Pengenalan Paradigma Objek

IF2210 – Semester II 2022/2023

Tim Pengajar IF2210

Tujuan

- Mahasiswa memahami dan dapat membedakan metodologi terstruktur (prosedural) dibandingkan dengan berorientasi objek dalam pengembangan perangkat lunak
- Mahasiswa mengenal Object-Oriented Language dan metodologi “terkait” OO (object-based programming, visual programming, event driven programming)
- Mahasiswa mendapatkan “sense” mengenai perbedaan aplikasi yang dikembangkan secara OO dan bukan, dengan harapan pada saat kelak mengembangkan aplikasi OO, akan mampu menerapkan konsep OO

Dua Pendekatan Pembangunan Perangkat Lunak

- **Struktural**
  - Sudut pandang pemrogram adalah "*how computers work*" (bagaimana data direpresentasikan dalam memori; **urutan instruksi**/proses terhadap data).
  - Biasanya analisis dimulai dari input-proses-output.
  - Inspirasi: proses bisnis yang diterjemahkan ke struktur program (urutan instruksi + *control flow* (*conditional*, *loop*))
- ***Object-Oriented***
  - Sudut pandang pemrogram adalah objek apa saja yang terlibat dalam domain persoalan.
  - Analisis dimulai dari identifikasi objek beserta **perilakunya** dan **bagaimana objek-objek saling berinteraksi**.
  - Inspirasi: sel biologis, "software IC"

Analogi OO

- **Sel biologis** – Alan Kay (pencipta OO, GUI, & bahasa Smalltalk)
  - Alan Kay's favorite metaphor for software objects is a biological system. Like cells, software objects don't know what goes on inside one another, but they communicate and work together to perform complex tasks.
- **Software IC** – Brad Cox (pencipta Bahasa Objective-C)
  - (Dalam buku *Object-oriented programming: an evolutionary approach*, 1986)
  - "a desirable trait for software objects is the ability to use them as standardized and interchangeable parts to 'mass-produce' larger constructs," seperti IC (integrated circuit) pada elektronika (hardware).

Contoh: structural vs. OO (1)

```
program frequency;
    const
        size = 80;
    var
        s: string[size];
        i: integer;
        c: character;
        f: array[1..26] of integer;
        k: integer;
begin
    writeln('enter line');
    readln(s);
    for i := 1 to 26 do f[i] := 0;
    for i := 1 to size do
    begin
        c := asLowerCase(s[i]);
        if isLetter(c) then
        begin
            k := ord(c) - ord('a') + 1;
            f[k] := f[k] + 1
        end
    end;
    for i := 1 to 26 do
        write(f[i], ' ')
end.
```

- Struktural: program Pascal untuk menghitung jumlah kemunculan setiap huruf pada sebuah String.

Contoh: structural vs. OO (2)

```
| s c f k |
f := Array new: 26.
s := Prompter prompt: 'enter line' default: ' '.
1 to: 26 do: [:i | f at: i put: 0].
1 to: s size do: [
    :I | c := (s at: i) asLowerCase.
    c isLetter ifTrue: [
        k := c asciiValue - $a asciiValue + 1.
        f at: k put: (f at: k) + 1.
                ].
            ].
^ f
```

- Bahasa OO (Smalltalk), cara berpikir masih prosedural.

Contoh: structural vs. OO (3)

