Tim Pengajar IF2250

IF2250 – Rekayasa Perangkat Lunak

Konsep Perancangan

SEMESTER II TAHUN AJARAN 2023/2024

KNOWLEDGE & SOFTWARE ENGINEERING

Perancangan Perangkat Lunak

- Perancangan PL
  - Terdiri dari sekumpulan **prinsip**, **konsep** dan **praktek** pengembangan perangkat lunak dengan tujuan menghasilkan sistem yang berkualitas tinggi
- Prinsip perancangan akan memandu pekerjaan perancangan yang akan dilakukan
  - Konsep ini harus dimengerti sebelum implementasi dilakukan
- Perancangan memberikan berbagai **variasi representasi** perangkat lunak sebagai dasar panduan pengembangan yang berikutnya
- Setelah kebutuhan PL diidentifikasi dan dimodelkan maka perancangan (perancangan) PL adalah aktivitas **pemodelan** terakhir sebelum konstruksi PL (koding dan pengujian)

Proses Perancangan

- Perancangan PL adalah proses iteratif mengubah kebutuhan menjadi cetak-biru (*blue-print*) untuk pengembangan perangkat lunak
  - Cetak-biru ini memberikan gambaran umum dari PL.
    - Perancangan adalah level tinggi dari abstraksi
      - Level yang dapat ditelusuri dari objektif sistem hingga kerincian data, fungsi, dan perilaku (*behavior*) dari kebutuhan
  - Ketika iterasi perancangan terjadi, hasil kebutuhan akan mengarah ke perancangan yang makin rinci (tingkat abstraksi yang lebih rendah).

Syarat perancangan yang baik

- Semua kebutuhan yang eksplisit dari model analisis diimplementasikan
  - Termasuk mengakomodasi kebutuhan implisit dari pengguna
- Dapat dibaca dan dimengerti, agar menjadi panduan dalam membuat koding program, pengujian dan panduan sistem
- Dapat memberikan gambaran lengkap (data, fungsi, dan perilaku) dari sudut pandang implementasi.

Panduan Umum Perancangan

- Perancangan harus **melihat berbagai sudut pandang**, misalnya:
  - Teknologi yang tersedia
  - Kemampuan pengguna
  - Ketersediaan infrastruktur
  - Kebutuhan perangkat lunak
- Perancangan sebaiknya **dapat dilacak** dari model analisis
  - Semua elemen hasil analisis harus muncul sebagai elemen perancangan
- Perancangan tidak dikembangkan dari awal (nol) - **Reuse**
  - Semaksimal mungkin memanfaatkan hasil rancangan yang pernah ada
- Perancangan **meminimisasi 'gap'** antara software dengan dunia nyata
  - Software menggantikan fungsional atau suatu refleksi elemen dunia nyata, sehingga software harus dibuat berdasarkan acuan di dunia nyata
- Perancangan bersifat **seragam** dan mengandung **kesatuan**
  - Misalnya interaksi yang dibuat harus konsisten
  - Contoh: Perhatikan interaksi penggunaan menu pada MS Word atau MS Excel atau MS PowerPoint

From Davis [DAV95]

Panduan Umum Perancangan (2)

- Hasil perancangan **distrukturkan secara baik** sehingga tidak 'rusak' hanya karena data yang tidak lengkap atau jika ditemui kondisi yang tidak biasa.
  - Jangan sampai terjadi 'tambal-sulam'
  - Sangat penting memiliki spesfiikasi kebutuhan yang lengkap, konsisten, singkat, padat
- Perancangan **bukanlah koding**, dan koding **bukanlah perancangan**
  - Koding dibuat berdasarkan hasil perancangan, sehingga banyak elemen dari design perlu penerjemahan yang benar oleh pemrogram.
- Perancangan sebaiknya dinilai **kualitasnya saat proses** pembentukan, dan bukan sesudahnya
  - Hasil perancangan (=program) mungkin bagus, tetapi proses pembuatan perancangan mungkin belum tentu bagus, sesuai standard, sehingga nantinya tidak *reusable*.
  - Proses **pembentukan yang bagus**, menjamin hasilnya sesuai dengan kebutuhannya,
- Perancangan sebaiknya **dikaji-ulang (*review*)** untuk mengurangi kesalahan konsep
  - *Review* perlu dilakukan bersama-sama, karena satu individu manusia masih memiliki kecenderungan untuk berbuat salah.

From Davis [DAV95]

Atribut Kualitas untuk Perancangan - FURPS (Hewlett-Packard)

- **Fungsionalitas**: sekumpulan fungsi & fitur serta kemampuan program
  - perancangan diharapkan memenuhi fungsi/fitur/kemampuan dari program sesuai dengan spesifikasi kebutuhan Sistem/Perangkat Lunak
- **Usability** (**penggunaan**) – faktor manusia (estetika, konsistensi, dokumentasi)
  - perancangan harus memperhatikan kemudahan pengguna dalam menjalankan sistem/perangkat lunak
- **Reliability** (**Keandalan**) – frekuensi dan kerugian terjadinya kegagalan
  - perancangan perlu memperhatikan keandalan dari sistem/perangkat lunak yang akan dikembangkan
- **Performansi** – Kecepatan proses, waktu respon, waktu keseluruhan dan efisiensi
  - perancangan perlu memperhatikan performansi dari sistem/perangkat lunak
- **Supportability** – **maintainability** (extensibility, adaptability, serviceability), testability, compatibility, configurability
  - perancangan perlu memperhatikan factor maintainability dari sistem/perangkat lunak

KNOWLEDGE & SOFTWARE ENGINEERING

Konsep Dasar Perancangan

- **Abstraksi** — terhadap data, prosedur dan kontrol (kendali)
- **Stepwise Refinement** — Elaborasi rinci dari setiap hasil abstraksi
- **Modularitas** — pengelompokan data dan fungsi
  - Functional Independence
- **Arsitektur** — struktur perangkat lunak
  - Structural properties
  - Extra-structural properties
  - Styles and patterns
- **Partisi struktural** – horisontal /vertical (factoring)
- **Struktur Data**
- **Prosedur** – Algoritma yang melakukan suatu fungsi tertentu yang diperlukan
- **Information Hiding** (penutupan informasi) – pengaturan interface
- **Patterns**
- **Separation of Concerns**
- **Aspects**
- **Refactoring**

KNOWLEDGE & SOFTWARE ENGINEERING

Abstraksi

ABSTRAKSI DATA

ABSTRAKSI PROSEDUR

ABSTRAKSI KENDALI

KNOWLEDGE & SOFTWARE ENGINEERING

Abstraksi

- Abstraksi adalah suatu cara untuk mengatur kompleksitas pada suatu sistem komputer *)
  - Pada **setiap level** dalam suatu sistem komputer memiliki **tingkat kompleksitas** yang berbeda-beda
  - Level yang lebih rendah memiliki kompleksitas yang lebih rendah, makin tinggi levelnya maka kompleksitas makin meningkat
  - Tingkat yang lebih tinggi mengandalkan tingkat bawah, sehingga bila tingkat bawah salah, maka akan berpengaruh pada tingkat yang lebih tinggi
  - Peningkatan kompleksitas menyebabkan perbedaan cara penanganan

KNOWLEDGE & SOFTWARE ENGINEERING

*) https://en.wikipedia.org/wiki/Abstraction_(computer_science)

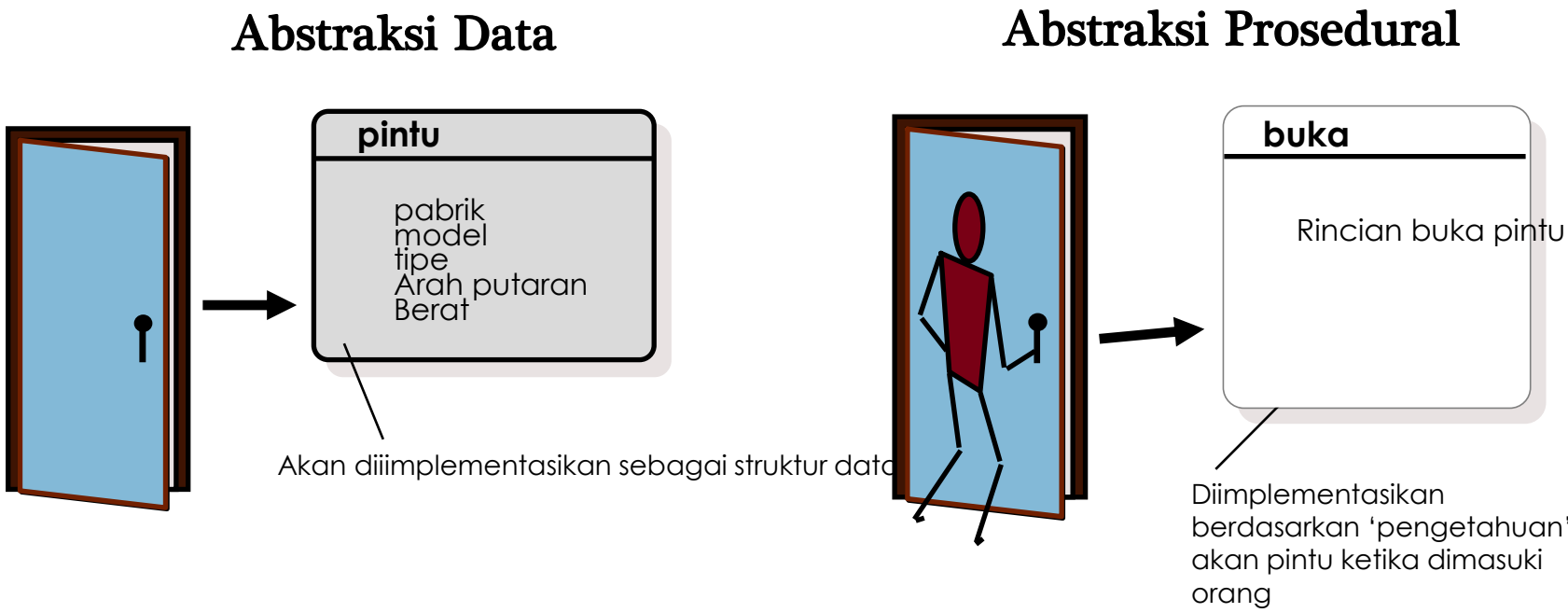
Abstraksi (lanjutan)

- Contoh 3 level abstraksi:
  - Penggunaan Email
    - User biasa ketika menggunakan email tidak peduli dengan bagaimana email itu bisa dikirim atau diterima, yang penting user tahu nama user dan passwordnya
  - Pengaturan (Administrasi) Email
    - Administrator harus menyiapkan email server
    - Administrator jaringan harus menyiapkan sistem infrastruktur internet atau kabel-kabel fisik
  - Pemrograman
    - Pemrogram aplikasi menggunakan tipe data integer dan hanya peduli dengan bagaimana suatu angka dapat disimpan sebagai tipe integer, tetapi dia tidak peduli dengan bagaimana 16 bit atau 32 bit integer di simpan
- Pada setiap lapisan kompleksitas, level di atas menggunakan abstraksi dari level di bawahnya.
- Contoh abstraksi:
  - Abstraksi data
  - Abstraksi prosedural/tindakan/action

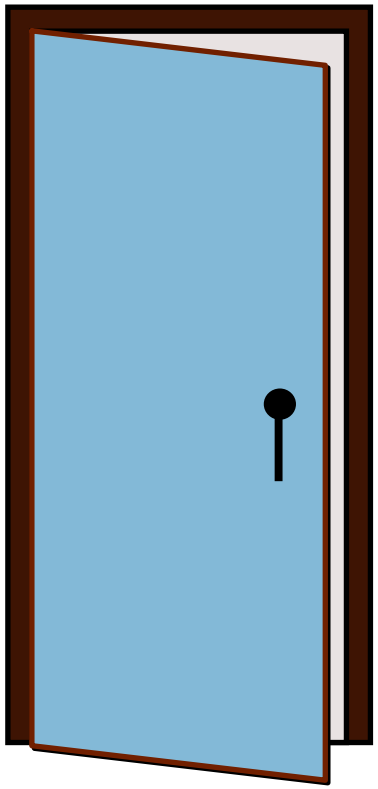
*) https://en.wikipedia.org/wiki/Abstraction_(computer_science)

