Catatan Kuliah

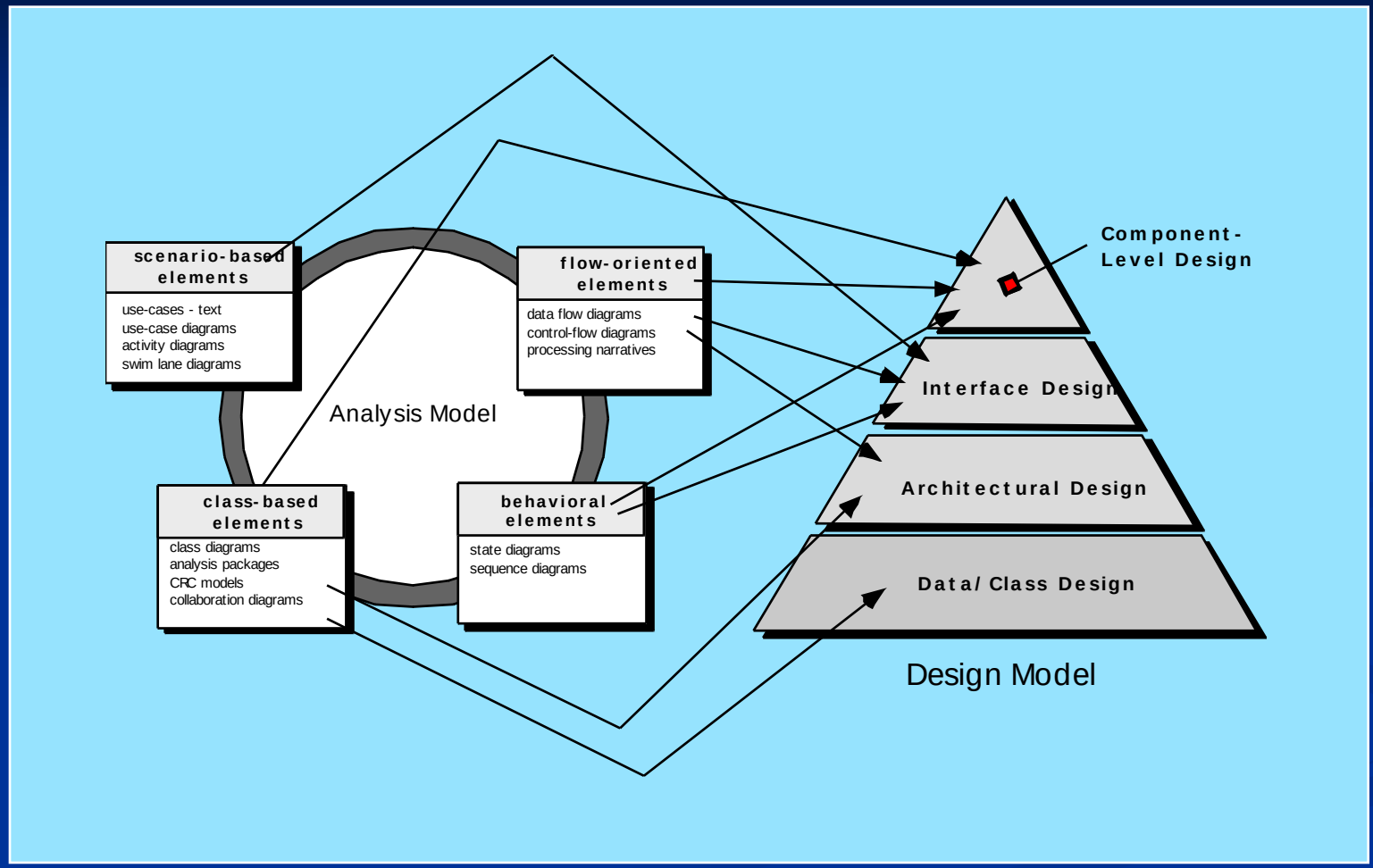
Rekayasa Perangkat Lunak (Software Engineering)

Bagian 2

Software Engineering: A Practitioner's Approach, 6/e

Chapter 9
Rekayasa Desain

Analysis Model -> Design Model



Desain dan Kualitas

- Desain harus mengimplementasikan semua kebutuhan eksplisit yang ada dalam model analisis, dan dia harus mengakomodasi semua kebutuhan implisit yang diinginkan oleh konsumen.
- Desain harus dapat berupa panduan yang dapat dibaca dan dipahami oleh orang-orang yang akan membuat kode, dan mereka yang menguji serta nantinya mendukung PL tersebut.
- Desain harus menyediakan gambaran utuh dari PL, menggambarkan domain data, fungsional, dan perilaku dari perspektif implementasi.

Panduan Kualitas

- Sebuah desain harus menampilkan arsitektur yang (1) dibuat menggunakan pola atau style arsitektural yang sudah dikenal, (2) terdiri dari komponen-komponen yang menunjukkan karakteristik desain yang baik dan (3) dapat diimplementasi dalam bentul yang evolusioner.
  - Untuk sistem yang lebih kecil, desain kadang dapat dikembangkan secara linear.
- Sebuah desain harus berbentuk modular; oleh karena itu PL harus secara logis dipartisi menjadi beberapa elemen subsistem
- Sebuah desain harus berisi representasi yang berbeda dari data ,arsitektur, antarmuka, dan komponen.
- Sebuah desain harus menuju struktur data yang tepat untuk class-class yang akan diimplementasi dan digambar dari pola data yang dikenal.
- Sebuah desain harus menuju komponen-komponen yang menunjukkan karakteristik fungsional yang dindpeneden.
- Sebuah desain harus menuju antarmuka yang mengurangi kompleksitas koneksi antara kompoenen-komponen dan dengan lingkungan eksternal.
- Sebuah desain harus diturunkan menggunakan method berulang yang diatur oleh informasi yang disebut selama analisis kebutuhan PL.
- Desain harus direpresentasikan menggunakan notasi yang secara efektif mengkomunikasikan maknanya.

