Forelesning 15 : Programvarearkitektur

- Hva er «Software Architecture» og Dekomponering ?
- Idealene «Høy Modulstyrke» og «Lave Modulkoblinger»
- Noen Grunnleggende Arkitekturmønstre :
  1. Model-View Controller
  2. Lagdeling (Layer)
  3. Repository-basert arkitektur
  4. Klient-Tjener arkitektur
  5. Pipe and Filter
- Gartner sin Tykk/Tynn-klientskisse (miks av lagdeling og K/T)
- Sentraliserte vs Hendelsesbaserte Kontrollmodeller
- Arkitektur i RUP - gjennomgang av SAD-artefaktet

Pensum : Sommerville – kap. 6 + RUP-SAD artefaktet

A general model of the design process

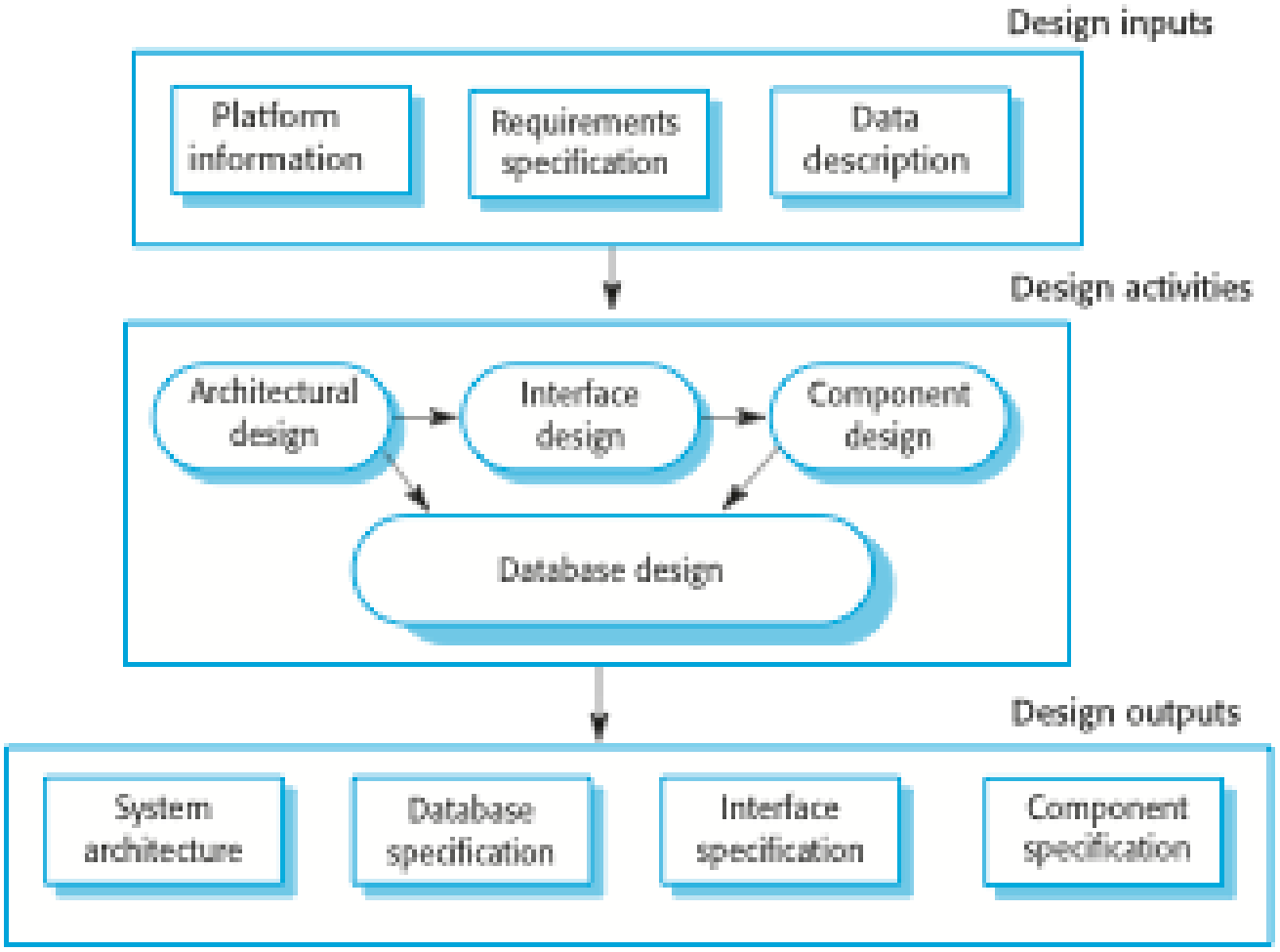


Fig. 2.5 , Sommerville

Dekomponering i subsystemer og moduler

- Ved å dekomponere oppnår man å
  - reduserer kompleksiteten
  - bevare oversikten over totalitet og detaljer
  - gir mulighet for parallelt utviklingsarbeid
  - bedre endrings- og gjenbruksmulighetene

- Dekomponering er et generelt prinsipp som benyttes innen alle former for Programvaredesign.

- En sentral avgjørelse innledningsvis i arkitekturarbeidet er å bestemme seg for hvilke prinsipper man skal basere inndelingen av programvaren på. Dagens fokus er å se på et sett alternative, men samtidig komplementære mønstre for dette.

Modulstyrke (Cohesion)

Modulstyrke er et mål på de interne forholdene i en systemmodul. Det er et ideal å komme frem til sterke moduler innen design

En modul som er sammensatt av enkle, tett sammenknyttede og veldefinerte oppgaver/funksjoner er ”sterk”.

Fordeler og ulemper med å tilstrebe høy modulstyrke :

- Høy stryke gir god vedlikeholdbarhet
- Overdreven dekomponering for å oppnå sterke moduler vil gi et kaos av mange små moduler

Modulkobling (Coupling)

Modulkobling er et mål på sammenkoblingene mellom modulene vi har satt opp i designet. Det er et ideal å komme frem til moduler med Lav kobling

- Høy modulkobling vanskeliggjør vedlikehold.
- Ved lav kobling er endringer og feilsøking enklere, da datautvekslingen er minimal
- Ved å holde datastrukturene internt i den enkelte modul oppnår man en lavere koblingsgrad (innkapsling innen OO).
- Globale data er et skrekk-eksempel på Høy kobling - men mange benytter seg av det likevel

Arkitekturmønster 1 : Model-View-Controller

(fig. 6.2 i Sommerville)

Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model. See Figure 6.3.
Example	Figure 6.4 shows the architecture of a web-based application system organized using the MVC pattern.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.

