BAB II

TINJAUAN PUSTAKA

II.1. Sistem

Sistem adalah kumpulan dari elemen-elemen yang berinteraksi untuk mencapai suatu tujuan tertentu.

II.2. Pakar

Pakar adalah seseorang yang mempunyai pengetahuan dan metode khusus serta mampu menerapkannya untuk memecahkan masalah atau memberi nasehat (T.Sutojo;2010:163).

II.3. Sistem Pakar

II.3.1. Pengertian Sistem Pakar

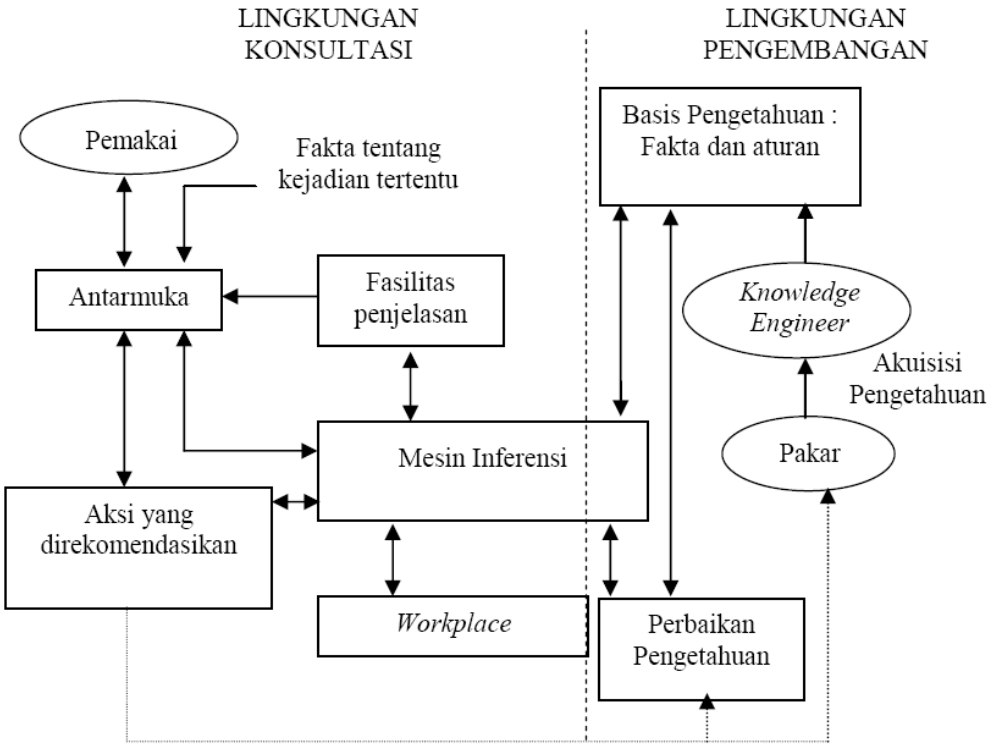
SistemPakar (*Expert System*) adalah sistem yang dirancang untuk dapat menirukan keahlian seorang pakar dalam menjawab pernyataan dan memecahkan suatu masalah (T. Sutojo;2010:13).

Sistem pakar adalah sebuah sistem yang menggunakan pengetahuan manusia dimana pengetahuan tersebut dimasukkan ke dalam sebuah komputer dan kemudian digunakan untuk menyelesaikan masalah-masalah yang biasanya membutuhkan kepakaran atas keahlian manusia(T.Sutojo;2010:161).

Dapat diambil kesimpulan bahwa sistem pakar adalah sebuah sistem yang dapat menirukan keahlian seorang pakar dalam menyelesaikan masalah.

II.3.2. Struktur Sistem Pakar

Ada dua bagian penting dari sistem Pakar, yaitu lingkungan pengembangan (development environment) dan lingkungan konsultasi (consultation environment). Lingkungan pengembangan digunakan oleh pembuatan sistem pakar untuk membangun komponen-komponennya dan memperkenalkan pengetahuan ke dalam *knowledge base* (basis pengetahuan). Lingkungan konsultasi digunakan oleh pengguna untuk berkonsultasi sehingga pengguna mendapatkan pengetahuan dan nasihat dari Sistem Pakar layaknya berkonsultasi degan seorang pakar. Gambar II.1 menunjukan komponen-komponen yang penting dalam sebuah sistem pakar.



Gambar II.1 komponen-komponen penting dalam sistem pakar

Komponen-komponen yang terdapat dalam arsitektur/struktur sistem Pakar :

A Antarmuka Pengguna (*user interface*)

Merupakan mekanisme yang digunakan oleh pengguna dan sistem pakar untuk berkomunikasi. Komunikasi ini paling bagus bila disajikan dalam bahasa alami (*natural language)* dan lengkapi dengan grafik, menu, dan formulir elektronik. Pada bagian ini akan terjadi dialog antara Sistem Pakar dan pengguna.

B. Basis Pengetahuan

Basis pengetahuan mengandung pengetahuan untuk pemahaman, formulasi, dan penyelesaian masalah. Basis pengetahuan itu dapat berasal dari dua elemen dasar,yaitu:

1. Fakta, Misalnya situasi, kondisi, atau permasalahan yang ada.
2. Rule (Aturan), untuk mengarahkan penggunaan pengetahuan dalam memecahkan masalah.

C. Akuisisi Pengetahuan (*Knowledge Acquisition*)

Subsistem ini digunakan untuk memasukkan pengetahuan dari seorang pakar dengan cara merekayasa pengetahuan agar bisa diproses oleh kompter dan menaruhnya ke dalam basis pengetahuan dengan format tertentu (dalam bentuk pengetahuan). Sumber-sumber pengetahuan bisa diperoleh dari pakar, buku, dokumen multimedia, basis data, laporan riset khusus, dan informasi yang terdapat diweb.