Prinsip-Prinsip Desain

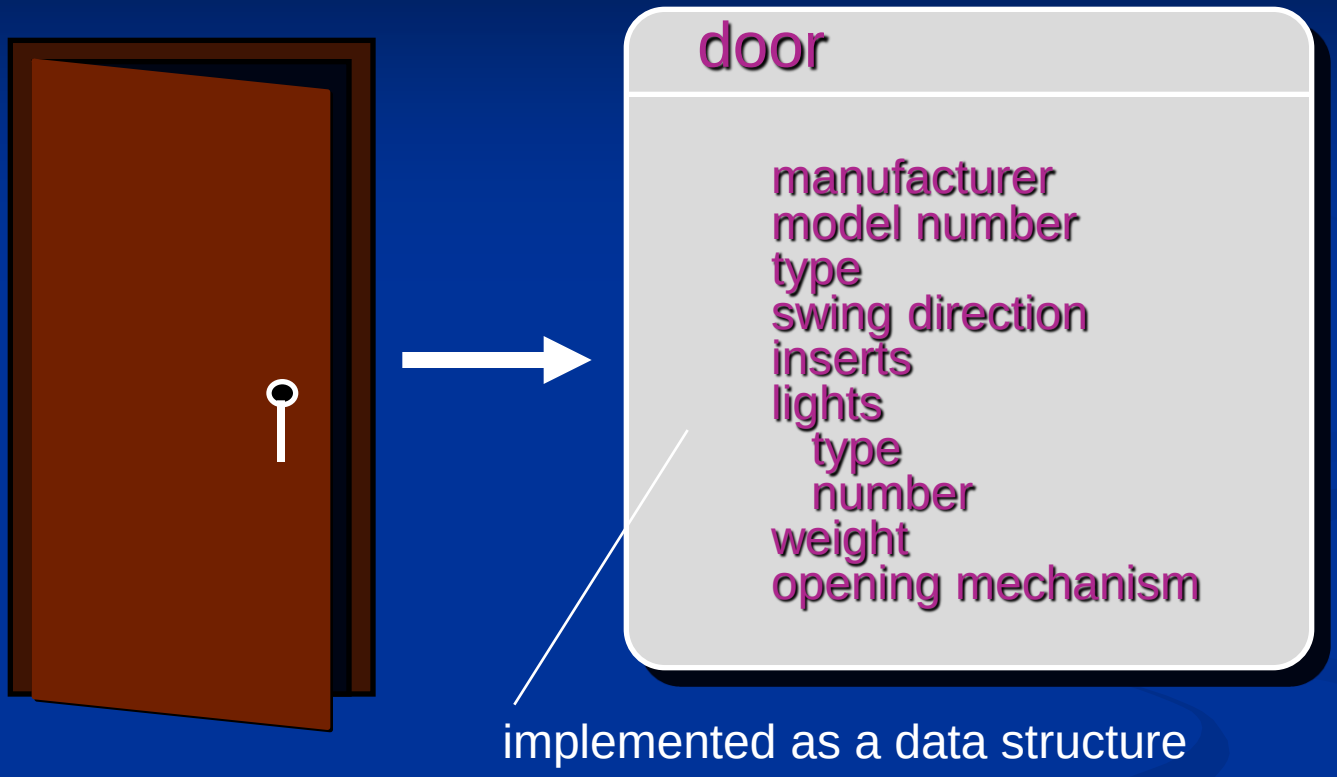
- Proses desain tidak boleh berjalan dengan “kacamata kuda”
- Proses desain harus bisa dirujuk dari model analisis.
- Proses desain tidak boleh mengulang penemuan-penemuan dasar.
- Desain harus dapat meminimalkan jarak intelektual antara PL dan permasalah yang ada di dunia nyata.
- Desain harus menampakkan keseragaman dan integrasi.
- Desain harus terstruktur untuk mengakomodasi perubahan.
- Desain harus terstruktur untuk turun secara bertahap, walaupun ketika data, event, atau kondisi operasi yang menyimpang ditemui.
- Desain bukan coding dan coding bukan desain.
- Desain harus dapat dipantau kualitasnya mulai dari dia dibuat, bukan setelah jadi.
- Desain harus direview untuk meminimalkan kesalahan semantik (konseptual).

From Davis [DAV95]

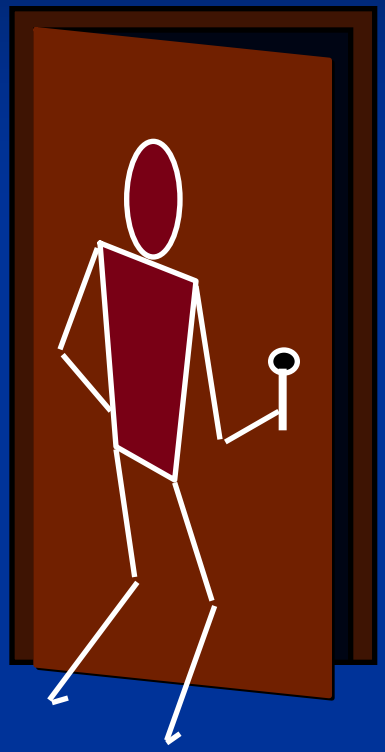
Konsep Dasar

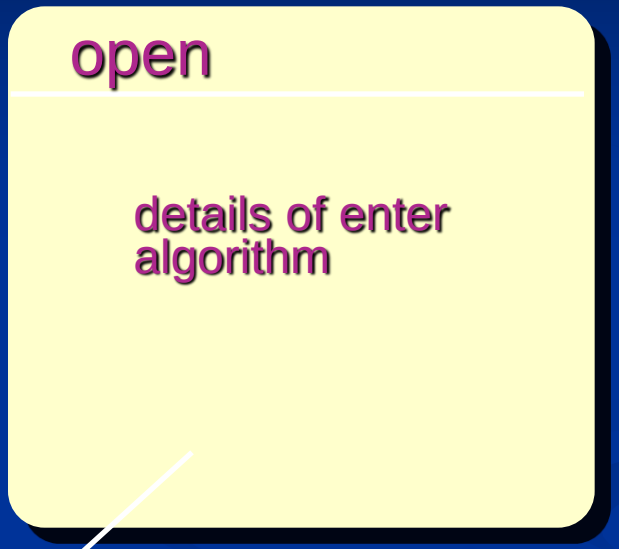
- abstraksi—data, prosedur, kontrol
- arsitektur—Struktur keseluruhan PL
- Patterns/pola—”memuat esensi” dari solusi desain yang sudah terbukti
- modularitas—Pembagian data dan fungsi
- menyembunyikan—interface terkendali
- Independensi fungsi—single-minded function dan low coupling
- refinement—elaborasi detail dari semua abstraksi
- Refactoring—sebuah teknik reorganisasi yang menyederhanakan desain

Abstraksi Data



Abstraksi Prosedur





implemented with a "knowledge" of the object that is associated with enter

Arsitektur

“Struktur keseluruhan dari PL dan cara dimana struktur menyediakan integritas konseptual bagi sebuah sistem” [SHA95a]

Properti Struktural. Aspek representasi desain arsitektur ini menentukan komponen-komponen sebuah sistem(mis : modul, objek, filter), dan pola komponen-komponen tersebut dipaket dan berinteraksi satu dengan yang lain. Sebagai contoh : objek dipaket untuk enkapsulasi baik data dan proses yang memanipulasi data dan berinteraksi dengan invokasi method

Properti extra-fungsional. Deskripsi desain arsitektur harus menggambarkan bagaimana arsitektur mencapai kebutuhan kinerja, kapasitas, reliabilitas, keamanan, adaptabilitas, dan karakteristik sistem yang lain.