Model-View-Controller

(fig. 6.3 i Sommerville)

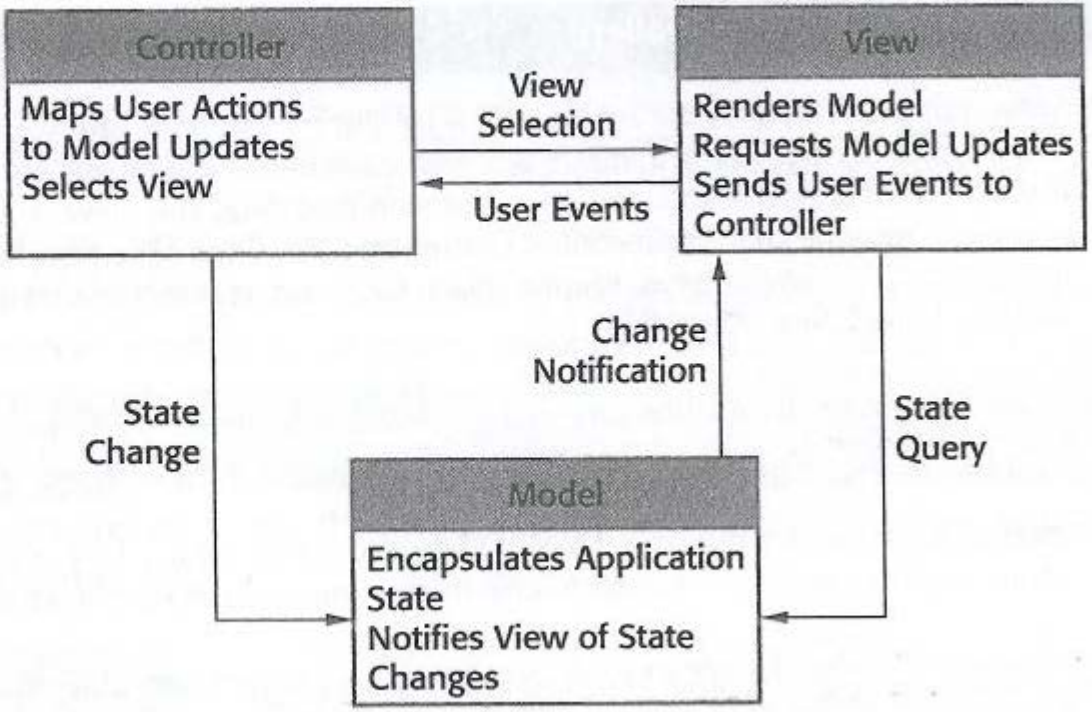
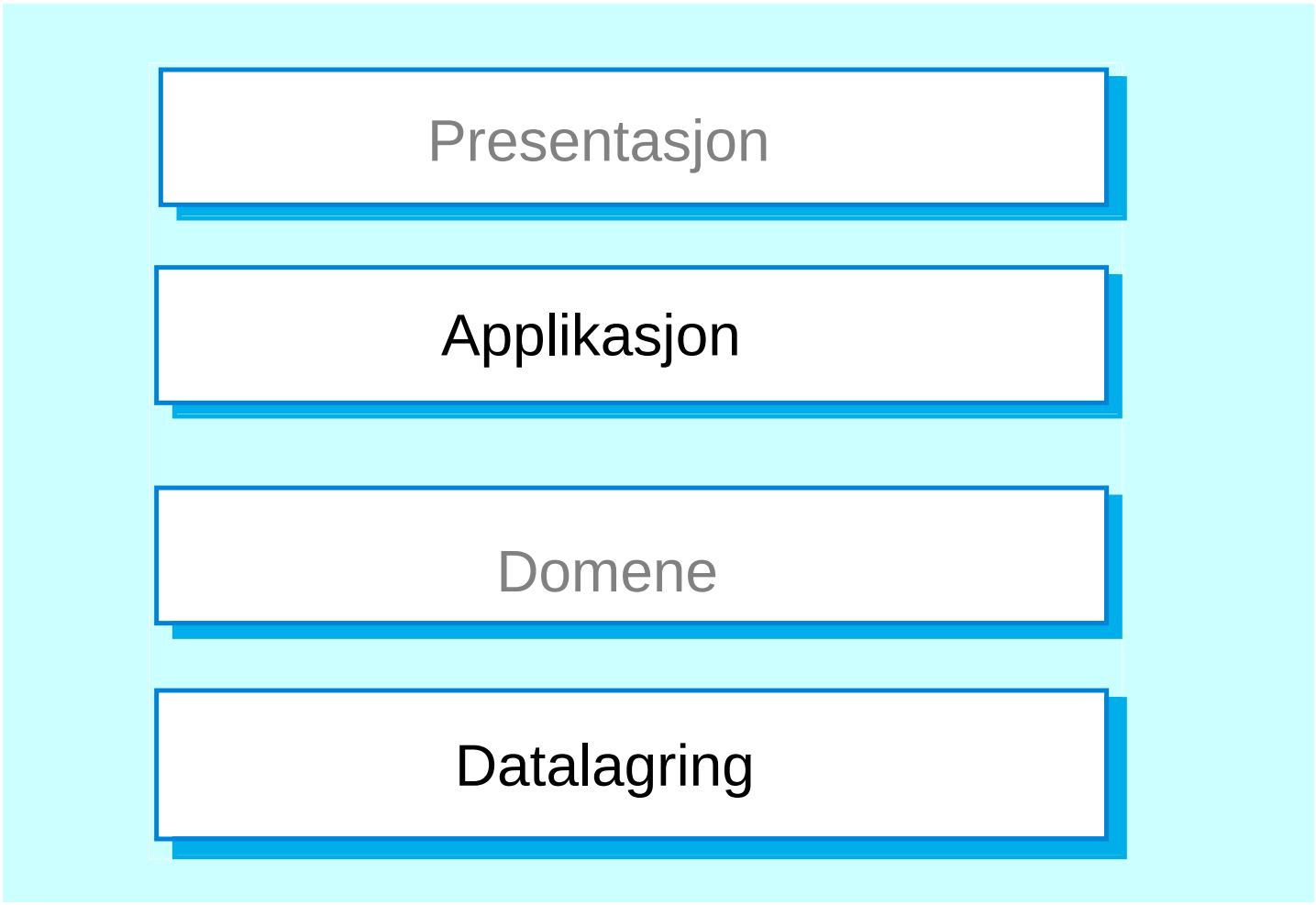
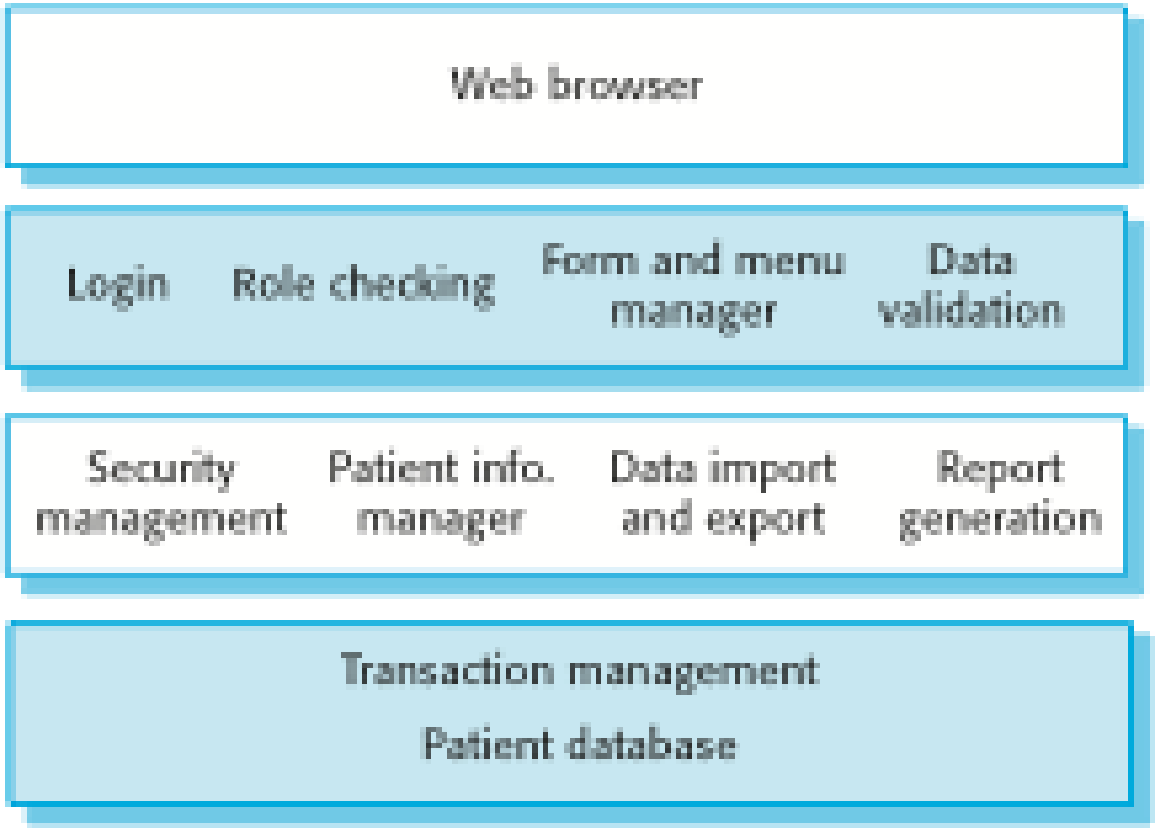


