

## **DAFTAR ISI**

BAB I REKAYASA PERANGKAT LUNAK.....	2
BAB II MODEL-MODEL PROSES PERANGKAT LUNAK .....	7
BAB III ANALISIS SISTEM .....	16
BAB IV DAD (DIAGRAM ALIRAN DATA) .....	20
BAB V BAGAN ALIR (FLOWCHART) .....	28
BAB VI UML (UNIFIED MODELLING LANGUAGE .....	33
BAB VII MACAM-MACAM DIAGRAM .....	38

# **BAB I**

## **REKAYASA PERANGKAT LUNAK**

### **Tujuan Praktikum :**

1. Pengenalan Sistem Komputer dan Perangkat Lunak
2. Pengenalan Rekayasa Perangkat Lunak dan Aplikasinya

### **Indikator :**

- ❖ Agar praktikan bisa menjelaskan komponen sistem komputer
- ❖ Agar praktikan bisa menjelaskan pengertian Perangkat Lunak dan contohnya
- ❖ Agar praktikan bisa menyebutkan dan menjelaskan contoh jenis dan aplikasi perangkat lunak

### **Materi :**

#### **Definisi Komputer**

Komputer merupakan suatu perangkat elektronika yang dapat menerima dan mengolah data menjadi informasi, menjalankan program yang tersimpan dalam memori, serta dapat bekerja secara otomatis dengan aturan tertentu.

#### **Sistem Komputer**

Sebuah sistem komputer tersusun atas tiga elemen, yaitu

1. Hardware (Perangkat Keras), merupakan rangkaian elektronika
2. Software (Perangkat Lunak), merupakan program yang dijalankan pada komputer
3. Brainware (SDM)

Pada praktikum Rekayasa Perangkat Lunak ini, sebelumnya kita harus mengerti apa yang dimaksud dengan Perangkat Lunak. Perangkat lunak kini sudah menjadi kekuatan yang menentukan. Perangkat lunak menjadi mesin yang mengendalikan pengambilan keputusan di dalam dunia bisnis, berfungsi sebagai dasar dari semua bentuk pelayanan serta penelitian keilmuan modern. Saat ini perangkat lunak memiliki dua peran. Di satu

sisi berfungsi sebagai sebuah produk, dan di sisi lain sebagai kendaraan yang mengantarkan sebuah produk.

Sebenarnya, apa yang dimaksud dengan **Perangkat Lunak**?

Perangkat Lunak (*Software*) adalah (1) *Perintah (program komputer) yang bila dieksekusi memberikan fungsi dan unjuk kerja seperti yang diinginkan.* (2) *Struktur data yang memungkinkan program memanipulasi informasi secara proporsional.* (3) *Dokumen yang menggambarkan operasi dan kegunaan program.*

Sedangkan yang dimaksud dengan **Rekayasa Perangkat Lunak** adalah *Suatu disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal requirement capturing (analisa kebutuhan pengguna), specification (menentukan spesifikasi dari kebutuhan pengguna), desain, coding, testing sampai pemeliharaan sistem setelah digunakan.*

Jadi yang perlu digaris bawahi, Rekayasa Perangkat Lunak merupakan serangkaian proses yang amat panjang untuk membuat atau menciptakan suatu perangkat lunak, bukan merupakan cabang ilmu *Computer Science* yang mempelajari tentang *technical coding*.

### **Klasifikasi Perangkat Lunak :**

1. Sistem Operasi, merupakan perangkat lunak yang mengoperasikan komputer serta menyediakan antarmuka dengan perangkat lunak lain atau dengan pengguna. Contoh sistem operasi : MS DOS, MS Windows (dengan berbagai generasi), Macintosh, OS/2, UNIX (dengan berbagai versi), LINUX (dengan berbagai distribusi), NetWare, dll
2. Program Utilitas, merupakan program khusus yang berfungsi sebagai perangkat pemeliharaan komputer, seperti anti virus, partisi hardisk, manajemen hardisk, dll. Contoh produk program utilitas : Norton Utilities, PartitionMagic, McAfee, dll
3. Program Aplikasi, merupakan program yang dikembangkan untuk memenuhi kebutuhan yang spesifik. Contoh : aplikasi akuntansi, aplikasi perbankan, aplikasi manufaktur, dll.

4. Program Paket, merupakan program yang dikembangkan untuk kebutuhan umum, seperti : - pengolah kata /editor naskah : Wordstar, MS Word, Word Perfect, AmiPro, dll - pengolah angka / lembar kerja : Lotus123, MS Excell, QuattroPro, dll - presentasi : MS PowerPoint, dll - desain grafis : CorelDraw, PhotoShop, dll.
5. Bahasa Pemrograman, merupakan perangkat lunak untuk pembuatan atau pengembangan perangkat lunak lain.

### **Aplikasi Perangkat Lunak**

Perangkat Lunak dapat diaplikasikan ke berbagai situasi dimana serangkaian langkah prosedural telah didefinisikan. Area Perangkat Lunak berikut menunjukkan luasnya aplikasi potensial :

- **Perangkat Lunak Sistem.**

Merupakan sekumpulan program yang ditulis untuk melayani program-program yang lain. Di dalam setiap kasus, area perangkat lunak sistem ditandai dengan eratnya interaksi dengan perangkat keras komputer; penggunaan oleh banyak pemakai; operasi konkuren yang membutuhkan penjadwalan-tukar-menukar sumber-dan pengaturan proses yang canggih; struktur-struktur data yang kompleks; serta interface eksternal ganda.

- **Perangkat Lunak Real-Time.**

Program-program yang memonitori/menganalisa/mengontrol kejadian dunia nyata pada saat terjadinya disebut perangkat lunak *real-time*. Elemen-elemen perangkat lunak *real-time* mencakup komponen pengumpul data yang mengumpulkan dan memformat informasi dari lingkungan eksternal, sebuah komponen analisa yang mentransformasi informasi pada saat dibutuhkan oleh aplikasi. Real-time berbeda dengan interaksi atau *timesharing*. Sistem real-time harus merespon di dalam suatu rentang waktu yang tetap. Waktu respon sebuah sistem interaktif (atau timesharing) secara normal dapat diperpanjang tanpa memberikan resiko kerusakan pada hasil.

- **Perangkat Lunak Bisnis.**

Pemrosesan informasi bisnis merupakan area aplikasi perangkat lunak yang paling luas. Sistem diskrit (contohnya payroll, account payable, inventory, dll)

telah mengembangkan perangkat lunak sistem informasi manajemen (SIM) yang mengakses satu atau lebih database besar yang berisi informasi bisnis.

- **Perangkat Lunak Teknik dan Ilmu Pengetahuan.**

Perangkat lunak teknik dan ilmu pengetahuan ditandai dengan algoritma *number crunching*. Perangkat lunak ini memiliki jangkauan aplikasi mulai dari aplikasi mulai dari astronomi sampai vulkanologi, dari analisa otomotif sampai dinamika orbit pesawat ruang angkasa, dan dari biologi molekuler sampai pabrik yang sudah diotomatisasi. Computer-aided design, simulasi sistem, dan aplikasi interaktif yang lain, sudah mulai memakai ciri-ciri perangkat lunak sistem genap dan real-time.

- **Embedded Software.**

Produk pintar telah menjadi bagian yang umum bagi hampir semua konsumen dan pasar industri. Embedded software dapat memberikan fungsi yang terbatas serta fungsi esoteris (misal *keypad control* untuk microwave) atau memberikan kemampuan control dan fungsi yang penting (contohnya fungsi digital dalam sebuah automobile seperti control bahan bakar, penampilan dashboard, sistem rem, dll)

- **Perangkat Lunak Komputer Personal.**

Pasar perangkat lunak komputer personal telah berkembang selama decade terakhir. Program pengolah kata, *spreadsheet*, multimedia, manajemen database, aplikasi keuangan bisnis dan personal, jaringan eksternal atau akses database hanya merupakan beberapa saja dari ratusan aplikasi yang ada.

- **Perangkat Lunak Kecerdasan Buatan.**

Perangkat lunak kecerdasan buatan (*Artificial Intelligent* (AI)) menggunakan algoritma non-numeris untuk memecahkan masalah kompleks yang tidak sesuai untuk perhitungan atau analisa secara langsung. Area kecerdasan buatan yang aktif adalah sistem pakar, disebut juga sistem berbasis pengetahuan. Sistem yang lain adalah Jaringan Syaraf Tiruan, *Voice and Image Recognition*, *game playing* dsb.

**TUGAS :**

Pilihlah sebuah aplikasi khusus (selain Sistem Informasi) dan tunjukkan

- (a) kategori dari aplikasi perangkat lunak
- (b) isi data (content) yang berhubungan dengan jenis aplikasi tersebut.

## **BAB II**

### **MODEL-MODEL PROSES PERANGKAT LUNAK**

#### **Tujuan Praktikum :**

1. Tahapan-tahapan pengembangan perangkat lunak
2. Mengenalkan model-model dalam proses perangkat lunak

#### **Indikator :**

- ❖ Agar praktikan bisa menjelaskan fase-fase dalam merekayasa perangkat lunak
- ❖ Agar praktikan bisa membedakan masing-masing model proses perangkat lunak dan mengaplikasikannya pada suatu kasus aplikasi perangkat lunak.

#### **Materi :**

#### **1. Pandangan umum tentang Rekayasa Perangkat Lunak**

Usaha yang berhubungan dengan Rekayasa Perangkat Lunak dapat dikategorikan ke dalam tiga fase umum dengan tanpa mempedulikan area aplikasi, ukuran proyek atau kompleksitasnya. Masing-masing fase akan memberi tekanan pada pertanyaan-pertanyaan yang sudah ditulis diatas :

- Fase Definisi (*Definition Phase*) berfokus pada “apa” (what), dimana pada definisi ini pengembang perangkat lunak harus mengidentifikasi informasi apa yang akan diproses, fungsi dan unjuk kerja apa yang dibutuhkan, tingkah laku system seperti apa yang diharapkan, interface apa yang akan dibangun, batasan desain apa yang ada dan kriteria validasi apa yang dibutuhkan untuk mendefinisikan sistem yang sukses. Kebutuhan (requirement) kunci dari sistem dan perangkat lunak yang didefinisikan. Metode yang diaplikasikan selama fase definisi berbeda, tergantung pada paradigma rekayasa perangkat lunak (atau kombinasi paradigma) yang diaplikasikan.
- Fase Pengembangan (*Development Phase*) berfokus pada *how* (bagaimana), yaitu dimana selama masa pengembangan perangkat lunak, teknisi harus mendefinisikan bagaimana data dikonstruksikan, bagaimana detil prosedur akan diimplementasikan, bagaimana interface ditandai dll.