Keluarga atau sistem-sistem yang berhubungan. Desain arsitektur harus dapat menggambar pola-pola yang diulang, yang secara umum ditemukan dalam disain keluarga atau sistem yang mirip. Esensinya, desain harus mempunyai kemampuan untuk menggunakan kembali blok-blok bangunan arsitektur

Patterns/Pola

Design Pattern Template

***Nama Pattern*—menggambarkan esensi pattern dalam nama yang singkat tapi ekspresif**

***Intent/Tujuan*—menjelaskan pattern dan apa yang dilakukan**

***Juga dikenal sebagai/Also-known-as*—Daftar sinonim untuk pattern terkait**

***Motivation/Motivasi*—menyediakan contoh masalah**

***Aplikabilitas*—menjelaskan situasi desain spesifik dimana pattern dapat diterapkan**

***Struktur*—menggambarkan class yang dibutuhkan untuk implementasi pattern**

***Participants*—menggambarkan tanggungjawab class-class yang diperlukan untuk mengimplementasikan pattern**

***Collaborations*—menggambarkan bagaimana participan berkolaborasi untuk memikul tanggung jwabanya.**

***Konsekuensi*—menggambarkan pengaruh desain terhadap pattern dan potensi masalah yang harus diperhatikan ketika pattern diimplementasi.**

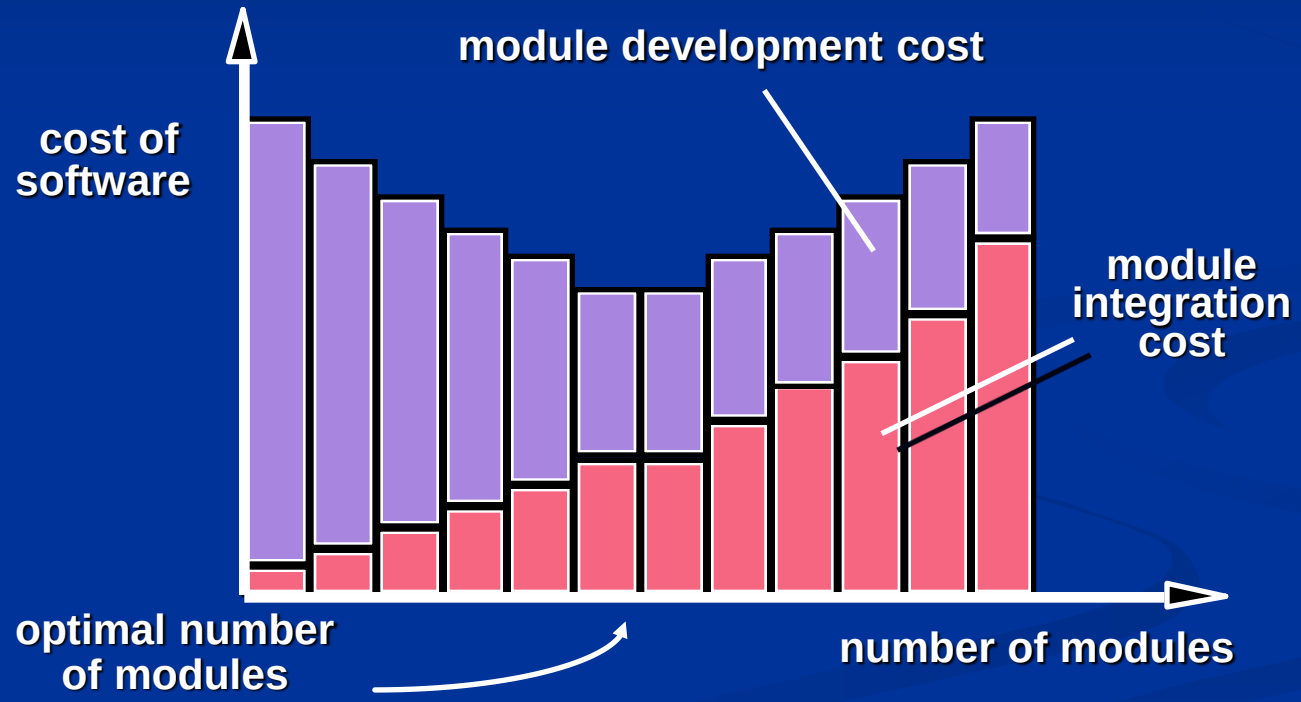
***Related patterns*—relasi referensi silang design patterns**

