

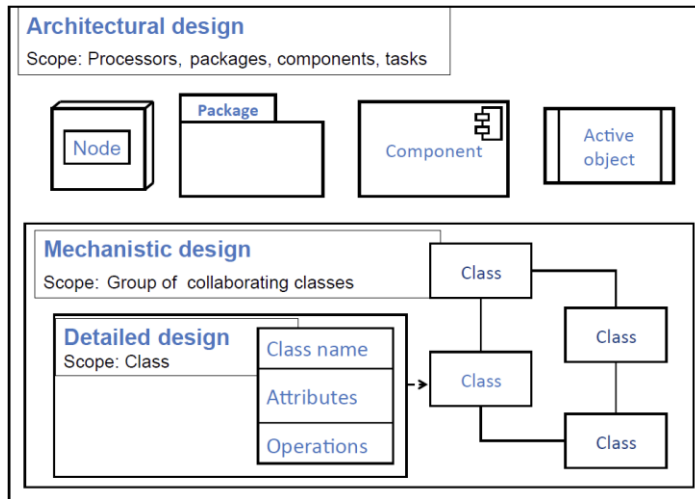
## 8. Software arkitektur

1. Redegør for begrebet softwarearkitektur
  - a. Forsøger at beskrive større elementer og komponenter i et system
    - i. Design forsøger at implementere disse
  - b. Højeste niveau hvor systemet brydes ned i bidder
    - i. Beskriver dele af et system som er svære at ændre
    - ii. Dokumentation af systemets opbygning
  - c. Arkitektur tager sig ikke af:
    - i. Mechanistic Design
    - ii. Typiske klassesdiagram: asso, kompo, arv
    - iii. Detaljeret design: klassenavne, attr, func
  - d. Arkitektur tager i højere grad stilling til:
    - i. Nodes:
      1. Fysiske enheder: computer, tablet, server, database osv.
      2. Hvor mange og hvilke CPU/hardware køres der med
      3. Hvordan kommunikeres der mellem noder
    - ii. Packages:
      1. UML Elementer: klasser eller Use Cases
      2. opdeling af logiske og fysiske delsystemer
    - iii. Component:
      1. Softwareenheder der kører i run-time: .exe, .dll, tabeller i database
      2. Nodes der indeholder run-time elementer kan vises på et deployment diagram
    - iv. Active Object
      1. objekt der køres i en tråd eller klasse
      2. Concurrency modellen og tråd-kommunikation mellem aktive enheder
  - e. Hvis ovennævnte er overholdt er man godt på vej til en god arkitektur
  - f. Andre typiske arkitekturer eksisterer som man kan bygge videre på
2. Eksempel på anden arkitektur: 3-tier arkitekturen
  - a. Presentation Tier: Visning af data
  - b. Business Logic: behandling af data
  - c. Database Tier: Persistering af data
  - d. Tegn eksempel, se disposition
    - i. Tier = fysisk
    - ii. Layer = logisk
  - e. Benyttes ofte til eksempelvis webapplikationer og windows applikationer
  - f. Fordele og Ulemper:
    - i. + Scalability: BLT kan køre på mange maskiner, PT har ikke forbindelse til database
    - ii. + Separation of Concern (SRP)
    - iii. + Reusability: Hvis grænsefladen er veldefineret, er sprog ligegyldigt.
    - iv. + Data Integrity: BLT sikrer at kun Valid data skrives til databasen
    - v. + Improved security: PT har ikke direkte adgang til database
    - vi. + Reduced Distribution: BLT ændres serverside, PT tilgår via API
    - vii. + Improved Availability
    - viii. – increased complexity