- Fase Pemeliharaan (*Maintenance Phase*) berfokus pada perubahan yang dihubungkan dengan koreksi kesalahan, penyesuaian yang dibutuhkan ketika lingkungan perangkat lunak berkembang, serta perubahan sehubungan dengan perkembangan yang disebabkan oleh perubahan kebutuhan pelanggan.

## 2. Fase Definisi :

### Proses Requirements Engineering

Hasil dari fase requirements engineering terdokumentasi dalam requirements specification. Requirements specification berisi kesepakatan bersama tentang permasalahan yang ingin dipecahkan antara pengembang dan customer, dan merupakan titik start menuju proses berikutnya yaitu software design. Sistemisasi proses kesepakatan pengembang dan customer dalam requirements engineering dibagi dalam 3 proses besar yaitu: **elicitation, specification, validation and verification.**

#### *Requirements Elicitation*

Adalah proses mengumpulkan dan memahami requirements dari user. Kadang masalah yang muncul berakar dari masalah knowledge domain (perbedaan disiplin ilmu yang dimiliki). Customer adalah expert pada domain yang softwrenya ingin dikembangkan (domain specialist) misal customer perbankan akan ahli dalam bidang perbankan, dilain pihak sang pengembang (requirements analyst) adakalanya sama sekali buta terhadap knowledge domain tersebut, meskipun tentu memahami dengan benar bagaimana sebuah software harus dikembangkan. Masalah knowledge domain tersebut yang diharapkan bisa diatasi dengan adanya interaksi terus menerus dan berulang (iterasi) antara pengembang dan customer. Proses interaksi tersebut kemudian dimodelkan menjadi beberapa teknik dan metodologi diantaranya adalah interviewing, brainstorming, prototyping, use case, dsb.

#### *Requirements Specification*

Setelah masalah berhasil dipahami, pengembang mendeskripsikannya dalam bentuk dokumen spesifikasi dokumen. Spesifikasi ini berisi tentang fitur dan fungsi yang diinginkan oleh customer, dan sama sekali tidak membahas bagaimana metode pengembangannya. IEEE mengeluarkan standard untuk dokumen spesifikasi equirements yang terkenal dengan nama IEEE Recommended Practice for Software



Requirements Specifications [IEEE-830]. Dokumen **spesifikasi requirements** bisa berisi **functional requirements, performance requirements, external interface requirements, design constraints**, maupun **quality requirements**.

#### *Requirements Validation and Verification*

Setelah spesifikasi requirements berhasil dibuat, perlu dilakukan dua usaha:

- Validation (validasi), yaitu proses untuk memastikan bahwa requirements yang benar sudah ditulis
- Verification (verifikasi), yaitu proses untuk memastikan bahwa requirements sudah ditulis dengan benar

Proses validasi dan verifikasi ini melibatkan customer (user) sebagai pihak yang menilai dan memberi feedback berhubungan dengan requirements.

Setelah melalui proses Requirement Engineering tentu saja masih panjang proses yang harus dilakukan pengembang sistem. Yaitu selanjutnya proses Analisa Sistem. Yang akan dijelaskan pada bab III.

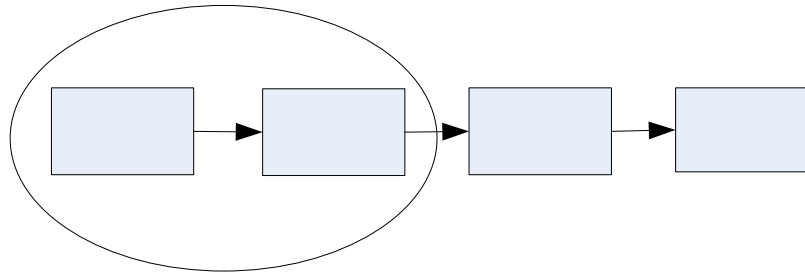
### **3. Model Proses Perangkat Lunak**

Untuk menyelesaikan masalah yang nyata dalam suatu hal, perekayasa perangkat lunak harus menggabungkan strategi pengembangan yang melingkupi lapisan proses, metode dan alat-alat bantu serta fase-fase generik (akan dijelaskan pada bab selanjutnya). Strategi yang dimaksud, sering diacukan sebagai *model proses* atau *paradigma rekayasa perangkat lunak*. Model proses untuk rekayasa perangkat lunak dipilih berdasarkan sifat aplikasi dan proyeknya, metode dan alat-alat bantu yang akan dipakai.

Pada bab ini selanjutnya akan didiskusikan bermacam-macam model proses yang berbeda pada perangkat lunak. Penting untuk diingat bahwa masing-masing model sudah ditandai dengan cara tertentu sehingga diharapkan bisa membantu di dalam kontrol dan koordinasi dari proyek perangkat lunak yang nyata. Dengan demikian, pada intinya semua model menunjukkan karakteristiknya.

#### **1. MODEL SEKUENSIAL LINIER (WATERFALL)**

Gambar berikut menggambarkan *sekuensial linier* untuk rekayasa perangkat lunak, yang sering disebut dengan *siklus kehidupan klasik* atau *model air terjun*.



**Gambar Model Sekuensial Linier (Waterfall Model)** Pemodelan sistem informasi

Sekuensial linier mengusulkan sebuah pendekatan kepada pengembangan perangkat lunak yang sistematis dan sekuensial yang mulai pada tingkat dan kemajuan system pada seluruh analisis, desain, kode, pengujian (tes), dan pemeliharaan. Berikut ini penjelasan yang bisa diberikan untuk masing-masing tahap :

- Analisis kebutuhan perangkat lunak

Proses pengumpulan kebutuhan difokuskan khususnya untuk perangkat lunak, perekayasa perangkat lunak (Analisis) harus memahami domain permasalahan (problem domain), tingkah laku, unjuk kerja dan antarmuka (interface) yang diperlukan. Kebutuhan baik untuk sistem maupun perangkat lunak didokumentasikan dan dilihat lagi dengan pelanggan.

- Desain

Desain perangkat lunak sebenarnya adalah proses multi langkah yang berfokus pada empat atribut sebuah program yang berbeda (struktur data, arsitektur perangkat lunak, representasi interface, dan detail (algoritma) prosedural. Proses desain menerjemahkan syarat/kebutuhan ke dalam sebuah representasi perangkat lunak yang dapat diperkirakan demi kualitas sebelum dimulai pemunculan kode (coding). Sebagaimana analisis, desain ini juga didokumentasikan.

- Generasi kode

Desain harus diterjemahkan ke dalam bentuk mesin yang bisa dibaca. Langkah pembuatan kode meliputi pekerjaan dalam langkah ini, dan dapat dilakukan secara mekanis.

- Pengujian (Tes)

Sekali kode dibuat, pengujian program dimulai. Proses pengujian berfokus pada logika internal perangkat lunak, memastikan bahwa semua pernyataan sudah diuji, dan pada eksternal fungsional, yaitu mengarahkan pengujian untuk menemukan kesalahan-kesalahan dan memastikan bahwa input yang dibatasi akan memberikan hasil aktual yang sesuai dengan hasil yang dibutuhkan.

- **Pemeliharaan**

Perangkat lunak akan mengalami perubahan setelah disampaikan kepada pelanggan (perkecualian yang mungkin adalah perangkat lunak yang dilekatkan). Perubahan akan terjadi karena kesalahan-kesalahan karena perangkat lunak harus disesuaikan untuk mengakomodasi perubahan-perubahan di dalam lingkungan eksternalnya atau karena pelanggan membutuhkan perkembangan fungsional atau unjuk kerja. Pemeliharaan perangkat lunak mengaplikasikan lagi setiap fase program sebelumnya dan tidak membuat yang baru lagi.

## 2. MODEL PROTOTYPE

Model prototipe ini dimulai dengan pengumpulan kebutuhan. Pengembang dan pelanggan bertemu dan mendefinisikan obyektif keseluruhan dari perangkat lunak, dan mengidentifikasi segala kebutuhan yang diketahui.



**Gambar. Model Prototipe**

Secara ideal prototipe berfungsi sebagai sebuah mekanisme untuk mengidentifikasi kebutuhan perangkat lunak. Prototipe bisa menjadi paradigma yang efektif bagi rekayasa perangkat lunak. Kuncinya adalah mendefinisikan aturan-aturan main pada saat awal, yaitu pelanggan dan pengembang keduanya harus setuju bahwa prototipe dibangun untuk berfungsi sebagai mekanisme pendefinisian kebutuhan. Prototipe kemudian disingkirkan dan perangkat lunak actual direkayasa dengan tertuju kepada kualitas dan kemampuan pemeliharaan.

### 3. MODEL RAD

*Rapid Application Development* (RAD) adalah sebuah model proses perkembangan perangkat lunak sekuensial linier yang menekankan siklus perkembangan yang sangat pendek. Model RAD ini merupakan sebuah adaptasi “kecepatan tinggi” dari model sekuensial linier dimana perkembangan cepat dicapai dengan menggunakan pendekatan konstruksi berbasis komponen. Jika kebutuhan dipahami dengan baik, proses RAD memungkinkan tim pengembangan menciptakan “sistem fungsional yang utuh” dalam periode waktu yang sangat pendek (kira-kira 60-90 hari). Pendekatan RAD melingkupi fase-fase sebagai berikut :

**Pemodelan Bisnis.** Aliran informasi diantara fungsi-fungsi bisnis dimodelkan dengan suatu cara untuk menjawab pertanyaan-pertanyaan berikut : Informasi apa yang mengendalikan proses bisnis? Informasi apa yang dimunculkan? Siapa yang memunculkannya? Kemana informasi itu pergi? Siapa yang memprosesnya?