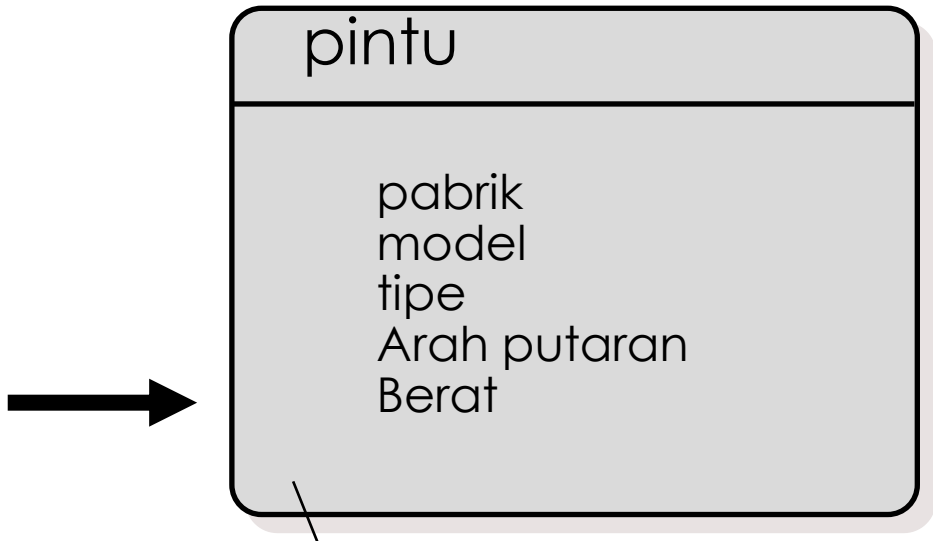
Abstraksi (lanjutan)

- Abstraksi dapat ditampilkan dalam berbagai level
- Abstraksi paling tinggi, solusi diterjemahkan dalam bentuk 'lingkungan masalah'
- Pada abstraksi yang lebih rendah, maka deskripsi solusi diberikan makin rinci



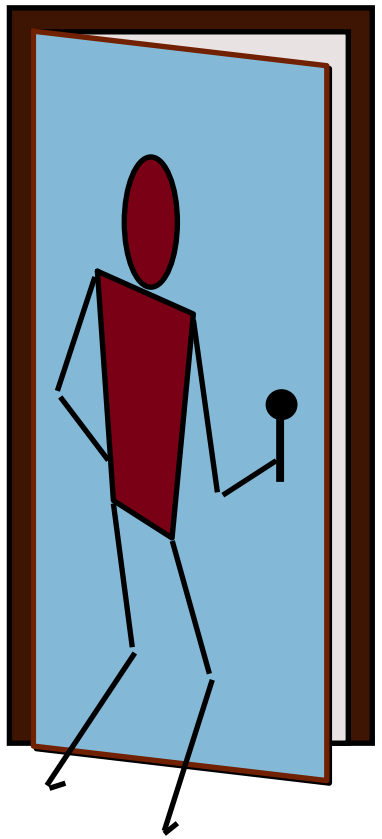
Abstraksi Data





Akan diiimplementasikan sebagai struktur data

Abstraksi Prosedural

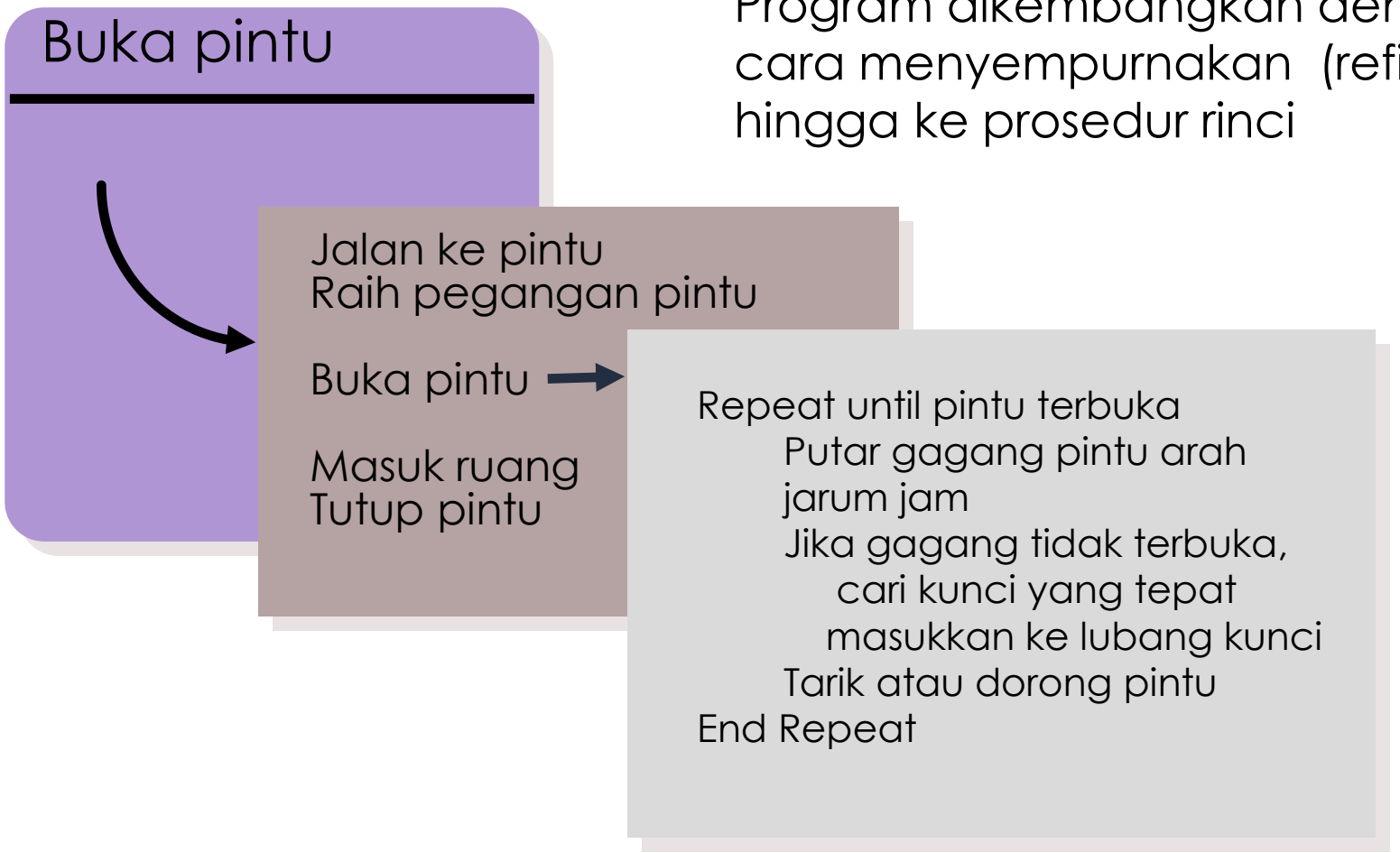


buka

Rincian buka pintu

Diimplementasikan berdasarkan 'pengetahuan' akan pintu ketika dimasuki orang

Stepwise Refinement



Program dikembangkan dengan cara menyempurnakan (refinement) hingga ke prosedur rinci

Modularitas

Apa itu Modul?

- Module:
  - "each of a set of standardized parts or independent units that can be used to construct a more complex structure, such as an item of furniture or a building" *(google's definition)*

- Modul dalam software bisa berbentuk
  - Kumpulan prosedur yang saling terkait secara fungsional
  - File yang berisi kumpulan prosedur yang terkait secara prosedur

- PL dibagi menjadi komponen yang terpisah dengan nama yang baru.
  - Komponen ini disebut modul
  - Komponen ini saling terintegrasi untuk memenuhi kebutuhan pemecahan masalah

Modul vs. Biaya

- Misalnya
  - Fungsi C(x) adalah mendefinisikan besar kompleksitas suatu masalah x dan fungsi E(x) adalah usaha (effort) untuk menyelesaikan masalah x
  - P1 dan P2 adalah program P1 dan P2

    Jika C(p1) > C(p2) maka E(p1) > E(p2) → *Artinya butuh waktu lebih lama untuk program dengan kompleksitas yang lebih tinggi*

    Hasil Eksperimen dengan penyelesaian dunia nyata **C(p1 + p2) > C(p1) + C(p2)**

    Implikasinya:

    E(p1 + p2) > E(p1) + E(p2) → *Artinya: kompleksitas masalah dari hasil gabungan p1 dan p2 adalah lebih besar dibandingkan dengan total pemecahan masalah p1 dan p2 secara terpisah.*

Jadi lebih mudah memecahkan masalah yang kompleks dengan memecahkannya menjadi bagian masalah yang lebih sederhana tetapi tetap *manageable*

Seberapa jauh harus kita pecah-pecah?

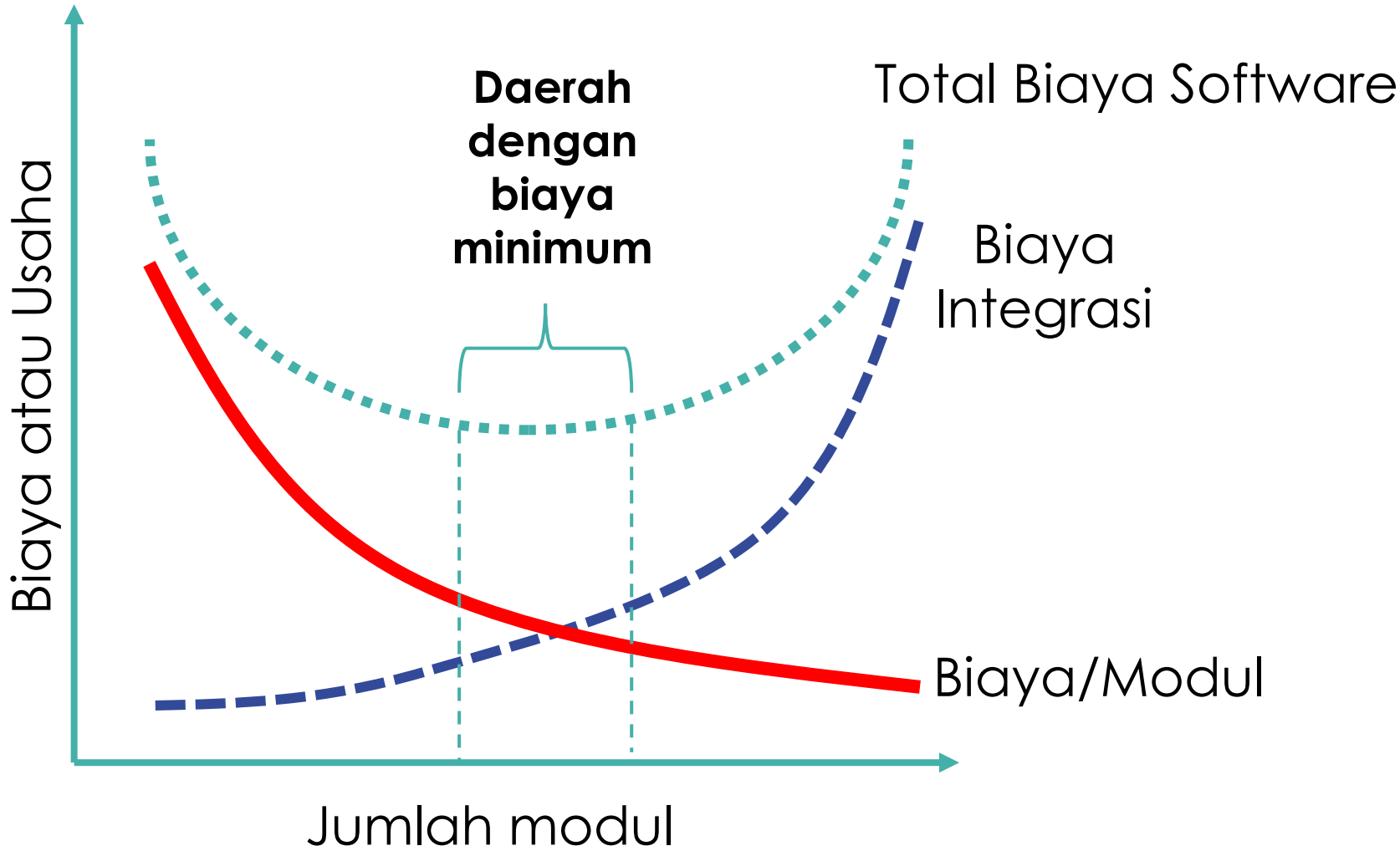
Jika kita bisa membagi suatu masalah menjadi **modul-modul lebih kecil** nampaknya akan menyebabkan biaya makin rendah,

tetapi

ternyata ada **biaya integrasi** yang harus dilakukan untuk menyatukan modul tadi menjadi program utuh

