

Tim Pengajar IF2250

# IF2250 – Rekayasa Perangkat Lunak Pendahuluan

SEMESTER II TAHUN AJARAN 2022/2023



KNOWLEDGE & SOFTWARE ENGINEERING



# *Sixty years ago no one could have predicted...*

- software would enable the creation of new technologies (e.g., genetic engineering and nanotechnology),
- the extension of existing technologies (e.g., telecommunications),
- the radical change in older technologies (e.g., the media);
- software would be the driving force behind the PC revolution;
- software applications would be purchased by consumers using their smart phones;
- software would slowly evolve from a product to a service as “on-demand” software companies deliver just-in-time functionality via a Web browser;
- a software company would become larger and more influential than all industrial-era companies;
- software-driven network would evolve (from library research to consumer shopping /political discourse / the dating habits).



KNOWLEDGE & SOFTWARE ENGINEERING

# Software

- Software is designed and built by software engineers.
- Software is used by virtually everyone in society.
- Software is pervasive in our commerce, our culture, and our everyday lives.
- Software engineers have a moral obligation to build **reliable** software that does no harm to other people.
- Software engineers view computer software, as being made up of the **programs**, **documents**, and **data** required to design and build the system.
- Software users are only concerned with whether or not software products meet their **expectations** and make their tasks **easier** to complete.



KNOWLEDGE & SOFTWARE ENGINEERING

# *When computer software succeeds?*

- when it **meets the needs of the people** who use it,
- when it **performs flawlessly** over a long period of time,
- when it is **easy to modify** and even **easier to use**  
it can and does change things for the better.



KNOWLEDGE & SOFTWARE ENGINEERING

# *When software fails?*

- when its **users are dissatisfied**,
- when it is **error prone**,
- when it is **difficult to change** and even **harder to use**  
bad things can and do happen



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 RPL

# *Important Questions for Software Engineers*

- Why does it take **so long** to get software finished?
- Why are development **costs so high?**
- Why can't we find all **errors** before we give the software to our customers?
- Why do we spend so much time and effort **maintaining** existing programs?
- Why do we continue to have difficulty in **measuring** progress as software is being developed?

# What is software ?

- Definitions:

Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system

**(IEEE Standard Glossary of Software Engineering Terminology, 1990)**



KNOWLEDGE & SOFTWARE ENGINEERING

# *Software Characteristics*

- Software is both a **product** and a **vehicle** for delivering a product (information).
- Software is **engineered** not manufactured.
- Software does **not wear out**, but it does **deteriorate**.
- Industry is moving toward **component-based software construction**, but most software is still **custom-built**.



KNOWLEDGE & SOFTWARE ENGINEERING

# *Software Application Domains*

- **System** software
- **Application** software
- **Engineering** or Scientific Software
- **Embedded** software
- **Product-line** software (includes entertainment software)
- **Web**-Applications
- **Mobile Based** Applications
- **Artificial intelligence** software

# *Legacy Software Evolves*

- The software must be adapted to meet the needs of new computing environments or technology.
- The software must be enhanced to implement new business requirements.
- The software must be extended to make it interoperable with other more modern systems or databases.
- The software must be re-architected to make it viable within a evolving computing environment.



KNOWLEDGE & SOFTWARE ENGINEERING

# *Software Engineering (I)*

- Software engineering is the establishment of sound **engineering principles** in order to obtain **reliable** and **efficient** software in an **economical** manner.
- Software engineering is the application of a **systematic**, **disciplined**, **quantifiable** approach to the **development**, **operation**, and **maintenance** of software.
- Software engineering encompasses a **process**, **management techniques**, **technical methods**, and the **use** of tools.

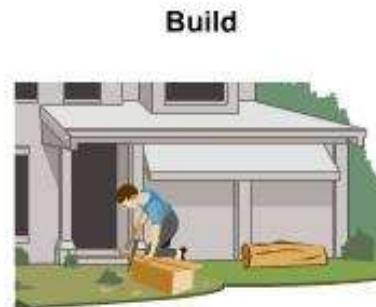
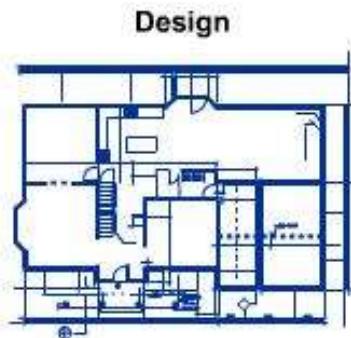


KNOWLEDGE & SOFTWARE ENGINEERING

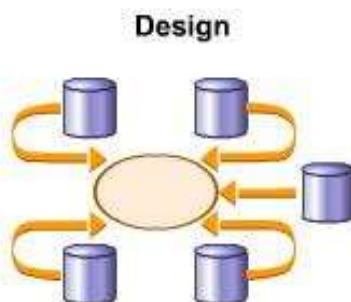
IF2250 RPL

# Software Engineering (2)

Engineering:



Software Engineering:



KT  
KNOWLEDGE & SOFTWARE ENGINEERING

# *Four broad categories of software are evolving to dominate the industry*

13

1. Web-based systems and applications (WebApps )
2. Mobile Applications
3. Cloud computing
4. Product Line Software



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 RPL

# *I. Web-based systems and applications*

- The augmentation of HTML by development tools (e.g., XML, Java) enabled Web engineers to provide computing capability along with informational content.
- Over the past decade, Semantic Web technologies (Web 3.0) have evolved into sophisticated corporate and consumer applications that encompass “semantic databases [that] provide new functionality that requires Web linking, flexible [data] representation, and external access APIs.”
- Sophisticated relational data structures will lead to entirely new WebApps that allow access to disparate information in ways never before possible.



KNOWLEDGE & SOFTWARE ENGINEERING

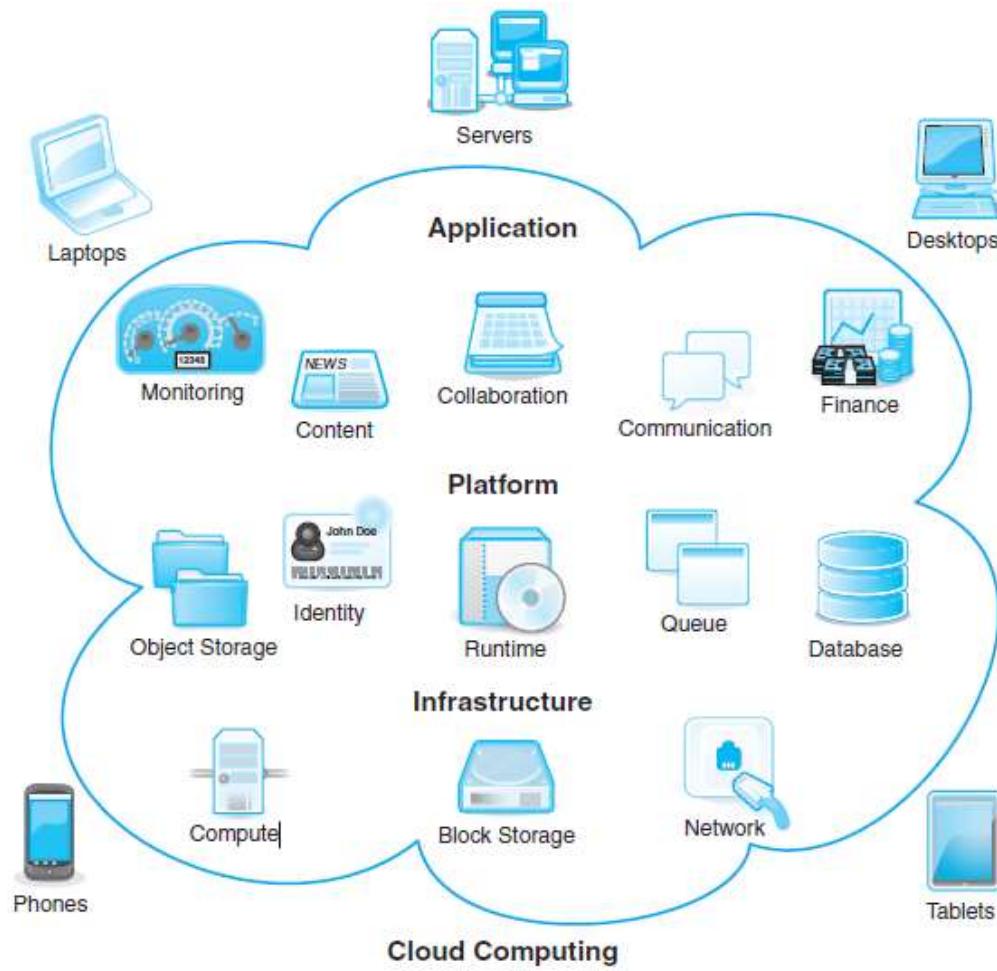
## 2. *Mobile Applications*

- The term app has evolved to connote software that has been specifically designed to reside on a mobile platform (e.g., iOS, Android, or Windows Mobile).
- encompass a user interface that takes advantage of the unique interaction mechanisms provided by the mobile platform,
- interoperability with Web-based resources



KNOWLEDGE & SOFTWARE ENGINEERING

### 3. Cloud computing



## 4. Product Line Software

- The Software Engineering Institute defines a *software product line* as “a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.”
- include requirements, architecture, design patterns, reusable components, test cases, and other software engineering work products



KNOWLEDGE & SOFTWARE ENGINEERING

# *Software Engineering – a layered technology*



KNOWLEDGE & SOFTWARE ENGINEERING

## ***Software Engineering – a layered technology (2)***

- The foundation for software engineering is the **process** layer, defines a framework that must be established for effective delivery of software engineering technology.
- Software engineering **methods** provide the technical how-to's for building software.
- Software engineering **tools** provide automated or semi-automated support for the process and the methods
  - computer-aided software engineering : e.g Rational Rose; various IDE (Integrated Development Environment) such as: VisualStudio, Eclipse, NetBeans; Software version, such as: CVS, SVN, and GitHub



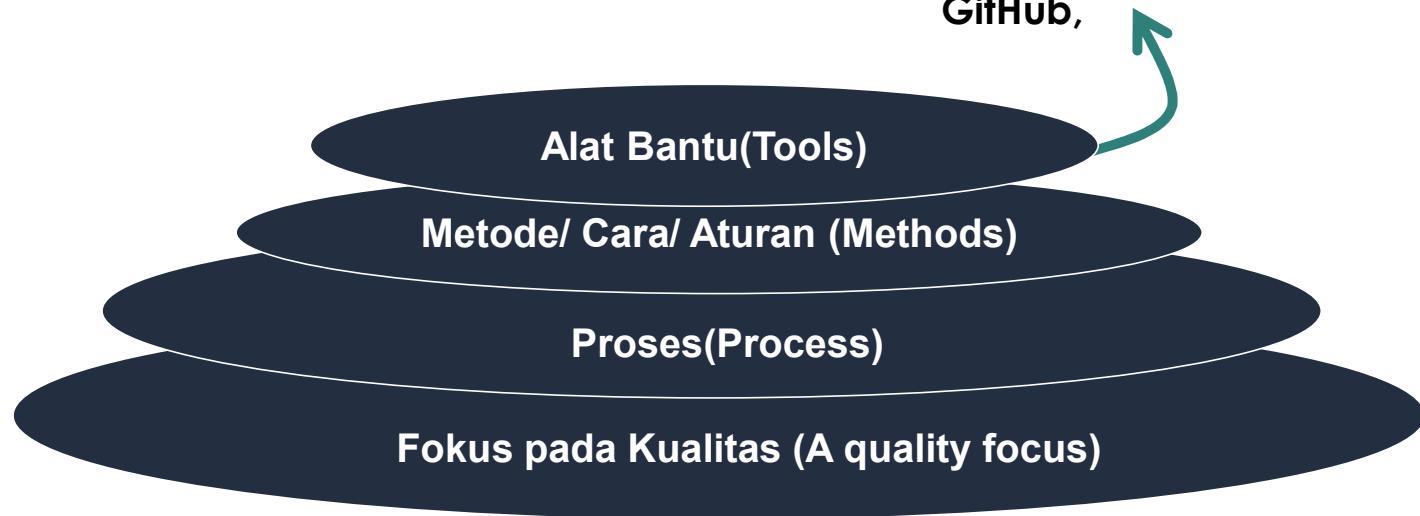
KNOWLEDGE & SOFTWARE ENGINEERING

# *Lapisan di RPL (1)*

20

Tiap Lapisan tidak bisa berdiri sendiri, masing-masing memiliki ketergantungan antar-lapisan.

- CASE Tool, contoh: Rational Rose,
- Berbagai jenis IDE (Integrated Development Environment) seperti: VisualStudio, Eclipse, NetBeans
- Versi Software, contoh: CVS, SVN, GitHub,



KNOWLEDGE & SOFTWARE ENGINEERING

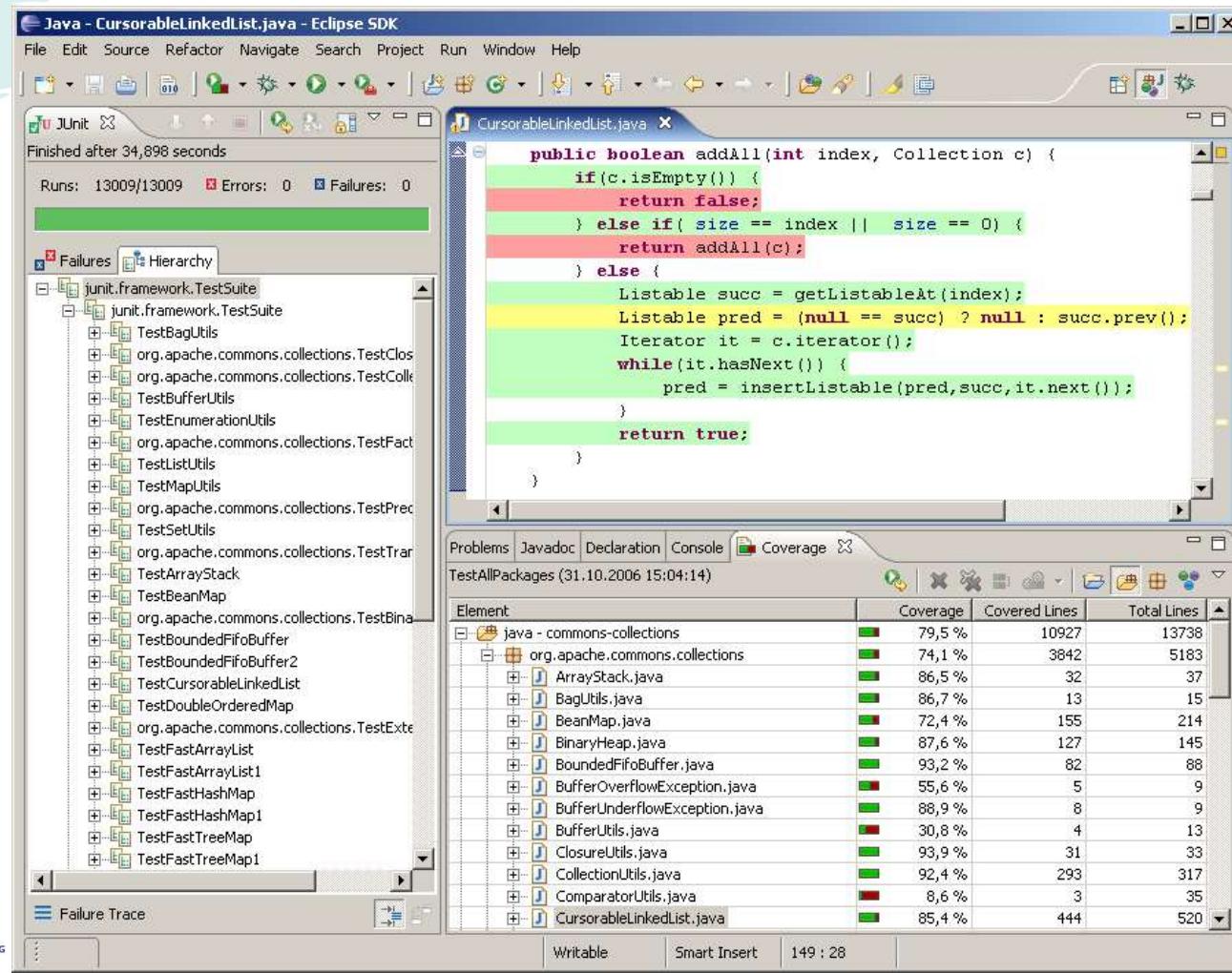
IF2250 RPL

# **CASE tools (Computer-Aided Software Engineering)**

- Software systems that are intended to provide automated support for software process activities
- CASE systems are often used for method support
- Upper-CASE
  - Tools to support the **early process** activities of **requirements** and **design**
- Lower-CASE
  - Tools to support **later activities** such as **programming**, **debugging** and **testing**

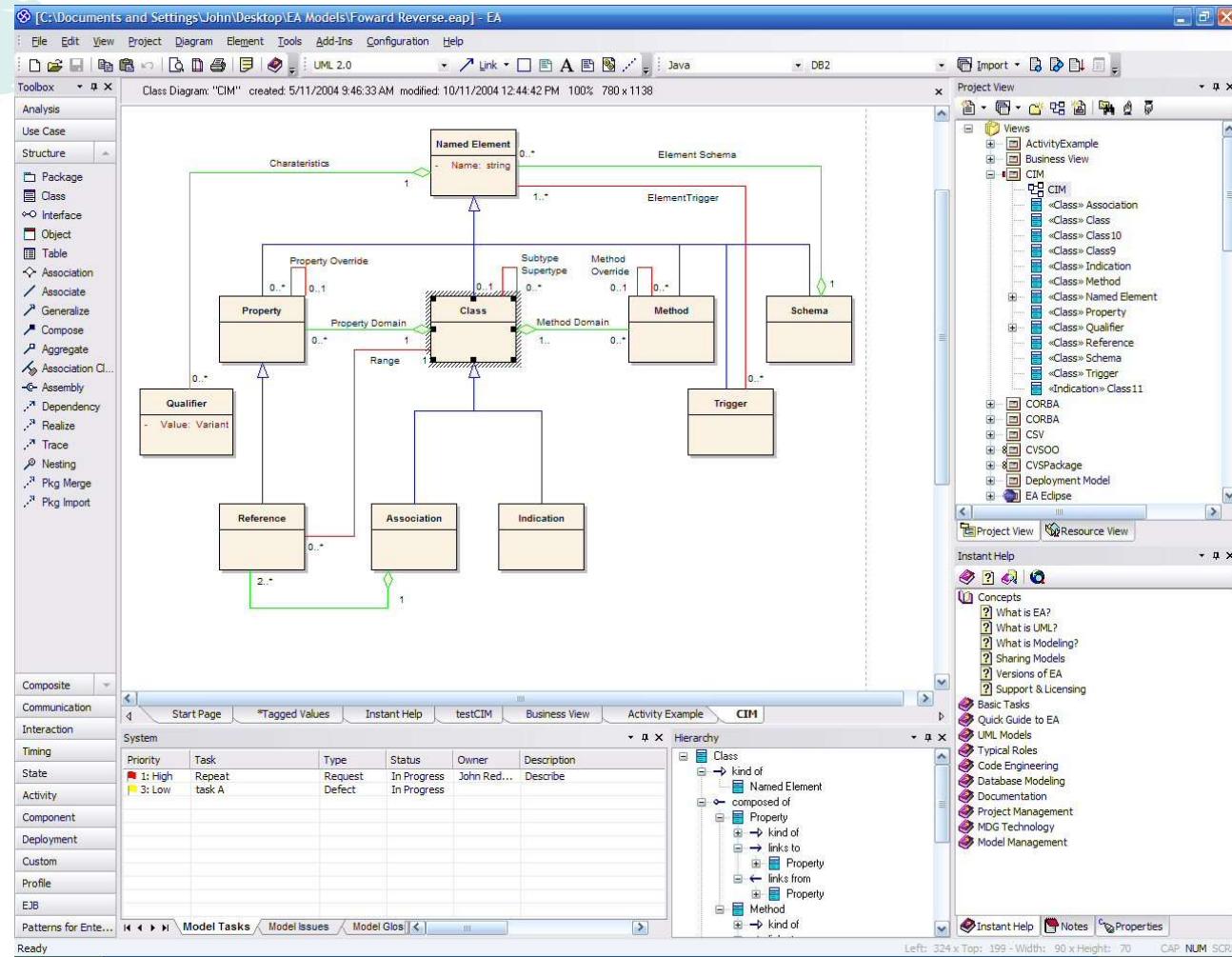
\* *Software Engineering 7<sup>th</sup> ed, Ian Sommerville*

# Contoh Case Tools untuk source code (IDE Eclipse)



# Contoh Case Tools – untuk diagram (IDE Eclipse)

23

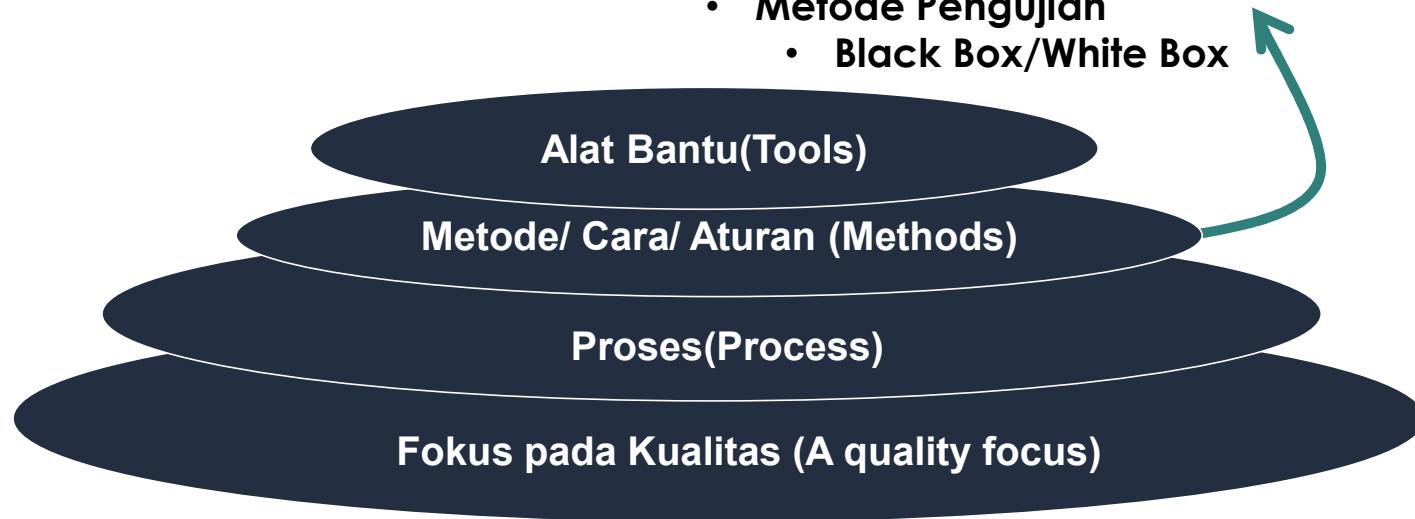


KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 RPL

# *Lapisan di RPL (2)*

- Metode Pengumpulan Kebutuhan Pengguna
  - Goal Oriented, Viewpoints, dll
- Metode Analisis
  - Terstruktur/OO
- Metode Perancangan
  - Terstruktur/OO
- Metode Pengujian
  - Black Box/White Box



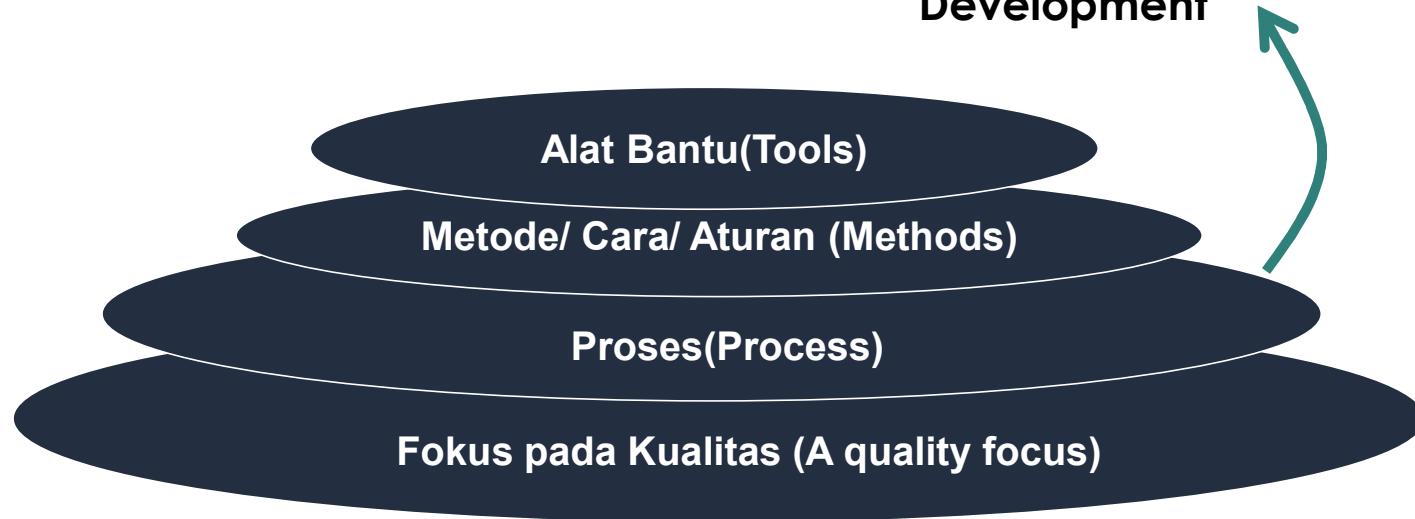
# What are *software engineering methods*?

- Structured approaches to software development which include system models, notations, rules, design advice and process guidance.
- Model descriptions
  - Descriptions of graphical models which should be produced
- Rules
  - Constraints applied to system models
- Recommendations
  - Advice on good design practice
- Process guidance
  - What activities to follow

# *Lapisan di RPL (3)*

26

- Waterfall Model
- Incremental Model/Incremental Process
- Spiral model
- Agile Development
- Rapid Application Development

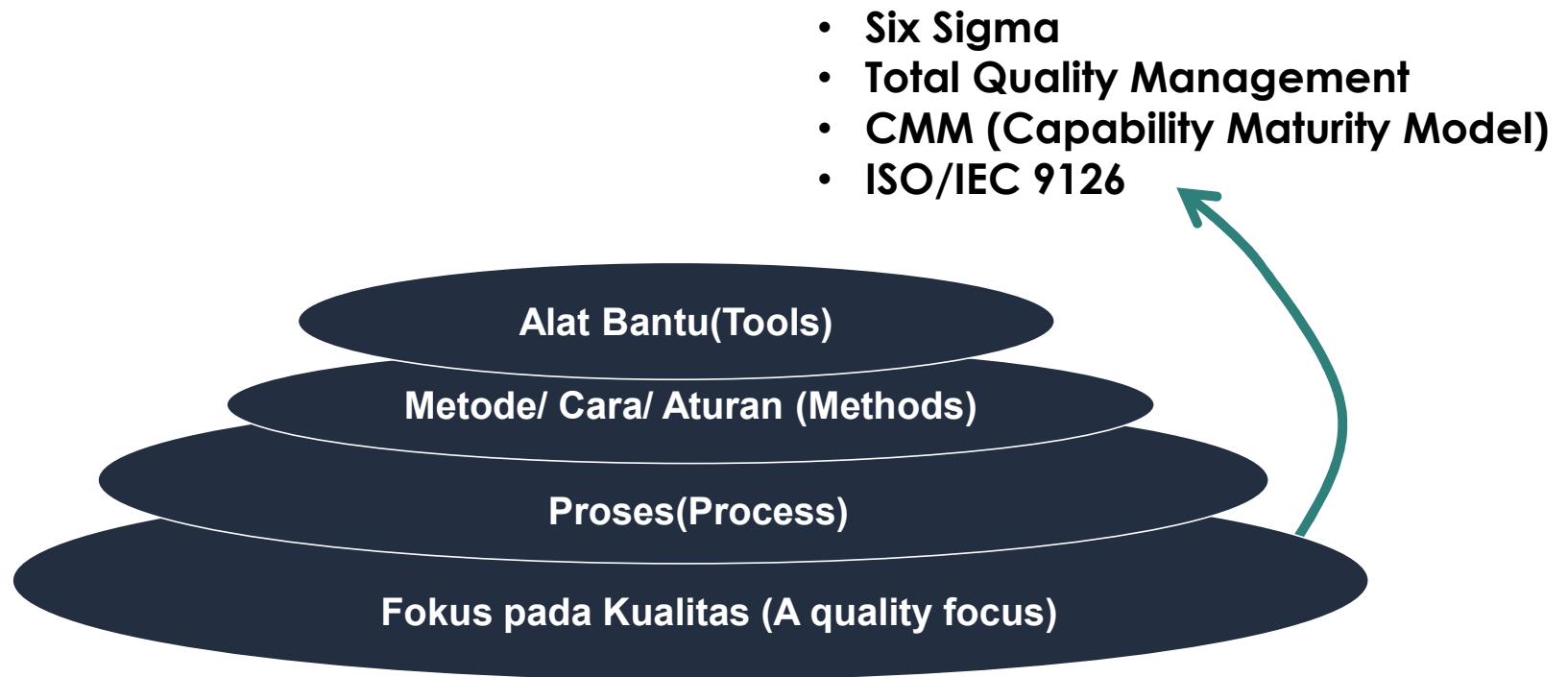


KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 RPL

# *Lapisan di RPL (4)*

27



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 RPL

# *The Software Process*

- THE PROCESS FRAMEWORK
- UMBRELLA ACTIVITIES
- PROCESS ADAPTATION



KNOWLEDGE & SOFTWARE ENGINEERING

# *Generic Software Process Framework*

- **Communication**
  - System analyst vs User
  - System analyst vs Programmer
- **Planning**
  - Cost, Time, human resources
- **Modeling**
  - Structured approach
  - Object oriented approach
- **Construction**
  - Coding and Testing
- **Deployment**
  - Software delivery to customer



KNOWLEDGE & SOFTWARE ENGINEERING

# *Umbrella Activities*

- **Software project tracking and control**
  - allows the software team to **assess progress** against the project plan and **take** any necessary **action** to maintain the schedule.
- **Risk management**
  - **assesses risks** that may affect the **outcome** of the project or the **quality** of the product.
- **Software quality assurance**
  - defines and conducts the activities required to **ensure** software **quality**.
- **Technical reviews**
  - assesses software engineering work products in an effort to **uncover** and **remove errors** before they are propagated to the next activity.

# *Umbrella Activities*

- **Measurement**

- defines and collects **process**, **project**, and **product measures** that assist the team in delivering software that meets **stakeholders' needs**; can be used in conjunction with all other framework and umbrella activities.

- **Software configuration management**

- manages the **effects of change** throughout the software process.

- **Reusability management**

- defines **criteria** for work product **reuse** (including software components) and establishes mechanisms to **achieve reusable** components.

- **Work product preparation and production**

- encompasses the activities required to create work products such as **models**, **documents**, **logs**, **forms**, and **lists**.

# *Process Adaptation*

- The software engineering process should be agile and adaptable
  - to the problem,
  - to the project,
  - to the team, and
  - to the organizational culture
- A process adopted for one project might be significantly different than a process adopted for another project.



KNOWLEDGE & SOFTWARE ENGINEERING

# *The essence of software engineering practice*

1. Understand the problem (communication and analysis).
2. Plan a solution (modeling and software design).
3. Carry out the plan (code generation).
4. Examine the result for accuracy (testing and quality assurance).



KNOWLEDGE & SOFTWARE ENGINEERING

# Software Practice Core Principles

- **The reason it all exist**
  - Software exists **to provide value** to its users
- **Keep it simple stupid (KISS)**
  - Keep the design as **simple as possible**, but not simpler
- **Maintain the vision**
  - **Clear vision** is essential to the success of any software project
- **We produce, others will consume**
  - Always specify, design, and implement knowing that **someone else will have to understand** what you **have done** to **carry out** his or her tasks
- **Open to the future**
  - Be **open to future changes**, don't code yourself into a corner
- **Plan for Reuse!**
  - Planning ahead for **reuse** reduces the cost and increases the value of both the reusable components and the systems that require them
- **Think First!**
  - Placing **clear** complete **thought** before any action almost always produces better results

# *Software Process Structure*

PROCESS FLOW

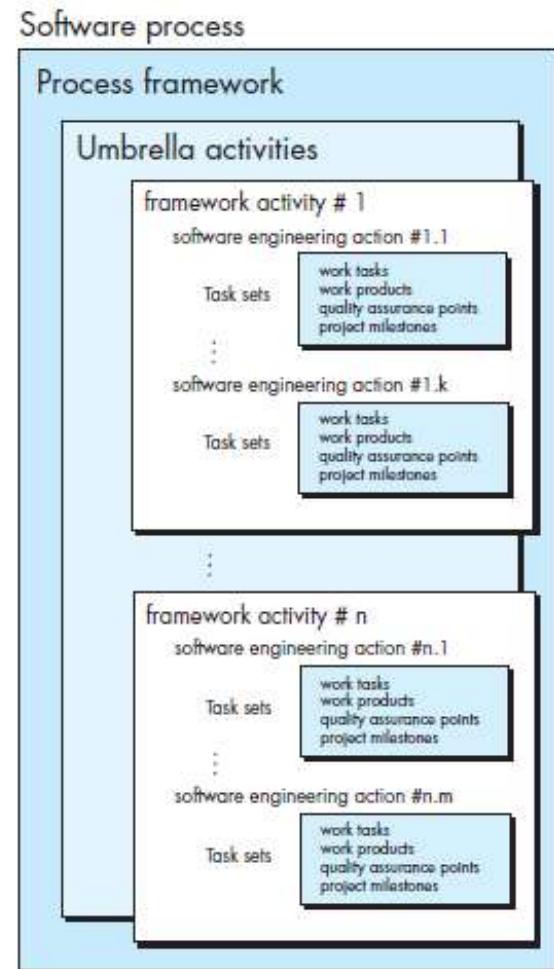


KNOWLEDGE & SOFTWARE ENGINEERING



# *A Software Process Framework*

36



# ***One additional aspect of the software process: Process flow***

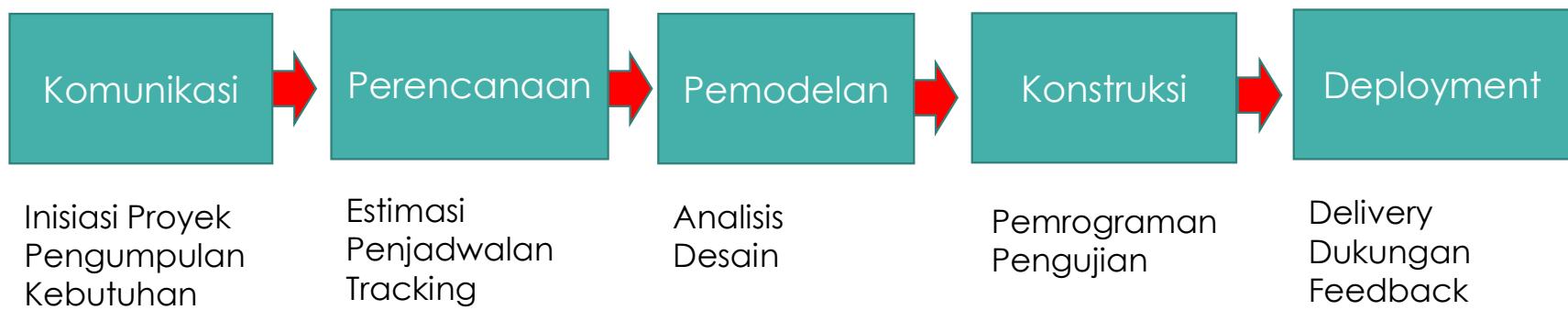
- Process framework:
  - communication,
  - planning,
  - modeling,
  - construction,
  - deployment ,
  - umbrella activities + process flow
- Organized with respect to sequence and time
  - *linear process flow*
  - *iterative process flow*
  - *evolutionary process flow*
  - *parallel process flow*



KNOWLEDGE & SOFTWARE ENGINEERING

# Process Flow (I)

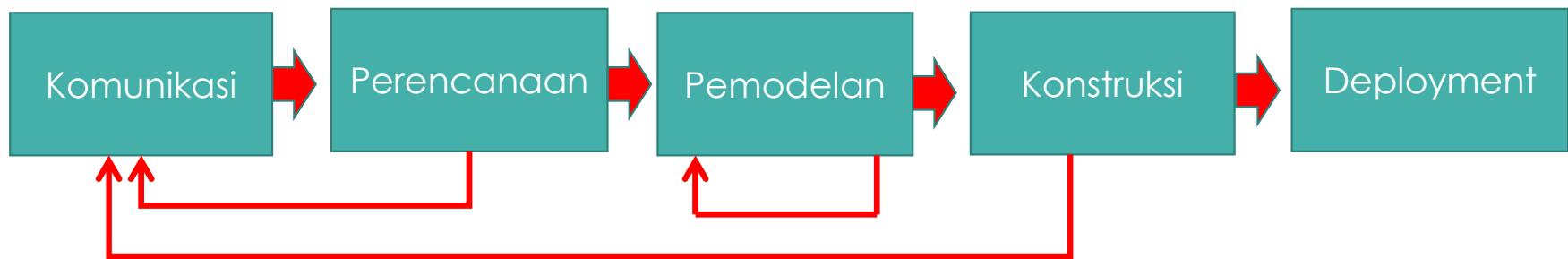
## Alur Proses Linear (*Linear Process Flow*)



KNOWLEDGE &amp; SOFTWARE ENGINEERING

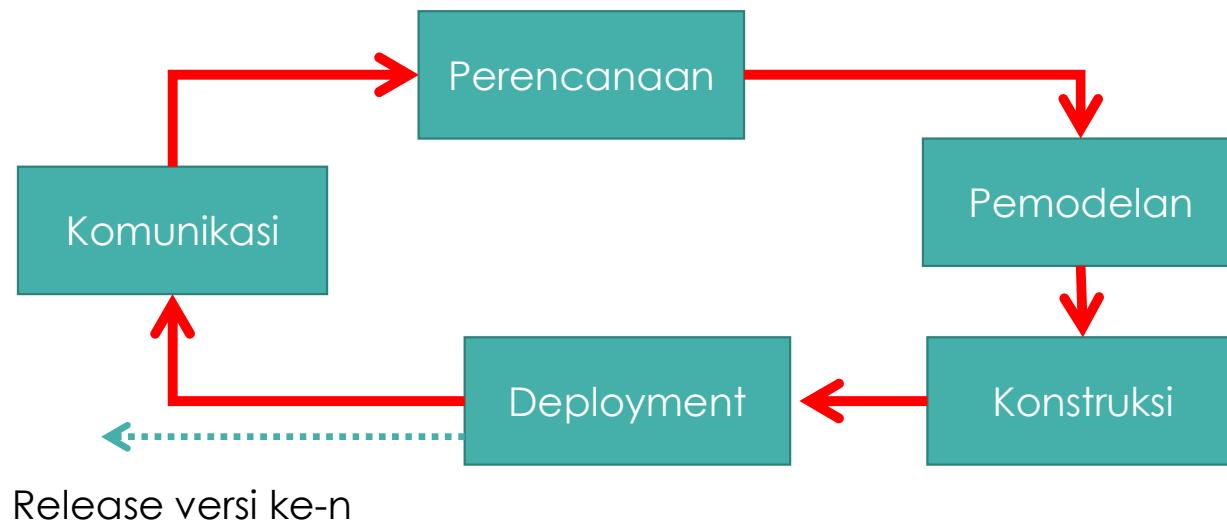
# Process Flow (2)

## Alur Proses Iteratif (*Iterative Process Flow*)



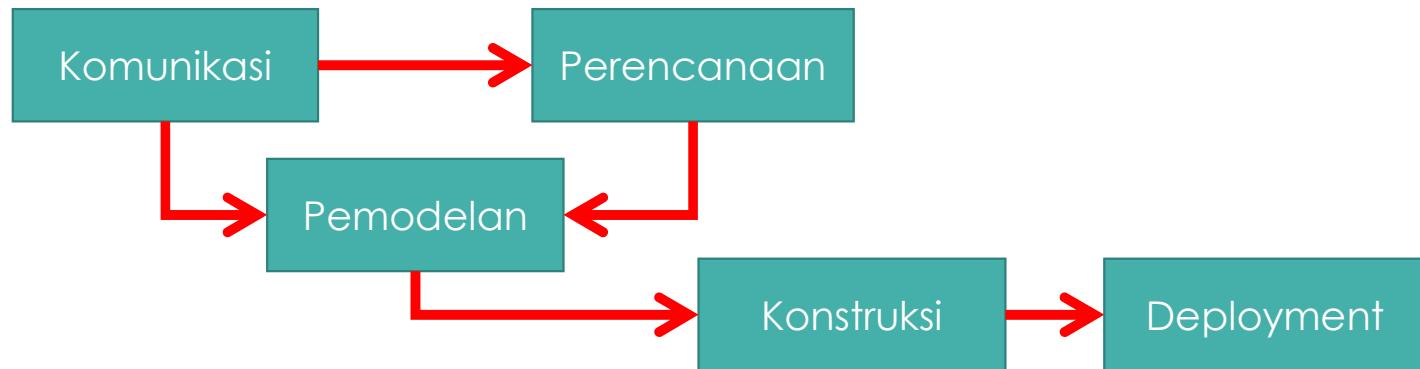
# Process Flow (3)

## Alur Proses Berevolusi (*Evolutionary process flow*)



# Process Flow (4)

## Alur Proses Paralel (*Parallel process flow*)



# *Process Models*

- PRESCRIPTIVE PROCESS MODELS
- SPECIALIZED PROCESS MODELS
- UNIFIED PROCESS

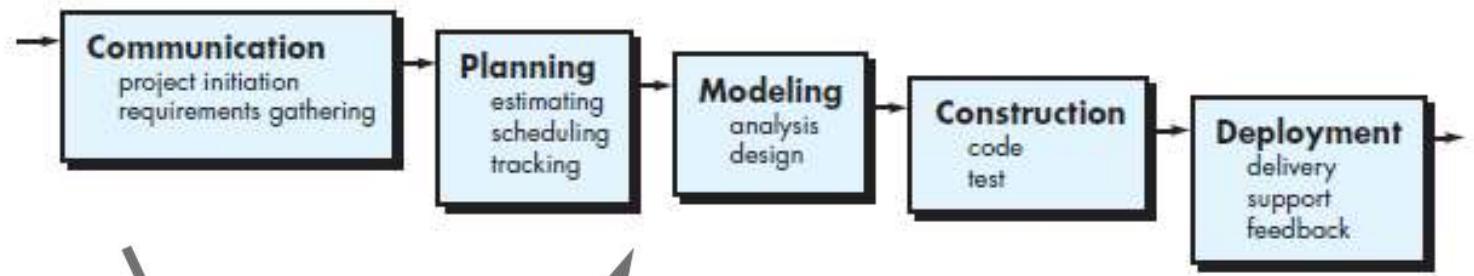


KNOWLEDGE & SOFTWARE ENGINEERING

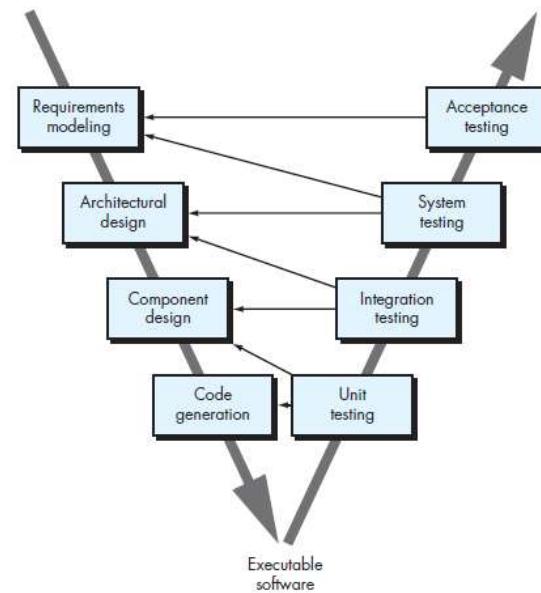


# Prescriptive Process Models

- The Waterfall Model - classic life cycle



- The V-model



# Karakteristik Waterfall

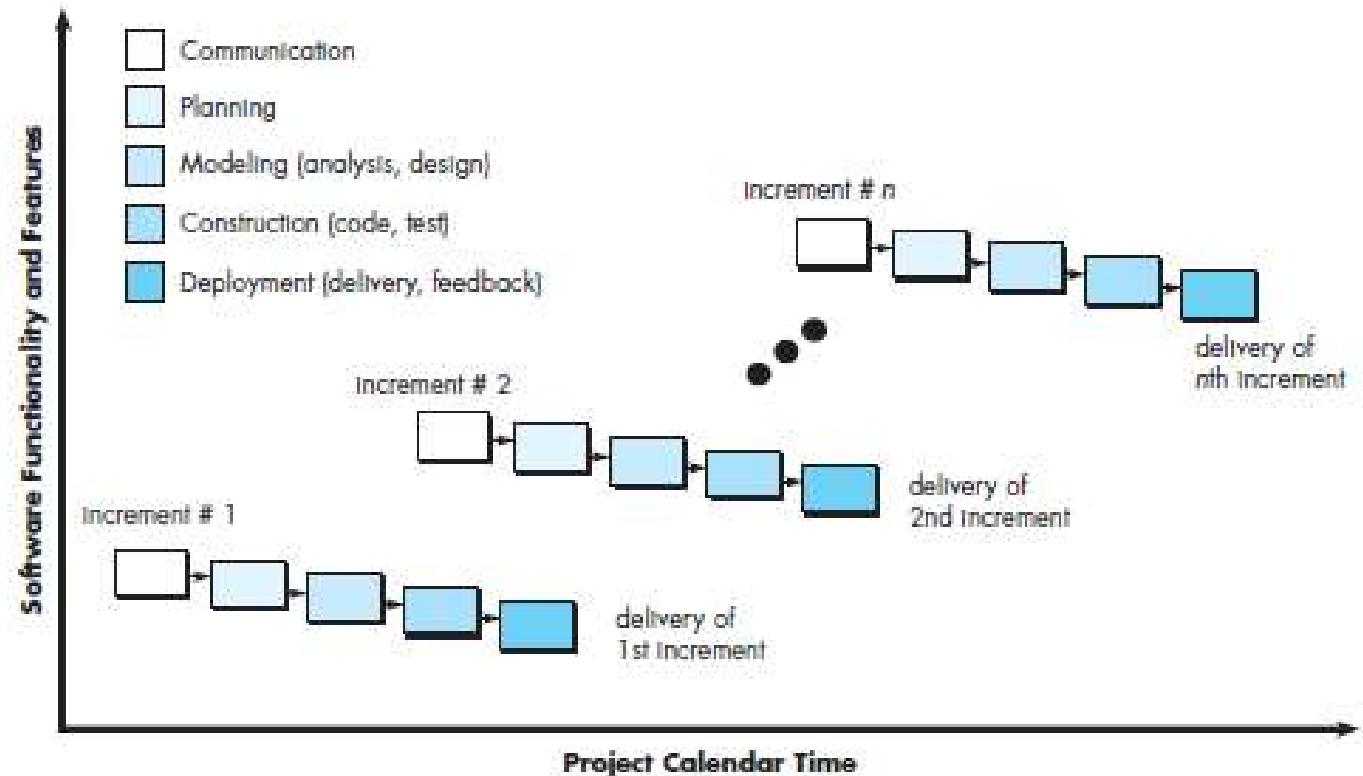
- Proses dijalankan secara sekuensial dari pengumpulan kebutuhan hingga perawatan
- Cocok untuk sistem yang sudah terdefinisi baik atau sistem yang mengutamakan keselamatan (safety)
  - Pengembangan auto-pilot untuk pesawat harus jelas dan lengkap di awal, jadi program harus sudah lengkap tidak bisa hanya sebagian yang di instalasi di pesawat.



KNOWLEDGE & SOFTWARE ENGINEERING

# *Prescriptive Process Models (2)*

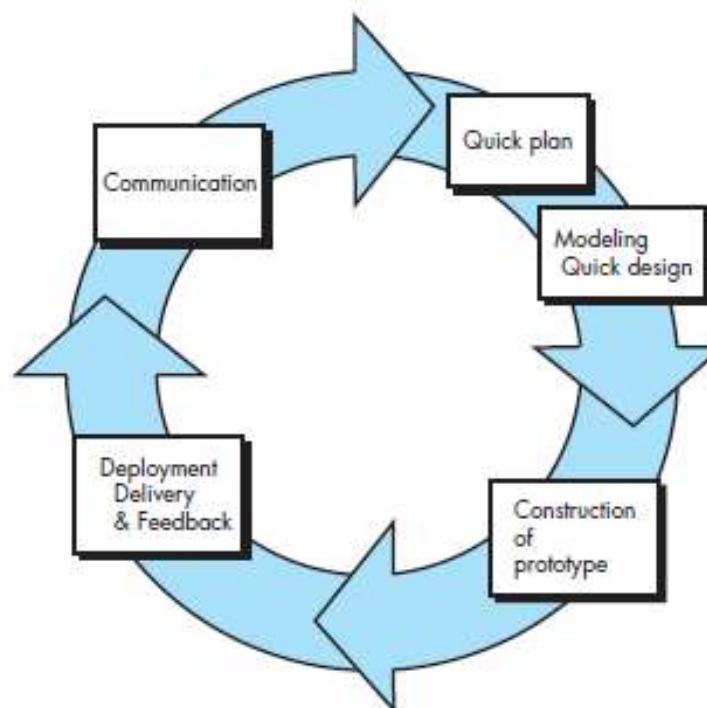
- Incremental Process Models



KNOWLEDGE & SOFTWARE ENGINEERING

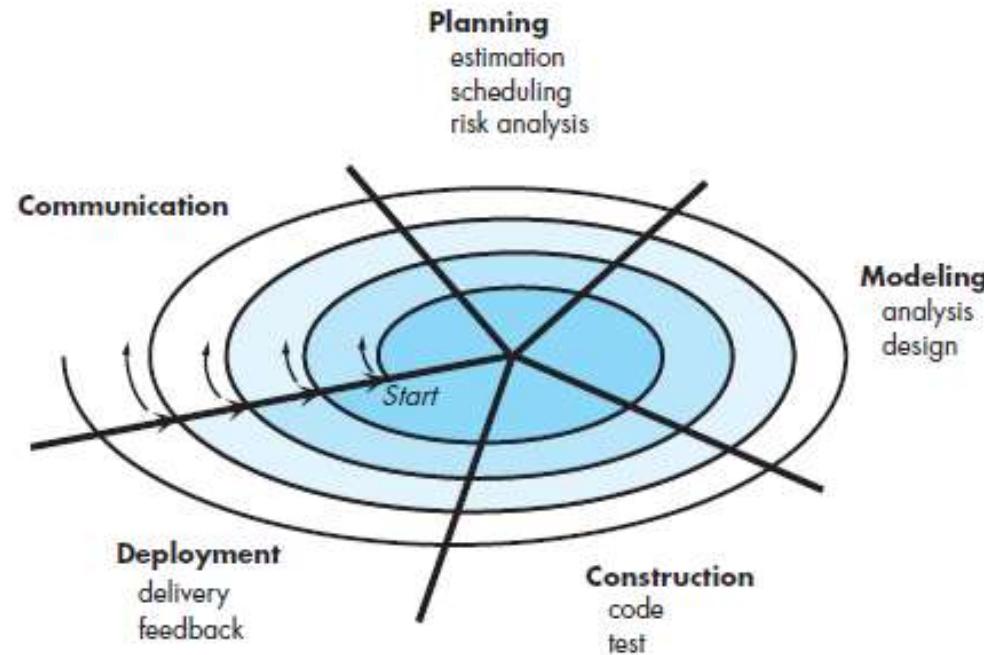
# *Prescriptive Process Models (3)*

- Evolutionary Process Models – prototyping paradigm



# Prescriptive Process Models (4)

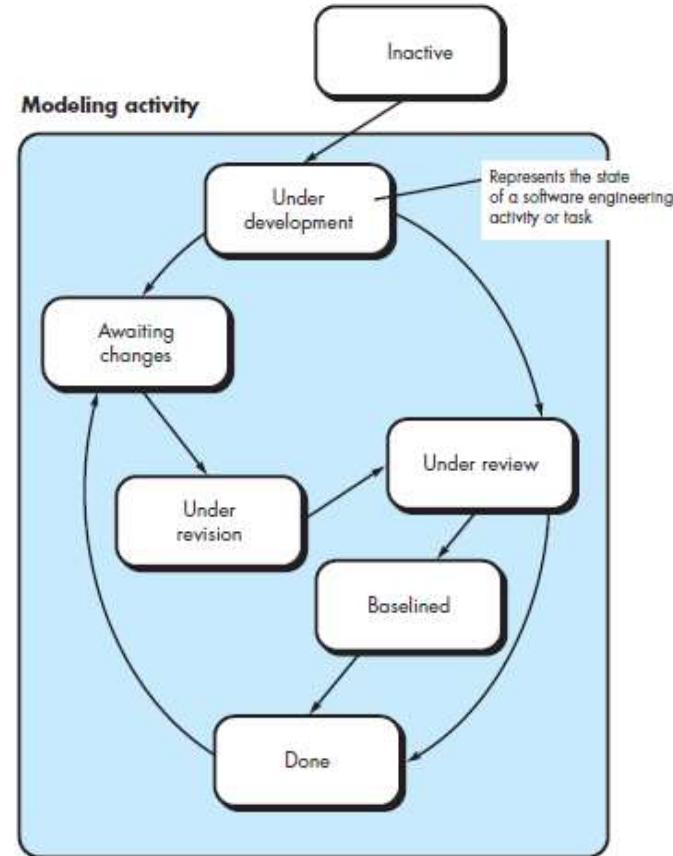
- Evolutionary Process Models – the Spiral Model



KNOWLEDGE & SOFTWARE ENGINEERING

# Prescriptive Process Models (5)

- Concurrent Models



KNOWLEDGE & SOFTWARE ENGINEERING

# Specialized process models

- **Component-Based Development** - comprises applications from prepackaged software components.
- **The Formal Methods Model**
  - encompasses a set of activities that leads to formal mathematical specification of computer software, enable you to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation.
  - the formal methods approach has gained adherents among software developers who must build safety-critical software (e.g., developers of aircraft avionics and medical devices) and among developers that would suffer severe economic hardship should software errors occur
- **Aspect-Oriented Software Development**
  - often referred to as aspect-oriented programming (AOP) or aspect-oriented component engineering
  - a relatively new software engineering paradigm that provides a process and methodological approach for defining, specifying, designing, and constructing aspects —“mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concern”



KNOWLEDGE & SOFTWARE ENGINEERING

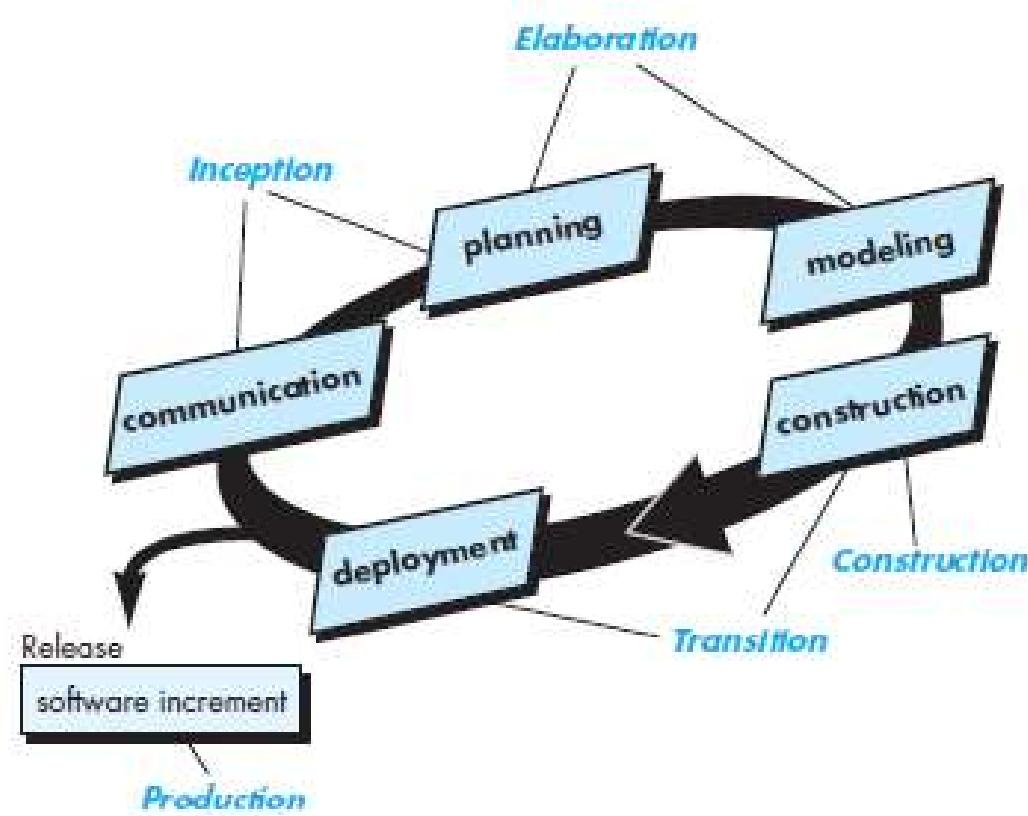
# Unified Process

- a “use case driven, architecture-centric, iterative and incremental”
- The result was UML—a *unified modeling language that contains a robust notation for the modeling and development of object- oriented systems.*



KNOWLEDGE & SOFTWARE ENGINEERING

# Unified Process (2)



KNOWLEDGE & SOFTWARE ENGINEERING

# *Agile - Scrum*



# ***System Engineering vs Software Engineering***



KNOWLEDGE & SOFTWARE ENGINEERING



# *System – Definition*

## *Webster's Dictionary*

- A set or **arrangement of things** so related as to form a unity or organic whole
- A set of facts, principles, rules, etc., **classified and arranged** in an orderly form so as to show a logical plan linking the various parts
- A method or plan of classification or arrangement
- An established way of doing something; method; procedure....
- .....
- ....



KNOWLEDGE & SOFTWARE ENGINEERING

# **Computer-Based Systems**

## **[PRE2007]**

- A set or arrangement of elements that are organized to accomplish some predefined goal by **processing information**
- The goal:
  - To support some business function or to develop a product that can be sold to **generate business revenue**
- To accomplish the goal, a computer-based system makes use of a variety of **system elements**



KNOWLEDGE & SOFTWARE ENGINEERING

# *Computer-Based System Elements*

- Software
- Hardware
- People
- Data
- Documentation
- Procedures

\* SEPA 6<sup>th</sup> ed, Roger S. Pressman



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 RPL

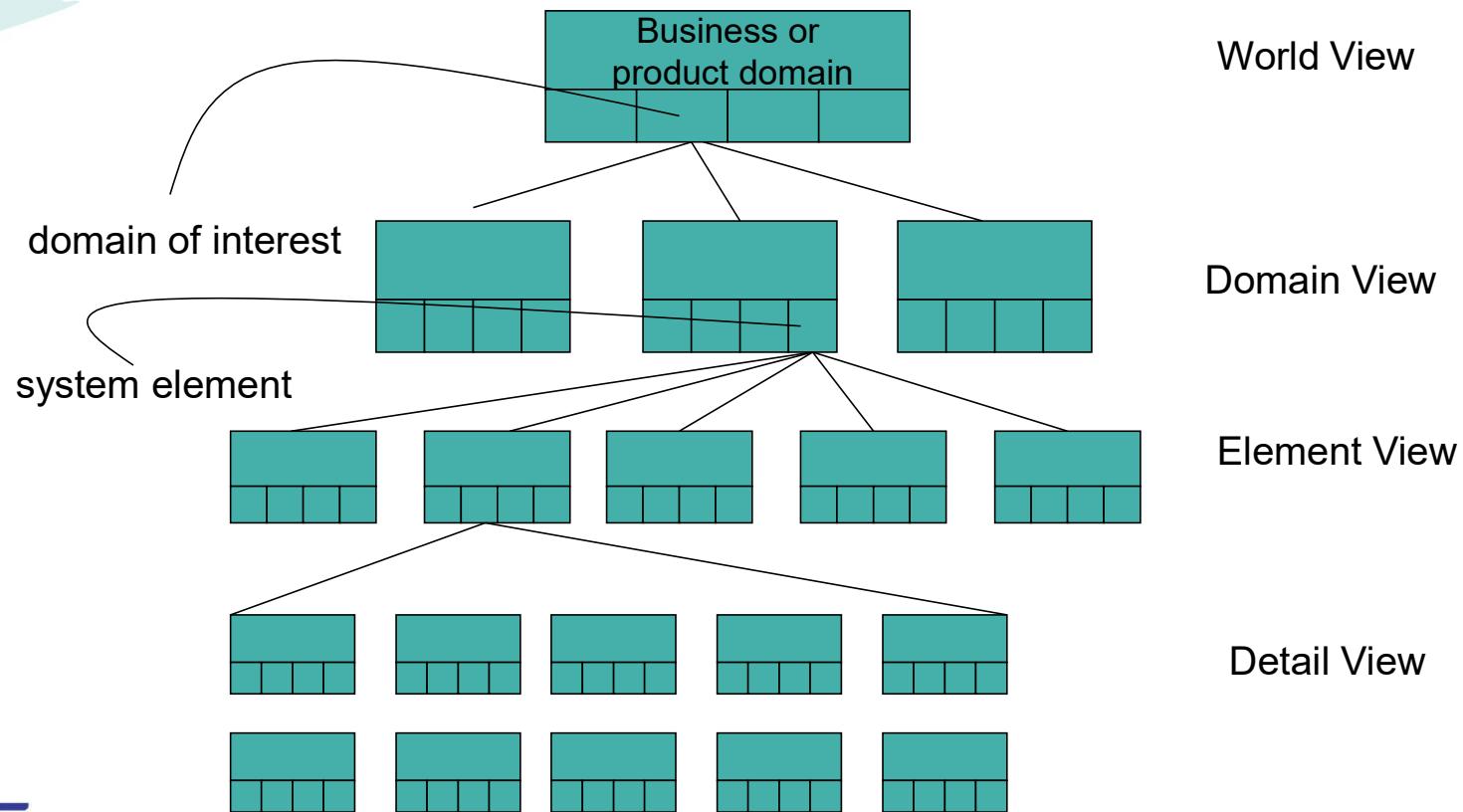
# *System Engineering Hierarchy*

- World view → WV = {D<sub>1</sub>, D<sub>2</sub>, D<sub>3</sub>, ..., D<sub>n</sub>}
  - Composed of a set of domains (D<sub>i</sub>) which can be each be a system or system of systems
- Domain view → DV = {E<sub>1</sub>, E<sub>2</sub>, E<sub>3</sub>, ..., E<sub>m</sub>}
  - Composed of specific elements (E<sub>j</sub>) each of which serves some role in accomplishing the objective and goals fo the domain or component
- Element view → EV = {C<sub>1</sub>, C<sub>2</sub>, C<sub>3</sub>, ..., C<sub>k</sub>}
  - Each element is implemented by specifying the technical component (C<sub>k</sub>) that achieve the necessary function for an element
- Detail view

\* SEPA 6<sup>th</sup> ed, Roger S. Pressman

# *System Engineering Hierarchy*

58



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 RPL

# Product Engineering

- Goal
  - to translate the customer's desire for a set of defined capabilities into a working product
- Hierarchy
  - Requirements engineering (world view)
  - Component engineering (domain view)
  - Analysis and Design modeling (element view - **software engineers**)
  - Construction and Integration (detailed view - **software engineers**)

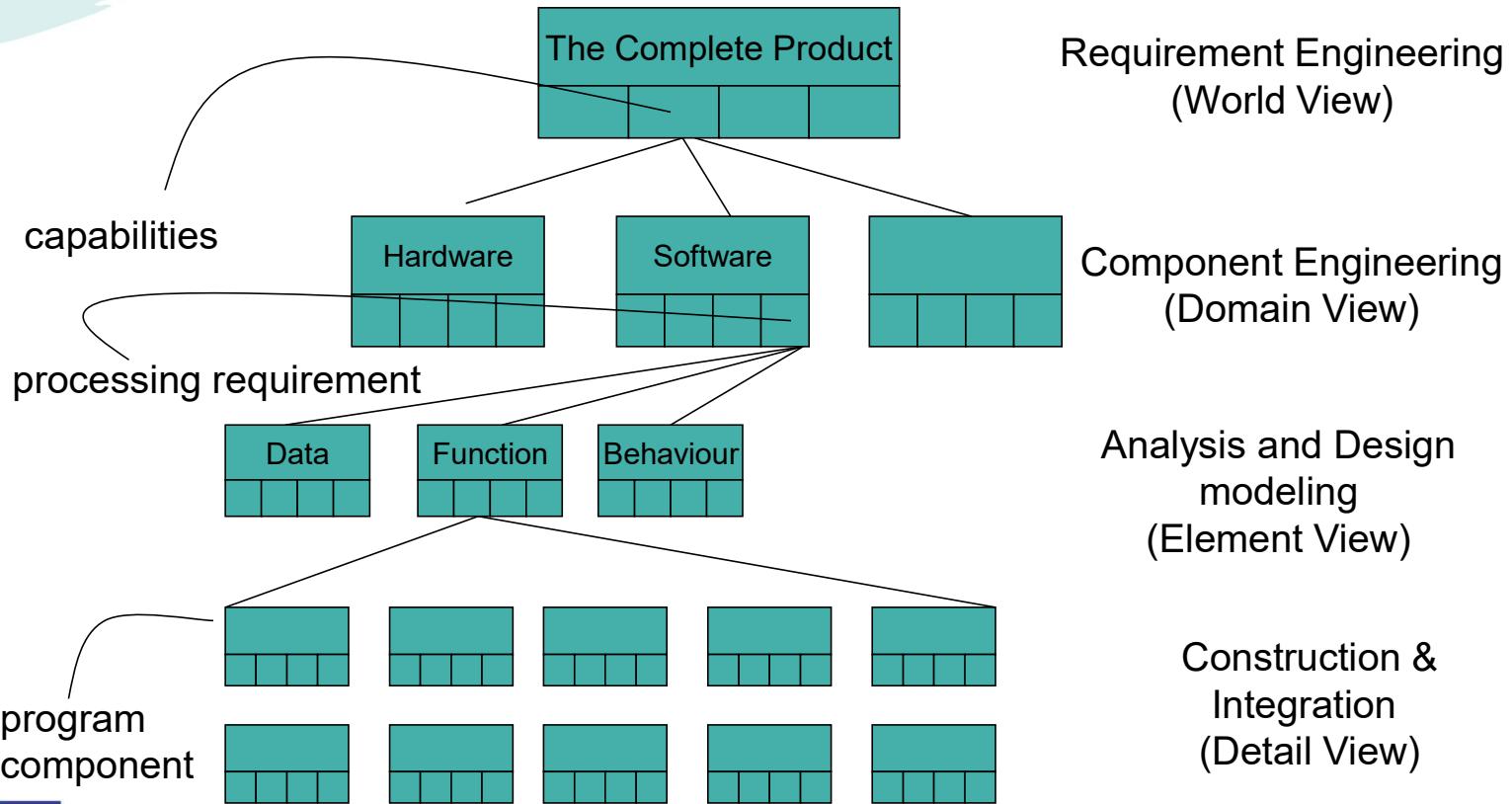


KNOWLEDGE & SOFTWARE ENGINEERING

\* *SEPA 6<sup>th</sup> ed, Roger S. Pressman*

IF2250 RPL

# The Product Engineering Hierarchy

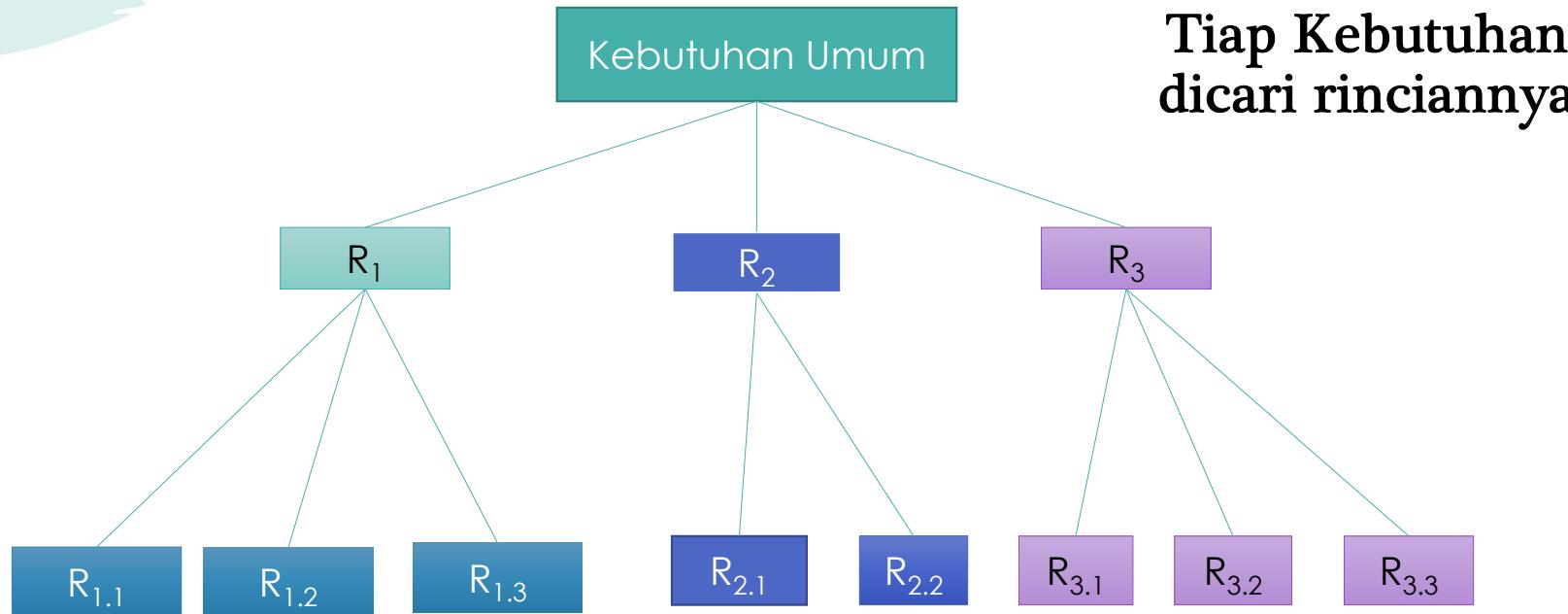


# *Aktivitas Pengembangan Perangkat Lunak*



KNOWLEDGE & SOFTWARE ENGINEERING

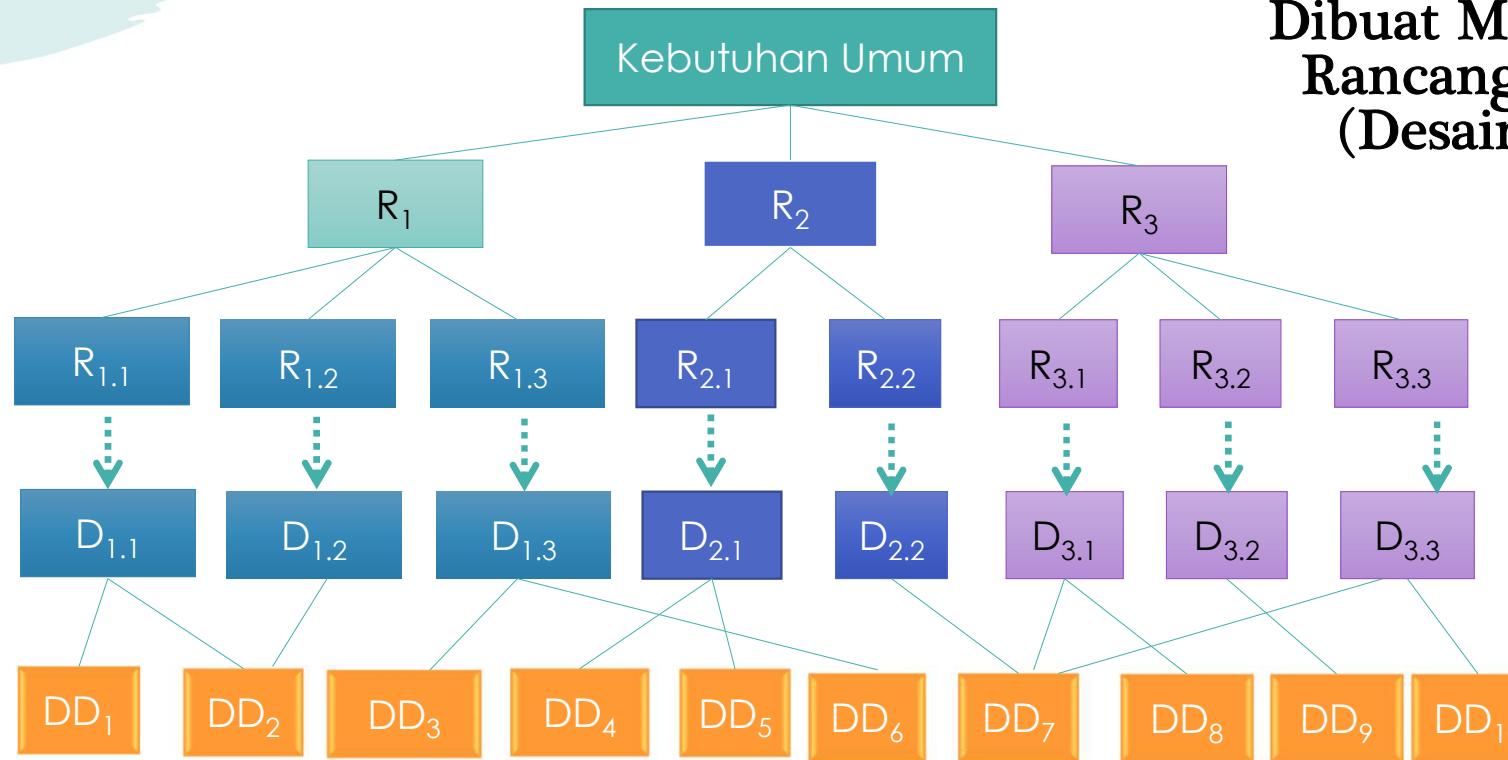
# Pengumpulan Kebutuhan(**Requirements**)



Hingga cukup detil!

Tapi sampai kapan kita memecah kebutuhan?

# *Dari Hasil Pengumpulan Kebutuhan*

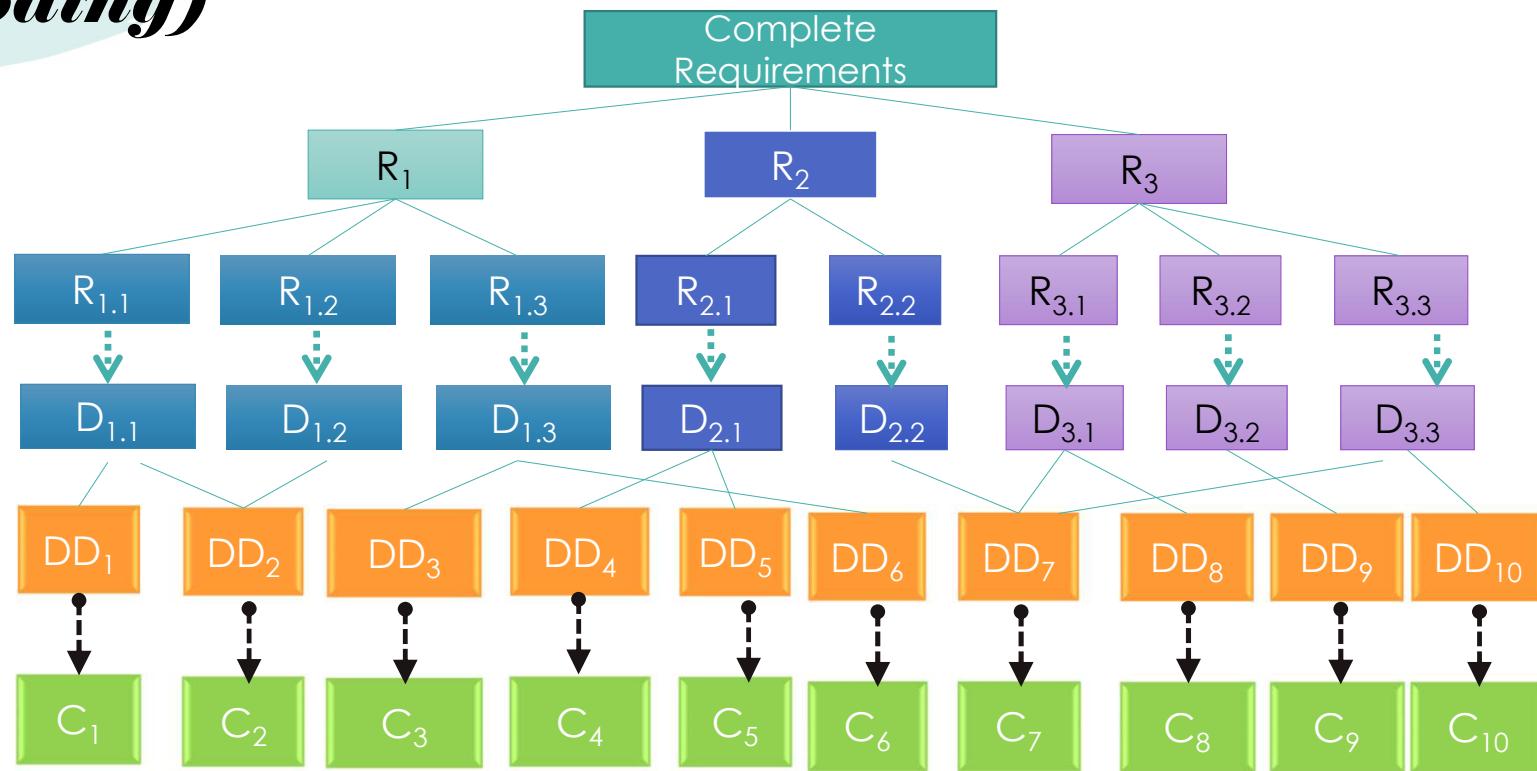


**Dibuat Model  
Rancangan  
(Desain)**

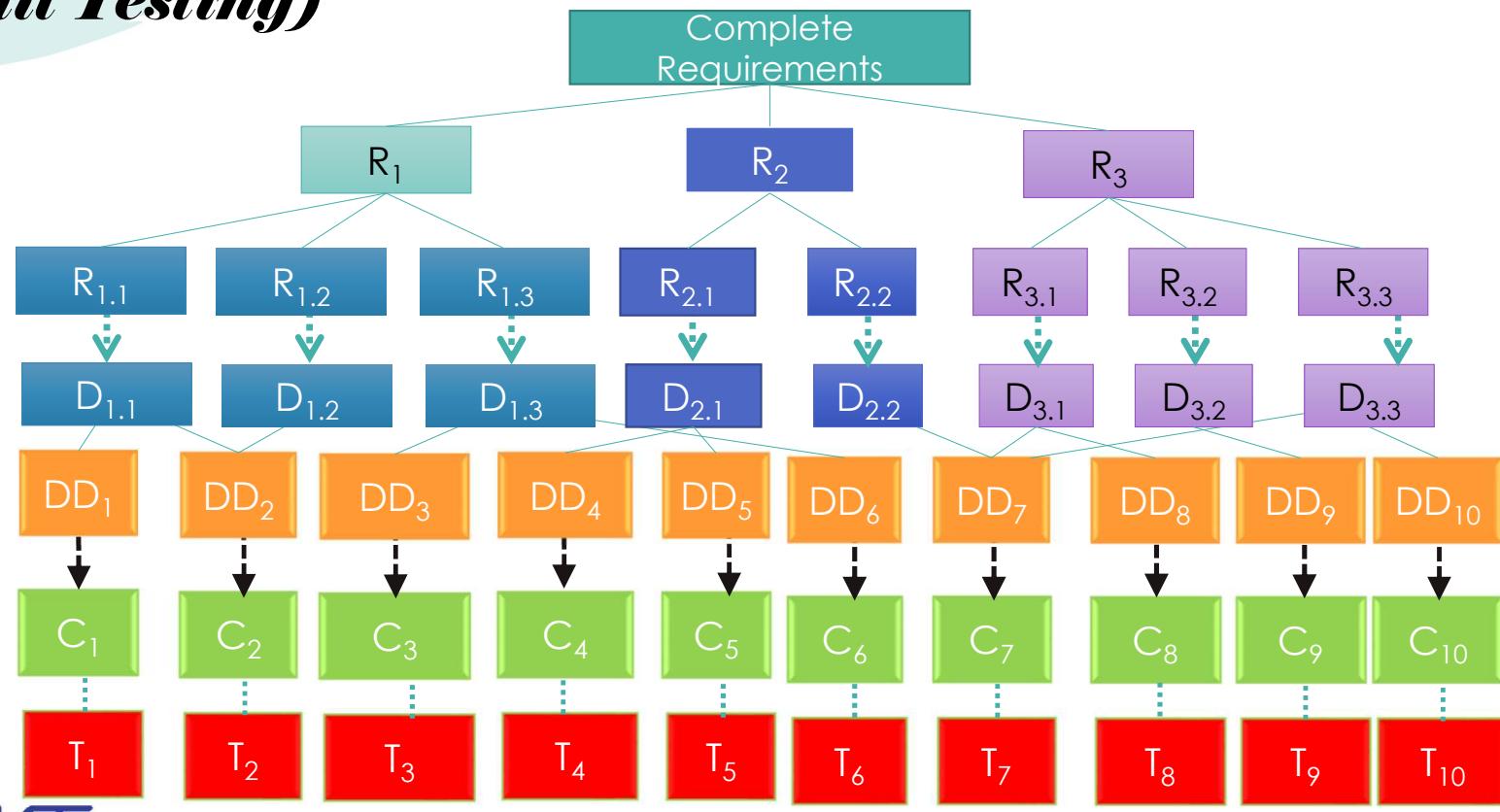
**Dari Rancangan Umum (Global) Hingga Lebih Rinci  
(Detil)**

# *Dari Perancangan hingga Pemrograman (Coding)*

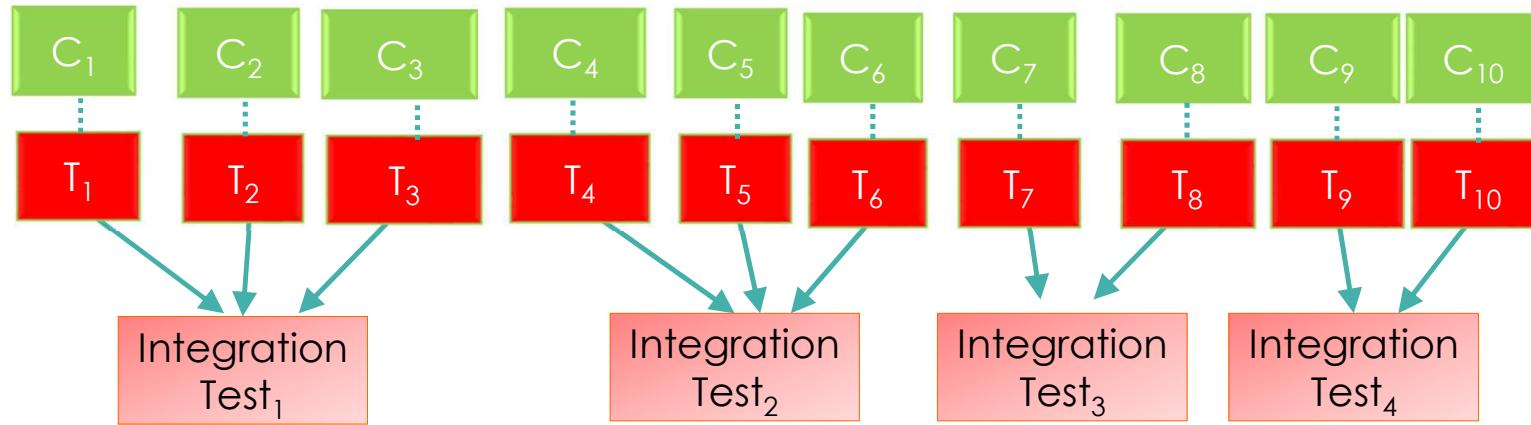
64



# *Setiap unit kode program harus diuji (Unit Testing)*

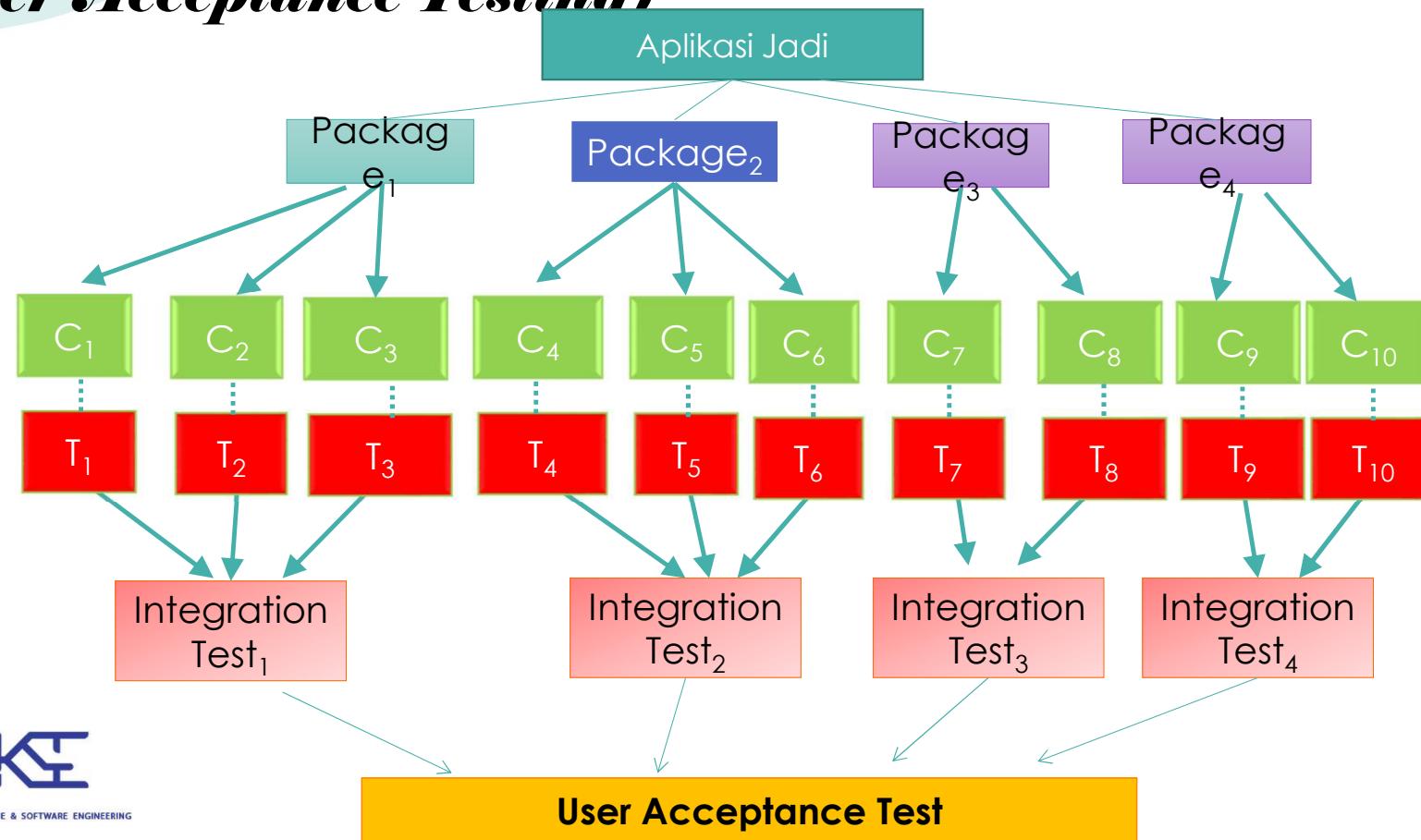


# *Setiap unit program harus digabung dan hasil penggabungannya di uji kembali (Integration Testing)*



Setiap hasil integrasi akan diuji, hingga kita mendapatkan pengujian yang lengkap, artinya semua unit sudah menjadi satu, dan dilakukan pengujian secara keseluruhan

## **Pengujian Lengkap di depan calon pengguna disebut Pengujian Penerimaan Pengguna (User Acceptance Testing)**



Tim Pengajar IF2250

# IF2250 – Rekayasa Perangkat Lunak Rekayasa Kebutuhan

SEMESTER II TAHUN AJARAN 2022/2023



KNOWLEDGE & SOFTWARE ENGINEERING



# *Proses Perangkat Lunak*

- **Komunikasi (Communication)**

- Antara Sistem Analis dengan Pengguna
- Antara Sistem Analis dengan Pemrogram

- **Perencanaan (Planning)**

- Perencanaan Biaya, Waktu dan Sumber daya (manusia/barang)

- **Pemodelan (Modeling)**

- Pendekatan Terstruktur (Structured approach)
- Pendekatan Berorientasi Objek (Object oriented approach)

- **Konstruksi (Construction)**

- Pemrograman/Pengujian (Coding and Testing)

- **Deployment**

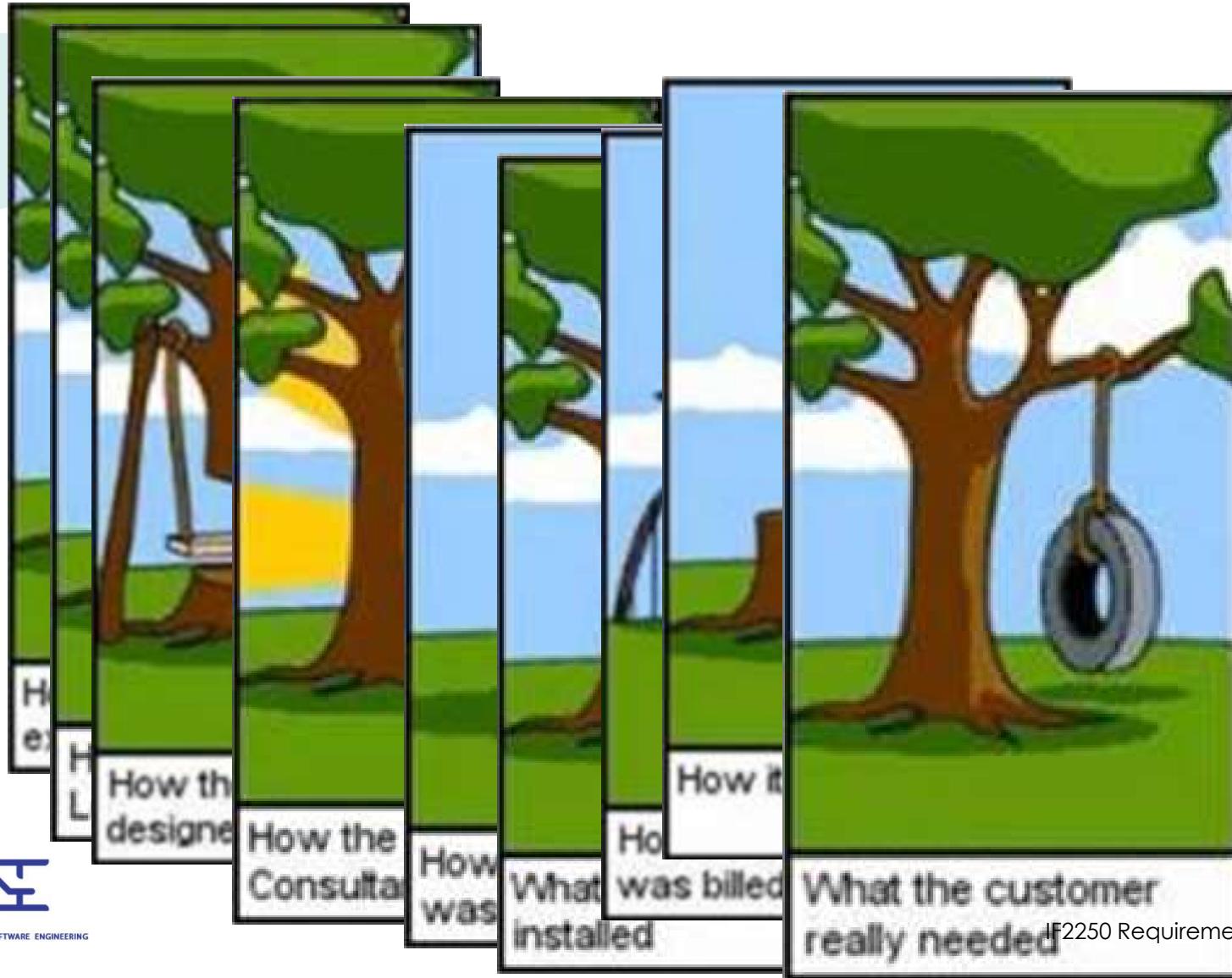
- Penyerahan dan instalasi program ke pengguna/pelanggan (user/customer)

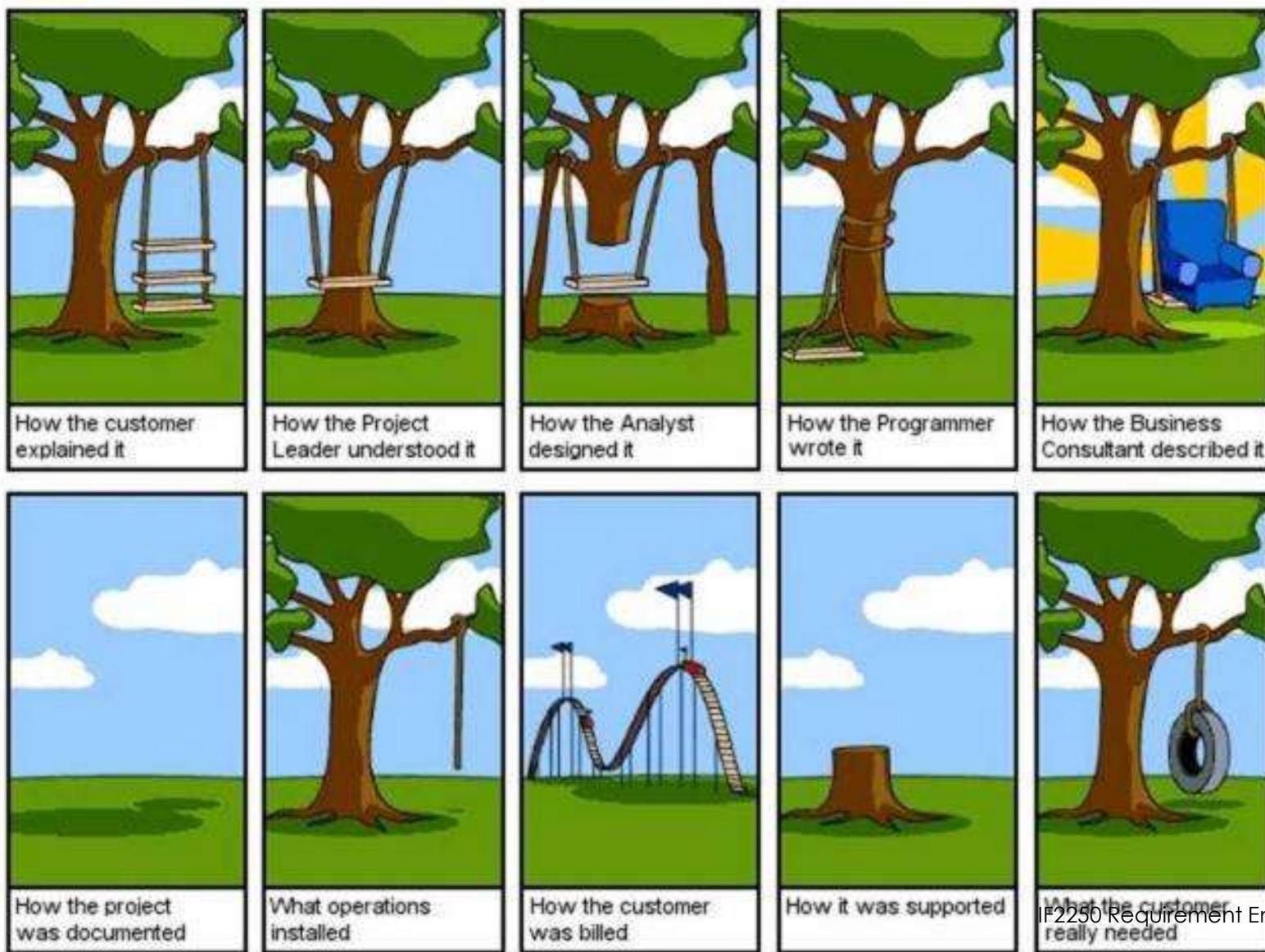
# *Apa yang terjadi bila terjadi ketidak sempurnaan dalam proses perangkat lunak?*

3

- **Contoh “ketidak sempurnaan”:**

- Komunikasi antara anggota tidak lancar
- Perencanaan pekerjaan yang salah
- Pemodelan masalah yang tidak sesuai dengan kondisi yang sebenarnya
- Pemrograman yang tidak mengikuti standard
- Cara pengujian yang tidak tepat
- Deployment yang tidak ikut aturan





# *Rekayasa Kebutuhan (Requirements Engineering)*



# *Requirements Engineering*

- Membantu pengembang (calon pengembang) mengerti masalah yang akan dibuatkan solusinya dalam bentuk perangkat lunak
  - Masalah pengguna dituangkan dalam bentuk tertulis
- Tahapan ini adalah bagian dari proses “Komunikasi” yang berlanjut pada “Pemodelan”
  - Proses “Perencanaan” dapat dilakukan makin rinci setelah “Kebutuhan” makin lengkap

# **Definisi Rekayasa Kebutuhan**

*“The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.*

*The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.”*

[Ian Sommerville]

- Rekayasa Kebutuhan: Proses membentuk layanan yang dibutuhkan pelanggan dari suatu sistem dan juga batasan sistem akan beroperasi dan dikembangkan
- Kebutuhan adalah deskripsi dari layanan sistem dan batasan (constraint) sistem yang dihasilkan selama proses rekayasa kebutuhan.

