FOVA 1-1 Utvikling av beslutningsstøttesystemer og persondatabaser

Informatiske emner

1 Introduksjon

Denne innledningen gir en oversikt over informatikk-delen av pensum i faget FOVA 1-1 ”Utvikling av beslutningsstøttesystemer og persondatabaser”. Spesielt vektlegges sammenhengene mellom de ulike deler av pensum som består av en rekke uavhengige bøker og publikasjoner. Referanser til pensum er inkludert for de ulike delene som omtales.

Hoveddelen av pensum legger vekt på *utviklingsprosessen* og hvilke hjelpemidler en har for å utvikle *databasekomponenten* i informasjonssystemer (ofte referert til som databaseserver). Spesielt omtales hvilke modelleringsspråk, metoder og teknikker som kan benyttes for å bestemme hvilke data som skal håndteres av en database, hvordan databasen kan realiseres ved hjelp av dagens teknologi og hvordan en kan få tilgang til og manipulere dataene i databasen.

Pensum er delt inn i fire hoveddeler som omtales mer detaljert under:

1. *Informasjonssystemer (IS) og utvikling av IS [1,2,3,7,9,11]*: gir en oversikt over IS og IS-utvikling (f.eks. begreper, ulike typer systemer, IS-utviklingsmodeller og utviklingsprosjekter)
2. *Modellering av statiske aspekter [1,2,4]*: gir en innføring i datamodellering ved hjelp av modelleringsspråkene Entity-Relationship (E-R) og Object-Role (O-R)
3. *Grunnleggende om databaser [1]*: gir en generell introduksjon til databaser, database-teknologi og bruk av databaser
4. *Trender innen IS utvikling og fokus på noen problemstillinger i praktisk systemutvikling [1,5,6,8,9,10,11,12,13,14,15,16,17]*: gir en oversikt over trender innen systemutvikling med vektlegging på databaser og CSCW / gruppevare. Denne delen gir også en innføring i verktøystøtte for systemutvikling samt kvalitetssikring og testing av programvare

Til sammen gir disse delene en bred innføring i informasjonssystemer og utvikling av databasekomponenten av disse. Både tradisjonell tilnærming til informasjonssystemutvikling og nyere trender blir omhandlet.

2 Informasjonssystemer (IS) og utvikling av IS

2.1 Generelt om informasjonssystemer

Informasjonssystemer finnes overalt i det moderne samfunn og i en rekke ulike versjoner. Generelt kan et IS defineres som et system for innsamling, bearbeiding, lagring, overføring og

presentasjon av informasjon [2]. Det finnes både manuelle og databaserte systemer. Databaserte informasjonssystemer er spesielt egnet der det kreves raskere bearbeiding av data, lagring og gjenfinning av store mengder data, større nøyaktighet i prosesser / resultater og der manuelle inngrep i verdikjeder må reduseres. Det finnes en rekke ulike informasjonssystemer samt en rekke forskjellige klassifiseringer av disse. Vi kan nevne noen typer:

- Transaksjonsbaserte systemer: online systemer som støtter den daglige drift av virksomheter og kjennetegnes ved et høyt antall transaksjoner som består av relativt enkle funksjoner (f.eks. ordre og faktureringssystemer)
- Embedded (sanntids) systemer: støtter sanntidsprosesser (f.eks. prosess-styringssystemer)
- Beslutningsstøttesystemer: støtter beslutningsprosesser i en virksomhet (f.eks. datawarehouse og data mining baserte systemer)
- Saksbehandlingssystemer: støtter saksbehandling/arbeidsflyt (f.eks. system for håndtering av lånesøknader i banker)
- Gruppevare (CSCW / datastøttet samarbeid/ samhandling / samarbeidsteknologi): støtter kommunikasjon, samarbeid og koordinering mellom mennesker (f.eks. PC-baserte konferansesystemer og møteromssystemer)

Da det ikke finnes noen enighet eller standard for hvordan systemer skal klassifiseres, blir de ulike systemtypene ofte brukt forskjellig avhengig av hvem som omtaler dem. Dette reflekteres også i de ulike fagretningene som finnes innen databehandling. De fokuserer ofte på ulike systemtyper eller ulike deler av systemutviklingen. Eksempler på noen fagretninger:

- Information Systems Engineering (på norsk ofte kalt “Informasjonsssystemer”)
- Requirements Engineering (konseptuell modellering)
- Business Process Re-Engineering
- CSCW (Computer Supported Cooperative Work)
- CASE-orientert utvikling
- Software Engineering

2.2 IS utvikling

IS utvikling eller systemutvikling omfatter alle aktiviteter fra forandringsstudie / problemdefinering til et system er ferdig implementert og satt i produksjon. Et system kan befinne seg i ulike tilstander avhengig av hvor i utviklingsprosessen en er. Dette blir ofte referert til som livssyklusen til et informasjonssystem der hver tilstand er resultatet av en rekke aktiviteter som inngår i systemutviklingen [2,3]:

- Forandringsanalyse (forandringsstudie / mulighets-studie): fokuserer på virksomhetens muligheter og problemer og resulterer i en problembeskrivelse som klart definerer målsetning, omfang, etc.
- Analyse: deles ofte inn i virksomhetsanalyse og systemanalyse. I første del analyseres virksomheten for å avgjøre hvordan systemet kan støtte den. Deretter vurderes og bestemmes systemets innhold (funksjonelle og ikke-funksjonelle krav). Resulterer i en kravsspesifikasjon
- Design (utforming / konstruksjon): hvor en først utvikler en prinsipiell teknisk løsning og deretter utvikler en prinsipiell utstyrstilpasset teknisk løsning (dvs. tar hensyn til leverandørspesifikke detaljer). Resulterer i en konstruksjonsspesifikasjon

- Implementasjon: realisering av systemet inkludert testing og det som behøves for å sette systemet i produksjon (f.eks. dokumentasjon). Resulterer i et system klar for produksjon der systemet overleveres til drift og vedlikehold
- Drift og vedlikehold (evolusjon / re-design): sørge for at systemet er tilgjengelig for brukerne, foreta feilretting når feilsituasjoner oppstår og små forbedringer av systemet. Hvis større forbedringer må foretas, blir dette ofte betraktet som et utviklingsprosjekt hvor alle fasene blir gjennomgått
- Utfasing (avvikling): systemet blir tatt ut av produksjon f.eks. byttet ut med et nytt system eller bare avviklet

En skiller ofte mellom systemutvikling og vedlikehold. Systemutvikling er alle aktivitetene inntil systemet er satt i produksjon. Dette er omtalt i størstedelen av pensum. Vedlikeholds-fasen dvs. feilretting og mindre endringer etter at systemet er satt i drift er hovedsaklig håndtert i [9].

.

Ulike problemer krever ulike løsninger og utviklingsmodellene kan brukes i ulike grad for ulike problemer. En rekke klassifiseringer av problemer har blitt foreslått. En grov gruppering er å skille mellom såkalte ”tamme” og “slemme” problemer. Tamme problemer karakteriseres ved at de kan lett beskrives i sin helhet, en kan skille mellom problem og løsning, løsningen kan testes, den har en naturlig form og kan lett gjenbrukes. Slemme problemer kjennetegnes ved at de har ingen endelige løsning, enhver formulering av problemet svarer til en formulering av løsningen og vice versa, problemet kan alltid løses på en bedre måte, kan bli sett på som et symptom av et annet problem og er essensielt unikt. I [3] beskrives ulike situasjoner som krever ulike krav til systemutviklerne. De deles inn i tre grupper henholdsvis rutine, problemløsning og problemdefinering. Rutine svarer til tamme problemer. I problemløsningssituasjoner vet vi hva som skal løses, men ikke hvilken arbeidsform som skal brukes. I problemdefineringsoppgavesituasjoner er det ikke klart hvilket problem som skal løses. En står ovenfor et slemt problem som man prøver å temme for å komme i en problemløsnings- og rutinesituasjon for delproblemer.

