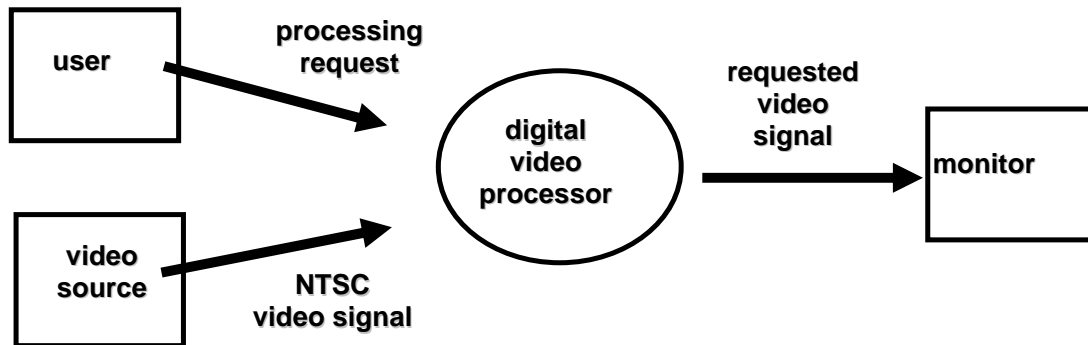


## Konsep Desain Software

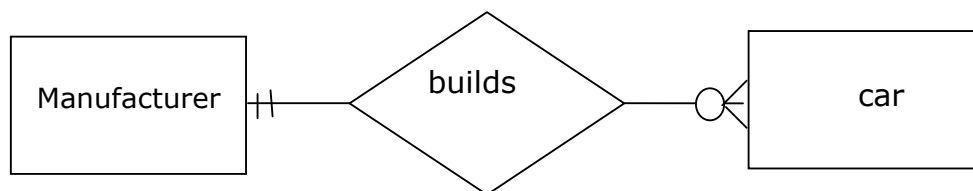
Disiapkan oleh Umi Proboyekti, S.Kom, MLIS

### Analisis dan desain model

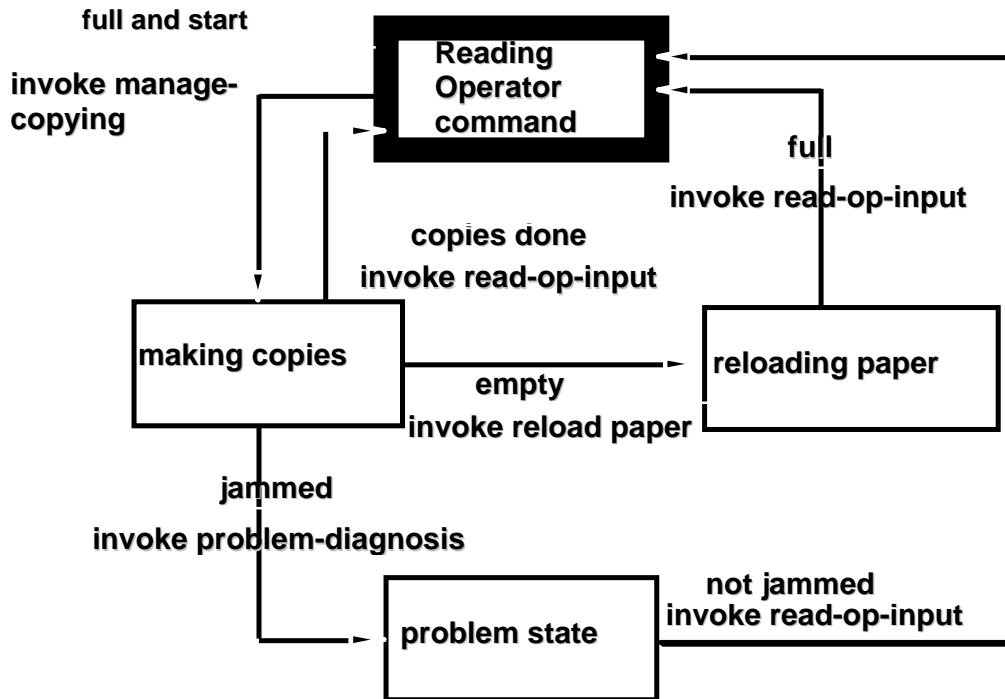
Setelah kebutuhan dikumpulkan, analisis terhadap kebutuhan dilakukan dengan menggunakan beberapa alat (tools) seperti DFD (Data Flow Diagram), ERD (Entity Relationship Diagram) dan STD (State Transition Diagram). Data Dictionary menjadi bekal dasar untuk menganalisis kebutuhan. Data Dictionary berisi gambaran dari semua objek data yang diperlukan dan dihasilkan oleh software nantinya. Diagram-diagram tadi mempunyai karakteristik masing-masing. DFD memberi gambaran bagaimana data berubah sejalan dengan alirannya dalam sistem dan menggambarkan fungsi-fungsi yang mengubah data-data tadi. ERD menggambarkan relasi antara objek data. STD menggambarkan bagaimana kerja sistem melalui kondisi (state) dan kejadian yang menyebabkan kondisi berubah. STD juga menggambarkan aksi yang dilakukan karena kejadian tertentu.