# Apa itu Kebutuhan? (*Requirements*)

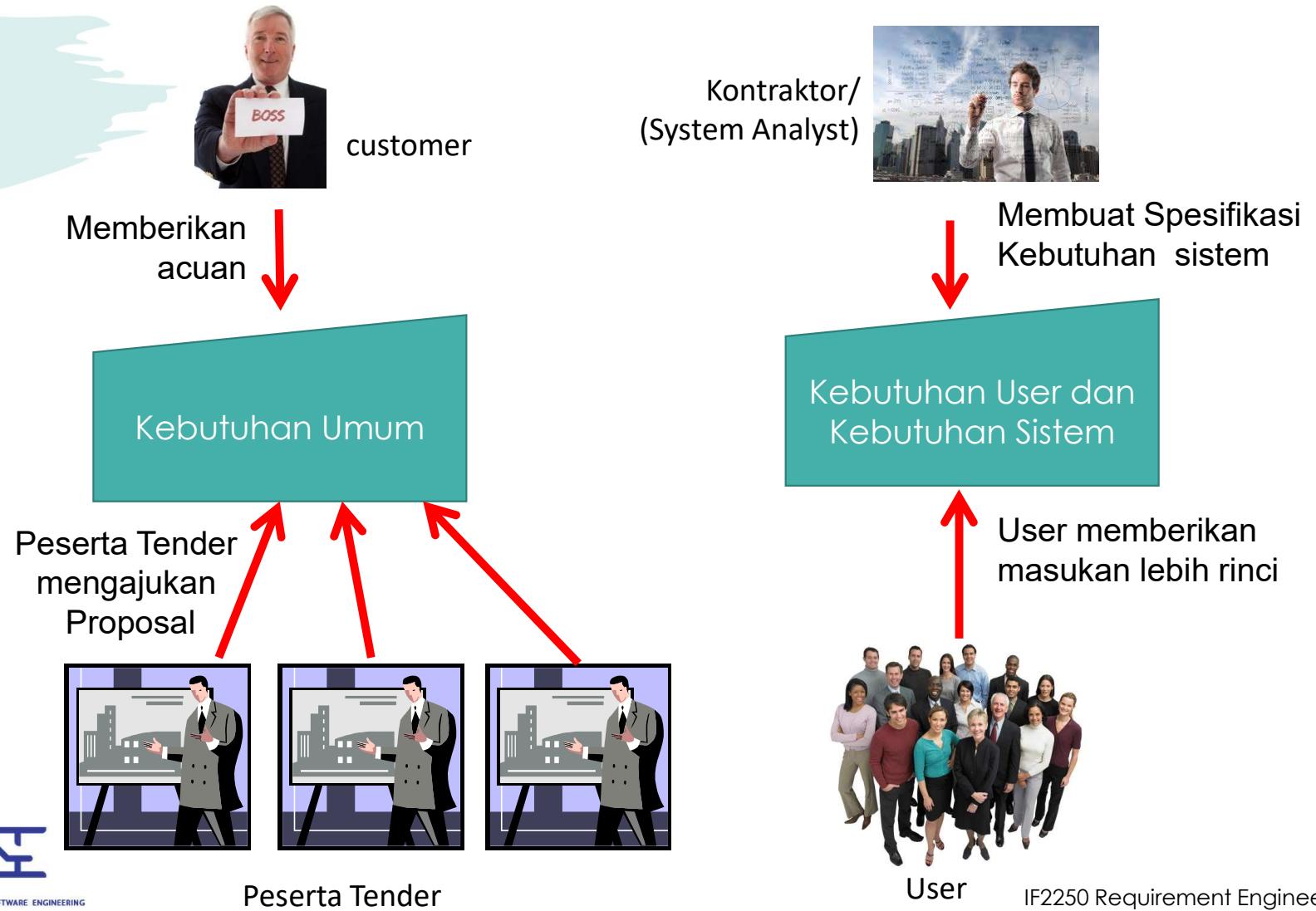
- Istilah
  - Kebutuhan (*requirements*) atau
  - Pernyataan Kebutuhan (*requirements statements*)
    - Kedua istilah ini mungkin digunakan bersamaan pada kuliah ini, tetapi maksudnya sama.
- Bentuk Kebutuhan/Pernyataan kebutuhan
  - Kalimat abstrak yang cenderung terdengar ‘high level’
    - yang masih di awang-awang, tidak rinci
  - Kalimat yang lengkap dan rinci, tapi mungkin justru tidak memberikan gambaran yang jelas secara keseluruhan
  - Bentuk fungsi matematika

# *Manfaat Pernyataan Kebutuhan*

- Pernyataan kebutuhan ini memiliki dua manfaat
  - Dasar untuk membuka/mengikuti tender pekerjaan
    - Pengembang perangkat lunak biasanya mengikuti ‘tender’ untuk mendapatkan suatu pekerjaan
    - Sifat pernyataan untuk kebutuhan ini ‘open to interpretation’
  - Dasar untuk pembuatan kontrak pekerjaan
    - Kontrak memiliki kekuatan hukum, pengembang harus mengikuti panduan yang ada di kontrak dan sekaligus merupakan jaminan bagi pelanggan/pengguna untuk mendapatkan perangkat lunak sesuai keinginannya.
- Keduanya disebut sebagai “**Requirements**” atau “**Kebutuhan**”

# *Abstraksi Kebutuhan [Davis 95]*

- Jika suatu perusahaan ingin mengembangkan kontrak untuk pengembangan perangkat lunak, maka perusahaan ini harus mendefinisikan kebutuhannya dengan cukup ‘abstrak’ sedemikian rupa sehingga kebutuhan itu harus cukup fleksibel bagi calon peserta tender.
  - Harapannya para peserta tender akan mengajukan berbagai solusi alternatif dengan berbagai pertimbangan fungsional ataupun biaya pengembangan.
- Sesudah pemenang ditetapkan, maka Kontraktor ini harus memberikan definisi rinci dari apa yang akan dikembangkan untuk Clientnya.
  - Definisi ini menjadi ‘**kontrak**’ yang menjadi dasar untuk melakukan validasi pekerjaan yang akan dilakukan kontraktor



# SISTEM

- Sistem

- Kombinasi dari elemen-elemen yang terorganisasi untuk mencapai suatu tujuan yang sudah ditetapkan
  - Hardware
  - Software (program komputer, aplikasi komputer)
  - Data
  - Manusia
  - Proses
  - Prosedur (SOP, UU, dll)
  - Fasilitas/Material terkait

# *Computer-Based System Elements*

- Software
- Hardware
- People
- Data
- Documentation
- Procedures



KNOWLEDGE & SOFTWARE ENGINEERING

# *Rekayasa Sistem*

- Systems engineering is an interdisciplinary field of engineering and engineering management that focuses on how to design and manage complex systems over their life cycles
- Rekayasa Sistem adalah rekayasa dan manajemen rekayasa yang melibatkan bidang antar disiplin yang berbeda yang fokus pada cara merancang dan mengatur aktivitas pengembangan sistem yang kompleks

# ***Kebutuhan Pengguna (user requirements)***

- Kebutuhan yang keluar dari Pengguna ataupun stakeholder terkait
- Bahasa yang digunakan adalah bahasa sehari-hari (natural)
  - Pernyataannya dalam bahasa ‘manusia’, kadang bisa disertai dengan gambar.
  - Pernyataan ini bisa berisi harapan apa yang dapat diberikan oleh sistem, juga kadang batasan-batasan operasional
- Kebutuhan ini akan dituliskan untuk diyakinkan kembali ke pengguna (atau pelanggan).

# *Kebutuhan Sistem*

- Menggunakan bahasa teknis
- Berisi pernyataan dalam dokumen yang terstruktur yang berisi deskripsi rinci dari **fungsi**, **layanan** dan batasan **operasional** dari suatu sistem
- Mendefinisikan apa yang harus diimplementasikan, hingga kemungkinan menjadi bagian dari kontrak antara 'client' dan kontraktor.
- Dokumentasi Kebutuhan Sistem



KNOWLEDGE & SOFTWARE ENGINEERING

# *Contoh: Sistem Perpustakaan*

18



# ***Kebutuhan pada Sistem Perpustakaan***

- Kebutuhan Pengguna (User Requirements)
  - Pimpinan perpustakaan ingin ada sistem komputer yang memungkinkan buku dapat dipinjam dan dikembalikan oleh peminjam. Sistem ini juga dapat memberikan laporan bulanan.
- Kebutuhan Organisasi/bisnis perpustakaan
  - Manajemen perpustakaan membutuhkan laporan akhir bulan setiap tanggal akhir bulan.
    - Laporan berisi data jumlah buku yang dipinjam, dan jumlah peminjam akan dibuat perhitungan akhirnya
    - Laporan akan berisi data buku yang dipinjam sesuai dengan topiknya, jumlah peminjam berdasarkan kategori peminjam (mahasiswa, dosen atau masyarakat umum).
  - Laporan harus dicetak setelah jam 17:00 di hari kerja terakhir akhir bulan.
    - Hanya bagian administrasi umum yang boleh mencetak laporan

# *Elemen pada Sistem Perpustakaan*

[https://www.lib.itb.ac.id/sites/default/files/Materi%20Orientasi%20Perpustakaan%202016\\_0.pdf](https://www.lib.itb.ac.id/sites/default/files/Materi%20Orientasi%20Perpustakaan%202016_0.pdf)

## Aturan-aturan Organisasi/ Bisnis

<b>Lokasi</b>	<b>Masa Perkuliahuan</b>	<b>Libur Semester</b>
Lantai 1 s/d 4	Senin s.d Kamis: 08.00-21.00	Senin s.d. Kamis : 08.00-16.30
	Jumat: 08.00-11.00   13.00-21.00	Jumat 08.00-11.00   13.00-16.30
	Sabtu 08.00-13.30	

- Apa saja larangannya?
- Bagaimana aturan peminjamannya?
- Fasilitas apa saja yang diberikan?
- Sanksi keterlambatan?



### • Bagaimana data buku disimpan?

#### Jenis Koleksi Cetak

Mingguan/ Koleksi Kerja		Koleksi Umum	
Koleksi TPB		Koleksi World Bank	
Rujukan/Referensi		Koleksi Goethe Institute	
Rujukan Bibliografi		Koleksi Doddy A. Tisna Amidjaja	
Koleksi American Corner		Koleksi Khusus :	
Koleksi Sampoerna Corner		<ul style="list-style-type: none"> <li>Buku Langka</li> <li>Buku Indonesia</li> <li>Dokumentasi ITB</li> <li>Laporan Penelitian ITB</li> <li>Tesis &amp; Disertasi ITB</li> <li>Majalah Seni Rupa</li> </ul>	
Koleksi Indonesia Nation Building Corner		Koleksi Majalah/Jurnal	

Jenis Koleksi	Lama Pinjam	Kuota Pinjam
• Textbook / Koleksi Mingguan (Lantai 2, 3 dan 4)	• 2 Minggu	• 8 Buku
• Koleksi Pengembangan ITB III (Lantai 1)	• 2 Minggu	• 2 Buku
• Koleksi Kerjasama AC, INBC, SC, GI (Lantai 1)	• 1 Minggu	• 2 Buku
• Koleksi TPB	• 1 Minggu	• 4 Buku
• Koleksi Umum	• 2 Minggu	• 4 Buku

Total Pinjaman @ mahasiswa = 10 buku

Engineering

- Tata tertib?

	MENGELUARKAN SUARA KERAS YANG MENGGANGGU ORANG LAIN
	MEMINDAHKAN BAHAN PUSTAKA KE RAK LAIN
	MENGEMBALIKAN KEMBALI BUKU KEPADA RAKNYA
	MENCORET, MEROBEK ATAU MELAKUKAN HAL YANG MERUSAK SUATU BAHAN PUSTAKA
	MEROKOK, MAKAN DAN MINUM
	MEMBUANG SAMPAH SEMBARANG

- Sanksi keterlambatan?

<input type="checkbox"/> Buku terlambat dikembalikan	Rp 1.000, /buku/hari
<input type="checkbox"/> Buku hilang, sobek atau kotor	Diganti dgn judul buku yg sama atau dengan buku subjek yg sama senilai <b>harga buku saat lapor hilang</b>

- Bagaimana system keamanan?



Sistem Keamanan di Perpustakaan

Gate yang aktif ketika buku langsung di bawa keluar

- Kegiatannya?



# Hardware Requirements

## DataManager™ Optical Mark Recognition (OMR) requirements

The table below lists the minimum system requirements for OMR.

Additional minimum system recommendations:

- Processor: 2 GHz or faster
- Memory: 2 GB RAM or more
- Hard drive space: 40 GB hard drive with 20 GB available space
- CD-ROM drive (for installation of ScanTools software)
- USB 2.0 (for OMR processing)
- JavaScript™ must be enabled
- 1024 x 768 or higher screen resolution
- High speed Internet connection
- Barcode reader attachment (*required for OpScan scanners only*)

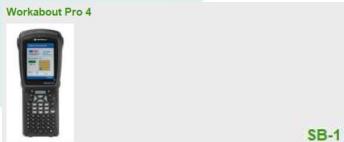
## Software Requirements

Operating System	Web Browser	Software
Windows® 7, 64 or 32 bit		
Windows® Vista, Business or higher or 32 bit SP1 or higher	Microsoft® Internet Explorer 8 or higher	<ul style="list-style-type: none"> <li>• ScanTools 8.0 or higher</li> <li>• Adobe Reader® 9.4.0 or higher (for viewing the scanner user's guide)</li> </ul>
Windows® XP Professional SP3 or higher		



KNOWLEDGE & SOFTWARE ENGINEERING

# Mungkin ada beberapa pilihan HW?



Workabout Pro 4

Sometimes applications require a mobile computer adapted to meet specific requirements. With its full range the Workabout Pro brings a flexibility to Sage® applications that make it the ideal work horse.

The Workabout Pro is the mobile computer that can evolve to meet your ever-changing business needs. Modularity lets you buy the features you need now, practically any feature you can imagine, right in you Start with Wi-Fi and add WWAN. Swap scan engine new types of bar codes. Add support for voice pickin and just about any type of RFID tag. Many ready-to-are available to meet the highly specialized needs o vertical markets. And since the Workabout brand ha businesses for over 20 years, you can choose the w Workabout Pro 4 with confidence.

Key features include:

- Impressive modularity for an extraordinary life cycle TCO
- The flexibility to choose the right model for every job
- Backwards compatibility with accessories
- A high-resolution color display for all applications



SB-1 Smart Badge

Sometimes applications require a mobile computer adapted to meet specific requirements. With its full range the Workabout Pro brings a flexibility to Sage® applications that make it the ideal work horse.

The Workabout Pro is the mobile computer that can evolve to meet your ever-changing business needs. Modularity lets you buy the features you need now, practically any feature you can imagine, right in you Start with Wi-Fi and add WWAN. Swap scan engine new types of bar codes. Add support for voice pickin and just about any type of RFID tag. Many ready-to-are available to meet the highly specialized needs o vertical markets. And since the Workabout brand ha businesses for over 20 years, you can choose the w Workabout Pro 4 with confidence.

Key features include:

- Impressive modularity for an extraordinary life cycle TCO
- The flexibility to choose the right model for every job
- Backwards compatibility with accessories
- A high-resolution color display for all applications



Barcode Scanners



LI 4278 Scanner

Barcode scanning has been a part of our history. Ensuring this key element of any barcode based scanner, exactly meets the requirements of the industry has been a key to our success.

The LI4278 takes 1D bar code scanning to the next level, allowing workers to scan faster and farther as they can capture virtually any 1D bar code. Built for all day and everyday use, the LI4278 offers cordless freedom with Bluetooth compatibility. It also offers better encryption for improved security and performance. The LI4278 is backward compatible with the cradle which works with LS4278 and DS6878. Superior battery life provides the largest number of scans per battery charge. Intensive applications. You can use it in environments and it can survive a 6-foot drop.

- Captures virtually all 1D bar codes from mobile phone displays
- Superior motion and angular tolerance
- Built-in rechargeable battery
- Backward compatible



Barcode Printers



PD 41 A member of the smartest bar-code printer family on the market, the Intermec PD41 is flexible and programmable, enabling customers to optimize their printing operations, streamline deployment, and achieve quick return on investment. Built to meet the needs of mission-critical applications, the rugged PD41 delivers advanced, secure connectivity and the latest network protocols, ensuring peace-of-mind today and a reliable, scalable solution for the long haul. The versatile PD41 Commercial Printer is part of Intermec's complete line of smart, strong and secure industrial printers. Strong metal construction for demanding, medium-duty applications. Smart Printing capabilities support stand-alone printer applications, eliminating PC expense and complexity. All-in-one, user-selectable printer languages Fingerprint/Direct Protocol (DP), IPL, ZSim, DSim and in every printer Secure wireless connectivity: CCX and WiFi certified with WPA2. Interchangeable print head provides choice of 203 or 300 dpi print resolution.



Alien ALR-9680 4 port RFID reader in RFID applications, the ability to accurately and repeatedly read tags is key to a successful system design. Sage Data uses the Alien reader as the basis for its main fixed reader installations. The Alien ALR-9680 is a commercial-grade UHF RFID reader that provides enterprise grade 4-port flexibility and Alien's industry leading ease-of-use and reader "intelligence". Feature-rich Alien Reader Protocol 4 monostatic reader ports POE eliminates cost of AC power drop EPC Gen 2 dense reader interoperable slim form-factor for installation in height-restricted places. Manageable and upgradeable.

IF2250 Requirement Engineering



# **Pertanyaan Untuk Pengembangan Sistem Informasi Perpustakaan**

Elemen apa yang ingin dikembangkan sebagai bagian dari sistem informasi perpustakaan?

- Aturan apa yang terkait (yang sudah ada ataupun yang ingin ada)?
- Data apa yang harus disimpan?
- Laporan apa saja yang diperlukan?
- Layanan apa saja yang ingin diberikan?
- Siapa saja yang terlibat dan apa tanggung jawabnya?
- Aturan pemerintah atau UU apa yang mungkin terkait atau perlu diikuti?
- dll

# *Some rules of thumb*

- Tidak semua elemen mungkin harus di-'komputer'-kan
  - Jumlah elemen tergantung pada permintaan pengguna (Kontrak)
  - Besar kecilnya elemen yang akan dikembangkan perlu memperhatikan
    - Waktu pengembangan
    - Biaya Pengembangan
  - Mengikuti lingkup masalahnya - Scoping
- Berbagai elemen tadi mungkin masih tidak lengkap, tidak jelas, ambigu, saling konflik, dll
- Penjelasan User (pengguna) cenderung sifatnya umum
  - Pengembangan/system analis bertanggung jawab menterjemahkan keinginan tadi menjadi bahasa yang bisa dimengerti oleh software designer/programmer

# *Karakteristik Pengguna*



KNOWLEDGE & SOFTWARE ENGINEERING



# *Berbagai Karakteristik Pengguna*

27



KNOWLEDGE & SOFTWARE ENGINEERING

# *Berbagai Karakteristik Pengguna (user)*

- Pengguna yang memiliki suatu hak akses tertentu
  - Tamu, pengguna anggota organisasi, petugas pelayanan masyarakat, administrator jaringan, administrator sistem
- Pengguna yang hanya mengerti satu bagian operasional, sehingga kadang tidak mengerti secara lebih umum tentang sistem yang akan dikembangkan
  - Contoh: petugas loket bagian peminjaman buku, hanya mengerti bagian peminjaman buku saja, tetapi tidak tahu bahwa ada masalah pembuatan laporan pada software yang akan dibuat
- Pengguna yang hanya mengerti satu sistem
  - Hanya mengerti windows, hanya tahu menggunakan MS Word, hanya tahu 'facebook', hanya tahu menggunakan aplikasi statistik
- Pengguna yang terbiasa berkomunikasi dengan bahasa ibunya
- Pengguna yang secara langsung atau tidak langsung berhubungan dengan pelanggan
  - Contoh: pengguna yang berperan sebagai manajer akan berbeda perlakunya dengan pengguna yang berperan sebagai penghubung dengan pelanggan
- Pengguna yang sibuk atau yang tidak sibuk (sulit ditemui)
- Pengguna juga mungkin memiliki karakter yang berbeda-beda
  - Pengguna yang sulit atau lambat dalam mengungkapkan ide,
  - Pengguna yang 'sok tahu'
  - Pengguna yang suka emosional
  - Dan lain-lain

Sistem analis harus 'sabar', mencoba mengerti, tidak mudah emosional, dll, agar tujuan pekerjaannya dapat dilaksanakan, yaitu **mendapatkan 'kebutuhan pengguna'**



## **Dengan Karakteristik tersebut, akibatnya pernyataan kebutuhan\* menjadi:**

- Terlalu umum, tidak rinci, tidak sederhana, terlalu ‘mengawang-awang’ atau bertele-tele
- Terlalu rinci, sehingga tidak menggambarkan sistem secara keseluruhan
- Terjadi konflik antara bagian yang berbeda dalam satu sistem
  - Perbedaan aturan, perbedaan tanggungjawab
- Tidak konsisten
  - *Hari ini bilang X, minggu depannya ingin Y*
- Mungkin kebutuhannya tidak realistik
  - Diluar anggaran, akan terlalu lama waktu pengembangannya

\* Pernyataan Kebutuhan = Requirements statement

# *Three Types of User by Job category\**

- **OPERATIONAL USER**

- Usually has a local view
- Carries out the function of the system
- Has a physical view of the system

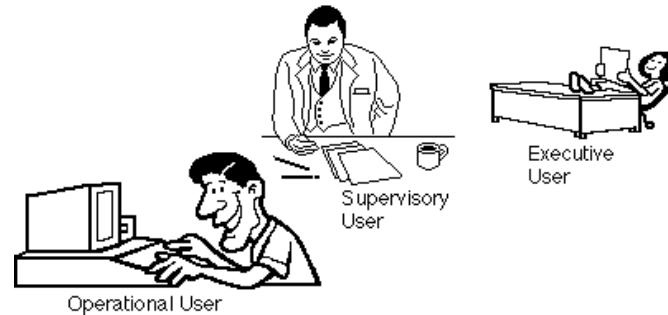
- **SUPERVISORY USER**

- May or may not have local view
- Generally familiar with operation
- Driven by budget considerations
- Often acts as a middleman between users and higher levels of management

- **EXECUTIVE USER**

- Has a global view
- Provides initiative for the project
- No direct operating experience
- Has strategic concerns

\*) Yourdon: "Just Enough Structured Analysis"

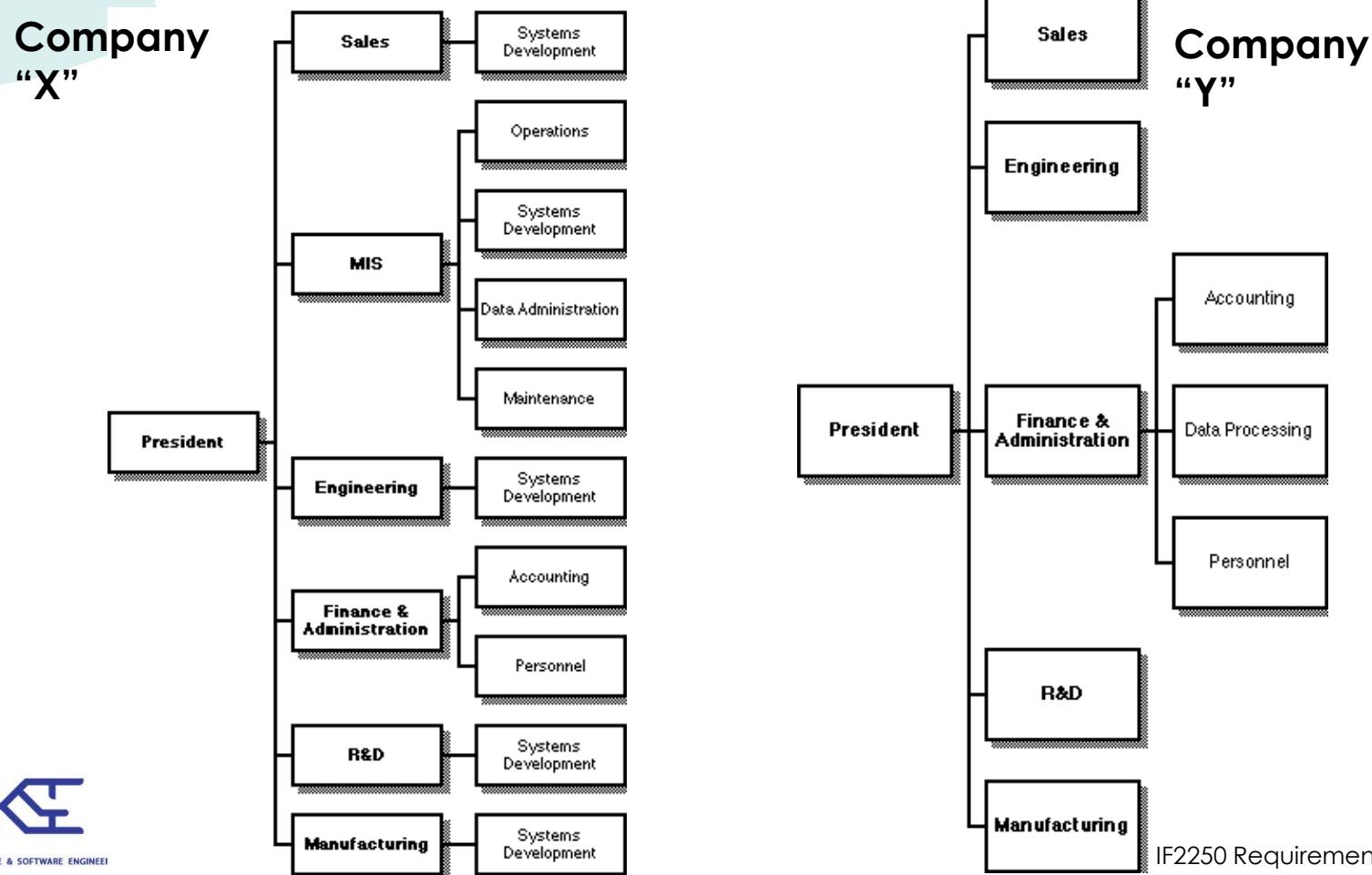


IF2250 Requirement Engineering



# *Users by Level of Experience*

31



KNOWLEDGE & SOFTWARE ENGINEER

IF2250 Requirement Engineering

# Kebutuhan Pengguna (User Requirements)

- Kebutuhan pengguna bisa dibagi menjadi dua sifat:
  - Kebutuhan bersifat fungsional (Functional Requirements)
  - Kebutuhan bersifat non-fungsional (Non Functional Requirements)
- Pengguna sering mencampur-adukkan kebutuhannya
  - Mungkin konflik, tidak konsisten, ambigu, dll
- Sistem analis wajib:
  - Menuliskan kembali suatu pernyataan kebutuhannya, dengan bahasa yang harus dapat dimengerti oleh pengguna
  - kadang-kadang tidak mengerti bahasa teknis
- Pernyataan kebutuhan pengguna ini dapat menggunakan bahasa 'sehari-hari' (Bahasa alami/'natural language')
  - dapat didukung dengan diagram atau tabel yang dapat dimengerti oleh semua calon pengguna.

# *Permasalahan Dengan Bahasa Alami*

- Kadang tidak jelas
  - Kurang presisi, tetapi kalau terlalu presisipun mungkin menyebabkan dokumen sulit dibaca
- Ketidakjelasan kebutuhan
  - Kebutuhan fungsional dan non-fungsional cenderung tercampur
- Kebutuhan yang berlebihan
  - Beberapa pernyataan kebutuhan mungkin diungkapkan bersamaan.

# Dari Cerita ke Coding



## Cerita si A

Dedi Wahyudi Riset

1. Bakalapak akan melakukan untuk rilis dataset yang sanitized. Pak ini juga bisa menjadi proyek niat bersenang-senang. Riset dengan data yang lebih mesesan bisa diakui oleh infrastruktur Bakalapak, tapi tetep untuk akses tidak akan memberikan akses ke lar. Jika membutuhkan data yang lebih detail, bisa diajak dan berkoordinasi bersama tim di Bakalapak untuk mengolah datanya.  
 2. Untuk HAKI bisa diskusikan per proyeknya Pak.  
 3. Mengenai pertengangan dan perih pengklomoran Pak saat belum dipublicasikan atau dikonsensualin dengan approval dari Bakalapak dan ITS, definitly adalah Pak! Besaranya antara Bakalapak dan ITS.



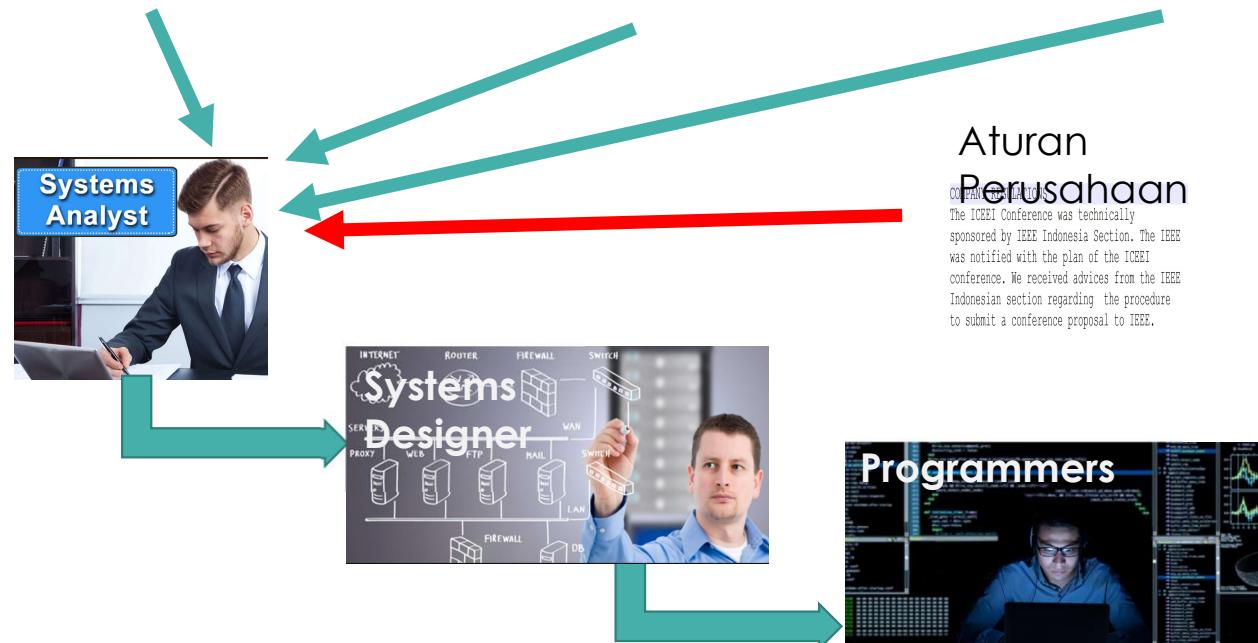
## Cerita si B

1. Seperti ini masih sesuai struktur yang sudah sempat saya share Pak, jadi yang akan pulang dengan struktur di sisi aplikasi Admin dan Pengguna datar Marawise, dan Tim Bakalapak sebagai support metode pengelolaan data. Mungkin akan ada satu dua hari Tim Bakalapak kudu di Sesi utama merencanakan jadwal, sebelumnya bisa dilakukan di remote.  
 2. Mengenai rekomendasi setuju Pak. Aku tetep pinggiran dalam PCG adalah proses pengguna dan pertumbuhan. Program plan dan budget dibentukkan per term.  
 3. Mengenai Rekomendasi, dan kunci setuju Pak. Kita mungkin request prosesnya bisa dimulai berbarengan dengan persiapan presentasi Pak dan dari Bakalapak ada yang akhirnya juga untuk perih pengklomoran.

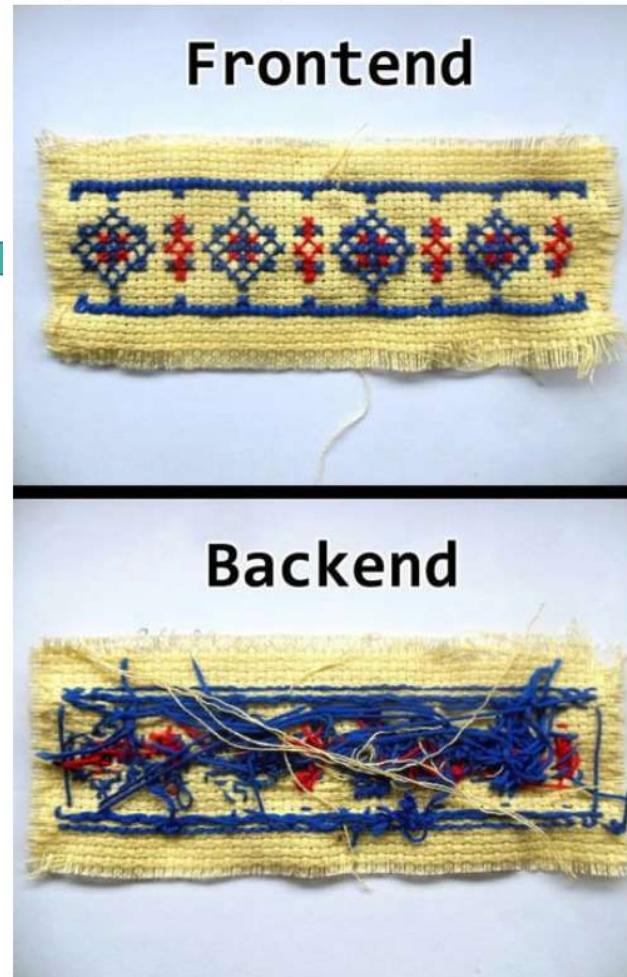


## Cerita si C

The ICEEI Conference was technically sponsored by IEEE Indonesia Section. The IEEE was notified with the plan of the ICEEI conference. We received advices from the IEEE Indonesian section regarding the procedure to submit a conference proposal to IEEE.



For programmers only



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 Requirement Engineering

# *Spesifikasi Kebutuhan PL dibuat berdasarkan berbagai Kebutuhan Sistem*



# *Sifat Kebutuhan*

- Kebutuhan Fungsional
- Kebutuhan Non Fungsional

## **Catatan:**

- Kebutuhan Fungsional dan Non Fungsional ada pada perangkat lunak dan ada pada suatu system
- Dalam pengembangan perangkat lunak, Spesifikasi Kebutuhan Perangkat Lunak juga harus membedakan kedua sifat kebutuhan ini.

# ***Kebutuhan Fungsional dan Non-Fungsional (NF)***

- Kebutuhan Fungsional
  - Pernyataan kebutuhan tentang layanan yang harus diberikan oleh sistem
  - Bagaimana sistem harus memberikan respon terhadap suatu masukan
  - Kadang-kadang termasuk juga “apa yang tidak perlu dilakukan sistem”
- Kebutuhan Non-Fungsional (NF)
  - Berisi pernyataan tentang batasan (constraint) dari layanan/fungsi yang ditawarkan oleh sistem
    - Contohnya: Timing constraint, Development constraint, Standard constraint

Kedua jenis kebutuhan ini bisa saling melengkapi, atau bahkan kebutuhan non-fungsional kadang menghasilkan kebutuhan baru yang kategorinya fungsional, juga sebaliknya

Contoh: kebutuhan NF akan jaminan Security, kemungkinan akan menghasilkan kebutuhan fungsional otentifikasi user



KNOWLEDGE & SOFTWARE ENGINEERING

# *Kebutuhan Fungsional*

- Berisi deskripsi fungsionalitas atau layanan sistem
- Bergantung pada tipe PL, calon pengguna dan juga tipe sistem yang akan digunakan
- Kebutuhan Fungsional Pengguna dapat berupa pernyataan umum tentang apa yang akan dilakukan sistem
- Kebutuhan Fungsional Sistem perlu menjelaskan layanan sistem secara rinci.

## *Contoh Kebutuhan Pengguna*

- Sistem Perangkat lunak untuk Perpustakaan
  - “Pengguna harus dapat mencari buku”
  - “Sistem dapat memberikan tampilan data buku untuk anggota perpustakaan”
  - “Sistem harus dapat memberikan laporan bulanan”
  - “Setiap pemesanan buku akan diberikan nomor pemesanan yang unik”
  - “Tidak boleh ada anggota yang bisa meminjam lebih dari 5 buku dan masa peminjaman tidak lebih dari 7 hari”

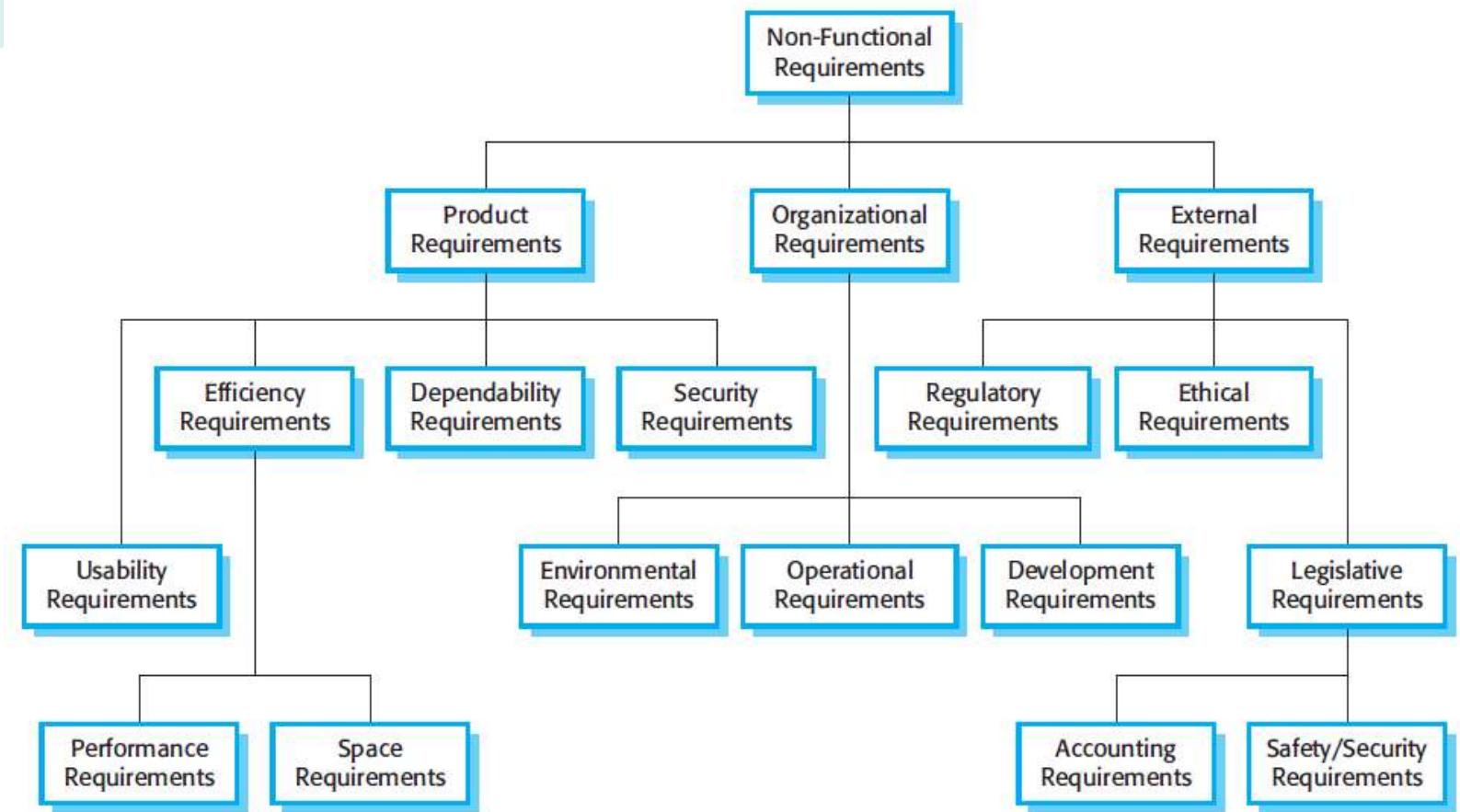
**Cari yang fungsional dan yang non-fungsional!**

# *Kebutuhan Non Fungsional*

- Kebutuhan NF kadang-kadang bisa lebih critical daripada kebutuhan fungsional
  - Contoh: Pada sistem auto-cruise mobil: saat rem di pijak, maka sistem auto-cruise harus stop
- Kebutuhan NF kadang bisa berakibat pada sistem secara keseluruhan (tidak hanya satu komponen tunggal)
  - Contoh: kebutuhan performansi suatu sistem, akan melibatkan berbagai komponen
- Satu kebutuhan NF mungkin bisa menghasilkan beberapa kebutuhan fungsional.

# Jenis-jenis Kebutuhan NF

42



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 Requirement Engineering

# *Kategori kebutuhan NF*

- **Kebutuhan Produk**

- Kebutuhan ini menjelaskan tentang perilaku dari software atau pembatasan perilaku
- Contoh: Besar memori yang dibutuhkan, kebutuhan security, kebutuhan usability

- **Kebutuhan Organisasional**

- Diturunkan dari kebijaksanaan atau aturan dalam lingkungan pengguna dan pengembang
- Contoh: Keperluan penggunaan bahasa pemrograman tertentu, kebutuhan untuk mengikuti aturan organisasi (misalnya pada suatu sistem perpustakaan ada aturan peminjaman buku maksimal 5 buku /peminjam) ataupun kebutuhan akan sistem operasi yang akan digunakan.

- **Kebutuhan Eksternal**

- Kebutuhan ini berasal dari luar sistem dan juga di luar dari proses pengembangan.
- Contoh: Pengembangan software untuk suatu Bank, harus memperhatikan juga aturan Bank Indonesia, pembangunan yang harus mengikuti peraturan UU yang ada agar software tidak beroperasi yang melanggar hukum.

# *Contoh Kebutuhan Non-Fungsional*

- Sistem perpustakaan
  - Tampilan browser diimplementasikan dengan HTML sederhana, tidak perlu ada frame atau Java applets
  - Sistem harus berjalan 24 jam, jika terjadi down, maka tidak melebihi 15 menit di jam kerja atau 3 jam di luar jam kerja. **(PRODUCT REQUIREMENTS)**
  - Proses pengembangan dan dokumen yang diserahkan harus sesuai dengan aturan di SNI-PL-1005
  - Semua pengguna harus terdaftar oleh sistem  
**(ORGANISATIONAL REQUIREMENTS)**
  - Sistem hanya akan menampilkan nama anggota perpustakaan, dan nomor referensinya. Sistem tidak akan menampilkan daftar informasi personal dari anggota.  
**(EXTERNAL REQUIREMENTS)**

# *Masalah pada kebutuhan NF*

- Kebutuhan NF harus semaksimal mungkin terkuantifikasi (agar dapat diuji (testable))
  - Pernyataan asli dari pengguna:
    - “Software harus mudah digunakan dan kesalahan pada saat penggunaan semaksimal mungkin di kurangi”
  - Pengembang perlu mengkuantifikasi sbb:
    - “Software harus dapat digunakan oleh staf setelah 8 jam training, dan sesudahnya bagi pengguna yang sudah berpengalaman jumlah rata-rata error yang muncul tidak melebihi 2 kali untuk setiap jam.
- Jika mungkin kebutuhan NF harus ditulis secara kuantitas agar mudah diuji secara objektif.

# **Bagaimana cara mengukur Kebutuhan NF?**

Metriks	Cara Mengukur
Kecepatan	Jumlah Transaksi/detik Waktu Respon Pengguna/Event Waktu Screen Refresh
Ukuran (size)	Mbytes, Gbytes
Kemudahan Pakai	Training Time Jumlah layar bantu (help)
Keandalan (reliability)	Mean Time to Failure Unavailability probability Kecepatan kemunculan kegagalan Availability
Robustness	Waktu restart sesudah failure Prosentase kegagalan even Probabilitas terjadinya data corrupt kalau ada failure
Portability	Jumlah target sistem



KNOWLEDGE &amp; SOFTWARE ENGINEERING

# Keterkaitan antar Kebutuhan

- Konflik sering terjadi pada kebutuhan NF
  - Terutama pada sistem yang kompleks
- Contoh Spacecraft system
  - Untuk mengurangi beban, jumlah chips harus dikurangi
  - Untuk mengurangi konsumsi daya, gunakan chips dengan daya rendah
  - Tetapi dengan chips daya rendah, mungkin berarti makin banyak chip yang harus digunakan
    - Mana requirements yang paling kritis untuk dipenuhi?

\* *Software Engineering 7<sup>th</sup> ed, Ian Sommerville*

# ***Kebutuhan Berdasarkan Domain (Domain Requirements)***

- Setiap domain memiliki kebutuhan yang berbeda-beda
  - Kebutuhan domain berisi penjelasan karakteristik dan fitur sistem yang memberikan gambaran tentang domain itu
- Kebutuhan berdasarkan domain ini bisa menjadi
  - kebutuhan fungsional baru;
  - batasan terhadap kebutuhan yang sudah ada; atau
  - Mendefinisikan komputasi yang spesifik
- Jika kebutuhan akan domain ini tidak terpenuhi, maka sistem mungkin tidak akan berjalan seperti yang diharapkan.

# Contoh Domain

- Domain E-Commerce
- Domain Logistik
- Domain Akademik
- Domain Perbankan

Setiap domain memiliki fokus kualitas solusi yang berbeda-beda.

- Contohnya fokus kualitas solusi domain perbankan, adalah untuk mencatat perhitungan keuangan dengan nilai angka yang besar dengan akurat (banyaknya jumlah digit);
- tetapi domain akademik berfokus pada pencatatan nilai-nilai akademik mahasiswa dengan benar, tingkat akurasi bilangan tidak terlalu menjadi perhatian

Setiap domain memiliki kata kunci (keywords) yang berbeda-beda

- Kata kunci ini bukan hanya kata, tapi mungkin “istilah”, hingga bisa berupa satu kata atau kata bentukan
- Kata kunci ini umumnya berbentuk kata benda ('noun')

# ***Contoh Kata Kunci***

- Perpustakaan
  - Peminjam, peminjaman buku, pengembalian buku, jenis buku, nomor catalog buku, sangsi, dll
- Perbankan
  - Saldo, ATM, Billing, Kliring, Transfer, Bunga, dll

Silahkan cari:

- Kata kunci untuk sistem finance
- Kata kunci untuk sistem healthcare
- Kata kunci untuk sistem insurance
- Kata kunci untuk sistem transportation
- Kata kunci untuk sistem academic
- Kata kunci untuk sistem pendidikan music

# *Masalah Kebutuhan berdasarkan Domain*

- Kemudahan dimengerti (Understandability)
  - Kebutuhan diungkapkan dalam bahasa dari domain aplikasi
  - Hal ini sering tidak dimengerti oleh pengembang perangkat lunak.
- Implicitness
  - Yang mengerti domain masalah sangat mengerti permasalahannya, sehingga kadang-kadang mereka tidak menyadari bahwa suatu masalah pada domain tersebut harus dinyatakan secara eksplisit.

\* *Software Engineering 7<sup>th</sup> ed, Ian Sommerville*

# *Kebutuhan Yang Tidak Presisi*

Pernyataan kebutuhan yang tidak presisi menyebabkan interpretasi yang berbeda antara pengembang dan pengguna

- Contoh kebutuhan yang tidak presisi:
  - “**Pengguna harus dapat mencari buku**”
- Interpretasi Pengguna:
  - Hanya mencari buku berdasarkan judul buku yang tidak sedang dipinjam
- Interpretasi Pengembang
  - Mencari suatu buku tanpa mempedulikan apakah buku sedang dipinjam atau tidak

# *Kelengkapan Kebutuhan dan Konsistensi*

- Suatu kebutuhan harus lengkap dan konsisten
  - Lengkap: semua deskripsi yang diperlukan sudah terungkapkan
  - Konsisten: tidak ada konflik atau kontradiksi pada deskripsi
- Dalam prakteknya, tidak mungkin membuat kebutuhan yang lengkap dan konsisten.
  - Tapi kita harus berusaha membuatnya



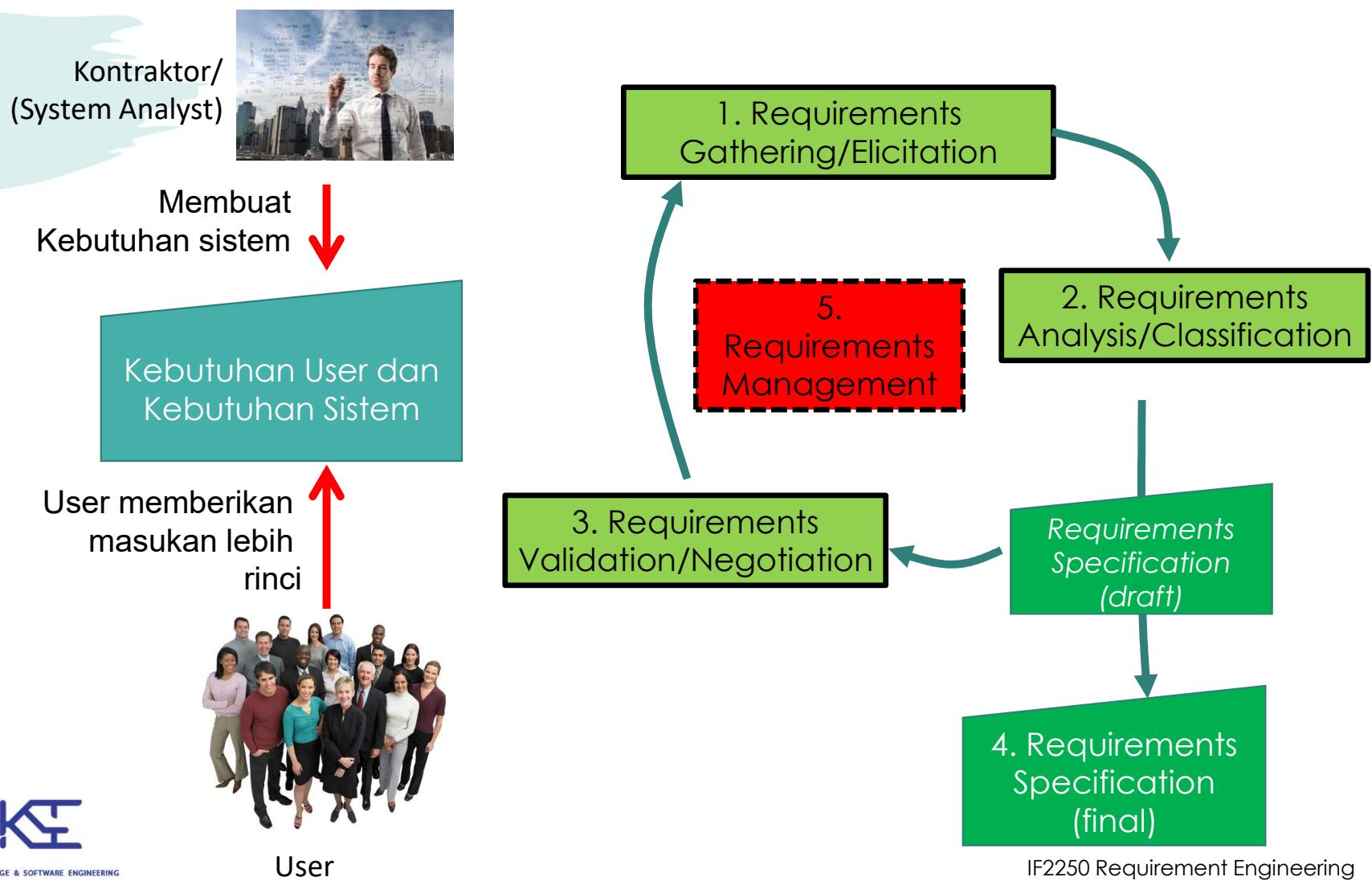
KNOWLEDGE & SOFTWARE ENGINEERING

# *Proses Rekayasa Kebutuhan Perangkat Lunak/Sistem*



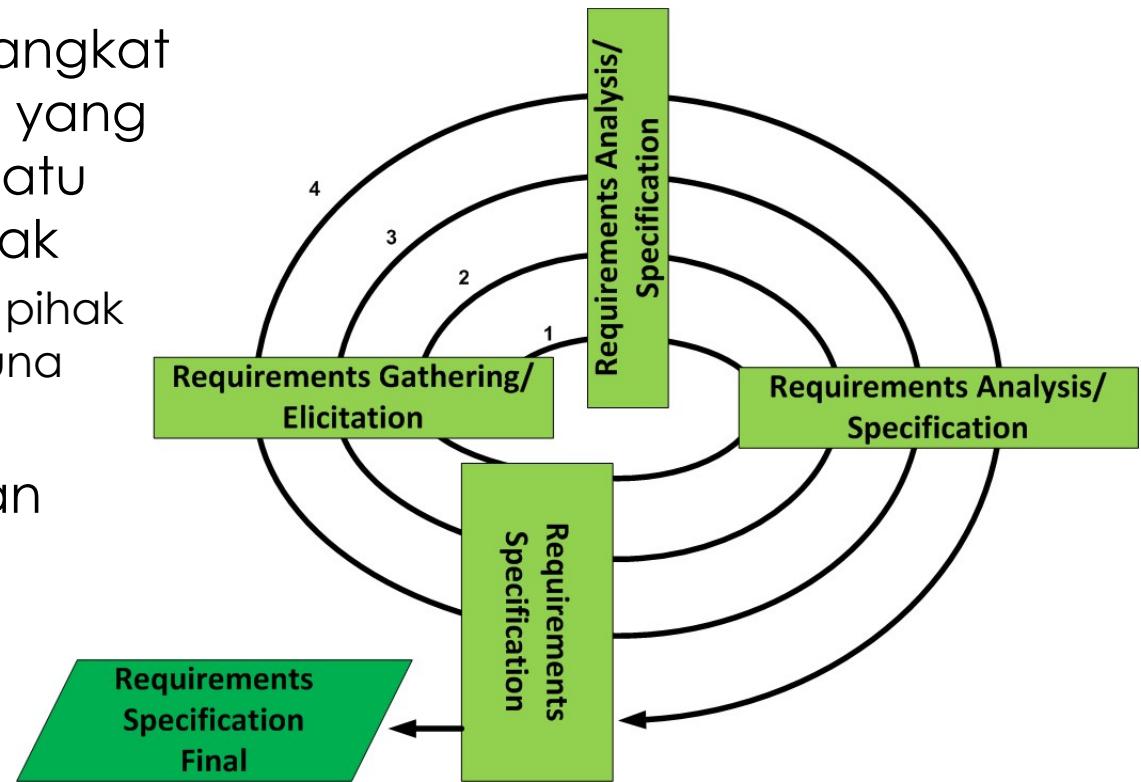
# *Proses Rekayasa Kebutuhan (RK)*

- Untuk setiap tahapan RK (atau RE – Requirements Engineering) perlu mengenali:
  - Domain aplikasi
  - Orang yang terlibat (stakeholders)
  - Organisasi yang mengembangkan kebutuhan
- Secara umum prosesnya:
  - **Requirements Elicitation**
  - **Requirements Analysis**
  - **Requirements Validation**
  - **Requirements Management**
- Hasil dari proses ini adalah **Requirements Specifications**
- Aktivitas RK dilakukan secara iteratif



# *Analisa kebutuhan PL adalah Proses Iteratif*

- Dokumen kebutuhan perangkat lunak adalah pernyataan yang sudah disepakati untuk suatu kebutuhan perangkat lunak
  - Harus diorganisasikan agar pihak pengembang dan pengguna dapat menggunakannya
- Proses rekayasa kebutuhan adalah proses iteratif



# *Requirements Elicitation and Analysis*

- Kadang disebut juga: Requirements Discovery (pencarian kebutuhan)
  - Teknik: **interview** (open interview vs. close interview)
  - **Skenario** sering dibutuhkan untuk mendapatkan situasi yang terjadi di dunia nyata
    - Skenario ini bisa berupa
      - rincian awal suatu situasi,
      - urutan normal suatu kejadian, ataupun
      - kemungkinan urutan yang tidak biasa,
      - aktivitas lain yang terjadi bersamaan, ataupun
      - berupa deskripsi apa yang terjadi kalau skenario selesai.
- Pengembang bekerja dengan pengguna untuk mengerti 'domain' aplikasi, layanan yang diperlukan dan juga batasan-batasannya.
- Melibatkan end-user, manajer, insinyur yang terlibat pada perawatan (maintenance), ahli domain masalah, dan lain-lain
  - Sering disebut sebagai **Stakeholders**

# *Teknik Pengumpulan Kebutuhan*

- Wawancara
- Brainstorming
- Focus Group Discussion
- Observasi
- Prototyping
- Workshop
- Reverse Engineering
- Survey

**PR (tidak dikumpul):**  
**Baca ini**

[http://www.brighthubpm.com/project-planning/  
60264-techniques-used-in-business-requirements-gathering/](http://www.brighthubpm.com/project-planning/60264-techniques-used-in-business-requirements-gathering/)



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 Requirement Engineering

# Proses Requirements Elicitation and Analysis



## ***Masalah dalam Analisis Kebutuhan (Requirements Analysis)***

- Stakeholder tidak tahu apa yang mereka inginkan sebenarnya
  - Tidak tahu cara menyatakan kebutuhannya
- Stakeholder menyatakan keinginannya dengan istilah mereka sendiri
- Stakeholder yang berbeda mungkin memiliki kebutuhan yang berbeda
- Faktor jenis/tipe organisasi dan juga politis dapat mempengaruhi kebutuhan sistem
- Kebutuhan mungkin berubah selama proses analisis
  - Stakeholder yang baru mungkin muncul dan lingkungan bisnis mungkin juga berubah

# Rangkuman

- Pengembang harus mengumpulkan berbagai kebutuhan (atau persyaratan) yang diperlukan dalam pengembangan system
  - User requirements, system requirements (organizations/business requirements, hardware requirements, software requirement, dll)
- Pengembang akan melakukan analisis terhadap semua kebutuhan tadi dan menetapkan **SPESIFIKASI KEBUTUHAN PERANGKAT LUNAK (SKPL)** yang akan dikembangkan
  - Pengembang harus memperjelas kebutuhan yang konflik, yang tidak lengkap, yang ambigu, yang tidak konsisten, yang tidak mungkin dikembangkan, dll
- Proses rekayasa kebutuhan adalah proses iteratif

# *Penyiapan Kelompok*

- Buatlah kelompok dengan anggota 5 orang. Ketua kelas bertanggung jawab memberikan nomor kelompok.
  - Ketua kelas akan meng-email nomor kelompok dan namanya, paling lambat hari Minggu 29 Januari 2023, jika ada kesulitan hubungi Dosen Pengajar



KNOWLEDGE & SOFTWARE ENGINEERING

Tim Pengajar IF2250

# IF2250 – Rekayasa Perangkat Lunak Analisa Kebutuhan P/L Pendekatan Terstruktur

SEMESTER II TAHUN AJARAN 2022/2023



KNOWLEDGE & SOFTWARE ENGINEERING



# *Pengembangan Spesifikasi Kebutuhan Perangkat Lunak (SKPL)*

- **Spesifikasi** adalah bentuk komitmen antara customer dengan pengembang (developer)
- Spesifikasi ditulis dalam dokumen SKPL ([Spesifikasi Kebutuhan Perangkat Lunak](#)) atau SRS ([Software Requirement Specification](#))
- Dokumen ini akan dibawa ke pertemuan untuk mereview perangkat lunak (SSR – [Software Specification Review](#))

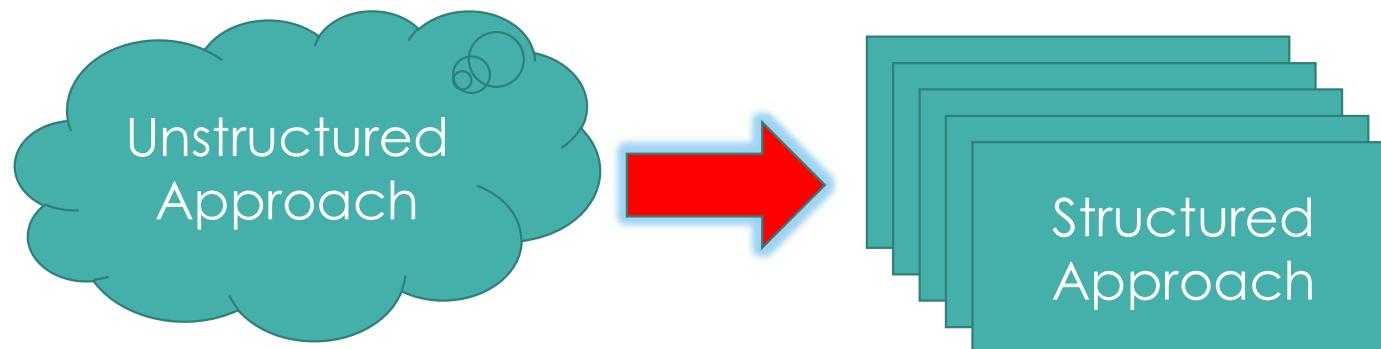
Hasil dari rapat ini adalah

- customer menyetujui penuh atau
- menyetujui dengan perbaikan atau
- menolak

kebutuhan yang akan ada di perangkat lunak.

Dokumen SKPL berisi  
“APA” yang akan dilakukan  
dan bukan “BAGAIMANA”  
cara mengimplementasikan.

# Pendekatan Terstruktur



- Program relatif **skala kecil**
  - Persoalan masih sederhana
  - Jumlah individu terlibat sedikit
  - Kompleksitas persoalan rendah
- Program relatif skala **menengah ke atas**
  - Persoalan melibatkan banyak elemen terkait
  - Jumlah individu makin banyak
  - Kompleksitas persoalan tinggi
  - Butuh waktu yang tepat
  - Butuh usaha besar

# Dekomposisi Fungsional

- **Dekomposisi fungsional** adalah teknik untuk mendekomposisi fungsional dari perangkat lunak yang akan dibangun
- Dekomposisi ini bergantung pada fungsi atau aksi yang akan dilakukan oleh PL.
- Caranya:
  - Tulis dalam satu kalimat apa yang akan dilakukan oleh PL
  - Jika PL melakukan beberapa fungsi, tulis setiap fungsi dalam satu baris
- Contoh:
  - Ingin membuat program robot untuk membuat secangkir kopi
  - Dekomposisi
    - Robot dapat mengambil cangkir
    - Robot dapat menyeduh kopi
    - Robot dapat ‘mengantar’ kopi kepada pemesannya

# *Pemodelan Analisa Kebutuhan dengan Pendekatan Terstruktur*

- Data Flow Diagram proses + aliran data antar proses
- Entity Relationship Diagram
- State Transition Diagram



KNOWLEDGE & SOFTWARE ENGINEERING

Analisis Kebutuhan P/L - Terstruktur

# *Analisis Terstruktur Data Flow Diagram*



KNOWLEDGE & SOFTWARE ENGINEERING



Analisis Kebutuhan P/L - Terstruktur

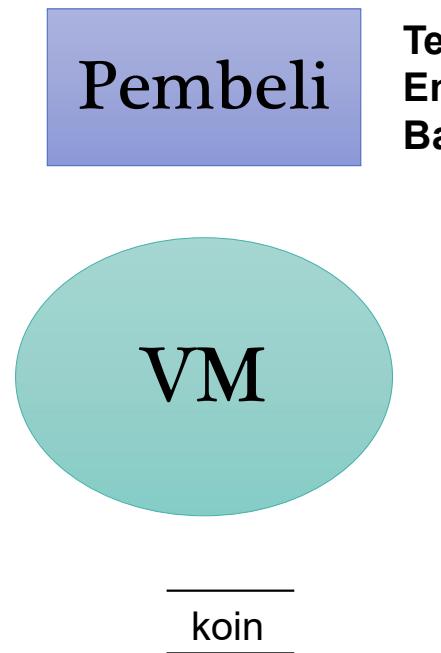
# *Analisis Aliran Data (Dataflow analysis)*

- Metode ini dapat/biasa digunakan sebagai dasar untuk merancang arsitektur P/L skala besar
- Caranya
  - Identifikasi **aliran data** yang akan terjadi pada PL
  - Identifikasi **transformasi** yang terkait dengan aliran data tsb
- Dengan metode ini, maka akan didapat
  - **Proses-proses** dalam PL
  - **Hubungan antar proses**
    - Dilihat dari aliran data dari suatu proses ke proses lain

# *Hubungan aktivitas lain*

- Hasil proses analisis kebutuhan ini akan terkait dengan:
  - Perancangan Sistem Awal (*Preliminary Design*)
  - Perancangan Sistem
  - Manajemen Konfigurasi
  - Kualitas
  - Validasi

# *Notasi Penulisan DFD*



**Terminator/  
Entitas eksternal/  
Bagian Input/output**

**Process**

**Data Store**

**Aliran Data**

harga

gabole ada > 1 bulatan/proses

sampah

object

koin

Analisis Kebutuhan P/L - Terstruktur



KNOWLEDGE & SOFTWARE ENGINEERING

# Pengembangan DFD

- Dimulai dengan pengembangan **Diagram Konteks**
  - **Diagram Konteks** digunakan untuk menjelaskan **keterhubungan sistem** yang dikembangkan dengan **sistem lain** atau **aktor luar** atau entitas eksternal
    - Contohnya Aktor pada Vending Machine: Para pengguna VM, Pemilik VM
    - Aktor ini bisa 'manusia' atau 'sistem'
      - Contohnya: sistem PL untuk mesin ATM dapat saja berhubungan dengan Bank A, Bank B, dan Bank C. Jadi Aktor/Entitas Eksternal nya adalah Bank A, B dan C
  - Pada level berikutnya dibuatlah **DFD Level 0**, **Level 1**, dan seterusnya
  - **Kamus Data (Data Dictionary)** diperlukan untuk menjelaskan data atau komposisi dari suatu data

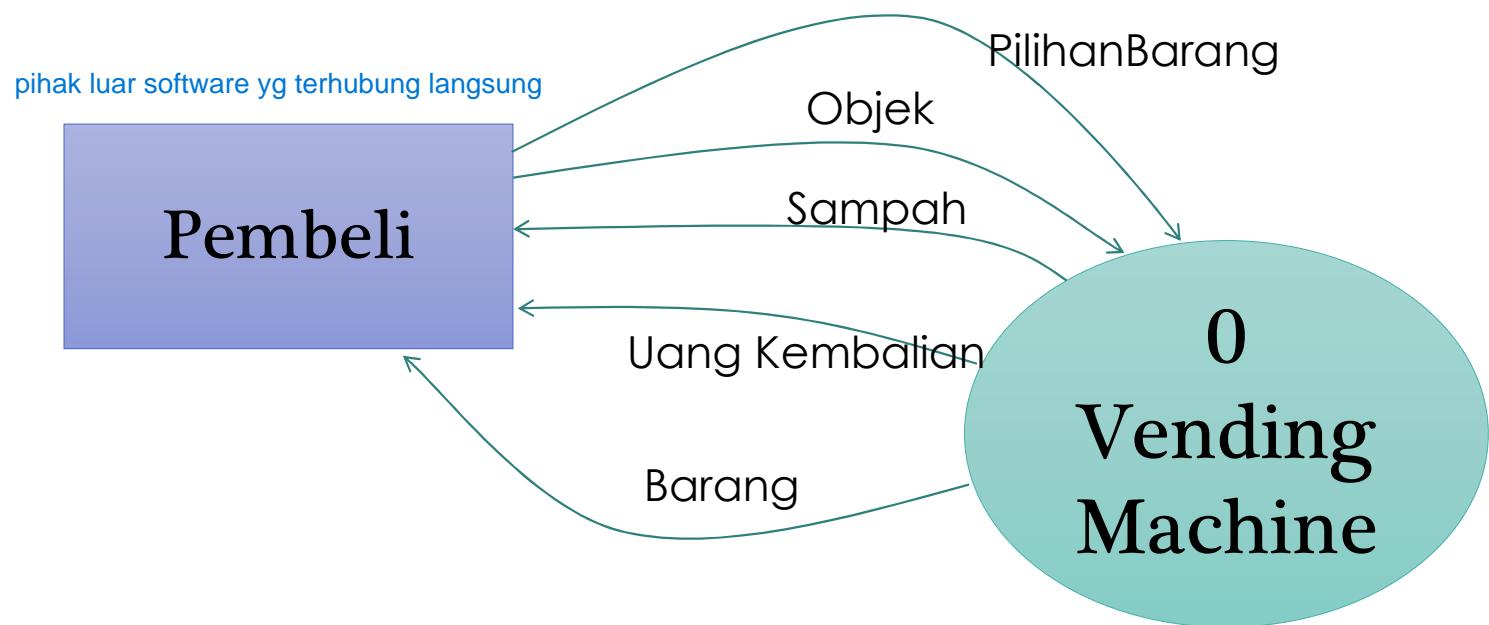
# Diagram Konteks

- Konteks diagram menjelaskan fungsi sistem dan mendefinisikan **jangkauan dari sistem**
- Hanya ada **satu lingkaran proses**
- Minimal ada **satu input** dan **satu output**
- Minimal ada **satu entitas eksternal/terminator**
- Semua terminator terhubung minimal satu aliran input atau output
- Tidak ada pertukaran data antara terminator



KNOWLEDGE & SOFTWARE ENGINEERING

# Diagram Konteks (I)



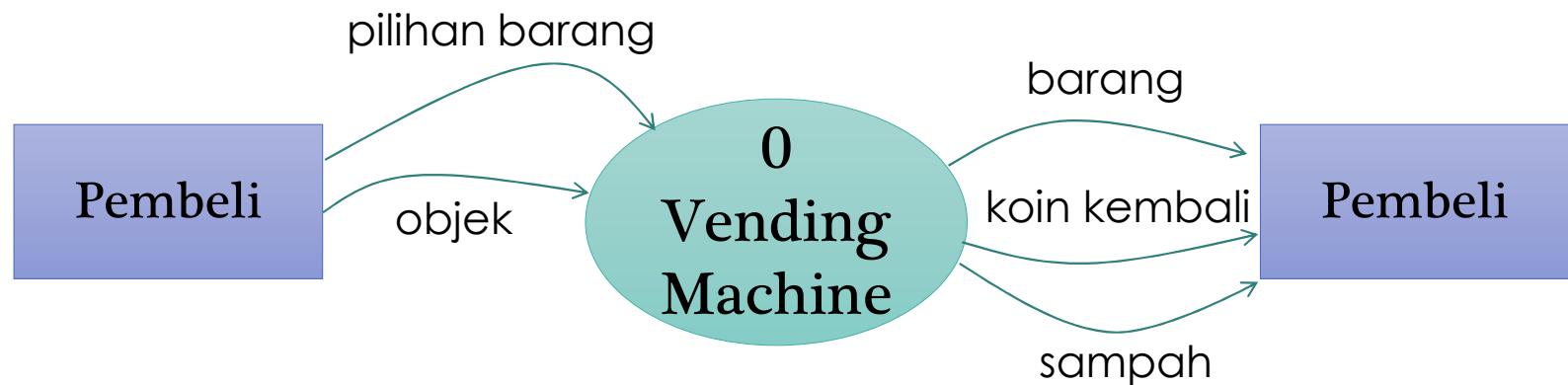
KNOWLEDGE & SOFTWARE ENGINEERING

Analisis Kebutuhan P/L - Terstruktur

# Diagram Konteks (2)

- Diagram konteks menunjukkan **pertukaran data** antara **sistem** dengan **lingkungannya**.
- Terminator ditunjukkan pada diagram konteks karena **terminator bukan bagian** dari sistem atau software yang akan dikembangkan
- **Data store** bukan bagian dari diagram konteks karena dianggap sebagai **bagian dari sistem**
- Diagram konteks hanya terdiri dari
  - **Satu proses** (sistem /software nya)
  - **Beberapa terminator/entitas eksternal**
  - **Aliran data**

# Diagram Konteks (3)



# Diagram Konteks (4)

- Konteks diagram dibentuk secara **iteratif**
  - Terutama untuk pembentukan batasan sistem (system boundary)
- Untuk sistem yang kompleks, kadang kita harus menggambar dulu beberapa DFD sebelum batasannya (boundary) dapat kita kembangkan
- Batasan ini juga dapat berubah sesuai dengan kebutuhan yang baru muncul.



KNOWLEDGE & SOFTWARE ENGINEERING

# *Dekomposisi Fungsional untuk DFD*

- Kebutuhan fungsional didekomposisi, dengan mengikuti aturan:
  - Fungsi-fungsi yang **terkait disatukan** dalam suatu kelompok
  - Fungsi-fungsi yang **tidak terkait dipisahkan**
  - **Setiap fungsi** hanya deskripsikan **satu kali** saja.



KNOWLEDGE & SOFTWARE ENGINEERING

# Panduan DFD (I)

- Setiap lingkaran menggambarkan satu proses
- Setiap **proses** akan **didekomposisi** menjadi proses lain dalam satu DFD atau menjadi **algoritma**  
max ada 9 bulatan pd 1 level
- Gunakan aturan **7 ± 2**
  - Sebaiknya 5 sampai 9 proses dalam satu gambar diagram
  - Keseimbangan antara kurang dari 4 yang artinya kurangnya informasi atau terlalu banyak bila lebih dari 9 proses dalam satu diagram
- Umumnya dekomposisi mencapai 4 level
- Tiap diagram DFD adalah dekomposisi dari lingkaran orangtuanya.

# *Panduan DFD (2)*

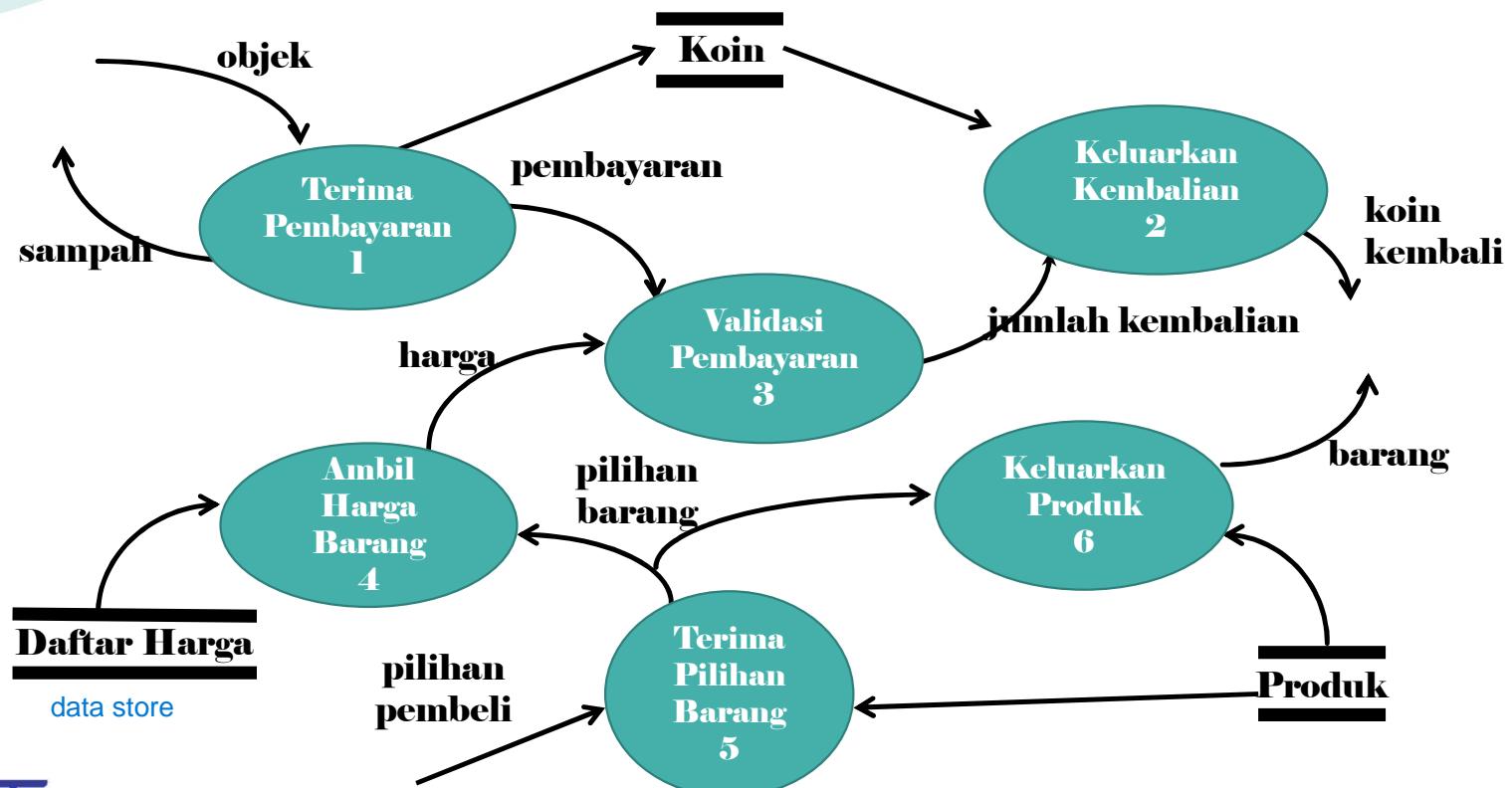
- Proses-proses ini adalah **elemen** yang **aktif** dalam model
- Proses melakukan **transformasi**
  - Jika dan hanya jika semua **informasi keluaran** tersedia
  - Semua **transformasi** informasi bersifat **langsung**
- Keluaran dari suatu proses sebaiknya adalah suatu **hasil fungsi** dari inputnya
- Aliran data memungkinkan transitnya data dalam sistem
- Data Store adalah tempat penyimpanan informasi

# *Panduan DFD (3)*

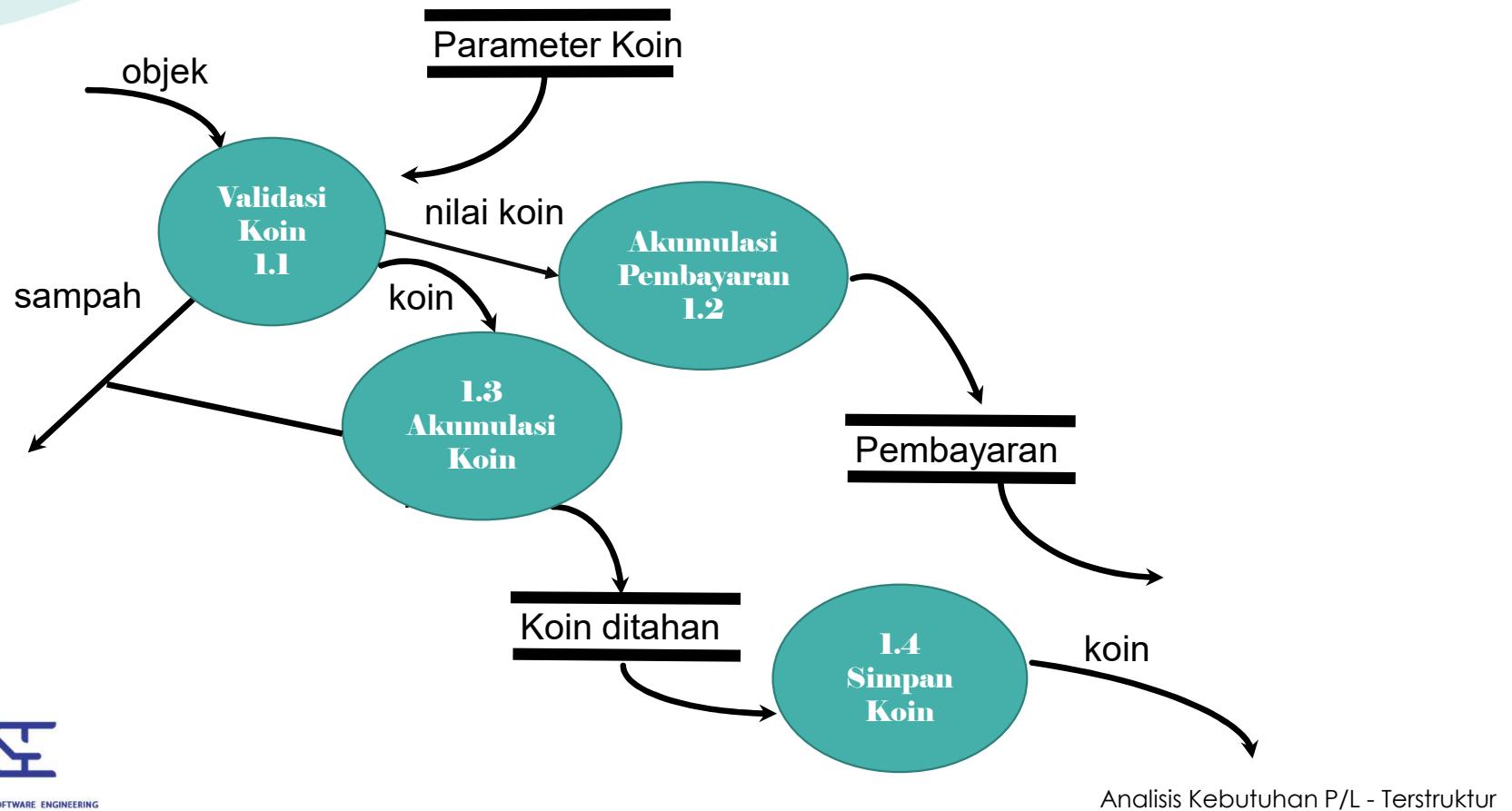
- Balancing
  - Input dan output dari setiap proses hasil dekomposisi suatu proses harus sesuai dengan induknya
- Leveling
  - Proses dekomposisi disebut sebagai leveling
- Ketika suatu proses **tidak bisa** lagi **dipecah**, maka akan menjadi **primitif fungsional** dan akan diperjelas dengan **PSPEC**
  - PSPEC ditulis hanya untuk primitif fungsional

# DFD Level 0

angka bukan as urutan



# ***DFD Level I: 1. Terima Pembayaran***



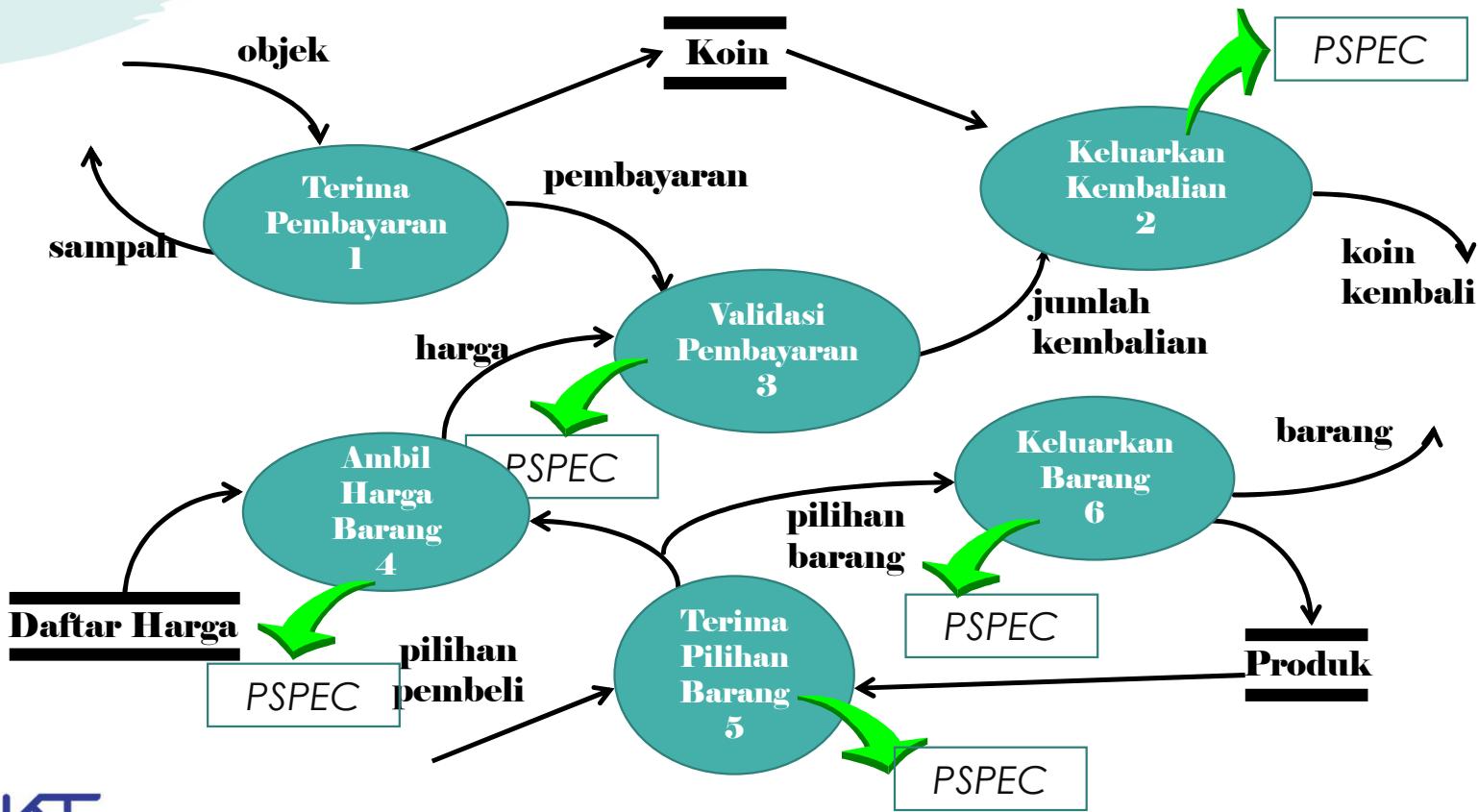
# *Spesifikasi Proses (PSPEC – Process Specification)*

- PSPEC adalah **level abstraksi** yang paling **rendah** (di DFD)
- PSPEC sepanjang kira-kira  $\frac{1}{2}$  halaman
- Menunjukkan hubungan antara input proses dan aliran output
- Dapat menggunakan berbagai bentuk spesifikasi
  - Gambar
  - Persamaan matematika
  - Bahasa sehari-hari

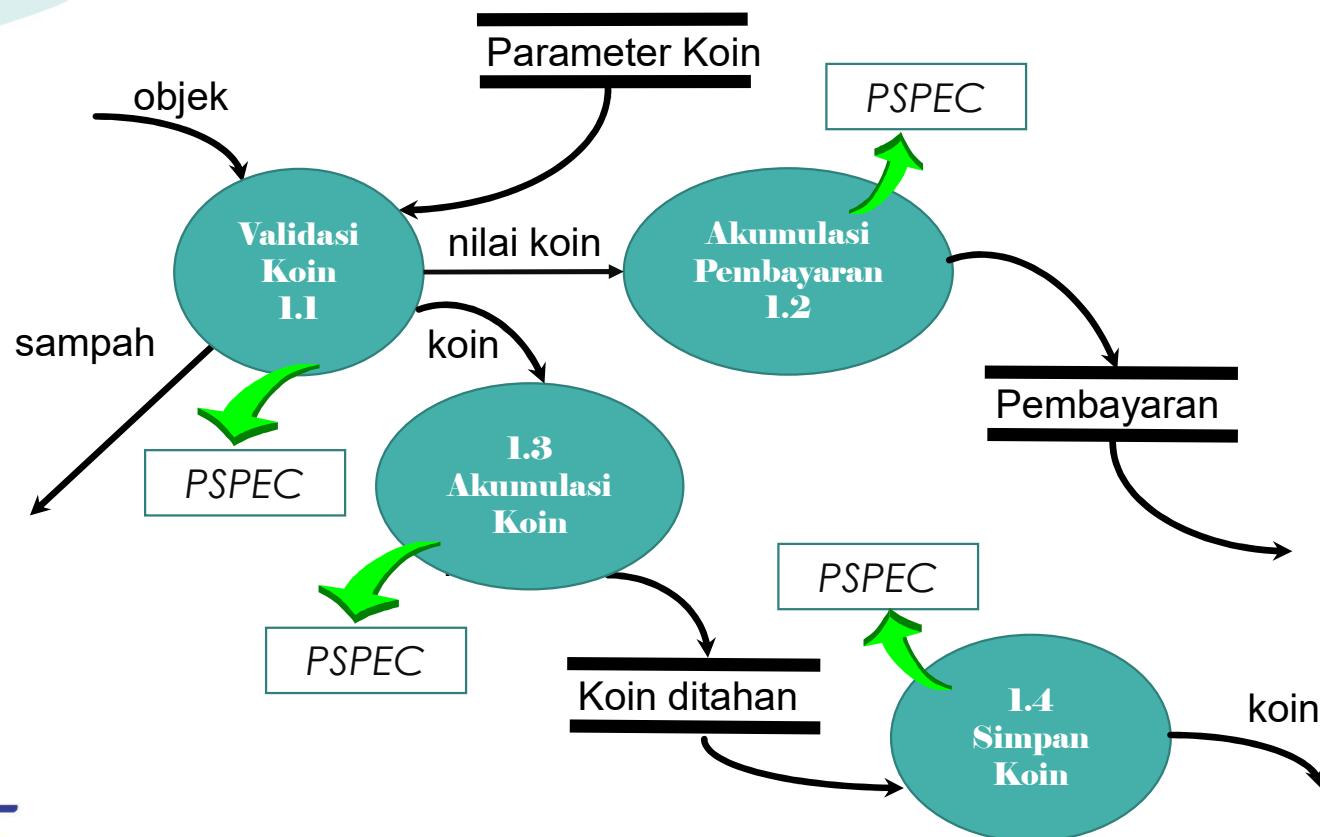


KNOWLEDGE & SOFTWARE ENGINEERING

# Spesifikasi Proses (PSPEC – Process Specification)



# **DFD Level I: 1. Terima Pembayaran**



# ***PSPEC 3: Validasi Pembayaran***

## **Inputs:**

pembayaran : data in  
harga : data in

## **Outputs:**

jumlah kembali : data out

## **Body:**

```
If (pembayaran >= harga)
    jumlah kembalian = payment – harga
else
    jumlah kembalian = 0
```

## ***PSPEC 2: Keluarkan Kembalian***

### **Inputs:**

koin : data in  
jumlah kembali : data in

### **Outputs:**

koin kembali : data out

### **Body:**

koin kembali adalah jumlah koin yang diambil dari sebanyak ‘jumlah kembali’

# **PSPEC lainnya (I)**

## **PSPEC 4: Ambil Harga Barang**

Keluarkan harga barang berdasarkan pilihan barang yang diambil dari tabel daftar harga

## **PSPEC 5: Terima Pilihan Barang**

Pilihan barang diambil berdasarkan pilihan pembeli jika produk yang dibeli ada dari daftar tabel ‘produk’

## **PSPEC 6: Keluarkan Barang**

Keluarkan Barang berdasarkan pilihan barang, update jumlah barang dari tabel produk

# ***PSPEC lainnya (2)***

## **PSPEC 1.1: Validasi Koin**

Periksa Objek sesuaikan dengan data dari Parameter Koin.