Hvilke faser en IS-utvikling består av (f.eks. navn på fasene, aktiviteter i de ulike fasene og rekkefølge), kommer an på hvilken prosessmodell som blir brukt (ofte refererert til som *utviklingsmodell* eller som *prosess modell for programvareutvikling*). En utviklingsmodell fokuserer på hva som skal gjøres i de ulike delene / fasene og hvor lenge det skal gjøres. Den gir derfor en beskrivelse av [7]:

  - Hvilke faser utviklingen består av
  - Hvilken rekkefølge fasene skal komme i
  - Etablerer kriteria for overgang fra en fase til neste (dvs. avslutningskriteria, kriteria for valg og inngangskriteria for de ulike fasene)

Det finnes en rekke forskjellige utviklingsmodeller og de kan grovt grupperes i seks hovedgrupper [7]:

- *Kod og fiks modeller* hører til de eldste måtene å utvikle systemer på. Utviklerne ble presentert et problem og satte umiddelbart igang med å utvikle programmer for deretter å rette / korrigere koden når feil ble oppdaget. Krav, konstuksjon, etc. ble håndtert underveis i kodinga
- *Stegvise modeller* deler utviklingsprosessen inn i et sett av steg som ble utført sekvensielt fra planlegging til koding og evaluering

- *Fossefall/vannfallsmodellen* ("waterfall") er en videreutvikling av stegvise modeller der feedback løkker mellom nabofaser tillates og prototyping går i parallell med kravs- og konstruksjonsspesifikasjon. Vannfallsmodellen er den mest utbredte og en rekke forskjellige varianter av denne har blitt utviklet
- *Evolusjonsmodeller* inndeler utviklingen i et sett av steg der løsningen blir gradvis utvidet basert på erfaringer i hver fase. Iterasjon og prototyping er sentrale begreper i denne modellen. I iterativ systemutvikling gjentas aktiviteter der omfanget av løsningen gradvis økes for hver iterasjon. Ved prototyping utvikles en prototype på et tidlig tidspunkt i prosessen og videreutvikles og kan eventuelt resultere i det systemet
- *Transformasjonsmodeller* ble utviklet for å håndtere problemer i de ovenevnte modellene for å håndtere forandringer i spesifikasjoner. I transformasjonsmodellene antas at høy nivås spesifikasjoner kan automatisk transformeres (omformes) til spesifikasjoner på lavere nivå dvs. i løpet av en rekke omforminger av modeller går en fra formelle spesifikasjoner til programvare. Endringer i systemet pga. ytelse problemer eller endringer i brukerkrav, kan utføres på høyere nivås spesifikasjoner for så å generere det ferdige systemet automatisk. På denne måten oppnås oppdaterte spesifikasjoner og reduserte kostnader til vedlikehold
- *Spiralmodeller* fokuserer på risiko og reduksjon av risiko i systemutviklingsprosessen. Utvikling følger en spiralmodell og deles i et sett av sykler i spiralen. Hver syklus starter med definering av målsetning (hva skal oppnås f.eks. funksjonalitet, ytelse, etc.), alternative måter å realisere løsning for dette (utvikle, kjøpe, etc.) og identifisering av begrensningene (kostnad, grensesnitt, etc.). Deretter følger evaluering av løsningsalternativer inkludert identifisering av risiki, forslag til reduksjon / eliminering av risiki, plan og aksept for neste syklus. Risiki håndteres systematisk fra syklus til syklus i spiralen

De ulike utviklingsmodeller representerer ulike tilnærminger til systemutvikling og gjenspeiler også evolusjonen innen programvareutvikling. En rekke utviklingsprosjekter har resultert i kostnadsoverskridelser, tidsplanen holder ikke og ofte er systemet ikke skikket til å løse det problemet det ble utviklet for. Det viser seg at en del av problemene kan spores tilbake til valg av utviklingsmodell. Fordeler og ulemper ved de ulike modellene blir diskutert i flere av pensumartiklene [2,3,7,8]. Kod og fiks modellen gir en rekke problemer på grunn av mangel på en strukturert tilnærmingsmåte til systemutviklingen. Mangel på kravs- og konstruksjonsspesifisering fører til at brukernes krav blir for dårlig ivaretatt, koden er dårlig strukturert og stadige endringer i koden blir kostbare og tidkrevende. Stegvise modeller ble utviklet som et resultat av problemene med kod og fiks modeller. De opprinnelige stegvise modellene viste seg også å være vanskelig å følge fordi en fase skulle gjøres helt ferdig før neste skulle påbegynnes og en kunne risikere at brukerne kun fikk se systemet like før det var planlagt til å bli satt i produksjon. Til tross for den mest brukte modellen, vannfallsmodellen, prøvde å bøte på dette med å inkludere feedback løkker og prototyping tidlig i utviklingsprosessen, viser det seg at den fremdeles har en rekke svakheter. Den er dokument-drevet og kravet til å dokumentere en fase før neste kan påbegynnes, gir en rigid modell og liten støtte til selve prosessen. Denne gir liten støtte til å løse oppgaver som krever problemløsning der problemet og kravene kanskje ikke er forstått før litt ut i prosessen og fokuserer på dokumentasjon istedet for på prosess. Den evolusjonære modellen har en del av de samme problemene som kod og fiks modeller hvis prototypen benyttes i det endelige systemet. Dette kan forøvrig bøtes på ved å bruke prototypen som en basis for å konstruere det endelige systemet. Bruk av prototyper uten nødevendig tid til å lage en konstruksjonsspesifikasjon for det endelige systemet og dens grensesnitt resulterer i systemer

som er vanskelig å vedlikeholde. I systemintegrasjonsprosjekter fører dette ofte til en rekke programgrensesnitt som må endres fordi en arkitekturbasert tenking er utelatt i prosessen.

Transformasjonsmodellene løser i teorien en rekke problemer med de foregående modellen med hensyn til vedlikeholdbare spesifikasjoner, men har en alvorlig hake. Den antar at man kan automatisk omforme en en høynivås spesifikasjon til spesifikasjoner på lavere nivå. I noen tilfelle er dette mulig, men på dette området er det fremdeles en rekke uløste problemer. I praksis har det vist seg at en kan støtte overgangen mellom de enkelte fasene, men kun i de siste fasene av utviklingen kan en automatisk generere spesifikasjoner på lavere nivå (f.eks. fra E-R modell til relasjonsdatabaseskjema). I [7] blir spiralmodellen introdusert som en radikal forbedring av de tradisjonelle modellene for systemutvikling, der risikokontroll og reduksjon av risiko står i fokus. Den har flere fordeler blant annet fokus på gjenbruk av spesifikasjoner, tar hensyn til ny innsikt og forandringer undervegs i prosessen og gir mulighet til å ta med programvarekvalitetsaspekter i utviklingsprosessen. Den er forøvrig lite utbredt og fremdeles er det flere uløste problemer før den kan benyttes i stort omfang i praksis. Den er vanskelig å håndtere kontraksmessig, eksperter innen risikoanalyse finnes det få av, selve modellen må forfines og støtte for syklusene må videreutvikles.

2.3 Prosjekter

Systemutvikling organiseres ofte som et prosjekt. Et prosjekt er en midlertidig organisatorisk enhet hvor arbeid utføres for å nå avgrensede mål [2]. Forløpet av prosjektet avhenger av situasjonen, arbeidsformene som velges og betingelsene (omgivelsene) for prosjektet. Ansvaret for resultatet av prosjektet blir vanligvis overført til linjeorganisasjonen i virksomheten. Utviklings-metodikken gir retningslinjer om hvordan de ulike fasene skal utføres og hvilken støtte som kan gis til de ulike aktivitetene, f.eks. hvilke modellerings-språk, verktøy, etc. som skal / bør brukes.

I et prosjekt defineres roller og ansvar for de enkelte aktørene som er innvolvert. Noen roller og ansvarsområder er kort beskrevet i det følgende. Prosjektet må alltid ha en eier og denne faller ofte sammen med linjeansvar i organisasjon og har budsjettet for prosjektet. Styringskomiten har overordnet ansvar for vurdering og godkjenning av delresultater, resultater, endringer, start/stopp, etc. Den har representanter fra ulike deler, for eksempel ledes av eier, representanter fra brukere, interessegrupper i organisasjon (finans, IT, nettverk, etc.) og representanter fra leverandør. Prosjektleder har typisk ansvar for prosess og resultat og ansvar for rapportering og eskalering. Brukere er de som skal bruke systemet i produksjon. En analytiker studerer brukerorganisasjonen, tekniske muligheter, utforming av løsning, etc. og er innvolvert i f.eks. kravsspesifisering. En utvikler utfører ofte design og implementerer. En konsulent er en ekstern rådgiver og kan også være prosjektdeltager. Vedlikeholder har ansvar for endringer etter system er i produksjon. Sammensetningen varierer sterkt fra prosjekt til prosjekt avhengig av faktorer som oppgaven dvs. type problem som skal løses, andre formål, interessenter, relasjon til brukerorganisasjon, standarder, lover, avtaler, etc., utstyr, bemanning og økonomisk situasjon.

