DAFTAR ISI

MODUL 1
Konsep Objek Oriented

A. Tujuan Khusus

Mahasiswa dapat memahami penerapan sistem secara nyata kedalam konsep berbasis objek .

B. Pertemuan

Pertemuan 1 dan Pertemuan 2

C. Teori Penunjang

A. PENGANTAR MODOLOGI BERORIENTASI OBJEK

Melihat laju perkembangan teknologi informatika yang demikian pesat dewasa ini, maka pendekatan berarah objek diperkirakan akan menggantikan pendekatan Struktural dalam merancang sistem di masa yang akan datang. Hal ini dikarenakan pendekatan berarah objek memiliki beberapa kelebihan diantaranya Aplikasi yang akan dibangun didukung oleh perkembangan teknologi *Software Engineering* yang baru menuju ke konsep *Object Oriented.*

1. **Tehnologi dengan pendekatan *object-oriented*** ini memungkinkan komponen dari program dapat dipakai ulang sehingga mempermudah dalam pengembangan sistem selanjutnya serta program yang dihasilkan pun makin berkualitas baik.
2. **Aplikasi yang dibangun dengan pendekatan ini** mudah dalam perawatan karena strukturnya mengalami proses inheritance. Sehingga teknologi dengan pendekatan object-oriented ini dapat mengurangi efek samping atau resiko ketika terjadi perubahan pada struktur.

   Sampai saat ini dikenal dua pendekatan yang utama dalam pengembangan sistem dengan metode berarah objek yaitu:

   1. **Pendekatan Langsung** (*Direct Approach*), adalah suatu pendekatan perancangan sistem alam metode berarah objek dimana pendesain sistem langsung mendesain objek-objek yang diperlukan sistem. Kelemahan dari pendekatan ini adalah karena seorang pendesain diberi kebebasan dalam membuat objek sehingga untuk

menghasilkan objek yang benar-benar baik maka diperlukan pengulangan proses untuk perbaikan objek.

2. **Pendekatan Sintesis** (*Synthesis Approach*), adalah suatu pendekatan dalam perancangan sistem dengan metode berarah objek dimana objek–objek yang dihasilkan diperoleh dari penerapan metode struktural. Kelemahan dari pendekatan ini adalah seorang pendesain melakukan tahapan kerja lebih banyak yaitu harus merancang system secara struktural terlebih dahulu kemudian mentransfernya ke dalam bentuk objek.

2.1. Pendekatan Sintesis (*Synthesis Approach*)

Pendekatan sintesis adalah pendekatan metode berarah objek secara tidak langsung. Maksud tidak langsung disini adalah bahwa objek-objek diperoleh dari hasil pendesainan secara struktural. Jadi dalam pendekatan ini, langkah pertama adalah membangun terlebih dahulu data dan proses, kemudian menyatukan data dan proses tersebut menjadi objek–objek dengan aturan-aturan tertentu.

2.2. Pendekatan Langsung (*Direct Approach*)

Pendekatan secara langsung adalah metode berarah objek yang meliputi:
analisis berarah objek (OOA), perancangan berarah objek (OOD), pemrograman berarah objek (OOP) dan pengujian berarah objek (OOT).

Istilah “objek” telah digunakan dalam cara yang berbeda dari dua
disiplin ilmu yang berbeda.

1. Dari permodelan Informasi adalah:
   “suatu representasi dari beberapa dunia nyata dan sejumlah kejadian”; dan

2. Dari bahasa pemrograman berarah objek adalah:
“suatu *runtime* beberapa proses dan nilai yang ditentukan dengan deskripsi yang disebut kelas”.

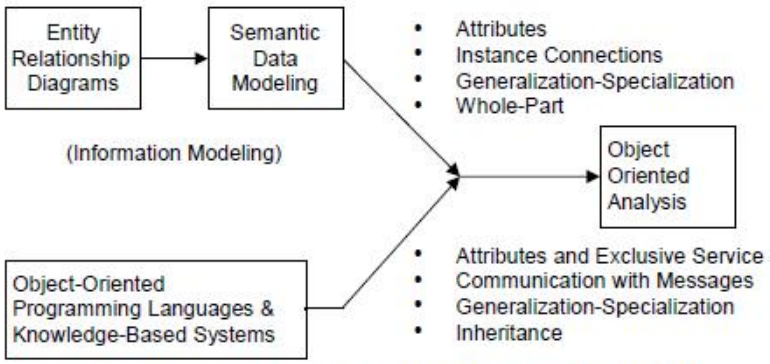
Objek juga didefinisikan sebagai berikut:
“Objek dapat didefinisikan sebagai suatu encapsulation atau penggabungan dari data (yang diwakilkan oleh atribut-atribut) dan operasi–operasi (disebut juga metode/prosedur) yang akan melakukan proses terhadap data-data tersebut.”

Persamaan dari pendekatan *object oriented* :

Object oriented = Class-&-Object + Inheritance + Communication with Massage

Analisis berarah objek dibangun sebagian besar diatas konsep permodelan Informasi, bahasa pemrograman berarah objek dan sistem berbasis pengetahuan (konsep yang mempunyai basis yang baku dan pengertian prinsip-prinsip untuk menangani kerumitan).

Dari model Informasi secara analogi mengambil bentuk *attribute, instance connection, generalize-specification* dan *whole-part*. Dari bahasa pemrograman berarah objek dan sistem berbasis pengetahuan secara analogi mengambil bentuk pembungkusan dari atribut dan *service, communication with massages, gen-spec* dan *inheritance*.



Gambar 1 OOA Gabungan dari beberapa Disiplin Ilmu

Metode pembangunan perangkat lunak berarah objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data aksi yang dilakukan terhadapnya. Penggabungan proses dan data didalam suatu objek dikenal sebagai pembungkusan. Ditambah dengan konsep *infohiding, object class or instant connection* dan *polymorphism* merupakan dasar dari metode berarah objek.

Adapun komunikasi antar proses yang dapat berarti komunikasi antar objek dilakukan dengan melalui pesan (*communication with massages*).

2.3. Prinsip-prinsip

Prinsip-prinsip pengelolaan kompleksi tersebut terdiri atas abstraksi (prosedural dan data), pembungkusan, asosiasi, komunikasi dengan pesan, metode organisasi, penskalaan, dan pengelompokan prilaku [COA90].

1. Abstraksi (*Abstraction*)

Abstraksi: prinsip yang menganggap bahwa aspek-aspek dari suatu objek yang tidak relevan dengan tujuan sekarang, digunakan untuk melengkapi tujuan tsb.

Mekanisme abstraksi:

a. Abstraksi Prosedural: sering dikarakteristikkan sebagai abstraksi fungsi atau subfungsi. Pemecahan proses kedalam tahapan-tahapan merupakan suatu metode dasar dalam penanganan kekomplekan suatu proses.
b. Abstraksi Data: prinsip pendefinisian suatu type data dalam istilah istilah pengoperasian yang menggunakan type tersebut dengan pembatasan terhadap nilai-nilai objek yang dapat dimodifikasi dan hanya dijalankan dengan menggunakan operasi, prinsip ini dapat menjadi suatu dasar untuk pengaturan tanggungjawab suatu system secara spesifik.

2. Pembungkusan (*Encapsulation*)

Pembungkusan adalah suatu prinsip yang digunakan ketika pengembangan struktur program secara keseluruhan, dimana tiap-tiap komponen program harus disimpan/disembunyikan dalam satu perancangan interaksi. Untuk tiap-tiap modul ditentukan dengan cara mengungkapkan sebagian kecil tentang informasi kerja secara pasti. Pembungkusan membantu memperkecil pekerja-an ketika pengembangan suatu sistem baru. Jika seorang perancang "*mem-bungkus*" bagian dari hasil analisis yang sering berubah-ubah, kemudian mengganti sesuai dengan kebutuhan, jika ada masalah, maka masalah tersebut lebih kecil resikonya bila dibandingkan dengan tanpa "*pembungkusan*".

3. Pewarisan (*Inheritance*)

Pewarisan adalah suatu cara kerja yang menyatakan kesamaan diantara kelas-kelas, penyederhanaan definisi kelas yang mirip ke suatu definisi yang telah ada sebelumnya. Pewarisan terdiri dari generalisasi dan spesialisasi.

4. Asosiasi (*Association*)

Asosiasi adalah penyatuan atau pengembangan ide-ide. Digunakan untuk me-nyatukan sesuatu secara bersama-sama dalam keadaan yang mirip.

5. Komunikasi dengan Pesan (*Communication with Massages*)

Pesan adalah suatu komunikasi, berupa tulisan atau ucapan, pengiriman pesan antar orang-orang. Hal ini dihubungkan dengan "*pembungkusan*", rincian itu merupakan kegiatan yang dilakukan dengan menerima pesan.

6. Metode Pengorganisasian (*Pervading Methods of Organization*)

Metode Pengorganisasian ini meliputi tiga bagian yaitu:

a. Perbedaan pengalaman kedalam bagian objek dan atributnya. (*object and attributes*)

b. Perbedaan antara keseluruhan objek dan komponen bagiannya. (*wholeand parts*)

c. Formasi dan perbedaan antara kelas-kelas objek yang berbeda. (*classes, members and distinguishing between them*)

7. Skala (*Scale*)

Skala adalah suatu prinsip yang menerapkan *whole-part* untuk membantu menghubungkan sesuatu yang berukuran besar tanpa perlu meliputi semuanya.

8. Klasifikasi Tingkah Laku (*Behavior Classification*)

Jenis klasifikasi tingkah laku yang digunakan secara umum yaitu:

a. Berdasarkan kepada kebutuhan yang mendesak

b. Berdasarkan perjalanan waktu (historis)

c. Berdasarkan fungsi

D. Spesifikasi S/W dan H/W : Easycase, Ms.Visio dan Pentium 4 atau Dual Core

E. Latihan Soal

1. Jelaskan apa perbedaan pemrograman terstruktur dan berbasis objek oriented ?
2. Bagaimana komunikasi antara program-program /class pada konsep objek oriented?
3. Gambarkan bentuk objek oriented dari kelas Hewan dengan memiliki beberapa jenis yang hidup didarat dan di air?

MODUL 2
METODOLOGY PERANCANGAN SISTEM

A. Tujuan Khusus

Mahasiswa dapat memahami dan menerapkan perancangan sistem dengan metodologi berbasis objek

B. Pertemuan

Pertemuan 3 dan 4

C. Teori Penunjang

Metodologi pengembangan sistem adalah suatu proses pengembangan sistem yang formal dan presisi yang mendefinisikan serangkaian aktivitas, metode, best practices, dan tool yang terautomasi bagi para pengembang dan manajer proyek dalam rangka mengembangkan dan merawat sebagian besar atau keseluruhan sistem informasi atau software.

• Pendekatan terstruktur mengenalkan penggunaan alat-alat dan teknik-teknik untuk mengembangkan sistem yang terstruktur.

Ciri-ciri utama yang mendukung pendekatan terstruktur adalah :

1. Memanfaatkan alat-alat pemodelan, menggunakan model untuk menjelaskan berbagai sistem, sub sistem untuk ditelah dan dievaluasi oleh pelanggan dan pengembang (sebagai alat komunikasi, eksperimentasi atau prediksi).

2. Merancang berdasar modul, Modularisasi adlah proses yang membagi suatu sistem menjadi beberapa modul yang dapat beroperasi secara independent.