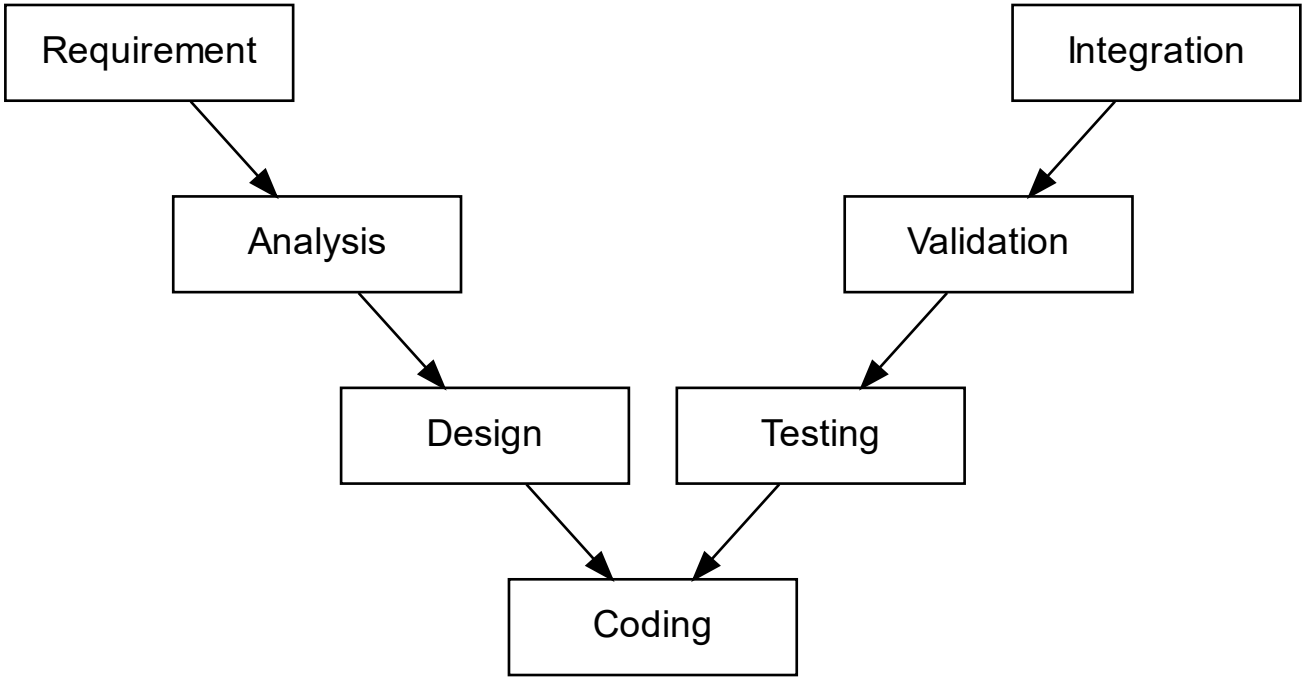
```
| s f |
s := Prompter prompt: ' enter line ' default: ' '.
f := Bag new.
s do: [ :c | c isLetter ifTrue: [f add: c asLowerCase]].
^ f.
```

"untuk setiap c bagian dari s, lakukan: jika c adalah letter, tambahkan c dalam bentuk huruf kecil ke dalam f"

- Real OO using Smalltalk.

OO∗ and Software Lifecycle (Konteks OOP)

- OOA
- OOD
- OOP
- OOT

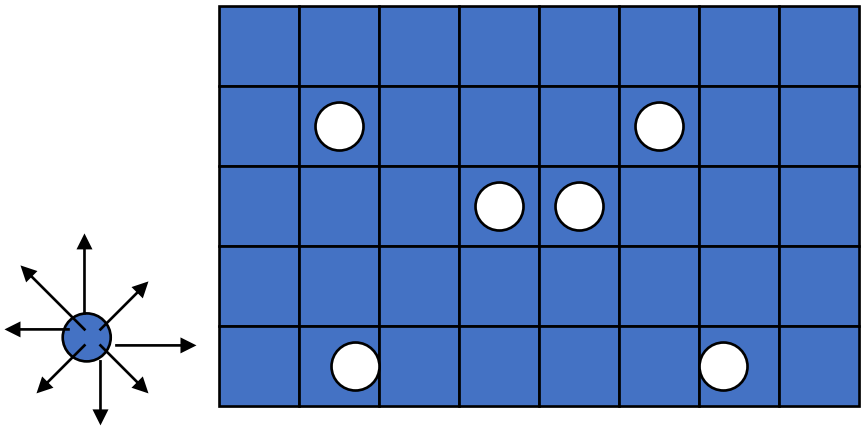


Dalam Software Lifecycle...

- Ada penentuan *requirement*, untuk apa SW dibuat, data, fungsi, behaviour
- Ada artefak desain (model PL):
  - Memodelkan "statis" (*state*, atribut) dan "dinamika" (*behaviour*, perilaku)
  - bagaimana dekomposisi menjadi modul, *class*, ... Dengan notasi tertentu, misalnya diagram kelas, CRC card, dll.
- Implementasi desain menjadi *source code* (*file*) dan artefak lain, sesuai dengan kaidah yang baik. Anda sudah belajar bagaimana membagi-bagi sebuah aplikasi menjadi sejumlah *file* dalam bahasa C.
- Antara desain dengan implementasi, harus *traceable*. Ibarat gambar rumah dengan rumah yang dibangun.
- Ada berbagai cara implementasi untuk mewujudkan efek yang sama ke pengguna.