Figure 6.3 The organization of the MVC

2 : Layer model (n-lag)



The architecture of the MHC-PMS



3 : Repository model

(fig. 6.9 i Sommerville)

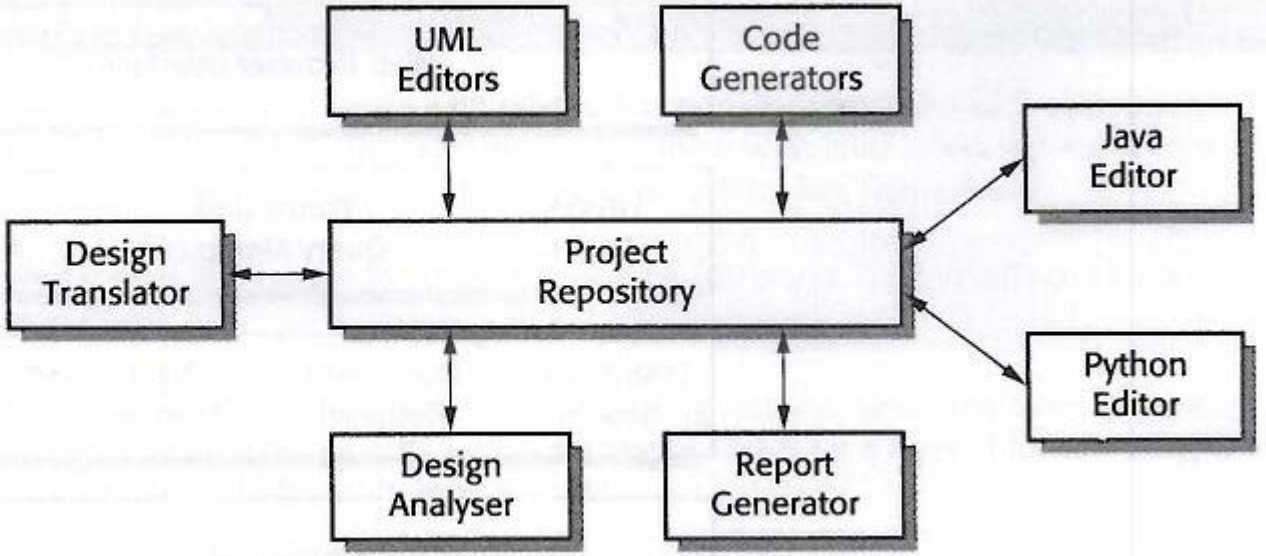
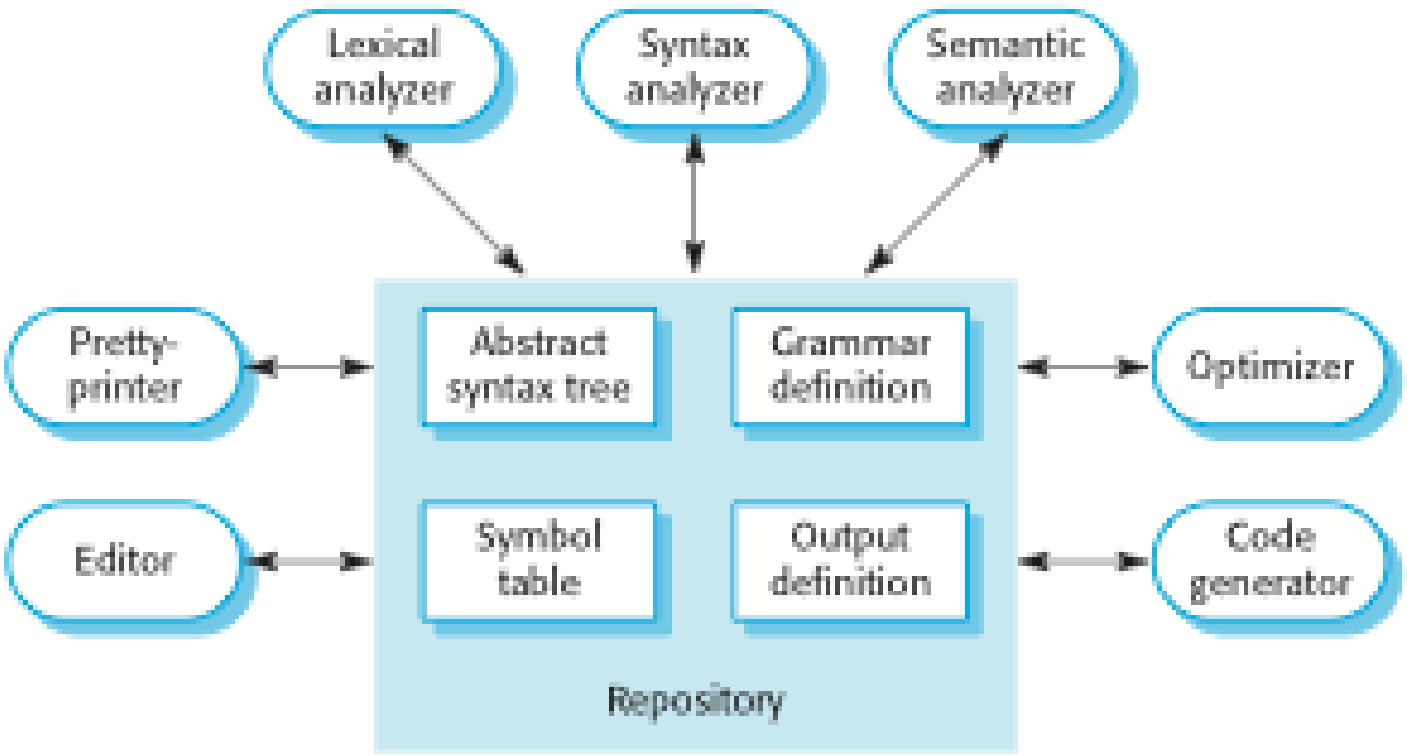


Figure 6.9 A repository architecture for an IDE

A repository architecture for a language processing system



4 : Client-Server model

(fig. 6.11 i Sommerville)