3. Bekerja dengan pendekatan top-down, Dimulai dari level atas (secara global) kemudian diuraikan sampai ke tingkat modul (rinci).

4. Dilakukan secara iterasi, Dengan iterasi akan didapat hasil yang lebih baik, terlalu banyak iterasi juga akan menurunkan hasilnya dan menunjukkan bahwa tahap sebelumnya tidak dilakukan dengan baik.

5. Kegiatan dilakukan secara paralel, Pengembangan subsistem-subsistem dapat dilakukan secara paralel, sehingga akan memperpendek waktu pengembangan sistem. Menggunakan

CASE, Dengan CASE computer aided software engineering) memungkinkan analis dapat membangun sistem dan menghasilkan executable secara otomatis.

• **Tujuan pendekatan terstruktur** adalah agar pada akhir pengembangan perangkat lunak dapat memenuhi kebutuhan user, dilakukan tepat waktu, tidak melampaui anggaran biaya, mudah dipergunakan, mudah dipahami dan mudah dirawat. Metodologi pengembangan sistem yang ada biasanya dibuat atau diusulkan oleh:

— Penulis Buku

— Peneliti

— Konsultan

— System House

— Pabrik Software

Mengapa perlu metodologi sistem informasi

– Menjamin adanya konsistensi proses.

– Dapat diterapkan dalam berbagai jenis proyek.

– Mengurangi resiko kesalahan dan pengambilan jalan Pintas.

– Menuntut adanya dokumentasi yang konsisten yang bermanfaat bagi personal baru dalam tim proyek

Pada prinsipnya metodologi dapat dikembangkan sendiri, bisa juga menggunakan metodologi yang sudah teruji penerapannya. Metodologi Pengembangan Sistem diklasifikasikan menjadi 3 golongan :

1**.** ***Functional Decomposition Methodologies*** (Metodologi Pemecahan Fungsional).

- HIPO (Hierarchy Input Process Output),
- SR (Stepwise Refinement),
- ISR (Iterative Stepwise Refinement),
- Information Hiding

2. ***Data Oriented Methodologies*** (Metodologi Orientasi Data)

- Data Flow Oriented Methodologies : SADT, Composite Design, SSAD
- Data Structure Oriented Methodologies : JSD, W/O

3. ***Prescriptive Methodologies*** ISDOS, PLEXSYS, PRIDE, SPEKTRUM Avison dan Fitzgerald (2003) menggunakan pendekatan lain dalam mengelompokkan (INFORMATION SYSTEM DEVELOPMENT METHODS) ISDM. Mereka mendasarkan pada filosofi dasar yang digunakan oleh ISDM. Menurut mereka terdapat enam kelompok ISDM, yaitu:

1. **Metodologi berorientasi proses** (processoriented methodologies) seperti Structured Analysis, Design, and Implementation of Information Systems (STRADIS) dan Yourdon Systems Method (YSM);

2. **Metodologi berorientasi obyek** (object-oriented methodologies) seperti Object Oriented Analysis (OOA) dan Rational Unified Process (RUP);

3. **Metodologi pengembangan cepat** (rapid development methodologies) seperti Extreme Programming (XP) dan Dynamic Systems Development Method (DSDM);

4. **Metodologi berorientasi orang** (people-oriented methodologies) seperti Effective Technical and 5. Metodologi berorientasi organisasi (organizational-oriented methodologies) seperti Soft Systems Methodology (SSM) dan Information Systems Work and Analysis Changes (ISAC); dan 6. **Metodologi campuran** (blended methodologies) seperti Merise dan Information Engineering (IE). Sebelum beranjak lebih jauh, kita harus mengenal Prinsip-prinsip dasar Pengembangan Sistem. Reprentasi Metodologi-Metodologi Pengembangan Sistem: Architected Rapid, Aplication Development, Dynamics Systems Developments, Joint Application Development, Information Enginering, Rapid Application Development, eXtreme Programming(XP), dan masih banyak yang lain yang biasanya disebut dengan Methodware.

Kembali ke prinsip metodologi pengembangan sistem :

· **Libatkan para pengguna sistem**. Para programmer dan sitem analis terkadang merasakan hal yang aneh dalam pembuatan sistemnya bahkan mereka adalah pembuat kekacauan dalam perusahaan. Oleh karena itu melibatkan para pengguna merupakan jalan yang terbaik, karena timbul komunikasi antar sesama yang nantinya akan membuat suatu pemecahan masalah yang jelas.

· **Pendekatan pemecahan masalah**. Berusaha untuk mempelajari dan memahami masalah, memikirkan solusi, mengidentifikasi solusi-solusi, merancang serta mengimplementasikan, mengatasi, mengevaluasi dan memperbaiki solusi-solusi. Pada kenyataannya harus meminimalkan dan menghilangkan resiko-resiko.

· **Bentuklah Aktivitas**. Setiap aktifitas-aktifitas atau fase berbeda-beda karena berbeda pula permasalahan pada intinya. Mulai dari Definisi, Analisis, Perancangan yang logis, Keputusa, Dan penerapan. Contohnya : Ketepatan waktu, Komplesitas, Strategi, Sumber Daya, dan sebagainya.

· **Dokumentasikan** sepanjang pengembangan. Maksudnya dalam perjalanan pengembangan sebaiknya laporan-laporan di dokumentasikan mulai dari awal hingga akhir. Karena hal ini dapat menutup kesalahan-kesalahan yang terjadi dengan cepat (kerja cerdas dan kritis).

· **Bentuklah standar**. Integrasi sistem telah menjadi kritis bagi keberhasilan semua sistem informasi perusahaan. Mulai dari Teknologi Database (Oracle, SQL Server, DB2, dll). Teknologi perangkat Lunak (Visual C++, Visual Basic, dll). Teknologi antar muka (xHtml, Dreamweaver, dll). Tanpa adanya arsitektur yang jelas, setiap sistem akan dibangun secara radikal.

· **Kelola Proses dan Proyek**. Proses manajemen memastikan bahwa suatu proses atau manajemen terpilih perusahaan digunakan secara konsisten agar proses yang sudah ad berjalan dengan baik. Proyek manajemen bahwa suatu sistem informasi digunakan dan dikembangkan dengan biaya minimal, dengan jangka waktu yang ditentukan, serta hasil akhir yang memuaskan.

· **Membenarkan sistem informasi** sebagai investasi modal. Perlu diketahui bahwa setiap adanya sistem informasi para pemilik sistem sudah berkomitmen untuk mensponsori dan mendanai seluruh sistem yang dibuat. Pemilik sistem harus mempertimbangkan masalah, beserta solusinya agar tidak ada faktor yang membahayakan dalam proses bisnis. Cost-Effectiveness, hasil yang didapat dengan menyeimbangkan dengan biaya yang telah dikeluarkan. Karena sistem informasi diacukan sebagai investasi modal, proyek pengembangan system ini sering dikendalikan dengan perencanaan enterprise. Rencana enterprise strategis, rencana formal jangka 5 tahun yang mendefinisikan misi, visi, tujuan,

strategi, titi tolak dan ukuran kemajuan dalam suatu tujuan, yang memetakan arah pada seluruh aspek bisnis.

· **Janganlah takut untuk Membatalkan** atau Merevisi Lingkup. Membatalkan proyek yang sudah tidak praktis lagi, mengevaluasi ulang, menyesuaikan biaya dan waktu yang akan ditingkatkan, mengurangi lingkup apabila biaya tidak cukup untuk membiayai sitem. Setelah itu diharapkan untuk mengelola resiko/ memanajemen resiko, mengontrol apa yang mungkin gagal dalam proyek sebelum nantinya pada proses penyelesaian timbul resiko yang besar.

· **Bagilah dan takhlukan**. Pada kerangka pembuatan system sebaiknya sang sistem analis membuat kerangka-kerangka atau biasanya dikatakan sebagai subsistem/pemfaktoran. Dengan pembagian subsitem, permasalahan akan menjadi jelas dan lebih mudah untuk dikelola. Pendekatan ini pun akan melengkapi komunikasi dan manajemen proyek.

· **Desainlah sistem** untuk pertumbuhan dan perubahan. Perlu kita ketahui di kehidupan ini berbagai pola berubah, mulai dari bisnis, kebutuhan-kebutuhan, hingga prioritas manusia berubah oleh karena itu sistem informasi pun turut berubah. Sistem harus di desain untuk persyaratan dalam akomodasi pertumbuhan dan perubahan. Perlu diketahui, sebagus-bagusnya kita mendesain sistem suatu saat sistem itu tidak dapat mendukung proses bisnis. Maka dengan itu diperlukan pengalaman-pengalaman, pembelajaran, dari setiap pengembangan dan cara mendesain sistem untuk membangun sistem yang lebih baik.

pengembangan SI (Siklus Hidup SI)

• **Model sekuensial linier** (clasic life cycle/waterfall model), terdiri dari tahapan perencanaan sistem (rekayasa sistem), analisa kebutuhan, desain, penulisan program, pengujian dan perawatan sistem.

- **Perencanaan sistem (rekayasa sistem),** pada tahapan ini dilakukan pengumpulan kebutuhan pada level sistem yaitu kebutuhan perangkat keras, perangkat lunak, orang dan basisdata. Pengumpulan kebutuhan ini penting dilakukan karena sistem informasi (PL) yang akan dibangun merupakan bagian dari sistem komputer. *PL=Perangkat Lunak

- **Analisa kebutuhan sistem informasi**, pada tahap ini dilakukan pengumpulan kebutuhan untuk sistem informasi (PL) yang berupa data input, proses yang terjadi dan output yang diharapkan dengan melakukan wawancara dan observasi, hasilnya berupa diagram yang dapat

berupa diagram aliran data (DFD) dengan kamus data, diagram keterhubungan entitas (ERD) atau diagram perubahan status (STD).

- **Desain**, pada tahap ini menterjemahkan analisa kebutuhan ke dalam bentuk rancangan sebelum penulisan program yang berupa perancangan antarmuka (input dan output), perancangan file-file atau basis data dan merancang prosedur (algoritma).

- **Penulisan program**, hasil rancangan di atas diubah menjadi bentuk yang dimengerti oleh mesin dalam bentuk bahasa pemrograman. Jika rancangannya rinci maka penulisan program dapat dilakukan dengan cepat.

• **Model prototipe (prototyping model),** dimulai dengan pengumpulan kebutuhan dan perbaikan, desain cepat, pembentukan prototipe, evaluasi pelanggan terhadap prototipe, perbaikan prototipe dan produk akhir.

- **Reaksi awal dari pengguna**, diawali dengan menampilkan sebuah protipe sistem informasi, kemudian melihat reaksi dari pengguna saat bekerja dengan prototipe apakah fitur-fitur sistem pada prototipe tersebut sudah sesuai dengan kebutuhannya. Reaksi tersebut dikumpulkan dalam lembar observasi, wawancara dan kuesioner.

- **Saran-saran pengguna**, saran-saran merupakan hasil interaksi pengguna dengan prototipe yang ditampilkan (evaluasi pengguna) yang merupakan masukan untuk perbaikan, pengubahan atau ‘menghentikan’ prototipe sehingga dapat memenuhi kebutuhan pengguan dengan lebih baik.

- **Inovasi,** adalah kemampuan-kemampuan sistem baru yang sebelumnya tidak ada pada saat pengguna berinteraksi dengan prototipe. Inovasi prototipe jika berhasil akan menjadi bagian dari sistem hasil jadi.