3. Hvordan udarbejdes en software arkitektur
  - a. Følgende trin skal udføres
    - i. Identificer arkitekturens formål
      1. hvad skal den?
      2. hvem skal bruge den?
      3. hvilke krav er der til den?
    - ii. Identificer vigtigste scenarier
      1. hvilke komponenter skal bruges til netop disse scenarier?
      2. hvordan skal systemet agere?
    - iii. Lav en applikationsoversigt
      1. identificer applikationstypen
      2. identificer deployment arkitektur
      3. identificer arkitektur stil (n-tier)
      4. identificer teknologier (.NET)
    - iv. opstil nøgle problemer
      1. opstil funktionelle krav for system
        - a. system qualities
        - b. run-time qualities
        - c. design qualities
        - d. user qualities
    - v. identificer løsninger
      1. opstil forbinder mellem noder, pakker, component og active object
4. hvordan dokumenteres en software arkitektur
  - a. der kan benyttes "4+1" eller "n+1"
  - b. gode til at fortælle stakeholders i projektet hvordan det hænger sammen
  - c. **Views**
    - i. Logical View beskriver
      1. Package Diagram
        - a. Klasse- og sekvensdiagrammer
        - b. Beskriver hvordan software er sat sammen logisk
    - ii. Implementation View beskriver
      1. Components i system
        - a. Eksistere kun under run-time (.exe, .dll etc.)
      2. Kan beskrive interfacet mellem to/flere programmer
    - iii. Deployment view beskriver
      1. tager sig ikke af fysiske enheder i system
      2. kommunikation mellem noder
      3. JEG VED IKKE HVORDAN DET RESTERENDE HÆNGER SAMMEN!!!
5. Network layers:
  - a. Se sidste side i disposition

# 8. Software arkitektur

## Redegør for begrebet softwarearkitektur.



Arkitektur forsøger at beskrive de større elementer og komponenter i ens system. Modsat design som søger at implementere disse. Man kan sige, at arkitekturen er det største niveau af abstraktion i beskrivelsen af ens system.

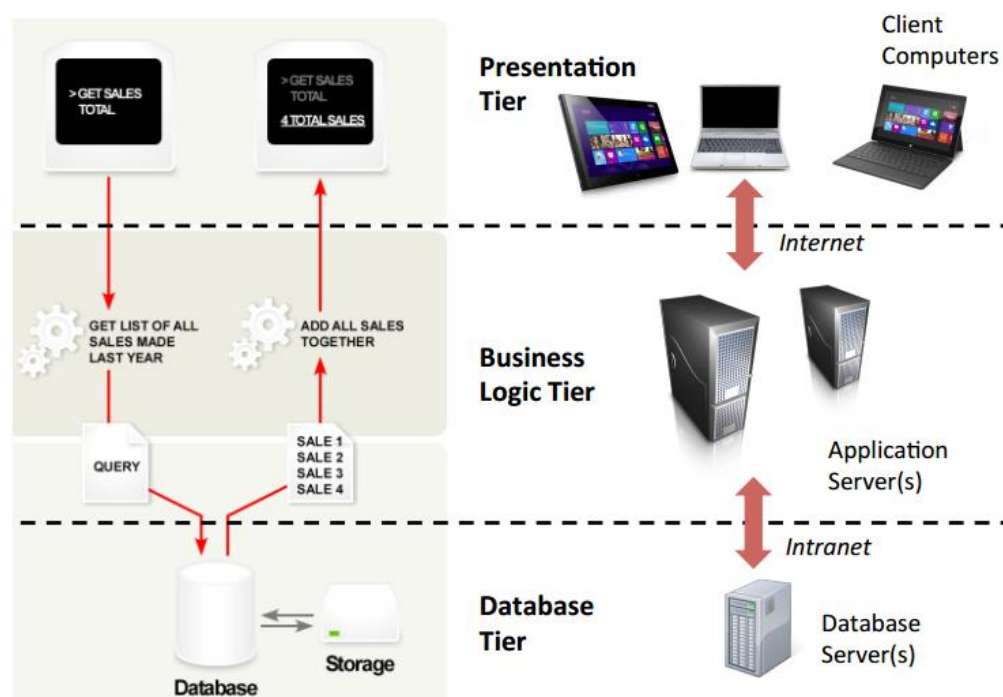
- Softwarearkitektur er det højeste niveau hvor man prøver at bryde sin applikation ned i bider
  - Det er her man beskriver de beslutninger om systemet som er svære at ændre (opbygning af systemet)
  - Man dokumentere hvordan systemet er bygget op (sat de overordnede dele sammen)
- I arkitekturen tager vi os ikke af
  - Mechanistic design dvs. hvordan klasserne arbejder sammen.
  - Typisk klassediagram med association, komposition, arv osv. eller sekvensdiagram.
  - Detaljeret design, dvs. klassernes navne, attributter og funktioner
- Vi er på et højere plan hvor man kan tage stilling til
  - Nodes
    - Fysiske enheder som eksisterer i systemet, computer, tablet, server, database osv. CPU.
    - Hvor mange, hvilke CPU/hardware kører de osv.
    - Hvordan snakker nodes sammen, hvilke teknologier.
  - Packages
    - Samling af UML elementer, typisk klasser eller use cases.
    - Opdeler logiske og fysiske delsystemer. Hvilke nodes de arbejder på.
  - Component
    - Beskriver en softwareenhed som eksisterer run-time (exe filer, programbiblioteker (dll), tabel i en database, filer)
    - Man kan explicit vise på deployment diagram hvilke nodes der indeholder run-time elementer