Kontrakter brukes for å håndtere forpliktelser mellom prosjekter, prosjektinteressenter og prosjektrelasjoner. En skiller mellom produktkontrakter (basert på f.eks. kravspesifikasjoner), prosesskontrakter (f.eks. overordnete tidsplaner) og interne kontrakter i prosjektgruppen (f.eks. prosjektplaner).

2.4 Organisasjonsaspekter

I [3,11] viser forfatterne hvordan systemutviklernes antagelser om, og holdninger til, systemutviklingsoppgaver, organisasjoner og forventninger til systemutviklerrollen påvirker hele utviklingsprosessen. I en organisasjon er det viktig at roller og ansvar er bestemt og at alle vet og aksepterer hverandres roller og ansvar. Dette gjelder både internt i et prosjekt og mellom de ulike aktørene som prosjektet har en relasjon til. Utviklingsorganisasjonen er leverandøren av produktet. Brukerorganisasjonen er brukeren av produktet av prosjektet. Utviklingsorganisasjonen kan spille rollen som konsulent eller leverandør. Som leverandør vil det være lite samarbeid mellom utviklings- og brukerorganisasjonene. Forholdet blir regulert ved hjelp av en kontrakt og regelmessige kontroller. Som konsulent vil det være tett samarbeid mellom utviklings- og brukerorganisasjonene.

Videre blir det i [11] diskutert hvorfor organisasjoner ofter støter på mange problemer internt i organisasjonen i forbindelse med endring knyttet til informasjonssystemer. Det blir fokusert på tre områder: sosial treghet i organisajonen, motstand mot forandring og motimplementasjon.

3 Modellering av statiske aspekter

3.1 Generelt om modeller

I alle faser av utviklingsprosessen brukes forskjellige typer modeller (ofte kalt spesifikasjoner) som uttrykkes ved hjelp av modelleringsspråk. Eksempler er kravs-spesifikasjoner og konstruksjonsspesifikasjoner som er skrevet på norsk eller uttrykt ved hjelp av et formelt språk. Behov for å bruke modeller vil alltid være sentralt i utviklingsprosessen. Noen grunner til dette er:

- Uhensiktsmessig å måtte observere virkeligheten for studere / vurdere den
- En modell gir ofte en bedre innsikt enn observasjon (abstraksjonsmuligheter)
- En modell holder virkeligheten stille og muliggjør analyser
- En modell kan endres på måter en ikke kan /våger å gjøre i virkeligheten

En modell gir et forenklet beskrivelse av en del av virkeligheten som er av interesse for problemløsningen og brukes derfor som:

- Basis for kommunikasjon og forståelse mellom ulike aktører i utviklingsprosessen. Dette stiller store krav til modelleringsspråkets uttrykkskraft og brukervennlighet
- Basis for implementasjon av systemet. Dette stiller store krav til modelleringsspråkets uttrykkskraft og formelle egenskaper

En modell skal best mulig reflektere de sentrale aspekter ved virkeligheten (UoD) samt være lett å forandre når virkeligheten forandres (vedlikeholdbare spesifikasjoner). Viktige forutsetninger for vellykket modellering dvs. bruke modelleringsspråk til å lage en modell:

- Tilstrekkelig tid har blitt brukt til problem forståelse, bestemme målsetning, etc.

- Viktige interessenter er aktivt involvert (f.eks. eier og brukere)
- Hensiktsmessig prosess for utvikling av spesifikasjonen samt uttrykt på en hensiktsmessig måte
- Krever samme referanseramme og representasjonsregler

En skiller mellom tre hovedtyper av modeller: uformelle, formelle og hybride. Et eksempel på en uformell spesifikasjon er en kravsspesifikasjon uttrykt ved hjelp av norsk språk. Formelle spesifikasjoner er uttrykt ved hjelp av formelle språk f.eks. en kravsspesifikasjon uttrykt ved hjelp av DFD og E-R. Det kan diskuteres i hvilken grad DFD og E-R er virkelig formelle språk, men vi nøyer oss med å slå fast at de er mer formelle enn naturlig språk siden dens syntaks og semantikk er formelt beskrevet. Eksempler på hybride spesifikasjoner er en kravsspesifikasjon der ulike deler er uttrykt ved hjelp av norsk og mer formelle språk.

I utviklingsprosessen er det en tendens til å introdusere mer og mer formelle språk til lenger frem en kommer i prosessen. Typisk vil naturlige språk (f.eks. norsk) benyttes mye i tidlige faser. De er lette å bruke for å kommunisere mellom mennesker, men modellene er ofte tvetydige og inkonsistente. Når en kommer til analyse, design og implementasjonsfasene vil tekstlige beskrivelse erstattes av modelleringsspråk og programmeringsspråk.

3.2 Modelleringsspråk

Det finnes mange klassifiseringer av modelleringsspråk. De kan grupperes etter hvilket perspektiv (orientering / modelleringsretning) de støtter, hvor perspektiv blir bestemt av språkets kjernebegreper og prinsipper. Eksempler er d*ata-orienterte språk (*fokus på hvilke data som behøves f.eks. E-R og O-R), *prosess-orienterte språk* (fokus på hvordan systemet skal fungere/hva det skal gjøre f.eks. dataflytdiagram), *objekt-orienterte språk* (fokus på objekter og meldingsutveksling mellom disse f.eks. OMT, OOA og Wirfs-Brock) og *regel-baserte språk* (fokus på hvilke regler som gjelder i virksomheten f.eks. XCON, Sapiens og triggere i Oracle og DB2). Ofte skiller en også mellom statiske, dynamiske og temporale modelleringsspråk. De statiske ignorer tidsdimensjonen og modellerer domenet (UoD – Universe of Discourse) som et øyeblikksbilde av begreper. Hvilke begreper som benyttes, er avhengig av hvilket språk som blir brukt. I ER-språket brukes f.eks. entiteter, relasjoner (forbindelser), attributter, etc. Dynamiske språk opererer med to tidstilstander, en tid før og en tid etter at en bestemt hendelse eller aktivititet har funnet sted. Dynamiske språk fokuserer på disse tilstandene og mulige transisjoner (overganger) mellom disse. Temporale språk er de mest uttrykkskraftige språkene og en kan referere til et vilkårlig tidspunkt i domenet.

Metoder gir retningslinjer for hvordan et modelleringsspråk kan benyttes til å utvikle modeller. For å effektivisere modelleringsprosessen brukes modelleringsverktøy (f.eks. editorer).

Hoveddelen av pensum fokuserer på konseptuell modellering av data ved hjelp av modelleringsspråkene Entity-Relationship (E-R) [1] og Object-Role (O-R), tidligere kalt NIAM [4]:

- Kapittel 5 i [1] beskriver viktige begreper i modellering og modelleringens rolle i utviklingsprosessen samt gir en oversikt over modelleringsspråket E-R og hvordan dette kan benyttes for å utvikle en database
- Kapittel 6 i [1]: gir en en oversikt over konstruksjon av en database fra tilegnelse av bruker krav til en har en konseptuell beskrivelse av databasen representert ved E-R. Ulike tilnærmingsmåter for database konstruksjon blir også beskrevet
- Kapittel 7 i [1]: fokuserer på logisk konstruksjon av databasen dvs. basert på en konseptuell beskrivelse i E-R blir et relasjonsdatabaseskjema utviklet. Kapitlet gir retningslinjer for hvordan denne omformingen kan utføres
- Kapittel 3 i [2]: gir en generell introduksjon til modellering ved hjelp av O-R
- Kapittel 4 i [2]: beskriver mer avanserte egenskaper ved O-R, spesielt hvordan begrensninger kan uttrykkes i O-R

I tillegg blir andre modelleringsspråk kort behandlet i følgende deler:

- Kapittel 11 i [1]: gir en kort oversikt over viktige egenskaper ved objekt-orienterte språk
- Kapittel 12 i [1]: beskriver aktive regler eller triggere og hvordan regel-baserte språk kan brukes i IS-utvikling
- Kapittel 13 i [1]: viser to alternatitve måter å modellere data for en datawarehouse-løsning, stjerneskjema og snøflakskjema

4 Databaser

4.1 Generelt om databaser

Databaser er en sentral komponent i de fleste informasjonssystemer og beskrives generelt i kapittel 1 i [1]. En database er en samling av relaterte data (f.eks. navn, adresse og telefonnummer). Den representerer en del av den virkelige verden (referert til som Universe of Discourse), er en logisk koherent samling av data som har en bestemt mening og er konstruert, bygget og fylt med data for en bestemt hensikt. Databasen har en bestemt gruppe brukere og bestemte anvendelser som brukerne er interessert i. En tilfeldig samling av data kan derfor ikke refereres til som en database.

