

**UNIVERSITAS ESA UNGGUL
FAKULTAS ILMU KESEHATAN
PROGRAM STUDI REKAM MEDIS**

Universitas
Esa Unggul

Universitas

Universitas
Esa Unggul



Universitas
Esa Unggul

**MODUL PRAKTIKUM
ANALISIS DAN PERANCANGAN SISTEM**

Universitas
Esa Unggul

Universitas
Esa Unggul

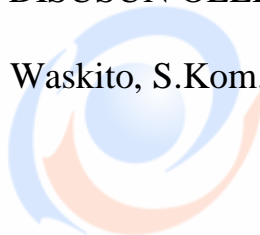


Universitas
Esa Unggul

Universitas
Esa Unggul

Universitas
Esa Unggul

DISUSUN OLEH



Andri Waskito, S.Kom., M.Kom

Universitas
Esa Unggul

Universitas
Esa Unggul

Universitas
Esa Unggul



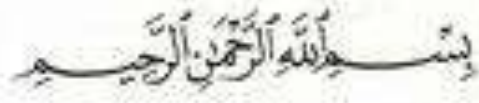
UNIVERSITAS ESA UNGGUL
2018

Universitas
Esa Unggul

Universitas
Esa Unggul

Universitas
Esa Unggul

KATA PENGANTAR



Assalaamu'alaikum Wr. Wb.

Segala puji bagi Allah SWT yang selalu memberikan rahmat dan hidayahnya serta nikmat yang tak terhingga, sehingga kami dapat menyelesaikan modul praktikum analisis dan perancangan sistem ini. Dalam penyusunan modul praktikum ini, kami sadar bahwa tanpa bantuan dan bimbingan berbagai pihak maka modul ini sulit untuk terwujud. Untuk itu dalam kesempatan ini kami ingin menyampaikan ucapan terima kasih yang sebesar-besarnya kepada semua pihak yang telah membantu.

Kami menyadari sepenuhnya bahwa dalam penyusunan modul praktikum ini masih banyak kekurangan, untuk itu dengan segala kerendahan hati kami mengharapkan saran dan kritik yang sifatnya membangun guna memperbaiki modul ini.

Akhir kata semoga modul praktikum ini dapat menambah ilmu pengetahuan dan bermanfaat bagi semua yang membaca dan memcobanya..

Wassalamu'alaikum Wr. Wb

Jakarta, Maret 2018

Andri Waskito, S.Kom., M.Kom

DAFTAR ISI

KATA PENGANTAR	ii
DAFTAR ISI.....	iii
DAFTAR TABEL.....	iv
DAFTAR GAMBAR	v
MODUL 1	1
MODUL 2	5
MODUL 3	9
MODUL 4	23
DAFTAR PUSTAKA.....	24





DAFTAR TABEL

Tabel 1. Bagan Notasi DFD	3
Tabel 2. Jenis hubungan dalam ERD	6
Tabel 3. Simbol <i>Data Dictionary</i>	7.
Tabel 4. Abstraksi konsep dasar UML	10



Universitas
Esa Unggul



Universitas
Esa Unggul



Universitas
Esa Unggul



Universitas
Esa Unggul



Universitas
Esa Unggul



Universitas
Esa Unggul



Universitas
Esa Unggul



Universitas
Esa Unggul



Universitas
Esa Unggul



Universitas
Esa Unggul



Universitas
Esa Unggul






Universitas
Esa Unggul



DAFTAR GAMBAR

Gambar 1. Contoh <i>use case diagram</i>	12
Gambar 2. Contoh <i>class diagram</i>	14
Gambar 3. Contoh <i>statechart diagram</i>	15
Gambar 4. Contoh <i>activity diagram</i> tanpa <i>swimlane</i>	16
Gambar 5. Contoh <i>sequence diagram</i>	17
Gambar 6. <i>Collaboration diagram</i>	18
Gambar 7. Contoh <i>component diagram</i>	19
Gambar 8. Contoh <i>deployment diagram</i>	20



Modul 1 :

a. Judul :

Perancangan Sistem Sederhana (Kasus Sistem Informasi Apotek) dengan tahapan Diagram Konteks dan Data Flow Diagram.

b. Estimasi Waktu (opsional) :

200 Menit (2 x Tatap Muka Prak.).

c. Tujuan Instruksional Khusus :

Mahasiswa mampu untuk melakukan analisis dan perancangan sistem melalui perangkat lunak.

d. Dasar Teori

Analisis

Sistem analis adalah orang yang menganalisis sistem dengan mempelajari masalah-masalah yang timbul dan menentukan kebutuhan-kebutuhan pemakai serta mengidentifikasi pemecahan yang beralasan (lebih memahami aspek-aspek bisnis dan teknologi komputer).

Dalam tahap analisis ini, digunakan oleh sistem analis untuk :

- a. Membuat keputusan apabila sistem saat ini mempunyai masalah atau sudah tidak berfungsi secara baik dan hasil analisisnya digunakan sebagai dasar untuk memperbaiki sistem.
- b. Mengetahui ruang lingkup pekerjaannya yang akan ditanganinya.
- c. Memahami sistem yang sedang berjalan saat ini.
- d. Mengidentifikasi masalah dan mencari solusinya.

Desain

Dalam tahap perancangan/Desain memiliki tujuan, yaitu:

Mendesain sistem baru yang dapat menyelesaikan masalah-masalah yang dihadapi perusahaan yang diperoleh dari pemilihan alternatif sistem yang terbaik. DFD (Data Flow Diagram)

Data flow diagram adalah alat pembuatan model yang memungkinkan profesional sistem untuk menggambarkan sistem sebagai suatu jaringan proses fungsional yang dihubungkan satu sama lain dengan alur data baik secara manual maupun komputerisasi. DFD ini sering disebut juga dengan nama *Bubble Chart* atau diagram, model proses, diagram alur kerja atau model fungsi. DFD ini adalah salah satu alat pembuatan model yang sering digunakan, khususnya jika fungsi-fungsi sistem merupakan bagian yang lebih penting dan kompleks daripada data yang digunakan untuk menjelaskan aliran informasi dan transformasi data yang bergerak dari pemasukan data hingga keluaran (Rachmat, 2004, hal. 25).