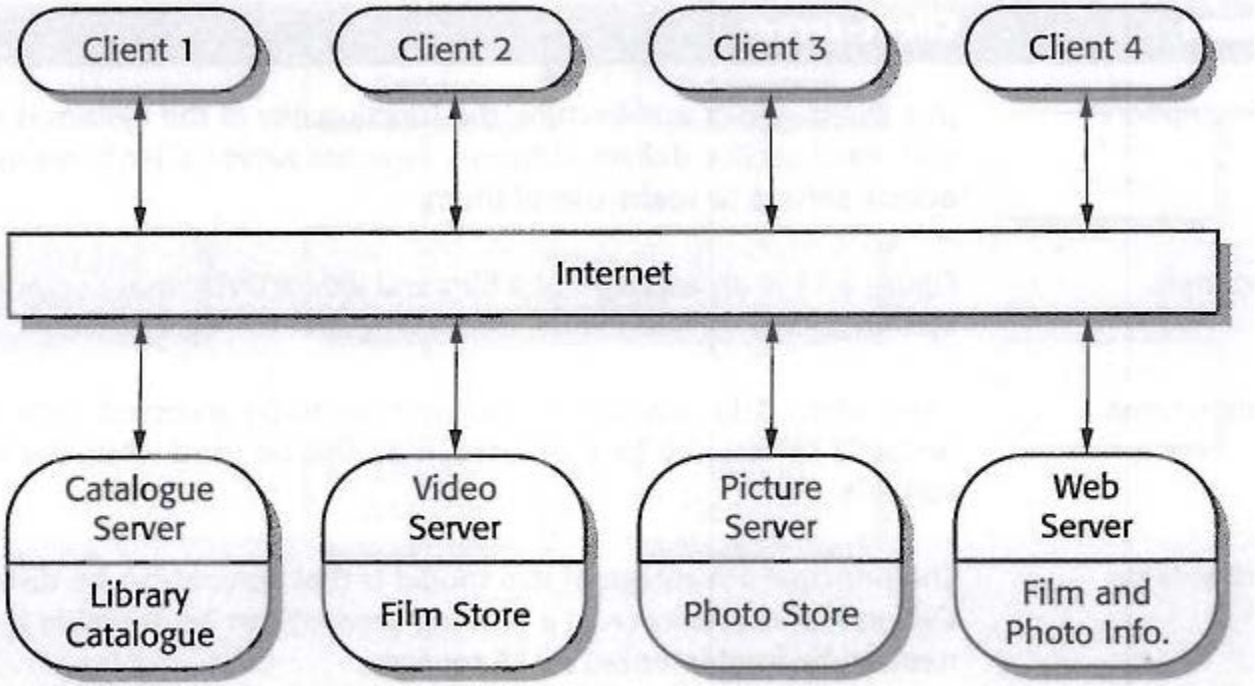
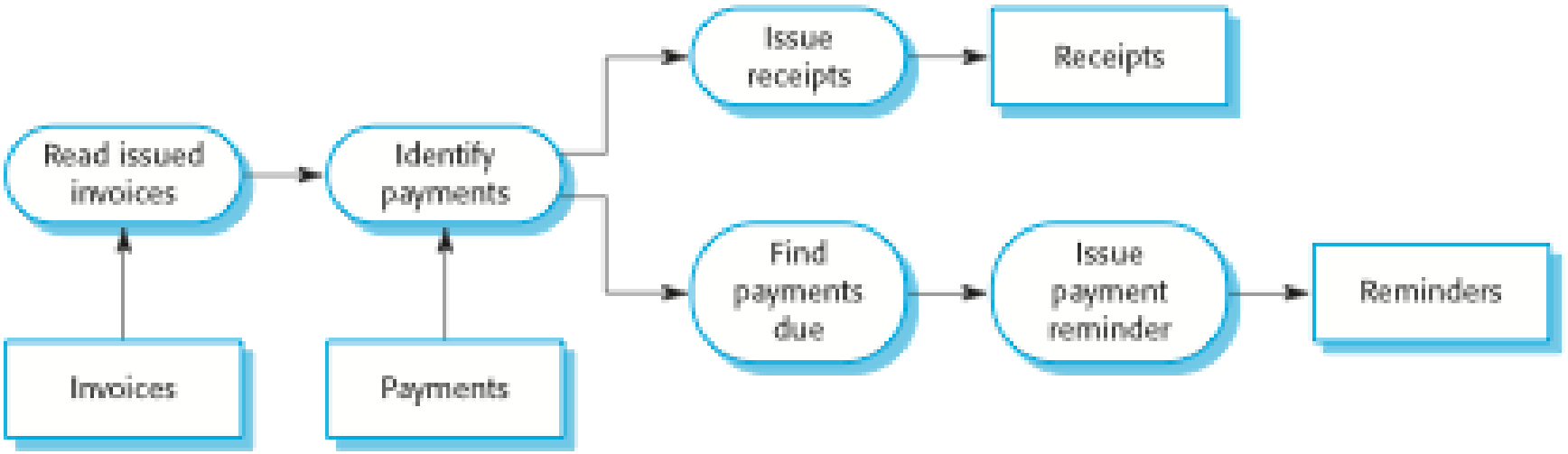


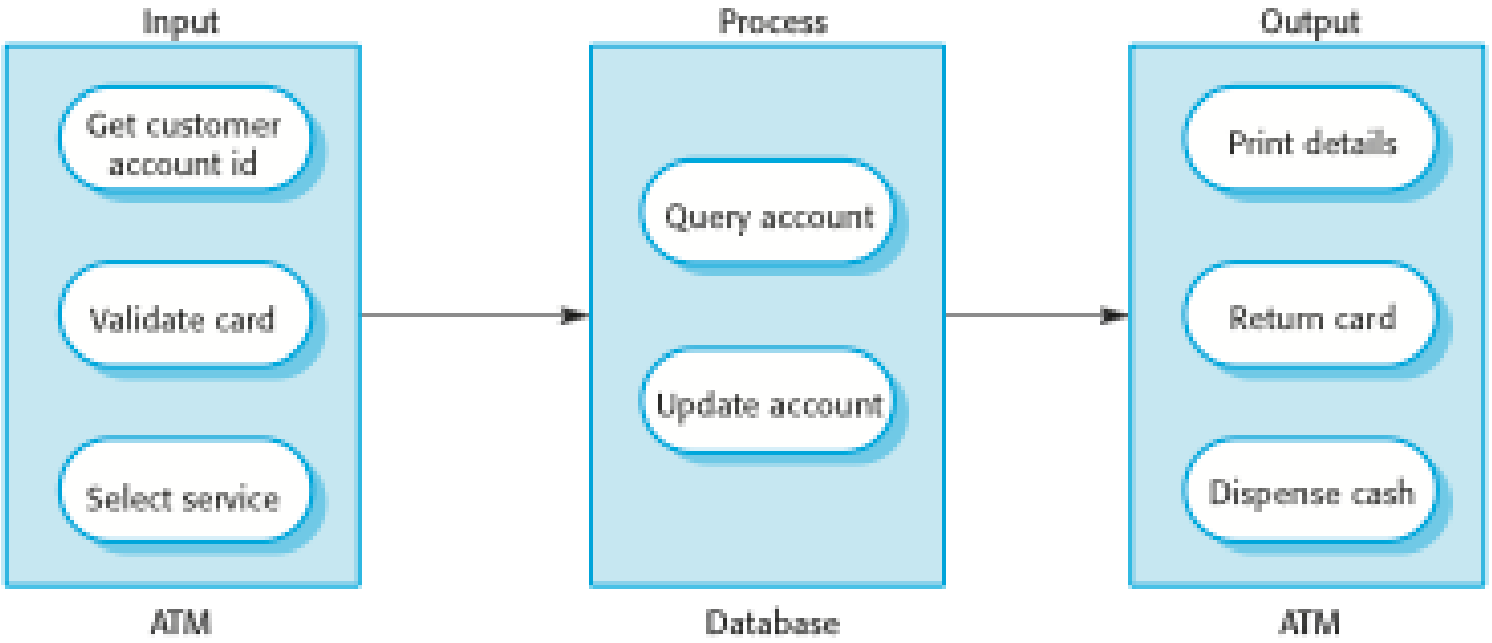
Figure 6.11 A client–server architecture for a film library

5 : Pipe and Filter

(fig. 6.13 i Sommerville)



