

## 6. Designprincipper og arkitektur

Hvad er Design?  
Eksempel på Design?  
Designprincipper  
Trinvis forfining  
Løs kobling  
Stor tæthed  
Arkitekturdesign: 5 stilarter  
Lagdeling som arkitektur

## Efter denne lektion (og projektet) skal du:

- Kunne opstille og dokumentere det overordnede design (arkitekturdesign) af et mindre IT-system

## Hvad er Design?

- Design i systemudvikling handler om at beskrive noget nyt, som ikke eksisterer i "nuet".
- Ofte vil det være givet *hvad* et nyt edb-system skal kunne
- Som regel få grænser for *hvordan*, så design kan sagtens være en meget kreativ proces.

## I Design indgår at:

- Studere det foreliggende problem fra flere forskellige synsvinkler
- Identificere mindst én og helst flere mulige løsninger. De fundne løsninger vurderes hver for sig f.eks. i forhold til nogle kriterier, f.eks. brugerens behov, enkelthed (en simpel løsning er bedre end en kompliceret), mulighed for genbrug osv.
- Beskrive de begreber og elementer der indgår i den valgte løsning.

## Eksempel på Designproces (1)

- En del af designet af et edb-system er en menu på skærmen hvor brugeren skal vælge et punkt
- (1) En mulig løsning er at lade brugeren bruge pile-tasterne til at bevæge sig op og ned i menuen.
- (2) En anden mulig løsning er, at lade brugeren udpege det ønskede menupunkt ved at pege med musen.
- (3) En tredje løsning er en kombination af begge dele.
- Fordele og ulemper diskuteres og der træffes et valg: Løsning 2 - Musen bruges til udpegning.

## Eksempel på Designproces (2)

- Når løsning 2 er valgt forestår et nyt designproblem: Hvordan skal musen bruges til udpegning?
- (1) Skal man blot pege
- (2) Skal man pege og klikke med venstre eller højre museknap
- (3) Eller skal man dobbeltklikke
- Som før opstilles en række løsninger, nogle kriterier for valg, og der vælges en løsning
- Processen hvor man langsomt men sikkert detaljerer og raffinerer sit design kaldes for *trinvis forfining*

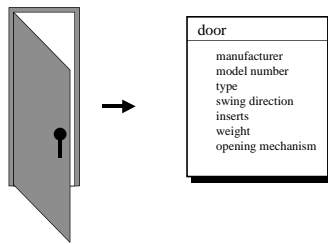
## Op til 4 slags Design

- *Arkitektur-design*: Opdeling af det samlede system i del-systemer. Relationen mellem del-systemerne beskrives
- *Del-system design*: Præcis specifikation af hvert del-system
- *Grænseflade-design*: Nøjagtig og præcis beskrivelse af grænsefladen til andre systemer. Hvornår udveksler man data? Hvilke signaler kan sendes fra del-system A til del-system B? Hvilket format skal inddata have?
- *Komponent-design*: Design af det indvendige i hver komponent. Den service og funktionalitet som komponenten skal levere designes

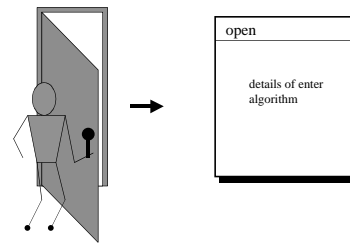
## Designprincipper

- Abstraktion
- Trinvis forfining
- Opdeling
  - løst koblet
  - stor tæthed

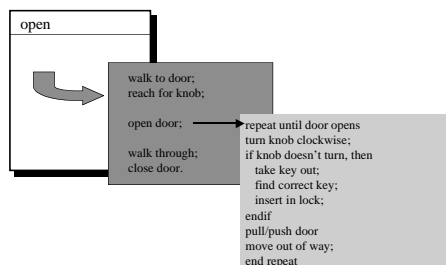
## Data-abstraktion



## Procedure-abstraktion



## Trinvis forfining af Design

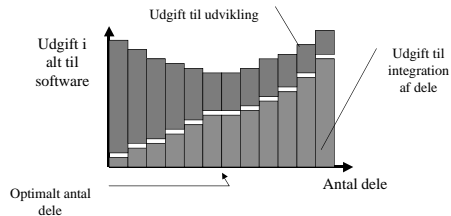


## Hvorfor opdeling?

- Software bliver nemmere at teste
- Software bliver nemmere at vedligeholde
- Software bliver nemmere at udvide
- En fejl får sværere ved at sprede sig til hele systemet (færre følgeføj "side effects")

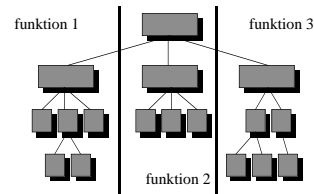
## Opdeling: En balance

Hvad er det "rigtige" antal dele?



