



Rosa A. S
M. Shalahuddin

REKAYASA PERANGKAT LUNAK

TERSTRUKTUR dan BERORIENTASI OBJEK

Bahasan meliputi:

- Analisis dan Desain Sistem ■
- SDLC ■
- Basis Data ■
- Pemrograman Terstruktur ■
- Pemrograman Berorientasi Objek ■
- Analisis dan Desain Berorientasi Objek ■
- Pemodelan dan UML ■
- Studi Kasus UML ■
- Perancangan Pola Berorientasi Objek ■
- Manajemen Proyek Perangkat Lunak ■

Penerbit INFORMATIKA



Nama : Apriliyana
NIM : A1316012

Rekayasa Perangkat Lunak

Terstruktur dan Berorientasi Objek

Rosa A. S.

M. Shalahuddin



INFORMATIKA
Bandung

Rekayasa Perangkat Lunak
(Terstruktur dan Berorientasi Objek)

Penulis : Rosa A.S.
M. Sholahuddin

Penerbit : Informatika Bandung

Pemasaran : BI-Obses
Pasar Buku Palasari No. 82
Bandung 40264
Telp. (022)7317812
Fax. (022)7317896

Cetakan Keempat : September 2016

ISBN : 978-602-1514-05-4

Copyright © 2016 pada Penerbit INFORMATIKA Bandung

Kata Pengantar

Segala puji bagi Allah SWT yang telah melimpahkan nikmat-Nya sehingga buku **Rekayasa Perangkat Lunak** ini bisa diselesaikan. Buku ini diharapkan dapat menunjang mahasiswa dalam memahami mata kuliah Rekayasa Perangkat Lunak dan juga bagi para pengembang perangkat lunak agar memahami korelasi rekayasa perangkat lunak dan pembangunan/pengembangan perangkat lunak. Buku ini dibuat berdasarkan sumber-sumber yang banyak digunakan dan pengalaman penulis selama sekitar satu dekade menjadi *software engineer* profesional.

Pada buku ini akan membahas tentang analisis dan desain sistem dengan disertai suatu studi kasus untuk memudahkan dalam pemahaman. Analisis dan desain sistem itu dimulai dengan analisis dan desain basis data, analisis dan desain sistem untuk pemrograman terstruktur dengan menggunakan DFD, dan analisis dan desain sistem untuk pemrograman berorientasi objek dengan menggunakan UML.

Setelah membaca studi kasus tersebut, pembaca diharapkan dapat memahami bagaimana melakukan analisis dan desain sistem untuk pemrograman terstruktur maupun pemrograman berorientasi objek. Untuk menghasilkan analisis dan desain perangkat lunak yang baik, seorang analis seharusnya memahami konsep pemrograman.

Tanpa pemahaman konsep pemrograman yang baik, analis tidak mungkin menghasilkan perancangan perangkat lunak yang baik.

Pada buku ini juga dijelaskan mengenai tahapan-tahapan yang harus dilalui dalam rakayasa perangkat lunak serta penjelasan secara umum tentang manajemen proyek perangkat lunak.

Pada kesempatan ini, penulis menyampaikan terima kasih yang tulus kepada berbagai pihak atas segala bantuan dan dukungannya sehingga penulis dapat menyelesaikan penulisan buku ini. Akhirnya, penulis mohon maaf jika dalam tulisan ini masih banyak kekurangan. Sumbangan ide, saran, dan kritik yang membangun untuk perbaikan buku ini sangat penulis harapkan.

Bandung, April 2013

Penulis

Daftar Isi

Kata Pengantar	iii
Daftar Isi.....	v
1 Pendahuluan	1
1.1 Perangkat Lunak.....	2
1.2 Rekayasa Perangkat Lunak.....	4
1.3 Proses Rekayasa Perangkat Lunak.....	8
1.4 Teknologi Informasi Sosial.....	11
2 Analisis dan Desain Sistem.....	17
2.1 Definisi Analisis Sistem.....	18
2.2 Teknik Pengumpulan Data.....	19
2.2.1 Teknik Wawancara.....	19
2.2.2 Teknik Observasi.....	20
2.2.3 Teknik Kuisioner.....	21
2.3 Jenis Kebutuhan	21
2.4 Definisi Desain Sistem.....	23
3 SDLC.....	25
3.1 Pengertian SDLC	25
3.2 Model SDLC.....	28
3.2.1 Model Waterfall.....	28
3.2.2 Model Prototipe	31
3.2.3 Model <i>Rapid Application Development (RAD)</i>	34
3.2.4 Model Iteratif.....	38
3.2.5 Model Spiral.....	39

8	Pemodelan dan UML.....	133
8.1	Kompleksitas Pengembangan Perangkat Lunak.....	134
8.2	Pemodelan	135
8.3	Pengenalan UML	137
8.4	Sejarah UML.....	138
8.5	Diagram UML.....	140
8.6	<i>Class Diagram</i>	141
8.7	<i>Object Diagram</i>	147
8.8	<i>Component Diagram</i>	148
8.9	<i>Composite Structure Diagram</i>	150
8.10	<i>Package Diagram</i>	153
8.11	<i>Deployment Diagram</i>	154
8.12	<i>Use Case Diagram</i>	155
8.13	<i>Activity Diagram</i>	161
8.14	<i>State Machine Diagram</i>	163
8.15	<i>Sequence Diagram</i>	165
8.16	<i>Communication Diagram</i>	168
8.17	<i>Timing Diagram</i>	169
8.18	<i>Interaction Overview Diagram</i>	171
9	Studi Kasus UML.....	175
9.1	<i>Use Case</i>	176
9.2	Diagram Kelas.....	205
9.3	Diagram Objek.....	207
9.4	Diagram Sekuen.....	208
9.5	Diagram Komunikasi.....	222
9.6	Diagram Kolaborasi	226
9.7	Diagram Status	229
9.8	Diagram Aktivitas	233
9.9	Diagram Komponen	235
9.10	Diagram <i>Deployment</i>	236
10	Perancangan Pola Berorientasi Objek	239
10.1	Design Pattern	240
10.2	Anti Pattern	256

11	Manajemen Proyek Perangkat Lunak.....	263
11.1	Pengertian Manajemen Proyek Perangkat Lunak...	264
11.2	Perencanaan Proyek	269
11.3	Pengujian Perangkat Lunak.....	271
11.3.1	Pengujian Unit.....	277
11.3.2	Pengujian Integrasi.....	278
	Glosarium.....	287
	Indeks.....	293
	Daftar Pustaka	295

1

Pendahuluan



Overview

Bab ini merupakan pendahuluan sebelum menjelaskan inti materi buku ini terkait dengan Rekayasa Perangkat Lunak (RPL) atau *software engineering*. Bab ini menjelaskan tentang hal-hal sebagai berikut:

1. Pengertian perangkat lunak atau *software*
2. Pengertian rekayasa perangkat lunak atau *software engineering*
3. Pengenalan proses rekayasa perangkat lunak
4. Pengenalan tentang faktor sosial yang berkaitan dengan teknologi informasi

Setelah membaca bab ini, pembaca diharapkan memahami pengertian bidang rekayasa perangkat lunak dan istilah-istilah yang berkaitan dengan hal tersebut.

1.1 Perangkat Lunak

Perangkat lunak (*software*) adalah program komputer yang terasosiasi dengan dokumentasi perangkat lunak seperti dokumentasi kebutuhan, model desain, dan cara penggunaan (*user manual*). Sebuah program komputer tanpa terasosiasi dengan dokumentasinya maka belum dapat disebut perangkat lunak (*software*). Sebuah perangkat lunak juga sering disebut dengan sistem perangkat lunak. Sistem berarti kumpulan komponen yang saling terkait dan mempunyai satu tujuan yang ingin dicapai.

Sistem perangkat lunak berarti sebuah sistem yang memiliki komponen berupa perangkat lunak yang memiliki hubungan satu sama lain untuk memenuhi kebutuhan pelanggan (*customer*). Pelanggan (*customer*) adalah orang atau organisasi yang memesan atau membeli perangkat lunak (*software*) dari pengembang perangkat lunak atau bisa dianggap bahwa pelanggan (*customer*) adalah orang atau organisasi yang dengan sukarela mengeluarkan uang untuk memesan atau membeli perangkat lunak. *User* atau pemakai perangkat lunak adalah orang yang memiliki kepentingan untuk memakai atau menggunakan perangkat lunak untuk memudahkan pekerjaannya.

Karakter perangkat lunak adalah sebagai berikut:

- Perangkat lunak dibangun dengan rekayasa (*software engineering*) bukan diproduksi secara manufaktur atau pabrikan.
- Perangkat lunak tidak pernah usang ("wear out") karena kecacatan dalam perangkat lunak dapat diperbaiki.
- Barang produksi pabrikan biasanya komponen barunya akan terus diproduksi, sedangkan perangkat lunak biasanya terus diperbaiki seiring bertambahnya kebutuhan.

Aplikasi dari perangkat lunak adalah sebagai berikut:

- Perangkat lunak sistem (*system software*) adalah kumpulan program dalam hal ini program yang satu ditulis untuk memenuhi kebutuhan program lainnya.

- Perangkat lunak waktu nyata (*real-time software*) merupakan perangkat lunak yang memonitor, menganalisis, mengontrol sesuatu secara waktu nyata (*real-time*). Reaksi yang dibutuhkan pada perangkat lunak harus langsung menghasilkan respon yang diinginkan.
- Perangkat lunak bisnis (*business software*) merupakan perangkat lunak pengelola informasi bisnis (seperti akuntansi, penjualan, pembayaran, penyimpanan (*inventory*)).
- Perangkat lunak untuk keperluan rekayasa dan keilmuan (*engineering and scientific software*) merupakan perangkat lunak yang mengimplementasikan algoritma yang terkait dengan keilmuan ataupun perangkat lunak yang membantu keilmuan, misalkan perangkat lunak di bidang astronomi, di bidang matematika dan lain sebagainya.
- Perangkat lunak tambahan untuk membantu mengerjakan suatu fungsi dari perangkat lunak yang lainnya (*embedded software*) misalnya perangkat lunak untuk mencetak dokumen ditambahkan agar perangkat lunak yang memerlukan dapat mencetak laporan, maka perangkat lunak untuk mencetak dokumen ini disebut *embedded software*.
- Perangkat lunak komputer personal (*personal computer software*) merupakan perangkat lunak untuk PC misalnya perangkat lunak pemroses teks, pemroses grafik dan lain sebagainya.
- Perangkat lunak berbasis web (*web based software*) merupakan perangkat lunak yang dapat diakses dengan menggunakan *browser*.

- Perangkat lunak berinteligensi buatan (*artificial intelligence software*) merupakan perangkat lunak yang menggunakan algoritma tertentu untuk mengelola data sehingga seakan-akan memiliki inteligensi seiring bertambahnya data yang diproses.

Produk perangkat lunak yang dibuat oleh pengembang (*developer*) perangkat lunak terdiri dari dua jenis:

- Produk generik
produk perangkat lunak yang dibuat oleh pengembang perangkat lunak untuk dijual atau dipopulerkan (*open source*) tanpa ada yang memesan terlebih dahulu, perangkat lunak yang termasuk dalam produk generik misalnya perangkat lunak sistem operasi, perangkat lunak pendukung perkantoran untuk membuat dokumen, *slide presentation*, atau perhitungan dalam bentuk *papersheet* dan lain sebagainya.
- Produk pemesanan
produk perangkat lunak yang dibuat karena ada pelanggan yang melakukan pemesanan, misalnya sebuah instansi memerlukan perangkat lunak untuk memenuhi proses bisnis yang terjadi di instansinya, maka instansi itu akan bekerja sama dengan pengembang untuk membuat perangkat lunak yang diinginkan.

1.2 Rekayasa Perangkat Lunak

Rekayasa perangkat lunak (*software engineering*) merupakan pembangunan dengan menggunakan prinsip atau konsep rekayasa dengan tujuan menghasilkan perangkat lunak yang bernilai ekonomi yang dipercaya dan bekerja secara efisien menggunakan mesin. Perangkat lunak banyak dibuat dan pada akhirnya sering tidak digunakan karena tidak memenuhi kebutuhan pelanggan atau bahkan karena masalah non-teknis seperti keenggan pemakai perangkat lunak (*user*) untuk mengubah cara kerja dari manual ke otomatis, atau ketidakmampuan *user* menggunakan komputer. Oleh karena itu,

rekayasa perangkat lunak dibutuhkan agar perangkat lunak yang dibuat tidak hanya menjadi perangkat lunak yang tidak terpakai.

Rekayasa perangkat lunak lebih fokus pada praktik pengembangan perangkat lunak dan mengirimkan perangkat lunak yang bermanfaat kepada pelanggan (*customer*). Adapun ilmu komputer lebih fokus pada teori dan konsep dasar perangkat komputer. Rekayasa perangkat lunak lebih fokus pada bagaimana membuat perangkat lunak yang memenuhi kriteria berikut:

- dapat terus dipelihara setelah perangkat lunak selesai dibuat seiring berkembangnya teknologi dan lingkungan (*Maintainability*)
- dapat diandalkan dengan proses bisnis yang dijalankan dan perubahan yang terjadi (*Dependability* dan *robust*)
- efisien dari segi sumber daya dan penggunaan
- kemampuan untuk dipakai sesuai dengan kebutuhan (*usability*)

Dari kriteria di atas maka perangkat lunak yang baik adalah perangkat lunak yang dapat memenuhi kebutuhan pelanggan (*customer*) atau *user* (pemakai perangkat lunak) atau berorientasi pada pelanggan atau pemakai perangkat lunak, bukan berorientasi pada pembuat atau pengembang perangkat lunak.



Perangkat lunak yang baik adalah perangkat lunak yang fokus pada pengguna atau pelanggan

Pekerjaan yang terkait dengan rekayasa perangkat dapat dikategorikan menjadi tiga buah kategori umum tanpa melihat area dari aplikasi, ukuran proyek perangkat lunak, atau kompleksitas perangkat lunak yang akan dibuat. Setiap fase dialamatkan pada satu atau lebih pertanyaan yang diajukan sebelumnya.

Fase pendefinisian fokus pada "what" yang artinya harus mencari tahu atau mengidentifikasi informasi apa yang harus diproses, seperti

apa fungsi dan performansi yang diinginkan, seperti apa perilaku sistem yang diinginkan, apa kriteria validasi yang dibutuhkan untuk mendefinisikan sistem.

Fase pengembangan yang fokus dengan "how" yang artinya selama tahap pengembangan perangkat lunak seorang perekayasa perangkat lunak (*software engineer*) berusaha untuk mendefinisikan bagaimana data distrukturkan dan bagaimana fungsi-fungsi yang dibutuhkan diimplementasikan didalam arsitektur perangkat lunak, bagaimana detail prosedural diimplementasikan, bagaimana karakter antarmuka tampilan, bagaimana desai ditranslasikan ke bahasa pemrograman, dan bagaimana pengujian akan dijalankan.

Fase pendukung (*support phase*) fokus pada perubahan yang terasosiasi pada perbaikan kesalahan (*error*), adaptasi yang dibutuhkan pada lingkungan perangkat lunak yang terlibat, dan perbaikan yang terjadi akibat perubahan kebutuhan pelanggan (*customer*). Fase pendukung terdiri dari empat tipe perubahan antara lain:

- **Koreksi (*correction*)**
walaupun dengan jaminan kualitas yang terbaik, akan selalu ada kecacatan atau keinginan pelanggan (*customer*) yang tidak tertangani oleh perangkat lunak. Pemeliharaan dengan melakukan perbaikan terhadap kecacatan perangkat lunak.
- **Adaptasi (*adaptation*)**
pada saat tertentu lingkungan asli (seperti CPU, sistem operasi, aturan bisnis, karakteristik produk luar) dimana perangkat lunak dikembangkan akan mengalami perubahan. Pemeliharaan adaptasi merupakan tahap untuk memodifikasi perangkat lunak guna mengakomodasi perubahan lingkungan luar dimana perangkat lunak dijalankan.
- **Perbaikan (*enhancement*)**
sejalan dengan digunakannya perangkat lunak, maka pelanggan (*customer*) atau pemakainya (*user*) akan mengenali fungsi tambahan yang dapat mendatangkan manfaat.

Pemeliharaan perfektif atau penyempurnaan melakukan ekstensi atau penambahan pada kebutuhan fungsional sebelumnya.

- **Pencegahan (*prevention*)**

keadaan perangkat lunak komputer sangat dimungkinkan untuk perubahan. Oleh karena itu, pemeliharaan pencegahan (*preventive*) atau sering disebut juga dengan rekayasa ulang sistem (*software reengineering*) harus dikondisikan untuk mampu melayani kebutuhan pemakainya (*user*). Untuk menanggulangi hal ini maka perangkat lunak harus dirancang dan dikondisikan untuk mengakomodasi perubahan kebutuhan yang diinginkan oleh pemakainya (*user*). Di lain sisi biasanya setelah perangkat lunak dikirimkan ke *user* maka masih dibutuhkan asistensi dan *help desk* dari pengembang perangkat lunak.

Tantangan yang dihadapi dari proses rekayasa perangkat lunak adalah sebagai berikut:

- tantangan warisan dimana perangkat lunak dikembangkan selama bertahun-tahun oleh orang-orang yang berbeda, hal ini dapat menyebabkan ketidakpahaman atau perubahan tujuan pembuatan perangkat lunak;
- tantangan heterogenitas dimana perangkat lunak harus dapat beradaptasi dengan teknologi yang terus berkembang dengan semakin luasnya lingkungan distribusi perangkat lunak;
- tantangan pengiriman bahwa perangkat lunak dengan sekala besar dan kompleks sekalipun dapat sampai ke tangan pelanggan (*customer*) atau *user* dengan cepat dan kualitas tetap terjaga.

Ada suatu hal tambahan dalam rekayasa perangkat lunak yang berbasis *web* pada tahap desainnya yaitu bagaimana menentukan aliran halaman *web*. Beberapa aliran halaman *web* misalnya adalah sebagai berikut:

Sedangkan DBMS versi *open source* yang cukup berkembang dan paling banyak digunakan saat ini adalah sebagai berikut:

- MySQL
- PostgreSQL
- Firebird
- SQLite

Hampir semua DBMS mengadopsi SQL sebagai bahasa untuk mengelola data pada DBMS.

4.3 SQL

SQL (*Structured Query Language*) adalah bahasa yang digunakan untuk mengelola data pada RDBMS. SQL awalnya dikembangkan berdasarkan teori aljabar relasional dan kalkulus.

SQL mulai berkembang pada tahun 1970an. SQL mulai digunakan sebagai standar yang resmi pada tahun 1986 oleh ANSI (*American National Standards Institute*) dan pada tahun 1987 oleh ISO (*International Organization for Standardization*) dan disebut sebagai SQL-86. Pada perkembangannya, SQL beberapa kali dilakukan revisi. Berikut ini sejarah perkembangan SQL sampai saat ini:

No.	Tahun	Nama
1	1986	SQL-86
2	1989	SQL-89
3	1992	SQL-92
4	1999	SQL:1999
5	2003	SQL:2003
6	2006	SQL:2006
7	2008	SQL:2008
8	2011	SQL:2011

Meskipun SQL diadopsi dan diacu sebagai bahasa standar oleh hampir sebagian besar RDBMS yang beredar saat ini, tetapi tidak semua standar yang tercantum dalam SQL diimplementasikan oleh

seluruh DBMS tersebut. Sehingga kadang-kadang ada perbedaan perilaku (hasil yang ditampilkan) oleh DBMS yang berbeda padahal query yang dimasukkan sama.

Berikut ini adalah contoh pengaksesan data pada DBMS dengan SQL yang secara umum terdiri dari 4 hal sebagai berikut:

- Memasukkan data (*insert*)

```
INSERT INTO Tabel_mahasiswa  
(nim, nama, tanggal_lahir)  
VALUES  
(‘13501058’, ‘Rosa’, ‘1986-01-01’);
```

Query di atas digunakan untuk memasukkan data mahasiswa dengan NIM 13501058, nama Rosa, dan tanggal lahir 1 Januari 1986 ke tabel “Tabel_mahasiswa”.

- Mengubah data (*update*)

```
UPDATE Tabel_mahasiswa  
SET  
    tanggal_lahir = ‘1990-03-04’  
WHERE  
nim = ‘13501058’;
```

Query di atas digunakan untuk mengubah data tanggal lahir mahasiswa dengan NIM = 13501058 menjadi 4 Maret 1990 dalam tabel “Tabel_mahasiswa”.

- Menghapus data (*delete*)

```
DELETE FROM Tabel_mahasiswa  
WHERE  
nim = ‘13501058’;
```

Query di atas digunakan untuk menghapus data mahasiswa dengan NIM = 13501058 dari tabel “Tabel_mahasiswa”.

- Menampilkan data (*select*)

```
SELECT nim,  
nama  
FROM Tabel_mahasiswa  
WHERE  
nim = '13501058';
```

Query di atas digunakan untuk menampilkan data mahasiswa yang tersimpan dalam "Tabel_mahasiswa" dengan NIM = 13501058.



Hampir sebagian besar DBMS yang beredar saat ini memanfaatkan SQL sebagai bahasa untuk pengelolaan data

4.4 Alur Hidup Basis Data

Tidak hanya perangkat lunak yang memiliki alur hidup, dalam membuat perencanaan basis data juga memiliki alur hidup atau *Database Life Cycle* (DBLC). Alur hidup basis data dapat dilihat pada gambar berikut:



Gambar 14 Alur Hidup Basis Data

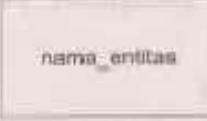
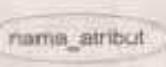
Fase-fase DBLC antara lain:

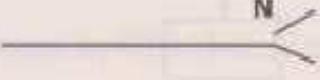
- Analisis kebutuhan / *requirement analysis*
Hal-hal yang harus dilakukan pada tahap ini adalah:
 - Didefinisikan dengan mewawancara produsen dan pernakai data, data apa sajakah yang butuh untuk disimpan dan terkait dengan aplikasi komputer yang akan dikembangkan
 - Membuat kontrak spesifikasi basis data
 - *Entity Relationship Diagram* (ERD) sebagai bagian dari desain konseptual
- Desain logik basis data / *logical database design*
Pada tahap ini harus dibuat rancangan logik basis data. Biasanya pada tahap ini dibuat *Conceptual Data Model* (CDM).
- Desain fisik basis data / *physical database design*
Pada tahap ini harus dibuat rancangan fisik basis data. Biasanya pada tahap ini dibuat *Physical Data Model* (PDM).