Jika benar maka terima objek sebagai koin, jika tidak maka keluarkan objek sebagai sampah

## **PSPEC 1.2: Akumulasi Pembayaran**

Tambahkan nilai koin ke data pembayaran.

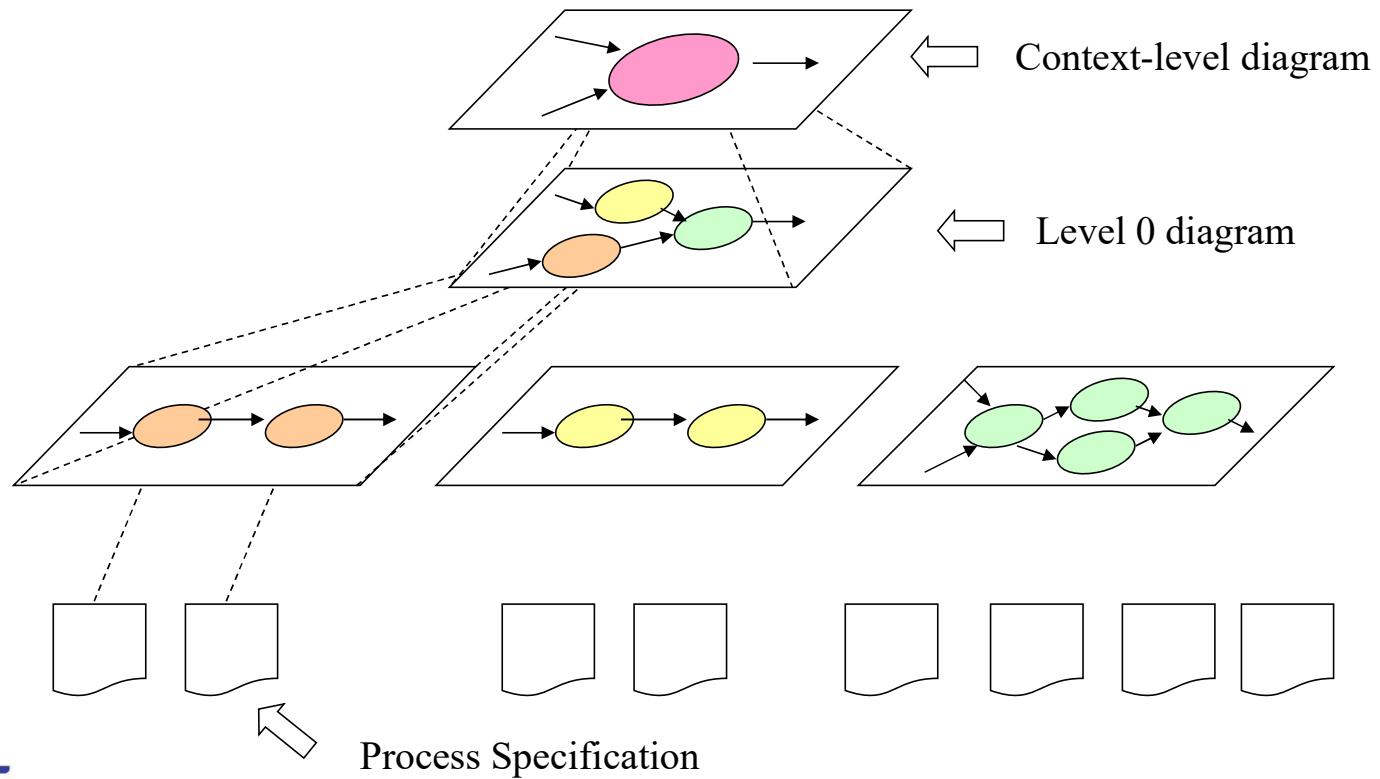
## **PSPEC 1.3: Akumulasi Koin**

Bila jumlah koin melebihi kapasitas, maka koin dianggap sampah

## **PSPEC 1.4: Simpan Koin**

Pindahkan koin yang di tahan ke penampungan koin

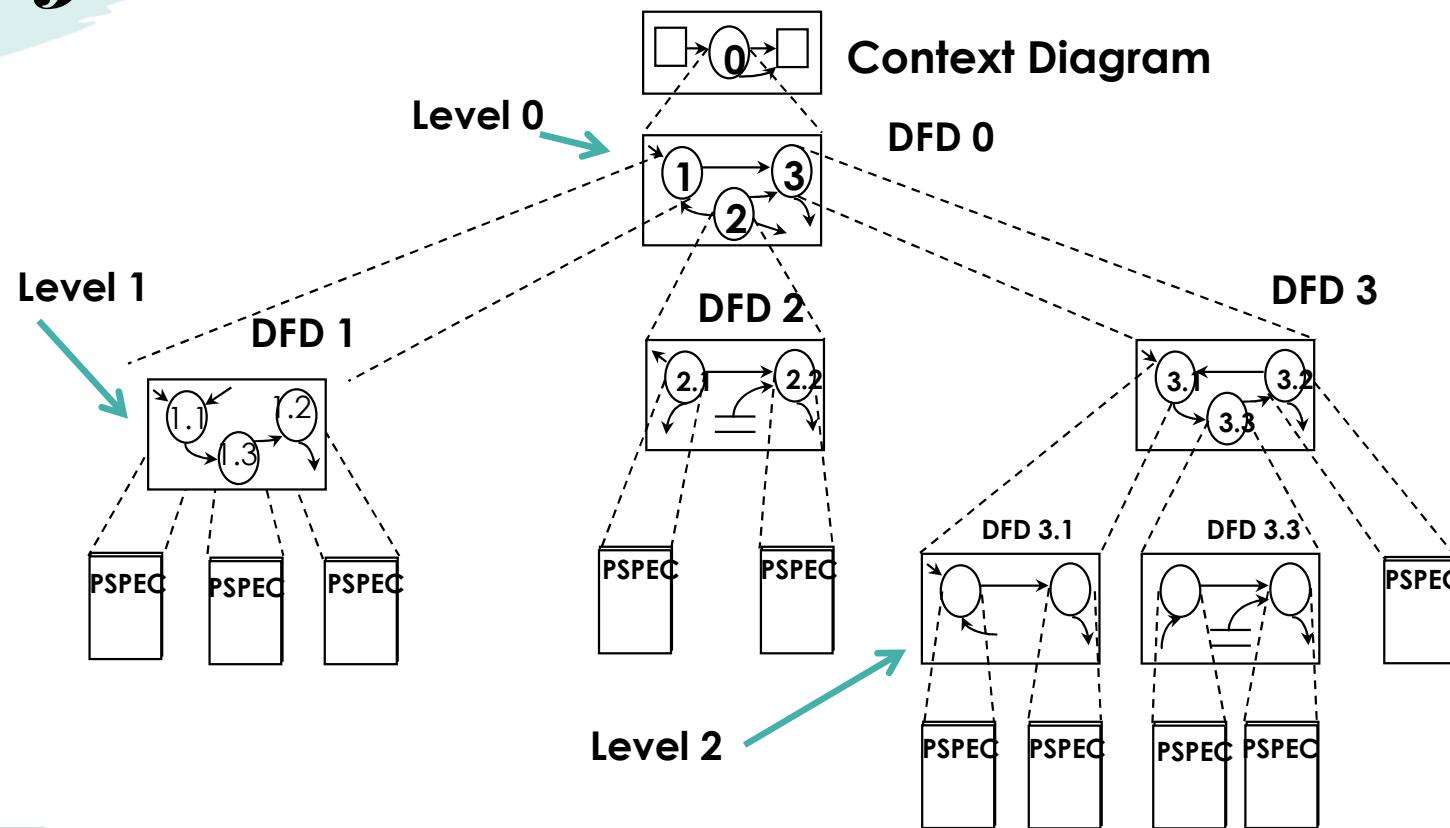
# *Diagram Layering and Process Refinement*



KNOWLEDGE & SOFTWARE ENGINEERING

Analisis Kebutuhan P/L - Terstruktur

# Rangkuman



KNOWLEDGE & SOFTWARE ENGINEERING

Analisis Kebutuhan P/L - Terstruktur

# *Kamus Data (Data Dictionary)*

- Berisi sekumpulan nama data dengan **definisinya**
- Setiap **aliran data** harus **ada kamusnya**
- Nama-nama kelompok harus **dipecah** hingga bentuk yang paling **elementer**
- **Elemen primitif** memiliki atribut seperti: **unit**, **range**, **akurasi**
- **Kamus data** ini akhirnya akan menjadi bagian dari **basisdata** yang digunakan untuk mendukung pemodelan.

# Aturan Penulisan

- = terdiri dari
- + digabungkan dengan
- { } pengulangan
- [..|..] terdiri dari
- ( ) optional item
- “ ” literal
- \* \* memberikan insight tambahan tentang arti

# *Kamus Data (silakan dikembangkan lagi)*

Nama	Deskripsi
Objek	= [koin   sampah]
Barang	= [soda   permen   chips]
koin	= [ 100an   200an   500an   1000an]
100an	= *uang koin 100 rupiah*



# Aturan Balancing Kebutuhan

- Setiap DFD harus seimbang (balance) dengan induknya
- Setiap PSPEC harus seimbang dengan proses primitif fungsi yang terkait
- Setiap **aliran data**, dan **data store** harus terdefinisi, dan harus **terdekomposisi** menjadi **elemen primitif**
  - Elemen primitif ini akan didefinisikan di kamus data



KNOWLEDGE & SOFTWARE ENGINEERING

# Evaluasi DFD

- Apakah semua **level** memang sudah yang **diperlukan**?
- Apakah **nama-nama** sudah **jelas**?
- Apakah setiap **bagian** sudah punya **nama** dan dapat dimengerti **maknanya**?
- Apakah ada aliran yang hilang?
- Apakah semua **proses** sudah sesuai dengan **kebutuhan** ?
- Apakah fungsi yang tidak terhubung sudah dipisahkan dan fungsi yang terhubung dikelompokkan bersama?

# ***ANALISIS TERSTRUKTUR***

## *State Transition Diagram*



Analisis Kebutuhan P/L - Terstruktur

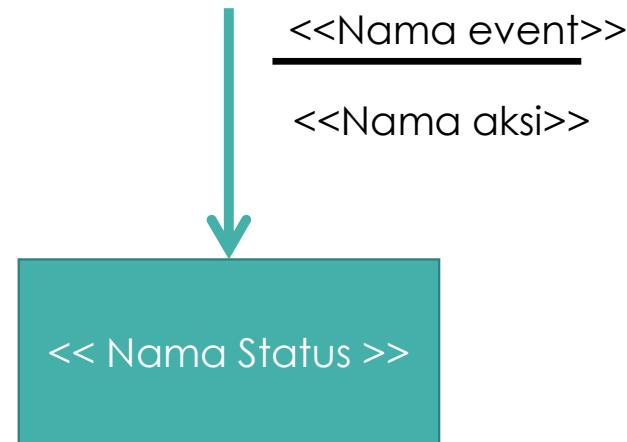
# *State Transition Diagram (STD) (I)*

- STD menjelaskan **perilaku dinamika** sistem
  - Sistem akan memberikan **respon** terhadap suatu **stimulus** atau **event** (kejadian)
  - **Transisi** (perubahan) **status** dari **suatu entitas**
  - **Trigger** atau **event** (kejadian) yang menyebabkan **perubahan status** dari **suatu entitas**.
  - Contoh:
    - Manusia menekan keyboard (event) maka di layar tampil huruf (status)
    - Pembeli memasukkan koin (event), sistem akan menampilkan jumlah koin (status)
    - Tombol lampu di-on-kan (event), maka lampu akan memiliki status hidup dari status mati

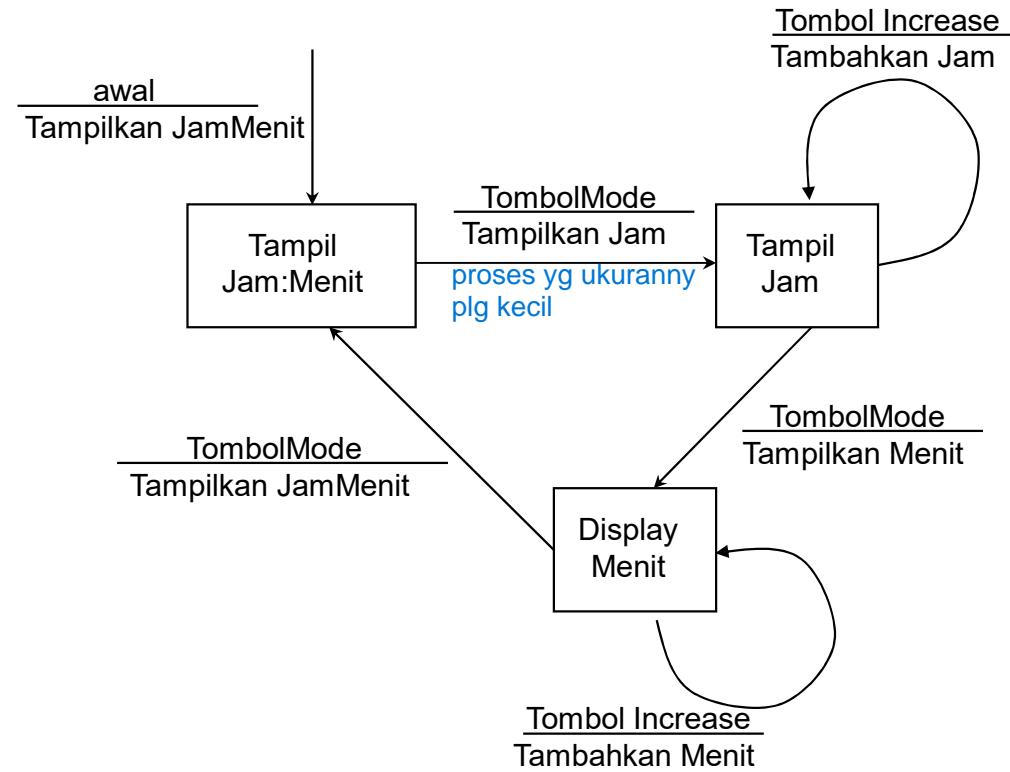
# *State Transition Diagram (STD) (2)*

- **Event** adalah suatu **kejadian** di **luar entitas**, biasanya memerlukan suatu **aksi** untuk menjalankannya
  - **Event** dapat disebabkan karena **kemunculan suatu data** atau suatu **stimulus** karena **aktivitas manusia**.
- **Aksi** adalah apa yang **dilakukan** oleh **entitas** yang dimodelkan sebagai **respon** terhadap suatu **event**
- **Status** (state) adalah **kondisi** dari **entitas** yang dimodelkan
- **Transisi** adalah **perubahan status** karena suatu **event**, transisi digambarkan sebagai **arah panah**

# *Penggambaran STD*

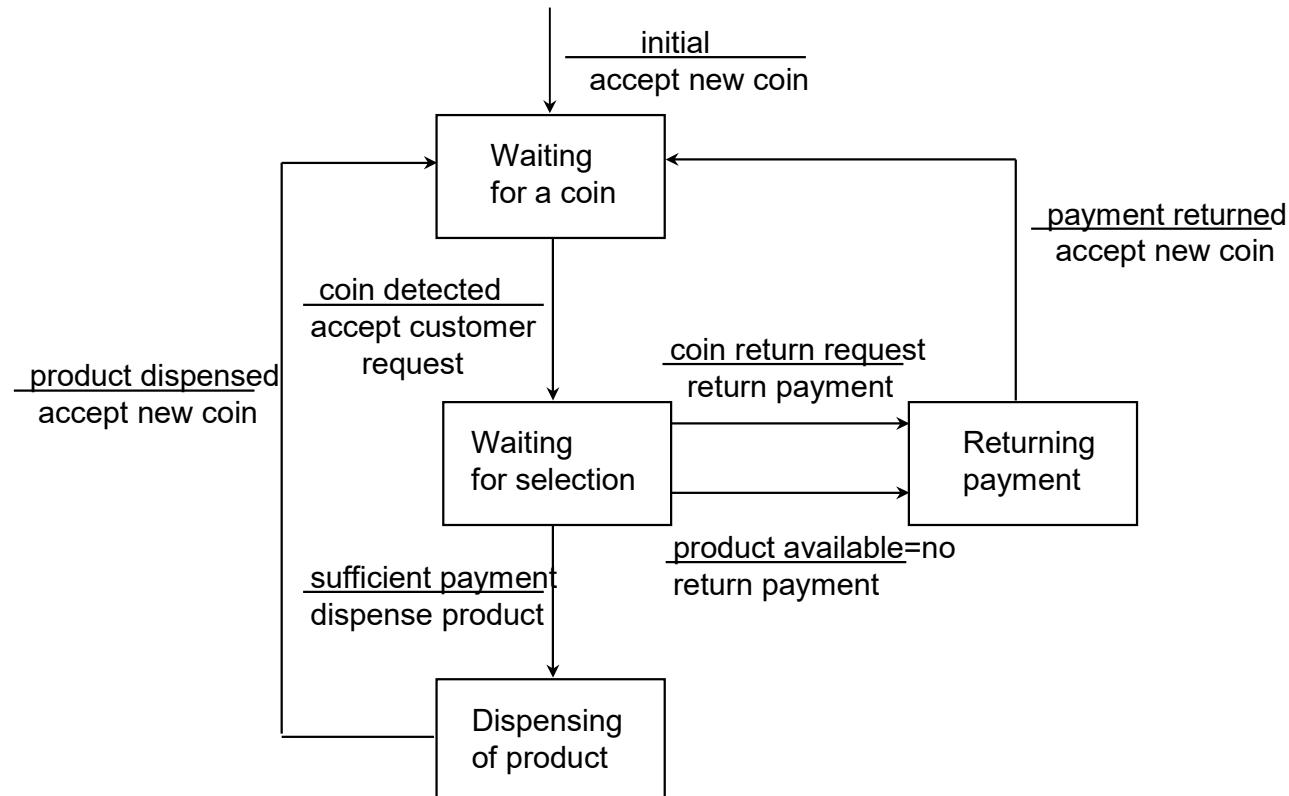


# *STD untuk Jam Digital*



KNOWLEDGE & SOFTWARE ENGINEERING

# *STD untuk Vending Machine <<draft>>*

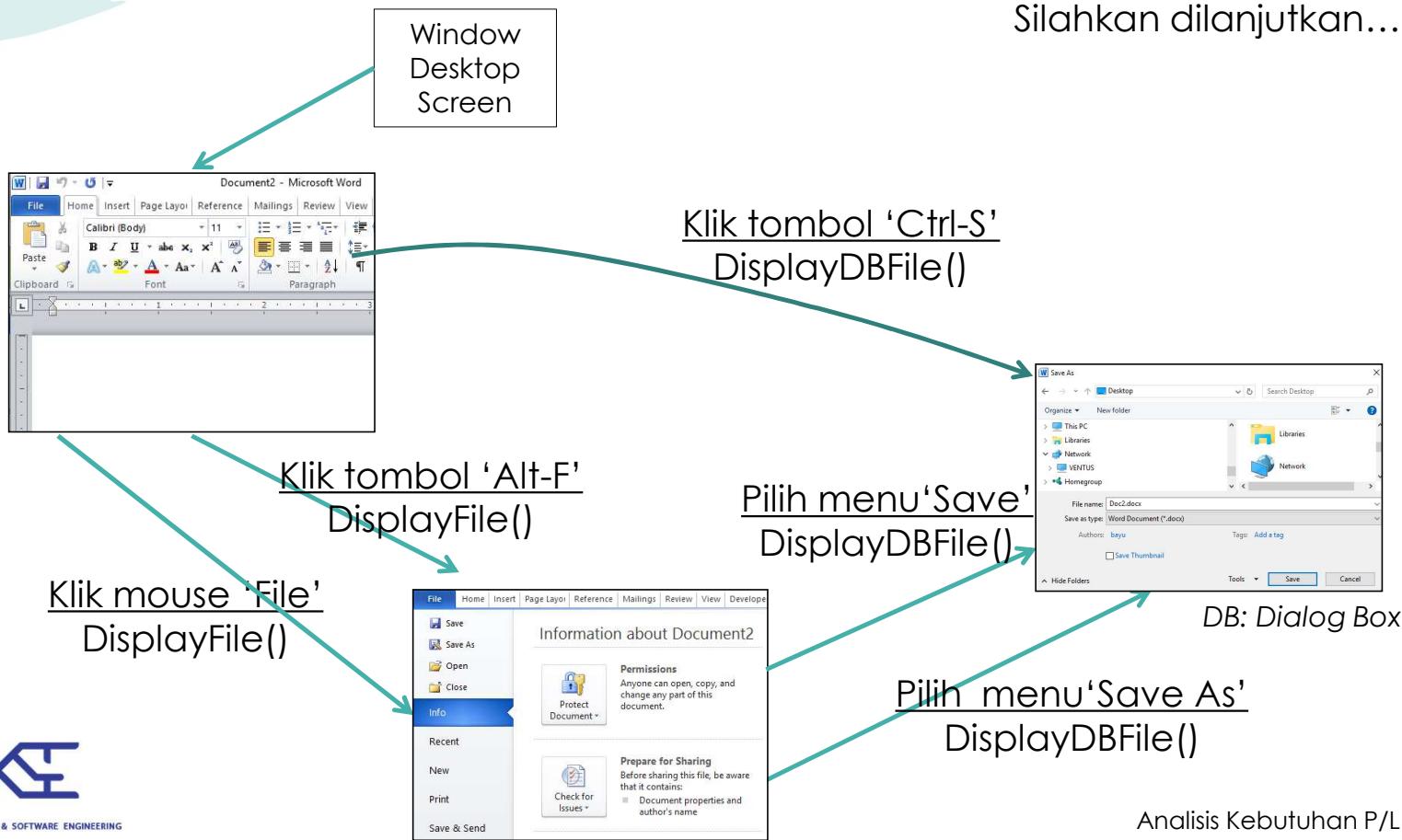


KNOWLEDGE & SOFTWARE ENGINEERING

# *STD untuk Eksekusi program pada Program MS Word*

42

Silahkan dilanjutkan...



# *Contoh implementasi STD dalam Program*

```
while (1)
{
    if (state == JamMenit) and (event == TombolMode) then
        TampilkanJam();
    if (state == Jam) and (event == TombolInc) then
       TambahkanJam();
    if (state == Jam) and (event == TombolMode) then
        TampilkanMenit();
    if (state == Menit) and (event == TombolInc) then
       TambahkanMenit();
    if (state == Menit) and (event == TombolMode) then
        TampilkanJamMenit();
    event = GetEvent();
}
```



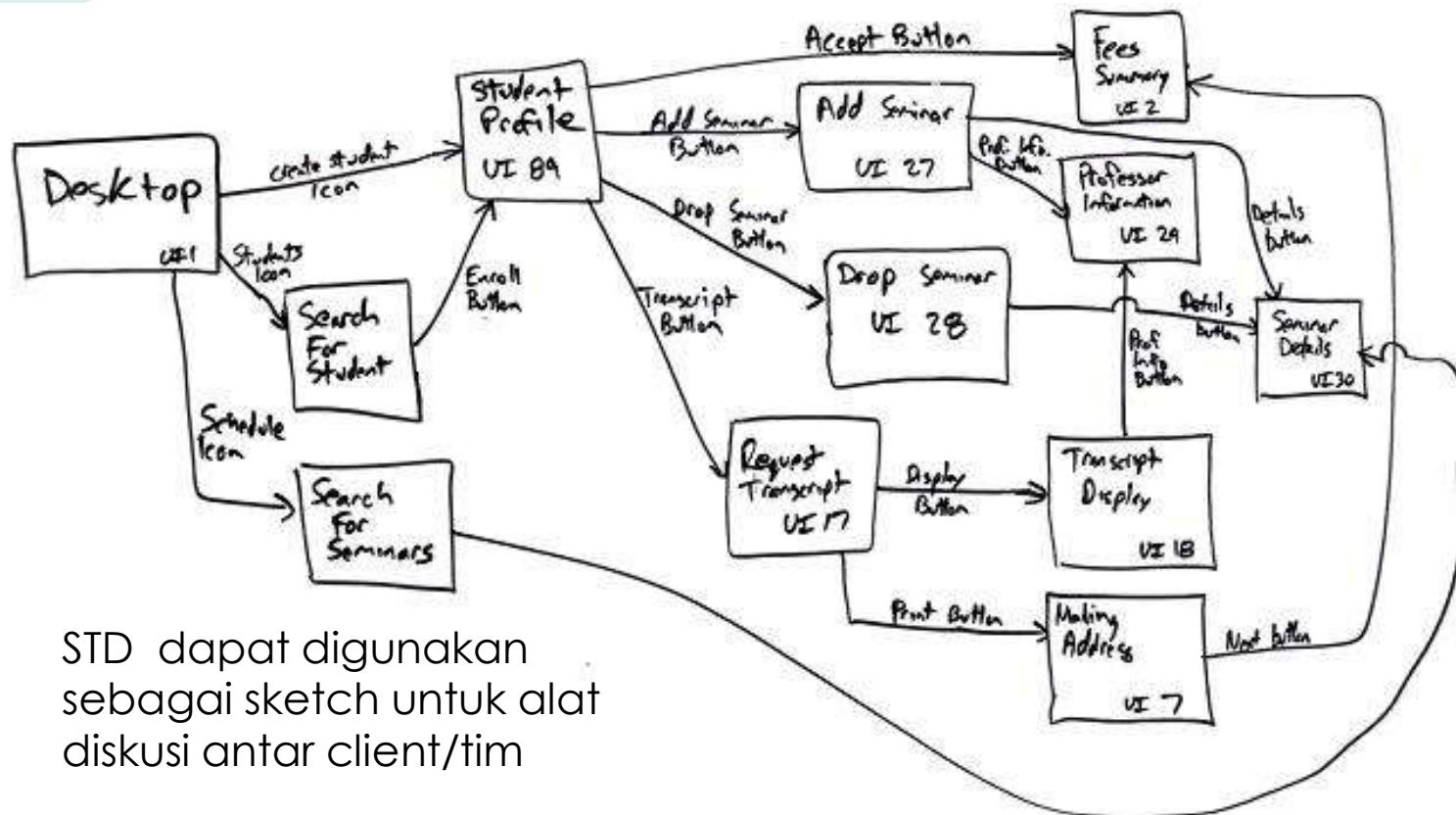
# Contoh implementasi STD dalam Program

```
while (1)
{
    switch (state)
        jamMenit: switch (event)
            TombolMode: TampilkanJam();
            TombolInc : -
            end;

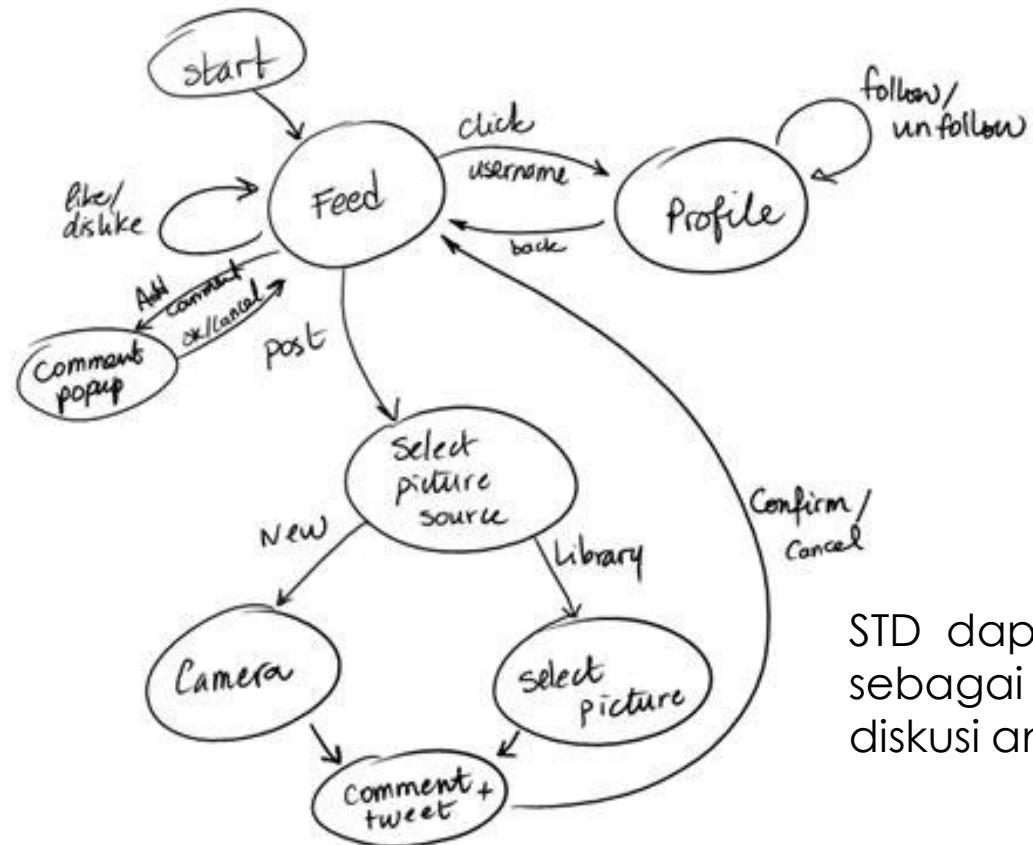
        jam :      switch (event)
            TombolMode: TampilkanMenit();
            TombolInc: TambahkanJam();
            end;
        menit:   switch (event)
            TombolMode: TampilkanJamMenit();
            TombolInc: TambahkanMenit();
            end;
    end;
    event = GetEvent();
}
```



# Varian Implementasi STD (1)



## Varian Implementasi STD (2)



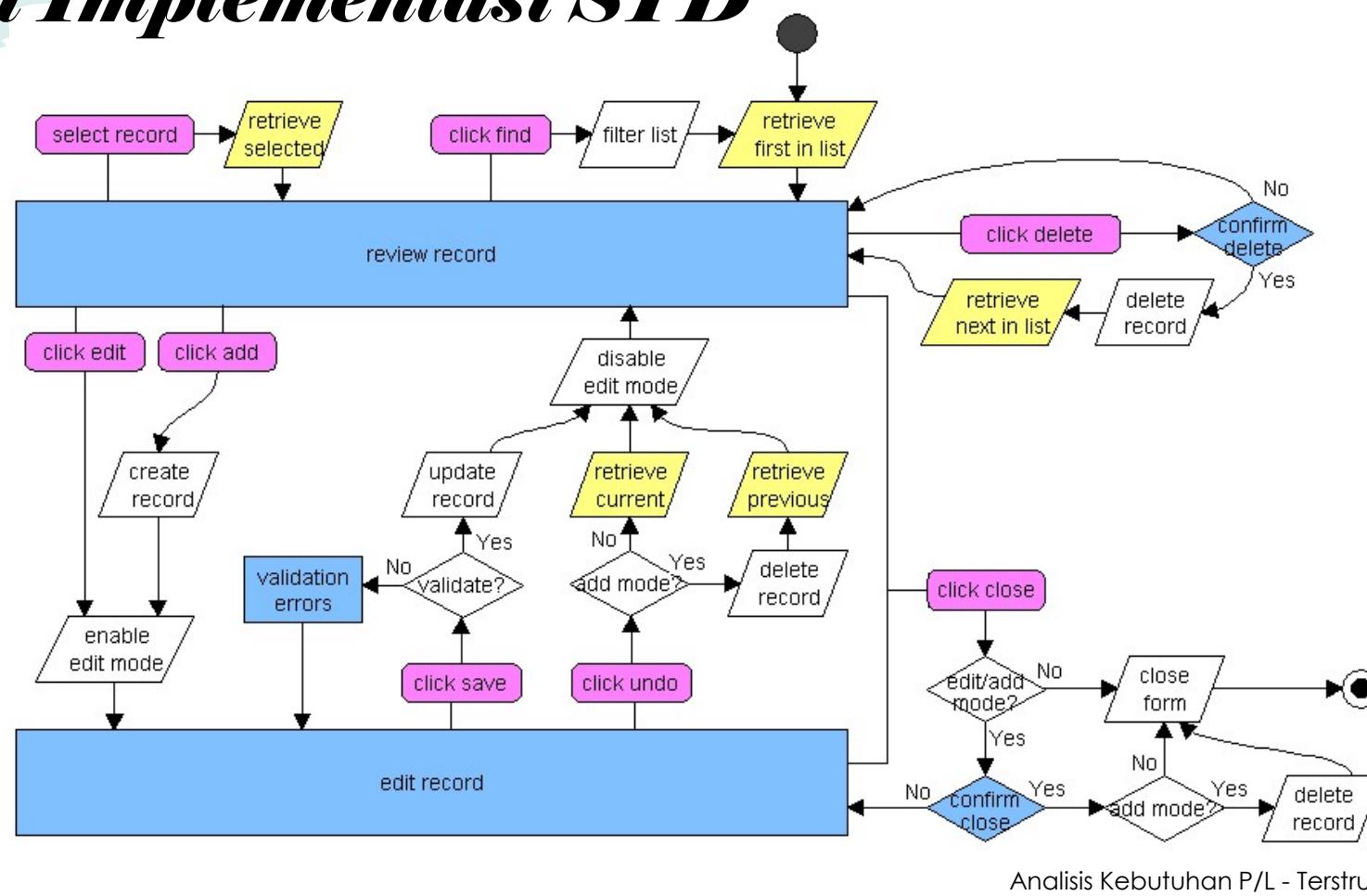
STD dapat digunakan sebagai sketch untuk alat diskusi antar client/tim



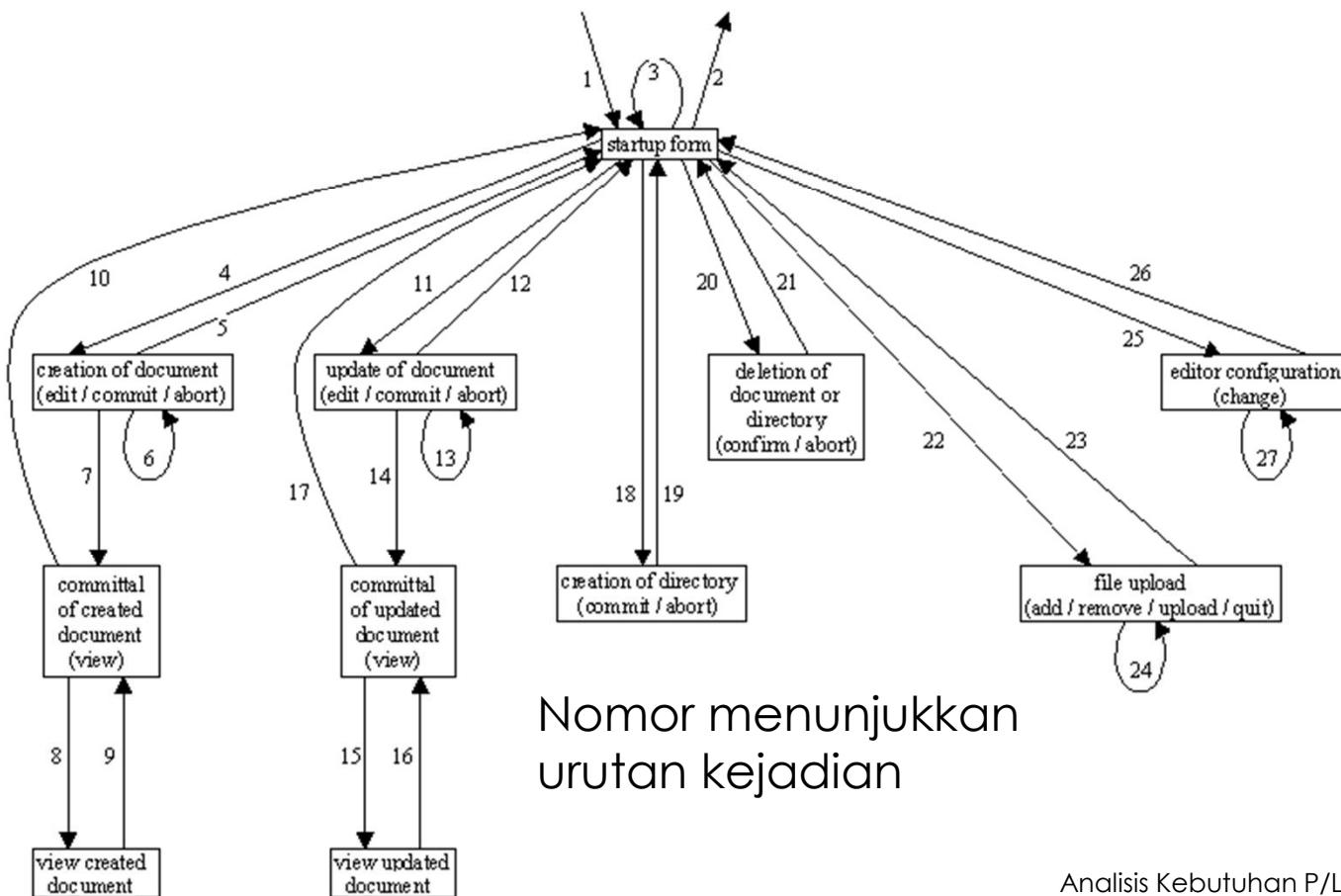
KNOWLEDGE & SOFTWARE ENGINEERING

Analisis Kebutuhan P/L - Terstruktur

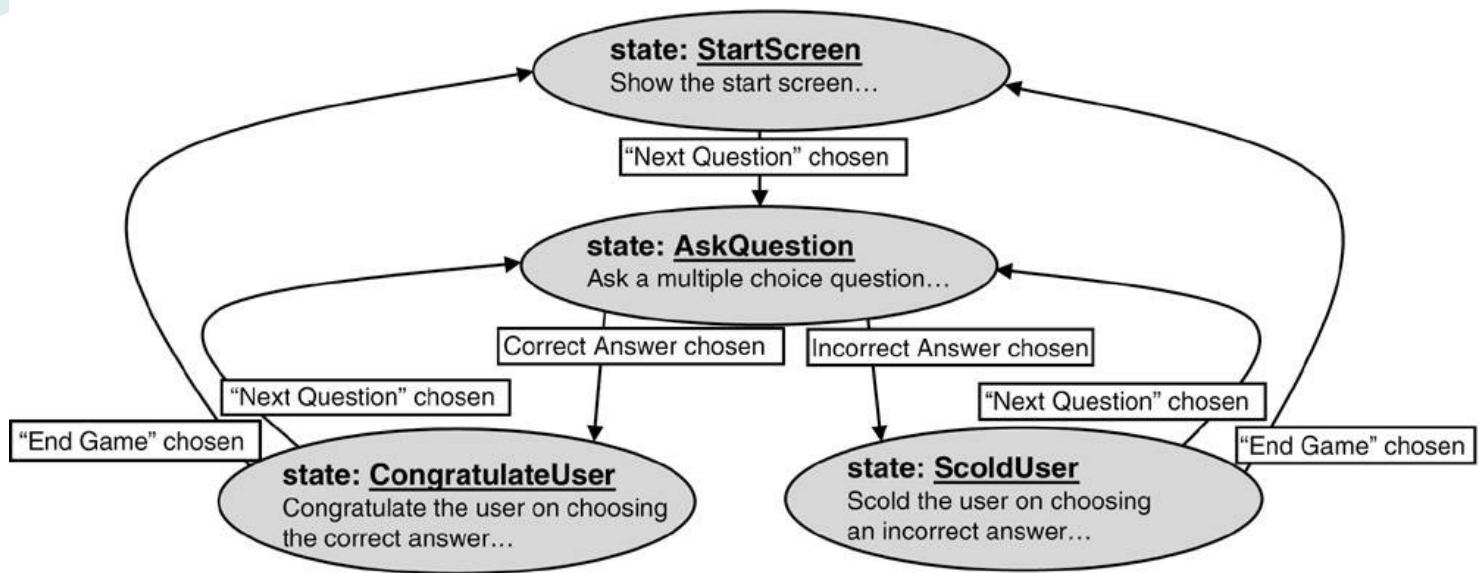
# Varian Implementasi STD



# Varian Implementasi STD

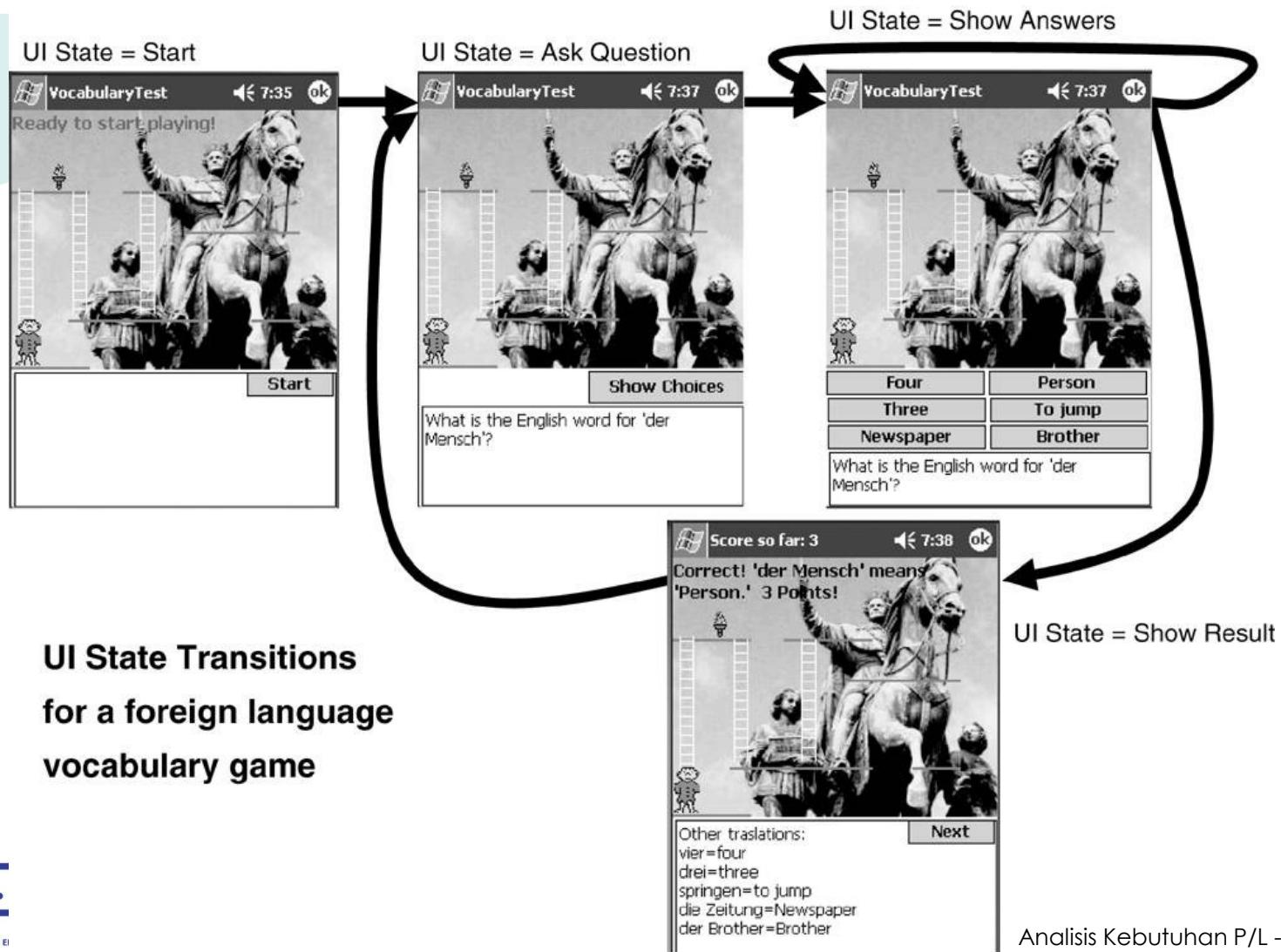


## State Machine for simple multiple choice game



Item	Meaning
oval	State
rectangle	External Input
arrow	Transition Logic





## UI State Transitions for a foreign language vocabulary game



ITB  
INSTITUT TEKNOLOGI BANDUNG  
1920  
KNOWLEDGE & SOFTWARE EI

Analisis Kebutuhan P/L - Terstruktur

# Rangkuman

relevan dgn real life

- DFD/ERD/STD adalah **abstraksi** model dari **dunia nyata**  
intinya yg penting" doang yg dikelola software
  - **DFD** menggambarkan **proses** dan **aliran data** yang **berpindah** antar **proses** DFD tidak menggambarkan urutan proses
  - **ERD** menggambarkan **hubungan** antar entitas
  - **STD** menggambarkan **urutan** perubahan **status** secara **dinamis** bila diberikan suatu **stimulus** (event)



KNOWLEDGE & SOFTWARE ENGINEERING

Analisis Kebutuhan P/L - Terstruktur

## Rangkuman (2)

- DFD/ERD/STD adalah **alat diskusi** antara **pengembang** dengan **user/customer**
  - Bentuk diagram ini umumnya lebih mudah dimengerti oleh user/customer
  - Bagi pengembang diagram ini juga membantu merancang dan mengimplementasikan solusi
  - Bentuk diagram ini **berevolusi** sesuai dengan aktivitas **pengumpulan kebutuhan**
    - Hasil final biasanya berbeda dengan diagram awal

# Rangkuman

## Iterasi, iterasi, iterasi!!

- Jangan takut untuk mulai dari awal
  - Sering terjadi ketika menyiapkan DFD di level bawah, maka induknya ikut berubah
- Kita tidak akan mengerti proses lengkap sebelum kita selesai
  - DFD akan menjadi jelas setelah diagram selesai di gambar

Tim Pengajar IF2250

# IF2250 – Rekayasa Perangkat Lunak Latihan DFD

SEMESTER II TAHUN AJARAN 2022/2023



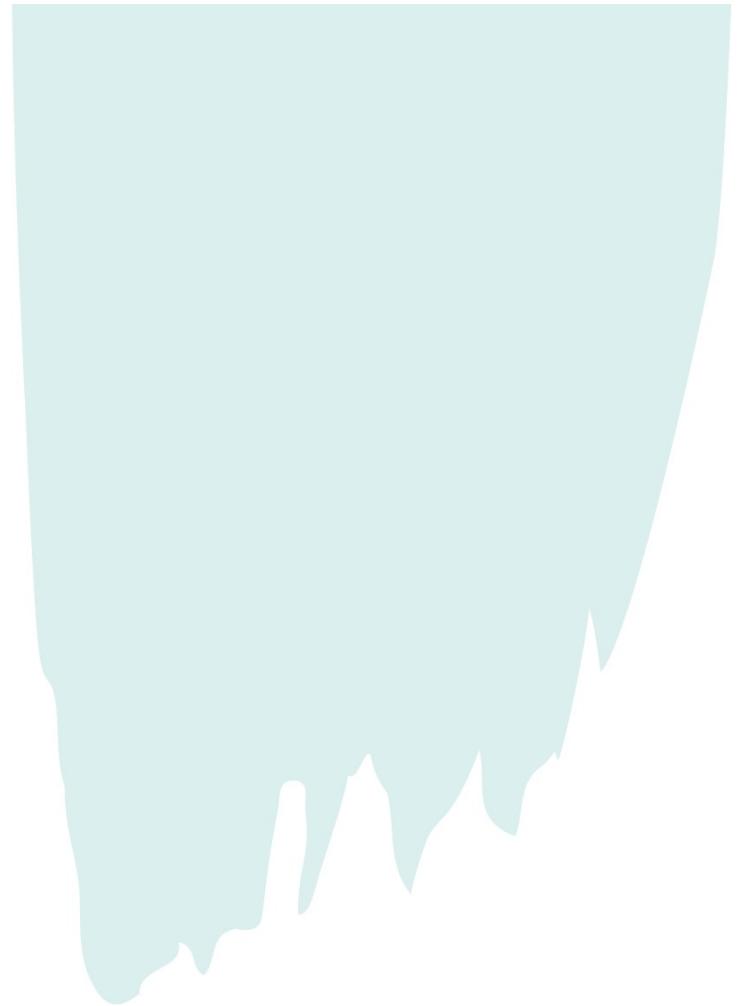
KNOWLEDGE & SOFTWARE ENGINEERING



# ***Latihan DFD***

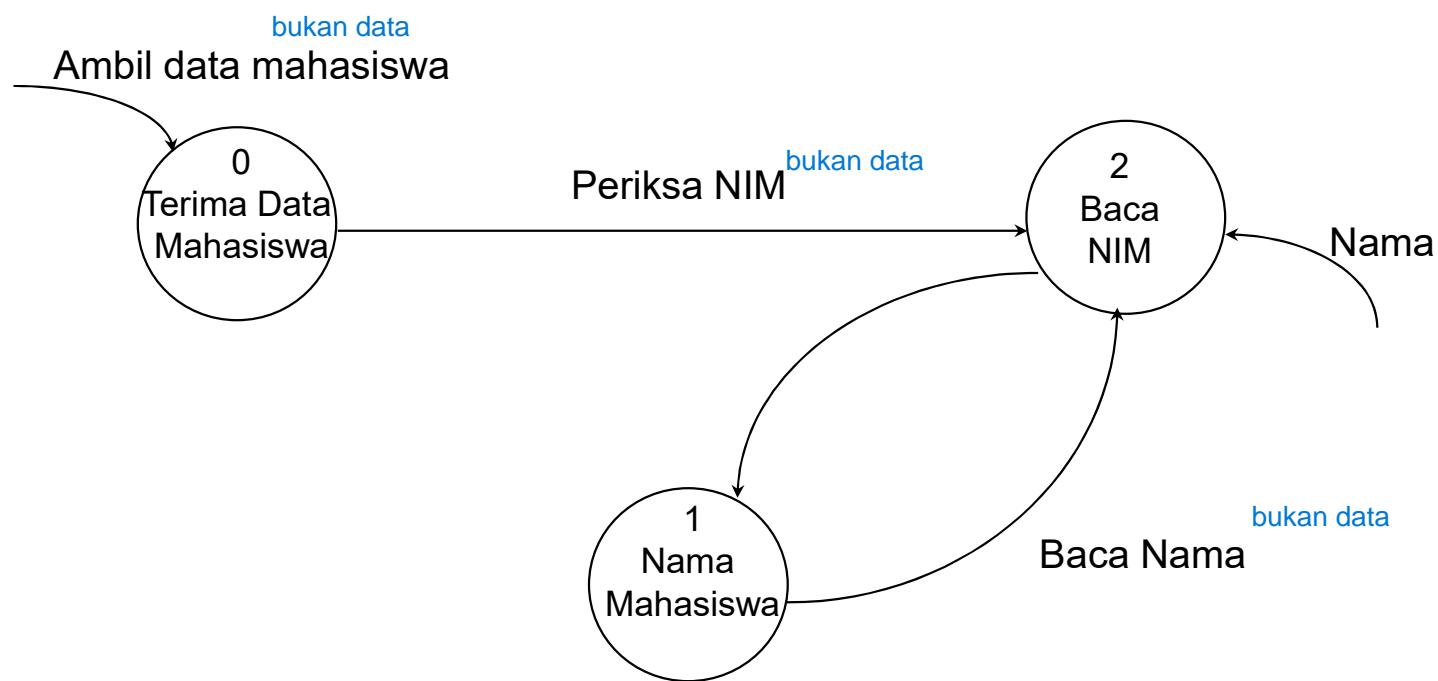


KNOWLEDGE & SOFTWARE ENGINEERING



Analisis Kebutuhan P/L - Terstruktur

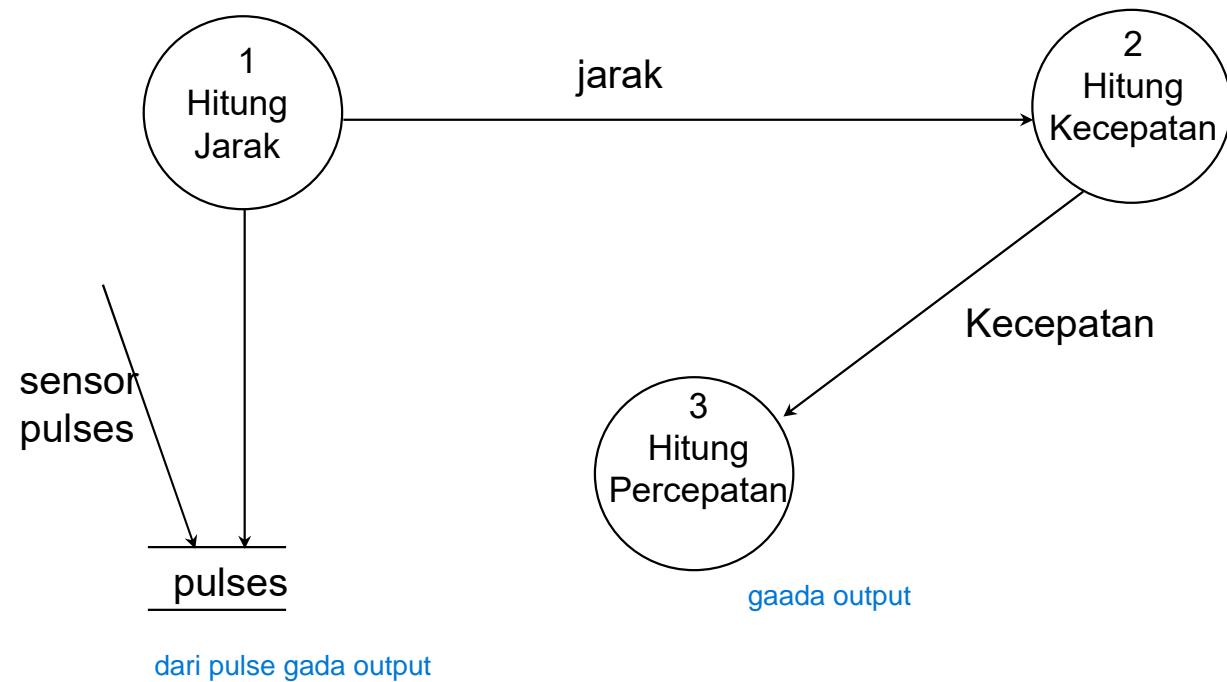
# Latihan: Cari Salah (I)



KNOWLEDGE &amp; SOFTWARE ENGINEERING

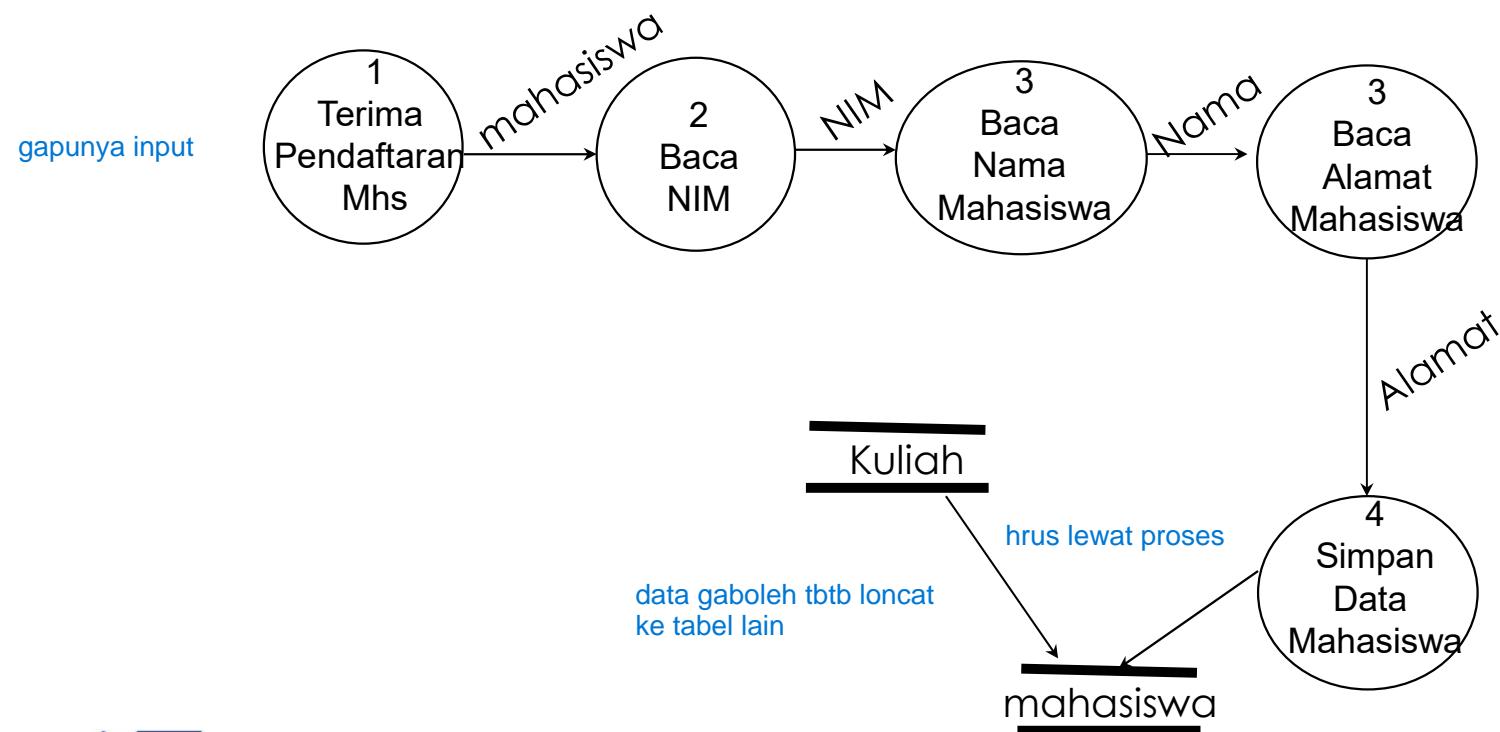
Analisis Kebutuhan P/L - Terstruktur

## Latihan: Cari Salah (2)



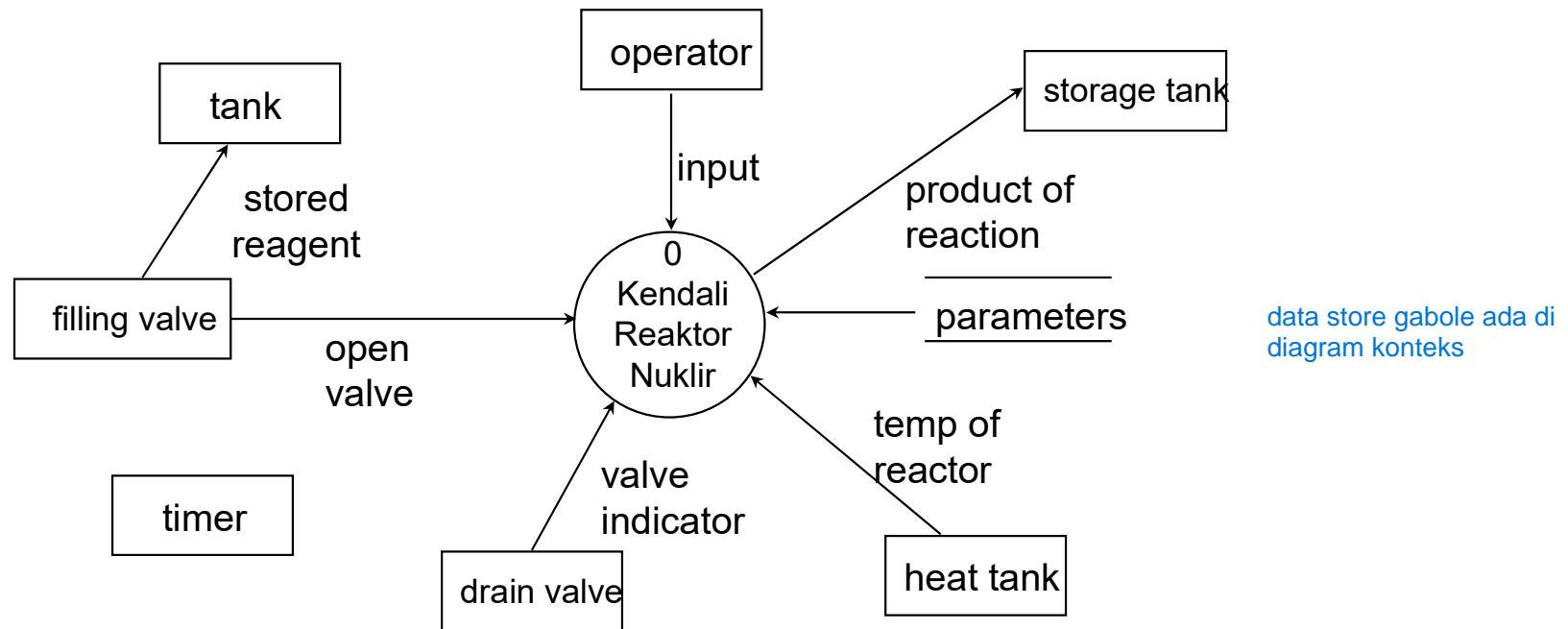
# Latihan: Cari Salah (3)

dfd ga menggambarkan urutan proses

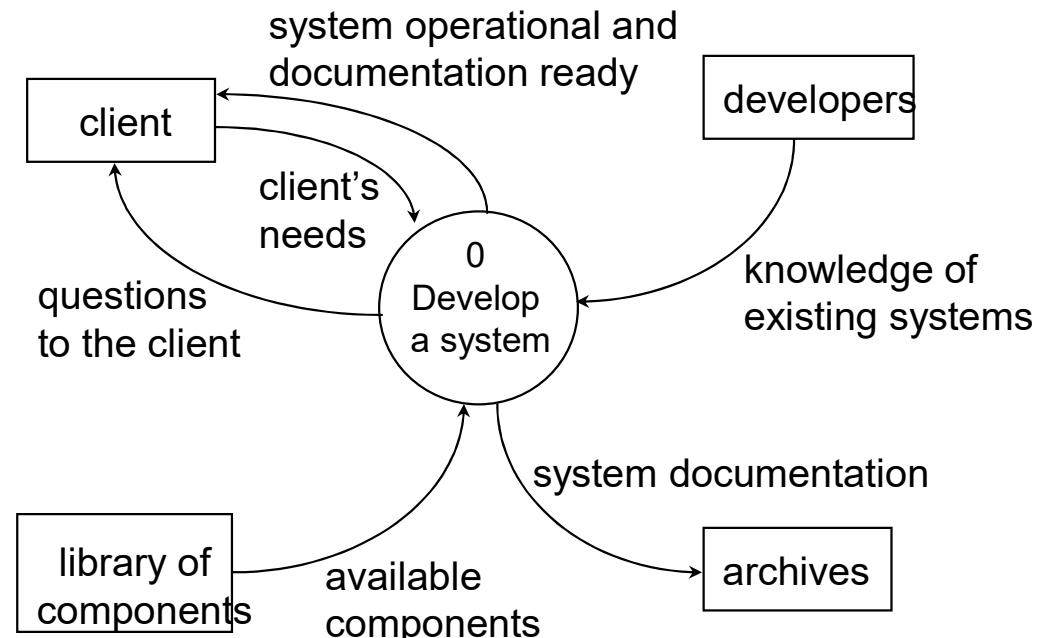


KNOWLEDGE & SOFTWARE ENGINEERING

# Latihan: Cari Salah (4)

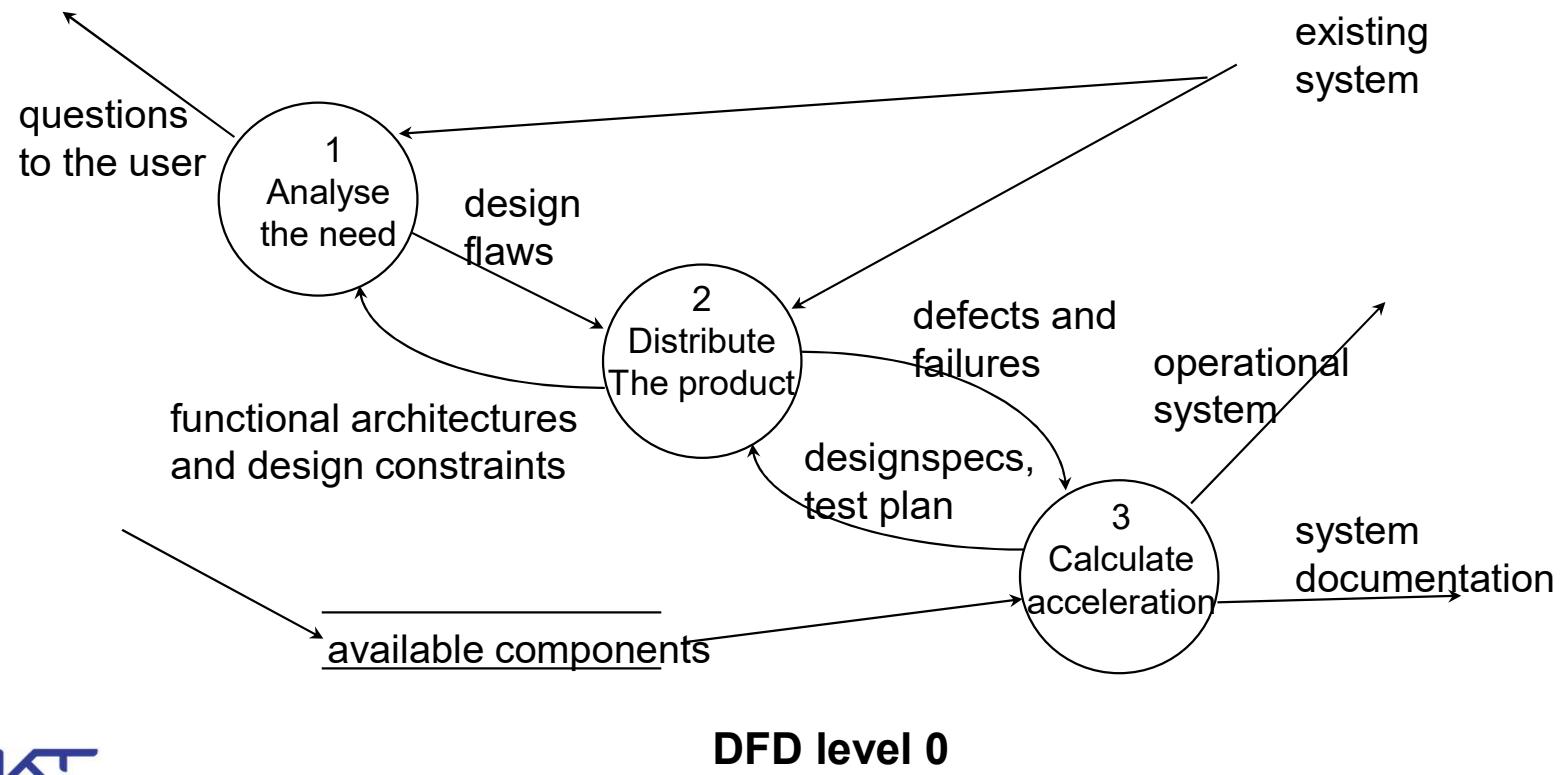


# *Latihan: Cari Salah (5)*



**Konteks Diagram**

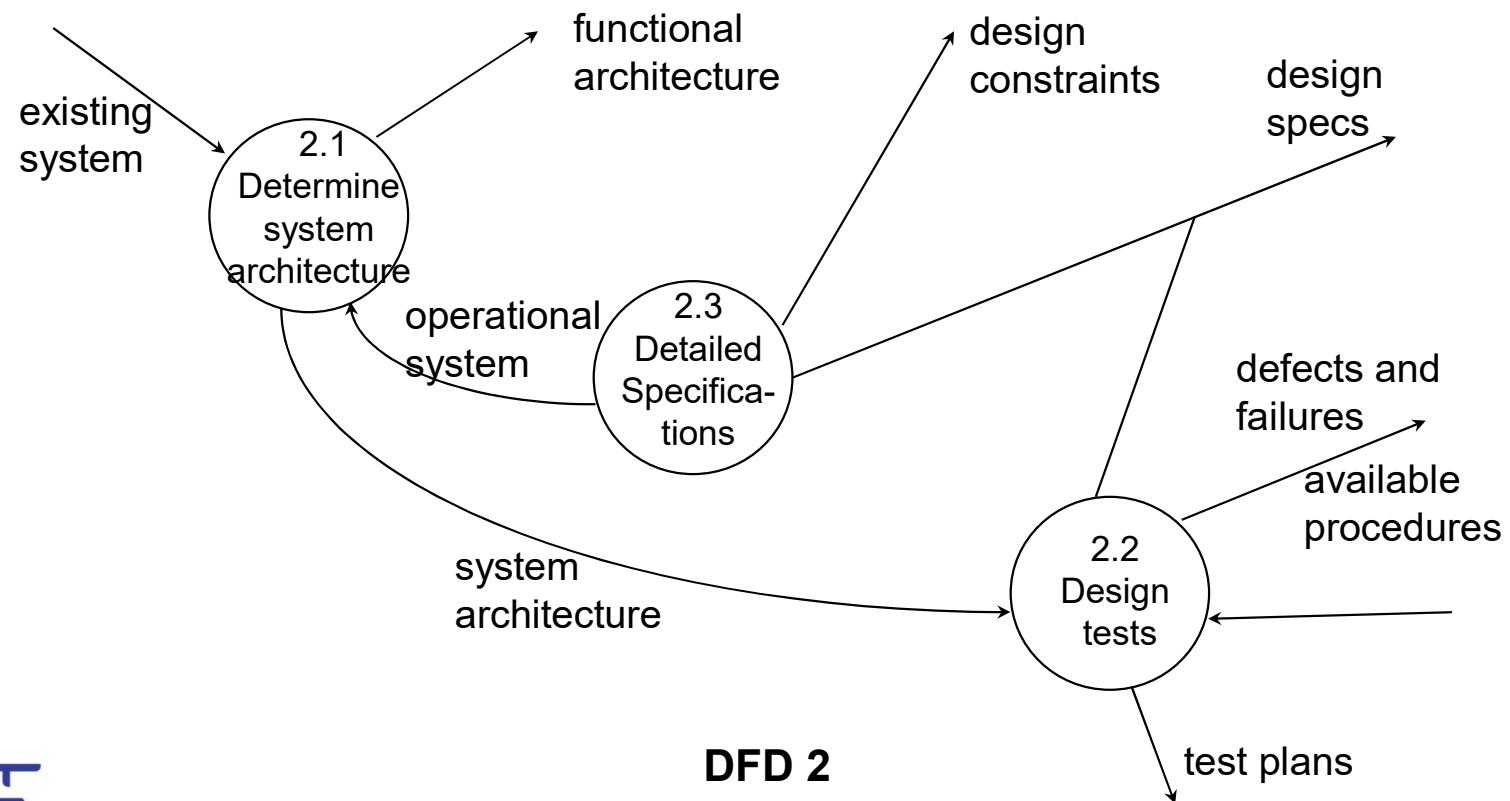
# *Latihan: Cari Salah (6)*



KNOWLEDGE & SOFTWARE ENGINEERING

Analisis Kebutuhan P/L - Terstruktur

# Latihan: Cari Salah (7)

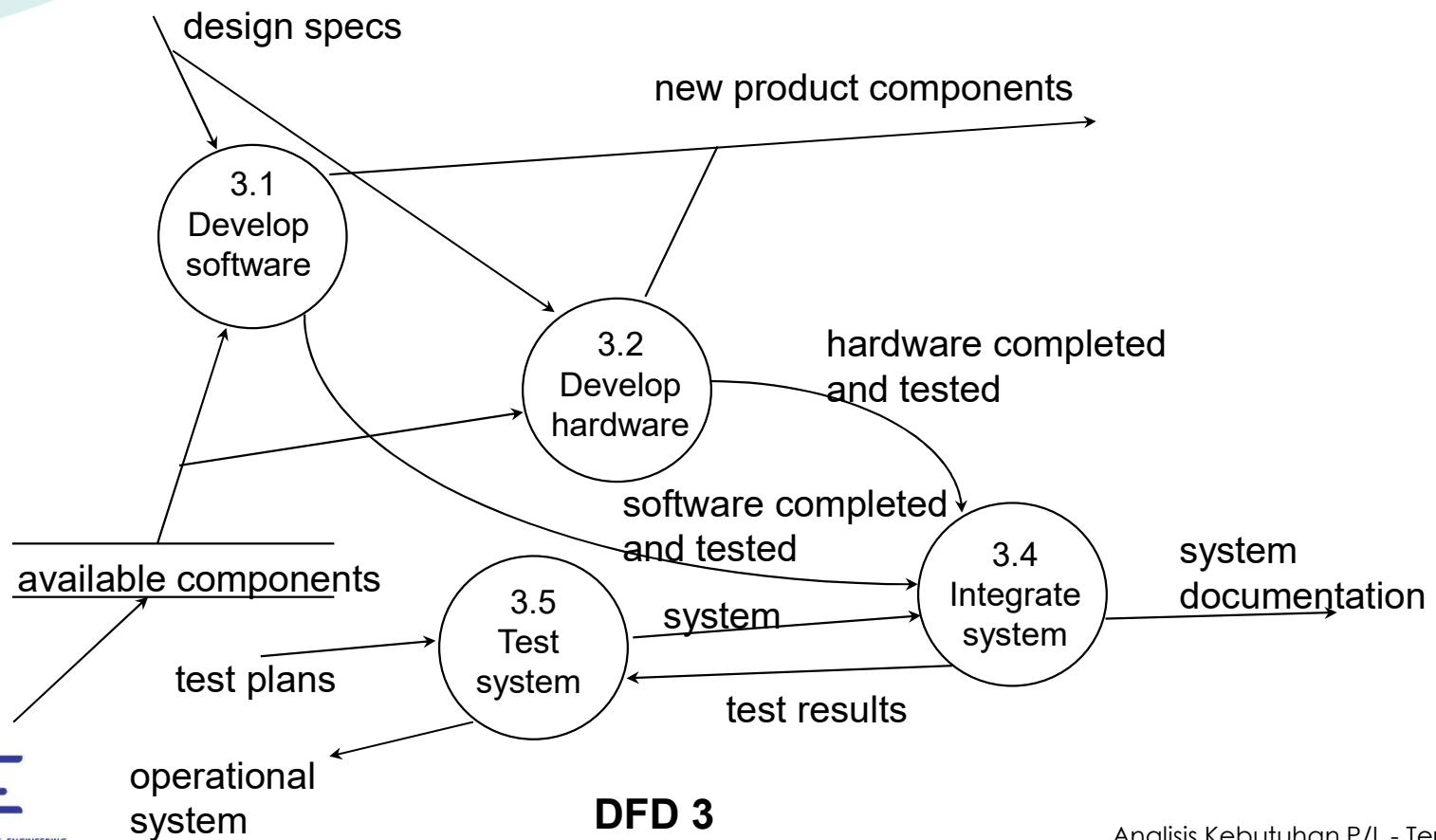


KNOWLEDGE & SOFTWARE ENGINEERING

**DFD 2**

Analisis Kebutuhan P/L - Terstruktur

# *Latihan: Cari Salah (8)*



# *Buat DFD: Resep Pancakes*

- Seorang pelanggan bisa memesan pancake apel atau original. Pesanan pelanggan akan di kirim ke chef (tukang masak). Chef hanya dapat melayani pesanan satu pancake setiap saat. Jika pesanan pancake sudah siap, maka chef akan meletakkan ke piring dan pancake sudah siap.
- Jika kulit pancake sudah habis, maka chef akan membuat yang baru.
- Resep pancake (untuk 12 potong)
  - 1 cangkir of tepung terigu
  - Garam secukupnya
  - 1 telur
  - 1  $\frac{1}{4}$  cangkir susu
  - Sedikit minyal buat memanaskan wajan
  - 1 potongan apel
- Campurkan tepung dan garam dalam mangkok besar, tambahkan telur dan susu
- Aduk hingga halus secara hati-hati
- Panaskan wajan, masukkan 3 sendok makan mentega ke wajan, dan goyangkan wajan hingga minyak tersebar merata.
- Untuk pancake apel, tambahkan potongan apel
- Jika warnanya sudah keemasan, angkat pinggiran pancake dengan pegangan yang sesuai dan balik hingga matang sisi sebelahnya.