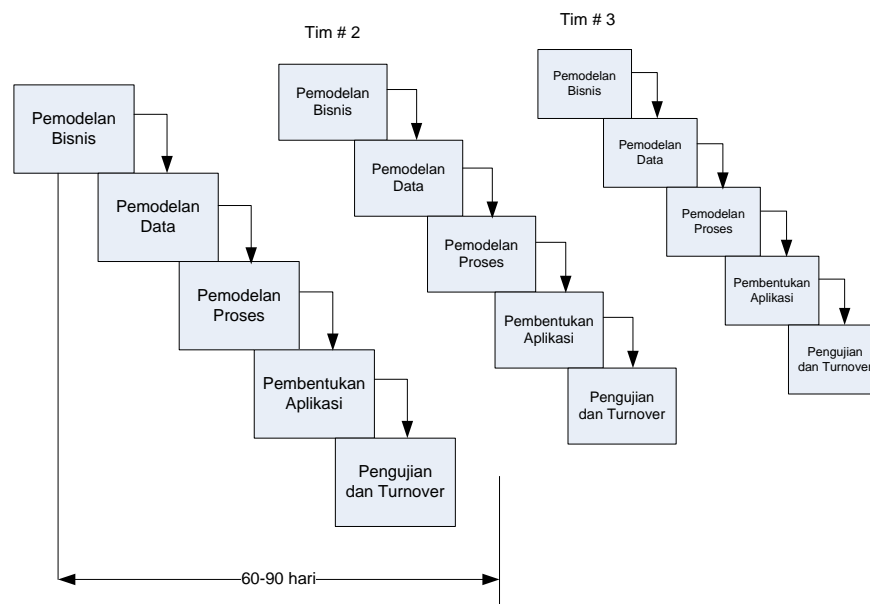
**Pemodelan Data.** Aliran informasi yang didefinisikan sebagai bagian dari fase business modelling disaring kedalam serangkaian objek data yang dibutuhkan untuk mendukung bisnis tersebut.

**Pemodelan Proses.** Aliran informasi yang didefinisikan di dalam fase data modelling ditransfirmasikan untuk mencapai aliran informasi yang perlu bagi implementasi

sebuah fungsi bisnis. Gambaran pemrosesan diciptakan untuk menambah, memodifikasi, menghapus, atau mendapatkan kembali sebuah objek data.

**Pembentukan Aplikasi.** RAD mengasumsikan pemakaian teknik generasi keempat. Selain menciptakan perangkat lunak dengan menggunakan bahasa pemrograman generasi ketiga yang konvensional, RAD lebih banyak memproses kerja untuk memakai lagi komponen program yang ada atau menciptakan komponen yang bisa dipakai lagi.

**Pengujian dan Turnover.** Karena proses RAD menekankan pada pemakaian kembali, banyak komponen program telah diuji. Hal ini mengurangi keseluruhan waktu pengujian. Tetapi komponen baru harus diuji dan semua interface harus dilatih secara penuh.



**Gambar. Model RAD (Rapid Application Development)**

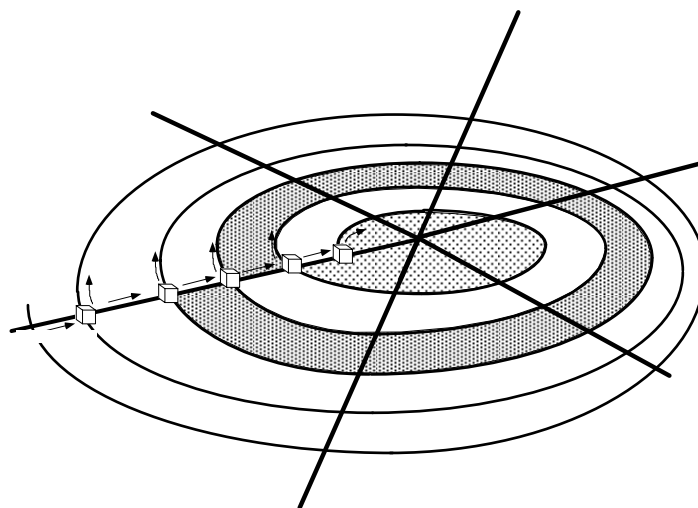
#### 4. MODEL SPIRAL

Model spiral (*spiral model*) yang pada awalnya diusulkan oleh Boehm adalah model proses perangkat lunak yang evolusioner yang merangkai sifat iteratif dari prototipe dengan cara kontrol dan aspek sistematis dari model sekuensial linier. Di dalam model spiral, perangkat lunak dikembangkan di dalam suatu deretan pertambahan.

Selama awal iterasi, rilis inkremental bisa merupakan sebuah model atau prototipe kertas. Selama iterasi berikutnya, sedikit demi sedikit dihasilkan versi sistem rekayasa yang lebih lengkap.

Model spiral dibagi menjadi sejumlah aktifitas kerangka kerja, disebut juga wilayah tugas, diantara tiga sampai enam wilayah tugas :

- Komunikasi pelanggan, tugas-tugas yang dibutuhkan untuk membangun komunikasi yang efektif diantara pengembang dan pelanggan.
- Perencanaan, tugas-tugas yang dibutuhkan untuk mendefinisikan sumber-sumber daya, ketepatan waktu, dan proyek informasi lain yang berhubungan.
- Analisis resiko, tugas-tugas yang dibutuhkan untuk menaksir resiko-resiko, baik manajemen maupun teknis.
- Perekayasaan, tugas-tugas yang dibutuhkan untuk membangun satu atau lebih representasi dari aplikasi tersebut.
- Konstruksi dan peluncuran, tugas-tugas yang dibutuhkan untuk mengkonstruksi, menguji, memasang (instal) dan memberikan pelayanan kepada pemakai (contohnya pelatihan dan dokumentasi).
- Evaluasi pelanggan, tugas-tugas yang dibutuhkan untuk memperoleh umpan balik dari pelanggan dengan didasarkan pada evaluasi representasi perangkat lunak, yang dibuat selama masa perekayasaan, dan diimplementasikan selama pemasangan.



**Gambar. Model Spiral**

Model spiral menjadi sebuah pendekatan yang realistis bagi perkembangan system dan perangkat lunak skala besar. Karena perangkat lunak terus bekerja selama proses bergerak, pengembang dan pemakai memahami dan bereaksi lebih baik terhadap resiko dari setiap tingkat evolusi. Model spiral menggunakan prototipe sebagai mekanisme pengurangan resiko. Tetapi yang lebih penting lagi, model spiral memungkinkan pengembang menggunakan pendekatan prototipe pada setiap keadaan di dalam evolusi produk. Model spiral menjaga pendekatan langkah demi langkah secara sistematis seperti yang diusulkan oleh siklus kehidupan klasik, tetapi memasukkannya ke dalam kerangka kerja iterative yang secara realistis merefleksikan dunia nyata.

## **BAB III**

### **ANALISIS SISTEM**

#### **Tujuan Praktikum :**

1. Menjelaskan Analisis Sistem dalam rekayasa perangkat lunak dan tahapannya
2. Menjelaskan Objek data, Atribut dan Hubungan antar objek data

#### **Indikator :**

- ❖ Agar praktikan bisa menjelaskan tahapan analisa sistem dalam rekayasa perangkat lunak dengan benar
- ❖ Agar praktikan bisa membedakan antara objek data, atribut dan hubungannya.

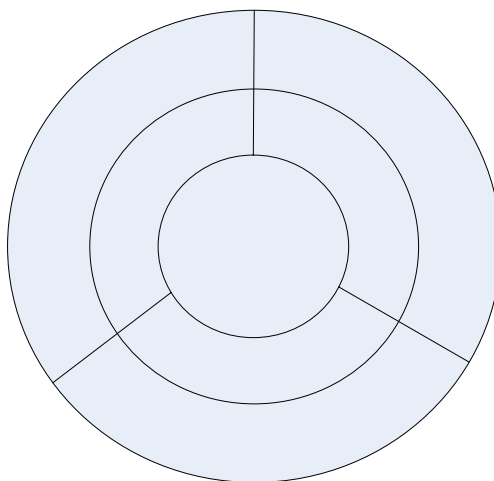
#### **Materi :**

#### **1. Analisis Sistem (Konvensional)**

##### **Tujuan Analisis :**

1. Menjabarkan kebutuhan pemakai
2. Meletakkan dasar-dasar untuk proses perancangan Perangkat Lunak
3. Mendefinisikan semua kebutuhan pemakai sesuai dengan lingkup kontrak yang disepakati kedua belah pihak

##### **Elemen Analisis :**



**Gambar Elemen Analisis**



- **Pemodelan Data**

Mendeskripsikan data yang terlibat dalam Perangkat Lunak.

Tools yang digunakan :

- ERD (Diagram Keterhubungan antar Objek Data),
- Data Object Description (Menyimpan semua atribut entitas dan relasi yang muncul pada ERD),
- Data Dictionary (Deskripsi semua objek data)

- **Pemodelan Fungsional**

Mendeskripsikan seluruh fungsi yang terlibat dalam Perangkat Lunak. Tools yang digunakan adalah :

- DAD (Diagram Aliran Data, yang menggambarkan bagaimana data ditransformasikan pada Perangkat Lunak)
- Proses Spesification (berisi deskripsi dari setiap fungsi yang muncul pada DAD)

- **Pemodelan Status**

Mendeskripsikan status sistem yang dapat muncul ketika Perangkat Lunak digunakan. Serta mendeskripsikan kelakuan sistem.

Tools yang digunakan :

- State Transition Diagram, menunjukkan bagaimana sistem bertingkah laku sebagai akibat dari kejadian eksternal.
- Control Spesification, menyajikan informasi tambahan mengenai aspek control dari perangkat lunak.

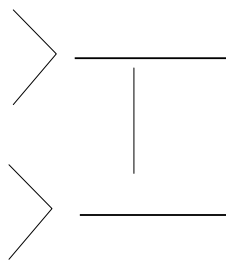
## **2. Objek Data, Atribut dan Hubungan**

Model data terdiri dari tiga informasi yang saling tergantung : objek data, atribut yang menggambarkan objek data tersebut, dan hubungan yang menghubungkan objek data yang satu dengan yang lain.

**Objek data** adalah representasi dari hampir semua informasi gabungan yang harus dipahami oleh perangkat lunak. Dengan informasi gabungan kita mengartikan sesuatu yang memiliki sejumlah sifat atau atribut yang berbeda.

Objek data dapat berupa entitas eksternal (misal semua yang menghasilkan atau mengonsumsi informasi), suatu benda (misal laporan), peristiwa (misalnya sambungan telepon) atau even/kejadian (misal alarm), peran misal (mahasiswa), unit organisasi (misal bagian gudang), dll.

Objek data dihubungkan satu dengan yang lainnya. Misal, seseorang dapat memiliki mobil, dimana hubungan memiliki mengkonotasikan suatu hubungan khusus antara seseorang dengan mobil. Hubungan itu selalu ditentukan oleh konteks masalah yang sedang dianalisis.