Desain Modular

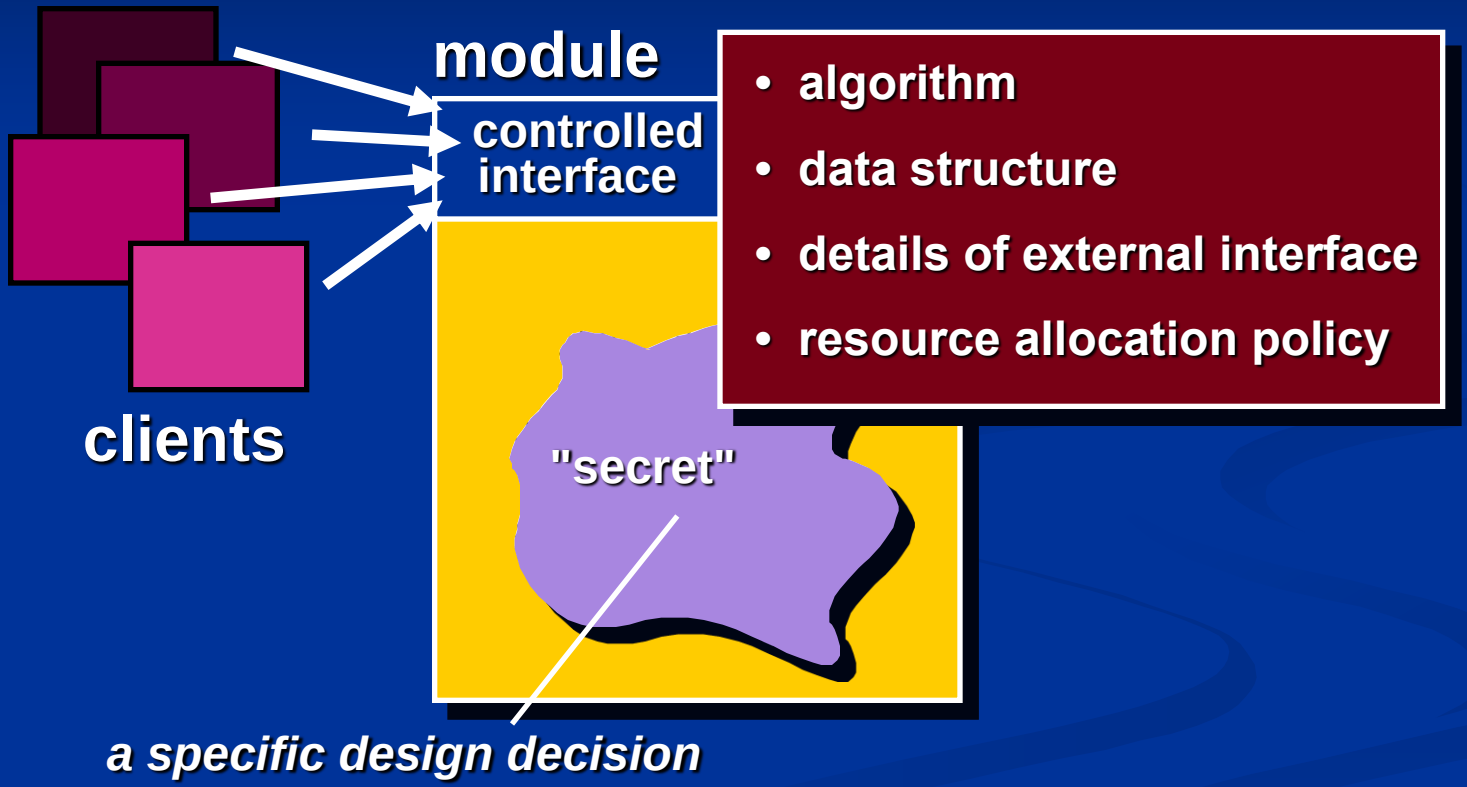
easier to build, easier to change, easier to fix ...

Modularitas: Permasalahan

Berapakah jumlah modul yang pas
Untuk desain PL tertentu ?



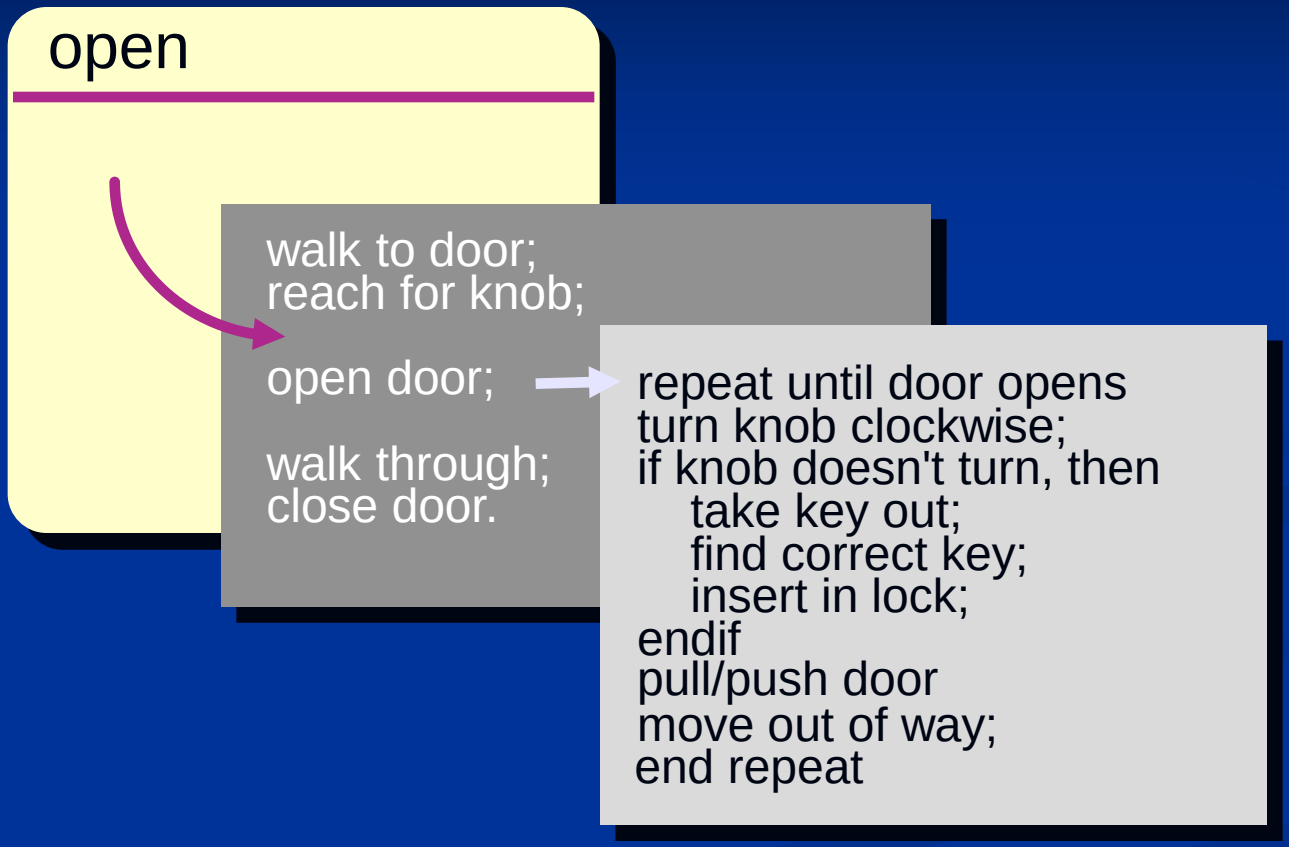
Penyembunyian Informasi



Mengapa Informasi Disembunyikan?