D. Mesin/Motor Inferensi (*Inference Engine*)

Mesin Inferensi (*Inference Engine*), merupakan otak dari Sistem Pakar, juga dikenal sebagai penerjemah aturan (*rule interpreter*). Komponen ini mengandung mekanisme pola pikir dan penalaran yang digunakan oleh pakar dalam menyelesaikan suatu masalah. Mesin inferensi adalah program komputer yang memberikan metodologi untuk penalaran tentang informasi yang ada dalam basis pengetahuan dan dalam *workplace*, dan untuk mem*for*mulasikan kesimpulan.

Ada 2 cara dalam melakukan inferensi :

1. *Forward Chaining* : pencocokkan fakta atau pernyataan dimulai dari fakta terlebih dahulu untuk menguji kebenaran hipotesis. Metode inferensi cocok digunakan untuk menangani masalah pengendalian (controlling) dan peramalan (prognosis).
2. *Backward Chaining* : pencocokkan fakta atau pernyataan dimulai dari hipotesis terlebih dahulu, dan untuk menguji kebenaran hipotesis tersebut harus dicari fakta-fakta yang ada dalam basis pengetahuan.

E. *Workplace / Blackboard*

Untuk merekam hasil sementara yang akan dijadikan sebagai keputusan dan untuk menjelaskan sebuah masalah yang sedang terjadi, Sistem Pakar membutuhkan *Blackboard,* yaitu area pada memori yang berfungsi sebagai basis data. Tiga tipe keputusan yang dapat direkam pada *blackboard,* yaitu:

a. Rencana : Bagaimana menghadapi masalah.
b. Agenda : Aksi-aksi potensial yang sedang menunggu untuk dieksekusi.
c. Solusi : Calon aksi yang akan dibangkitkan.

F. Fasilitas Penjelasan (*Explaination Facility*).

Berfungsi member penjelasan kepada pengguna, bagaimana suatu kesimpulan dapat diambil. Kemampuan seperti ini sangat penting bagi pengguna untuk mengetahui proses pemindahan keahlian pakar maupun dalam pemecahan masalah.

G. Sistem Perbaikan Pengetahuan

Kemampuan memperbaiki pengetahuan dari seorang pakar diperlukan untuk menganalisis pengetahuan, belajar dari kesalahan masa lalu, kemudian memperbaiki pengetahuannya sehingga dapat dipakai pada masa mendatang.

H. Pengguna (User)

Pada umumnya pengguna sistem pakar bukanlah seorang pakar yang membutuhkan solusi, saran, atau pelatihan (training) dari berbagai permasalahan yang ada.

II.3.3. Manfaat Sistem Pakar

Sistem pakar menjadi sangat populer karena sangat banyak kemampuan dan manfaat yang diberikan, diantaranya:

1. Meningkatkan produktivitas, karena Sistem Pakar dapat bekerja lebih cepat daripada manusia.
2. Membuat seorang yang awam bekerja seperti layaknya seorang pakar.
3. Meningkatkan kualitas, dengan memberikan nasehat yang konsisten dan mengurangi kesalahan.
4. Mampu menangkap pengetahuan dan kepakaran seseorang.
5. Memudahkan akses pengetahuan seorang pakar.

6. Meningkatkan kapabilitas sistem komputer. Integritas Sistem Pakar dengan sistem komputer lain membuat sistem lebih efektif dan mencangkup lebih banyak sistem.
7. Andal, sistem pakar tidak pernah menjadi bosan dan kelelahan atau sakit.
8. Mampu bekerja dengan informasi yang tidak lengkap atau tidak pasti. Berbeda dengan sistem komputer konvensional, Sistem Pakar dapat bekerja dengan informasi yang tidak lengkap.
9. Bisa digunakan sebagai media pelengkap dalam pelatihan. Pengguna pemula bekerja dengan Sistem Pakar akan menjadi lebih berpengalaman karena adanya fasilitas penjelas yang berfungsi sebagai guru.
10. Meningkatkan kemampuan untuk menyelesaikan masalah karena Sistem Pakar mengambil sumber pengetahuan dari banyak pakar.

II.3.4. Kekurangan Sistem Pakar

Selain manfaat, sistem pakar juga memiliki beberapa kelemahan, diantaranya:

1. Memerlukan biaya yang sangat mahal untuk membuat dan memelihatanya.
2. Sulit dikembangkan karena keterbatasan keahlian dan ketersediaan pakar.
3. Sistem Pakar tidak selamanya 100% bernilai benar.

II.3.5. Ciri-ciri Sistem Pakar

Sistem Pakar memiliki beberapa ciri-ciri, diantanya sebagai berikut:

1. Terbatas pada domain keahlian tertentu.
2. Dapat memberikan penalaran untuk data-data yang tidak lengkap atau tidak pasti.

3. Dapat menjelaskan alasan-alasan dengan cara yang dapat dipahami.
4. Bekerja berdasarkan kaidah/*rule* tertentu.
5. Mudah dimodifikasi
6. Basis pengetahuan dan mekanisme inferensi terpisah.
7. Keluarannya bersifat anjuran.
8. Sistem dapat mengaktifkan kaidah secara searah yang sesuai, dituntun oleh dialog dengan pengguna.