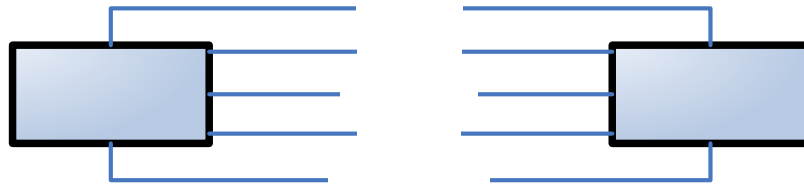
**Atribut** menentukan properti suatu objek data dan mengambil salah satu dari tiga karakteristik yang berbeda. Atribut dapat digunakan untuk :

1. menamai sebuah contoh dari objek data
2. menggambarkan contoh
3. membuat referensi ke contoh yang lain pada tabel yang lain.

Pada gambar diatas, contoh dari suatu atribut dari objek orang adalah *Nama*, *Alamat*, *Umur* dan *No SIM*. Atribut dari objek mobil adalah *Model*, *Nomor ID*, *Body Type* dan *Warna*. Selain contoh atribut di atas, tentunya masih banyak komponen atribut lain yang bias menjelaskan objek Orang maupun Mobil.

**Hubungan** Objek data disambungkan satu dengan yang lainnya dengan berbagai macam cara. Andaikan ada dua objek data, buku dan perpustakaan. Objek tersebut dapat diwakilkan dengan menggunakan notasi sederhana yang diunjukkan pada gambar berikut :

*Nama*  
*Alamat*  
*Umur*  
*No SIM*  
*Model*  
*Nomor ID*  
*Body Type*  
*Warna*



Dibangun suatu hubungan di antara buku dan perpustakaan karena kedua objek data tersebut berhubungan. Tetapi apa hubungan tersebut? Untuk menentukan jawabannya, kita harus memahami posisi buku dan toko buku dalam konteks perangkat lunak yang akan dibangun. Kita dapat mendefinisikan *object relationship pairs* yang mendefinisikan hubungan yang relevan.

## Buku

Misalnya :

- perpustakaan memesan buku
- perpustakaan memajang buku
- perpustakaan memiliki persediaan buku
- perpustakaan mempunyai buku
- perpustakaan mengembalikan buku

Hubungan antara *memesan*, *memajang*, *persediaan*, *mempunyai* dan *mengembalikan* mendefinisikan hubungan yang relevan antara buku dan perpustakaan.

Dalam melakukan analisis, pendefinisian dari objek data, atribut dan hubungan antar objek perlu dilakukan, karena dengan pendefinisian ini berhubungan dengan pendefinisian semua kebutuhan yang diperlukan untuk pengembangan perangkat lunak. Dan pendefinisian ini diperlukan pada tahap pemodelan data (tahap 1)

## BAB IV

### DAD (DIAGRAM ALIRAN DATA)

#### Tujuan Praktikum :

1. Mengenalkan DAD (Diagram Aliran Data) sebagai tool perancangan sistem.
2. Memahami Tahapan-tahapan Levelisasi DAD

#### Indikator :


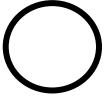


- ❖ Agar praktikan bisa menjelaskan pengertian DAD dan kegunaannya
- ❖ Agar praktikan bisa membuat DAD berdasar level-levelnya dari kasus yang diberikan oleh dosen dengan baik dan benar.

#### Materi :

**DAD** (Diagram Aliran Data) atau yang juga dikenal dengan sebutan **DFD** (Data Flow Diagram) merupakan alat perancangan sistem yang berorientasi pada alur data dengan konsep dekomposisi dapat digunakan untuk penggambaran analisa maupun rancangan sistem yg mudah dikomunikasikan oleh profesional sistem kepada pemakai maupun pembuat program.

Pada umumnya, DAD digunakan untuk merancang sistem yang menggunakan data store dalam mengelola informasi dalam sistem.

Komponen DAD, menurut Yourdan dan DeMarco adalah sebagai berikut :

Simbol	Keterangan
	Entitas Luar/Terminator
	Proses
	Aliran Data (Data Flow)
	Penyimpan Data (Data Store)

Keterangan :

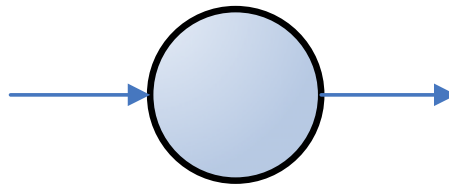
- **Entitas Luar** : kesatuan diluar sistem yang akan memberikan input atau menerima output dari sistem dapat berupa orang atau organisasi sumber informasi lain atau penerima akhir dari suatu laporan.

**Contoh :**



- **Proses** adalah transformasi input menjadi output, merupakan kegiatan atau pekerjaan yang dilakukan oleh orang atau mesin komputer, dimana aliran data masuk ditransformasikan ke aliran data keluar. Penamaannya sesuai dengan proses yang sedang dilakukan.

**Contoh :**



**Ada beberapa hal yang perlu diperhatikan tentang proses :**

1. Proses harus memiliki input dan output.
2. Proses dapat dihubungkan dengan komponen *entitas luar*, *data store* atau *proses* melalui alur data.
3. Sistem/bagian/divisi/departemen yang sedang dianalisis oleh profesional sistem digambarkan dengan komponen proses.

- **Aliran data/Arus data** digunakan untuk menjelaskan perpindahan data atau paket data dari satu bagian ke bagian lain

Aliran data dapat berbentuk sebagai berikut :

- Formulir atau dokumen yang digunakan perusahaan
- Laporan tercetak yang dihasilkan sistem
- Output dilayar komputer
- Masukan untuk komputer
- Komunikasi ucapan

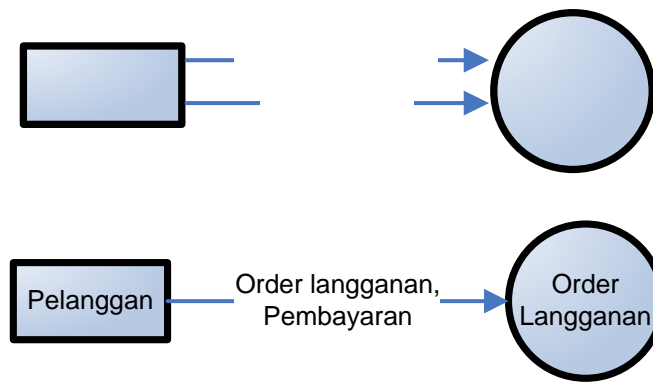
- Surat atau memo
- Data yang dibaca atau direkam di file
- Suatu isian yang dicatat pada buku agenda
- Transmisi data dari suatu komputer ke komputer lain

Catatan : aliran data tidak dalam bentuk kalimat.

### Konsep Arus Data :

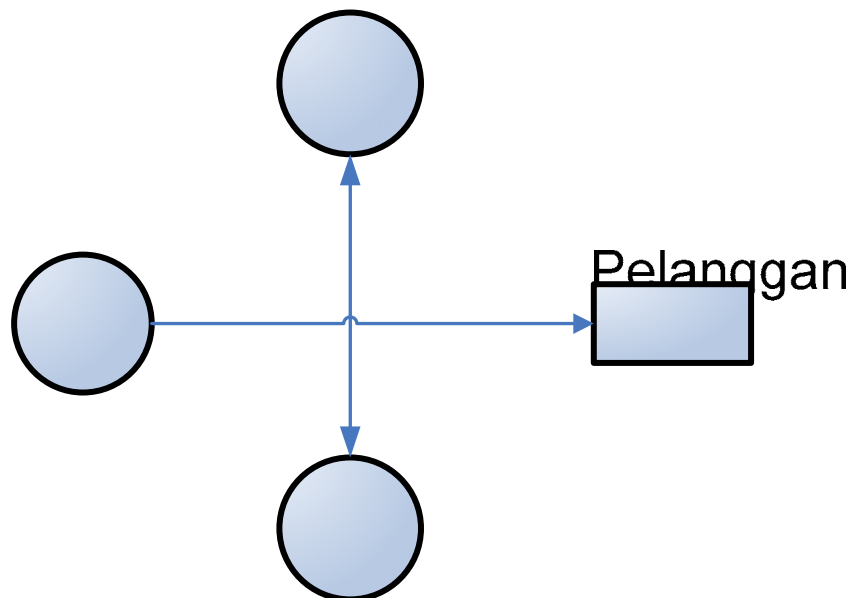
#### 1. Packet of Data (Paket Data)

Bila dua data mengalir dari suatu sumber yang sama menuju ke tujuan yang sama, maka harus dianggap sebagai suatu arus data yang tunggal



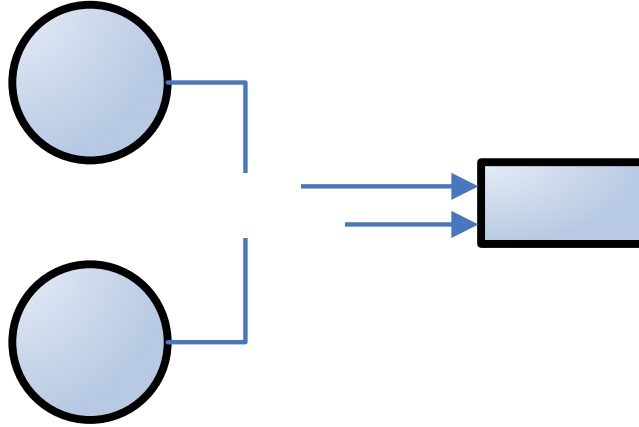
#### 2. Diverging Data Flow (Arus Data Menyebar)

Arus data yang data yang menyebar menunjukkan sejumlah tembusan dari arus data yang sama dari sumber sama ke tujuan berbeda.



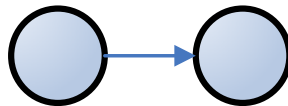
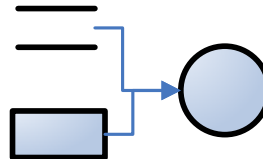
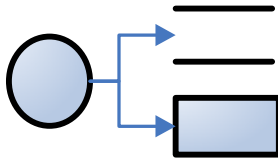
### 3. Convergen Data Flow (Arus Data Mengumpul)

Arus data yang mengumpul yaitu Arus data yang berbeda dari sumber yang berbeda mengumpul ke tujuan yang sama.



### 4. Sumber dan Tujuan

Arus data harus dihubungkan pada proses, baik dari maupun yang menuju proses.