En database kan utvikles og vedlikeholdes manuelt (f.eks. kortarkiv i et bibliotek) eller ved hjelp av en datamaskin (f.eks. utviklet ved hjelp av et DBMS). Et database management system (DBMS) er generell programvare som støtter definering, konstruering og manipulering av databaser. Ved definering av en database spesifiseres datatyper, relasjoner og begrensninger (”constraints”). Konstruering omfatter lagring av data på et lagringsmedium som blir kontrollert av et DBMS og manipulering omfatter operasjoner mot databasen f.eks. lesing og oppdatering.

Før databasebegrepet ble innført ble filprosessering benyttet. En fil tilhørte alltid kun ett program. Hvis programmer skulle dele data, ble filene kopierte. Dette førte til store problemer med dataadministrasjon og datakvalitet. Introduksjonen av databaser utgjorde derfor store forbedringer for virksomheters håndtering av data. De viktigste fordelene er bedret kontroll av redundans, begrensning av ikke-autorisert aksess til data, persistent lagring av program objekter og data strukturer, database interferens kunne håndteres ved hjelp av regler, støtte til flerbrukergrensesnitt, kompliserte relasjoner mellom data kunne representeres, støtte til

backup og recovery, muligjør bruk av standarder, reduserer systemutviklingstid, tilgjengeliggjør oppdatert informasjon og ikke minst “economies of scale”.

Det er forøvrig en rekke utfordringer ved databaseutvikling som må håndteres for å oppnå disse fordelene. Dette inkluderer blant annet utvikling av en adekvat datastruktur, etablering regime for å sikre datakvalitet samt håndtering av og møte kravene fra mange ulike brukere (både mennesker og andre systemer). De fleste organisasjoner som har store informasjonssystemer har fått føle hva det koster å utvikle en adekvat datastruktur. Selv om en rekke hjelpemidler som modelleringsspråk og verktøy er tatt i bruk, er fremdeles modellering av data en av de store utfordringene i systemutviklingsprosessen. Det er ofte store mengder data involvert og disse blir brukt i en årrekke. Mislykkes datamodelleringen kan det ha store organisasjonsmessige / forretningsmessige konsekvenser.

En skiller mellom databasers *intensjon* og *ekstensjon*. Intensjonen refererer til beskrivelsen av databasen som kalles databaseskjemaet eller meta-data. Databaseskjemaet er resultatet av databasedesign og lagres i en datakatalog. Selve databasen som inneholder instansene eller dataene (f.eks. anne helga, oslo, etc.) kalles ekstensjon.

4.2 Databasearkitektur

En tre-lagsmodell for databaser er allment akseptert, men ikke alle kommersielle systemer følger den stringent. I tre-lagsmodellen skilles det mellom internt, konseptuelt og eksternt skjema [1]. Det interne skjemaet utgjør den fysiske representasjonen av data i en database (filorganisering, aksess-strukturer, etc.). Alle dataene beskrives i det konseptuelle skjemaet. Utsnitt eller ”views” av det konseptuelle skjemaet blir beskrevet i det eksterne skjemaet og det er dette de ulike brukerne ser. På denne måten oppnås uavhengigheter mellom ulike representasjoner og en kan for eksempel gjøre tekniske oppgraderinger som kun impliserer at en må gjøre endringer i det interne skjema. Det konseptuelle, eksterne skjema (og dermed applikasjoner) kan være (tilnærmet) uforandret. Databasetilnærmingsmåten tilrettelegger for dataabstraksjon ved å skjule detaljer som ikke er relevante for noen brukere/ brukergrupper.

Under databasekonstruksjon defineres de ulike skjemaene. Den konseptuelle modellen som beskriver dataene i ved hjelp av f.eks. modelleringsspråket E-R blir reorganisert og omformet til en logisk datamodell. En rekke logiske datamodeller er utviklet. Eksempler er hierarkiske modeller (basert på trestruktur), nettverksmodeller (CODASYL), relasjonsbaserte modeller og objektorienterte modeller. I første del av et typisk databasedesign tar en kun hensyn til prinsipielle tekniske aspekter. Dette innebærer blant annet en omforming av E-R modellen til f.eks. relasjonsmodellen. I andre del tar man hensyn til utstyrsspesifikke hensyn f.eks. hvilke spesielle tekniske hensyn som gjelder for Sybase DBMS. Under reorganiseringen i begge fasene tar man også hensyn til den estimerte databaselasten for systemet i bruk (f.eks. datavolumer og type spørringer) for å bedre ytelsen.

4.3 Relasjonsdatabaser

I kapitlene 2-3 i [1] blir relasjonsmodellen som er den mest utbredte datamodellen beskrevet. En datamodell er et sett av konsepter som kan beskrive databasestrukturen og består av to hoveddeler: *datastrukturer* (datatyper, relasjoner og begrensninger som gjelder for dataene)

og et sett av *operasjoner* for lesing og oppdatering av dataene i databasen. I tillegg er det vanlig at en datamodell kan utvides med brukerdefinerte operasjoner.

Dataene i en relasjonsdatabase representeres av en samling av relasjoner (tabeller). En tabell består av rader eller tupler og kolonner som svarer til attributter. For tabellene gjelder følgende regler:

- Ingen tuppel redundans, dvs. et tuppel er representert bare et sted i tabellen
- Ingen tuppel ordning, dvs. tupler kan lagres i vilkårlig rekkefølge i tabellen
- Attributt ordning, men refereres ved navn
- Atomiske attributt verdier (enkle dataverdier)

For å identifisere dataene i tabellene introduseres nøkler (identifikatorer). En skiller mellom kandidatnøkler (nøkler som identifiserer et tuppel i en tabell entydig), primærnøkler (viktigste kandidatnøkkel) og fremmednøkler (referential integrity – dvs. refererer til nøkkelen /nøklene i en annen relasjon). I tillegg introduseres et nytt konsept for å håndtere at en attributts verdi ikke finnes, at verdien er ukjent eller at en simpelthen ikke har informasjon. Disse håndteres ved hjelp av null-verdier. Ulike begrensninger (beskrankninger / "constraints") kan være av to typer: intra-relasjon (innenfor en tabell og kan være tuppel eller domene f.eks. key constraint) og inter-relasjon (på tvers av tabeller, f.eks. referential integrity).

For relasjonsbaserte databaser brukes relasjonsspråk for å definere operasjonene mot dataene i databasen (oppdatere, manipulere, etc.). En skiller mellom to klasser av relasjonsspråk: relasjonsalgebra og relasjonskalkyle. Relasjonsalgebra er et prosedurelt språk og beskriver hvordan man skal komme frem til et resultat. Det består av et sett av operasjoner mot tabeller, der hver operasjon gir en tabell som resultat. Relasjonskalkyle er et deklarativt språk og en spesifiserer egenskapene ved resultat når en formulerer en spørring mot databasen. Det består av utvelgelse av tabellinformasjon ved hjelp av en mapping (predikater).

Et DBMS kan tilby flere språk for å definere en databasestruktur og for å manipulere dataene. En skiller mellom et *datadefineringsspråk* (DDL - Data Definition Language) og et *datamanipuleringsspråk* (Data Manipulation Language). DDL brukes for å definere konseptuelt og internt skjema og brukes av DBA (DataBase Administrator) og database designere. DML brukes for sette inn og manipulere dataene. SQL (Structured Query Language) ble utviklet ved IBM San Jose og er det mest brukte databasespråket (de facto standard fra1983). SQL er basert på tuppel-basert relasjonskalkyle og har både en DDL og en DML del.