Ada empat komponen DFD, yaitu sebagai berikut: (Pohan, Iskandar, 1996, hal 16)




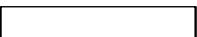
1. **Process;** Menunjukkan transformasi dan masukan menjadi keluaran. dalam hal ini sejumlah masukan dapat menjadi hanya satu keluaran ataupun sebaliknya. Proses umumnya didefinisikan dengan kata tunggal atau kalimat sederhana dan lebih sering mengidentifikasi subjek proses daripada objek proses itu sendiri. Proses direpresentasikan dalam bentuk lingkaran atau bujur sangkar dengan sudut melengkung.
2. **Data Flow Lines;** Direpresentasikan dengan menggunakan panah. Aliran digunakan untuk menggambarkan gerakan paket data atau informasi dan satu bagian ke bagian lain dan sistem dimana penyimpanan mewakili lokasi penyimpanan data. Ujung panah menunjukkan ke mana data bergerak ke/dari proses, penyimpanan ataupun terminator atau keduanya. Aliran yang digambarkan sebagai panah dengan dua ujung menggambarkan terjadinya dialog. Aliran dapat juga menyebar atau menyatu, misalnya sejumlah atribut dapat membentuk satu aliran. atau aliran menyebar menjadi sejumlah atribut. Atribut dalam hal ini dapat merupakan bagian atau duplikasi aliran.
3. **External Entity;** Komponen ini direpresentasikan dengan menggunakan persegi panjang yang mewakili entitas luar dimana sistem berkomunikasi. Biasanya komponen

ini melambangkan orang atau organisasi diluar sistem dan berada diluar kendali dari sistem yang dimodelkan.

4. Penyimpanan (*Data Store*); Komponen ini digunakan untuk memodelkan kumpulan data atau paket data. Notasi yang digunakan adalah garis sejajar, segi empat, melengkung atau persegi panjang. Notasi ini dapat juga didefinisikan *file* atau *database* atau mendefinisikan bagaimana penyimpanan diimplementasikan dalam sistem komputer. Panah yang bergerak ke penyimpanan mendeskripsikan penulisan, perubahan atau penghapusan. Satu atau lebih paket dimasukkan ke penyimpanan sebagai bagian dan paket lama atau paket baru, ataupun satu atau lebih paket dihapus/dipindahkan dan penyimpanan.

Data flow diagram digunakan untuk bermacam-macam tujuan analisis sistem, termasuk menggambarkan alur logical dari data melalui proses. *Data flow diagrams* terdiri dari 4 simbol seperti pada gambar yaitu :

Tabel 1. Bagan Notasi DFD

No	Gambar	Simbol	Keterangan
1		Intity eksternal	Prosedure atau kosumer informasi yang ada diluar sistem untuk dimodelkan
2		Proses	Transfer informasi (fungsi) yang ada didalam sistem untuk dimodelkan
3		Objek data	Anak panah menunjukan arah aliran data
4		Penyimpanan data	Responentasi data yang disimpan untuk digunakan oleh satu atau lebih

Tingkatan DFD

Untuk memudahkan pembacaan DFD, maka penggambaran DFD disusun berdasarkan tingkatan atau level dari atas ke bawah, yaitu yaitu (Rachmat, 2004, hal. 26) :

- Diagram Konteks (Level 0)

Merupakan diagram paling atas yang terdiri dari suatu proses dan menggambarkan ruang lingkup proses. Hal yang digambarkan dalam diagram konteks adalah hubungan terminator dengan sistem dan juga sistem dalam suatu proses. Sedangkan hal yang tidak digambarkan dalam diagram konteks adalah hubungan antar terminator dan *data store*.

- Diagram *Zero* (Level 1)

Merupakan diagram yang berada di antara Diagram Konteks dan Diagram Detail serta menggambarkan proses utama dari DFD. Hal yang digambarkan dalam Diagram *Zero* adalah proses utama dari sistem serta hubungan entitas, Proses, alur data dan *data store*.

- Diagram Detail (Rinci)

Merupakan penguraian dalam proses yang ada dalam Diagram *Zero*. Diagram yang paling rendah dan tidak dapat diuraikan lagi. Jika tidak dapat diturunkan lagi maka dikatakan diagram primitif.

Konsep *Balancing*

Balancing dalam DFD adalah keseimbangan antar *level* pada DFD. Jika keseimbangan antar *level* tercapai berarti sudah didapatkan *level balance*. Persyaratan untuk mencapai *level balance* adalah: “aliran data yang masuk ke dalam dan ke luar dari suatu proses harus sama dengan aliran data yang masuk ke dalam dan ke luar dari rincian proses tersebut.” (Pressman, 2002).

e. Tugas Pendahuluan

Mengenal fungsi icon-icon dan menu yang ada di perangkat lunak, untuk hal ini perangkat lunak / alat bantu yang digunakan adalah easycase.

f. Praktikum :

Langkah yang dapat membantu dalam menggambarkan CD dan DFD :

1. Identifikasikan seluruh informasi yang dibutuhkan.
2. Identifikasikan seluruh data yang dibutuhkan proses/informasi.
3. Identifikasikan seluruh tujuan setiap informasi bagi penggunanya.
4. Identifikasikan seluruh sumber data yang dibutuhkan proses/informasi

g. Tugas Pasca Praktikum

Membuat resume atas pekerjaan.

Modul 2 :

a. Judul :

Perancangan Basis Data

b. Estimasi Waktu (opsional) :

200 Menit (2 x Tatap Muka Prak.).

c. Tujuan Instruksional Khusus

1. Membuat model sistem yang akan mereka kembangkan dengan ERD
2. Membuat perancangan database.

d. Dasar Teori

Entity Relationship Diagram (ERD)

Entity Relationship Diagram adalah suatu cara untuk merepresentasikan struktur dari suatu relational *database*. *Entity* merepresentasikan suatu objek yang nyata dan biasanya berupa kata benda, seperti pegawai, lagu, dll. *Relationship* menggambarkan bagaimana hubungan antara 2 atau lebih entitas dan biasanya berupa kata kerja, seperti “memiliki” sebagai relasi antara perusahaan dan komputer.