1.0

Pembuatan  
Faktur

Slip p

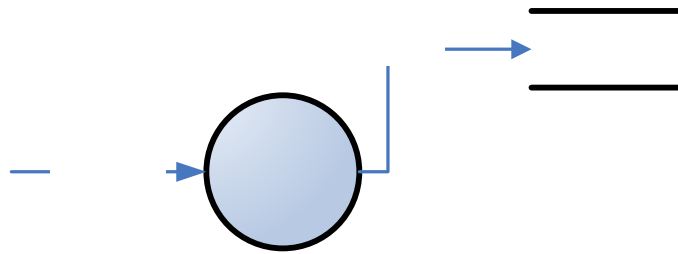
2.0

Pembuatan  
Slip  
Pengepakan

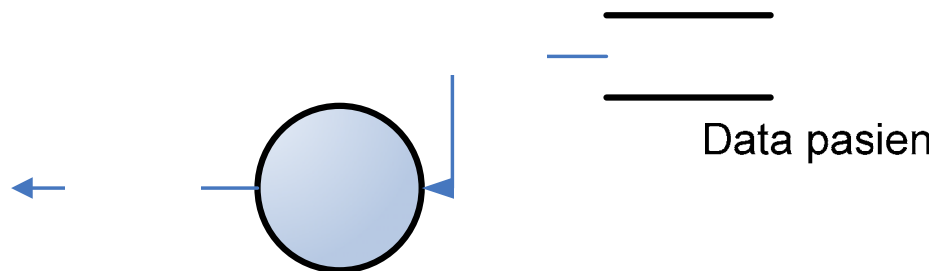
- **Data Storage** adalah komponen untuk membuat model sekumpulan data, dapat berupa suatu file atau suatu sistem database dari suatu komputer, suatu arsip/dokumen, suatu agenda/buku.

**Yang perlu diperhatikan tentang data store :**

1. Alur data dari proses menuju data store, hal ini berarti data store berfungsi sebagai tujuan/tempat penyimpanan dari suatu proses (proses write).

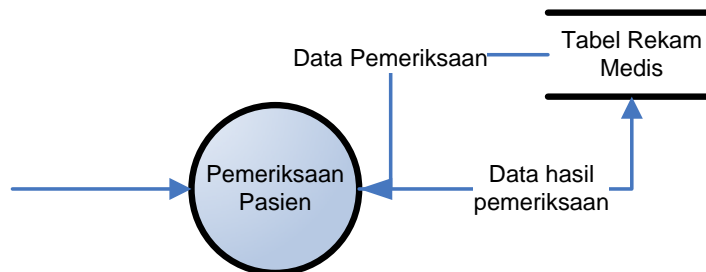


2. Alur data dari data store ke proses, hal ini berarti data store berfungsi sebagai sumber/ proses yang memerlukan data (proses read).



Penda  
Pas

3. Alur data dari proses menuju data store dan sebaliknya berarti berfungsi sebagai sumber dan tujuan.



#### Langkah-langkah Pembuatan DAD :

1. Identifikasi semua kesatuan luar yang terlibat dengan sistem
2. Identifikasi input dan output yang berhubungan dengan kesatuan luar
3. Buatlah gambaran diagram konteksnya.

Data  
Pemeriksaan

Pemerik  
Pasien

#### Jenis Diagram dalam DAD :

- Diagram Konteks

Disebut juga diagram tingkat atas, merupakan diagram sistem yang menggambarkan aliran-aliran data yang masuk dan keluar dari sistem dan yang masuk dan keluar dari entitas luar.

Hal yang harus diperhatikan :



- Memberikan gambaran tentang seluruh sistem
- Terminal yang memberikan masukan ke sistem disebut *source*
- Terminal yang menerima keluaran disebut *sink/destination*
- Hanya ada satu proses
- Tidak boleh ada data store
- Diagram Level 1
 

Setelah pembuatan DAD Level Konteks, selanjutnya adalah pembuatan DAD Level 1, dimana pada DAD Level adalah penggambaran dari Diagram Konteks yang lebih rinci (Overview Diagram) atau biasanya disebut sebagai **dekomposisi**. Hal yang harus diperhatikan :

  - Perlihatkan data store yang digunakan
  - Pada proses yang tidak dirinci lagi, tambahkan tanda \* pada akhir penomoran proses
  - Keseimbangan antara diagram konteks dan diagram level 1 harus dipelihara.
- Diagram Level 2
- Diagram Level 3, dst

### Penomoran Proses

Nama Level	Nama Diagram	Nomor Proses
0	Konteks	0
1	Diagram Level 1	1.0, 2.0, 3.0
2	Diagram Rinci 1.0	1.1, 1.2, 1.3
2	Diagram Rinci 2.0	2.1, 2.2, 2.3
2	Diagram Rinci 3.0	3.1, 3.2, 3.3
3	Diagram Rinci 1.1	1.1.1, 1.1.2, 1.1.3
3	Diagram Rinci 1.2	1.2.1, 1.2.2, 1.2.3
3	Diagram Rinci 1.3	1.3.1, 1.3.2, 1.3.2
dst		

## Penggambaran DAD

Tidak ada aturan baku untuk menggambarkan DFD, tapi dari berbagai referensi yang ada, secara garis besar:

### 1. Membuat Diagram Konteks

Diagram ini adalah diagram level tertinggi dari DAD yg menggambarkan hubungan sistem dengan lingkungan luarnya.

Cara :

- Tentukan nama sistemnya. Misal, Sistem Informasi Rumah Sakit, Aplikasi Pengenalan Jenis Ikan, Aplikasi Pemilihan Jamur berkualitas dsb.
- Tentukan batasan sistemnya.
- Tentukan terminator/entitas luar apa saja yg ada dalam sistem.
- Tentukan apa yg diterima/diberikan terminator dari/ke sistem.
- Gambarkan diagram konteks.

### 2. Buat Diagram Level 1

Diagram ini adalah dekomposisi dari diagram Context.

Cara :

- Tentukan proses utama yang ada pada sistem.
- Tentukan apa yang diberikan/diterima masing-masing proses ke/dari sistem sambil memperhatikan konsep keseimbangan (alur data yg keluar/masuk dari suatu level harus sama dengan alur data yg masuk/keluar pada level berikutnya)
- Apabila diperlukan, munculkan data store sebagai sumber maupun tujuan alur data.
- Gambarkan diagram level satu.
- Hindari perpotongan arus data
- Beri nomor pada proses utama (nomor tidak menunjukkan urutan proses). Misal. 1.0, 2.0, 3.0 dst

### 3. Buat Diagram Level 2

Diagram ini merupakan dekomposisi dari diagram level satu.

Cara :

- Tentukan proses yang lebih kecil (sub-proses) dari proses utama yang ada di level satu.
- Tentukan apa yang diberikan/diterima masing-masing sub-proses pada/dari sistem dan perhatikan konsep keseimbangan.
- Apabila diperlukan, munculkan data store (transaksi) sebagai sumber maupun tujuan alur data.
- Gambarkan DFD level Dua
- Hindari perpotongan arus data.
- Beri nomor pada masing-masing sub-proses yang menunjukkan dekomposisi dari proses sebelumnya. Contoh : 1.1, 1.2, 1.3 dst

#### 4. DFD Level Tiga, Empat, dst

Diagram ini merupakan dekomposisi dari level sebelumnya. Proses dekomposisi dilakukan sampai dengan proses siap dituangkan ke dalam program. Aturan yang digunakan sama dengan level dua.

### Pengembangan Diagram Alir Data

Bagaimana mengembangkan Diagram Alir Data. Berikut ini beberapa aktivitas yang dibutuhkan untuk pengembangan DAD :

- Mengidentifikasi aliran data kunci
- Mengidentifikasi semua entitas eksternal
- Mengidentifikasi semua fungsi
- Mengidentifikasi semua jalur aliran data
- Mengidentifikasi batas sistem
- Mengidentifikasi semua proses
- Mengidentifikasi semua data store
- Mengidentifikasi antara proses dan data store
- Pada Level yang lebih rendah, mengisi tiap detil proses.

## **BAB V**

### **BAGAN ALIR (FLOWCHART)**

#### **Tujuan Praktikum :**

1. Mengenalkan Flowchart sebagai tool perancangan sistem
2. Pengenalan Jenis-jenis flowchart dan aplikasi yang sesuai
3. Mengenalkan studi kasus dengan menggunakan tools flowchart

#### **Indikator :**

- ❖ Agar praktikan bisa menjelaskan fungsi flowchart dalam perancangan sistem
- ❖ Agar praktikan bisa membedakan aplikasi mana yang menggunakan flowchart sebagai tools perancangan sistem
- ❖ Agar praktikan bisa menyebutkan jenis flowchart serta aplikasi yang sesuai
- ❖ Membuat flowchart dari studi kasus yang diberikan oleh dosen dengan benar

#### **Materi :**

#### **Flowchart atau bagan alir :**

Merupakan alat bantu yang bisa digunakan untuk kegiatan analisa sistem dan perancangan (desain) sistem. Suatu skema representasi suatu proses atau algoritma. Flowchart merupakan salah satu tool yang digunakan untuk Quality Control.

Flowchart adalah bentuk gambar/diagram yang mempunyai aliran satu atau dua arah secara sekuensial. Flowchart digunakan untuk merepresentasikan maupun mendesain program. Oleh karena itu flowchart harus bisa merepresentasikan komponen-komponen dalam bahasa pemrograman. Baik flowchart maupun algoritma bisa dibuat sebelum maupun setelah pembuatan program. Flowchart dan Algoritma yang dibuat sebelum membuat program digunakan untuk mempermudah pembuat program untuk menentukan alur logika program, sedangkan yang dibuat setelah pembuatan program digunakan untuk menjelaskan alur program kepada orang lain.

Flowchart berbeda dengan DAD, yang paling jelas adalah dari tahapan/langkah alur data serta simbol-simbol yang digunakan. Untuk itu, perhatikan penjelasan berikut sehingga bisa mengetahui apa saja perbedaan yang ada pada flowchart dan DAD.

## Macam-macam Flowchart :

### 1. Bagan Alir Sistem (System Flowchart)

Merupakan bagan yang menunjukkan arus pekerjaan secara keseluruhan dari sistem. Contoh. Pendaftaran pasien di RS, Pendaftaran matakuliah praktikum, dll.

### 2. Bagan Alir Dokumen (Document Flowchart)

Merupakan bagan alir yang menunjukkan arus data dari laporan dan formulir-formulir termasuk tembusannya. Contoh. Pelaporan bulanan Perusahaan.

### 3. Bagan Alir Skematik (Schematic Flowchart)

Menggambarkan prosedur di dalam sistem. Bagan ini menggunakan simbol bagan alir sistem, juga menggambarkan komputer dan peralatan lainnya. Contoh. Bagan alir proses robot, bagan alir proses pencetakan dokumen.