Tim Pengajar IF2250

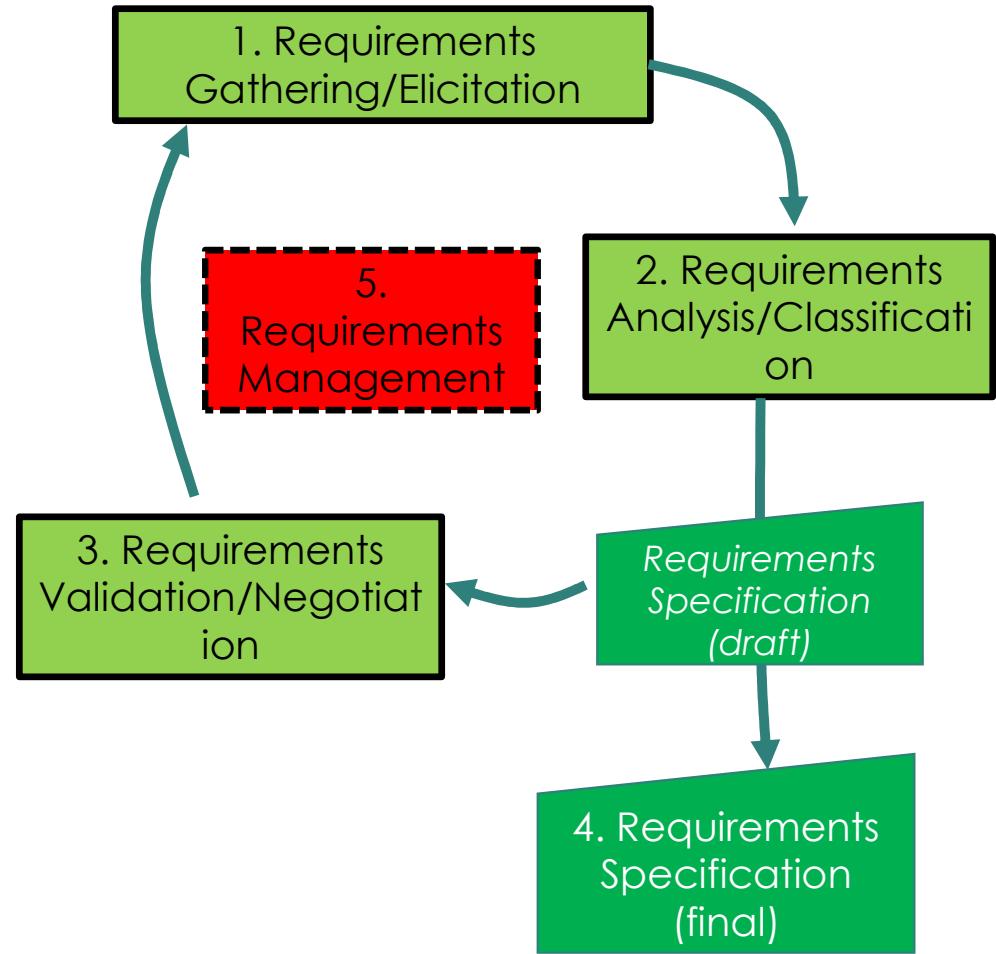
IF2250 – Rekayasa Perangkat Lunak  
**Penulisan Spesifikasi Kebutuhan  
Perangkat Lunak**

SEMESTER II TAHUN AJARAN 2022/2023



KNOWLEDGE & SOFTWARE ENGINEERING





# Berbagai Karakteristik Pengguna (user)

- Pengguna yang memiliki suatu hak akses tertentu
  - Tamu, pengguna anggota organisasi, petugas pelayanan masyarakat, administrator jaringan, administrator sistem
- Pengguna yang hanya mengerti satu bagian operasional, sehingga kadang tidak mengerti secara lebih umum tentang sistem yang akan dikembangkan
  - Contoh: petugas loket bagian peminjaman buku, hanya mengerti bagian peminjaman buku saja, tetapi tidak tahu bahwa ada masalah pembuatan laporan pada software yang akan dibuat
- Pengguna yang hanya mengerti satu sistem
  - Hanya mengerti windows, hanya tahu menggunakan MS Word, hanya tahu 'facebook', hanya tahu menggunakan aplikasi statistik
- Pengguna yang terbiasa berkomunikasi dengan bahasa ibunya
- Pengguna yang secara langsung atau tidak langsung berhubungan dengan pelanggan
  - Contoh: pengguna yang berperan sebagai manajer akan berbeda perlakunya dengan pengguna yang berperan sebagai penghubung dengan pelanggan
- Pengguna yang sibuk atau yang tidak sibuk (sulit ditemui)
- Pengguna juga mungkin memiliki karakter yang berbeda-beda
  - Pengguna yang sulit atau lambat dalam mengungkapkan ide,
  - Pengguna yang 'sok tahu'
  - Pengguna yang suka emosional
  - Dan lain-lain

Sistem analis harus 'sabar', mencoba mengerti, tidak mudah emosional, dll, agar tujuan pekerjaannya dapat dilaksanakan, yaitu **mendapatkan 'kebutuhan pengguna'**



## **Dengan Karakteristik tersebut, akibatnya pernyataan kebutuhan\* menjadi:**

- Terlalu umum, tidak rinci, tidak sederhana, terlalu ‘mengawang-awang’ atau bertele-tele
- Terlalu rinci, sehingga tidak menggambarkan sistem secara keseluruhan
- Terjadi konflik antara bagian yang berbeda dalam satu sistem
  - Perbedaan aturan, perbedaan tanggungjawab
- Tidak konsisten
  - *Hari ini bilang X, minggu depannya ingin Y*
- Mungkin kebutuhannya tidak realistik
  - Diluar anggaran, akan terlalu lama waktu pengembangannya

\* Pernyataan Kebutuhan = Requirements statement

# *Spesifikasi Kebutuhan PL dibuat berdasarkan berbagai Kebutuhan Sistem*



# *Penulisan Spesifikasi Kebutuhan Perangkat Lunak (SKPL)*



KNOWLEDGE & SOFTWARE ENGINEERING



# *Spesifikasi Kebutuhan*

- Membuat spesifikasi kebutuhan artinya
  - Proses penulisan kembali kebutuhan pengguna (user) dan sistem menjadi dokumen spesifikasi kebutuhan perangkat lunak
- Spesifikasi Kebutuhan akan menjadi **KONTRAK** bagi Pengguna juga Pengembang
  - Pengguna dan pelanggan (customer) harus dapat mengerti
    - Pengguna umumnya tidak memiliki latar belakang teknis
  - Pengembang system harus dapat mengerti untuk membuat Perancangan dan Implementasi Program

# *Penulisan Spesifikasi Kebutuhan PL (SKPL)*

Spesifikasi kebutuhan dapat diungkapkan dengan cara berikut:

- Bahasa sehari-hari
- Penulisan yang lebih terstruktur
- Bahasa yang mirip dengan deskripsi perancangan
  - Algoritma
- Notasi Grafis
- Notasi matematis (metode formal)

**Mudah  
dituliskan**



**Sulit  
dituliskan**



KNOWLEDGE & SOFTWARE ENGINEERING

# Contoh:

## Bahasa Sehari-hari

Seorang diijinkan untuk meminjam dokumen

## Bahasa Terstruktur

Siapa: Seorang

Aksi: diijinkan untuk meminjam

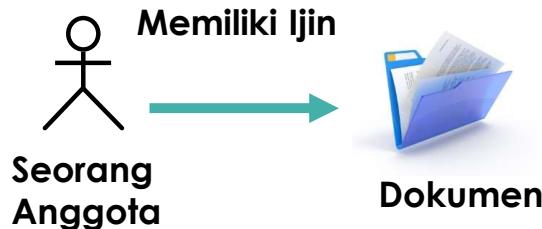
Objek: dokumen

## Bahasa Deskripsi (mirip algoritma)

Jika (seseorang punya hak) maka  
akan diijinkan untuk meminjam  
dokumen

## Metode Formal (Notasi Z)

### Notasi Grafis



CheckOut

$\Delta$  Documents  
p?: PERSON  
d?: DOCUMENT

$d? \notin \text{dom checked\_out}$   
 $(d?, p?) \in \text{permission}$   
 $\text{checked\_out}' = \text{checked\_out} \cup \{(d?, p?)\}$



KNOWLEDGE & SOFTWARE ENGINEERING

# ***Penulisan SKPL yang baik***

1. Jelas/Lengkap (Clear, Complete, Concise)
2. Konsisten (Consistent)
3. Tepat (Correct)
4. Mudah di ubah (Modifiable)
5. Terurut (Ranked)
6. Dapat diuji (Testable)
7. Dapat ditelusuri (Traceable)
8. Tidak Ambigu (Unambiguous)

## ***I. Jelas/Lengkap (Clear, Complete, Concise)***

- Sudah mencakup semua aspek yang terkait pada situasi dunia nyatanya
- Hanya ‘satu’ kebutuhan (tidak tercampur aduk), dan tidak terduplikasi
- Mudah dibaca dan tidak meragukan
  - Tidak menggunakan istilah yang tidak jelas atau tidak ada penjelasannya atau arti yang sama menggunakan istilah yang berbeda-beda
  - Tidak menuliskan situasi yang tidak akan ditemui atau fitur yang tidak perlu



KNOWLEDGE & SOFTWARE ENGINEERING

## 2. Konsisten (*Consistent*)

- Tidak ada kebutuhan yang konflik antara setiap kebutuhan
  - Misalnya konflik bisa terjadi karena ada suatu perilaku yang diharapkan terjadi tetapi mungkin menyebabkan perilaku lain menjadi tidak terjadi
  - Contoh:
    - Pegawai di bagian peminjaman: Peminjaman buku dapat dipinjam untuk 10 buku/anggota
    - Pegawai di bagian pengembalian: Sistem membolehkan pengembalian maksimum 5 buku

### 3. Benar (Correct)

- Spesifikasi kebutuhan yang **benar**, harus **akurat** dan **presisi** dalam mengidentifikasi setiap situasi dan **keterbatasannya**, termasuk suatu kemampuan yang mungkin dihadapi, dan termasuk mendefinisikan respons yang terkait dengan kemampuan tadi
  - Spesifikasi harus mendefinisikan lingkungan operasional pada dunia nyata, bagaimana interface dan interaksinya dengan lingkungan
- Kebutuhan mungkin **benar** di **satu masa**, tetapi mungkin menjadi **tidak benar** di waktu yang **berbeda**, atau dengan kata lain kebutuhan pada dunia nyata sering di selaraskan menjadi **spesifikasi yang benar**.
  - User membutuhkan perangkat lunak yang dapat diakses setiap saat
    - Pada jaman **sebelum** era internet, kebutuhan ini mungkin perlu diperjelas apakah akses setiap saat ini hanya **diakses** di semua **cabang perusahaan**?
    - Di **era internet**, maka kebutuhan ini dapat diterjemahkan sebagai pengembangan aplikasi **berbasis web** dengan menggunakan **client browser** untuk aksesnya
    - Di **era mobile**, maka kebutuhan ini dapat diterjemahkan sebagai pengembangan aplikasi berbasis mobile dengan karakteristik ‘mobile’ yang sifatnya **mudah dipindah-pindahkan** dan **tetap terkoneksi** internet



KNOWLEDGE & SOFTWARE ENGINEERING

## 4. Mudah diubah (*Modifiable*)

- Kebutuhan dari pengguna relatif mudah berubah, sehingga struktur dari spesifikasi tadi harus dibuat **pengelompokan** atau **strukturisasi** yang rapi
  - Perlu pengelompokan struktur yang **logis**
  - **Tetapi**, pengelompokkan ini dapat menyebabkan terjadinya **konflik** antar **spesifikasi**, atau urutan skala prioritas dalam kelompok yang berbeda

## 5. Terurut (*Ranked*)

- Spesifikasi dapat **diurutkan** sesuai dengan **prioritas**, atau **kepentingan** atau Stabilitas dari suatu kebutuhan
  - Urutan ini dapat dibentuk dalam **struktur dokumen**
- Makin **tinggi kompleksitas** masalah, maka makin **sulit pengurutan** spesifikasi ini,
  - Tetapi hal ini juga menunjukkan akan pentingnya pengurutan spesifikasi ini dilakukan dengan benar
  - **Pengurutan** spesifikasi kebutuhan ini dapat mempengaruhi **pengembangan**.
    - Kesalahan **urutan pekerjaan** pengembangan spesifikasi kebutuhan dapat mengakibatkan pengembangan yang **tidak efisien**



KNOWLEDGE & SOFTWARE ENGINEERING

## 5. Terurut (*Ranked*) (2)

- Contoh:
  - Pembangunan sistem perpustakaan melibatkan beberapa SKPL, contoh:
    - “Perangkat Lunak memiliki fungsi peminjaman buku oleh anggota” (SKPL01)
    - “Perangkat Lunak memiliki fungsi pengembalian buku oleh anggota” (SKPL02)
    - “Perangkat lunak memiliki fungsi layanan pendaftaran anggota baru” (SKPL03)
    - “Perangkat lunak memiliki fungsi layanan pendataan buku baru” (SKPL04)
  - Keempat SKPL tersebut, dapat diurutkan berdasarkan:
    - Fungsi utamanya: SKPL01, SKPL02, SKPL03, SKPL04
    - Susunan urutan langkah: SKPL03, SKPL04, SKPL01, SKPL02
      - Buku dan anggota baru dapat melakukan transaksi peminjaman setelah data buku/anggota sudah tersedia



KNOWLEDGE & SOFTWARE ENGINEERING

## 6. Dapat diujikan (*Testable*)

- Spesifikasi kebutuhan harus dapat **diujikan**, atau artinya dapat **dibuktikan** atau dinilai secara **kuantitatif**
  - Kebutuhan bahwa “Sistem harus mudah digunakan” sifatnya sangat subjektif, jadi kebutuhan semacam ini **tidak dapat diujikan**
    - Pada **SKPL** kita harus **mendefinisikan kembali** apa itu ‘sistem yang mudah digunakan’ bagi pengguna, pada beberapa kasus dapat dituliskan dengan.
    - Contoh Spesifikasi Kebutuhan yang **baik**:
      - “Akan dilakukan pelatihan 5 hari sebelum operasional” (dengan harapan pengguna akan mudah menggunakan sistem baru setelah dilakukan pelatihan)
      - “Sistem menggunakan sistem operasi Windows” (dengan harapan pengguna yang sudah terbiasa menggunakan Windows akan tidak sulit menjalankan aplikasi yang dikembangkan dengan Windows)
      - “Sistem memiliki fasilitas Bantuan/Panduan” (dengan harapan pengguna yang kesulitan akan dapat menggunakan fasilitas ini).



KNOWLEDGE & SOFTWARE ENGINEERING

## 7. Dapat ditelusuri (*Traceable*)

- Setiap Spesifikasi kebutuhan yang ditulis pada dokumen harus diberikan **nomor identifikasi** yang **unik**
  - Agar mudah **ditelusuri**, juga mudah diucapkan
- Penulisan nomor identifikasi ini harus **konsisten** dan dengan struktur yang **logis**
  - Contoh:
    - SRS001: “PL memiliki fungsi pencatatan anggota baru”
    - SRS002: “PL memiliki fungsi peminjaman buku”
    - SRS003: “PL memiliki fungsi pengembalian buku”

## 8. Tidak Ambigu (Unambiguous)

- Spesifikasi kebutuhan ini harus **tidak ambigu**
  - Tidak ada interpretasi **ganda**
- Ketidakambiguan ini relatif paling **sulit** jika menggunakan **Bahasa sehari-hari** (Bahasa natural)
- Penggunaan **ungkapan** yang **tidak tegas** atau **struktur** yang **jelek** dapat menyebabkan ambiguitas atau salah mengerti/interpretasi bagi yang membaca.
  - Hati-hati menggunakan kata hubung "dan", "atau" , "seperti"
    - Sistem dapat melakukan X dan Y
      - Apakah artinya: X dan Y harus dilakukan **bersamaan**?
      - Atau artinya ada **dua kebutuhan**:
        - Sistem dapat melakukan X
        - Sistem dapat melakukan Y



KNOWLEDGE & SOFTWARE ENGINEERING

# Lain-lain (I)

- Hindari kata-kata seperti “kecuali”, “tetapi” atau “jika diperlukan”
  - Contoh:
    - *Perangkat lunak memiliki fungsi pembayaran denda jika diperlukan*
- Satu spesifikasi harus ditulis **lengkap** tapi **atomik** (tidak mengandung spesifikasi lain)
  - Hati-hati menuliskan spesifikasi yang intinya tersebar di beberapa spesifikasi
- Hindari suatu spesifikasi dengan kata-kata semacam buzzword
  - Contoh: system mampu melayani **Big Data** (apa ini?)

# Lain-lain (2)

- Setiap kebutuhan harus ditulis sebagai:
  - **Subjek – Predikat (- objek)**
    - Subjek: Nama identifikasi dari sistem
    - Predikat: melakukan suatu aksi, hasil yang diinginkan
- Hindari penulisan **singkatan**, apalagi yang dapat menimbulkan ambiguitas
  - Contoh:
    - “PL melakukan penyimpanan data, pencetakan, dan lain-lain”
    - “PL melakukan proses seperti di word”
- Hindari penggunaan kata jumlah yang **tidak terukur** atau tidak jelas
  - Contoh: “kira-kira”, “mendekati jumlah”, “efek minimal”, “mungkin”, “sebaiknya”
  - “PL memiliki fungsi penghitungan yang kira-kira tidak lebih dari 100ribu data”

## *Lain-lain (3)*

- Hindari kata-kata yang **mengambang/tidak perlu**
- Hindari istilah yang **berbeda** untuk **mengacu** pada ‘benda’ atau sesuatu yang **sama**
- Hindari **spekulasi harapan sistem** yang tidak perlu
  - Jangan menuliskan yang mungkin tidak akan tercapai
- Hindari penulisan **kebutuhan** yang **belum jelas**
- Gunakan kalimat ‘**positif**’ dan bukan ‘negative’
  - Contoh: “PL dapat melakukan <sesuatu>”
  - Contoh salah: “PL tidak dapat melakukan <sesuatu>”



KNOWLEDGE & SOFTWARE ENGINEERING



# Lain-lain (4)

- Rules of thumbs, dalam Bahasa Indonesia gunakan kalimat
  - <**Perangkat Lunak/Sistem/Nama**> <kata kerja> <obyek>
  - Perhatikan **konsistensi** penggunaan “**nama**” pada **Subyek**
  - Contoh:
    - PL memiliki layanan pendaftaran anggota baru
    - PL memiliki layanan pembelian barang
    - PL memberikan fungsi pembayaran barang yang sudah dipesan
    - PL melakukan permintaan Informasi login/password untuk setiap transaksi pembelian dan penjualan
  - Contoh lain:
    - Sistem Ecommerce memiliki layanan pendaftaran anggota baru
    - Sistem Ecommerce memiliki layanan pembelian barang

## Lain-lain (5)

- Requirement dalam Bahasa Inggris:
  - **Shall** digunakan di mana **persyaratan** (requirement) sedang **dinyatakan**
  - **Will** harus digunakan untuk mewakili **pernyataan fakta**
  - **Should** mewakili **tujuan** yang ingin **dicapai**



KNOWLEDGE & SOFTWARE ENGINEERING

# ***Kebutuhan Pengguna vs. SKPL***

<b>Kebutuhan Pengguna</b>	<b>SKPL</b>
Software harus 'user friendly' (bagaimana mengukurnya?)	PL memiliki rancangan yang menggunakan menu, dialog box,, dropdown menu.
Semua tampilan di layar harus terlihat di monitor dengan cepat (seberapa cepat?)	PL dapat menampilkan hasil dalam kurang dari 2 detik saat pengguna mengakses suatu fungsi
Jika software hang, maka sistem akan dapat di restart secepatnya	PL dapat di-restart dalam waktu kurang dari 30 menit jika terjadi hang
Upgrade terhadap sistem dapat dilakukan tanpa sama sekali mengganggu sistem produksi yang berjalan 24 jam (harapan yang tidak realistik)	PL tidak menyebabkan sistem produksi berhenti lebih dari 2 hari jika terjadi upgrade terhadap sistem.
Kalau user gagal login tiga kali, maka program javascript harus di jalankan untuk mengunci user dari sistem (javascript program adalah aspek implementasi)	PL akan mengunci user yang gagal login 3 kali.



# *Kesalahan Umum*

- Jangan membuat **asumsi** yang ‘Jelek’
- Jangan menuliskan ‘HOW’, tapi seharusnya ‘WHAT’
  - **Jangan** menuliskan **penjelasan operasinya**
  - **Jangan** terlalu **rinci** menuliskan kebutuhannya, cenderung **bias** menjadi penjelasan HOW-nya.
- Contoh:
  - Kalau user gagal login tiga kali, maka program javascript harus di jalankan untuk mengunci user dari system
    - javascript program adalah aspek implementasi

# Kesalahan Umum

- Jangan menggunakan **istilah** yang **salah**
- Jangan menggunakan **struktur** kalimat yang **tidak tepat**
  - Grammar yang jelek
- Jangan ada **kebutuhan** yang ‘**hilang**’
  - Mungkin **terlupakan** saat wawancara, atau **tidak tercatat** dalam notulen hasil dari suatu rapat/ pertemuan



KNOWLEDGE & SOFTWARE ENGINEERING

# *Jangan membuat Asumsi yang Jelek'*

- Biasanya terjadi karena analis **tidak punya informasi** yang cukup
  - Informasi dapat diperjelas dengan menanyakan lebih lanjut
  - Walaupun sering pengguna/customer mempercayai pendapat kita, tetapi mereka biasanya lebih tahu apakah asumsi yang kita gunakan sudah cukup atau berlebihan atau masih kurang
- Asumsi ini sering terkait dengan:
  - Anggaran/Biaya: Anggaran/biaya yang terlalu sedikit menyebabkan suatu solusi disederhanakan, sehingga asumsi kadang diperlukan, tentunya harus di konfirmasi dulu ke calon pengguna/customer
- Asumsi ini harus jelas untuk pengguna/customer, juga jelas untuk semua anggota tim pengembang
  - Selalu di review dengan kebutuhan yang terkait

# *Jangan menuliskan 'HOW'; tapi seharusnya 'WHAT'*

- Kenapa?
  - Untuk menghindari pemaksaan suatu solusi 'Desain/Perancangan'
    - Bila suatu kebutuhan sudah berisi solusi rancangan, maka ada kemungkinan solusi ini tidak/kurang memperhatikan kebutuhan yang lain
      - Solusi harus perancangan harus memperhatikan semua kebutuhan lain
    - Solusi 'bagaimana' pada suatu kebutuhan menyebabkan rancangan menjadi tidak fleksibel
      - Sering terjadi pengguna yang tidak pengalaman menyatakan suatu solusi rancangan padahal yang kita butuhkan adalah 'APA' yang dia butuhkan.
        - Ada kemungkinan solusi rancangannya hanya bersifat sementara atau hanya memandang dari sudut pandang teknologi tertentu
  - Jadi yang ditulis adalah 'Requirements'/Kebutuhan bukan 'Operasi'
    - Sulit memverifikasi suatu kebutuhan yang dinyatakan dalam suatu 'operasi'

# ***Jangan menggunakan istilah yang salah (I)***

- Dalam bahasa Inggris dibedakan istilah:
  - Shall: sesuatu yang harus dapat di verifikasi, biasanya requirements menggunakan 'shall'
  - Will: menyatakan suatu fakta
  - Should: menyatakan suatu obyektif/tujuan
  - Kata 'are', 'is', 'was', 'must' tidak digunakan pada requirements
    - Kata ini dapat digunakan sebagai penjelasan suatu kebutuhan
- Dalam Bahasa Indonesia, disarankan menggunakan:
  - Kata 'mampu' atau 'dapat' atau 'akan' untuk menggantikan 'shall'
  - Kata 'memiliki' untuk menggantikan 'Will'
  - Kata 'mampu' atau 'dapat' atau 'akan' untuk menggantikan 'should'

## *Jangan menggunakan istilah yang salah (2)*

- **Hindari** penggunaan kata:
  - Support (mendukung)
  - But not limited to (tapi tidak dibatasi pada...)
  - Etc (dan lain-lain)
  - And/or (dan/atau)
- Mendukung..
  - Salah:
    - Sistem akan mendukung pengguna dalam memperbaiki data lama
  - Benar
    - Sistem akan menyediakan layar untuk memperbaiki data pengguna yang lama
- Istilah lain tidak jelas berapa batasnya atau tidak jelas apa jumlahnya, atau juga tidak jelas apakah ‘dan’ atau ‘atau’

# Struktur Kalimat (I)

- Pernyataan kebutuhan harus **mudah dibaca** dan **dimengerti**.
- Dimulai dengan APA yang dapat/mampu dilakukan oleh sistem/perangkat lunak
  - Bukan 'HOW'
- Bentuk umum (panduan)
  - <sistem> <akan | dapat | mampu> memberikan/melakukan <suatu aksi>
  - <sistem> dapat <kata kerja>
- Nama <sistem> dapat diganti dengan <perangkat lunak> atau suatu nama sistem atau nama perangkat lunak
  - Contoh:
    - PL dapat menyimpan data perubahan
    - Sistem PL mampu menyimpan data perubahan
    - Aplikasi mampu menyimpan data perubahan
    - Sistem Perpustakaan mampu menyimpan data perubahan

# Struktur Kalimat (2)

- Penulisan dengan penggunaan daftar (*list*) sebaiknya dihindari, Karena setiap item kebutuhan harus diverifikasi, kecuali memang dalam daftar tersebut mungkin dapat diverifikasi dengan satu cara saja
- Contoh
  - Sistem dapat melakukan:
    - Mencatat peminjaman buku
    - Mencatat anggota perpustakaan baru
  - Sebaiknya ditulis sebagai:
    - Sistem dapat mencatat peminjaman buku
    - Sistem dapat mencatat anggota perpustakaan baru

# *Struktur Kalimat: Subjek*

- Hindari penggunaan subjek yang salah
  - Contoh salah
    - Database akan melakukan penyimpanan data perubahan
  - Perhatikan bahwa walaupun memang Database yang faktanya akan melakukan proses penyimpanan, tetapi gunakan Nama Sistem/ perangkat lunak.
  - Contoh:
    - Sistem akan melakukan penyimpanan data perubahan

# *Kalimat yang terlalu panjang*

- Kebutuhan harus dituliskan dengan kalimat yang sifatnya '**atomik**' atau tidak terlalu panjang atau terlalu kompleks
  - Mungkin saja sebenarnya kebutuhan yang **panjang** tadi seharusnya ditulis sebagai **lebih dari satu** kebutuhan
- Contoh salah:
  - Sistem perpustakaan dapat memberikan fasilitas peminjaman dan pengembalian buku khususnya untuk anggota yang sudah mendaftar
- Contoh benar:
  - Sistem perpustakaan dapat memberikan layanan peminjaman buku untuk anggota terdaftar
  - Sistem perpustakaan dapat memberikan layanan pengembalian buku untuk anggota terdaftar.

# *Hindari Kata yang Ambigu*

- Contoh:
  - Minimisasi, maksimisasi, cepat, user-friendly (mudah digunakan), cukup mudah, secukupnya
  - “Seminimum/semaksimum mungkin” harus jelas seberapa minimum atau maksimum

# *Kebutuhan atau objektif*

- Kadang kita sulit mendefinisikan apa yang dibutuhkan
  - Maka yang harus kita tulis adalah obyektifnya atau tujuannya
  - Jadi pernyataannya akan ditulis sebagai obyektif/tujuan dan bukan kebutuhan
- Struktur kalimat:
  - <sistem> <diharapkan | sebaiknya> <dapat | mampu>  
<melakukan suatu aksi>
  - Dalam Bahasa Inggris digunakan istilah ‘Should’ dan bukan ‘shall’

# *Kalimat Kebutuhan bersifat Imperative (keharusan)*

- Menyatakan sesuatu yang ‘harus’ dibuat atau ‘harus’ ada pada perangkat lunak, walaupun demikian kata ‘harus’ perlu dihindari, kata ‘harus’ sebaiknya digunakan untuk menyatakan suatu performasi/kualitas
- Contoh:
  - PL mampu melakukan kalkulasi gaji pegawai, kalkulasi harus menghitung gaji seluruh karyawan



KNOWLEDGE & SOFTWARE ENGINEERING

# *Panduan Umum penulisan SKPL*

- Mengikuti **aturan tata tulis** (grammar)
- Dokumen harus bebas dari **salah ketik** (kurang huruf), salah penulisan ataupun penggunaan tanda baca yang salah
- Kebutuhan ini perlu dituliskan mengikuti **template** penulisan yang sama dalam satu organisasi
- Kebutuhan perlu dituliskan pada **bagian** yang sudah **disiapkan** (dari template).

# *Struktur Kalimat (yang disarankan)*

- [Lokalisasi] [Aktor/Owner] [aksi] [target | owned][constraint]
- Contoh:
  - Jika sistem terkena virus, sistem dapat beroperasi kembali kurang dari 2 hari.
  - [Lokalisasi] → Jika sistem kena virus
  - [Aktor/Owner] → Sistem
  - [Aksi] → beroperasi
  - [Target/Owned] →

# *Pemberian Contoh*

- Suatu contoh dapat digunakan untuk melengkapi pernyataan kebutuhan
- Contoh yang sama dapat digunakan jika diperlukan agar memperjelas masalah
- Harus jelas dituliskan '**Contoh**' jangan tertukar dengan 'spesifikasi kebutuhan'
  - Contohnya adalah .....

# Hindari ... (1)

- Hindari penggunaan kata “**harus**”
  - “Sistem harus melakukan pencetakan laporan”
    - Penulisan cukup dengan pernyataan:
      - “Sistem akan melakukan pencetakan laporan”
      - “Sistem memiliki fungsi pencetakan laporan”
    - Penggunaan kata harus memiliki sifat ‘wajib’ dilakukan, padahal pernyataan yang sederhana sudah menunjukkan sesuatu yang perlu/akan dilakukan oleh pengembang

# Hindari ... (2)

- Hindari kata ‘**sebaiknya**’
  - “sistem sebaiknya bisa mengkalkulasi data total penerimaan harian”
  - Seharusnya ditulis:
    - “Sistem akan melakukan kalkulasi data total penerimaan harian”
    - “Sistem memiliki fungsi kalkulasi data total penerimaan harian”
  - Penambahan kata “sebaiknya” membuat tidak jelas, apakah **bisa dilakukan** ataukah mungkin **tidak perlu dilakukan**.

# Hindari ... (3)

- Hindari penggunaan kata-kata yang mungkin berarti ganda atau tidak tegas.
  - Contoh: “**mendukung**”
    - “Sistem mendukung pencetakan laporan bulanan”
    - Kata mendukung, tidak secara langsung atau tidak tegas menyatakan apa yang akan ditunjukkan pada SKPL
      - Contoh yang benar: “Sistem akan mencetak laporan bulanan”



KNOWLEDGE & SOFTWARE ENGINEERING

# Hindari ... (4)

- Hindari kalimat yang memiliki arti tidak pasti. Misalnya kata “**mungkin**”
  - “Sistem dapat melakukan X, tapi tidak dibatasi hanya X, mungkin sistem dapat juga melakukan Y”
  - Sebaiknya: “Sistem akan melakukan X dan Y”
    - Kata mungkin juga perlu dihindari, karena tidak jelas status pernyataannya.
- Kata sambung “atau” juga bisa membingungkan, karena “atau” bisa berarti salah satu atau dua-duanya.
  - User: “sistem perlu melakukan operasi A atau operasi B”
  - Kalau memang dua-dua perlu dilakukan, maka harus dituliskan: “Sistem akan melakukan operasi A dan B”



KNOWLEDGE & SOFTWARE ENGINEERING

# Hindari ... (5)

- Contoh yang tidak pasti, adalah kalimat dengan kata: “... **dan lain-lain**”
  - Sistem akan mampu menyimpan laporan harian, bulanan, mingguan dan lain-lain”
  - Seharusnya: “Sistem akan menyimpan laporan harian, bulanan, mingguan dan tahunan”
- Perhatikan bahwa kita harus menuliskan dengan jelas dan lengkap apa saja yang diperlukan. Kata “dan-lain-lain” membuat interpretasi menjadi tidak jelas

# *Penulisan Referensi*

- Semua bagian yang merujuk ke suatu informasi lain, harus jelas dituliskan acuannya
  - Acuan dapat dituliskan sebagai kode
  - Contoh:
    - “Sistem dapat memiliki fungsi khusus yang didefinisikan di [DEF100]”
- Referensi juga harus jelas
  - Contoh:
    - [DEF100] “Data Management Company”, by Ali Budi, 2017.
- Acuan terhadap suatu bagian dari dokumen dapat dituliskan sebagai contoh berikut:
  - “Sistem menggunakan fungsi yang dijelaskan di bab 3, paragraph ke 4”

# *Penggunaan Tabel dan Gambar*

- Tiap tabel dan gambar diberikan nomor yang unik
- Daftar judul tabel dan gambar dicantumkan pada daftar isi
  - Agar mudah dicari
  - Berikan penjelasan untuk setiap tabel/gambar

Program Editor untuk dokumen memiliki fungsi ‘referensi’ yang dapat digunakan secara otomatis  
Contoh: ‘Reference’ di MS Word

- Penulisan Spesifikasi Kebutuhan Perangkat Lunak saat ini:
  - Belum ada panduan standard untuk Bahasa Indonesia
  - Aturan umumnya:

<NamaPerangkatLunak atau Sistem>  
<mampu | dapat | akan>  
<predikat>  
<obyek>

# *Pengembangan SKPL*

- Cari **fungsional** yang akan ada pada **perangkat lunak**
  - Tentunya juga jelas kenapa suatu fungsi itu diperlukan.
- Cari **proses-proses** apa saja yang akan ada nanti di aplikasi perangkat lunak ini
  - Pada bagian apa suatu proses akan digunakan, bagaimana dan apa efeknya.
- Jika proses dalam software ini nantinya berbeda dengan kenyataannya, maka kita harus berkonsultasi dengan user.  
Contoh:
  - Perubahan dalam kebijaksanaan baru organisasi kadang mengubah kebiasaan yang ada.
  - Pergantian staf dalam perusahaan

# *Analisis Kebutuhan*

- Melakukan pengkajian ulang (**review**) terhadap semua dokumen yang relevan
- Mengembangkan konsep awal
- Konsultansi dengan pengguna
  - Mendapatkan kejelasan
  - Negosiasi jika diperlukan, misalnya karena
    - Ada kebutuhan yang mungkin di luar anggaran
    - Butuh waktu yang lebih lama karena mungkin alasan teknis/non teknis
    - Ada kebutuhan di luar batasan yang pernah di bicarakan sebelumnya.
- Konsolidasi semua data masukan ke sistem
- Konsolidasi semua hasil keluaran dari sistem

# *Studi Kasus*

## *Mesin Jual Otomatis (Vending Machine)*



IF2250 Penulisan SKPL



KNOWLEDGE & SOFTWARE ENGINEERING



KNOWLEDGE & SOFTWARE E



) Penulisan SKPL

## ***Kebutuhan Pengguna untuk Vending Machine (VM)***

Suatu VM menjual barang secara otomatis. VM akan menampilkan barang dan harganya. Seorang pembeli akan memasukkan sejumlah koin dan kemudian memilih barang yang akan dibeli.

Transaksi pembelian hanya akan terjadi jika pengguna telah memasukkan koin yang benar (100, 200, 400, 500 dan 1000). Artinya jika benda lain dimasukkan selain koin yang benar, maka benda itu akan otomatis dikeluarkan.

Sistem yang dikembangkan harus berjalan di sistem Linux, dan software harus menyesuaikan dengan kebutuhan perangkat keras VM.

## ***Kebutuhan Sistem untuk VM (I)***

- Mesin memiliki mekanisme untuk memeriksa apakah objek yang dimasukkan adalah koin yang benar dengan melakukan validasi terhadap ukuran, berat, ketebalan dan juga sisi logamnya.
- Mesin menerima koin 100, 200, 500 dan 1000 rupiah. Selain itu akan dianggap sebagai objek yang tidak valid.
- Mesin dapat melakukan proses perhitungan pembayaran dan juga pemilihan produk, setelah koin yang benar di deteksi.
- Mesin akan menerima masukan jenis barang yang akan dibeli oleh pembeli
- Mesin akan memeriksa apakah suatu barang yang dipilih tersedia, jika tidak ada maka koin akan dikembalikan secara otomatis, dan akan ada pemberitahuan ke pembeli.

## ***Kebutuhan Sistem untuk VM (2)***

- Jika pembeli tidak melakukan pemilihan barang (ingin dibatalkan), maka koin akan dikembalikan oleh mesin
- Mesin akan mengeluarkan barang jika memang masih tersedia dan jumlahnya koinnya sudah cukup
- Mesin akan memberikan uang kembali jika uang yang dimasukkan lebih dari harga barang.
- Tombol pemilihan barang di non-aktifkan jika suatu barang sudah dikeluarkan dan selama jumlah koin yang cukup sudah dimasukkan
- Mesin bisa menerima pergantian jenis barang oleh pemilik mesin. Dengan demikian harga dari produk yang baru harus dapat diperbaiki.

# **Spesifikasi Kebutuhan PL untuk VM (I)**

- Sistem VM menjual barang secara otomatis
- Sistem VM menampilkan barang dan harganya
- Pembeli memasukkan koin
- Pembeli memilih barang
- Sistem VM berjalan di system operasi LINUX
- Sistem VM memeriksa validasi koin dengan ukuran, berat, ketebalan, sisi logam
- Sistem VM hanya menerima koin 100, 200, 500, dan 1000
- Sistem VM akan menolak masukan koin yang tidak sah
- Sistem VM otomatis akan mengeluarkan koin yang tidak sah
- Sistem VM akan mengembalikan koin jika pembeli batal melakukan pemilihan barang
- Sistem VM melakukan kalkulasi penghitungan pembayaran
- Sistem VM dapat memberikan kembalian jika pembayaran berlebih atau jika barang ternyata tidak tersedia
- Pemilik dapat mengganti jenis barang, VM akan mengupdate harga produk

## **Spesifikasi Kebutuhan PL untuk VM (2)**

- Sistem VM menjual barang secara otomatis
- Sistem VM menampilkan barang dan harganya
- Pembeli memasukkan koin
- Pembeli memilih barang
- Sistem VM berjalan di system operasi LINUX
- Sistem VM memeriksa validasi koin dengan ukuran, berat, ketebalan, sisi logam
- Sistem VM hanya menerima koin 100, 200, 500, dan 1000
- Sistem VM akan menolak masukan koin yang tidak sah
- Sistem VM otomatis akan mengeluarkan koin yang tidak sah
- Sistem VM akan mengembalikan koin jika pembeli batal melakukan pemilihan barang
- Sistem VM melakukan kalkulasi penghitungan pembayaran
- Sistem VM dapat memberikan kembalian jika pembayaran berlebih atau jika barang ternyata tidak tersedia
- Pemilik dapat mengganti jenis barang, VM akan mengupdate harga produk



KNOWLEDGE & SOFTWARE ENGINEERING

## ***Spesifikasi Kebutuhan PL untuk VM (3)***

- Sistem VM menjual barang secara otomatis
- Sistem VM menampilkan barang dan harganya **F**
- Pembeli memasukkan koin
- Pembeli memilih barang
- Sistem VM berjalan di system operasi LINUX **NF**
- Sistem VM memeriksa validasi koin dengan ukuran, berat, ketebalan, sisi logam
- Sistem VM hanya menerima koin 100, 200, 500, dan 1000
- Sistem VM akan menolak masukan koin yang tidak sah
- Sistem VM otomatis akan mengeluarkan koin yang tidak sah
- Sistem VM akan mengembalikan koin jika pembeli batal melakukan pemilihan barang **F**
- Sistem VM melakukan kalkulasi penghitungan pembayaran **F**
- Sistem VM dapat memberikan kembalian jika pembayaran berlebih atau jika barang ternyata tidak tersedia **F**
- Pemilik dapat mengganti jenis barang, VM akan mengupdate harga produk **F**

# *Spesifikasi Kebutuhan PL*

- Kebutuhan Fungsional
  - Sistem VM dapat menerima masukan koin
  - Sistem VM dapat menerima masukan jenis barang
  - Sistem VM menampilkan barang dan harganya
  - Sistem VM melakukan kalkulasi penghitungan pembayaran
  - Sistem VM dapat memberikan kembalian jika pembayaran berlebih atau jika barang ternyata tidak tersedia
  - Sistem VM dapat menerima update barang dan harga dari pemilik mesin
- Kebutuhan Non Fungsional
  - Sistem VM berjalan di system operasi LINUX

# *Spesifikasi Kebutuhan PL (Berikan Kode)*

- Kebutuhan Fungsional

- VM01: Sistem VM dapat menerima masukan koin
- VM02: Sistem VM dapat menerima masukan jenis barang
- VM03: Sistem VM menampilkan barang dan harganya
- VM04: Sistem VM melakukan kalkulasi penghitungan pembayaran
- VM05: Sistem VM dapat memberikan kembalian jika pembayaran berlebih atau jika barang ternyata tidak tersedia
- VM06: Sistem VM dapat menerima update barang dan harga dari pemilik mesin

- Kebutuhan Non Fungsional

- VM07: Sistem VM berjalan di system operasi LINUX

*Cara pengkodean tidak ada standar,  
yang penting kode digunakan konsisten  
dan mudah digunakan*

# *Dokumentasi Spesifikasi Kebutuhan Perangkat Lunak (SKPL)*



62



IF2250 Penulisan SKPL

# Pendahuluan

- Hasil analisa kebutuhan itu nanti harus didokumentasikan dalam dokumen yang sudah standard
  - Dengan standard, maka komunikasi akan lebih lancar
- Dokumen kebutuhan ini sering disebut sebagai **SRS** (Software Requirement Specification) atau **SKPL** (Spesifikasi Kebutuhan Perangkat Lunak)

# **Dokumen Kebutuhan (Requirements Document)**

- Dokumen kebutuhan ini berisi pernyataan resmi tentang apa yang harus dikerjakan oleh pengembang
- Dokumen ini sebaiknya mengikutsertakan definisi dari kebutuhan pengguna (user requirements) dan juga spesifikasi dari kebutuhan sistem (system requirements).
- Dokumen ini adalah BUKAN dokumen perancangan
  - Dokumen ini harus berisi ‘APA’ yang dilakukan oleh sistem dan seminimal mungkin tentang ‘BAGAIMANA’ mengerjakannya.

# *Siapa pengguna dalam dokumen kebutuhan?*

Peran	Apa guna dokumen?
Pengguna Sistem	Pengguna sistem memberikan spesifikasi kebutuhan dan membaca dokumen untuk memeriksa apakah sudah sesuai dengan keinginannya. Keinginan mereka bisa merubah isi dokumen kebutuhan ini
Manajer	Dokumen ini kadang digunakan untuk bahan mengajukan tender. Pada situasi lain, manajer menggunakan dokumen ini untuk melakukan perencanaan pengembangan.
System Engineers	Dokumen digunakan untuk mengerti apa yang harus dikembangkan.
System Test Engineers	Dokumen digunakan untuk membuat validasi dari setiap kebutuhan
System Maintenance Engineers	Dokumen digunakan untuk membantu mengerti sistem dan keterhubungan antar bagian dalam sistem.



KNOWLEDGE &amp; SOFTWARE ENGINEERING

\* *Software Engineering 7<sup>th</sup> ed, Ian Sommerville*

IF2250 Penulisan SKPL



# ***Struktur Umum Dokumen Kebutuhan***

- Preface
- Introduction
- Glossary
- User requirements definition
- System architecture
- System requirements specification
- System models
- System evolution
- Appendices
- Index

# Struktur Dokumen Kebutuhan (I)

Chapter	Deskripsi
Preface	Berisi apa yang akan dibaca dari dokumen. Termasuk juga berisi penjelasan sejarah versi dari dokumen, termasuk kenapa versi terakhir dikembangkan.
Introduction	Berisi penjelasan mengapa sistem ini dibutuhkan. Bagian ini secara umum menjelaskan fungsi utama dari sistem dan menjelaskan keterhubungannya dengan sistem lain. Bagian ini juga menjelaskan keberadaan sistem ini dalam konteks bisnis perusahaan secara umum ataupun tujuan strategis mengapa perusahaan memerlukan sistem ini.
Glossary	Bagaimana ini menjelaskan istilah teknis dalam dokumen. Bagian ini mungkin dibaca oleh orang awam, jadi jangan diasumsikan pembaca adalah orang yang memang sudah ahli atau berpengalaman.
User requirements definition	Layanan sistem bagi user dijelaskan di bagian ini. Kebutuhan NF juga di nyatakan di bagian ini. Deskripsi ini dapat menggunakan bahasa natural, diagram atau notasi lain yang dapat dimengerti pengguna. Standard produk dan proses dapat dispesifikasikan juga.
System architecture	Bagian ini berisi pandangan arsitektur sistem secara umum. Caranya dengan menunjukkan fungsi-fungsi yang tersebar ke modul sistem lain. Bagian komponen arsitektur yang di gunakan lagi dapat di tekankan lagi



# Struktur Dokumen Kebutuhan (2)

Chapter	Description
System requirements specification	Bagian ini menjelaskan kebutuhan fungsional dan NF secara lebih rinci. Bahkan interface ke sistem lain juga dapat dijelaskan dibagian ini.
System models	Bagian ini mungkin menggunakan model grafik untuk menunjukkan keterhubungan antar komponen sistem dan juga sistem dengan lingkungannya. Contohnya: Model Objek, Model data-flow atau model Data Semantik.
System evolution	Bagian ini menjelaskan asumsi dasar dari sistem yang akan dikembangkan, dan antisipasi perubahan karena evolusi perangkat keras, dan perubahan kebutuhan pengguna. Bagian ini berguna bagi perancangan sistem agar keputusan perancangannya tidak dibatasi oleh perubahan sistem di masa depan.
Appendices	Bagian ini berisi informasi rinci/spesifik yang terkait dengan aplikasi yang sedang dikembangkan. Misalnya deskripsi hardware dan basisdata.
Index	Index dari dokumen.



## ***Dokumen SKPL memiliki standard yang berbeda-beda***

- IEEE SRS std 830-1998
- BS 6719:1986
- ESA Space Agency Standards
- US DoD Std 7935A
- NASA standard
- Canadian Standard (Z242.15.4-1979)
- dll

# Struktur SRS - MIL-STD-498

SOFTWARE REQUIREMENT SPECIFICATION- DI-IPSC-81433

1. Scope
2. Reference Documents
3. Requirements
  - 3.1 Required states and modes
  - 3.2 CSCI capability requirements
  - 3.3 CSCI external interface requirements
  - 3.4 CSCI internal interface requirements
  - 3.5 CSCI internal data requirements
  - 3.6 Adaptation requirements
  - 3.7 Safety requirements
  - 3.8 Security & privacy requirements
  - 3.9 CSCI environment requirements
  - 3.10 Computer resource requirements
  - 3.11 Software quality factors
  - 3.12 Design and Implementation constraints
  - 3.13 Personnel-related requirements
  - 3.14 Training-related requirements
  - 3.15 Logistics-related requirements
  - 3.16 Other requirements
  - 3.17 Packaging requirements
  - 3.18 Precedence and criticality of requirements
4. Qualification Provisions
5. Requirements Traceability
6. Notes
- A. Appendixes

*Dengan struktur seperti ini dokumen lengkap biasanya membutuhkan minimal 10 halaman*

**MIL-STD- Military Standard, digunakan oleh Department of Defense Amerika**



KNOWLEDGE & SOFTWARE ENGINEERING

# **IEEE Std 830-1998**

## **A.1 Template of SRS Section 3 organized by mode: Version 1**

71

- 3. Specific requirements
  - 3.1 External interface requirements
    - 3.1.1 User interfaces
    - 3.1.2 Hardware interfaces
    - 3.1.3 Software interfaces
    - 3.1.4 Communications interfaces
  - 3.2 Functional requirements
    - 3.2.1 Mode 1
      - 3.2.1.1 Functional requirement 1.1
      - .
      - .
      - 3.2.1.*n* Functional requirement 1.*n*
    - 3.2.2 Mode 2
      - .
      - .
      - 3.2.2.*m* Mode *m*
        - 3.2.2.*m*.1 Functional requirement *m*.1
        - .
        - .
        - 3.2.2.*m*.*n* Functional requirement *m*.*n*
    - 3.2.3 Mode 3
    - 3.2.4 Mode 4
  - 3.3 Performance requirements
  - 3.4 Design constraints
  - 3.5 Software system attributes
  - 3.6 Other requirements



- 3. Specific requirements
    - 3.1. Functional requirements
      - 3.1.1 Mode 1
        - 3.1.1.1 External interfaces
          - 3.1.1.1.1 User interfaces
          - 3.1.1.1.2 Hardware interfaces
          - 3.1.1.1.3 Software interfaces
          - 3.1.1.1.4 Communications interfaces
        - 3.1.1.2 Functional requirements
          - 3.1.1.2.1 Functional requirement 1
          - 3.1.1.2.n Functional requirement n
        - 3.1.1.3 Performance
      - 3.1.2 Mode 2
        - .
        - .
        - .
    - 3.1.m Mode m
    - 3.2 Design constraints
    - 3.3 Software system attributes
    - 3.4 Other requirements



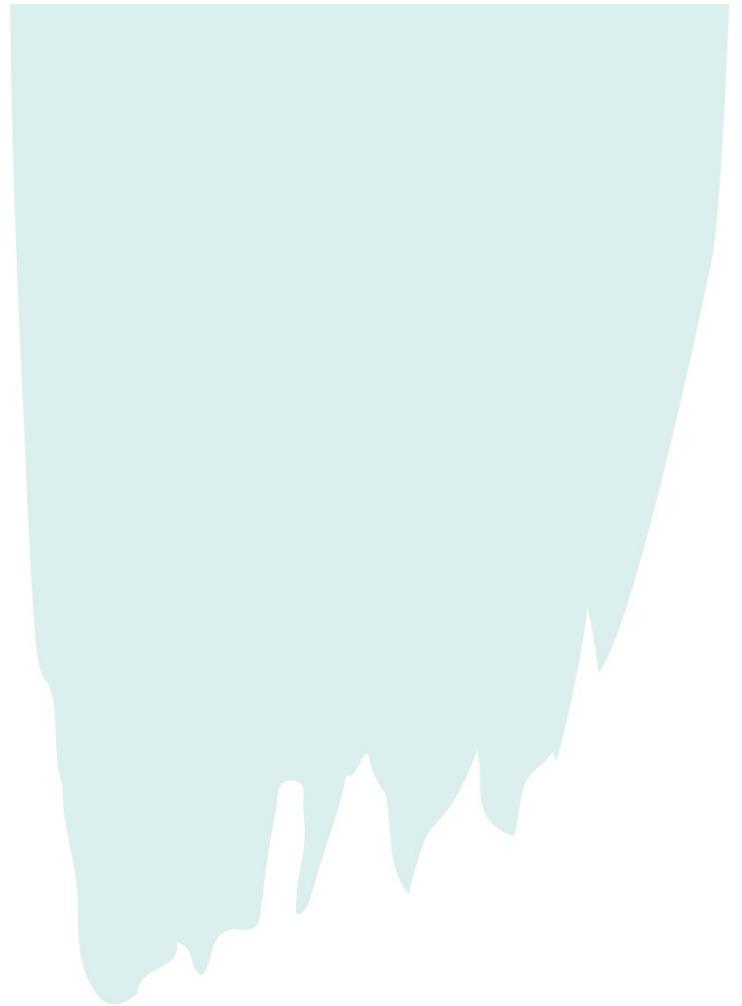
KNOWLEDGE & SOFTWARE ENGINEERING



IF2250 Penulisan SKPL

# **Dokumen SKPL**

**(Tentang dokumen akan  
dibantu penjelasannya  
oleh asisten)**



74



Analisis Kebutuhan P/L - Terstruktur

# Cover

**GL01**

## SPESIFIKASI KEBUTUHAN PERANGKAT LUNAK

<Nama Perangkat Lunak atau Nama Sistem>

untuk:

<Nama User atau Nama Perusahaan>

Dipersiapkan oleh:

<Nomor Grup & Anggota>

Program Studi Teknik Informatika

STEI - ITB

Jl. Ganesha 10, Bandung 40132

 Program Studi Teknik Informatika STEI – ITB	Nomor Dokumen		Halaman
	<b>GL01-SKPL</b>		<#/>/<jml #>
	Revisi	<nomor revisi>	Tgl: <isi tanggal>



KNOWLEDGE & SOFTWARE ENGINEERING

ANALISIS KEDUTUNAN P/L - Terstruktur

- Daftar Perubahan
  - Versi awal tidak memiliki revisi
  - Revisi pertama, diberikan penjelasan apa yang direvisi
- Dokumen perlu ditandatangani oleh:
  - Yang bertanggung jawab dalam penulisan
  - Pemeriksaan dokumen
  - Penyetujuan dokumen

**DAFTAR PERUBAHAN**

Revisi	Deskripsi
A	
B	
C	
D	
E	
F	
G	

INDEX TGL	-	A	B	C	D	E	F	G
Ditulis oleh								
Diperiksa oleh								
Disetujui oleh								



- Isikan dengan halaman yang berubah dibandingkan dengan revisi sebelumnya.
- Tuliskan juga nomor revisi

Daftar Halaman Perubahan

Halaman	Revisi	Halaman	Revisi



KNOWLEDGE & SOFTWARE ENGINEERING

Analisis Kebutuhan P/L - Terstruktur

## Daftar Isi

1.	Pendahuluan .....	5
1.1	Tujuan Penulisan Dokumen.....	5
1.2	Lingkup Masalah .....	5
1.3	Definisi, Istilah dan Singkatan.....	5
1.4	Aturan Penomoran.....	5
1.5	Referensi.....	5
1.6	Deskripsi umum Dokumen (Ikhtisar) .....	5
2	Deskripsi Umum Perangkat Lunak.....	6
2.1	Deskripsi Umum Sistem .....	6
2.2	Karakteristik Pengguna.....	6
2.3	Batasan .....	6
2.4	Lingkungan Operasi .....	6
3	Deskripsi Kebutuhan .....	7
3.1	Kebutuhan Antarmuka Eksternal.....	7
3.1.1	Antarmuka pemakai.....	7
3.1.2	Antarmuka Perangkat Keras .....	7
3.1.3	Antarmuka Perangkat Lunak .....	7
3.1.4	Antarmuka Komunikasi.....	7
3.2	Kebutuhan Fungsional.....	7
3.2.1	Diagram Konteks.....	7
3.2.2	DFD Level 1 .....	7
3.2.2.1	DFD Level 2 <??> .....	7
3.2.2.2	DFD Level 2 <??> .....	7
3.3	Kebutuhan Data .....	7
3.3.1	E-R diagram.....	8
3.4	Kebutuhan Non Fungsional .....	8
3.5	Batasan Perancangan .....	8
3.6	Kerunutan (traceability).....	8
3.6.1	Data Store vs E-R .....	8
3.7	Ringkasan Kebutuhan.....	9
3.7.1	Kebutuhan Fungsional .....	9
3.7.2	Kebutuhan Non Fungsional .....	9



KNOWLEDGE & SOFTWARE ENGINEERING

## 1. Pendahuluan

### 1.1 Tujuan Penulisan Dokumen

Tuliskan dengan ringkas tujuan dokumen SKPL ini dibuat, dan digunakan oleh siapa.

### 1.2 Lingkup Masalah

Tuliskan dengan ringkas nama aplikasi dan deskripsinya. Maksimal 1 paragraf

### 1.3 Definisi, Istilah dan Singkatan

Semua definisi dan singkatan yang digunakan dalam dokumen ini dan penjelasannya

### 1.4 Aturan Penomoran

Tuliskan jika anda memakai aturan penomoran

### 1.5 Referensi

Dokumentasi PL yang dirujuk oleh dokumen ini.

Buku, Panduan, Dokumentasi lain yang dipakai dalam pengembangan PL ini.

### 1.6 Deskripsi umum Dokumen (Ikhtisar)

Tuliskan sistematika pembahasan dokumen SKPL ini.

## 2 Deskripsi Umum Perangkat Lunak

### 2.1 Deskripsi Umum Sistem

Tuliskan overview P/L, dalam bentuk gambar dan narasi yang dapat memberikan gambaran tentang aplikasi dan konteksnya, yaitu hubungannya dengan dunia luar (gambar yang mirip dengan diagram konteks, tetapi dengan notasi yang lebih mudah dimengerti orang awam).

### 2.2 Karakteristik Pengguna

Minimal sebuah tabel dengan Kolom : Pengguna, Pekerjaan, Hak Akses. Kolom Hak Akses dihubungkan dengan Fungsi utama yang muncul pada Fungsi Produk

Kategori Pengguna	Tugas	Hak Akses ke aplikasi

### 2.3 Batasan

Batasan (jika ada), ketergantungan SW terhadap SW/HW/sistem lain (misalnya modul Konsolidasi baru dapat dijalankan ketika rekapitulasi data akuntansi dari Aplikasi AKUNT sudah dijalankan dan datanya dinyatakan OK oleh petugas

Batasan yang harus dipakai. Misalnya :

- harus memakai file data dari Sistem lain (sebutkan),
- harus memakai format data yang sama dengan sistem lain
- harus berfungsi multi platform (di Windows dan linux)

### 2.4 Lingkungan Operasi

Operating system, DBMS, ...

Aplikasi Client server ini akan berfungsi dengan spesifikasi :

Server : ???

Client : ?????

OS :

DBMS :

Analisis Kebutuhan P/L - Terstruktur

### 3 Deskripsi Kebutuhan

### **3.1 Kebutuhan Antarmuka Eksternal**

*Hanya diisi jika P/L memerlukan fasilitas khusus .*

### 3.1.1 Antarmuka pemakai

User interface untuk mengoperasikan Perangkat Lunak : keyboard, mouse

### 3.1.2 Antarmuka Perangkat Keras

*Hanya diisi jika perlu perangkat keras khusus, misalnya CARD XXX, CABLE XYZ*

### 3.1.3 Antarmuka Perangkat Lunak

Hanya diisi jika PL memakai interface (berupa PL), misalnya API Windows.

### 3.1.4 Antarmuka Komunikasi

Hanya diisi jika PL beroperasi di jaringan dan membutuhkan alat komunikasi khusus, misalnya RS232.

### **3.2 Kebutuhan Fungsional**

*Diawali dengan membuat daftar kebutuhan fungsional P/L, lengkap dengan ID dan penjelasan jika perlu. Bisa dibuat dalam bentuk tabel.*

Pada subbab berikutnya, buatlah diagram konteks dan DFD level berikutnya.

### 3.2.1 Diagram Konteks

### 3.2.2 DFD Level 1

### 3.2.2.1 DFD Level 2 <????>

### 3.2.2.2 DFD Level 2 <????>

### 3.2.3 Spesifikasi Proses (P-SPEC)

### 3.2.4 Data Store

### 3.3 Kebutuhan Data

*Diisi untuk kebutuhan kuliah basisdata*

### 3.3.1 E-R diagram

### 3.4 Kebutuhan Non Fungsional

Uraikan dengan ringkas kebutuhan non fungsional dalam tabel sebagai berikut. Isilah Kolom Kebutuhan dengan kalimat yang jelas dan kelak dapat diujicobakan apakah dipenuhi. ID adalah nomor kebutuhan yang harus ditelusuri pada saat test. Tuliskan N/A bila Not Applicable..

ID	Parameter	Kebutuhan
	Availability	
	Reliability	
	Ergonomics	
	Portability	
	Memory	
	Response time	
	Safety	N/A
	Security	
	Others 1: Bahasa komunikasi	Misalnya : semua tanya jawab harus dalam bahasa Indonesia
		Setiap layar harus mengandung logo PT Pos Indonesia

Catatan :

Availability : ketersediaan aplikasi, misalnya harus terus menerus beroperasi 7 hari per minggu, 24 jam per hari tanpa gagal

Reliability : keandalan, misalnya tidak pernah boleh gagal (atau kegagalan yang ditolerir adalah ...%) sehingga harus dipikirkan fault tolerant architecture. Biasanya hanya perlu untuk Critical Application yang jika gagal akan berakibat fatal.

Ergonomics : kenyamanan pakai bagi pengguna

Portability : kemudahan untuk dibawa dan dioperasikan ke mesin/sistem operasi/platform yang lain

Memory : jika perhitungan kapasitas memori internal kritis (misalnya untuk SW yang harus dijadikan CHIPS dan ukurannya harus kecil)

Response time : Batasan waktu yang harus dipenuhi. Sangat penting untuk aplikasi Real Time. Contoh: "Aplikasi harus mampu menampilkan hasil dalam 4 detik", atau "ATM harus menarik kembali kartu yang tidak diambil dalam waktu 3 menit"

Safety: yang menyangkut keselamatan manusia, misalnya untuk SW yang dipakai pada sistem kontrol di pabrik

Security : aspek keamanan yang harus dipenuhi.

### 3.5 Batasan Perancangan

Sebutkan batasan perancangan jika ada. Contoh : harus memakai library yang ada, harus memakai sepotong kode yang sudah pernah dikembangkan, harus memperhatikan hal-hal tertentu

### 3.6 Kerunutan (traceability)

Diisi dengan tabel yang berisi traceability dari hasil analisis. Gunanya untuk menilai apakah hasil analisis "runut" dan logik. Untuk sementara, baru didefinisikan Data-store versus E-R.

### 3.6.1 Kebutuhan Fungsional vs Proses

Mapping kebutuhan fungsional dengan proses pada DFD

ID Kebutuhan Fungsional	Nomor Proses pada DFD

Analisis Kebutuhan P/L - Terstruktur



### 3.6.2 Data Store vs E-R

### *Mapping data store pada DFD dengan Entity - Relasi*

Data Store	Entity	Relasi

### **3.7 Ringkasan Kebutuhan**

Bab ini berisi ringkasan semua kebutuhan. Kebutuhan ini mencerminkan semua hal yang harus dipenuhi, dan nantinya akan menjadi arahan untuk tahapan testing, karena pada dasarnya, semua kebutuhan harus dapat ditest supaya dapat dibuktikan dipenuhi. Dibagi menjadi dua bagian: fungsional dan non fungsional.

### **3.7.1 Kebutuhan Fungsional**

### 3.7.2 Kebutuhan Non Fungsional

# *Analisis Kebutuhan dan Pemodelan*



# *Pemodelan Kebutuhan*

- Spesifikasi kebutuhan perlu diperjelas dengan suatu model atau diagram
- Pemodelan hingga saat ini menggunakan 2 pendekatan
  - Model dengan pendekatan Terstruktur (**Structured Approach**) – Tradisional/Konvensional
    - Diagram konteks – DFD
    - Diagram ER
  - Model dengan pendekatan Obyek (**Object Orientation**) – lebih baru, diagram UML (Unified Modeling Language)
    - Diagram Use case
    - Diagram Kelas/Objek
    - Diagram Sekuens, dll



KNOWLEDGE & SOFTWARE ENGINEERING

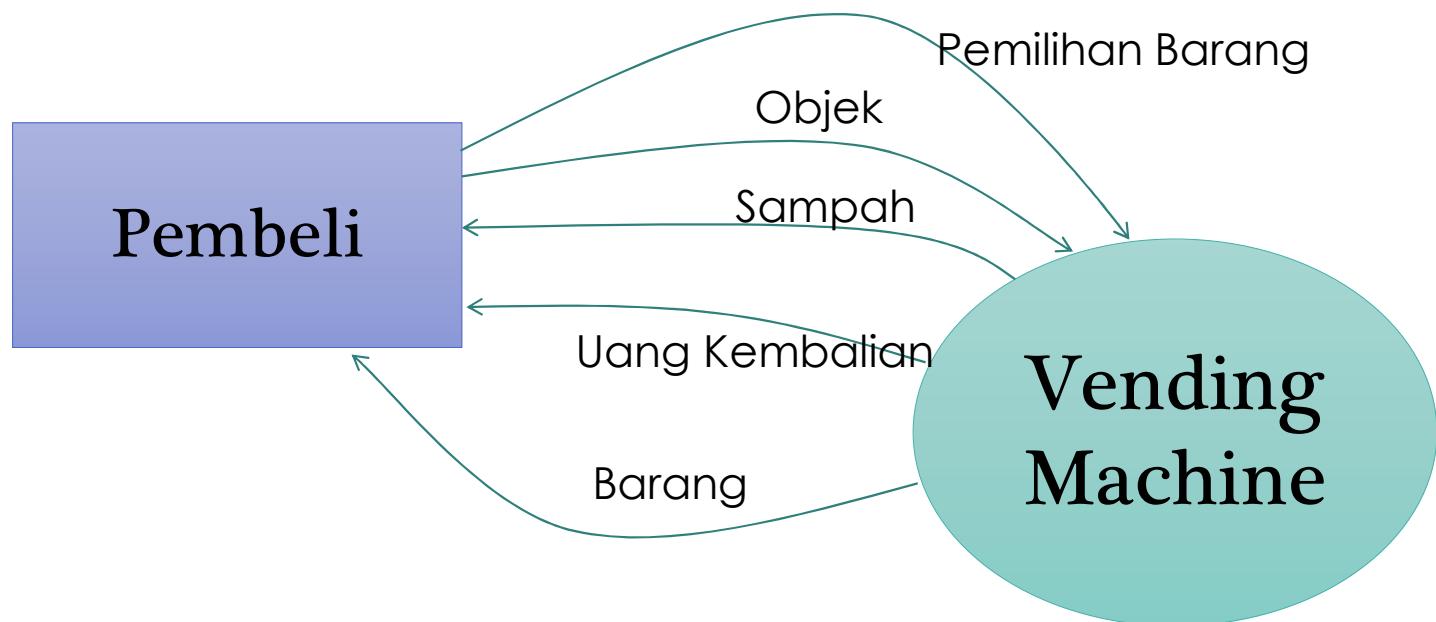
# **Model Kebutuhan Apa Yang Dipilih?**

- Selain pendekatan terstruktur dengan pemodelan Diagram Konteks, untuk pendekatan lain dapat juga dengan pendekatan berorientasi objek
  - Use case (UC) dapat digunakan untuk tujuan yang sama dengan Diagram Konteks (DK).
- Diagram Konteks dan Use case digunakan untuk menggambarkan interaksi sistem dengan entitas/aktor yang terkait
  - DK dapat menggambarkan hubungan antara entitas dengan sistem yang akan dikembangkan
    - DK memberikan deskripsi aliran data dari/ke suatu entitas dari/ke sistem
    - Berawal dari ide pendekatan **Input-Proses-Output**
  - UC menggambarkan suatu aktor dapat melakukan atau memiliki kemampuan apa saja terhadap sistem yang akan dikembangkan.
- Menggunakan DK harus dilengkapi dengan proses dan aliran data yang lebih rinci untuk melengkapi model kebutuhannya.
  - Model ini lebih dikenal dengan nama DFD (Data Flow Diagram)
  - Menggunakan pendekatan terstruktur (Structured Approach)
- Use Case hanya fokus pada kebutuhan yang ada pada sistem tanpa terlalu memikirkan data apa yang akan diperlukan oleh sistem.
  - Model ini banyak digunakan untuk pendekatan berorientasi objek (Object Oriented Approach)

# *Interaksi dengan Sistem*

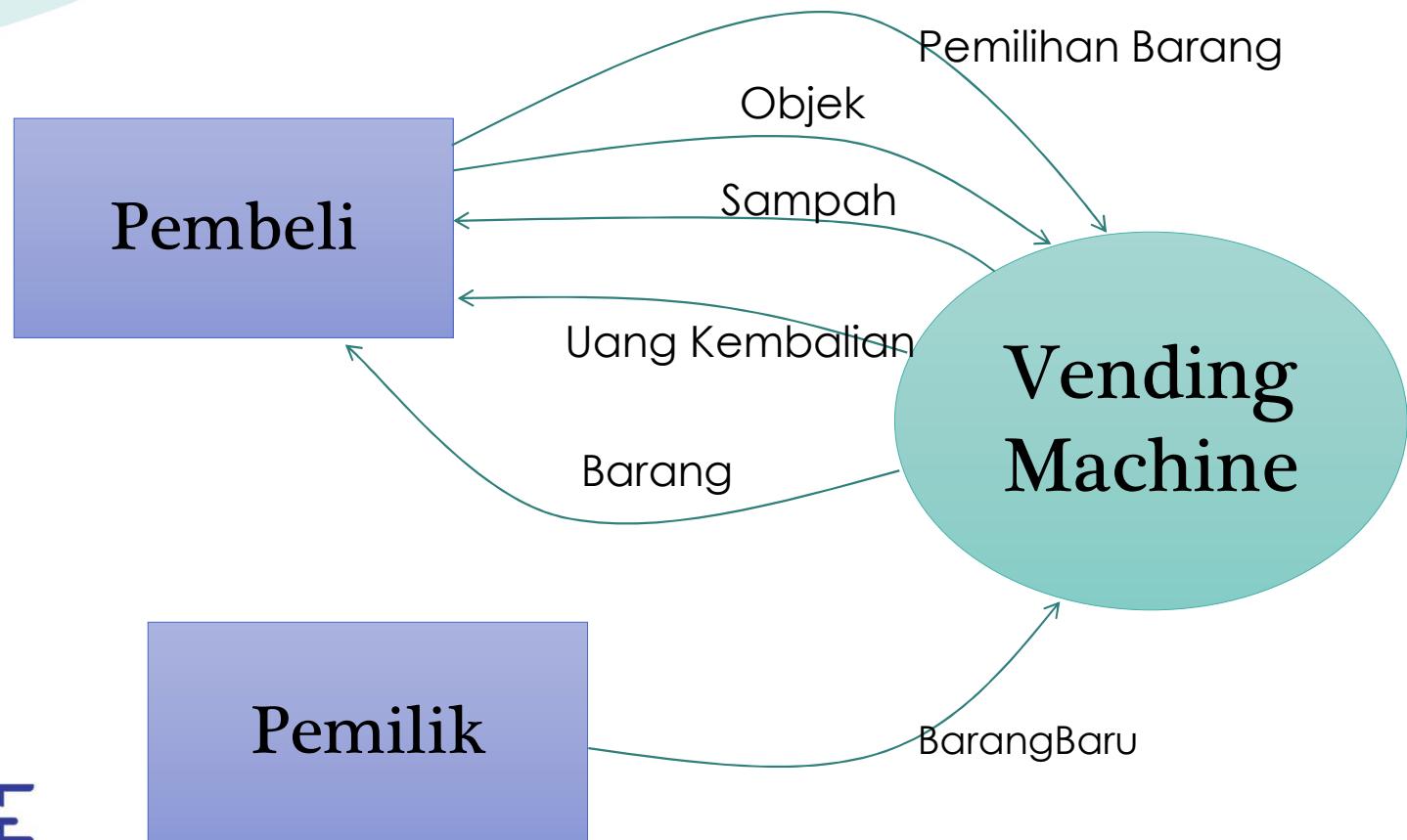
- Pada tahap awal pengembangan perangkat lunak kita harus mendefinisikan **siapa** atau **apa** saja yang terkait dengan sistem/perangkat lunak(software) yang akan dibuat.
- Pengembangan model diperlukan untuk memudahkan analisa dan juga komunikasi
  - Diagram konteks dapat digunakan untuk menggambarkan siapa saja yang berinteraksi dengan sistem
  - Use-case digunakan untuk menggambarkan keterlibatan ‘aktor’ dengan sistem yang akan dikembangkan.

# Diagram Konteks VM



KNOWLEDGE & SOFTWARE ENGINEERING

# Diagram Konteks V:M (Lengkap)



# Catatan

- Diagram Konteks biasanya digunakan dalam pendekatan model struktural
- Use-Case biasanya digunakan dalam pendekatan pemodelan berorientasi objek
  - akan *dijelaskan di bagian slide terakhir*
- Bentuk diagram di atas mungkin bukan solusi tunggal, solusi lain bisa ditawarkan tergantung pemahaman kita, dan juga jenis/kategori persoalan yang akan dipecahkan
  - Jumlah aktor atau entitas luar yang terlibat mungkin bertambah
    - Contoh : Pemilik Sistem, User, Sistem Bank, Sensor, dan lain-lain
- Diagram ini digunakan sebagai
  - Pemodelan awal yang dapat digunakan sebagai alat komunikasi dengan user atau customer.
  - Menunjukkan ruang lingkup permasalahan

# *Checklist untuk validasi kebutuhan*

- Apakah semua kebutuhan sudah dinyatakan dengan jelas? Hati-hati salah interpretasi
- Apakah sumber kebutuhan sudah teridentifikasi dengan jelas?
  - Sumber dapat berasal dari 'nama orang', nama dokumen,
  - Apakah SKPL sudah dicek lagi dengan sumber ?
- Apakah semua kebutuhan sudah jelas secara 'kuantitatif'
- Kebutuhan apa yang mungkin berhubungan atau terkait dengan kebutuhan lain
  - Harus jelas dinyatakan bagaimana hubungan atau kaitannya
  - Harus jelas urutan kebutuhannya
- Apakah suatu kebutuhan tidak sesuai dengan 'constraint' dari suatu domain
  - Misalnya: apakah meminjam 10 buku sesuai dengan aturan organisasi perpustakaan
- Apakah suatu kebutuhan dapat diselesaikan (sesuai dengan rencana waktu dan biaya)?
- Apakah kebutuhan non-fungsional telah terdefinisi dengan baik bagaimana merealisasikannya
  - Misalnya kebutuhan 'aplikasi' harus cepat, harus terdefinisi 'cepat' itu bagaimana?
    - Apakah 10 transaksi per detik? Atau penyimpanan suatu data tidak lebih dari 1 detik.

Tim Pengajar IF2250

IF2250 – Rekayasa Perangkat Lunak  
**Perancangan Terstruktur**  
*(Structured Design)*

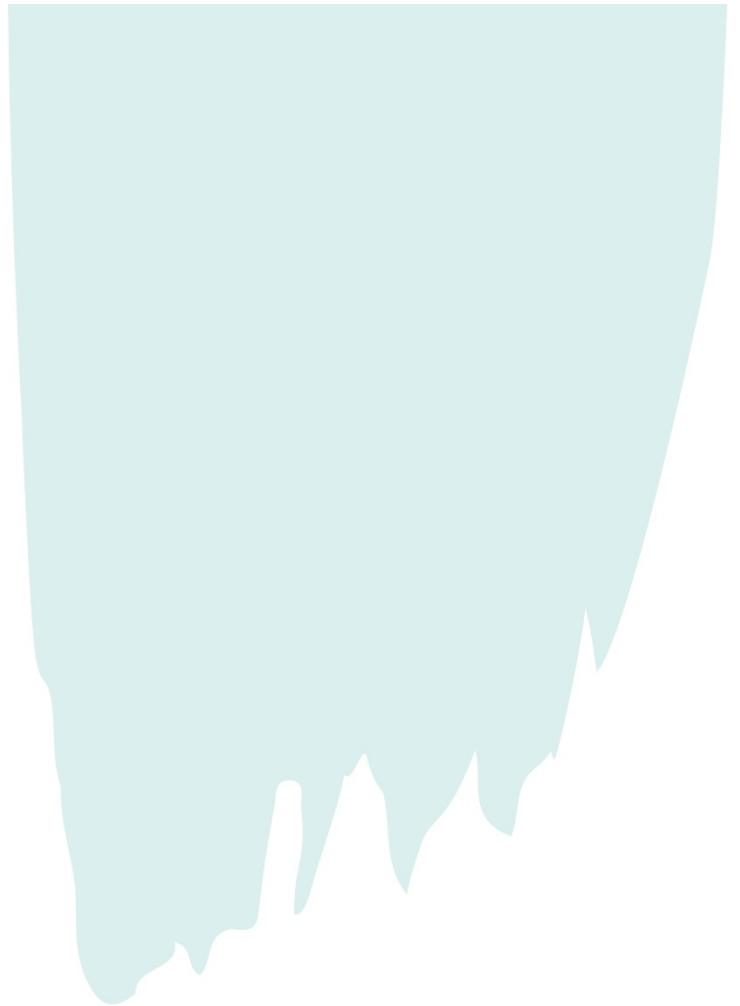
SEMESTER II TAHUN AJARAN 2022/2023



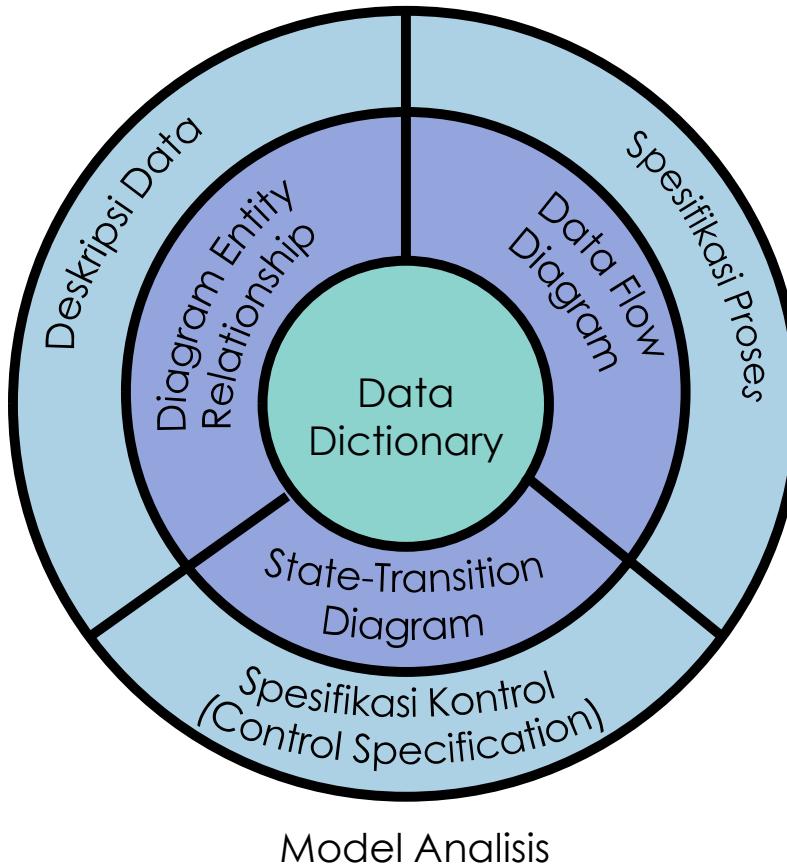
# ***Pemodelan Secara Terstruktur (Structured Approach Modeling)***



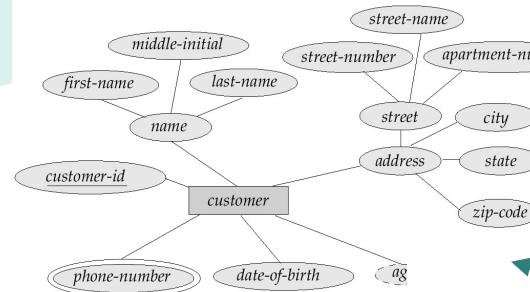
KNOWLEDGE & SOFTWARE ENGINEERING



# *Model Analisis Terstruktur*



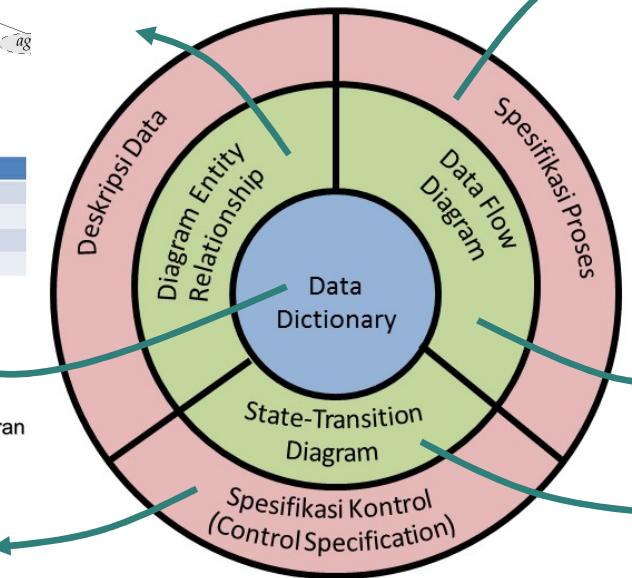
# Model Analisis Terstruktur



Nama	Deskripsi
objek	[koin   sampah]
barang	[soda   permen   chips]
koin	[ 100an   200an   500an   1000an ]
...	...

CSPEC 3: Validasi Pembayaran

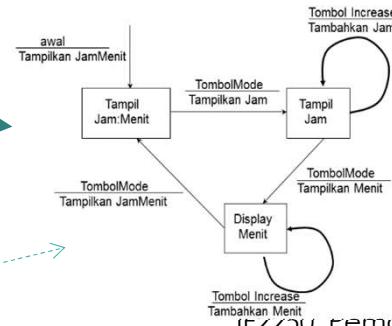
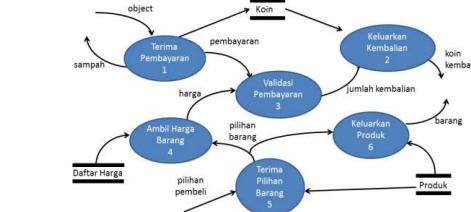
**Inputs:**  
pembayaran : data in  
harga : data in  
**Outputs:**  
jumlah kembalian : data out  
**Body:**  
while (pembayaran) do  
pembayaran ← pembayaran + 1



Biasanya digunakan pada sistem Real Time

PSPEC 3: Validasi Pembayaran

**Inputs:**  
pembayaran : data in  
harga : data in  
**Outputs:**  
jumlah kembalian : data out  
**Body:**  
If (pembayaran >= harga)  
jumlah kembalian = payment - harga  
else  
jumlah kembalian = 0



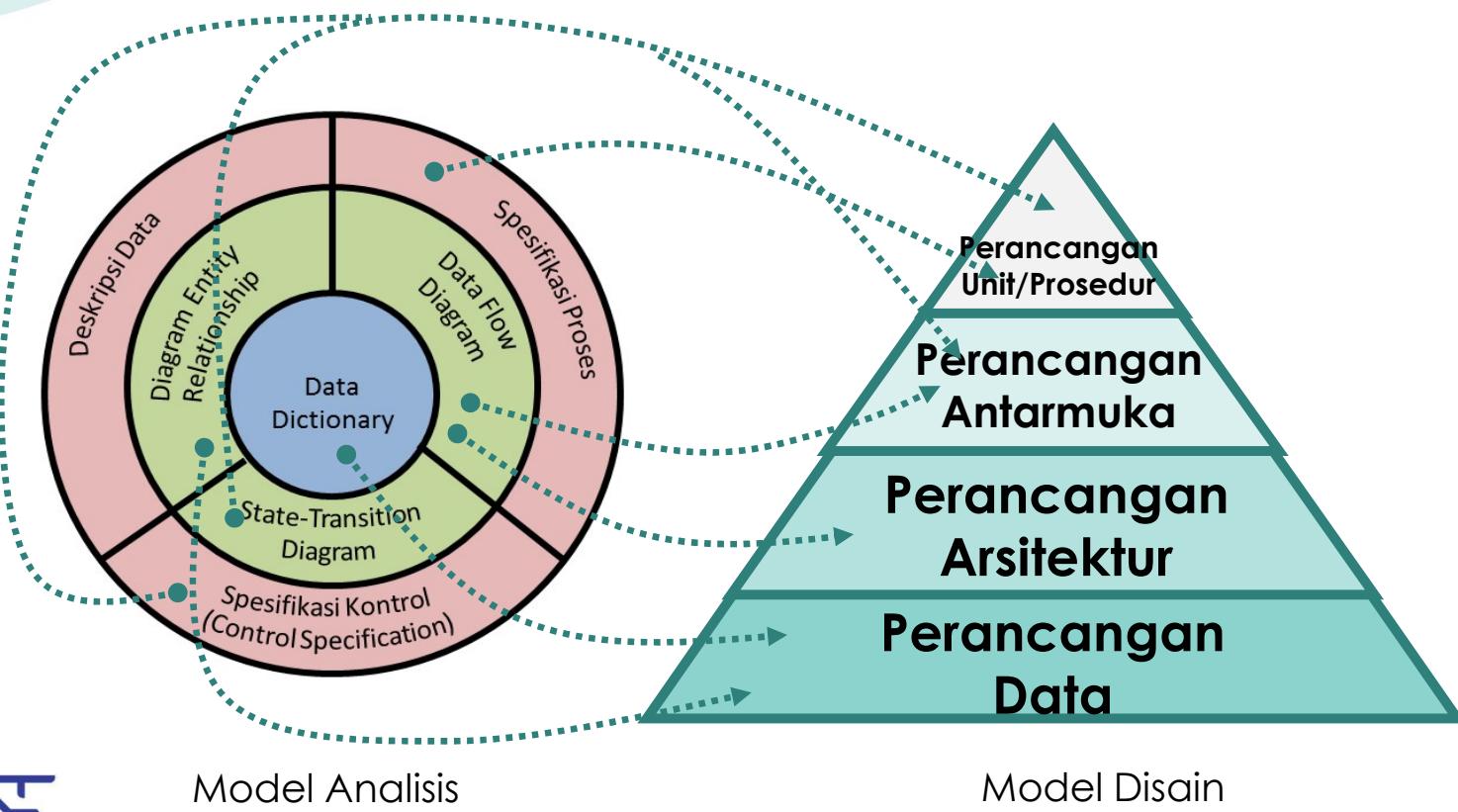
IRZIZOU remodelan Terstruktur



# *Elemen-elemen pada Perancangan*



# Dari Analisis ke perancangan



# 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)

# Spesifikasi Perancangan

- **Perancangan Data**

- Dibentuk dari hasil transformasi analisis model informasi (**Kamus data** dan **Diagram ER**) menjadi **struktur data** untuk implementasi program

- **Perancangan Arsitektur**

- Mendefinisikan hubungan antara elemen struktural utama, yang diturunkan dari **spesifikasi sistem**, **model analisis proses** dan keterhubungannya (dari DFD)

- **Perancangan Antarmuka**

- Mendefinisikan bagaimana elemen dari software saling terhubung dan berkomunikasi, baik dengan sistem lain ataupun dengan pengguna (manusia).
- Diagram **DFD** dan **STD** akan memberikan informasi ini

- **Perancangan Unit/Prosedur**

- Adalah hasil transformasi dari elemen struktural dari arsitektur software menjadi deskripsi suatu unit atau komponen dalam perangkat lunak.