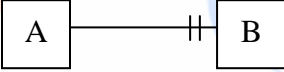
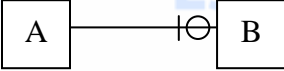
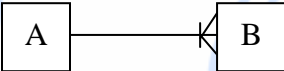
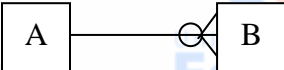
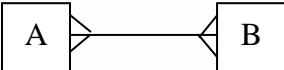
Entitas dan *relationship* dapat memiliki atribut yang menggambarkan sifat yang dimiliki oleh objek tersebut. Setiap entitas harus memiliki identitas atribut yang unik yang disebut dengan *primary key*. *Entity relationship diagram* tidak hanya menampilkan entitas tunggal, tetapi ditampilkan satu set entitas dan set *relationship*. Garis dibuat untuk menghubungkan set *entitas* dan *relationship* yang ada (http://en2.wikipedia.org/wiki/Entity-Relationship_Diagram).

ERD menggunakan sejumlah notasi dan simbol untuk menggambarkan struktur dan hubungan antar data. Terdapat 3 (tiga) simbol dasar yang digunakan, yaitu: (Pohan, Iskandar, 1997, hal 35)

- *Entitas* adalah suatu obyek yang dapat didefinisikan dalam lingkungan pemakai, sesuatu yang penting bagi pemakai dalam konteks sistem yang akan dibuat. Contohnya: Pelanggan, Pekerja, dll.

- **Atribut** merupakan elemen dan entitas yang berfungsi untuk mendeskripsikan karakter entitas. Sebagai contoh adalah atribut nama pekerja dan entiti pekerja
- **Relationship** adalah hubungan yang menunjukkan relasi antar entitas. Ada beberapa jenis relationship seperti yang tercantum pada tabel dibawah berikut ini:

Tabel 2. Jenis hubungan dalam ERD

Relasi <i>Minimum</i>	Relasi Maksimum	Notasi Grafik
1	1	
0	>1 (banyak)	
1	>1 (banyak)	
0	>1 (banyak)	
>1 (banyak)	>1 (banyak)	

Event List

Event List adalah daftar narasi *stimuli* (daftar kejadian) yang terjadi dalam lingkungan dan mempunyai hubungan dengan respon yang diberikan oleh sistem. (Pohan, Iskandar, 1997, hal 14)

Kamus Data (Data Dictionary)

Data Dictionary (kamus data yang selanjutnya kita sebut sebagai DD) tidak menggunakan notasi grafik seperti halnya DFD. Mirip dengan kamus yang membantu

kita dalam mencari arti kata baru, maka DD juga mempunyai fungsi yang sama dalam pemodelan sistem.

Selain itu DD berfungsi membantu pelaku sistem untuk mengerti aplikasi secara detil, dan mengorganisasi semua elemen data yang digunakan dalam sistem secara presisi sehingga pemakai dan penganalisa sistem mempunyai dasar pengertian yang sama tentang masukan, keluaran, penyimpanan dan proses (Pohan, Bahri, 1997, hal. 21). DD mendefinisikan elemen data dengan fungsi sebagai berikut :

- ◆ Menjelaskan arti aliran data dan penyimpanan dalam DFD
- ◆ Mendeskripsikan komposisi paket data yang bergerak melalui aliran, misalnya alamat diuraikan menjadi kota, negara, dan kode pos.
- ◆ Mendeskripsikan komposisi penyimpanan data
- ◆ Mendeskripsikan nilai dan satuan yang relevan bagi penyimpanan dan aliran
- ◆ Mendeskripsikan hubungan detil antar penyimpanan yang akan menjadi titik perhatian dalam ERD

Pada kebanyakan sistem dalam dunia nyata dimana kita bekerja, kadang-kadang elemen data terlalu kompleks untuk didefinisikan. Kekompleksan tersebut seharusnya diuraikan melalui sejumlah elemen data yang lebih sederhana. Kemudian elemen data yang lebih sederhana tersebut didefinisikan kembali hingga nilai dan satuan relevan dan elementer. Pendefinsian tersebut menggunakan notasi yang umum digunakan dalam menganalisa sistem dengan menggunakan sejumlah simbol, yaitu (Pohan, Bahri, 1997, hal. 22):

Tabel 3. Simbol *Data Dictionary*

No	Simbol	Uraian
1	=	terdiri dari, mendefinisikan, diuraikan menjadi, artinya
2	+	dan
3	()	opsional (boleh ada atau tidak)
4	{ }	pengulangan
5	[]	memilih salah satu dari sejumlah alternatif, seleksi
6	**	komentar
7	@	identifikasi atribut kunci
8		pemisah sejumlah alternatif pilihan antara simbol []

Sebagai contoh, kita akan mendefinisikan nama dengan menggunakan aturan di atas. Nama dalam hal ini mempunyai sejumlah atribut pendukung seperti gelar, nama_pertama, nama_tengah, dan nama_akhir.

nama = gelar+nama_pertama+nama_tengah+nama_akhir

gelar = [Tuan | Nyonya | Nona | Doktor | Profesor]

nama_pertama = karakter_valid

nama_tengah = karakter_valid

nama_akhir = karakter_valid

karakter_valid = [A-Z | a-z | 0-9 | ' | - | |]

e. Tugas Pendahuluan

Memahami analisis suatu kasus dan desainnya berupa DFD.

f. Praktikum :

Membuat ERD dari hasil analisis dan desain yang telah dilakukan.

g. Tugas Pasca Praktikum .

- Membuat resume atas pekerjaan.
- Memperkenalkan bentuk model dalam rangka pengkodean.