II.3.6. Konsep Dasar Sistem Pakar

Konsep dasar sistem pakar meliputi enam hal berikut ini:

II.3.6.1. Kepakaran (*Expertise*)

Kepakaran merupakan suatu pengetahuan yang diperoleh dari pelatihan, membaca dan pengalaman. Kepakaran inilah yang memungkinkan para ahli dapat mengambil keputusan lebih cepat dan lebih baik daripada seseorang yang bukan pakar. Kepakaran itu sendiri meliputi pengetahuan tentang.

1. Fakta-fakta tentang bidang permasalahan tertentu.
2. Teori-teori tentang bidang permasalahan tertentu.
3. Aturan-aturan dan prosedur-prosedur menurut bidang permsalahan umumnya.
4. Atuan *heuristic*yang harus dikerjakan dalam suatu situasi tertentu.
5. Strategi global untuk memcahkan permasalahan
6. Pengetahuan tentang pengetahuan (*meta knowledge*).

II.3.6.2. Pakar (*Expert*)

Pakar adalah seseorang yang mempunyai pengetahuan, pengalaman dan metode khusus serta mampu menerapkannya untuk memecahkan masalah atau

memberi nasehat. Seorang pakar harus mampu menjelaskan dan mempelajari hal-hal baru yang berkaitan dengan topik permasalahan, jika perlu harus mampu menyusun kembali pengetahuan-pengetahuan yang didapatkan dan dapat memecahkan aturan-aturan serta menentukan relevansi kepakarannya. Jadi seorang pakar harus mampu melakasanankan kegiatan-kegiatan berikut ini:

1. Mengenali dan memformulasikan permasalahan.
2. Memecahkan permasalahan secara cepat dan tepat.
3. Menerangkan pemecahannya.
4. Belajar dari pengalaman.
5. Merestrukturasikan pengetahuan.
6. Memecahkan aturan-aturan.
7. Menentukan relevansi.

II.3.6.3. Pemindahan Kepakaran (*Transfering Expertisi*)

Tujuan dari Sistem Pakar adalah memindahkan kepakaran dari seorang pakar ke dalam komputer, kemudian kepada orang lain yang bukan pakar. Proses ini melibatkan empat kegiatan, yaitu:

1. Akuisisi pengetahuan (dari pakar atau sumber lain).
2. Representase pengetahuan (pada komputer).
3. Inferensi pengetahuan.
4. Pemindahan pengetahuan ke pengguna.

II.3.6.4. Inferensi (*Inferencing*)

Inferensi adalah sebuah prosedur (program) yang mempunyai kemampuan dalam melakukan penalaran. Inferensi ditampilkan pada suatu komponen yang

disebut mesin inferensi yang mencangkup prosedur-prosedur mengenai pemecahan masalah. Semua pengetahuan yang dimiliki oleh seorang pakar disimpan pada basis pengetahuan oleh sistem pakar. Tugas mesih adalah mengammbil kesimpulan berdasarkan basis pengetahuan.

II.3.6.5. Aturan-aturan (*Rules*)

Kebanyakan software pakar komersil adalah sistem yang berbasis *rule* (*rule-based system*), yaitu pengetahuan disimpan terutama dalam bentuk *rule*, sebagai prosedure pemecahan masalah.

II.3.6.6. Kemampuan Menjelaskan (*Explanation Capability*)

Fasilitas lain dari sistem pakar adalah kemampuannya untuk menjelaskan saran atau rekomendasi yang diberikannya. Penjelasan dilakukan dalam subsistem yang disebut subsistem penjelasan (*explanation*). Bagian dari sistem ini memungkinkan sistem untuk memerikasa penalaran yang dibuatnya sendiri dan menjelaskan operasi-operasinya.

Karakteristik dan kemampuan yang dimiliki oleh sistem pakar berbeda dengan sistem konvensional. Perbedaan ini ditunjukkan pada Tabel II.1

Tabel II.1 Perbandingan antara Sistem Konvensional dengan Sistem Pakar

Sistem Konvensional	Sistem Pakar
Informasi dan pemprosesannyabiasanya digabungkan dalam satu program.	Basis pengetahuan diposahkan secara jelas dengan mekanisme inferensi.

Program tidak membuat kesalah (yang membuat kesalahan: Pemrogram atau Pengguna).	Program dapat berbuat kesalahan.
Biasanya tidak menjelaskan mengapa data masukan diperlakukan atau bagaimana output dihasilkan.	Penjelasan merupakan bagian terpenting dari semua sistem pakar.
Perubahan program sangat menyulitkan.	Perubahan dalam aturan-aturan mudah untuk dilakukan.
Sistem hanya bisa beroperasi setelah lengkap atau selesai.	Sistem dapat beroperasi hanya dengan aturan-aturan yang sedikit (sebagai prototipe awal).
Eksekusi dilakukan berdasarkan langkah demi langkah (*algoritmatic*).	Eksekusi dilakukan dengan menggunkaan *heuristic* dan logika pada seluruh basi pengetahuan.
Perlu informasi lengkap agar bisa beroperasi.	Dapat beroperasi dengan infomasi yang tidak lengkap atau mengandung ketidakpastian.
Menaipulasi efektif dari basis data yang besar.	Manipulasi efektif dari basis pengetahuan yang besar.

Menggunkan data.	Menggunakan pengetahuan.
Tujuan utama efisiensi.	Tujuan utama efektivitas.
Mudah berurusan dengan data kuantutatif.	Mudah berurusan dengan data kulaitatif.
Menagkap, menambah dan mendistribusikan akses ke data numerik atau informasi.	Menangkap, menambah dan mendistribusikan akses ke pertimbangan dan pengetahuan.

Sumber: (T. Sutojo;2011:165)

II.4. Penyakit Typus

Typus (*Tyhpoid Fever*) adalah penyakit infeksi bakteri, yang disebabkan oleh Salmonella Typhi . Penyakit ini ditularkan melalui konsumsi makanan atau minuman yang terkontaminasi oleh tinja atau urin orang yang terinfeksi. Hal ini terjadi untuk semua golongan umur.