I pensum behandles relasjonsmodellen og konstruksjon av relasjonsdatabaser på følgende steder:

- Kapittel 7 i [1] beskriver hvordan E-R modellen kan omformes til en relasjonsmodell.
- Kapittel 2 i [1] beskriver generelle egenskaper ved relasjonsmodellen med vektlegging på de ulike komponentene som behøves for å definere et relasjonsskjema (dvs. datastrukturdelen av relasjonsmodellen)
- Kapittel 3 i [1] beskriver to generelle språk for å manipulere data i en relasjonsmodell henholdsvis relasjonsalgebra og relasjonskalkyle
- Kapittel 4 i [1] presenterer SQL som blir brukt i virkelige systemer for å definere, oppdatere og utforme spørringer i en relasjonsdatabase (dvs. kombinerer egenskapene beskrevet i kapitlene 2-3 i [1])

4.4 Generelt om databaseteknologi

Et informasjonssystem består hovedsakelig av en databasedel, en applikasjonsdel og en del som utgjør brukergrensesnittet. Applikasjoner varierer i stor grad fra å ha alle disse delene som en del til å ha de ulike delene på ulike noder (dvs. ulike maskiner). Senere vil vi referere til dette som henholdsvis et-lags, to-lags og tre-lags arkitekturer. I databasedelen som vanligvis blir kalt databaseserver eller databasetjener kan man skille mellom de som utgjør selve databasen, beskrivelsen av disse dataene og de mekanismer som skal til for å håndtere dette. Disse mekanismene blir stort sett i alle nyere systemer (i flerbrukermiljø over en viss størrelse) tilbudt av et DBMS (DataBase Management System). Eksempler på kommersielle DBMS er Informix, Oracle og Sybase.

Et DBMS består av fem hovedkomponenter:

- Optimaliseringsmodul for spørringer (”optimizer”): velger ut strategi for data aksess for å sikre den beste responstida for spørringer
- Aksessmetode håndterer (”access method manager”): aksesserer dataene basert på strategien bestemt av optimaliseringsmodulen
- Buffer håndterer (”buffer manager”): håndterer overføringen av sider (også referert til som blokker i operativsystem termer) mellom sekundær- og primærminne. Den håndterer også store deler av primærminne som er allokert til DBMS og er som regel delt av mange applikasjoner
- Pålitelighetskontroll (”reliability control system”): ansvarlig for å sikre dataene i databasen ved eventuelle feilsituasjoner
- Samtidighetskontroll (”concurrency control system”): håndterer samtidig aksess til databasen og at interferens mellom applikasjoner ikke fører til tap av data og dermed konsistensproblemer

I dette faget antas at teorier for å utvikle ulike filstrukturer, bestemme strategier for å aksessere data og selve aksesseringen av data har inngått i andre kurs som studentene har hatt tidligere i studiet (f.eks. filsystemer og operativsystemer). I pensum er det derfor fokus på håndtering av transaksjoner, samtidighetskontroll og feilhåndtering som er sentrale konsepter i alle DBMS.

4.5 Transaksjonsbegrepet

Sentralt i håndtering av databasespørringer er transaksjonsbegrepet. En transaksjon er en utførelse av et program som aksesserer eller endrer innholdet av en database (lesing/skriving). I enbruker miljøer utføres en transaksjon i sin helhet før neste starter. I flerbruker systemer er ikke dette effektivt og eksekvering av transaksjoner skjer i såkalt blandet (”interleaved”) modus. Med dette menes at aksjoner eller instruksjoner fra en ulike transaksjoner blir utført på en slik måte at CPU-en ikke blir stående ledig å vente på at f.eks. sider skal leses inn i minnet. Isteden blir aksjoner fra andre transaksjoner utført i mellomtiden. For å øke ytelsen kan en forandre konfigurasjonen av systemer og legge til flere CPU-er.

Det er nødvendig å sikre at transaksjoner som blir utført blandet eller i parallell modus ikke får tilgang til hverandres data på en ukontrollert måte. Hvis dette skjer, kan resultatet av transaksjonen innholde feil. Typer problemer som må håndteres er [1]:

- En transaksjon leser en verdi som blir oppdatert av en annen transaksjon
- En transaksjon leser en verdi som har blitt oppdatert av en annen transaksjon, men som i mellomtiden blir kansellert
- Aggregering av data i en transaksjon som i mellomtiden blir oppdatert av andre transaksjoner

For å håndtere transaksjoner på korrekt måte og dermed unngå feilsituasjoner som beskrevet over har man utviklet flere teorier som blant annet tar utgangspunkt i atomiske transaksjoner og utviklet ulike algoritmer for å håndtere at dataene i en database blir oppdatert korrekt og samtidig som ytelsen til systemet blir ivaretatt (teknikker for samtidighetskontroll). I tillegg må systemer sørge for disse dataene forblir tilgjengelige selv om ytre feilsituasjoner oppstår (gjenopprettingsteknikker).

En atomisk transaksjon er en logisk enhet for database prosessering som oppfyller ACID-egenskapene**:**

- A(tomicity): En atomisk transaksjon blir enten utført i sin helhet eller ikke utført i hele tatt
- C(onsistency): En korrekt utførelse av en transaksjon tar databasen fra en konsistent tilstand til en annen konsistent tilstand
- I(solation): En transaksjon skal ikke gjøre sine oppdateringer synlig for andre transaksjoner før den er fullført i sin helhet (dvs. “committed”)
- D(uration): Når en transaksjon har endret databasen og transaksjon er fullført, må disse endringene ikke gå tapt pga. feil i etterkant

Atomicity og varighets egenskapene blir håndtert ved hjelp av pålitelighets teknikker. Konsistens blir tatt hånd om av programmerere som lager database programmer. Håndteres av DBMS modulen som sjekker at operasjoner ikke er i konflikt med integritets-begrensningene / reglene. Isoleringsegenskapen blir ivaretatt av teknikker for samtidighetskontroll.

4.6 Samtidighetskontroll og gjenoppretting

Hovedtilnærmingsmåter for å håndtere samtidighetskontroll er beskrevet i kapittel 9 i [1]:

- View ekvivalens
- Konflikt ekvivalens
- To-fase låsing
- Tidstempling

Gjenoppretting (“recovery”) behøves for å håndtere feilsituasjoner dvs. systemet skal ikke miste data om noe uforutsatt skjer f.eks. at systemet går ned. En kan klassifisere ulike typer feil: system kræsj (hw/sw), transaksjon/system feil (integer overflow, deling med null, ^C i Unix, etc.), unntak oppdaget av transaksjon (f.eks. ikke funnet data), disk kræsj og katastrofer.

Gjenoppretting er beskrevet i kapittel 9 i [1]. Sentralt i håndtering av feilsituasjoner er en systemlog. Den brukes for å logge alle operasjoner i transaksjoner som endrer database

tilstanden (dvs. oppdateringer). Hver enkelt transaksjon identifiseres ved hjelp av en transaction_id som genereres automatisk av databasesystemet. Systemloggen lagres regelmessig på disk og brukes for å gjenopprette tilstanden i databasen rett før feilsituasjonen oppstod.

