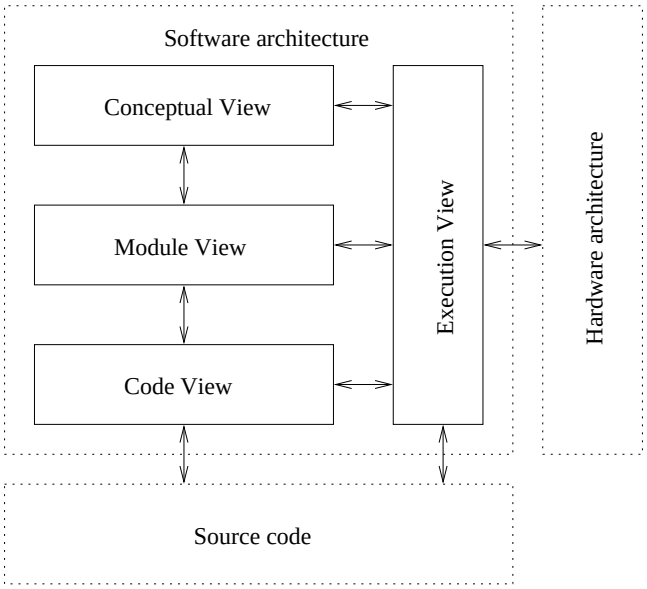
The Process View (Processvy)
Denna vy avser olika aspekter i och med körning av systemet. Olika aspekter kan vara trådar eller olika processer och deras interaktion. I denna vy behandlas områden som skalbarhet, parallellitet, att köra igång och stänga ner system, svarstider och distribuering av objekt.

The Deployment View (Distribueringsvy)
Denna vy visar hur de olika exekverbara filerna och de andra körbara komponenterna fördelas på de underliggande plattformarna. Det är i denna vy mjukvara möter hårdvara. Frågor inom t.ex. drift, installation och prestanda hör hemma i denna vy.

The Use-Case View (Användarvy)
Denna vy har en speciell roll i mjukvaruarkitekturen. Den innehåller en liten uppsättning huvudscenarior eller användarfall för att beskriva hur de övriga vyerna fungerar. I början av arbetet med mjukvaruarkitekturen används dessa för att leda arbetet medan de i slutet används för att bekräfta att de olika vyerna utformats på lämpligaste sätt.

9.2.2 Hofmeister, Nord och Soni

Hofmeister, Nord och Soni har genom en undersökning av mjukvaruarkitekturer bakom stora komplexa industriella system kommit fram till att mjukvaruarkitekturen kan ses utifrån 4 olika vyer; *conceptual*, *module*, *code* och *execution*. I sin undersökning observerade de att arkitekter har använt sig av dessa olika vyer i sitt arbete utan att nödvändigtvis ha sett dessa som separata arkitekturella vyer. I figur 11 presenteras dessa fyra vyer och hur de förhåller sig till den underliggande hårdvaran och tillhörande källod [Hof00].



Figur 11: Indelning i olika arkitekturella vyer enligt Hofmeister, Nord och Soni [Hof00]

Conceptual View (Konceptuell vy)
Den konceptuella vyn beskriver systemet utifrån dess element och relationerna mellan dessa. Systemets funktionalitet presenteras i arkitekturella element, konceptuella komponenter. De element som hanterar koordination och utbyte mellan dessa komponenter kallas kopplare.

Module View (Modulvy)
Då systemen är komplexa krävs en god hantering och fördelning av dess moduler. Abstraktion, inkapsling och tydliga gränssnitt är då viktiga koncept. Uppdelning i delsystem och organisering av modulerna i lager är huvudmålet för denna vy. De konceptuella komponenterna och kopplarna från den konceptuella vyn arrangeras i delsystem och moduler. Utifrån denna vy kan man undersöka hur moduler och delsystem ska utformas för att ge ökad återanvändning och minskade beroenden mellan dessa.

Code View (Kodvy)
I denna vy beskrivs de filer som källkoden fördelas i. Filerna arrangeras i bibliotek och en konfigurationshantering för dessa filer måste anskaffas eftersom det fortlöpande kommer skapas nya versioner. Att ha en ordentlig organisation på filer, binärkod och bibliotek underlättar återanvändningen avsevärt

som i sin tur effektiviserar utvecklingstiden för systemet.

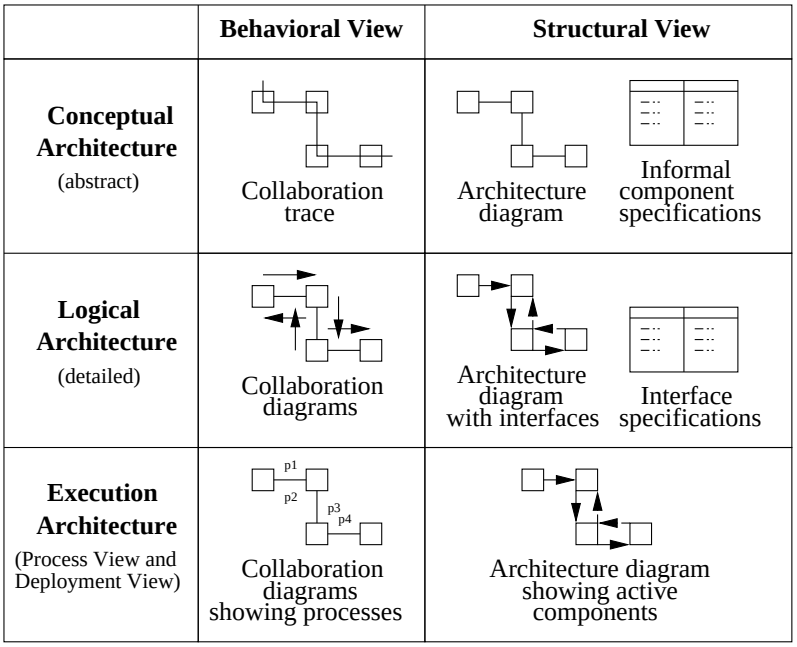
Execution View (Exekveringsvy)
Då systemen blivit allt mer distribuerade krävs funderingar över kommunikation, koordination och synkronisering mellan systemets enheter och hur dessa ska mappas till hårdvaran. Tidigare låg ansvaret för dessa frågor på programmerarna, dvs. i implementationsfasen, men i dagens läge tas dessa frågor upp tidigare, under arbetet med mjukvaruarkitekturen. Till skillnad från den konceptuella vyn som ser systemets kontrollflöde ur en logisk synvinkel ser man i denna vy systemets kontrollflöden ur den underliggande plattformens synvinkel.

9.2.3 Bredemyer och Malan

Till skillnad från de två tidigare indelningarna i vyer har Bredemeyer och Malan delat in arkitekturen i tre olika delarkitekturer, *Conceptual Architecture*, *Logical Architecture*, *Execution Architecture* [Bre02]. Som komplement till de tre delarkitekturerna väljer Bredemeyer och Malan att se dessa ur två vyer, *behavioral view* och *structural view*, dvs. en för delarkitekturernas struktur och en för dess beteende. Dessa vyer appliceras på varje delarkitektur. Figur 12 visar hur Bredemeyer och Malan valt att dela upp den arkitekturella beskrivningen.

Conceptual Architecture (Konceptuell arkitektur)
Här identifieras komponenter i systemet på hög nivå och relationerna mellan dessa. Dessa strukturer beskrivs i en slags samarbetsskiss (eng. collaboration trace). Syftet med denna del av arkitekturen är att skapa en lämplig uppdelning av systemet utan att gå in på detaljer. Genom denna uppdelning skapas ett hjälpmedel för att diskutera arkitekturen för de icke tekniskt inblandade. I denna vy skapas även ett arkitekturellt diagram (eng. architecture diagram) och tillhörande beskrivningar för varje komponent. Dessa beskrivningar är dock informella och utan detaljer.

Logical Architecture (Logisk arkitektur)
I denna del av arkitekturen förfinas de externt synliga egenskaperna hos komponenterna så att de blir precisa och otvetydiga. Välformulerade gränssnitt och komponentspecifikationer skapas. Även arkitekturella mekanismer beskrivs detaljerat. Utifrån den logiska arkitekturen ska utvecklarna för de enskilda komponenterna kunna arbeta relativt oberoende av de andra komponenternas utvecklare. Det arkitekturella diagrammet från den konceptuella vyn utökas med gränssnitt- och komponentbeskrivningar, beskrivningar av



Figur 12: Indelning i vyer enligt Bredemeyer och Malan [Bre02]

hur komponenter samarbetar och beskrivningar av arkitekturella mekanismer, protokoll mm. .

Execution Architecture (Exekveringsarkitektur)
En *execution architecture* behövs främst vid utveckling av distribuerade system. Bredemyer och Malan ser denna arkitektur i två vyer, en *process view* och en *deployment view*. I *process view* beskrivs hur komponenter ska mappas till processer i det underliggande fysiska systemet med hänsyn till avseenden som t.ex. skalbarhet och genomströmning. I *deployment view* beskrivs hur de exekverbara modulerna ska utplaceras på olika noder i det fysiska systemet. Dessa två vyer kan liknas vid RUPs vyer, *The Process view* och *The Deployment View*.

Structural View (Strukturell vy)
Denna vy består av det arkitekturella diagrammet, utformat som ett UML-diagram med tillhörande beskrivningar av komponenterna och dess gränssnitt. Noggrannheten och detaljerna i dessa beskrivningar beror på vilken del av mjukvaruarkitekturen som för tillfället struktureras.

Behavioral View (Beteendevy)
Då Bredemeyer och Malan anser att det inte räcker att beskriva mjukvaruarkitekturen utifrån strukturer och diagram har de utformat denna vy som komplement till *structural view*. Denna vy ska beskriva hur komponenternas interaktion och samarbete fungerar, detta i form av sekvensdiagram.

9.3 Architecture Description Languages (ADL)

I och med att mjukvarusystem har blivit större och komplexare har det blivit allt mer viktigt att beskriva arkitekturen på systemen. Dessa beskrivningar behandlar både hög-nivå-aspekter på systemen och den övergripande strukturingen, uppdelningen i komponenter, komponenternas funktionalitet och hur de interagerar [Sha95]. Beskrivningen har tidigare oftast bestått av boxar och linjer där boxar representerar en komponent och linjer representerar någon form av kommunikation eller relation mellan komponenterna [Bas98]. Arkitekturella stilar har ofta representerats av fraser som "pipe-and-filter" eller "client-server" [Sha95]. Denna representation har sin fördel i att den på ett enkelt sätt ger en bild av systemet, men den klarar oftast inte av att svara på exakt vad komponenterna är, vad de gör, hur de beter sig, vilka andra komponenter de är beroende av och vad kopplingarna mellan komponenterna betyder [Bas98].

Medan arkitekturella beskrivningar använder hög-nivå-abstraktioner är de motsvarande implementationerna skrivna i vanliga programmeringsspråk. Att sätta ihop ett system från delsystem är en stor skillnad motför att programmera de underliggande algoritmerna och datastrukturerna. Det existerar med andra ord ett stort gap mellan krav och programmering. För att överbrygga detta gap behövs bättre modeller och notationer för de mellanliggande stegen [Sha96]. Dessa svagheter går, enligt Bass et al, att komma ifrån med hjälp av ett formellt språk för att beskriva arkitekturen, ett ADL [Bas98].

Ett ADL är ett formellt språk för att på ett otvetydigt sätt specificera ett systems arkitektur och syftar till att öka förståelsen och återanvändbarheten av arkitekturell design [SEI02]. Det är varken ett kravspråk, ett programmeringsspråk eller ett modelleringsspråk och skiljer sig från dessa främst på några punkter.

Skillnad mot kravspråk. Det skiljer sig från ett kravspråk genom att ett kravspråk beskriver problemområden och ett ADL beskriver lösningar [Bas98]. Krav delas ofta upp i mindre delar för att lättare kunna beskrivas och vissa kravspråk kan därför fungera bra för att beskriva arkitektur trots

att det inte var det ursprungliga målet med språken [Cle96].

Skillnad mot programmeringsspråk. Ett ADL skiljer sig från programmeringsspråk genom att då ett programmeringsspråk fokuserar på en lösning undviker ett ADL att binda sig vid en specifik lösning så länge som möjligt [Bas98]. Många programmeringsspråk kan dock representera arkitektur på ett relativt bra sätt. Ett exempel på detta är Ada som tillhandahåller möjligheten att titta på ett system via komponenternas gränssnitt, men som inte tillhandahåller någon möjlighet till analys på arkitekturell nivå [Cle96]

Skillnad mot modelleringsspråk. Ett ADL skiljer sig från modelleringsspråk då ADLer koncentrerar sig på att beskriva beteendet hos komponenter och modelleringsspråk koncentrerar sig på helheten [Bas98].

En existerande standardnotation för att representera arkitektur, ett standard ADL, skulle framhäva fördelarna med en genomtänkt arkitektur. Det skulle förbättra kommunikationen eftersom författaren och läsaren av en arkitekturell beskrivning skulle dela förståelsen för beskrivningen och alltså slippa förtydligande frågor. Det skulle förbättra möjligheten till analys av tidiga designval eftersom det skulle underlätta byggandet av verktyg för att utföra detta [Bas98]. Ett sådant språk skulle dessutom medföra en mer formell arkitekturbeskrivning vilken lättare följs och bibehålls eftersom den ses som mer auktoritär än en informell beskrivning [Cle96].

9.3.1 Beståndsdelar i ett ADL

Tekniken att använda ADLer för att representera och analysera arkitektur växer men är fortfarande i forskningsstadiet och de existerande ADLerna är relativt olika. Det finns därför många olika definitioner på vad som krävs av ett språk för att få kallas för ett ADL. Bass et al har definierat följande [Bas98]:

Förmedling av arkitekturen. Ett ADL måste kunna förmedla arkitekturen till alla intresserade parter genom att anpassa detaljnivån på beskrivningen. Alla strukturer en arkitektur kan ha måste gå att representera och komponenter, kopplare och deras typer måste identifieras för varje struktur.

Systemarkitekturprocess. Processen med systemarkitekturen, d.v.s. skapandet av arkitekturen, förfinandet och valideringen ska stödjas. Det måste även finnas regler över vad en fullständig och konsistent arkitektur innebär.

Arkitekturella stilar. Det måste finnas möjligthet att representera de mest vanliga arkitekturella stilarna.

Abstraktion. Ett ADL måste tillhandahålla strukturer som kan uttrycka arkitekturell information och samtidigt dölja icke-arkitekturell information.

Utökning. En bas för fortsatt implementation måste tillhandahållas. Det måste vara möjligt att addera information till specifikationen av ADLet.

Analys. Ett ADL måste antingen kunna stödja möjligheten till analys av arkitekturell information eller stödja möjligheten till snabb generation av implementationsprototyper.

Shaw et al har definierat följande önskvärda egenskaper hos ett ADL [Sha95]:

Möjlighet att modellera komponenter. Alla komponenter ska ha ett gränssnitt som definierar dess egenskaper, så som typ eller subtyp, funktionalitet m.m.

Möjlighet att modellera kopplare. Alla kopplare ska ha ett protokoll som specificerar dess egenskaper, så som typ eller subtyp, regler om vilka gränssnitt den jobbar mot, m.m.

Abstraktion och inkapsling. Abstraktion är nödvändigt för att kunna beskriva sammansatta komponenter och avancerade komponenter på ett relativt lättförståeligt vis. Abstraktion är även nödvändigt för att beskriva abstrakta egenskaper hos komponenter eller kopplare som inte implementeras direkt och för att kunna anpassa detaljnivån på beskrivningen av arkitekturen.