Typus(*Tyhpoid Fever)* pada masyarakat dengan standar hidup dan kebersihan rendah, cenderung meningkat dan terjadi secara endemis. Biasanya angka kejadian tinggi pada daerah tropik dibandingkan daerah berhawa dingin. Sumber penularan penyakit demam tifoid adalah penderita yang aktif, penderita dalam fase konvalesen, dan kronik karier. Typus(*Tyhpoid Fever)* juga dikenali dengan nama lain yaitu *Typhus Abdominalis, Typhoid fe*ver atau *Entericfever*. Typus(*Tyhpoid Fever)* adalah penyakit sistemik yang akut yang mempunyai

karakteritik demam, sakit kepala dan ketidakenakan abdomen berlangsung lebih kurang 3 minggu yang juga disertai gejala-gejala perut pembesaran limpa dan erupsi kulit. Typus(*Tyhpoid Fever*) termasuk para-tifoid disebabkan oleh kuman *Salmonella Typhi, S paratyphi A, S paratyphi B* dan *S paratyphi C*. Jika penyebabnya adalah *S paratyphi*, gejalanya lebih ringan dibanding dengan yang disebabkan oleh *S typhi*.

Typus(*Tyhpoid Fever)* timbul akibat dari infeksi oleh bakteri golongan *Salmonella* yang memasuki tubuh penderita melalui saluran pencernaan. Sumber utama yang terinfeksi adalah manusia yang selalu mengeluarkan mikroorganisme penyebab penyakit, baik ketika ia sedang sakit atau sedang dalam masa penyembuhan. Pada masa penyembuhan, penderita pada masih mengandung Salmonella spp didalam kandung empedu atau didalam ginjal. Sebanyak 5% penderita demam tifoid kelak akan menjadi karier sementara, sedang 2 % yang lain akan menjadi karier yang menahun. Sebagian besar dari karier tersebut merupakan karier intestinal *(intestinal type*) sedang yang lain termasuk urinarytype. Kekambuhan yang yang ringan pada karier Typus(*Tyhpoid Fever)*, terutama pada karier jenis intestinal,sukar diketahui karena gejala dan keluhannya tidak jelas.

II.5. *Certainty Factor*

Teori *Certainty Factor* diusulkan oleh Shortliffe dan Buchanan pada 1975 untuk mengakomadasi ketidakpastian pemikiran (*inexact reasoning*) seorang pakar. Seorang pakar, misalnya dokter sering kali menganalisis informasi yang ada dengan ungkapan seperti “mungkin”, “kemungkinan besar”, “hampir pasti”.

Untuk mengakomodasi hal ini perlu menggunakan *Certainty Factor* untuk menggambarkan tingkat keyakinan pakar terhadap masalah yang sedang dihadapi.

Ada pun rumusan dasar dari *certainty factor*, sebagai berikut :

$$\mathbf{CF[h,e] = MB[h,e] - MD[h,e]}$$

Keterangan :

CF[h,e] = Certainty Factor dalam hipotesis h yang dipengaruhi oleh fakta e.

MB[h,e] = *Meansure of Believe*, merupakan nilai kenaikan dari kepercayaan hipotesis h dipengaruhi oleh fakta e.

MD[h,e] = *Meansure of Disbelieve*, merupakan nilai kenaikan dari ketidak percayaan hipotesis h dipengaruhi oleh fakta e.

h = Hipotesis.

e = Evidence.

1. Dengan cara yang mewawancarai seorang pakar

   Nilai CF(Rule) didapat dari interprestasi "term" dari pakar, yang diubah menjadi nilai CF tertentu.

II.6. *JAVA*

Program adalah sekumpulan perintah-perintah *(instruction*) yang diatur secara sistematis untuk menyesuaikan suatu masalah. Salah satu bahasa pemrograman komputer adalah bahasa pemrograman Java. Bahasa pemrograman Java adalah bahasa pemrograman berorientasi objek(PBO). Java bersifat netral,tridak tergantung tidak bergantung pada suatu platform, dan mengikuti prinsip WORA (*Write Once and Run Anywhere*). Disini penulis menggunakan Netbeans IDE 7.3 RC2.

II.7. *MySQL*

MySQL adalah nama *database server*. *Database server* adalah server yang berfungsi untuk menangani database. *Database* adalah sesuatu pengorganisasian data dengan tujuan memudahkan penyimpanan dan pengaksesan data. Dengan menggunakan *MySQL*, kita bisa menyimpan data dan kemudian data bisa diakses dengan cara yang mudah dan cepat.

MySQL tergolong sebagai database relasional. Pada model ini, data dinyatakan dalam bentuk dua dimensi yang secara khusus dinamakan rabel. Tabel tersusun atas baris dan kolom.

II.8. ERD (*Entity Relationship Diagram*)

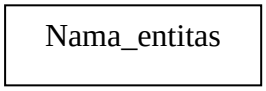
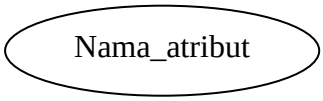
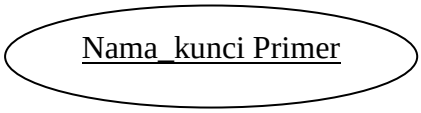
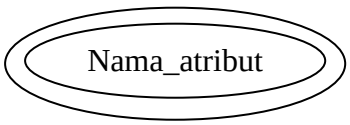
ERD (*Entity Relationship Diagram*) dikembangkan berdasarkan teori himpunan dalam bidang matematika. ERD (*Entity Relationship Diagram*) digunakan untuk pemodelan basis data relasional. Sehingga jika penyimpanan

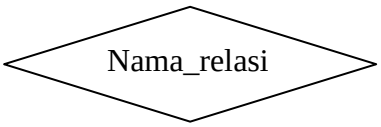
basis data menggunakan OODBMS maka perancangan basis data perlu menggunakan ERD (*Entity Relationship Diagram*) (Rosa A.S, M. Shalahuddin;2011:49).