Gambar 1: Data Flow Diagram

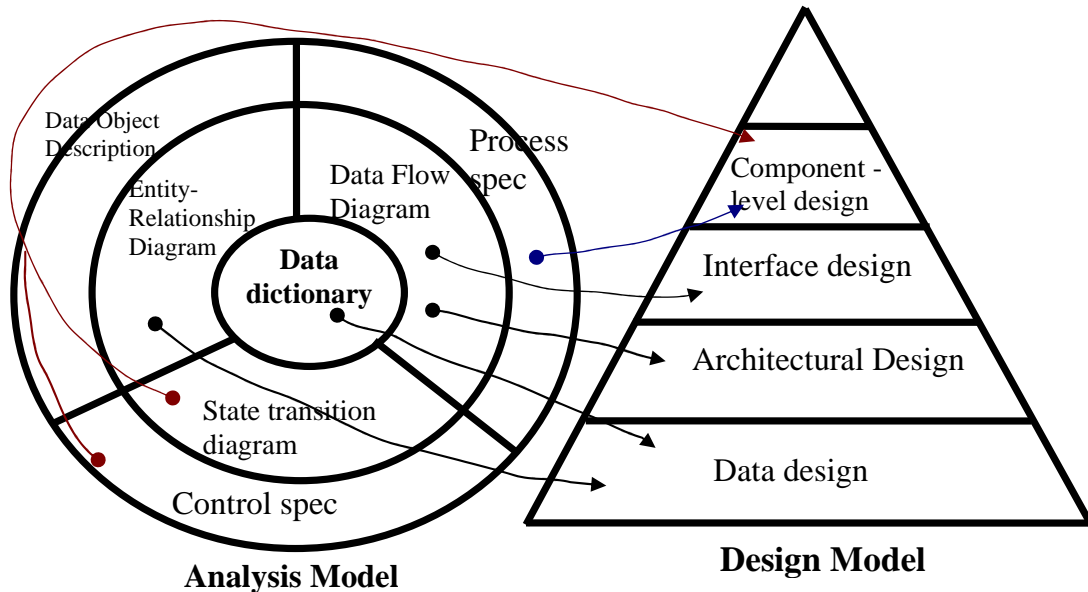


Gambar 2: Entity Relationship Diagram



Gambar 3: State Transition Diagram

Hasil yang diperoleh dari analisis kebutuhan adalah model analisis yang kemudian menjadi bekal untuk melakukan desain. Setiap bagian dari analisis model pada gambar 4 sebelah kanan menjadi bekal pada proses desain pada piramida model desain pada sebelah kiri gambar 4.



Gambar 4: hubungan antara model analisis dan model desain

## Model Desain

*Data design* mengubah informasi menjadi struktur data untuk mengimplementasikan software. Data design dibuat berdasarkan data dictionary dan ERD.

*Architectural design* mendefinisikan relasi antara elemen-elemen struktural utama, pola desain yang digunakan untuk mencapai kebutuhan yang ditentukan untuk sistem dan batasan-batasan yang mempengaruhi bagaimana desain arsitektural ini diterapkan. Desain ini berdasarkan spesifikasi sistem, model analisis (bagian DFD) dan interaksi antara subsistem.

*Interface design* menjelaskan bagaimana software berkomunikasi dalam dirinya, dengan sistem yang bertukar informasi dengannya, dan dengan manusia yang menggunakannya. DFD diperlukan untuk desain ini.

*Component-level design* menghasilkan deskripsi prosedur software.

## Konsep desain

### 1. abstraction

Abstraction adalah gambaran dari fungsi suatu program. Gambaran ini bisa bertingkat-tingkat. Tingkat yang paling atas adalah gambaran suatu fungsi program dengan menggunakan bahasa alami. Pada tingkat terendah, menghasilkan abstraksi yang bersifat prosedural/ langkah perlangkah dengan menggunakan istilah yang teknis dan bisa diimplementasikan menjadi fungsi program.

Pada saat beralih dari tingkat ke tingkat, kita menggunakan procedural dan data abstraction. Procedural abstraction adalah urutan instransi yang mempunyai tujuan khusus, dan data abstraction adalah koleksi data yang digunakan pada fungsi tersebut.

Contoh:

Program : Iklan Part-time Job

Fungsi: Pendaftaran calon part-timer

Abstraction 1 (highest level):

Calon part-timer dalam melakukan upload syarat-syarat yang diperlukan untuk melamar: surat lamaran, CV, foto, transkrip, data diri.

Abstraction 2 (lower level):

Procedural abstraction :

- tampilkan pilihan part-time job
- input data
- verifikasi format

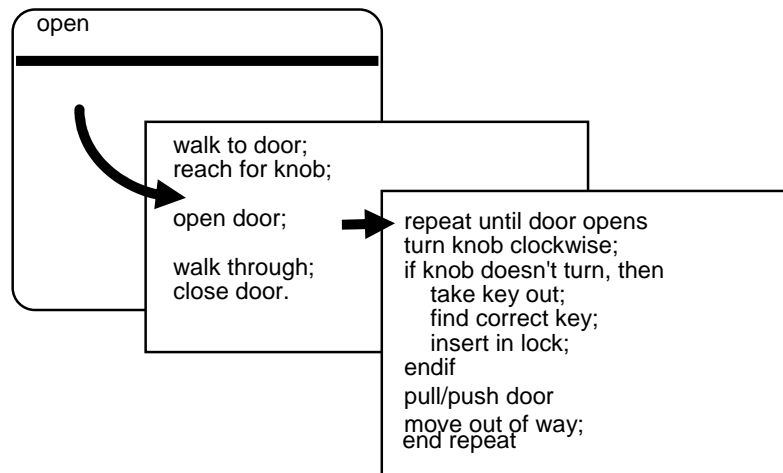
- kirim data

Data abstraction

- nama is STRING
- nim is STRING
- foto is IMAGE FILE
- surat\_lamaran is PDF FILE