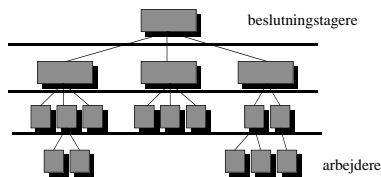
## Horisontal opdeling

- opdeling efter grupper af funktionalitet
- kontrolmoduler til at styre arbejdsdeling mellem funktioner



## Vertikal opdeling

- adskil beslutningstagen og arbejde
- beslutningstagningsmoduler i toppen af arkitektur



## Løst koblet

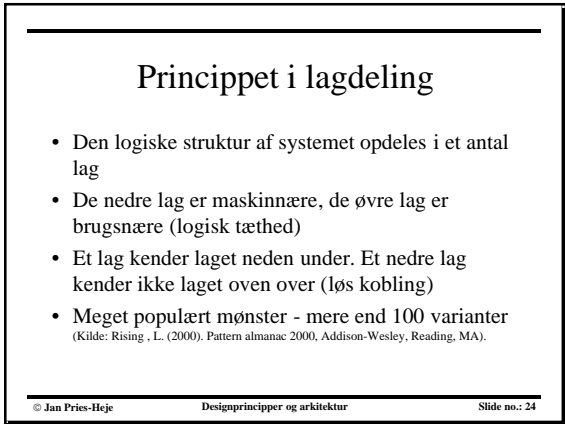
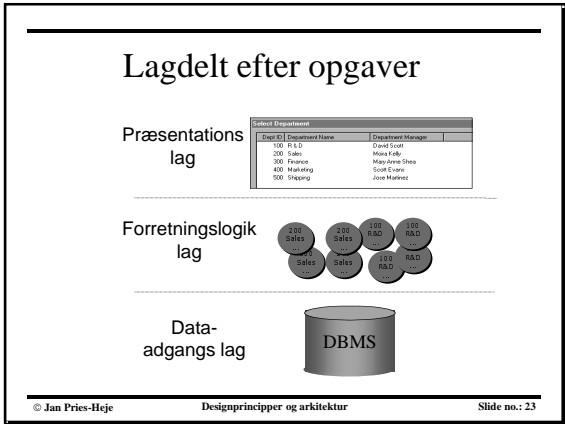
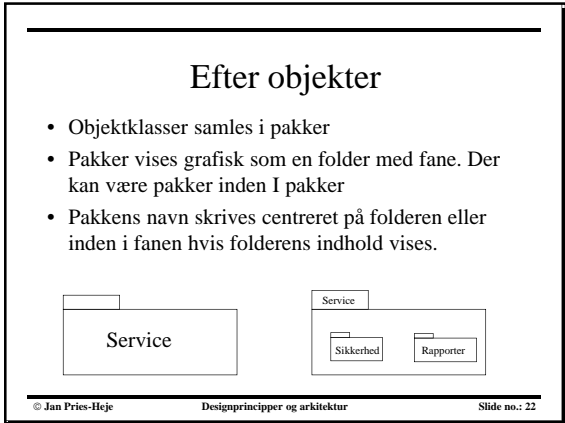
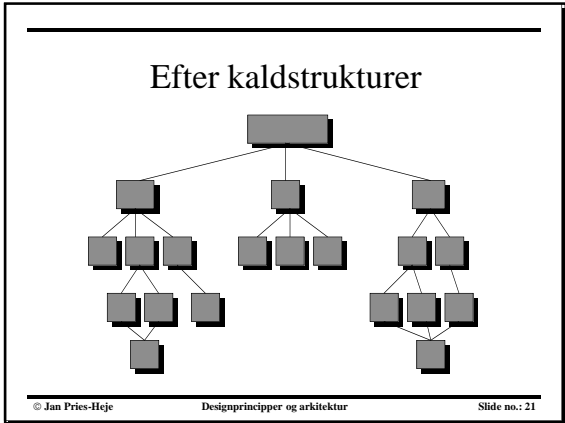
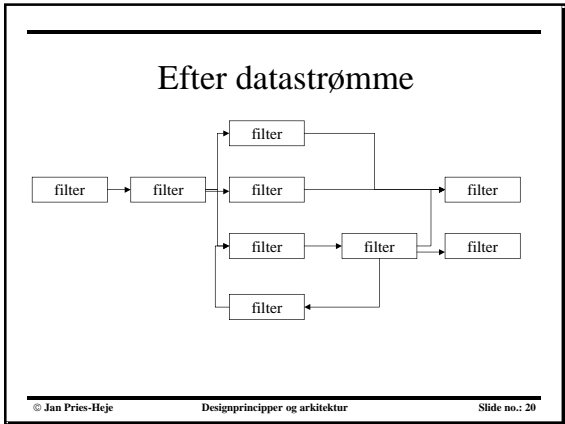
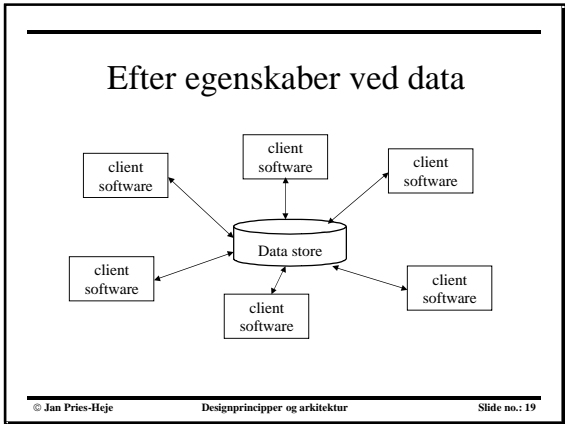
- At et design er *løst koblet* handler om at minimalisere grænseflader - så få som muligt
- Hvis samme data bruges af f.eks. fire del-systemer, så er det meget svært at lave en lille ændring uden at skulle lave alting om
- Hvis derimod data til de fire del-systemer er opdelt så hvert del-system har sin egen pulje af data, og kun de data som skal deles er fælles via grænsefladerne, så bliver det meget nemmere at ændre i et del-system uden at lave alle de andre del-systemer om, og det bliver meget nemmere at finde fejl