5 Trender innen IS utvikling og fokus på noen problemstillinger i praktisk systemutvikling

Noen trender som har hatt stor påvirkning på dagens IS utvikling:

- Nyere typer databasesystemer (distribuerte databaser, objektorienterte databaser og aktive databaser).
- Dataanalyse
- WWW-teknologi og databaser
- CSCW / gruppevare

5.1 Distribuerte databaser

Skiller mellom to hovedtyper funksjonalitet:

- OLTP fokuserer på effektiv og pålitelig behandling av transaksjoner samt høyt antall transaksjoner. Dette beskrives i kapittel 10 i [1]
- OLAP fokuserer på dataanalyse som kjører på dedikerte data warehouse servere, spesialisert for datahåndtering for beslutningsstøtte. Dette beskrives i kapitlene 10 og 13 i [1]

En distribuert database er en samling av data som logisk hører til det samme systemet, men som fysisk er distribuert på en rekke noder som er koplet i et nettverk. Fordeler ved distribuerte systemer er:

- Database applikasjoners distribuerte natur
- Økt pålitelighet og tilgjengelighet
- Tillater deling av data samtidig som beholder lokal kontroll (varierende grad!)
- Forbedret ytelse

Dette forutsetter mulighet til å aksessere fjernnoder og overføre forespørsler og data mellom de ulike nodene via et nettverk samt mulighet til å holde styr på datadistribuering og replikering i en DDBMS. Dette er fremdeles meget store utfordringer for de fleste organisasjoner/bedrifter. Det finnes flere ulike paradigmer for datadistribusjon. Klient-tjener arkitektur / klient-tjener teknologi er sentral i de fleste nyere informasjonssystemer. I en klient-tjener arkitektur er det logiske entiteter som arbeider sammen over et nettverk for å utføre en oppgave. Klient/tjener applikasjoner er et forhold mellom prosesser som kjører på like eller ulike maskiner. En klient forespør/bruker tjenester og en tjener (eller server) tilbyr tjenester. Middleware er limet som gjør det mulig å kommunisere mellom klienten og

tjenere. Ressurser deles ved at en tjener kan yte tjenester til mange klienter samtidig og regulere aksessen til klientene.

En konfigurasjon der brukergrensesnittet og applikasjonskoden er lokalisert på klienten blir ofte referert til som en to-lags arkitektur. I dette tilfellet snakker vi også om en tjukk klient ("fat client"). Hvis en legger til en server der applikasjonskoden legges, kaller vi konfigurasjonen er tre-lags klient-tjener arkitektur.

Distribuerte databaser består av mange database servere som brukes av samme applikasjon. Parallelle databaser består av mange datalagringsenheter og prosessorer som opererer i parallell for å øke ytelsen. Replikerte databaser har data som logisk representerer den samme informasjon, men som er fysisk lagret på ulike servere. Datawarehouses er servere som er spesialisert for håndtering av data dedikert for beslutningsstøtte.

Distribuering av data skjer ved hjelp av fragmentering. En skiller mellom to typer av fragmentering: horisontal fragmentering og vertikal fragmentering. Data kan fragmenteres på mange måter. Den er korrekt hvis den er komplett og gjenopprettelig. I tillegg vil grad av redundans varierer avhengig av krav til f.eks. responstid og pålitelighet.

For distribuerte systemer skiller en mellom ulike transparensnivåer dvs. i hvilken grad distribueringen av data må være kjent for programmererne av systemene:

- Fragmenteringstransparens: programmerer behøver ikke å ta hensyn til om databasen er distribuert eller fragmentert
- Allokeringstransparens: programmerer må kjenne til fragmenteringsstruktur, men ikke hvor den er lokalisert
- Språkstransparens: programmerer må kjenne til både fragmenteringsstruktur og lokasjon for fragmenter
- Hvis ingen transparens støttes, har hvert DBMS sin egen SQL dialekt dvs. heterogene systemer og DBMSs har ingen felles interoperabilitetsstandard

På samme måte som for sentraliserte databaser, er transaksjonskonseptet. Transaksjoner kan klassifiseres på følgende måte:

- Remote requests: read-only transaksjoner som består av et vilkårlig antall SQL spørringer som blir sendt til en fjern DBMS (kun forespørsler)
- Remote transaksjoner: består av et vilkårlig antall SQL kommandoer (select, insert, etc.) som blir sendt til en fjern DBMS (skriver)
- Distribuerte transaksjoner: består av et vilkårlig antall SQL kommandoer (select, insert, etc.) som blir sendt til et gitt vilkårlig antall fjerne DBMSs, men hver SQL kommando refererer til et enkelt DBMS (oppdaterer mer enn en DBMS, krever 2-fase commit)
- Distribuerte forespørsler er vilkårlige transaksjoner, der hver SQL kommando kan referere til hvilken som helst

ACID-egenskapene gjelder også for transaksjoner i et distribuert system. Data distribuering influerer ikke på konsistens og er ikke avhengig av distribusjon fordi integrity constraints beskriver bare lokale egenskaper og varighet blir garantert av hvert system med lokale recovery mekanismer. Andre deler må ha store forbedringer som for eksempel optimaliseringsmodulen for spørringer, samtidighetskontroll og pålitelighetshåndtering.

Felles for alle distribuerte databaser er at programvare og data er distribuert på ulike noder og koplet sammen via et nettverk. Det finnes mange forskjellige typer og de beskrives i forhold til grad av programvare homogenitet, grad av lokal selvstendighet og grad av distribusjonstransparens (skjema integrasjon). Dagens distribuerte databasesystemer fungerer ikke tilfredstillende i heterogene databasemiljøer. Noen årsaker til dette er at de innkapsler dårlig data og tjenester (f.eks. endring på en server som aksesseres av andre skaper ofte trøbbel) og krever for mange lav-nivås kommandoer for å fungere godt (SQL nivå, ikke stored procedures eller RPCs styrt av en TP monitor).

Portabilitet og interoperabilitet er to egenskaper som er sentrale for distribuerte systemer. Portabilitet er muligheten for å overføre og kjøre et program fra et miljø til et annet miljø (f.eks. fra stormaskin til Unix). Interoperabilitet er muligheten for interaksjon mellom heterogene systemer. Eksempler på standarder er ODBC og DTP.

5.2 Objektorienterte databaser

En generell beskrivelse av objektorienterte databaser presenteres i kapittel 11 i [1]. De integrerer egenskaper til databaser med egenskaper til objektorientert programmeringsspråk. Typiske begreper som blir støttet:

- Objekter: informasjon og operasjoner (metoder) innkapslet ("encapsulation")
- Objekter i et generaliseringshierarki arver superobjekters egenskaper ("inheritance")
- Objekter kommuniserer ved hjelp av meldinger
- Abstraksjon ("information hiding"): intern representasjon som aksesseres via et veldefinert eksternt grensesnitt

Objektorienterte DBMS er ofte basert på en klient-tjener arkitektur som håndterer persistente objekter for deling av data i et flerbrukermiljø. De håndterer samtidig aksess til objekter, låsing og transaksjonsbeskyttelse og håndterer objektene i tillegg til vanlig DBMS funksjonalitet (backup, recovery, etc.). I [10] omhandles bruk av objektorientering i heterogene distribuerte systemer.

5.3 Aktive databaser

Aktive databasesystemer er beskrevet i kapittel 12 i [1]. De er DBMS-er som støtter definering og håndtering av produksjonsregler (også kalt aktive regler eller triggere). Det uavhengig subsystemet som kontrollerer utføringen av regler kalles en regelmotor. Den holder greie på hvilke hendelser som skjer og har ansvaret for at eksekvering av reglene blir utført korrekt. Regler har en struktur som typisk ser slik ut (finnes en rekke varianter):

- WHEN <event>
- IF <condition>
- THEN <action>

En aktiv database kan utføre transaksjoner initiert av brukere eller av regler (dvs. kontrollert av systemet). Systemer som støtter transaksjoner initiert av regler sies å ha en reaktiv

oppførsel. Kunnskap av reaktiv type behøves ikke plasseres i applikasjonskoden, men kan uttrykkes ved hjelp av regler (kalles også kunnskapsuavhengighet).

Basis karakteristika for triggere:

- Triggere er basert på event-condition-action (ECA) paradigmet
- Triggere defineres generelt ved hjelp av et DDL
- Triggere blir definert i forhold til hendelser som skjer i forhold til en tabell (ofte referert til som måltabell)
- Skiller mellom to nivåer av granularitet for relasjonsbaserte triggere: rad nivå (eller tuppel-orientert) der aktivering skjer for hvert tuppel i operasjonen og setningsnivå der aktivering skjer en gang for hvert SQL primitiv dvs. refererer til alle tuplene på en sett orientert måte
- Evaluering av triggere kan ha umiddelbar eller utsatt funksjonalitet

I tillegg til basis egenskapene for trigger, har en rekke tilleggsegenskaper for aktive regler blitt utviklet (og som øker uttrykkskraften til aktive regler):

- Temporale hendelser (f.eks. 19.10.1999)
- Bruker-definerte hendelser (f.eks. 'presidentbesøk')
- Hendelsesuttrykk: en trigger kan avhenge av flere hendelser som er relatert ved ulike operatorer
- Istedenfor modus: utvidelse i forhold til før / etter modi
- 'Detached' betraktning / utføring av regler: kan endre hvilken transaksjon en regel skal betraktes / utføres i forhold til
- Prioritet kan brukes for å løse konflikter mellom regler som blir aktivert av samme hendelse
- Regelsett: regler kan organiseres i sett som kan bli aktivert / deaktivert separat

En skiller mellom interne og eksterne anvendelser av aktive regler. Eksempler på interne anvendelser i en database er håndtering av integritetsbegrensninger, utledning av data, håndtering av replikerte data, versjonshåndtering og håndtering av sikkerhet. Eksterne anvendelser er forretningsregler som uttrykker kunnskap som er spesifik for en anvendelse.