## 2. refinement—penjelasan detil dari abstraction

Refinement membantu designer untuk memperlihatkan detil dari lowest level dari abstraction. Abstraction dan refinement merupakan konsep yang saling melengkapi. Contoh dari refinement tentang fungsi sebuah pintu ada pada gambar 5.



Gambar 5: hasil refinement fungsi sebuah pintu

## 3. modularity—membagi software menjadi modul

Software dibagi-bagi menjadi beberapa component yang disebut modul-modul. Modul-modul ini nantinya disatukan/diintegrasikan untuk memenuhi kebutuhan sistem. Dalam pembentukan modul-modul berlaku pernyataan-pernyataan berikut:

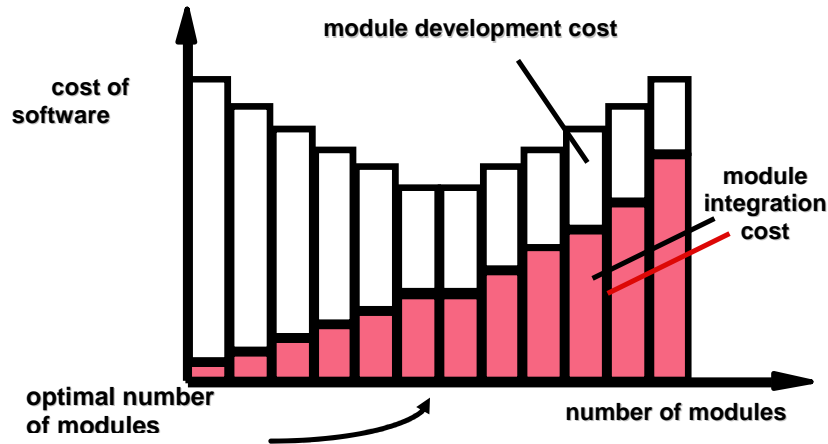
Jika  $C(p1) > C(p2)$  dimana  $C$  adalah complexity dari suatu modul, maka  $E(p1) > E(p2)$  dimana  $E$  adalah waktu yang diperlukan. Artinya semakin rumit sebuah modul, maka waktu yang digunakan untuk menyelesaikan modul tersebut makin banyak.

$$C(p1+p2) > C(p1) + C(p2)$$

Dan

$$E(p_1+p_2) > E(p_1) + E(p_2)$$

Untuk itu, modul yang rumit dipecah lagi menjadi beberapa modul untuk memudahkan penyelesaian masalah. Namun semakin banyak modul, maka waktu/biaya untuk integrasikan modul-modul tersebut juga makin tinggi. Seperti pada grafik pada gambar 6.



Gambar 6: Hubungan jumlah modul dan harga/biaya integrasi

4. software architecture—struktur software secara keseluruhan

struktur hirarki/berjenjang dari modul-modul program. Untuk menggambarkan struktur modul-modul tersebut beberapa model yang ada adalah :

- framework model : identifikasi pola yang berulang-ulang
- dynamic model : identifikasi bagaimana konfigurasi sistem berubah karena kejadian-kejadian tertentu
- process model: fokus pada proses teknis yang harus dikerjakan sistem
- functional model : menggambarkan hirarki sistem berdasarkan fungsinya

5. Software procedure

Fokus pada detail proses pada tiap modul. Prosedur menjelaskan proses, urutan kejadian, proses perulangan, penentuan keputusan/arah. Ini bisa digambarkan dengan menggunakan Flow Chart yang bertingkat.

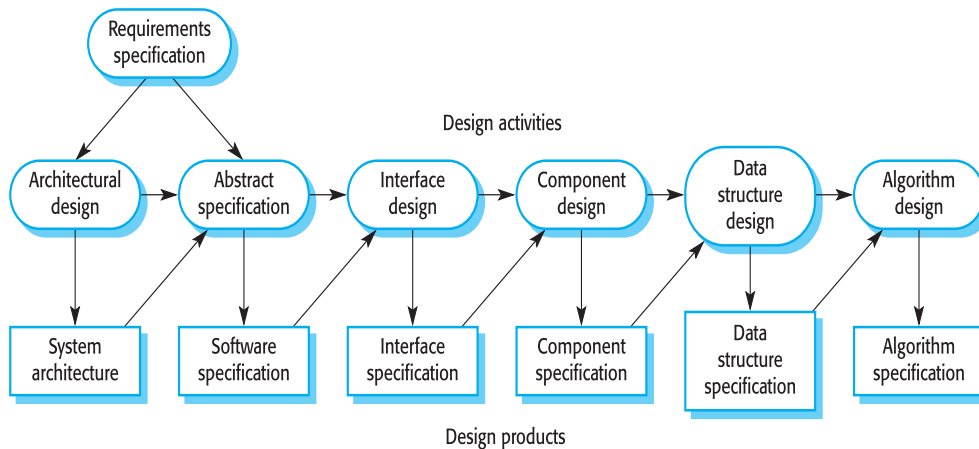
6. Information hiding

Ide dari information hiding (menyembunyikan informasi) adalah modul dirancang sedemikian rupa sehingga informasi (prosedur dan data) yang di dalamnya tidak dapat diakses oleh modul lain yang tidak memerlukannya.