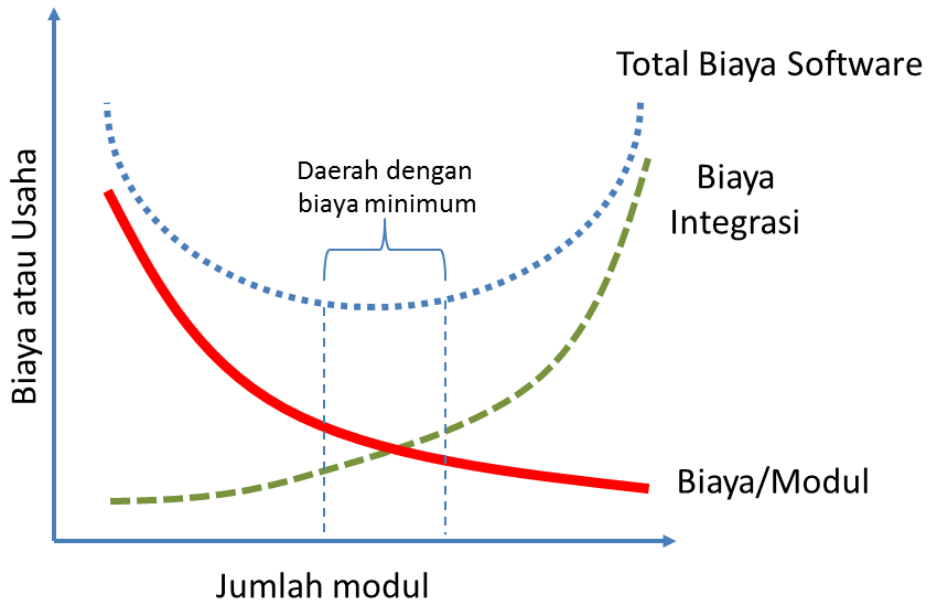
Makin banyak integrasi harus dilakukan, maka usaha akan makin besar

Modularitas dan Biaya Software



Modularitas dan Biaya Software

To Evaluate Design Method for Effective modularity



Kurva itu menunjukkan bahwa kita harus melakukan **modularisasi**, tetapi jangan sampai terjadi **Under-modularity** atau **Over-modularity**

- **Modular decomposability**
- **Modular composability**.
- **Modular understandability**
- **Modular continuity**
- **Modular protection**

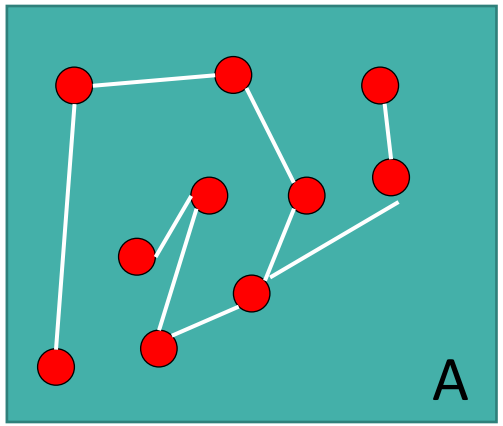
KNOWLEDGE & SOFTWARE ENGINEERING

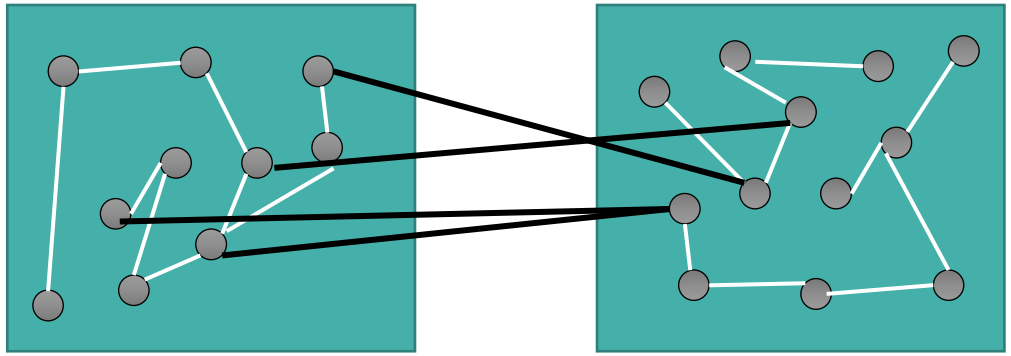
Kriteria Evaluasi Metode perancangan Modular

- **Modular Decomposability**
  - Teknik yang secara sistematis **memecah masalah** menjadi masalah yang lebih kecil
- **Modular Composability**
  - Teknik yang mendukung **reuse** akan memudahkan **pembangunan** sistem baru
- **Modular Understandability**
  - Suatu modul dapat **dimengerti** sebagai unit **individu tunggal**
- **Modular Continuity**
  - **Perubahan** kecil dimasa depan pada satu individual modul akan memberikan **dampak** (side-effect) yang **minimal**
- **Modular Protection**
  - Jika terjadi **kesalahan** dalam operasinya, maka **efek** sampingnya rendah atau dapat **diminimumkan**

Independensi Fungsional

- Kebergantungan fungsional tiap elemen dalam satu modul





- Kebergantungan fungsional antar modul

Independensi Fungsional

- Kriteria kualitatif independensi
  - **Kohesi (*Cohesion*)**
    - Ketergantungan fungsionalitas dari unit atau elemen **dalam suatu modul**
  - **Coupling**
    - Ketergantungan fungsionalitas suatu modul dengan fungsionalitas **modul lain**

> *"Suatu perangkat lunak sebaiknya dibentuk dari komponen-komponen yang memiliki **interaksi antar** komponen **serendah-rendahnya** (low-coupling) dan juga sebaliknya **interaksi dalam** komponen **setinggi-tingginya** (high-cohesion)"*

KNOWLEDGE & SOFTWARE ENGINEERING

Independensi Fungsional

- Prinsip Ideal
  - Kohesi **Tinggi**
    - Unit-unit **dalam satu modul** sebaiknya memiliki ketergantungan fungsional yang **setinggi-tingginya**
  - *Coupling* **Rendah**
    - Ketergantungan fungsionalitas **antar modul** sebaiknya **serendah-rendahnya**

Perhatikan ketergantungan fungsional pada modul A, B, C dan D

Rancangan Modul A
Fungsi Cetak Ke Printer
Fungsi Cetak Ke Layar
Fungsi Cetak ke Projektor

Rancangan Modul B
Fungsi Cetak Ke Printer
Fungsi Cetak Ke Layar
Fungsi Cetak ke Projektor
Fungsi Baca Data Mahasiswa

Kohesi A lebih tinggi dari Kohesi B atau dapat dibaca: **"Rancangan modul A lebih baik daripada Rancangan Modul B"**

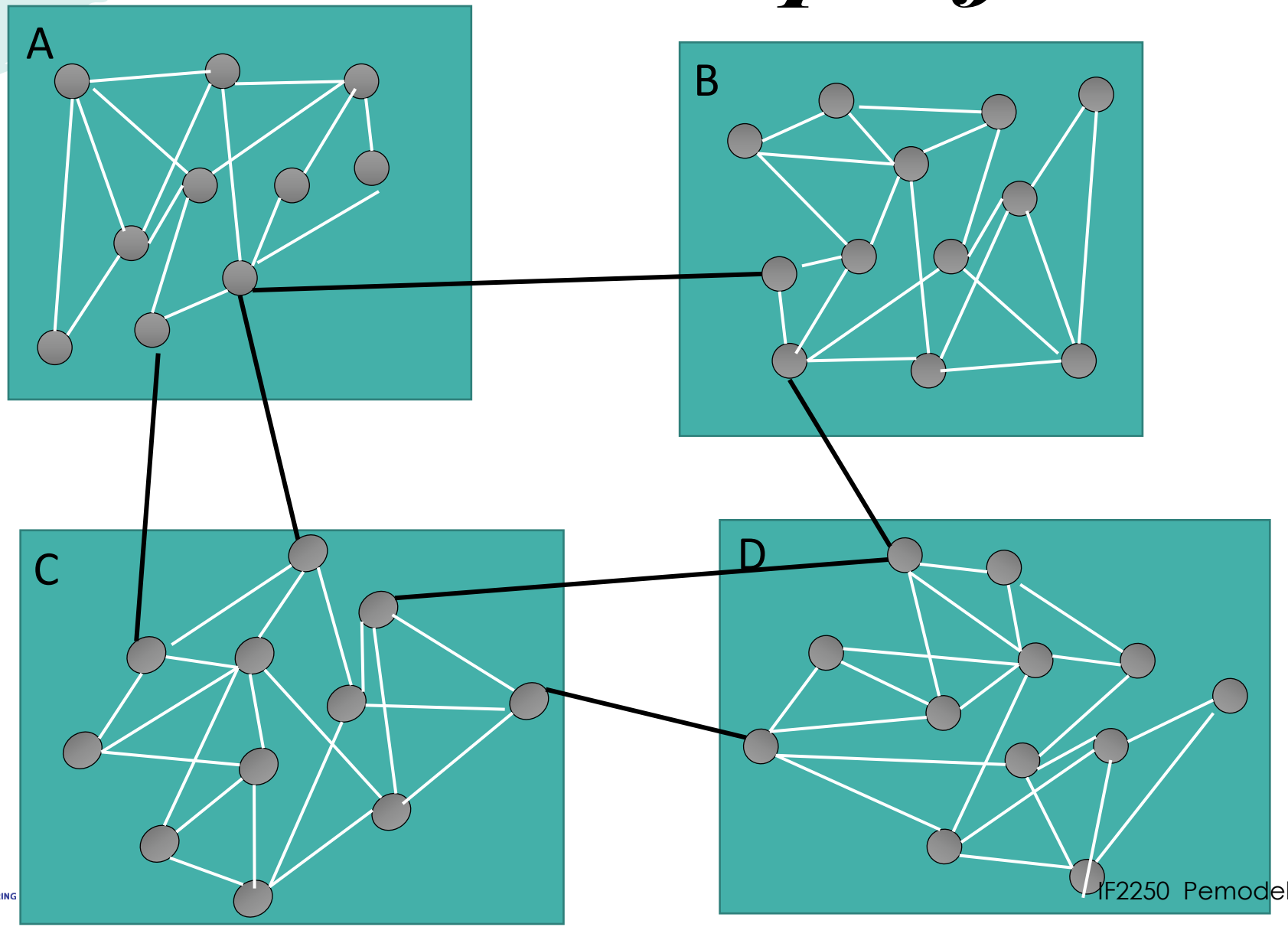
Rancangan Modul C
Fungsi Baca NIM Mahasiswa
Fungsi Baca Nama Mahasiswa
Fungsi Baca Alamat Mahasiswa
Fungsi Baca MataKuliah
Fungsi Baca Nama Dosen

Rancangan Modul D
Fungsi Baca NIM Mahasiswa
Fungsi Baca Nama Mahasiswa
Fungsi Baca Alamat Mahasiswa