II.8.1. Simbol-simbol ERD (*Entity Relationship Diagram*)

Adapaun simbol-simbol ERD (*Entity Relationship Diagram*) ditunjukkan pada tabel II.2.

Tabel II.2. Simbol-simbol ERD (*Entity Relationship Diagram*)

Simbol	Deskripsi
Entitas/ *entity* 	Entitas merupakan data inti yang akan disimpan; bakal tabel pada basis data
Atribut 	*Field* atau kolom data yang perlu disimpan dalam suatu entitas
Atribut kunci primer 	*Field* atau kolom data yang butuh disimpan dalam suatu entitas dan digunakan sebagai kunci akses *record* yang diinginkan; biasanya berupa id
Atribut multinal/ *multivalue* 	*Field* atau kolom data yang butuh disimpan dalam suatu entitas yang dapat memiliki nilai lebih dari satu

Relasi  	Relasi yang menghungkan antarentitas; relasi biasanya diawali dengan kata kerja
Asosiasi/ *Association* 1 0..*	Penghubung antar relasi dan entitas dimana di kedua ujungnya memiliki *multiplicity* kemungkinan jumlah pemakaian

Sumber: (Rossa A.S-M. Shalahuddin;2011:49-50)

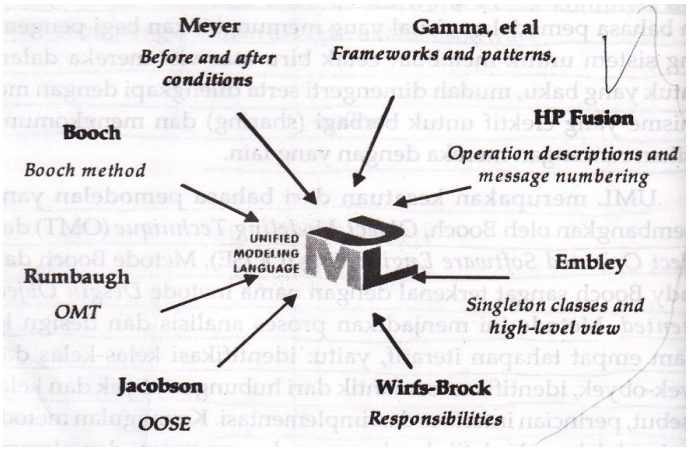
II.9. UML (*Unified Modelling Language*)

UML(*Unified Modelling Language*) adalah salah satu alat bantu yang sangat handal di dunia pengembangan sistem yang berorientasi objek. Hal ini disebabkan karena UML menyediakan bahasa pemodelan visual yang memungkinkan bagi pengembangan sistem untuk membuat cetak biru atas visi mereka dalam bentuk yang baku, mudah dimengerti serta dilengkapi dengan mekanisme yang efektif untuk berbagi (Sharing) dan mengkomunikasikan rancangan dengan lain (Munawar;2005:17).

UML merupakan kesatuan bahasa pemodelan yang dikembangkan oleh Booch, *Object Modeling Technique* (OMT) dan *object Oriented Engineering* (OOSE). Metode Booch dari Grady Booch sangat terkenal dengan nama metode *Design Object Oriented.* Metode ini menjadikan proses analisis dan design ke dalam empat tahapan iteratif, yaitu: identifikasi kelas-kelas dan objek-objek,

identifikasi semantik dari hubungan objek dan kelas tersebut, perincian interface dan implementasi. Keunggulan metode Booch adalah pada detil dan kayanya dengan notasi dan elemen. Pemodelan OMT yang dikembangkan oleh Rumbaugh didasarkan pada analisis terstruktur dan pemodelan *entity-relationship*. Tahapan utama dalam metodologi in adalah analisis, desain sistem, desain objek dan impelmentasi. Keunggulan motode ini adalah dalam penotasian yang mendukung semua konsep OO. Metode OOSE dari Jacobson lebih memberi penekanan dan *use case.* OOSE memiliki tiga tahapan yaitu membuat model *requirement* dan analisis, desain dan implementasi dan model pengujian (test Model). Keunggulan metode ini adalah mudah dipelajari karena memiliki notasi yang sederhananamun mencangkup seluruh tahapan dalam rekayasa perangkat lunak.

Dengan UML, metode Booch, OMT dan OOSE digabungkan dengan membuang elemen-elemen yang tidak praktis ditambah dengan elemen-elemen dari metode lain yang lebih efektif dan elemen-elemen baruyang belum ada pada metode terdahulu sehingga UML lebih ekspresif dan seragam daripada metode lainnya. Unsur-unsur yang membentuk UML ditunjukkan dalam Gambar II.2

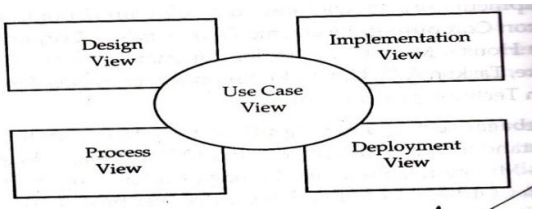


Gambar II.2 Unsur-unsur yang membentuk UML

Sumber: (Munawar;2005:18)