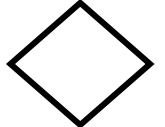
### 4. Bagan Alir Program (Program Flowchart)

Merupakan bagan yang menjelaskan secara rinci langkah-langkah dari proses program. Contoh. Bagan alir untuk proses penghitungan faktorial, bagan alir untuk proses penghitungan suhu ruangan.

### 5. Bagan Alir Proses (Process Flowchart)

Merupakan bagan alir yang banyak digunakan di teknik (misal. teknik industri).

## Simbol FLOWCHART

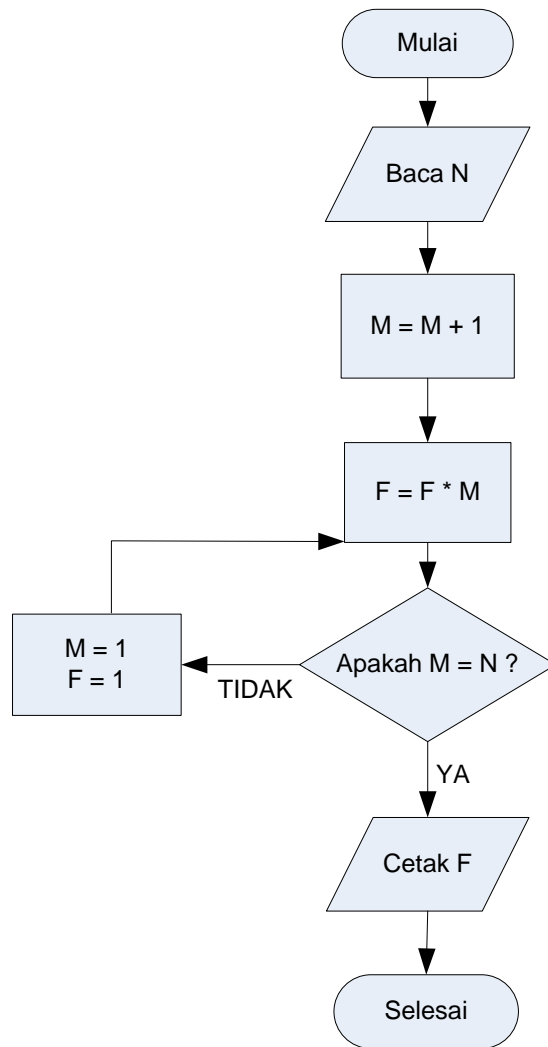
Nama	Simbol	Keterangan
Oval		Menunjukkan notasi untuk awal dan akhir bagan alir
Aliran data		Menunjukkan petunjuk dari aliran fisik pada program
Data		Menunjukkan suatu operasi input atau suatu operasi output.
Rectangle		Menunjukkan suatu proses yang akan digunakan
Diamond		Menotasikan suatu keputusan (atau cabang) yang akan dibuat. Program akan memilih satu dari dua rute.

Keterangan :

- Tanda Oval, biasanya disebut sebagai *start and end symbol*. Biasanya diberi nama “Mulai” dan “Selesai”, atau “Start” dan “End”. Ataupun frase lain yang menunjukkan bahwa program/bagan alir tersebut mulai dan selesai.
- Diamond sebagai tanda pemilihan adalah **Conditional** (or **decision**). Simbol ini mengandung pertanyaan Yes/No atau Ya/Tidak atau Benar/Salah. Simbol ini memiliki dua simbol aliran data yang keluar, biasanya dari sudut yang bawah dan sudut yang kanan, satunya menyatakan “Benar(true)” dan satunya menyatakan “False(salah)”.

Contoh kasus yang menggunakan Flowchart :

1. Berikut ini adalah contoh dari bagan alir untuk proses menghitung Faktorial N ( $N!$ ), dimana  $N! = 1 * 2 * 3 * 4 * 5 * \dots N$ . fungsi dari bagan alir untuk menghitung N Faktorial ini akan mempermudah kita dalam menuliskan pada program komputer, karena dari bagan alir ini, kita akan mudah untuk memahami algoritma yang digunakan, karena flowchart ini merupakan tahapan suatu instruksi seperti halnya algoritma suatu program.  
 $N$  = bilangan faktorial yang dicari,  $M$  = bilangan 1, 2, 3, 4 ..  $N$



**Flowchart penghitungan N factorial**

## 2. RENTAL VCD

Yang perlu diperhatikan saat akan membuat diagram alir proses peminjaman VCD yaitu :

- ❖ Investigasi data
- ❖ Wawancara/Observasi sistem
- ❖ Narasi
- ❖ Investigasi Data
  1. Kartu Anggota
  2. Fotokopi KTP/SIM
  3. Lembar Identitas

4. Nota Peminjaman
5. Data VCD
6. Laporan bulanan

❖ Wawancara

Hasil Wawancara

1. Orang yang terlibat: Anggota, adm, manager
2. Calon anggota harus mendaftar dengan membawa identitas diri (KTP/SIM)

❖ Hasil Quisioner

1. Keterlambatan tidak didenda didenda
2. Kesulitan dalam mencari data vcd pada arsip vcd

❖ Narasi

1. Calon anggota penyewaan vcd mengisi lembar identitas dan memberikan kartu pengenalan SIM/KTP
2. Petugas memeriksa apakah data tersebut sudah ada pada arsip anggota
3. Jika tidak ada petugas akan membuat kartu anggota baru dan memberikannya pada anggota
4. Pada saat meminjam anggota harus menunjukkan kartu anggota dan memberikan data vcd yang akan dipinjam
5. Petugas akan mencari dari arsip vcd
  - Jika vcd tidak ada atau sedang dipinjam maka petugas akan memberitahukan status kosong ke anggota
  - Jika ada maka petugas akan membuat nota peminjaman dan memberikannya ke anggota



## **BAB VI**

### **UML (UNIFIED MODELLING LANGUAGE)**

#### **Tujuan Praktikum :**

1. Mengenalkan Pengertian UML (Unified Modelling Language)
2. Menjelaskan Konsep Dasar UML

#### **Indikator :**

- ❖ Agar praktikan bisa memahami pengertian UML beserta kegunaannya.
- ❖ Agar praktikan bisa menjelaskan Konsep Dasar UML dan berbagai macam jenis diagram UML

#### **Materi :**

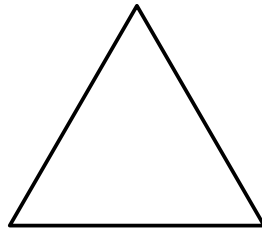
#### **Pendahuluan**

Saat ini piranti lunak semakin luas dan besar lingkupnya, sehingga tidak bisa lagi dibuat asal-asalan. Piranti lunak saat ini seharusnya dirancang dengan memperhatikan hal-hal seperti *scalability*, *security*, dan eksekusi yang robust walaupun dalam kondisi yang sulit. Selain itu arsitekturnya harus didefinisikan dengan jelas, agar *bug* mudah ditemukan dan diperbaiki, bahkan oleh orang lain selain *programmer* aslinya. Keuntungan lain dari perencanaan arsitektur yang matang adalah dimungkinkannya penggunaan kembali modul atau komponen untuk aplikasi piranti lunak lain yang membutuhkan fungsionalitas yang sama.

**Pemodelan** (*modeling*) adalah proses merancang piranti lunak sebelum melakukan pengkodean (*coding*). Model piranti lunak dapat dianalogikan seperti pembuatan blueprint pada pembangunan gedung. Membuat model dari sebuah sistem yang kompleks sangatlah penting karena kita tidak dapat memahami sistem semacam itu secara menyeluruh. Semakin kompleks sebuah sistem, semakin penting pula penggunaan teknik pemodelan yang baik. Dengan menggunakan model, diharapkan pengembangan piranti lunak dapat memenuhi semua kebutuhan pengguna dengan lengkap dan tepat, termasuk faktor-faktor seperti *scalability*, *robustness*, *security*, dan sebagainya. Kesuksesan suatu

pemodelan piranti lunak ditentukan oleh tiga unsur, yang kemudian terkenal dengan sebuah segitiga sukses (*the triangle for success*). Ketiga unsur tersebut adalah metode pemodelan (*notation*), proses (*process*) dan *tool* yang digunakan.

Memahami notasi pemodelan tanpa mengetahui cara pemakaian yang sebenarnya (proses) akan membuat proyek gagal. Dan pemahaman terhadap metode pemodelan dan proses disempurnakan dengan penggunaan tool yang tepat.



## Pengertian

**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.

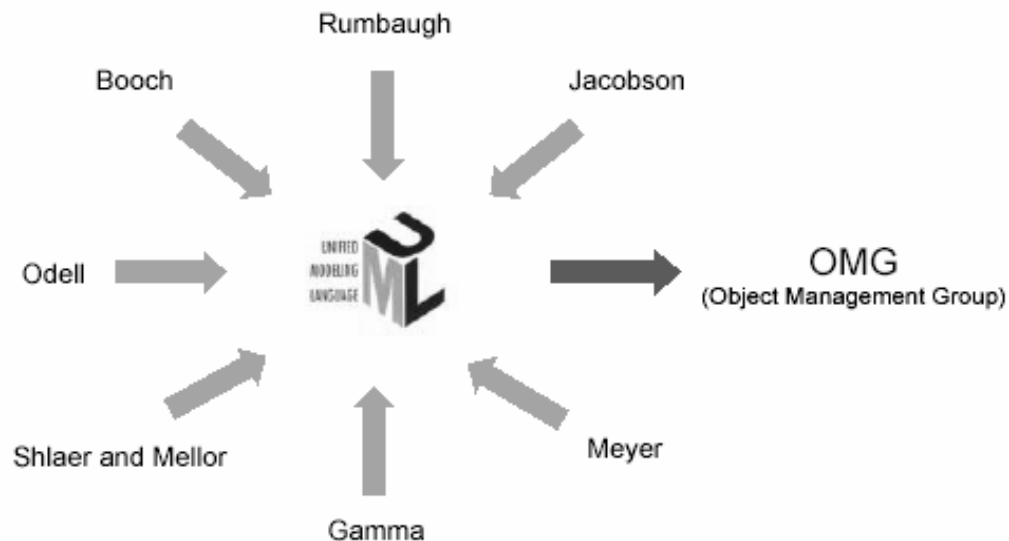
Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax*/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD

Metodologi

Proses

(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 wirfs-brock [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.

## Konsep Dasar UML

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

<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 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 diagram 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 diagram-diagram sebagai berikut:

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

#### **Tugas :**

1. Apakah perbedaan bahasa berorientasi objek dengan aplikasi prosedural?
2. UML juga menggunakan class dan operation dalam konsep dasarnya, bagaimana hubungannya dengan penulisan piranti perangkat lunak?
3. Apakah yang dimaksud dengan use case diagram, class diagram sequence diagram, collaboration diagram, activity diagram, state diagram didalam UML?