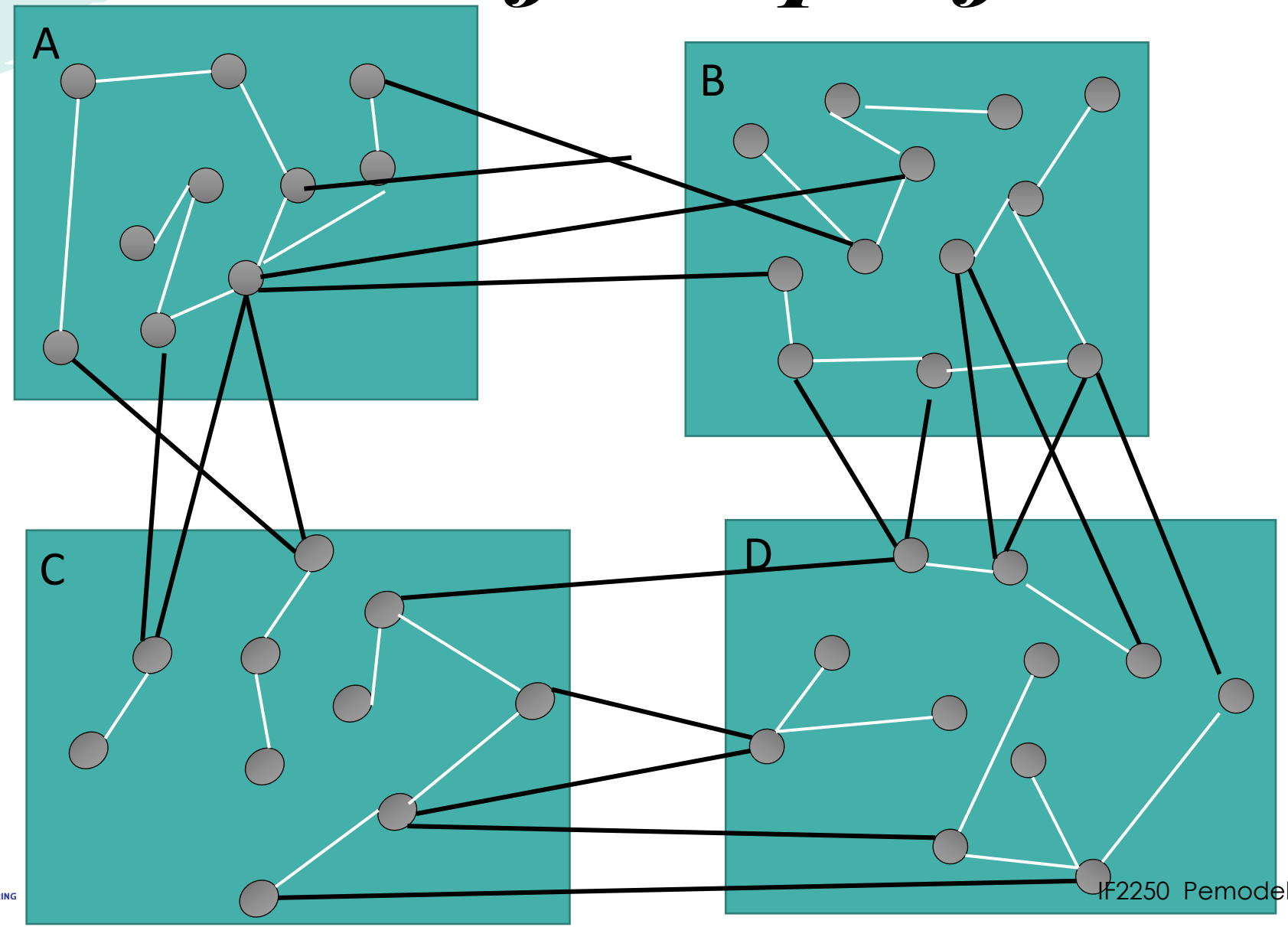
Coupling C lebih tinggi dari *coupling* D atau dapat dibaca:
"Rancangan modul D lebih baik daripada Rancangan Modul C"

KNOWLEDGE & SOFTWARE ENGINEERING

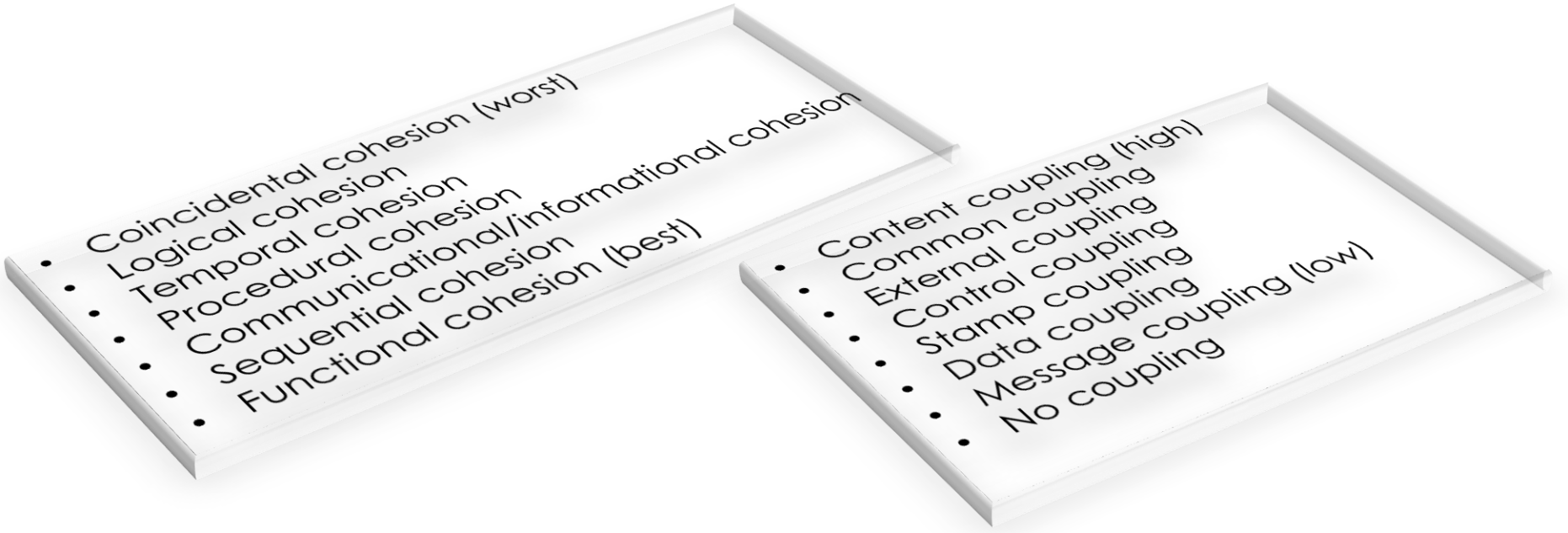
High Cohesion /Low Coupling



Low Cohesion /High Coupling



Ada berbagai macam bentuk coupling/cohesion



Bentuk Coupling

- Urutan **teratas**, adalah **coupling** yang paling **dihindari** dari pada urutan di bawahnya, *atau* makin ke **bawah** adalah **coupling** yang makin dapat **ditoleransi**:
  - Ganti kode komponen lain
  - Bercabang ke lokasi kode lain
  - Mengakses data dalam komponen lain
  - Menggunakan global data atau shared data
  - Pemanggilan prosedur/fungsi dengan switch sebagai parameter
  - Pemanggilan prosedur/fungsi dengan data parameter biasa
  - Memindahkan data stream dari satu komponen ke komponen lain

KNOWLEDGE & SOFTWARE ENGINEERING

Bentuk Coupling (1)

- Mengganti kode komponen lain
  - Hanya bisa dalam bahasa assembly atau bahasa COBOL (jaman dulu)
    - Kode program mengganti kode program dalam komponen lain.
- Bercabang ke lokasi kode lain
  - Menggunakan instruksi Goto yang bisa dilakukan dalam bahasa C ataupun bahasa assembly

Bentuk Coupling (2)

- Mengakses data dalam komponen lain
  - Mengubah isi data milik komponen lain, sedikit lebih tidak membahayakan dibandingkan mengubah kode program komponen lain, tetapi tetap tidak dianjurkan karena kesalahan bisa terjadi karena keterlibatan komponen lain.
- Menggunakan global data atau shared data
  - *Lihat penjelasan sebelumnya*

Bentuk Coupling(3)

- Pemanggilan prosedur/fungsi dengan 'switch' sebagai parameter
  - Adanya parameter kendali (control) tidak dianjurkan. Sebaiknya setiap control ditempatkan sebagai metode/prosedur terpisah.
- Contoh:

```
void Cetak( int perintah, char* Kata, int Bilangan)
{
   switch( perintah)
   { case 1: printf("cetak 1: %s", Kata); break;
     case 2: printf("cetak 2: %d", bilangan); break;
     otherwise: printf("none");
   }
}
```

*Cara yang **tidak** disarankan…*

Cara yang disarankan…

```
void CetakKata(char* Kata)
{
   printf("cetak 1: %s", Kata);
}
```

```
void CetakBilangan(int Bilangan)
{
   printf("cetak 2: %d", Bilangan);
}
```

KNOWLEDGE & SOFTWARE ENGINEERING

Bentuk Coupling (4)

- Pemanggilan prosedur/fungsi dengan data parameter biasa
  - Ini bentuk coupling yang 'boleh' atau 'ideal'
  - Interaksi antar prosedur/fungsi dilakukan dengan jumlah parameter yang seminimal mungkin
- Memindahkan data stream dari satu komponen ke komponen lain
  - Misalnya keluaran suatu prosedur direkam ke suatu file keluaran, dan hasil ini dibaca oleh prosedur lain. Jadi tidak ada interaksi langsung terjadi antar modul.
    - Perpindahan data melalui media file (stream) lain.

Bentuk Cohesion

1. COINCIDENTAL
2. LOGICAL
3. TEMPORAL
4. COMMUNICATIONAL
5. FUNCTIONAL

KNOWLEDGE & SOFTWARE ENGINEERING

I. Coincidental Cohesion

- Coincidental
  - Elemen-elemen dari metode terbentuk secara **kebetulan** (tanpa direncanakan)
  - Tidak ada keterhubungan antar elemen

2. Logical Cohesion

- Suatu procedure/fungsi melakukan **sekumpulan fungsi** yang **sama secara logika**
- Contoh: ketika kita melakukan perancangan, kadang kita mengidentifikasi adanya aktivitas keluaran dari sistem dan menggabungkannya menjadi satu metode.
  - Contoh: Fungsi CetakSemua(); /* isinya mencetak dengan berbagai media berbeda */
  - Procedure/Fungsi ini berisi multi-fungsi, karena isinya meliputi aktivitas seperti
    - Cetak teks ke layar
    - Cetak teks ke printer
    - Simpan teks ke file
- Prosedur/fungsi ini nampaknya rasional, bahkan secara logika benar.
- Contoh lain: fungsi calculate yang menghitung (log, sinus, cosinus)
- Masalah dengan Logical Cohesion adalah fungsi yg 'multifungsional'
  - Prosedur/fungsi tsb akan **melakukan beberapa aksi** dan bukan satu aksi tunggal. Prosedur/Fungsi ini menjadi kompleks padahal tidak seharusnya kompleks. Jika kita ingin **memperbaiki salah satu aksi** di dalamnya, maka akan sulit untuk tidak **memeriksa juga elemen lain** (akibatnya proses pengujian dan maintenance juga akan lebih kompleks).

3. Temporal Cohesion

- Prosedur/Fungsi yang melakukan **sekumpulan aksi** yang hubungannya adalah **hanya** karena harus dilakukan **bersama-sama**.
- Contoh berikut berisi temporal cohesion

```
ClearScreen();
Openfile();
Total = 0;
```

- Sekumpulan urutan ini sering dilakukan pada banyak program yang kadang tidak bisa dihindari. Tetapi seperti pada contoh di atas, **masing-masing tidak memiliki keterhubungan fungsional**.
- Solusinya adalah dengan membuat metode inisialisasi yang terdiri dari pemanggilan secara berurutan seperti pada contoh berikut:

```
Initialize_terminal()
Initialize_files()
Initialize_calculation()
```

- Pada pemrograman OO, inisialiasi dilakukan bersamaan saat **penciptaan objek**.
  - Metode constructor harus dieksekusi untuk melaukan proses inisialisasi dari suatu objek.
  - Konstruktor dibuat sebagai bagian dari suatu kelas dan memiliki tanggung jawab yang khusus.

KNOWLEDGE & SOFTWARE ENGINEERING

4. Communicational Cohesion

- Fungsi-fungsi dari suatu modul yang melakukan **aksi pada data yang sama** dikelompokkan bersama. Misalnya:
  - Serangkaian prosedur/fungsi yang menampilkan dan mencatat log suhu
  - Sekumpulan prosedur/fungsi yang memformat dan mencetak suatu angka
- Kohesi komunikasi ini dapat dijelaskan dengan dilakukannya beberapa aksi pada suatu benda
  - Kelemahan dengan cara ini, adalah makin kompleks padahal dapat dihindari.
  - Setiap aksi sebenarnya dapat dibedakan sebagai prosedur/fungsi yang berbeda.

KNOWLEDGE & SOFTWARE ENGINEERING

5. Functional Cohesion

- Beberapa bagian modul yang berbeda dikelompokkan karena modul-modul ini **melakukan suatu task** yang sudah **terdefinisi jelas**
- Ini adalah bentuk kohesi yang paling baik. Suatu metode dengan functional cohesion melakukan aksi tunggal pada suatu subjek.
  - 1 aksi dan 1 objek (yang dikenakan aksi)
- Contoh:

```
HitungRataRata();
CetakHasil();
InputTransaksi();
```

Arsitektur Perangkat Lunak

KNOWLEDGE & SOFTWARE ENGINEERING

Arsitektur Perangkat Lunak

Definisi

"The overall structure of the software and the ways in which that structure provides conceptual integrity for a system." [SHA95a]

Properti Struktural. Design akan melibatkan komponen-komponen yang saling terhubung dan saling berinteraksi dalam sistem.