5.4 Dataanlyse

Dagens virksomheter setter ofte opp egne miljø for å håndtere dataanalyse. Det er utviklet en rekke spesialiserte verktøy for å støtte dataanalyse for derigjennom å forbedre beslutningskvalitet og redusere beslutningstid. Teknologien for dataanalyse kalles OLAP (OnLine Analysis Processing) og baseres på dedikerte data warehouse servere som spesialiserte for håndtering av data for beslutningsstøtte. Disse karakteriseres ved få men ofte kompliserte spørringer). Datawarehouses er servere som er spesialisert for håndtering av data dedikert for beslutningsstøtte. Resultatet av dataanalysen blir ofte brukt som basis for f.eks. planlegging, strategiutforming og kundesegmentering.

Data blir ekstrahert og filtrert fra diverse datakilder for deretter å bli lastet inn i datawarehouse-løsningen. En typisk datawarehousearkitektur består av følgende:

- Filtrering: sjekker nøyaktighet av data før de blir satt inn i databasen

- Eksport: ekstraherer data fra datakildene (f.eks. fra produksjonssystemer). Inkrementell prosess der en samling av endrete data blir laget
- Laster: initiell lasting av data inn i et datawarehouse og har også ansvaret for sortering, aggregering og konstruksjon av de rette data strukturene
- Oppfrisker (refresher): oppdaterer innholdet av et datawarehouse inkrementelt avhengig av endringene som har skjedd i datakildene. En skiller mellom data shipping og transaction shipping
- Data aksess: utfører dataanalyse operasjonene mot dataene i et datawarehouse (generelle operasjoner og datawarehouse spesifikke operasjoner f.eks. roll up, drill down, data cube)
- Data mining: analyse for å identifisere / oppdage gjemte / ukjente mønster i dataene
- Eksport: eksportere data til andre datawarehouse-løsninger

I tillegg til disse inneholder en datawarehouse-løsning ofte verktøy for design og administrasjon av løsningen. Omfanget av dataene i en datawarehouse-løsning er ofte begrenset til subsett av den totale datamengde i en virksomhet. På samme måte som i utviklingen av en database brukes modelleringsspråk for å modellere dataene. Stjerneskjema og snøflaksskjema er de mest utbredte modelleringsspråkene. Stjerneskjema er basert på E-R, men deler konseptene i to deler: fakta og dimensjoner. Modellene blir strukturert rundt ulike fakta. Snøflakskjema er utvidelse av stjerneskjemaet til struktuerering av dimensjonene hierarkisk. På samme måte som for en vanlig RDBMS har datawarehouse-løsninger språk for definering av skjema og språk for manipulering av skjema.

Datamining er analyseteknikker for å finne mønster i dataene i et datawarehouse. De er basert på teknikker for datahåndtering, statistikk og kunnskapsbaserte teknikker. En typisk datamining prosess består av [1]: forståelse av domene, forberede datasett, oppdagelse av mønster i dataene og evaluering av mønster.

5.5 WWW og databaser

Basisteknologiene for WWW har vært kjent lenge. En løsning som ble basert på menneskets tankeprosess lansert på starten av 60-tallet i USA (hypertekst basert). En primitiv del av kommunikasjonsløsningen ble utviklet av akademiske og militære miljøer i USA på 70-tallet. Den ble videreutviklet av CERN i Sveits for å gjøre det mulig å dele informasjon mellom arbeidsgrupper som jobbet i ulike land. På starten av 90-tallet lyktes det å komme til enighet om et kommunikasjonsspråk i nettet som kunne brukes uavhengig av plattform. Til sammen har dette gjort internet revolusjonen mulig. Kapittel 14 i [1] gir en kort introduksjon til WWW (World Wide Web) samt koplinger mellom WWW og databaser.

En skiller mellom ulike tre hovedløsninger: intranett, intranett og ekstranett. Internett er det verdensomspennende offentlige nettet der det ikke er noen restriksjoner på tilgang til nettet. Intranett er en løsning som er tilgjengelig kun for en virksomhets medlemmer. Ekstranett er intranett løsninger som er delvis åpnet opp for andre enn virksomhetens medlemmer. Alle løsningene er hovedsakelig basert på TCP/IP og web teknologi. TCP/IP protokollene er en leverandør-uavhengig og åpen kommunikasjonsplattform. Den brukes for kommunikasjon mellom klient-tjener applikasjoner og som transportprotokoll for deling av informasjon på nettet. Web teknologi omfatter alle IT-løsninger som er basert på "menneskelig tenkning" (dvs. basert på assosiasjoner, også referert til som hypertekst) i et nettverk. De ulike løsningene er basert på klient-tjener teknologi der en skiller det som vises på skjermen (og lagret lokalt) og det som er lagret på datamaskin(er) i nettet. Det finnes en rekke web servere

rundt omkring i nettet og på klientsiden brukes browsere (f.eks. Explorer og Netscape) for å aksessere informasjonen som er gjort tilgjengelig. Informasjon er organisert i et sett med sider som er beskrevet ved hjelp av HTML (HyperText Markup Language). Sidene er bundet sammen ved hjelp av linker som angir addressene til sidene (URL – Universal Resource Locator). Kommunikasjon mellom browsere og web servere skjer ved hjelp av http (hypertext transaction protocol). Web-servere kan kalle program som etablerer koplinger mellom web og andre miljø. Disse programmene refereres til som gateways (skrevet i C, Java, etc.). Kommunikasjonsmekanismen mellom web-servere og gateways kalles CGI (Common Gateway Interface).

Typiske basistjenester tilgjengelig via web er elektronisk post, fjerninnlogging (telnet) og filoverføring (FTP). Diskusjonsgrupper er informasjonsutveksling mellom brukere innenfor emner. Informasjonssystemer (WIS) er systemer som støtter søking, henting og oppdatering av informasjon. De har en rekke anvendelser og krever oftest kopling til databaser. Kapittel 14 i [1] presenterer både design av web-sites og oversikt over aksesseringsmåter til en database fra web. Den enkleste måten å aksessere en database fra web er via CGI protokollen. Det finnes også server- og tjenerbaserte alternativer til dette.

5.6 CSCW / gruppevare

Gruppevare er datastøtte for samhandling mellom mennesker. I [13] defineres gruppevare som databaserte systemer som støtter grupper av mennesker som er engasjert i å løse en felles oppgave (eller å nå et felles mål), samt sørger for en felles grensesnitt til et delt miljø. I denne definisjonen er det verdt å merke seg kravene til felles oppgave og delt miljø.

Det er forøvrig ingen skarp skillelinje som gjør det mulig å klassifisere systemer som gruppevare eller ikke. I stedet snakker en ofte om et spektrum av gruppevare som i ulik grad støtter en felles oppgave og et delt miljø. Eksempelvis er et transaksjonsbasert IS lavt på et gruppevare-spektrum, mens en flerbruker editor scorer høyt. Tilsvarende er et email system lavt og et elektronisk klasseromsystem høyt på et gruppevarespektrum. To hovedtaksonomier for gruppevare defineres i [13]:

- Tid-rom taksonomi:
  - Samme plass - samme tid: ansikt til ansikt interaksjon (f.eks. møteromsteknologi)
  - Samme plass - forskjellig tid: asynkron interaksjon (f.eks.oppslagstavler)
  - Forskjellig plass - samme tid: synkron distribuert interaksjon (f.eks. desktop konferanser og real time editorer)
  - Forskjellig plass - forskjellig tid: f.eks. email

- Applikasjonsnivå taksonomi:
  - Meldingssystemer
  - Flerbruker editorer / samforfatterskapssystemer
  - Mediarom
  - Elektronisk møterom
  - Konferansesystemer
  - Intelligente agenter
  - Konversasjonssystemer
  - Arbeidsflyt

Prosesser som passer for bruk av arbeidsflytsystemer kan lett inndeles i deloppgaver, regler kan spesifiseres for overgangen mellom deloppgaver, deloppgavene bruker digitale informasjonsressurser og må kommuniseres til aktører og det er behov for prosesskontroll. Hovedkomponenter for et arbeidsflytsystem er arbeidsflytsmetodikk, implementasjonsmiljø, arbeidsflytmodellering og arbeidsflytmotor (engine). Tre hovedarkitekturer for arbeidsflyt-systemer er meldingsbasert, delt database og klient-tjener database.