- Mengurangi “efek samping”
- Membatasi pengaruh global dari keputusan desain lokal
- Menekankan komunikasi melalui interface yang terkendali
- Mengurangi penggunaan data global
- Merujuk pada enkapsulasi—sebuah atribut dari desain kualitas tinggi
- Menghasilkan PL dengan kualitas tinggi

Langkah-langkah Refinement

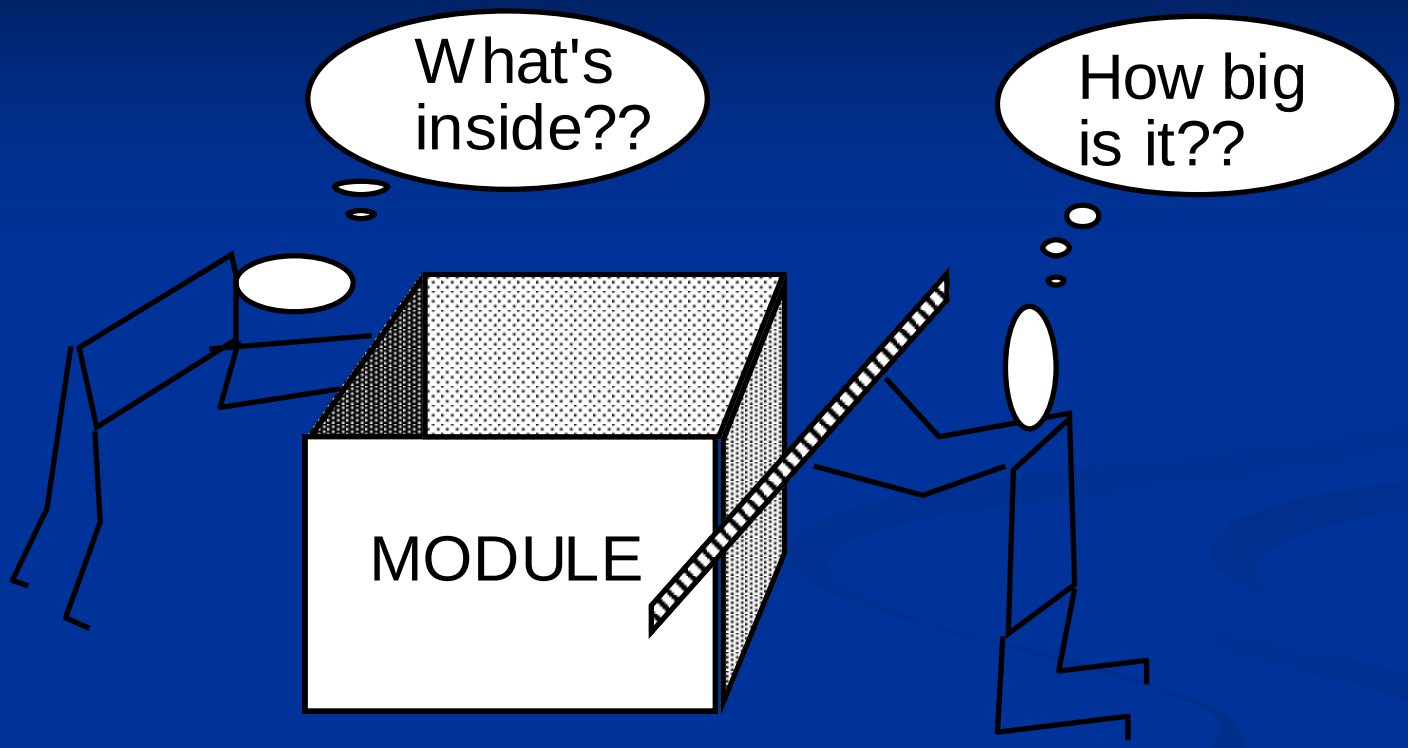


Independensi Fungsi

COHESION - the degree to which a module performs one and only one function.

COUPLING - the degree to which a module is "connected" to other modules in the system.

Mengukur modul: dua sudut pandang



Refactoring

- Fowler [FOW99] mendefinisikan refactoring sbb :
  - "Refactoring adalah proses mengubah sistem PL dimana dia tidak mengubah perilaku eksternal dari kode (desain) sehingga menigkatkan struktur internal."
- Ketika PL refactored, desain akan diperiksa terhadap :
  - redundancy
  - Elemen desain yang tidak berguna
  - Algoritma yang tidak efisien atau tidak perlu
  - Struktur data dengan konstruksi yang buruk atau tidak tepat
  - Atau kesalahan desain lain yang dapat diperiksa untuk menghasilkan desain yang lebih baik.

Konsep Desain OO

- Desain Class
  - Entity classes
  - Boundary classes
  - Controller classes
- Inheritance—semua tanggung jawab superclass akan diwarisi oleh semua subclassnya
- Messages—stimulasi beberapa perilaku yang dapat terjadi pada objek penerima pesan
- Polymorphism—sebuah karakteristik yang mengurangi usaha yang dibutuhkan untuk memperluas desain