UML adalah hasil kerja dari konsorsium berbagai organisasi yang berhasil dijadikan sebagai standar baku dalam OOAD (*Object Oriented Analysis dan Design*). UML tidak hanya dominan dalam penotasian di lingkungan OO tetapi juga populer di luar lingkungan OO. Ada tiga karakter penting yang melekat di UML yaitu sketsa, cetak biru dan bahasa pemrograman. Sebagai sebuah sketsa UML bisa berfungsi sebagai sebuah cetak biru kerena sangat lengkap dan detil. Dengan cetak biru ini maka akan bisa diketahui informasi detil tentang coding program (*Forward Engineering*) atau bahkan membaca program dan mengiterpretasikannya kembali ke dalam diagram (*reverse engineering*). *Reverse engineering* sangat berguna pada situasi dimana kode program yang tidak terdokumentasi asli hilang atau bahkan belum dibuat sama sekali. Sebagai bahasa pemrogaraman, UML dapat menterjemahkan diagaram yang ada di UML menjadi kode program siap untuk dijalankan.

UML dibangun atas model 4+1 *view*. Model ini didasarkan pada fakta bahwa struktur sebuah sistem dideskripsikan dalam *view* dimana salah satu diantaranya *use case view*. *Use case view* ini memegang peran khusus untuk mengintegrasikan *content* ke *view* yang lain. Model 4+1 *view* ditunjukkan pada gambar II.3



Gambar II.3 Model 4+1 *View*

Sumber: (Munawar;2005:20)

Kelima *view* tersebut tidak berhubungan dengan diagram yang dideskripsikan di UML. Setiap *view* berhubungan dengan perspektif tertentu dimana sistem akan diuji. *View* yang berbeda akan menekankan pada aspek yang berbeda dari sistem yang mewakili tentang sistem bisa dibentuk dengan menggabungkan informasi-informasi yang ada pada kelima *view* tersebut.

Use case view mendefinisikan perilaku eksternal sistem. Hal ini menjadi daya tari bagi *end user,* analis dan tester. Pandangan ini mendefinisikan kebutuhan sistem karena mengandung semua *view* yang lain yang mendeskripsikan aspek-aspek tertentu dari peran dan sering dikatakan yang mendrive proses pengembangan perangkat lunak.

Design view mendeskripsikan struktur logika yang mendukung fungsi-fungsi yang dibutuhkan di *use case*. *Design view* ini berisi definisi komponen program, class-class utama bersama-sama dengan spesifikasi data, perilkau dan interaksinya. Informasi yang terkandung di *view* ini menjadi perjatian para programer karena menjelaskan secara detil bagaimana fungsionalitas sistem akan diimplementasikan.

Implemantasi *view* menjelaskan komponen-komponen fisi dari sistem yang akan dibangun. Hal ini berbeda dengan komponen logic yang dideskripsikan pada *design view.* Termasuk disini diantaranya *file exe*, *library* dan *database.* Informasi yang ada di *view* dan integrasi sistem.

Proses *view* berhubungan dengan hal-hal yang berkaitan dengan *concurrency* do dalam sistem. Sedangkan *deployment view* menjelaskan bagaimana komponen-komponen fisik didistribusikan ke lingkungna fisik seperti

jaringan komputer dimana sistem akan dijalankan. Kedua *view* ini menunjukkan kebutuhan non fungsional dari sistem seperti toleransi kesalahan dan hal-hal yang berhubungan dengan kinerja (Munawar;2005:17-21).

II.9.1. Use Case Diagram

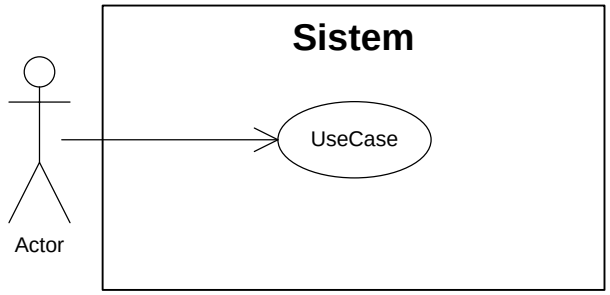
Use case adalah deskripsi fungsi dari sebuah sistem dari perspektif pengguna. *Use case* bekerja dengan cara deskripsikan tipikal interaksi antara *user* (pengguna) sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai. Urutan langkah-langkah yang menerangkan antara pengguna dan sistem disebut *scenario.* Setiap *scenario* mendeskripsikan urutan kejadian. Setiap urutan diinisialisasi oleh orang, sistem yang lain, perangkat keras atau urutan waktu. Dengan demikian secara singkat bisa dikatakan *use case* adalah serangkaian *scenario* yang digabungkan bersama-sama oleh tujuan umum pengguna.

Dalam pembicaraan tentang *use case,* pengguna biasanya disebut dengan *actor. Actor* adalah sebuah peran yang bisa dimainkan oleh pengguna dalam interaksinya dengan sistem.