- Implementasi
 - Membuat Query SQL
 - Aplikasi ke DBMS atau file

4.5 ERD

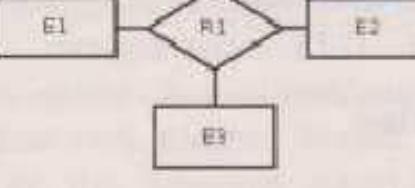
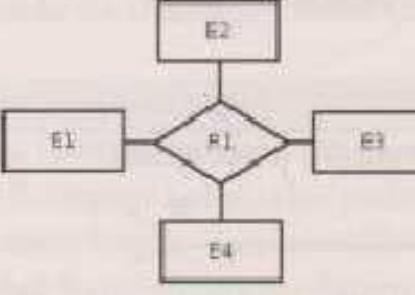
Pemodelan awal basis data yang paling banyak digunakan adalah menggunakan *Entity Relationship Diagram* (ERD). ERD dikembangkan berdasarkan teori himpunan dalam bidang matematika. ERD digunakan untuk pemodelan basis data relasional. Sehingga jika penyimpanan basis data menggunakan OODBMS maka perancangan basis data tidak perlu menggunakan ERD. ERD memiliki beberapa aliran notasi seperti notasi Chen (dikembangkan oleh Peter Chen), Barker (dikembangkan oleh Richard Barker, Ian Palmer, Harry Ellis), notasi Crow's Foot, dan beberapa notasi lain. Namun yang banyak digunakan adalah notasi dari Chen. Berikut adalah simbol-simbol yang digunakan pada ERD dengan notasi Chen:

Simbol	Deskripsi
Entitas / <i>entity</i> 	Entitas merupakan data inti yang akan disimpan; bakal tabel pada basis data; benda yang memiliki data dan harus disimpan datanya agar dapat diakses oleh aplikasi komputer; penamaan entitas biasanya lebih ke kata benda dan belum merupakan nama tabel
Atribut 	<i>Field</i> atau kolom data yang butuh disimpan dalam suatu entitas
Atribut kunci primer 	<i>Field</i> atau kolom data yang butuh disimpan dalam suatu entitas dan digunakan sebagai kunci akses <i>record</i> yang diinginkan; biasanya

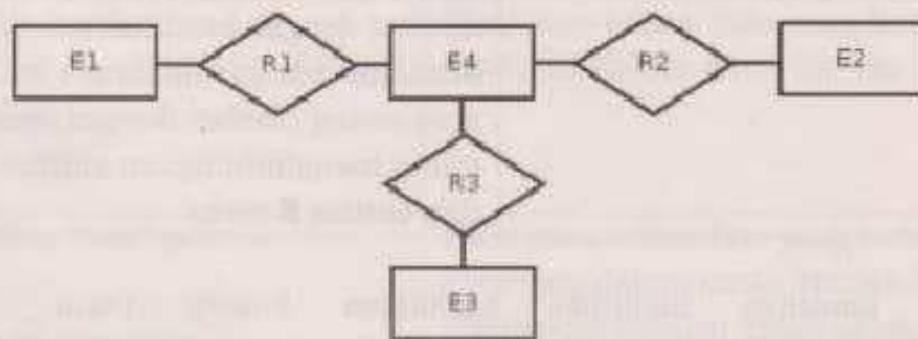
Simbol	Deskripsi
	berupa id; kunci primer dapat lebih dari satu kolom, asalkan kombinasi dari beberapa kolom tersebut dapat bersifat unik (berbeda tanpa ada yang sama)
Atribut multinilai / <i>multivalue</i> 	<i>Field</i> atau kolom data yang butuh disimpan dalam suatu entitas yang dapat memiliki nilai lebih dari satu
Relasi 	Relasi yang menghubungkan antar entitas; biasanya diawali dengan kata kerja
Asosiasi / <i>association</i> 	<p>Penghubung antara relasi dan entitas di mana di kedua ujungnya memiliki <i>multiplicity</i> kemungkinan jumlah pemakaian</p> <p>Kemungkinan jumlah maksimum keterhubungan antara entitas satu dengan entitas yang lain disebut dengan kardinalitas. Misalkan ada kardinalitas 1 ke N atau sering disebut dengan <i>one to many</i> menghubungkan entitas A dan entitas B maka</p>

ERD biasanya memiliki hubungan *binary* (satu relasi menghubungkan dua buah entitas). Beberapa metode perancangan ERD menoleransi hubungan relasi *ternary* (satu relasi

menghubungkan tiga buah relasi) atau *N-ary* (satu relasi menghubungkan banyak entitas), tapi banyak metode perancangan ERD yang tidak mengizinkan hubungan *ternary* atau *N-ary*. Berikut adalah contoh bentuk hubungan relasi dalam ERD:

Nama	Gambar
<i>binary</i>	
<i>ternary</i>	
<i>N-ary</i>	

Beberapa metode perencanaan ERD memberikan solusi untuk perencanaan ERD yang tidak *binary* diubah sebagai berikut:



Gambar 15 Cara Menghindari Relasi Ternary

Entitas E₄ berasal dari relasi R₁ yang dijadikan entitas, karena banyak metode perencanaan ERD yang menyatakan bahwa jika terjadi relasi *ternary*, maka sebenarnya relasinya lebih layak dijadikan entitas dibandingkan menjadi relasi.



ERD adalah bentuk paling awal dalam melakukan perancangan basis data relasional. Jika menggunakan OODBMS maka perancangan ERD tidak perlu dilakukan

4.6 Studi Kasus

Dalam buku ini, untuk mempermudah pemahaman tentang materi yang dijelaskan akan menggunakan sebuah studi kasus tentang Sistem Informasi Manajemen Perpustakaan. Studi kasus ini akan dijadikan studi kasus sampai di akhir buku. Diharapkan dengan adanya studi kasus yang menyatu ini pembaca dapat memahami proses rekayasa perangkat lunak dari awal dan akhir secara komprehensif / menyeluruh.

Nama aplikasi: Sistem Informasi Manajemen Perpustakaan

Deskripsi:

Sistem informasi manajemen perpustakaan merupakan sebuah sistem informasi untuk mengelola informasi yang diperlukan dalam suatu perpustakaan yang meliputi pendaftaran pustaka, anggota, dan proses peminjaman pustaka. Aturan perpustakaan yang harus diatasi pada sistem informasi manajemen perpustakaan yang akan dimodelkan adalah sebagai berikut:

1. Pustaka dapat memiliki lebih dari satu pengarang
2. Anggota dapat memiliki lebih dari satu nomor telepon
3. Seorang anggota dapat melakukan sebuah peminjaman dalam satu waktu dan boleh lebih dari satu pustaka

4. Seorang anggota dapat mengembalikan pustaka yang dipinjam tidak dalam waktu yang bersamaan walaupun pustaka-pustaka itu dipinjam pada waktu yang sama.
5. Pengunjung yang bukan anggota tidak diperbolehkan meminjam pustaka.
6. Proses pendaftaran pustaka, anggota, dan peminjaman dilakukan oleh petugas perpustakaan.
7. Anggota dan pengunjung dapat melakukan pencarian pustaka.
8. Satu pustaka akan disimpan sebagai satu data dengan id yang unik.

Manajemen perpustakaan meliputi fungsi-fungsi sebagai berikut:

1. Validasi Petugas
 - a. *Login*
 - b. *Logout*
2. Mengelola data pustaka, meliputi:
 - a. Memasukkan data pustaka
 - b. Mengubah data pustaka
 - c. Menghapus data pustaka
 - d. Mencari data pustaka
 - e. Melihat data pustaka
3. Mengelola data anggota, meliputi:
 - a. Memasukkan data anggota
 - b. Mengubah data anggota
 - c. Menghapus data anggota
 - d. Mencari data anggota
 - e. Melihat data anggota
4. Mengelola data peminjaman, meliputi:
 - a. Memasukkan data peminjaman
 - b. Mengubah data peminjaman (mekanisme pengembalian pustaka)
 - c. Mencari data peminjaman
 - d. Melihat data peminjaman
5. Mengelola data petugas untuk petugas yang memiliki hak akses untuk mengelola data petugas, meliputi:
 - a. Memasukkan data petugas baru
 - b. Mengubah data petugas

- c. Menghapus data petugas
- d. Mencari data petugas
- e. Melihat data petugas

4.7 Studi kasus ERD

Studi kasus untuk membuat ERD menggunakan sistem informasi manajemen perpustakaan dengan deskripsi seperti yang ada pada subbab di atas.

4.7.1 Definisi Entitas dan atribut

Berikut adalah definisi entitas dan atribut dari sistem informasi manajemen perpustakaan:

No	Entitas	Atribut
1.	Pustaka Entitas yang menyimpan data pustaka	id atribut yang menjadi identitas pustaka
		judul atribut judul pustaka
		jenis atribut jenis pustaka
		penerbit atribut penerbit pustaka
		tahun atribut tahun pustaka
		pengarang atribut pengarang pustaka (bisa lebih)

No	Entitas	Atribut
2.	Anggota Entitas yang menyimpan data anggota	dari satu) id atribut yang menjadi identitas anggota
		nama atribut nama anggota
		alamat atribut alamat rumah anggota
		email atribut alamat email anggota
		telepon atribut nomor telepon anggota (bisa lebih dari satu)
3.	Peminjaman Entitas yang menyimpan data anggota	tgl_peminjaman atribut tanggal peminjaman pustaka
		tgl_kembali atribut tanggal kembali pustaka
4.	Petugas Entitas yang menyimpan data petugas yang berhak login ke aplikasi untuk mengelola data	username atribut untuk melakukan proses login password atribut sebagai kata sandi untuk melakukan login nama atribut nama petugas no_petugas atribut untuk nomor petugas

No	Entitas	Atribut
		hak_akses atribut untuk mengetahui hak akses petugas yang berhak mengelola data petugas atau tidak (biasanya disebut sebagai admin/administrator/yang mengurus administrasi)

4.7.2 Definisi Relasi

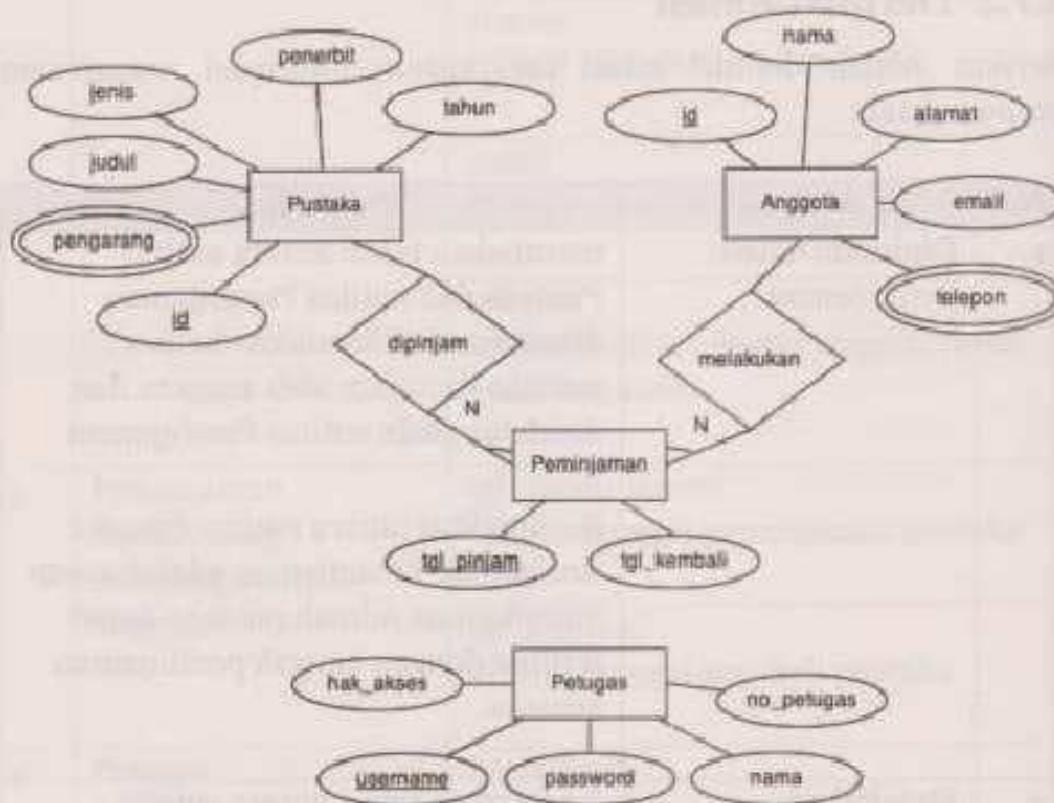
Berikut adalah definisi relasi dari sistem informasi manajemen perpustakaan:

No	Relasi	Deskripsi
1.	Dipinjam dalam peminjaman	merupakan relasi antara entitas Pustaka dan entitas Peminjaman dimana memiliki makna bahwa pustaka dipinjam oleh anggota dan disimpan pada entitas Peminjaman Kardinalitas antara entitas Pustaka dan entitas Peminjaman adalah <i>one to many</i> karena sebuah pustaka dapat terlibat dengan banyak peminjaman pustaka
2.	Melakukan peminjaman	merupakan relasi antara entitas Anggota dan entitas Peminjaman dimana memiliki makna bahwa anggota melakukan peminjaman pustaka dan disimpan pada entitas Peminjaman Kardinalitas antara entitas Anggota dan entitas Peminjaman adalah <i>one to many</i> karena seorang anggota dapat

No	Relasi	Deskripsi
		terlibat dengan banyak peminjaman pustaka

4.7.3 Diagram ER

Berikut ini gambar diagram ER untuk studi kasus Sistem Informasi Manajemen Perpustakaan di atas.



Gambar 16 Diagram ER Studi Kasus

Entitas Petugas dalam perancangan ERD di atas tidak memiliki relasi dengan tabel lain karena data petugas tidak dikaitkan atau tidak masuk dalam data yang ada di entitas-entitas lain. Entitas Petugas diperlukan menyimpan data untuk keperluan proses *login*.

4.8 CDM

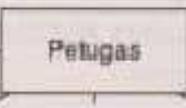
CDM (*Conceptual Data Model*) atau model konsep data merupakan konsep yang berkaitan dengan pandangan pemakai terhadap data yang disimpan dalam basis data. CDM dibuat sudah dalam bentuk tabel-tabel tanpa tipe data yang menggambarkan relasi antar tabel untuk keperluan implementasi ke basis data.

CDM merupakan hasil penjabaran lebih lanjut dari ERD. Ada aturan-aturan yang harus diikuti dalam melakukan konversi ERD menjadi CDM. Dalam buku ini aturan-aturan tersebut tidak dijabarkan karena diasumsikan hal tersebut bisa dipelajari pada buku-buku yang secara khusus membahas basis data.

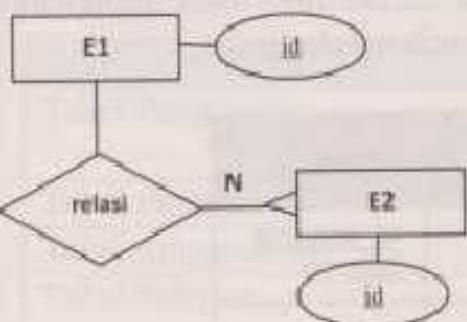
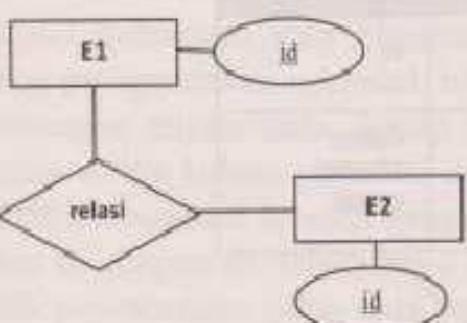
Berikut adalah simbol-simbol yang ada pada CDM:

Simbol	Deskripsi
Entitas / tabel 	entitas atau tabel yang menyimpan data dalam basis data
Relasi 	relasi antar tabel yang terdiri atas nama relasi dan <i>multiplicity</i>

Aturan untuk mengubah ERD menjadi CDM secara umum adalah sebagai berikut:

ERD	CDM
 entitas	

ERD	CDM
<p>Pelugas</p> <p>PK: username password nama no_pelugas hak_akses</p>	<p>menjadi sebuah tabel tersendiri</p>
<p>Pengarang</p> <p>PK: id_pustaka pengarang</p>	<p>menjadi sebuah tabel tersendiri dengan kunci primer (<i>primary key</i>) adalah kunci primer pada entitas dan memiliki atribut dengan nama seperti pada atribut entitas</p>
<p>Relasi</p> <p>PK: id_E1, id_E2 atribut_relası</p>	<p>menjadi sebuah tabel tersendiri dengan kunci primer adalah atribut yang menjadi kunci primer di kedua entitas yang direlasikannya</p>

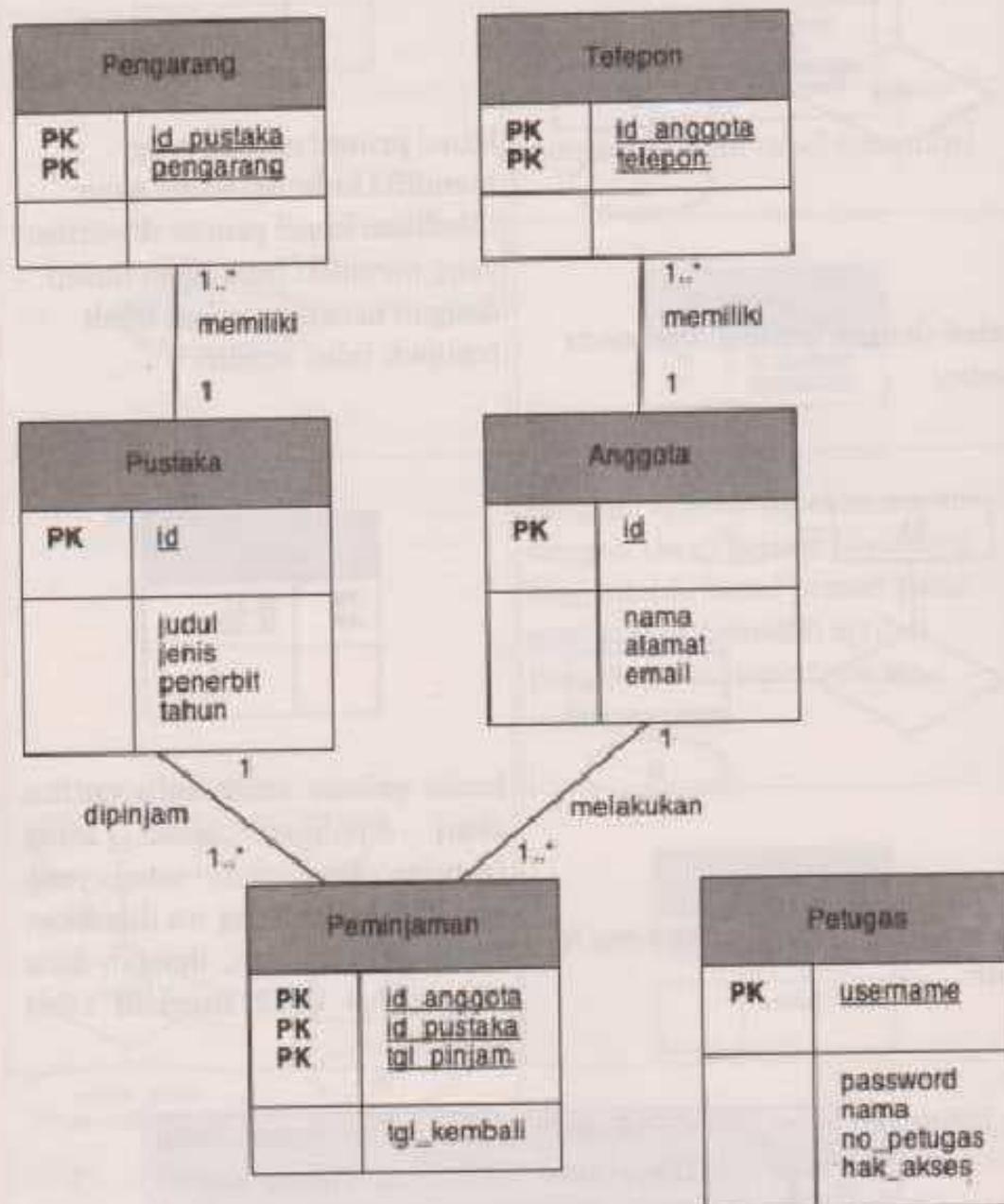
ERD	CDM								
 <p>relasi dengan kardinalitas <i>one to many</i></p>	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td colspan="2" style="text-align: center; background-color: #cccccc;">E2</td> </tr> <tr> <td style="text-align: center;">PK</td> <td style="text-align: center;">id_E1</td> </tr> <tr> <td style="text-align: center;">PK</td> <td style="text-align: center;">id_E2</td> </tr> <tr> <td> </td> <td> </td> </tr> </table> <p>kunci primer entitas yang memiliki hubungan <i>one</i> akan dijadikan kunci primer di entitas yang memiliki hubungan <i>many</i> dengan kata lain, relasi tidak menjadi tabel sendiri</p>	E2		PK	id_E1	PK	id_E2		
E2									
PK	id_E1								
PK	id_E2								
 <p>relasi dengan kardinalitas <i>one to one</i></p>	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td colspan="2" style="text-align: center; background-color: #cccccc;">E2</td> </tr> <tr> <td style="text-align: center;">PK</td> <td style="text-align: center;">id_E1</td> </tr> <tr> <td style="text-align: center;">PK</td> <td style="text-align: center;">id_E2</td> </tr> <tr> <td> </td> <td> </td> </tr> </table> <p>kunci primer salah satu entitas akan dijadikan kunci asing (<i>foreign key</i>) pada tabel yang lain dan kunci asing itu dijadikan kunci primer juga, dengan kata lain, relasi tidak menjadi tabel sendiri.</p>	E2		PK	id_E1	PK	id_E2		
E2									
PK	id_E1								
PK	id_E2								



CDM merupakan penjabaran lebih lanjut dari ERD. Ada aturan-aturan cara menjabarkan ERD menjadi CDM yang bisa dipelajari pada buku yang secara khusus membahas basis data.