Properti Fungsi-Ekstra (*Extra-functional properties*). Deskripsi rancangan memasukkan bagaimana kebutuhan arsitektur sistem terhadap kualitas dari sistem (performansi, capacity, reliability, keamanan, adaptability dll)

Kumpulan sistem terkait (***Families of related systems***)**.** Deskripsi rancangan melibatkan pola-pola berulang yang sering ditemui pada sistem yang mirip. Jadi perancangan sebaiknya memiliki kemampuan reuse.

Bentuk- bentuk Arsitektur
(Architectural Styles)

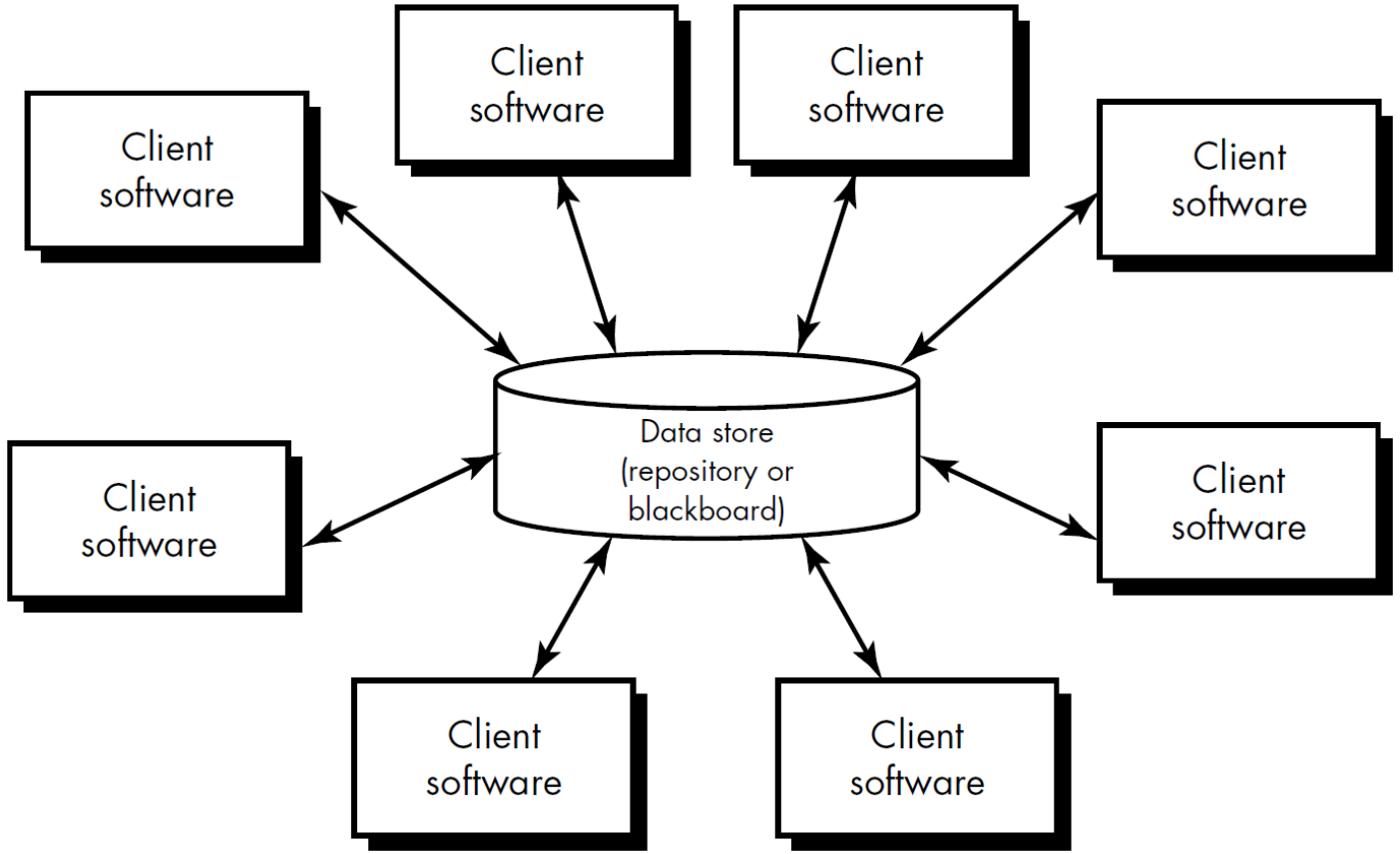
1. ***Data-centered architectures***
   - Fokus pada data
2. ***Data flow architectures***
   - Fokus pada aliran data
3. ***Call and return architectures***
   - Fokus pada pemanggilan fungsi dan return values
4. ***Object-oriented architectures***
   - Fokus pada objek
5. ***Layered architectures***
   - Arsitektur berlapis

KNOWLEDGE & SOFTWARE ENGINEERING

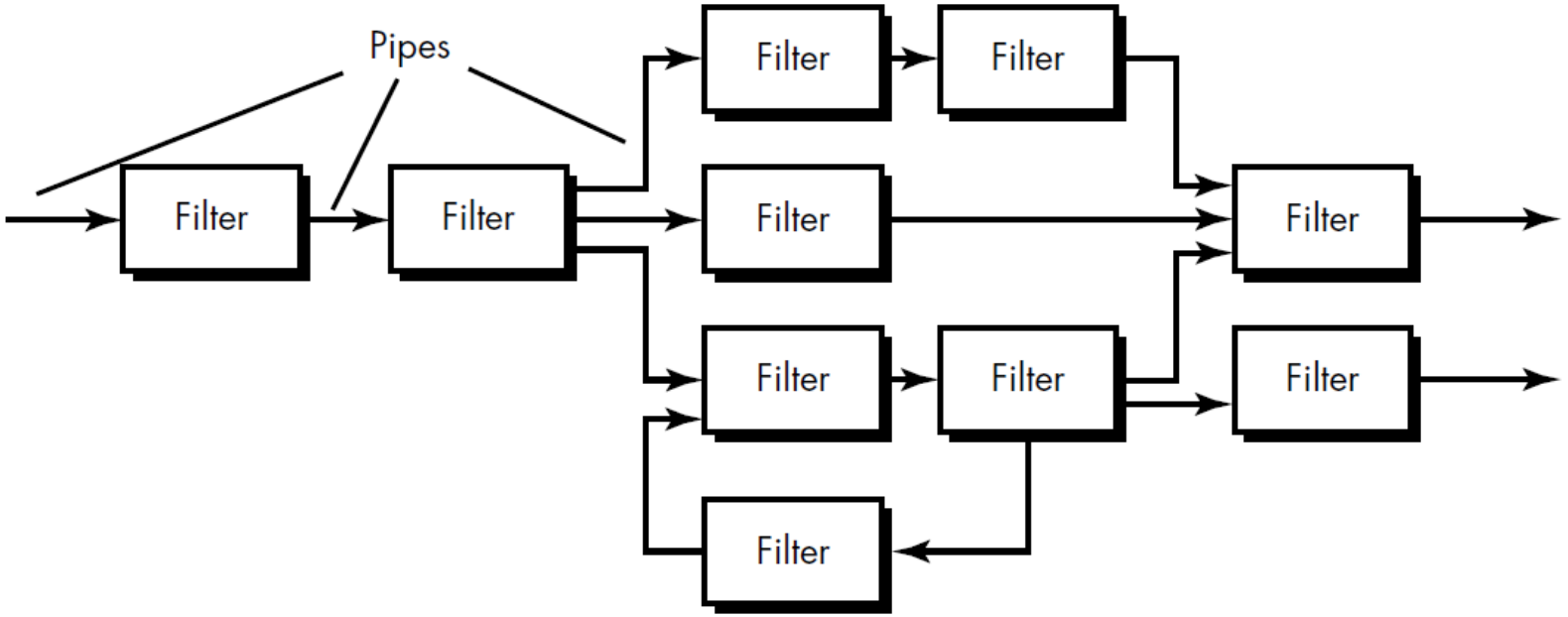
Karakteristik Setiap Bentuk Arsitektur

- Kumpulan komponen yang melakukan suatu **fungsi/peran** yang dibutuhkan dari suatu sistem
  - Komponen ini bisa berupa basisdata, modul procedure/function, class/objects
- Kumpulan penghubung yang memungkinkan "Komunikasi", "Koordinasi" dan "Kooperasi" antar komponen
  - Jelaskan perbedaannya!
- **Batasan (Constraint)** yang mendefinisikan bagaimana komponen saling berintegrasi membentuk sistem
- Model "**Semantik**" yang memungkin perencana mengerti properti keseluruhan dari suatu sistem
  - Dengan menganalisis semua properti yang menjadi elemen-elemen dari arsitektur