- Active object
  - Objekt som køres i en tråd eller klasse.
  - Concurrency modellen og tråd-kommunikationen mellem de aktive enheder.
    - **concurrency** refers to the ability of different parts or units of a program, algorithm, or problem to be executed out-of-order or in partial order, without affecting the final outcome.
- Tager man stilling til de ting, er man godt på vej i sin softwarearkitektur.
- Men der findes typiske softwarearkitekturer man kan benytte og bygge videre på

## Giv et eksempel på en typisk softwarearkitektur og dens anvendelse.

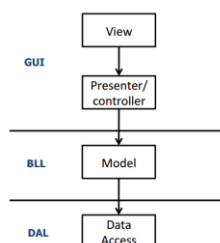
Den mest udbredte arkitektur-stil er 3-Tier arkitekturen. Her opdeles systemet i 3-lag:

- Presentation Tier (PT): Har ansvar for visning af data.
- Business Logic Tier (BLT): Har ansvar for behandling af data.
- Database Tier (DT): Har ansvar for persistering af data.

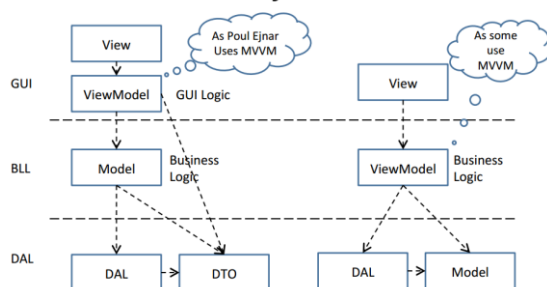


Tier = fysisk, Layer = logisk

Mapping MVC/MVP  
To 3-layer Architecture?



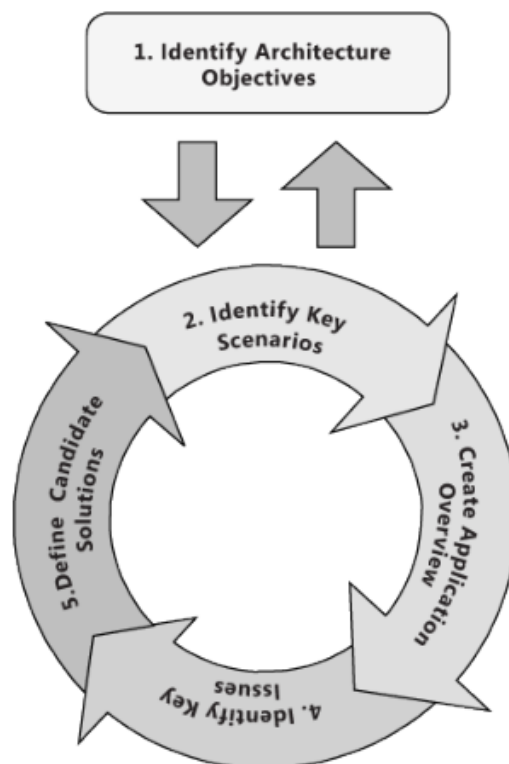
## MVVM and N-layer Architecture



3-tier kan både bruges i web applikationer såvel som en rich client (f.eks. windows applikation).  
Fordele og ulemper ved at bruge sådan en struktur:

- **+ Scalability:** BLT kan udstilles på mange maskiner. Hver PT klient har ikke brug for en forbindelse til DB.
- **+ Separation of Concern (SRP)**
- **+ Reusability:** Hvis man har defineret grænsefladen, er implementeringssproget ligegyldigt.
- **+ Data Integrity:** BLT sikrer at kun valid data, bliver skrevet i databasen.
- **+ Improved Security:** PT klient har ikke direkte adgang til databasen. BLT er placeret på en sikker server.
- **+ Reduced Distribution:** BLT ændres på en server, PT klienter tilgår en API.
- **+ Improved Availability:** Der kan indskydes en redundant BLT eller database server.
- **- Increased complexity/effort:** Det siger vel sig selv?

## Hvordan udarbejdes en software arkitektur?



Følgende trin skal udføres:

1. Identificér arkitekturens formål
  - Hvad skal arkitekturen?
  - Hvem skal bruge arkitekturen?
  - Hvilke krav er der til arkitekturen?
2. Identificér vigtige scenarier
  - Hvilke komponenter skal bruges til at implementere arkitekturen?

- Hvordan skal systemet agere (use-cases)?

### 3. Lav en applikationsoversigt

- Identificér applikationstypen (mobile, rich-client, web interface)
- Identificér deployment arkitektur (distributed/nondistributed, runtime, network)
- Identificér arkitektur stil (n-tier, 3-tier)
- Identificér teknologier (.NET, ASP.NET, knockout.js)

### 4. Identificér nøgle problemer

- Opstil ikke-funktionelle krav for ens arkitektur: system qualities, run-time qualities, design qualities, user qualities (authentication and authorization, caching, communication, configuration management, exception management, logging and instrumentation, validation)

### 5. Identificér løsninger

- Opstil forbindelser mellem noder, pakker, component, active object.

## Hvorledes dokumenteres en software arkitektur?

Der kan bruges "4+1" eller "n+1" til at dokumentere arkitekturen. Den er god fordi den kan fortælle alle de stake-holders der er i projektet hvordan det hænger sammen men fra forskellige vinkler.

Logical view:

- Til det logiske view i forbindelse med software arkitekturen er package-diagram

o I software-design så: klasse-diagrammer, sekvensdiagrammer

- Package-diagram beskriver pakker af softwareklasser eller use-cases som logisk er sat sammen