Contoh: bola dalam bidang

- Sebuah bidang mengandung sekumpulan bola. Setiap bola mempunyai arah
- Bola bergerak sesuai dengan arahnya
- Jika “bersitabrak”, maka bola akan mati
- Jika bola habis sama sekali, sistem mati
- Jika bola tersisa “sedikit”, akan lahir bola-bola baru hingga jumlah tertentu



Berbagai solusi bola dalam bidang

- Solusi prosedural sekuensial:
  - Sebuah main program
  - ADT list of bola (linked list, matriks), yang dibuat, ditambah, dikurangi bolanya
- Solusi dengan proses konkuren:
  - Satu main program menumbuhkan banyak proses
  - setiap bola adalah proses
  - Semua bola dipetakan ke sebuah bidang (matriks posisi)
- Jika benda bergerak bukan hanya bola, namun perilaku sama: Solusi dengan proses konkuren dan ADT generik (list of list, atau list of “things”)
- Sekilas solusi OO:
  - Bola-bola adalah objek, *instance* dari sebuah kelas Bola.
  - Jika benda bergerak bukan hanya bola, maka objek-objek adalah instance dari kelas-kelas yang memiliki hubungan inheritance.

Metodologi OO SW Development

- Memandang persoalan dari sudut “Objek”
  - Coadd
  - Rumbaugh
  - Jacobson
  - Booch
  - Martin Odel
  - UML

Standar:

- Notasi
- Diagram
- Bahasa Spesifikasi

OO Methodology (1)

Coadd Yourdon

- Object diagram
- Class diagram
- Notion of Subject

Rumbaugh

- Object modeling - Object diagram
- Dynamic Modeling - State diagram
- Functional Modeling - DFD

OO Methodology (2)