KNOWLEDGE & SOFTWARE ENGINEERING

# 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 mengantikan 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 spesifikasi 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]



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 Pemodelan Terstruktur

# *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

# *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.



KNOWLEDGE & SOFTWARE ENGINEERING

# **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

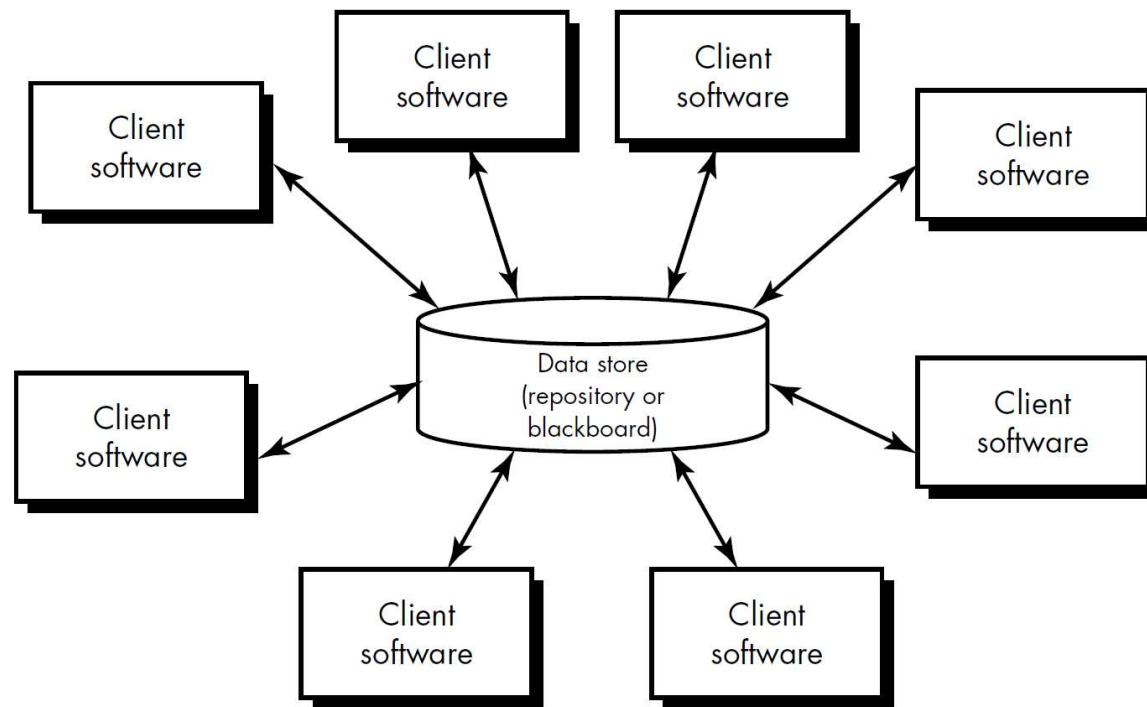
## **5. Layered architectures**

- Arsitektur berlapis

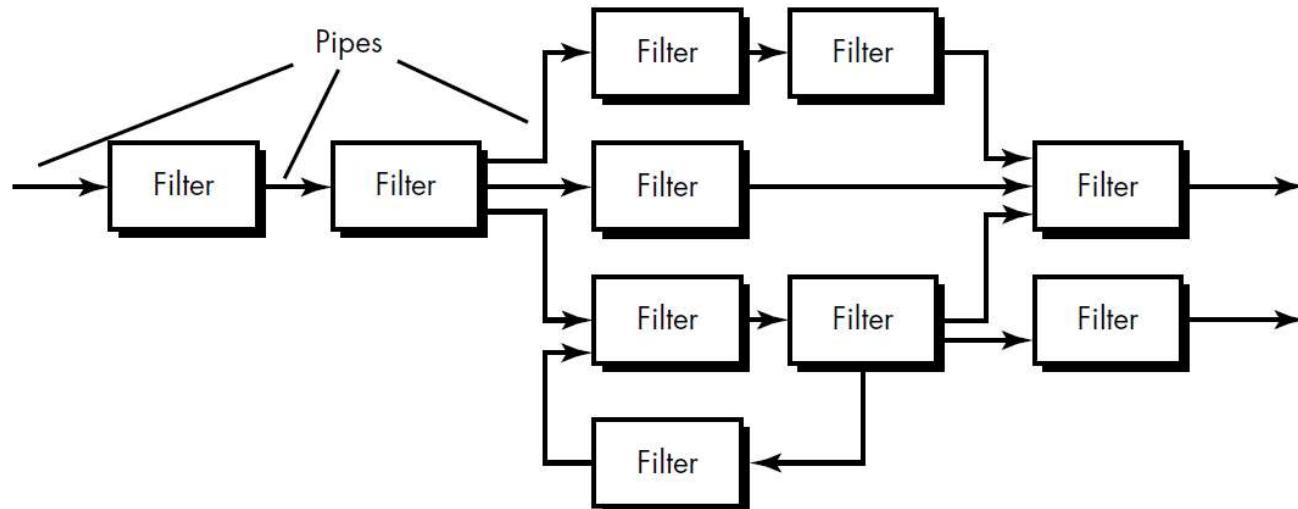
# Karakteristik Setiap Bentuk Arsitektur

- 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

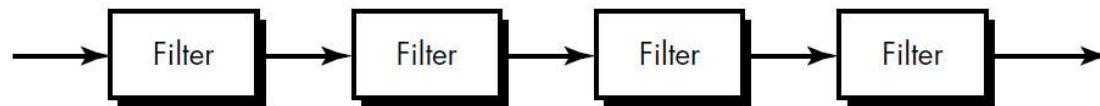
# I. Data-Centered Architecture



## 2. Data Flow Architecture

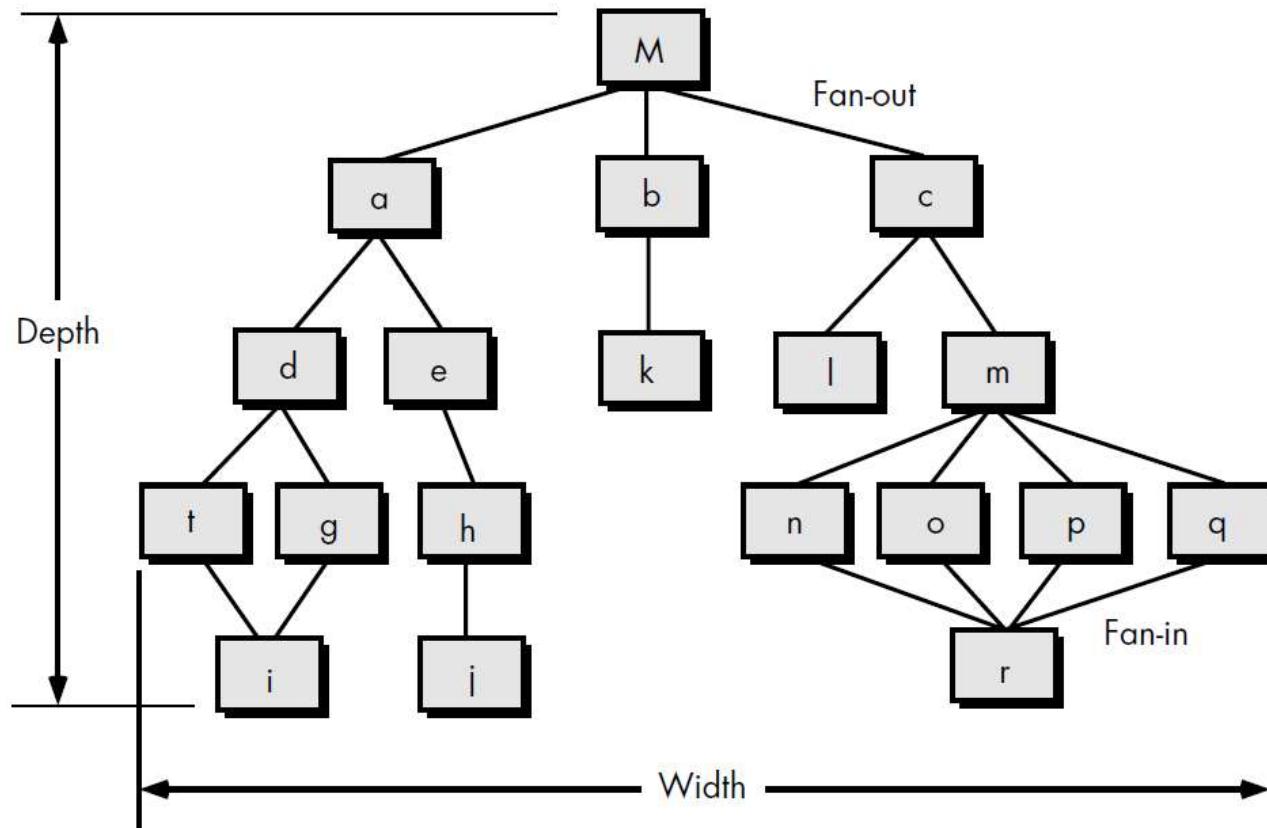


(a) Pipes and filters

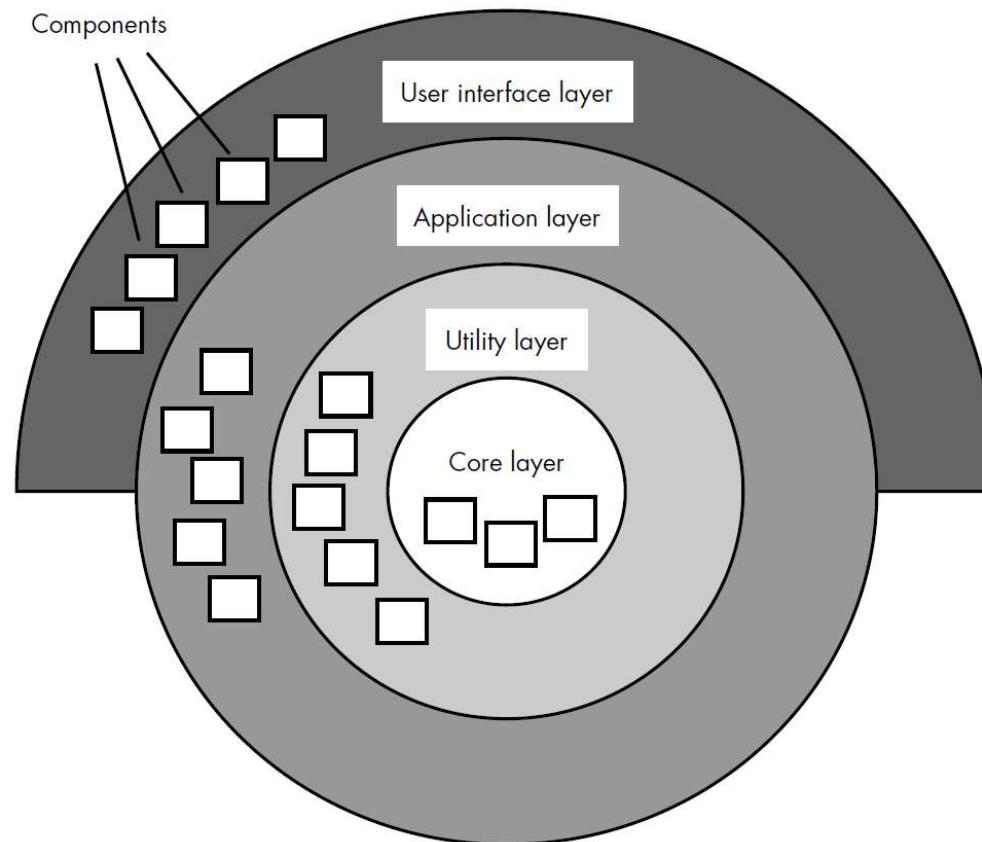


(b) Batch sequential

### 3. Call and Return Architecture

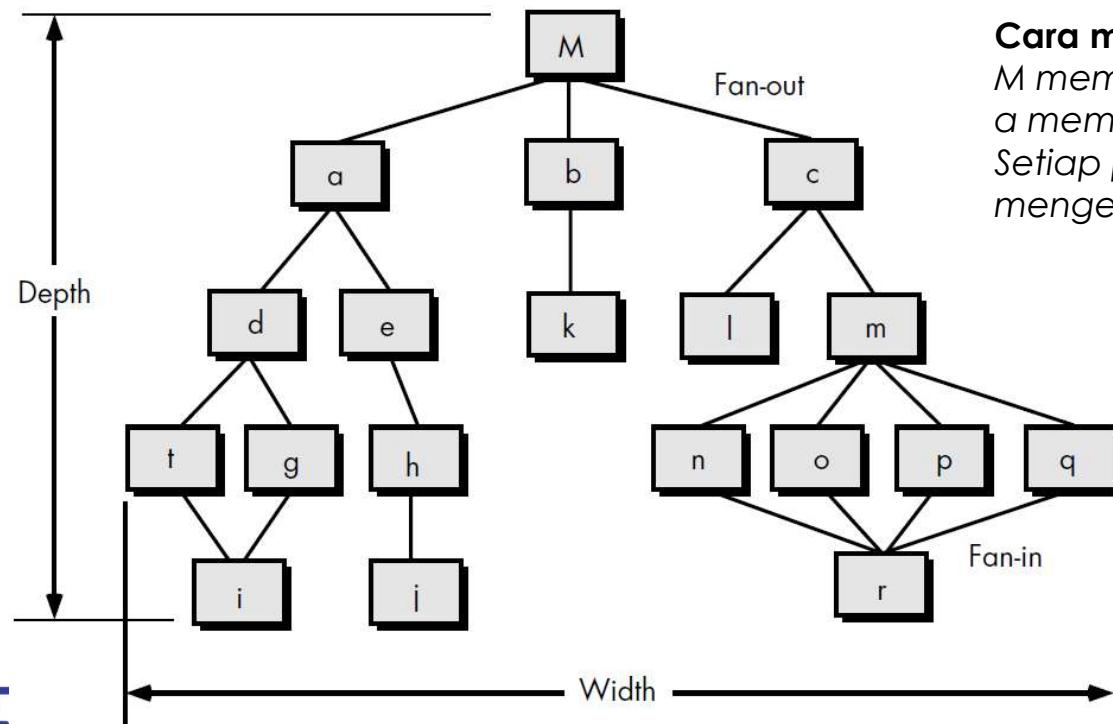


## 5. Layered Architecture



KNOWLEDGE & SOFTWARE ENGINEERING

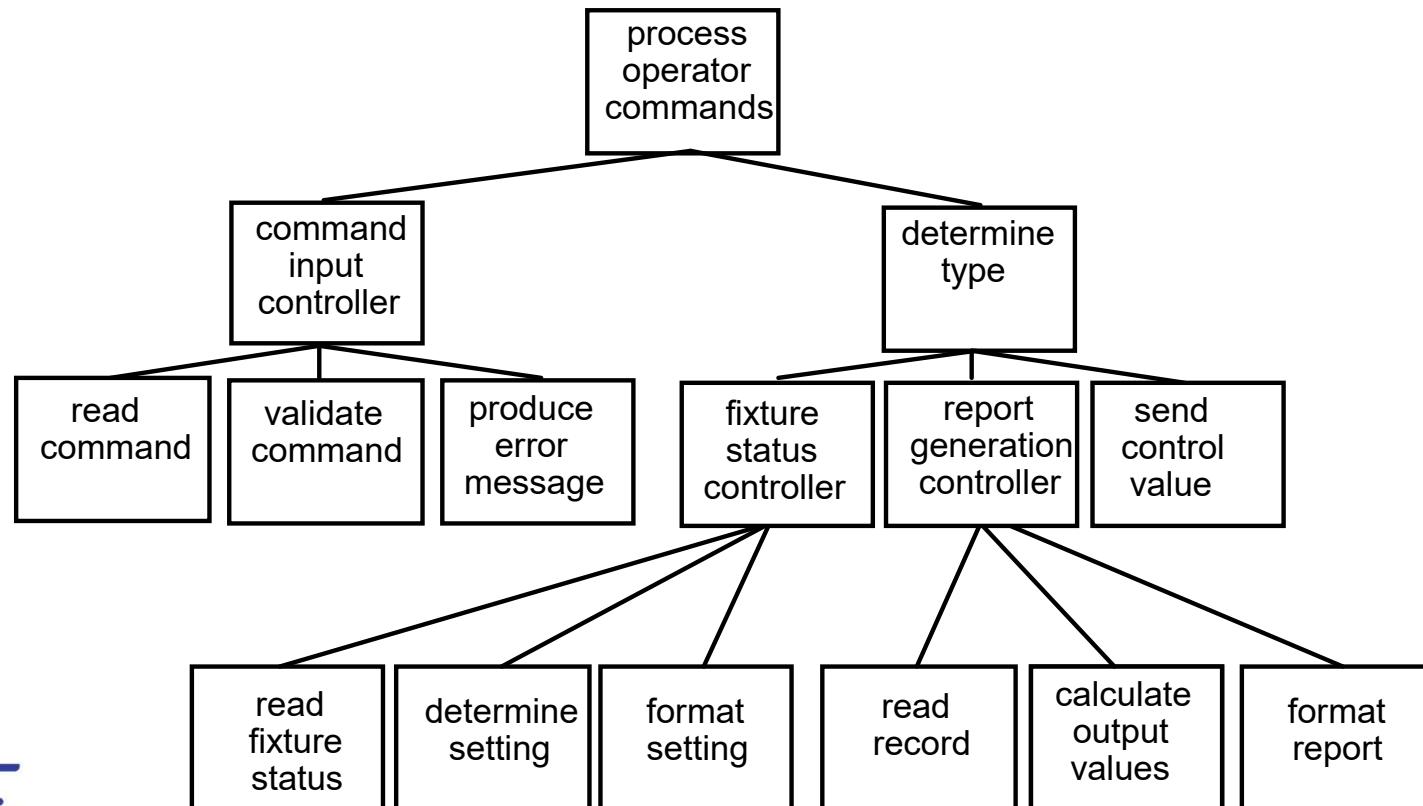
# Contoh Structure Chart



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



# Structure Chart (Diagram Terstruktur)



KNOWLEDGE &amp; SOFTWARE ENGINEERING

# *Perancangan Arsitektur*

(PEMETAAN DFD KE STRUCTURE CHART)



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 Pemodelan Terstruktur

# *Perancangan Terstruktur*

- **Tujuan**

- Membentuk arsitektur program

- **Pendekatan**

- DFD dipetakan menjadi arsitektur program
  - PSPEC dan STD memberikan indikasi isi dari setiap modul

- **Notasi**

- *Structure Chart* (Diagram Terstruktur)

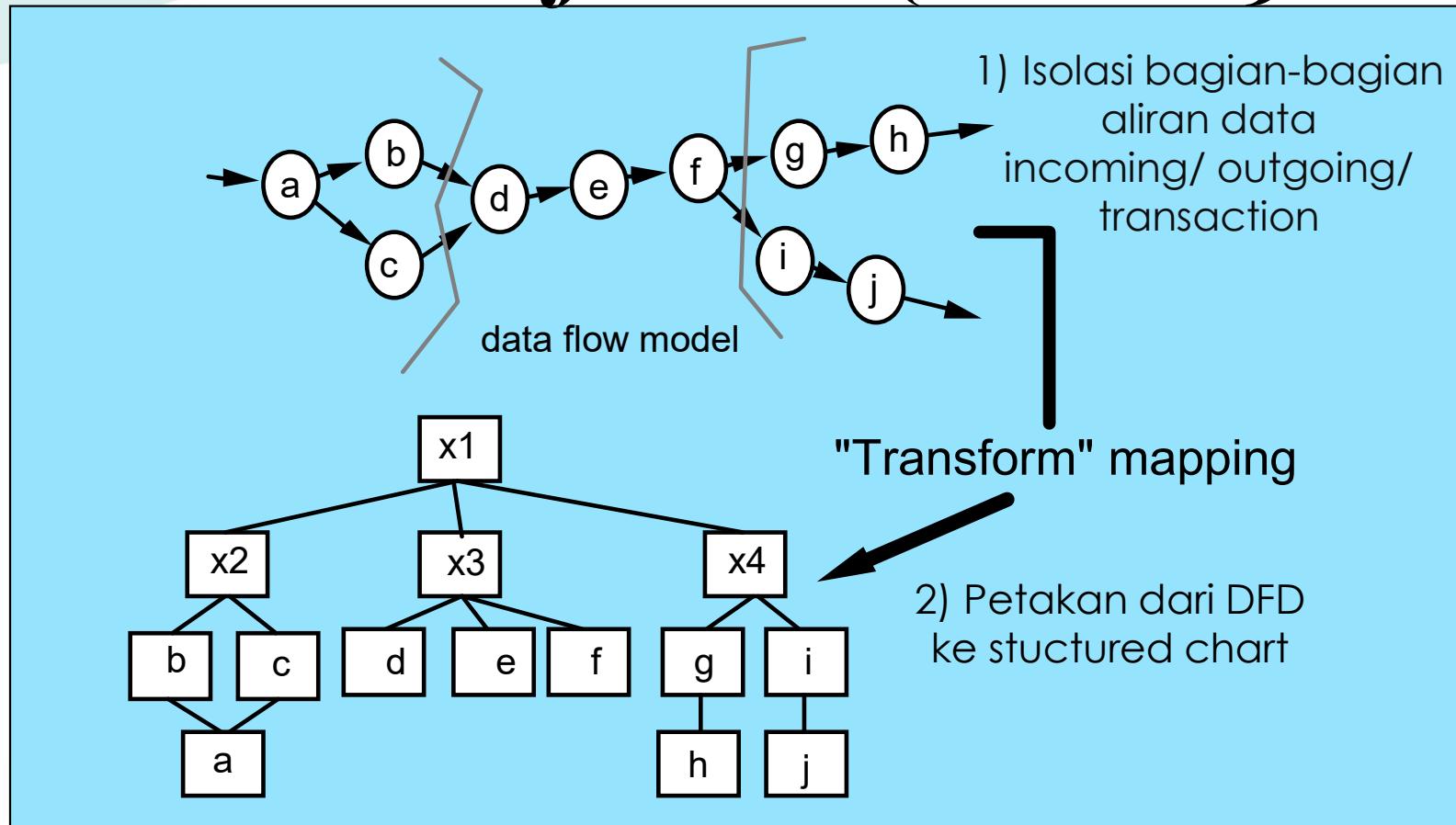


KNOWLEDGE & SOFTWARE ENGINEERING

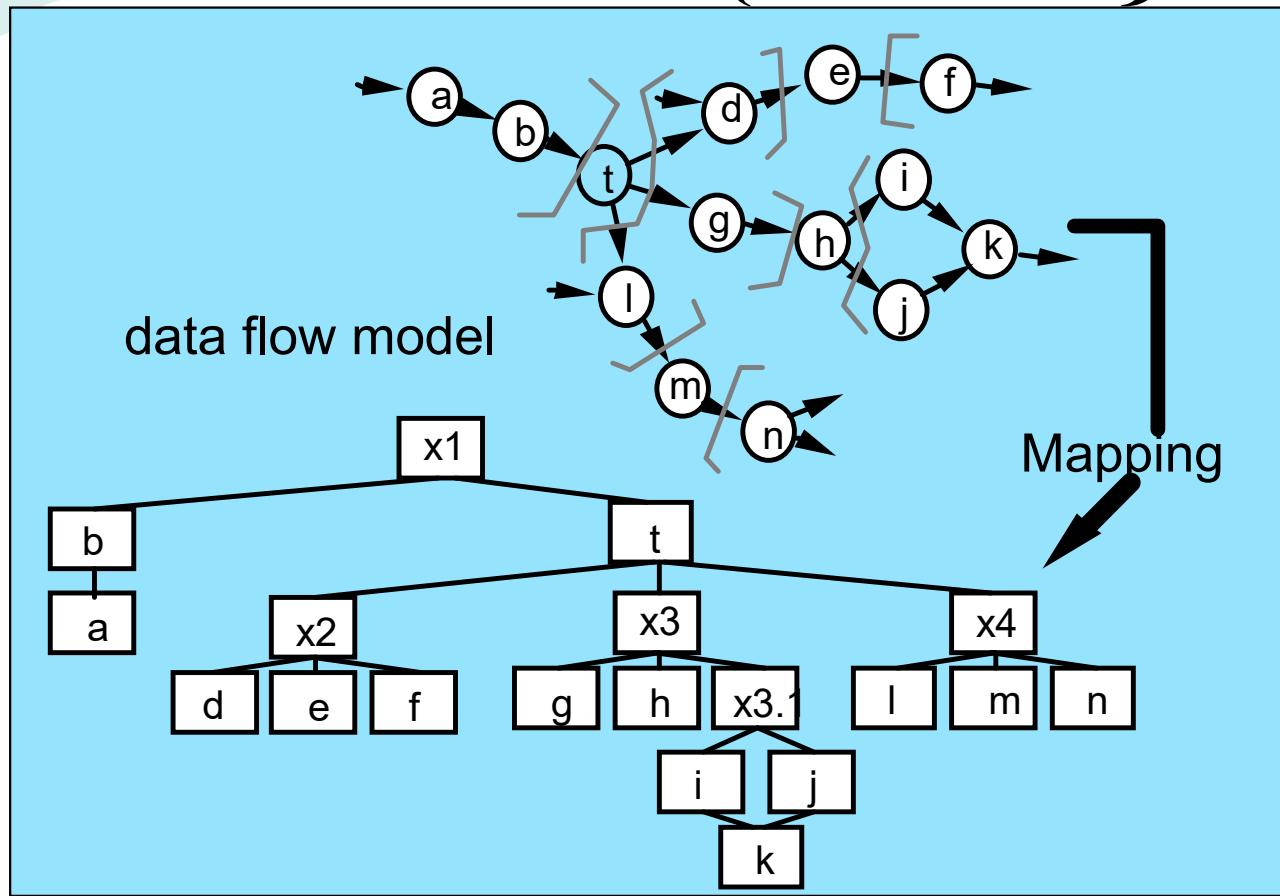
# **Panduan Umum Pembuatan Structure Chart**

- Isolasi *incoming flow* dan *outgoing flow*, berikan batas
  - Untuk flow transaksi, isolasi bagian transaksinya
- Dari batas tadi, petakan DFD, transformasikan menjadi modul yang sesuai
- Tambahkan modul ‘antara’ jika diperlukan
- Perbaiki hasil program struktur dengan memperhatikan modularitas dari struktur hirarki modul

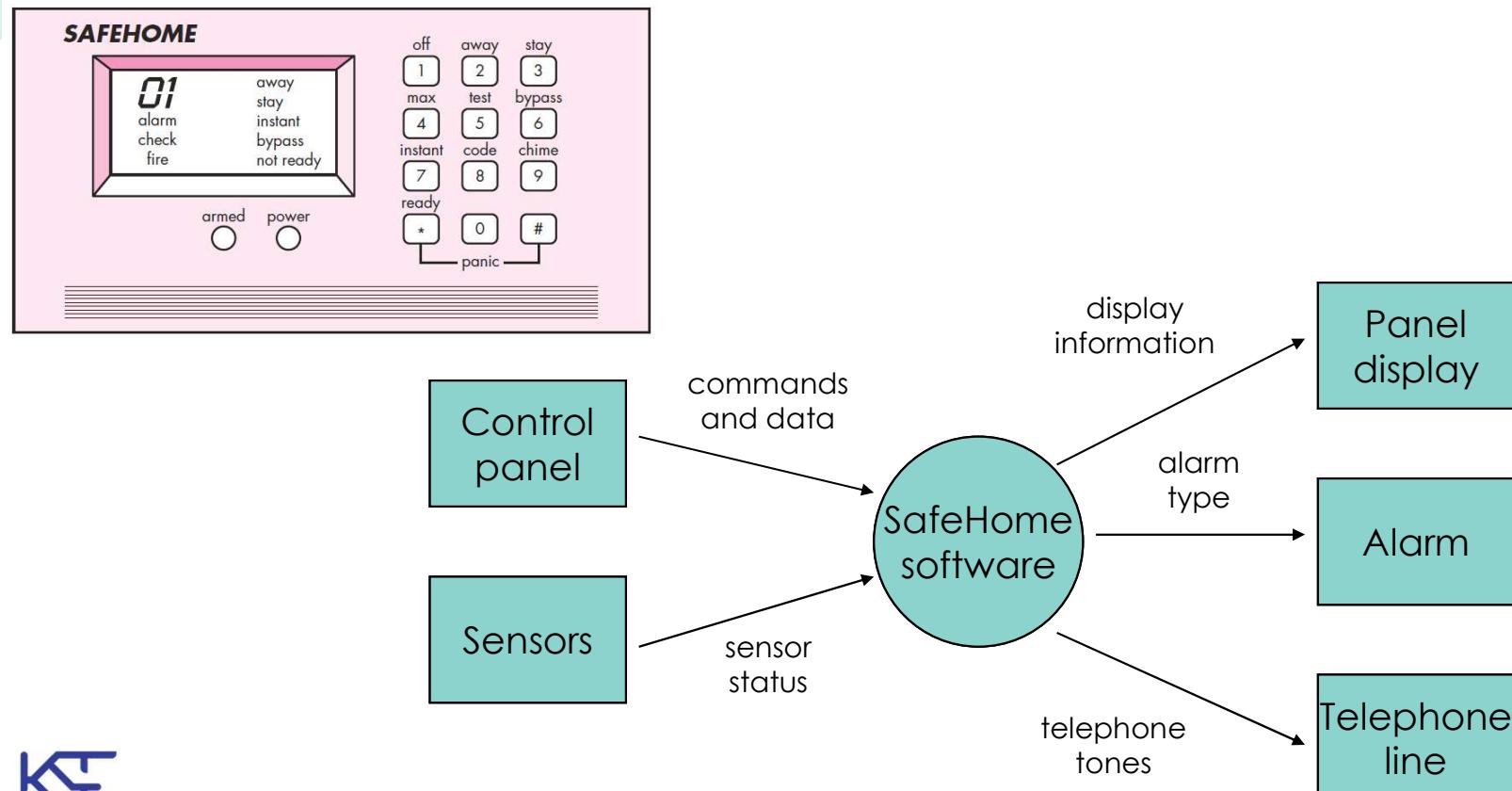
# Pemetaan Transformasi (contoh 1)



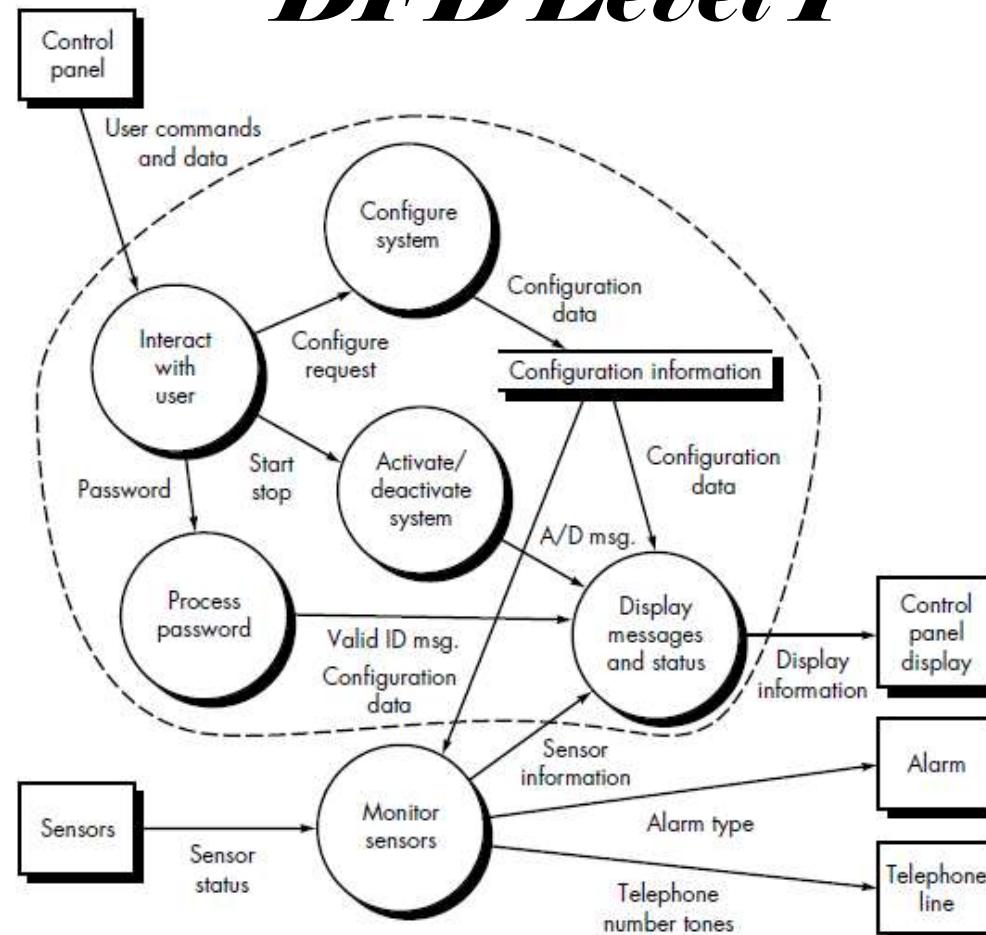
## Pemetaan Transaksi (contoh 2)



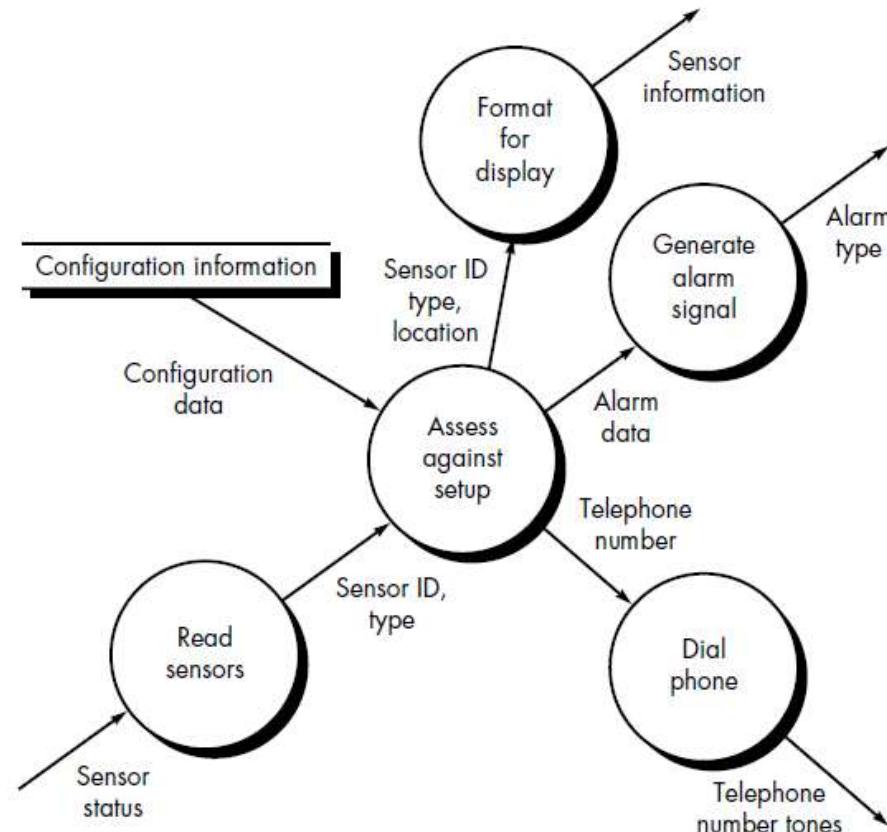
# Diagram Konteks SafeHome



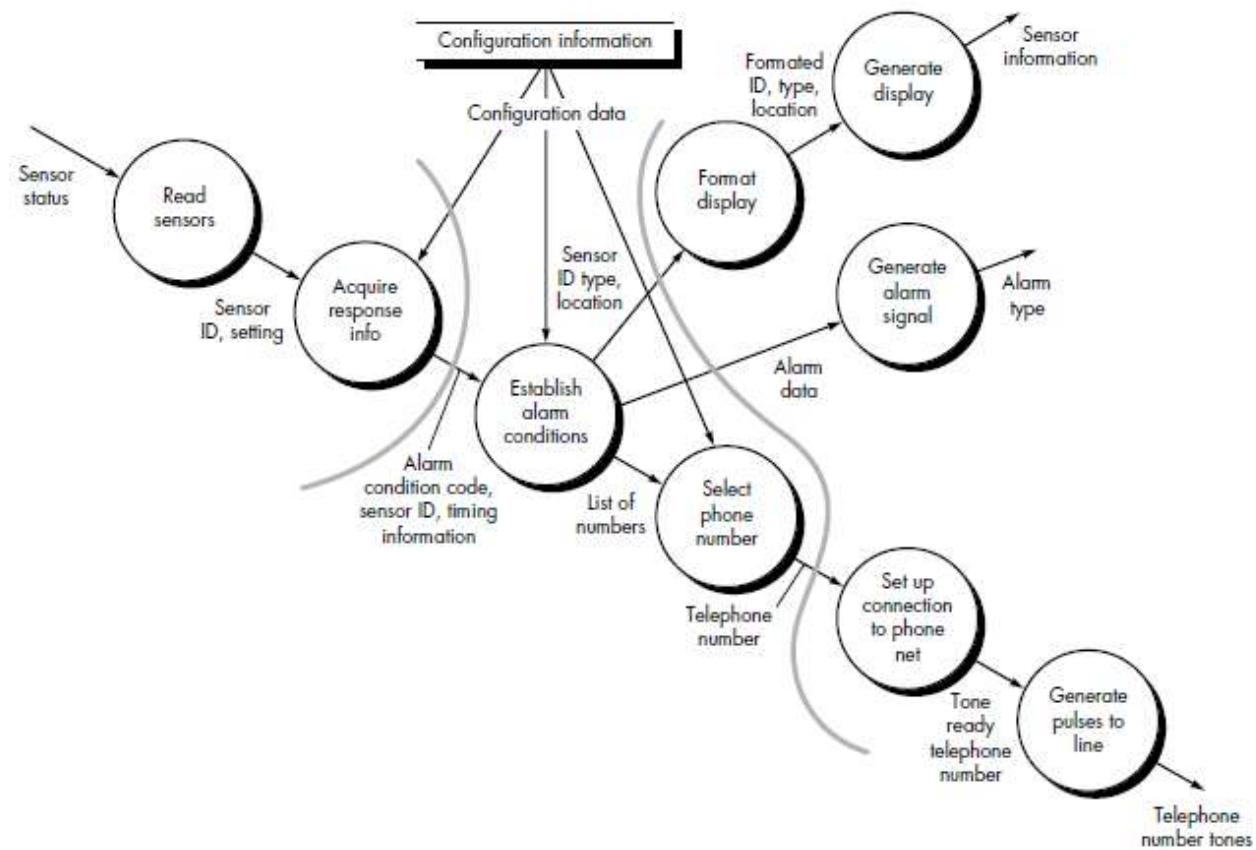
# *DFD Level 1*



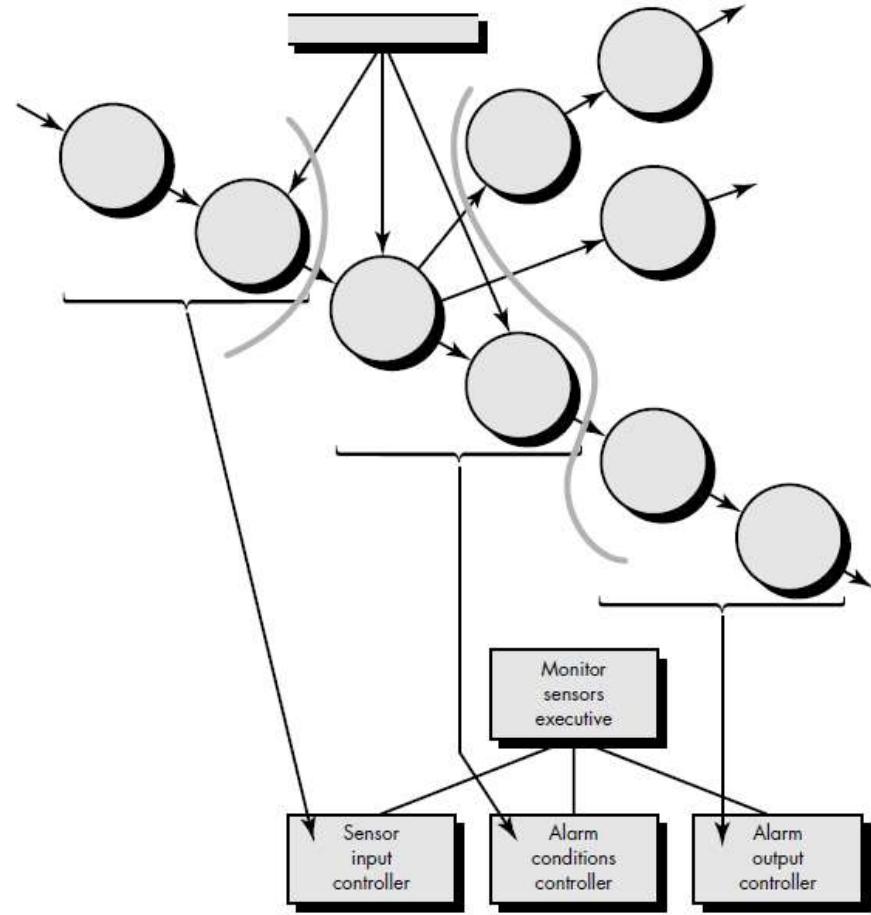
## ***DFD Level 2 dari Proses “Monitor Sensor”***



## *DFD level 2 dari 'monitor sensor' dengan flow boundary*



# *Iterasi Faktoring pertama untuk “monitor sensor”*

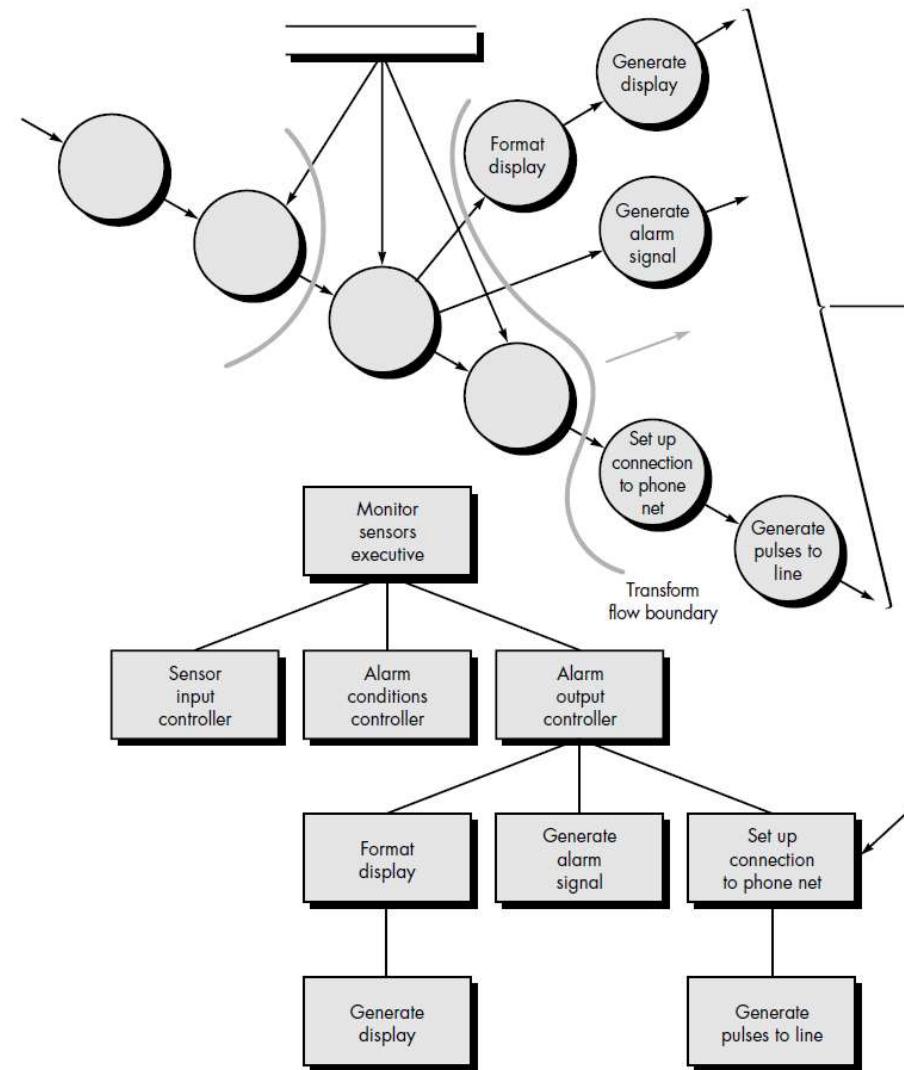


IF2250 Pemodelan Terstruktur



KNOWLEDGE & SOFTWARE ENGINEERING

# *Iterasi Faktoring kedua*



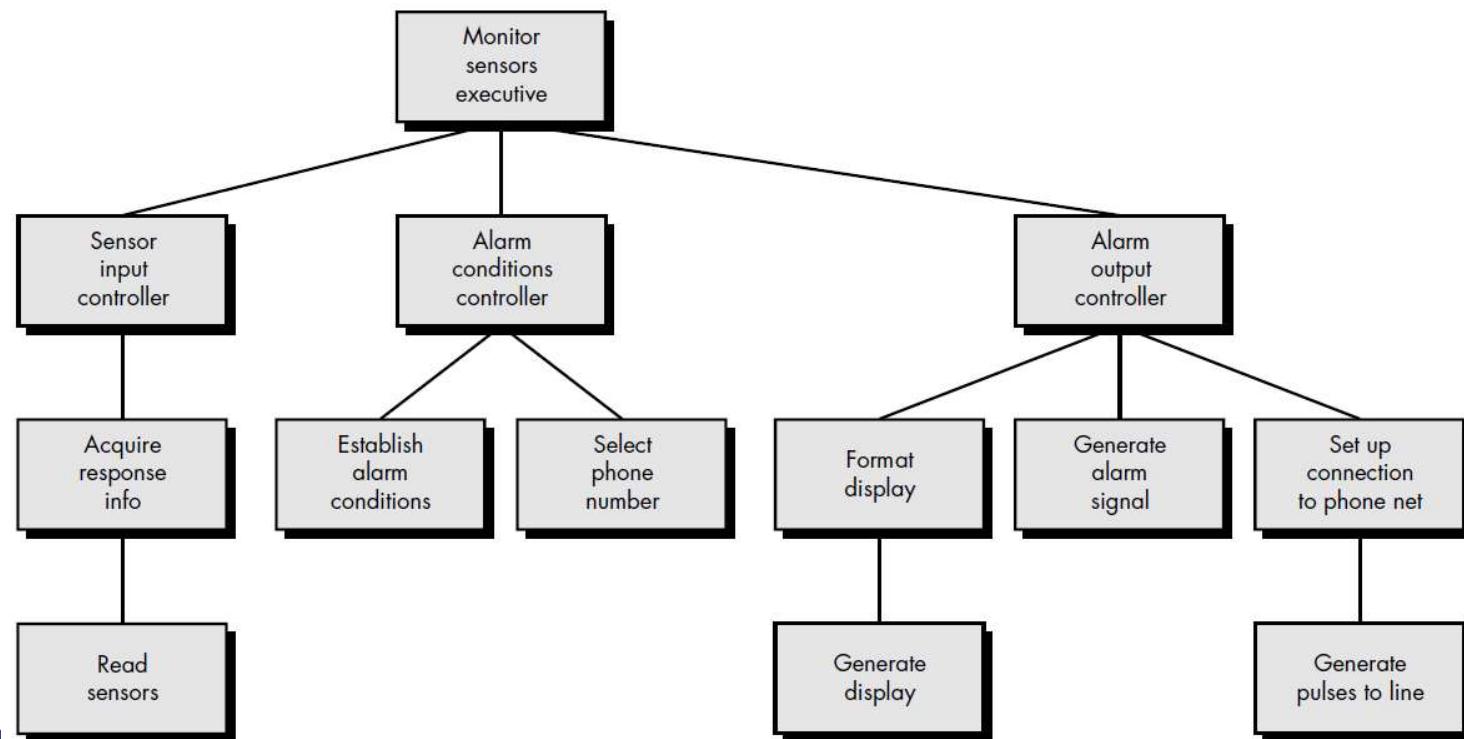
IF2250 Pemodelan Terstruktur



KNOWLEDGE & SOFTWARE ENGINEERING

# *Hasil Structured Chart (SC) untuk SafeHome (Iterasi I)*

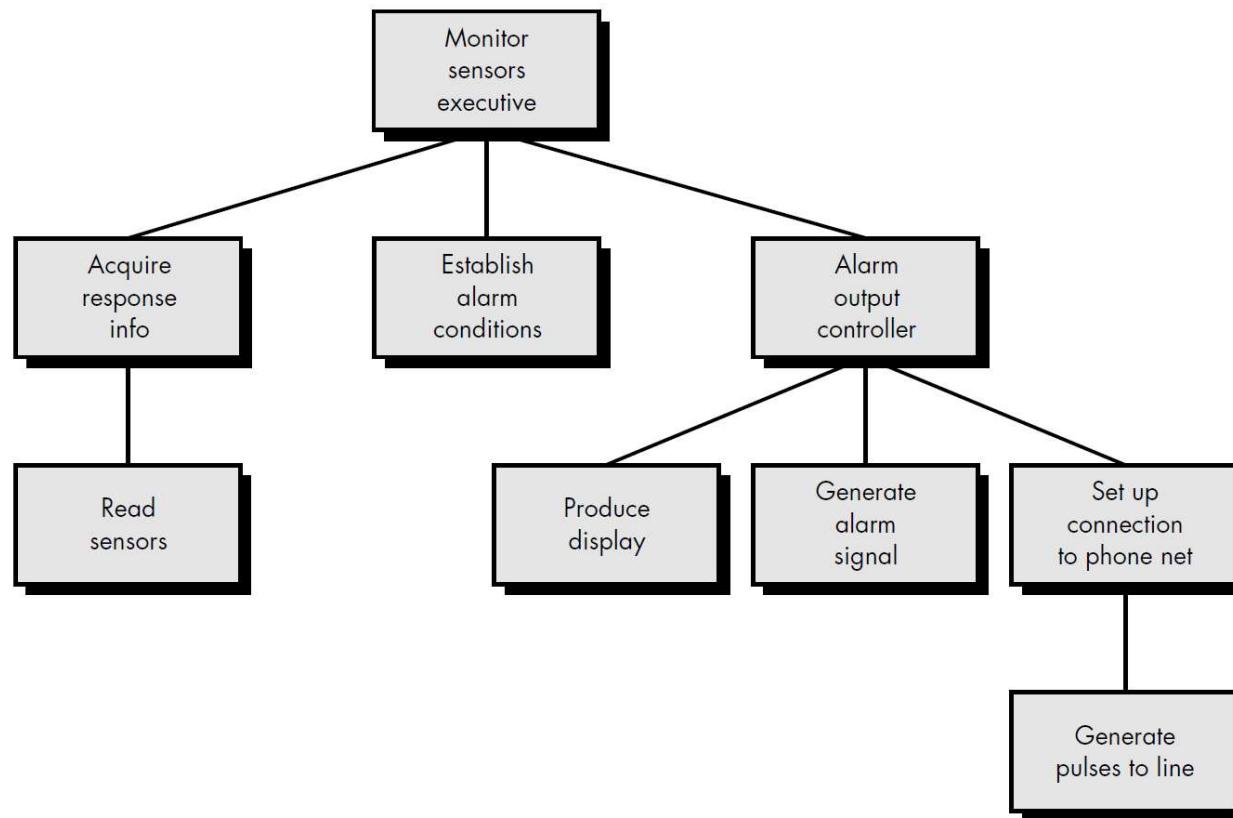
35



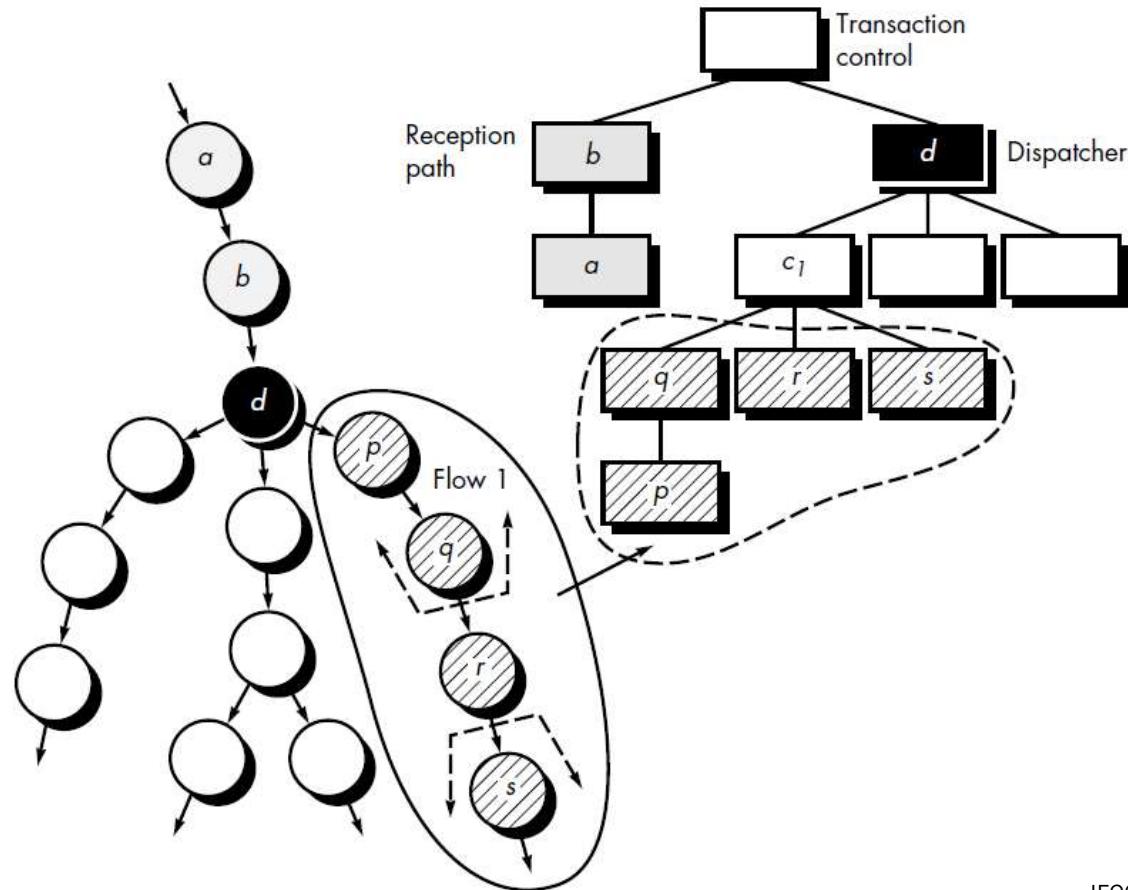
KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 Pemodelan Terstruktur

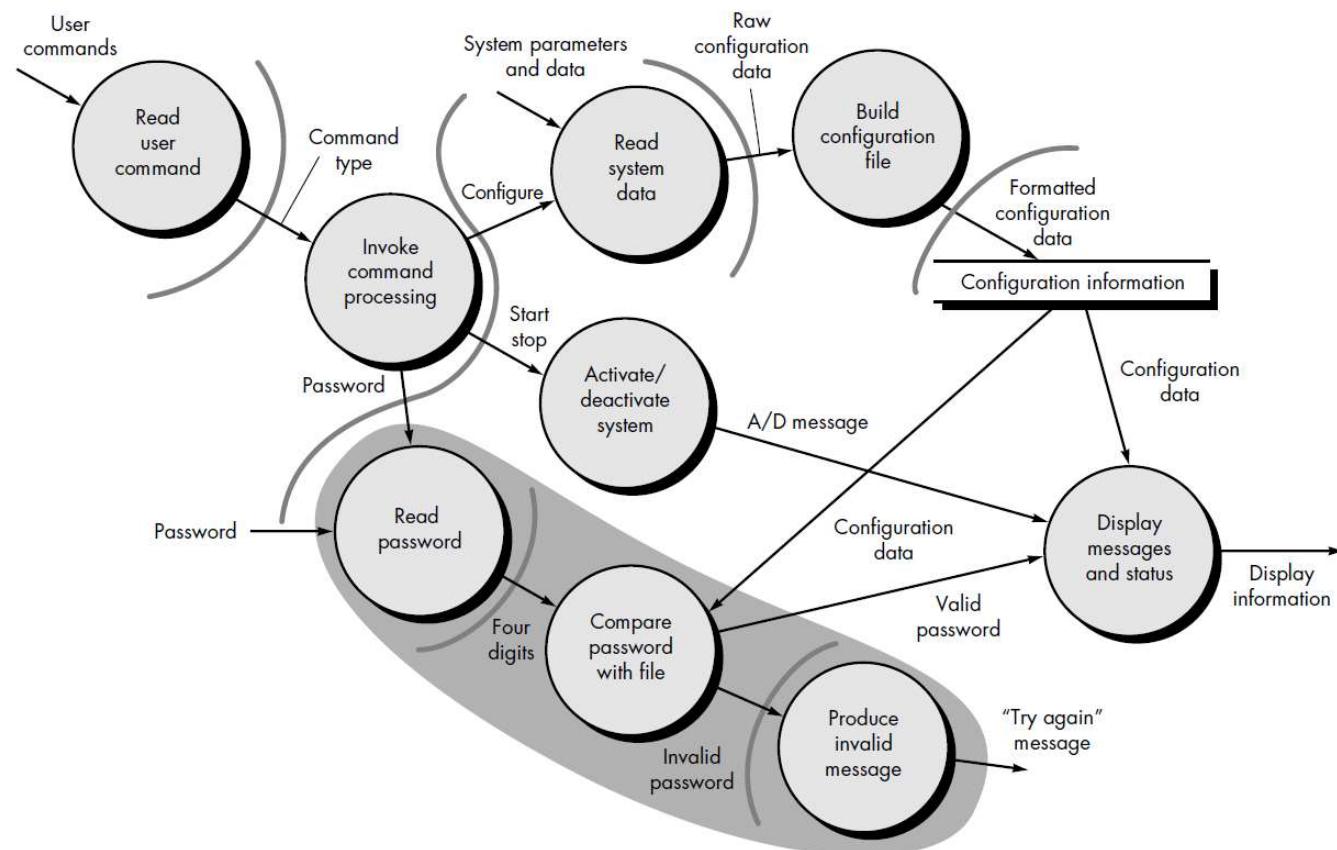
# Hasil Structured Chart (SC) untuk SafeHome (Iterasi 2)



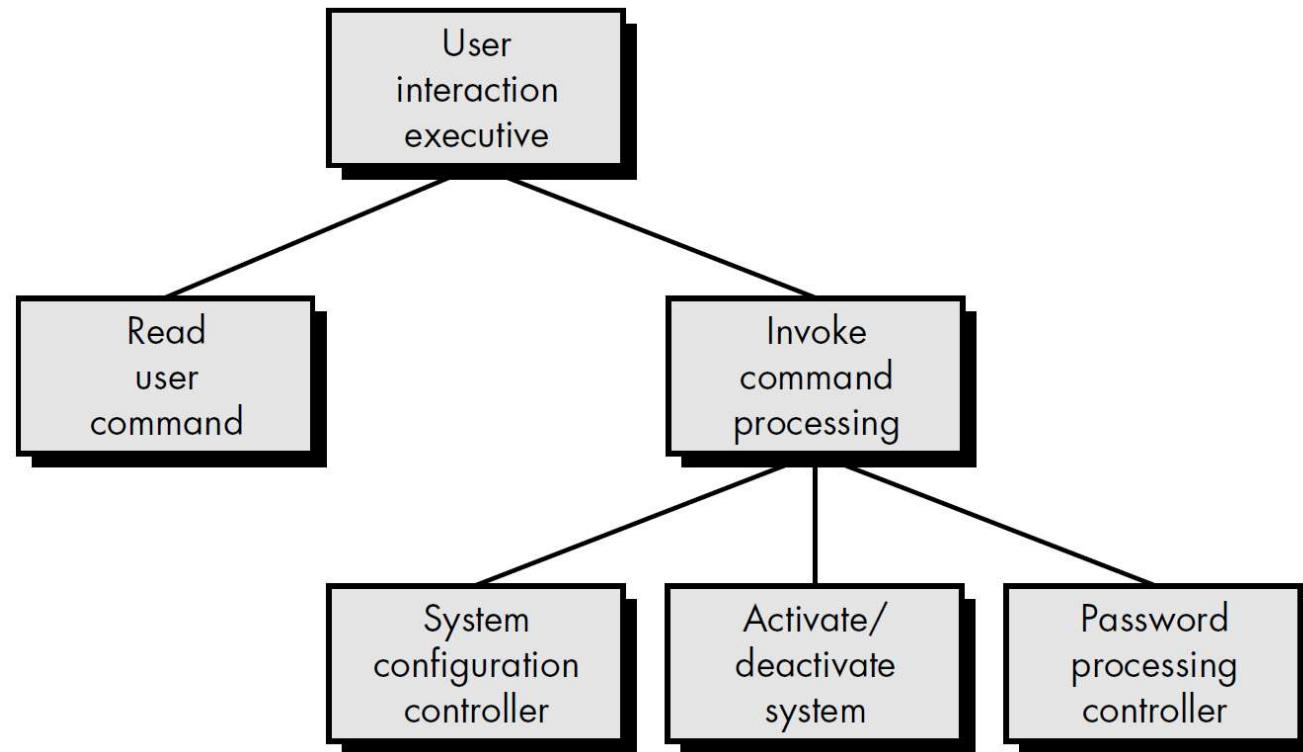
# Pemetaan Transaksi



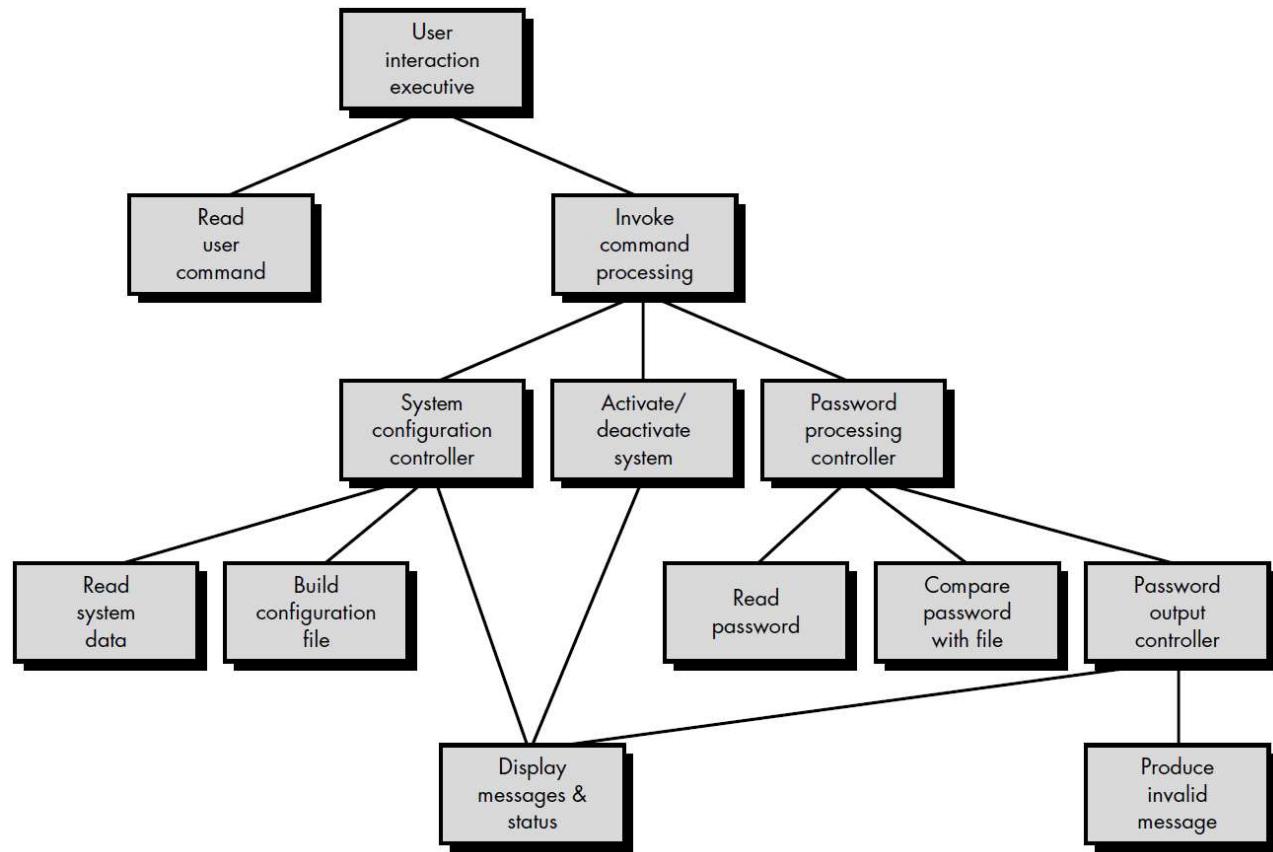
# *DFD untuk proses “User Interaction System”*



# *SC untuk User Interaction System (Iterasi I)*



# *SC untuk User Interaction System (Iterasi 2)*



# *Partisi Arsitektur*



KNOWLEDGE & SOFTWARE ENGINEERING



# *Arsitektur dipartisi agar...*

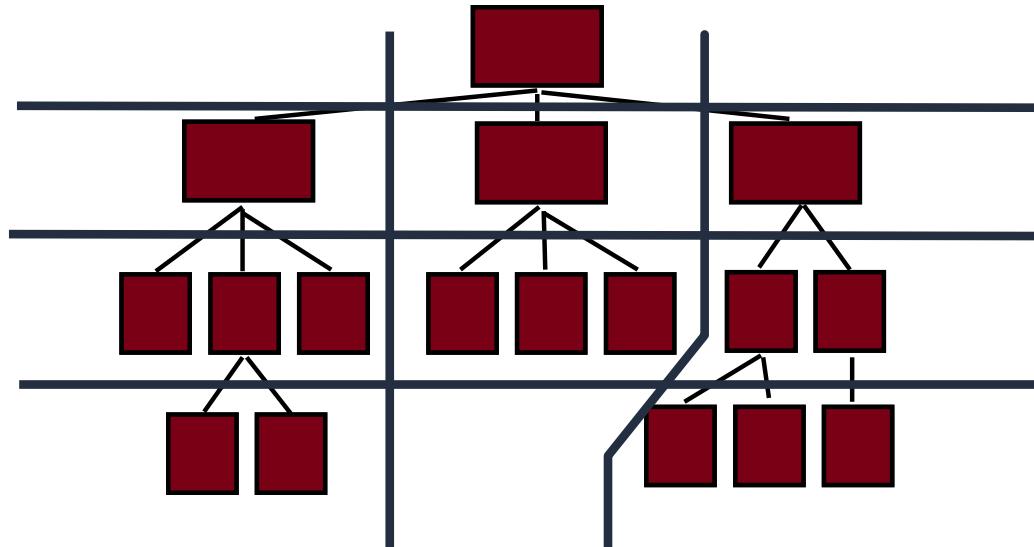
- Software akan lebih mudah diuji
- Software akan lebih mudah dirawat ('maintain')
- Efek propagasi kesalahan berkurang
- Software lebih mudah ditambah kurang modulnya.



KNOWLEDGE & SOFTWARE ENGINEERING

# Partisi Arsitektur

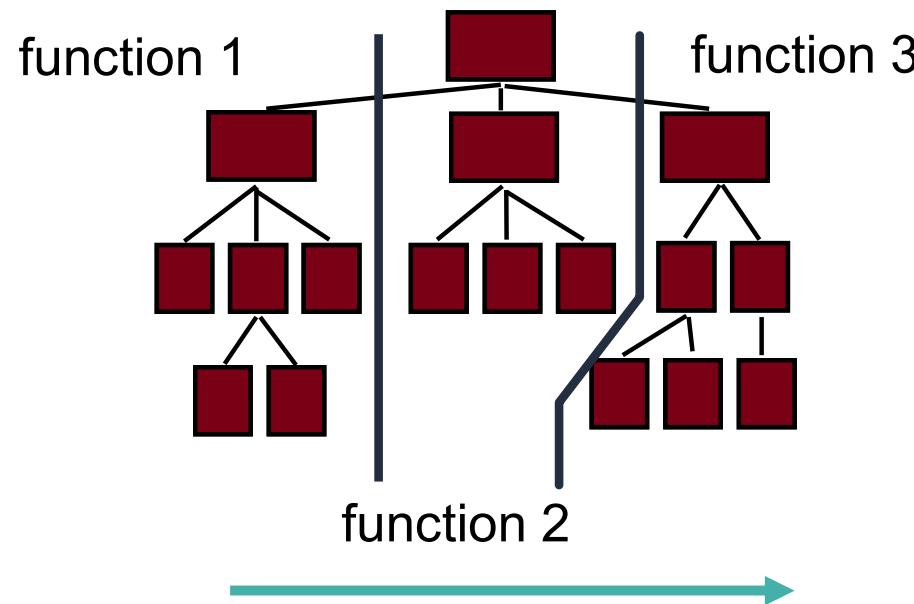
- Partisi ‘horizontal’ dan ‘vertikal’



KNOWLEDGE & SOFTWARE ENGINEERING

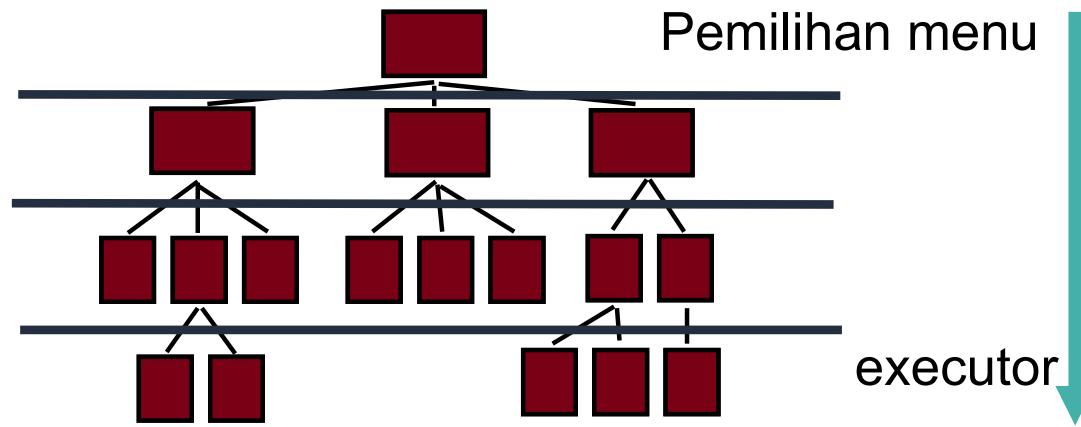
# *Partisi arah Horizontal (horizontal partitioning)*

- 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)



# 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

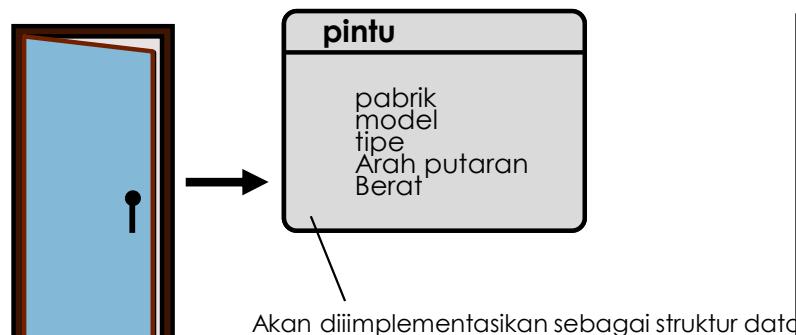
# Abstraksi (lanjutan)

- 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

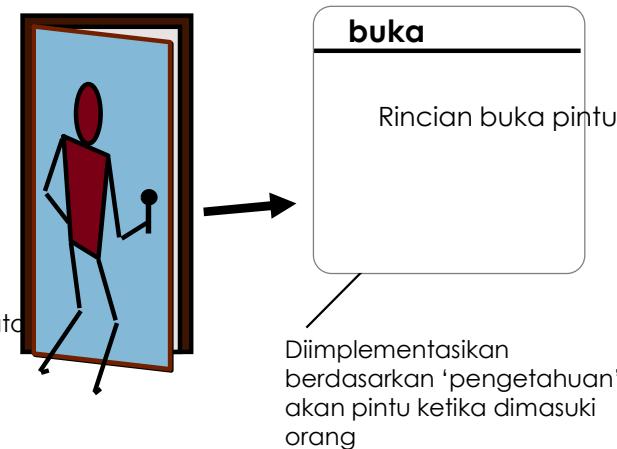
# Abstraksi (lanjutan)

- 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



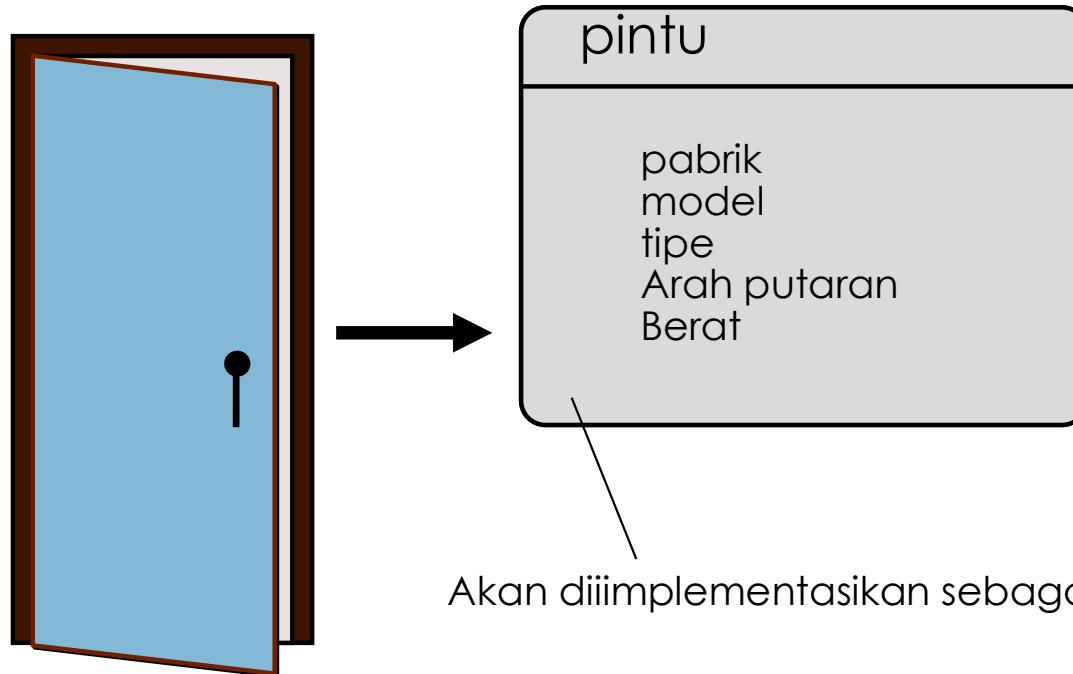
Abstraksi Prosedural



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 Pemodelan Terstruktur

# Abstraksi Data



KNOWLEDGE & SOFTWARE ENGINEERING

# Abstraksi Prosedural



buka

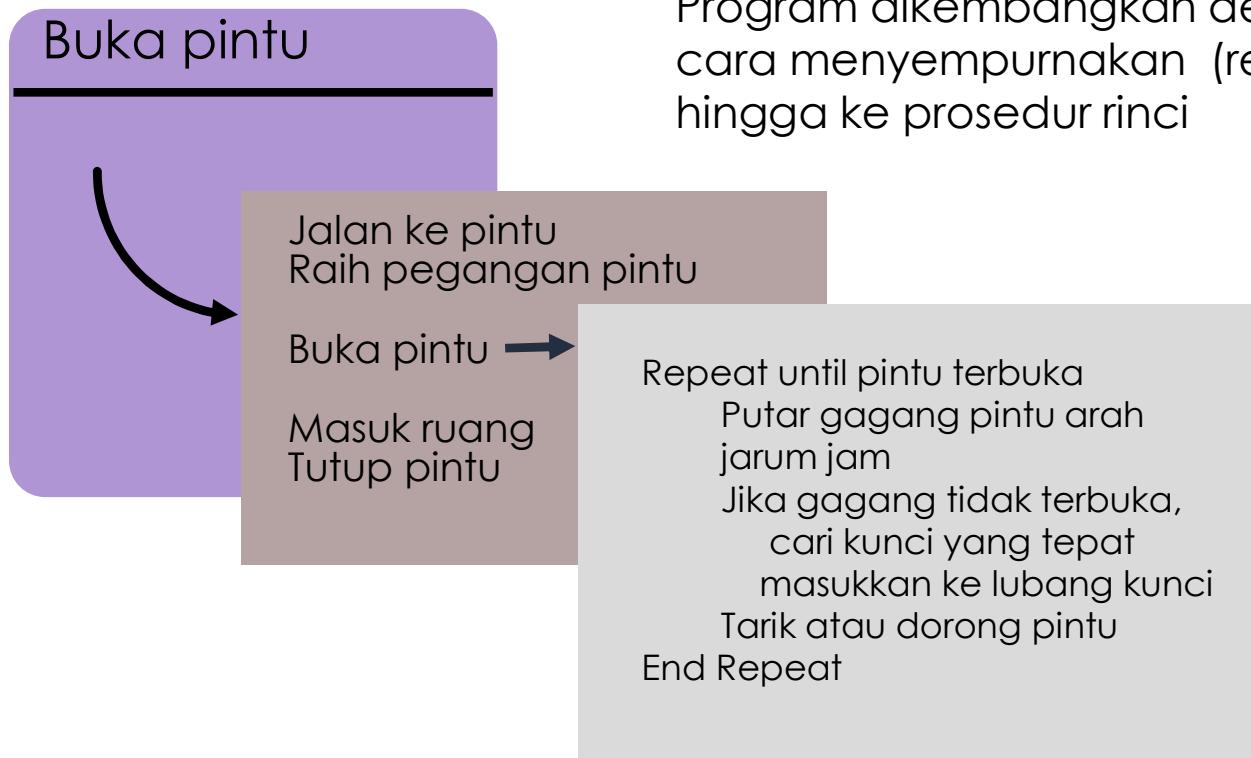
Rincian buka pintu

Dilakukan berdasarkan  
'pengetahuan' akan pintu  
ketika dimasuki orang



KNOWLEDGE & SOFTWARE ENGINEERING

# *Stepwise Refinement*



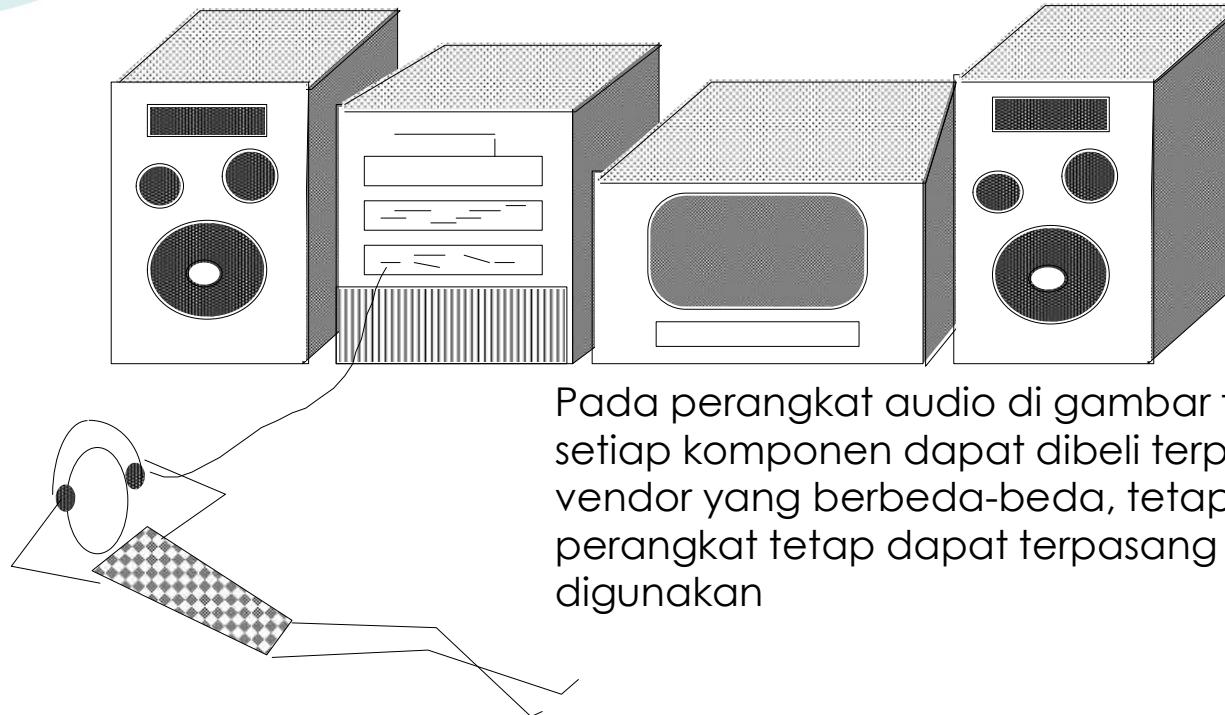
# *Modularitas*



KNOWLEDGE & SOFTWARE ENGINEERING



# *perancangan Modular*



Pada perangkat audio di gambar tersebut, setiap komponen dapat dibeli terpisah, dari vendor yang berbeda-beda, tetapi setiap perangkat tetap dapat terpasang dan dapat digunakan

# 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



KNOWLEDGE & SOFTWARE ENGINEERING

# Modul vs. Biaya

- Misalnya
  - Fungsi  $C(x)$  adalah mendefinisikan besar kompleksitas suatu masalah  $x$  dan fungsi  $E(x)$  adalah usaha (effort) untuk menyelesaikan masalah  $x$
  - $P_1$  dan  $P_2$  adalah program  $P_1$  dan  $P_2$

**Jika  $C(p_1) > C(p_2)$  maka  $E(p_1) > E(p_2)$**  → Artinya butuh waktu lebih lama untuk program dengan kompleksitas yang lebih tinggi

Hasil Eksperimen dengan penyelesaian dunia nyata  **$C(p_1 + p_2) > C(p_1) + C(p_2)$**

Implikasinya:

**$E(p_1 + p_2) > E(p_1) + E(p_2)$**  → Artinya: kompleksitas masalah dari hasil gabungan  $p_1$  dan  $p_2$  adalah lebih besar dibandingkan dengan total pemecahan masalah  $p_1$  dan  $p_2$  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**

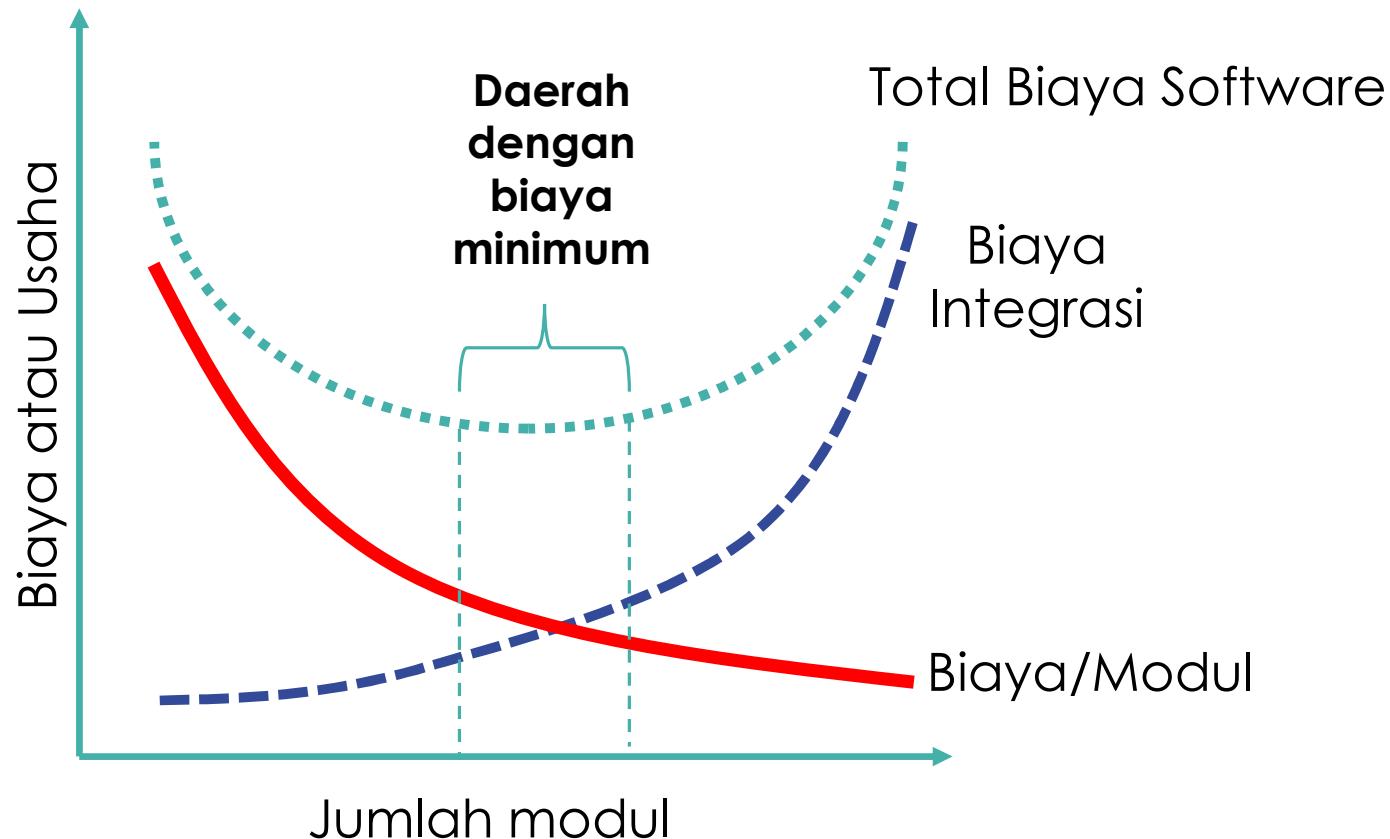
ternyata ada **biaya integrasi** yang harus dilakukan untuk menyatukan modul tadi menjadi program utuh

Makin banyak integrasi harus dilakukan, maka usaha akan makin besar



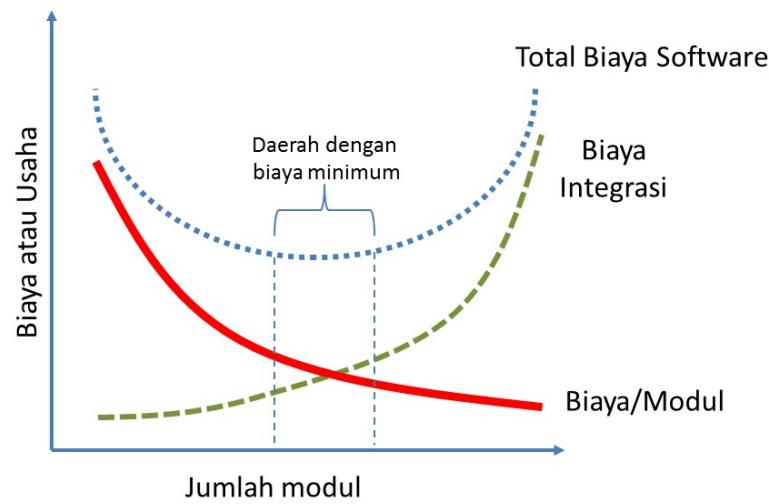
KNOWLEDGE & SOFTWARE ENGINEERING

# *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**

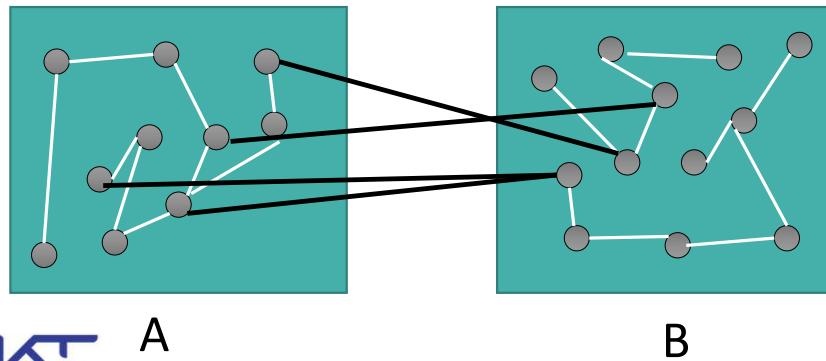
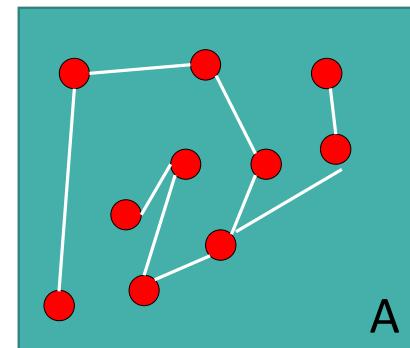
- **Perubahan** kecil dimasa depan pada satu individual modul akan memberikan **dampak** (side-effect) yang **minimal**

- **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)”

# *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:

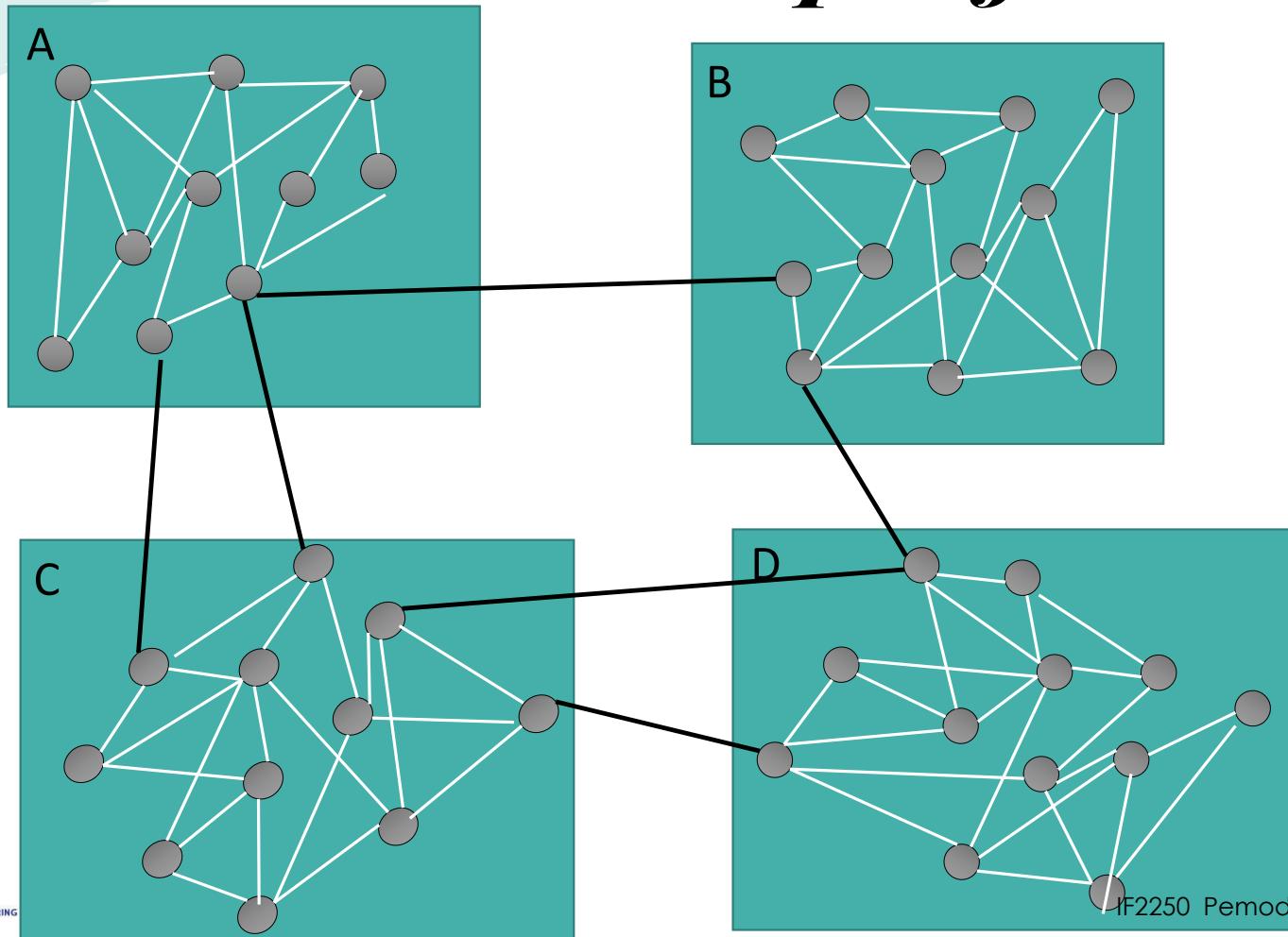
“**Rancangan modul D lebih baik daripada Rancangan Modul C**”



KNOWLEDGE & SOFTWARE ENGINEERING

# *High Cohesion / Low Coupling*

66

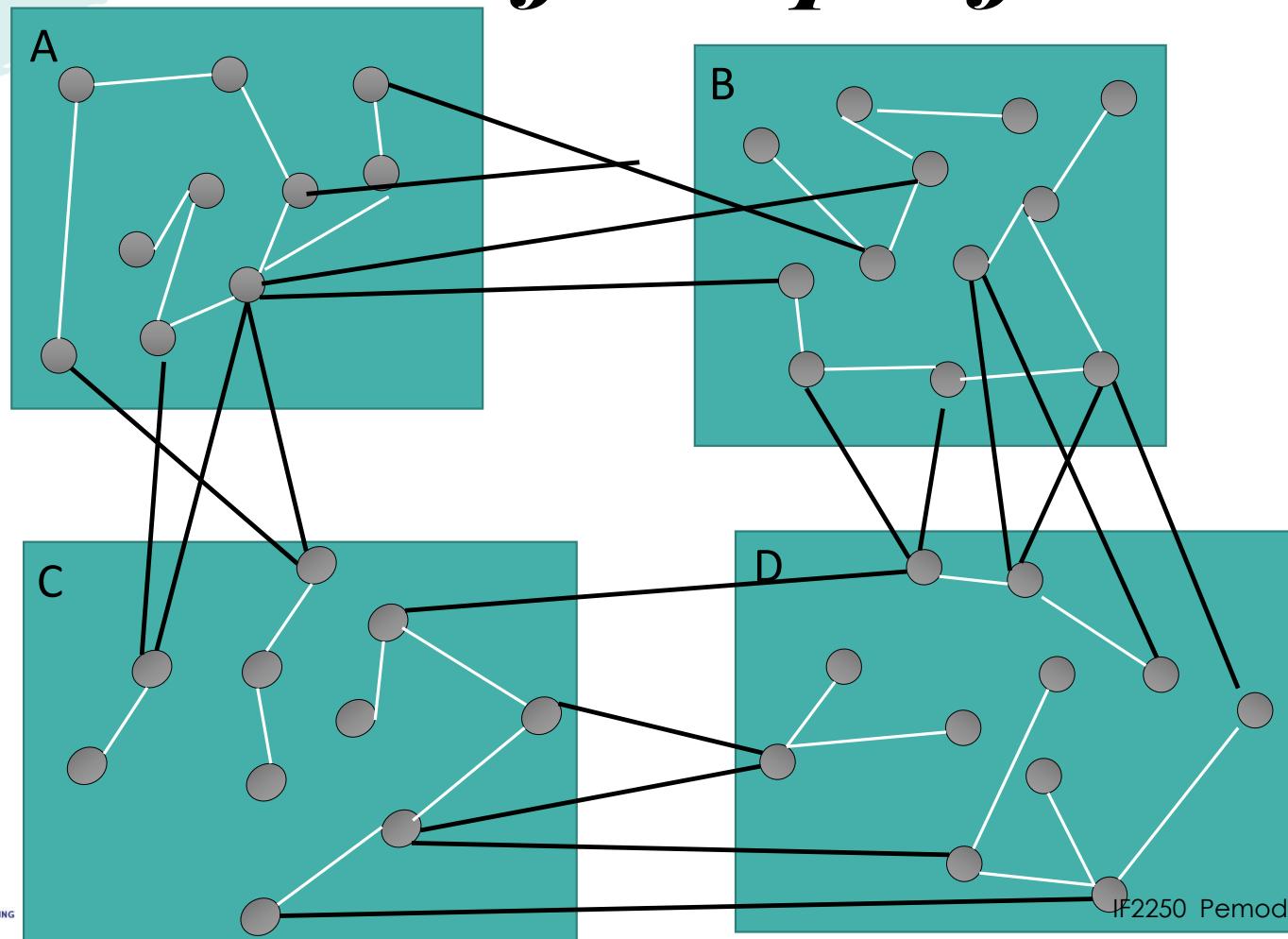


KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 Pemodelan Terstruktur

# *Low Cohesion /High Coupling*

67



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 Pemodelan Terstruktur

# *Ada berbagai macam bentuk coupling/cohesion*

68

- Coincidental cohesion (worst)
  - Logical cohesion
  - Temporal cohesion
  - Procedural cohesion
  - Communicational/informational cohesion
  - Sequential cohesion
  - Functional cohesion (best)
- 
- Content coupling (high)
  - Common coupling
  - External coupling
  - Control coupling
  - Stamp coupling
  - Data coupling
  - Message coupling
  - No coupling