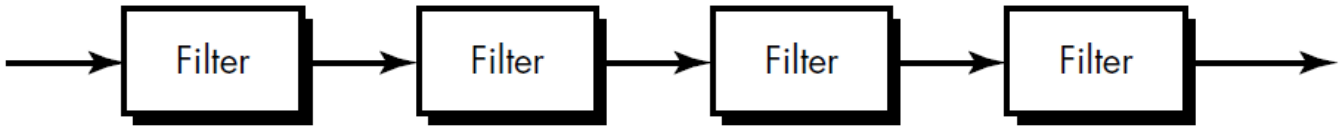
1. Data-Centered Architecture



2. Data Flow Architecture

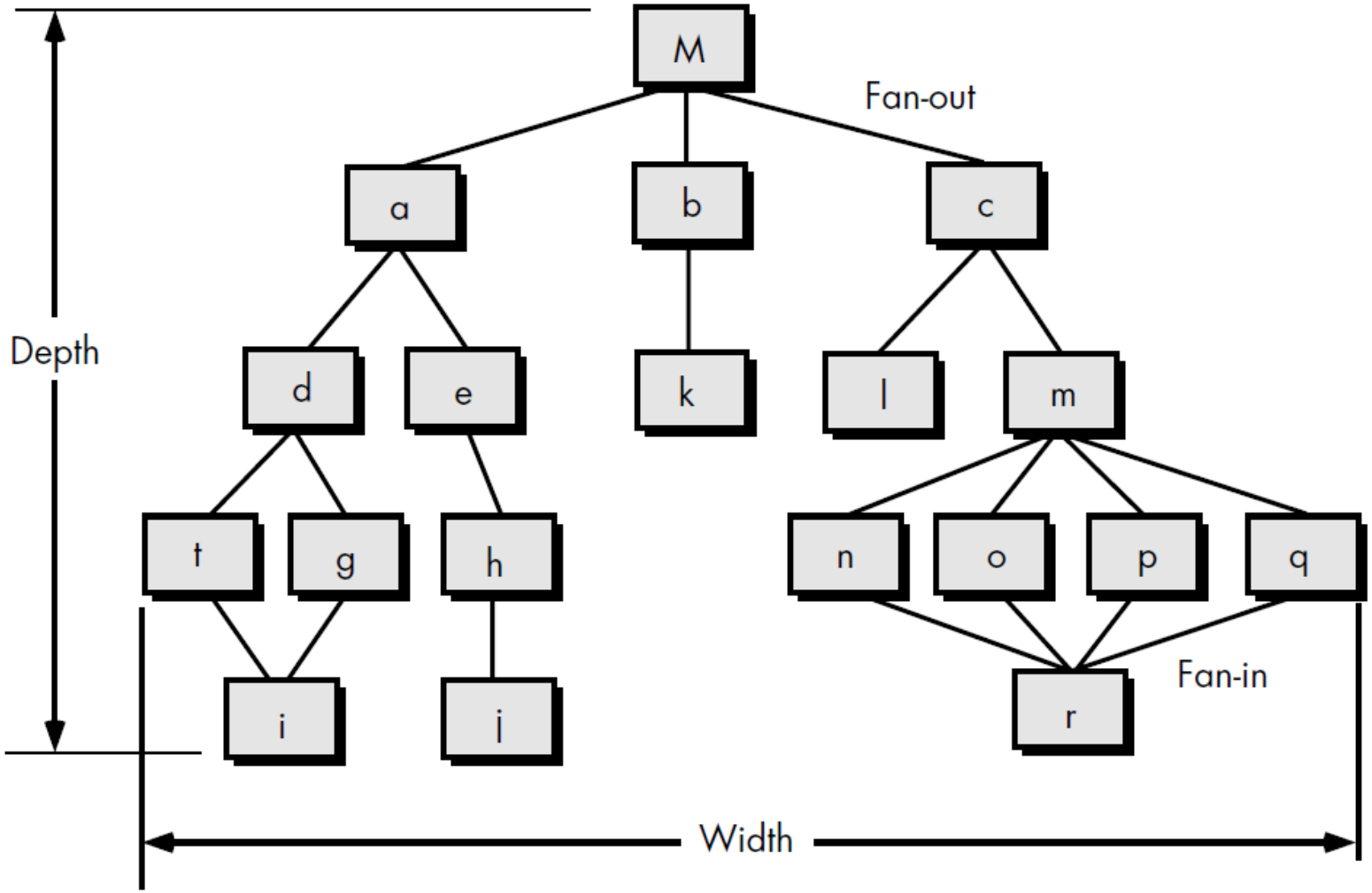


(a) Pipes and filters

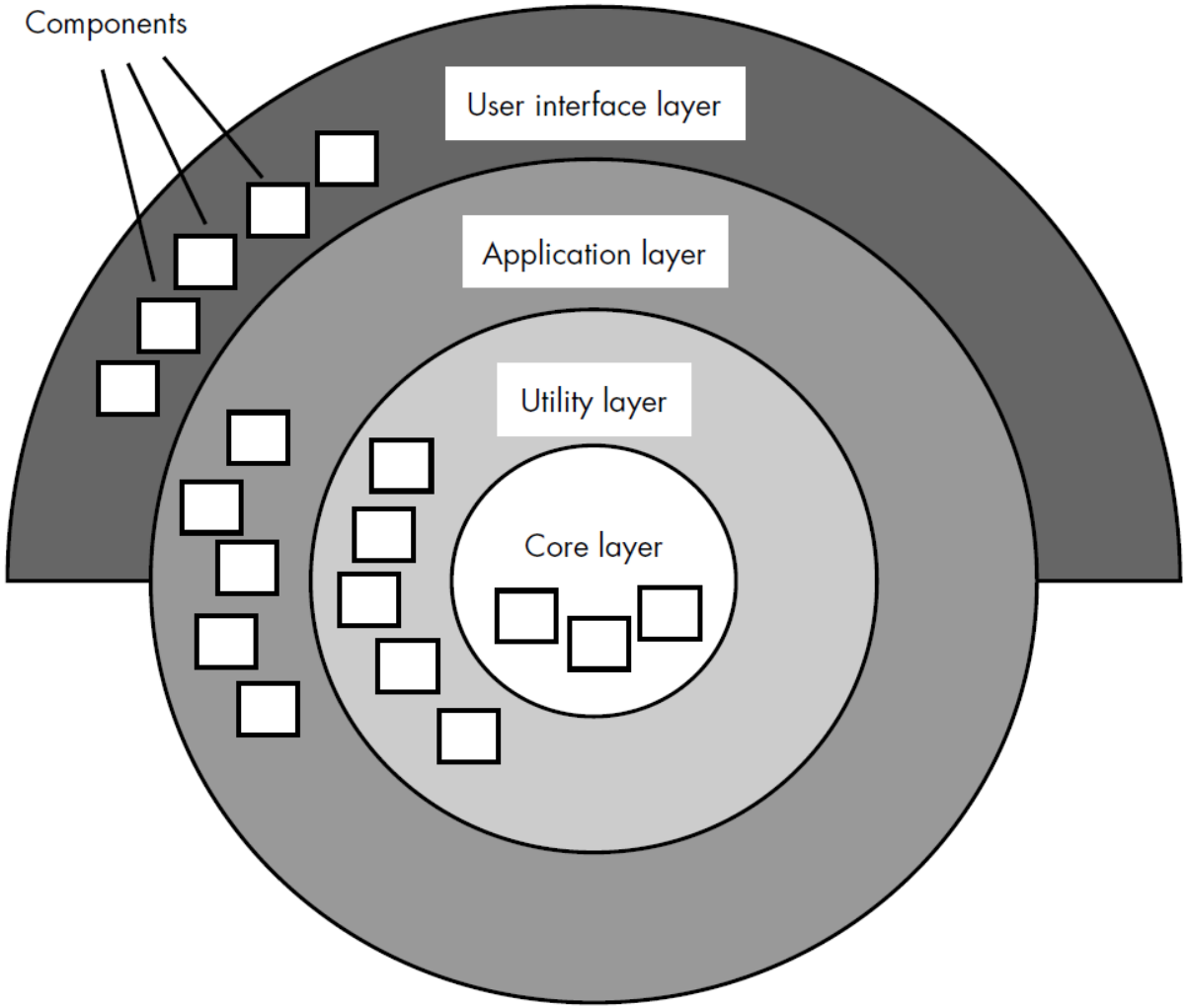


(b) Batch sequential

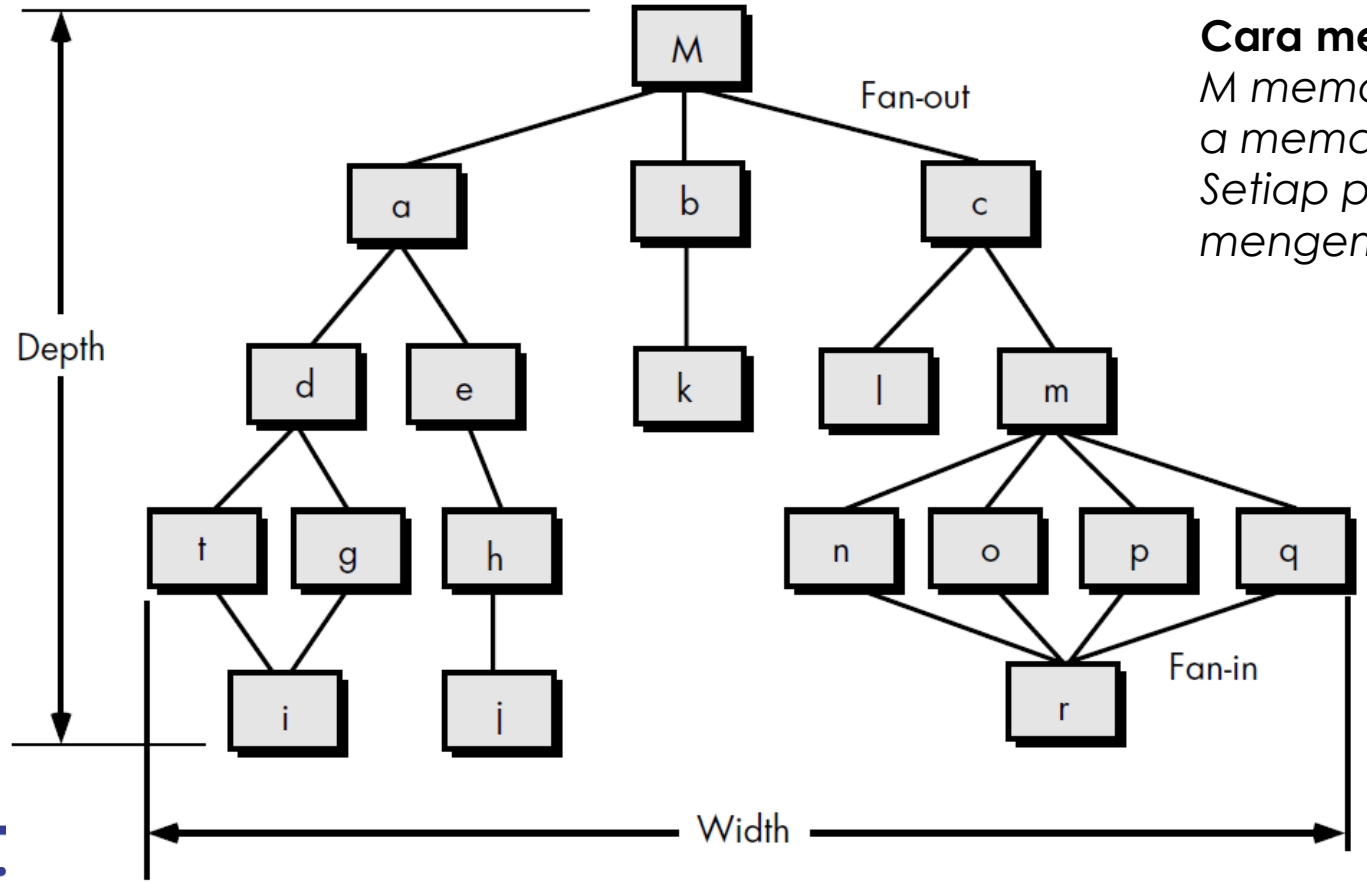
3. Call and Return Architecture



5. Layered Architecture

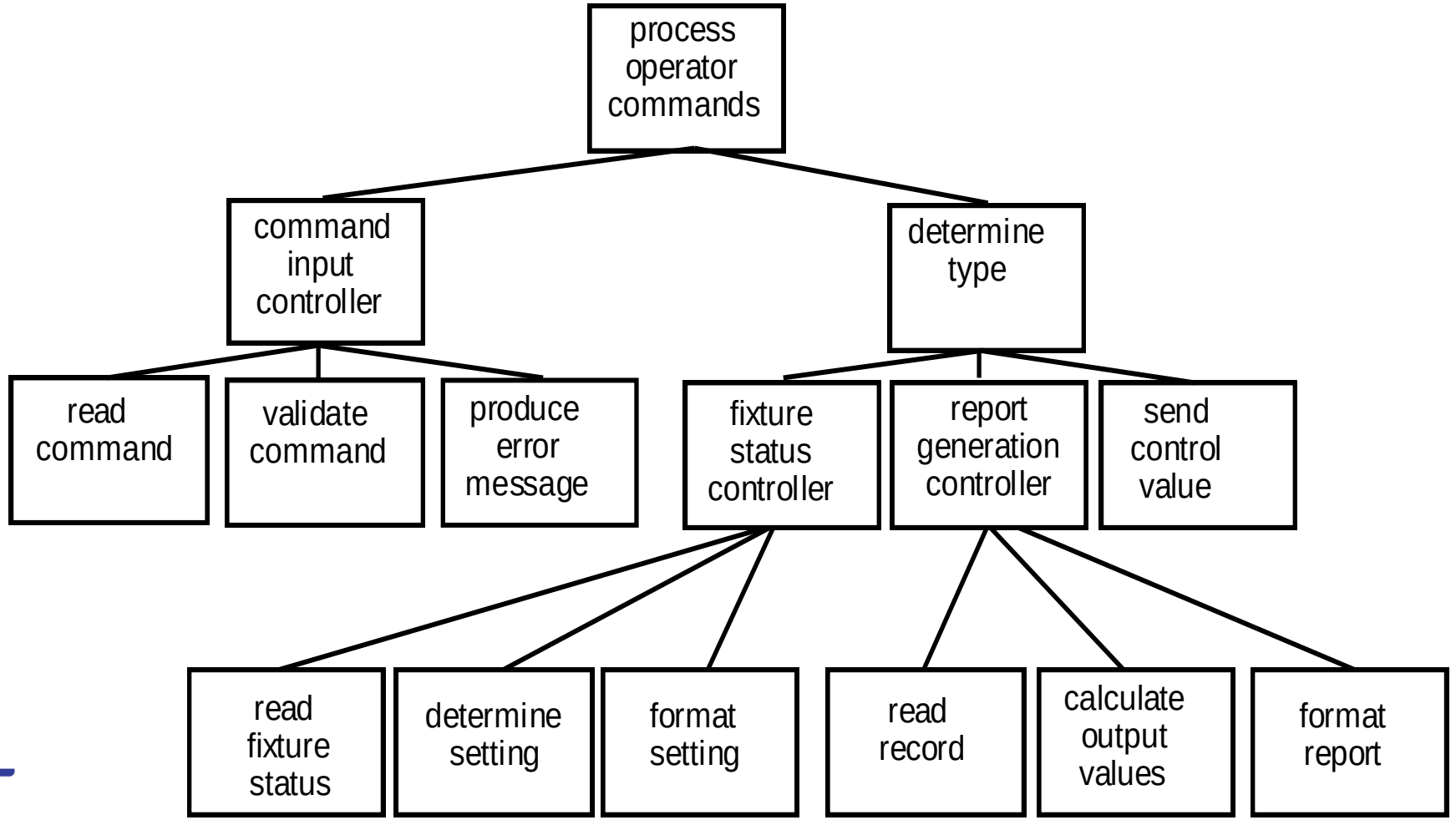


Contoh Structure Chart



Cara membaca:
M memanggil a,
a memanggil d, dst
Setiap pemanggilan mungkin mengembalikan nilai (return value)