## BAB VII

### MACAM-MACAM DIAGRAM

#### Tujuan Praktikum :

1. Mengetahui contoh-contoh diagram UML beserta penjelasannya

#### Indikator :

- ❖ Agar praktikan bisa menjelaskan pengertian masing-masing jenis diagram dalam UML

#### Materi :

#### 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.

*Use Case Diagram* menggambarkan interaksi antara actor dengan proses atau sistem yang dibuat. *Use Case Diagram* mempunyai beberapa bagian penting seperti : *Actor, Use Case, Unidirectional Association, Generalization*.

### 1. Actor

*Actor* merupakan bagian dari *Use Case* yang bertindak sebagai subjek (pelaku) dalam suatu proses.

### 2. Use Case

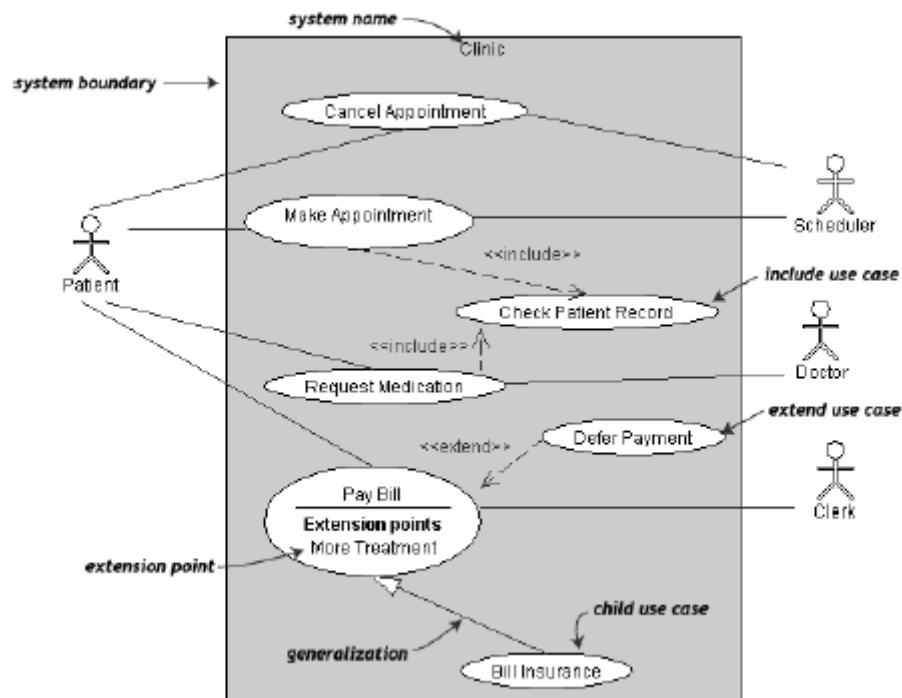
*Use Case* adalah proses-proses yang terjadi dalam suatu software. *Use Case* juga menggambarkan apa yang sedang dilakukan oleh seorang *Actor*.

### 3. Relasi

Relasi menggambarkan hubungan antara *actor* dan *use case*. Relasi-relasi tersebut dapat dibagi menjadi :

- *Unidirectional Association*
- *Generalization*
- *Dependency*

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*. Kita juga dapat membuat diagram yang terdiri atas *package*.

## Hubungan Antar Class

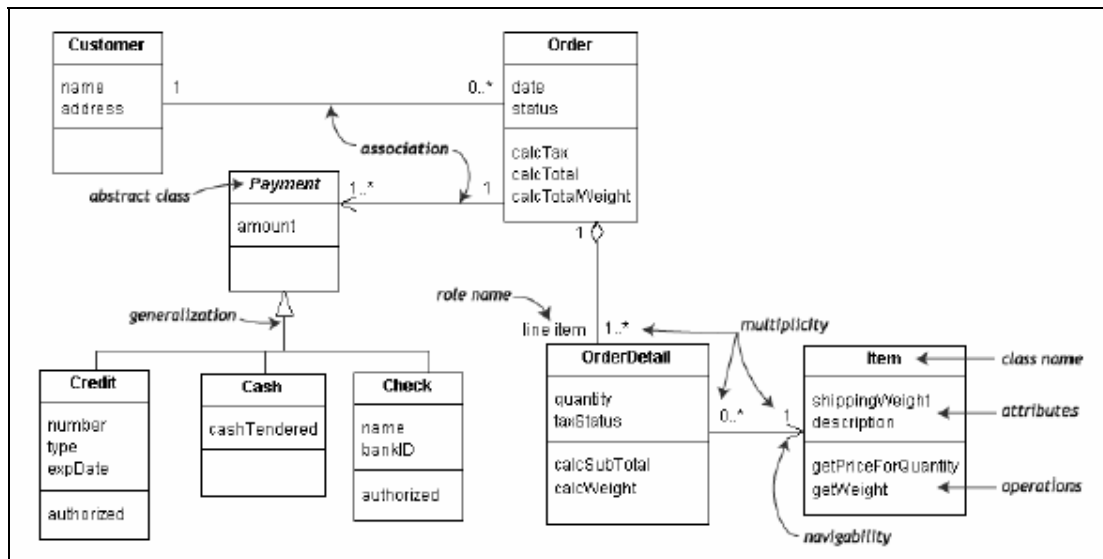
1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah *query* antar *class*.
2. Agregasi, yaitu hubungan yang menyatakan bagian (“terdiri 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 :

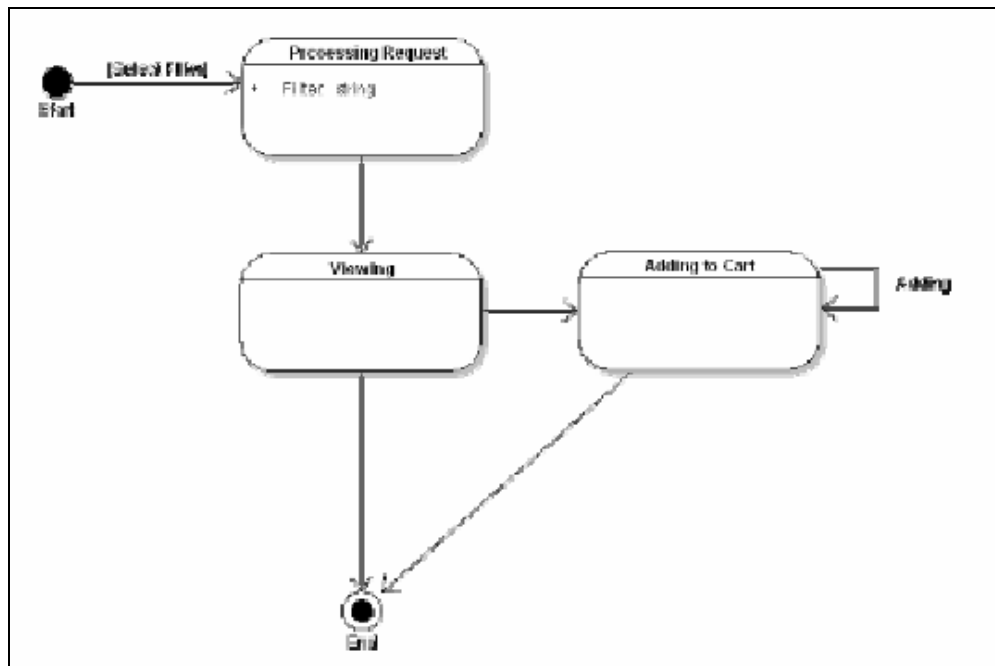


## Statechart Diagram

*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* :



## 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.

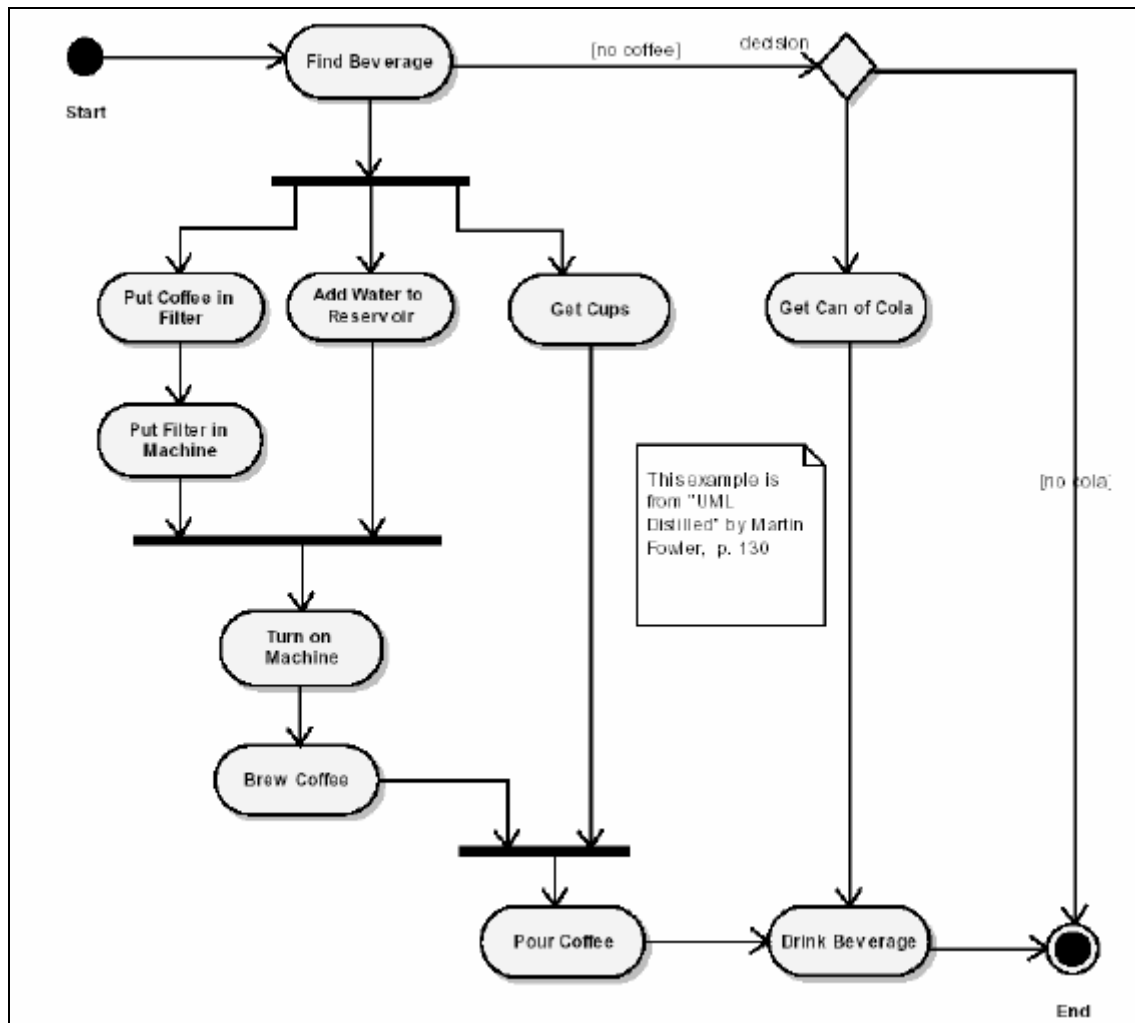
*Activity diagram* merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-trigger oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi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