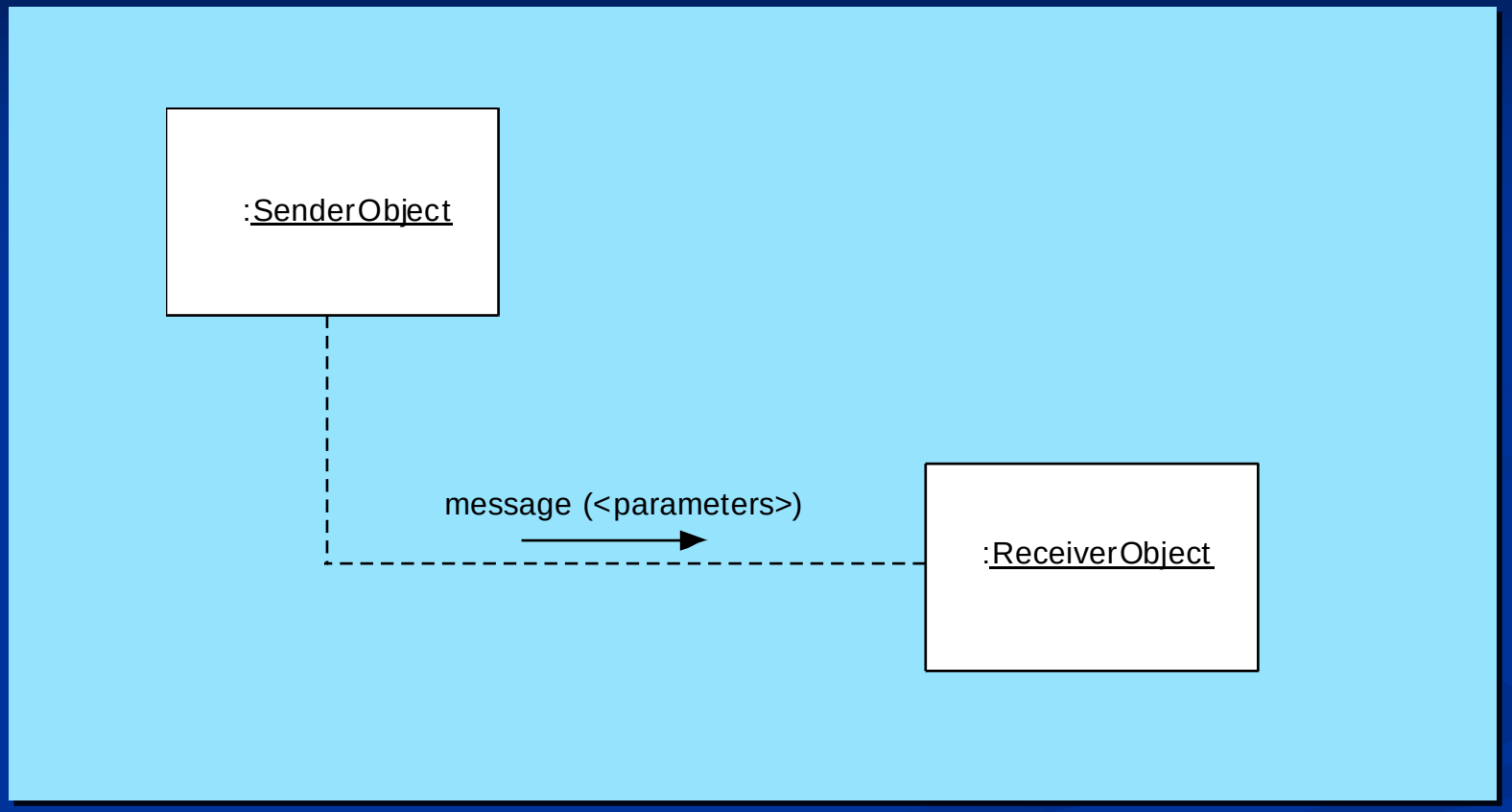
Desain Class

- Analisis class disempurnakan dalam desain untuk menjadi class-class entitas
- Class-class boundary dikembangkan selama desain untuk membuat interface(mis. Layar interaktif atau laporan cetak) yang dilihat pengguna dan berinteraksi.
  - Class-class boundary didesain dengan tanggungjawab untuk mengelola cara objek entitas ditampilkan kepada user.
- Class-class control didesain untuk mengelola :
  - Pembuatan atau perubahan objek entitas;
  - Instansiasi object boundary dengan mengambil informasi dari objek entitas;
  - Komunikasi kompleks antara sekelompok objek;
  - Validasi data yang dikomunikasikan antar objek atau antar pengguna dan aplikasi.

Inheritance

- Pilihan-pilihan desain :
  - Class dapat didesain dan dibangun dari nol. Jika demikian, inheritance tidak digunakan.
  - Hierarki class dapat dicari untuk mencari kemungkinan jika sebuah class yang lebih tinggi pada hierarki (superclass) memiliki atribut-atribut dan operasi-operasi yang paling banyak dibutuhkan. Class baru ini diturunkan dari superclass dan beberapa tambahan dapat diberikan jika dibutuhkan.
  - Hierarki class dapat di restrukturisasi sehingga atribut-atribut dan operasi-operasi yang dibutuhkan dan diturunkan oleh class baru tersebut.
  - Karakteristik dari class yang sudah ada dapat di override dan atribut-atribut dan operasi-operasi dengan versi berbeda dapat diimplementasikan untuk class baru tersebut.

Messages



Polymorphism

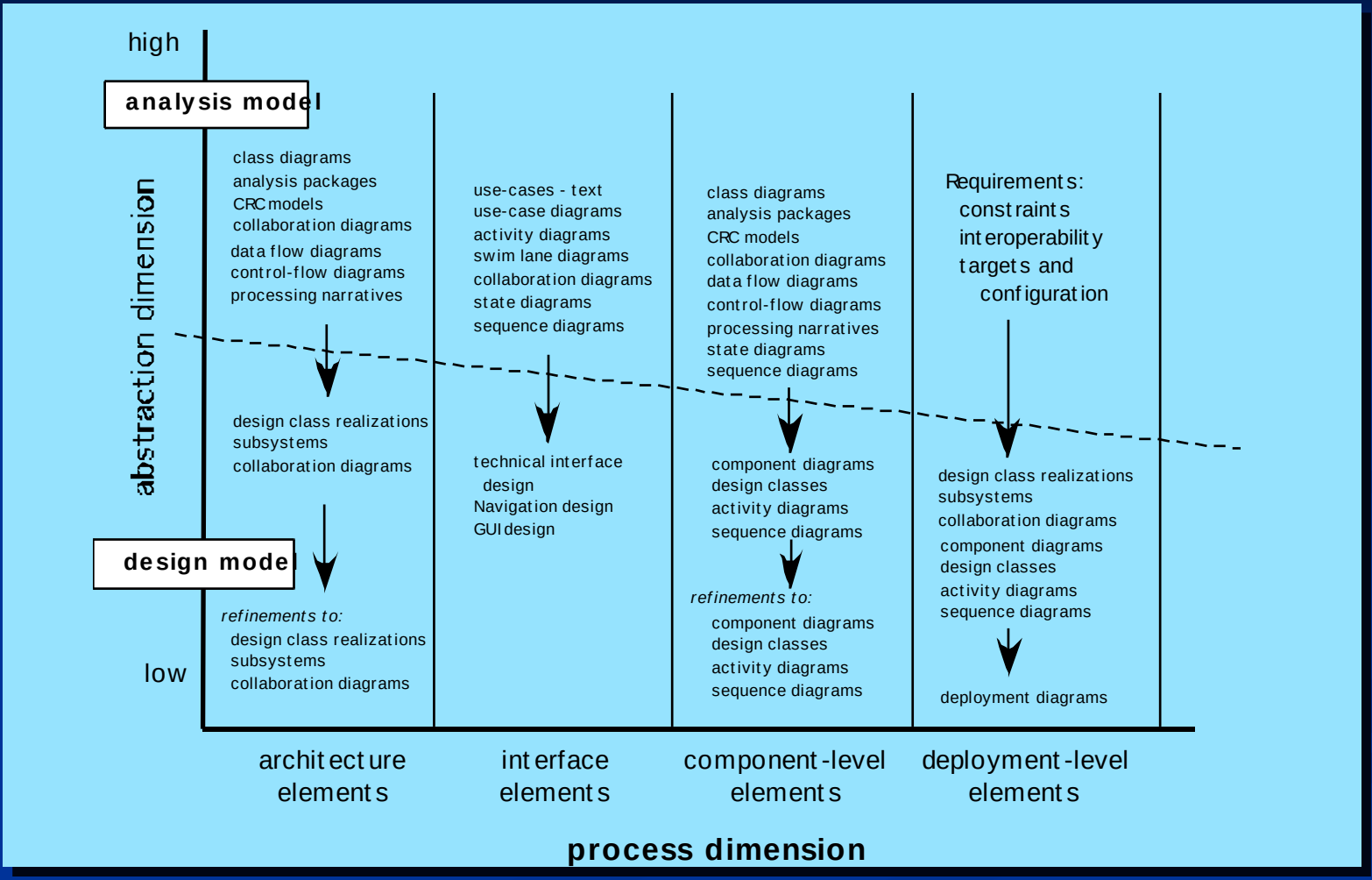
Pendekatan konvensional …

```
case of graphtype:
        if graphtype = linegraph then DrawLineGraph (data);
        if graphtype = piechart then DrawPieChart (data);
        if graphtype = histogram then DrawHisto (data);
        if graphtype = kiviat then DrawKiviat (data);
end case;
```