4.9 Studi Kasus CDM

Berikut adalah CDM dari studi kasus sistem informasi manajemen perpustakaan seperti pada studi kasus bab sebelumnya:



Gambar 17 CDM Studi Kasus

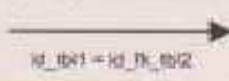
Tabel-tabel di atas merupakan hasil pertanggungjawaban dari diagram ER yang telah dibuat dengan rincian sebagai berikut:

CDM	ERD
Tabel Pengarang	atribut <i>multivalue</i> pengarang dari entitas Pustaka
Tabel Pustaka	entitas Pustaka
Tabel Anggota	entitas Anggota
Tabel Telepon	atribut <i>multivalue</i> telepon dari entitas Anggota
Tabel Peminjaman	entitas Peminjaman
Tabel Petugas	entitas Petugas

4.10 PDM

Model Relasional atau *Physical Data Model* (PDM) adalah model yang menggunakan sejumlah tabel untuk menggambarkan data serta hubungan antara data. Setiap tabel mempunyai sejumlah kolom di mana setiap kolom memiliki nama yang unik beserta tipe datanya. PDM merupakan konsep yang menerangkan detail dari bagaimana data disimpan di dalam basis data. PDM sudah merupakan bentuk fisik perancangan basis data yang sudah siap diimplementasikan ke dalam DBMS sehingga nama tabel juga sudah merupakan nama asli tabel yang akan diimplementasikan ke dalam DBMS.

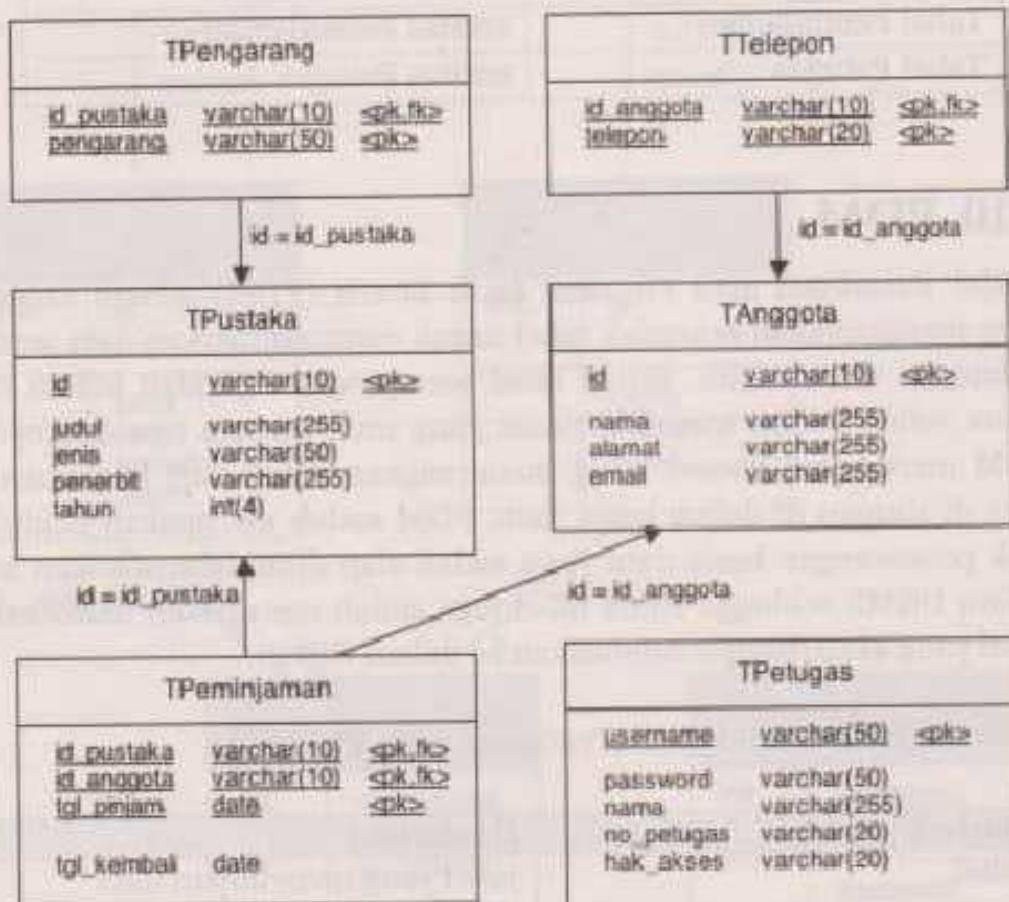
Berikut adalah simbol-simbol yang ada pada PDM:

Simbol	Deskripsi
Tabel 	tabel yang menyimpan data dalam basis data
Relasi 	relasi antar tabel yang terdiri dari persamaan antara <i>primary key</i> (kunci primer) tabel yang diajukan

Simbol	Deskripsi
	dengan kunci yang menjadi referensi acuan di tabel lain.

4.11 Studi Kasus PDM

Berikut adalah PDM dari studi kasus sistem informasi manajemen perpustakaan seperti pada studi kasus bab sebelumnya:



Gambar 18 PDM Studi Kasus



1. Apakah yang dimaksud dengan basis data?
2. Apakah fungsi basis data pada suatu sistem informasi?
3. Apakah yang dimaksud dengan DBMS dan apa fungsinya?
4. Sebutkan syarat-syarat suatu sistem bisa disebut sebagai DBMS!
5. Sebutkan kelebihan dan kekurangan masing-masing DBMS sebagai berikut!
 - a. Oracle
 - b. Microsoft SQL server
 - c. Microsoft Access
 - d. IBM DB2
 - e. MySQL
 - f. PostgreSQL
 - g. Firebird
 - h. SQLite
6. Jelaskan tentang organisasi ANSI dan ISO! Apa peran yang mereka lakukan?
7. Apa pengertian SQL dan apa kegunaannya?
8. Apa pengertian *query*?
9. Mengapa implementasi/adopsi standar SQL pada masing-masing DBMS bisa berbeda-beda?
10. Apa yang dimaksud dengan entitas, relasi, dan atribut pada ERD?
11. Bagaimana cara/aturan untuk melakukan pemetaan dari ERD ke CDM?
12. Apakah perbedaan antara CDM dan PDM?
13. Sebutkan dan jelaskan beberapa basis data non-relasional! Apa kelebihan dan kekurangan basis data tersebut jika dibandingkan dengan basis data relasional?
14. Buatlah desain basis data untuk Sistem Informasi Apotek!

Hal terpenting adalah kolom telepon tetap dapat diakses dari perancangan kelas. Kelas data biasanya adalah kelas yang terkait dengan pengaksesan tabel pada basis data sesuai dengan nama kelasnya, misalnya kelas Anggota maka akan digunakan untuk mengakses tabel yang menyimpan data anggota.

Jenis-jenis kelas di atas juga dapat digabungkan satu sama lain sesuai dengan pertimbangan yang dianggap baik asalkan fungsi-fungsi yang sebaiknya ada pada struktur kelas tetap ada. Susunan kelas juga dapat ditambahkan kelas utilitas (*utility class*) seperti Koneksi ke basis data, membaca *file* teks, dan lain sebagainya sesuai kebutuhan.

Dalam mendefinisikan metode yang ada di dalam kelas perlu memperhatikan apa yang disebut dengan *cohesion* dan *coupling*. *Cohesion* adalah ukuran seberapa dekat keterkaitan instruksi di dalam sebuah metode terkait satu sama lain sedangkan *coupling* adalah ukuran seberapa dekat keterkaitan instruksi antara metode yang satu dengan metode yang lain dalam sebuah kelas. Sebagai aturan secara umum maka sebuah metode yang dibuat harus memiliki kadar *cohesion* yang kuat dan kadar *coupling* yang lemah.

Perhatikan juga jika sebuah kelas hanya terdiri dari atribut saja, atau sebuah atribut saja, atau hanya sebuah metode. Berikut pertimbangan dalam membuat kelas:

Pertimbangan	Keterangan
sebuah kelas yang hanya berisi sebuah atribut	Kelas seperti ini kurang efisien sehingga perlu dipikirkan kembali perancangan yang dilakukan, karena secara fisik kelas dengan satu atribut adalah sebagai berikut: <pre>class Pustaka { judul : string }</pre> sebuah berkas atau <i>file</i> hanya berisi seperti di atas tentu saja tidak efisien.

Pertimbangan	Keterangan
	<p>Cara memperbaiki rancangan adalah dengan menggabungkan satu atribut tersebut ke kelas lain yang memiliki keterkaitan lebih dekat.</p>
sebuah kelas yang hanya berisi atribut	<p>Kelas seperti ini kurang efisien sehingga perlu dipikirkan kembali perancangan yang dilakukan, karena secara fisik kelas dengan hanya berisi atribut saja adalah sebagai berikut:</p> <pre>class Pustaka{ id : string judul : string penerbit : string jumlah : int tahun : int }</pre> <p>sebuah berkas atau <i>file</i> hanya berisi seperti di atas tentu saja tidak efisien.</p> <p>Cara memperbaiki rancangan adalah dengan menggabungkan atribut tersebut ke kelas lain yang memiliki keterkaitan lebih dekat.</p>
sebuah kelas yang hanya berisi sebuah metode	<p>Kelas seperti ini kurang efisien sehingga perlu dipikirkan kembali perancangan yang dilakukan, karena secara fisik kelas dengan hanya berisi sebuah metode saja adalah sebagai berikut:</p> <pre>class Pustaka{ id : string judul : string penerbit : string jumlah : int tahun : int }</pre>

Pertimbangan	Keterangan
	<pre> procedure queryMelihatPustaka() query = "SELECT" execute(query) </pre> <p>sebuah berkas atau <i>file</i> hanya berisi seperti di atas tentu saja tidak efisien.</p> <p>Cara memperbaiki rancangan adalah dengan menggabungkan atribut dan metode tersebut ke kelas lain yang memiliki keterkaitan lebih dekat.</p>

Berikut adalah simbol-simbol yang ada pada diagram kelas:

Simbol	Deskripsi
kelas	kelas pada struktur sistem
antarmuka / <i>interface</i> 	sama dengan konsep <i>interface</i> dalam pemrograman berorientasi objek
asosiasi / <i>association</i> 	relasi antarkelas dengan makna umum, asosiasi biasanya juga disertai dengan <i>multiplicity</i>
asosiasi berarah / <i>directed association</i> 	relasi antarkelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan <i>multiplicity</i>
generalisasi 	relasi antarkelas dengan makna generalisasi-spesialisasi (umum khusus)
kebergantungan / <i>dependency</i>	relasi antarkelas dengan makna

Simbol	Deskripsi
	kebergantungan antarkelas
<i>agregasi / aggregation</i>	relasi antarkelas dengan makna semua-bagian (<i>whole-part</i>)



Arah panah relasi pada diagram kelas mengarah pada diagram kelas yang lebih besar kontrolnya atau yang dipakai.

8.7 Object Diagram

Diagram objek menggambarkan struktur sistem dari segi penamaan objek dan jalannya objek dalam sistem. Pada diagram objek harus dipastikan semua kelas yang sudah didefinisikan pada diagram kelas harus dipakai objeknya, karena jika tidak, pendefinisian kelas itu tidak dapat dipertanggungjawabkan. Diagram objek juga berfungsi untuk mendefinisikan contoh nilai atau isi dari atribut tiap kelas.

Untuk apa mendefinisikan sebuah kelas sedangkan pada jalannya sistem, objeknya tidak pernah dipakai. Hubungan *link* pada diagram objek merupakan hubungan memakai dan dipakai dimana dua buah objek akan dihubungkan oleh *link* jika ada objek yang dipakai oleh objek lainnya.

Berikut adalah simbol-simbol yang ada pada diagram objek:

Simbol	Deskripsi
Objek	objek dari kelas yang berjalan saat sistem dijalankan

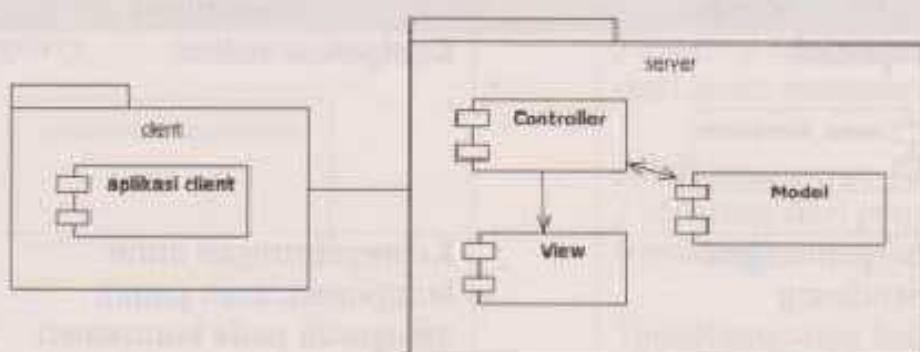
Simbol	Deskripsi
<div style="border: 1px solid black; padding: 5px; width: fit-content; margin: auto;"> name_objek : nama_kelas atribut = nilai </div>	
Link 	relasi antar objek

Penulisan nilai sama dengan penulisan pada kamus data DFD. Misalkan jika sebuah atribut dapat berisi lebih dari satu string maka akan ditulis dengan lambang {"nilai1", "nilai2", ..., "nilain"}, atau misalkan ingin menuliskan bahwa nilai sebuah atribut string dapat diisi nilai1 atau nilai2 maka akan ditulis ["nilai1" | "nilai2"].

8.8 Component Diagram

Diagram komponen atau *component diagram* dibuat untuk menunjukkan organisasi dan ketergantungan diantara kumpulan komponen dalam sebuah sistem. Diagram komponen fokus pada komponen sistem yang dibutuhkan dan ada di dalam sistem. Diagram komponen juga dapat digunakan untuk memodelkan hal-hal berikut:

- *source code* program perangkat lunak
- komponen *executable* yang dilepas ke *user*
- basis data secara fisik
- sistem yang harus beradaptasi dengan sistem lain
- *framework* sistem, *framework* pada perangkat lunak merupakan kerangka kerja yang dibuat untuk memudahkan pengembangan dan pemeliharaan aplikasi, contohnya seperti Struts dari Apache yang menggunakan prinsip desain Model-View-Controller (MVC) dimana *source code* program dikelompokkan berdasarkan fungsinya seperti pada gambar berikut:



Gambar 44 Ilustrasi Framework

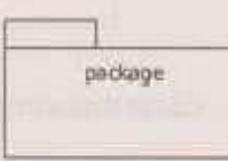
Di mana *controller* berisi *source code* yang menangani *request* dan validasi, *model* berisi *source code* yang menangani manipulasi data dan *business logic*, dan *view* berisi *source code* yang menangani tampilan.

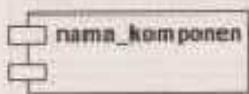
Komponen dasar yang biasanya ada dalam suatu sistem adalah sebagai berikut:

- Komponen *user interface* yang menangani tampilan
- Komponen *bussiness processing* yang menangani fungsi-fungsi proses bisnis
- Komponen *data* yang menangani manipulasi data
- Komponen *security* yang menangani keamanan sistem

Komponen lebih terfokus pada penggolongan secara umum fungsi-fungsi yang diperlukan.

Berikut adalah simbol-simbol yang ada pada diagram komponen:

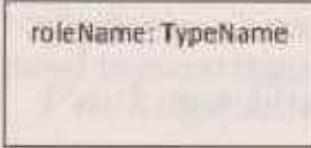
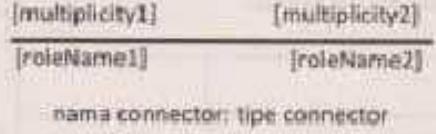
Simbol	Deskripsi
Package 	<i>package</i> merupakan sebuah bungkus dari satu atau lebih komponen

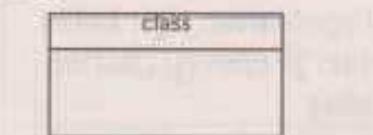
Simbol	Deskripsi
Komponen 	Komponen sistem
Kebergantungan <i>dependency</i> 	Kebergantungan antar komponen, arah panah mengarah pada komponen yang dipakai
Antarmuka / <i>interface</i> 	sama dengan konsep <i>interface</i> pada pemrograman berorientasi objek, yaitu sebagai antarmuka komponen agar tidak mengakses langsung komponen
Link 	relasi antar komponen

8.9 Composite Structure Diagram

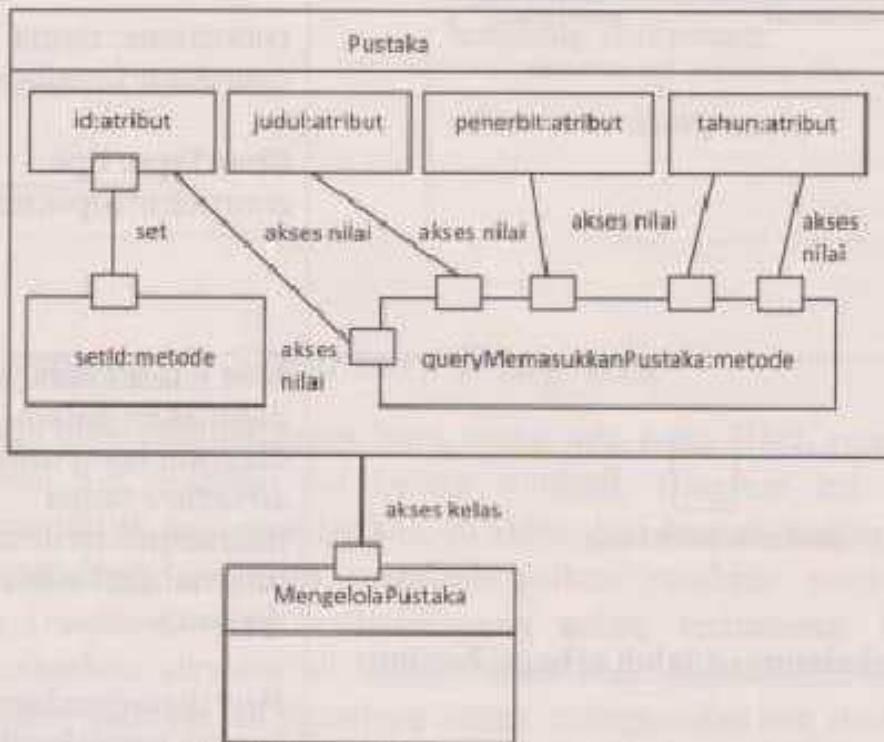
Composite structure diagram baru mulai ada pada UML versi 2.0, pada versi 1.x diagram ini belum muncul. Diagram ini dapat digunakan untuk menggambarkan struktur dari bagian-bagian yang saling terhubung maupun mendeskripsikan struktur pada saat berjalan (*runtime*) dari *instance* yang saling terhubung. Dapat menggambarkan struktur di dalam kelas atau kolaborasi. Contoh penggunaan diagram ini misalnya untuk menggambarkan deskripsi dari setiap bagian mesin yang saling terkait untuk menjalankan fungsi mesin tersebut, menggambarkan aliran data *router* pada jaringan computer, dan lain-lain.

Berikut adalah simbol-simbol yang ada pada diagram *composite structure*:

Simbol	Deskripsi
Property 	<p><i>Property</i> adalah satu set dari suatu <i>instance</i>.</p> <p><i>roleName</i>: peran / nama / identitas dari <i>property</i> (opsional)</p> <p><i>TypeName</i>: tipe kelas dari <i>property</i> (harus ada)</p>
Connector 	<p><i>Connector</i> adalah cara komunikasi dari 2 buah <i>instance</i>.</p> <p><i>connName</i>: nama <i>connector</i> (opsional)</p> <p><i>ConnType</i>: tipe <i>connector</i> (opsional)</p>
Port  <i>portName: EntityName[n]</i> <p>pemakaiannya adalah sebagai berikut:</p> 	<p><i>Port</i> adalah cara yang digunakan dalam diagram <i>composite structure</i> tanpa menampilkan detail internal dari suatu system.</p> <p><i>Port</i> digambarkan dalam bentuk kotak kecil yang menempel atau di dalam suatu <i>property</i>.</p> <p><i>Port</i> digambarkan menempel <i>property</i> jika fungsi tersebut dapat diakses <i>public</i>.</p>

Simbol	Deskripsi
	Sedangkan <i>port</i> digambarkan di dalam suatu <i>property</i> jika fungsi tersebut bersifat <i>protected</i> .
class 	Kelas; jika yang akan dijabarkan strukturnya adalah sebuah kelas

Contoh diagram *composite* adalah sebagai berikut:



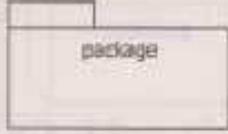
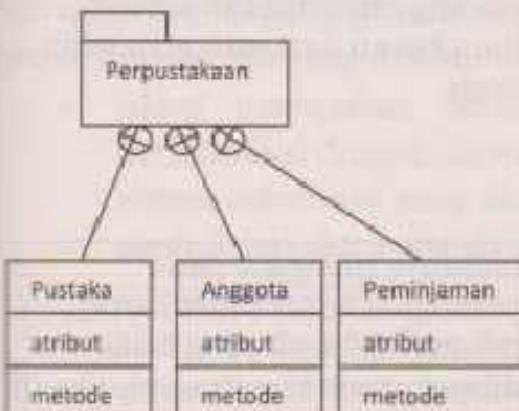
Gambar 45 Contoh Diagram Composite

Maksud dari diagram di atas adalah bahwa atribut id, judul, penerbit, dan tahun diakses nilainya oleh metode queryMemasukkanPustaka, dan atribut id diakses pada metode setId dengan memberikan nilai

kepada atribut id. Kelas Pustaka diakses oleh kelas MengelolaPustaka.