Modul 3 :

a. Judul

Perancangan Sistem Berorientasi Objek Dengan Uml

b. Estimasi Waktu (opsional)

200 Menit (2 x Tatap Muka Prak.).

c. Tujuan Instruksional Khusus

- Mengetahui tujuan penggunaan UML.
- Mengetahui sejarah singkat UML.
- Mengenal bagian-bagian (diagram-diagram) UML.
- Menggunakan UML untuk membuat model sederhana

d. Dasar Teori

Konsepsi Dasar UML

Tabel 4. Abstraksi konsep dasar UML

<i>Major Area</i>	<i>View</i>	<i>Diagrams</i>	<i>Main Concepts</i>
structural	static view	class diagram	class, association, generalization, dependency, realization, interface
	use case view	use case diagram	use case, actor, association, extend, include, use case generalization
	implementation view	component diagram	component, interface, dependency, realization
	deployment view	deployment diagram	node, component, dependency, location
dynamic	state machine view	statechart diagram	state, event, transition, action
	activity view	activity diagram	state, activity, completion transition, fork, join
	interaction view	sequence diagram	interaction, object, message, activation
collaboration diagram		collaboration, interaction, collaboration role, message	
model management	model management view	class diagram	package, subsystem, model
extensibility	all	all	constraint, stereotype, tagged values

Abstraksi konsep dasar UML yang terdiri dari *structural classification*, *dynamic behavior*, dan *model management*, bisa dipahami dengan mudah apabila melihat gambar diatas dari *Diagrams*. *Main concepts* bisa dipandang sebagai term yang akan muncul pada saat membuat diagram. Dan view adalah kategori dari diagram tersebut. Untuk menguasai UML, sebenarnya cukup dua hal yang harus kita perhatikan:

1. Menguasai pembuatan diagram UML
2. Menguasai langkah-langkah dalam analisa dan pengembangan dengan UML

Seperti juga tercantum pada gambar diatas UML mendefinisikan diagram-diagram sebagai berikut:

- *use case diagram*
- *class diagram*
- *statechart diagram*
- *activity diagram*
- *sequence diagram*
- *collaboration diagram*
- *component diagram*
- *deployment diagram*

Use Case Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya.

Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

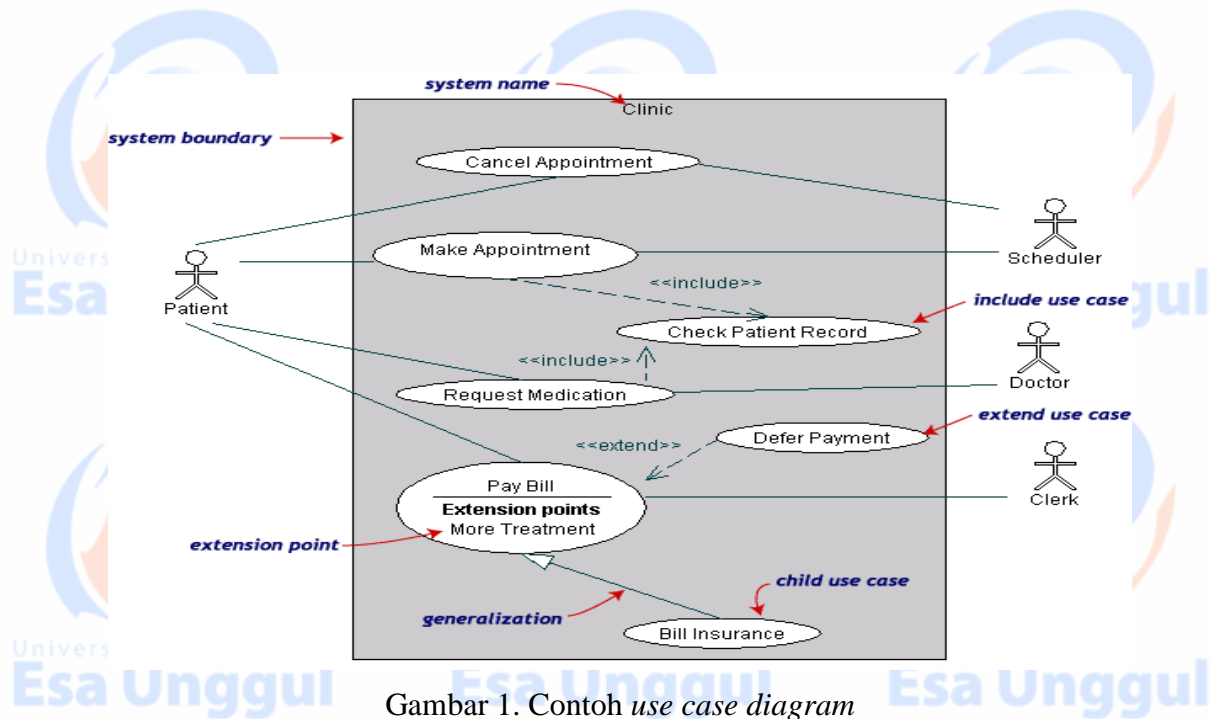
Use case diagram dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.

Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal.

Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*.

Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri.

Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.



Gambar 1. Contoh use case diagram

Class Diagram

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).

Class diagram menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

Class memiliki tiga area pokok :

1. Nama (dan stereotype)
2. Atribut
3. Metoda

Atribut dan metoda dapat memiliki salah satu sifat berikut :

- *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan
- *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya
- *Public*, dapat dipanggil oleh siapa saja