- **Rencana revisi**, prototipe menggambarkan sistem di masa datang. Rencana revisi membantu mengidentifikasikan prioritas-prioritas apa saja yang akan diprototipekan selanjutnya.

• **Rapid Application Development (RAD)** model, dengan kegiatan dimulai pemodelan bisnis, pemodelan data, pemodelan proses, pembangkitan aplikasi dan pengujian.

- **Pemodelan bisnis**, aliaran informasi dimodelkan dengan menjawab

pertanyaan : informasi apa yang mempengaruhi proses bisnis ? informasi apa yang akan dimunculkan ? siapa yang memunculkannya ? kemana informasi tersebut diberikan ? siapa yang memprosesnya ?

- **Pemodelan data**, pengumpulan objek data yang dibutuhkan, mengidentifikasi karakteristik setiap objek dan mendefinisikan hubungan antar objek tersebut.

- **Pemodelan proses**, mendeskripsikan proses (input ditransformasi menjadi output).

- **Pembangkitan aplikasi**, jika mungkin menggunakan kembali komponen program yang ada.

- **Pengujian**, dengan penggunaan kembali komponen program yang telah siap maka akan mengurangi waktu pengujian.

• **Model evolusioner** yang dapat berupa model incremental atau model spiral Model incremental merupakan gabungan model sekuensial linier dengan prototyping (mis perangkat lunak pengolah kata dengan berbagai versi). Sedangkan model spiral menekan adanya analisa resiko. Jika analisa resiko menunjukkan ada ketidakpastian terhadap kebutuhan, maka pengembangan system dapat dihentikan.

a. **Model incremental**, pada model ini tahapannya sama dengan model sekuensial linier dengan increment pertama sering merupakan produk inti. Dengan adanya penambahan kemampuan pada produk inti maka akan dimasukkan pada increment kedua dst. (mis. Perangkat lunak pengolah kata dengan berbagai versi).

b. **Model spiral**, menekan adanya analisa resiko. Jika analisa resiko menunjukkan ada ketidakpastian terhadap kebutuhan, maka pengembangan sistem dapat dihentikan. Model ini dibagi menjadi 6 kegiatan yaitu :

- **Komunikasi pelanggan**, komukasi antara pengembang dengan pelanggan untuk menentukan kebutuhan kerja.

- **Perencanaan**, mendefinisikan sumberdaya, batas waktu dan hubungan informasi proyek lain.

- **Analisa resiko**, untuk menentukan resiko teknis dan manakemen.

- **Rekayasa**, membangun satu atau lebih aplikasi yang dapat mewakili.

- **Konstruksi** dan peluncuran, untuk mengkonstruksi, menguji, menginstal dan memberi dukungan pemakai (mis. pelatihan).

- **Evaluasi pelanggan**, untk memperoleh umpan balik pelanggan berdasarkan pada penilaian terhadap hasil rekayasa.

• **Teknik generasi ke-empat (4GT),** dimulai dengan pengumpulan kebutuhan, strategi perancangan, implementasi menggunakan 4GL dan pengujian. Untuk aplikasi yang kecil dimungkinkan dari pengumpulan kebutuhan langsung mengimplementasikannya menggunakan 4GL, tetapi untuk aplikasi yang besar perlu adanya pengembangan strategi perancangan.

Dari berbagai model pengembangan sistem informasi di atas, maka proses dari pengembangan sistem yang utama adalah analisis sitem, desain sistem dan implementasi sistem. Tahap perencanaan system sebanarnya merupakan tahapan sebelum dilakukan pengembangan sistem dan tahap pemeliharaan sistem sebenarnya juga merupakan tahapan setelah pengembangan sistem selesai dilakukan dan system telah dioperasikan.

Tahap pemeliharaan membutuhkan waktu dan biaya 48 – 60 % dari pengembang sistem. Ada dua alasan dilakukannya pemeliharaan :

1. Memperbaiki kesalahan dalam perangkat lunak setelah sistem diberikan ke pelanggan.

2. Meningkatkan kemampuan perangkat lunak untuk merespon

perubahan kebutuhan-kebutuhan organisasional, yang dapat berupa :

a. adanya permintaan fitur-fitur tambahan dari pemakai.

b. bisnis berubah seiring dengan waktu.

c. teknologi perangkat keras dan perangkat lunak berubah dengan pesat.

D. Spesifikasi S/W dan H/W : Easycase, Ms.Visio dan Pentium 4 atau Dual Core

E. Latihan Soal

1. Dari pengamatan kita terhadap alam sekitar coba lakukan beberapa pemodelan berbasis objek oriented untuk merancang sebuah system Rental Mobil apa saja yang harus terdapat, jabarkan objek-objek yang saling keterhubungan/Inheritance.

MODUL 3
ANALISIS BERORIENTASI OBJEK

A. Tujuan Khusus

Mahasiswa dapat memahami dan melakukan analisa sistem menggunakan objek oriented dengan UML (*Unifield Markup Language*)

B. Pertemuan

Pertemuan 5 dan 6

C. Teori Penunjang

Pendekatan analisa berarah objek terdiri dari lima pokok aktivitas, yaitu:

a. Menentukan Kelas-&-Objek (*finding class-&-objects*)

b. Identifikasi Struktur (*identifying structures*)

c. Identifikasi Subjek (*identifying subject*)

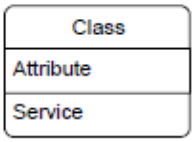
d. Pendefinisian Atribut (*defining attributes*)

e. Pendefinisian Service (*defining services*)

3.1. Penentuan Kelas-&-Objek (*class-&-object*)

Pengertian **objek** adalah suatu abstraksi dari suatu lingkup permasalahan / implementasi yang menggambarkan kemampuan dari system untuk menangkap informasi tentang objek tersebut, berinteraksi dengan atau keduanya; "pembung-kusan" pada nilai atribut dan service-nya.

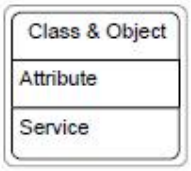
Kelas adalah suatu gambaran dari satu atau lebih objek, penggambaran dengan sekumpulan atribut dan service yang sama. Menggambarkan bagaimana untuk menciptakan objek-objek baru dalam suatu kelas.



Gambar 3.1 Simbol Kelas

Simbol ini digunakan untuk merepresentasikan generalisasi kelas dari lingkup permasalahan yang berhubungan dengan objek-objeknya yang digambar-kan dengan spesialisasi objek tersebut.

Kelas-&-objek adalah suatu istilah yang berarti suatu kelas-&-objek yang ada di kelas tersebut.

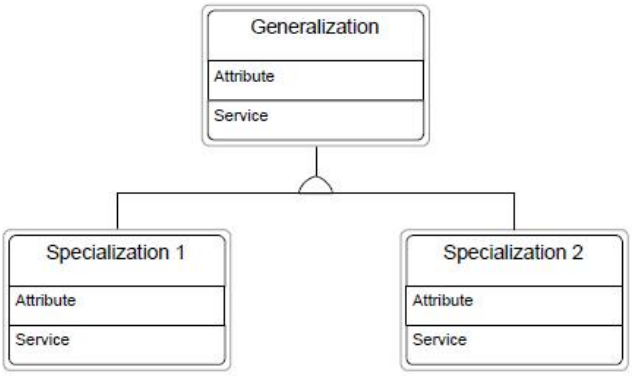


Gambar 3.2 Simbol Kelas-&-Objek

Simbol "**kelas-&-objek**" merepresentasikan suatu kelas dan objek-objeknya. Simbol kelas digambarkan dengan segiempat yang dicetak tebal dan di bagi ke dalam tiga bagian. Simbol objek-objeknya digambarkan dengan segiempat yang ada disebelah luar dari simbol kelas. Hubungan dari satu objek ke objek lain atau hubungan satu kelas ke kelas lain atau hubungan satu objek ke suatu kelas, merupakan hal yang dapat dilakukan terhadap simbol tersebut. Simbol diberi nama dengan kelas-&-objeknya. Atribut dan service dapat digunakan untuk tiap-tiap objek disuatu kelas. Nama kelas-&-objek adalah suatu kata kerja atau kata sifat. Nama kelas-&-objek harus menggambarkan suatu objek tunggal dengan kelasnya. Contoh: ketika tiap objek menggambarkan sesuatu yang berhubungan dengan "**meja**" maka nama kelas-&- objeknya: "**anggota_meja**". Tiap objek minimal mempunyai satu item.

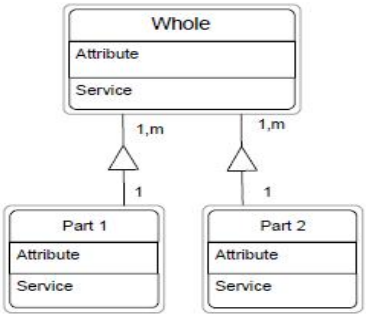
3.2. Identifikasi Lapisan Struktur

Struktur adalah suatu ekspresi dari lingkup permasalahan yang komplek, berhubungan dengan tanggungjawab terhadap sistem. Istilah struktur digunakan sebagai sebuah istilah yang menggambarkan struktur generalisasi-spesialisasi (*gen-spec*) dan struktur keseluruhan dan bagiannya (*whole-part*). Struktur *gen-spec* termasuk kebagian "perbedaan antar kelas" dalam tiga aspek dasar metode pengorganisasian. Struktur *gen-spec* dikenal sebagai struktur "*is a*" atau "*is a kind of*".



Gambar 3.3 Simbol *Generalize-Specification*

Dengan struktur *gen-spec,* pewarisan digambar secara eksplisit terhadap atribut dan servisnya. *Gen-spec* digambarkan dengan kelas atas generalisasi dan kelas dibawahnya adalah spesialisasi dengan penghubung diantaranya.setengah lingkaran membedakan antara kelas-kelas dalam *gen-spec.* Struktur *Whole-parts* digambarkan dengan objek pada bagian atas dan objek pada bagian bawah (dari simbol kelas-&-objek) dengan garis penghubung diantaranya, *whole-parts* dikenal sebagai struktur "*has a*".

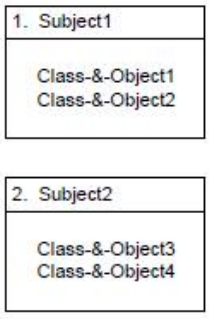


Gambar 3.4 Simbol *Whole-parts*

2.3. Identifikasi Lapisan Subjek

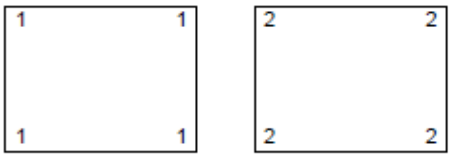
Subjek adalah suatu mekanisme yang membantu pembaca untuk menganalisa model permasalahan dari yang kecil hingga ke yang besar dan komplek. Pada lapisan subjek, gambarkan setiap subjek sebagai kotak persegi yang sederhana, dengan nama subjek dan nomor didalamnya sebagai tambahan, daftarkan kelas-kelas yang termasuk didalam subjek tersebut. Pada lapisan yang lain indikasikan subjek dengan label subjek pada sekat pemisah

kotak subjek untuk memandu pembaca dari subjek ke subjek lainnya. Dalam model yang besar dan membutuhkan fasilitas komunikasi, pertimbangkan penggunaan "set" dari lapisan secara terpisah untuk setiap subjek. Sebuah kelas dari subjek berada didalam lebih dari satu subjek. Subjek boleh berisi subjek lainnya. Persiapkan pemetaan multi level untuk membantu pembaca dalam suatu model yang besar.