Structure Chart (Diagram Terstruktur)



KNOWLEDGE & SOFTWARE ENGINEERING

Partisi Arsitektur

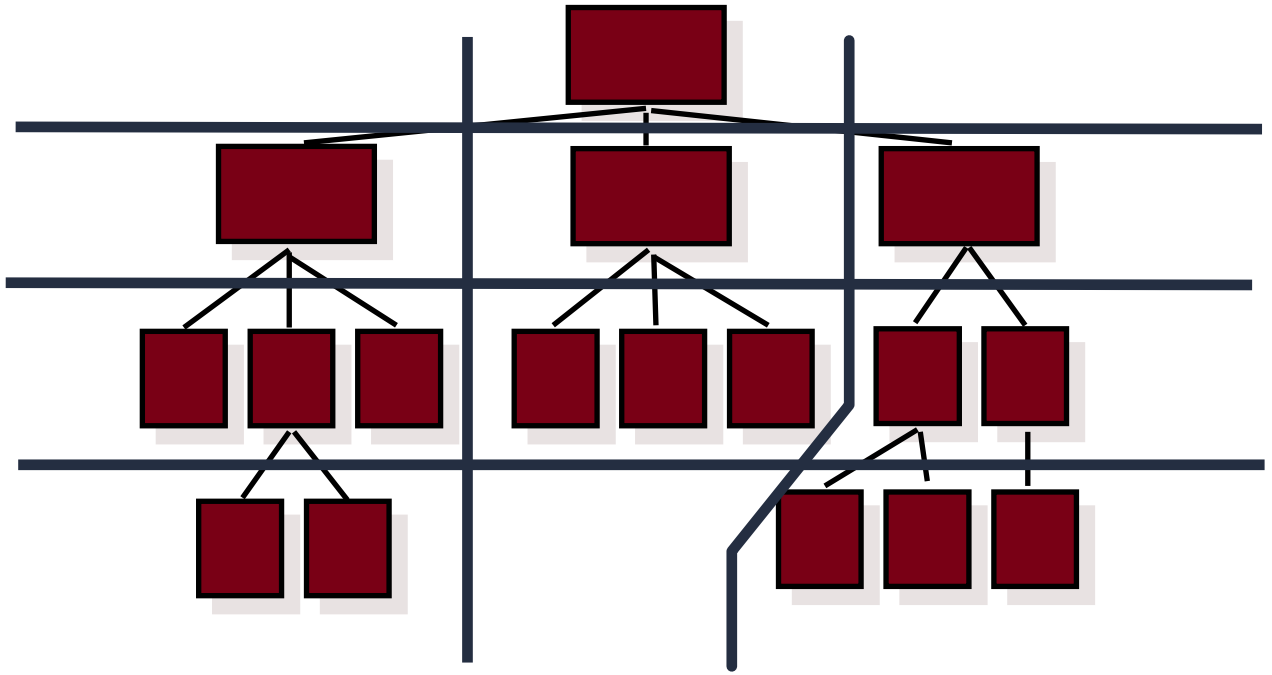
KNOWLEDGE & SOFTWARE ENGINEERING

Arsitektur dipartisi agar…

- Software akan lebih mudah diuji
- Software akan lebih mudah dirawat ('maintain')
- Efek propagasi kesalahan berkurang
- Software lebih mudah ditambah kurang modulnya.

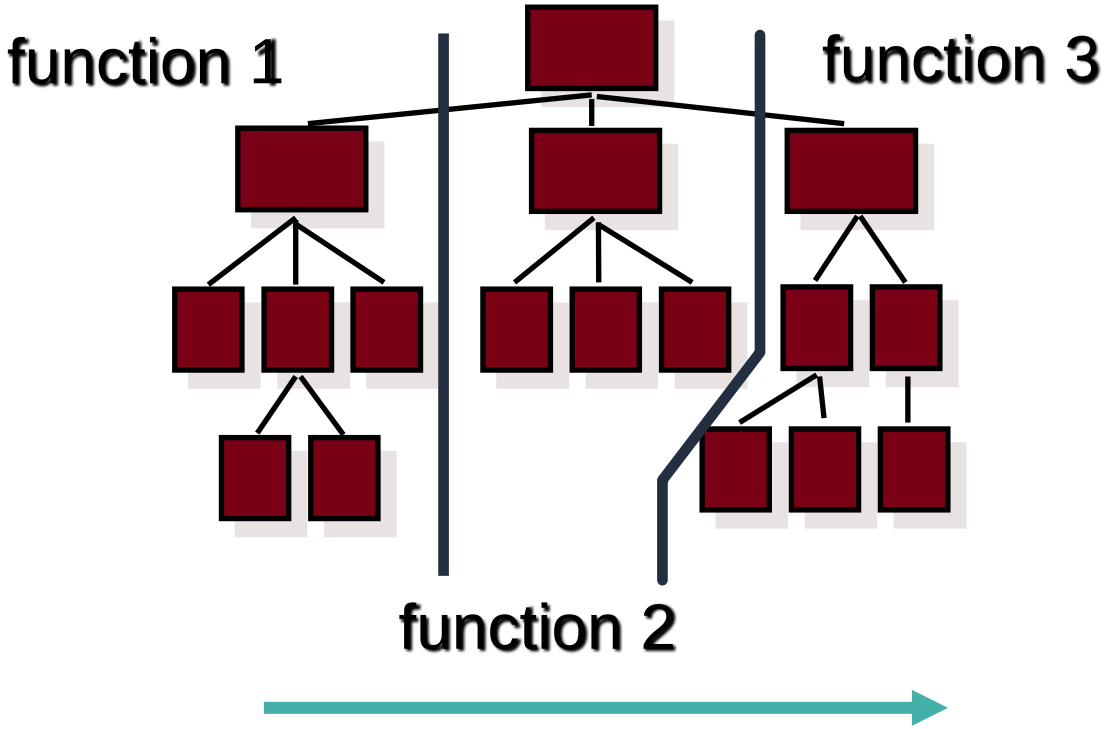
Partisi Arsitektur

- Partisi ‘horizontal’ dan ‘vertikal’



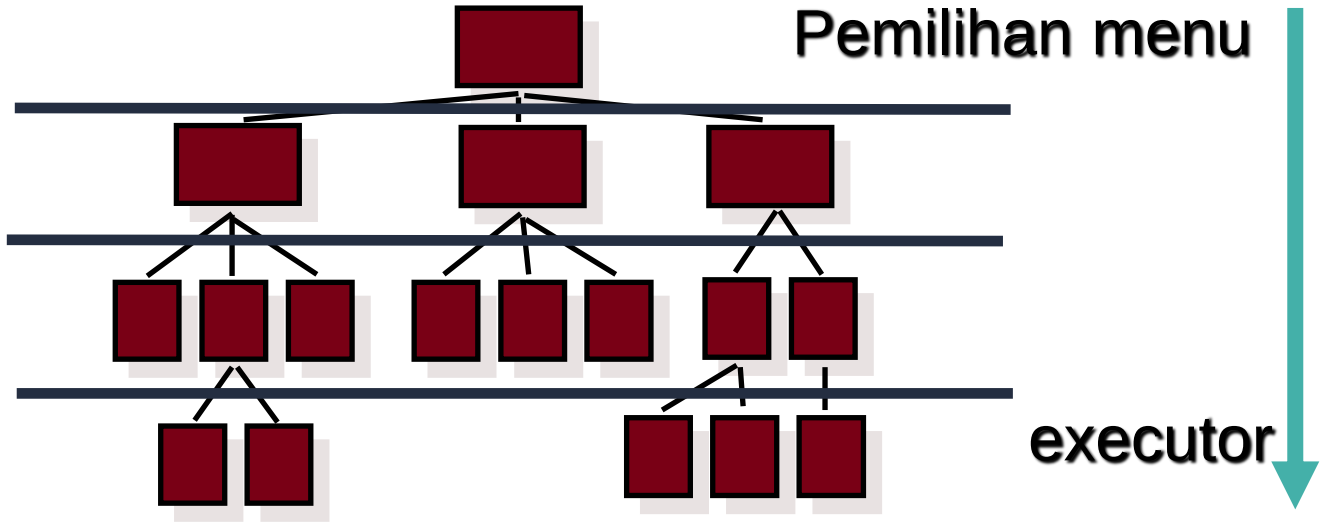
Partisi arah Horizontal (horizontal partitioning)

- Buat cabang terpisah dari hirarki modul untuk setiap fungsi utama
- Buat modul penghubung untuk kordinasi antar fungsi



Partisi Vertikal atau Factoring

- Perancangan yang membagi menjadi bagian level atas dan bagian bawah
- Bagian atas biasanya berbentuk modul-modul yang sifatnya untuk pengambilan keputusan (misalnya pemilihan menu), dan bagian bawah bagian 'pekerja' nya (fungsi yang melakukan pekerjaan yang lebih spesifik)



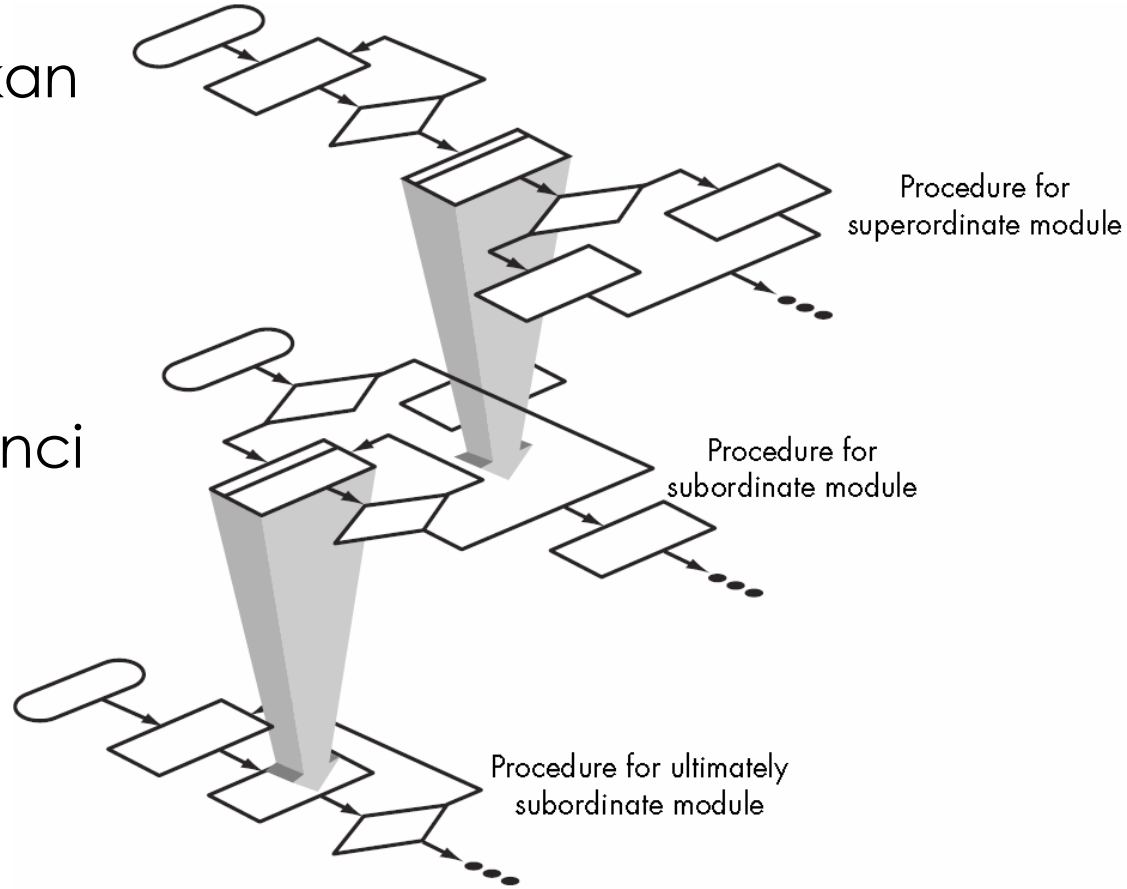
Struktur Data

- Struktur data: definisi
  - Representasi dari **hubungan logis** antar elemen-elemen data
- Struktur data sudah digunakan untuk merepresentasikan berbagai masalah, misalnya
  - Susunan suatu organisasi
    - Struktur data siswa: nim, nama, alamat, tgl lahir,
  - Alternatif informasi, dan lain-lain, tergantung perancang
- Bentuk representasinya:
  - Array dalam bentuk vektor, n-dimensi, dll
  - Linked List
  - Stack, Queue