Typer och typkontroll. Typer måste kunna kontrolleras till exempel vid associering av komponenters aktörer med kopplares roller. Tillräcklig information måste finnas för att kunna kontrollera om komponenter får associeras med kopplaren .

Analysverktyg. Ett ADL ska vara ”öppet” för analysverktyg. Analyser som kan utföras är till exempel ett systems realtidsegenskaper.

Luckham och Vera har under arbetet med språket Rapide arbetat fram egenskaper de anser är krav för ett ADL [Luc95]:

Komponentabstraktion. Alla gränssnitt ska definiera vad komponenten tillhandahåller och vad komponenten kräver. Gränssnittet ska även definiera komponentens beteende i en form som tillåter analys och exekvering.

Kommunikationsabstraktion. Kopplingar, som definierar kommunikationen mellan komponenter får bara använda komponenternas gränssnitt och definiera kommunikation i en form som tillåter analys och exekvering.

Kommunikationsintegritet. Gränssnitt får bara kommunicera direkt om det finns en arkitekturell koppling mellan gränssnitten.

Dynamik: Det ska gå att modellera dynamisk arkitektur där antalet komponenter och kopplare kan variera under tiden för exekvering.

Tillfällighet och tid. Tillfälliga beroenden och oberoenden mellan gränssnitt och kopplare ska gå att beskriva.

Hierarkisk förfining. Både komponenter och kopplare ska kunna bytas ut mot subarkitekturer för att kunna forma nya arkitekturer.

Relativitet. Det måste finnas egenskaper för att relatera beteende mellan arkitekturer. Det ska till exempel gå att relatera arkitekturer på olika abstraktionsnivåer till varandra.

Även Shaw och Garlan har definierat vissa egenskaper som ett idealt ADL bör ha [Sha96b]. Dessa är:

Komposition. Det ska vara möjligt att beskriva ett system som en sammansättning, komposition, av oberoende komponenter och kopplare. Det måste även vara möjligt att dela upp ett komplext system i mindre, mer lätthanterliga delar.

Abstraktion. Det ska vara möjligt att beskriva komponenter och deras interaktioner på ett sätt som tydligt fastställer deras roller inom systemet. Abstraktion används i alla mjukvarusystem, men på den arkitekturella nivån behövs en annan typ av abstraktion då man till exempel ska kunna se att komponenter har en klient-server-relation. Denna relation är svår att upptäcka genom att bara titta på gränsnittet på procedursanrop.

Återanvändbarhet. Det ska vara möjligt att återanvända komponenter, kopplare och arkitekturella mönster i olika arkitekturella beskrivningar.

Heterogenitet. Det ska vara möjligt att kombinera multipla, heterogena arkitekturella beskrivningar och mönster i ett system. Det ska till exempel gå att definiera en komponent som kommunicerar med en annan komponent genom en pipa samtidigt som den har åtkomst till en delad databas. Det ska även gå att kombinera komponenter som är skrivna i olika språk.

Analys. Det ska vara möjligt att göra utförliga och varierande analyser av arkitekturella beskrivningar. Olika arkitekturer tillåter olika typer av analys och det ska vara möjligt att skräddarsy analystypen beroende på typ av arkitektur. I en pipe-and-filter-arkitektur ska det till exempel vara möjligt att analysera genomströmning, dödlägen och användning av resurser.

Likheter mellan de önskvärda egenskaperna
De ovan nämnda egenskaperna skiljer sig en hel del, men likheter mellan dem finns dock.

Abstraktion. Hos alla författare har abstraktion en viktigt del i ett ADL. Bass et al nämner vikten av att kunna uttrycka arkitekturell information och samtidigt gömma icke-arkitekturell information. De anser även att detaljnivån på den arkitekturella beskrivningen ska gå att anpassa beroende på vilken information som ska förmedlas, vilket även är en typ av abstraktion [Bas98]. Shaw med fler beskriver precis som Bass med fler vikten av abstraktion genom anpassning av detaljnivån och nödvändighet av att kunna beskriva arkitekturen på ett lättförståeligt sätt [Sha95]. Även Luckham och Vera nämner abstraktion men då i form av gränssnitt till komponenter och kopplare och möjligheten till att relatera arkitekturer på olika abstraktionsnivå till varandra [Luc95]. Shaw och Garlan beskriver det som möjligheten att tydligt kunna beskriva komponenter och deras interaktioner [Sha96b].

Komposition av komponenter. Författarna nämner även komposition av komponenter men med lite olika definitioner. Shaw och Garlan anser att det ska vara möjligt att beskriva ett system som en komposition av komponenter och kopplare och dessutom möjligt att dela upp systemet i mindre delar [Sha96b]. Luckham och Vera kallar det för hierarkiskt förfining och nämner att både komponenter och kopplare ska kunna bytas ut mot subarkitekturer för att kunna forma nya arkitekturer [Luc95].

Komponenter och kopplare. Shaw et al och Luckham och Vera diskuterar kring önskvärda egenskaperna ur ett komponent/kopplar-perspektiv där kommunikation endast får ske genom gränssnitt [Sha95], [Luc95]. Trots

att de andra författarna inte nämnt detta uttryckligen i samband med de viktigaste egenskaperna ett ADL ska ha, bygger hela deras resonemang på att en arkitektur beskrivs med hjälp av komponenter och kopplare [Bas98], [Sha96b].

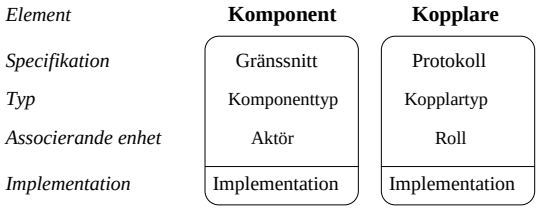
Analys. Bass et al, Shaw med fler och Shaw och Garlan beskriver alla möjligheten till att kunna utföra olika typer av analyser på arkitekturen som en viktig del i ett ADL. I [Luc95] nämner Luckham och Vera inte analys som en viktig del av ett ADL, men har dock skrivit en annan artikel angående analys av arkitekturer med hjälp av Rapide vilket tyder på att de ändå anser det som en viktig del av ett ADL [Luc95b].

Alla har dessutom som grundpelare att kommunikationen mellan de inblandade i utvecklingen av mjukvarusystem ska förbättras med hjälp av ett ADL.

9.3.2 UniCon

UniCon är ett ADL som skapats för att hjälpa mjukvarudesigners att definiera arkitektur genom att göra nödvändiga abstraktioner. En annan viktig aspekt i UniCon är möjligheten att göra en enkel överföring från arkitektur till körbar kod.

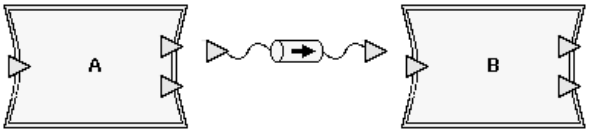
UniCon beskriver system med hjälp av elementen komponenter och kopplare. Varje element har ett namn, en specifikation, en typ, en mängd associerade enheter och en implementation, se figur 13. En *komponent* är ett element



Figur 13: En komponent och en kopplare i UniCon [Sha96b]

som utför någon typ av beräkning. Komponentens specifikation kallas för gränssnitt och gränssnittet definierar dess egenskaper och en implementation som kan vara primitiv eller sammansatt. Ett gränssnitt måste innehålla komponentens typ, begränsningar och aktörer. Aktören bestämmer hur komponenten ska interagera med omvärlden.

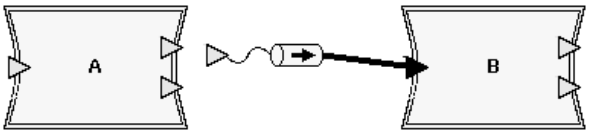
En kopplares specifikation kallas för protokoll. Protokollet definierar kopplarens egenskaper och en implementation. Dessa egenskaper är till exempel kopplarens typ eller subtyp och regler om vilka gränssnitt den fungerar tillsammans med. I protokollet finns även kopplarens roller specificerade. En roll bestämmer hur kopplaren ska förmedla meddelanden. Kopplarens associerade enheter är komponenters aktörer och komponentens associerade enheter är kopplares roller [Sha96b].



Figur 14: Ett Unix-filter representerat i UniCon av två icke ihopkopplade komponenter och en kopplare [Cs03]

Figur 14 är ritad med en grafisk UniCon-editor och visar två komponenter, A och B, som ska föreställa Unix-filter. Båda har tre aktörer, ritade som trianglar. Aktörerna till vänster i komponenterna representerar ”standard in” och de till höger i komponenterna ”standard out” och ”standard error”. Mellan komponenterna finns en kopplare som ska representera en Unix-pipa. Kopplaren har två roller, den till vänster representerar var den tar meddelanden från och den till höger var den ”slänger dem”.

I figur 14 finns ingen interaktion mellan komponenterna och kopplaren. För att det ska kunna finnas måste en aktör associeras med en roll, vilket är gjort i figur 15 [Cs03].



Figur 15: Ett Unix-filter representerat i UniCon av två komponenter och en kopplare. Kopplaren är ihopkopplad med komponent B [Cs03]

Både komponenter och kopplare kan vara primitiva eller sammansatta. Primitiva komponenter kan vara skrivna i ett programspråk eller ett shellscript. Sammansatta komponenter definierar konfigurationer med en notation som är oberoende av programspråk. Notationen måste tilllåta identifikation

av ingående komponenter och kopplare, matcha komponenters aktörer med kopplares roller och kontrollera att sammansättningen av komponenter och kopplare är tillåten [Sha96b].

Ett sammansatt element innehåller:

- en lista över ingående komponenter och kopplare
- instruktioner över sammansättning av element
- abstraktionsmappning

UniCon tillhandahåller tre sorters typkontroll. Kontroll av komponenter och kopplares typer, kontroll av komponenters och kopplares överensstämmande med en stil och kontroll av associeringen mellan komponenters aktörer och kopplares roller.

Nedan är ett exempel på implementationen av en primitiv komponent.

```
COMPONENT sort
   INTERFACE IS
   TYPE Filter
   PLAYER input IS StreamIn
      SIGNATURE ("line")
      PORTBINDING (stdin)
   END input
   PLAYER output IS StreamOut
      SIGNATURE ("line")
      PORTBINDING (stdout)
      END output
   PLAYER error IS StreamOut
      SIGNATURE ("line")
      PORTBINDING (stderr)
      END error
   END INTERFACE

---------------------------------

   IMPLEMENTATION IS
      VARIANT sort IN "sort"
         IMPLTYPE (Executable)
         INITACTUALS ("-f")
      END sort
```

```
   END IMPLEMENTATION
END sort
```

9.3.3 ROOM

ROOM är ett grafiskt ADL som används inom telekommunikationsindustrin. Det är baserat på en mängd modelleringsabstraktioner av aktiva objekt som kallas aktörer [Bas98].

ROOM tillhandahåller två typer av diagram. Aktörsdiagram beskriver den hierarkiska strukturen på aktörerna och alla bindningar mellan aktörerna. Ett aktörsdiagram definierar både den interna strukturen och de externa gränssnitten. *ROOMcharts* är definitioner av aktörernas beteende och identifierar en aktörs möjliga tillstånd, signaler som orsakar tillståndsförändringar och handlingarna som utförs på förändringarna. ROOM-modellen innehåller, enligt Rumpe et al, följande [Rum99]:

Aktörer. En aktör är ett element som utför någon form av beräkning. Aktörer kan skicka och ta emot signaler genom en eller fler portar och kan organiseras hierarkiskt med arv.

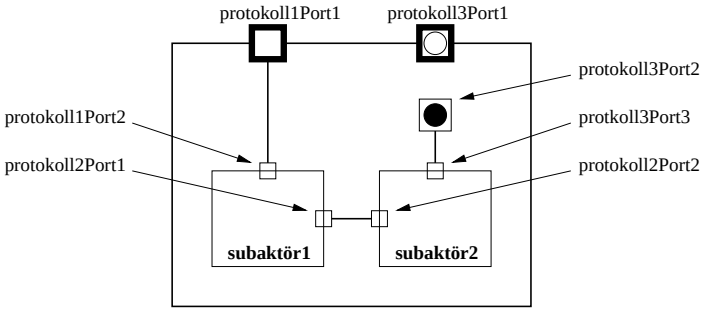
Protokoll. Ett protokoll definierar en mängd tillåtna signaler som skickas mellan två ihopkopplade aktörer. Signalerna definieras som ingående eller utgående i förhållande till protokollet.

Portar. En mängd namngivna portar definierar gränssnittet på aktörer. Portarna placeras där en aktör tillhandahåller eller kräver någon form av service. Portarna tillåter signaler i båda riktningarna. En port som är associerad med ett protokoll skickar protokollets utsignaler och tar emot protokollets insignaler.

Konjugerade portar. För att kunna koppla ihop två portar måste den ena vara konjugerad. Den är då konjugerad med avseende på protokollet det är associerat med. En konjugerad port tar emot de utsignaler och skickar de insignaler som protokollet definierat.

Kopplare. En kopplare sätter upp en koppling mellan två portar. Den binder en aktörs port med en annan aktörs konjugerade port med ett kompatibelt protokoll.

Figur 16 är ett enkelt exempel för att visa ROOM-notationen. Figuren visar:



Figur 16: Ett enkelt system representerat med ROOM-notation [Bas98]

- En huvudaktör med två subaktörer, *subaktör1* och *subaktör2*
- en vidarebefodringsport, *protokoll1Port1*
- en slutport, *protokoll3Port1*
- en intern port, *protokoll3Port2*
- fyra portar tillhörande subaktörerna, *protokoll2Port1*, *protokoll1Port2*, *protokoll2Port2* och *protokoll3Port3*

Vidarebefordringsporten, *protokoll1Port1*, används för att skicka signaler mellan *subaktör1* och en extern aktör, via huvudaktören, medan slutporten, *protokoll3Port1*, används för kommunikation mellan huvudaktören och andra aktörer. Den interna porten, *protokoll3Port2*, tillhandahåller mekanism för kommunikation mellan huvudaktören och subaktörerna och då speciellt *subaktör2*. Typerna på subaktörernas portar göms av notationen eftersom det inte är viktigt att veta om dessa är slutportar eller vidarebefodringsportar så länge de sköter sina åtaganden enligt protokollen [Bas98].

Portarnas protokoll definierar, som tidigare nämnts, två typer av signaler, inkommande och utgående. Nedan är ett enkelt exempel på hur detta implementeras.

```
PROTOCOL CLASS Protocoll
   SERVICE BasicCommunication
   IN MESSAGE
   {
      DEFINE InSignal1 ISA Null;
      DEFINE InSignal2 ISA Boolean;
```

```
   } /* end of in message */
   OUT MESSAGE
   {
      DEFINE OutSignal1 ISA Integer;
      DEFINE OutSignal2 ISA String;
   } /* end of out message */
; /* end of Protcol */
```

En av ROOMs fundamentala principer är den operationella modellen. Så fort en ROOM-modell är tillräckligt beskriven kan den köras. Detta för att kunna validera krav och demonstrera beteendet på modellen. ROOM-modeller kan alltså fungera som systemprototyper tidigt i en utvecklingscykel [Bas98].

När en modell är fullständigt specificerad kan den kompileras till ett körbart system. Den måste dock utökas med aktörskroppar skrivna i något programmeringsspråk. På grund av dessa egenskaper kan ett system analyseras för att kontrollera run-time-defekter redan när bara lite av funktionaliteten har specificerats och det finns heller inget konceptuellt gap mellan specifikation, design och körning [Bas98].