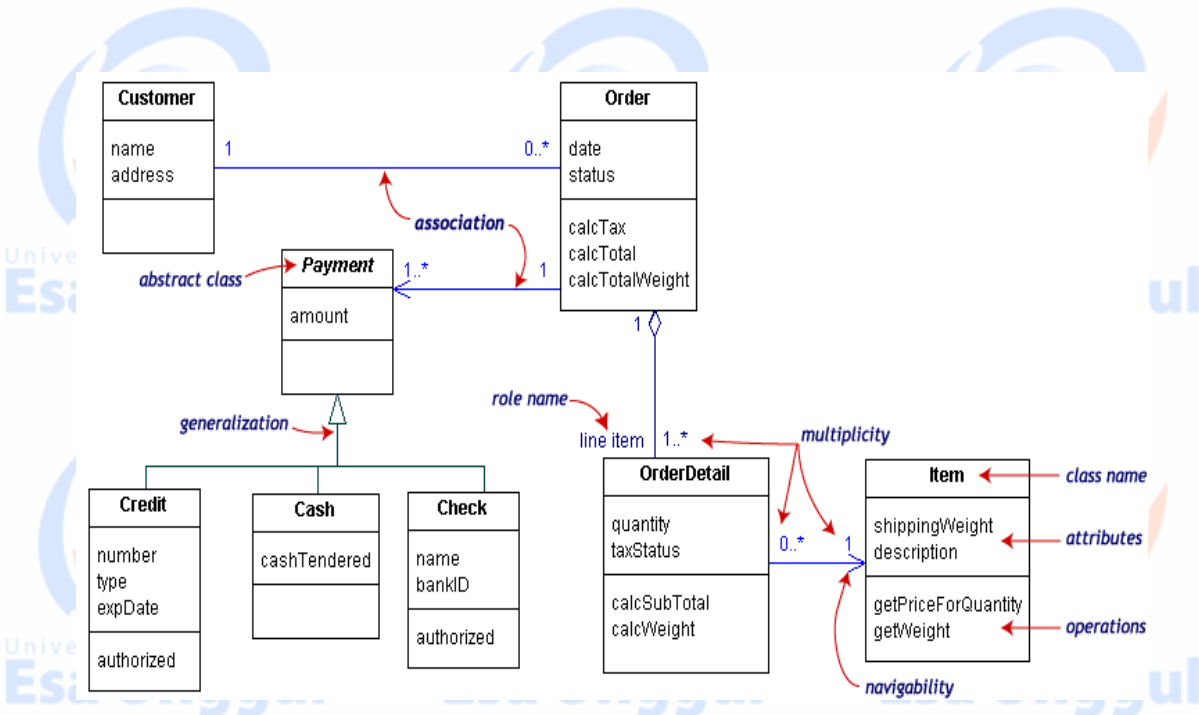
Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class*. Dengan demikian *interface* mendukung resolusi metoda pada saat *run-time*.

Sesuai dengan perkembangan *class* model, *class* dapat dikelompokkan menjadi *package*. juga dapat membuat diagram yang terdiri atas *package*.

Hubungan Antar Class

- Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah *query* antar *class*.
- Agregasi, yaitu hubungan yang menyatakan bagian (“terdiri atas..”).
- Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
- Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-*passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram* yang akan dijelaskan kemudian.





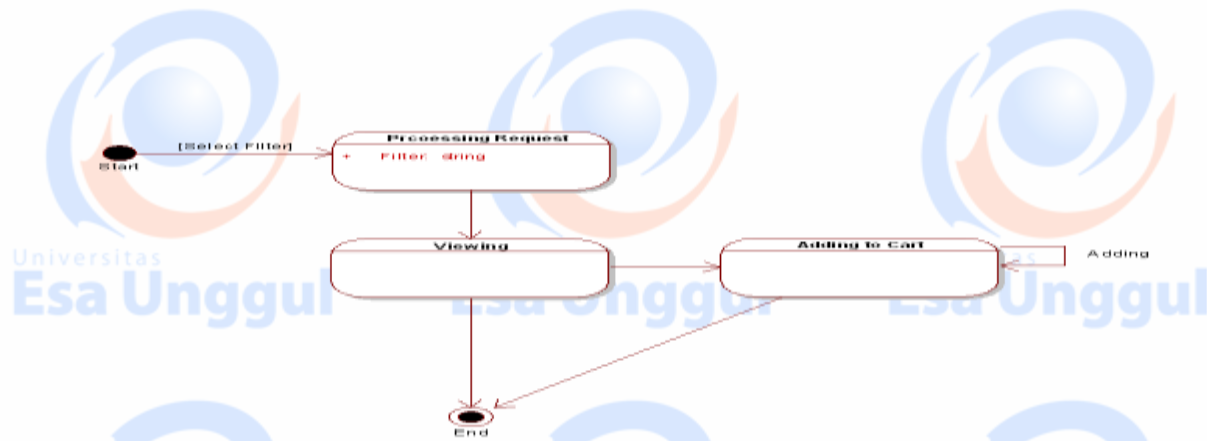
Gambar 2. Contoh class diagram

Statechart Diagram

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu state ke state lainnya) suatu objek pada sistem sebagai akibat dari stimuli yang diterima. Pada umumnya statechart diagram menggambarkan class tertentu (satu class dapat memiliki lebih dari satu statechart diagram).

Dalam UML, state digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya saat itu. Transisi antar state umumnya memiliki kondisi guard yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku. Action yang dilakukan sebagai akibat dari event tertentu dituliskan dengan diawali garis miring.

Titik awal dan akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah.