- Man kan sætte afhængigheder på men kun "bruger" afhængigheder

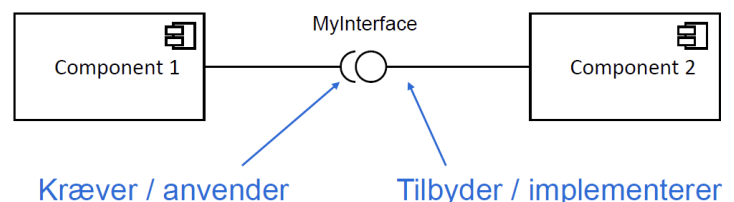
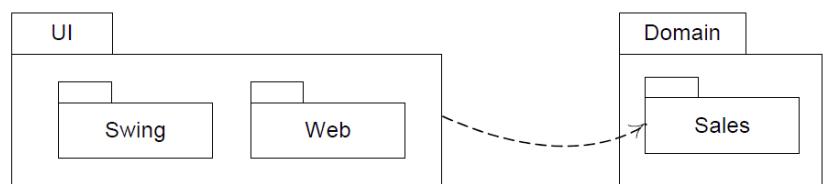
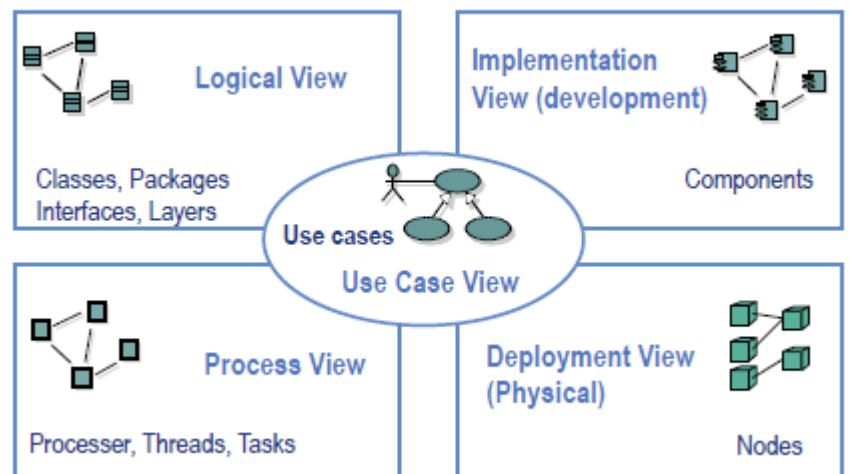
Implementation view:

- Viser de components systemet indeholder

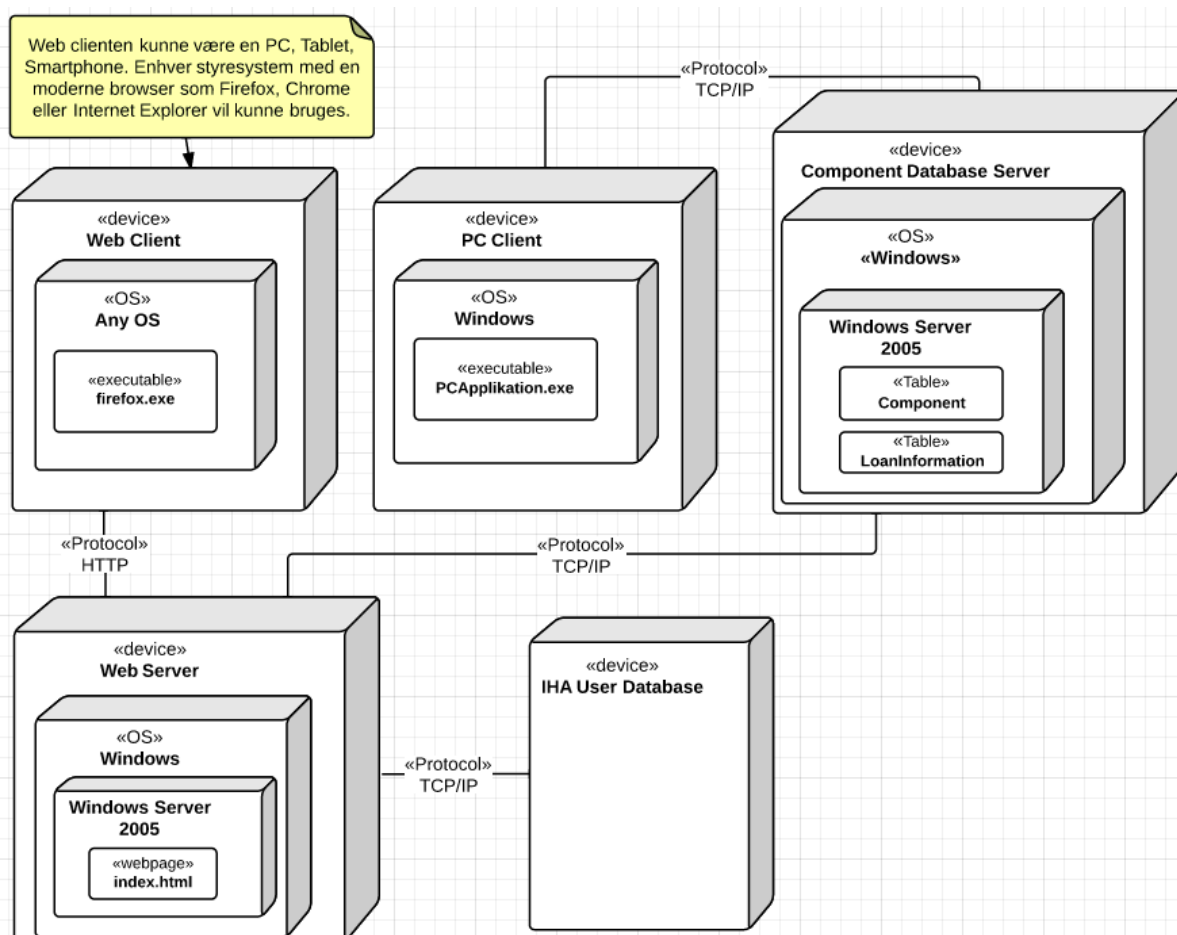
o Components eksisterer run-time (exe filer, programbiblioteker (dll), tabel i en database, filer) ol.