Modul yang efektif adalah modul yang berdiri sendiri dan berkomunikasi dengan modul lain yang memang diperlukan.

## Desain Arsitektur Software

Suatu sistem, entah itu besar atau tidak, dibangun dari sub-sub sistem yang lebih kecil. Sub-sub sistem ini memiliki fungsi sendiri-sendiri. Proses merancang untuk menentukan sub-sub sistem dan membangun kerangka kerja untuk kendali dan komunikasi antar sub sistem disebut design arsitektural. Proses merancang ini menghasilkan arsitektur software atau arsitektur sistem. Desain arsitektur adalah aktifitas desain yang pertama dalam pembangunan software seperti yang digambarkan pada Gambar 1.



Gambar 1: Aktifitas Desain dan hasil rancangan

Desain arsitektur memberikan 3 keuntungan yaitu:

1. arsitektur software menjadi media komunikasi dan diskusi karena mudah dipahami
2. memberi kemudahan dalam melakukan analisis terhadap software yang akan dibangun
3. arsitektur-nya bisa digunakan lagi untuk sistem selanjutnya (reusable)

Tiap perancang sistem memiliki kemampuan dan pengetahuan yang berbeda dalam merancang arsitektural. Aktifitas-aktifitas berikut adalah aktifitas dalam merancang dan aktifitas ini tidak dikerjakan satu persatu berurutan, tapi bisa dilakukan bersamaan.

1. Menyusun sistem (system structuring) : sistem disusun menjadi beberapa subsistem utama, dimana subsistem adalah unit bagian software yang berdiri sendiri.
2. Membuat model kendali (Control modelling) : berkaitan dengan hubungan antara bagian dalam sistem.
3. Membuat pembagian sistem menjadi modul-modul (modular decomposition) : membagi sub-sub sistem menjadi modul-modul

Untuk menghindari kesalahan dalam pemahaman terhadap istilah modul dan sub sistem, perlu diketahui bahwa sub sistem adalah bagian dari sistem yang bisa berdiri sendiri dan tidak bergantung pada layanan sub sistem lain. Sub sistem terdiri dari beberapa modul dan dilengkapi interface untuk berkomunikasi/ bertukar data dengan sub sistem lain.

### **System Structuring (struktur sistem)**

Struktur sistem menggambarkan sub-sub sistem dan interaksi antara sub-sub sistem. Desain dengan menggunakan diagram-diagram untuk menggambarkan sub sistem dan interaksinya agar mudah dipahami.

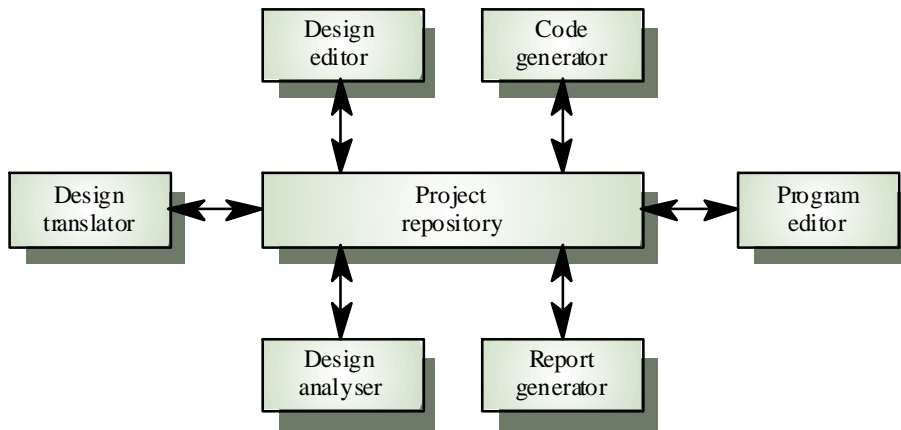
Beberapa model dari struktur sistem yang menjelaskan bagaimana sub sistem berbagi data, bagaimana sub sistem terdistribusi dan bagaimana sub-sub sistem saling berinteraksi adalah:

#### **1.Repository Model**

Pada model ini data disimpan secara terpusat. Ada dua cara terpusat pada model ini:

1. Database terpusat dan dapat diakses oleh semua sub sistem dalam sistem tersebut. Contoh : sistem informasi perpustakaan UKDW.
2. setiap sub sistem menyimpan database sendiri dan bisa bertukar data dengan sub sistem lain melalui pengiriman pesan (message).

Diagram menggambarkan model ini seperti pada Gambar 2. Sub-sub sistem pada Gambar 2 mendapatkan data dari repository (kumpulan data). Data tersimpan secara terpusat pada satu tempat.