KNOWLEDGE & SOFTWARE ENGINEERING

Procedure

- Struktur program mendefinisikan struktur kendali tanpa memperhatikan urutan pemrosesan dan titik pencabangan
- Procedure fokus pada rincian pemrosesan untuk setiap modul
- Procedure memberikan spesifikasi rinci untuk suatu pemrosesan
  - Kumpulan event
  - Titik pencabangan
  - Operasi yang berulang
  - Organisasi data dan struktur



KNOWLEDGE & SOFTWARE ENGINEERING

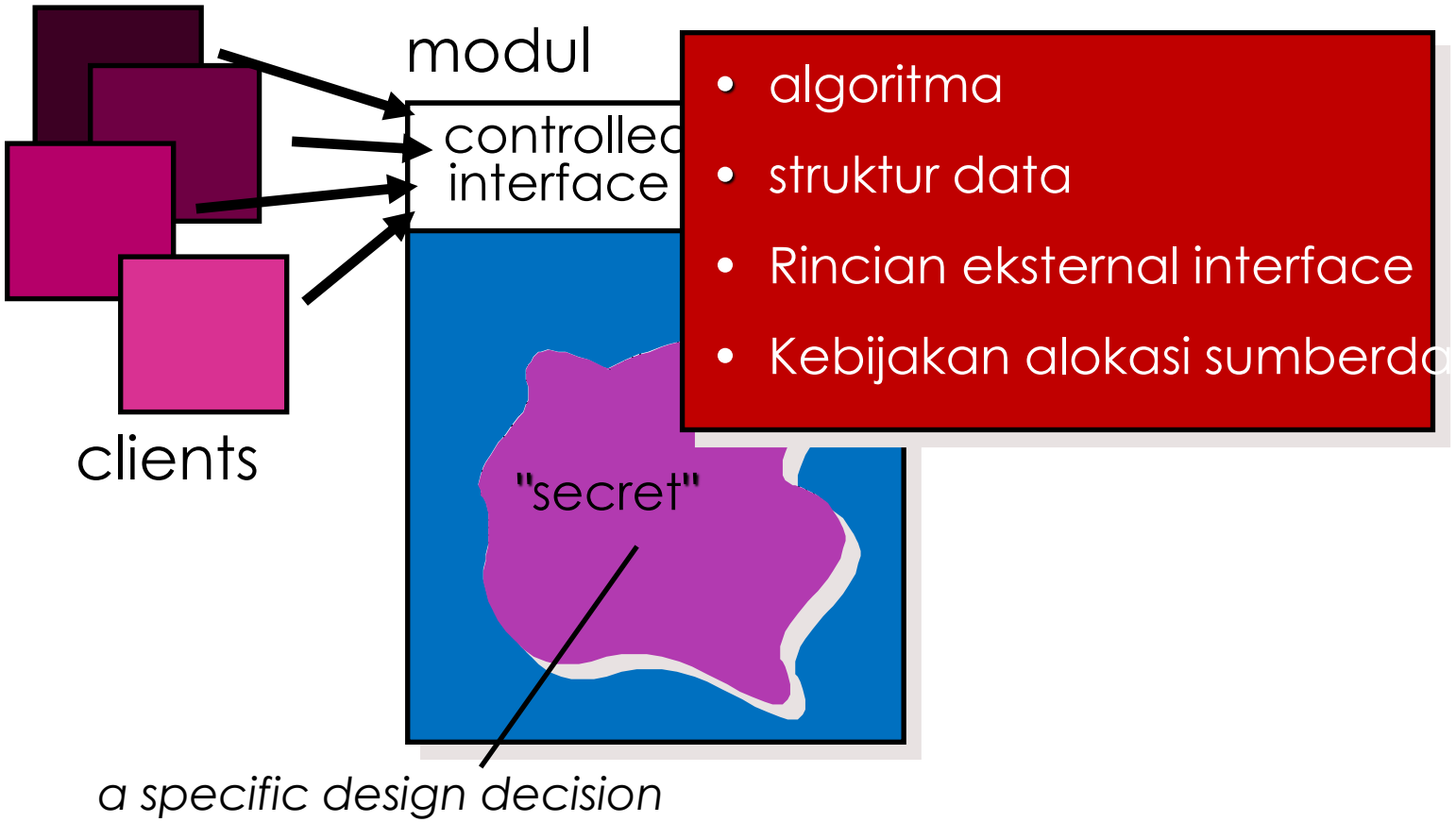
Information Hiding

DIKENAL JUGA SEBAGAI

- DATA HIDING
- ENCAPSULATION

KNOWLEDGE & SOFTWARE ENGINEERING

Information Hiding



Kenapa perlu 'Information Hiding'

- **Mengurangi** kemungkinan '**efek samping**'
  - Tidak terjadi dampak global bila dilakukan **perubahan lokal**
- **Menekankan komunikasi** melalui manajemen interface
- **Mengurangi** pemakaian **data global**
- Mengarah ke ide '**encapsulation**'
  - Encapsulation adalah salah satu atribut dari perancangan yang berkualitas bagus

Data Lokal vs. Data global

- **Hindari data global** (atau shared data)
  - Jika ada tiga modul program A, B, dan C yang mengakses variabel global X, kalau kita ingin mempelajari perilaku X, maka kita harus pelajari modul A, B dan C sekaligus.
    - Shared data adalah data yang hanya bisa diakses oleh dua atau lebih modul berbeda.
- **Gunakan data lokal**
  - Mudah dipelajari dalam lingkup satu modul tunggal
  - Dengan data lokal, suatu modul akan lebih dipisahkan untuk dipakai kembali (reuse) di keperluan berbeda.
- **Data Global** kadang tidak bisa dihindari dalam perancangan, tetapi jumlahnya **seminimal mungkin**

KNOWLEDGE & SOFTWARE ENGINEERING

Struktur Umum Information Hiding

- Dalam satu modul atau komponen,
  - ... berisi **struktur data tunggal** yang menjelaskan modul atau sebagai atribut dari modul
  - … berisi statement yang **mengakses struktur data** itu
  - … berisi statement yang **mengubah struktur data** itu
- Data dalam modul **tidak dapat diakses** secara **langsung**
  - **Akses** dilakukan melalui suatu **metode khusus**
- Implementasi
  - Dikenal sebagai ADT (Abstract Data Type)
  - Class dalam pendekatan Object Oriented (OO)

KNOWLEDGE & SOFTWARE ENGINEERING

Contoh: Stack (tumpukan)

- Metode yang disediakan adalah Push, Pop, IsEmptyStack, atau..
- Pengguna stack **tidak disarankan** untuk **mengakses langsung** ke elemen stack
- Dengan demikian maka pengembang stack akan **bebas mengimplementasikan** stack baik dengan array ataupun linked list.

Keuntungan Information Hiding

- Mudah diperbaiki
  - Perbaikan hanya pada **satu modul** atau **satu komponen** atau **satu unit** saja
- Pengembangan yang independen (ketergantungan yang rendah)
  - Programmer/pengembang **tidak tergantung** pada **modul lain**
  - **Komunikasi** antar modul dilakukan lewat **interface**
- Mudah dimengerti (*Comprehensibility*)
  - Kemudahan dimengerti ini akan menguntungkan dari sisi perancangan (dan review), pengujian (*testing*) dan perawatan (*maintenance*) untuk suatu modul/komponen tunggal

Pola (Pattern)

- Suatu pola perancangan (*design pattern*) menjelaskan **struktur perancangan** untuk memecahkan suatu masalah perancangan pada **suatu konteks**
  - Yang dapat memberikan dampak hasil aplikasi pola tersebut
- Tujuan setiap pola perancangan adalah memberikan deskripsi yang memungkinkan perancang menentukan:
  - Apakah suatu pola **cocok** untuk diaplikasikan pada **kasus** tertentu
  - Apakah suatu pola dapat **di-*reuse***
  - Apakah suatu pola dapat menjadi **panduan** untuk pengembangan pola perancangan yang **mirip** tetapi berbeda struktur pola fungsional/struktural.

Separation of Concern

- *Separation of concerns* adalah konsep perancangan yang menyarankan bahwa setiap masalah yang kompleks dapat lebih **mudah ditangani** jika dibagi menjadi **lebih kecil** agar dapat **dipecahkan** secara **independen**.
- *Concern* adalah **fitur** atau **perilaku** (*behavior*) yang menjadi bagian dari model kebutuhan PL.
- Dengan memecah *concerns* menjadi lebih kecil dan *managable*, maka masalah akan dapat dipecahkan dalam **usaha** dan **waktu** se**singkat**nya .

Aspect

- Setelah **analisis kebutuhan** dilakukan, sekumpulan '**concerns**' akan bisa **ditemukan**
  - Concerns ini meliputi kebutuhan, use-case, fitur, struktur data, masalah 'quality-of-service', variant, intellectual property boundaries, kolaborasi, pola dan kontrak
- Ketika perancangan dimulai, **kebutuhan** akan diperbaiki menjadi representasi **perancangan** yang lebih **modular**
  - Jika ada dua kebutuhan, A dan B. Kebutuhan A crosscuts kebutuhan B jika hasil dekomposisi tadi menyebabkan B tidak bisa dipenuhi jika A tidak dilibatkan.
  - **Aspek** adalah representasi dari suatu **concern** yang **crosscutting**.

Refactoring

KNOWLEDGE & SOFTWARE ENGINEERING

Refaktor (Refactoring)

- Fowler [FOW99] mendefinisikan refactoring sbb:
  - "Refactoring is the **process of changing** a software system in such a way that it **does not alter the external behavior** of the code [design] yet **improves** its **internal structure**."
- Kalau PL direfaktor, maka perancangan akan diperiksa terhadap
  - **Redundansi** (duplikasi yang tidak perlu)
  - Elemen **perancangan** yang **tidak pernah digunakan**
  - **Algoritma** yang **tidak efisien** atau **tidak perlu**
  - **Struktur data** yang **jelek konstruksi**nya atau bahkan tidak cocok penerapannya
  - Atau setiap **perancangan** yang harus **diperbaiki** untuk menghasilkan perancangan yang **lebih baik**.

Refactoring: Hilangkan Goto

- Perintah Goto masih dikenal dalam bahasa C dan turunannya
  - Hilangkan perintah ini dengan menggunakan teknik pemrograman terstruktur, dengan memperhatikan struktur kendali dari program (loop, kondisi pencabangan)

Refactoring: Hilangkan Kode Yang Sama

- Kode program yang sama mungkin muncul di banyak tempat
  - Perbaikan dilakukan dengan membuat method (procedure/fungsi) baru
  - Ganti kode yang sama dengan pemanggilan method

Refactoring: Metode yang Panjang

- Suatu method yang terlalu panjang akan sulit di mengerti, sulit diubah dan diguna-ulang (*reuse*)
- Suatu method mungkin memiliki jumlah baris yang panjang
  - Seberapa panjang? Tidak ada aturan yang standard. Panduannya kira-kira lebih dari 20 baris mungkin perlu diperiksa kalau memang bisa dipecah. Dibawah 10 baris biasanya di rekomendasikan
  - Caranya: dengan memecah menjadi lebih dari satu metode.

Refactoring: Perintah "Switch"

- Hati-hati dengan pernyataan 'switch' (case-of)
  - Switch kadang berisi logika untuk instans yang berbeda dari class yang sama
  - Dalam OO, biasanya mengindikasikan perlunya sub-class yang baru
- Kadang struktur switch ini muncul di beberapa tempat
- Cara perbaikan:
  - Buat sub-kelas baru
  - Bagian blok dalam case nya di pindahkan ke metode baru dalam sub-kelas baru.

KNOWLEDGE & SOFTWARE ENGINEERING

Refaktoring jika:

- Ada duplikasi kode
- Suatu kode terlalu panjang
- Loop terlalu lama atau loop dalam loop dalam loop…
- Suatu kelas punya kohesi rendah
- Suatu kelas terlalu banyak coupling
- Level abstraksi tidak konsisten
- Terlalu banyak parameter
- Perubahan di satu tempat mempengaruhi tempat lain
- Modifikasi hirarki inheritansi berjalan secara paralel
- Pengelompokan data yang berulang

KNOWLEDGE & SOFTWARE ENGINEERING