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 Pemodelan Terstruktur

# 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 (I)*

- 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



KNOWLEDGE & SOFTWARE ENGINEERING

# 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);
}
```

# *Bentuk Coupling (4)*

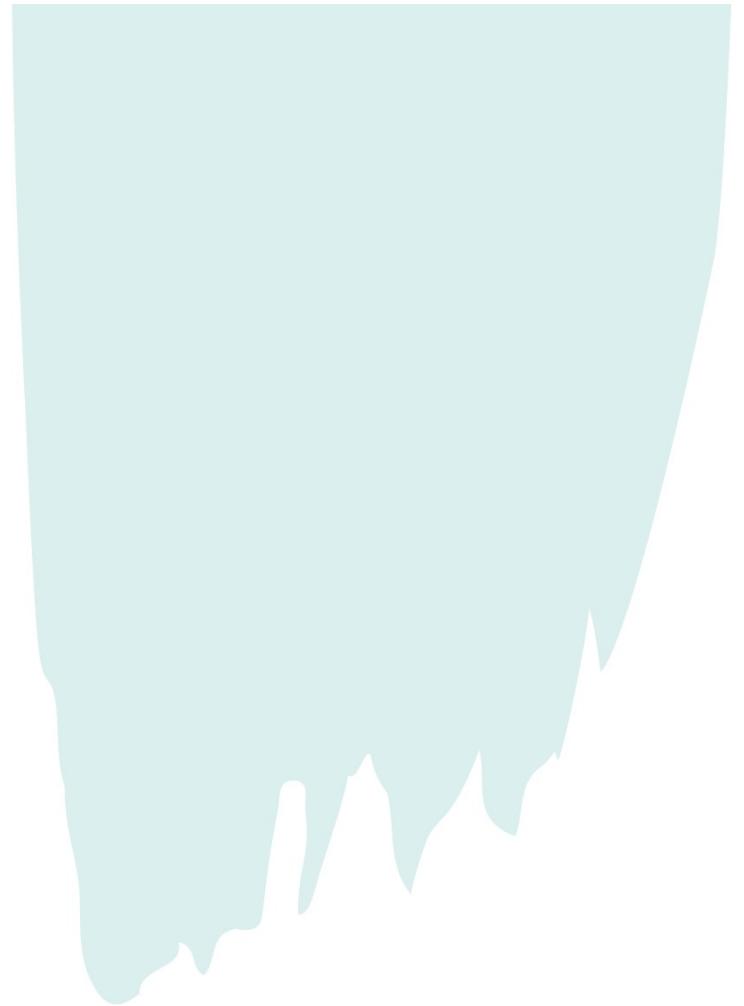
- Pemanggilan prosedur/fungsi dengan data parameter biasa
  - Ini bentuk coupling yang ‘boleh’ atau ‘ideal’
  - Interaksi antar prosedur/fungsi dilakukan dengan jumlah parameter yang seminimal mungkin
- Memindahkan data stream dari satu komponen ke komponen lain
  - Misalnya keluaran suatu prosedur direkam 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, inisialisasi dilakukan bersamaan saat **penciptaan objek**.
  - Metode constructor harus dieksekusi untuk melakukan proses inisialisasi dari suatu objek.
  - Konstruktor dibuat sebagai bagian dari suatu kelas dan memiliki tanggung jawab yang khusus.

## 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.

## 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:

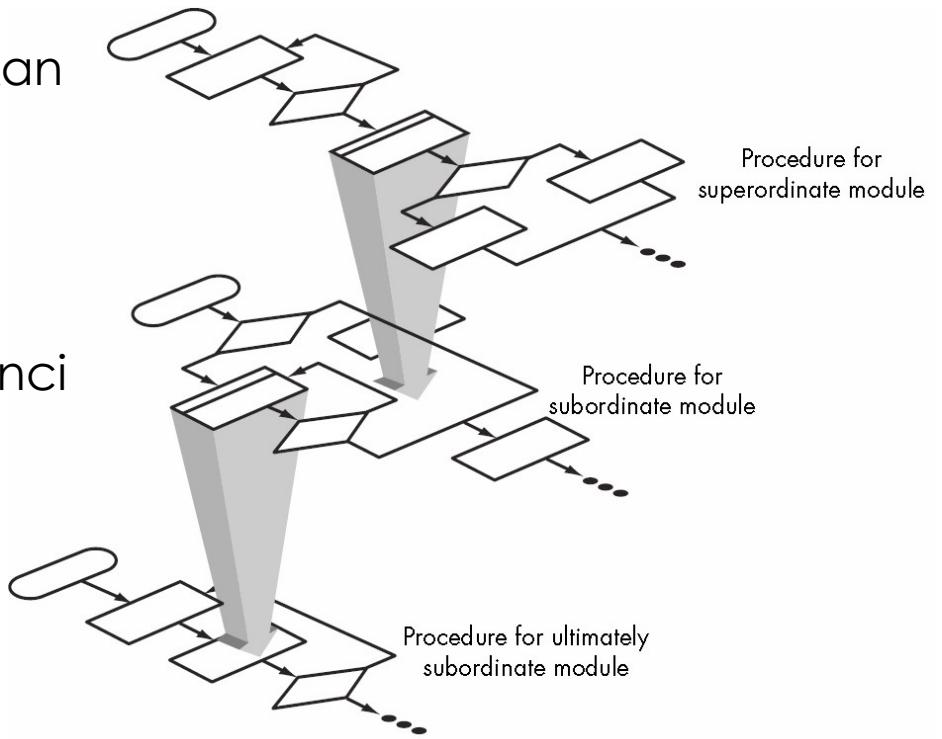
```
HitungRataRata();  
CetakHasil();  
InputTransaksi();
```

# 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

# 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

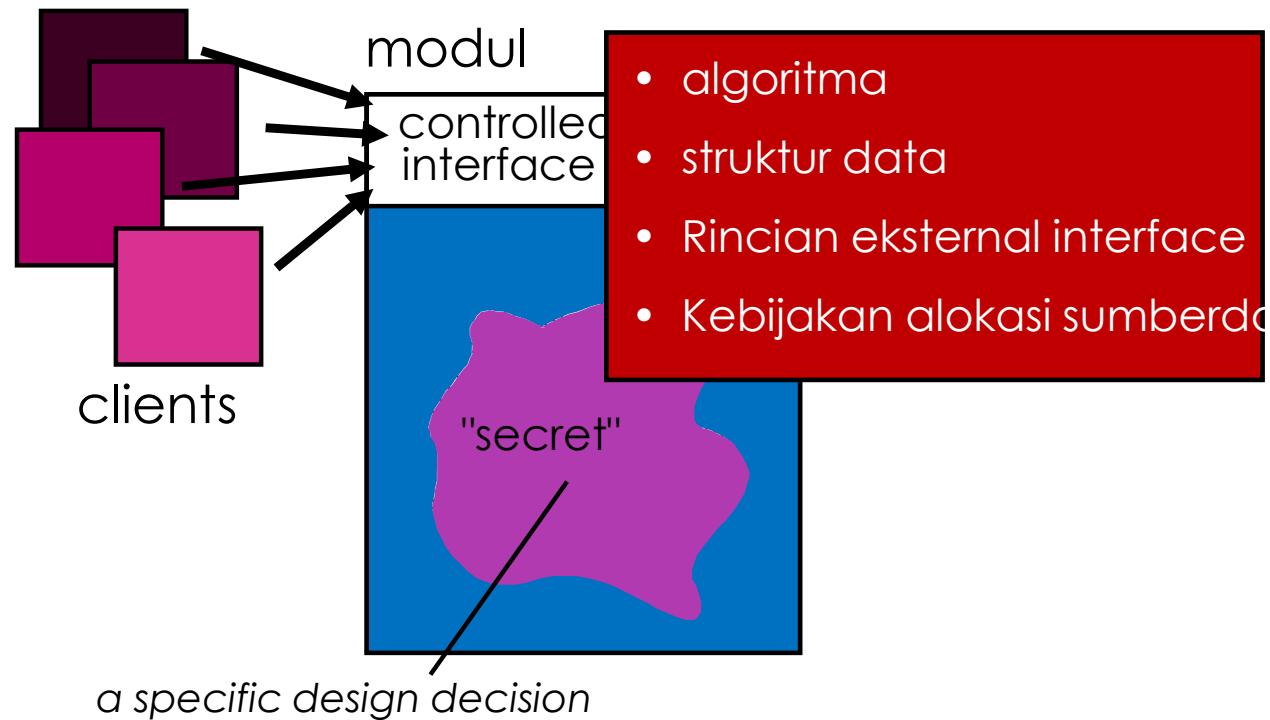
# *Information Hiding*

DIKENAL JUGA SEBAGAI

- DATA HIDING
- ENCAPSULATION



# 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**

# *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)

## 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.



KNOWLEDGE & SOFTWARE ENGINEERING

# *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** sesingkatnya .

# 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



IF2250 Pemodelan Terstruktur

# 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 konstruksinya** 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



KNOWLEDGE & SOFTWARE ENGINEERING

# ***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.

# *Refactoring 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

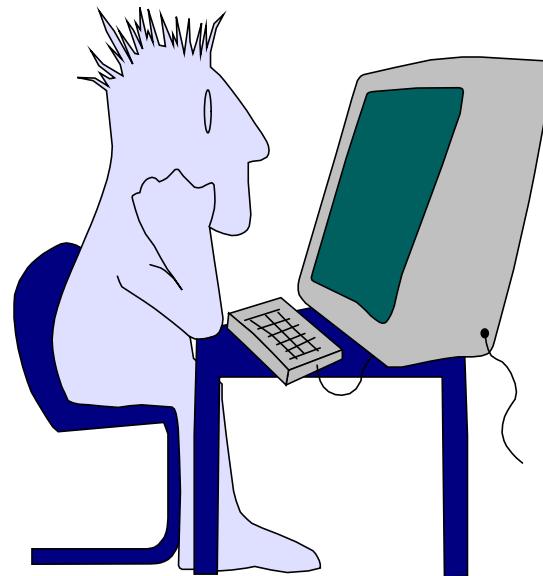
# *Perancangan Antarmuka*



IF2250 Pemodelan Terstruktur

# *Perancangan Antarmuka (Interface Design)*

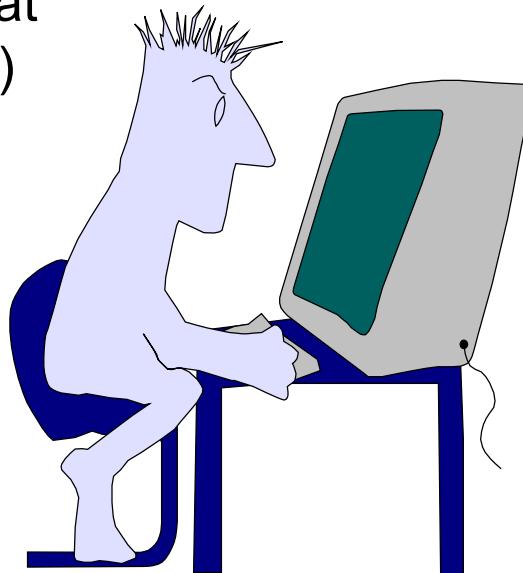
**Mudah dipelajari?  
Mudah digunakan?  
Mudah di mengerti**



KNOWLEDGE & SOFTWARE ENGINEERING

# *Kesalahan umum dalam perancangan antarmuka*

- Tidak konsisten
- Kurangi objek yang harus diingat
- Tidak ada panduan (help menu)
- Respon lambat
- Tidak mudah digunakan



KNOWLEDGE & SOFTWARE ENGINEERING

# *Golden Rules*

- Pengguna harus menjadi pengendali
- Kurangi hal-hal yang mengharuskan pengguna harus mengingat-ingat (less memory load)
- Buat tampilan yang konsisten

# *Pengguna sebagai Pengendali*

- Buat interaksi yang tidak memaksa pengguna harus melakukan aksi yang tidak perlu atau yang tidak diinginkan
- Interaksi dibuat seflexibel mungkin
- Interaksi pengguna dapat di interrupt (interruptible) atau dibatalkan (undoable)
- Interaksi dapat dibuat lebih fleksibel ketika kemampuannya makin meningkat, dan memungkinkan interaksi dapat dicustomized
- Hindari pengguna biasa untuk mengerti masalah-masalah tekniks
- perancangan dibuat untuk memungkinkan interaksi langsung dengan objek yang ada di layar

# ***Kurangi Objek yang Harus diingat***

- Kurangi kebutuhan pengguna untuk mengingat
  - Short-term memory reduction
- Buat perancangan yang praktis/default
- Buat shortcut yang intuitif
- Tataletak visual dari interaksi harus berdasarkan metafora dunia nyata
- Informasi ditampilkan secara progresif



KNOWLEDGE & SOFTWARE ENGINEERING

## *Buat antarmuka yang konsisten*

- Perancangan antarmuka perlu memperhatikan konteks yang dilakukan oleh Pengguna
- Konsistensi sepanjang aplikasi dijalankan
- Jika model interaksi sebelumnya telah membentuk apa yang diinginkan pengguna, maka hati-hati dalam melakukan perubahan, kecuali jika ada alasan yang kuat



KNOWLEDGE & SOFTWARE ENGINEERING

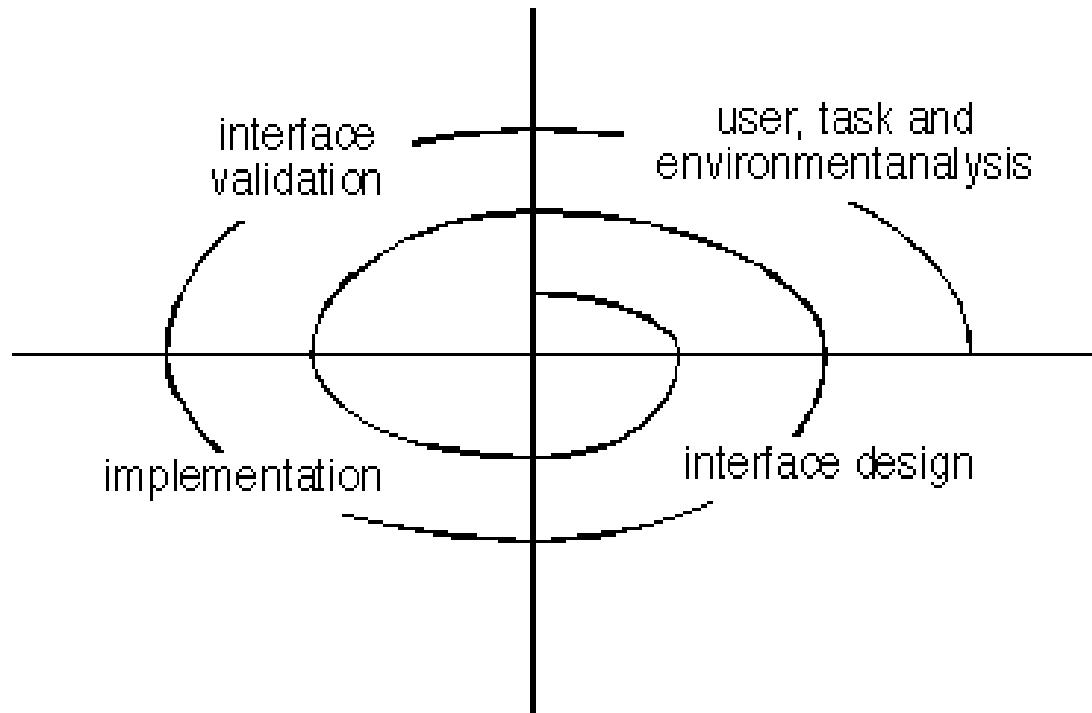
# ***Model Perancangan Interaksi Pengguna***

## ***User Interface Design Models***

- Persepsi Sistem
  - Melihat sistem dari sudut pandang pengguna (end-user)
- Model Pengguna (User model)
  - Buat profile dari setiap end-user dari sistem
- System image —
  - Bentuk ‘presentasi’ dari sistem dengan interface yang lengkap
- Model perancangan (Design model)
  - Representasi software dalam bentuk perancangan data, arsitektural, interface and procedural

# *Proses Perancangan Antarmuka Pengguna*

107



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 Pemodelan Terstruktur

# *Analisis dan Pemodelan Task*

- Semua task yang harus dilakukan harus terdefinisi dan terklasifikasi dengan jelas
- Objek (yang akan dimanipulasi) dan aksi (fungsi yang dilakukan pada objek) harus jelas teridentifikasi untuk setiap task
- Task harus di perbaiki secara iteratif hingga terdefinisi lengkap



KNOWLEDGE & SOFTWARE ENGINEERING

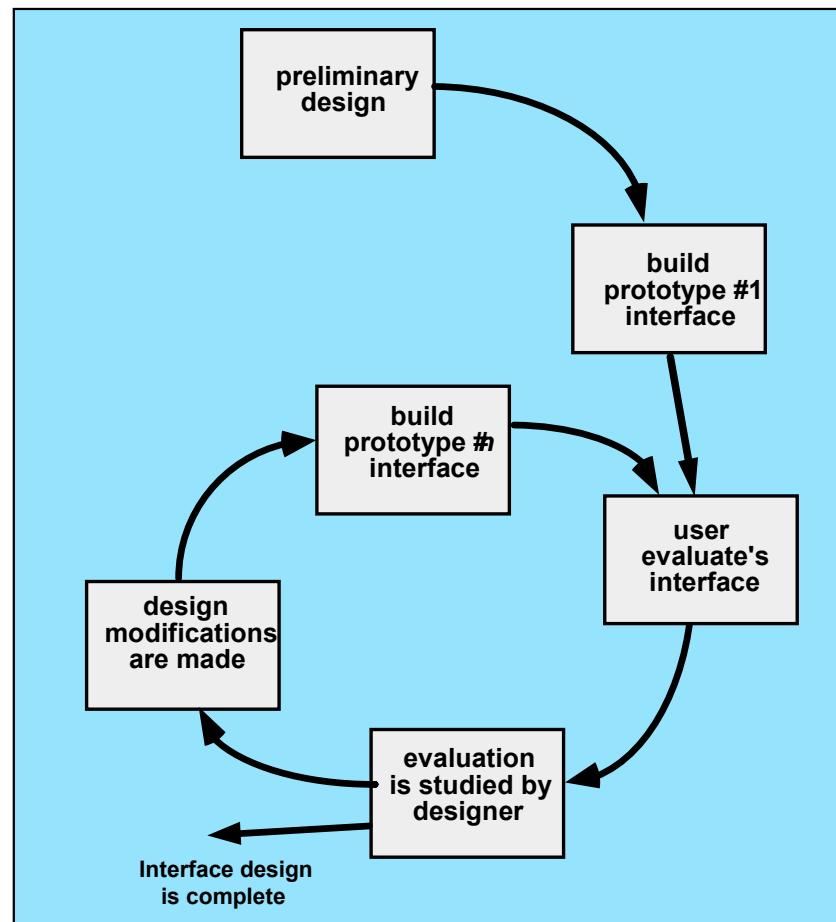
# *Aktivitas Perancangan Antarmuka*

1. Buat tujuan (goal) untuk setiap task
2. Petakan tujuan tadi menjadi sekumpulan aksi
3. Tentukan urutan aksi untuk setiap task/subtask
  - Disebut juga sebagai user-scenario (skenario pengguna) yang akan dieksekusi di level antarmuka
4. Tentukan 'state' dari sistem
  - Apa bentuk interface yang harus ditampilkan jika suatu user-scenario di jalankan
5. Definisikan mekanisme kendali
  - Objek/aksi yang harus ada saat pengguna mengubah 'state' dari sistem
6. Tunjukkan bagaimana mekanisme kendali akan berefek pada state dari sistem
7. Berikan indikasi bagaimana pengguna mengartikan 'state' dari sistem melalui interface

# *Masalah pada Perancangan Antarmuka*

- Waktu respon sistem
  - Ketika pengguna memberikan suatu aksi dan waktu sistem memberikan respon
- Fasilitas panduan pengguna
  - Terintegrasi, context-sensitive help
- Penanganan terjadinya kesalahan
  - Pesan tidak bersifat ‘mengadili’, masalah dijelaskan dengan rinci dan juga berikan solusinya
- Nama Istilah Perintah
  - Penamaan menggunakan istilah yang dapat dimengerti pengguna, dan termasuk penggunaan singkatan yang konsisten.

# Daur Evaluasi Perancangan Antarmuka



# *Kriteria mengevaluasi perancangan antarmuka*

- Spesifikasi antarmuka yang panjang dan kompleks memberikan indikasi kompleksitas yang akan dipelajari oleh pengguna
- Jumlah task dari pengguna dan jumlah rata-rata aksi/task memberikan indikasi waktu interaksi dan juga efisiensi dari sistem
- Jumlah task, aksi dan state dari sistem dalam perancangan memberikan indikasi jumlah hal yang harus diingat oleh pengguna (memory load dari pengguna akan besar)
- Gaya antarmuka, fasilitas panduan, dan protokol penanganan error memberikan indikasi kompleksitas sistem dan tingkat acceptance dari user.

Tim Pengajar IF2250

IF2250 – Rekayasa Perangkat Lunak  
**Scenario-based Modeling**

SEMESTER II TAHUN AJARAN 2022/2023



KNOWLEDGE & SOFTWARE ENGINEERING

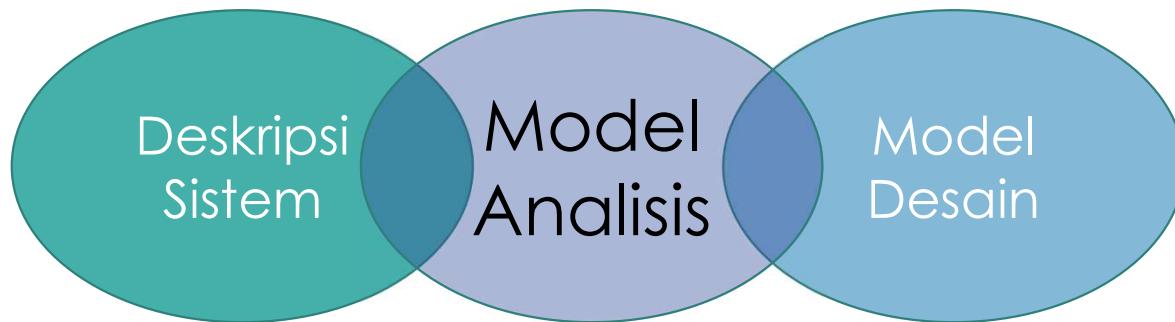


# *Model Analysis*

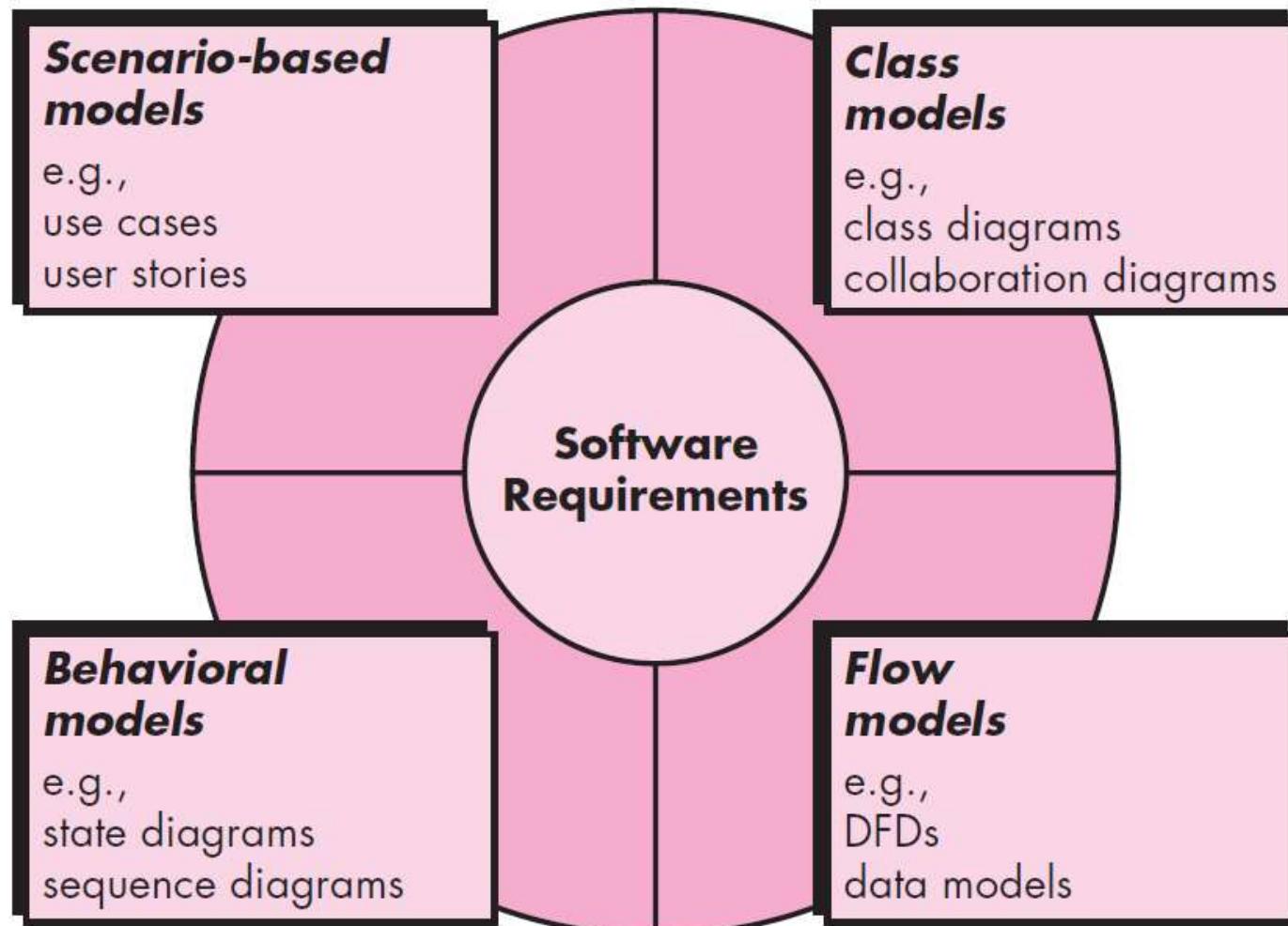


IF2250 - Scenario Modeling

# ***Model Analisis adalah “jembatan” antara Deskripsi Sistem dan Model Desain***



- Model harus difokuskan pada kebutuhan yang ada dalam domain masalah/bisnis
- Tiap elemen digunakan untuk mengerti kebutuhan sistem
- Model dibuat sederhana



# *Model Analisis Kebutuhan PL*

- Model Berorientasi Aliran (**Flow-Oriented Models**)
  - Contoh: DFD
- Model Perilaku (**Behavioral Models**)
  - Contoh: *State Transition Diagram* atau *State Diagram*
- Model Berbasis Skenario (**Scenario-Based Models**)
  - Fungsional – pemrosesan cerita untuk suatu fungsi perangkat lunak
  - Use-Case – deskripsi hubungan interaksi antara aktor dan sistem
- Model Berbasis Kelas (**Class-Based Models**)
  - Dikembangkan berdasarkan entitas yang ditemukan saat melakukan pemodelan skenario dan/atau fungsional

# ***Pemodelan Berbasis Skenario (Scenario Based Modeling)***



IF2250 - Scenario Modeling

# *Scenario-based Modeling*

- Apa yang dimodelkan?
  - Cara pengguna **berinteraksi** dengan sistem
- Seperti apa modelnya?
  - Diagram use-case
  - Diagram aktivitas
  - Diagram aktivitas dalam bentuk *swimlane*
  - Diagram interaksi
- Bagaimana membuat modelnya?

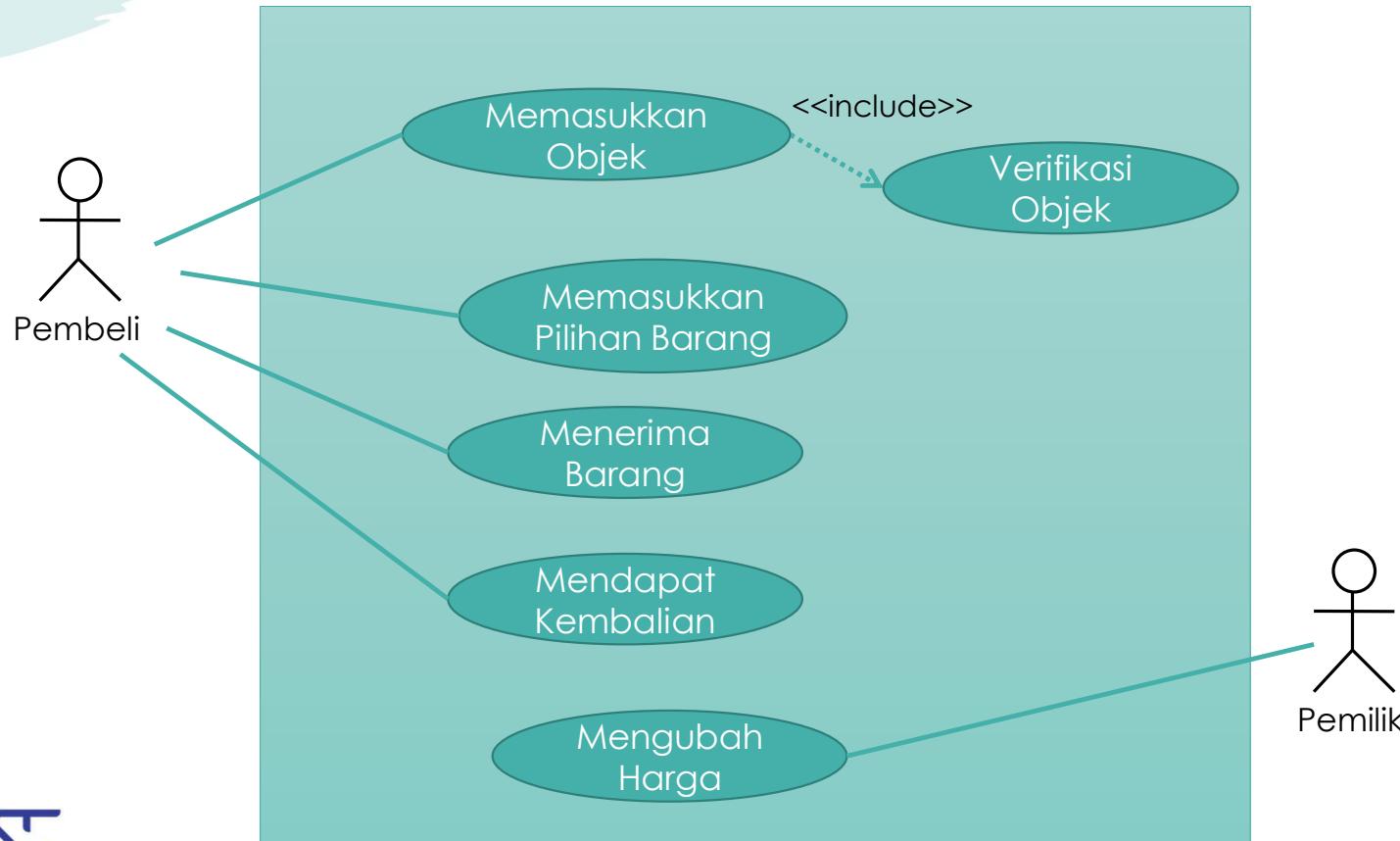
# Diagram Use-case

- Apa itu use-case?
- Apa itu aktor?
- Apa itu skenario?

Ivar Jacobson: “[**Use-cases**] are simply an aid to defining what exists outside the system (**actors**) and what should be performed by the system (**use-cases**).”

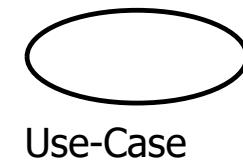
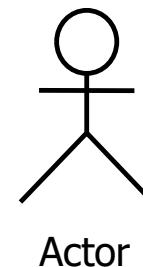
# Use Case untuk VM

ga memperhatikan urutan



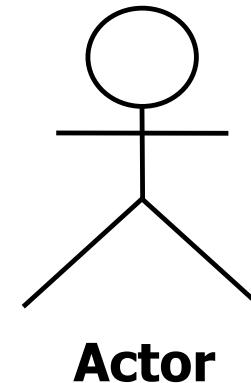
# Konsep Pemodelan use-case

- Aktor mewakili semua yang **berinteraksi** dengan sistem
  - Aktor adalah unsur '**eksternal**'
- Use-case adalah **urutan aksi-aksi** dalam sistem yang melakukan suatu pekerjaan yang memberikan suatu hasil untuk aktor
  - use-case bertindak sebagai **penghubung** antara pengguna dengan pengembang
  - use-case berguna sebagai **alat komunikasi** antara pengguna dan pengembang



# Aktor (Actor)

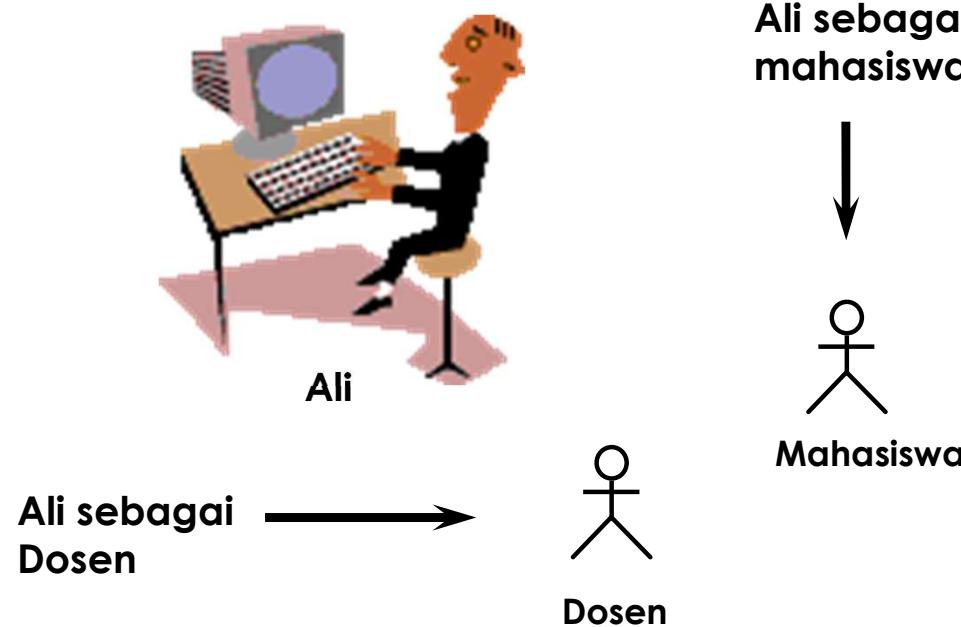
- Aktor bukan bagian dari sistem
  - Mereka berperan sebagai '**pengguna**' dari sistem
- Aktor mungkin secara aktif bertukar informasi dengan sistem
- Aktor mungkin berfungsi pasif sebagai penerima informasi
- Aktor bisa merepresentasikan
  - Manusia,
  - Mesin,
  - Sistem lain



***Aktor mewakili unsur Eksternal***

# *Seorang pengguna mungkin memiliki peran berbeda*

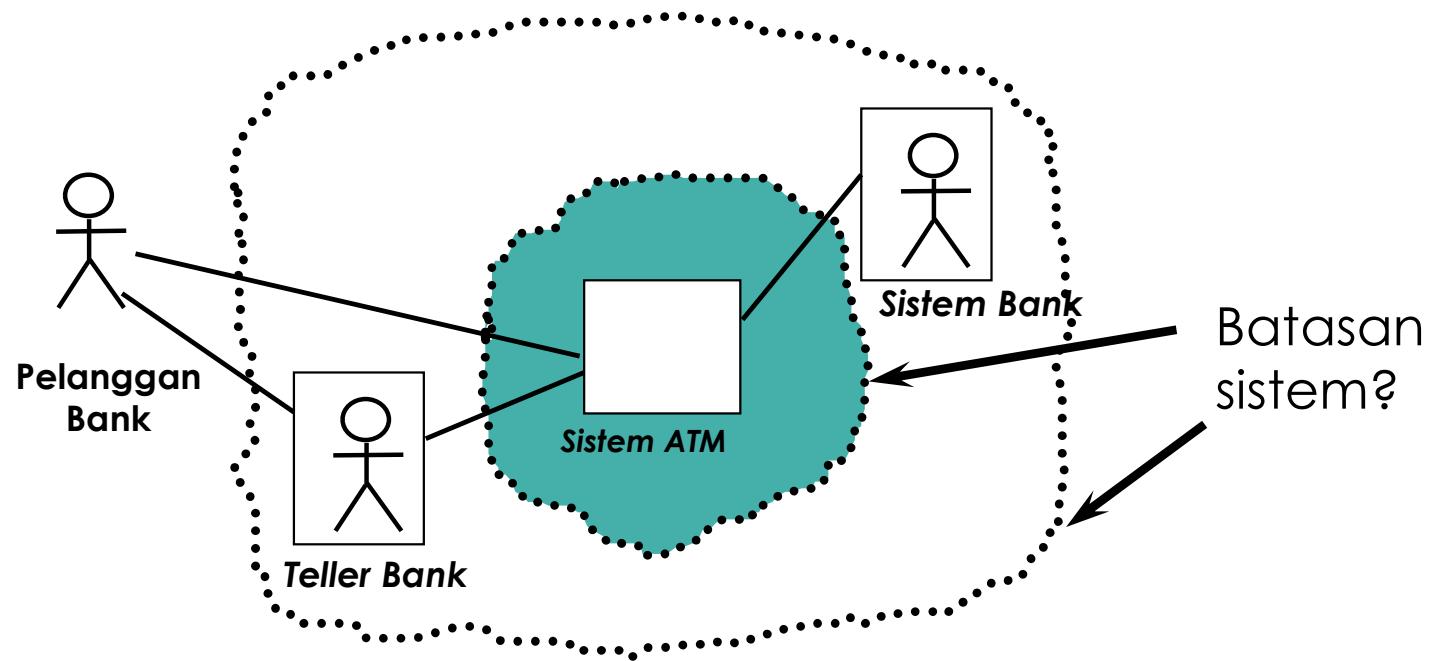
12



Dengan use-case peran ini harus digambarkan berbeda  
walau dalam kenyataannya orangnya mungkin hanya satu

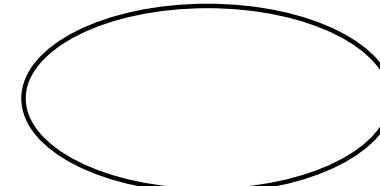
# *Aktor dan Batasan Sistem (System Boundaries)*

13



# Use-Case

- Use-case merepresentasikan **dialog** antara aktor dengan sistem
  - use-case di '**inisiasi**' oleh **aktor** untuk melakukan suatu fungsi tertentu dalam sistem
- Use-case merepresentasikan dialog antara satu atau lebih aktor
  - sistem akan **mengembalikan** suatu **nilai** ke aktor
- Use-case perlu menggambarkan **event** yang lengkap dan memiliki makna bagi sistem
- Use case juga dapat dilihat sebagai tujuan sistem secara umum yang mungkin melibatkan satu atau lebih aktor
- Semua use-case mengarahkan ke semua penggunaan sistem



**Use-Case**

# Penjelasan Use-case

- Gambar use-case (termasuk aktornya) perlu disertai dengan **keterangan** yang akan membantu **menjelaskan** gambaran yang diberikan
  - Keterangan ini tidak perlu panjang lebar, untuk setiap use-case lebih kurang **dua baris** saja.
  - Deskripsi langkah-demi langkah apa yang perlu dikerjakan sistem ketika berinteraksi dengan aktor
- Perlu dijelaskan bagaimana antar use-case **saling terkait**
  - Atau suatu use case yang terlibat pada suatu use-case lain
  - Bagaimana use case itu dilakukan oleh aktor
  - Use case dapat dikelompokkan sebagai paket (package)

# Isi Rincian dari Use-case

- Deskripsi struktur Use-case
  - Menyatakan bahwa instansiasi dari use-case akan memiliki perubahan **transisi antar state** (status)
    - Perubahan ini digambarkan dalam aliran event (*event flow*)
  - Setiap transisi adalah sekumpulan aksi
    - Mungkin bisa makin kompleks, jadi penjelasan sesederhana mungkin
- Deskripsi Use-case bisa dimulai dari **Basic Path**
  - **Alternatif path** diberikan penjelasan pada bagian yang berbeda
- Berikan penjelasan yang mudah dibaca

# *Apa saja isi deskripsi*

- Penjelasan **Kondisi awal**
- Bagaimana dan kapan use-case **dimulai**
  - Urutan kejadian (aliran event)
  - Bagaimana dan kapan use-case selesai
  - Kemungkinan post-kondisi alternatif
  - Jalur eksekusi yang tidak diperbolehkan
  - Jalur alternatif (diambil dari jalur dasar)
- Interaksi sistem dengan aktor
- Penggunaan objek, sumber daya dalam sistem
- Penjelasan apa yang dilakukan oleh sistem

# Deskripsi Formal

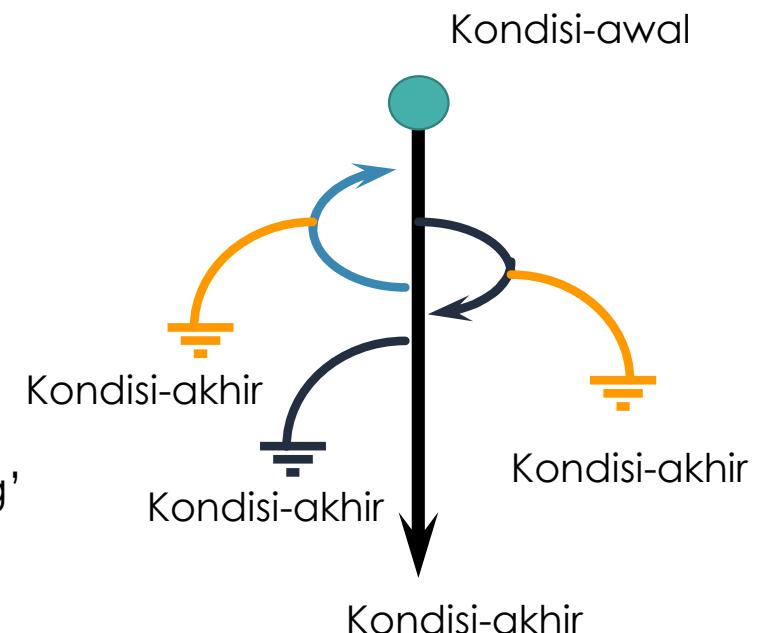
- Untuk jumlah use-case yang **besar** dengan berbagai **alternatif**, maka penulisan teks menjadi tidak praktis, jadi kadang digunakan **diagram**
  - Diagram Statechart**
    - Untuk menggambarkan use-case yang kompleks
    - Berisi penjelasan state dan transisi dalam use-case
  - Diagram Aktivitas**
    - Menggambarkan transisi antar state dalam bentuk urutan aksi
    - Bentuk yang lebih umum dari *State Transition Diagram*
  - Diagram Interaksi**
    - Menjelaskan interaksi antar instansiasi dari aktor dan instansiasi dari use-case

# Aliran Event (Flow of Event) (I)

- Aliran ini bisa bersifat
  - Aliran yang '**normal**', atau aliran 'dasar' atau jalur dasar (basic path) atau "Happy path"
  - Aliran **alternatif**
    - Varian yang regular
    - Penanganan kasus khusus
    - Aliran yang khusus menangani terjadinya 'error'

Contoh:

- Basic Path untuk use case 'Terima Barang' adalah ketika pelanggan memasukkan suatu item barang dan meminta tanda terima
- Alternatif path adalah use case ketika suatu barang terjepit di slot pemasukan barang



# Aliran Event (*Flow of Event*) (2)

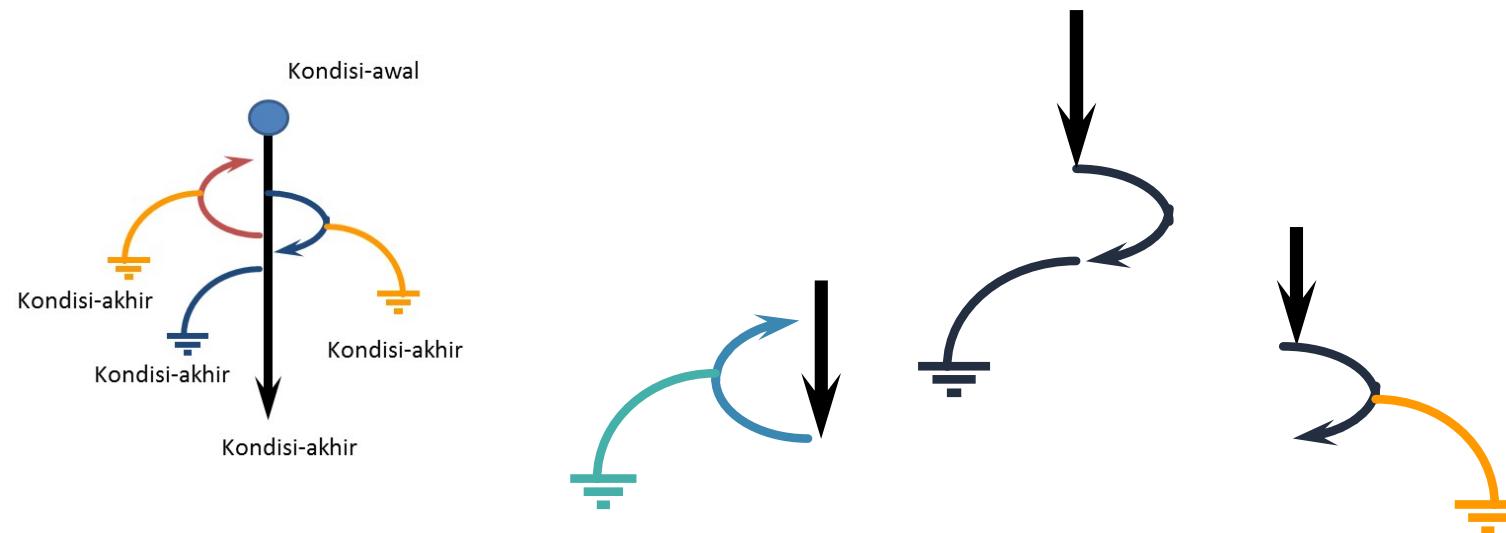
- Aliran Basic (**basic flow**) atau **normal** (flow) menjelaskan
  - Apa yang terjadi pada ‘umum’nya kalau suatu use-case dilakukan
  - Sering disebut sebagai ‘**happy path**’ atau skenario ‘happy path’
- Aliran event dalam use-case bisa dibagi menjadi beberapa ‘**sub flow**’
  - Beberapa bagian dari *flow event* bisa dipecah dan pecahannya dapat diberikan penjelasan secara terpisah; untuk meningkatkan kemudahan pembacaan (readability), dan menjadi perbaikan dari struktur model use-case
  - Pemisahan ini juga membuat *flow dasar* menjadi lebih jelas
  - Jika suatu *subflow* melibatkan hanya sebagian kecil dari *event flow* lengkapnya, mungkin cukup dijelaskan dalam teks saja.
  - *Sub flow* kadang sebaiknya dipisahkan sebagai suplemen terpisah dari bagian *flow of events*

# Aliran Event (Flow of Event) (3)

- Use-case harus mencakup **semua flow**, baik yang **normal, alternatif** maupun **flow pengecualian (exceptional)**.
  - use-case harus digambarkan sedemikian rupa sehingga mudah untuk melihat flow dan mudah dimengerti apa yang terjadi pada satu saat
- Contoh subflow
  - Bila menempati bagian yang cukup besar dalam event flow
  - Jika merupakan event flow varian, atau exception
  - Bisa dieksekusi pada beberapa interval yang dalam event flow yang sama
  - Sub flow mungkin dilakukan pada waktu yang sama, dan mungkin bersifat optional

# Skenario

- Skenario adalah hasil ‘instansiasi’ dari use-case
  - Berisi satu aliran (**flow**) dalam suatu use-case



Garis hitam tebal merepresentasikan skenario yang mungkin untuk aliran basic dan alternatif

# Skenario (2)

- Setiap use-case akan memiliki sekumpulan *flow-event* dengan skenario sebagai bagian dari instansiasi suatu aliran event
  - Skenario mungkin melibatkan *basic flow* dan alternatif *flow* (dalam berbagai kombinasi)
- Berapa skenario yang dibutuhkan?
  - Sebanyak mungkin, agar sistem dapat dimengerti hingga mudah dikembangkan
  - Dan perlu ditekankan jika ada use-case yang ‘menarik’ atau memiliki resiko tinggi.
    - Skenario dapat digunakan untuk mengerti, juga untuk mem-validasi aliran event dalam use-case
- Dalam pemodelannya, ada yang menulis skenario lalu use-case atau sebaliknya
- Skenario dapat memudahkan pengembangan kasus pengujian (*test case*)

# *Langkah pemodelan berbasis skenario*

- Cari aktor
- Cari use case
- Gambarkan diagram use case
- Buat skenario tiap use case



KNOWLEDGE & SOFTWARE ENGINEERING

# Cari aktor

- Siapa '**pengguna**' sistem atau yang terkait dengan sistem
  - Pelanggan
  - Operator
- Pada kasus ini,
  - Ada aktor yang **menggunakan** sistem
  - Ada aktor yang bertugas melakukan perawatan (**maintenance**)
- **Peran** aktor harus **berbeda**
  - Mungkin bisa terjadi peran yang saling tumpang tindih
    - Perlu identifikasi yang jelas (mungkin melibatkan diskusi panjang dengan calon pemilik sistem)
  - Perlu **nama** yang 'relevan' dengan makna semantik dari peran
    - Contoh:
      - **Pengguna sistem** dengan **pelanggan sistem**, harus jelas apa yang dimaksud dengan peran ini. Misalnya pengguna ATM adalah juga pelanggan (customer) dari Bank. Gunakan istilah yang jelas perannya, bila kurang jelas, perlu diperjelas dalam penjelasan deskripsinya.
      - Istilah pengguna sistem mungkin juga adalah sistem lain (bukan orang).
- Juga harus jelas apa '**kebutuhan**' dan '**tanggung jawab**' si aktor!

# Cari use-case

- Daftarkan **aktivitas** yang dilakukan oleh **aktor** untuk melakukan suatu **fungsi/kegiatan**
  - Lakukan untuk semua aktor
- Beri **nama**
  - Nama ini sebaiknya membuat kita berpikir terhadap sekumpulan aksi yang akan terkait dengan aktor
  - Biasanya dimulai dengan '**kata kerja**'
- Use-case biasanya harus '**lengkap**' atau dapat '**berdiri sendiri**'
- Use-case ini memberikan suatu '**hasil**' untuk **aktor** ini
- Use-case biasanya ditulis dalam bentuk **cerita** yang kemudian dipetakan ke suatu template
- Setiap skenario utama harus dikaji ulang (review) dan diperbaiki untuk melihat alternatif interaksi yang mungkin
  - Apakah aktor bisa punya peran lain atau melakukan aksi lain
  - Apakah mungkin aktor mengalami kondisi 'error'
  - Apakah mungkin aktor melakukan perilaku/aksi yang berbeda pada suatu titik?

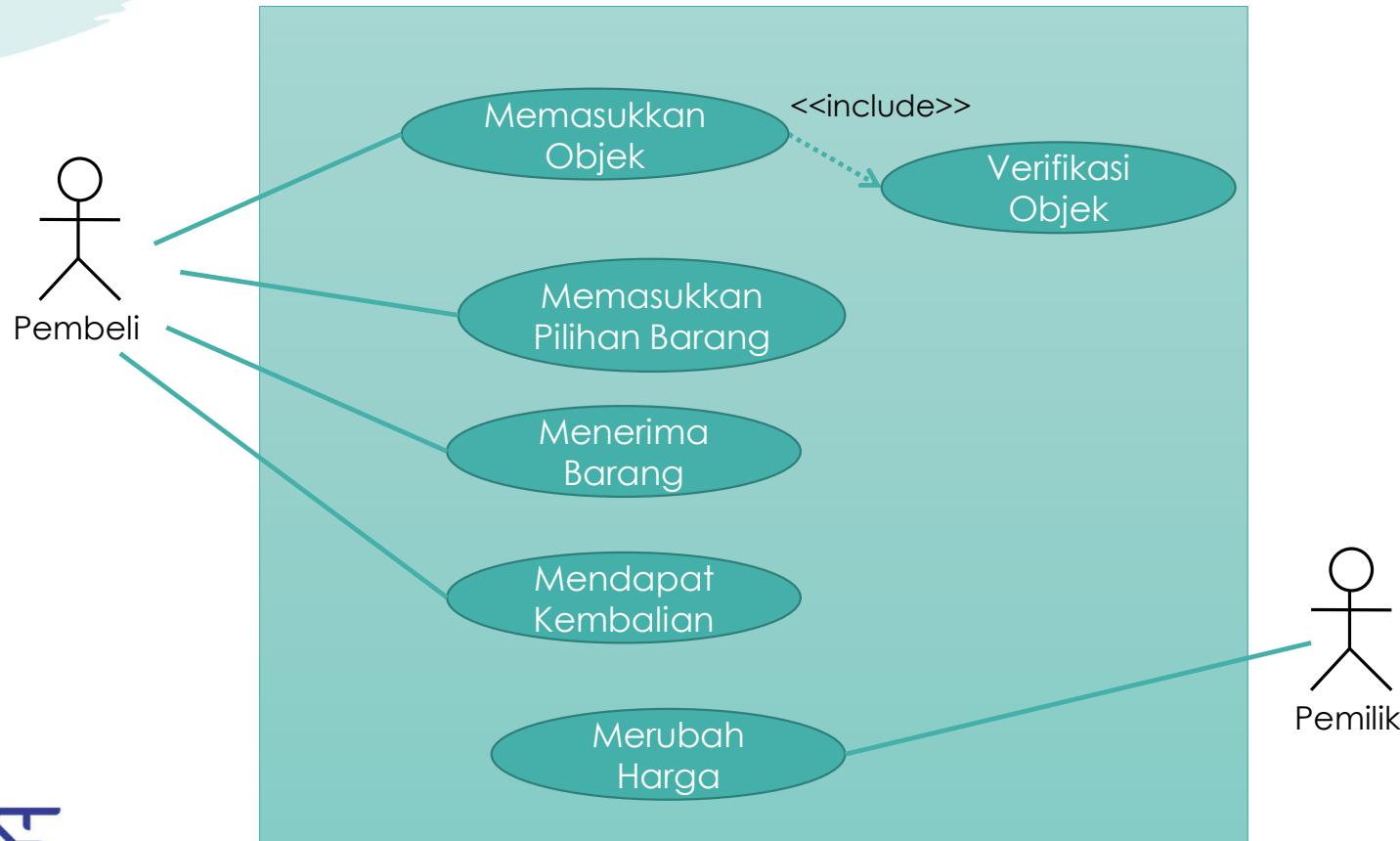
# Gambarkan diagram use case

- Setiap **use case** harus **terhubung** dengan minimal **satu aktor**
- Jika use case terhubung dengan **lebih** dari satu aktor, maka harus diperjelas aktor yang men-**trigger** use case pertama kali (gunakan **tanda panah**)
- Mungkin ada hubungan antar use case: **include** (uses) atau **extends** (insert)



KNOWLEDGE & SOFTWARE ENGINEERING

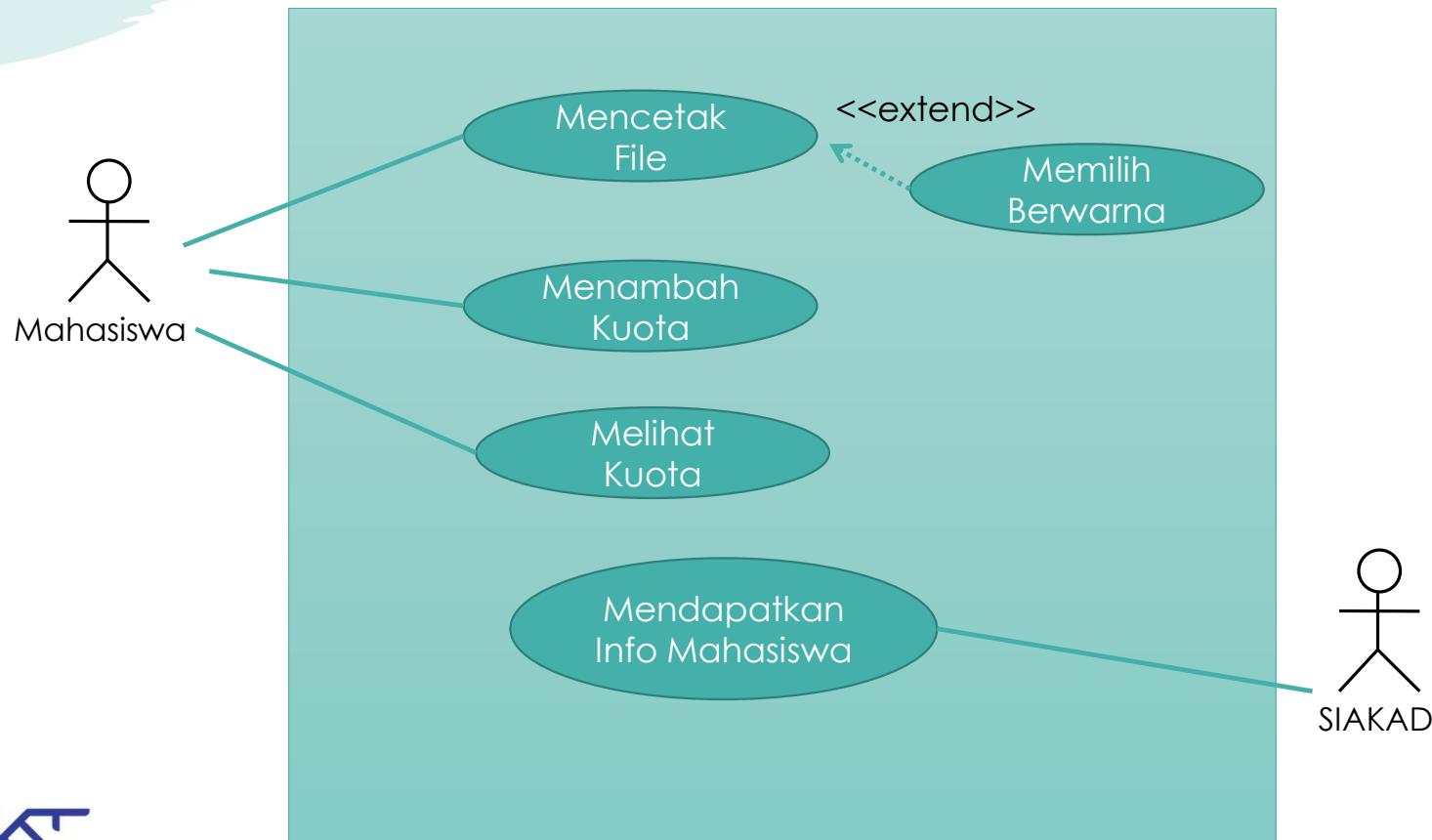
# Use Case untuk VM



KNOWLEDGE &amp; SOFTWARE ENGINEERING

IF2250 - Scenario Modeling

# Use Case Layanan Pencetakan Mahasiswa



# Membuat scenario

- Sekumpulan skenario dari pengguna digunakan untuk **menjelaskan pemakaian sistem**
- Setiap skenario **dilihat dari sudut pandang ‘aktor’**
  - Aktor adalah orang atau sistem yang melakukan interaksi terhadap dengan software
- Setiap skenario **menjawab pertanyaan** berikut:
  - Siapa aktor utama, siapa aktor pendukung
  - Apa tujuan si aktor
  - Kondisi awal apa yang harus ada sebelum suatu cerita atau ‘story’ dimulai
  - Apa tugas/fungsi utama yang dilakukan oleh si aktor
  - Apa tugas/fungsi tambahan yang dapat diberikan
  - Apa variasi yang memungkinkan dalam interaksi si aktor
  - Informasi dari sistem apa yang dibutuhkan, diproduksi atau diubah dari/oleh si aktor
  - Apakah si aktor harus memberitahukan sistem bila terjadi perubahan pada lingkungan eksternal?
  - Informasi apa yang diinginkan oleh si aktor dari sistem
  - Apakah si aktor ingin diberitahu bila ada perubahan yang di luar rencana?

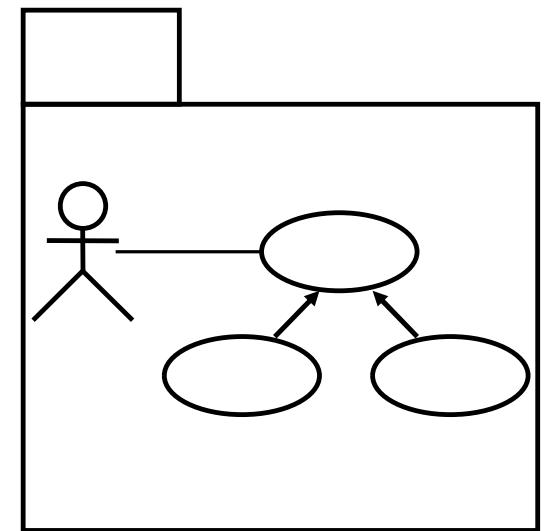


KNOWLEDGE & SOFTWARE ENGINEERING



# **Paket (Packages) dalam model Use-case**

- **Paket** digunakan untuk **mengelompokkan** elemen-elemen yang terkait secara **semantik**
- Kegunaan:
  - Use-case lebih terstruktur, bisa disesuaikan dengan fungsi atau tipe pengguna
  - Sebagai **batasan** lingkup dari satu atau beberapa use-case
  - Paket dalam use-case juga bisa digunakan untuk
    - Menunjukkan urutan sistem
    - Konfigurasi sistem
    - Delivery unit
  - Pengalokasian sumber daya/kompetensi dalam tim pengembang dapat dibagi menjadi beberapa kelompok yang berbeda pada tempat yang berbeda



# Contoh: Mesin Recycle



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 - Scenario Modeling

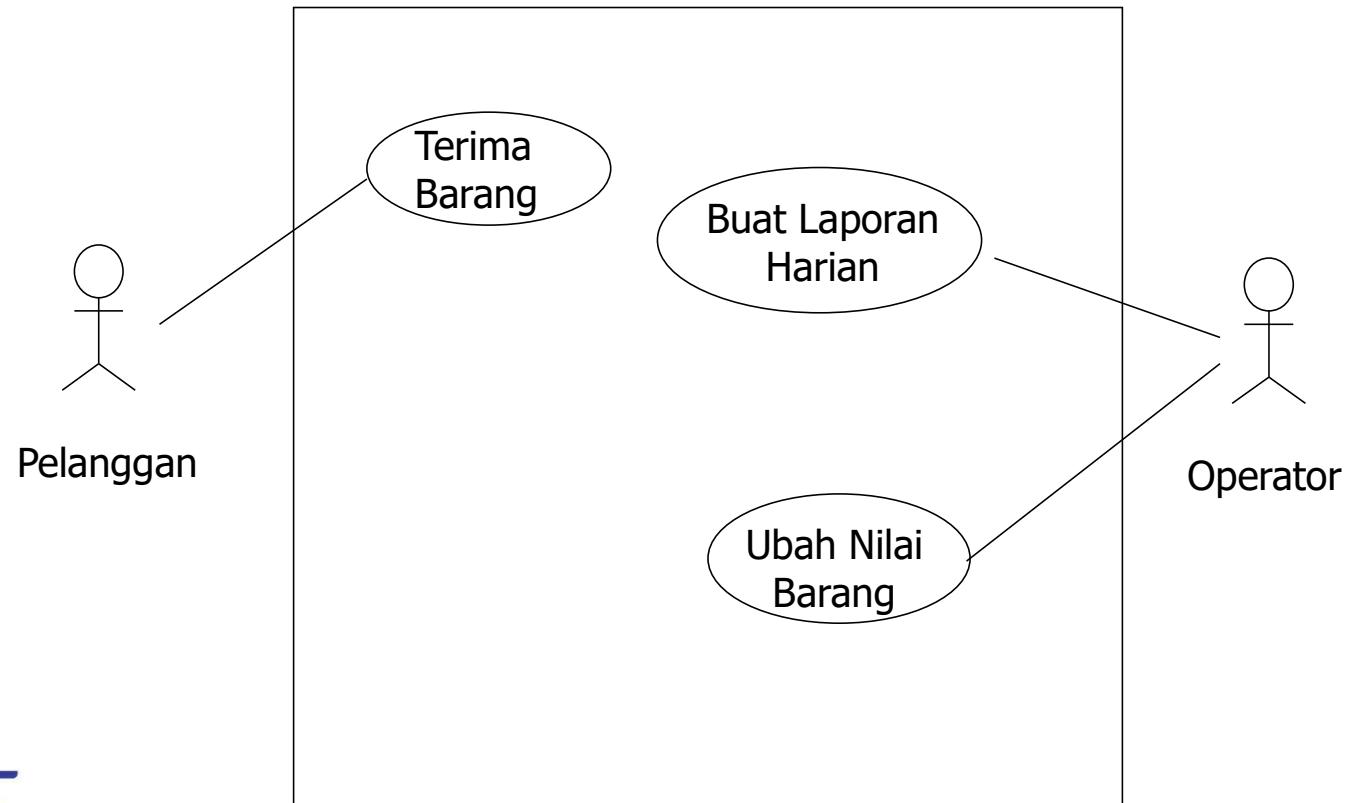
# *Sistem Mesin Recycle*

- Suatu mesin melakukan *recycle* terhadap botol dan kaleng
- Mesin dapat digunakan oleh pengguna yang berbeda-beda
- Sistem akan mencatat tipe dan jumlah item yang dimasukkan
- Sistem akan mencetak tanda terima agar dapat diganti dengan uang oleh pengguna
- Operator dapat meminta laporan harian
- Operator dapat merubah nilai dari item (benda) yang dikembalikan
- (Operator akan diinformasikan bila ditemukan *malfunction*)

# Mesin Recycle

- Aktor?
  - Pelanggan
    - Memasukkan kaleng, botol ke mesin recycling
    - Tekan tombol
    - Menerima tanda terima yang bisa ditukar dengan uang
  - Operator
- Ada berapa use-case?
  - Aksinya:
    - Memasukkan item (barang)
    - Tekan tombol
    - Mencetak tanda terima
  - Tapi dalam kenyataannya,
    - Ada tiga kejadian, tapi masing-masing saling terkait, dan bagi pelanggan detil kejadian itu tidak terlalu penting.
    - Agar pelanggan mudah mengerti, maka ketiga aksi itu dapat dilihat sebagai proses yang lengkap
      - Mulai dari memasukkan barang(item), tekan tombol hingga dapat tanda terima
  - Jadi sebagai suatu instansiasi yang lengkap hanya akan ada satu use-case
- Gambarkan diagram use case (slide berikutnya)
- Detilkan skenarionya

# Contoh: Sistem Mesin Recycle



## **Contoh: Jalur Dasar untuk use-case “Terima Barang”**

- Kondisi Awal(*Pre-condition/Initial State*): Pelanggan mau meletakkan botol dan kaleng ke mesin
  - Pelanggan meletakkan setiap barang ke mesin
  - Sistem akan menaikkan jumlah barang yang diterima, juga mencatat jumlah total barang secara keseluruhan
  - Pelanggan menekan tombol selesai untuk mendapatkan tanda terima
  - Sistem akan mencetak tanda terima berisi jumlah barang yang dimasukkan.
- Jalur alternatif
  - Pada langkah pertama, barang mungkin terjepit di slot mesin
    - Pada kasus ini, pelanggan harus diinformasikan, operator juga diberikan notifikasi, dan mesin akan mencetak tanda terima barang yang sudah terlebih dulu dimasukkan