Gambar 3.5 Simbol *Subject*

Penggambaran dari simbol diatas dengan lapisan yang berbeda dapat dilihat dalam gambar dibawah ini.



Gambar 3.6 Simbol *Subject* dengan Lapisan Berbeda

2.4. Pendefinisian Lapisan Atribut

Atribut adalah beberapa data (*state information*) dimana setiap objek di dalam suatu kelas mempunyai nilai tersendiri. Mendefinisikan atribut dengan cara:

a. Identifikasi atribut

b. Mencari posisi atribut

c. Identifikasi *instance connection*

Untuk identifikasi atribut perlu menjawab beberapa pertanyaan:

1. Bagaimana kita mendeskripsikannya secara umum?
2. Bagaimana kita pada permasalahannya
3. Bagaimana kita mendeskripsikannya pada bentuk tanggungjawab thd. sistem?
4. Apa yang harus diketahui?

5. Apa acuan untuk pelaksanaannya?

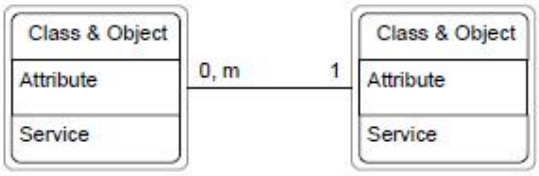
Periksalah kembali hasil analisis pada permasalahan yang sama/serupa. Pertimbangkan penggunaan atribut untuk merancang keputusan dalam segi waktu dengan memori yag dipergunakan. Buatlah tiap-tiap atribut dalam konsep *atomic*. Tambahkan "id" (*identifier*) dan "cid" (*connection identifier*), sehingga dapat dipergunakan pada spesifikasi teks jika diperlukan. Masukan setiap atribut kedalam kelas dan objeknya (periksa permasalahannya).

Daftarkan *inheritance* pada struktur *gen-spec*. posisi atribut yang bersifat umum letakan di tingkat yang lebih atas, sedangkan posisi atribut yang lebih spesifik diletakan ditingkat yang berikutnya. *Instance connection:* suatu model pemetaan lingkup permasalahan pada satu objek yang memerlukan objek lainnya, untuk memenuhi "tanggungjawab" terhadap proses yang harus dilakukannya. *Instance connection* merepresentasikan bagian dari *state information* yang diperlukan oleh sebuah objek.

Periksa hasil analisis pada permasalahan yang sama. Untuk setiap objek, tambahkan garis koneksi. Tambahkan subjek-bahan pemetaan (*subject-matter mappings*) diantara objek, perhatikan keterhubungan dengan struktur *gen-spec*.

Gambar 3.7 Simbol *Attributes*

Atribut ditempatkan dibagian tengah dari **kelas-&-objek** dan dari **kelas**.



Gambar 3.8 Simbol *Instance Connection*

Instance connection digambarkan dengan garis yang mehubungkan antar objek. Garis tersebut menghubungkan antar individu objek dari pada antar kelas. Tiap objek mempunyai jumlah atau batas penandaan (m, n) untuk *Instance connection*-nya.

2.5. Pendefinisian Lapisan Service

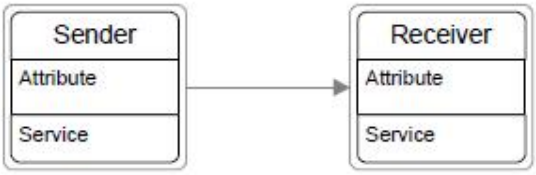
Service adalah tingkah laku yang spesifik, yaitu tingkah laku yang memperlihatkan "tanggungjawabnya" terhadap sebuah objek. Identifikasi uraian objek (*object state*):

1. Ujilah nilai yang potensial dari atribut
2. Bedakan apakah tanggungjawab sistem termasuk perbedaan kelakuan (*behavior*) untuk masing-masing nilai yang potensial
3. Periksa dengan hasil analisis terhadap permasalahan
4. Deskripsikan kedalam diagram objek (*object state diagram*)

Gambar 3.9 Simbol *Service*

Service ditempatkan dibagian bawah dari simbol **kelas-&-objek** dan **kelas**.

Message connection adalah model yang memproses ketergantungan dari sebuah objek, indikasi yang diperlukan untuk *service* dalam melaksanakan pekerjaan yang berhubungan dengan "tanggungjawabnya".



Gambar 3.10 Simbol *Message Connection*

Simbol *message connection* merupakan sebuah anak panah dari pengirim ke penerima. Anak panah berarti pengirim mengirim sebuah pesan, sedangkan penerima menerima pesan tersebut dan mengembalikan pesan tersebut setelah diproses terlebih dahulu. Biasanya *message connection* menghubungkan antar objek dari pada antar kelas.

Analisis

D. Spesifikasi S/W dan H/W : Easycase, Ms.Visio dan Pentium 4 atau Dual Core

E. Latihan Soal

1. Buatlah pemrograman berbasis objek mengacu dari rancangan class diagram berikut ini:

2. Lakukan Analisis system informasi untuk Akademik atau Perpustakaan, dan gambarkan siapa yang terlibat dalam perancangan system Akademik tersebut dengan menggunakan *Use Case Diagram dan Class diagram.*

MODUL 4
PEMODELAN UML

A. Tujuan Khusus

Mahasiswa dapat memahami dan menerapkan metodologi pemodelan dengan menggunakan UML (Unifield Markup Language) berbasis objek

B. Pertemuan

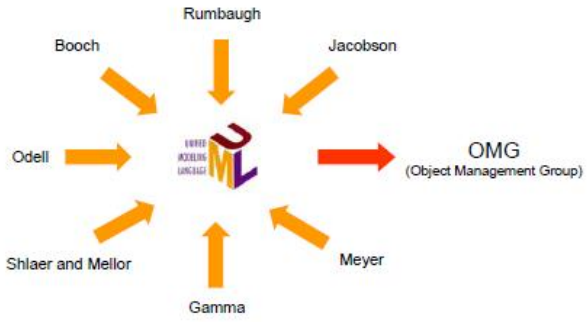
Pertemuan 7 dan 8

C. Teori Penunjang

Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa-bahasa berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C. Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax*/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD (Object-Oriented Design), Jim Rumbaugh OMT (Object Modeling Technique), dan Ivar Jacobson OOSE (Object-Oriented Software Engineering).

Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: metodologi booch [1], metodologi coad [2], metodologi OOSE [3], metodologi OMT [4], metodologi shlaer-mellor [5], metodologi wirfsbrock [6], dsb. Masa itu terkenal dengan masa perang metodologi (***method war***) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila

kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan.



Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease draft pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh **Object Management**

Group (OMG – http://www.omg.org). Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 1.5 yang dirilis bulan Maret 2003. Booch, Rumbaugh dan Jacobson menyusun tiga buku serial tentang UML pada tahun 1999 [7] [8] [9]. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek.

Konsepsi Dasar UML

Dari berbagai penjelasan rumit yang terdapat di dokumen dan buku-buku UML. Sebenarnya konsepsi dasar UML bisa kita rangkumkan dalam gambar dibawah.

Major Area	View	Diagrams	Main Concepts
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 value

Abstraksi konsep dasar UML yang terdiri dari *structural classification*, *dynamic behavior*, dan *model management*, bisa kita pahami dengan mudah apabila kita melihat gambar diatas dari *Diagrams*. *Main concepts* bisa kita pandang sebagai term yang akan muncul pada saat kita membuat diagram. Dan view adalah kategori dari diagaram tersebut.

Lalu darimana kita mulai ? 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

Tulisan ini pada intinya akan mengupas kedua hal tersebut. Seperti juga tercantum pada gambar diatas UML mendefinisikan diagramdiagram sebagai berikut:

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

4.1 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*

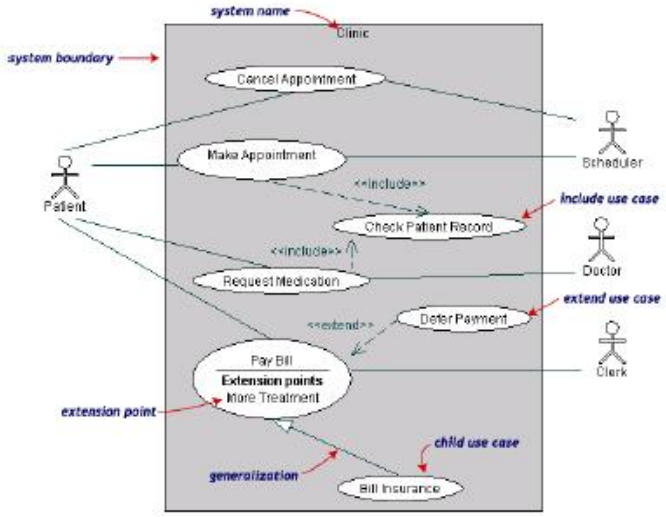
merepresentasikan sebuah interaksi antara actor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya.

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.

Contoh *use case diagram* :



4.2 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 :

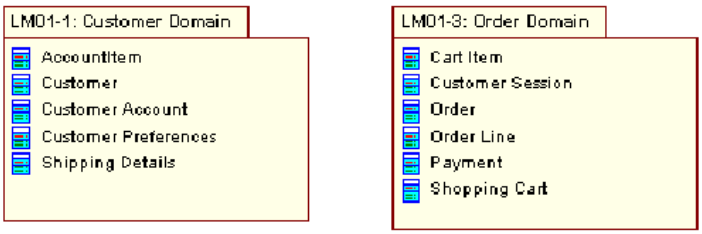
• *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan

• *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*.

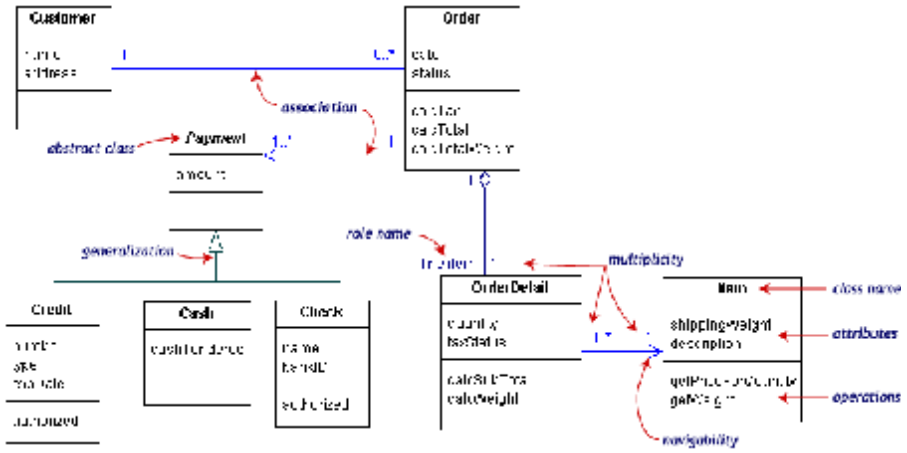
Sesuai dengan perkembangan *class* model, *class* dapat dikelompokkan menjadi *package*. Kita juga dapat membuat diagram yang terdiri atas *package*.



Hubungan Antar *Class*

1. 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*.
2. Agregasi, yaitu hubungan yang menyatakan bagian ("terdiri atas..").
3. 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.
4. 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.