The software architecture of an ATM system



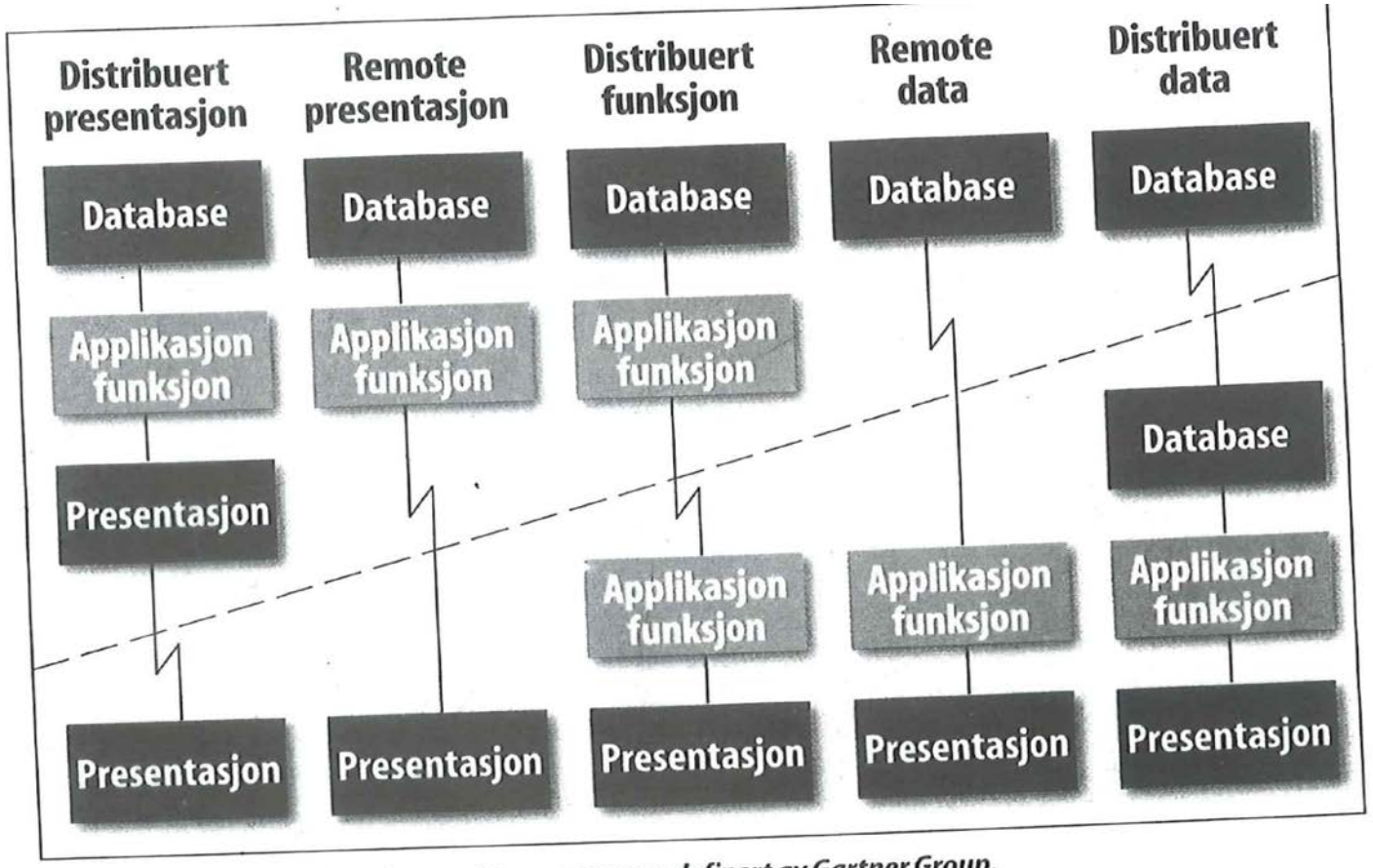
Gartner Group - nivåer i K/T

Definerer 5 ulike nivåer å basere sin Klient / Tjener inndeling på :

1. Distribuert presentasjon :
    - Klienten besørger deler av brukergrensesnittet, resten ligger sentralt
2. Remote presentasjon :
    - Klienten besørger brukergrensesnitt og kall til sentralt system genereres. Resten ligger sentralt
3. Distribuert funksjon :
    - Deler av applikasjonen ligger ute på klienten, resten ligger sentralt
4. Remote data :
    - Kun dataene ligger på tjeneren. All prosessering skjer lokalt
5. Distribuerte data :
    - Dataene ligger delvis på tjeneren og delvis på klienten

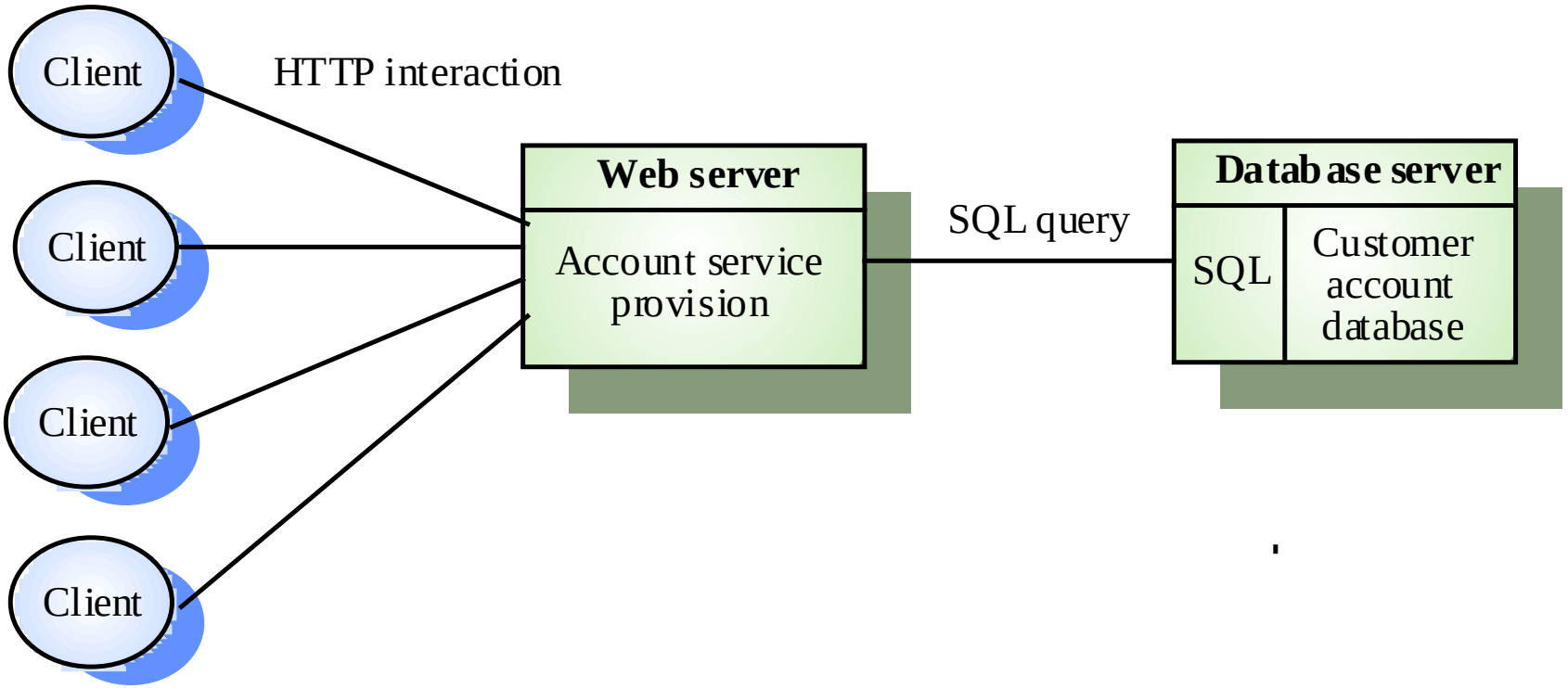
Gartner Group nivåene:

Merk at Klienten her er nederst og Server øverst



Figur 1 viser de fem klient/server-klassene som er definert av Gartner Group.