8.10 Package Diagram

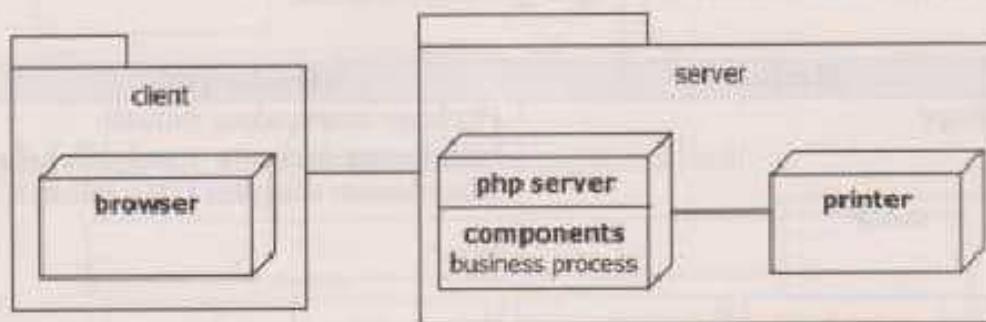
Package diagram menyediakan cara mengumpulkan elemen-elemen yang saling terkait dalam diagram UML. Hampir semua diagram dalam UML dapat dikelompokkan menggunakan *packagediagram*. Berikut ini simbol-simbol yang digunakan dalam

Simbol	Deskripsi
<i>Package</i> 	<i>Package</i> merupakan sebuah bungkus dari satu atau lebih kelas atau elemen diagram UML lainnya.
Elemen dalam <i>package</i> digambarkan di dalam <i>package</i>	
	
Elemen dalam <i>package</i> digambarkan di luar <i>package</i>	
	

8.11 Deployment Diagram

Diagram *deployment* atau *deployment diagram* menunjukkan konfigurasi komponen dalam proses eksekusi aplikasi. Diagram *deployment* juga dapat digunakan untuk memodelkan hal-hal berikut:

- sistem tambahan (*embedded system*) yang menggambarkan rancangan *device*, *node*, dan *hardware*.
- sistem *client/server* misalnya seperti gambar berikut:

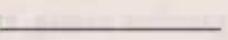


Gambar 46 Diagram Deployment Sistem Client / Server

- sistem terdistribusi murni
- rekayasa ulang aplikasi

Berikut adalah simbol-simbol yang ada pada diagram *deployment*:

Simbol	Deskripsi
Package	<i>package</i> merupakan sebuah bungkusan dari satu atau lebih <i>node</i>
Node	biasanya mengacu pada perangkat keras (<i>hardware</i>), perangkat lunak yang tidak dibuat sendiri (<i>software</i>), jika di dalam <i>node</i> disertakan

Simbol	Deskripsi
	komponen untuk mengkonsistenkan rancangan maka komponen yang diikutsertakan harus sesuai dengan komponen yang telah didefinisikan sebelumnya pada diagram komponen
Kebergantungan / <i>dependency</i> 	Kebergantungan antar <i>node</i> , arah panah mengarah pada <i>node</i> yang dipakai
<i>Link</i> 	relasi antar <i>node</i>

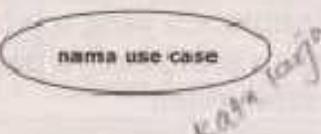
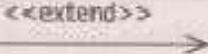
8.12 Use Case Diagram

Use case atau diagram *use case* merupakan pemodelan untuk kelakuan (*behavior*) sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu.

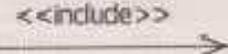
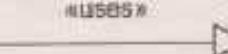
Syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu pendefinisian apa yang disebut aktor dan *use case*.

- Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.
- *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor.

Berikut adalah simbol-simbol yang ada pada diagram *use case*:

Simbol	Deskripsi
<i>Use case</i> 	fungsiionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor; biasanya dinyatakan dengan menggunakan kata kerja di awal di awal frase nama <i>use case</i>
<i>Aktor / actor</i> 	orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal frase nama aktor
<i>Asosiasi / association</i> 	komunikasi antara aktor dan <i>use case</i> yang berpartisipasi pada <i>use case</i> atau <i>use case</i> memiliki interaksi dengan aktor
<i>Ekstensi / extend</i> 	relasi <i>use case</i> tambahan ke sebuah <i>use case</i> dimana <i>use case</i> yang ditambahkan dapat berdiri sendiri walaupun tanpa <i>use case</i> tambahan itu; mirip dengan prinsip <i>inheritance</i> pada pemrograman berorientasi objek; biasanya <i>use case</i> tambahan memiliki nama depan yang sama dengan <i>use case</i> yang ditambahkan, misal

Simbol	Deskripsi
<pre> graph TD A([validasi username]) -- "<<extend>>" --> B([validasi user]) B -- "<<extend>>" --> C([validasi sidik jari]) </pre>	<p>arah panah mengarah pada <i>use case</i> yang ditambahkan; biasanya <i>use case</i> yang menjadi <i>extend</i>-nya merupakan jenis yang sama dengan <i>use case</i> yang menjadi induknya</p>
Generalisasi / generalization 	<p>Hubungan generalisasi dan spesialisasi (umum - khusus) antara dua buah <i>use case</i> dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya, misalnya:</p> <pre> graph TD A([ubah data]) --> B([mange data]) B --> C([hapus data]) </pre> <p>arah panah mengarah pada <i>use case</i> yang menjadi generalisasinya (umum)</p>
Menggunakan / include / uses	<p>relasi <i>use case</i> tambahan ke sebuah <i>use case</i> di mana <i>use case</i> yang ditambahkan memerlukan <i>use case</i> ini untuk menjalankan fungsinya atau sebagai syarat</p>

Simbol	Deskripsi
 	<p>dijalankan <i>use case</i> ini</p> <p>ada dua sudut pandang yang cukup besar mengenai <i>include</i> di <i>use case</i>:</p> <ul style="list-style-type: none"> • <i>include</i> berarti <i>use case</i> yang ditambahkan akan selalu dipanggil saat <i>use case</i> tambahan dijalankan, misal pada kasus berikut: <pre> graph TD A([validasi username]) -- "<<include>>" --> B([login]) </pre> <ul style="list-style-type: none"> • <i>include</i> berarti <i>use case</i> yang tambahan akan selalu melakukan pengecekan apakah <i>use case</i> yang ditambahkan telah dijalankan sebelum <i>use case</i> tambahan dijalankan, misal pada kasus berikut: <pre> graph TD A([validasi user]) -- "<<include>>" --> B([ubah data]) </pre> <p>Kedua interpretasi di atas dapat dianut salah satu atau keduanya tergantung pada pertimbangan dan interpretasi yang dibutuhkan.</p>



Arah panah relasi pada *use case* mengarah pada *use case* yang lebih besar kontrolnya atau yang dipakai.

Aksi Aktor	Reaksi Sistem
	anggota (semua kolom) dari anggota yang dipilih

Nama *Use case*: Melihatanggota
Skenario:

Aksi Aktor	Reaksi Sistem
Skenario Normal	
	1. Memeriksa status login
	2. Menampilkan data anggota yang dicari (belum semua kolom data anggota ditampilkan dan bisa banyak data anggota yang memenuhi data pencarian)
3. Memilih anggota yang dicari	
	4. Menampilkan data anggota (semua kolom) dari anggota yang dipilih

Nama *Use case*: Memasukkan peminjaman
Skenario:

Aksi Aktor	Reaksi Sistem
Skenario Normal	
	1. Memeriksa status login
2. Memasukkan data peminjaman sesuai kolom yang ada	
	3. Memeriksa valid tidaknya data masukan
	4. Menyimpan data peminjaman ke basis data
	5. Menampilkan pesan sukses disimpan
Skenario Alternatif	
	1. Memeriksa status login

Aksi Aktor	Reaksi Sistem
2. Memasukkan data peminjaman sesuai kolom yang ada	
	3. Memeriksa valid tidaknya data masukan
	4. Mengeluarkan pesan bahwa data masukan tidak valid
5. Memperbaiki data masukan yang tidak valid	
	6. Memeriksa valid tidaknya data masukan
	7. Menyimpan data peminjaman ke basis data
	8. Menampilkan pesan sukses disimpan

Nama *Use case*: Mengubah peminjaman
Skenario:

Aksi Aktor Skenario Normal	Reaksi Sistem
2. Memasukkan kata kunci dan kategori pencarian	1. Memeriksa status login
	3. Mencari data peminjaman yang akan diubah
5. Memilih data peminjaman yang akan	4. Menampilkan data peminjaman yang dicari (belum semua kolom data peminjaman ditampilkan dan bisa banyak data peminjaman yang memenuhi data pencarian)

Aksi Aktor	Reaksi Sistem
diubah	<p>6. Menampilkan semua kolom data peminjaman yang akan diubah</p>
7. Mengubah data peminjaman	<p>8. Memeriksa valid tidaknya data masukan</p> <p>9. Menyimpan data yang telah diubah ke basis data</p>
	<p>10. Menampilkan pesan bahwa data sukses disimpan</p>
Skenario Alternatif	
2. Memasukkan kata kunci dan kategori pencarian	<p>1. Memeriksa status login</p>
	<p>3. Mencari data peminjaman yang akan diubah</p>
5. Mengubah data peminjaman	<p>4. Menampilkan data peminjaman yang dicari</p>
	<p>6. Memeriksa valid tidaknya data masukan</p>
8. Memperbaiki data masukan yang diubah dan tidak valid	<p>7. Menampilkan pesan bahwa data masukan tidak valid</p>
	<p>9. Memeriksa valid tidaknya data masukan</p>
	<p>10. Menyimpan data yang telah diubah ke basis data</p>
	<p>11. Menampilkan pesan</p>

Aksi Aktor	Reaksi Sistem
	bahwa data sukses disimpan

Nama *Use case*: Menghapus peminjaman

Skenario:

Aksi Aktor	Reaksi Sistem
Skenario Normal	
2. Memasukkan kata kunci dan kategori pencarian	1. Memeriksa status login
	3. Mencari data peminjaman yang akan dihapus
	4. Menampilkan data peminjaman yang dicari (belum semua kolom data peminjaman ditampilkan dan bisa banyak data peminjaman yang memenuhi data pencarian)
5. Memilih data peminjaman yang akan dihapus	
	6. Menampilkan pesan konfirmasi apakah data akan benar-benar dihapus
7. Mengklik pilihan setuju data dihapus	
	8. Menghapus data peminjaman dari basis data
	9. Menampilkan pesan bahwa data sukses dihapus
Skenario Alternatif	
2. Memasukkan kata kunci	1. Memeriksa status login

Aksi Aktor dan kategori pencarian	Reaksi Sistem
	3. Mencari data peminjaman yang akan dihapus 4. Menampilkan data peminjaman yang dicari (belum semua kolom data peminjaman ditampilkan dan bisa banyak data peminjaman yang memenuhi data pencarian)
5. Memilih data peminjaman yang akan dihapus	
7. Mengklik pilihan tidak setuju data dihapus	6. Menampilkan pesan konfirmasi apakah data akan benar-benar dihapus 8. Kembali ke form pencarian peminjaman

Nama Use case: Mencari peminjaman

Skenario:

Aksi Aktor Skenario Normal	Reaksi Sistem
	1. Memeriksa status login
2. Memasukkan kata kunci dan kategori pencarian	3. Mencari data peminjaman yang akan dicari 4. Menampilkan data peminjaman yang dicari (belum semua kolom data peminjaman ditampilkan dan bisa

Aksi Aktor	Reaksi Sistem
	banyak data peminjaman yang memenuhi data pencarian)
5. Memilih peminjaman yang dicari	6. Menampilkan data peminjaman (semua kolom) dari peminjaman yang dipilih
Skenario Alternatif	
	1. Memeriksa status login
2. Memasukkan kata kunci dan kategori pencarian	3. Mencari data peminjamanyang akan dicari
	4. Menampilkan pesan data peminjaman tidak ada
5. Memasukkan kata kunci dan kategori pencarian	6. Mencari data peminjaman yang akan dieari
	7. Menampilkan data peminjamanyang dicari (belum semua kolom data peminjaman ditampilkan dan bisa banyak data peminjaman yang memenuhi data pencarian)
8. Memilih peminjaman yang dicari	
	9. Menampilkan data peminjaman (semua kolom) dari peminjaman yang dipilih

Nama Use case: Melihatpeminjaman

Skenario:

Aksi Aktor	Reaksi Sistem
Skenario Normal	
	1. Memeriksa status login
	2. Menampilkan data peminjaman yang dicari (belum semua kolom data peminjaman ditampilkan dan bisa banyak data peminjaman yang memenuhi data pencarian)
3. Memilih peminjaman yang dicari	
	4. Menampilkan data peminjaman (semua kolom) dari peminjaman yang dipilih

Nama Use case: Memasukkan petugas

Skenario:

Aksi Aktor	Reaksi Sistem
Skenario Normal	
	1. Memeriksa status login
2. Memasukkan data petugas sesuai kolom yang ada	
	3. Memeriksa valid tidaknya data masukan
	4. Menyimpan data petugas ke basis data
	5. Menampilkan pesan sukses disimpan
Skenario Alternatif	
	1. Memeriksa status login
2. Memasukkan data petugas sesuai kolom yang ada	
	3. Memeriksa valid tidaknya data masukan

Aksi Aktor	Reaksi Sistem
	4. Mengeluarkan pesan bahwa data masukan tidak valid
5. Memperbaiki data masukan yang tidak valid	
	6. Memeriksa valid tidaknya data masukan
	7. Menyimpan data petugas ke basis data
	8. Menampilkan pesan sukses disimpan

Nama *Use case*: Mengubah petugas
Skenario:

Aksi Aktor	Reaksi Sistem
Skenario Normal	
2. Memasukkan kata kunci dan kategori pencarian	1. Memeriksa status login
	3. Mencari data petugas yang akan diubah
	4. Menampilkan data petugas yang dicari (belum semua kolom data petugas ditampilkan dan bisa banyak data petugas yang memenuhi data pencarian)
5. Memilih data petugas yang akan diubah	
	6. Menampilkan semua kolom data petugas yang akan diubah
7. Mengubah data petugas	8. Memeriksa valid tidaknya data masukan
	9. Menyimpan data yang telah diubah ke basis

Aksi Aktor	Reaksi Sistem
	data
	10. Menampilkan pesan bahwa data sukses disimpan
Skenario Alternatif	
2. Memasukkan kata kunci dan kategori pencarian	1. Memeriksa status login 3. Mencari data petugas yang akan diubah 4. Menampilkan data petugas yang dicari (belum semua kolom data petugas ditampilkan dan bisa banyak data petugas yang memenuhi data pencarian)
5. Memilih data petugas yang akan diubah	6. Menampilkan semua kolom data petugas yang akan diubah 8. Memeriksa valid tidaknya data masukan 9. Menampilkan pesan bahwa data masukan tidak valid
7. Mengubah data petugas	
10. Memperbaiki data masukan yang diubah dan tidak valid	11. Memeriksa valid tidaknya data masukan 12. Menyimpan data yang telah diubah ke basis data 13. Menampilkan pesan bahwa data sukses disimpan

Nama Use case: Menghapus anggota

Skenario:

Aksi Aktor	Reaksi Sistem
Skenario Normal	
2. Memasukkan kata kunci dan kategori pencarian	1. Memeriksa status login 3. Mencari data petugas yang akan dihapus 4. Menampilkan data petugas yang dicari (belum semua kolom data petugas ditampilkan dan bisa banyak data petugas yang memenuhi data pencarian)
5. Memilih data petugas yang akan dihapus	6. Menampilkan pesan konfirmasi apakah data akan benar-benar dihapus
7. Mengklik pilihan setuju data dihapus	8. Menghapus data petugas dari basis data 9. Menampilkan pesan bahwa data sukses dihapus
Skenario Alternatif	
2. Memasukkan kata kunci dan kategori pencarian	1. Memeriksa status login 3. Mencari data petugas yang akan dihapus 4. Menampilkan data petugas yang dicari (belum semua kolom data petugas ditampilkan dan bisa banyak data)

Aksi Aktor	Reaksi Sistem
	petugas yang memenuhi data pencarian)
5. Memilih data petugas yang akan dihapus	
	6. Menampilkan pesan konfirmasi apakah data akan benar-benar dihapus
7. Mengklik pilihan tidak setuju data dihapus	
	8. Kembali ke form pencarian petugas

Nama Use case: Mencari petugas
 Skenario:

Aksi Aktor	Reaksi Sistem
Skenario Normal	
2. Memasukkan kata kunci dan kategori pencarian	1. Memeriksa status login
	3. Mencari data petugas yang akan dicari
	4. Menampilkan data petugas yang dicari (belum semua kolom data petugas ditampilkan dan bisa banyak data petugas yang memenuhi data pencarian)
5. Memilih petugas yang dicari	
	6. Menampilkan data petugas (semua kolom) dari petugas yang dipilih
Skenario Alternatif	
8. Memasukkan kata kunci dan kategori pencarian	7. Memeriksa status login
	9. Mencari data petugas

Aksi Aktor	Reaksi Sistem
	yang akan dicari
	10. Menampilkan pesan data petugas tidak ada
11. Memasukkan kata kunci dan kategori pencarian	
	12. Mencari data petugas yang akan dicari
	13. Menampilkan data petugas yang dicari (belum semua kolom data petugas ditampilkan dan bisa banyak data petugas yang memenuhi data pencarian)
14. Memilih petugas yang dicari	
	15. Menampilkan data petugas (semua kolom) dari petugas yang dipilih

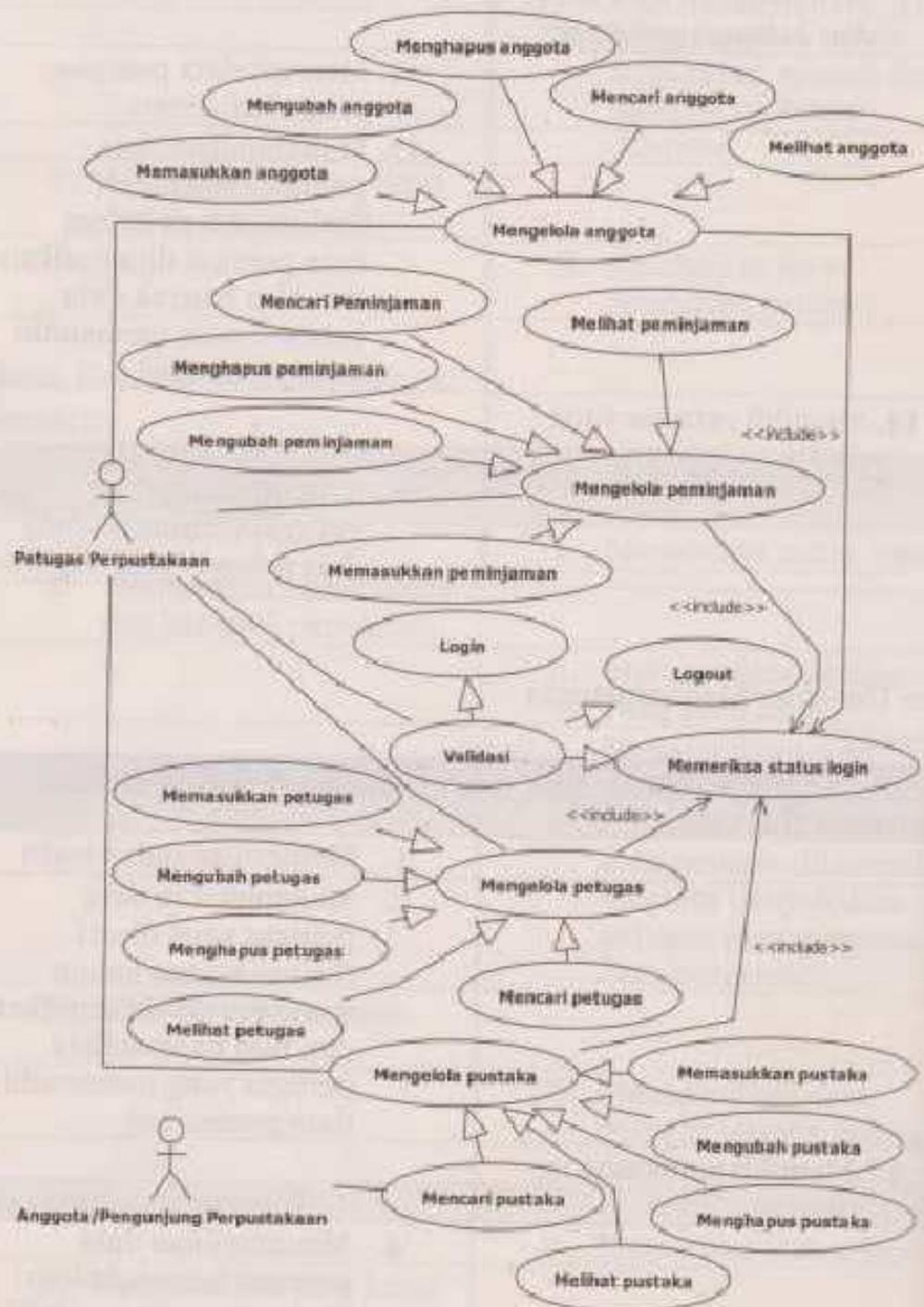
Nama *Use case*: Melihatpetugas

Skenario:

Aksi Aktor	Reaksi Sistem
Skenario Normal	
	1. Memeriksa status login
	2. Menampilkan data petugas yang dicari (belum semua kolom data petugas ditampilkan dan bisa banyak data petugas yang memenuhi data pencarian)
3. Memilih petugas yang dicari	
	4. Menampilkan data petugas (semua kolom) dari petugas yang dipilih