Contoh *class diagram* :



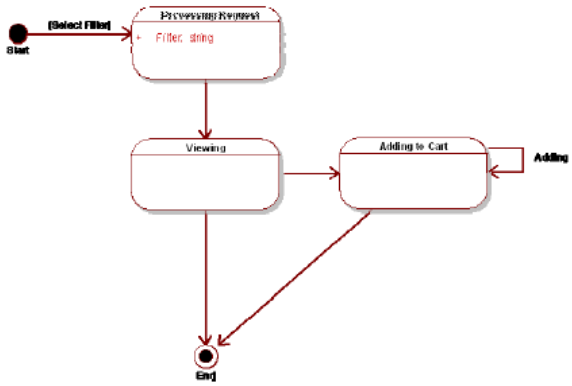
4.3 Statechart Diagram

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.

Contoh *statechart diagram*



4.4 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.

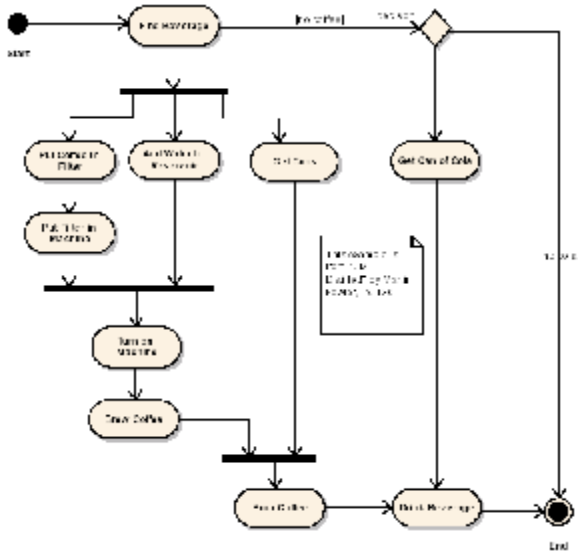
Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi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

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.

Contoh *activity diagram* tanpa *swimlane*:



4.5 Sequence Diagram

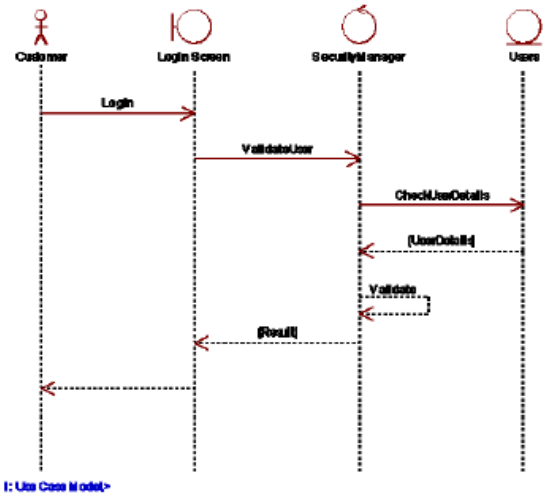
Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

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 men-*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*.

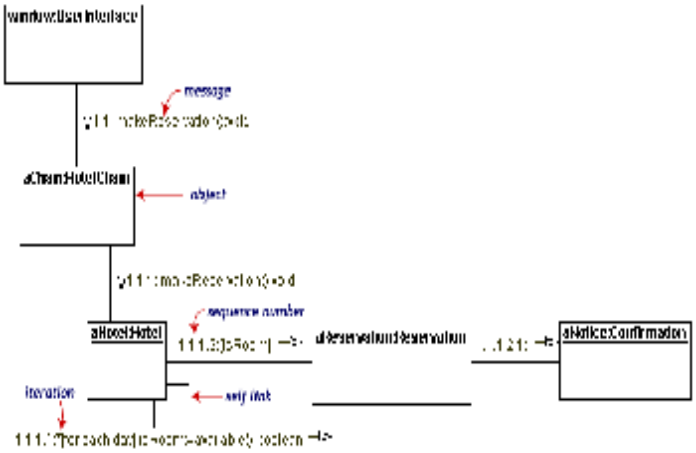
Activation bar menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah message. Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk objek *boundary*, *controller* dan *persistent entity*.

Contoh *sequence diagram* :



4.6 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.

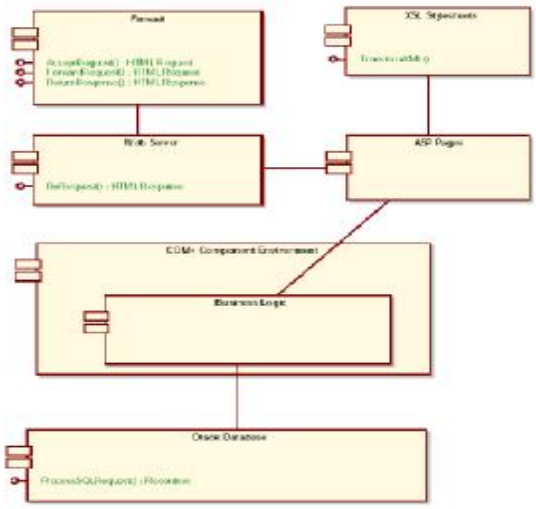


4.7 Component Diagram

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.

Contoh *component diagram*:

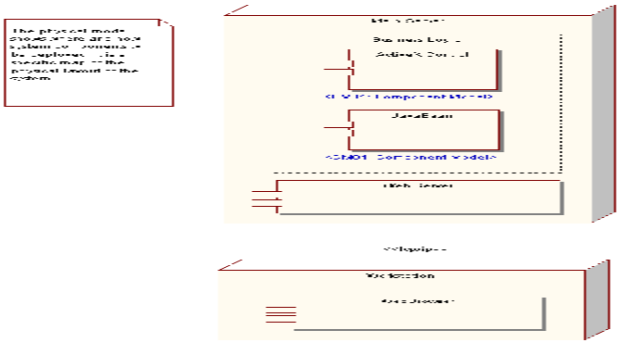


4.8 Deployment Diagram

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 fisikal

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.

Contoh *deployment diagram* :



D. Spesifikasi S/W dan H/W : Easycase, Ms.Visio dan Pentium 4 atau Dual Core

E. Latihan Soal

Kerjakanlah soal dibawah ini dengan mengacu pada studi kasus atau Perusahaan sehingga memudahkan dalam penjabaran.

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.

MODUL 5
USE CASE, ACTIVITY DAN SEQUENCES

A. Tujuan Khusus

Mahasiswa dapat menerapkan tools pemodelan dengan menggunakan Usecase Diagram, Activity Diagram, Sequences Diagram untuk perancangan sistem

B. Pertemuan

Pertemuan 9 dan 10

C. Teori Penunjang

1. 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.

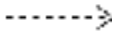
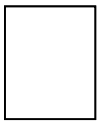
Sebuah use case dapat meng-include fungsionalitas use case lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa use case yang di-include akan dipanggil setiap kali use case yang meng-include dieksekusi secara normal.

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.

Simbol Use Case

NO	GAMBAR	NAMA	KETERANGAN
1		*Actor*	Menspesifikasikan himpuan peran yang pengguna mainkan ketika berinteraksi dengan *use case*.
2		*Dependency*	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri *(independent)* akan mempengaruhi elemen yang bergantung padanya elemen yang tidak mandiri (*independent*).
3		*Generalization*	Hubungan dimana objek anak (*descendent*) berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (*ancestor*).
4		*Include*	Menspesifikasikan bahwa *use case* sumber secara *eksplisit*.
5		*Extend*	Menspesifikasikan bahwa *use case* target memperluas perilaku dari *use case* sumber pada suatu titik yang diberikan.
6		*Association*	Apa yang menghubungkan antara objek satu dengan objek lainnya.
7		*System*	Menspesifikasikan paket yang menampilkan sistem secara terbatas.
8		*Use Case*	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu aktor
9		*Collaboration*	Interaksi aturan-aturan dan elemen lain yang bekerja sama untuk menyediakan prilaku yang lebih besar dari jumlah dan elemen-elemennya (sinergi).
10		*Note*	Elemen fisik yang eksis saat aplikasi dijalankan dan mencerminkan suatu sumber daya komputasi

2. 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.

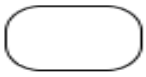
Activity diagram merupakan state diagram khusus, di mana sebagian besar state adalah action dan sebagian besar transisi di-trigger oleh selesainya state sebelumnya (internal processing). Oleh karena itu activity diagram tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) 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

Simbol Activity Diagram

NO	GAMBAR	NAMA	KETERANGAN
1		*Actifity*	Memperlihatkan bagaimana masing-masing kelas antarmuka saling berinteraksi satu sama lain
2		*Action*	State dari sistem yang mencerminkan eksekusi dari suatu aksi
3		*Initial Node*	Bagaimana objek dibentuk atau diawali.
4		*Actifity Final Node*	Bagaimana objek dibentuk dan dihancurkan
5		*Fork Node*	Satu aliran yang pada tahap tertentu berubah menjadi beberapa aliran

3. Sequences Diagram

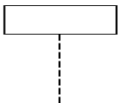
Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, display, dan sebagainya) berupa message yang digambarkan terhadap waktu. Sequence diagram terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

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 men-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.

Simbol Sequences Diagram :

NO	GAMBAR	NAMA	KETERANGAN
1		*LifeLine*	Objek *entity*, antarmuka yang saling berinteraksi.
2		*Message*	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktifitas yang terjadi
3		*Message*	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktifitas yang terjadi

D. Spesifikasi S/W dan H/W : Easycase, Ms.Visio dan Pentium 4 atau Dual Core

E. Latihan Soal

1. Dari modul 1 lanjutkan dengan memodelkan kasus tersebut kedalam beberapa diagram seperti Usecase, Activity , sequences sehingga terbentuk sistem informasi perpustakaan.

MODUL 6
FLOW DIAGRAM

A. Tujuan Khusus

Mahasiswa dapat memahami penggunaan Flow Diagram untuk perancangan sistem berbasis terstruktur atau manual

B. Pertemuan

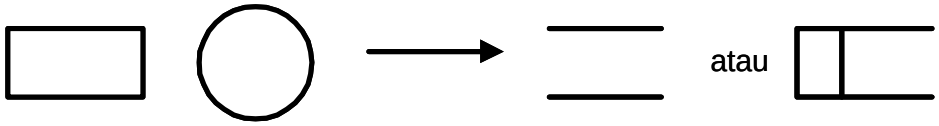
Pertemuan 11 dan 12

C. Teori Penunjang

DFD merupakan salah satu komponen dalam serangkaian pembuatan perancangan sebuah sistem komputerisasi. DFD menggambarkan aliran data dari sumber pemberi data (input) ke penerima data (output). Aliran data itu perlu diketahui agar si pembuat sistem tahu persis kapan sebuah data harus disimpan, kapan harus ditanggapi (proses), dan kapan harus didistribusikan ke bagian lain.

Komponen-komponen DFD

Komponen-komponen DFD terdiri atas :



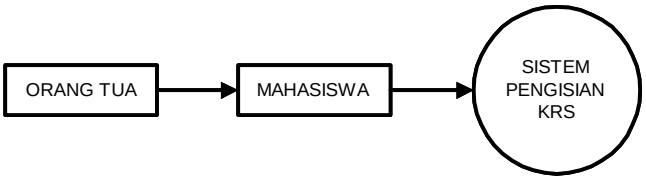
Terminator Proses Alur Data Penyimpan Data (data store)

Gambar 1. Komponen-komponen DFD

(1). Terminator

Terminator dapat disebut juga ‘Kesatuan Luar,’ yaitu suatu unit kerja/ jabatan, atau sejenisnya yang berada di luar sistem tetapi memberi andil atas pemberian atau penerimaan data dari sistem secara langsung. Terminator dapat pula disebut dengan ‘Sumber Pemberi Data (input),’ maupun ‘Tujuan Pemberian Data (output).