Gambar 3. Contoh *statechart diagram*

Activity Diagram

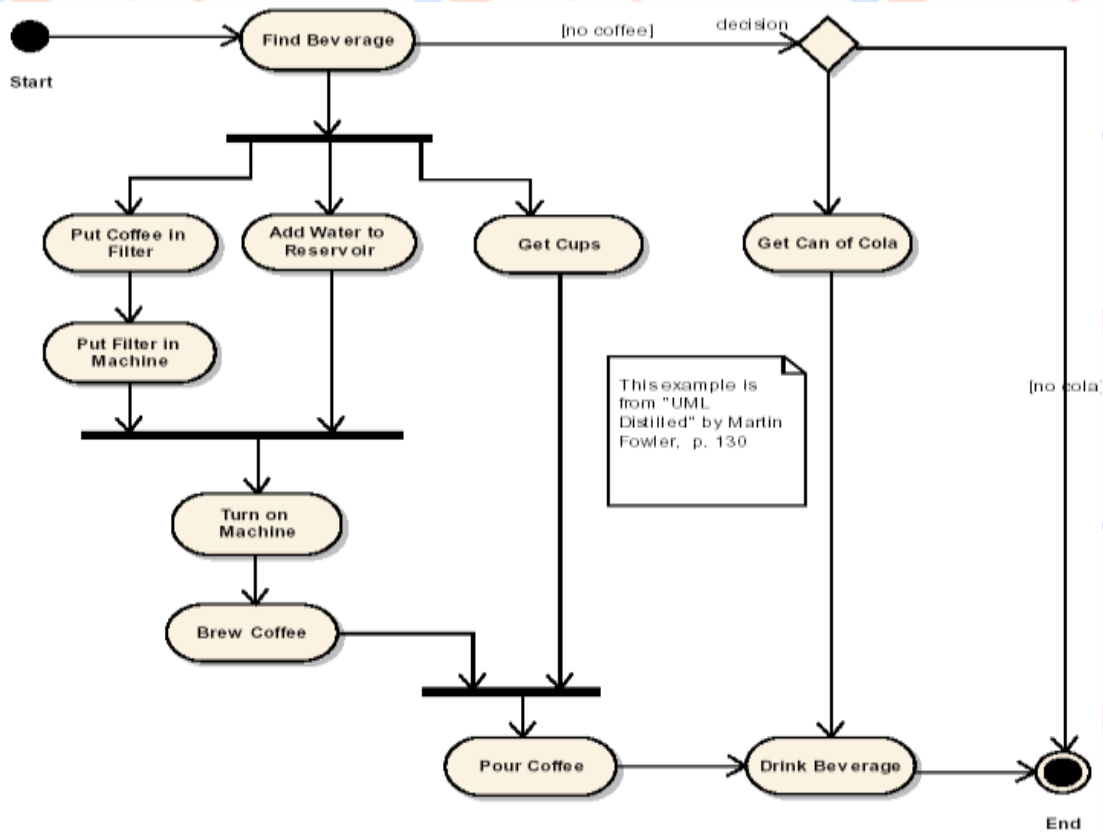
Activity diagrams menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-trigger oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal.

Activity diagram dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.



Gambar 4. Contoh *activity diagram* tanpa *swimlane*

Sequence Diagram

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atas dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

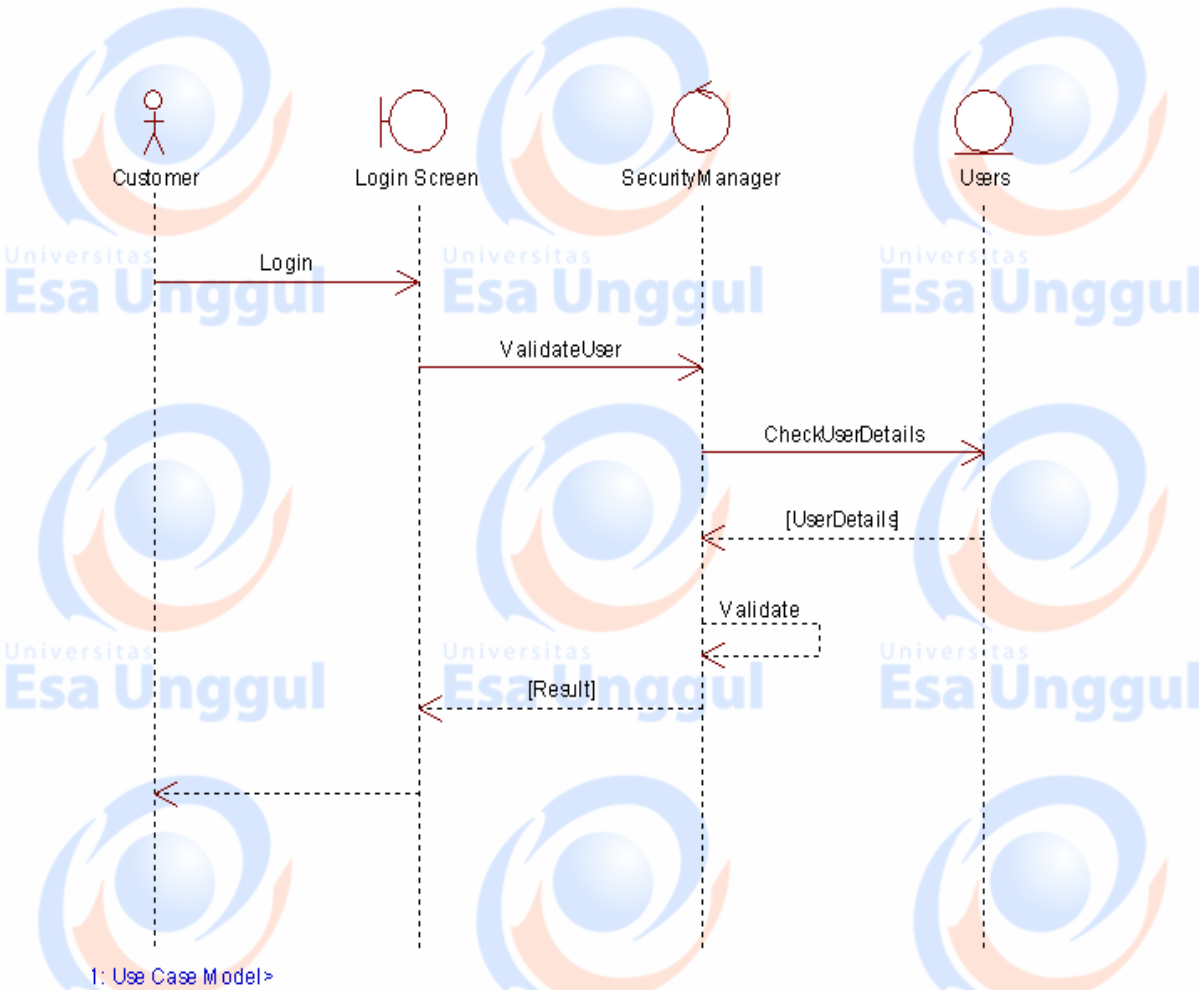
Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang *trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan.

Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal.

Message digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*.

Activation bar menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*.

Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk objek *boundary*, *controller* dan *persistent entity*.