Gambar 2: Repository model

- Keuntungan
  - Efisien untuk share jumlah data yang besar. Tidak perlu kirim data secara langsung dari satu sub sistem ke sub sistem yang lain
  - Sub-sistem tidak perlu memikirkan bagaimana data digunakan oleh sub sistem lain.
  - manajemen data seperti backup, keamanan, re-index, dan kontrol akses dilakukan secara terpusat. Itu merupakan tanggung jawab manager repository
- Kerugian
  - Sub-system harus mengikuti model yang sudah ditetapkan. Jadi jika ada sub sistem baru, maka yang baru harus menyesuaikan dengan model yang ada.
  - Evolusi data sulit dan mahal karena volume informasi yang besar dihasilkan dengan model tertentu. Mengubahnya ke model yang lain pun tidak mudah
  - Sulit untuk distribusi layanan secara efisien, karena yang melayani hanya satu.

Contoh untuk model repository dengan database terpusat adalah : sistem informasi perpustakaan UKDW, sistem registrasi akademik UKDW

## 2. Client-Server Model

Model ini terdiri dari server yang berdiri sendiri dan menyediakan layanan untuk client-client. Ada client-client (sub-sistem) yang menggunakan layanan server dan tersedia network yang mengijinkan client untuk akses layanan dari server.

Komponen utama pada model ini :

1. Ada stand-alone server yang menyediakan layanan ke sub-sub sistem

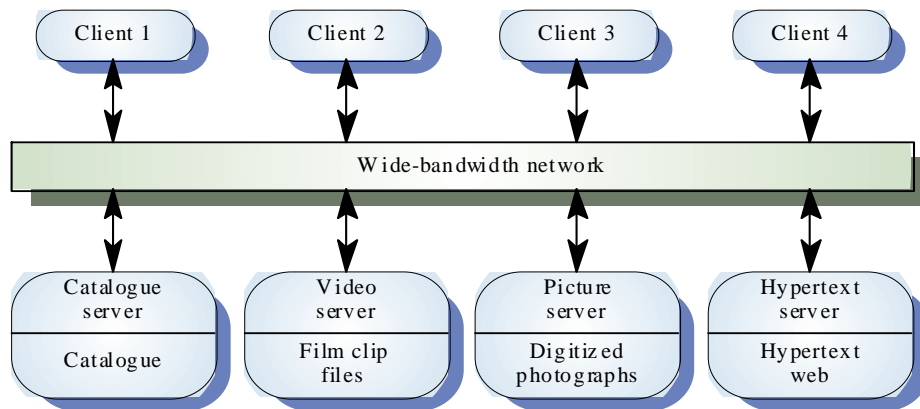
2. Ada sub sistem yang disebut juga client yang memanggil/mengakses layanan di server-server
3. Ada jaringan memungkinkan sub-sub sistem mengakses layanan-layanan pada server.

Untuk mengakses suatu server maka sub sistem atau client harus mengetahui alamat atau nama server yang diakses dan juga layanan yang diberikan. Sebaliknya, server tidak perlu tahu berapa client/sub sistem yang mengaksesnya dan sub sistem mana yang menggunakannya.

Arsitektur client server memiliki struktur yang terdiri dari 3 lapisan yang harus ada yaitu:

1. business logic/ application
2. data management
3. presentation layer

Gambar 3 dan 4 adalah contoh-contoh dari model client-server, dimana ada beberapa server dan client. Masing-masing server menyimpan datanya sendiri dan setiap client bisa mengakses/menggunakan layanan pada tiap server.

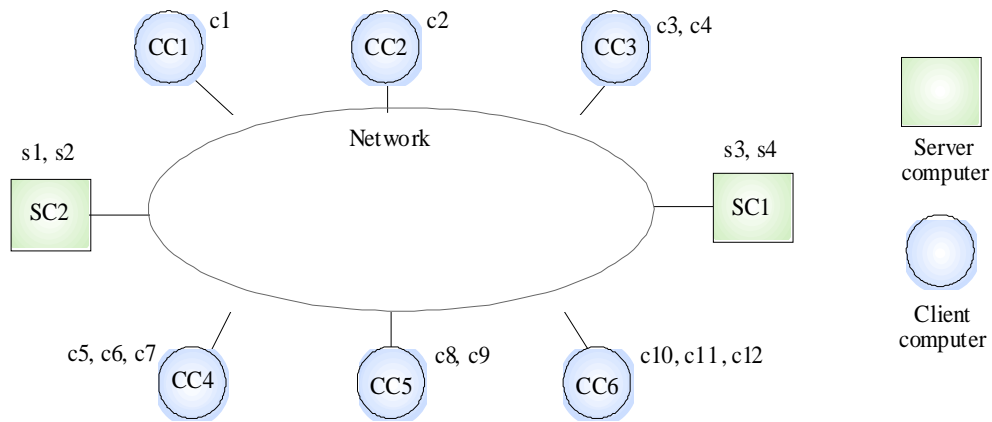


Gambar 3 : Client-Server arsitektur

- Keuntungan
  - Distribusi data secara langsung melalui jaringan
  - Penggunaan sistem jaringan secara efektif –hardware jadi murah
  - Mudah untuk tambahkan server baru atau up-grade server yang sudah ada
- Kekurangan
  - Tidak ada data model, jadi organisasi data macam-macam, sehingga integrasi data sulit
  - Redundant management