Pemberi data dan penerima data yang dimaksud adalah pihak yang sangat dekat dan memiliki hubungan langsung dengan sistem. Adapun pihak luar yang berhubungan dengan pihak luar lainnya tidak boleh digambarkan. Misalkan, dalam pengisian KRS, mahasiswa berhubungan dengan sistem. Orang tua berhubungan dengan mahasiswa, tetapi tidak berhubungan dengan sistem, karenanya, kesatuan luar ‘orang tua’, tidak boleh digambarkan.



Gambar 2. Contoh Hubungan Terminator yang Salah

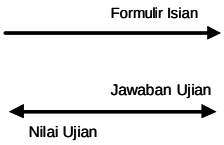
(2). Proses

Proses adalah suatu tindakan yang akan diambil terhadap data yang masuk. Karena proses adalah tindakan, maka proses berisi kata kerja, Proses diberikan identifikasi (nomor) agar mempermudah sekuen untuk diagram detilnya.

Gambar 3. Contoh Proses

(3). Alur Data

Alur data menggambarkan data yang mengalir dari terminator ke proses atau dari proses ke proses lainnya. Data yang dibawa oleh alur data harus disebutkan dan diletakkan di atas lambang alur data dan bila alur data digambar panjang, sebaiknya penulisan data mendekati lambang anak panahnya.



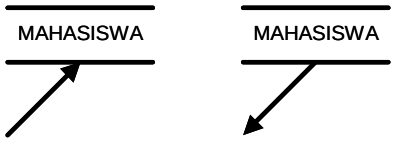
Gambar 4. Contoh Alur Data Searah dan Dua Arah

Data yang menempati alur data dapat berupa elemen data tunggal, maupun kumpulan elemen data. Misalkan, pada kumpulan elemen data : ‘Jawaban Ujian’, dapat ditulis secara lengkap dengan menyebutkan setiap elemen data yang ada di sana, yaitu : ‘Lembar Jawaban’, dan ‘Naskah Soal’.

(4). Penyimpan Data (Data Store)

Data yang akan disimpan perlu ditempatkan ke satu tempat penyimpanan data. Data yang disimpan dapat berupa data manual maupun data digital. Untuk data digital, penyimpan data tersebut kelak akan dijadikan file data di komputer. Alur data yang anak panahnya menuju penyimpan data, kegiatannya adalah ‘menulis/ merekam’ data, sehingga isi file data akan berubah karenanya. Sedangkan alur data yang anak panahnya menuju ke proses dari penyimpan data, kegiatannya adalah ‘membaca’ data, sehingga isi file data tidak akan berubah karenanya.

Penyimpan data harus diberi nama, misalkan data yang berisi biodata mahasiswa diberi nama ‘MAHASISWA’.



Gambar 5. Menulis dan Membaca data di Penyimpan Data

LEVELISASI DFD

DFD digambarkan secara bertingkat, dari tingkat yang global berturut-turut hingga tingkat yang sangat detil. Tingkat yang global (umum) disebut dengan ‘Diagram Konteks’ atau ‘Context Diagram’. Ini termasuk level 0.

Selanjutnya, dari diagram konteks, prosesnya dijabarkan lebih rinci lagi di ‘Diagram Nol’ atau ‘Zero Diagram.’ Ini disebut level 1. Pada diagram nol ini yang berkembang hanya proses dan alur data yang menghubungkan proses-prosesnya, sedangkan jumlah terminator dan alur data yang masuk atau keluar dari terminator, tetap.

Bila, masih dirasakan perlu memerinci proses berikutnya, maka diagram selanjutnya disebut dengan ‘Diagram Detil’ atau ‘Diagram primitif.’ Ini disebut dengan level 2. Dalam diagram detil, yang digambar cukup proses (nomor berapa) yang perlu didetilkan saja, selain itu (proses lainnya, atau terminatornya) tidak perlu digambarkan.

Contoh Kasus

Di sebuah tempat penyewaan *Video Compact Disk* (VCD), masih dilakukan pencatatan manual untuk Penyewaan dan pengembalian VCD oleh Penyewa. Dalam kasus ini, akan dirancang sistem komputerisasi Penyewaan (saja) VCD tersebut.

Analisis

1. Pihak-pihak yang terkait :

   a. Penyewa;
   b. Pemilik usaha;
   c. Petugas.

   Petugas berada di dalam sistem (yang menjalankan sistem), sehingga tidak perlu digambarkan. Dari sini, terdapat 2 terminator, yaitu a dan b.

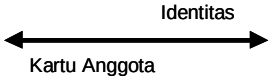
   1.a. Penyewa

   Data apa saja yang akan diberikan oleh Penyewa kepada sistem, dan data apa saja yang diberikan sistem kepada penyewa ?. Analisis ini bertujuan untuk menentukan data apa saja yang akan mengalir di alur data dari terminator Penyewa ke sistem (proses), dan sebaliknya.

   1.a.1. Penyewa Baru

   Penyewa baru (di kasus ini) harus membuat Kartu Anggota terlebih dulu. Pembuatan Kartu Anggota tidak dipungut biaya tetapi si Penyewa harus menunjukkan identitas diri (contoh : KTP).

   Petugas akan mencatat identitas Penyewa, membuatkan Kartu Anggota, dan bersama dengan KTP tersebut diserahkan kembali ke Penyewa.



Proses manual bahwa KTP tersebut dikembalikan ke Penyewa tidak harus digambarkan di dalam arus data.

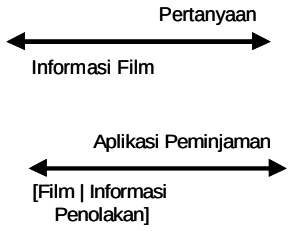
1.a.2. Prosedur Penyewaan oleh Penyewa

Penyewa yang akan meminjam film dipersilakan mencari sendiri filmnya, namun, bila mereka enggan mencarinya (tidak ketemu), mereka dapat langsung bertanya ke petugas. Petugas akan mengecek data film yang dicari dan akan dipinjam tersebut ke *file* di komputer. Hasil pengecekan itu diinformasikan kepada Penyewa.

Bila film dicari ada dan mereka mau meminjamnya, maka si Penyewa harus menyerahkan Kartu Anggotanya (di lapangan, bisa saja hanya dengan menyebutkan identitasnya saja), dan uang sewanya.

Adakalanya, petugas yang tidak yakin akan keanggotaan si Penyewa, dia melakukan cek keanggotaan ke *file* komputer. Bila ternyata data keanggotaannya tidak ada, maka si Petugas akan melakukan penolakan (pembatalan transaksi).

Bila benar anggota, maka Petugas akan mencatat data *film* yang dipinjam si Penyewa tersebut (transaksi) dan akan menyerahkan kembali Kartu Anggota dan film yang akan dipinjam tersebut ke Penyewa.

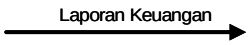


[Film | Informasi Penolakan] bisa ditulis : Film, Informasi Penolakan.

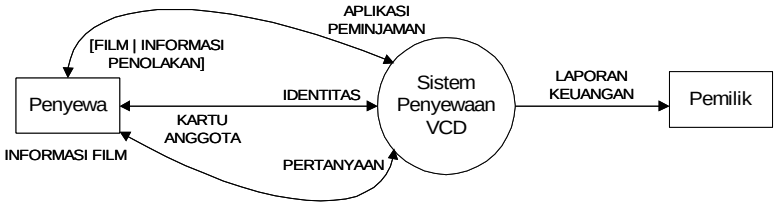
1.b. Pemilik Usaha (disingkat dengan Pemilik).

Apa saja data yang dibutuhkan oleh pemilik atas sistem, dan data apa saja yang diberikan oleh pemilik kepada sistem, perlu di analisis. Analisis ini akan menghasilkan alur data apa saja yang mengalir dari Terminator ke sistem dan sebaliknya.

Pada kasus ini, dicontohkan bahwa Pemilik hanya butuh laporan keuangan harian.



Dari analisis di atas, dapat dirancang DFD konteksnya :



Gambar 6. DFD Konteks Kasus di Atas

"Aplikasi Peminjaman" yang tergambar di atas bisa saja ditulis secara detil, misalkan Bukti Keanggotaan, Uang Sewa, dan Daftar Film yang akan Disewa. "Identitas" boleh saja ditulis [KTP|SIM].

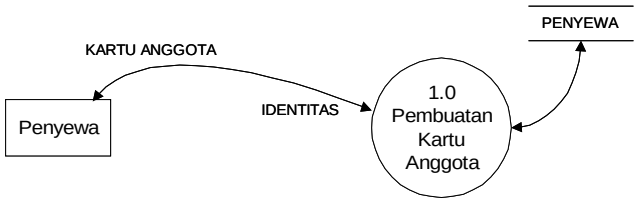
Sekali lagi, yang mengalir adalah data yang akan mempegaruhi proses komputerisasi, sedangkan untuk proses manualnya tidak perlu digambarkan. Misalkan, sewaktu akan meminjam film, Penyewa menyerahkan Kartu Anggota dan sewaktu menerima film, Kartu Anggota tersebut dikembalikan. Hal itu tidak perlu digambarkan.

2. Pembuatan Diagram Nol (*Level* 1)

Diagram Nol adalah pengembangan proses yang lebih mendetil dari proses (sistem) yang ada di konteksnya. Jadi, jumlah terminator dan alur data yang masuk dan keluar dari terminator harus tetap.