- Typisk meget store systemer, ikke noget jeg har arbejdet med

- Kan beskrive interfacet mellem fx to programmer



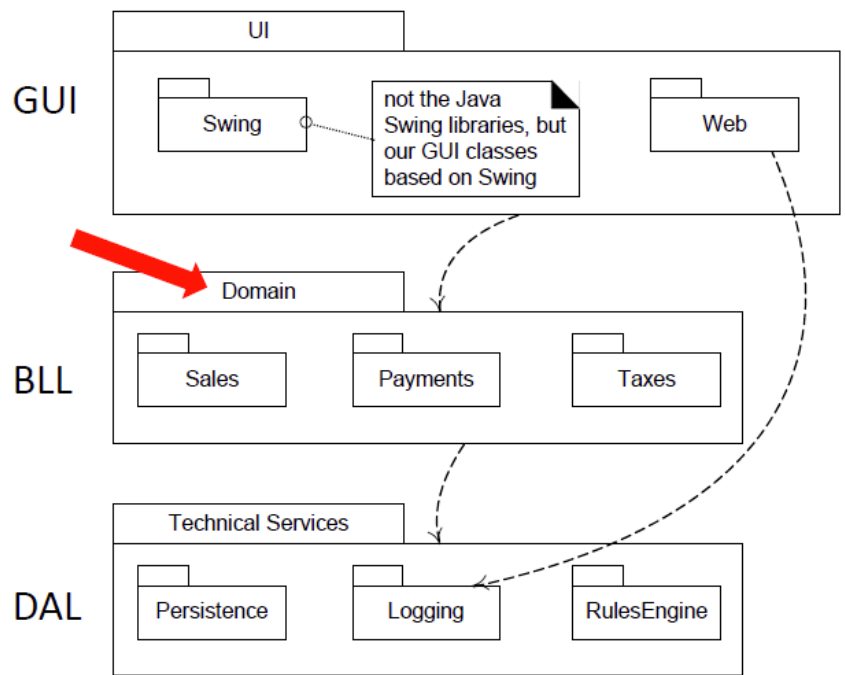
Deployment view:



- Tager sig af at vise de fysiske enheder der er i systemet
  - o Kan evt. vise hvilket software der kører på hver node fx PCApplikation, Table Component
- Viser hvordan nodes snakker sammen
- Vigtigt at skrive noget tekst til figuren. Figuren beskriver hvordan, men hvorfor?
- Evt. design alternativer.

Overvejelser ved layers arkitekturen

- Layer er en logisk separation af ansvar
- Separation of concerns er i fokus
- Reducerer kobling (færre afhængigheder)
  - Højere lag kan kalde ned i lavere lag
  - ikke umiddelbart omvendt
  - Selvom det kan implementeres vha. subject/observer pattern
- Fordel at lag kan genbruges da de er lavt koblede
- Pille et lag ud eller indsætte et nyt
- Kan kodes parallelt



## Network Layers