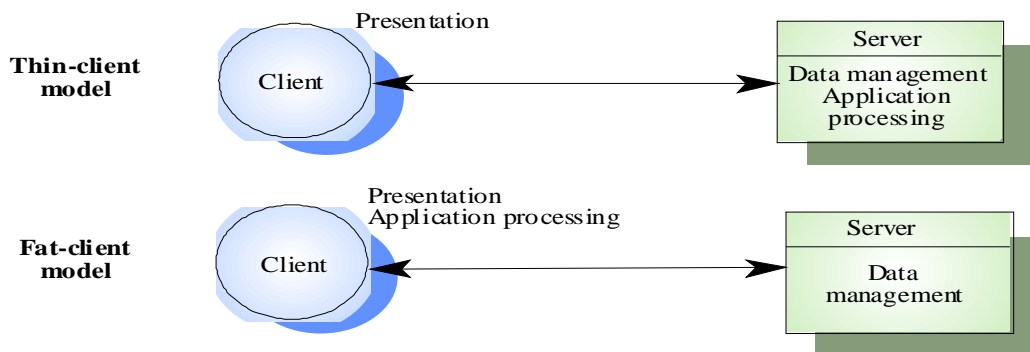
- Tidak ada pusat register nama dan service, sehingga kalau tidak tahu nama server dan service-nya sulit ditemukan
- Manajemen data dilakukan pada tiap server, tidak terpusat karena masing-masing server memiliki karakteristiknya sendiri.

Client server merupakan arsitektur terdistribusi. Bisa digunakan secara efektif pada jaringan dengan prosesor yang terdistribusi.



Gambar 4: Client-server arsitektur

Tiga lapisan pada client-server arsitektur menentukan model dari client-server. Perbedaan model-model tersebut adalah pada distribusi 3 lapisan tersebut. Model distribusi 3 lapisan client-server adalah : two-tier, three-tier dan n-tier (multi-tier).



### 2.1 Two-tier architecture:

- Thin-Client model

Menempatkan business logic/application process dan data management pada server dan presentation pada client. Server mengerjakan pekerjaan berat

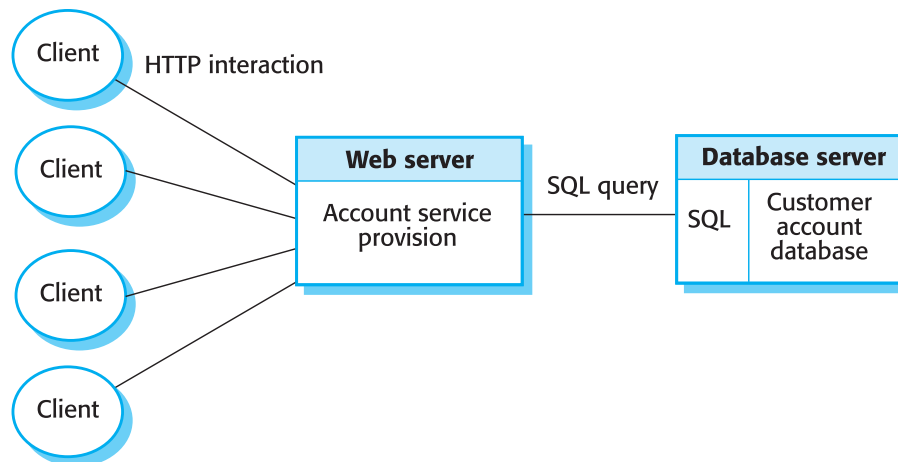
yaitu menjalankan application process dan data management Contoh :  
website

- Fat Client model

Menempatkan business logic/application process dan presentation pada client dan server hanya mengurus data management. Contoh : suatu aplikasi dibangun dengan VFP dan mengakses database Oracle. Semua application process dan presentation di client yang menggunakan VFP.

## 2.2 Three-tier

Memisahkan secara logic, presentation yang ada di client dengan application process yang berada terpisah secara logic dengan data management. Contoh: Internet Banking system



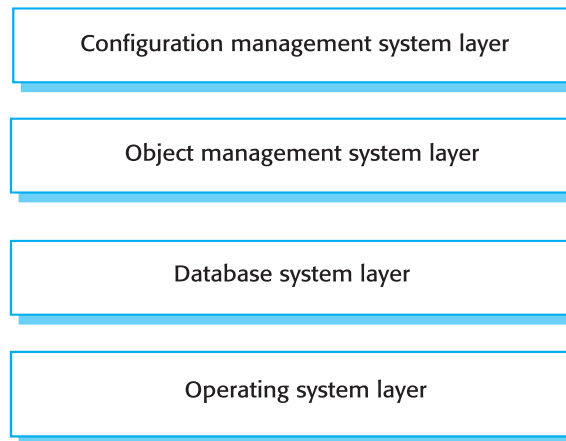
Gambar 4

## 2.3 Distributed object arsitektur

Pada model ini komponen yang terpenting adalah objek yang menyediakan antarmuka untuk layanan-layanannya guna dipanggil oleh objek lain. Masing-masing objek dapat dipanggil oleh objek lain dalam sistem tersebut. Tidak ada lagi pembagian client-server, karena tiap objek dapat berperan menjadi client dan server bergantung pada operasi yang dilakukan. Jika objek tersebut memberikan layanan pada objek lain, berarti objek yang memberi layanan berperan sebagai server, dan objek yang menggunakan layanan berperan sebagai client

### 3. Abstract Machine Model

Model ini juga disebut dengan layered model. Pada model ini sistem terdiri dari serangkaian lapisan (layer) yang masing-masing menyediakan layanan-layanan khusus. Setiap lapisan (layer) merupakan satu abstract machine yang layanannya digunakan pada abstract machine pada tingkat berikutnya.