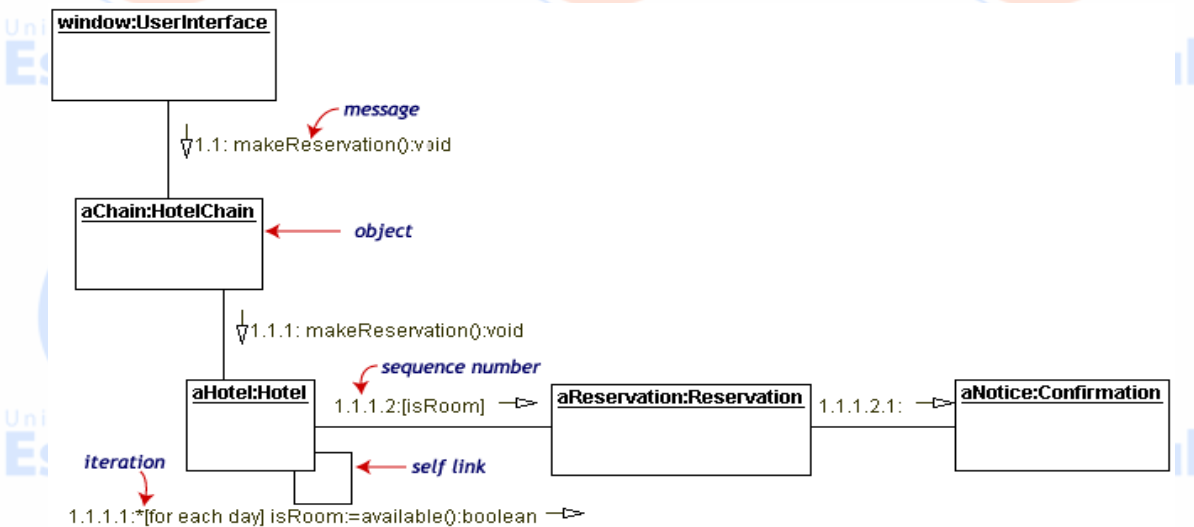


Gambar 5. Contoh *sequence diagram*

Collaboration Diagram

Collaboration diagram juga menggambarkan interaksi antar objek seperti *sequence diagram*, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*.

Setiap *message* memiliki *sequence number*, di mana *message* dari level tertinggi memiliki nomor 1. Messages dari level yang sama memiliki prefiks yang sama.



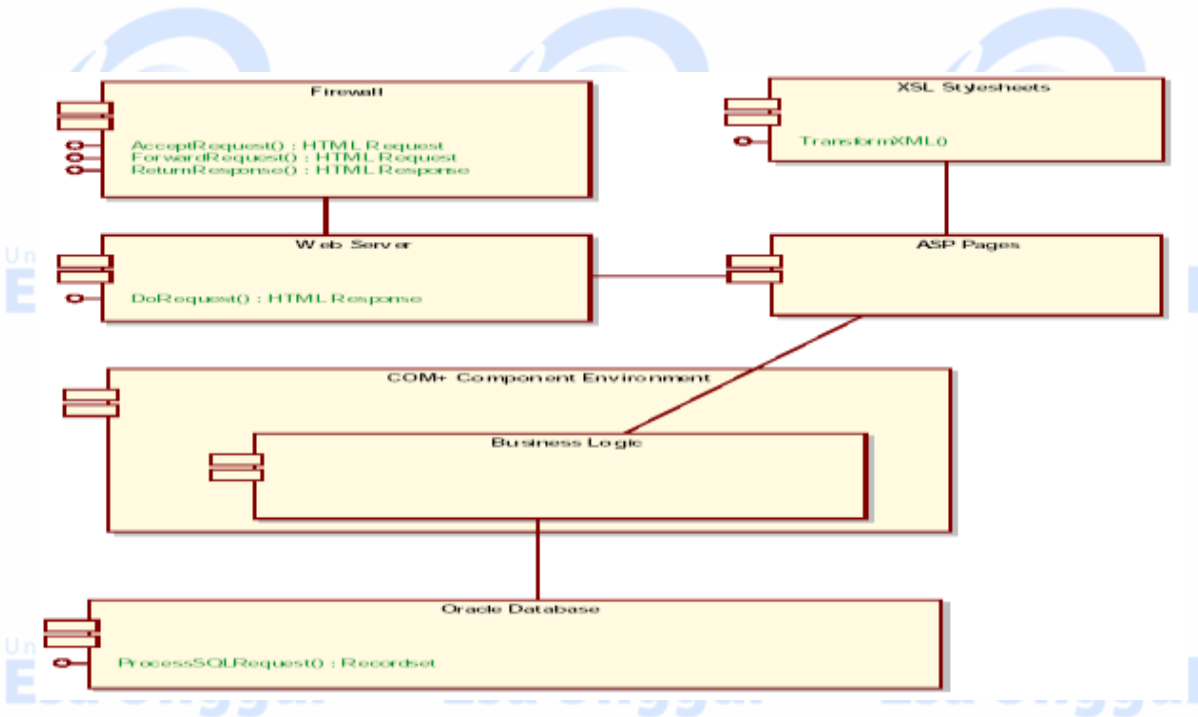
Gambar 6. Collaboration diagram

Component Diagram

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) di antaranya.

Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*, baik yang muncul pada *compile time*, *link time*, maupun *run time*. Umumnya komponen terbentuk dari beberapa *class* dan/atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil.

Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.



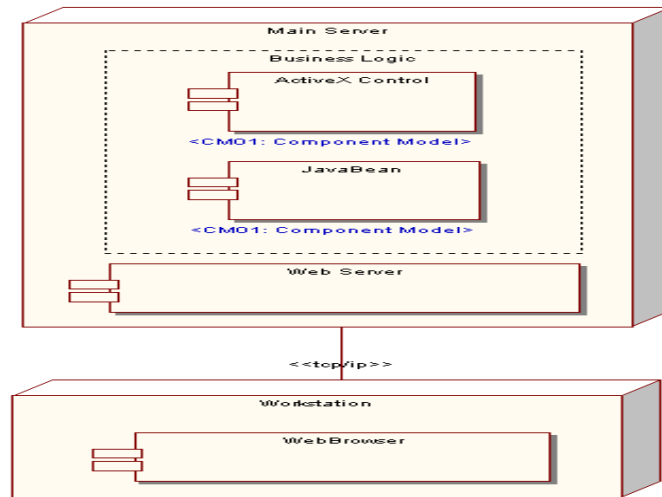
Gambar 7. Contoh *component diagram*

Deployment Diagram

Deployment/physical diagram menggambarkan detail bagaimana komponen di-deploy dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, server atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisik

Sebuah *node* adalah server, *workstation*, atau piranti keras lain yang digunakan untuk men-deploy komponen dalam lingkungan sebenarnya. Hubungan antar *node* (misalnya TCP/IP) dan *requirement* dapat juga didefinisikan dalam diagram ini.