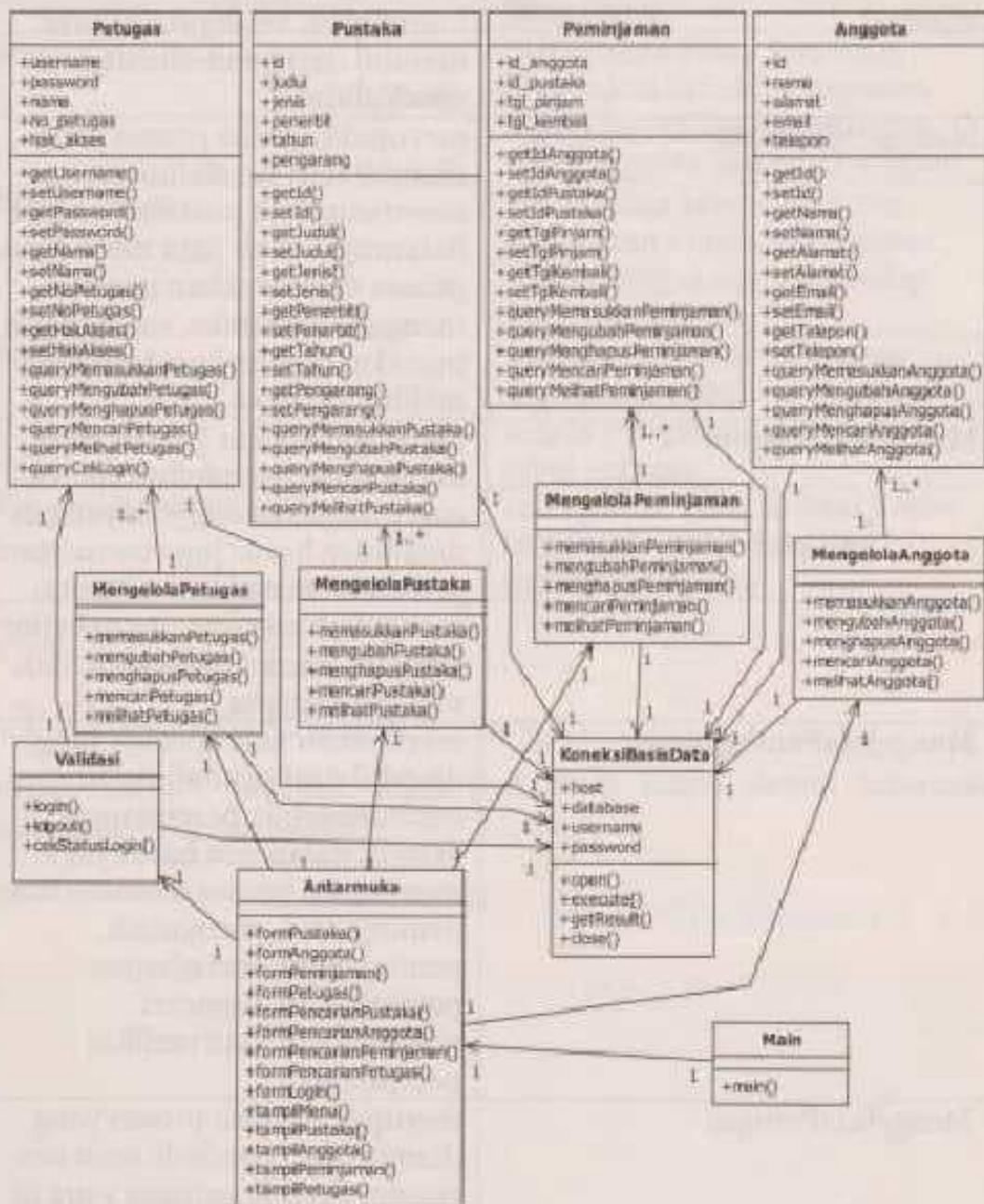
4. Diagram Use case

Berikut adalah diagram use case dari sistem informasi manajemen perpustakaan:



Gambar 53 Diagram Use case Perpustakaan

9.2 Diagram Kelas



Gambar 54 Diagram Kelas Studi Kasus

Keterangan:

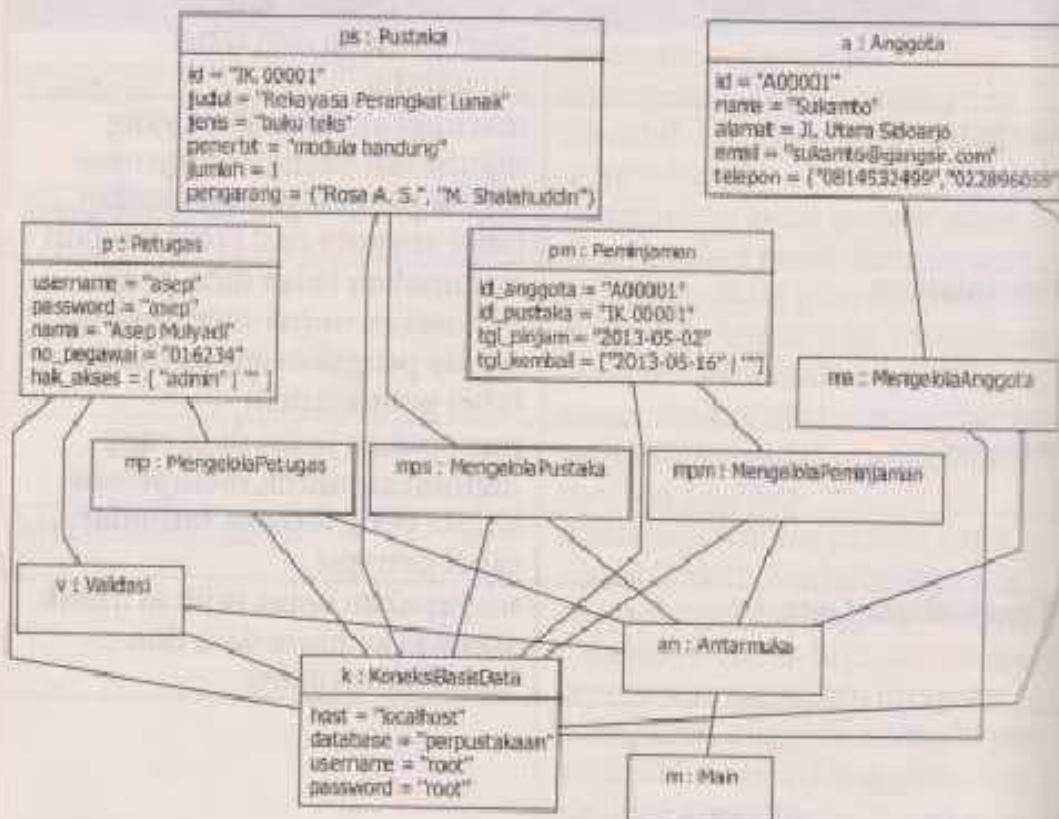
Nama Kelas	Keterangan
Main	merupakan kelas main
Antarmuka	merupakan kelas yang

Nama Kelas	Keterangan
Validasi	menangani tampilan merupakan kelas proses yang diambil dari pendefinisian <i>use case</i> Validasi
MengelolaPustaka	merupakan kelas proses yang diambil dari pendefinisian <i>use case</i> mengelola pustaka yang di dalamnya harus juga menangani proses memasukkan pustaka, mengubah pustaka, menghapus pustaka, mencari pustaka, dan melihat pustaka
MengelolaAnggota	merupakan kelas proses yang diambil dari pendefinisian <i>use case</i> mengelola anggota yang di dalamnya harus juga menangani proses memasukkan anggota, mengubah anggota, menghapus anggota, mencari anggota, dan melihat anggota
MengelolaPeminjaman	merupakan kelas proses yang diambil dari pendefinisian <i>use case</i> mengelola peminjaman yang di dalamnya harus juga menangani proses memasukkan peminjaman, mengubah peminjaman, menghapus peminjaman, mencari peminjaman, dan melihat peminjaman
MengelolaPetugas	merupakan kelas proses yang diambil dari pendefinisian <i>use case</i> mengelola petugas yang di dalamnya harus juga menangani proses memasukkan petugas, mengubah petugas, menghapus petugas, mencari petugas, dan melihat petugas
Pustaka	merupakan kelas data yang digunakan untuk memproses segala pengaksesan terhadap

Nama Kelas	Keterangan
	tabel pustaka dan tabel pengarang
Anggota	merupakan kelas data yang digunakan untuk memproses segala pengaksesan terhadap tabel anggota dan tabel telepon
Peminjaman	merupakan kelas data yang digunakan untuk memproses segala pengaksesan terhadap tabel peminjaman
Petugas	merupakan kelas data yang digunakan untuk memproses segala pengaksesan terhadap tabel petugas
KoneksiBasisData	merupakan kelas utilitas untuk koneksi ke basis data dan melakukan <i>query</i>

9.3 Diagram Objek

Berikut adalah diagram objek dari studi kasus sistem informasi manajemen perpustakaan:

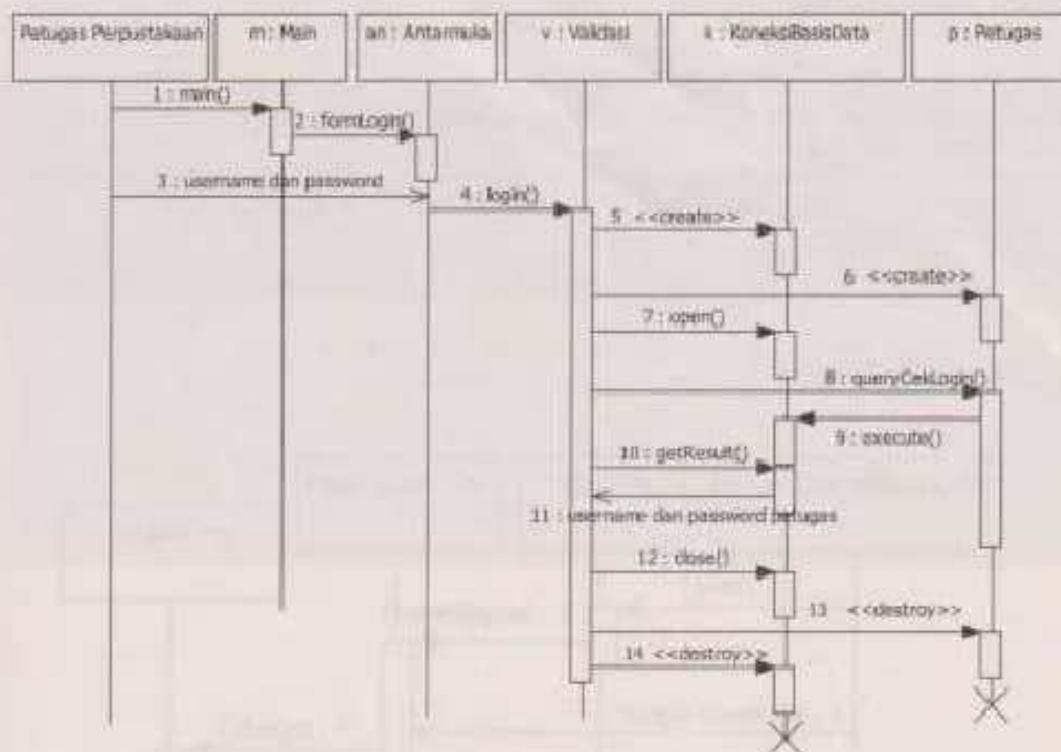


Gambar 55 Diagram Objek Studi Kasus

9.4 Diagram Sekuen

Berikut adalah diagram sekuen dari sistem informasi manajemen perpustakaan:

1. Use case: Login



Gambar 1 Diagram sekuen - Login

Dari diagram sekuen di atas maka urutan proses pada metode *login* di kelas *Validasi* adalah sebagai berikut (*dalam pseudocode*):

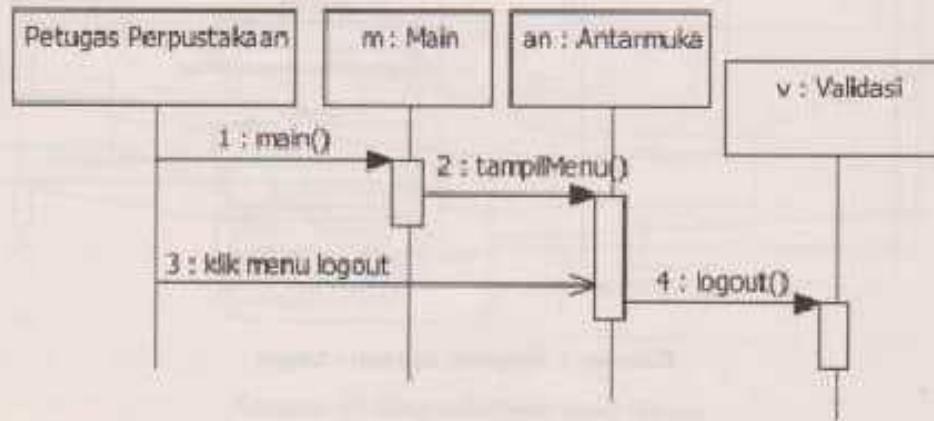
```
procedure login(username: string, password: string) {
    KoneksiBasisData k = new KoneksiBasisData // create
    Petugas p = new Petugas // create
    k.open()
    p.queryCekLogin(k, username, password)
    if (x.getResult() <> NULL){
        SESSION(status) = "valid"
    } else{
        SESSION(status) = "unvalid"
    }
    k.close()
    delete(p) //destroy
    delete(k) //destroy
}
```

Metode *execute* dijalankan di dalam metode *queryCekLogin()* dan hasil dari *query* yaitu *username* dan *password* didapat dari

metode getResult(). Berikut adalah tahapan metode queryCekLogin() dalam *pseudocode*:

```
procedure queryCekLogin(k: KoneksiBasisData, username: string,  
password: string)  
    query = "SELECT username, password FROM tptugas where  
username=''" + username + "' AND password=''" + password + "'";  
k.execute(query)
```

2. Use case: Logout



Gambar 57 Diagram Sekuen untuk Use Case Logout

Proses logout hanya mengosongkan session status login.

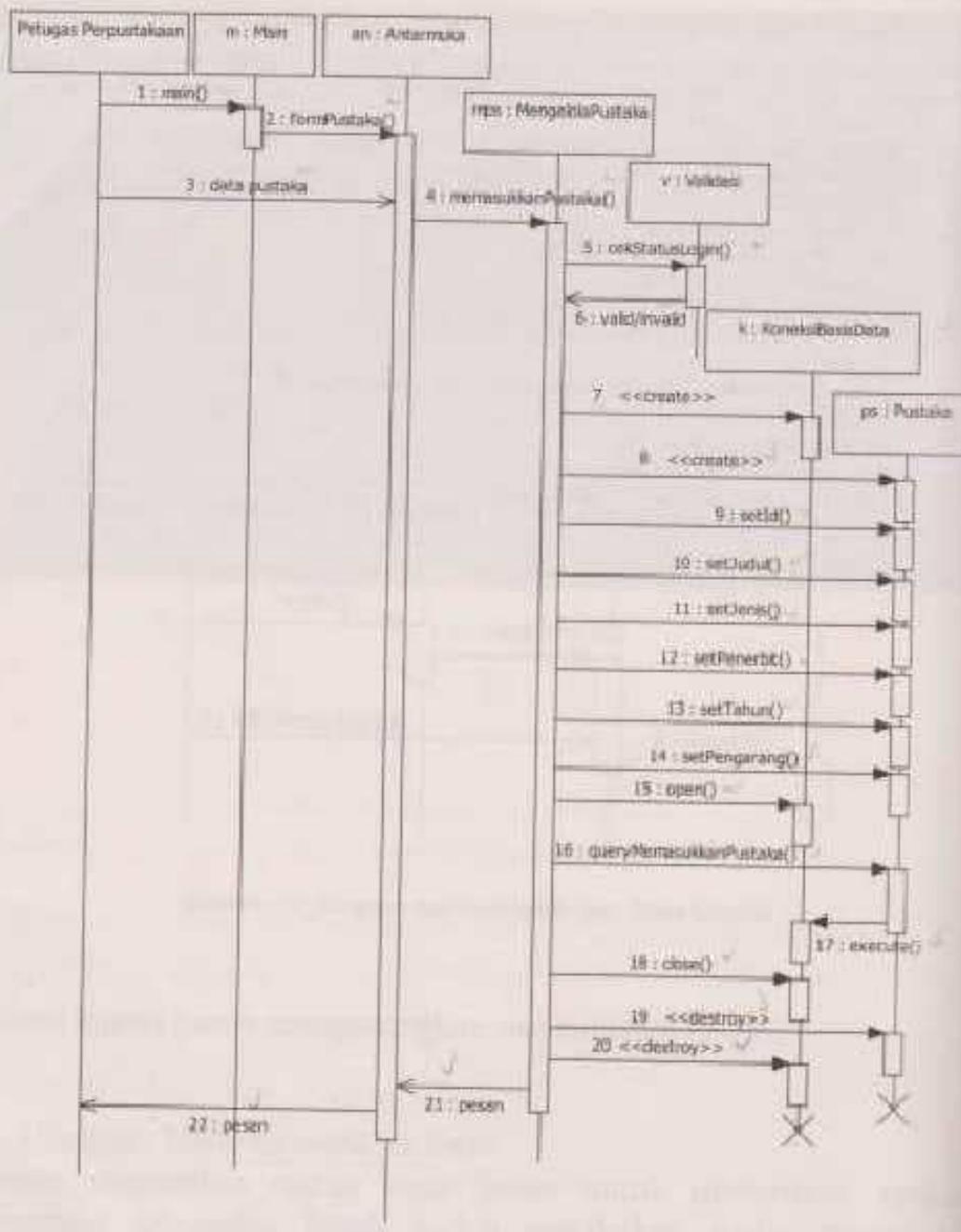
3. Use case: Memeriksa status login

Proses memeriksa status login berisi untuk memeriksa apakah pengguna perangkat lunak sudah melakukan *login*. Proses ini digunakan oleh *use case* lain sehingga akan menjadi bagian diagram sekuen dari *use case* lain yang menggunakaninya.

4. Use case: Memasukkan pustaka

Dari gambar diagram sekuen di bawah ini maka tahapan yang ada dalam metode memasukkanPustaka() adalah sebagai berikut (dalam *pseudocode*):

```
function memasukkanPustaka(id: string, judul: string, jenis: string,
penerbit: string, tahun: string, pengarang: string[], v: Validasi){
    if(v.cekStatusLogin()){
        KoneksiBasisData k = new KoneksiBasisData //create
        Pustaka ps = new Pustaka //create
        ps.setId(id)
        ps.setJudul(judul)
        ps.setJenis(jenis)
        ps.setPenerbit(penerbit)
        ps.setTahun(tahun)
        ps.setPengarang(pengarang)
        k.open()
        ps.queryMemasukkanPustaka(k) //k.execute ada didalamnya
        k.close()
        delete(ps) //destroy
        delete(k) //destroy
        return "Penyimpanan Berhasil!"
    }
}
```

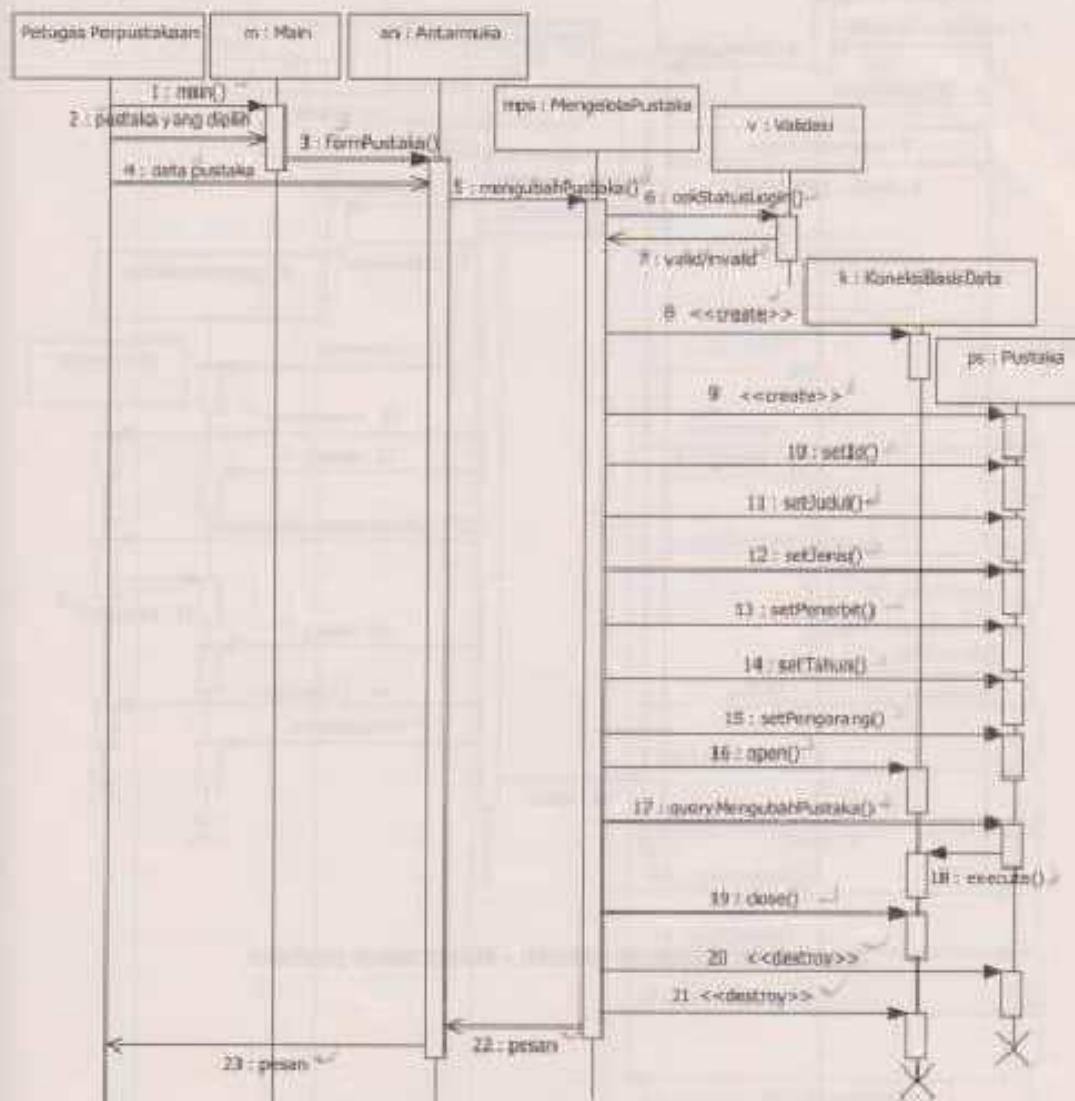


Gambar 58 Diagram sekuen - Memasukkan pustaka

5. Use case: Mengubah pustaka

Proses Mengubah pustaka diawali dengan proses Mencari pustaka yang dapat dilihat pada diagram sekuen Mencari pustaka.