UML + ROOM

Inom forskarvärlden råder viss förhoppning om att den rådande diskussionen kring vilket ADL som är bäst och vilken terminologi som är korrekt ska lösas genom att använda Unified Modeling Language (UML) som en startpunkt i att utveckla ett standard ADL. Detta skulle vara möjligt eftersom UML är en väl accepterad OMG-standard[16] med många egenskaper som passar för arkitekturell beskrivning, så som abstraktion, arv och flera olika typer av diagram (klassdiagram, tillståndsdiagram, komponentdiagram).

UML saknar dock en del egenskaper som är önskvärda för att kunna användas framgångsrikt som ett ADL. En nackdel är att UMLs komponentdiagram inte är utvecklade för att representera den logiska uppdelningen av ett system i återanvändbara subsystem. UML tillhandahåller heller inte någon möjlighet att se kopplare som egna objekt. På grund av detta försöker forskare koppla ihop UML med ROOM som har dessa egenskaper, men som bland annat saknar den vida acceptans som UML har [Rum99].

[16] *Object Management Group* är en organisation som fastställer riktlinjer och detaljerade specifikationer för att tillhandahålla gemensamma ramverk för mjukvaruindustrin (www.omg.com).

10 Realisering av arkitekturen

Att beskriva hur en arkitektur bör byggas, vilka olika stilar som kan användas, hur dessa fungerar och när dessa är lämpligast att använda är relativt enkelt jämfört med att utforma en lämplig arkitektur. Arkitekten måste vid varje val ta hänsyn till alla krav som finns på det kommande systemet. Eftersom kraven på systemet kommer från olika inblandade parter så som användare, verksamhetsledning, utvecklare, de som sköter underhållet av systemet, marknaden m fl. så kan det ofta krävas att något eller några krav prioriteras framför andra. I slutändan kommer alla dessa inblandade se på systemet utifrån olika synvinklar. Som exempel kommer användarna säkerligen att se på systemet utifrån dess funktionalitet och dess användbarhet. Verksamhetsledningen är kanske intresserade av hur personalen anpassar sig till systemet och hur det leder till en effektivare verksamhet. De kommer antagligen även att relatera resultaten till den kostnad som införandet av ett nytt system inneburit. Att utvärdera resultatet är därmed inte en trivial uppgift. Det kan även vara svårt för alla inblandade att enas om någon form av bedömning av resultatet. Har utvecklingen av det nya systemet resulterat i ett lyckosamt resultat eller har projektet misslyckats? Att besvara denna fråga är oerhört komplext då det finns många aspekter att ta hänsyn till. Ett lyckosamt resultat måste kanske inte innebära ett fullständigt korrekt resultat och ett till synes misslyckat system kanske ändå baseras på en så pass bra systemlösning att det med viss omarbetning och förbättring kan resultera i ett fullt dugligt system.

I detta avsnitt kommer vi ta upp frågeställningar kring problemet med att realisera det önskade systemet. Vi kommer även att beskriva hur resultatet kan mätas i form av olika kvalitetsattribut.

10.1 Kvalitetsattribut

Arbetet med utvecklingen av ett kommande system har i inledande skedet ett antal krav att arbeta mot, vilka det ofta råder konflikt emellan. Det är arkitektens uppgift att hantera dessa konfliktande krav på bästa sätt och därefter utforma den lämpligaste arkitekturen. Att skapa en bra arkitektur är med andra ord att skapa en bra balans mellan kraven. Denna balans skapas olika beroende på vilket arkitekturteam som tilldelas uppgiften och utifrån vilka förutsättningar de arbetar. Samma mängd krav kan alltså resultera i olika systemlösningar. Att därefter avgöra vilken som är den mest lämpliga är inte alltid så trivialt. Det kan därför vara bra att utvärdera systemet efter ett antal kvalitetsattribut och se hur det färdiga systemet påverkar dessa. Enligt Bass, Clements och Kazman kan dessa kvalitetsattribut delas in i

två huvudsakliga kategorier, de som är observerbara när systemet körs och de som ej går att observera när systemet körs. Kvalitetsattribut inom den första kategorin, de observerbara, mäter egenskaper så som systemets prestanda, dess funktionalitet, säkerhet. Kvalitetsattribut i den andra kategorin, de icke observerbara, är inte lika påtagliga. Exempel på egenskaper som dessa kvalitetsattribut mäter är modifierbarhet, portabilitet, utvecklingskostnad [Bas98].

Att fastställa hur systemet uppfyller ett kvalitetsattribut i den ena kategorin säger ingenting om hur systemet uppfyller kvalitetsattributen i den andra kategorin. Om ett system uppfyller alla de funktionella kraven kan det mycket väl ha kostat mycket mer än den givna budgeten att utveckla eller kanske är omöjligt att modifiera. Detsamma gäller kvalitetsattribut inom samma kategori. Ett system kan t.ex. ha bra tillgänglighet, med andra ord sällan krascha, medan det å andra sidan är opålitligt och producerar dåliga resultat [Bas98]. Dock kan uppfyllandet av något kvalitetsattribut ha effekt på möjligheten att uppfylla ett annat.

För att vidare beskriva kvalitetsattributen har Bass et al delat in kvalitetsattributen i fyra olika klasser där den första klassen motsvarar de observerbara kvalitetsattributen och de övriga tre en uppdelning av de icke observerbara kvalitetsattributen.

- systemets egenskaper som kan observeras när systemet körs och används
- systemets egenskaper som inte kan observeras när systemet körs och används
- affärskrav eller affärsmål som påverkats av arkitekturen
- arkitekturens utformning och dess inre påverkan på projektet t.ex. implementationen av systemet

Nedan presenteras ett antal olika kvalitetsattribut inom de fyra olika klasserna.

10.1.1 Kvalitetsattribut observerbara vid körning

Prestanda. Med systemets prestanda menas dess svarstid vid olika händelser eller det antal händelser som utförs inom ett visst tidsintervall. Prestanda uttrycks ofta i form av antal transaktioner per tidsenhet. Eftersom kommunikation mellan olika systemenheter oftast tar längre tid än beräkningar så

handlar prestanda mycket om interaktionen och kommunikationen mellan de olika delarna i systemet, vilket är en oerhört viktig del av systemarkitekturen. Prestanda är ett oerhört viktigt kvalitetsattribut främst när det gäller distribuerade system där systemets delar kan vara placerade på olika fysiska delar. Länge har prestanda varit den viktigaste kvaliteten i systemutvecklingssammanhang men då förhållandet mellan pris och prestanda på hårdvara har förbättrats och kostnaderna för utveckling av mjukvara har ökat, så har andra egenskaper blivit lika viktiga i dessa sammanhang.

Säkerhet. Säkerhet är ett mått på hur bra systemet kan stå emot obehörigas försök att använda det, utan att ge försämrad service till dess behöriga användare. Systemet bör kunna motstå olika typer av attacker som utförs på systemet. Att upptäcka attacker och försvara sig mot dem kan kräva olika arkitekturella lösningar, så som en autentiseringsserver för att hindra obehöriga från att nå systemet, nätverksmonitorer för att övervaka och logga trafiken och händelser på systemets nätverk eller sätta upp brandväggar.

Tillgänglighet. Ett systems tillgänglighet mäter hur stor del av tiden som systemet är tillgängligt för användarna. Tillgängligheten, α, mäts utifrån medeltiden mellan systemkraschar (Mean Time Between Failures, MTBF) och medeltiden över hur lång tid det tar för systemet att återhämta sig efter en krasch (Mean Time To Repair, MTTR) [Bas98], enligt följande formel:

$$\alpha = \frac{\text{MTBF}}{\text{MTBF+MTTR}}$$

Pålitlighet är en närbesläktad egenskap. Med pålitlighet avses endast medeltiden mellan systemets kraschar. Både pålitlighet och tillgänglighet är kvaliteter på arkitekturell nivå. Det kan handla om att tillhandahålla reservkomponenter som ska möjliggöra att systemet alltid kan nås trots en systemkrasch eller att utforma systemets komponenter så att dessa är lätta att modifiera och enkla att felsöka i.

Funktionalitet. Hur väl systemet uppfyller arbetet som det är tänkt att göra är ett mått på funktionalitet. Systemets funktionalitet avslöjar hur bra komponenterna sköter det ansvar de blivit tilldelade. Utan påverkan från övriga kvalitetskrav skulle den önskade funktionaliteten vara relativt lätt att uppfylla. Systemet skulle i så fall t.ex. kunna byggas som en enda komponent utan någon inre struktur, vilket antagligen skulle medföra att systemet svårt skulle kunna modifieras, underhållas, felsökas mm. Utformningen av ett systems funktionalitet måste alltså relateras till alla de övriga kvalitetskraven.

Användbarhet. Användbarhet kan delas upp i följande delar:

- **Förmåga att lära** - hur snabbt lär sig användaren att använda systemet?
- **Effektivitet** - är svarstiderna på användarnas förfrågningar godtagbara?
- **Förmåga att få användaren att minnas** - kan användaren komma ihåg systemets funktioner?
- **Förmåga att undvika fel** - kan systemet förutspå och förebygga vanliga felaktiga handlingar utförda av användaren?
- **Felhantering** - hjälper systemet användaren att återfinna sig vid fel?
- **Förmåga att möta användarens krav** - underlättar systemet använd-arens arbete?

Att avgöra hur systemets funktioner ska presenteras, t.ex. knappar eller menyer är inte ett arkitekturellt val. Det arkitekturella i detta sammanhang är att försäkra sig om att alla funktioner som önskas av systemet finns tillgängliga för användaren och att användarens instruktioner till systemet tas om hand på rätt sätt och fördelas till rätt systemkomponenter.

10.1.2 Kvalitetsattribut icke observerbara vid körning

Modifierbarhet. Modifierbarhet kan vara det kvalitetsattribut som kräver mest av systemarkitekturens utformning. Systemarkitekturen har ansvar för att modifieringar kan ske kostnads- och tidseffektivt. Systemarkitekturen ska definiera komponenter och dess ansvar men även hur dessa ska förändras. Förändringar av komponenter orsakas oftast av förändringar i verksamheten där systemet används. Förändringar kan enligt Bass et al delas in i följande grupper [Bas98]:

- utöka eller förändra systemets funktioner
- ta bort befintliga funktioner
- anpassa systemet till nya operatoriska enheter i dess omgivning
- omstrukturera

Portabilitet. Med portabilitet menas systemets möjlighet att köras på olika underliggande plattformar. Typer av plattformar kan vara olika hårdvara, mjukvara eller en kombination av dessa. De plattformsspecifika funktionerna kan kapslas in i arkitekturen i form av ett portabilitetslager, dvs. en mängd mjukvarutjänster som ger systemets applikationer ett abstrakt gränssnitt till dess underliggande plattform. Detta gränssnitt är då bestående även om själva portabilitetslagret förändras beroende på vilken plattform systemet körs på.

Återanvändning. Att designa ett system så att systemstrukturer och komponenter i framtiden kan återanvändas är den enkla innebörden av återanvändning. Återanvändning på arkitekturell nivå handlar om att utforma återanvändbara komponenter och återanvändbara arkitekturella ramverk, såsom referensarkitektur, domänspecifik arkitektur eller product-line-arkitektur.

Integrerbarhet. Att utveckla de olika systemkomponenterna separat och därefter integrera dessa är inte alltid så lätt. Förutom att komponenterna ska kunna kommunicera och interagera ska de fungera korrekt tillsammans. Komponenternas externa mekanismer och protokoll måste överensstämma med de gränssnitt som systemarkitekturen planerat för dessa. En väl formulerad specifikation av gränssnitten är alltså viktig på arkitekturell nivå. En annan närbesläktad kvalitet är *interoperabilitet*. Till skillnad från integrerbarhet som endast avser komponenters integrering inom ett system avser interoperabilitet att mäta ett systems förmåga att integreras med andra system.

Testbarhet. Med testbarhet menas systemets möjlighet att genom tester, oftast i form av speciella testprogram, demonstrera sina fel och brister. Testbarhet innebär att kända fel ska kunna framkallas genom en enda exekvering. Om ett system har bra testbarhet ska det vara möjligt att kontrollera varje komponents input, dess interna tillstånd och därefter kunna observera dess output. På arkitekturell nivå kan detta handla om dokumentation, ansvarsfördelning över de olika komponenterna mm. .

10.1.3 Kvalitetsattribut i form av affärskrav

Förutom de nyligen nämnda kvalitetsattributen som har direkt samröre med systemet finns även ett antal kvalitetsattribut i form av affärsmål som starkt påverkar utformningen av systemarkitekturen. De huvudsakliga kvalitetsattributen i detta sammanhang är inom tid och kostnad.

Time-to-market. Om det finns konkurrerande påtryckningar eller om utvecklingstiden är tidsbegränsad är tiden en viktig faktor i systemstruktureringen. Genom att köpa in komponenter som redan finns på marknaden, COTS eller återanvända redan utvecklade komponenter kan utvecklingstiden reduceras. Det är då viktigt att utforma en systemarkitektur där dessa färdiga komponenter lätt kan integreras.

Kostnad. Utvecklingen av ett system har vanligtvis en budget som inte får överskridas. Olika typer av arkitekturer medför olika kostnader. Ett exempel på arkitekturellt val med avseende på kostnad kan vara valet av en viss teknik. Att basera systemarkitekturen på en teknik där kunskap om denna redan finns är billigare än att använda en teknik där kunskapen inte finns. I detta fall måste kunskapen införskaffas. t.ex. genom utbildning eller genom rekrytering av ny personal.

10.1.4 Kvalitetsattribut i form av arkitekturens utformning

Eftersom systemarkitekturen ska fungera som en plan över hur det fortsatta utvecklingsarbetet ska ske är det viktigt att den kan vägleda utvecklare och andra inblandade parter på ett lättförståeligt och överblickbart sätt. Det finns ett antal kvalitetsattribut för att mäta denna förmåga.

Konceptuell integritet. Med konceptuell integritet menas systemarkitekturens underliggande vision eller stil som förenar de olika nivåerna av design. Liknande arkitekturella delar ska utformas på liknande sätt. Detta medför att utvecklare och de som ska underhålla systemet lättare kan känna igen stilar och mönster och på så vis få bättre förståelse för dessa.

Korrekt och komplett. Detta är en ganska självklar kvalitet. Systemarkitekturen ska vara utformad korrekt och komplett. Felaktigheter i systemarkitekturen kan få förödande konsekvenser om de upptäcks sent. Det är även viktigt att systemarkitekturen omfattar alla val på arkitekturell nivå, så att inga viktiga arkitekturella beslut fattas på lokal nivå, inom utvecklingsgruppen för en viss komponent.

Byggbarhet. Systemarkitekturen måste vara utformad så att den är möjlig att realisera av den tilltänkta utvecklingsgruppen och inom en godtagbar tidsperiod. Den måste även vara öppen för förändringar allt eftersom utvecklingen av systemet fortlöper.

10.2 Utvecklingsarbetet

Förutom de arkitekturella besluten har arbetsgången från den planerade arkitekturen till det färdiga systemet stor påverkan på resultatet. Det finns många orsaker till det lyckade eller misslyckade resultatet.

10.2.1 Utvecklingsprocessen

Systemutvecklingsprojekt, stora som små, misslyckas ofta på grund av brister i utvecklingsprocessen och arbetsgången kring detta. Enligt Philippe Krutchen kan de misslyckade utvecklingsprocesserna avslöja sina brister i ett antal symptom, t.ex. moduler som ej passar ihop. De enskilda symptomen kan gå att korrigera men det kan vara för sent för den totala utvecklingsprocessen att återhämta sig. Olika systemutvecklingsprojekt misslyckas av en mängd olika anledningar, men enligt Krutchen är det ofta en kombination av följande brister som i slutändan leder till ett misslyckat resultat [Kru01].