- Kondisi Akhir (*Post-condition/final state*): Use-case selesai setelah tombol tanda terima (tombol selesai) ditekan

## Jalur dasar: Buat Laporan harian

- Kondisi-awal: Operator mau mencetak laporan harian untuk semua barang yang dimasukkan
  - Sistem akan mencetak jumlah barang yang diterima untuk setiap tipe
  - Sistem akan mencetak jumlah barang
  - Sistem akan di-reset kembali nol
- Kondisi-akhir: Jumlah harian di-set nol setelah use-case selesai

# *Jalur Dasar: Ubah Nilai Barang*

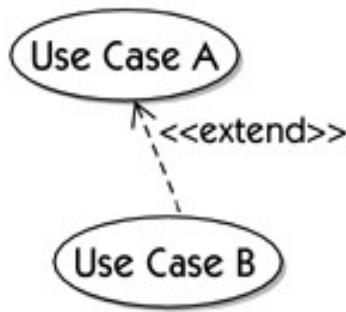
- Kondisi-awal: Operator mau mengubah nilai keterangan barang
  - Nilai jumlah setiap barang yang akan diubah
  - Ukuran barang yang akan dirubah
  - Tipe barang baru dapat ditambahkan
- Kondisi-akhir: Nilai keterangan barang baru berubah



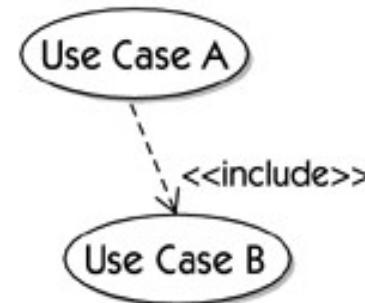
KNOWLEDGE & SOFTWARE ENGINEERING

# Ketergantungan antar use-case

- Hubungan **extend** mendefinisikan suatu instansiasi dari suatu use-case yang menunjukkan adanya perilaku (behaviour) **tambahan**
- Hubungan **include** mendefinisikan hubungan **langsung** dua use-case
  - use-case yang dimasukkan akan dilakukan oleh use-case dasarnya



use-case B  
**mungkin** dilakukan  
oleh use-case A



use-case B **dilibatkan**  
ketika use-case A  
dilakukan

# Extend/Include

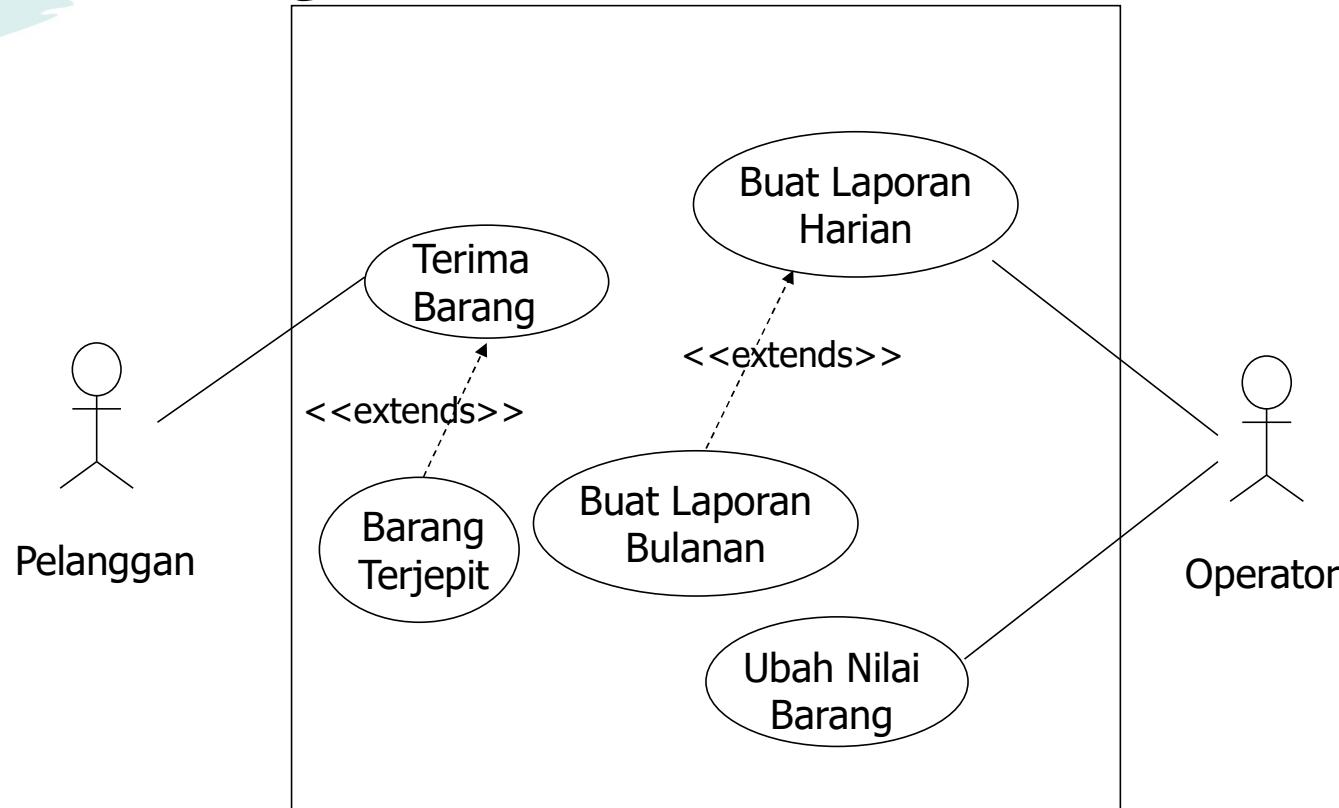
- Dalam model use-case, dimungkin adanya pilihan event alternatif ataupun ada use-case yang harus dilakukan jika suatu use-case dieksekusi
  - **Alternatif** use-case ditunjukkan sebagai **<<extends>>**
    - Extend ini menjadi bagian yang **optional** (bisa dilakukan dan bisa tidak)
  - Use-case yang **harus** dilakukan ditunjukkan sebagai **<<include>>**
    - **Harus dilakukan**
- Walau jarang terjadi, tetapi use-case extends dan include bisa terjadi.
  - Digunakan untuk kasus tertentu
- Kenapa ada extends/include?
  - Karena perkembangan kebutuhan, sehingga harus ditambahkan fitur/ fungsionalitas baru
  - use-case perlu dikembangkan untuk mendapatkan aksi tambahan dalam suatu kondisi lain



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 - Scenario Modeling

# Contoh dengan Extends



KNOWLEDGE & SOFTWARE ENGINEERING

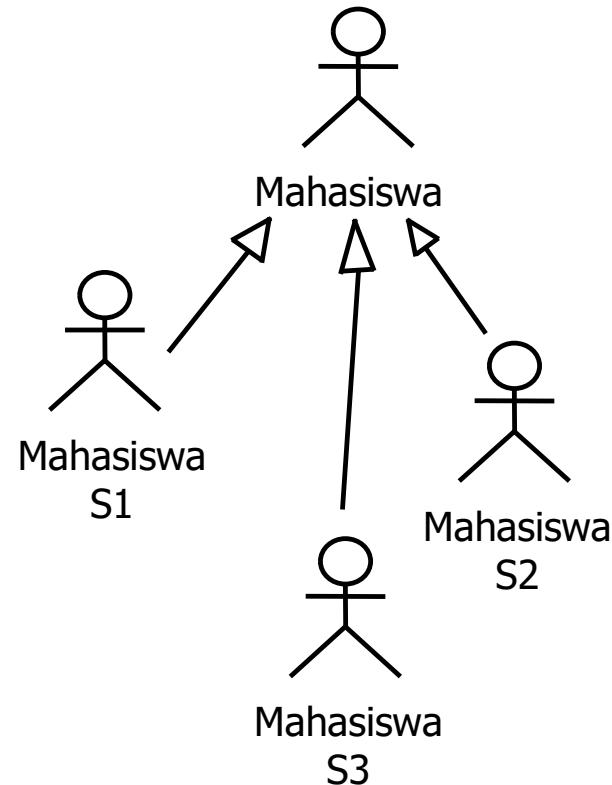
# Contoh dengan Include



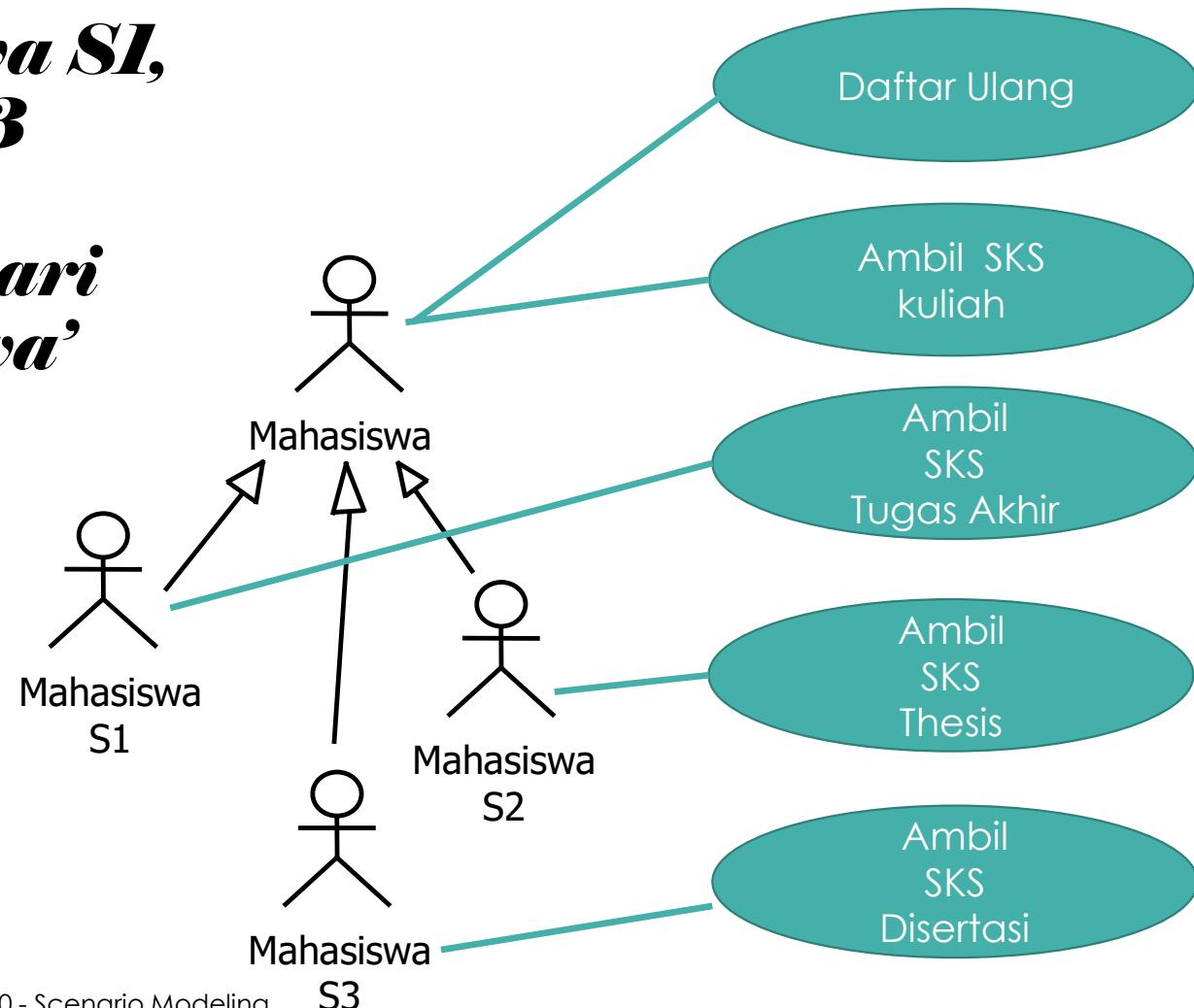
KNOWLEDGE &amp; SOFTWARE ENGINEERING

# **Generalisasi Aktor (Actor Generalization/Actor Inheritance)**

- Beberapa aktor dapat memiliki **peran** yang **sama** pada suatu use-case
- Contoh:
  - Ada mahasiswa S1, S2 dan S3 yang ketiganya terdaftar suatu kuliah
  - Ketiga akan terlihat sebagai entitas eksternal oleh use-case ‘Daftar Ulang’ atau ‘Ambil mata kuliah’
  - Mahasiswa S1, S2 atau S3 dapat dimodelkan sebagai ‘Mahasiswa’ saja, karena ketiganya memiliki banyak kesamaan
  - Hal ini disebut ‘**generalisasi aktor**’

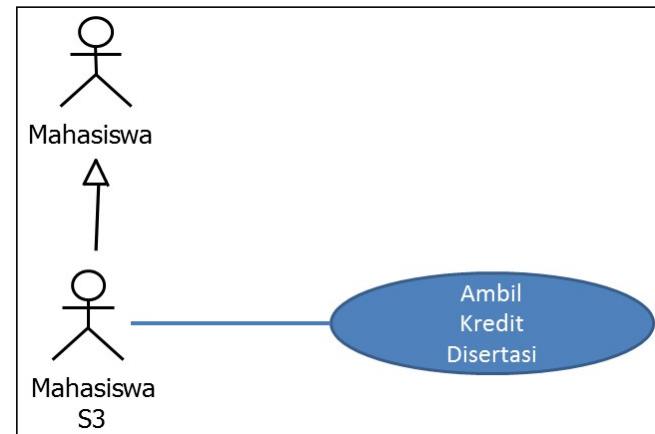
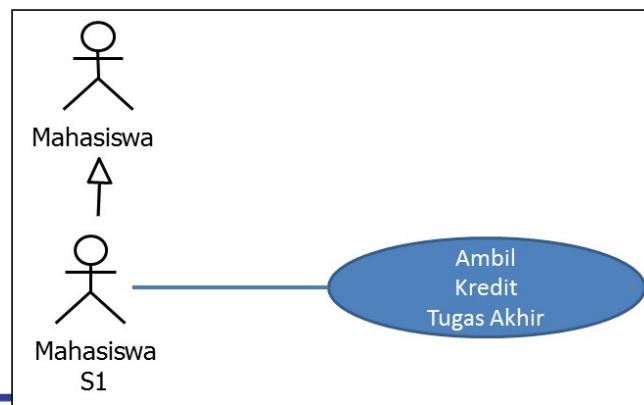
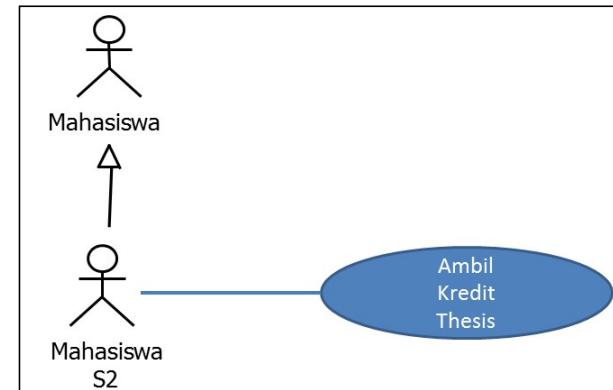
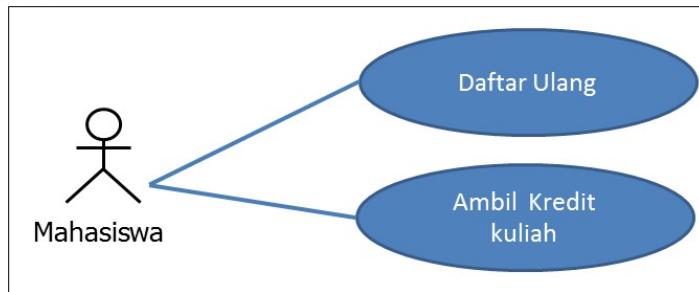


***Mahasiswa SI,  
S2, dan S3  
adalah  
turunan dari  
'Mahasiswa'***



KNOWLEDGE & SOFTWARE ENGINEERING

# *Generalisasi Aktor dapat dipisahkan penggambarnya*



# *Pemakaian Use-case*

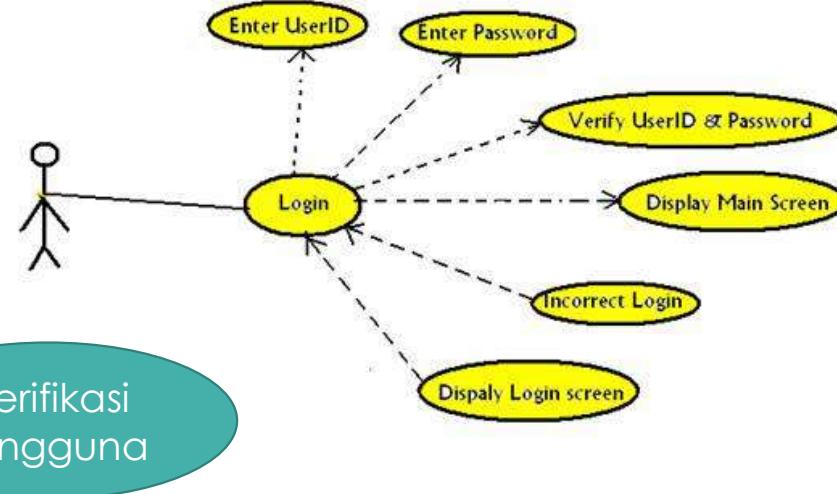
- Dalam pengembangan perangkat lunak, Use-case akan berperan:
  - Membantu proses iterasi untuk mengerti masalah
  - Membantu mencari prioritas pengembangan



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 - Scenario Modeling

# Apa yang salah?



## Skenario:

- Display Login Screen
- Enter User Id
- Enter Password
- Verify User Id dan Password
- If LoginError then Incorrect Login
- Else display Main Screen

# Latihan membuat diagram use-case

Akan dibangun sebuah perangkat lunak untuk mendukung proses pendaftaran ulang mahasiswa secara online. Melalui aplikasi tersebut, mahasiswa dapat mengajukan usulan pengambilan matakuliah.

Selanjutnya, dosen wali dapat melihat usulan pengambilan matakuliah untuk disetujui/ditolak. Usulan yang ditolak dapat direvisi kembali oleh mahasiswa.

Usulan yang telah disetujui wali dapat langsung diproses oleh Petugas Administrasi untuk pencetakan KSM. KSM hanya bisa dicetak apabila status pembayaran SPP mahasiswa sudah beres. Informasi status pembayaran SPP diperoleh dari perangkat lunak lain yaitu SISKEU (Sistem Informasi Keuangan). Perangkat lunak ini juga akan berhubungan dengan perangkat lunak SIKAD (Sistem Informasi Akademik) untuk mendapatkan informasi tentang matakuliah yang ditawarkan pada semester tersebut, serta informasi transkrip nilai mahasiswa, agar dosen wali mendapatkan referensi untuk menyetujui/menolak usulan pengambilan matakuliah.



KNOWLEDGE &amp; SOFTWARE ENGINEERING

IF2250 - Scenario Modeling

mahasiswa dpt merevisi usulan  
doswal dpt melihat usulan  
doswal dpt setujui/tolak usulan  
petugas administrasi dpt mencetak ksm  
siskeu --> pasif (panahnya nanti keluar)

# Cari Aktor!

Akan dibangun sebuah perangkat lunak untuk mendukung proses pendaftaran ulang mahasiswa secara online. Melalui aplikasi tersebut, mahasiswa dapat mengajukan usulan pengambilan matakuliah.

Selanjutnya, dosen wali dapat melihat usulan pengambilan matakuliah untuk disetujui/ditolak. Usulan yang ditolak dapat direvisi kembali oleh mahasiswa.

Usulan yang telah disetujui wali dapat langsung diproses oleh Petugas Administrasi untuk pencetakan KSM. KSM hanya bisa dicetak apabila status pembayaran SPP mahasiswa sudah beres. Informasi status pembayaran SPP diperoleh dari perangkat lunak lain yaitu SISKEU (Sistem Informasi Keuangan). Perangkat lunak ini juga akan berhubungan dengan perangkat lunak SIKAD (Sistem Informasi Akademik) untuk mendapatkan informasi tentang matakuliah yang ditawarkan pada semester tersebut, serta informasi transkrip nilai mahasiswa, agar dosen wali mendapatkan referensi untuk menyetujui/menolak usulan pengambilan matakuliah.



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 - Scenario Modeling

# Aktor

Akan dibangun sebuah perangkat lunak untuk mendukung proses pendaftaran ulang **mahasiswa** secara online. Melalui aplikasi tersebut, mahasiswa dapat mengajukan usulan pengambilan matakuliah.

Selanjutnya, **dosen wali** dapat melihat usulan pengambilan matakuliah untuk disetujui/ditolak. Usulan yang ditolak dapat direvisi kembali oleh mahasiswa.

Usulan yang telah disetujui wali dapat langsung diproses oleh **Petugas Administrasi** untuk pencetakan KSM. KSM hanya bisa dicetak apabila status pembayaran SPP mahasiswa sudah beres. Informasi status pembayaran SPP diperoleh dari perangkat lunak lain yaitu **SISKEU** (Sistem Informasi Keuangan). Perangkat lunak ini juga akan berhubungan dengan perangkat lunak **SIKAD** (Sistem Informasi Akademik) untuk mendapatkan informasi tentang matakuliah yang ditawarkan pada semester tersebut, serta informasi transkrip nilai mahasiswa, agar dosen wali mendapatkan referensi untuk menyetujui/menolak usulan pengambilan matakuliah.



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 - Scenario Modeling

# Aktor

- Mahasiswa
- Dosen Wali
- Petugas Administrasi
- SISKEU
- SIKAD



KNOWLEDGE & SOFTWARE ENGINEERING

# Cari Use-case!

Akan dibangun sebuah perangkat lunak untuk mendukung proses pendaftaran ulang mahasiswa secara online. Melalui aplikasi tersebut, mahasiswa dapat mengajukan usulan pengambilan matakuliah.

Selanjutnya, dosen wali dapat melihat usulan pengambilan matakuliah untuk disetujui/ditolak. Usulan yang ditolak dapat direvisi kembali oleh mahasiswa.

Usulan yang telah disetujui wali dapat langsung diproses oleh Petugas Administrasi untuk pencetakan KSM. KSM hanya bisa dicetak apabila status pembayaran SPP mahasiswa sudah beres. Informasi status pembayaran SPP diperoleh dari perangkat lunak lain yaitu SISKEU (Sistem Informasi Keuangan). Perangkat lunak ini juga akan berhubungan dengan perangkat lunak SIKAD (Sistem Informasi Akademik) untuk mendapatkan informasi tentang matakuliah yang ditawarkan pada semester tersebut, serta informasi transkrip nilai mahasiswa, agar dosen wali mendapatkan referensi untuk menyetujui/menolak usulan pengambilan matakuliah.



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 - Scenario Modeling

## Use-case

Akan dibangun sebuah perangkat lunak untuk mendukung proses pendaftaran ulang mahasiswa secara online. Melalui aplikasi tersebut, mahasiswa dapat **mengajukan usulan** pengambilan matakuliah.

Selanjutnya, dosen wali dapat **melihat usulan** pengambilan matakuliah untuk **disetujui/ditolak**. Usulan yang ditolak dapat **direvisi** kembali oleh mahasiswa.

Usulan yang telah disetujui wali dapat langsung diproses oleh Petugas Administrasi untuk **pencetakan** KSM. KSM hanya bisa dicetak apabila **status pembayaran** SPP mahasiswa sudah beres. Informasi status pembayaran SPP diperoleh dari perangkat lunak lain yaitu SISKEU (Sistem Informasi Keuangan). Perangkat lunak ini juga akan berhubungan dengan perangkat lunak SIKAD (Sistem Informasi Akademik) untuk **mendapatkan informasi tentang matakuliah** yang ditawarkan pada semester tersebut, serta **informasi transkrip** nilai mahasiswa, agar dosen wali mendapatkan referensi untuk menyetujui/menolak usulan pengambilan matakuliah.



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 - Scenario Modeling

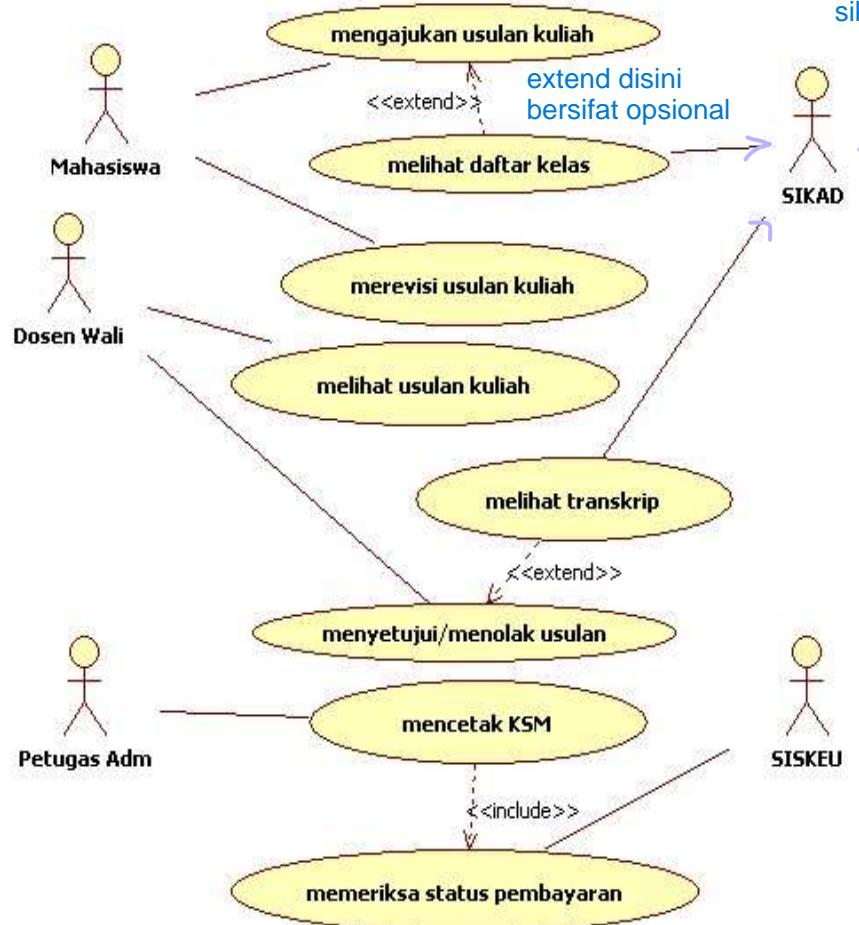
# *Use-case*

- Mengajukan Usulan
- Melihat Usulan
- Menyetujui Usulan
- Menolak Usulan
- Merevisi Usulan
- Memeriksa Status Pembayaran
- Melihat Daftar Kelas
- Melihat Transkrip
- Mencetak KSM

# Use-case Diagram

55

sikad sbg aktor pasif



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 - Scenario Modeling

# *Skenario Mengajukan Usulan*

- Mahasiswa memilih menu entri usulan kuliah
- Sistem menampilkan form entri FRS
- Mahasiswa mengisikan kode kuliah
- Sistem menampilkan informasi detil matakuliah (nama, sks)
- Mahasiswa menekan tombol SIMPAN
- Sistem menyimpan data usulan ke dalam basisdata



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 - Scenario Modeling

# ***Alternatif skenario Mengajukan Usulan dan Melihat Daftar Kelas***

- Mahasiswa memilih menu entri usulan kuliah
- Mahasiswa memilih untuk melihat daftar kelas
- Sistem menampilkan daftar kelas yang dibuka
- Mahasiswa memilih matakuliah dari daftar
- Mahasiswa menekan tombol SIMPAN
- Sistem menyimpan data usulan ke dalam basisdata



KNOWLEDGE & SOFTWARE ENGINEERING

## *Alternatif skenario... (2)*

- Mahasiswa memilih menu entri usulan kuliah
- Sistem menampilkan form entri FRS
- Mahasiswa mengisikan kode kuliah
- Sistem menampilkan pesan bahwa kelas untuk kuliah tersebut tidak dibuka



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 - Scenario Modeling

# Deskripsi Use Case

<b>Use Case:</b>	Mengajukan Usulan Kuliah
<b>Iteration :</b>	ke-2, Modifikasi terakhir 1 Maret 2018
<b>Primary Actor:</b>	Mahasiswa
<b>Goal in Context:</b>	Untuk pengajuan usulan kuliah oleh mahasiswa
<b>Preconditions:</b>	Mahasiswa sudah terdaftar dan mahasiswa sudah memasukkan nama user dan password sebelumnya
<b>Trigger:</b>	Jika Mahasiswa memutuskan untuk mengambil kuliah di awal semester
<b>Scenario:</b>	<ol style="list-style-type: none"> <li>1. Mahasiswa memilih menu entri usulan kuliah</li> <li>2. Sistem menampilkan form entri FRS</li> <li>3. Mahasiswa mengisikan kode kuliah</li> <li>4. Sistem menampilkan informasi detil matakuliah (nama, sks)</li> <li>5. Mahasiswa menekan tombol SIMPAN</li> <li>6. Sistem menyimpan data usulan ke dalam basisdata</li> </ol>

## Exception:

1. Mahasiswa memilih menu entri usulan kuliah
2. Mahasiswa memilih untuk melihat daftar kelas
3. Sistem menampilkan daftar kelas yang dibuka
4. Mahasiswa memilih matakuliah dari daftar
5. Mahasiswa menekan tombol SIMPAN
6. Sistem menyimpan data usulan ke dalam basisdata

**Priority:** Prioritas sedang

**When available:** Iterasi ketiga

**Secondary Actor:** tidak ada

## Open Issues:

1. Bagaimana mekanisme mendeteksi, jika ada suatu kuliah memiliki prerequisite kuliah lain
2. Bagaimana jika mahasiswa ingin membatalkan usulan

# *Activity Diagram*



KNOWLEDGE & SOFTWARE ENGINEERING



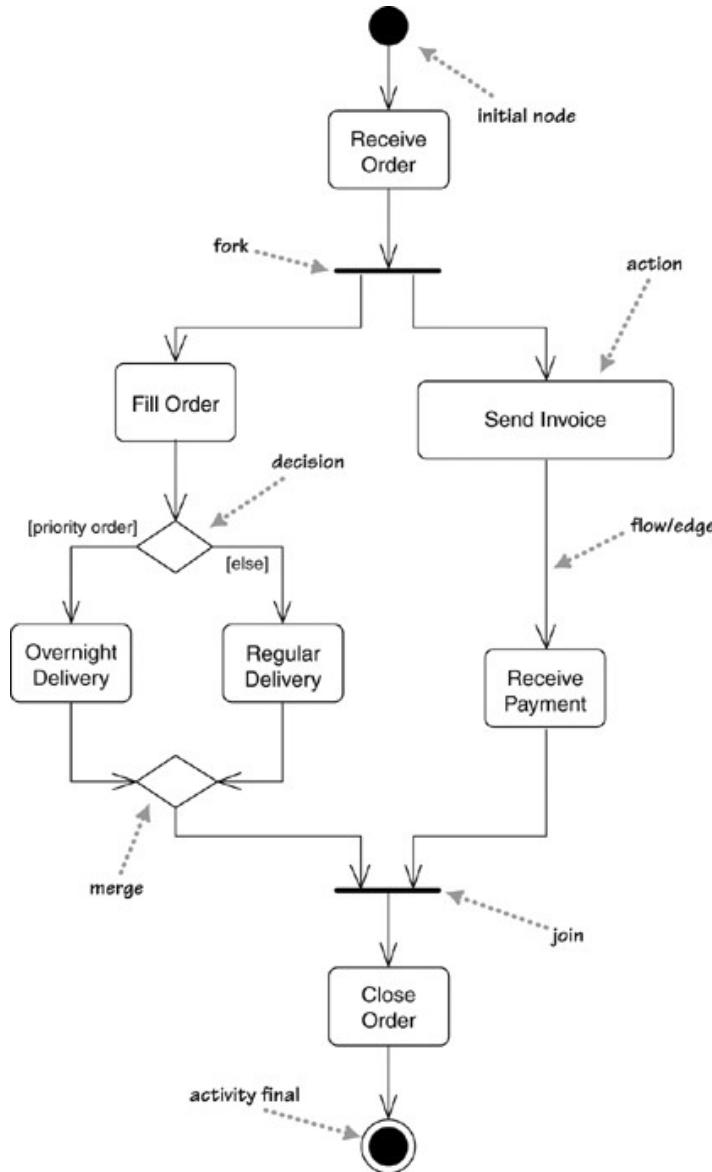
60

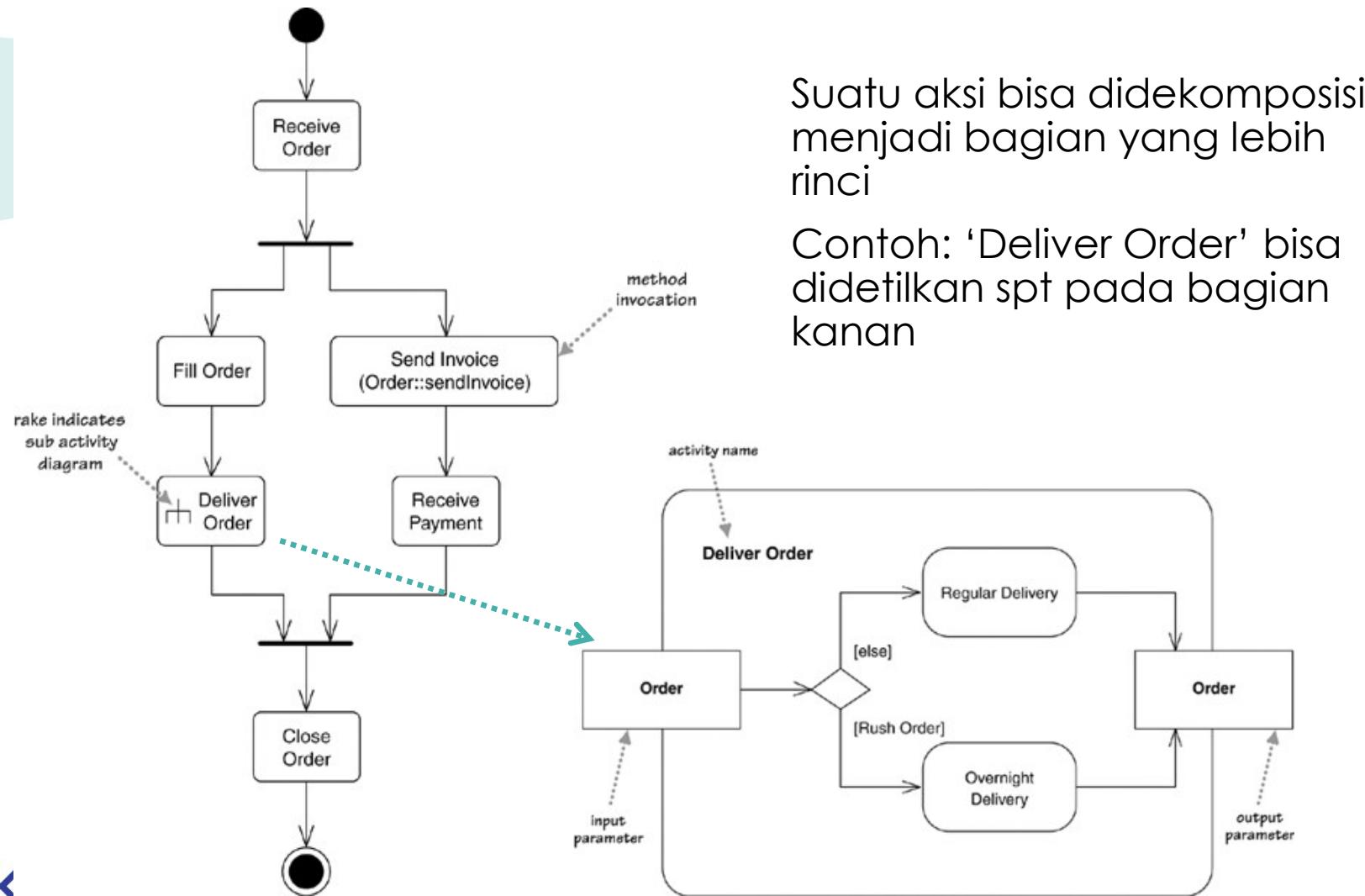
IF2250 - Scenario Modeling

# Activity Diagram

- Diagram aktivitas menjelaskan
  - Urutan proses prosedural
  - Urutan bisnis proses
  - Urutan kerja (*work flow*)
- Diagram aktivitas ini mirip seperti ‘Flow chart’
  - Tetapi flow-chart tidak mengenal ‘perilaku paralel’/‘konkuren’

# Contoh



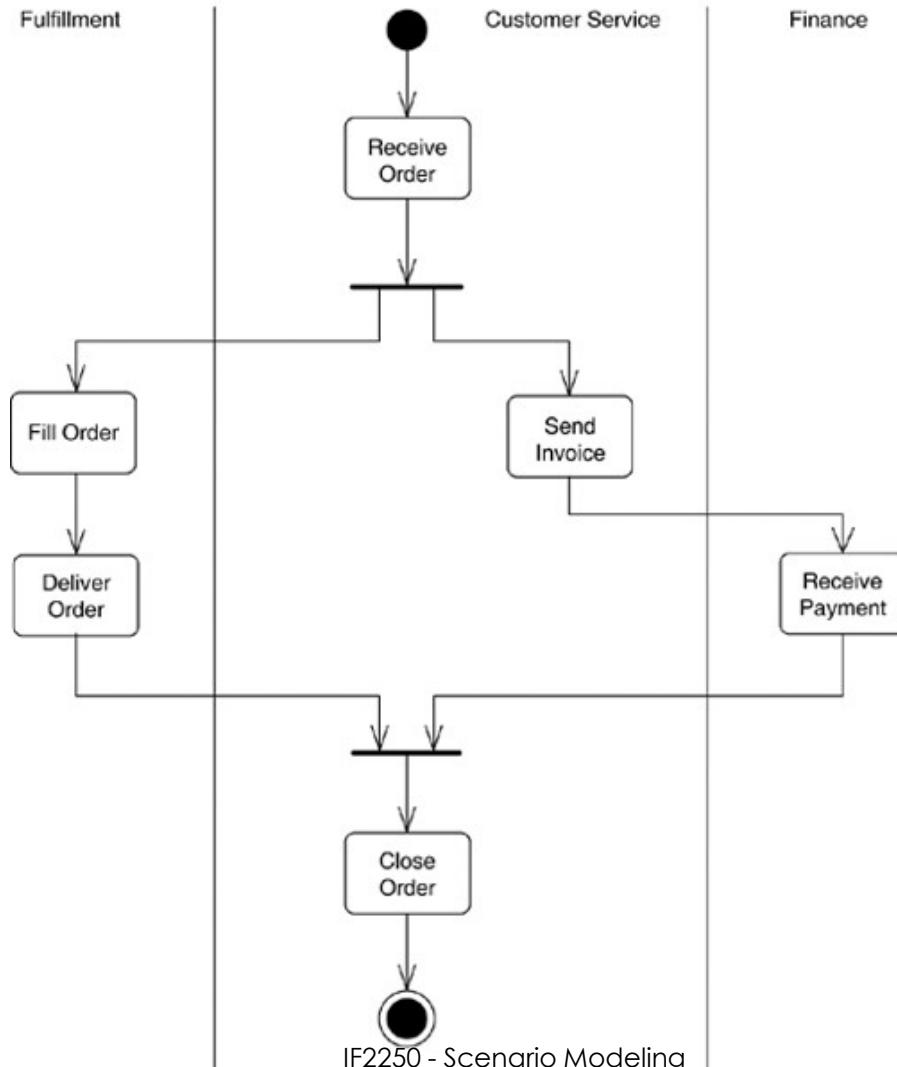


Suatu aksi bisa didekomposisi menjadi bagian yang lebih rinci

Contoh: ‘Deliver Order’ bisa didekomposisikan spt pada bagian kanan

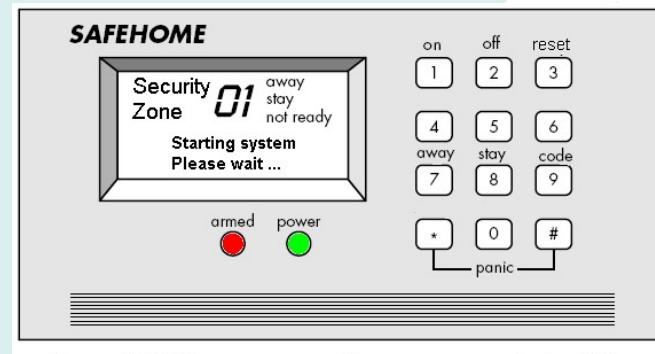
# *Partisi*

Diagram aktivitas dapat dipartisi berdasarkan '**siapa**' yang melakukan '**apa**'



# *Contoh Pengembangan Use-Case untuk Safe Home*

(Referensi: R. Pressman, "Software Engineering: A Practitioner's Approach" 7<sup>th</sup> edition)



See SEPA 193,231pg for more details



IF0250 Scenario Modeling

# *Skenario Awal (Safe Home)*

**The scene:** A meeting room, during the second requirements gathering meeting.

**The players:** Jamie Lazar, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

## **The conversation:**

**Facilitator:** It's time that we begin talking about the SafeHome surveillance function. Let's develop a user scenario for access to the surveillance function.

**Jamie:** Who plays the role of the actor on this?

**Facilitator:** I think Meredith (a marketing person) has been working on that functionality. Why don't you play the role?

**Meredith:** You want to do it the same way we did it last time, right?

**Facilitator:** Right . . . same way.

**Meredith:** Well, obviously the reason for surveillance is to allow the homeowner to check out the house while he or she is away, to record and play back video that is captured . . . that sort of thing.

**Ed:** Will we use compression to store the video?

**Facilitator:** Good question, Ed, but let's postpone implementation issues for now. Meredith?

**Meredith:** Okay, so basically there are two parts to the surveillance function . . . the first configures the system including laying out a floor plan—we have to have tools to help the homeowner do this—and the second part is the actual surveillance function itself. Since the layout is part of the configuration activity, I'll focus on the surveillance function.

Facilitator (smiling): Took the words right out of my mouth.

**Meredith:** Um . . . I want to gain access to the surveillance function either via the PC or via the Internet. My feeling is that the Internet access would be more frequently used. Anyway, I want to be able to display camera views on a PC and control pan and zoom for a specific camera. I specify the camera by selecting it from the house floor plan. I want to selectively record camera output and replay camera output. I also want to be able to block access to one or more cameras with a specific password. I also want the option of seeing small windows that show views from all cameras and then be able to pick the one I want enlarged.

**Jamie:** Those are called thumbnail views.

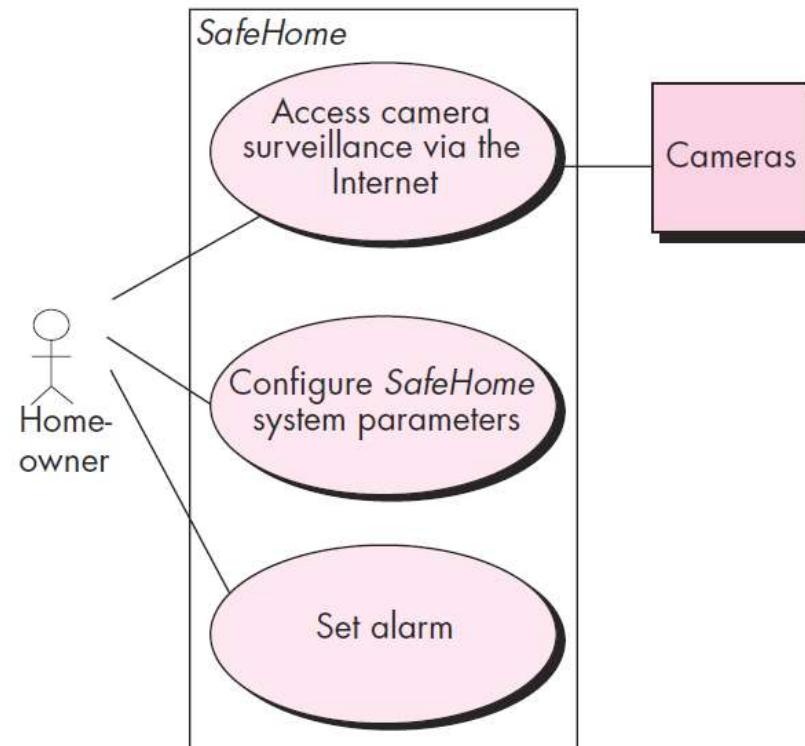
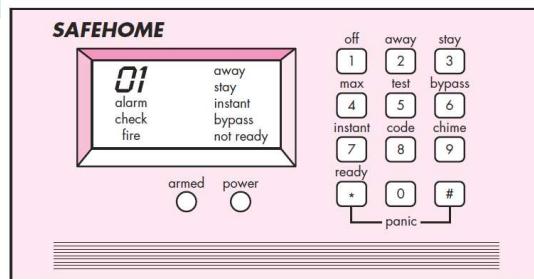
**Meredith:** Okay, then I want thumbnail views of all the cameras. I also want the interface for the surveillance function to have the same look and feel as all other SafeHome interfaces. I want it to be intuitive, meaning I don't want to have to read a manual to use it.

**Facilitator:** Good job. Now, let's go into this function in a bit more detail . . .

# *Fungsi "HomeOwner"*

- Select camera to view.
- Request thumbnails from all cameras.
- Display camera views in a PC window.
- Control pan and zoom for a specific camera.
- Selectively record camera output.
- Replay camera output.
- Access camera surveillance via the Internet.

# Diagram Use-case Safe Home



IF2250 - Scenario Modeling



KNOWLEDGE & SOFTWARE ENGINEERING

# Skenario use case



**SAFEHOME**

 **Use Case Template for Surveillance**

**Use case:** Access camera surveillance via the Internet—display camera views (ACS-DCV)

**Iteration:** 2, last modification: January 14 by V. Raman.

**Primary actor:** Homeowner.

**Goal in context:** To view output of camera placed throughout the house from any remote location via the Internet.

**Preconditions:** System must be fully configured; appropriate user ID and passwords must be obtained.

**Trigger:** The homeowner decides to take a look inside the house while away.

**Scenario:**

1. The homeowner logs onto the *SafeHome Products* website.
2. The homeowner enters his or her user ID.
3. The homeowner enters two passwords (each at least eight characters in length).
4. The system displays all major function buttons.
5. The homeowner selects the “surveillance” from the major function buttons.
6. The homeowner selects “pick a camera.”
7. The system displays the floor plan of the house.
8. The homeowner selects a camera icon from the floor plan.
9. The homeowner selects the “view” button.
10. The system displays a viewing window that is identified by the camera ID.
11. The system displays video output within the viewing window at one frame per second.

**Exceptions:**

1. ID or passwords are incorrect or not recognized—see use case **Validate ID and passwords**.
2. Surveillance function not configured for this system—system displays appropriate error message; see use case **Configure surveillance function**.
3. Homeowner selects “View thumbnail snapshots for all camera”—see use case **View thumbnail snapshots for all cameras**.
4. A floor plan is not available or has not been configured—display appropriate error message and see use case **Configure floor plan**.
5. An alarm condition is encountered—see use case **Alarm condition encountered**.

**Priority:** Moderate priority, to be implemented after basic functions.

**When available:** Third increment.

**Frequency of use:** Moderate frequency.

**Channel to actor:** Via PC-based browser and Internet connection.

**Secondary actors:** System administrator, cameras.

**Channels to secondary actors:**

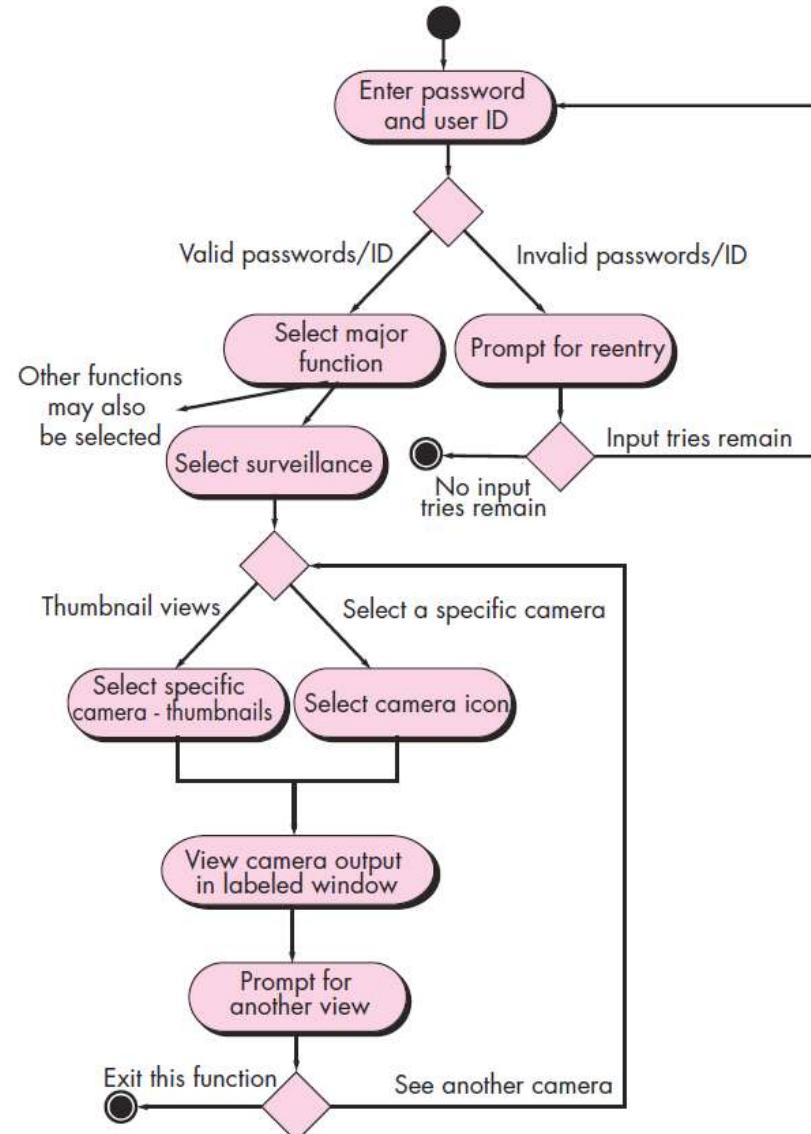
1. System administrator: PC-based system.
2. Cameras: wireless connectivity.

**Open issues:**

1. What mechanisms protect unauthorized use of this capability by employees of *SafeHome Products*?
2. Is security sufficient? Hacking into this feature would represent a major invasion of privacy.
3. Will system response via the Internet be acceptable given the bandwidth required for camera views?
4. Will we develop a capability to provide video at a higher frames-per-second rate when high-bandwidth connections are available?

# Diagram Aktivitas untuk Use case Access camera surveillance via the internet

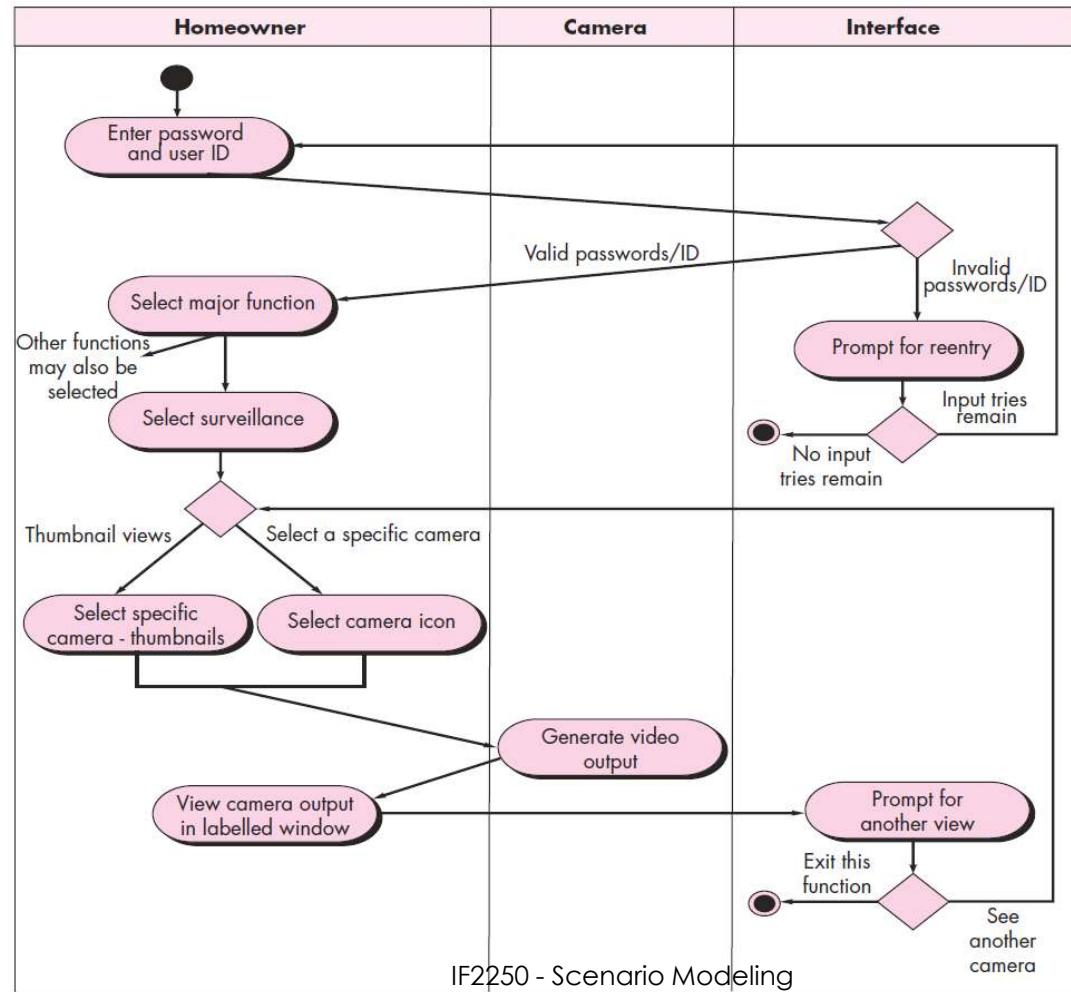
70

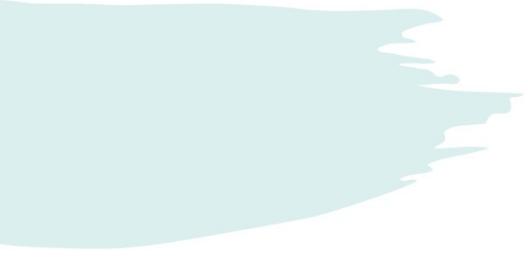


# Diagram Aktivitas dalam bentuk swimlane

71

Diagram ini menunjukkan aliran aktivitas yang dijelaskan dengan use-case dan pada waktu yang sama menunjukkan aktor yang bertanggung jawab terhadap suatu aksi (juga akan berguna untuk analisis kelas)





***Terima kasih***



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 - Scenario Modeling

# Acknowledgement

- Pengembangan dari slide: "Scenario Based Modeling" IF2036 Sem II - 2012/2013
- Pengembangan dari slide: "Pemodelan Berbasis Skenario" oleh Bayu Hendradjaya, IF2250 Sem II - 2015/2016



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250 - Scenario Modeling

Tim Pengajar IF2250

IF2250 – Rekayasa Perangkat Lunak  
**Class Modeling (Object-oriented)**

SEMESTER II TAHUN AJARAN 2022/2023



KNOWLEDGE & SOFTWARE ENGINEERING



# *Analisis Sistem*

Sudut Pandang **Fungsional**



Sudut Pandang **Statik**

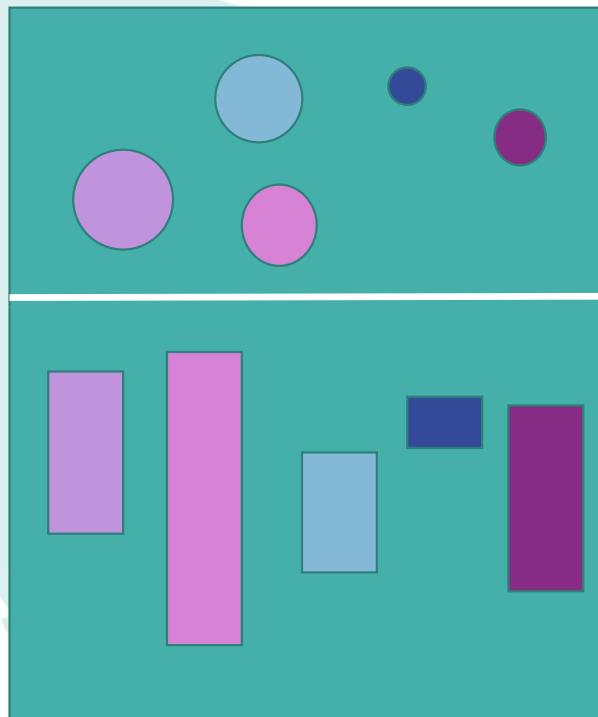


Sudut Pandang **Dinamik**

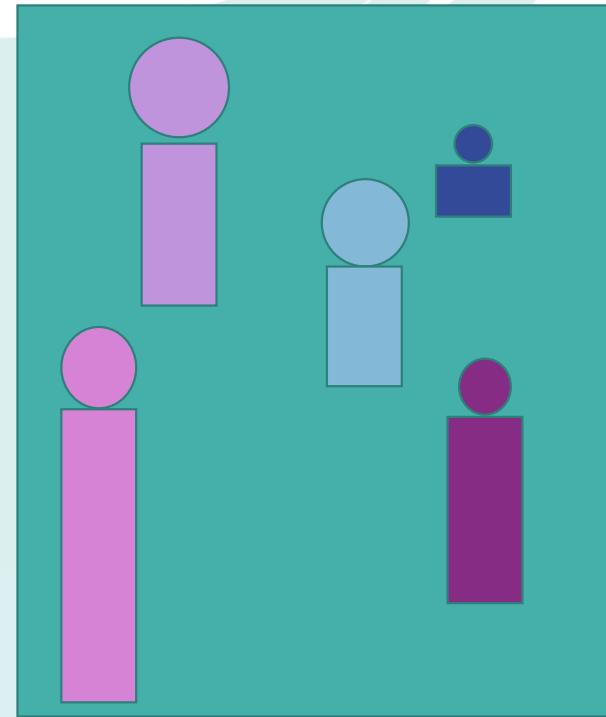


IF0250 Class Modeling

# *Pendekatan Terstruktur vs. Pendekatan Objek*



**Pendekatan Terstruktur**  
Data dan Procedure/fungsi  
dipisahkan



**Pendekatan Objek**  
Data dan Procedure/fungsi yang  
terkait secara semantik  
diintegrasikan



KNOWLEDGE & SOFTWARE ENGINEERING

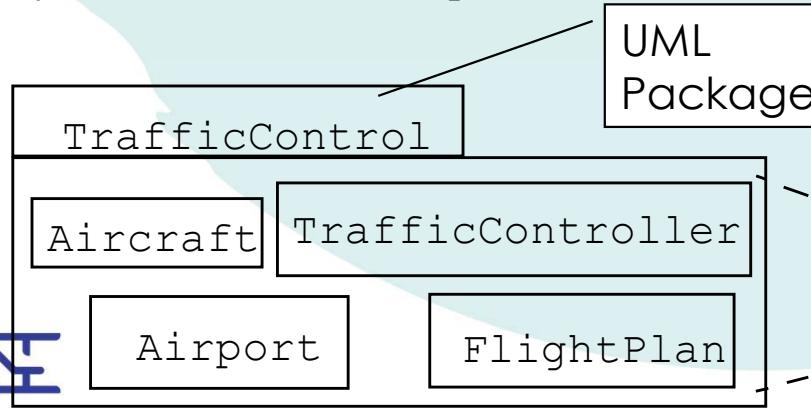
# Object-oriented Modeling



Application Domain  
(Phenomena)

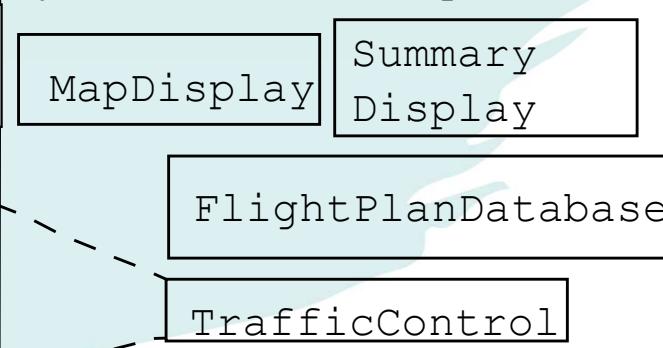


System Model (Concepts) (*Analysis*)



Solution Domain  
(Phenomena)

System Model (Concepts) (*Design*)



# *Sejarah dari “Structured Oriented” hingga “Object Oriented”*

- Perkembangan Bahasa
- Perkembangan Metode Desain
- Perkembangan Metode Analisis



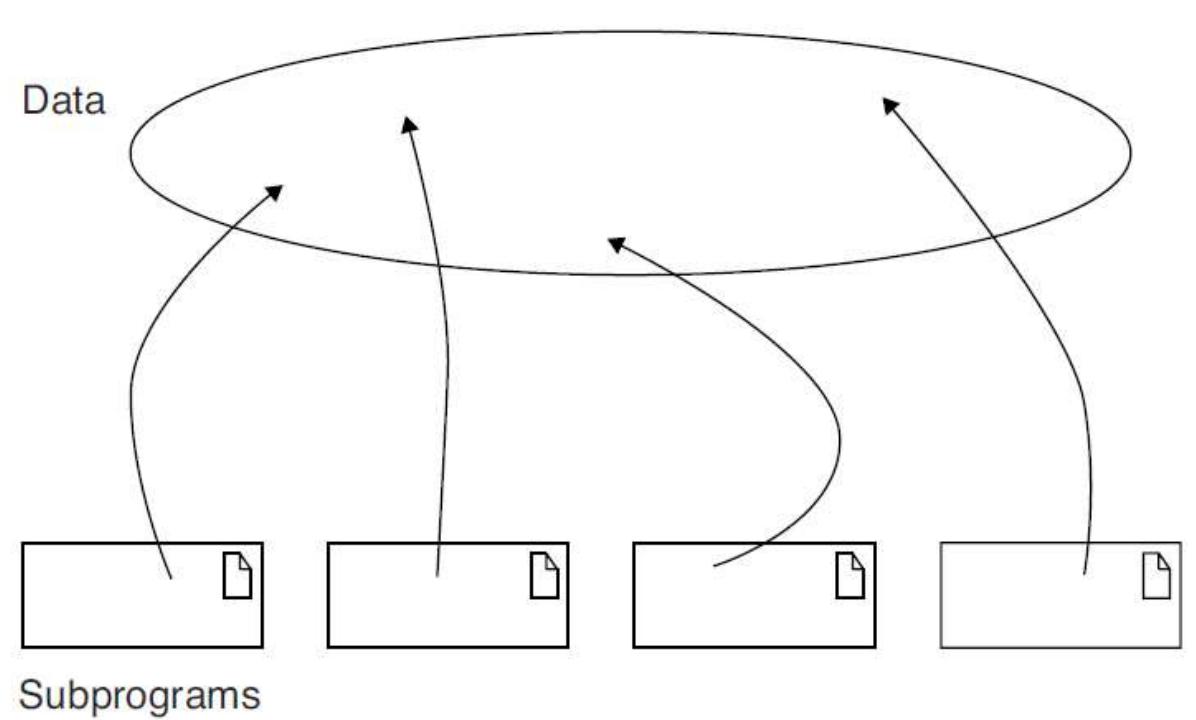
KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling

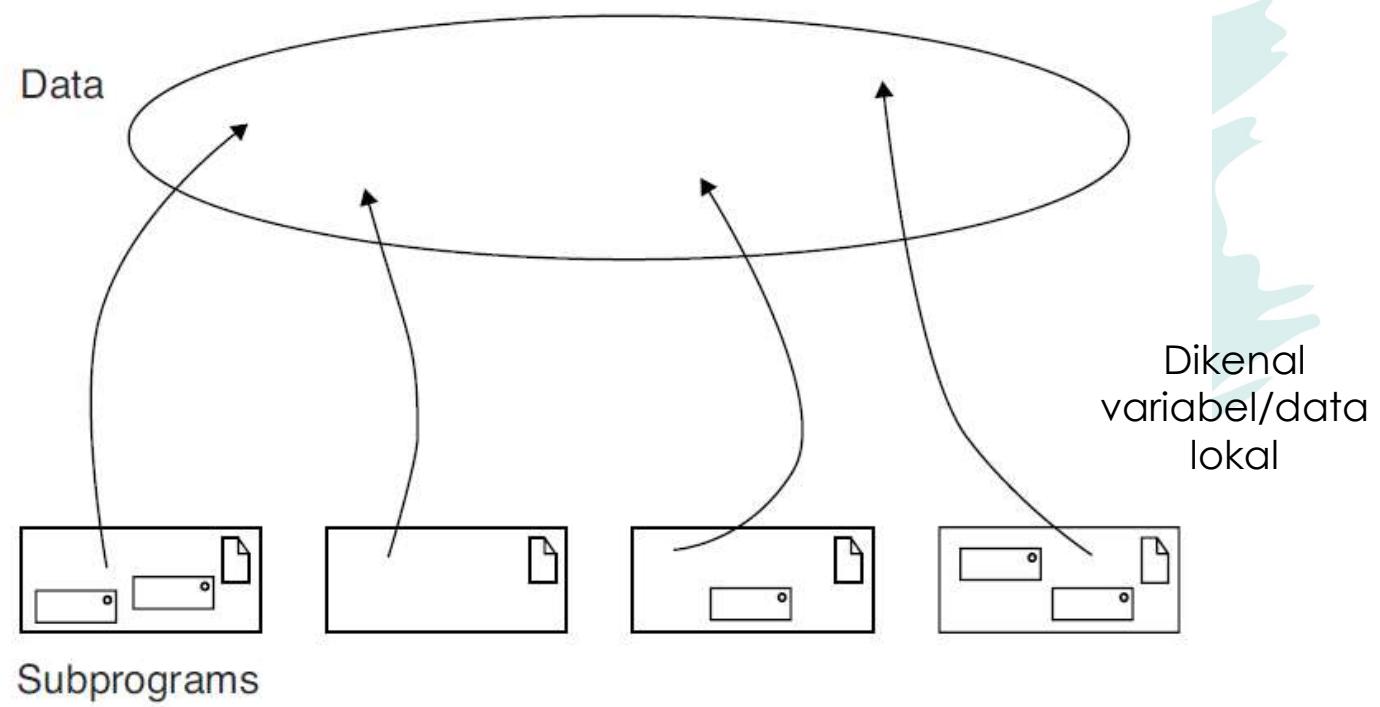
# Sejarah

- 1970an: Desain dan Pemrograman Terstruktur
  - Awal strukturisasi program
  - Dipengaruhi oleh bahasa-bahasa terstruktur seperti Pascal
  - Pendekatan solusi untuk Program skala besar (terutama masalah modularitas, maintenance, upgrade)
- 1980: Desain dan Pemrograman Objek
  - Perkembangan RPL
  - Pemrograman Modular (ADA)
  - Pemrograman Objek (C++)
  - Metode perancangan berorientasi objek
- 1990: Pemrograman objek, Desain dan Analisis
  - Metode analisis dan desain berorientasi objek (Rumbaugh's OMT, Booch, dll.)

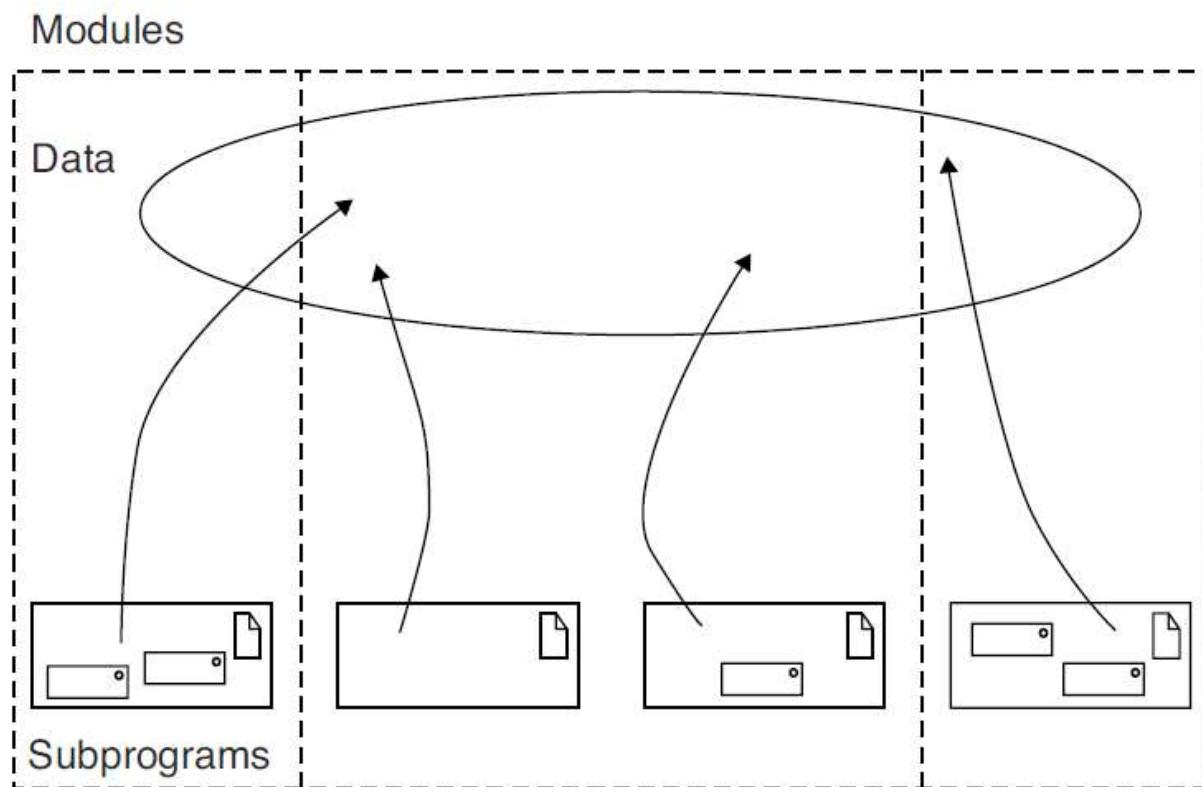
# *Topologi Bahasa Pemrograman Generasi I dan 2 (sebelum 1960an)*



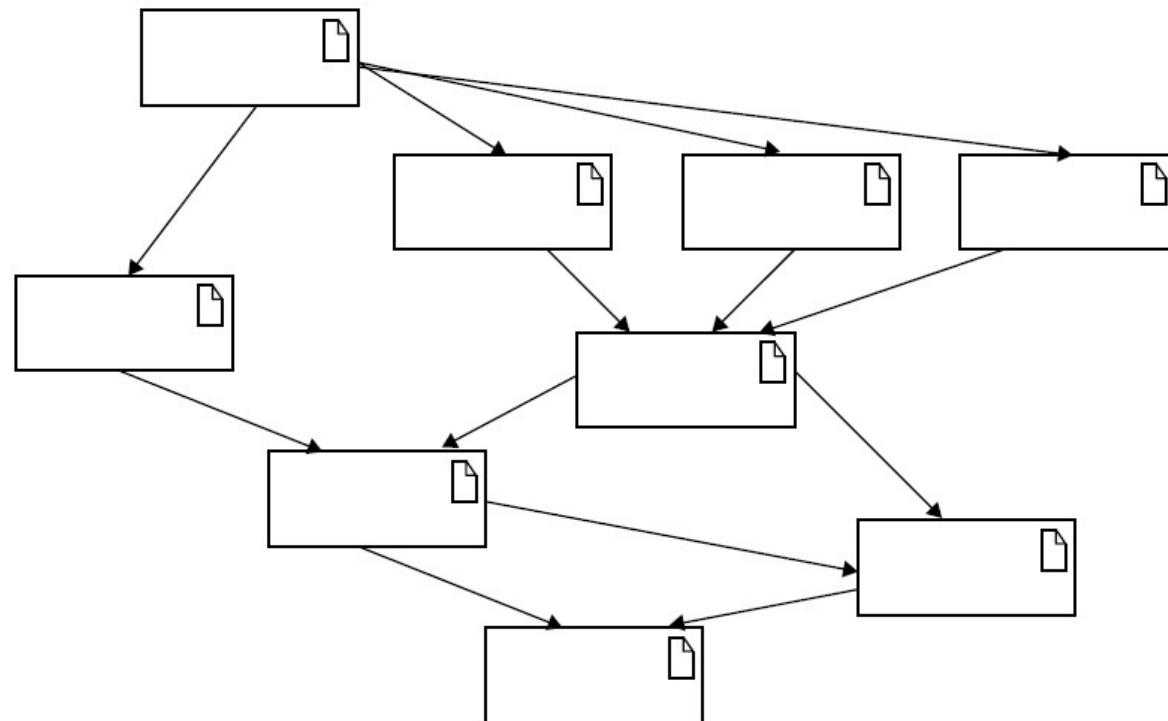
# ***Topologi Bahasa Pemrograman akhir generasi ke 2, awal ke-3 (1960-1970an)***



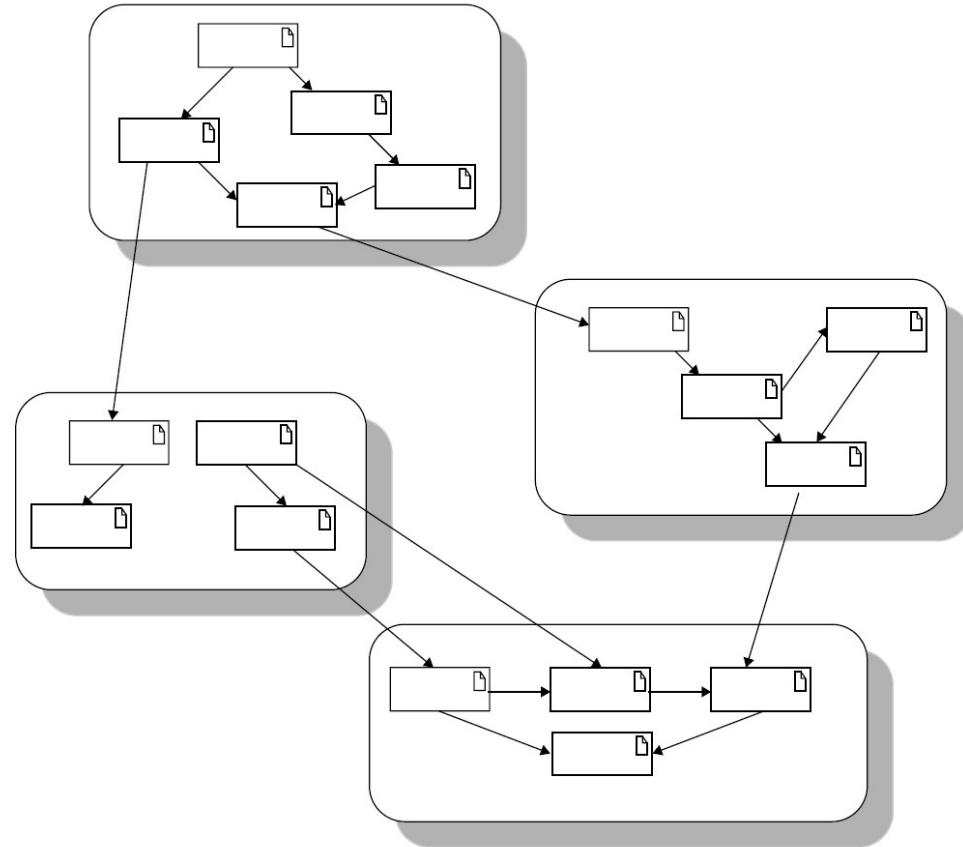
# *Topologi Bahasa Pemrograman akhir generasi ke-3 (1970-1980an)*



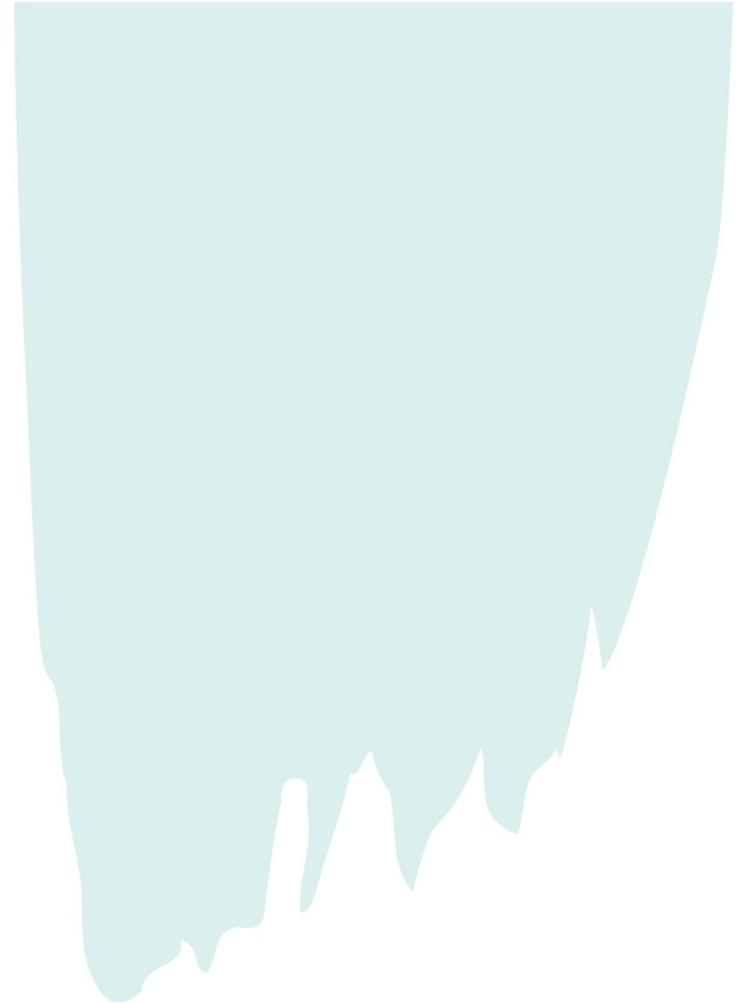