Gambar 5: Abstract Machine Model

- Keuntungan
  - mudah diubah
  - perubahan yang terjadi pada satu lapisan hanya berpengaruh pada lapisan yang terdekat
- Kerugian
  - akses terhadap satu layanan pada lapisan yang dalam tidak bisa langsung karena harus melewati lapisan-lapisan sebelumnya
  - kinerja sistem bisa jadi lambat karena harus melewati lapisan-lapisan sebelumnya

### Control Models (Model kendali)

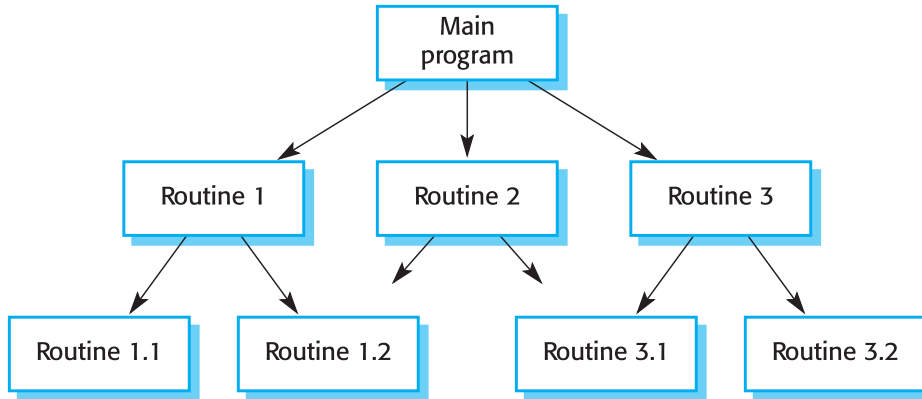
Sub-sub sistem perlu kendali agar bisa bekerja sebagai suatu sistem. Model struktur di atas tidak menggambarkan kendali, tapi model kendali yang menjelaskan aliran kendali antar sub-sub sistem.

Dua pendekatan model kendali dan variannya yang umum adalah :

#### 1. Centralised control

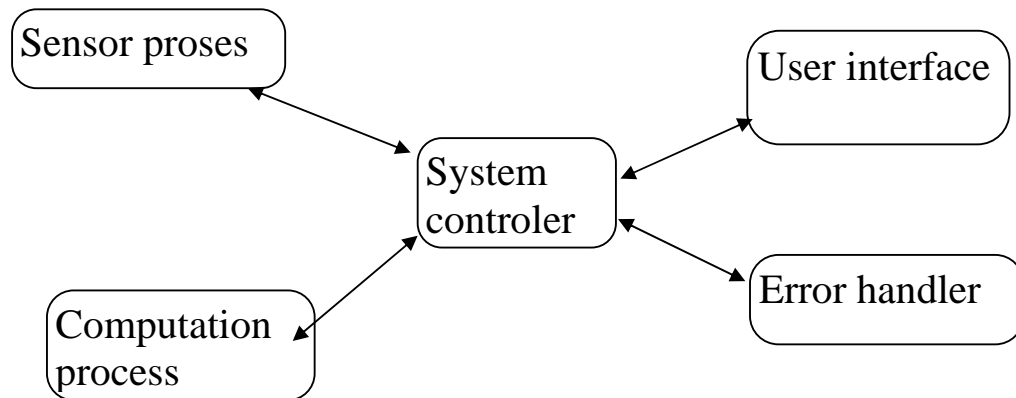
Satu sub-system bertanggung jawab untuk mengatur eksekusi sub-system lain

- The call-return model : Top-down subroutine model. Control mulai dari paling atas dari hirarki subroutine dan mengalir ke bawah



Gambar 6 : Call Return Model

- The manager model : Satu komponen sistem jadi manager dan mengendalikan start-stop dan koordinasi proses sistem. Satu proses adalah satu sub-sistem yang bisa dieksekusi secara paralel dengan sub-sistem lain