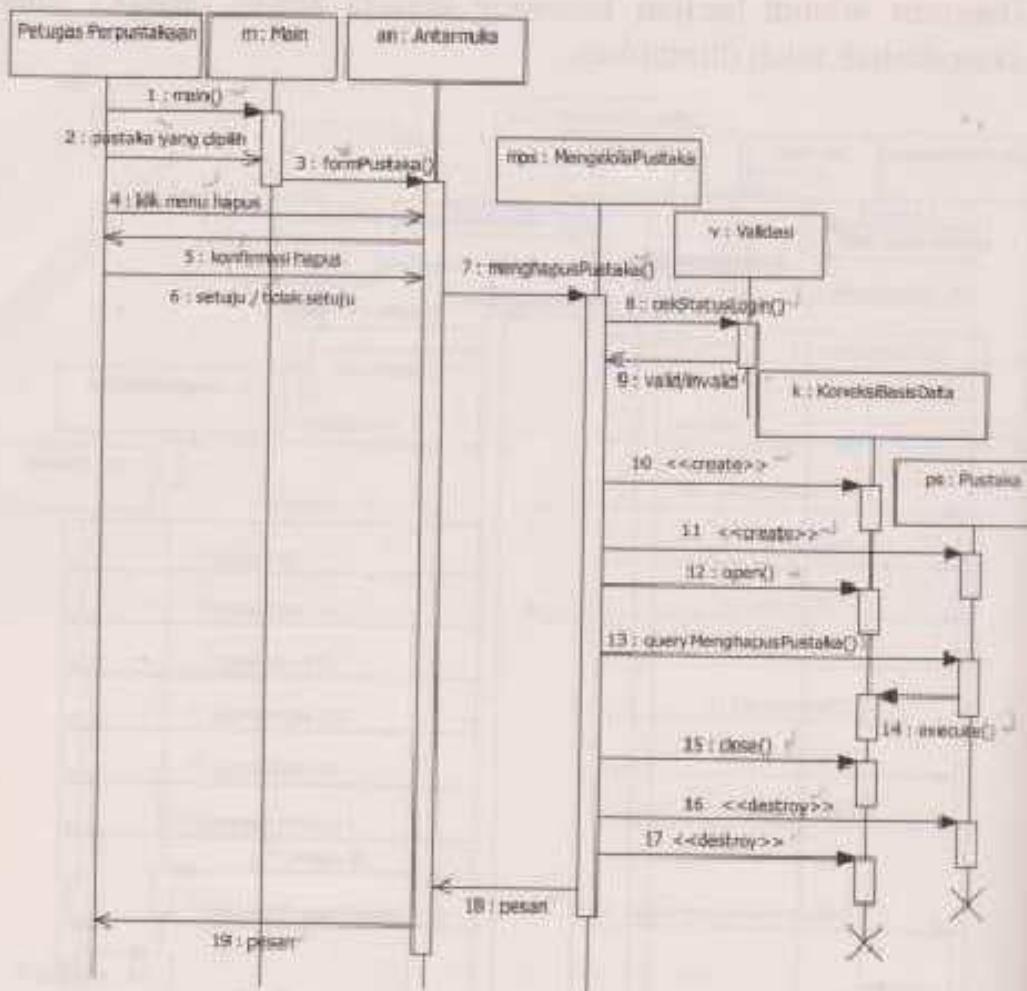
Diagram sekuen berikut langsung kepada ketika pustaka yang akan diubah telah ditemukan.



Gambar 59 Diagram sekuen - Mengubah pustaka

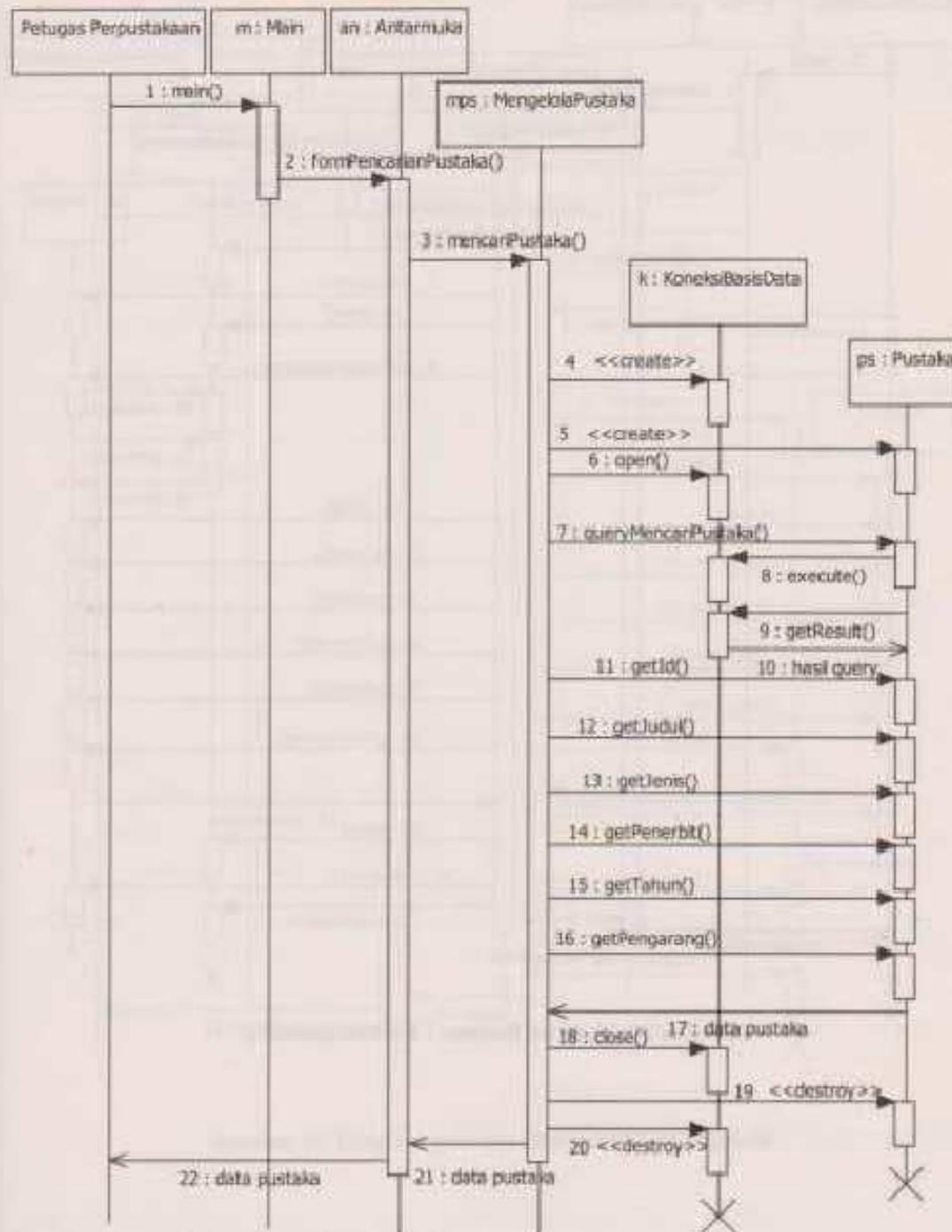
Use case: Menghapus pustaka

Proses Menghapus pustaka diawali dengan proses Mencari pustaka yang dapat dilihat pada diagram sekuen Mencari pustaka. Diagram sekuen berikut langsung kepada ketika pustaka yang akan dihapus telah ditemukan.



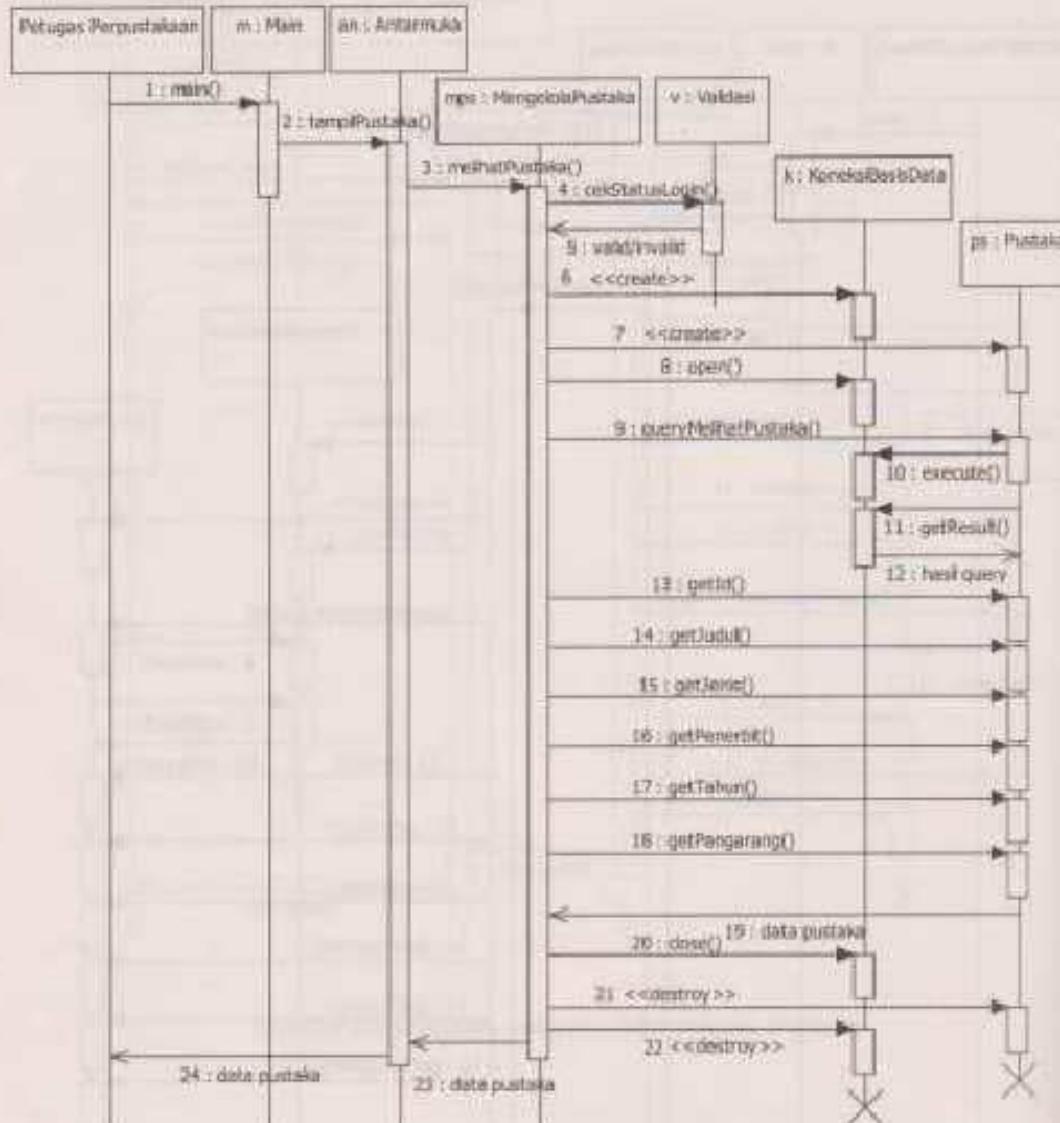
Gambar 60 Diagram sekuen - Menghapus pustaka

6. Use case: Mencari pustaka



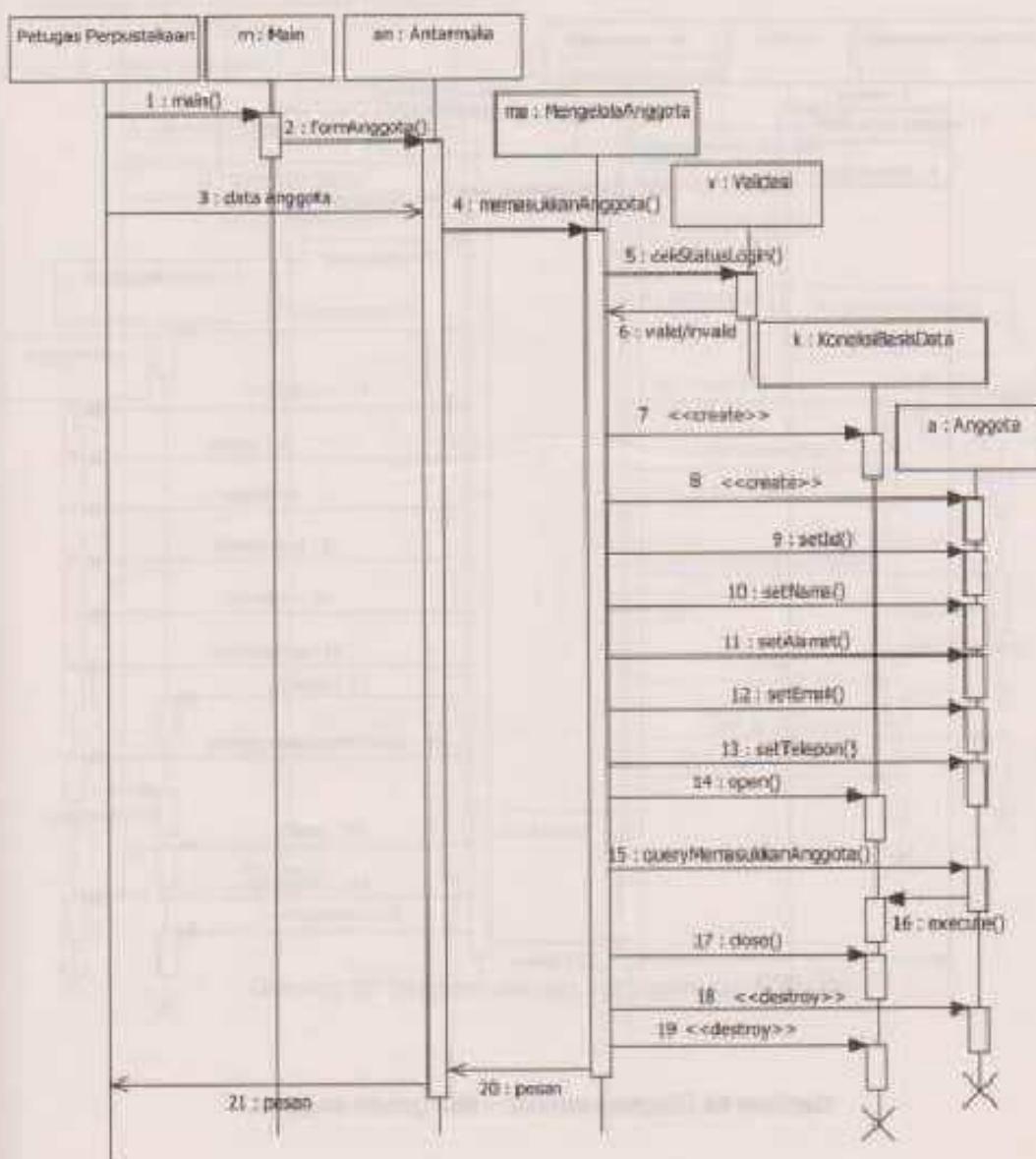
Gambar 61 Diagram Sekuan - Mencari pustaka

7. Use case: Melihat pustaka



Gambar 62 Diagram Sekuen - Melihat pustaka

8. Use case: Memasukkan anggota

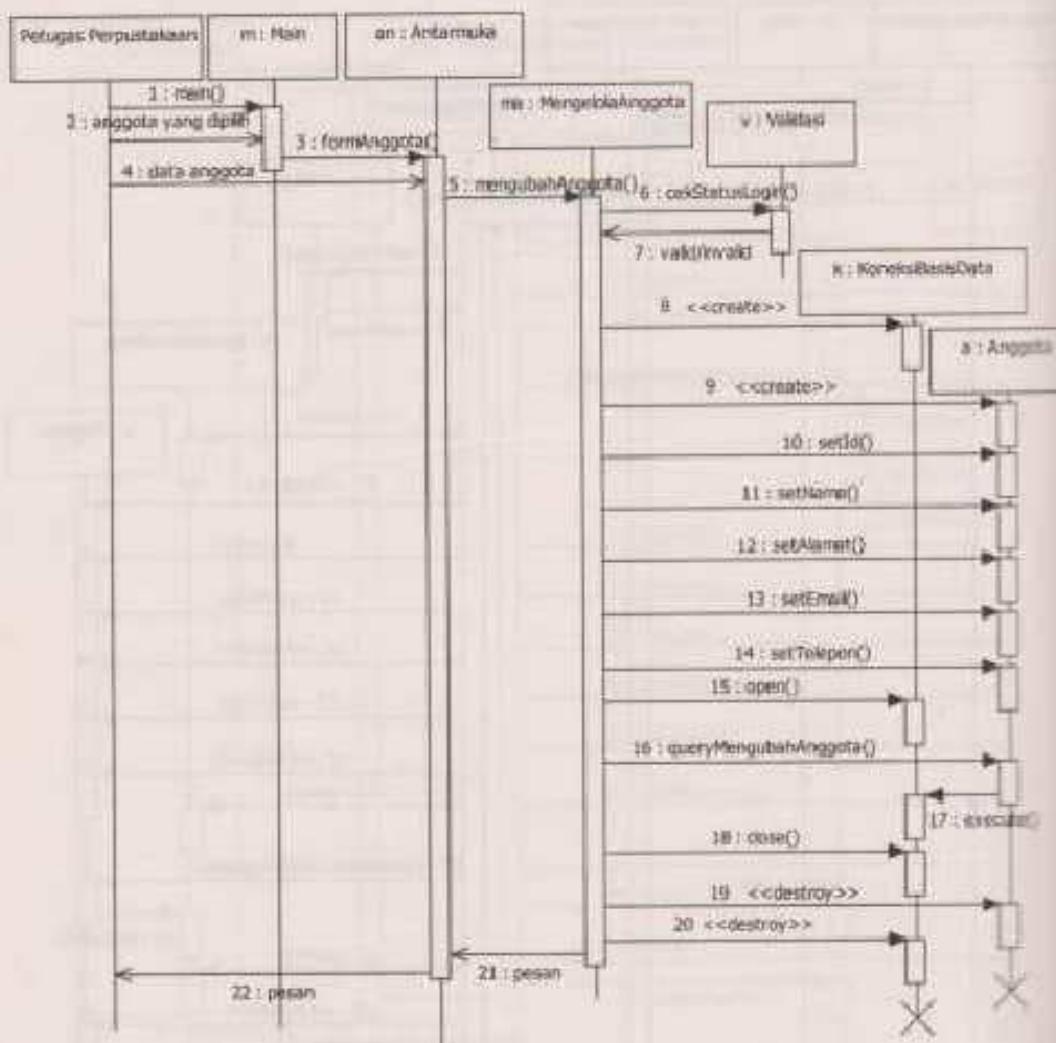


Gambar 63 Diagram sekuen - Memasukkan anggota

9. Use case: Mengubah anggota

Proses Mengubah anggota diawali dengan proses Mencari anggota yang dapat dilihat pada diagram sekuen Mencari anggota. Diagram

sekuen berikut langsung kepada ketika anggota yang akan diubah telah ditemukan.



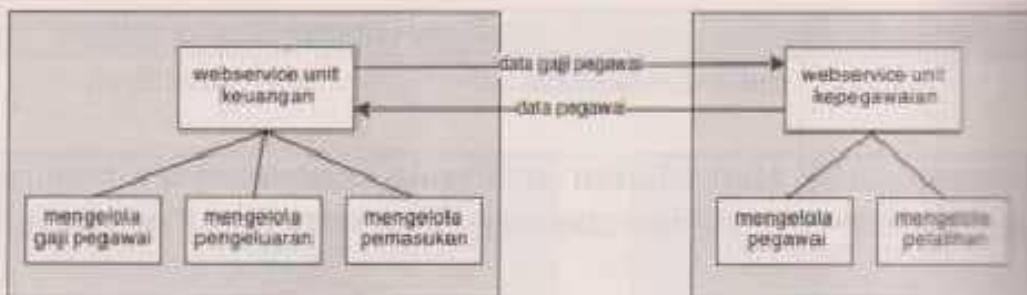
Gambar 64 Diagram sekuen - Mengubah anggota

10. Use case: Menghapus anggota

Proses Menghapus anggota diawali dengan proses Mencari anggota yang dapat dilihat pada diagram sekuen Mencari anggota. Diagram sekuen berikut langsung kepada ketika anggota yang akan dihapus telah ditemukan.