- kravhanteringen sköts ad-hoc
- tvetydig kommunikation
- instabil arkitektur
- projektet är allt för komplext
- krav, design och implementation innehåller dold inkonsistens
- testningen är otillräcklig
- projektets tillstånd bedöms subjektivt
- misslyckande i att förebygga attacker
- ändringar förespråkas okontrollerat

Att förebygga brister i projektet leder till en större sannolikhet för ett lyckat resultat. Det finns olika modeller och standarder för att förbättra utvecklingsprocessen. Dessa går ut på att skapa en mer kvalitativ utvecklingsprocess för att på så vis med större säkerhet uppnå ett slutresultat, en produkt, med hög kvalitet. Exempel på sådana kvalitetsmodeller är *Capability Maturity Model* (CMM), ISO 9000 och *Rational Unified Process* (RUP).

10.2.2 Inverkande faktorer

Utformningen av systemarkitekturen består till stor del av ett antal avvägningar och beslut där, förutom kraven på systemet, diverse faktorer spelar en avgörande roll. I de arkitekturella besluten måste faktorer och krav vägas in. Faktorer och krav motsträvar ofta varandra och kompromisser måste då göras. Det är dock viktigt att förstå att faktorer även kan inverka stödjande och möjliggöra bättre systemlösningar. Mycket kunskap är exempel på en stödjande faktor, ju mer kunskap som finns inom projektet desto bättre systemlösningar borde gå att uppnå.

Håkan Enquist har beskrivit några grupper av faktorer som kan ha inverkan på de arkitekturella besluten [Enq02].

- **Ekonomiska** - nytta/kostnad, utvecklingskostnader mm.
- **Ledningsmässiga** - ansvar och befogenheter för utformning och användning av resurser, ledningsformer (centraliserad/decentraliserad), samordning mm.
- **Verksamhetsmässiga** - omorganisering, dynamik i verksamheten mm.
- **Organisatoriska** - resurstillgång/resursbrist, organisatorisk flexibilitet mm.
- **Kulturella** - kunskaper, attityder, sedvänjor, normer, traditioner
- **Verksamhetsutveckling/Förändring** - flexibilitet och förändringsbarhet
- **Tekniska** - teknik möjliggör eller förhindrar

Nedan förklarar vi ett antal faktorer som kan ha både positiv och negativ inverkan på de arkitekturella besluten.

Budget
Budget är en organisatorisk faktor. En budget är den finansiella planen av systemutvecklingsprojektet. Allokering av egen personal eller externa konsulter måste ske i enlighet med budgeten. Även eventuella inköp av moduler måste ske i enlighet med budgeten. Budgeten har nära samröre med tidsåtgången eftersom en förlängning av projektet måste ske inom ramen för förändringar av budgeten. Valet av en viss teknik kan som vi tidigare nämnt vara en kostnadsfråga.

Kunskap och erfarenhet

För att hitta de bästa lösningarna i förhållande till krav och andra påverkande faktorer krävs personal med goda kunskaper och erfarenhet. Vid den övergripande systemstruktureringen måste man ha de kunskaper som krävs för att se systemstrukturen ur ett helikopterperspektiv. Som komplement till de övergripande kunskaperna krävs specialistkunskaper inom diverse områden. Det systemarkitekturella arbetet kräver därmed kompetenser inom en stor mängd olika områden [Are01].

Gamla system

När ett system har varit väldigt viktig för en verksamhet kan det vara svårt att byta ut detta även om tekniken det är byggt med är föråldrad. Dessa system, ofta kallade legacysystems, används ofta även efter att en organisation bytt till en ny, annorlunda arkitektur som till och med kan vara inkompatibel med den förra. Dessa system är av stordatortyp och det ligger en stor svårighet i vad som ska göras med dessa. Beslut måste tas om ifall de bör behållas, förändras eller helt bytas ut. För att kunna möta marknadens krav på till exempel flexibilitet och nya teknologier, kan det vara attraktivt att byta ut systemen, men att göra detta är i regel ett väldigt riskfyllt och dyrt projekt. De gamla systemen får alltså ofta vara kvar, helt enkelt beroende på att de fungerar [Lid99].

Andra system

En faktor som kan skapa stora problem vid arkitekturarbete är ifall det ska förekomma kommunikation med andra system. Detta är vanligt idag i och med att många myndigheter är ”ihopkopplade” och att många företag kommunicerar elektroniskt med underleverantörer, banker, återförsäljare, etc.

Personal

Medlemmar i ett projekt kanske inte känner känner varandra sedan tidigare och gruppen kan bestå av personer i olika åldrar, kön och med olika social eller kulturell bakgrund. Detta kan både vara bra och dåligt. Det kan medföra risk för konflikter eftersom medlemmarna då kan ha svårigheter att kommunicera och ha dålig förståelse för varandra. Klarar man av att hantera eventuella konflikter och får medlemmarna att kommunicera obehindrat så borde dock det faktum att projektmedlemmarna är olika och då även tänker på olika sätt leda till en bättre helhetssyn och medföra att problem förhoppningsvis kan angripas ur olika synvinklar [Loo99].

Personalens engagemang kan påverka både arbetet och de andra i gruppen. Människor som inte känner sig engagerade tenderar att smitta av sig på and-

ra, vilket kan medföra en dålig stämning i en grupp. På samma sätt fungerar det omvända.

Det kan vid arbete i grupp vara bra att sätta upp en "riskplan" ifall någon av personerna i projektet blir sjuka, byter arbetsplats eller liknande så att störningen av detta inte blir alltför stor.

Verktyg
För att kunna vara effektiv bör en utvecklingsprocess stödjas av ett eller flera lämpliga verktyg. Ett verktyg kan även påverka processen negativt genom att begränsa arkitekturen och utvecklingen efter sin funktionalitet. Valet av verktyg är med andra ord ytterst viktigt. Om det redan finnas ett verktyg inom organisationen och kan det kännas som ett naturligt val eftersom det redan är välbekant för de inblandade.

Tid
I de olika arkitekturella valen kan tid vara en avgörande faktor. Vid tidsbegränsade projekt kan det vara nödvändigt att t.ex. till så stor utsträckning som möjligt återanvända moduler eller kanske köpa in färdiga moduler. Noggranna tidsuppskattningar är oerhört avgörande för att kunna genomföra en hållbar projektplanering. När de olika komponenterna fördelas på olika utvecklingsteam måste en tiduppskattning av utvecklingstiden göras så exakt som möjligt. Tid är en genomsyrande faktor i hela projektet och kan ha både positiv och negativ inverkan. Tidsbegränsning kan vara positivt om denna fungerar som en sporre.

Projekt och planering
Enligt Turban, McLean och Wetherbe klarar få informationssystemprojekt av att färdigställas inom beräknad tid och budget. Projektplaneringar stämmer sällan överens med verkligheten eftersom tidskraven och resurskraven oftast underskattas. Ofta blir efterfrågad funktionalitet i systemet utesluten för att projektet ska klara ett visst tidskrav eller hålla sig inom en viss budget. Sådan uteslutning av funktionalitet kan leda till misstyckande hos användarna. Denna funktionalitet måste då läggas till i systemet i senare versioner. Bättre projektplanering kan förhindra att funktionalitet ofrivilligt måste uteslutas [Tur99].

Dokumentation
Att ha bra rutiner för dokumentation är viktigt i många avseenden. Eftersom arbetet med systemarkitekturen ska fungera som en planering över det fortsatta utvecklingsarbetet och systemets underhåll och utveckling är det viktigt

att formuleringar och beskrivningar är begripliga och inte kan missuppfattas. Dokumentation är till för att skapa förståelse för systemets uppbyggnad. Dokumentation ingår i alla faser av systemutvecklingen, från verksamhetsmässiga diskussioner till diskussioner om specifika tekniska tillämpningar. Med hjälp av ordentlig dokumentation kan man bl.a. få förståelse för projektets utveckling, underlätta samarbete med andra projekt, enklare klara av personalomsättning och lättare se möjligheter till återanvändning [Niv02].

11 Reflektioner kring litteraturstudien

Denna litteraturstudie kan förefalla aningen sporadiskt uppdelad i delavsnitt. Detta kan förklaras av ämnets oerhörda komplexitet. Eftersom ämnet innehåller många olika aspekter och synvinklar är det svårt att välja ut lämpliga angripssätt för en så pass generell och övergripande litteraturstudie som detta är. Vi har dock försökt att ta upp de moment och aspekter inom informationssystemarkitektur som vi under litteraturstudiens gång ansett höra hemma i denna uppsats. Efter avslutad litteraturstudie drar vi slutsatsen att det krävs mycket kunskap och erfarenhet av ämnet för att kunna avgöra vilka aspekter och delavsnitt som är de bäst lämpade att ta upp i en litteraturstudie som denna. Det är även nödvändigt att kritiskt kunna avgöra relevansen i den mängd avhandlingar och artiklar som finns i ämnet, då vi upplever att det i stor utsträckning handlar om aktuella trender och personliga åsikter.

Del II
Praktisk studie

I denna del av rapporten ska vi undersöka problemet kring hur man i arbetet med arkitekturen lyckas realisera önskningarna på det kommande systemet. Det är intressant att studera vilka faktorer som påverkar utvecklingsprocessen och hur dessa har en positiv eller negativ inverkan på realiseringen av den önskade arkitekturen. Utifrån dessa faktorer bör man kunna avgöra om systemutvecklingen medfört ett lyckat resultat trots att alla önskningar eventuellt inte realiserats fullständigt.

Första steget i vår undersökning av detta problem är att utforma en utvärderingsmodell för att identifiera de inverkande faktorerna i ett verkligt systemutvecklingsprojekt. Denna utvärderingsmodell ska vara generell och därmed kunna användas på ett godtyckligt befintligt systemutvecklingsprojekt. Det andra steget i vår studie av problemet är att testa utvärderingsmodellen. Detta ska ske genom att undersöka ett befintligt systemutvecklingsprojekt och dess utvecklingsprocess med hjälp av modellen. Utifrån denna undersökning ska modellen utvärderas.

12 Metod

Följande delar klargör och motiverar val av metod för den undersökning som utförts och för den utvärderingsmodell som skapats.

12.1 Metodteori

Den kunskapsutvecklande processen är till stor del styrd av grundläggande föreställningar om vetenskap. Både formulering av problem och sätt att samla in och analysera data formas till stor del av valt metodsynsätt. Olika metodsynsätt gör skilda antaganden om det som ska studeras och problem angrips på olika sätt. Det finns i dag två vetenskapliga huvudinriktningar: *positivism* och *hermeneutik* som står till grund för de olika metodsynsätten *kvalitativ metod* och *kvantitativ metod* [Pat94].

12.1.1 Positivism

Auguste Comte, en fransk sociolog verksam under mitten av 1800-talet, gav namnet åt positivismen. Han hade fysiken som förebild och ville skapa en

vetenskaplig metodologi som var lika för alla vetenskaper. Den kunskap som söktes skulle vara verklig och tillgänglig för våra sinnen och vårt förnuft och dessutom logiskt prövbar. Inom positivismen är det viktigt att endast beakta det som går att observera med sinnena och att förklara och beskriva verkligheten som anses vara lika för alla. Egna åsikter och förutfattade meningar ska helt bortses från och undersökningar inriktas på det som är mätbart och kvantifierbart. Inom naturvetenskapen har detta förhållningssätt varit det dominerande sedan en lång tid tillbaka [Pat94].

12.1.2 Hermeneutik

Hermeneutiken har sin grund i tolkning av texter och utgör ofta en motvikt mot det positivistiska synsättet. Till skillnad från positivisterna är hermeneutikerna inte intresserade av att förklara företeelser utan menar att människor har intentioner och avsikter som yttrar sig i språk och handling. Dessa intentioner går att tolka och förstå innebörden av. En hermeneutisk forskare närmar sig forskningobjekt subjektivt utifrån sin egen förförståelse och försöker se helheten i forskningsproblemet [Pat94].

12.1.3 Kvalitativ/kvantitativ metod

I debatten mellan positivister och hermeneutiker har positivismen fått stå för kvantitativa, statistiska hårddatametoder för analys och hermeneutiken fått stå för kvalitativa förståelse- och tolkningssystem. Beteckningarna ”kvantitativ” och ”kvalitativ” syftar, något förenklat, på hur man väljer att samla in, bearbeta och analysera data. Med *kvantitativt* inriktad forskning menas forskning som använder sig av statistiska bearbetnings- och analysmetoder. Med *kvalitativt* inriktad forskning menas forskning som använder sig av verbala analysmetoder [Pat94].

12.1.4 Undersökningsupplägg

Det vanligaste uppläggen för undersökningar är *survey*, *fallstudie* och *experiment*.

Vid en *survey* sker undersökningen på en större avgränsad grupp. Om resurser finns för att undersöka en hel population görs en totalundersökning. Annars görs urval genom stickprov och om urvalet gjorts på rätt sätt ska resultatet som erhålls vara representativt för populationen.

Vid en *fallstudie* görs en undersökning på en mindre avgränsad grupp. Ett fall kan vara en individ, en grupp, en organisation eller en situation. Man

försöker vid en fallstudie se till helheten och få så täckande information som möjligt. Fallstudien kan t.ex. utföras med djupintervjuer.

Experiment är beteckningen på en undersökning där ett fåtal variabler studeras och där man försöker få kontroll över saker som påverkar dessa variabler.

12.1.5 Insamlingsmetod

Vid arbete med att insamling av information, både i form av enkäter och med hjälp av intervjuer, måste två aspekter beaktas. Vid planering av undersökningen måste beslut tas kring hur mycket ansvar som lämnas till intervjuaren när det gäller frågornas utformning och inbördes ordning. Detta kallas grad av *standardisering*. Den andra aspekten är i vilken utsträckning frågorna är fria för intervjuaren att tolka fritt efter sin egen erfarenhet och inställning. Detta kallas grad av *strukturering*.

Intervjuer med låg grad av standardisering eller helt ostandardiserade görs när frågorna formuleras under intervjun. Vid helt standardiserade intervjuer ställs likalydiga frågor i samma ordning till alla personer. Denna typ av undersökning utförs ofta med enkäter. Vid strukturering beaktas hur stort "svarsutrymme" intervjupersonen får. En helt strukturerad intervju lämnar ett mycket litet utrymme för intervjupersonen att svara inom genom att tillhandahålla ett antal fasta svarsalternativ att välja mellan. I en ostrukturerad intervju lämnas maximalt utrymme för intervjupersonen att svara inom. Exempel på detta är frågor av typen "Vad anser du om ...?" [Pat94].

12.2 Metodval

Vi har, för att kunna utföra undersökningen på bästa sätt, valt att se på vetenskapen enligt ett hermeneutiskt synsätt och valt att insamla, bearbeta och analysera data med en kvalitativ metod.

Undersökningen valdes att göras i form av en fallstudie eftersom intervjupersonerna var relativt få och då möjlighet till slumpmässigt val av dessa personer inte fanns. Insamling av data skedde i form av djupintervjuer kompletterade med studie av dokument.