To hovedtyper modeller (komplementære) for å beskrive arbeidsflyt beskrives i [17]: IPO (Input-Process-Output) og arbeidsflyt paradigmet. IPO tilsvarer den tradisjonelle måten å modellere arbeidsflyt der f.eks. DFD har blitt mye brukt. Ifølge [17] passer IPO best for å beskrive prosedyrer og rutiner og et annet språk behøves for å håndtere mer avansert arbeidsflyt. Arbeidsflyt paradigme er basert på arbeid av F. Flores og har røtter fra speech act teori. Arbeidsflyt paradigmet passer for håndtering av forhandlingssituasjoner som involverer generelle forretningsforhold, markeder, kunder, investorer, etc. En forretningsprosess deles inn i fire deler (konversasjon mellom to mennesker): åpning, avtale, ytelse og akseptanse.

5.7 Kvalitetssikring

Kvalitet defineres generelt som "fitness for purpose". Når en omtaler kvaliteten til et produkt kreves en beskrivelse av hensikten med produktet (f.eks. kravsspesifikasjon). Det finnes en rekke kvalitetsfaktorer for programvareutvikling og i [6] blir følgende definert: faktorer spesifisert i kravsspesifikasjon, kulturelle faktorer og faktorer som er relevant / viktig for utviklerne. Moderne kravspesifikasjoner vil ofte inneholde alle. Generelt vil et kvalitetssystem inneholder prosedyrer, standarder og retningslinjer. I [8] defineres ulike typer kvalitet: teknisk kvalitet, brukskvalitet, estetisk kvalitet, organisasjonskvalitet og symbolsk kvalitet.

Noen prinsipper for kvalitet blir skissert i [8]:

- Uavhengighet
- Vedlikeholdbarhet
- Sporbarhet
- Inkrementell
- Tidlig validering
- Viktighet av kravsspesifikasjon
- Dynamisk natur ved kvalitetssystem

ISO-9000 blir diskutert i [8]. Standarden vektlegger teknisk kvalitet og karakteriseres ved fase-orientert systemutvikling, dokument-drevet utvikling og faste kravsspesifikasjoner. Den observante leser vil huske at dette er sammenfallende med en del av de samme problemene som har blitt identifisert for vannfallsmodellen. ISO-standarden er dermed ingen garantist for kvalitet for systemutvikling.

5.8 Testing

I kapittel 22 i [5] beskriver testing. Sentralt i testing er verifikasjon og validering. Verifikasjon har som mål å sikre at produktet blir utviklet korrekt (sjekke at produktet er

riktig i forhold til spesifikasjonene). Validering har som mål å sikre at det rette produktet blir utviklet (sjekke at produktet møter kundens forventninger). Verifikasjon er å sjekke at en modell er internt konsistent, konsistent med andre tilgjengelige spesifikasjoner og i overensstemmelse med modelleringsspråkets syntaks og semantikk. Validering er å sjekke at en modell er i overensstemmelse med kundens ønsker og behov.

En skiller mellom statisk og dynamisk testing. Statisk testing omfatter analyse og sjekking av spesifikasjoner. Dynamisk testing er testing av implementasjoner av systemet (ulike test cases). Testdata brukes i statistisk testing der et programs ytelse blir testet og i defekt testing for å finne feil ved systemet. Testing deles inn i et sett av faser:

- Enhetstesting: individuelle komponenter blir testet for å sikre at de fungerer korrekt
- Modul testing: en samling av avhengige komponenter blir testet for å sikre at de fungerer korrekt
- Sub-system testing: samling av moduler som er integrert i sub-systemer
- System testing: sub-systemene integreres til et hele dvs. hele systemet
- Akseptanse testing: bruker testing før systemet skal settes i produksjon dvs. i brukermiljøet

Testing er sentralt i utviklingsprosessen og det er viktig at testplanen integreres i den overordnede prosjektplanen. Hovedkomponenter i en testplan er:

- Overordnet beskrivelse av testprosessen
- Sporbarhet av krav
- Testobjekter
- Testplan og ressurser
- Prosedyrer for dokumentasjon av test
- Programvare og maskinvare krav
- Begrensninger

Det finnes fire hovedstrategier for validering: top-down, bottom-up, thread testing, stress testing og back-to-back testing.

5.9 Utviklingsverktøy

Ulike typer utviklingsverktøy støtter ulike deler av utviklingen av programvare. Det finnes en rekke klassifiseringer av utviklingsverktøy. I kapittel 25 i [5] beskrives tre hovedtyper:

- CASE-verktøy. Computer-Aided Software Engineering omfatter verktøystøtte for systemutviklingsprosessen og ble utviklet fra tidlig på 80-tallet. Det er tre nivåer av CASE-teknologi: teknologi for støtte til produksjonprosesser, teknologi for prosesstyring og meta-CASE teknologi
- CASE workbenches
- Software engineering workbenches

Referanseliste

1. Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Richardo Torlone: *Database Systems: Concepts, Languages and Architectures,* McGraw-Hill, 1999, ISBN 0-07-709500-6:
2. Andersen, E.S.: *Systemutvikling*, NKI Forlaget, Bekkestua, 1989, kapittel 1-2 (Informasjonssystemer og systemutvikling – en begrepsklargjøring og Husbygging og systemutvikling)
3. Andersen, N.E. m.fl.: Professionel S*ystemudvikling*, Teknisk Forlag, København 1986, kapittel 2-3 (Situation og handling og Hvad er systemudvikling?)
4. Skagestein: *Dataorientert systemutvikling*, Universitetsforlaget, Oslo, 1996, kapittel 3-4
5. Sommerville: *Software Engineering*, Addison-Wesley, Reading, Mass., 1989, kapittel 22-23, 25-27
6. Ince: *An Introduction to Software Quality Assurance*, McGraw-Hill, London, 1994, kapittel 1-3
7. Boehm: *A Spiral Model of Software Development and Enhancement*, Computer, vol. 21 s. 61-72, mai 1988
8. Braa m.fl.: *Critical View of the ISO Standard for Quality Assurance*, Information Systems Journal, vol 5, s. 253-269, 1994
9. Kling m.fl.: *The Control of Information Systems After Implementation*, Communications of the ACM, vol. 27, no. 12, s. 1218-1226, 1984
10. Nicol m.fl.: *Object Orientation in Heterogeneous Distributed Computing Systems*, IEEE Computer, vol. 26, no. 6, pp. 57-67, June1993
11. Keen P.G.W: *Information Systems and Organizational Change*, Communications of the ACM, vol. 28, no. 1, s. 24-33, 1981
12. J. Grudin: *Computer-Supported Cooperative Work: History and Focus*, IEEE, May 1994
13. C.A. Ellis, S.J. Gibbs and G.L. Rein: *Groupware – Some Issues and Experiences*, Communications of ACM, Vol. 34, No. 1, Jan. 1991
14. L.J. Bannon and K. Schmidt: *CSCW: Four Characteristics in Search of a Context*, Studies in Computer Supported Cooperative Work, J.M. Bowers and S.D. Benford (editors), Elsevier Science Publishers B.V. (North-Holland), 1991
15. R.T. Marshak: *Workflow White Paper – An Overview of Workflow Software*, In Proceedings Workflow '94, San Jose, The Conference Group, Scottsdale, AZ, 1994
16. K.R. Abbott and S.K. Sarin: *Experiences with Workflow Management: Issues for the Next Generation*, In CSCW-94, Proceedings of the Conference on Computer Supported Cooperative Work, Chapel Hill, NC, ACM Press (side 113-120), October 1994
17. K. Center and S. Henry: *A new Paradigm for Business Processes*, IBM Report, 1992