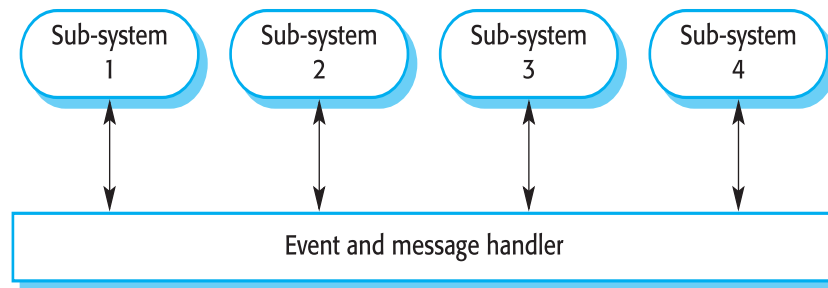


Gambar 7 : Manager Model

## 2. Event-based control

### 3.2.1 broadcast model :

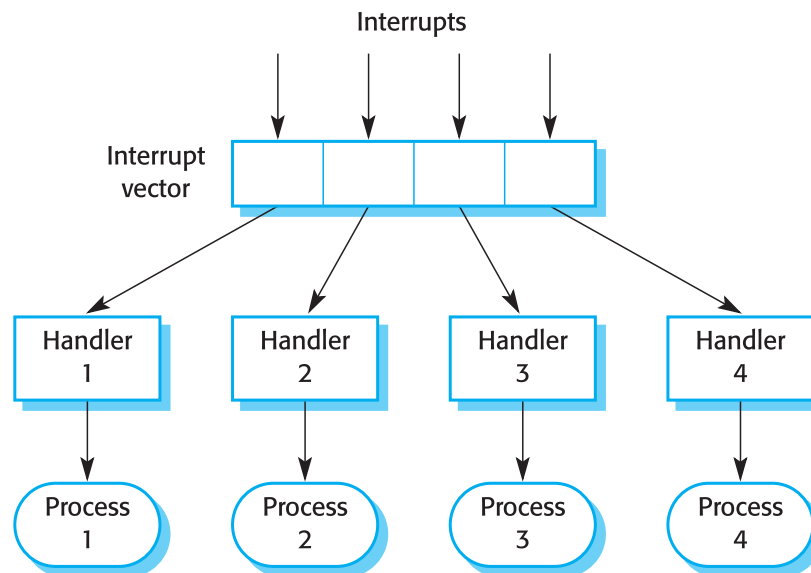
- Event di broadcast ke semua sub-system
- Sub-system register ke specific event, saat ini terjadi control ditransfer ke sub-system yang handle event tersebut
- Sub-system tentukan event yang dibutuhkan



Gambar 8: Contoh arsitektur broadcast model

### 3.2.2. interrupt-driven models

- Digunakan secara khusus pada real-time system yang mendeteksi interupsi luar
- Memberikan respon yang cepat pada suatu kejadian
- Membutuhkan pemrograman yang rumit dan testing yang sulit.
- Contoh: Real-time system pada air safety bag di mobil



Gambar 9: Contoh arsitektur event-driven model

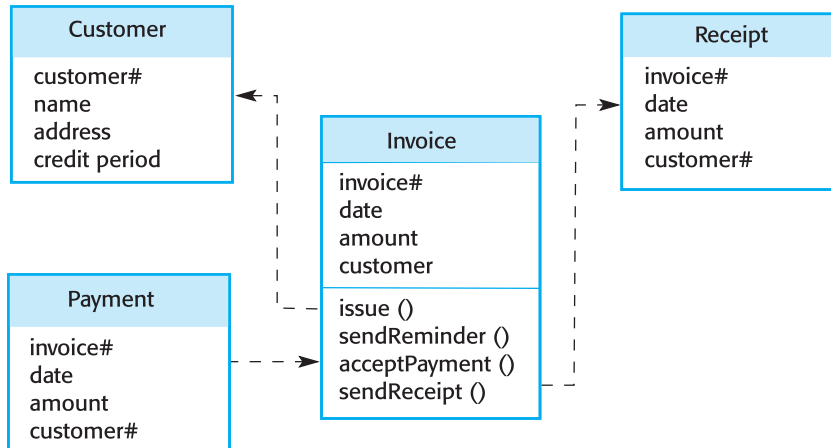
## Modular Decomposition

Membagi sub-sistem menjadi beberapa modul. Ada 2 model dalam desain arsitektur jenis ini:

### 1. Object-oriented Models

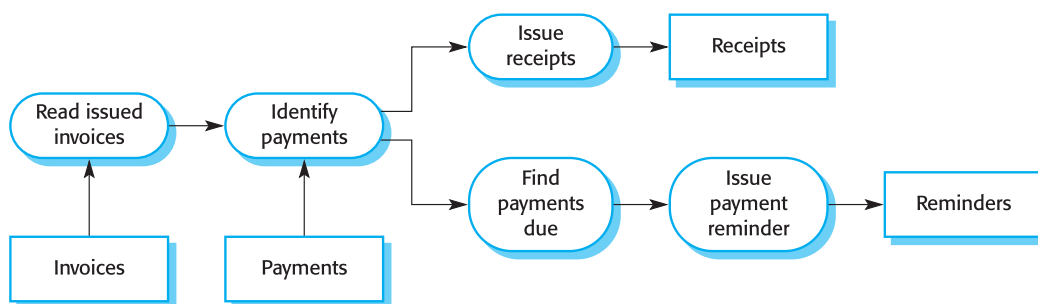
- Sistem dibagi-bagi menjadi objek-objek yang saling berkomunikasi

- Berkaitan dengan class yang terdiri dari atribut-atribut dan operasi-operasinya.
- Perubahan pada object tidak mempengaruhi object lain
- Cenderung mudah dipahami karena mewakili keadaan objek yang sebenarnya di dunia nyata
- Penggunaan layanan/service objek lain harus mengacu pada nama atau antarmuka dari objek tersebut



## 2. Data Flow models

- Sistem dibagi-bagi menjadi fungsi-fungsi yang menerima input dan mengubahnya menjadi output. Model ini juga disebut pendekatan pipeline
- Aliran data mengalir dari satu proses ke proses lain secara sekuensial dan setiap proses merupakan transformasi data.
- Proses ada yang dilakukan secara sekuensial, tadi ada juga yang paralel



Diadaptasi dari:

1. Pressman, Roger.S. "Software Engineering : A Practioner's Approach." 5th . McGrawHill. 2001.
2. Sommerville, Ian. "Software Engineering" .6th . Addison Wesley. 2001