2.1. Proses Pembuatan Kartu Anggota

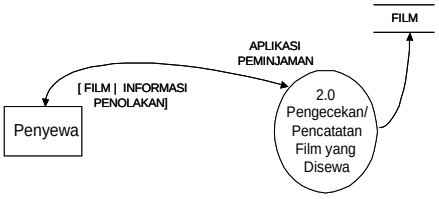
Lihat poin 1.a.1. di atas. Gambar DFD-nya :



Gambar 7. Penggalan Diagram Nol

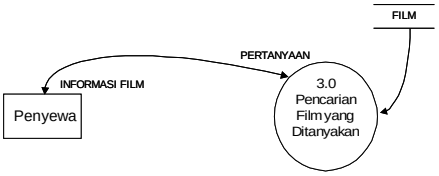
2.2. Proses Penyewaan VCD

Lihat poin 1.a.2. di atas. DFD-nya akan digambarkan sebagai :



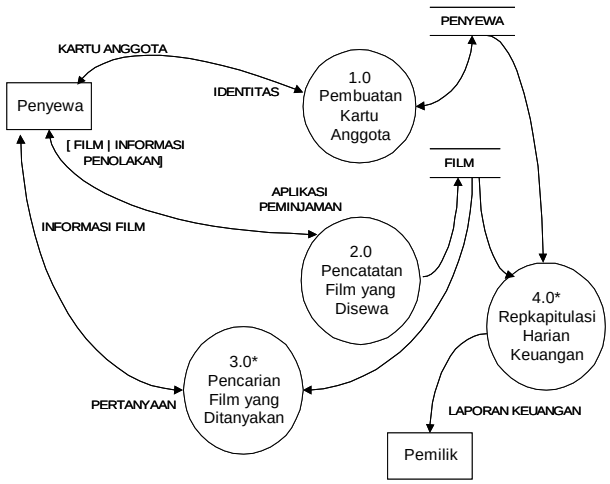
Gambar 8. Penggalan Diagram Nol

2.3. Proses Permintaan Informasi Keberadaan Film



Gambar 9. Penggalan Diagram Nol

2.4. Gambar DFD *Zero* (*level* 1) Lengkapnya



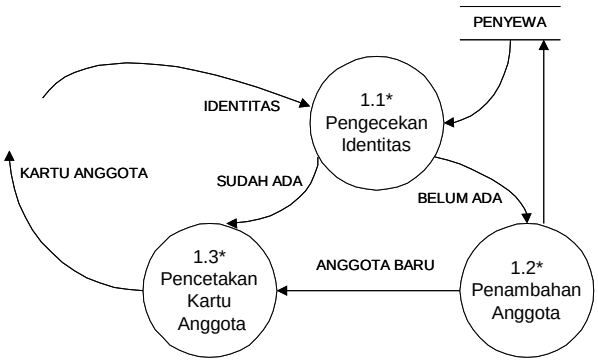
Gambar 10. DFD *Level* 1 Kasus di Atas

Beberapa catatan tambahan :

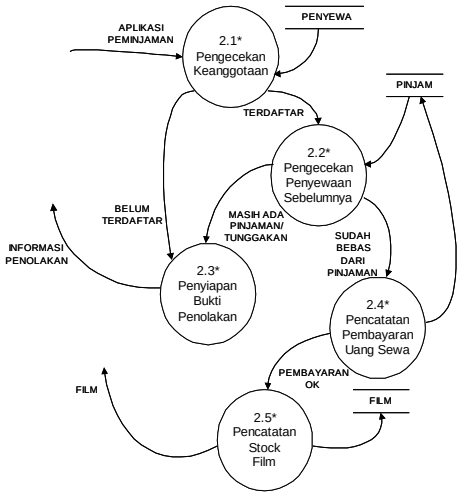
(1) Pembuatan rancangan DFD harus sesuai dengan prosedur yang berlaku di tempat penelitian (jadi harus ada pembahasan mengenai prosedur yang berlaku, dan prosedur tersebut bukan penguji yang menentukan);
(2) Penggambaran DFD hendaknya dibuat sebaik mungkin (mudah ditelusuri, dan tidak rumit, misalkan dengan tidak adanya alur data yang bersilangan).
(3) Bila akan terjadi persilangan alur di penyimpan data, maka penyimpan data tersebut dapat digambar kembali dan diberi tanda '*' yang menandakan bahwa penyimpan data tersebut sama dengan nama penyimpan data sebelumnya (*copy*).
(4) Tanda '*' di nomor proses berarti proses tersebut tidak perlu didetilkan lagi.

3. Pembuatan Diagram Detil (*level* 2)

Diagram detil perlu digambarkan bila masih ada suatu proses yang bisa dirinci lebih lanjut. Di sini dimisalkan penggambaran dari proses 1.0 (Pembuatan Kartu Anggota).



Gambar 11. Diagram 1.0 Level 2



Gambar 12. Diagram 2.0 *Level* 2

D. Spesifikasi S/W dan H/W : Easycase, Ms.Visio dan Pentium 4 atau Dual Core

E. Latihan Soal

1. Buatlah diagram DFD untuk system yang akan dirancang dengan mengacu pada analisis pada modul sebelumnya.

MODUL 7
User Interface dan Persistance

A. Tujuan Khusus

Mahasiswa dapat memahami dan mengimplementasikan rancangan tampilan dengan menggunakan pemodelan dan teknik desain interface.

B. Pertemuan

Pertemuan 13

C. Teori Penunjang

A. 12 Teknik Antar Muka yang baik

1. Penebalan (Highlight)

Salah satu elemen yang paling penting dari sebuah antarmuka pengguna yang baik adalah visibilitas status sistem. Pengguna harus melihat langsung apa yang terjadi di balik layar dan apakah tindakan mereka telah benar-benar menyebabkan hasil yang diharapkan. Untuk mencapai tingkat yang lebih canggih visibilitas sistem

2. Gunakan Shortcut pada Keyboard

Sebagai fitur canggih aplikasi Web modern (seperti menyeret dan menjatuhkan, jendela modal, dll) terus mendapatkan pada orang-aplikasi desktop, pengembang aplikasi ini mencoba untuk menawarkan pengguna antarmuka pengguna yang lebih responsif dan interaktif. Salah satu teknik yang digunakan untuk mencapai ini adalah integrasi dari cara pintas keyboard atau navigasi. Sama seperti dengan aplikasi klasik, fitur ini sedikit signifikan dapat meningkatkan alur kerja pengguna Anda dan membuat lebih mudah bagi mereka untuk mendapatkan tugas-tugas mereka selesai.

3. Meng-Upgrade dari account halaman pilihan

Jika aplikasi Anda memiliki beberapa fitur , pastikan untuk menghilangkan gesekan antarmuka bagi pelanggan untuk memutuskan untuk meng-upgrade. Sebagian besar pengguna ingin mencoba versi dasar aplikasi pertama untuk mendapatkan rasa lebih baik dari apa yang ditawarkan ,Jika mereka yakin aplikasi memenuhi harapan mereka,.

4. Iklan fitur aplikasi

Meskipun kita membuat sebuah halaman pemasaran rinci, menguraikan aplikasi Anda setiap fitur, dan menciptakan sebuah bagian bantuan menyeluruh di situs kita, pengguna kita mungkin tidak semuanya dibaca. Mereka mungkin tidak akrab dengan semua fitur dari produk Anda dan akan mendapat manfaat dari tips kecil di dalam aplikasi itu sendiri

5. Gunakan Warna sebagai Kode

Beberapa fitur aplikasi feed yang berbagai jenis agregat konten. Sebagai contoh, Anda mungkin memiliki aplikasi manajemen proyek yang menunjukkan Anda semua terbaru, tugas dan file pada halaman rumah. Jika barang-barang ini semua muncul bersama-sama dalam satu daftar, mungkin sulit untuk mengatakan apa apa. Banyak aplikasi menggunakan kode warna untuk membantu secara visual membedakan antara berbagai jenis entri. Sebuah cara sederhana untuk melakukan ini adalah untuk menempatkan label teks di dalam kotak berwarna

6. Tawarkan Pilihan personalisasi

Banyak aplikasi kustom untuk menyediakan ruang kerja orang dan bisnis. Personalisasi dapat membantu membuat pengguna Anda merasa lebih di rumah. Hal ini dapat dilakukan dengan memberikan pengguna pilihan untuk menyesuaikan tampilan dan nuansa dari antarmuka aplikasi. Biarkan mereka memilih tema warna, warna link, latar belakang dan sebagainya. Bahkan sejumlah kecil kustomisasi akan memungkinkan pengguna Anda untuk membuat halaman mereka sendiri

7. Tampilkan pesan menarik untuk membantu

Setiap aplikasi Web yang berbeda dan memiliki cara sendiri dalam melakukan sesuatu. Jika fungsi dari elemen tertentu tidak langsung terlihat, Anda dapat memberikan pesan bantuan singkat untuk bisa orang-orang mulai. Satu hal penting untuk dicatat adalah bahwa jika Anda ingin membantu orang yang tidak yakin apa yang mereka lakukan nya, Anda perlu untuk menarik perhatian mereka ke pesan ini. Salah satu cara untuk menarik perhatian dengan warna - menempatkan kuning "lengket" pesan.

8. Desain pesan umpan balik dengan hati-hati

Sebuah praktik yang baik di sini adalah untuk melakukan beberapa hal.

1. Pertama, kode warna berbagai jenis pesan.
2. Pesan yang memberitahu pengguna dari tindakan sukses biasanya berwarna hijau. Ini menggunakan analogi lampu lalu lintas makna hijau "Pergilah." Peringatan dan pesan kesalahan yang berwarna kuning. Sama analogi lampu lalu lintas di sini: kuning berarti memperlambat dan menunggu. Anda juga dapat membedakan antara pesan peringatan dan pesan kesalahan dengan mewarnai peringatan kesalahan merah dan kuning.

9. Gunakan tab navigasi

Banyak aplikasi Web telah mengadopsi pendekatan navigasi tab untuk menu navigasi utama mereka. Navigasi tab adalah menu yang terlihat seperti setiap item tab pada folder file, dengan tab aktif terhubung ke tubuh halaman. Tab navigasi tidak hanya permen mata; itu memberikan manfaat kegunaan.

- 10. Menggelapkan latar belakang bawah jendela modal
- 11. Light Boxes and Slideshows
- 12. Short sign-up forms

Bentuk sign-up yang berpotensi salah satu hambatan terbesar antara Anda dan pelanggan potensial. Semakin lama bentuk, upaya lebih banyak pengunjung Anda akan harus membuat sebelum menjadi anggota dari situs Anda atau, mungkin, pelanggan yang membayar. Untuk meminimalkan penghalang, kita harus mempercepat proses. Ini berarti menghapus semua elemen opsional dari bentuk dan hanya menyisakan penting inti. Hal-hal opsional dapat diisi nanti.

B. Perancangan Tampilan

1. Story board

Apa yang dimaksud dengan Story board?

Storyboard

Adalah kolom teks, audio dan visualisasi dengan keterangan mengenai content dan visualisasi yang digunakan untuk produksi sebuah course. Derajat storyboard bisa berbeda karena ada berbagai tahap yang harus di lalui sesuai tujuan pembuatan story board tersebut.

Storyboard merupakan konsep komunikasi dan ungkapan kreatif, teknik dan media untuk menyampaikan pesan dan gagasan secara visual, termasuk audio dengan mengolah elemen desain grafis berupa bentuk dan gambar, huruf dan warna, serta tata letaknya, sehingga pesan dan gagasan dapat diterima oleh sasarannya. Storyboard juga tidak terbatas hanya pada pembuatan iklan saja karena produksi game, cd multimedia dan elearningpun menggunakan story board.

Apa yang harus diperhatikan pada penulisan storyboard?

Prinsip penulisan storyboard

Pesan visual harus kreatif (asli, luwes dan lancar), komunikatif, efisien dan efektif, sekaligus indah/ estetis

Konsep, Strategi dan Proses Perancangan Grafis

KONSEP 5 W + 1 H =‘What, Why, Who, Which, Where, How.’

1. Materi pembelajaran dan pesan apa yang akan disampaikan
2. Apa saja jenis dan cakupan materi pembelajaran
3. Apa keunggulannya dan bagaimana konsep membawakannya

5. Kepada siapa materi tersebut diperuntukkan.

6. Bagaimana cara pendekatan dengan audience

7. Apa peluang dan target dari pembelajaran tersebut

8. Apa yang diperlukan untuk menggali potensi audience

9. Kebiasaan, pola dan cara masyarakat dalam belajar

10. Pendekatan komunikasi dan kreatif apa yang tepat untuk itu

STRATEGI

Strategi diperlukan dalam upaya proses menyampaikan pesan secara efektif dan efisien.

Cara yang biasa dipergunakan yaitu :

1. Merancang Strategi Komunikasi

2. Menyusun Strategi Kreatif

PROSES PERANCANGAN

Proses perancangan selalu dimulai dengan penelitian yaitu:

1. **Scanning, data collecting/pengumpulan data**, sebagai bahan dasar untuk dianalisa. Data berupa data tertulis (verbal), dan data gambar (visual), atau data lainnya seperti suara (audio), data teraba (bentuk 3 dimensi) dan aroma atau rasa (kecap).