Insamlingsmetoden var standardiserad i den mening att ett antal "stödfrågor" formulerats i förväg. Ur dessa frågor valdes dock vissa ut och anpassades för att passa aktuell intervjuperson beroende på befattning och arbetsuppgifter. Ytterligare frågor tillkom dessutom under intervjuernas gång för att föra dis-

kussionerna framåt. Intervjupersoner stod helt fria att tolka frågorna efter tidigare erfarenheter och inställning. Vi tillhandahöll inga svarsalternativ att välja mellan utan de fick svara fritt vilket innebär en låg grad av strukturering.

13 Utvärderingsmodell

För att undersöka faktorers påverkan på arkitekturens utformning har vi skapat en generell modell för utvärdering av genomförda systemutvecklingsprojekt. Utvärderingsmodellen ska kunna fungera som en vägledningsmodell vid uppföljning av projekt. Syftet med denna modell är att studera arbetsgången och att identifiera hur olika faktorer eventuellt begränsat eller underlättat arbetet. Eftersom vi anser att det inte räcker att studera problemet i att realisera det man har planerat har vi delat upp utvärderingen i ytterligare två steg. Systemen kommer i slutändan huvudsakligen att jämföras med användarnas önskemål på systemet och inte med hur systemet planerades. Vi har därför utformat modellen med början vid önskemålen. Vi vill se om man under arbetsgången hela tiden "kontrollerat bakåt". Motsvarar det planerade det önskade? Motsvarar det realiserade det planerade? Motsvarar det realiserade det önskade?

13.1 Modellens mål

För att tydligt förstå modellens syfte har vi formulerat tre huvudsakliga mål med modellen. Dessa är:

- att identifiera en mängd faktorer som inverkat stödjande
- att identifiera en mängd faktorer som inverkat begränsande
- att utifrån identifierade faktorer avgöra huruvida man lyckats eller misslyckats med projektet

13.2 Tillvägagångssätt

Utvärderingsmodellen förutsätter att undersökningen sker enligt en kvalitativ metod, dvs. i form av en verbal analysmetod, eventuellt kompletterat med studie av relevanta dokument. Utvärderingen bör inledas med en presentation av den övergripande arkitekturen och dess huvudsakliga delar. Beroende på vilken fas i modellen som studeras och utifrån vilka faktorer kan det vara lämpligt att intervjua olika roller/personer som varit inblandade i projektet. Ett antal "stödfrågor" till de olika faserna finns formulerade för att lättare kunna utse rätt personer att intervjua.

13.3 Förutsättningar

Det kan vara lämpligt att innan utvärderingen få tillgång till dokument från arbetet med arkitekturen som kan vara relevanta för undersökningen. Exem-

pel på typer av relevanta dokument är följande:

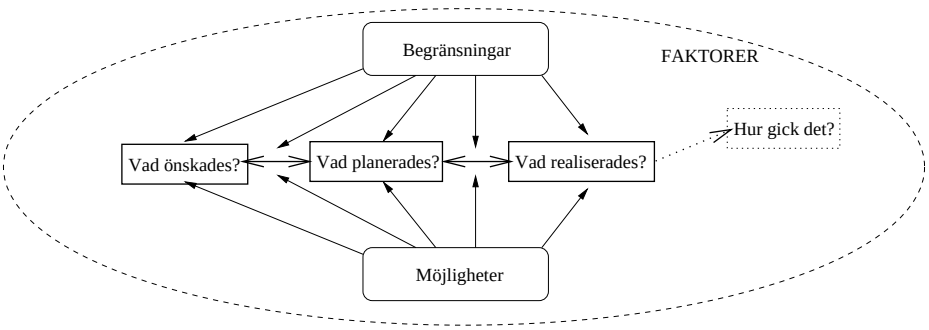
- plan över verksamhetsanalys, verksamhetsmodell, verksamhetens krav på systemet
- kravspecifikation, prioriteringar på kraven, målmodell
- övergripande systemstruktur, modell över delsystem och dyl.
- principer, riktlinjer

Utvärderingsmodellen är alltså inte intresserad av systemarkitekturen på detaljnivå, så som enskilda modulers gränssnitt, utan på en mer övergripande nivå.

För att utvärderingen ska kunna göras i form av en jämförelse av det önskade systemet med det realiserade systemet måste projektet som utvärderas ha uppnått någon form av konkret resultat, t.ex. en första version av systemet. Systemet bör även ha varit i bruk under så pass lång tid att brister i systemet påvisats.

13.4 Modellens form

Utvärdering sker genom att studera fyra huvudfaser men även övergångarna där emellan. I övergångarna kan de faktorer som agerat begränsande identifieras. Faktorer som inte begränsar stödjer utformningen av arkitekturen och medför ett friare val av lämpliga lösningar.



Figur 17: Utvärderingsmodell

13.5 Faktorer

Olika faktorers inverkan på de fyra faserna ska identifieras för att se hur dessa påverkat arbetet och utformningen av arkitekturen. En diskussion kring huruvida de olika faktorerna varit stödjande eller begränsande ska också ske och då ska även eventuella begränsningar ifrågasättas. Det är viktigt att fundera kring ifall begränsningar gjorts i onödan eller om de var nödvändiga. Det är även intressant att undersöka vilka begränsande faktorer som övervunnits.

Följande faktorers inverkan ska undersökas:

- budget
- kunskap och erfarenhet
- gamla system
- andra system
- personal
- verktyg
- tid
- projekt och planering
- dokumentation

13.6 Faser

Det ideala vid utformning av arkitektur vore att basera alla beslut på olika krav på systemet. I verkligheten påverkas dock besluten av mycket mer än bara krav. Denna modell ska undersöka hur de olika besluten även på olika faktorers inverkan. För varje fas ska arbetsgången och de arkitekturella besluten analyseras med utgång från de olika faktorernas inverkan. I den sista fasen, Hur gick det?, undersöks hur faktorers inverkan i de tidigare faserna medfört att kvalitetsattribut uppfyllts eller inte i det resulterande systemet.

Varje fas består av ett antal stödfrågor för att vägleda identifieringen av inverkande faktorer. Vid intervjuerna är det viktigt att hela tiden försöka identifiera påverkande faktorer. Vissa frågor syftar dock inte till att identifiera just detta utan är till för att ge en bättre förståelse av arbetsgången från ett önskat till ett realiserat system.

Vi har valt att dela in stödfrågorna i varje fas i olika grupper.

13.6.1 Vad önskades?

Denna fas i utvärderingen studerar hur man redan i önskningarna och kravutformningen begränsades eller fick stöd av faktorer. Det är intressant att se hur krav och önskningar på systemet blev direkt avvisade på grund av inverkan av faktorer. Observera att vi i denna fas gör skillnad på önskningar och krav på systemet. Krav anses vi baseras på de önskningar som anses vara realiserbara och viktiga för systemets uppgift. Stödfrågorna i denna fas är:

Visioner. Vilka inledande visioner hade man på systemet? Vilka visioner blev direkt avvisade och vilka fick begränsas? Vad orsakade avvisningarna och begränsningarna? Hur hårt begränsades visionerna? Vilka visioner var realiserbara? Vilka faktorer gav möjlighet till detta?

Önskningar. Vem hade önskningar på det kommande systemet? Utfördes en verksamhetsanalys och av vem? Var önskningarna på systemet neutrala eller färgades dessa av olika faktorer? Vilka faktorer hade isåfall inverkan på önskningarna och var de begränsande eller stödjande? Till hur stor utsträckning uteslöts önskningar i kravspecifikationen?

Krav. Bestod kravspecifikationen endast av realiserbara krav i förhållande till kända faktorers inverkan, t.ex. kunskap? Utformades kraven endast för att uppfylla kvalitetskraven eller utformades kraven även med avseende på faktorers inverkan? Vilka faktorer påverkade kravens prioritering?

13.6.2 Vad planerades?

Denna fas går till stor del ut på att identifiera vilka faktorer som påverkat de rena arkitekturella besluten. Det kan vara lämpligt att för varje typ av arkitekturellt beslut, t.ex. val av arkitekturella stilar, gå igenom olika faktorer och diskutera dess inverkan. En annan viktig del i denna fas är att identifiera vilka faktorer som medfört att vissa krav i kravspecifikationen blivit tvungna att uteslutas. Eftersom den planerade arkitekturen även fungerar som en plan över det fortsatta konstruktionsarbetet är det intressant att identifiera faktorer som har påverkat fördelning av arbete på olika utvecklingsteam. Följande är stödfrågorna i denna fas:

Arkitekturella beslut. Hur påverkades arkitekturella beslut av faktorer? Exempel på arkitekturella beslut är val av arkitekturella stilar, mönster, komponentindelning, delsystem, datamodell, arkitekturell beskrivningsteknik, återanvändning mm. Ifrågasattes de arkitekturella besluten i förhållande till kommande inverkande faktorer, t.ex. personal? Omöjliggjorde faktorer vissa "ideala" tekniska lösningar?

Tillgodose kraven. Vilka krav med hög prioritet fick uteslutas? Vilka krav av lägre prioritet fick uteslutas? Vilka faktorer orsakade dess uteslutande?

Arkitekturell plan. Hur planerades den vidare konstruktionen? Vilka faktorer inverkade avgörande för hur konstruktionen skulle delas upp? Kunde konstruktionen planeras precis som de inblandade i arkitekturarbetet ville?

13.6.3 Vad realiserades?

I denna fas studeras främst hur de arkitekturella besluten och den arkitekturella plan som tidigare utformats varit realiserbar. Det är intressant att se om den arkitekturella planen fungerat som en bra konstruktionsplan. I och med realiseringen av den arkitekturella planen kan eventuella dolda faktorers inverkan identifieras, dvs. faktorer som inte beaktats i planeringsfasen och som sedan gör sig synliga i realiseringen. Sådana faktorer kan innebära att omstruktureringar måste ske och att nya arkitekturella beslut måste fattas. Stödfrågorna i denna fas är:

Realisering av arkitekturella beslut. Vilka arkitekturella beslut gick inte att realisera och i så fall på grund av vilka faktorer? Vilka faktorer har inverkat stödjande i realiseringen av arkitekturella beslut? Gick kravens prioritering att vidhålla? Realiserades systemet helt enligt den arkitekturella planen? Togs några arkitekturella beslut på lokal nivå? Vilka arkitekturella beslut realiserades trots faktorers negativa inverkan?

Konstruktionsplan. Var konstruktionsplanen lämplig? Blev utvecklingsteamen tvungna att omarrangeras? Var uppdelningen av komponenter lämplig? Gick eventuell planerad återanvändning av moduler att uppnå? Vilka huvudsakliga problem uppstod i det vidare konstruktionsarbetet? Orsakades i så fall dessa av arkitekturella beslut? Var den arkitekturella beskrivningstekniken lämplig?

13.6.4 Hur gick det?

Den huvudsakliga frågeställningen i denna fas är huruvida systemet uppfyllt kvalitetsattributen godtagbart trots olika faktorers begränsande inverkan, med andra ord om systemet kan anses vara lyckat. Självklart är det även oerhört intressant att se vilka faktorer som agerat stödjande i realiseringen av kvalitetsattributen. I denna fas bör man kunna avgöra om de arkitekturella besluten har påverkats så mycket av rådande faktorer att kvaliteten i det resulterande systemet blivit lidande. Man bör även kunna identifiera de faktorer som projektet kan anse sig ha "övervunnit" med innebörden att kvalitet uppnåtts trots en viss faktors talan för motsatsen. Stödfrågorna i denna fas är:

Uppfyllande av kvalitetsattribut. Vilka kvalitetsattribut uppfylldes? Vilka faktorer talade för eller mot olika kvalitetsattributs uppfyllelse? Vilka kvalitetsattribut uppfylldes inte? Vilka faktorer orsakade detta? Vad realiserades dåligt trots stödjande faktorer?

Misstag att lära av. Vilka var de huvudsakliga misstagen? Kan misstagen förklaras av dolda faktorers inverkan? Kan misstagen härledas till felaktiga val tidigare i utvecklingen?

Jämförelse med det önskade. Anses systemet som lyckat? Anses systemet som lyckat med avseende på det önskade systemet? Anses systemet som lyckat med avseende på begränsande faktorer? Vilka faktorer har i huvudsak inverkat positivt och stödjande? Vilka faktorer har i huvudsak inverkat negativt och begränsande?

14 Fallstudie

Som en del i detta examensarbete utförs en studie av ett befintligt informationsystem och dess utvecklingsprocess. Denna studie använder sig av den utvärderingsmodell vi utformat. Vi har valt att studera det studiestödssystem som används i dagsläget på Centrala Studiestödsnämnden, CSN. CSN är den statliga myndighet som administrerar det svenska studiestödet. Studien genomförs på IT-avdelningen på CSN i Sundsvall.

14.1 Mål med fallstudie

Vi har tre primära mål med vår utvärdering:

- Undersöka CSN:s utveckling av det nya studiestödet i Stis2000 för att identifiera vilka faktorer som påverkat systemarkitekturens utformning positivt respektive negativt. Vi vill undersöka hur negativt inverkande faktorer lett till brister i systemarkitekturen och hur positivt inverkande faktorer lett till en bättre systemarkitektur. Vi är därmed inte intresserade av att finna detaljerade brister i systemarkitekturen annat än som exempel.
- Genomföra utvärderingen för att testa vår utvärderingsmodell och på så vis förbättra denna, t.ex. genom att lägga till ytterligare inverkande faktorer.
- Genomföra utvärdering så att CSN kan dra nytta av resultatet.

14.2 Beskrivning av studerat projekt

År 1999 beslutade regeringen om ett reformerat studiestödssystem med ett nytt regelverk som skulle träda i kraft 1 juli 2001. För att hantera de nya reglerna utvecklade CSN ett nytt datasystem. Detta utvecklingsprojekt kallades Studiestöd 2001. Projektet delades huvudsakligen in i två delar, en för beviljning av studiemedel och en för återbetalning av studiemedel. Vi kommer senare i denna presentation av studien omnämna dessa som Beviljningen och Återbetalningen. Denna studie omfattar endast Beviljningen. Beviljningen delades i sin tur in i fyra delprojekt. Vi har emellertid inte tagit hänsyn till denna indelning under denna studie.

Totalt har ca 80 antal personer varit inblandade i Beviljningen. Projektets officiella startdatum var 2000-08-07 och en första version av systemet levererades vid fyra olika leveranstillfällen.

1. 2001-02-03 Utbildningsregistret
2. 2001-04-07 Basuppgifter t.ex. ärenderegler
3. 2001-05-12 Grundregistrering, beslut mm.
4. 2001-06-21 Studiekontroller och utbetalningar

Regelverket trädde i kraft 1 juli 2001 men systemet kunde börja användas i form av handläggning av studiestödsansökningar efter den tredje delleveransen, 2001-05-12.

14.3 Tillvägagångssätt

Åtta personer som varit inblandade i Studiestöd 2001 intervjuades. Efter att vi presenterat syfte och problemfrågeställning för vår kontaktperson på CSN presenterade han ett antal personer som skulle vara lämpliga att intervjua, utöver honom själv. Dessa personer kontaktades och tid för intervju bokades in, om möjligt. Utvärderingsmodellen med tillhörande stödfrågor skickades till de personer som bokats in för intervju, dock inte med kravet att de måste läsa igenom denna. Det var därmed frivilligt för dem att förbereda sig inför intervjun eller ej.

Intervjuerna skedde enligt den utvärderingsmodell som utformats. Inledningsvis identifierades den aktuella personens eller personernas arbetsuppgifter i projektet för att på så vis kunna anpassa frågorna på lämpligt sätt.

Intervjuerna dokumenterades med hjälp av handskrivna anteckningar under intervjuernas gång. Anteckningarna renskrevs och förfinades i efterhand. Dessa dokumentationer tillsammans med de intryck vi fått ligger till grund för det resultat vi presenterar i följande avsnitt.