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

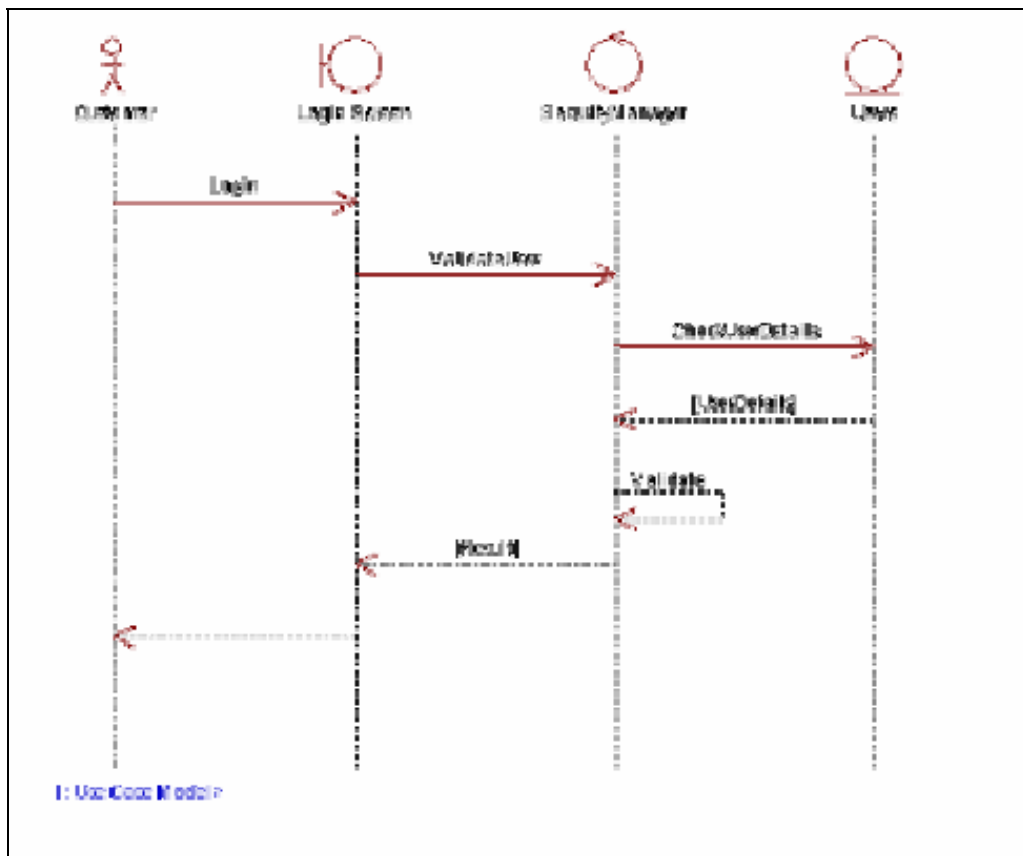
menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan.

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* :



## Collaboration Diagram

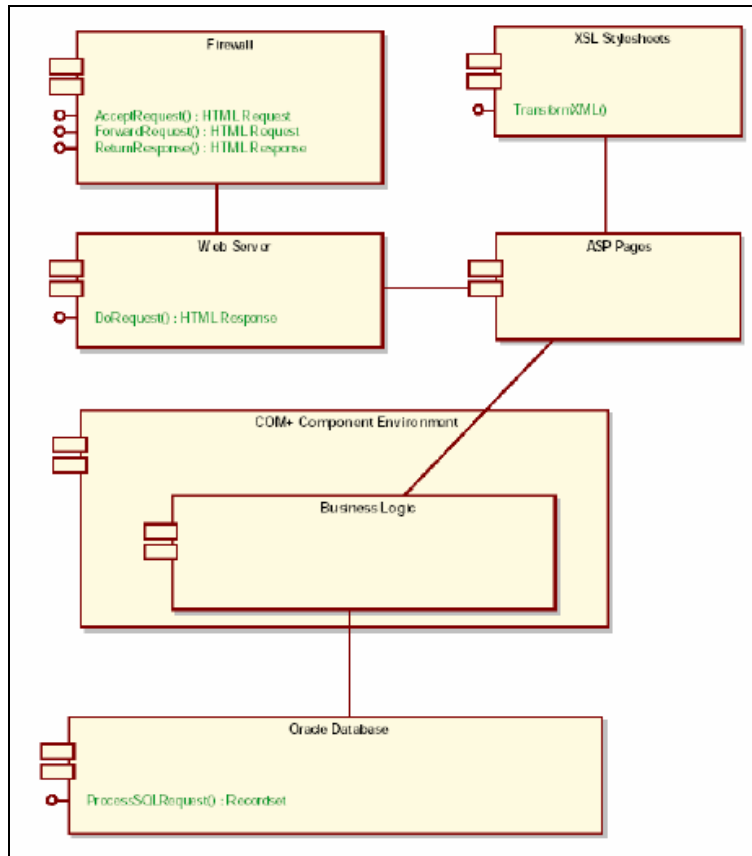
*Collaboration diagram* juga menggambarkan interaksi antar objek seperti *sequence diagram*, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*. Setiap *message* memiliki *sequence number*, di mana *message* dari level tertinggi memiliki nomor 1. Messages dari level yang sama memiliki prefiks yang sama.

## 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.

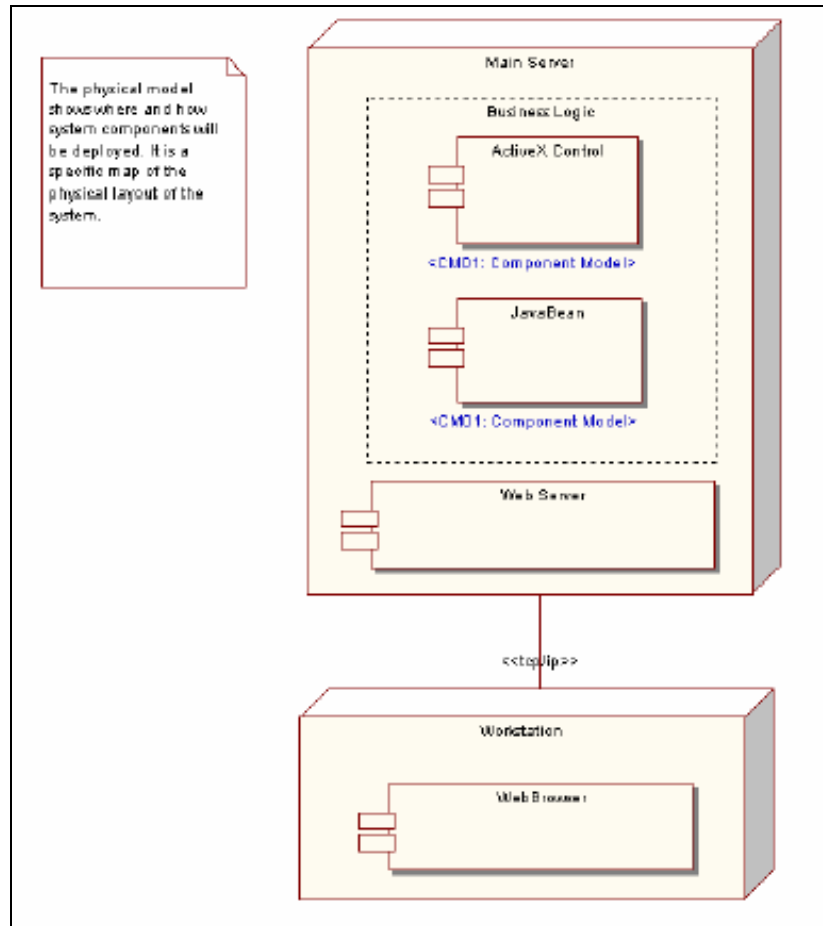
Contoh Component Diagram :



## 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-*deployn* komponen dalam lingkungan sebenarnya. Hubungan antar *node* (misalnya TCP/IP) dan *requirement* dapat juga didefinisikan dalam diagram ini.

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. Buarlah 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 berserta *codenya*. Model harus selalu sesuai dengan *code* yang aktual.
13. Piranti lunak siap dirilis.

### **Tool Yang Mendukung UML**

Saat ini banyak sekali tool pendesainan yang mendukung UML, baik itu tool komersial maupun opensource. Beberapa diantaranya adalah:

- Rational Rose ([www.rational.com](http://www.rational.com))



- Together ([www.togethersoft.com](http://www.togethersoft.com))
- Object Domain ([www.objectdomain.com](http://www.objectdomain.com))
- Jvision ([www.object-insight.com](http://www.object-insight.com))
- Objecteering ([www.objecteering.com](http://www.objecteering.com))
- MagicDraw ([www.nomagic.com/magicdrawuml](http://www.nomagic.com/magicdrawuml))
- Visual Object Modeller ([www.visualobject.com](http://www.visualobject.com))

Data seluruh tool yang mendukung UML, lengkap beserta harganya (dalam US dolar) bisa anda pelajari di situs

[http://www.objectsbydesign.com/tools/umltools\\_byCompany.html](http://www.objectsbydesign.com/tools/umltools_byCompany.html) .

Disamping itu, daftar tool UML berikut fungsi dan perbangingan kemampuannya juga dapat dilihat di <http://www.jeckle.de/umltools.htm>.

### **Pointer Penting UML**

Sebagai referensi dalam mempelajari dan menggunakan UML, situs-situs yang merupakan pointer

- [http://www.cetus-links.org/oo\\_uml.html](http://www.cetus-links.org/oo_uml.html)
- <http://www.omg.org>
- <http://www.omg.org/technology/uml/>
- <http://www.rational.com/uml>
- <http://www.uml.org/>