The physical model shows where and how system components will be deployed. It is a specific map of the physical layout of the system.



Gambar 8. Contoh *deployment diagram*

Langkah-Langkah Penggunaan UML

Berikut ini adalah tips pengembangan piranti lunak dengan menggunakan UML:

1. Buatlah daftar *business process* dari level tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul.
2. Petakan *use case* untuk tiap *business process* untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh sistem. Kemudian perhalus use case diagram dan lengkapi dengan *requirement*, *constraints* dan catatan-catatan lain.
3. Buatlah *deployment diagram* secara kasar untuk mendefinisikan arsitektur fisik sistem.
4. Definisikan *requirement* lain (non-fungsional, *security* dan sebagainya) yang juga harus disediakan oleh sistem.
5. Berdasarkan *use case diagram*, mulailah membuat *activity diagram*.

6. Definisikan objek-objek level atas (*package* atau *domain*) dan buatlah *sequence* dan/atau *collaboration diagram* untuk tiap alir pekerjaan. Jika sebuah *use case* memiliki kemungkinan alir normal dan error, buatlah satu diagram untuk masing-masing alir.

7. Buatlah rancangan *user interface* model yang menyediakan antarmuka bagi pengguna untuk menjalankan skenario *use case*.

8. Berdasarkan model-model yang sudah ada, buatlah *class diagram*. Setiap *package* atau *domain* dipecah menjadi hirarki *class* lengkap dengan atribut dan metodenya.

Akan lebih baik jika untuk setiap *class* dibuat *unit test* untuk menguji fungsionalitas *class* dan interaksi dengan *class* lain.

9. Setelah *class diagram* dibuat, kita dapat melihat kemungkinan pengelompokan *class* menjadi komponen-komponen. Karena itu buatlah component diagram pada tahap ini. Juga, definisikan tes integrasi untuk setiap komponen meyakinkan ia berinteraksi dengan baik.

10. Perhalus *deployment diagram* yang sudah dibuat. Detilkan kemampuan dan *requirement* piranti lunak, sistem operasi, jaringan, dan sebagainya. Petakan komponen ke dalam node.

11. Mulailah membangun sistem. Ada dua pendekatan yang dapat digunakan :

- Pendekatan *use case*, dengan meng-*assign* setiap *use case* kepada tim pengembang tertentu untuk mengembangkan *unit code* yang lengkap dengan tes.
- Pendekatan komponen, yaitu meng-*assign* setiap komponen kepada tim pengembang tertentu.

12. Lakukan uji modul dan uji integrasi serta perbaiki model beserta *codenya*. Model harus selalu sesuai dengan *code* yang aktual.

13. Piranti lunak siap dirilis.

e. Tugas Pendahuluan

- Pengenalan UML.
- Bagian-bagian UML.
 - View.
 - Diagram.
- Langkah-langkah Pembuatan UML.

f. Praktikum

Menggunakan UML untuk membuat model sederhana.

g. Tugas Pasca Praktikum .

Membuat resume atas pekerjaan.

Modul 4 :

a. Judul :

Pembangunan Sistem Informasi Sederhana (studi kasus bebas).

b. Estimasi Waktu (opsional) :

400 Menit (4 x Tatap Muka Prak.).

c. Tujuan Instruksional Khusus :

Mahasiswa mampu untuk melakukan Pembangunan Sistem Informasi

d. Dasar Teori : dapat dibaca pada modul sebelumnya.

e. Tugas Pendahuluan

-----tidak ada -----

f. Praktikum :

1. Identifikasikan seluruh informasi yang dibutuhkan.
2. Identifikasikan seluruh data yang dibutuhkan proses/informasi.
3. Identifikasikan seluruh tujuan setiap informasi bagi penggunaannya.
4. Identifikasikan seluruh sumber data yang dibutuhkan proses/informasi
5. Lakukan proses pengkodean

g. Tugas Pasca Praktikum

Membuat laporan urutan kerja dari tahapan analisis, perancangan hingga pembangunan sistem informasi tersebut.

DAFTAR PUSTAKA

1. Al Fatta, Hanif. (2007), **Analisis dan Perancangan Sistem Informasi untuk Keunggulan Bersaing Perusahaan dan Organisasi Modern**, Andi, Yogyakarta.
2. Fathansyah. (1995), **Basis Data**, Penerbit Informatika, Bandung.
3. Hartono, Jogyanto. (1990), **Analisis dan Disain Sistem Informasi**, Andi , Yogyakarta.
4. Hartono, Jogianto. (2007), **Model Kesuksesan Sistem Teknologi Informasi**, Andi, Yogyakarta.
5. Korth, Henry F. (1991), **Database System Concept**, McGraw-Hill Publishing Company.
6. Nugroho, Adi. (2003), **Analisis dan Perancangan Sistem Informasi dengan Metodologi Berorientasi Objek**, Informatika, Bandung.
7. Pohan, Husni Iskandar. (1997), **Pengantar Perancangan Sistem**, Penerbit Erlangga, Jakarta.
8. Senn, James A. (1985), **Analysis and Design of Information Systems**, McGraw-Hill Publishing Company.
9. Suhendar, A dan Gunadi, Hariman. (2002), **Visual Modeling Menggunakan UML dan RATIONAL ROSE**, Informatika, Bandung.
10. Sutopo, Ariesto Hadi. (2002), **Analisis dan Desain Berorientasi Objek**, J&J Learning, Yogyakarta.
11. Tunas, Billy. (2007), **Memahami dan Memecahkan Masalah dengan Pendekatan Sistem**, PT. Nimas Multima, Jakarta.
12. Whitten, Bentley, Barlow. (1989). **Systems Analysis & Design Methods**, Penerbit IRWIN, USA.