3-nivå klient-tjener arkitektur

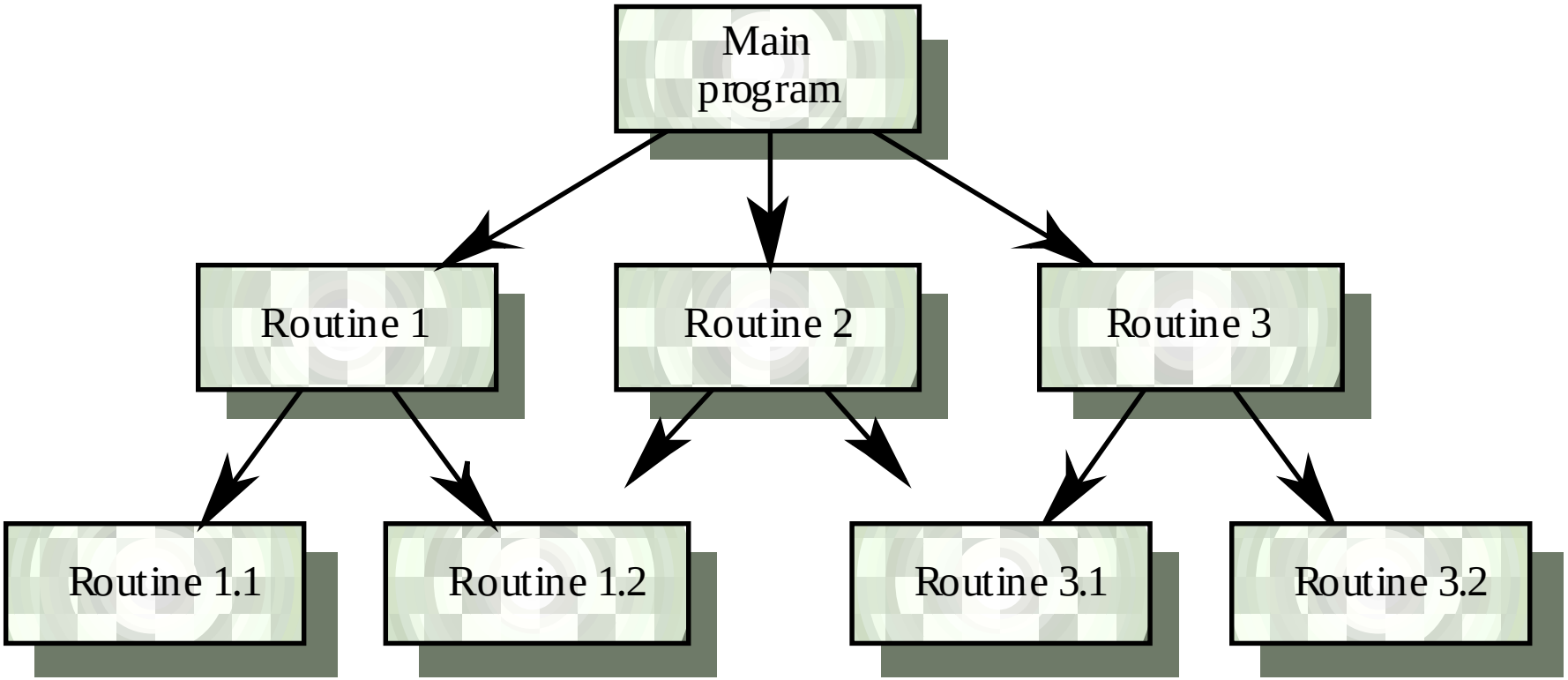


Kontroll modeller

- Etter å ha definert de overordnede trekk i systemstrukturen, bør det foretas bevisste valg for kontrollmekanismene for at riktig tjeneste leveres til riktig tid i systemet.
  - Sentralisert kontroll
    - call-return model
    - manager model
  - Hendelsesbasert kontroll
    - broadcast model
    - interrupt-driven models

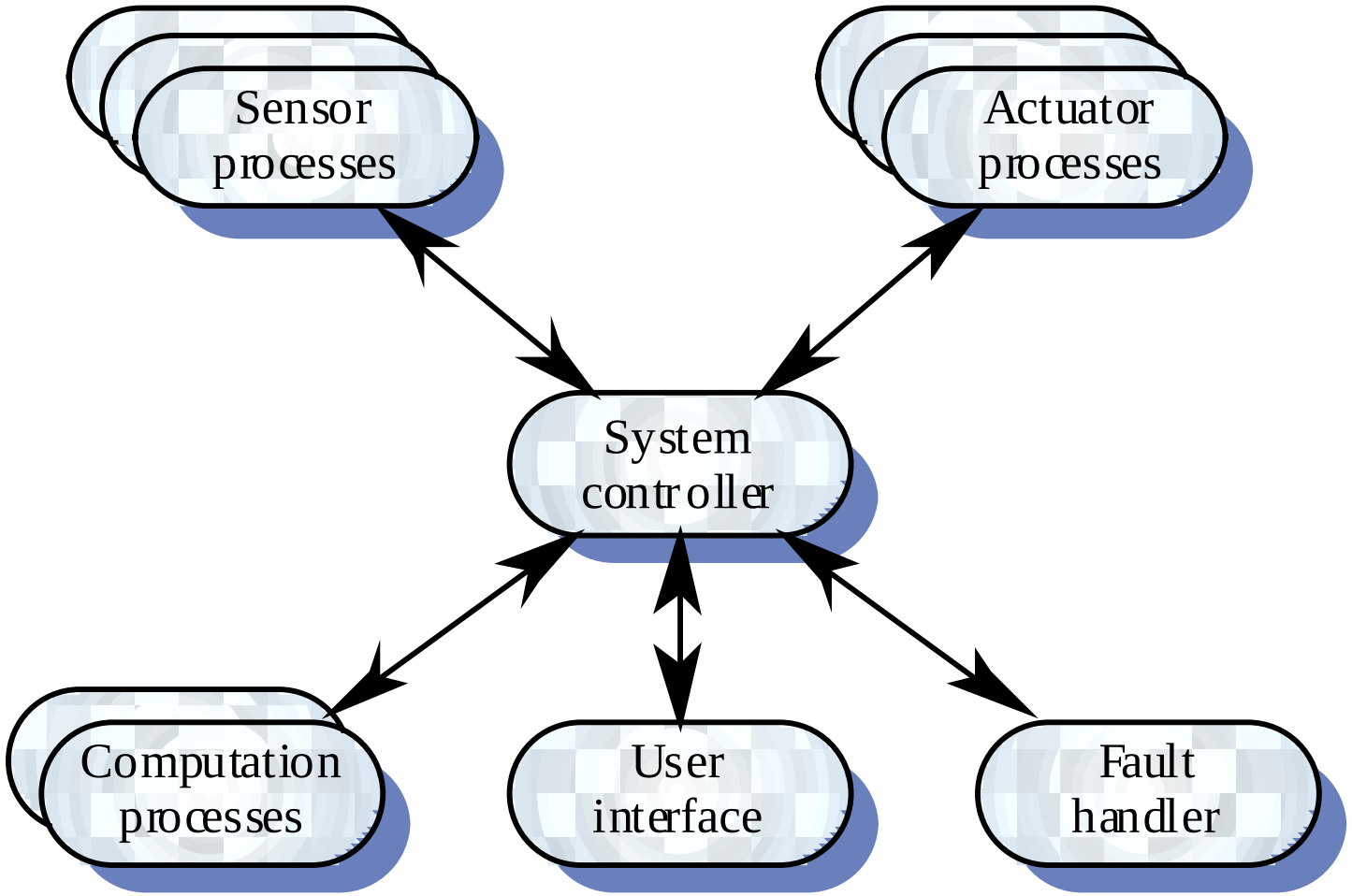
Sentralisert : call-return

(tidl. Sommerville)