# ***Topologi Bahasa Pemrograman Small-middle scale Application dengan OO (1990an)***



# *Topologi Bahasa Pemrograman Large-scale Application dengan OO (1990- sekarang)*



# *Kelas dan Objek*

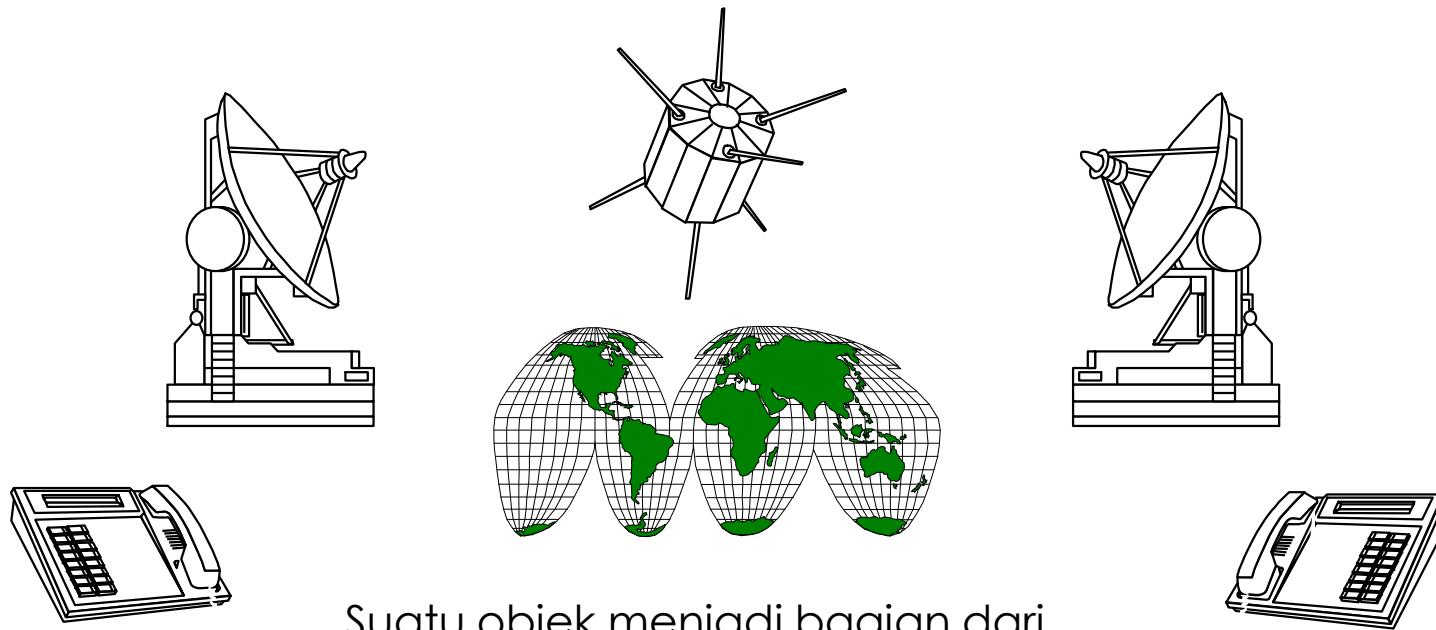


# Objek

13

## POSTULATE

Dunia nyata dibentuk oleh entitas otonom yang saling berinteraksi



Suatu objek menjadi bagian dari pengetahuan terhadap suatu domain tertentu



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling

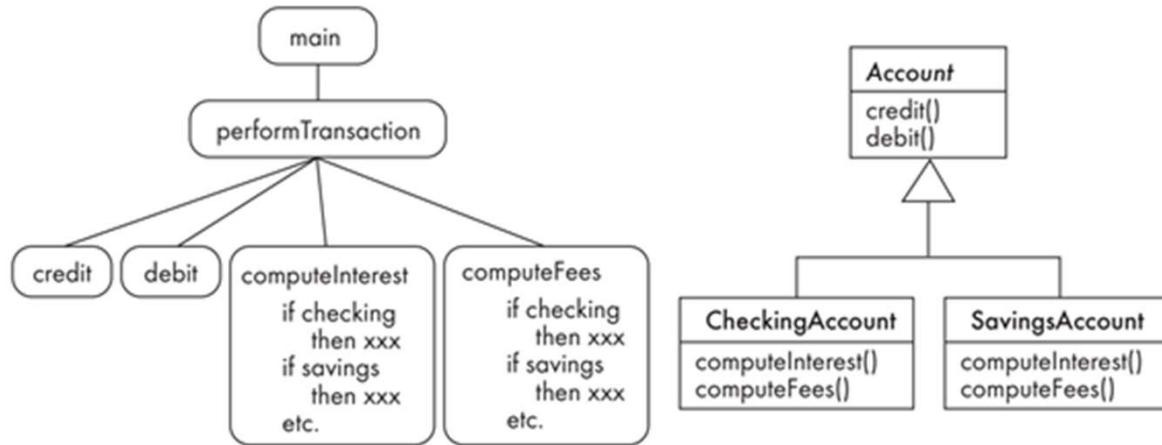
# *Review of object orientation*

- Teknik Object-oriented (OO) didasarkan pada penggunaan kelas yang bertindak sebagai abstraksi data, dan yang berisi seperangkat prosedur yang bertindak atas data tersebut.
- Sekarang diakui secara luas bahwa orientasi objek adalah pendekatan desain yang efektif untuk mengelola kompleksitas yang melekat pada sebagian besar sistem besar
- Sistem berorientasi objek memanfaatkan abstraksi untuk membantu membuat perangkat lunak menjadi tidak terlalu rumit.
- Abstraksi adalah sesuatu yang membebaskan dari keharusan berurusan dengan detail.
- Sistem berorientasi objek menggabungkan (a) abstraksi prosedural/paradigma procedural dengan (b) abstraksi data.

# *Review of object orientation (2)*

- Paradigma berorientasi objek adalah pendekatan untuk solusi masalah di mana semua perhitungan dilakukan dalam konteks objek.
- Objek adalah contoh konstruksi pemrograman, biasanya disebut kelas, yang merupakan abstraksi data dan yang berisi abstraksi prosedural yang beroperasi pada objek.
- Dalam paradigma berorientasi objek, program yang berjalan dapat dilihat sebagai kumpulan objek yang berkolaborasi untuk melakukan tugas yang diberikan.

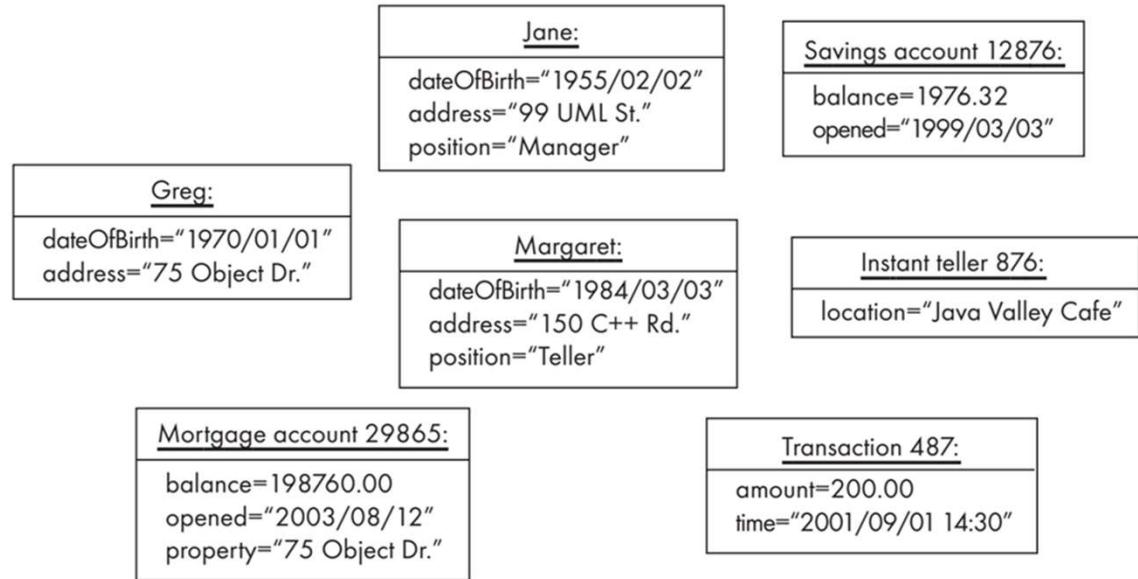
# *Review of object orientation (3)*



- Perbedaan penting antara paradigma berorientasi objek dan prosedural.
  - Dalam paradigma prosedural (gambar kiri), kode diatur ke dalam prosedur yang masing-masing memanipulasi berbagai jenis data.
  - Dalam paradigma berorientasi objek (gambar kanan), kode diatur ke dalam kelas yang masing-masing berisi prosedur untuk memanipulasi instance dari kelas itu saja.

# Objek (I)

- Objek adalah data terstruktur dalam sistem perangkat lunak yang berjalan, yang bisa mewakili apa saja yang bisa dikaitkan dengan properti dan perilaku.
- Properti mencirikan objek, menggambarkan keadaan saat ini.
- Perilaku adalah cara objek bertindak dan bereaksi, mungkin mengubah kondisinya.



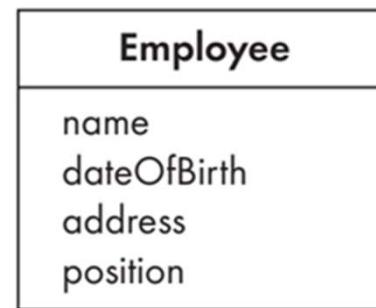
# Objek (2)

- Berikut ini adalah beberapa contoh objek lainnya:
  - Dalam program penggajian, akan ada objek yang mewakili setiap karyawan.
  - Dalam program pendaftaran universitas, akan ada objek yang mewakili setiap siswa, setiap kursus dan setiap anggota fakultas.
  - Dalam sistem otomasi pabrik, mungkin ada objek yang mewakili setiap jalur perakitan, setiap robot, setiap item yang diproduksi, dan setiap jenis produk.
- Pada contoh di atas, semua objek mewakili hal-hal yang penting bagi pengguna program.
- Penggunakan proses yang sering disebut analisis berorientasi objek untuk memutuskan objek mana yang penting bagi pengguna, dan untuk mengetahui struktur, hubungan, dan perilaku objek-objek ini.

# *Kelas dan Instannya (I)*

- Kelas adalah unit abstraksi data dalam program berorientasi objek. Lebih khusus lagi, kelas adalah modul perangkat lunak yang mewakili dan mendefinisikan satu set objek yang sama, instansnya. Semua objek dengan properti dan perilaku yang sama adalah contoh dari satu kelas.
- Gambar sebelumnya menunjukkan bagaimana karyawan bank Jane dan Margaret dapat direpresentasikan sebagai instan kelas `Employee` yang menyatakan bahwa semua mesin virtualnya memiliki:

- `name`,
- `dateOfBirth`,
- `address`, dan
- `position`



IF2250- Class Modeling



KNOWLEDGE & SOFTWARE ENGINEERING

# *Kelas dan Instannya (2)*

- Sebagai modul perangkat lunak, kelas berisi semua kode yang berkaitan dengan objeknya, termasuk:
  - Kode yang menggambarkan bagaimana objek kelas disusun - yaitu data yang disimpan dalam setiap objek yang mengimplementasikan properti.
  - Prosedur, yang disebut method, yang menerapkan perilaku objek.
- Dengan kata lain, selain mendefinisikan properti seperti name dan address, seperti yang ditunjukkan pada di atas, kelas Employee juga akan menyediakan method untuk menciptakan employee baru, dan mengubah name, address, dan position dari employee.

# *Kelas dan Instannya (3)*

- Dua aturan untuk membantu memutuskan apa yang harus menjadi kelas dan apa yang harus menjadi instance:
  - Secara umum, sesuatu harus berupa kelas jika dapat memiliki instance.
  - Secara umum, sesuatu harus menjadi contoh jika itu jelas merupakan anggota tunggal dari set yang ditentukan oleh kelas.
- Misalnya, dalam aplikasi untuk mengelola rumah sakit, salah satu kelas mungkin adalah Doctor, dan yang lain mungkin Hospital. Anda mungkin berpikir bahwa Hospital harus menjadi instance jika hanya ada satu dari mereka dalam sistem; Namun, fakta bahwa secara teori mungkin ada beberapa Hospital memberi tahu kita bahwa Hospital harus kelas.

# Apa itu objek

- Objek

- Dapat 'ditemukan' (*discoverable*) dan dapat dipisahkan dari lingkungannya
- Memiliki karakteristik yang dapat diamati
- Dikelilingi oleh objek lain
- Bereaksi terhadap suatu event yang terjadi pada suatu lingkungan atau event yang terjadi karena suatu aksi
- Menginformasikan lingkungannya atau event yang terkait dengan lingkungannya

- Model Objek

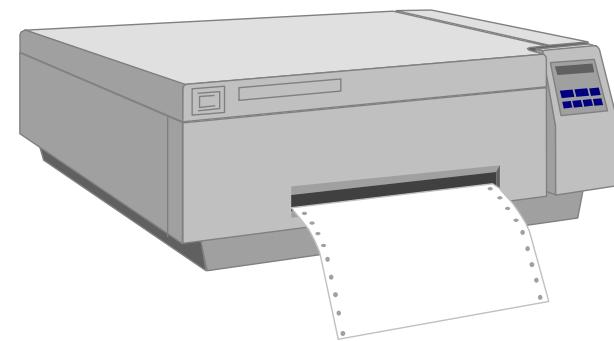
- Objek yang nyata dalam dunia nyata
- Suatu konstruksi hasil buah pikiran (account bank, jalur )
- Suatu sistem atau aplikasi adalah hasil kerjasama antar objek

# Apa itu objek

- Suatu objek mewakili suatu individu, item, unit atau entitas
  - Bisa abstrak atau juga nyata, dengan peran yang jelas dalam suatu domain masalah [Smith and Tockey]
- *...semua yang memiliki batas terdefinisi dengan jelas [Cox]*
- *Objek memiliki status (state), perilaku dan identitas [Booch]*
  - *Struktur dan perilaku dari objek yang sama didefinisikan dalam suatu kelas yang umum*
    - *Istilah instansiasi dan objek memiliki arti yang sama*

# *Objek memiliki status (state)*

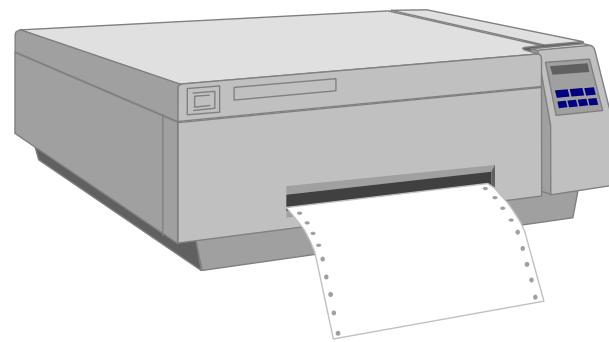
- Example:
  - PRINTER    Weight  
Size  
Serial number  
ON/OFF  
Number of sheets in the tray  
Number of jobs queued  
...  
...



*State suatu objek meliputi semua karakteristik (umumnya statik) dari objek termasuk nilai dari tiap karakteristik (nilai ini bersifat dinamik) [Booch]*

# Objek memiliki Perilaku (*Behavior*)

- Contoh
  - Switch on / Switch off
  - Send a print job
  - Print one page
  - Display an error message
  - ...

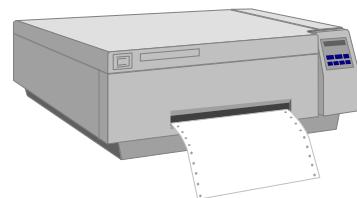


*Perilaku adalah bagaimana suatu objek beraksi dan bereaksi yang ditandai dengan adanya perubahan state dan juga pengiriman pesan (message passing) [Booch].*

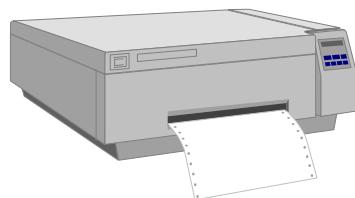
*State dari suatu objek adalah menunjukkan hasil kumulatif dari perilakunya  
[Booch]*

# *Objek memiliki identitas*

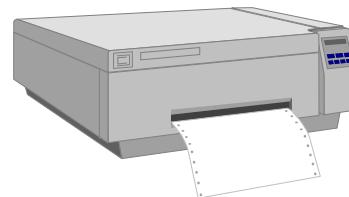
LPT1:



LPT2:



LPT3:



*Identitas adalah karakteristik dari objek yang membedakannya dengan objek lain [Khoshafian and Copeland]*



KNOWLEDGE &amp; SOFTWARE ENGINEERING

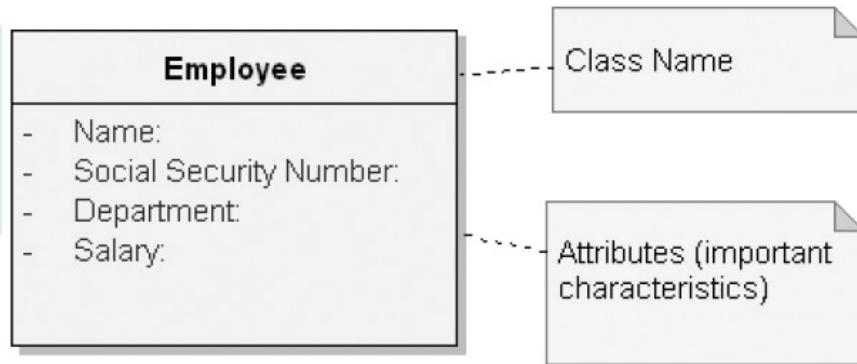
IF2250- Class Modeling

# Kelas

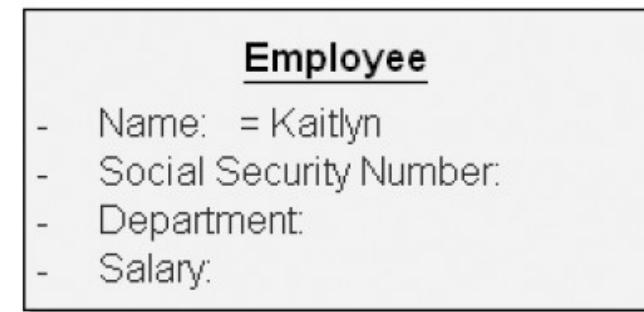
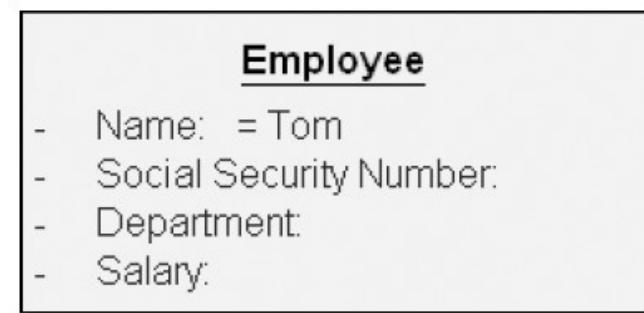
- Suatu objek yang menjadi bagian dari suatu kelas artinya objek tersebut adalah instansiasi dari kelas atau contoh dari kelas itu [Booch]
- Hasil instansiasi dari suatu kelas
  - Memiliki nilai atributnya sendiri (yang spesifik)
  - Nama atribut dan event dapat digunakan dengan objek lain



KNOWLEDGE & SOFTWARE ENGINEERING



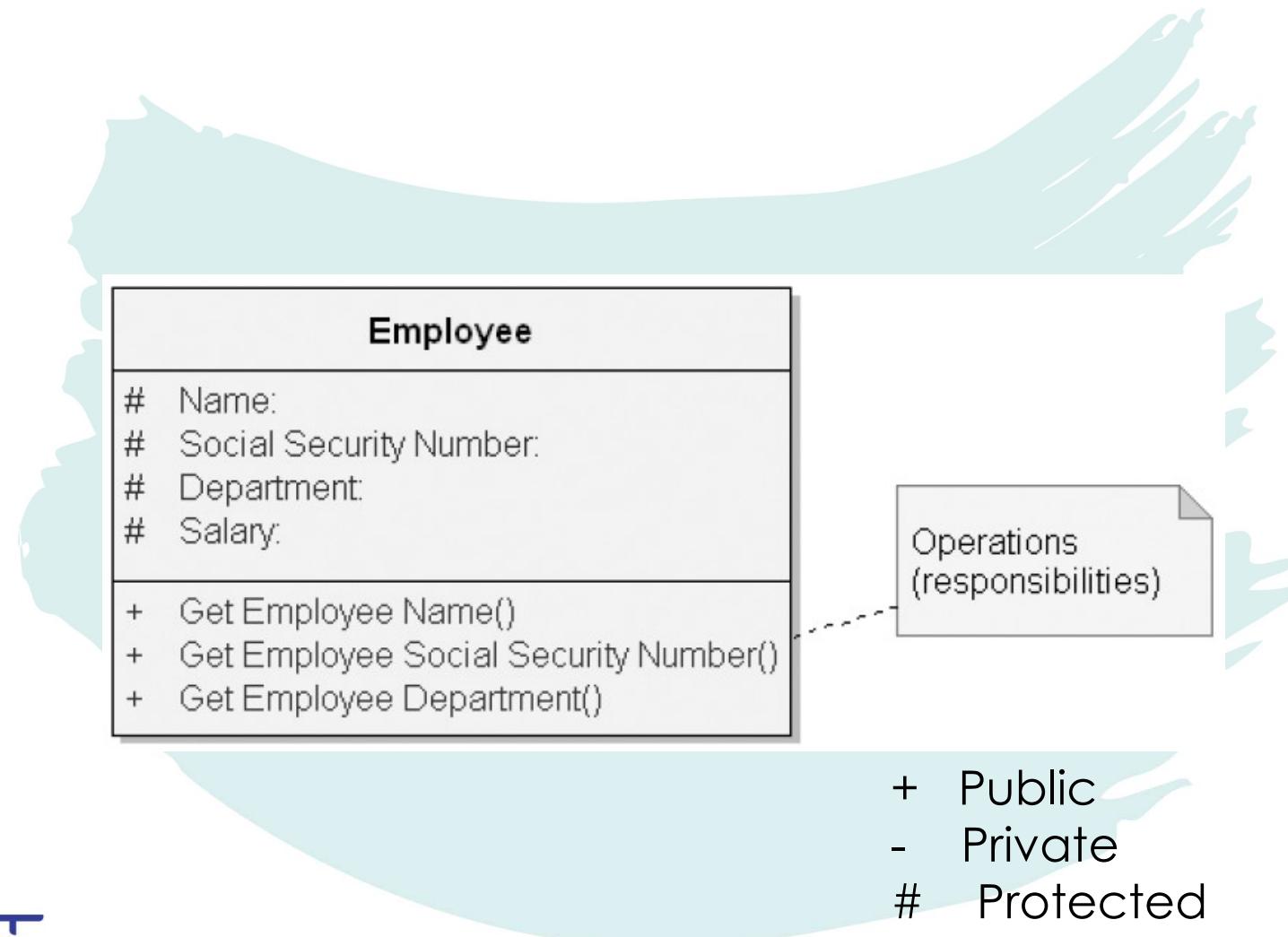
# *Kelas Employee*



# Objek Employee



KNOWLEDGE & SOFTWARE ENGINEERING



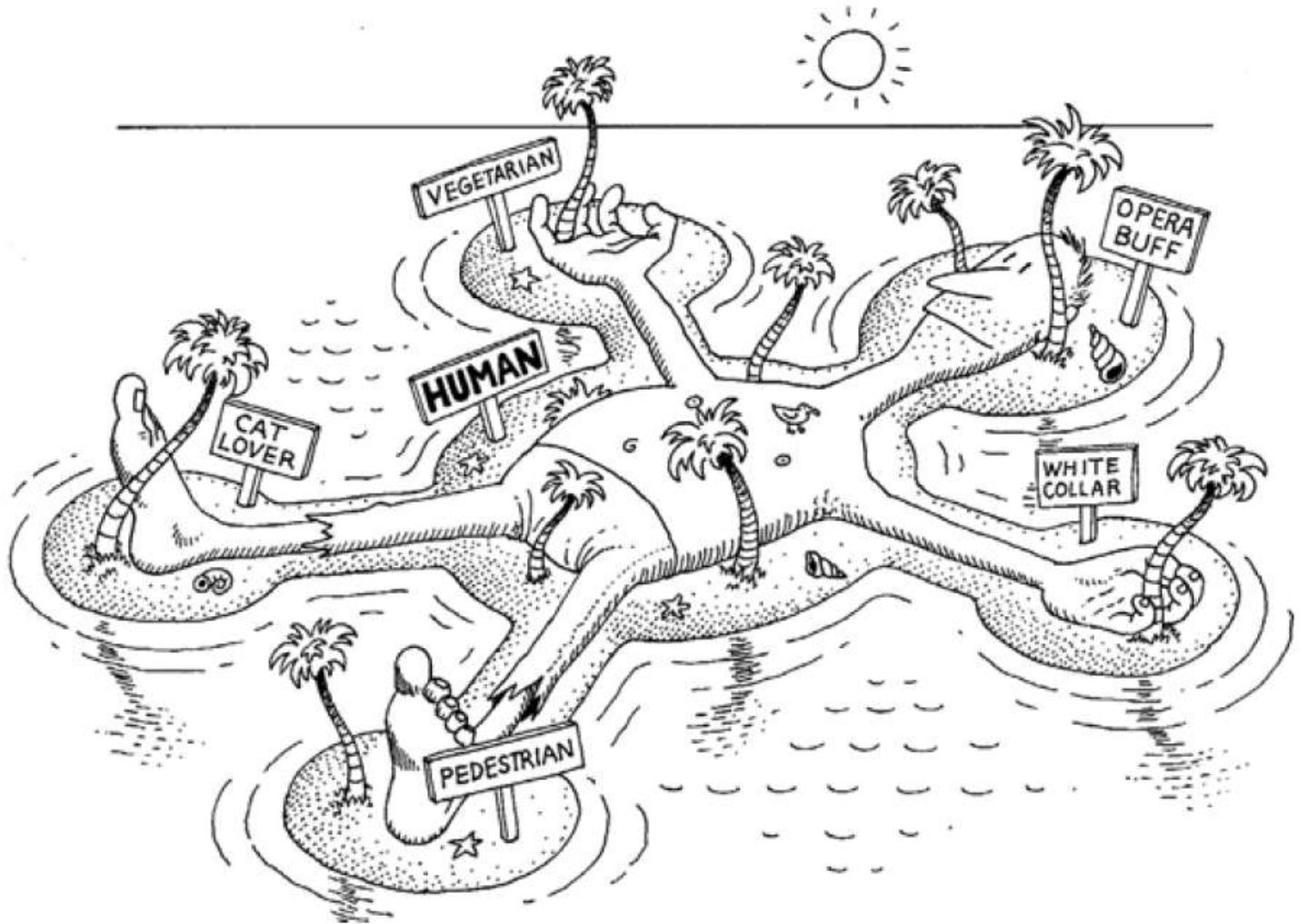
# **Perilaku (*Behavior*)/ Operasi/ metode/procedure/function**

- Suatu metode menjelaskan aksi (tiap langkah), dan reaksi dari suatu objek, sesuai dengan kelasnya
  - Hasilnya: **perubahan state** atau **message passing**
- Suatu metode dieksekusi dari suatu objek, yang terikat pada suatu kelas
- Objek bisa berperan dalam peran yang berbeda



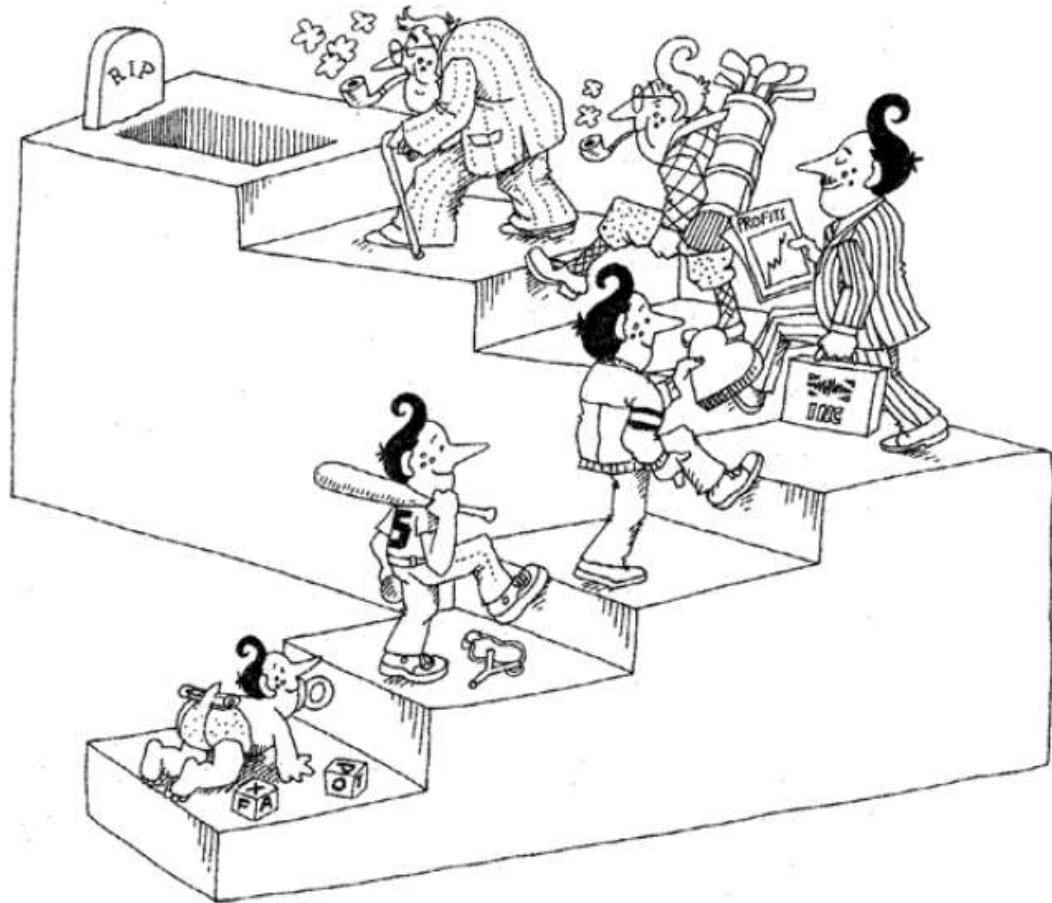
KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling



Objects can play many different roles.





KNOWLEDGE & SOFTWARE ENGINEERING

Objects play many different roles during their lifetimes.

IF2250- Class Modeling

# Kelas vs. Objek

	<b>Class</b>	<b>Objek</b>
Definisi	Kelas adalah mekanisme penggabungan atribut dan metode/operasi dalam satu unit tunggal	Objek adalah instansiasi dari suatu kelas atau variabel dari suatu kelas
Keberadaan	Ada secara logik	Ada secara fisik (ada di memori)
Alokasi Memori	Tidak ada alokasi memori	Objek dialokasikan tempat di memori saat dibuat
Life-Span	Tidak ada	Sesuai dengan definisi publik(global) atau lokal)
Pendefinisian	Hanya perlu didefinisikan satu kali	Bisa dibuat sesuai kebutuhan

Dalam bahasa C, atau bahasa lain, diketahui tipe data Integer. Tipe Integer ini dapat dilihat sebagai konsep suatu kelas yang memiliki atribut integer (bilangan negatif, nol dan positif, tapi bukan bilangan pecahan). Suatu variabel X yang bertipe kelas Integer, akan memiliki sifat kelas integer. X adalah objek dari kelas integer.



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling

# *Pemodelan Berbasis Kelas*

- Pemodelan ini merepresentasikan manipulasi terhadap objek, operasi yang diaplikasikan pada objek dan kolaborasi yang terjadi antar kelas dari objek
- Elemen yang digunakan pada pemodelan ini
  - Kelas, Objek, Atribut, Operasi, Kolaborasi, dan Paket



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling

# *Tahapan Pembuatan Diagram Kelas*

1. Identifikasi Kelas
2. Penentuan Atribut
3. Penentuan Operasi
4. Penentuan Hubungan (asosiasi) Antar Kelas
5. Membuat Diagram Kelas
6. Membuat Diagram Paket (jika perlu)



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling

# I. Identifikasi kelas

- Pelajari masalah (Problem Statement)
  - Kenali kata ‘BENDA’ yang terkait dengan keluar/masuk informasi
    - Entitas eksternal (sistem, alat, orang)
    - Benda-benda seperti laporan, layar, surat, sinyal
    - Kemunculan atau kejadian (event) selama sistem berjalan
    - Peran (Roles) misalnya manajer, insinyur, sales
    - Unit organisasi (divisi, group, team)
    - Tempat/Lokasi
    - Struktur (sensor, mobil, komputer)

# *Kriteria pemilihan Kelas*

- Berisi informasi yang harus dijaga/disimpan
- Memberikan suatu bentuk layanan
- Berisi beberapa atribut
- Punya atribut yang sama yang terpakai di semua instans dari kelas
- Punya operasi yang sama yang terpakai di semua instans dari kelas
- Mewakili entitas eksternal yang menghasilkan atau menggunakan informasi



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling

# *Kelas untuk Analisis*

- Kelas Entitas (Entity Class)/ Kelas Model/ Kelas Bisnis
  - Diambil dari masalah yang akan dipecahkan
  - Biasanya disimpan di basisdata, dan digunakan selama aplikasi berjalan
- Kelas Batas (Boundary Class)
  - digunakan sebagai interface/ antarmuka yang dilihat atau digunakan sebagai interaksi dengan/ oleh pengguna
- Kelas Pengendali (Controller Class)
  - Menangani suatu unit pekerjaan dari awal hingga selesai
    - Create/ update objek entitas
    - Instansiasi objek batas
    - Melakukan komunikasi antara sekumpulan objek
    - Validasi komunikasi data antar aktor



KNOWLEDGE & SOFTWARE ENGINEERING

## 2. Penentuan Atribut dari Kelas

- Periksa cerita pemrosesan atau use case/scenario dari pengguna
  - Pilih karakteristik yang mewakili suatu kelas atau menjadi ciri suatu kelas
- Temukan item data yang mendefinisikan kelas ini sesuai konteks permasalahannya.
  - Item data bisa sederhana (elementary) atau item data gabungan (composite)



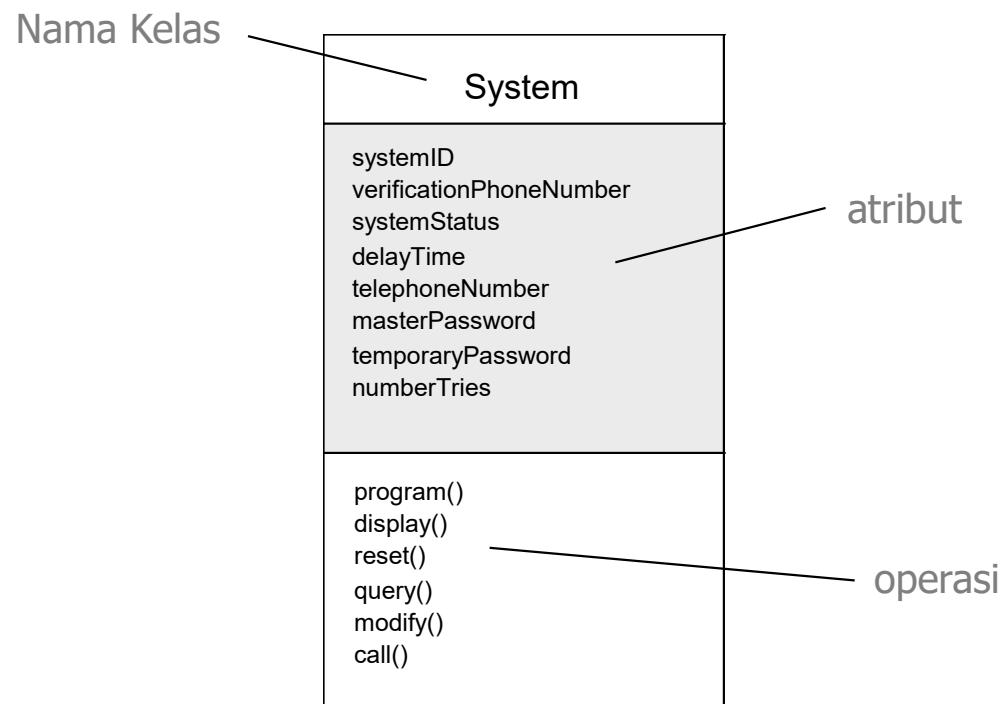
KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling

### 3. Penentuan Operasi

- Cari kata ‘kerja’ dalam cerita/skenario tadi, dan identifikasi operasi yang kira-kira menjadi milik suatu kelas. Misalnya, terkait dengan:
  - Manipulasi data
  - Melakukan komputasi
  - Pertanyaan tentang state suatu objek
  - Memantau objek untuk melihat kemunculan suatu even pengendali
- Operasi dipecah menjadi operasi yang lebih sederhana/kecil jika dibutuhkan
- Perhatikan juga komunikasi yang terjadi antara objek dan definisikan operasi yang dibutuhkan

# Class



KNOWLEDGE &amp; SOFTWARE ENGINEERING

IF2250- Class Modeling

## *4. Penentuan hubungan antar kelas*

- Asosiasi
- Agregasi
- Komposisi
- Dependensi
- Generalisasi (*inheritance*)



KNOWLEDGE & SOFTWARE ENGINEERING

# Asosiasi

- Dari hasil analisa kelas-kelas dari suatu sistem, sering ditemukan keterkaitan secara semantik antara dua kelas.
  - Keterkaitan ini disebut sebagai hubungan asosiasi
    - Hubungan ini juga dapat diperjelas dengan hubungan multiplicity (kardinalitas dalam ER diagram)



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling

# Asosiasi dan Links

- Link dan asosiasi membentuk keterhubungan antar objek atau kelas
  - Link: menghubungkan dua instans objek
  - Asosiasi (association): sekumpulan link dengan struktur dan semantik yang sama



- Cara membaca suatu asosiasi, biasanya dari kiri ke kanan atau dari atas ke bawah (kecuali dinyatakan arahnya dengan ◀ or ▶)

# Multiplicity

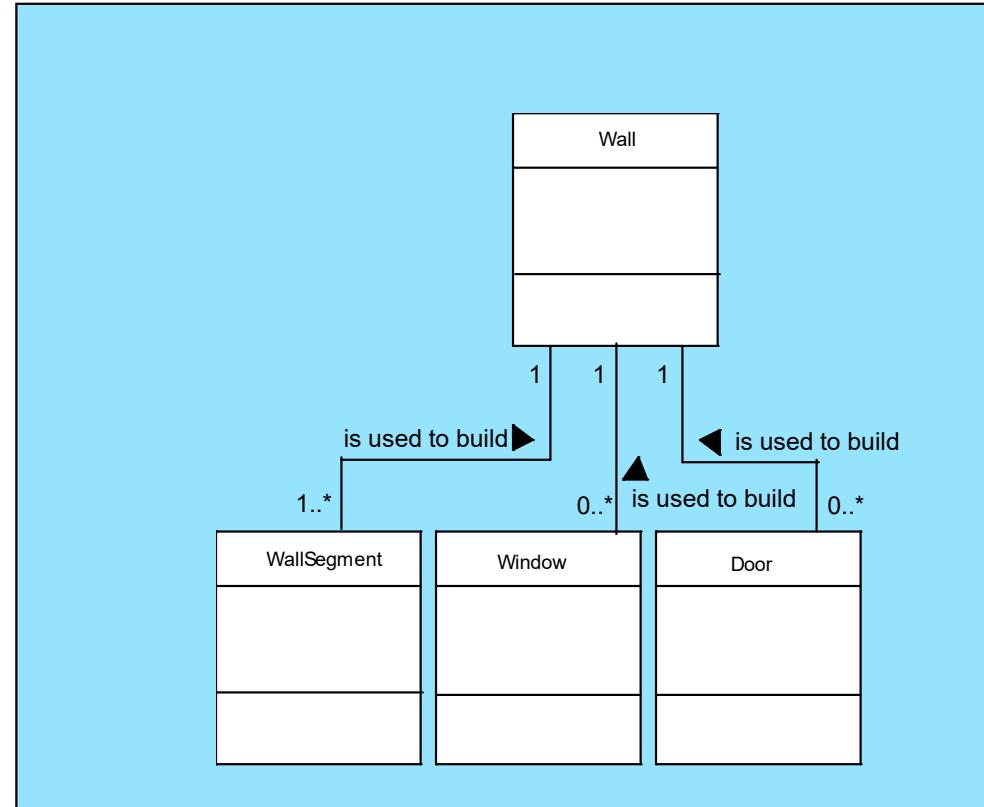
- Digunakan untuk menunjukkan jumlah instansiasi yang mungkin terjadi pada suatu asosiasi



Seorang dosen  
mungkin mengajar  
banyak Prodi atau  
mungkin tidak  
mengajar

Suatu Prodi mungkin  
diajari oleh banyak  
dosen atau tidak ada  
dosen

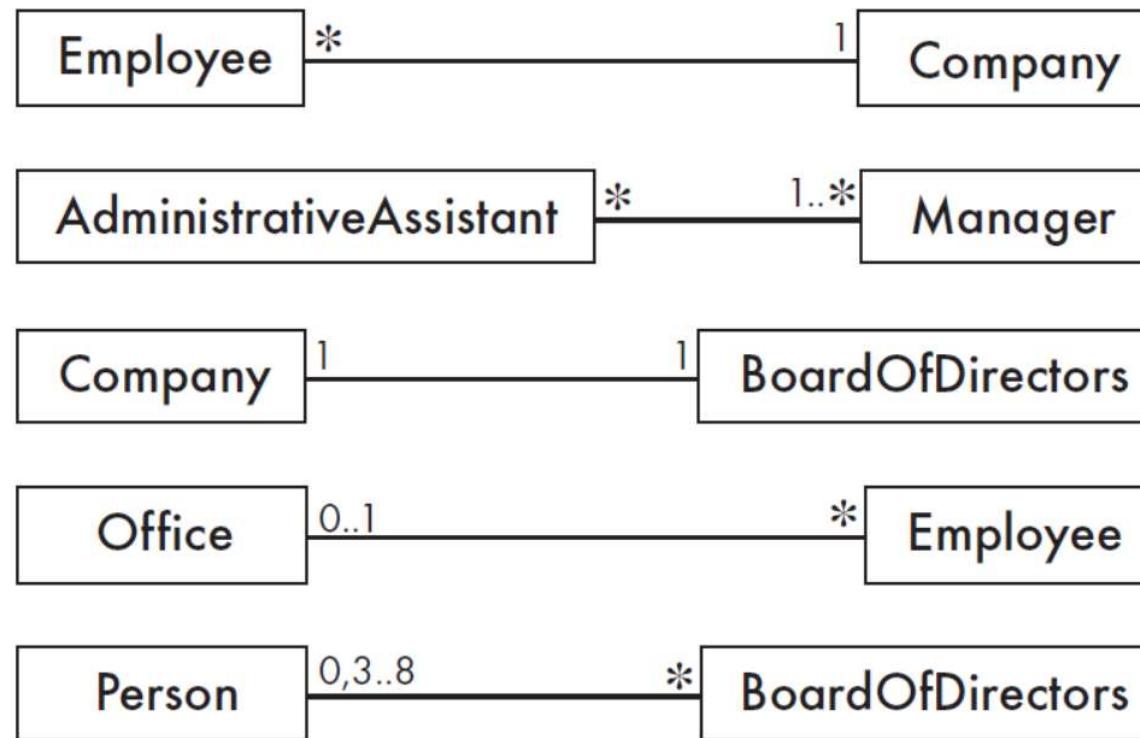
# Multiplicity



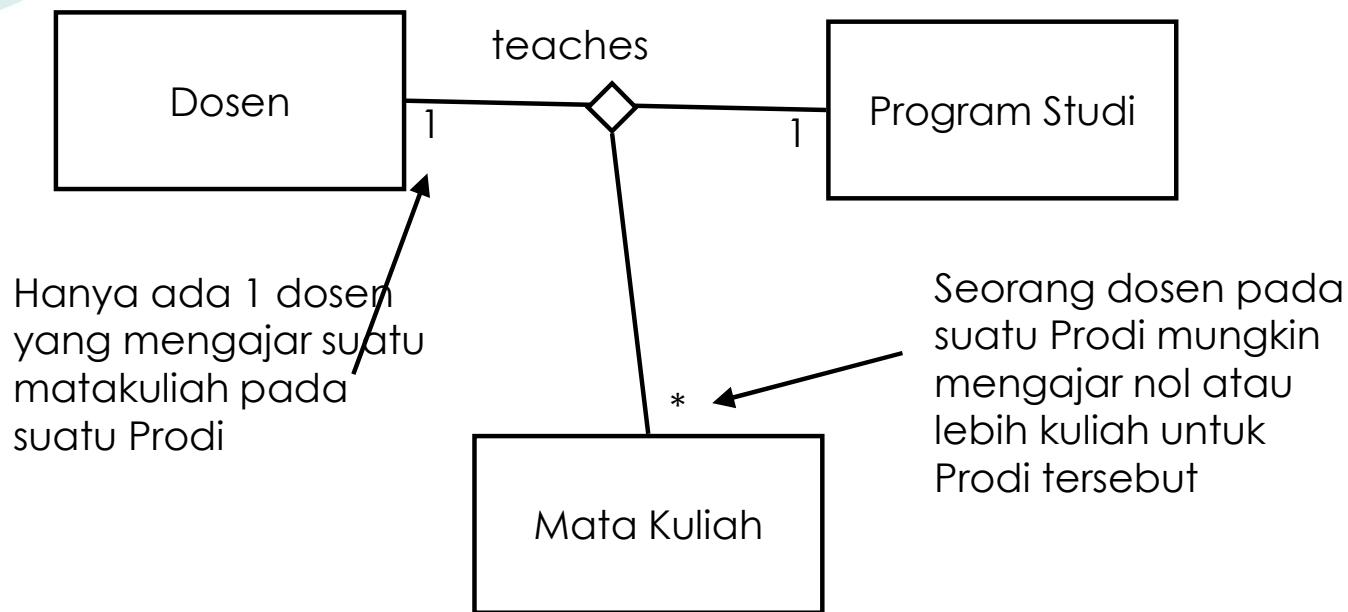
KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling

## Contoh: Multiplicity

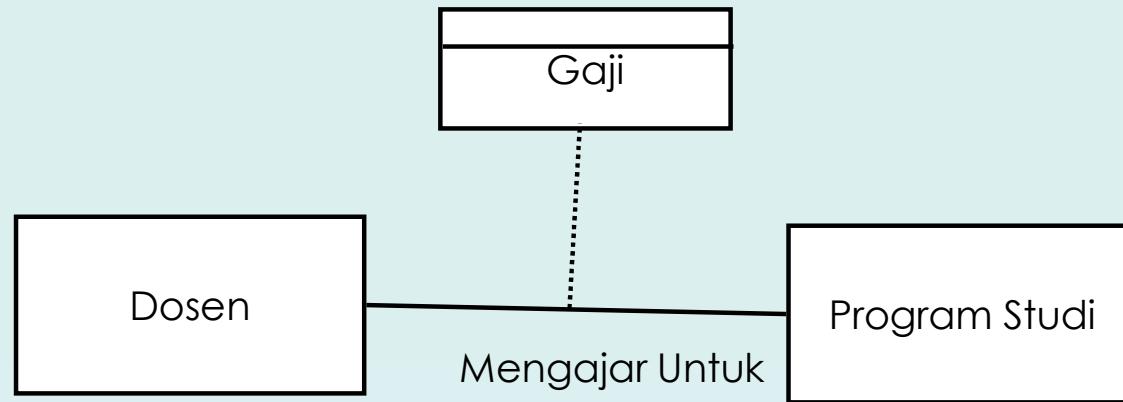


# *Asosiasi N-ary (N-Ary Associations)*



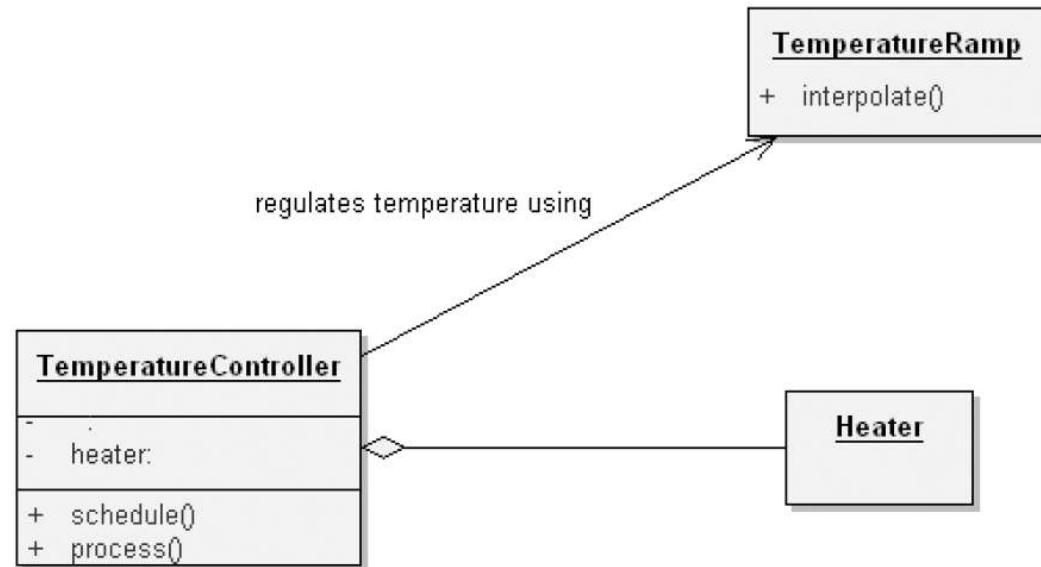
Hubungan n-ary sering membingungkan, sehingga kalau bisa dihindari tetapi jika diperlukan maka perlu disertai dengan penjelasan tertulis.

## *Atribut pada Asosiasi*



# Agregasi

- Agregasi lebih ke hirarki '**whole-part**'
  - Dari 'whole' (semua) ke 'part' (bagian)



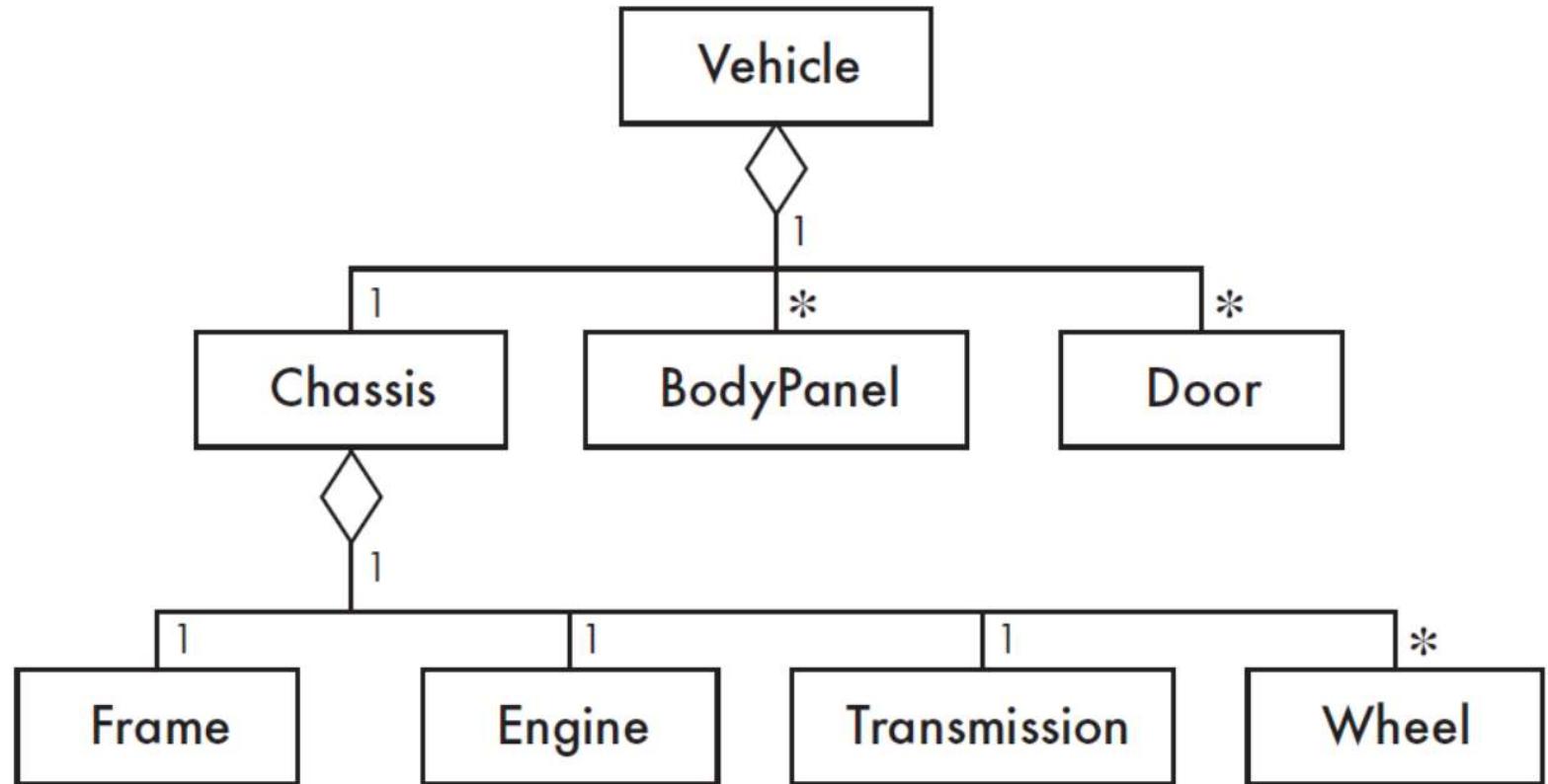
## Agregasi ‘fisik’

- Pesawat
  - Sayap, mesin, ban

## Agregasi ‘konsep’

- PemegangSaham (ShareHolder)
  - Pemegang saham lain

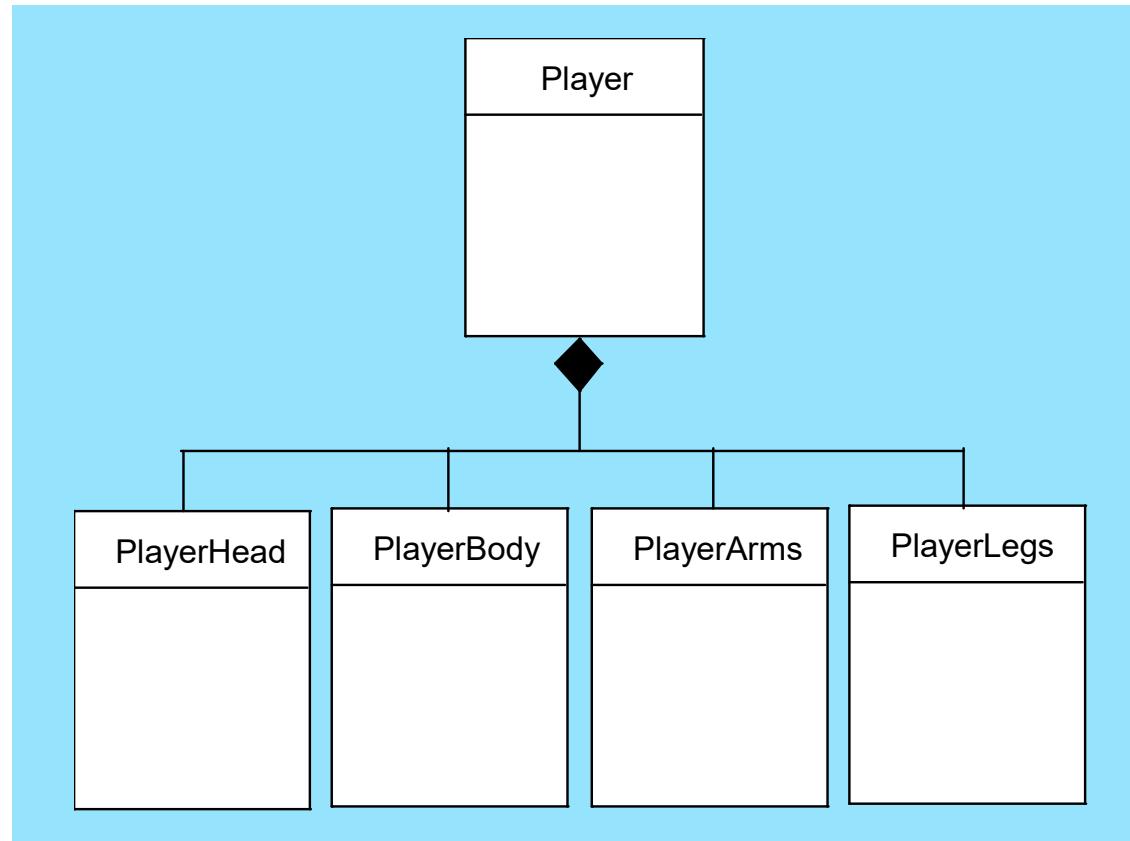
# Contoh Agregasi



KNOWLEDGE &amp; SOFTWARE ENGINEERING

IF2250- Class Modeling

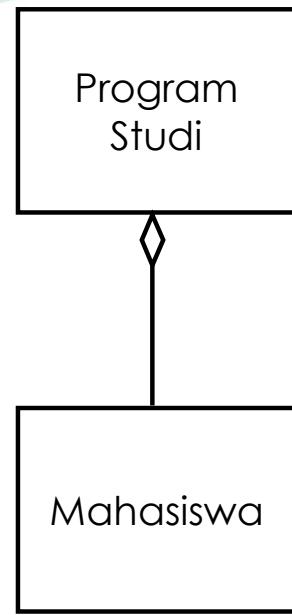
# Komposisi



KNOWLEDGE & SOFTWARE ENGINEERING

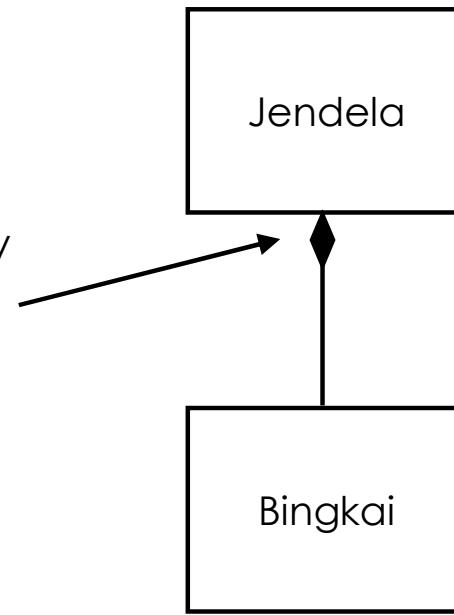
IF2250- Class Modeling

# *Aggregasi (Part-Of)*



## **Agregasi**

(Mahasiswa adalah bagian dari Program Studi, dan suatu program studi mungkin tidak memiliki mahasiswa)



## **Komposisi**

(suatu bingkai harus ada untuk membentuk jendela)



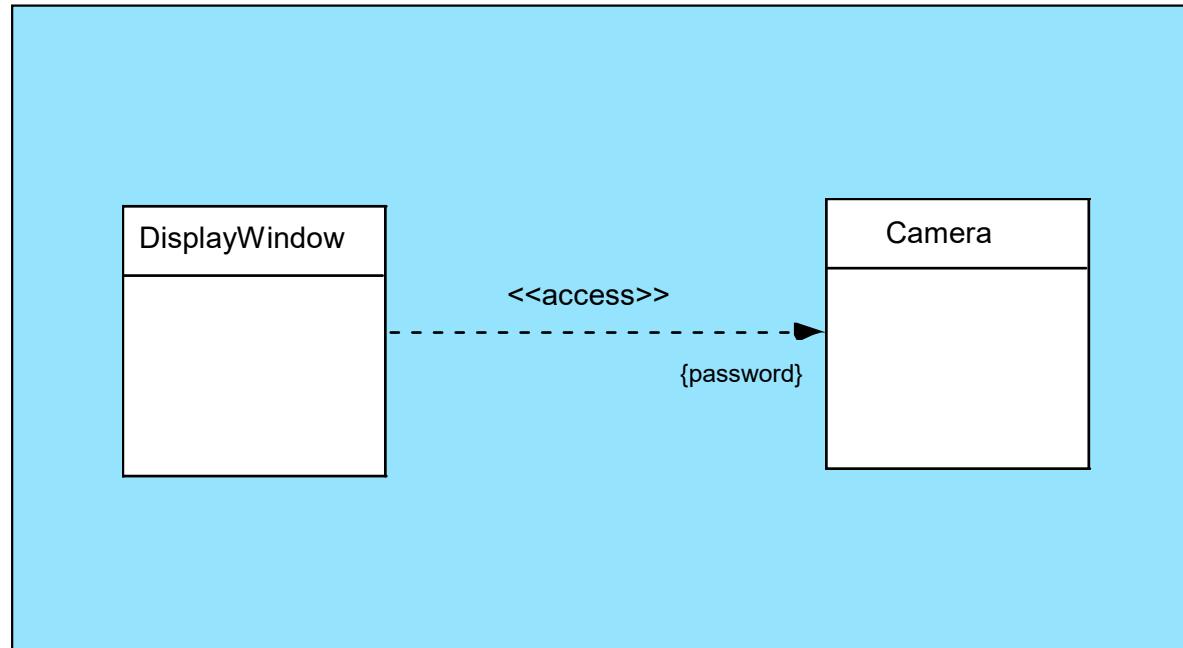
# Dependensi

- Hasil analisis juga dapat menunjukkan hubungan ketergantungan/*Dependencies/ client-server* antar kelas
  - Kelas Client bergantung pada Kelas Server



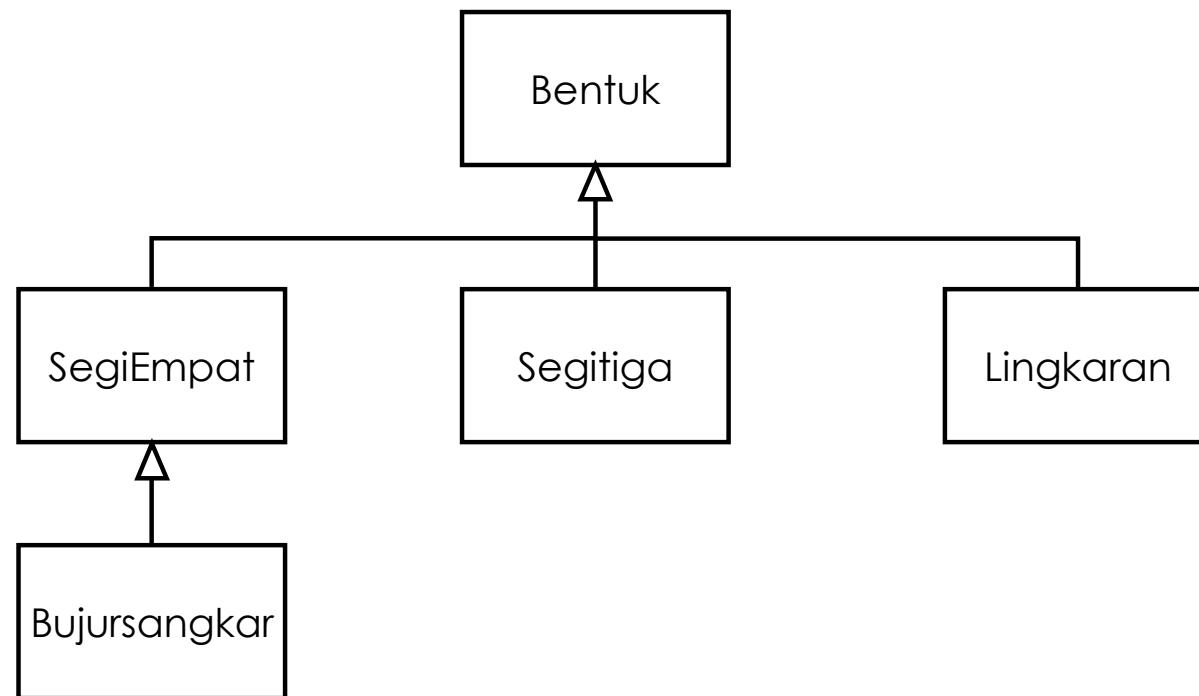
KNOWLEDGE & SOFTWARE ENGINEERING

# Contoh dependensi



Kelas **DisplayWindow** memerlukan password untuk mengakses kelas **Camera**

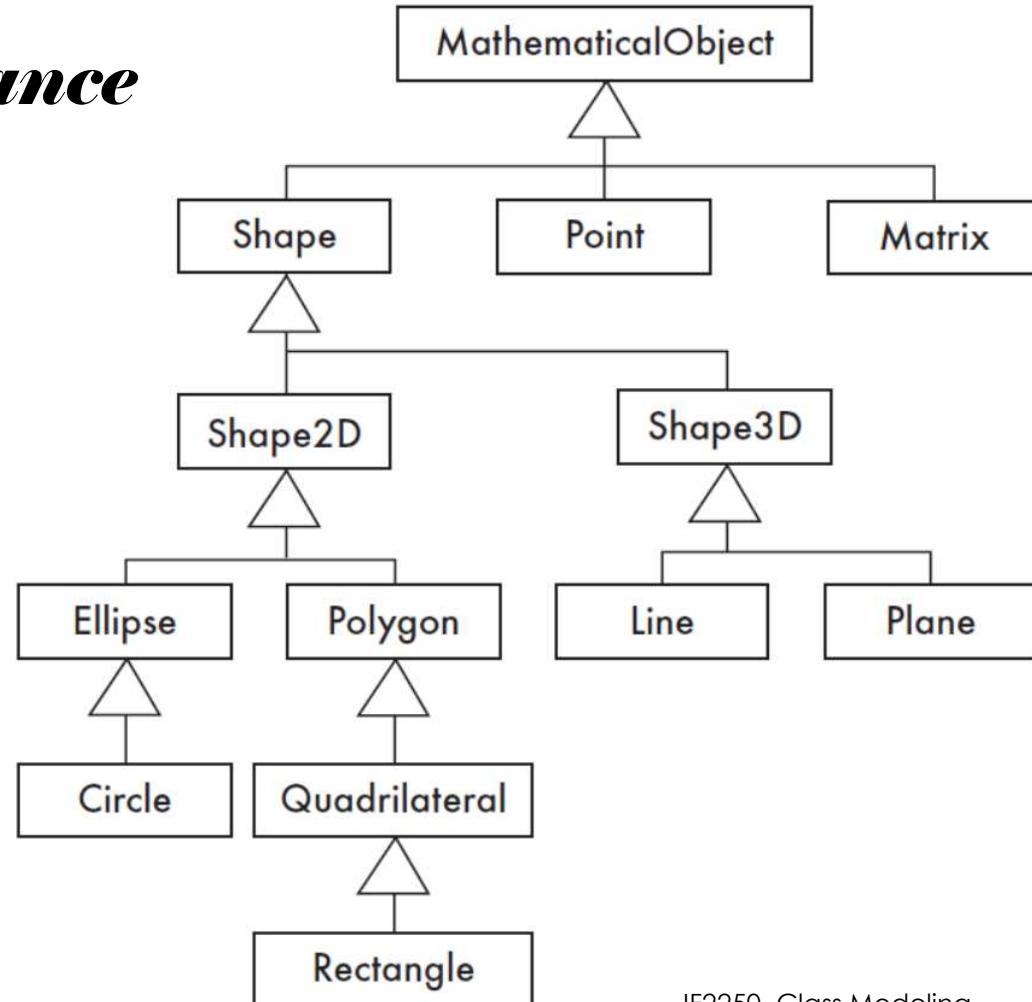
# Generalisasi (a.k.a. Inheritance, is-a)



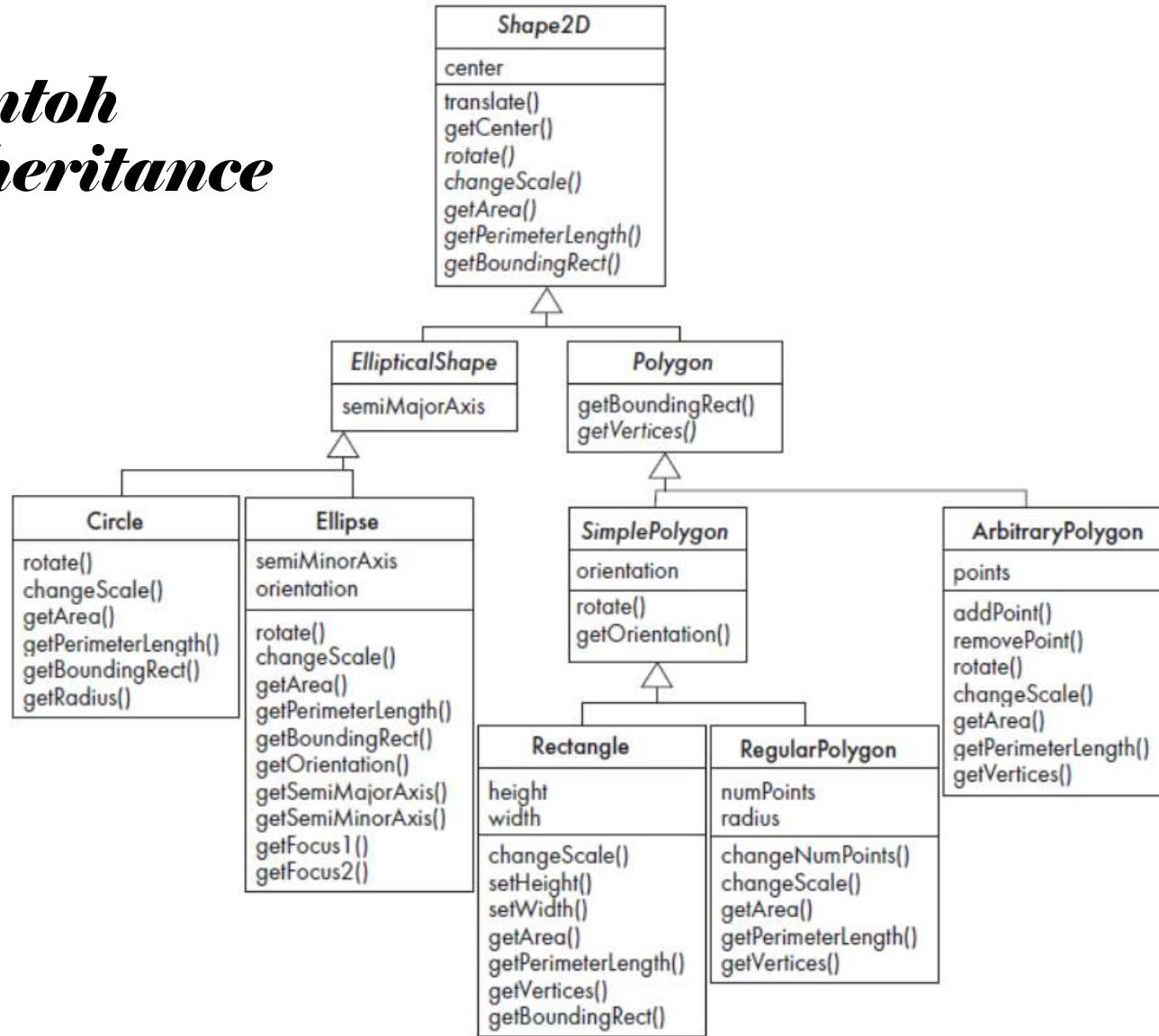
KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling

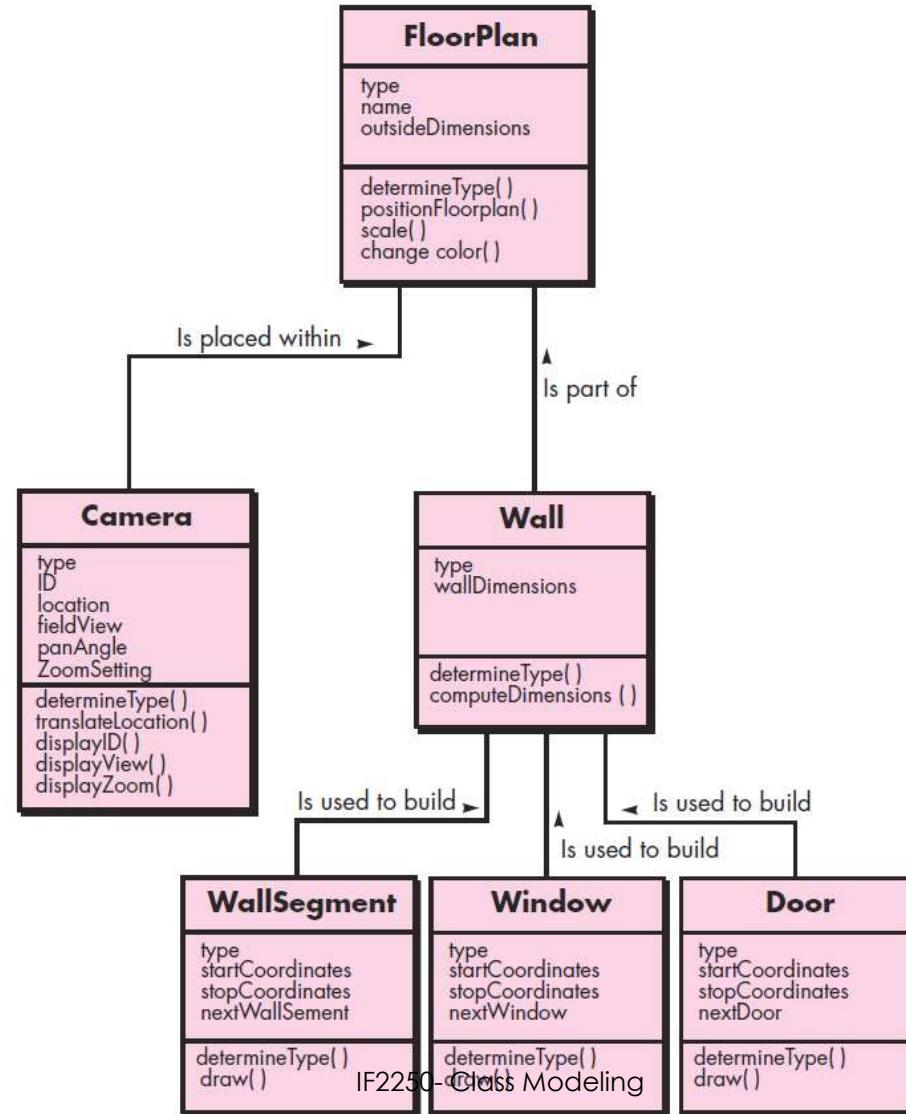
## Contoh Inheritance



# Contoh Inheritance



# Contoh Diagram Kelas



# *Latihan*

Akan dibangun sebuah perangkat lunak untuk mendukung proses pendaftaran ulang mahasiswa secara *online*. Melalui aplikasi tersebut, mahasiswa dapat mengajukan usulan pengambilan matakuliah.

Selanjutnya, dosen wali dapat melihat usulan pengambilan matakuliah untuk disetujui/ditolak. Usulan yang ditolak dapat direvisi kembali oleh mahasiswa.

Usulan yang telah disetujui wali dapat langsung diproses oleh Petugas Administrasi untuk pencetakan KSM. KSM hanya bisa dicetak apabila status pembayaran SPP mahasiswa sudah beres. Informasi status pembayaran SPP diperoleh dari perangkat lunak lain yaitu SISKEU (Sistem Informasi Keuangan). Perangkat lunak ini juga akan berhubungan dengan perangkat lunak SIKAD (Sistem Informasi Akademik) untuk mendapatkan informasi tentang matakuliah yang ditawarkan pada semester tersebut, serta informasi transkrip nilai mahasiswa, agar dosen wali mendapatkan referensi untuk menyetujui/menolak usulan pengambilan matakuliah.



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling

# *Kandidat Kelas ?*

Akan dibangun sebuah perangkat lunak untuk mendukung proses pendaftaran ulang mahasiswa secara *online*. Melalui aplikasi tersebut, mahasiswa dapat mengajukan usulan pengambilan matakuliah.

Selanjutnya, dosen wali dapat melihat usulan pengambilan matakuliah untuk disetujui/ditolak. Usulan yang ditolak dapat direvisi kembali oleh mahasiswa.

Usulan yang telah disetujui wali dapat langsung diproses oleh Petugas Administrasi untuk pencetakan KSM. KSM hanya bisa dicetak apabila status pembayaran SPP mahasiswa sudah beres. Informasi status pembayaran SPP diperoleh dari perangkat lunak lain yaitu SISKEU (Sistem Informasi Keuangan). Perangkat lunak ini juga akan berhubungan dengan perangkat lunak SIKAD (Sistem Informasi Akademik) untuk mendapatkan informasi tentang matakuliah yang ditawarkan pada semester tersebut, serta informasi transkrip nilai mahasiswa, agar dosen wali mendapatkan referensi untuk menyetujui/menolak usulan pengambilan matakuliah.

# *Kandidat Kelas*

Akan dibangun sebuah perangkat lunak untuk mendukung proses **pendaftaran ulang mahasiswa** secara *online*. Melalui aplikasi tersebut, mahasiswa dapat mengajukan **usulan pengambilan matakuliah**.

Selanjutnya, dosen **wali** dapat melihat usulan pengambilan matakuliah untuk disetujui/ditolak. Usulan yang ditolak dapat direvisi kembali oleh mahasiswa.

Usulan yang telah disetujui wali dapat langsung diproses oleh **Petugas Administrasi** untuk **pencetakan KSM**. KSM hanya bisa dicetak apabila **status pembayaran SPP** mahasiswa sudah beres. Informasi status pembayaran SPP diperoleh dari perangkat lunak lain yaitu **SISKEU** (Sistem Informasi Keuangan). Perangkat lunak ini juga akan berhubungan dengan perangkat lunak **SIKAD** (Sistem Informasi Akademik) untuk mendapatkan informasi tentang matakuliah yang ditawarkan pada semester tersebut, serta informasi **transkrip** nilai mahasiswa, agar dosen wali mendapatkan **referensi** untuk menyetujui/menolak usulan pengambilan matakuliah.

# *Kandidat Kelas*

- Pendaftaran → bisa jadi bagian dari Mahasiswa
- Mahasiswa
- Usulan pengambilan → FRS
- Matakuliah
- Wali
- Petugas Adm → di luar sistem
- Pencetakan
- KSM → sama dg FRS tetapi statusnya beda
- Status Pembayaran → jadi atribut
- Kelas dibuka
- SIKAD → di luar sistem
- SISKEU → di luar sistem
- Transkrip → Kumpulan Nilai



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling

# *Kandidat Kelas*

65

- **Entity** Classes:

- Mahasiswa
- Wali
- Matakuliah
- Kelas\_dibuka
- Usulan → FRS
- KSM

- **Controller** Classes:

- PendaftaranController
- TranskripManager
- ..

- **Boundary** Classes:

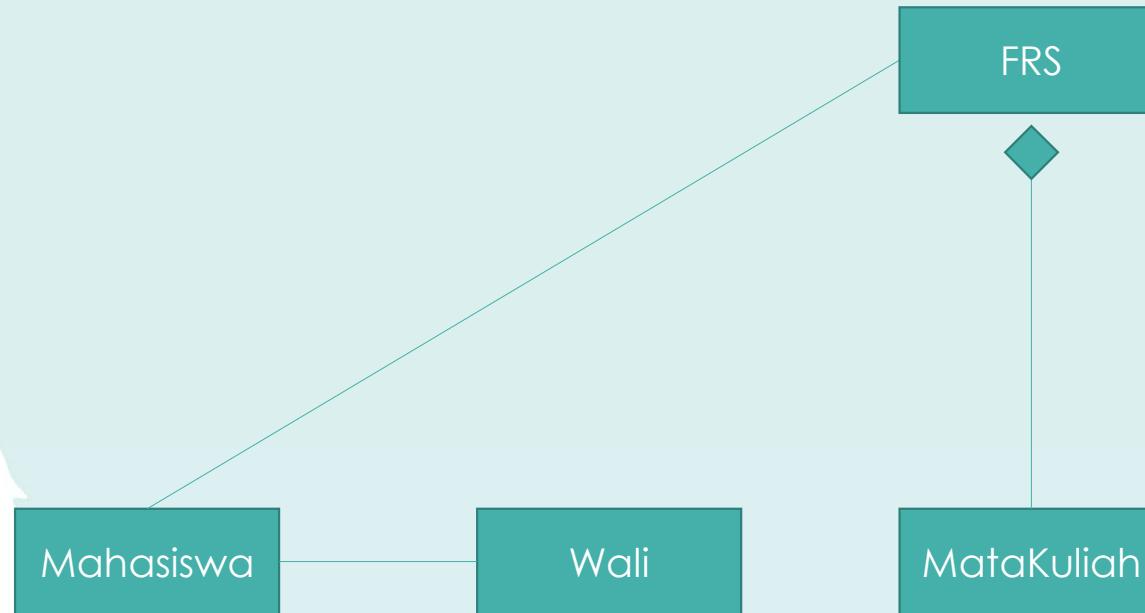
- FormEntriFRS
- TranskripDisplay
- ....



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling

## ***Diagram Kelas (Tahap Analisis – Iterasi awal)***



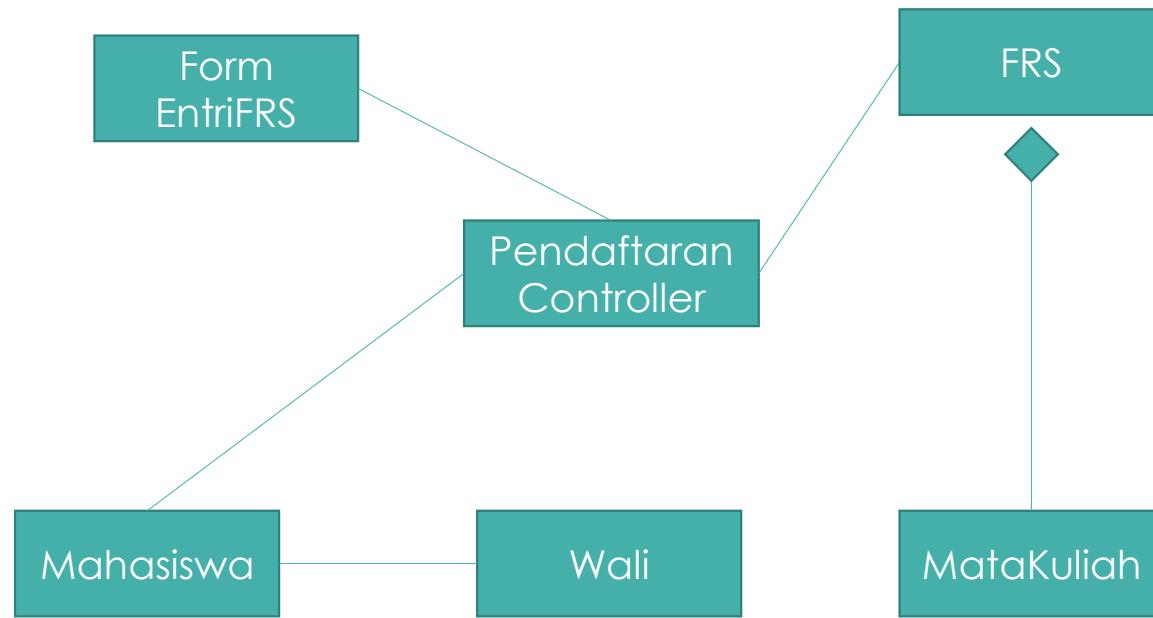
### ***Catatan:***

Pada tahapan implementasi (koding), jumlah kelas mungkin berubah lagi



# *Diagram Kelas (Tahap Desain atau Iterasi ke-n)*

67



KNOWLEDGE & SOFTWARE ENGINEERING

IF2250- Class Modeling

# Atribut

- Mahasiswa:
  - NIM
  - Nama
  - Alamat
  - Tgl\_Lahir
  - Status\_Daftar
- FRS
  - Sem
  - Tahun
  - Kd\_kul1
  - Kelas1
  - Kd\_kul2
  - Kelas2
  - ...



KNOWLEDGE & SOFTWARE ENGINEERING

# Operasi

- Mahasiswa
  - DaftarUlang()
  - Cuti()
  - GantiAlamat()
- FRS
  - TambahUsulan()
  - HapusUsulan()
  - UbahUsulan()
  - MintaPersetujuan()

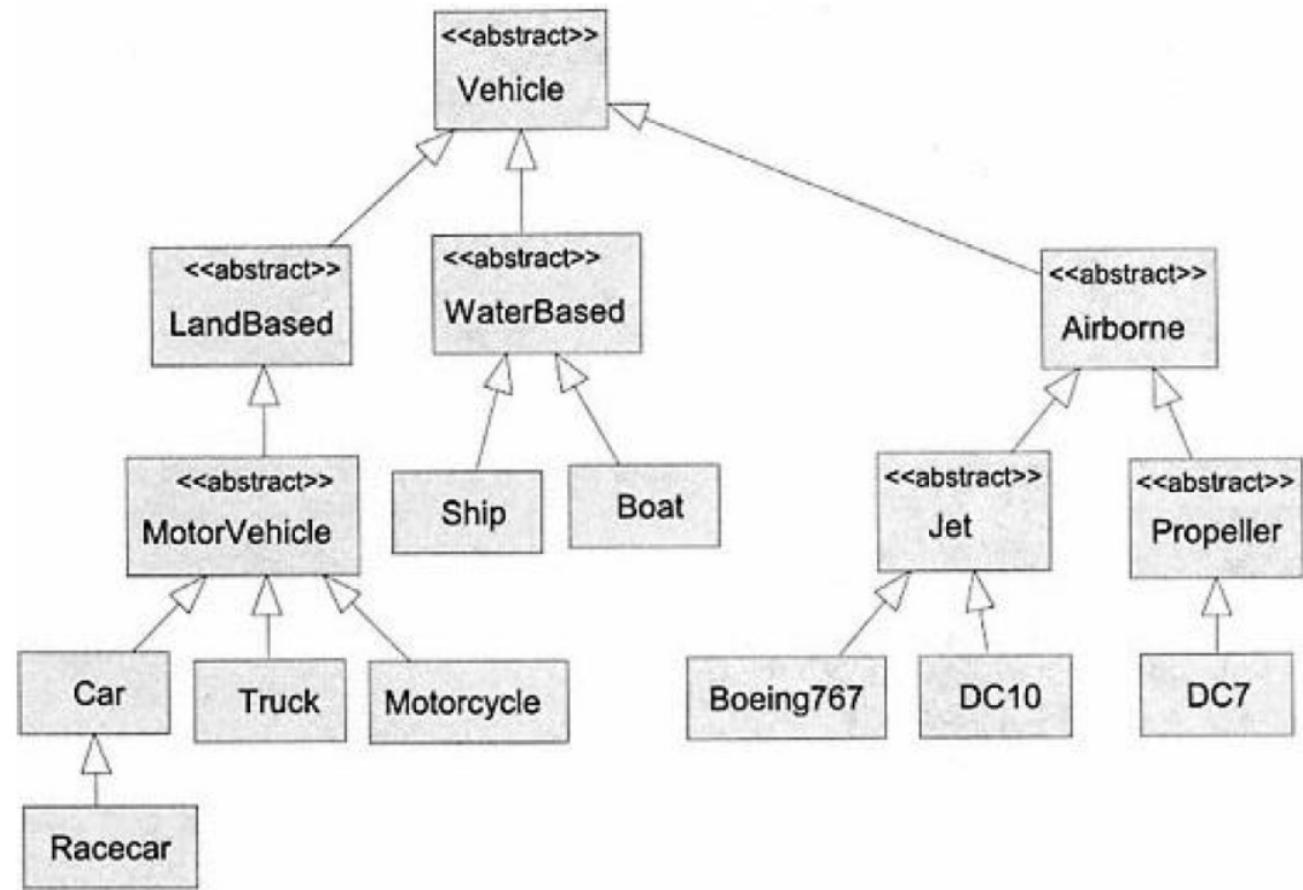


KNOWLEDGE & SOFTWARE ENGINEERING

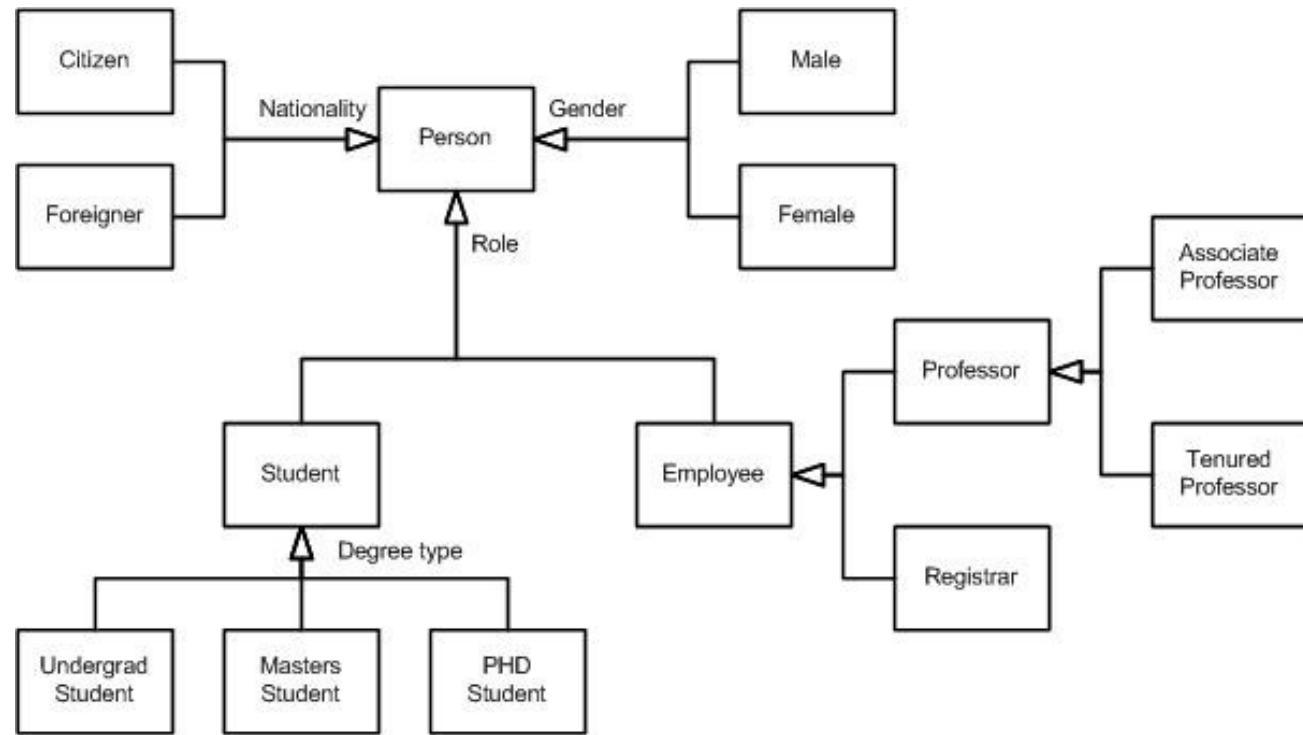
# *Contoh Kelas Diagram*



# Vehicle Class

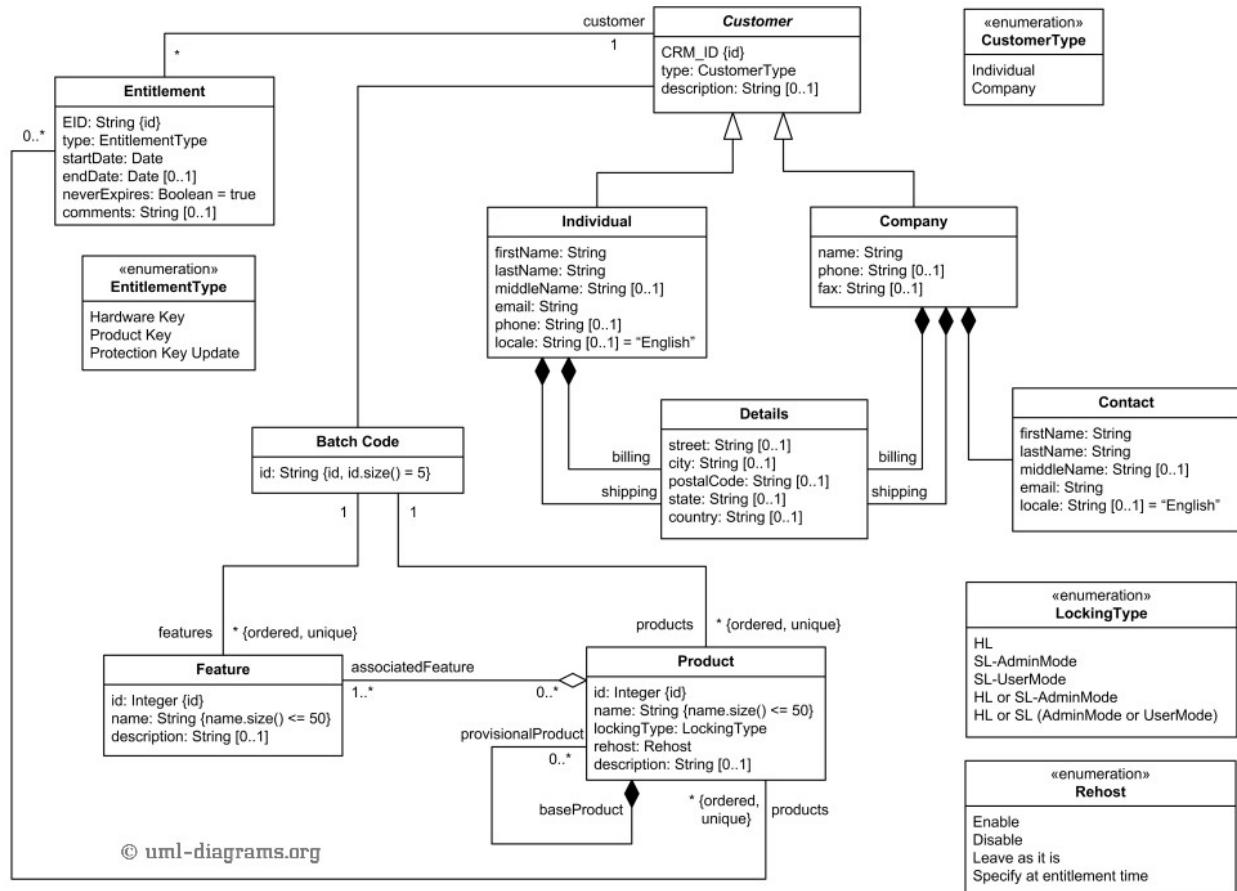


# Kelas Diagram untuk Sistem Warga Negara

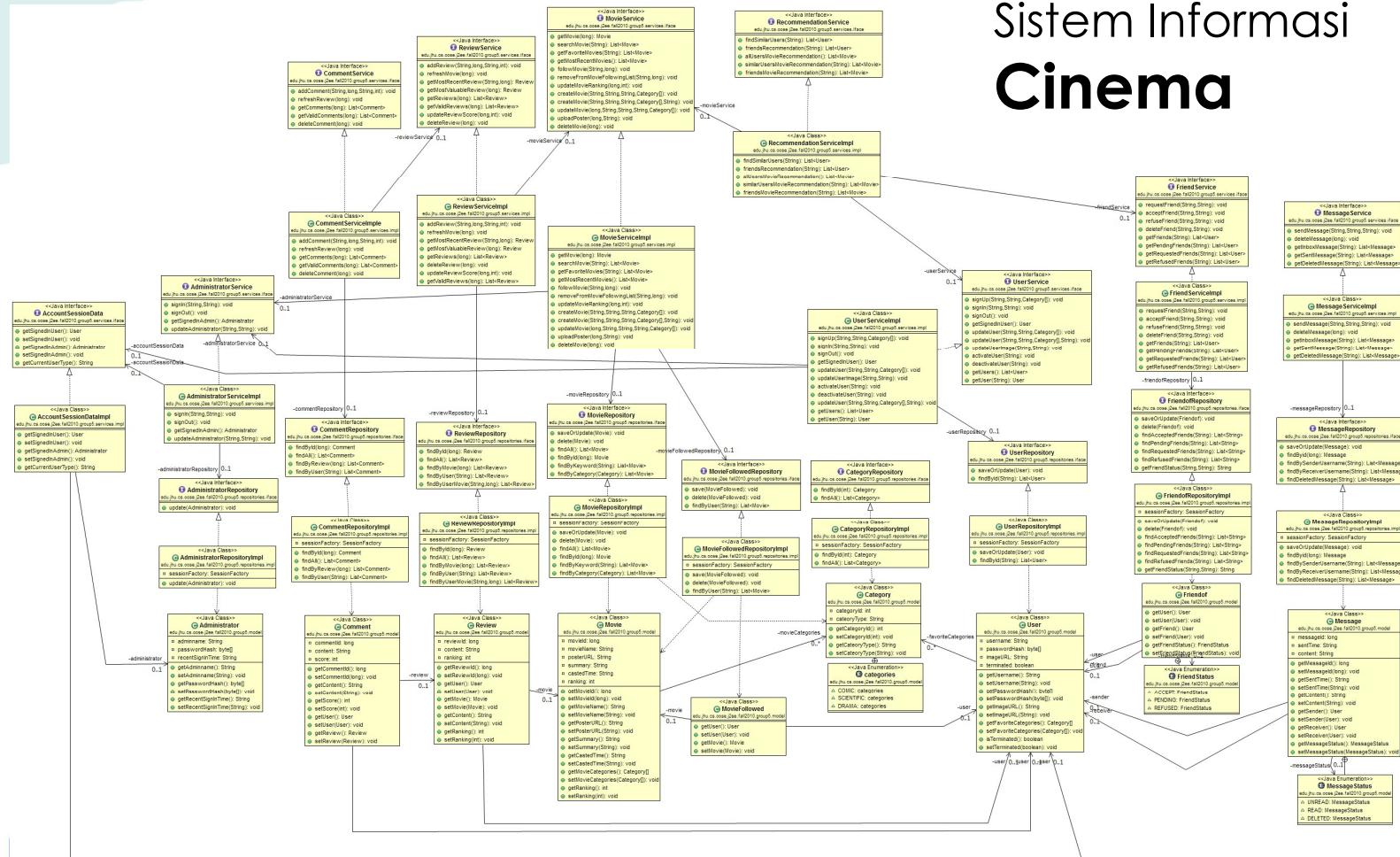


# *Customer Record*

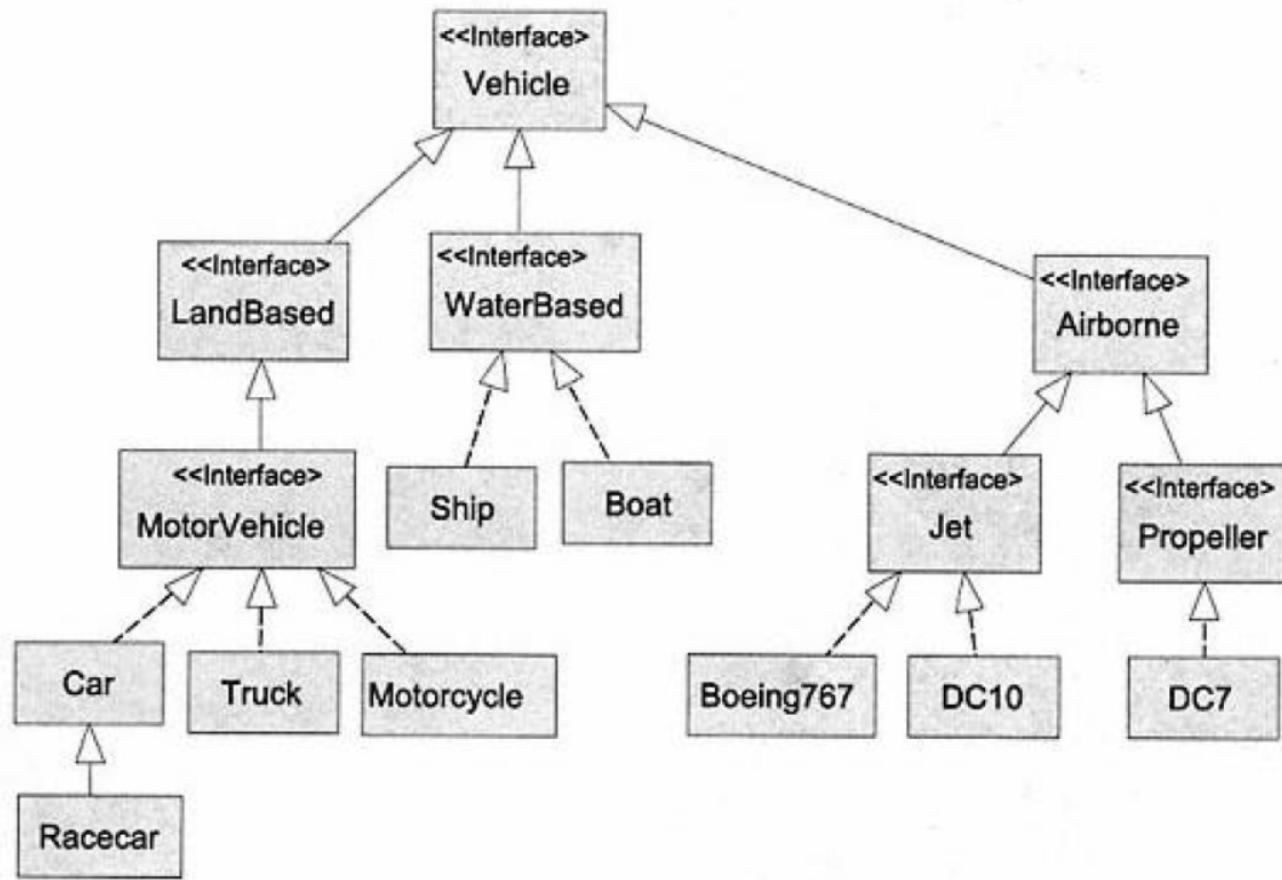
73



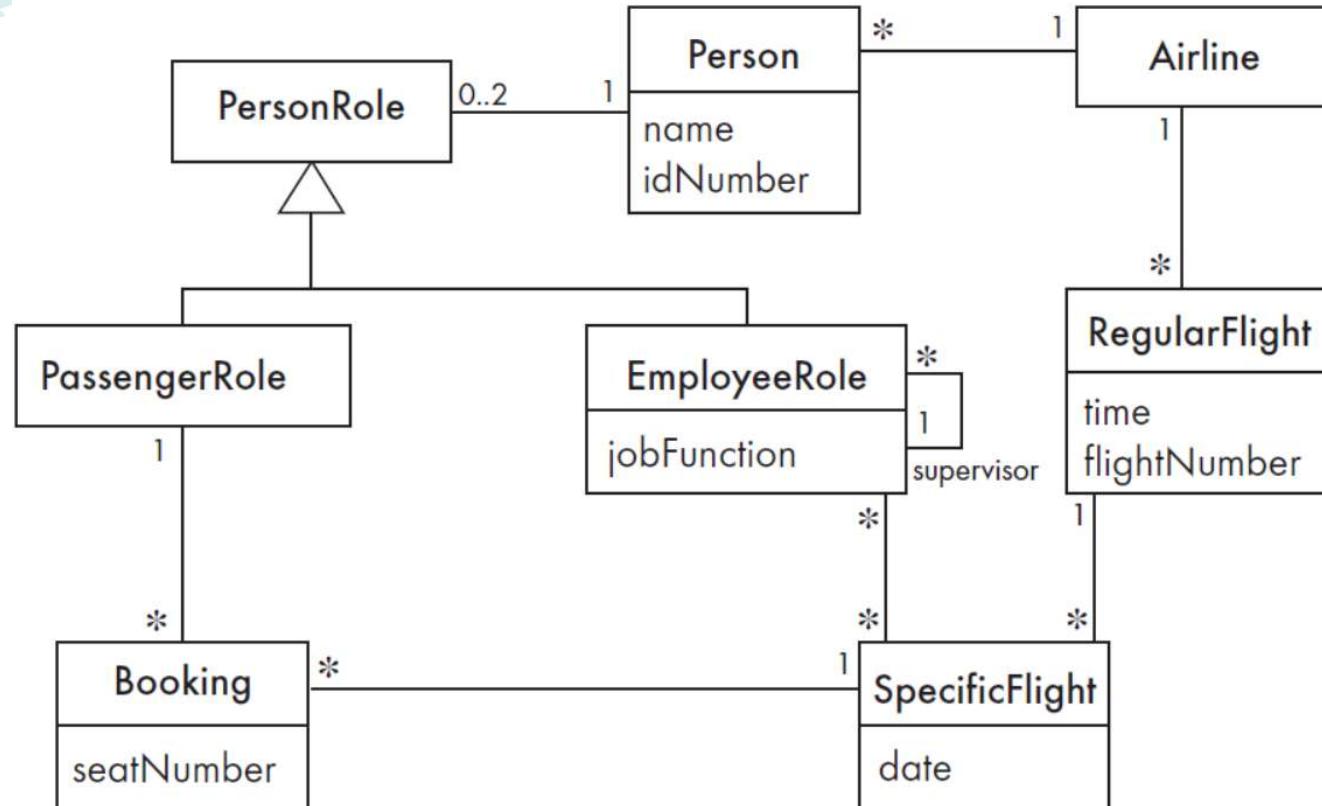
# Sistem Informasi Cinema



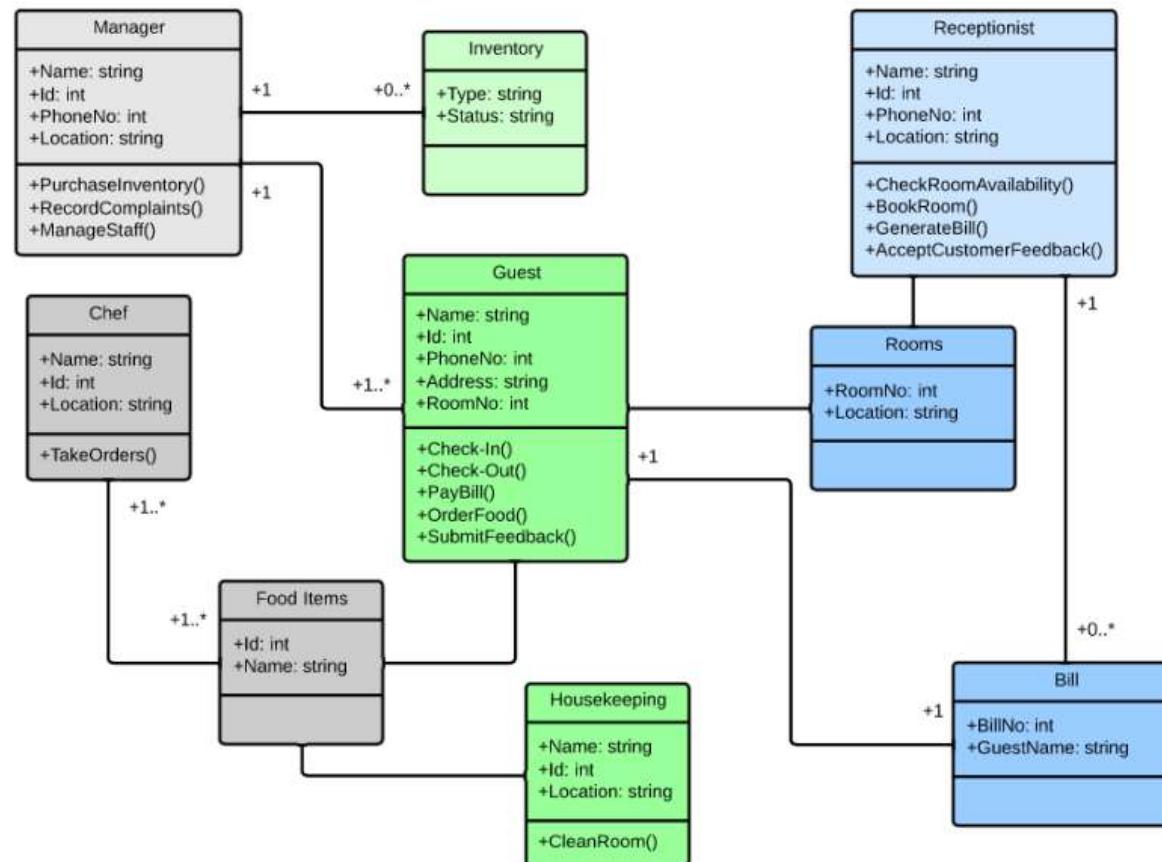
# *Class Vehicle dengan Abstract Class*



# System Airlines

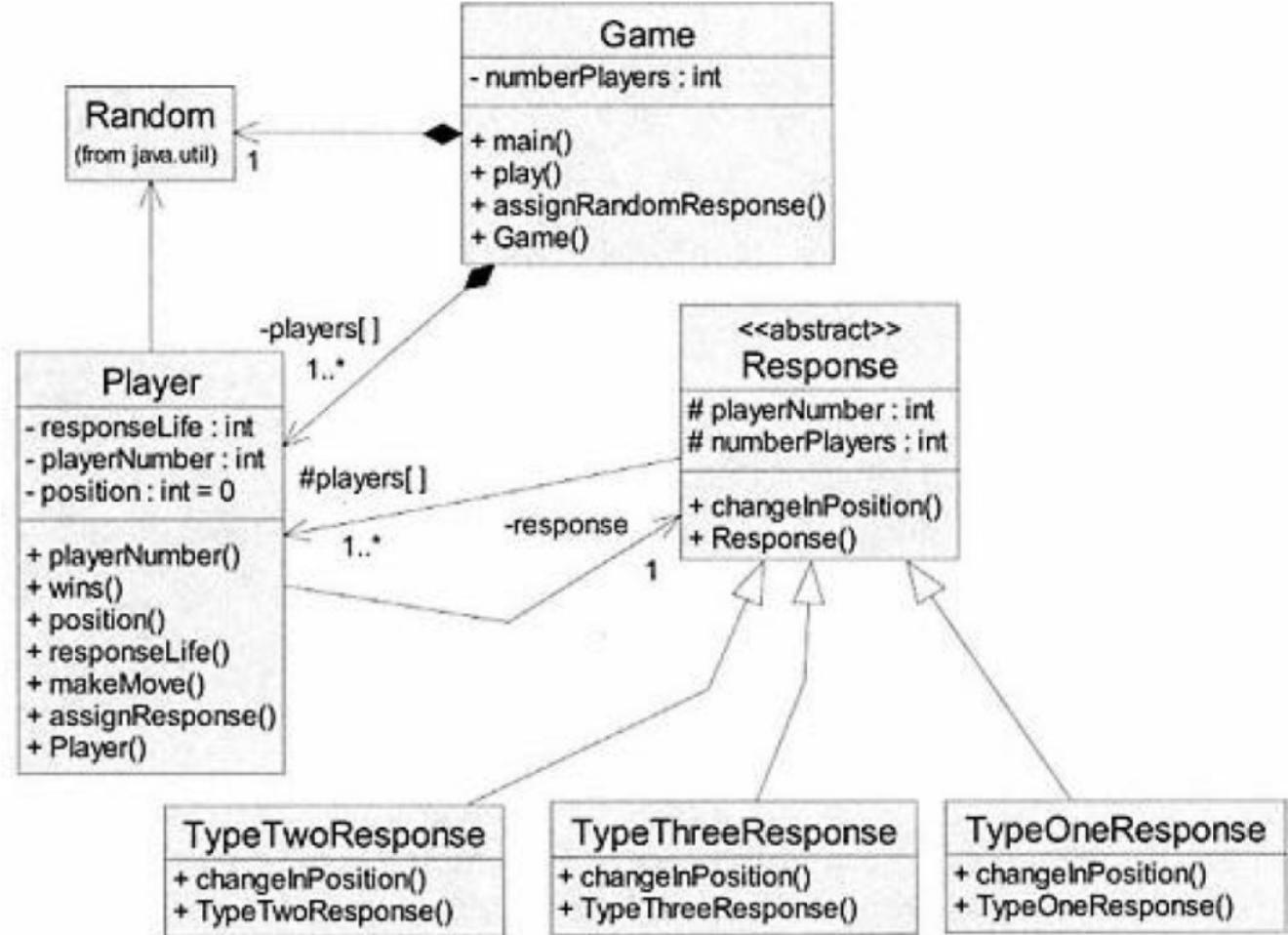


# Kelas Diagram untuk Sistem Manajemen Hotel



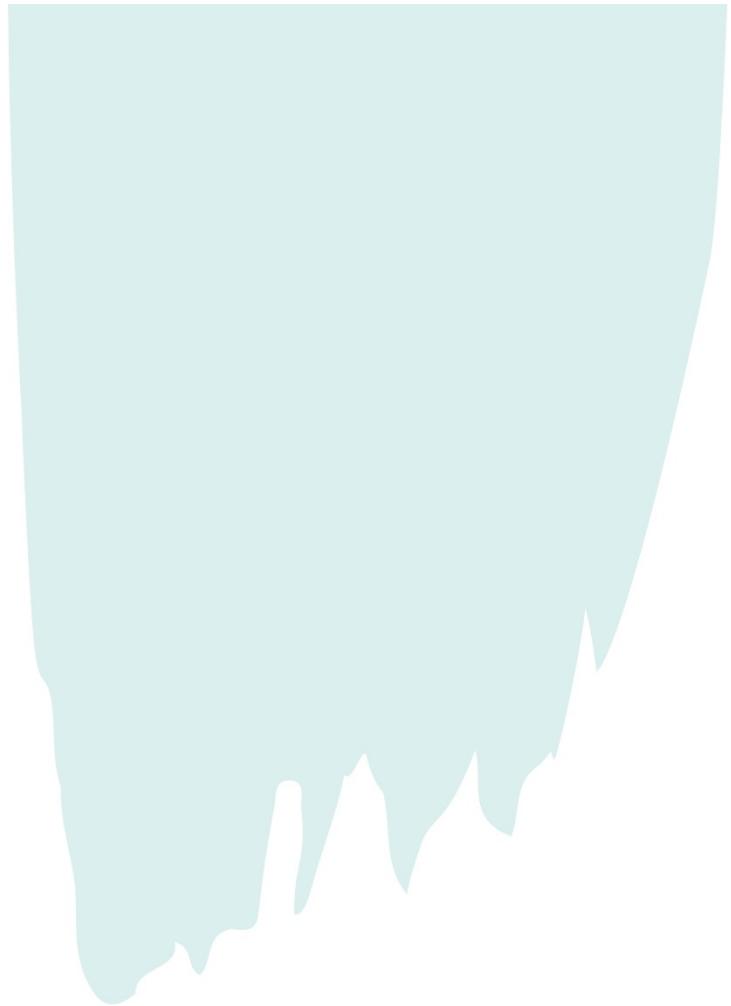
# Game Class Diagram

78



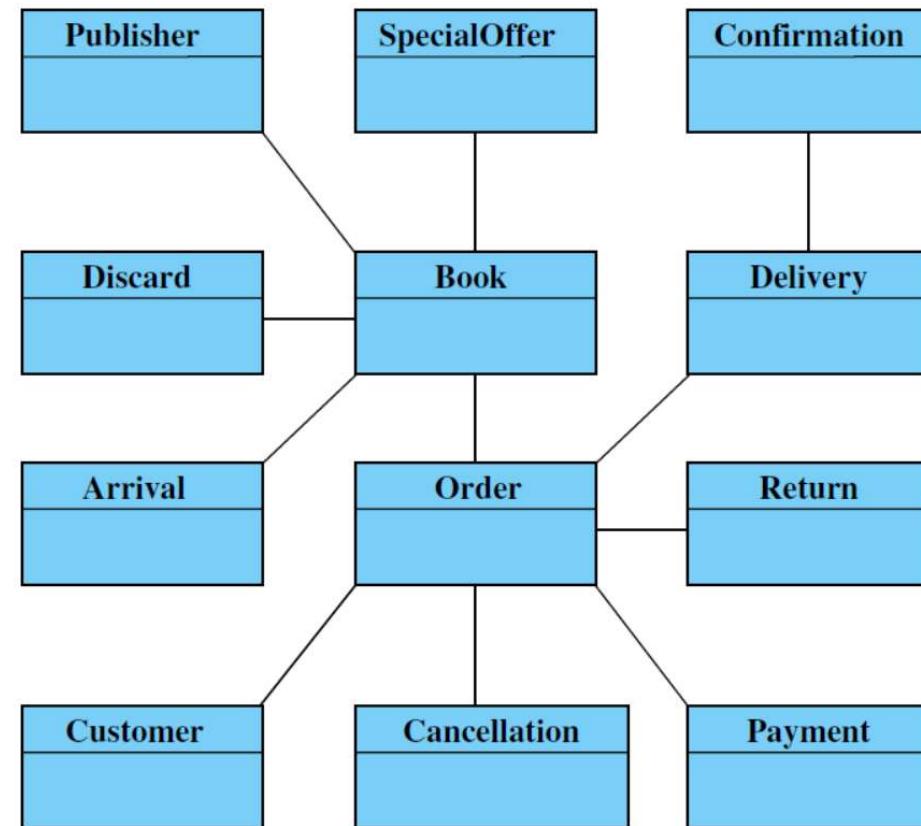
***Dari tahap analisis,  
sering terjadi diagram  
kelas berevolusi menjadi  
lebih rinci***

ILUSTRASI BERIKUT  
MENGGAMBARKAN APA YANG  
TERJADI PADA SETIAP ITERASI



IF00250 Class Modeling

# *First Iteration of Conceptual Model for Book Ordering*

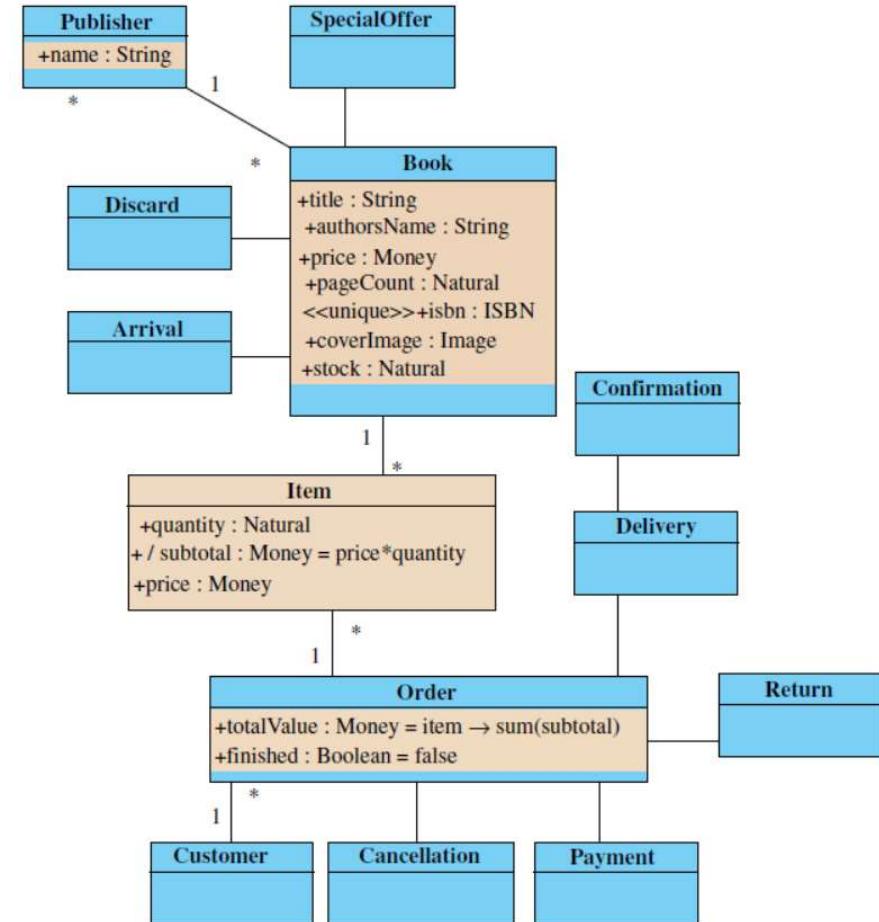


KNOWLEDGE & SOFTWARE ENGINEERING

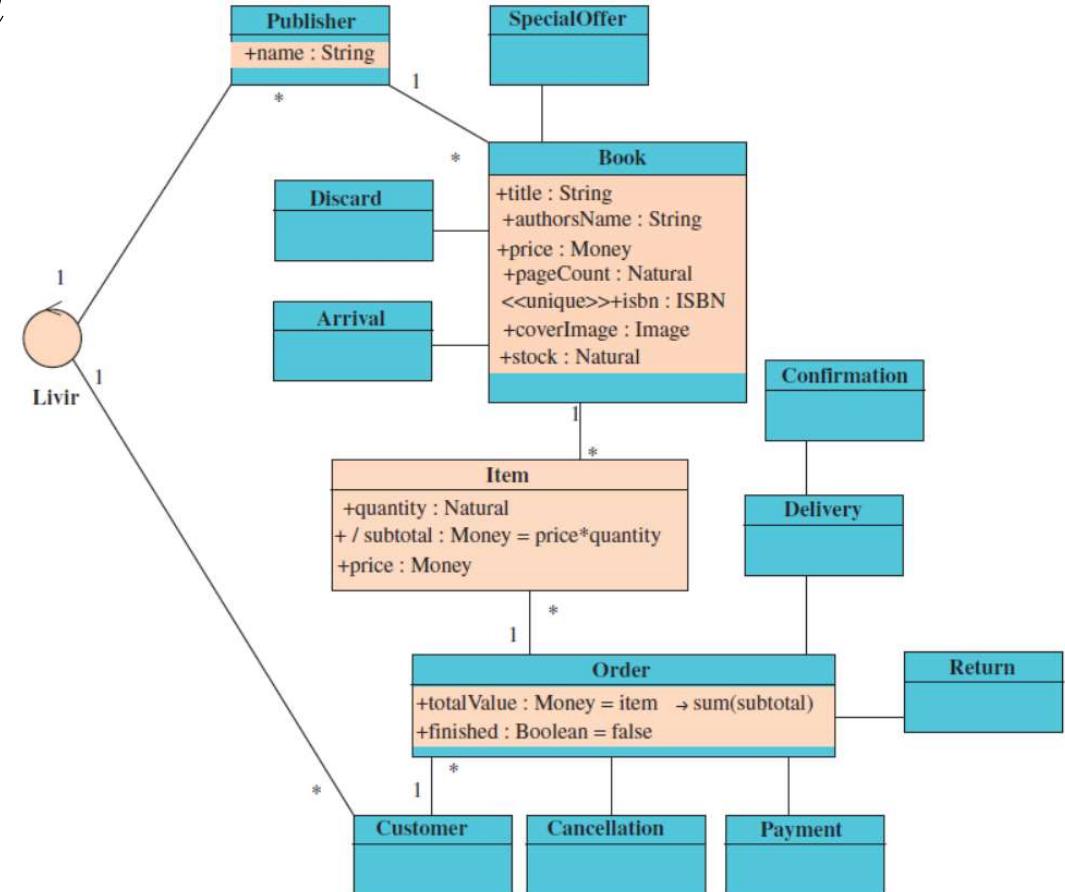
IF2250- Class Modeling

# *Second iteration of Book Ordering*

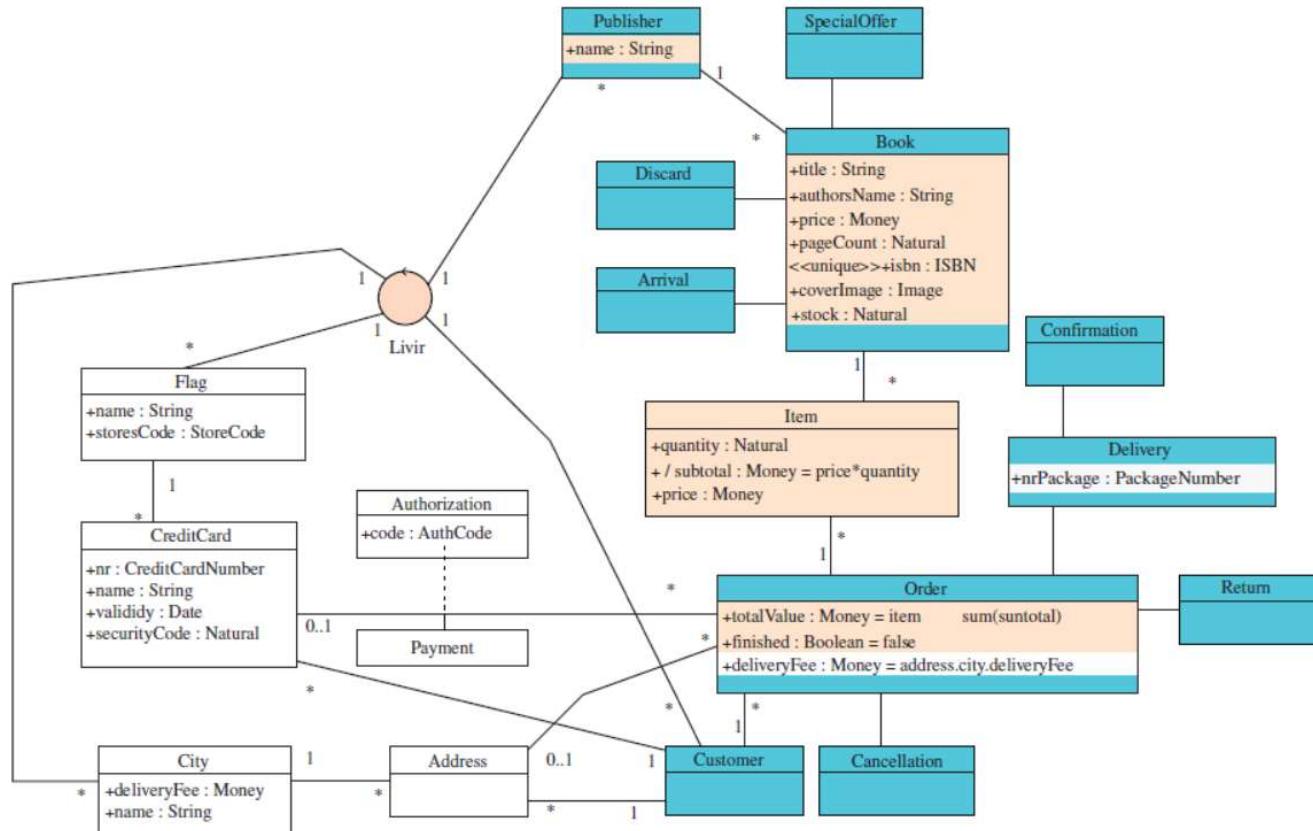
81



# Third Iteration



# Fourth Iteration



# References

- Lethbridge, Timothy Christian, and Robert Laganiere. *Object-oriented software engineering*. New York: McGraw-Hill, 2005.
- Wazlawick, Raul Sidnei. *Object-oriented analysis and design for information systems: modeling with UML, OCL, and IFML*. Elsevier, 2014.
- Booch, Grady. *Object oriented analysis & design with application 3<sup>rd</sup> edition*. Pearson Education India, 2006.