2. Formulasi, data dasar dianalisa untuk **proses pemilahan, pengelompokkan (klasifikasi),** lalu dirumuskan.

Hasil rumusan tersebut merupakan bahan penyusunan :

- Konsep Umum, lebih ditekankan pada konsep komunikasinya
- Konsep Kreatif, lebih ditekankan pada konsep kreatifnya.

3. **Implementasi**

Adalah **perwujudan visual (visualisasi)** kreatif ke dalam media yang telah dipilih berdasar pada kesesuaian dengan visi, misi, maksud,tujuan, sasaran pesan agar efisien, efektif, komunikatif serta keindahannya. Pada proses implementasi ini diperlukan strategi serta pemikiran proses produksi media dan penerapan pada media serta penyebarannya, serta pemasangan di lokasi yang tepat (strategis).

4. Biasanya dilakukan **pretest (uji coba sebelum storyboard** yang Anda tulis dituangkan dalam bentuk visual dan audio.

Konsep desain storyboard yang baik adalah:

Konsep yang mampu memberikan jawaban/jalan keluar terhadap problem-problem yang ada sesuai dengan kebutuhan SME/audience. Ini menggunakan riset, eksperimentasi, kritik, dan

analisa. Dari segi pendekatan visual maupun copywriting mampu menarik khalayak untuk melihat, mengerti dan kemudian mengambil tindakan yang diharapkan sebenar-benarnya.

Storyboard yang baik harus bisa menjawab pertanyaan

1.Apa yang sebenarnya ingin dicapai?

2.Berapa lama tujuan tersebut akan dapat dicapai?

3.Apa strategi yang paling cocok untuk mencapai tujuan tersebut ?

Yang harus diperhatikan setiap kali menyerahkan storyboard pada setiap tahap

1. Apakah **struktur content** sudah sesuai dengan ekspektasi SME

2. Apakah content setiap page sudah **merepresentasikan tujuan pembelajaran**
dari SME

3. Apakah kolom content sudah **typo error free dan gramatical error free**

4. Content mudah dipahami dan dipelajari

5. Ide visualisasi sudah merepresentasikan pesan yang ingin disampaikan

6. Kesesuaian kolom content, audio dan visual

7. Apakah semua modul dalam satu course sudah konsisten sesuai standard yang disepakati, seperti jumlah page, feedbacknya, assesment

8. Setiap perubahan harus dicatat dan disimpan dengan penamaan file berdasarkan tanggal agar **control version** bisa diketahui.

Masalah yang sering muncul saat bekerja dengan GD

1. Bekerjalah dengan graphic GD anda secara **hati-hati**. Lihatlah karya-karya mereka sebelumnya. GD terbaik tidak memiliki "style yang terlihat". Karya mereka akan terlihat berbeda.

2. Tinggalkan praduga yang belum tentu benar di pintu depan. Terkadang klien/audience anda meminta sebuah course yang seperti milik orang lain tetapi dalam warna yang berbeda. Anda harus **terbukalah terhadap ide** yang baru dan diluar dugaan. Jangan takut pada sesuatu yang berbeda. Biarkanlah ide-ide baru mengalir.

3. **Katakan pada GD apa yang ingin anda capai,** bukannya bagaimana anda menginginkan sesuatu terlihat. Jangan meminta sebuah warna, bentuk, atau style. Mintalah arti atau emosi.

4. Pastikan tentang **fitur spesifik yang anda butuhkan**, misalnya untuk materi yang membutuhkan gambar formulir, bagan, skema untuk pendukung visualisasi. Anda ingin GD anda untuk membuat sebuah spesifikasi desain yang anda inginkan. Jika anda mencoba untuk menambahkan fitur selama perjalanan, desainnya tidak akan cocok.

5. Kerjakan riset anda dan **jelaskan secara spesifik tentang kebutuhan anda**.

Semakin detil dan spesifik saat permulaan, semakin baik daya kerja GD untuk membuatkan visualisasi untuk kebutuhan anda. Jika anda menambahkan kebutuhan anda di kemudian hari, sang GD kemungkinan hanya bisa untuk menyelipkannya, dimana ini tidak akan memberikan anda hasil yang terbaik.

6. Pastikan **pesan dan isi anda telah jelas.** Semakin siap isi yang ingin anda masukkan, semakin baik GD dapat membuat storyboard anda. Seorang GD yang bagus dapat membuat usulan untuk memperjelas isi content anda agar pesan yang ingin anda sampaikan dapat dipahami lebih cepat dan lebih jelas. Tetapi semakin banyak isi content yang telah siap, semakin banyak yang bisa dikerjakan oleh GD.

7. **Desainlah untuk SME anda dan audiencenya,** bukan untuk anda sendiri, teman anda, atau kolega anda. Jelaskan secara spesifik sehingga GD anda tahu bagaimana pelanggan anda dan apa yang mereka inginkan. Adalah lebih penting jika mereka menyukai course anda daripada anda sendiri. Selalu ingatlah "Apa yang diinginkan mereka". Jika desainnya meyenangkan hati audience anda, mereka akan meyenangkan anda. Jika anda bertahan pada desain yang hanya menyenangkan hati

anda, maka audience pun mungkin tidak akan tertarik untuk belajar.

8. Milikilah alasan yang baik untuk pilihan anda. Anda dapat menunjukkan pada GD contoh course yang menarik hati anda, tetapi galilah lebih dalam dan jelaskanlah mengapa course tersebut menarik hati anda. Berpikirlah dalam konteks perasaan.

Desain membuat anda merasakan senang, jadi beritahu GD anda bagaimana desain tersebut mewakili perasaan audience anda. Daripada berkata "Saya suka warna kuning", galilah ke akarnya dan berkata "Saya ingin course yang terasa hangat" atau "Saya ingin sesuatu yang riang dan bersahabat." Berfokus pada kesan logis dan emosional anda akan memberikan GD hal yang lebih banyak untuk dikerjakan.

Kenapa ? Karena audience anda mungkin tidak "suka" hal yang sama dengan anda, tetapi GD yang baik dapat menterjemahkan kesan yang ada ingin didapatkan audience anda.

9. **Jangan mendesain dengan komite.** Tidak ada desain yang bagus yang pernah tercipta dari kesepakatan bersama. Semakin banyak orang yang memiliki suara dalam proses, semakin bias desain yang dihasilkan. Teman-teman dan rekan-rekan kerja anda akan sering memberikan nasehat yang bertentangan dan orang seringkali memiliki motif tertentu saat memberikan komentar (mereka mungkin iri atau terintimidasi jika anda mendapatkan sesuatu yang terlalu bagus, atau mereka mungkin hanya tidak peduli). Anda dapat menunjukkannya pada beberapa orang yang terpercaya dan mendapatkan komentar mereka, tetapi hanya boleh satu orang yang

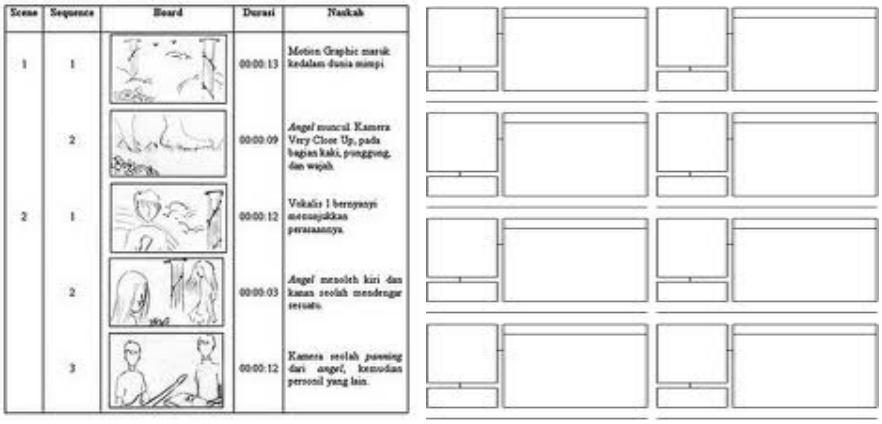
mengambil keputusan. Jangan terombang-ambing dan mencoba untuk mengubah arah saat-saat terakhir dalam proses.

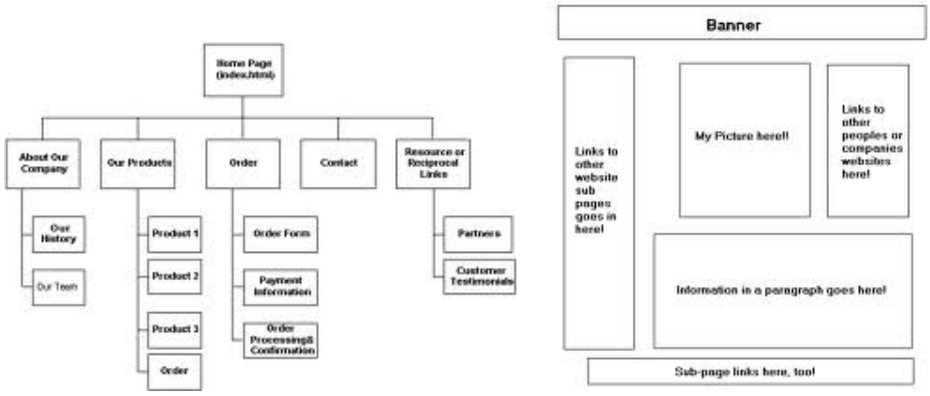
10. **Jangan memberitahu GD bagaimana caranya mendesain.** Itu bukan keahlian anda. Berikan GD kebutuhan dan catatan anda, tapi juga beri mereka kebebasan untuk menciptakan sesuatu yang menjawab semua itu seefektif mungkin. Jika anda terlalu mengatur GD, mereka tidak akan termotivasi untuk melakukan apapun kecuali menguangkan cek anda.

11.**Percayai GD anda.** Maka ketika mereka mulai menunjukkan desainnya, berikan komentar yang spesifik. Jangan hanya berkata, "Saya tidak suka warna coklat." Itu tidak menunjukkan nilai yang sebenarnya. Jika anda berkata "Saya khawatir warnanya terlihat muram dan kita butuh sesuatu yang menunjukkan pertumbuhan," maka anda memberikan sesuatu yang berguna kepada GD, karena anda berbicara tentang isinya dan bukan menyuruh mereka bagaimana mendesain sesuatu.

12. Anda tidak dapat menyenangkan semua orang setiap waktu. Bill Cosby pernah berkata "Satu-satunya jalan yang pasti untuk gagal adalah mencoba untuk menyenangkan hati semua orang." Jika semua berpikir situs anda "OK", maka kemungkinan situs anda terlalu membosankna untuk mendapatkan reaksi dari seseorang. Jika Anda mendesain sebuah situs TANPA kepribadian, tidak ada orang yang membencinya atau menyukainya.

Contoh Storyboard:





C. Spesifikasi S/W dan H/W : Easycase, Ms.Visio dan Pentium 4 atau Dual Core

D. Latihan Soal :

Kerjakanlah perancangan dari Tampilan dari kasus-kasus sebelumnya dengan memperhatikan 12 teknik antar muka yang baik:

1. Buatlah tampilan untuk user dan Admin
2. Tampilan untuk form masukan dan form keluaran