Atribut	Penjelasan
	yang diwakilinya
Kemampuan diaplikasikan	Menyediakan antarmuka/ <i>interface</i> yang mampu merepresentasikan semua kelas yang diwakilinya
Partisipasi dan kolaborasi	Menyediakan antarmuka untuk mewakili kumpulan kelas lain
Konsekuensi	Memperpanjang pengaksesan objek-objek yang diwakilinya
Implementasi	Mengapsulasi dengan membuat kelas abstrak yang mewakili kelas-kelas yang diwakilinya

Pola *bridge* biasanya digunakan untuk dua perangkat lunak yang saling terhubung namun membutuhkan akses dua arah. Misalkan perangkat lunak yang membutuhkan pertukaran data dimana dari kedua perangkat lunak itu memiliki DBMS yang berbeda, maka biasanya sering menggunakan teknologi *webservice* di kedua perangkat lunak yang saling terhubung tersebut. Sebagai contoh riil misalkan ada dua buah unit, yaitu unit keuangan yang bertanggung jawab memberikan gaji pada karyawan dan unit kepegawaian. Kedua unit itu memiliki perangkat lunak yang berbeda. Unit keuangan membutuhkan data pegawai dan unit kepegawaian membutuhkan data jumlah gaji pegawai. Maka kedua perangkat lunak itu harus saling menarik data. Agar keamanan tetap terjaga dan agar unit lain tidak terlalu banyak memiliki akses pada perangkat lunak unit lain, maka dipakailah teknologi *webservice* di kedua perangkat lunak untuk saling mengirim data yang dibutuhkan. Berikut ilustrasinya:

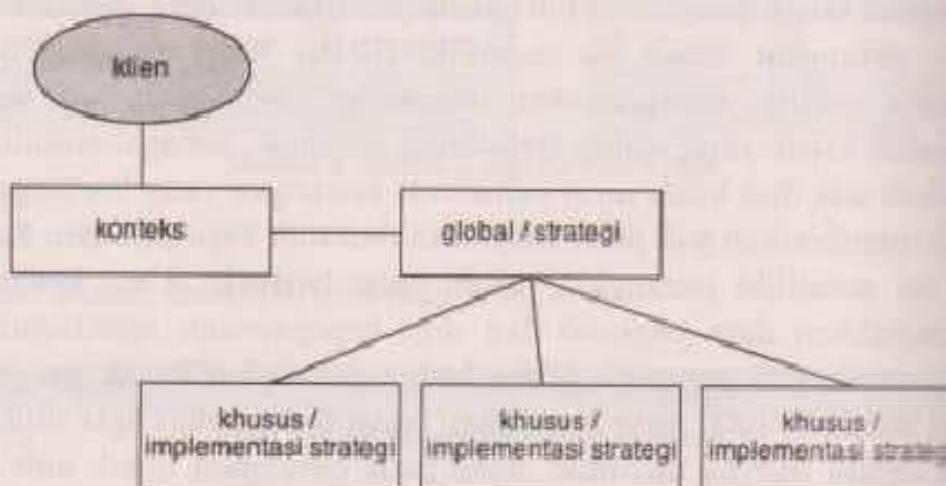


Gambar 9 Ilustrasi Implementasi Pola Bridge

3. Pola yang Berbasis pada Kebiasaan (*Behavioral patterns*)

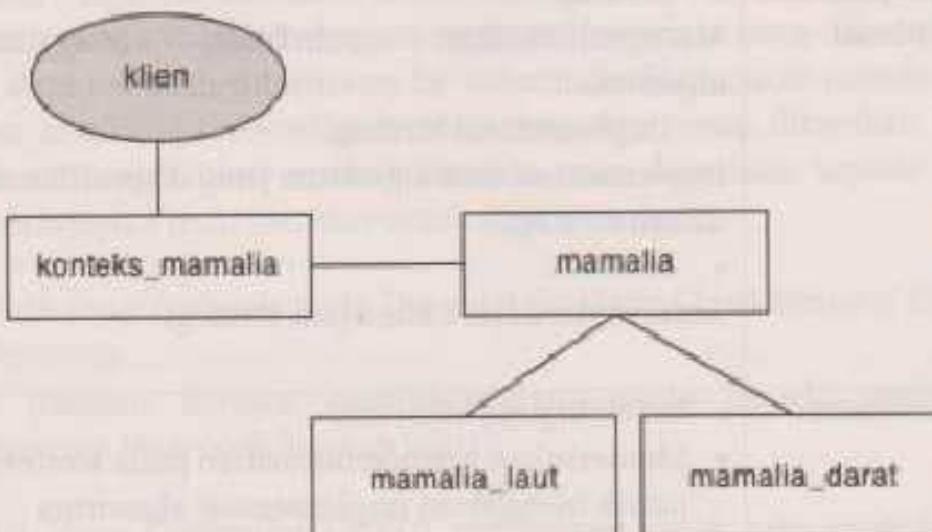
Pola disusun dengan menitikberatkan pada spesifikasi kelakuan sistem aplikasi yang akan dibuat apakah pola yang dibuat sudah sesuai dengan kelakuan atau kemampuan aplikasi yang diharapkan.

- a. *Strategy pattern* merupakan sebuah antarmuka (*interface*) yang berperan sebagai penggeneralisasi dari kelas yang lebih bersifat khusus. Arsitektur pola *strategy* adalah sebagai berikut:



Gambar 10 Arsitektur pola *strategy*

Misalkan:



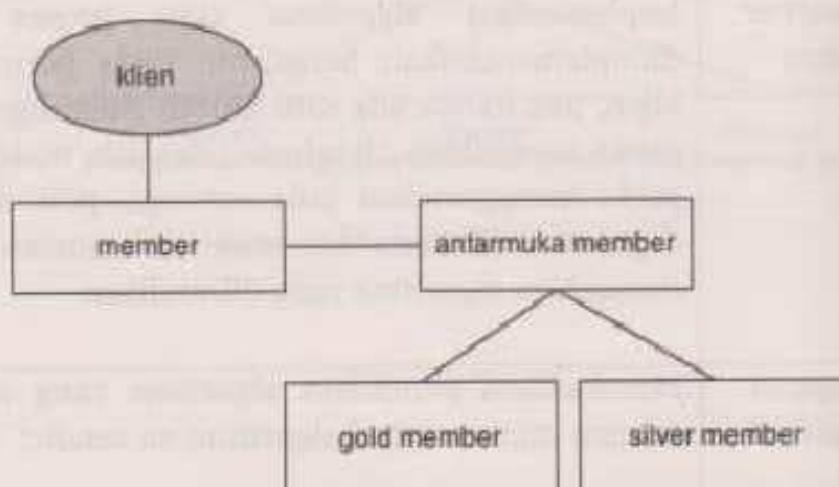
Gambar 11 Contoh arsitektur pola strategy

Berikut adalah deskripsi dari pola strategi:

Atribut	Penjelasan
Nama	Pola Strategi(<i>Strategy Pattern</i>)
Tujuan	Mengizinkan penggunaan aturan atau metode yang berbeda tergantung pada konteksnya
Motivasi/Permasalahan	Implementasi algoritma atau proses yang diimplementasikan bergantung pada permintaan klien, jika hanya ada satu aturan yang digunakan untuk mengakses algoritma yang ada maka tidak perlu menggunakan pola strategi, pola strategi digunakan jika ada dua atau lebih aturan untuk mengakses algoritma yang dibutuhkan
Kemampuan diaplikasikan	Memisahkan pemilihan algoritma yang diakses dengan implementasi algoritma itu sendiri

Atribut	Penjelasan
Partisipasi dan kolaborasi	<ul style="list-style-type: none"> • Strategi Menspesifikasikan pemilihan penggunaan algoritma • Implementasi Strategi Implementasi dari algoritma yang dispesifikasikan dalam strategi • Konteks Antarmuka antara klien dan strategi
Konsekuensi	<ul style="list-style-type: none"> • kekeluargaan algoritma • Memerlukan metode tambahan pada konteks untuk mengakses implementasi algoritma
Implementasi	Membuat kelas yang menggunakan algoritma (konteks) dan kelas strategi yang memiliki metode abstrak untuk menspesifikasikan pengaksesan implementasi algoritma

Pola strategi digunakan jika klien akan mengakses sebuah layanan dengan satu nama namun sebenarnya dari sisi penyedia layanan sebenarnya ada beberapa layanan yang dapat diakses dengan satu layanan di sisi klien. Misalkan ilustrasi di bawah ini:



Gambar 12 Ilustrasi Implementasi Pola Strategi

Dari ilustrasi di atas misalkan klien akan mengakses layanan untuk *member*, maka di sisi klien hanya akan melihat layanan itu sebagai layanan untuk *member* dengan memasukkan data yang dibutuhkan. Lalu data itu akan diteruskan ke antarmuka (*interface*) *member*. Di dalam antarmuka *member* berdasarkan data yang diberikan klien maka akan dipilihkan apakah layanan yang diberikan kepada klien adalah layanan *gold member* atau *silver member*.

4. Pola yang Berbasis pada Desain Arsitektur (*Architectural Design Pattern*)

Pola disusun dengan menitikberatkan pada hirarki arsitektur (pemisahan blok-blok lingkup kerja).

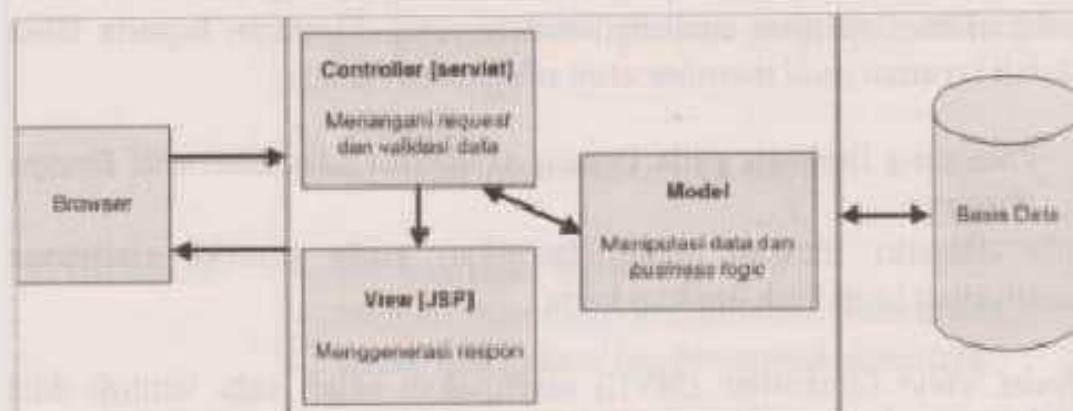
Model View Controller (MVC) merupakan salah satu contoh dari *Architectural Design Pattern*. Pada saat ini banyak *framerwork* yang didedikasikan pemrograman berorientasi objek.

Apa itu *framework*? *Framework* adalah kerangka kerja yang memudahkan *programmer* untuk membuat sebuah aplikasi sehingga *programmer* akan lebih mudah melakukan perubahan (*customize*) terhadap aplikasinya dan dapat memakainya kembali untuk aplikasi lain yang sejenis. Apa yang disebut sebagai *framework* biasanya sudah memiliki aplikasi implementasinya, maka beda antara pola desain dan *frarnework* adalah pola desain merupakan sebuah konsep sedangkan *framework* bisa merupakan aplikasi hasil implementasi dari pola desain atau aplikasi kerangka kerja yang tidak berasal dari implementasi sebuah konsep.

Contoh-contoh *framework* yang sering digunakan antara lain sebagai berikut:

- PHP: CodeIgniter, Smarty, cakePHP, Zend, Symfony, Yii, dan lain-lain
- Java: Spring, Hibernate, Struts, dan lain-lain
- Ruby: Ruby on Rails, dan lain-lain
- Python: Django, dan lain-lain
- .NET: ASP .NET MVC Framework, dan lain-lain

Beberapa *framework* untuk membangun aplikasi web berbasis Java telah banyak dibuat, misalnya Struts dari Apache. Struts menggunakan prinsip desain *Model-View Controller* (MVC). Berikut adalah arsitektur dari MVC:



Gambar 13 Arsitektur Struts

Strut sebenarnya merupakan implementasi dari MVC yang berupa pustaka-pustaka (*library*) standar Struts. Konsep MVC bertujuan agar sebuah aplikasi dapat mudah dipelihara oleh orang-orang di dalam tim pengembangan yang berbeda spesifikasi pekerjaan, misalnya *Database Administrator* (DBA) untuk mengurusi masalah basis data, blok *controller* untuk *programmer*, dan blok *view* untuk desainer antarmuka (*interface designer*). Struts dapat digunakan untuk membangun sebuah aplikasi web *enterprise* yang memiliki banyak proses kompleks. Komponen dasar dari Struts adalah:

- *framework* dasar
Menyediakan fungsi dasar MVC dan kumpulan blok dasar dari aplikasi. Pondasi dari *framework* dasar adalah servlet yang memegang kontrol (*controller servlet*) dan sisanya adalah kelas-kelas yang menangani fungsi-fungsi yang diperlukan.
- pustaka tag JSP (*Java Server Pages*)
Struts dibungkus dengan beberapa pustaka tag JSP sebagai berikut:
 - **HTML** digunakan untuk menggenerasi *form* yang berinteraksi dengan API Struts.

- **Bean** digunakan untuk bekerja dengan objek *Java Bean* misalnya mengakses nilai *bean*.
 - **Logic** digunakan untuk implementasi kondisi logik dalam JSP.
 - **Nested** digunakan untuk penanganan level *nested* dari tag HTML, *Bean*, dan logik.
- *tiles plugin*
Struts dibungkus dengan *tiles subframework*. *Tiles* adalah kumpulan *framework template JSP* yang memfasilitasi penggunaan kembali kode-kode yang telah ada.
 - *plugin validator*
Struts dibungkus dengan *subframework validator* yang menyediakan kumpulan *framework* untuk kebutuhan validasi data pada klien maupun server.

Struts dapat di-download pada <http://jakarta.apache.org/struts/>. Pada site tersebut juga terdapat keterangan lebih lanjut mengenai Struts dan penggunaanya dengan Apache Tomcat.



MVC merupakan salah satu contoh *architectural design pattern*. Saat ini banyak *framework* yang menggunakan prinsip desain MVC

Masih banyak lagi *design pattern* yang belum di bahas di buku ini karena jumlahnya sangat banyak. *Design pattern* sangat menarik untuk dipelajari jika akan mengembangkan perangkat lunak yang cukup kompleks atau mengalami perkembangan yang cukup kompleks di masa mendatang.

10.2 Anti Pattern

Anti pattern (pola kebalikan) merupakan pola yang dihasilkan dari penelusuran solusi-solusi yang buruk yang pernah digunakan untuk mencari solusi yang baik. *Anti pattern* lebih kepada mengambil pelajaran pada kegagalan-kegagalan yang pernah dialami sebelumnya dan bagaimana cara memperbaikinya. *Anti pattern* lebih bersifat pada perbaikan perangkat lunak berorientasi objek yang sudah ada atau proses pengembangan yang sudah berjalan untuk diperbaiki dengan proses-proses tertentu yang sesuai dengan permasalahan yang dialami. Langkah-langkah membuat *anti pattern* yang baik:

- Tidak menganggap separuh langkah buruk yang sudah ada sebagai sesuatu yang buruk tapi dipandang sebagai sebuah bagian langkah untuk menghasilkan pola yang positif untuk kedepannya.
- Mencoba mencari tahu apa yang salah dengan pola sebelumnya untuk menghasilkan pola yang baik (menangani kesalahan pola yang sudah ada).

Berikut adalah contoh-contoh kesalahan yang digunakan untuk memperbaiki pola pemrograman berorientasi objek:

1. *Spaghetti code* (kode spageti)

Banyak kode program yang tidak efisien dan diulang-ulang. Biasanya terjadi karena *programmer* membuat program dengan asal jalan tanpa memahami konsep pemrograman itu sendiri.

Penyebab terjadinya *spaghetti code*:

- *lone ranger programmer* (pembuat program solo, tanpa tim)
- dokumentasi kuno dan tidak lengkap
- melibatkan pembuat program yang sering ragu-ragu

Cara memperbaiki (*refactoring*):

memilah-milah fungsi yang digunakan ke dalam sub kelas atau membuat super kelas untuk memilah kode.

2. *Stovepipe system* (sistem pipa kompor)

Keterkaitan antar kelas tidak jelas, sehingga ada beberapa kelas yang sebenarnya secara implisit melakukan fungsi yang sama.

Penyebab *Stovepipe system*:

- Implementasi kode program tidak sesuai dengan dokumentasi
- Analisis yang tidak terbiasa dengan integrasi dan aspek-aspek kunci dari pembuatan perangkat lunak

Cara memperbaiki (*refactoring*):

Mengelompokkan fungsi-fungsi pada kelas menggunakan antarmuka dengan memperhatikan keterkaitan antar kelas.

3. *Analysis paralysis*

Memperlihatkan ketidakseimbangan upaya pada fase analisis dari sebuah proyek, hal ini terjadi karena proses analisis dilakukan oleh banyak orang yang kurang baik koordinasi timnya.

Penyebab *Analysis paralysis*:

- Koordinasi tim analis yang tidak baik
- Pemaksaan penyelesaian analisis sebelum proses desain
- Terpaku pada paradigma baru dan sudut pandang yang berbeda-beda

Cara memperbaiki (*refactoring*):

- Tidak menggunakan banyak analis
- Memperkerjakan arsitek perangkat lunak profesional
- Menyediakan prototipe untuk analisis

4. *Design by committee*

Desain kelas-kelas yang digunakan tidak detail sehingga menimbulkan asumsi sendiri pada permasalahan teknis detail dari anggota tim proyek. Hal ini terjadi biasanya karena desain aplikasi dilakukan pada pertemuan-pertemuan anggota tim dengan kepemimpinan pertemuan yang kurang baik.

Penyebab *Design by committee*:

- Dokumen terlalu rumit

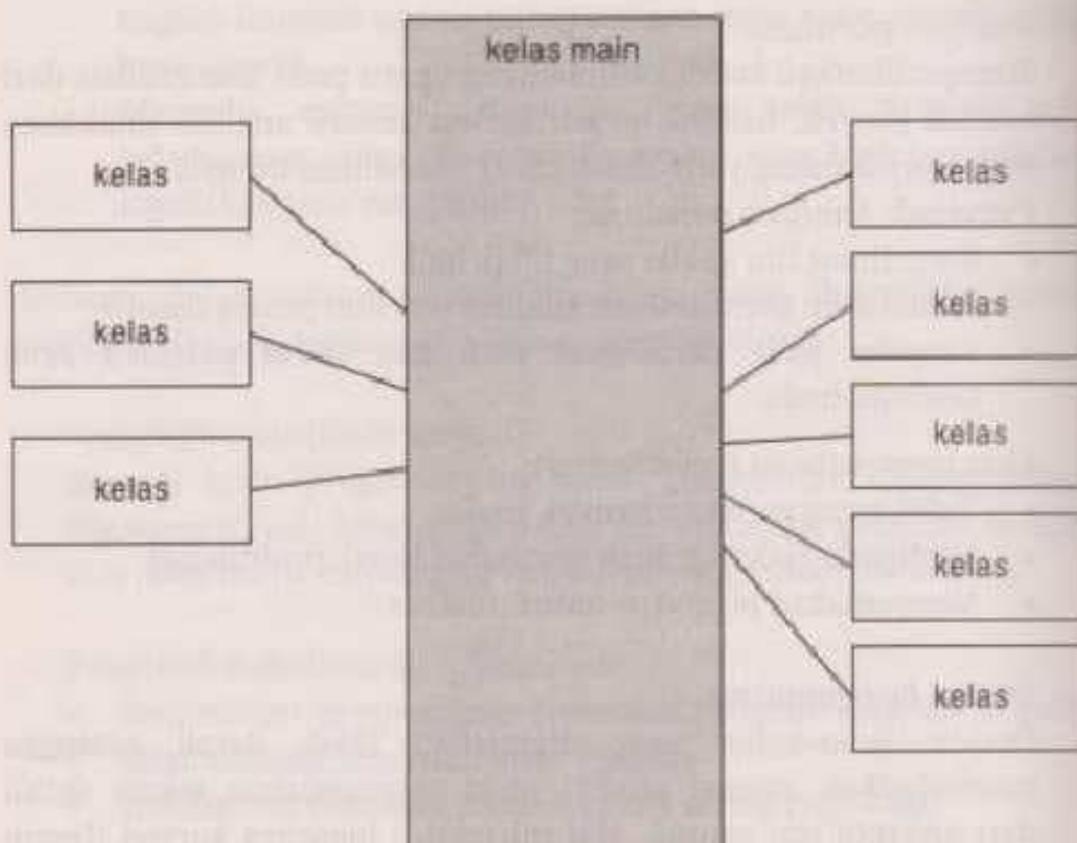
- Pertemuan yang tidak efektif
- Tidak mampu menentukan prioritas fitur
- Tidak ada arsitek perangkat lunak

Cara memperbaiki (*refactoring*):

- Mengefektifkan pertemuan dengan tim yang efektif
- Ada arsitek perangkat lunak

5. God class

Ada kelas yang memiliki peranan sangat dominan dalam perancangan kelas dalam sebuah aplikasi sehingga hampir semua kelas yang dibuat bergantung secara langsung maupun tidak langsung pada kelas ini. *God class* biasanya berisi kode program yang sangat panjang karena harus menangani banyak hal.



Gambar 14. Ilustrasi god class

Penyebab *God Class*:

- *Single* kelas dengan sejumlah besar atribut, operasi, atau keduanya. Sebuah kelas dengan 60 atau lebih atribut dan operasi mengindikasikan terjadinya *God Class* (kelas Tuhan)
- Tidak memperhatikan kohesi (*cohesion*) atau keterkaitan atribut dan operasi di dalam kelas.
- Berpikiran bahwa kelas yang memegang kontrol hanya satu
- Tidak ada desain berorientasi objek

Cara memperbaiki (*refactoring*):

Mengategorikan atau mengelompokkan atribut-atribut dan operasi-operasi di dalam kelas yang menjadi *God Class* menggunakan prinsip *cohesion* dan *coupling* untuk penyusunan kelas-kelas baru sebagai penguraian dari *God Class*.

6. *Mythical man month*

Desain atau struktur program yang digunakan hanya bergantung pada keputusan satu orang di dalam tim, hal ini akan menimbulkan ketidakdetailan struktur desain aplikasi apalagi jika orang yang dominan tidak memiliki kemampuan yang baik.