Semua graphs menjadi subclass dari class umum yang disebut graph. Menggunakan konsep overloading, setiap subclass mendefinisikan operasi yang disebut *draw.* Sebuah object dapat mengirim pesan draw pada salah satu instansiasi objek dari salahsatu subclassnya. Objek yang menerima message akan menjalankan operasi draw nya sendriri untuk membuat graph yang sesuai..

```
graphtype draw
```

Model Desain



Elemen-Elemen Model Desain

- Elemen-elemen Data
  - Data model --> struktur data
  - Data model --> arsitektur database
- Elemen-elemen arsitektur
  - Domain aplikasi
  - Class-class analisis, relasinya, kolaborasi dan perilaku diubah menjadi realisasi desain
  - Patterns dam “styles” (Chapter 10)
- Elemen-elemen interface
  - user interface (UI)
  - Interface external pada sistem lain, piranti-piranti, jaringan-jaringan atau produsen maupun konsumen informasi lainnya
  - Interface internal antara komponen-komponen desain.
- Elemen-elemen komponen
- Elemen-elemen deploy

Elemen-elemen Interface

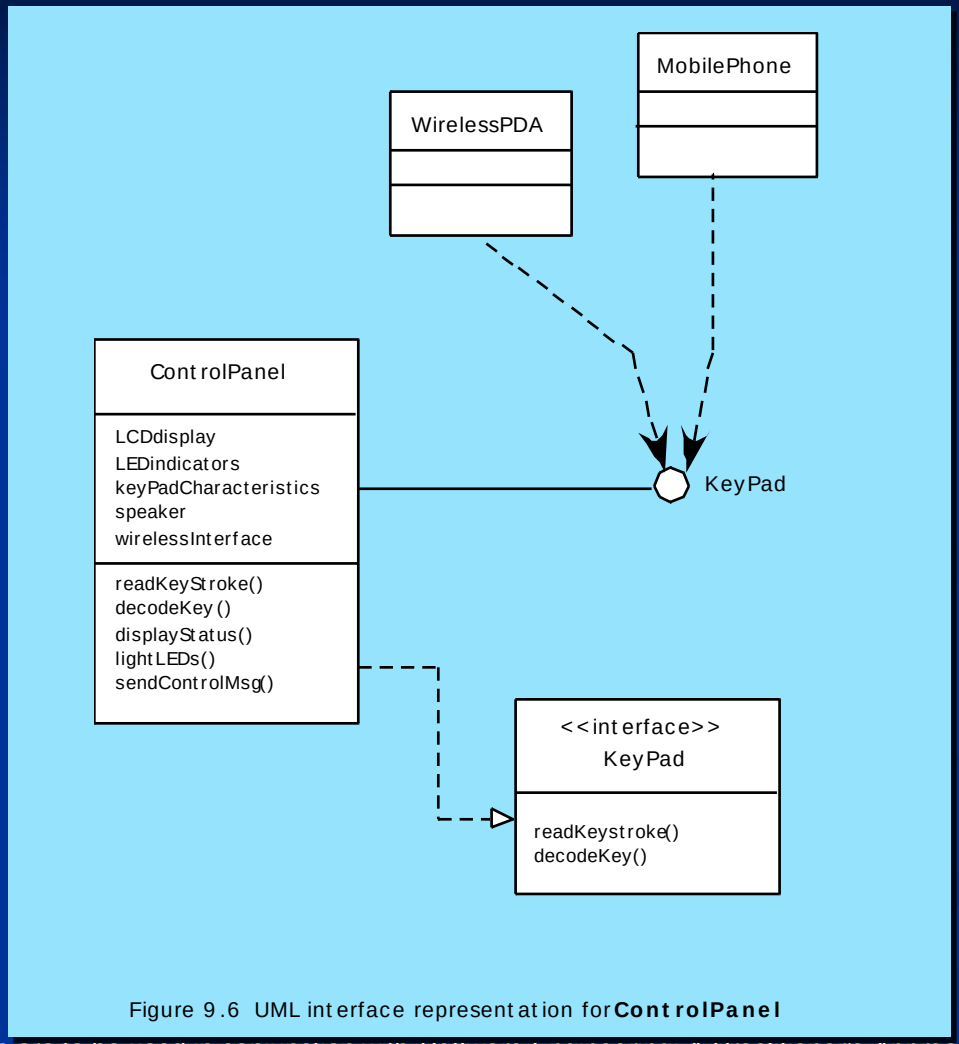
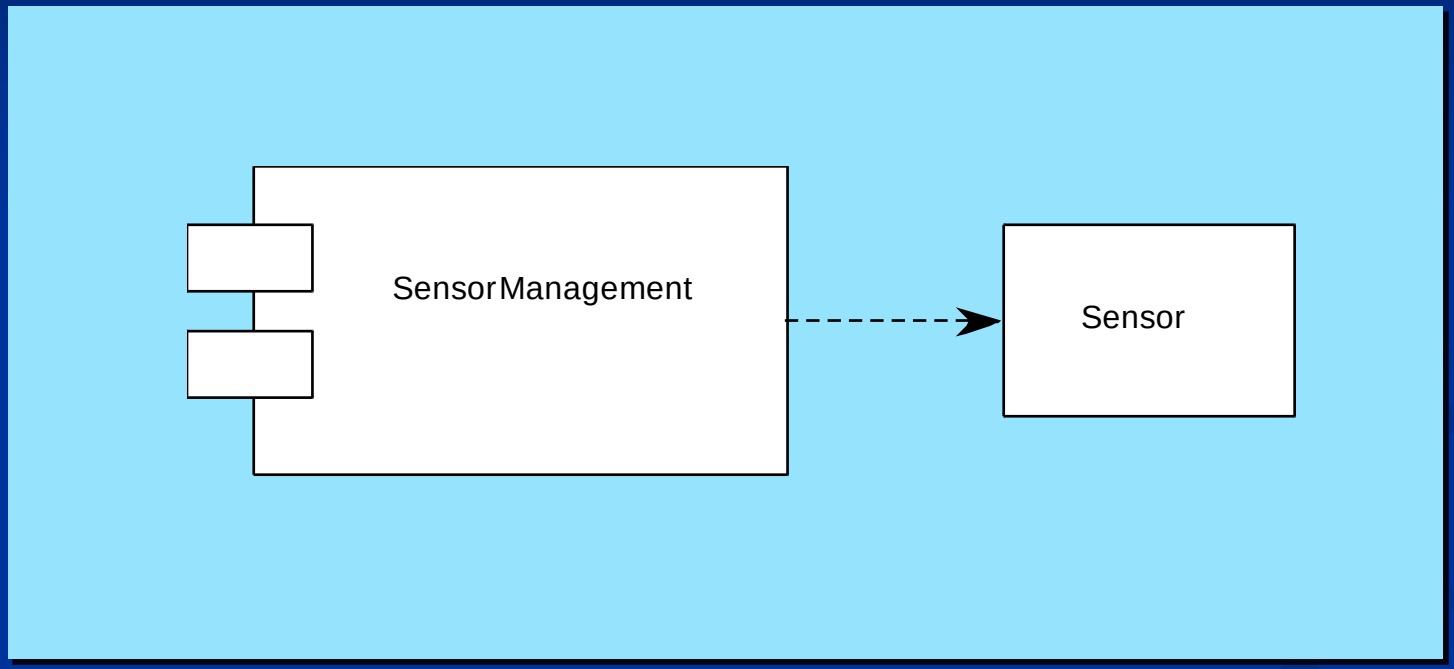