Model *use case* adalah bagai dari model *requirement.* Termasuk disini adalah problem domain object dan penjelasan tentang *user interface*. *Use case* memberikan spesifikasi fungsi-fungsi yang ditawarkan oleh sistem dari *perspectif user.*

Notasi *use case* menunjukkan 3 aspek dari sistem yaitu *actor use case* dan *system/sub system boundary. Actor* mewakili peran orang, *system* yang lain atau

alat ketika berkomunikasi dengan *use case.* Ilustrasi *actor*, *usecase* dan *system* ditunjukkan pada gambar II.4



Gambar II.4*Usecase* Diagram

Sumber: (Munawar;2005:64)

Untuk mengidentifikasi *actor,* harus ditentukan pembagian tenaga kerja dan tugas-tugas yang berkaitan dengan peran pada konteks targer sistem. *Actor* adalah *abstraction* dari orang dan sistem yang lain yang mengaktifkan fungsi dari target sistem. Orang atau sistem bisa muncul dalam beberapa peran. Perlu dicatat bahwa *actor* berinteraksi dengan *use case*, tetapi tidak memiliki kontrol atas *use case.*

Use case adalah abstraksi dari interaksi antara sistem dan *actor*. Oleh karena itu sangat penting untuk memilih abstraksi yang cocok. *Use case* dibuat berdasarkan keperluan *actor*. *Use case* harus merupakan 'apa' yang dikerjakan *software* aplikasi, bukan 'bagaimana' *software* aplikasi mengerjakannya. Setiap *use case* harus diberi nama yang menyatakan apa hal yang dicapai dari hasil interaaksinya dengan *actor*. Namun *use case* boleh terdiri dari beberapa kata dan tidak boleh ada dua *use case* yang memiliki nama yang sama (Munawar;2005:63-66).

II.9.2. Class Diagram

Class adalah sebuah spesifikasi yang jika diinstansiasiakan 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.

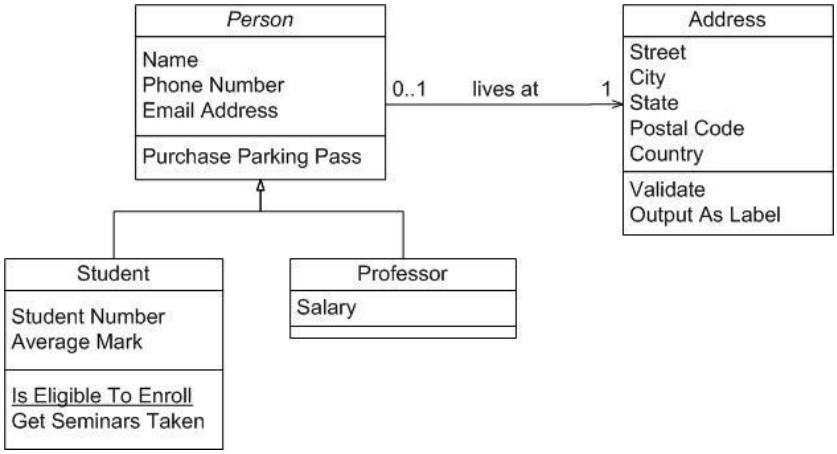
Class memiliki tiga area pokok:

1. Namakelas
2. Atribut
3. Metode

Atribut dan metode dapat memiliki salah satu sifat berikut :

1. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan.
2. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan.
3. *Public*, dapat dipanggil oleh siapa saja.

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metode. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class*. Contoh diagram *class* dapat dilihat pada gambar II.5 dibawah ini:



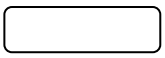
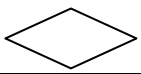
Gambar II.5*Class* Diagram

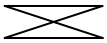
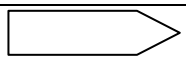
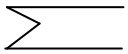
Sumber: (Munawar;2005:220)

II.9.3. Activity Diagram

*Activity Diagram*adalah teknik untuk mendiskripsikan logika prosedural, proses bisnis dan aliran kerja dalam banyak kasus.*Activity Diagram* mempunyai peran seperti halnya *flowchart*, akan tetapi perbedaannya dengan *flowchart* adalah *activity* diagram bisa mendukung perilaku paralel sedangkan flowchart tidak bisa.Adapun simbol *activity diagram* dapat dilihat pada table II.3.

Tabel II.3. Simbol *Activity Diagram*

Notasi	Keterangan
	Titik Awal
	Titik Akhir
	*Activity*
	Pilihan untuk pengambilan keputusan
	*Fork*digunakan untuk menunjukkan kegiatan yang dilakukan secara paralel atau untuk menggabungkan dua kegiatan paralel menjadi

	satu
	*Rake*menunjukkanadanyadekomposisi
	Tandawaktu
	Tandapengiriman
	Tandapenerimaan
	AliranAkhir (*Flow Final*)

Sumber : (Munawar;2005:110)

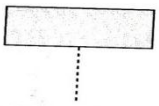
II.9.4. Squence Diagram

Sequence diagram digunakan untuk menggambarkan perilaku pada sebuah sekenario. Diagram ini menunjukkan sejumlah contoh objek dan pesan yang diletakkan diantaran objek-objek ini di dalam *use case.*

Komponen utama *sequence diagram* terdiri atas objek yang ditulisakan dengan kotak segiempat bernama. *Messege* diwakili oleh garis dengan tanda panah dan waktu yang ditunjukkan dengan *progress vertical.*

1. Objek /*participant*

Objek diletakkan di dekat bagian atas diagram dengan urutan dari kiri ke kanan. Mereka diatur dalam urutan guna menyederhanakan diagram. Setiap *participant* dihubungkan dengan garis titik-titik yang disebut *lifeline.* Sepanjang *lifeline* ada kotak yang disebut *activation. Activation* mewakili sebuah eksekusi operasi dari *participant.* Panjang kotak ini berbanding lurus dengan durasi *activation.* Bentuk *participant* dapat dilihat pada gambar II.6.



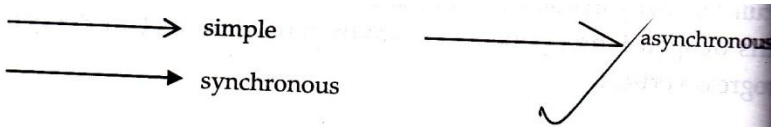
Gambar II.6 Bentuk *Participant*

Sumber: (Munawar;2005:88)

2. *Messege*

Sebuah *messsage* bergerak dari satu *participant* ke *participant* yang lain dan dari satu *lifeline* ke *lifeline* yang lain. Sebuah *participant* bisa mengirim sebuah *message* kepada dirinya sendiri.