## Stor tæthed

- At tæt relaterede funktioner er lagt tæt ved hinanden
- (1) Tidsmæssig tæthed er når funktioner som udføres på samme tid er lagt sammen, f.eks. al initialisering
- (2) Logisk tæthed opnås ved at samle alle funktioner der logisk udfører samme funktion f.eks. alle skærbilleder der indgår i brugergrænsefladen
- (3) Procedural tæthed er når alle elementer der proceduremæssigt er ens bliver samlet, f.eks. alle søgeprogrammer.
- (4) Kommunikations-tæthed opstår ved samling af alle elementer, der arbejder på de samme inddata eller uddata

## Arkitekturdesign: 5 stilarter

- Efter egenskaber ved data
- Efter datastrømme
- Efter kaldstrukturer
- Efter objekter
- Lagdelt efter opgaver



## Eksempel på opdeling i 6 lag

- **Præsentationslag:** Vinduer, rapporter, HTML, XML, Javascript ...
- **Applikationslag:** Modtager forespørgsler fra præsentationslag, specifikke arbejdsgange, konsolidering/ændring af data til præsentation
- **Domænelag:** Modtager forespørgsler fra applikationslag, domænergler, service på tværs af applikationer
- **Infrastrukturlag:** Modtager forespørgsler fra domænelag, generel lav-niveau service på tværs af domæner
- **Teknisk servicelag:** Modtager forespørgsler fra infrastrukturlag, persistens, sikkerhed
- **Fundament:** Modtager forespørgsler fra det tekniske servicelag, datastrukturer, træde, databaser, netværks I/O

## Eksempel fra hotel

- **Præsentationslag:** Skærbilledet som hotel-receptionisten ser
- **Applikationslag:** Den arbejdsgang der følges ved indcheckning af gæst på hotel
- **Domænelag:** Regler om hvordan "guld" gæster skal behandles I receptionen, I restauranten, på nettet når vedkommende bestiller
- **Infrastrukturlag:** Konvertering af valuta
- **Teknisk servicelag:** Sikkerhed omkring modtagelse af betaling med kreditkort. Persistent opbevaring af kunde og betalingsdata
- **Fundament:** Databasen hvor kundedata ligger og "sover" mellem brug, netværket som bruges til at hente og registrere betalingsdata

## To lags arkitektur

- Den simpleste lagdeling er selvfølgelig to lag
- I det øverste lag findes forretningslogikken sammen med vinduerne. Der læses og skrives direkte til en database
- Øverste lag f.eks. programmeret i Visual Basic eller Powerbuilder
- I det nederste lag finder vi databasen
- Kan være et godt valg til enkeltstående databasetunge systemer med hovedsagligt simple transaktioner (af typen CRUD - Create, Read, Update, Delete)

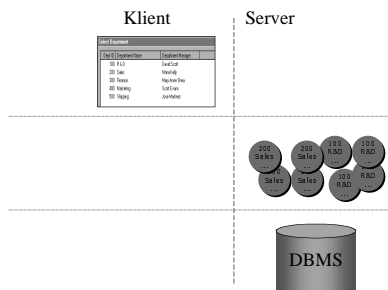
## Tre lags arkitektur

- Den hyppigst anvendte lagdeling
- Øverst grænseflade med vinduer, rapporter etc.
- I midten forretningslogik incl. forretningsgange, regler og service på tværs så som valutaomregning
- Nederst lagring af data (database)

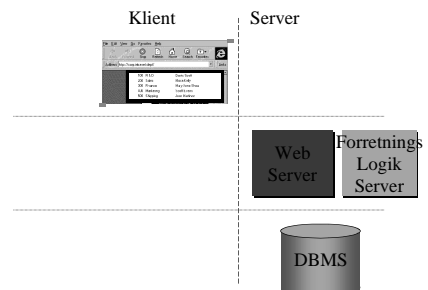
EmpID	Department Name	Employee Name
100	IT	David Scott
200	Sales	Michael Kelly
300	Finance	Maya Aver Shee
400	Marketing	Scott Crane
500	Shipping	Jose Martinez



## Eksempel på en tynd klient



## Eksempel: Web-server



# Eksempel: Transaktion-Server

Klient      Server

Data base	
Oplysninger	Opdateringer
01 123	01 123
02 456	02 456
03 789	03 789
04 101	04 101
05 202	05 202

Transaktions-Server  
Forretningskomponenter      Data-adgangs komponenter