Figure 9.6 UML interface representation for **ControlPanel**

Elemen-elemen Komponen



Elemen-elemen Deployment

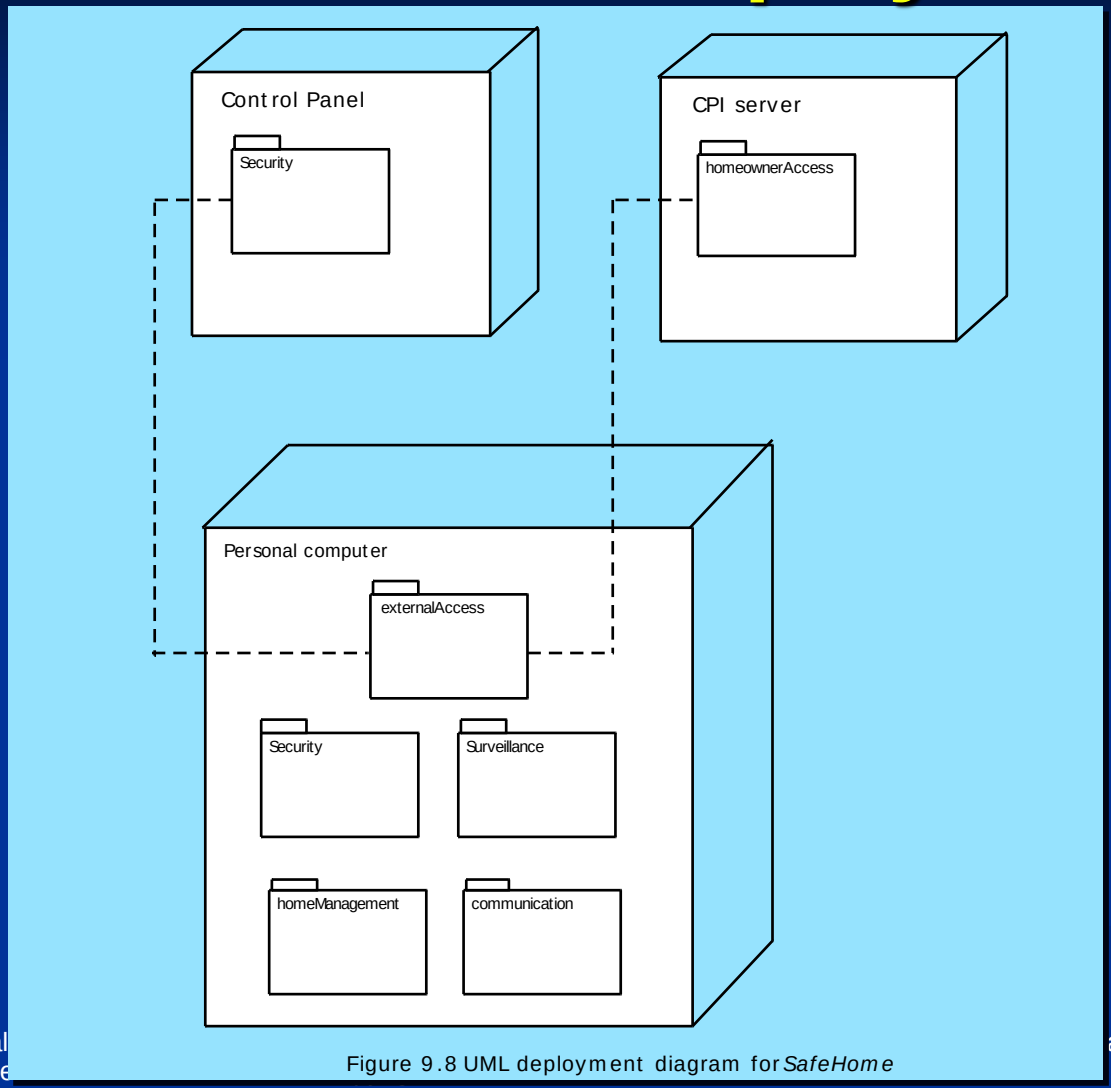


Figure 9.8 UML deployment diagram for *SafeHome*

Design Patterns

- Desainer terbaik di segala bidang tetap mempunyai keterbatasan untuk melihat pola yang mencirikan sebuah masalah dan menghubungkannya dengan pola yang dapat dikombinasikan untuk membuat solusi
- Sebuah deskripsi dari design pattern dapat juga dilihat sebagai sekumpulan design forces.
  - *Design forces* menjelaskan kebutuhan non fungsional (misalkan : kemudahan perawatan, portabilitas) yang dihubungkan dengan PL dimana pattern akan diaplikasikan.
- Karakteristik pattern (class, tanggungjawab, dan kolaborasi) mengindikasikan atribut-atrobit desain yang harus diatur untuk memungkinkan pattern mengakomodasi permasalahan yang bervariasi.

Frameworks

- Sebuah framework bukan merupakan pattern arsitektur, namun lebih merupakan kerangka dengan sekumpulan “plug points” (yang juga disebut *hooks* dan *slots*) yang memungkinkannya untuk beradaptasi dengan domain permasalahan tertentu.
- Gamma et al mencatat bahwa:
  - Design patterns lebih abstrak dari frameworks.
  - Design patterns adalah elemen-elemen arsitektural yang lebih kecil daripada frameworks
  - Design patterns lebih umum daripada frameworks