I tabellen på följande sida finns en beskrivning av de åtta personer som intervjuades.

Person	Befattning	Anställning	Arbetsuppgifter
A	IT-arkitekt	ca 5 år på CSN	arkitektur, kvalitetssäkring
B	utvecklingsledare i utvecklingsprojekt	> 15 år på CSN, konsult	analys, design, datamodell
C	utvecklingsledare i utvecklingsprojekt	> 20 år på CSN, konsult	analys, design, dokumenthantering
D	utvecklingsledare i utvecklingsprojekt, datamodellansvarig	> 15 år på CSN, konsult	framtagning av beställning, analys, datamodell, design
E	huvudprojektledare i utvecklingsprojekt	< 5 år på CSN, konsult	projektledare för ett delprojekt
F	IT-arkitekt	> 20 år på CSN	arkitektur, kvalitetssäkring, rådgivning
G	processägare för Kundserviceprocessen	> 15 år på CSN	framtagning av beställning
H	systemförvaltare	> 15 år på CSN	utvecklingsledare för verksamheten i ett delprojekt

15 Resultat av fallstudie

Ett av de primära målen med vår studie var att identifiera olika faktorers inverkan på systemarkitekturens utformning. Vi upptäckte dock redan vid första intervjun att detta skulle bli svårt i och med att det studerade projektet inte haft något uttalat arkitekturarbete med ledning av en arkitektenhet. Vår utvärderingsmodell baseras till stor del på att så skulle skett. Vi fick därmed ändra målet med studien till att gälla faktorers inverkan på projektet som helhet och vid sidan av detta undersöka hur systemarkitekturen implicit utformats och vilken roll IT-arkitekterna haft under projektets gång. Vi kommer därför presentera resultatet av studien dels i form av beskrivning av olika faktorers inverkan men även presentera arkitektens roll och delaktighet i systemutformningen. Vi inleder presentationen av resultatet med en diskussion kring den realiserade systemarkitekturen.

Vi har valt att inte beskriva de enskilda tillfrågades åsikter, annat än som exempel, utan försökt sammanväga deras svar i denna sammanställning. Vi har därmed inte omnämnt de tillfrågade vid namn.

15.1 Systemarkitekturen

Systemlösningen har genererats med hjälp av verktyget Cool:Gen. Den övergripande arkitekturella stilen är en klient-server lösning. Klienten är skriven i C och servern, som arbetar mot en DB2-databas är skriven i COBOL. Servern består av tre huvuddelar, Beviljning, Återbetalning och Kundservice. Då vi inte har studerat Kundservice- och Återbetalningsdelarna eftersom dessa tillhör ett annat projekt än Studiestöd 2001 kommer vi därför inte beskriva systemarkitekturen i dessa närmare.

Beviljningen är moduluppbyggd, dock inte oberoende i den mening att de endast känner till varandra genom definierade gränssnitt. Förändring i en modul medför att ett flertal moduler måste omgenereras och testas på nytt. Detsamma gäller den tillhörande datamodellen. Den är oerhört komplex, med många beroende och främmande nycklar. Arkitekterna försöker finna metoder för att bryta isär datamodellen och skapa mindre beroenden. Detta är svårt och kan komma att kräva expertishjälp.

Ur arkitekturell synvinkel är målbilden en komponentbaserad systemstruktur med ungefär 40-60 autonoma verksamhetskomponenter, vilket skulle underlätta testning och modifiering avsevärt. I nuläget har testarna svårt att förstå vad som påverkas vid förändringar eftersom systemet är allt för in-

tegrerat. Dock är inte organisationen i dagsläget redo att ta steget till ett fullt komponentbaserat system då detta skulle kräva en stor kostnads- och tidssatsning. Av den anledningen genomförs förändringar steg för steg mot idealbilden. Ett exempel på ett steg mot en komponentbaserad systemarkitektur är ett projekt med syfte att utveckla en komponent för avskrivningshantering. Denna komponent ska sköta avskrivning av studiestödsveckor för behörighetsgivande studier på gymnasial nivå. Då denna funktion varken tillhör enbart Beviljningen eller enbart Återbetalningen har en fristående komponent med gränssnitt till Beviljning och Återbetalning planerats.

En annan önskan på arkitekturen är att införa ett mellanting mellan en transaktionsorienterad och en batch-orienterad systemstruktur. Detta genom att låta de autonoma komponenterna vara batch-orienterade och kopplingen där emellan vara transaktionsorienterad.

I struktureringen av Beviljningen upplevdes projektet, av arkitekterna men även av några av de övriga tillfrågade, vara alltför tids- och kostnadsbegränsat för att tillåta teknisk innovation och nya idéer. De tillfrågade menar att systemarkitekturen kan därför tyckas vara gammalmodig men till dess försvar är den inarbetad och stabil. Arkitekturen bakom Beviljningen är tillräckligt bra med avseende på systemets funktionalitet men i många kvalitetsaspekter upplevs den som bristande. Arkitekterna riktar kritik mot den bristande helhetssynen under konstruktionen av Studiestöd 2001 vilket medförde att arkitekturella beslut togs på lokal nivå.

15.2 Faktorer

Förutom de faktorer som fanns med i utvärderingsmodellen innan studiens genomförande har vi undersökt ytterligare två faktorers inverkan, nämligen *lagar och regelverk* samt *ledningen*. Lagar och regelverk har stor påverkan på CSN:s arbete främst på grund av att CSN är en statlig nämnd och att projekt ofta initieras p.g.a. lagförändringar. Ledningen påverkar genom att de tillhandahåller resurser i form av personal och pengar men även genom att besluta om genomförande av tekniska projekt. Ledningens inställning till teknik påverkar då mycket.

Faktorn *dokumentation* som finns med i utvärderingsmodellen har i denna sammanställning ersatts med faktorn *modeller*. Detta beror på att dokumentationen på CSN styrs av ett antal modeller.

15.2.1 Lagar och regler

Eftersom CSN är en myndighet och projektet Studiestöd 2001 dessutom var ett resultat av en reform hade faktorn *lagar och regler* stor påverkan. Ett av de stora problemen var att regelverket inte var fastställt innan projektet startade. Eftersom tidschemat var otroligt pressat fick projektmedlemmarna lov att använda sig av förhandsbesked från utbildningsdepartementet för att systemet skulle bli klart i tid. Vissa delar lämnades helt eftersom osäkerheten var för stor, men en stor del av systemet var redan färdigbyggt när regelverket slutligen fastställdes. Detta medförde att delar som inte stämde fick lov att ändras. Detta upplevdes av vissa leda till ett ”lapptäcke av nödlösningar”.

Regler fungerade även som en begränsande faktor i avseende på leverans av systemet. Leveransen fick inte förskjutas vid eventuella problem eftersom regelverket trädde i kraft ett visst datum och då krävde ett fungerande system.

Ett typiskt problem med avseende på denna faktor är dessutom att departementen sällan håller tidsramarna för de nya regelverken, vilket försvårar CSN:s arbete betydligt då vissa delar i utvecklingen kräver att ramverket är fastställt.

15.2.2 Verktyg

CSN använder ett verktyg som vid tidpunkten för Studiestöd 2001 hette Cool:Gen. Verktyget heter numera dock Advantage:Gen, men omnämns fortfarande som Cool:Gen av de anställda på CSN. Cool:Gen är en utvecklingsmiljö framställd för att utveckla verksamhetssystem. Att arbeta med Cool:Gen innebär att skapa verksamhetsobjekt i form av pseudokod och modellera dessa för att sedan generera färdiga klient-server-system men verktyget kan även hjälpa till med att integrera ett gammalt system med det nya systemet. Cool:Gen infördes under början på 90-talet för att ersätta det tidigare verktyget som enbart fungerade som applikationsgenerator.

Valet av verktyg var givet inför Studiestöd 2001 och några seriösa funderingar på att använda något nytt fanns inte. Detta på grund av att det inte skulle finnas tid att prova ut något nytt verktyg och på grund av att konstruktörerna var vana vid Cool:Gen sen tidigare utvecklingsprojekt. Att skriva kod på traditionellt vis var heller inget alternativ på grund av tidsbrist och skulle enligt en av de tillfrågade, även om tiden funnits, vara ”samhällsekonomiskt vansinne”. Användandet av Cool:Gen i Studiestöd 2001 medförde även möjligheten att återanvända vissa delar ur det befintliga systemet eftersom detta var

byggt med samma verktyg.

Alla tillfrågade är mycket nöjda med Cool:Gen och säger att verktyget var en av de faktorer som möjliggjorde att Studiestöd 2001 blev klart enligt tidsramarna. En nackdel är dock att dataåtkomsten upplevs vara något långsam. På grund av detta finns funderingar på att använda andra verktyg vid vissa tillfällen men att ändå ha kvar Cool:Gen som basverktyg. Cool:Gen kan även upplevas som något innovationsbegränsande för de som tycker om att programmera eftersom skrivandet av pseudokod inte möjliggör några ”häftiga” lösningar, som en av de tillfrågade uttrycker sig. Cool:Gen påverkar även konstruktionsfasen något negativt eftersom in- och ut-checkningen av filer är relativt långsam och kan medföra långa väntetider om många är inblandade i fasen.

Cool:Gen är, enligt en av de tillfrågade, inte fullkomlig på att generera effektiv COBOL men upplevs ändå som ett bättre alternativ än att anställa några riktigt duktiga COBOL-kodare, eftersom Cool:Gen genererar felfri kod och eftersom man slipper personberoendet som skulle uppstå vid den alternativa lösningen.

En av de tillfrågade förklarar att vissa tror att det är Cool:Gen som är orsaken till dåliga lösningar och tror då att problemen inte går att lösa, men oftast beror de dåliga lösningarna på felaktiga beslut i till exempel designfasen. Den tillfrågade menar här att det därför är viktigt att ha kunskap om hur övriga delar i systemet fungerar för att kunna undvika att göra olämpliga lösningar i Cool:Gen som till exempel kan medföra extra stor belastning på databasen.

15.2.3 Kunskap och erfarenhet

Eftersom många i projektet med det nya studiestödet tidigare deltagit i andra stora projekt hade man god erfarenhet att dra nytta av. Detta upplevdes av alla tillfrågade som en bidragande orsak till att de olika delprojekten kunde fördelas och genomföras lyckosamt utan en arkitekturell överblick över dessa. Många i personalen har lång erfarenhet av verksamheten och de gamla systemen. De flesta projektmedlemmarna hade även mycket god kunskap om verktyget sedan tidigare vilket medförde att verktyget kändes som en trygg teknisk miljö.

Även om det inte rådde kunskapsbrist i projektet var det svårt att få rätt personer till rätt arbete. Personal var inbokad på många olika delprojekt och

det var därför svårt att koordinera alla inblandade så att alla kunde delta där de behövdes.

Det uttrycktes en tanke och önskan från några av de tillfrågade om bättre erfarenhetsåtervinning så att erfarenhet och kunskap blir mindre personberoende. I dagsläget utförs uppföljningar efter ett projekts avslutande i form av frågeformulär där de inblandades tankar om projektet efterfrågas. Dock finns ingen uttalad metod för uppsamling och spridning av kunskap efter ett projekt och kunskapen blir därför personberoende. Med anledning av detta är det viktigt att se till att kunskapen i form av personer finns kvar inom organisationen, dvs i den egna personalen. En av de tillfrågade menade att det kan vara en fara i att förlita sig allt för mycket på erfarenheter och kunskap hos konsulter då dessa lättare kan försvinna från organisationen. Dagens dåliga ekonomiska läge i datakonsultbranschen, med få nya arbetstillfällen, medför att konsulter är relativt långvariga på den aktuella arbetsplatsen men vid förändrade omständigheter skulle detta kunna leda till ett allvarligare problem. Inom CSN finns dock medvetenhet om problemet med erfarenhetsåtervinning och olika försök till åtgärder har gjorts, t.ex. genom att låta konsulter ansvara för rekrytering av en anställd till CSN och lära upp denne. Det finns även andra tankar på hur erfarenhetsåtervinning skulle kunna ske men dock inga som genomförts.

15.2.4 Andra system

Viss information inom CSN skickas till och hämtas från andra myndigheter och organisationer. Hur detta informationsutbyte sker kan inverka begränsande för CSN:s effektivitet eftersom många överföringar sker med en något föråldrad teknik, t.ex. genom att filer skickas mellan de båda parterna istället för t.ex. automatisk informationsöverföring, där ny information i ett system direkt förmedlas till andra berörda parter och dess system. Det finns en vilja hos de tillfrågade att förnya dessa tekniker men det är svårt att koordinera den egna organisationen med den andra parten så att en lämplig överföringsteknik kan införas. Båda parter måste tjäna på en förnyelse och dessutom ha tid för ett sådant projekt. Eftersom CSN ofta står under tidspress prioriteras inte förhandlingar med andra organisationer angående en förnyelse av informationsöverföringen.

15.2.5 Ledningen

CSN är en framgångsrik organisation med avseende på IT. Ledningen har som målsättning att vara i teknikens framkant vilket upplevs som mycket

positivt hos de tillfrågade. Dock måste tekniska projekt relateras till kostnad och tid vilket innebär att de kan vara svårt för IT-avdelningen att motivera ledningen till att satsa på rena förbättringsprojekt.

I avseende på arkitektens roll i verksamheten upplevs ingen motvilja från ledningen till att arkitekten ska bli mer verksamhetsnära, dock ges heller inget direkt stöd från ledningen till detta. Arkitekterna har en vision att en IT-strategi ska vara nära förknippad med verksamhetens övergripande strategi vilket kräver att arkitektenheten bör samarbeta mer med ledning och verksamhetsnära enheter. På så sätt kan arkitektenheten lättare motivera tekniska projekt och lättare framföra hur verksamheten kan tjäna på dessa. En annan fördel med ett ledningsnära samarbete är att arkitekterna lättare kan förbereda IT-avdelningen för kommande projekt.

15.2.6 Gamla system

Vid införandet av det nya studiestödet försökte CSN överföra de lyckade delarna från det gamla studiemedelsystemet till det nya. Undersökning över vilka delar i det gamla systemet som fungerade bra och vilka som fungerade mindre bra utfördes. På så vis återanvändes en del komponenter från det gamla systemet. Återanvändning av delar från det gamla studiemedelsystemet underlättades oerhört av att det gamla systemet var utvecklat med samma verktyg som det som användes för att utveckla det nya.

I och med det nya studiestödet sammanfördes flera gamla system till ett stort system. Det nya systemet blev då väldigt komplext, vilket upplevs av några av de tillfrågade som negativt med tanke på modifiering och förvaltning av systemet.

Eftersom övergången från det gamla till det nya systemet var lagstyrt och skedde på ett visst datum var man tvungen att lösa problemet med överlappning, dvs de studerande som studerade en kurs som överlappade både det nya och det gamla t.ex. i form av sommarstudier.