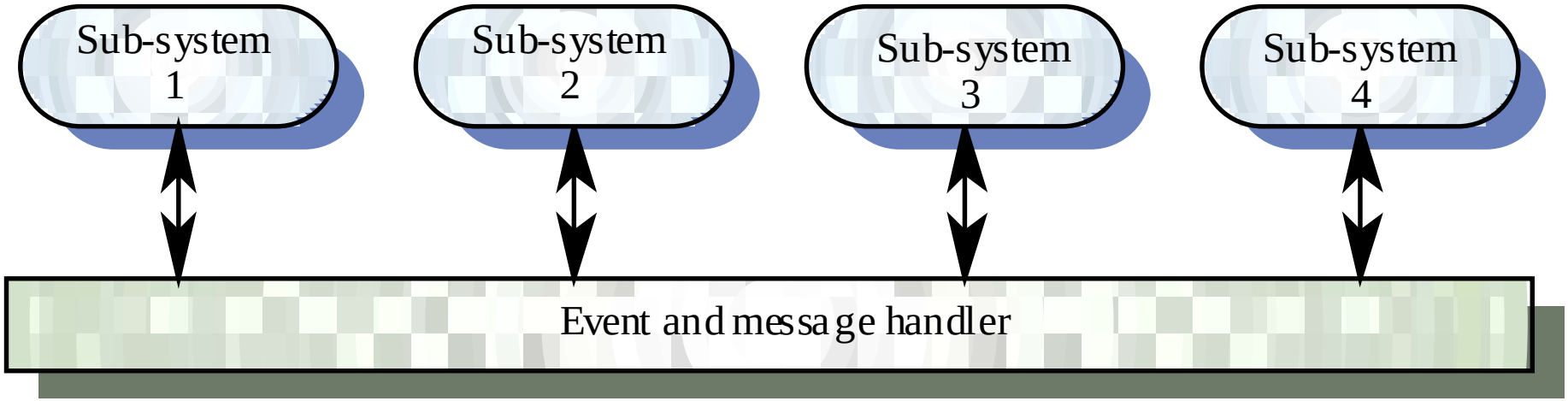
Sentralisert : manager

(tidl. Sommerville)

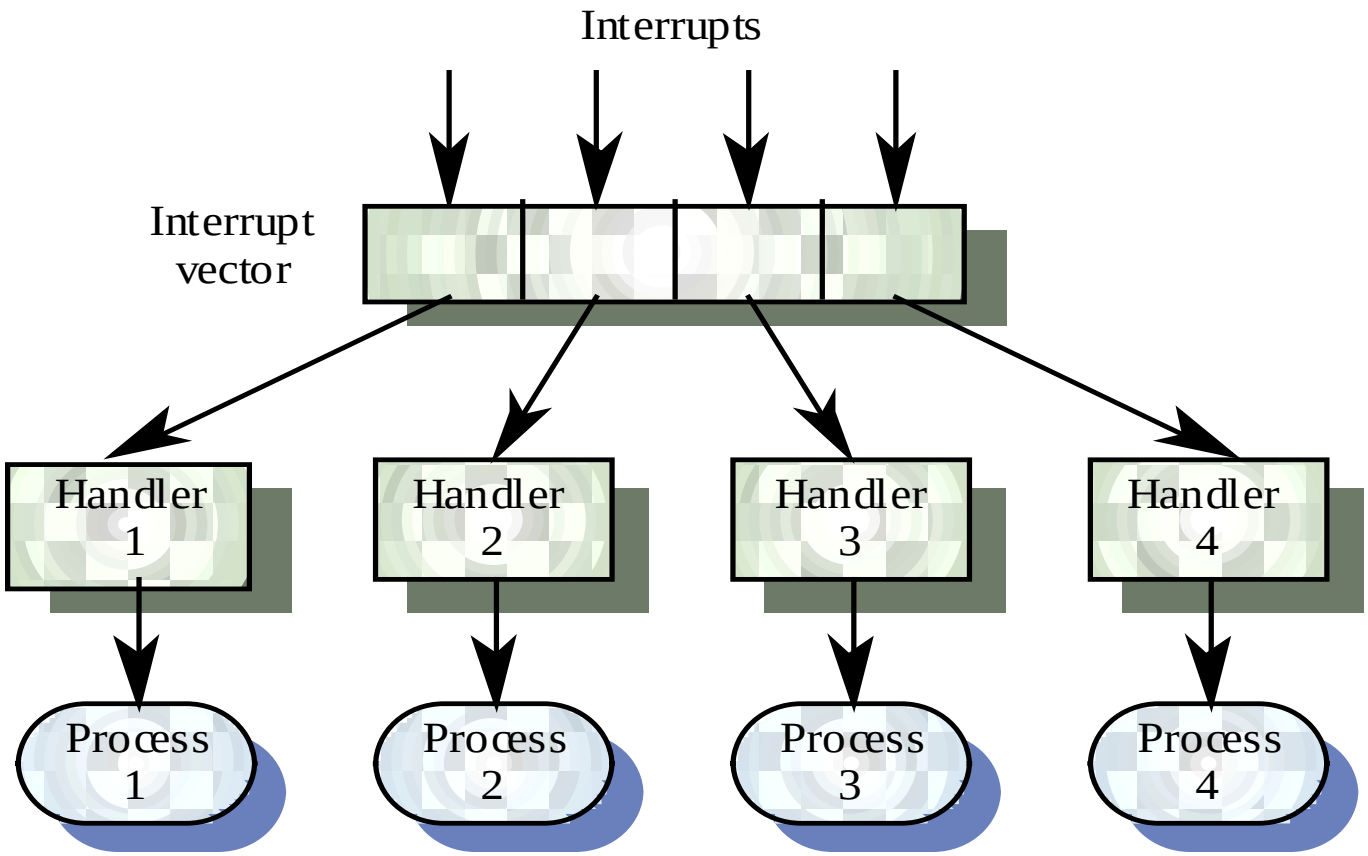


Hendelses basert: Broadcast

(tidl. Sommerville)



Hendelses basert : Interrupt-driven (tidl. Sommerville)



Rational Unified Process

(figur hentet fra bok av P. Kruchten)