Penyebab *Mythical man month*:

- Orang yang memegang peranan penting bukan pembuat program (*programmer*) yang andal
- Menambah banyak orang di tengah proyek karena ketidakmampuan tim yang menjalankan proyek.

Cara memperbaiki (*refactoring*):

- Membuat tim atau sub tim dengan ukuran yang ideal dan memiliki orang berkemampuan baik sebagai pemegang peranan penting dalam tim
- Menyerahkan proyek kepada *developer* berbakat daripada membangun sendiri perangkat lunak dengan staf yang tidak berbakat

7. *Death march project*

Desain dihasilkan dari perkiraan proyek yang tidak realistik, misalnya waktu yang tidak realistik untuk mengembangkan aplikasi, spesifikasi kemampuan aplikasi yang tidak realistik,

sehingga dapat dikatakan desain seperti ini memang ditujukan untuk kegagalan.

Penyebab *Death march project*:

- Desain dilakukan oleh orang yang kurang memiliki kemampuan untuk mendesain perangkat lunak
- Kepala proyek tidak memiliki pengetahuan yang baik mengenai pembangunan perangkat lunak

Cara memperbaiki (*refactoring*):

Memberikan pekerjaan desain dan memimpin proyek kepada orang yang memiliki kemampuan dalam pembangunan perangkat lunak sehingga semua manajemen proyek dapat realistik

Semua kesalahan di atas dapat berakibat fatal yaitu sistem aplikasi tidak dipakai lagi karena akan diganti baru. Hal ini terjadi karena struktur aplikasi sangat sulit untuk dipelihara jika ada perubahan maupun tambahan pada spesifikasi sistem aplikasi.

Anti pattern mencoba menghubungkan apa yang terjadi di dunia nyata dalam pengembangan perangkat lunak dengan perancangan pola yang baik dengan tujuan dapat menyelesaikan permasalahan-permasalahan yang ada pada dunia pengembangan perangkat lunak. *Anti pattern* biasanya diselesaikan dengan perekayasaan kembali (*reengineering*) agar aplikasi dapat lebih mudah untuk dipelihara (*software maintenance*) salah satunya adalah proses *refactoring* dimana perancangan kode program diperbaiki dan disesuaikan dengan kaidah-kaidah pemrograman berorientasi objek yang benar.



Pengembangan perangkat lunak sebaiknya dilakukan dengan baik sejak awal proses agar tidak perlu melakukan *anti pattern*

11

Manajemen Proyek Perangkat Lunak



Overview

Bab ini membahas tentang manajemen proyek perangkat lunak. Pengembangan perangkat lunak biasanya dikerjakan dalam bentuk proyek. Ada hal-hal yang harus dikelola agar perangkat lunak yang dihasilkan memenuhi kualitas yang diharapkan dengan waktu, biaya, dan sumber daya sesuai dengan perencanaan awal.

Pada bab ini juga dibahas tentang hal-hal yang dilakukan dalam pengujian perangkat lunak agar menghasilkan perangkat lunak dengan kualitas sesuai yang diharapkan.

11.1 Pengertian Manajemen Proyek Perangkat Lunak

Proyek adalah urutan kegiatan yang unik, kompleks, dan saling terkait, memiliki satu tujuan, dan tujuan harus diselesaikan dalam waktu tertentu, sesuai anggaran, dan memenuhi spesifikasi. Manajemen/pengelolaan proyek perangkat lunak bertujuan agar perangkat lunak yang dibuat sampai ke tangan pelanggan (*customer*) tepat waktu dan sesuai dengan harapan pelanggan (*customer*).

Proyek perangkat lunak dapat didapatkan dari proses-proses berikut:

- Memiliki koneksi atau rekan yang sedang membutuhkan rekanan untuk mengerjakan proyek teknologi informasi (*Information Technology (IT)*)).
- Mencari proyek perangkat lunak dari internet. Beberapa situs luar negeri menyediakan iklan para pencari rekanan proyek perangkat lunak untuk mengerjakan perangkat lunak yang mereka butuhkan. Biasanya orang yang mendapat proyek dari internet paling tidak dapat berbahasa inggris dan menjelekkan reputasi karena jika tidak orang asing sering tidak bisa percaya lagi.
- Pihak swasta yang sedang membutuhkan rekanan untuk mengerjakan proyek teknologi informasi (*Information Technology (IT)*)).
- Penunjukan langsung dari instansi pemerintah yang sedang membutuhkan rekanan untuk mengerjakan proyek teknologi informasi (*Information Technology (IT)*)).
- Proses pelelangan oleh instansi pemerintah yang membutuhkan rekanan untuk mengerjakan proyek teknologi informasi (*Information Technology (IT)*)).

Macam-macam proyek IT yang dapat diperoleh adalah sebagai berikut:

- pengadaan perangkat keras (*hardware*)
- pengembangan perangkat lunak (*software*)
- pemeliharaan perangkat lunak (*software*)
- konsultasi IT (biasanya dilakukan oleh konsultan IT)

Salah satu proyek IT adalah pengembangan dan pemeliharaan perangkat lunak. Manajemen proyek yang akan dibahas pada buku ini lebih terfokus pada proyek perangkat lunak.

Manajemen proyek perangkat lunak sangat dibutuhkan karena permasalahan yang sering timbul pada proyek perangkat lunak tanpa pengelolaan yang baik adalah pembengkakan biaya proyek dan waktu pengerjaan tidak sesuai rencana (molor waktu). Padahal waktu dan biaya biasanya sudah dianggarkan oleh pelanggan (*customer*) dan *developer* perangkat lunak harus mampu mengelolanya agar target perangkat lunak yang akan dibuat dapat dicapai.

Aktifitas manajemen perangkat lunak adalah sebagai berikut:

- Manajemen aktivitas proyek

Aktifitas yang dilakukan pada proyek perangkat lunak adalah sebagai berikut:

- menulis proposal
- perencanaan dan penjadwalan
- pembiayaan proyek
- pengawasan dan peninjauan proyek
- seleksi dan evaluasi anggota proyek
- penulisan laporan dan presentasi

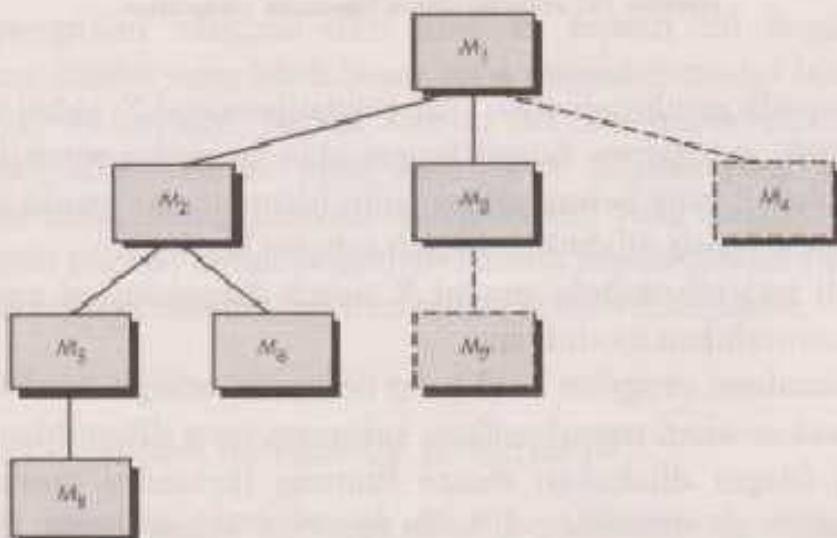
- Perencanaan Proyek

Perencanaan proyek biasanya terdiri dari:

- organisasi proyek
- analisis risiko
- kebutuhan perangkat keras dan perangkat lunak

11.3.2.1 Pengujian Top-Down Integration

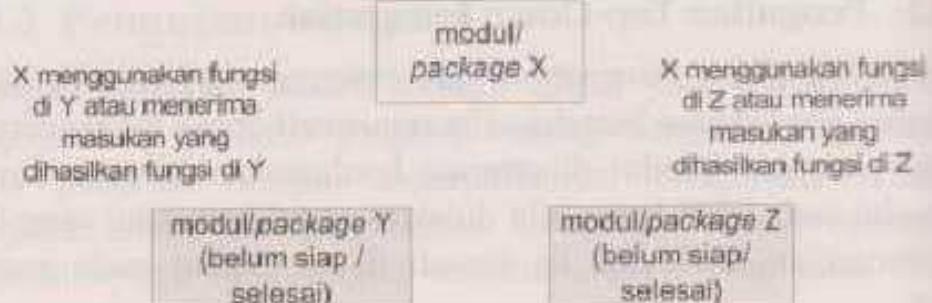
Pengujian *top-down integration* atau integrasi dari atas ke bawah merupakan pendekatan bertahap (*incremental*) untuk mengonstruksi struktur program. Modul diintegrasikan berdasarkan hierarki, dimulai dari modul yang lebih besar lalu didekomposisi ke modul yang lebih kecil. Pengujian dari atas ke bawah dapat dilihat pada gambar berikut:



Gambar 106 Ilustrasi hierarki *top-down Integration*

Di mana urutan pengujian setiap modul sama dengan ilustrasi gambar di atas.

Pengujian dari atas ke bawah biasanya sering digunakan untuk menguji prototipe dimana sebuah prototipe biasanya pada awalnya dibuat secara sistem tapi berupa prototipe (alur kerja sudah dapat dilihat tapi sebenarnya fungsi-fungsi yang ada belum berjalan benar, fungsinya hanya untuk menunjukkan ke *user* atau pelanggan seperti apa sistem yang akan dihasilkan). Berikut adalah ilustrasi siklus dari pengujian dari atas ke bawah:



Gambar 107 Ilustrasi siklus *top-down integration*

Misalkan pada gambar di atas, akan menguji modul X, yakni modul X perlu untuk mengakses fungsi-fungsi atau prosedur-prosedur pada modul Y dan Z yang belum selesai atau belum siap digunakan, maka langkah yang harus dilakukan adalah sebagai berikut:

1. uji terlebih dahulu modul X untuk fungsi-fungsi yang tidak memerlukan modul lain.
2. membuat program kecil yang berperan sebagai modul Y yang seakan-akan menghasilkan keluaran yang dibutuhkan modul X (dapat dilakukan secara *dummy* (keluaran random atau *hardcode* dituliskan di kode program keluarannya, jadi tidak harus menggunakan modul Y yang sudah jadi)), misalkan jika membutuhkan keluaran berupa angka nilai mata uang:

```
//modul YDummy
function getSaldo(){
    return 599999;
}
```

maka modul X dapat menggunakan modul Y sebagai berikut:

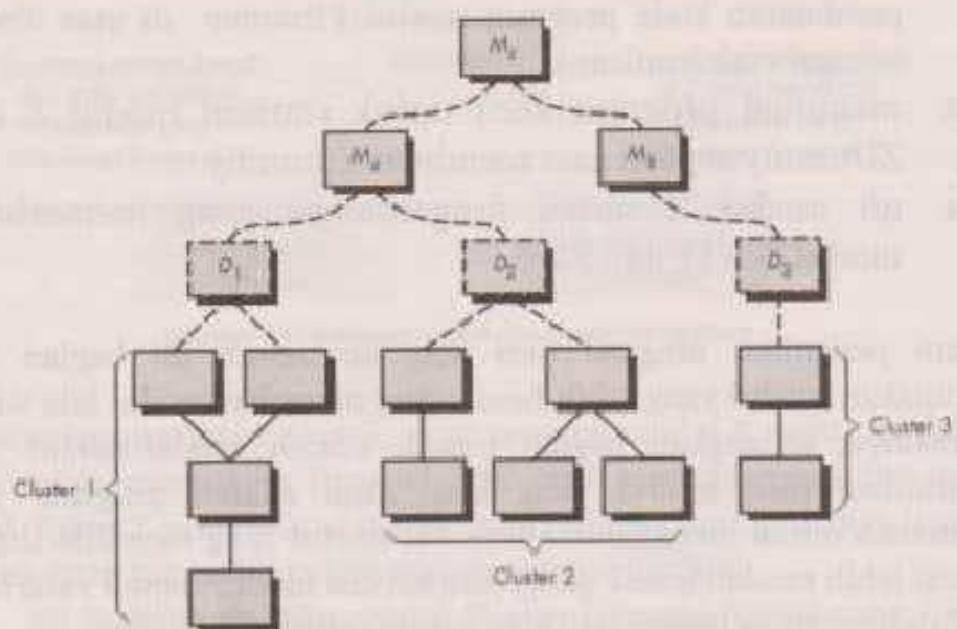
```
//modul X
#include "YDummy"
main()
{
    *****
    print(getSaldo());
    *****
}
```

- pembuatan kode program modul YDummy di atas disebut sebagai *stub* (rintisan).
3. membuat program kecil untuk rintisan modul Z atau ZDummy seperti saat membuat Ydummy.
 4. uji modul X untuk fungsi-fungsi yang memerlukan modul lain (Y dan Z).

Dalam pengujian integrasi dari atas ke bawah ini bagian atas merupakan modul yang lebih besar yang memakai modul lain dalam operasinya, sedangkan bagian bawah adalah modul-modul yang dibutuhkan oleh modul yang lain. *Stub* adalah program yang digunakan untuk menggantikan modul-modul yang memiliki tingkat hirarki lebih rendah untuk pengujian karena modul-modul yang lebih rendah hirarkinya belum siap atau belum selesai dikerjakan.

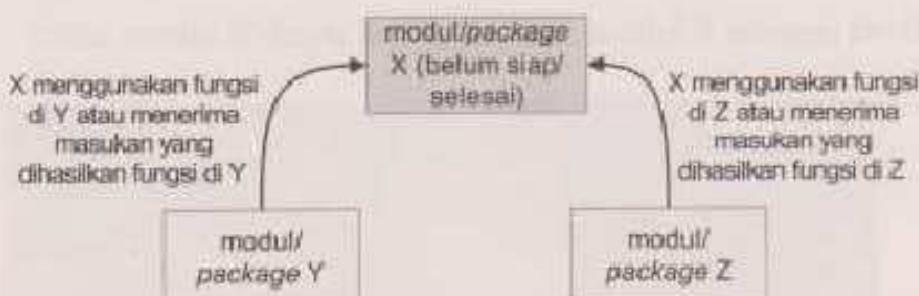
11.3.2.2 Pengujian Bottom-Up Integration

Pengujian integrasi dari bawah ke atas (*bottom-up integration*) memulai pengujian dari modul yang paling kecil ke modul yang lebih besar. Pengujian dari bawah ke atas sering dilakukan untuk pengembangan perangkat lunak yang tidak menggunakan alur prototipe sehingga perangkat lunak dibangun dari modul-modul yang kecil ke modul-modul yang besar sesuai dengan hirarki pemakaiannya. Ilustrasi pengujian dari bawah ke atas adalah sebagai berikut:



Gambar 108 Ilustrasi hirarki *bottom-up integration*

Beberapa modul yang memiliki hirarki paling rendah dibuat menjadi beberapa *cluster* (kelompok) berdasarkan keterkaitan fungsinya. Dari setiap *cluster* dibuat sebuah program untuk menguji setiap *cluster* dimana setiap program yang digunakan untuk menguji ini harus mampu menguji setiap fungsi yang ada di dalam *cluster*. Program yang digunakan untuk menguji fungsionalitas *cluster* disebut *driver*. Setelah setiap *cluster* selesai diuji dengan menggunakan program *driver* maka dilanjutkan pengujian dengan modul yang lebih tinggi hirarkinya. Ilustrasi pengujian tiap *cluster* adalah sebagai berikut:



Gambar 109 Ilustrasi pengujian tiap *cluster*

Misalkan pada gambar di atas, akan menguji modul Y dan Z, dimana modul Y dan Z diakses oleh modul X yang belum selesai atau belum siap digunakan, maka langkah yang harus dilakukan adalah sebagai berikut:

1. uji terlebih dahulu modul Y dan Z.
2. membuat program yang berperan sebagai modul X yang seakan-akan adalah modul X yang mengakses modul Y dan Z, misalkan sebagai berikut:

```
//modul Y  
function getSaldo()  
{  
    .....  
}
```

```
//modul Z  
function getMasabah()  
{  
    .....  
}
```

maka modul XDummy dapat menggunakan modul Y dan modul Z sebagai berikut:

```
//modul XDummy  
include Y;  
include Z;  
  
main()  
{  
    .....  
    print(getSaldo());  
    .....  
    getMasabah();  
    .....  
}
```

Pembuatan kode program modul XDummy di atas disebut sebagai *driver* (pemegang kontrol). *Driver* adalah program yang digunakan untuk mengantikan modul yang memiliki tingkat hirarki lebih tinggi

untuk pengujian karena modul-modul yang lebih tinggi hirarkinya belum siap atau belum selesai dikerjakan.

11.3.2.3 Pengujian Regression Integration

Setiap modul baru ditambahkan sebagai bagian dari integrasi maka kondisi perangkat lunak menjadi berubah. Hal ini mungkin saja menyebabkan masalah pada fungsionalitas yang sudah diuji sebelumnya. Pengujian regresi (*regression*) adalah eksekusi dari beberapa subset pengujian yang sudah terhubung atau saling terkait untuk menjamin bahwa modul yang baru masuk pengujian tidak mengubah fungsionalitas yang sudah diuji sebelumnya. Pengujian regresi dapat dilakukan secara manual dengan mengeksekusi sebuah subset untuk semua kasus uji dari subset itu atau bisa juga menggunakan perangkat (*tools*).

Pengujian regresi lebih sesuai menggunakan tiga kelompok kasus pengujian sebagai berikut:

- ▲ kelas kasus uji yang berisi contoh kasus pengujian yang dapat menguji semua fungsi perangkat lunak
- ▲ kelas kasus uji yang berisi kasus tambahan yang fokus pada fungsi perangkat lunak yang akan terpengaruh jika ada tambahan modul baru untuk diuji
- ▲ kelas kasus uji yang berisi kasus yang fokus pada komponen atau modul baru atau yang mengalami perubahan

Pengujian regresi sesui digunakan untuk pengujian perangkat lunak yang memiliki struktur yang baik, karena jika digunakan pada perangkat lunak dengan struktur yang tidak baik maka bisa jadi lingkup kasus pengujian menjadi besar dan melebar. Pengujian regresi lebih fokus pada pengujian area dari komponen yang mengalami tambahan, perubahan, atau perbaikan.

11.3.2.4 Pengujian Smoke Integration

Pengujian "asap" (*smoke testing*) adalah sebuah pendekatan pengujian integrasi yang biasa digunakan ketika waktu penggerjaan perangkat lunak cukup singkat dan biasanya untuk komponen atau modul yang ditambahkan pada perangkat lunak. Pengujian asap awalnya berasal dari industri perangkat keras (*hardware*) dimana setelah sebuah komponen perangkat keras diubah atau diperbaiki maka perangkat keras akan dinyalakan, jika tidak ada asap yang keluar maka komponen itu berhasil lolos uji. Pengujian "asap" meliputi aktivitas-aktivitas berikut:

- ▲ mempersiapkan komponen atau modul perangkat lunak yang sudah ditranslasi menjadi kode program diintegrasikan dengan komponen lain yang terkait seperti berkas (*file*) data, pustaka (*libraries*), modul lain yang digunakan kembali (*reusable*), dan komponen rekayasa lainnya yang diperlukan untuk implementasi satu atau lebih fungsi perangkat lunak (merupakan komponen yang diubah atau diperbaiki)
- ▲ mempersiapkan sekumpulan pengujian yang didesain untuk menemukan kesalahan (*error*) yang menjaga perangkat lunak tetap memenuhi fungsinya
- ▲ mengintegrasikan kumpulan kode program, berkas (*file*) data, pustaka (*libraries*), modul lain yang digunakan kembali (*reusable*), dan komponen rekayasa lainnya yang diperlukan dengan kumpulan yang lain dan diuji per hari agar setiap pertambahan komponen per hari dapat teruji, pendekatan yang dilakukan bisa menggunakan *top-down* atau *bottom-up*.

Kelebihan pengujian "asap" jika dilakukan pada perangkat lunak yang dikembangkan dengan waktu pendek adalah sebagai berikut:

- ▲ risiko integrasi dapat diminimalisasi karena pengujian dilakukan per hari sehingga ketidaksesuaian antar komponen dapat cepat terdeteksi
- ▲ kualitas produk dapat diperbaiki karena fokus pada pengujian fungsionalitas produk

REKAYASA PERANGKAT LUNAK

TERSTRUKTUR dan BERORIENTASI OBJEK

Rosa A. S
M. Shalahuddin

Pada buku ini akan membahas tentang analisis dan desain sistem dengan disertai suatu studi kasus untuk memudahkan dalam pemahaman. Analisis dan desain sistem itu dimulai dengan analisis dan desain basis data, analisis dan desain sistem untuk pemrograman terstruktur dengan menggunakan DFD, dan analisis dan desain sistem untuk pemrograman berorientasi objek dengan menggunakan UML.

Setelah membaca studi kasus tersebut, pembaca diharapkan dapat memahami bagaimana melakukan analisis dan desain sistem untuk pemrograman terstruktur maupun pemrograman berorientasi objek. Untuk menghasilkan analisis dan desain perangkat lunak yang baik, seorang analis seharusnya memahami konsep pemrograman. Tanpa pemahaman konsep pemrograman yang baik, analis tidak mungkin menghasilkan perancangan perangkat lunak yang baik.

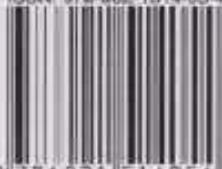
Pada buku ini juga dijelaskan mengenai tahapan-tahapan yang harus dilalui dalam rakayasa perangkat lunak serta penjelasan secara umum tentang manajemen proyek perangkat lunak.



Penerbit **INFORMATIKA**

Pemasaran: BI-OBSES
Pasar buku Palasari 82 Bandung 40264
Tel.(022) 7317812 Fax.(022) 7317896
www.biobses.com

ISBN 978-602-1514-05-4



9786021514054