15.2.7 Modeller

Inom CSN finns ett antal modeller för de olika utvecklingsmomenten i och med utveckling av system. De olika modellerna är projektstyrningsmodell, systemutvecklingsmodell och testmodell. Utöver dessa modeller finns en övergripande modell, CSN-modellen. Modellernas syfte är främst att säkerställa en kontrollerad utvecklingsprocess men de ger även trygghet åt projektdel-

tagare och skapar likformighet vad gällande t.ex. dokumentation. Denna likformighet underlättar uppföljningar och granskningar av dokument avsevärt. Modeller av sådan typ att den uppmanar personal till att arbeta efter mallar kan ibland vara svåra att införa men på CSN upplevdes detta inte som något problem då personalen hade god vilja att anpassa sitt arbetssätt. Numera ifrågasätts inte modellernas grundstruktur eller användningen av dessa. En bidragande orsak till att modellerna accepterades utan större problem och att de dessutom används är, enligt en av de tillfrågade, att det var starka personer som drev igenom dessa förändringar av arbetssättet i projekt. Dessa personer lyckades förankra modellerna bra så att personalen förstod nyttan med modellerna. Det händer dock att delar av modellerna ifrågasätts och då främst i små projekt. I dessa fall kan mindre vikt läggas vid delar som inte är av så stor betydelse för det aktuella projektet. Inom CSN sker fortlöpande arbetet med att förbättra modellerna genom att studera slutrapporter från avslutade projekt för att identifiera eventuella brister i modellerna.

Sedan modellerna infördes har CSN lyckats genomföra alla projekt inom avsedd tidsram. En av orsakerna till detta, förklarar en av de tillfrågade, är att önskningar på systemen ifrågasätts tidigare i utvecklingsprocessen och hänvisar till en fas i modellerna där en entydig beställning ska formuleras. Denna beställning står som grund för vad som ska levereras. På så vis realiseras det planerade systemet även om systemets önskade funktioner ibland fördelas på ett antal leveranser. Tidigare kunde kravspecifikationer ibland upplevas som en uppsättning önskningar vilket medförde att man i senare planering och realisering av systemet fick begränsa kraven betydligt. Modellerna har alltså medfört att man tidigare utesluter eller nedprioriterar önskningar på systemet. I Studiestöd 2001 var beställaren delaktig i formulering och prioritering av kraven vilket underlättade realisering av ett godtagbart resultat.

I och med införandet av modellerna görs alltid en riskanalys i början av projektet. Denna riskanalys syftar till att bedöma eventuella risker och dess inverkan. Beroende på riskernas storlek utformas en reservplan. Ju större riskerna är desto mer arbete läggs på reservplanen. Om riskerna anses vara för stora för att resultera i ett lyckat resultat kan projektet läggas ner. Detta är emellertid ovanligt då projekt alltid måste godkännas av utvecklingsledningen, vilka ser till att alltför riskfyllda och ”onödiga” projekt inte startas.

Det finns önskningar bland de tillfrågade att komplettera modellerna och då främst systemutvecklingsmodellen. T.ex. anses rutiner i analysfasen vara för grovt beskrivna eftersom analysen omfattar så många olika moment, så

som systemanalys, utbildning, handläggarrutiner, ärenderegler och beslutsutformning. Det riktas även kritik mot kvalitetssäkringen och det faktum att inga detaljerade krav på formulering av kvalitetsmål i form av enskilda kvalitetsattribut finns. Exempel på kvalitetsattribut kan vara prestanda, användbarhet, modifierbarhet, tillgänglighet etc. Genom att formulera systemets önskade uppfyllnad av dessa kvalitetsattribut kan det färdiga systemets kvalitet lättare mätas. Vidare kritiseras rutiner kring test och framtagning av testfall då dessa utvecklas för sent. En önskan finns att dessa ska utvecklas då användarfall identifieras.

Funderingar på utökning av systemutvecklingsmodellen finns i form av metodik för parallell utveckling av olika versioner av system, dvs. olika delleveranser. Detta klarar CSN inte av i dagsläget vilket medför att en version måste slutföras innan konstruktion av den nästkommande kan påbörjas.

15.2.8 Projekt och planering

Projektstyrningen och organisationen kring detta anses av alla tillfrågade vara en succéfaktor vid genomförandet av Studiestöd 2001. Projektstyrningen var en bra grund att stå på trots att projektstyrningsmodellen användes för första gången i och med detta projekt. Efter designfasen delades projektet in i fyra delprojekt med var sin projektledare. Dessa delprojekt hölls ihop av en styrgrupp. Dock upplevdes det negativt av vissa tillfrågade att det inte fanns någon övergripande projektledare. Uppdelningen i de fyra delprojekten skedde utan någon arkitekturell översikt. Denna gjordes helt enligt verksamhetsfunktioner och de inblandades erfarenhet från tidigare stora projekt. Resultatet av uppdelningen anses som lyckad.

En risk som upplevdes som störst i detta projekt var dess storlek och komplexitet. Många människor var inblandade vilket i regel innebär en hel del organisatoriska problem. Detta lyckades man dock på CSN övervinna delvis genom att överdimensionera projektstyrningen.

Med anledning av tidspressen planerades händelser som skulle ske tidigare än systemleverans inte i början av projektet. Exempel på en sådan händelse är uppflyttning av studenter, dvs utskick av förtryckta ansökningsblanketter till studerande på exempelvis program. Dessa händelser observerades inte förrän i konstruktionsfasen.

15.2.9 Personal

Eftersom projektet Studiestöd 2001 var stort och omfattade många personer var personalen en viktig faktor. Deras engagemang och vilja att anpassa sig till modellerna tillsammans med deras kunskap och erfarenhet upplevdes som en bidragande orsak till att tidspressen gick att övervinna och att systemet kunde levereras på avsett leveransdatum. Enligt slutrapporten var den viktigaste framgångsfaktorn personalens goda stämning och stora engagemang.

15.2.10 Tid

Tiden var den främsta begränsande faktorn i Studiestöd 2001. Som vi nämnt tidigare upplevdes projektet på grund av tidsbristen vara alltför leveransfokuserat. Tidsplanen i vissa delprojekt var enligt slutrapporten alldeles för optimistiska med avseende på konstruktion och integrationstest. Trots tidpressen lyckades projektet leverera systemet på avsett leveransdatum.

15.2.11 Budget

De tillfrågade upplevde inte budgeten som en begränsande faktor. Enligt den slutrapport som skrivits angående Studiestöd 2001 har budgeten underskridits. Detta kan bland annat förklaras av att inplanerade resurser ej utnyttjats och att enskilda personer inte rapporterat korrekt antal arbetstimmar. Konsulterna har dock överskridit budgeten en aning.

15.3 IT-arkitektens roll

Under intervjuerna framkom faktumet att det inom organisationen finns olika syn på arkitektens roll. Olika personer har olika uppfattningar om IT-arkitekternas uppgifter och ansvarsområden. Arkitekterna ser arkitektrollen som ansvarig för den IT-strategi som verksamheten bör ha och arbeta efter. Denna IT-strategi ska baseras på en övergripande verksamhetsstrategi. Det upplevs dock, av arkitekterna, svårt att identifiera den exakta verksamhetsstrategin på CSN. Arkitekten ska även tillhandahålla regler och ramverk för nya metoder och verktyg som införs och dessutom övervaka att dessa efterföljs. Arkitekten har även en rådgivande roll och kan gå in i projekt i stödjande syfte. Det väsentliga i arkitektens roll är att hela tiden se systemet utifrån ett bredare perspektiv och med helikoptersyn för att kunna styra arbetet mot den uppsatta visionen av systemstrukturen. Arkitekten ska mer vara en generalist än en specialist. Dock upplever arkitekterna att vissa förlitar sig på arkitekten i fråga om detaljerade beskrivningar och instruktioner av t.ex. ramverk och verktyg, vilket inte är deras ansvar.

Det upplevs ibland svårt för arkitekten att få sina idéer och visioner accepterade av alla inom organisationen. Projekt har genomförts med lyckat resultat tidigare utan inblandning av arkitekten vilket innebär att nya främmande tankesätt för systemuppbyggnad kan kännas onödigt och otryggt. Arkitekterna upplever dock inte detta som något allvarligt problem eftersom personalen får förståelse för visionerna om tid för diskussioner finns.

Från ledningens sida upplevs ingen motvilja mot arkitekternas visionsarbete men detta arbete prioriteras inte. Fokusering upplevs ligga på rena leveransprojekt dvs projekt som levererar nya system eller delar av system och inte på projekt vars syfte endast är att förbättra systemstrukturen eller förbereda system för kommande krav, t.ex. olika lagändringar. Det kan vara svårt att motivera inför ledningen varför viss förbättringsprojekt behövs eftersom dessa inte genererar några intäkter och inte levererar något system eller del av system. Eftersom det ofta råder kostnadsjakt prioriteras inte dessa projekt.

Under Studiestöd 2001 existerade ingen IT-arkitektroll. Personerna som i nuläget har benämningen IT-arkitekt kallades då IT-stöd. Arkitektrollen är därmed ny och upplevs därför, av några av de tillfrågade, ännu inte funnit sin plats i projekten. Utvecklingspersonalen inblandade i Studiestöd 2001 hade mycket erfarenhet och kunskap från tidigare projekt och upplevde därför att de inte behövde någon överblickande arkitekturell vägledning. De kunde dela upp projektet i delprojekt själva. Arkitektrollen blev därmed satt utanför det inledande struktureringsarbetet.

Arkitekten fick senare i utvecklingen av systemet utföra granskningar av färdiga dokument. Detta upplevdes enbart som en formalia utan någon möjlighet till påverkan på systemstrukturen. Att granska innebar att arkitekten fick en mängd dokument rörande systemet att granska efter varje avslutad fas i systemutvecklingsmodellen. Oftast var det ont om tid att läsa igenom dessa dokument och dessutom svårt att ha några synpunkter på den övergripande strukturen eftersom krav på stora förändringar skulle innebära mycket förlorad tid och arbete.

Granskningarna har nu gjorts om till avstämningsmöten. Projektledningen kallar då till avstämning där problem och risker belyses för arkitekten. Detta har gjort situationen bättre ut arkitekturellt perspektiv men ännu finns inga möjligheter för arkitekten att påverka den övergripande systemstrukturen. Det finns förhoppningar om att arkitekten ska sitta med redan i förstudier av projekt för att förbereda sig på kommande tekniskt arbete

och kunna påverka i ett tidigare skede. Detta skulle medföra att arkitekterna agerade mer proaktivt istället för nulägets reaktiva arbete. Ett exempel på arkitekternas strävan mot mer inflytande över systemarkitekturens utformning är konstruktionen av en integrationsplattform som hanterar koppling mellan kanaler, så som websvar, talsvar och andra organisationer, och studiestödsystemet. I detta projekt har arkitekturarbetet varit av stor vikt och utförliga tester på skalbarheten har gjorts. I och med denna integrationsplattform är man bättre förberedd för det kommande kravet på att vara en 24-timmarsmyndighet.

15.4 Resultatet

Eftersom en rättvis bedömning av huruvida systemet är lyckat eller ej skulle kräva undersökningar utifrån många olika perspektiv och med fler personer än det antal som ingått i denna studie, har vi valt att inte presentera någon slutgiltig bedömning av resultatet. Dock anser alla de tillfrågade i denna studie att systemet är lyckat. De valde då att se resultatet utifrån de rådande förutsättningarna. Projektet levererade ett system enligt kontraktet, dvs. det som lovats. Det finns inplanerade förbättringsåtgärder och uppdatering av systemet sker kontinuerligt i form av nya leveranser och versioner.

Även om det färdiga systemet kan anses lyckat finns ett antal relativt allvarliga problem. Förklaringen till dessa kan enligt de tillfrågade förklaras av den rådande tidspress som rådde i projektet och det faktum att projektet blev allt för leveransfokuserat. I och med detta fanns inte tid till att förebygga problemen trots att man delvis kunde förutse dem. Problemen är emellertid inte så allvarliga att de inte går att åtgärda.

16 Diskussion kring studie

Det största problemet med studien var att uttalat arkitekturarbete inte var så pass moget inom CSN som vi hade förutsatt. Även om arbete med IT-arkitektur i dagsläget är ett populärt ämne i tidsskrifter och böcker inser vi att det tar tid innan nya tankesätt når organisationer och specificeras i modeller och arbetssätt. Problemet kring IT-arkitektens roll kan antagligen även förklaras av detta. Dock verkar CSN som verksamhet inte vara ovillig till teknisk förändring och förnyelse.

Eftersom studien endast bestod av ett fåtal personer var det svårt att få en rättvis bild av projektet och systemarkitekturen då dessa personer har olika uppfattningar och åsikter. De tillfrågade kan dessutom vara hemmablinda i den mening att de inte naturligt ser alternativ till valda systemlösningar eller olika moment i projektet. Trots detta fick vi ändå en enhetlig uppfattning angående de olika faktorernas inverkan. I detta avseende var de tillfrågade relativt överens.

Vi upplevde även att olika personer tolkar begrepp olika vilket försvårade intervjuerna en aning. Ett exempel är tolkning av begreppet komponentbaserad systemstruktur, där vissa tillfrågade upplevdes ha annan uppfattning av dess innebörd än sina kollegor. Även uppfattning av IT-arkitekternas arbetsuppgifter skilde stort mellan de tillfrågade.

Då vi i förväg inte visste vad de enskilda tillfrågade haft för arbetsuppgifter under projektet gick mycket tid åt till att identifiera dessa. Det hade varit en fördel att veta detta innan för att på så vis spara intervjutid och kunna förbereda och anpassa frågorna bättre.

16.1 Utvärdering av modellen

Då utvärderingsmodellen bygger på ett uttalat arkitekturarbete med ledning av en övergripande enhet, en arkitekturenhet var det svårt att anpassa denna till Studiestöd 2001. Visionen vid utformningen av modellen var att arbetet från verksamhetsanalys fram till en färdig konstruktionsplan skulle omfångas av ett övergripande arkitekturarbete. Arkitekturarbetet skulle dessutom genomsyras av en långsiktig IT-strategi. Detta var dock en alldeles för idealiserad bild för att kunna appliceras på Studiestöd 2001 vilket medförde att modellen inte gick att följa som förväntat. Detta visar med andra ord på en huvudsaklig brist i modellen eftersom denna inte går att applicera till fullo på godtyckligt systemutvecklingsprojekt.

Utvärderingsmodellen baseras på tre faser, önskat, planerat och realiserat. CSN:s systemutvecklingsmodell är även denna linjär på så vis att arbetet sker i seriellt följande faser. Trots detta var det svårt att se någon motsvarighet mellan faser i vår modell och i CSN:s systemutvecklingsmodell. Tanken i vår modell är att så gott som allt arkitekturarbete sker i planeringsfasen och att realiseringsfasen motsvarar implementering av det planerade. Detta med reservation för att vissa arkitekturella beslut måste korrigeras i senare skeden i utvecklingen. Arkitekturarbete är inte lika avgränsat i CSN:s systemutvecklingsmodell och det är därför svårt att applicera vår utvärderingsmodell och dess faser.

Som utvärderingsmodellen ser ut nu fungerar den bäst på ett projekt som genomsyras av ett övergripande arkitekturarbete. Om ett sådant projekt skulle utvärderas skulle antagligen olika faktorers inverkan på val i systemarkitekturens utformning kunna identifieras. För att utvärderingsmodellen ska fungera på systemutvecklingsprojekt där arkitekturarbetet inte varit i fokus skulle en annan typ av modell krävas. En sådan modell skulle ha som främsta uppgift att identifiera arkitekturella moment i projektet och därefter faktorers inverkan. Det kan alltså vara nödvändigt att använda olika modeller på olika projekt beroende på hur arkitekturarbetet skett.

Utvärderingsmodellen bör även utökas i form av gruppering av frågor beroende på den tillfrågades arbetsuppgifter och roll i projektet. Det bör finnas frågor anpassade för olika roller eftersom faktorerna haft olika stor betydelse beroende på vilket typ av arbete som utförts.