Sebuah *message* bisa jadi *simple, synchronous* atau *asynchoronous. Message* yang *simple* adalah sebuah perpindahan (transfer), contoh dari satu *participant* ke *participant*yang lainnya. Jika sebuah *participant*mengirimkan sebuah *messagae* tersebut akan ditunggu sebelum diproses dengan urusannya. Namun jika *message asynchoronous* yang dikirimkan, maka jawabannya atas *message* tersebut tidak perlu ditunggu. Simbol *message* pada *squnence diagram* dapat dilihat pada gambar II.7.



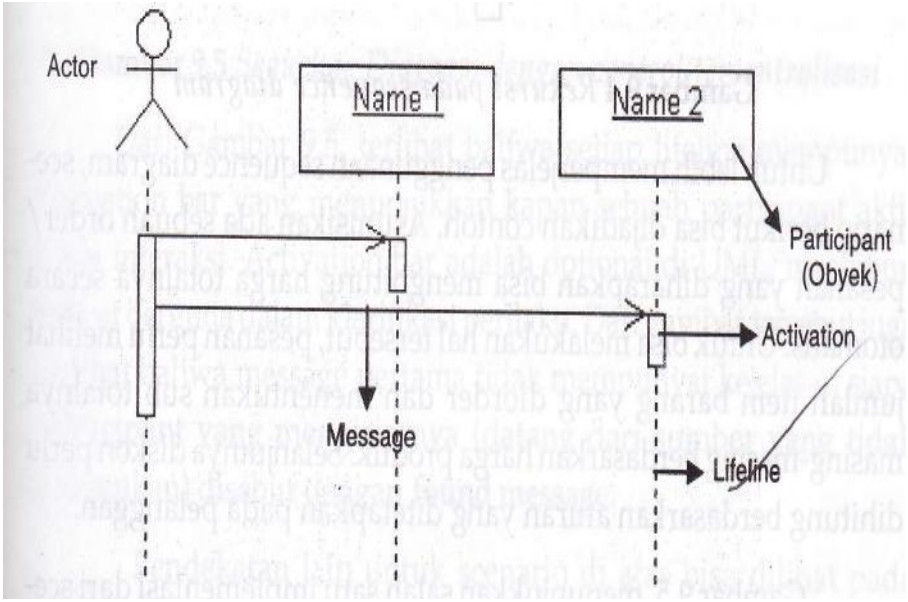
Gambar II.7 Bentuk *Messege*

Sumber: (Munawar,2005:88)

3. Time

Time adalah diagram yang mewakili waktu pada arah vertikal. Waktu dimulai dari ata ke bawah. *Message* yang lebih dekat dari atas akan dijalankan terlebih dahulu dibanding *message* yang lebih dekat ke bawah.

Terdapat dua dimensi pada *squence diagram* yaitu dimensi dari kiri ke kanan menunjukkan tata letak *participant* dan dimensi dari atas ke bawah menunjukkan lintasan waktu. Simbol-simbol yang ada pada *squence diagram* ditunjukkan pada gambar II.8



Gambar II.8 *Squence Diagaram*

Sumber: (Munawar,2005:89)

II.10. Normalisasi

Normalisasi merupakan bentuk transformasi tinjauan pemakai yang kompleks dimana data tersimapan ke dalam sekumpulan bagian struktur data yang kecil dan stabil. Normalisasi merupakan kegiatan perlakuan data untuk

menyederhanakan sebuah tabel data agar lebih terstruktur dan mudah digunakan (Idris Asmuni;2005:4).

Pemahaman normalisasi merupakan keterampilan yang harus diperhatikan oleh programmer sistem dengan mengadakan pengamatan dan analisis yang memadai pada formulir atau dokumen masukan menjadi laporan utama (Idris Asmuni;2005:4).

Tahapan normalisasi terdiri dari beberapa bentuk yaitu sebagai berikut:

1. Bentuk Normal Pertama (1NF/*First Normal Form*)

Bentuk normal pertama memiliki ciri yaitu data berbentuk *flat file* (file datar), *record* disusun sesuai kedatangan, masih mungkin terjadi penyimpangan data (*anomali* data). *Anomali* data dapat berupa *insert anomali, delete anomali, update anomali* dan *redudancy* data (data duplikat).

2. Bentuk Normal Kedua (2NF/*Second Normal Form*)

Bentuk normal kedua memiliki ciri yaitu tidak terjadi *anomali* data, setiap *field/ atribute* bukan kunci harus tergantung fungsi (*functional dependency*) terhadap *field/ atribute* kunci, masih mungkin terjadi *transitive dependency* (*field* bukan kunci tergantung pada *field* bukan kunci dalam satu tabel).

3. Bentuk Normal Ketiga (3NF/ *Third Normal Form*)

Bentuk normal ketiga memuliki syarat yaitu tabel harus tidak terdapat *transitive dependency* (*field* bukan kunci tergantung pada *field* bukan kunci dalam satu tabel).

4. Bentuk Normal B*oyce Codd* (BCNF/*Boyce Codd Normal Form*)

Pada tahap ini menghilangkan ketergantungan *field* bukan kunci secara persial (bagian) kunci dalam satu tabel. Apabila pada normal ketiga tidak lagi ditemukan *field* bukan kunci tergantung secara persial dalam satu tabel, maka normal ketiga juga merupakan bentuk BCNF.