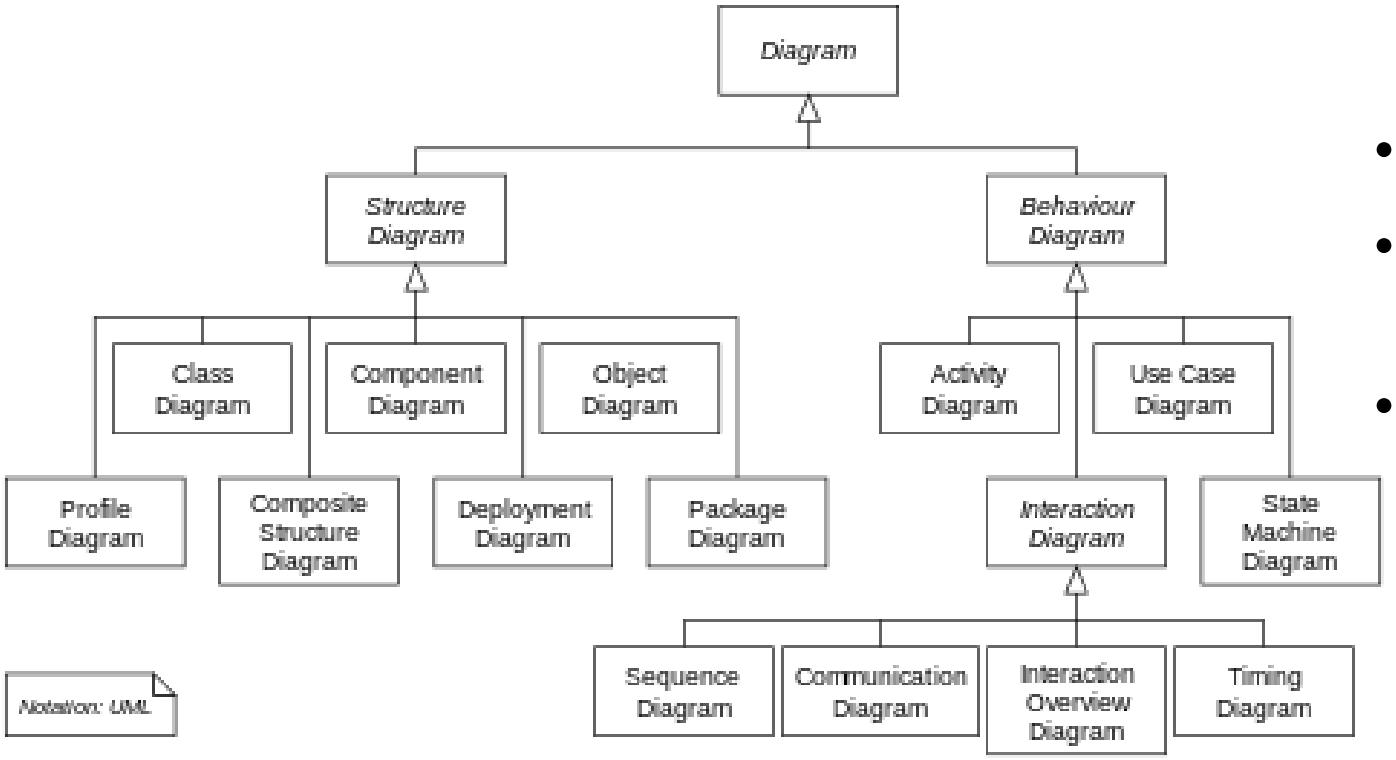
Booch

- Class Diagram
- State transition Diagram
- Object Diagram
- Timing Diagram
- Module Diagram
- Process Diagram

UML Diagram [yang utama]

- Structural Diagram:
  - Class, object diagram
  - Component diagram
  - Deployment diagram
- Behavioural Diagram
  - Use-case
  - Collaboration diagram
    Sequence diagram
  - State chart diagram
  - Activity diagram

UML Diagram



- Use Case driven
- Architecture-Centric Process
- Iterrative and Incremental process

OO Language

- Pure OO: ex. Smalltalk, Eiffel, Java(?)
- Procedural OO: ex. C++, Ada 95
- Functional and OO: ex. CLOS, Object LISP, Scala
- Declarative: prolog extension, ex. Flora-2, Logtalk, Oblog
- Bacaan :
  - http://en.wikipedia.org/wiki/Object-oriented_programming
  - http://en.wikipedia.org/wiki/List_of_object-oriented_programming_languages
  - http://en.wikipedia.org/wiki/Comparison_of_programming_languages_%28object-oriented_programming%29
  - http://en.wikipedia.org/wiki/Encapsulation_%28object-oriented_programming%29
  - http://en.wikipedia.org/wiki/Object-based_language

Apakah ada “ukuran” derajat “OO” sebuah program?

- Prosedural dan sekuensial, tanpa ada definisi “kelas” misalnya menggunakan bahasa C++ , atau karena dalam bahasa Java kita diharuskan mempunyai kelas, maka hanya berisi sebuah class dengan “void main(...)”
- Program tidak mengandung definisi “kelas”, tetapi
  - Menggunakan class library Class (misalnya STL)
  - It uses available class/“object” (⟶ object-based programming)
- Program mengandung definisi “kelas” namun hanya diperlakukan sebagai “ADT”.
- Program anda dimodelkan dengan OOAD, dan secara konsisten diprogram/ dikoding dengan bahasa OO ataupun yang lain (misalnya C yang tidak OO)
- Domain dimodelkan dengan OOAD, diprogram dengan/tanpa OOL dan direlasikan dengan GUI object (windows, button...) – “event driven programming”

Batasan Kuliah IF2210 OOP

- Pemodelan perangkat lunak dengan notasi/diagram di atas menjadi bagian dari kuliah Rekayasa Perangkat Lunak
- Pada kuliah OOP:
  - Memrogram harus berdasarkan spesifikasi yang jelas, kuliah dimulai dari analisis persoalan secara umum, langsung memodelkan software dalam diagram kelas
  - Diagram yang banyak dipakai adalah diagram kelas. Diagram-diagram lain yang diperlukan untuk pendukung dapat disinggung/dipakai

OOP dan paradigma terkait

- Pada contoh bola dan bidang, seringkali gambar visual dapat diplot ke layar dengan menggunakan objek visual yang tersedia:
  - OOP vs event driven programming
  - OOP vs Visual programming [VB is not visual programming, but GUI object-based programming]
  - Object-based programming