Del III
Avslut

17 Egna tankar och reflektioner

För att kunna skriva uppsatsen har vi studerat mycket material, böcker, artiklar och avhandlingar. De flesta artiklar och avhandlingar refererar till samma böcker, då utbudet av dessa inom området är begränsat. Det tror vi beror på att ämnet informationssystemarkitektur är fyllt av så mycket trender och subjektiv kunskap, dvs. enskilda forskares eller andra branschfolks aktuella åsikter. Det kan då kännas mer lämpligt att uttrycka dessa åsikter i en artikel än att skriva en bok, eftersom en bok kan kännas alltför bestående. Många av de artiklar vi läst är skrivna av samma personer, så de borde ha tillräckligt med material för en bok. Ämnet är dessutom relativt ”nytt” vilket också borde påverka bokutbudet.

När vi började med examensarbetet var ämnet nytt för oss. Under vår utbildning har begreppet arkitektur aldrig specifikt introducerats. Detta kan bero på att kurserna sällan tar upp stora och övergripande systemutvecklingsprojekt utan fokuserar på de uppgifter som studenterna ska genomföra. Det finns många kurser där det skulle passa med en teoretisk del om systemkonstruktion på arkitekturell nivå. Det skulle vara bra att låta dessa aspekter finnas med mer i de flesta programvarutekniska kurserna, men helst ser vi en hel kurs i ämnet.

Det är en sak att ämnet var nytt för oss som inte har arbetat med omfattande systemkonstruktion, men att ämnet ännu inte verkar ha etablerats mer hos de som bygger stora system förvånar oss. Denna uppfattning baserar vi dels på vår fallstudie, men även på det faktum att många företag och organisationer i dagsläget efterfrågar arkitekturell kunskap i form av nyrekrytering eller utbildningar. Del två av vårt arbete var därför oerhört intressant och gav oss perspektiv på hela del ett och en tankeställare vad gäller organisationers medvetna satsning på strategiskt arkitekturarbete.

Den största lärdomen vi fått, utöver ämneskunskaperna, är svårigheten att avgränsa sig inom ett ämne som man inte är så insatt i. Det var väldigt lätt att anta att vissa delar ”måste” var med, t.ex. genom att läsa en artikel om ett visst arkitekturellt fenomen, trots att de kanske inte är så viktiga för den övergripande förståelsen för ämnet. Av den anledningen har vi ofta planerat

att skriva eller t.o.m. skrivit avsnitt som vi senare insåg var överflödiga. Ett exempel är databasarkitektur som vi till en början ansåg hörde hemma i denna uppsats, men senare uteslöt. Vi har därför inte heller skrivit om några tekniker, t.ex. CORBA eller andra typer av middleware[17]. Detta inser vi nu vara ett väldigt bra val.

Vi har ännu inte hittat något negativt i att arbeta två personer med ett så här stort arbete. Vi ser det tvärtom som en stor fördel att ha någon att diskutera och "bolla tankar" med vilket ger oss djupare förståelse för det vi läser och skriver. Som vi redan visste, och än en gång fick bekräftat, arbetar vi otroligt bra tillsammans vilket var en förutsättning för att ro iland detta arbete.

[17]Ordet finns förklarat i ordlistan.

18 Tack

Då arbetet pågått under så lång tid har många personer varit till hjälp. Vi vill först och främst tacka vår handledare Oscar Appelgren för hans engagemang och åsikter om innehållet. Ingen åsikt har varit överflödig eller kunnat ifrågasättas. Givetvis vill vi tacka alla på CSN för deras deltagande i fallstudien och då främst vår kontaktperson Örjan Carlsson. De var väldigt tillmötesgående och villiga att svara på våra frågor vilket underlättade studien betydligt. Ett speciellt tack till Anita Hansson för otroligt mycket hjälp och inte minst boende och mat under fallstudien. Vi vill även tacka våra underbara familjer, vänner och pojkvänner för stöd och ”pepp”. Sist men inte minst vill vi tacka varandra!

A Ordlista

`adhoc` (1) Till detta, tillsatt för ett särskilt ändamål. (2) En icke planerad systemlösning endast i syfte att lösa ett specifikt problem.

`artefakt` (1) En artefakt är resultatet av en aktivitet. Det kan vara dokument, källkod, komponenter. (2) En artefakt är ett föremål som avsiktligt eller oavsiktligt tillverkats av människor.

`middleware` Programvara som kopplar ihop olika tillämpningar och datorsystem för integrerad användning.

`modifierbarhet` Möjligheten att förändra ett system.

`object request broker (ORB)` En databuss som ser till att olika distribuerade objekt i ett system kan kommunicera oavsett om de flyttats sedan de senast kommunicerade med varandra eller om objekten är skivna i olika programspråk.

`plug-and-play` Enheter som ska sammankopplas behöver ingen manuella konfiguration för att passa ihop.

`portabilitet` Möjligheten att kunna överföra ett system mellan olika omgivningar.

`redundans` Lagring av samma data mer än en gång. Vanligen i samband med databaser och kan leda till en mängd uppdateringsproblem.

`skalbarhet` Möjligheten att utöka ett systems kapacitet utan omfattande kostnader och utan att göra förändringar i applikationerna.

`wrapping` En teknik för att anpassa en mjukvarukomponent till sin omgivning utan att förändra dess innehåll. Mjukvarukomponenten omges med ett skal, t.ex. genom att komponenten kapslas in i ett annat objekt eller genom att koppla ytterligare ett gränssnitt till komponenten.

Referenser

[Age02] Agero *Verksamhetsmodellering* <www.agero.se/Content/Objects/Tjanster/Tj%C3%A4nstebeskrivning%20%20Verksamhetsmodellering%20v1.pdf> (nov 2002): 1 nov 2002. webbsida

[Alw89] Jens Alwood (1989) *Modellering som analys- och specificeringsmetod* Eskilstuna: MILINF 90

[And98] Bengt E W Andersson (1998) *Samverkande informationssystem mellan aktörer i offentliga åtaganden* Linköping: Licentiatavhandling

[Are01] Arete Libro AB *Verksamhetsanalys* <http://www.libro.se/services/VA_kompetensblad.pdf> (- 2001): 1 nov 2002. webbsida

[Axe98] Karin Axelsson (1998) *Metodisk systemstrukturering* Linköping: triva-tryck ab

[Axe98b] Karin Axelsson, Göran Goldkuhl (1998) *Strukturering av informationssystem - arkitektursstrategier i teori och praktik* Lund: Studentlitteratur

[Bas98] Len Bass, Paul Clemens, Rick Kazman (1998) *Software Architecture in Practice* Addison-Wesley

[Ber99] Maria Bergenstjerna, Lena Johansson, Marina Wojtasik (1999) *Metoder för strategisk IT-management* Göteborg: Institutionen för informatik, Handelshögskolan, Göteborgs universitet

[Bre02] Bredemeyer Consulting *Software Architecture: Central Concerns, Key Decisions* <http://www.bredemeyer.com/pdf_files/Architecture-Definition.PDF> (maj 2002): 18 okt 2002. webbsida

[Cee02] Department of Computing and Electrical Engineering at Heriot-Watt University <http://www.cee.hw.ac.uk/> (jul 2002): 22 feb 2003. webbsida

[Cbl94] Computer Based Learning Unit at University of Leeds <http://cbl.leeds.ac.uk/ www/home.html> (dec 1994): 22 feb 2003. webbsida

[Chr98] Benneth Christiansson (1998) *Komponentbaserade informationssytem* Karlstad: Högskolan i Karlstad, Institutionen för informationsteknologi

[Chr99] Benneth Christiansson, Lars Jakobsson (1999) *Component-Based Software Development Life Cycles* Karlstad: Karlstad Universitet, Institutionen för informationsteknologi.

[Cle96] Paul C. Clements (1996) *A Survey of Architecture Description Languages* Pittsburgh, Pennsylvania, USA: Carnegie Mellon University

[Com01] Combitech Systems *On Time / Tema: Programvaruarkitektur* <http://www.combitechsystems.com/files/site/combitech/on-time/pdf/OT0024_webbnr.pdf> (jun 2001): 3 feb 2003. webbsida

[Cs03] School of Coputer Science <http://www.cs.cmu.edu/> (- -): 18 feb 2003. webbsida

[Cza97] Krzysztof Czarnecki (1997) *Leveraging Reuse Through Domain-Specific Software Architectures* Ulm, Tyskland: Daimler-Benz AG Research and Development

[Dav00] Thomas H. Davenport (2000) *Mission Critical.* Boston, Massachusetts, USA: Harward Business Scholl Press

[DFK02] Dataföreningen i Sverige *Certifierad IT-arkitekt* <http://www.dfs.se/kompetens/utbildning/harhant/default.asp?ID=541> (jan 2002): 24 okt 2002. webbsida

[DFK02b] Dataföreningen i Sverige *Certifierad databas- och informationsarkitekt* <http://www.dfs.se/kompetens/utbildning/harhant /default.asp?ID=741> (okt 2002): 27 jan 2003. webbsida

[Dsv02] Institutionen för Data- och systemvetenskap *Informationssystem* <http://www.dsv.su.se/utbildning/su/dsv160/is.html> (sept 2002): 12 sept 2002. webbsida

[Elm00] Ramez Elmasri, Shamkant B. Navathe (2000) *Fundamentals of Database Systems* Addison-Wesley

[Enq02] Håkan Enquist (2002) *Informationsarkitektur och militär ledning* Göteborg: Institutionen för informatik, Handelshögskolan, Göteborgs universitet

[Eri89] B Anders Eriksson (1989) *Systemering - från informationsbehov till informationssystem* Lund: Studentlitteratur

[Eur02] *European Software Institute* <http://www.esi.es/Euromethod/overview.html> (apr 2002): 25 sept 2002. webbsida

[Fle85] Per Flensburg, Hans Köhler (1985) *Hur skapas bra informationssystem?* Stockholm: Liber Tryck AB

[Gol82] Göran Goldkuhl, Anders Nilsson, Annie Röstlinger (1982) *Att specifiera informationssystem* Stockholm: LiberTryck

[Gom99] H. Gomaa, G.A. Farrukh *Composition of Software Architectures from Reusable Architecture Patterns* Worcester, USA: Worcester Polytechnic Institute, Computer Science

[Gor83] Nils-Göran Svensson (1983) *Vad är IRM/IA?* Stockholm: LiberTryck

[Hay97] The Data Administration Newsletter *The Zachman Framework: An Introduction* The Data Administration Newsletter Issue 1 <http://www.tdan.com/i001fe01.htm> (jun 1997): 4 nov 2002. webbsida

[Hof00] Christine Hofmeister, Robert Nord, Dilip Son (2000) *Applied Software Architecture* Addison-Wesley

[Hpi00] HP Invent *Meta Architecture* <http://www.architecture.external.hp.com/Overview/arch-_meta_architecture.htm> (- 2000): 12 nov 2002. webbsida

[IEEE03] Institute of Electrical and Electronics Engineers, Inc *IEEE Std 1471-2000* <http://www.ieee.org> (feb 2003): (10 jan 2003). webbsida

[Kin02] Metodnätet *IT-arkitekturer.* <www.metodnatet.org/Dokument/Konfmtrl/IT-arkitektur%20Tasse%20var2000.ppt> (maj 2000): 24 okt 2002. webbsida

[Kru01] Philippe Kruchten (2001) *The Rational Unified Process An Introduction* Addison-Wesley

[Lan78] Börje Langefors (1978) *Theoretical Analysis of Information Systems*. Lund: Studentlitteratur

[Lid99] Torun Lidfeldt (1999) *Välkommen till middleware och EAI*. Datateknik 3.0 nr. 7 nov 99

[Loo99] Monica Lööw *Att leda och arbeta i projekt - en praktisk handbok om att lyckas projekt*. Liber Ekonomi, Malmö.

[Luc95] David C. Luckham, James Vera *An Event-Based Architecture Definition Language* IEEE Transactions on Software Engineering, Vol 21, Nr 9, s.717-734. sep 1995

[Luc95b] David C. Luckham, John J. Kenney, Larry M. Augustin, James Vera, Doug Bryan, Walter Mann *Specification and Analysis of System Architecture Using Rapide* IEEE Press Piscataway, s 336-355 Periodical-Issue-Article 1995

[Lud99] Frank Lüders (1999) *Architectural Styles in Component-Based Software Enigneering* Västerås: Mälardalens Högskola, Instutionen för Datavetenskap

[Mag91] Thanos Magoulas, Kalevi Pessi (1991) *En studie om informationssystemarkitekturer* Göteborg: Chalmers Tekniska Högskola och Göteborgs Universitet.

[Max01] Bruce R. Maxim *Software Reuse and Component-Based Software Engineering*
<http://www.engin.umd.umich.edu/CIS/course.des
/cis376/ppt/lec22.ppt> (apr 2001): 6 jan 2002. webbsida

[Nil00] Anders G. Nilsson (2000) *Anskaffning av standardsystem för att utveckla verksamheter* Stockholm: Gotab

[Niv02] Stefan Nivall *Medicinsk systemteknik - Föreläsning 10, Systemarkitektur*
<http://www.s2.chalmers.se/iths/pdf/F10.pdf> (mar 2002): 27 nov 2002. webbsida

[Nis82] Hans-Erik Nissen *When People design an Information System - then the Information System Designs People* Evolutionary Information Systems IFIP, North-Holland s. 87-100 1982

[Pag03] *Paginas IT-ordbok* <http://www.pagina.se/itord/default.asp> (- -): 7 mars 2003: webbsida

[Pat94] Patel, R. Davidsson, B (1994) *Forskningmetodikens grunder* Lund: Studentlitteratur

[Rum99] B. Rumpe, M. Schoenmakers, A. RaderMacher, A. Schurr (1999) *UML + ROOM as a Standard ADL?* Engineering of Complex Computer Systems, ICECCS'99 Proceedings

[SEI02] *The Software Engineering Institute.* <www.sei.cmu.edu> (sept 2002): 8 jan 2003. webbsida

[Sha95] Mary Shaw, Robert DeLine, Daniel V. Klein, Theodore L. Ross, David M. Young, Gregory Zelesnik (1995) *Abstractions for Software Architecture and Tools to Support Them*journal = Software Engineering, Vol 21, Nr 4, s 314-335, 1995

[Sha96] Mary Shaw, Paul Clements (1996) *A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems* Pittsburgh, Pennsylvania, USA: Carnegie Mellon University.

[Sha96b] Mary Shaw, David Garlan (1996) *Software Architecture - Perspectives on an emerging discipline* New Jersey: Pearson Education

[TOG03] *The Open Group Home Page* <http://www.opengroup.org/> (feb 2003]: 24 nov 2002. webbsida

[Tur99] Efraim Turban, Ephraim McLean, James Wetherbe (1999) *Information technology for management* New York: John Wiley & Sons Inc

[Va02] *Office of Information and Technology* <http://www.va.gov/oirm/> (sept 2002): 6 nov 2002. webbsida

[Zac99] John Zachman (1999) *A Framework for information system architecture* IBM Systems Journal, Vol 39 1999 s.454-470

[Zac97] John Zachman *Information System Architecture - ISA.* <http://www.istis.unomaha.edu/isqa/vanvliet/arch /isa/isa.htm> (- 1997): 31 okt 2002. webbsida

[Zac02a] John Zachman *The Challange is Change - A Management Paper.* <http://members.ozemail.com.au/ visible/papers/ zachman2.htm> (mar 2002): 4 nov 2002.

[Zac02b] John Zachman *Enterprise Architecture and Legacy Systems* <http://members.ozemail.com.au/ visible/papers/ zachman1.htm> (mar 2002): 31 okt 2002. webbsida

[Zif02] *The Zachman Institute for Framework Advancement* <http://www.zifa.com/> (- 2002): 5 nov 2002. webbsida