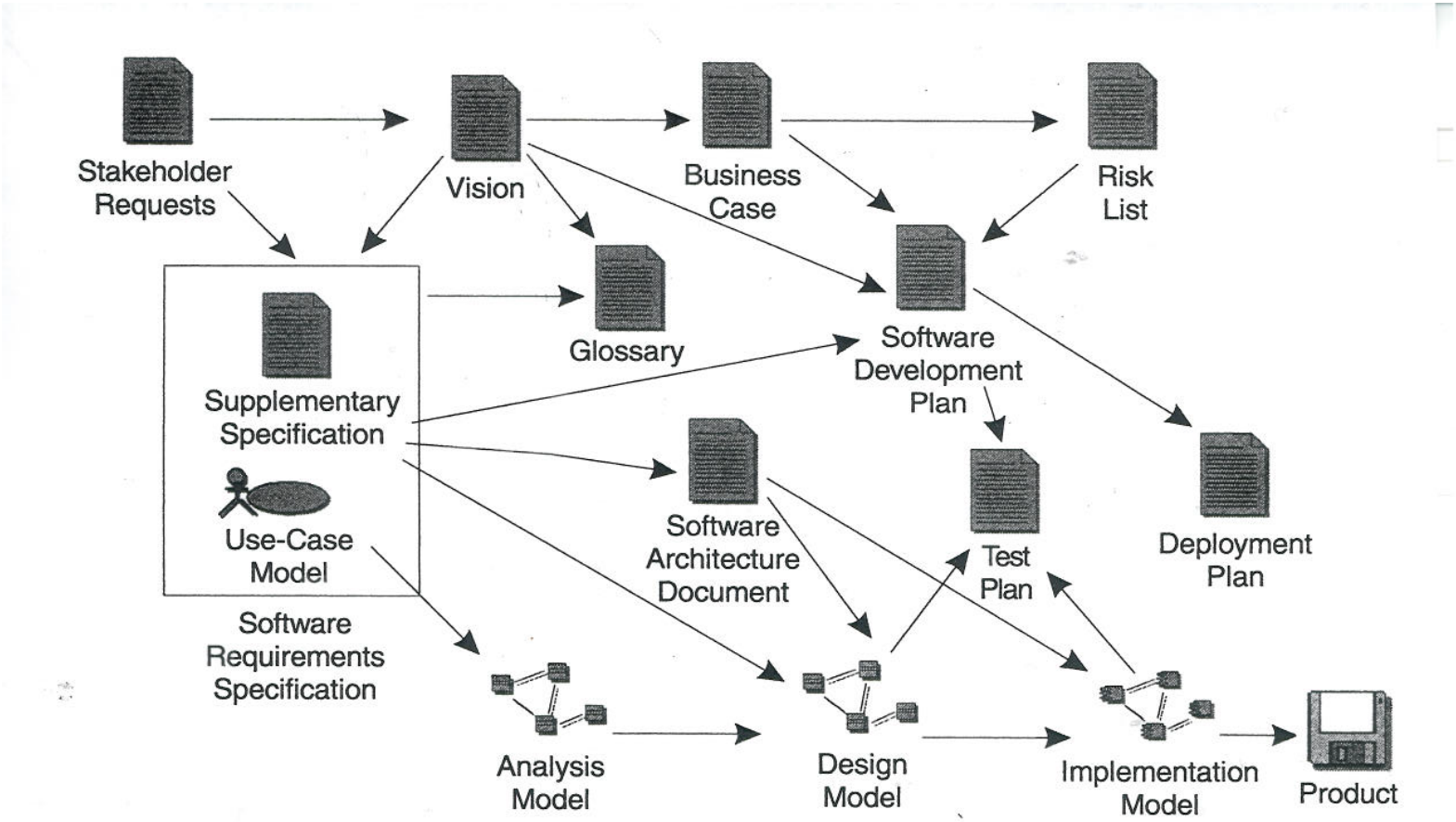
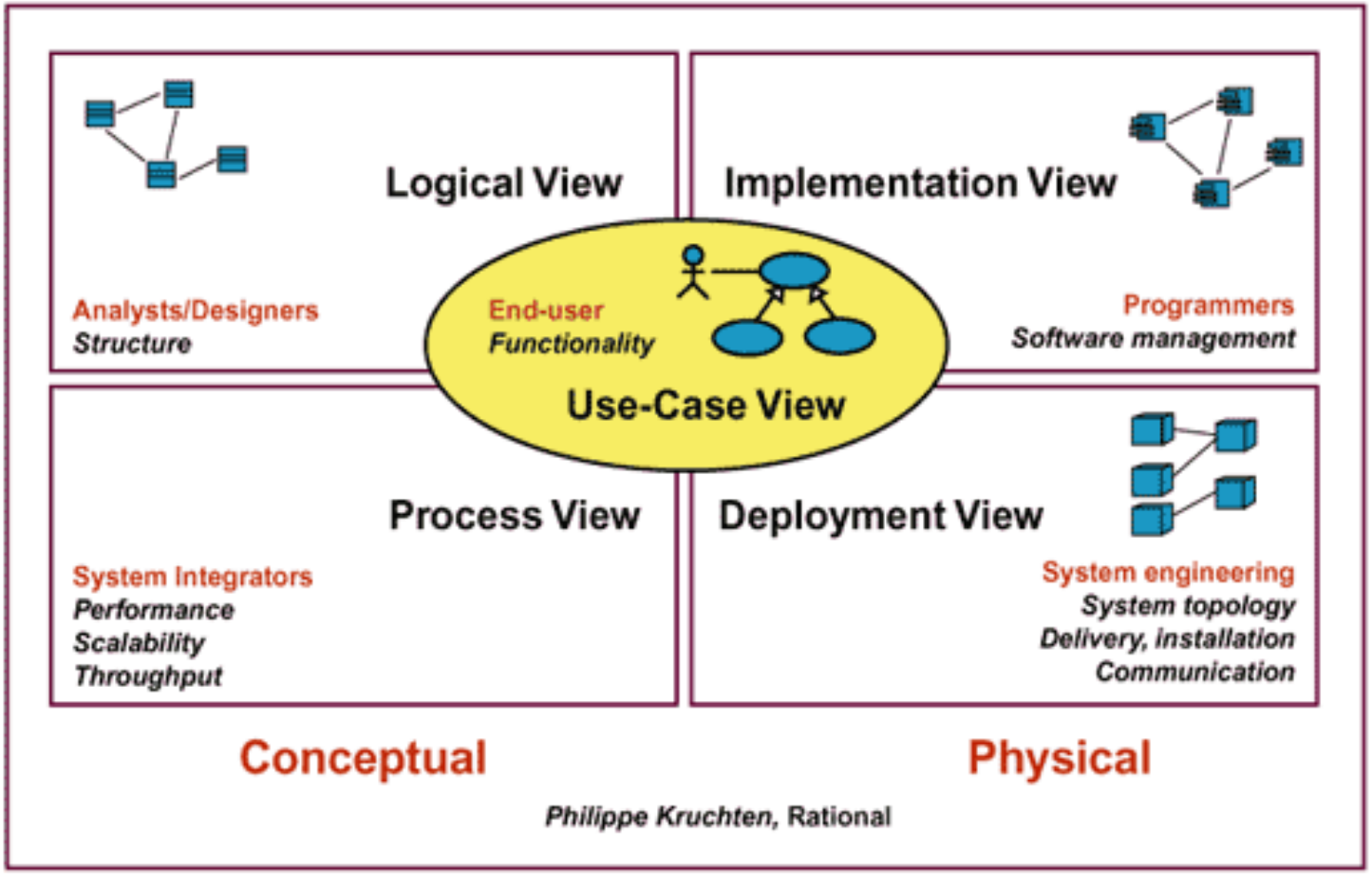


FIGURE 3-3 *Major artifacts of the Rational Unified Process*

RUP Arkitektur : 4 + 1 View



RUP-arkitektur

- Rollen «Software Architect» er totalansvarlig for arkitektur. En hovedrolle og viktigste sparringpartner til «Project Manager»
- Artefaktet «Software Architeture Document» dokumenterer arkitekturen man ved utgangen av Elaborationfasen er blitt enige om (milepælen LCA – LifeCycle Architecture)
- Har en klar «Big Up Front»-tendens til arkitektur i RUP (i motsetning til f.eks. Scrum og spesielt XP
- Sett dere inn malen for SAD og de to eksemplene som ligger under Ressurser i Fronter.