BAB II

TINJAUAN PUSTAKA

II.1. Kecerdasan Buatan

Kecerdasan Buatan (*Artificial Intelligence*) merupakan salah satu bagian dari ilmu komputer yang mempelajari bagaimana membuat mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang dilakukan oleh manusia bahkan bisa lebih baik daripada yang dilakukan manusia. Aplikasi atau program kecerdasan buatan dapat ditulis dalam semua bahasa komputer, baik dalam bahasa C, *pascal*, *basic* dan bahasa pemrograman lainnya.

Menurut John McCarthy, 1956, AI adalah untuk mengetahui dan memodelkan proses-proses berpikir manusia dan mendesain mesin agar dapat menirukan perilaku manusia. Cerdas, berarti memiliki pengetahuan ditambah pengalaman, penalaran (bagaimana membuat keputusan dan mengambil tindakan), moral yang baik.

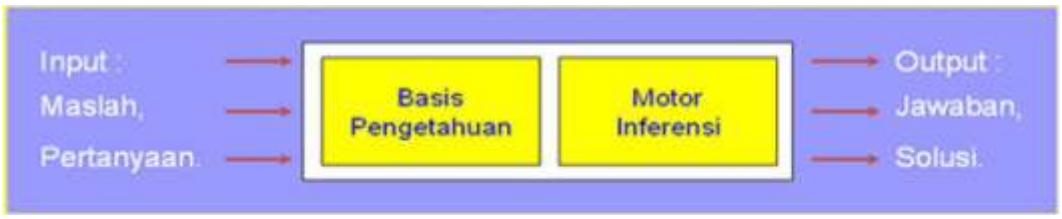
Manusia cerdas (pandai) dalam menyelesaikan permasalahan karena manusia mempunyai pengetahuan dan pengalaman. Pengetahuan diperoleh dari belajar. Semakin banyak bekal pengetahuan yang dimiliki tentu akan lebih mampu menyelesaikan permasalahan. Tapi bekal pengetahuan saja tidak cukup, manusia juga diberi akal untuk melakukan penalaran, mengambil kesimpulan berdasarkan pengetahuan dan pengalaman yang dimiliki. Tanpa memiliki kemampuan untuk menalar dengan baik, manusia dengan segudang pengalaman dan pengetahuan tidak akan dapat menyelesaikan masalah dengan baik. Demikian

juga dengan kemampuan menalar yang sangat baik, namun tanpa bekal pengetahuan dan pengalaman yang memadai, manusia juga tidak akan bisa menyelesaikan masalah dengan baik.

Demikian juga agar mesin bisa cerdas (bertindak seperti dan sebaik manusia) maka harus diberi bekal pengetahuan, sehingga mempunyai kemampuan untuk menalar. Untuk membuat aplikasi kecerdasan buatan ada 2 (dua) bagian utama yang sangat dibutuhkan, yaitu :

1. Basis Pengetahuan (*Knowledge Base*), bersifat fakta-fakta, teori, pemikiran dan hubungan antar satu dengan yang lainnya.
2. Motor Inferensi (*Inference Engine*), kemampuan menarik kesimpulan berdasarkan pengetahuan dan pengalaman. (Muhammad Dahria ; 2008 : 185)

Penerapan konsep kecerdasan buatan pada komputer adalah sebagai berikut :



Gambar II.1. Penerapan Konsep Kecerdasan Buatan Di Komputer

(Sumber : Muhammad Dahria ; 2008 : 186)

II.2. Sistem Pakar

Istilah sistem pakar (*expert system*) berasal dari istilah sistem pakar berbasis pengetahuan. Sistem pakar adalah suatu sistem yang menggunakan pengetahuan manusia yang terekam dalam komputer untuk memecahkan persoalan yang biasanya memerlukan keahlian manusia. Sistem pakar diterapkan untuk mendukung aktivitas pemecahan masalah.

Sistem pakar merupakan cabang dari kecerdasan buatan (*Artificial Intelligence*) yang cukup tua karena sistem ini mulai dikembangkan pada pertengahan 1960. Sistem ini bekerja untuk mengadopsi pengetahuan manusia ke komputer yang menggabungkan dasar pengetahuan untuk menggantikan seorang pakar dalam menyelasaikan suatu masalah. Sistem pakar berasal dari istilah *knowledge base expert system*. Sistem pakar adalah suatu sistem yang dirancang agar dapat menyelesaikan suatu permasalahan tertentu dengan meniru kerja dari para ahli dalam menjawab pertanyaan dan memecahkan suatu masalah. Dengan sistem pakar ini orang awam pun dapat menyelesaikan masalah yang cukup rumit yang sebenarnya hanya dapat diselesaikan dengan bantuan para ahli. Bagi para ahli sistem pakar ini juga membantu aktivitasnya sebagai asisten yang sangat berpengalaman. (Dodi Harto ; 2013 : 23)

II.2.1. Keuntungan Sistem Pakar

Ada banyak keuntungan bila menggunakan sistem pakar, diantaranya adalah sebagai berikut :

1. Menjadikan pengetahuan dan nasihat lebih mudah didapat.
2. Meningkatkan *output* dan produktivitas.
3. Menyimpan kemampuan dan keahlian pakar.
4. Meningkatkan penyelesaian masalah, menerusi paduan pakar, penerangan, sistem pakar khas.
5. Meningkatkan reliabilitas.
6. Memberikan *respons* (jawaban) yang cepat.
7. Merupakan panduan yang *intelligence* (cerdas).

8. Dapat bekerja dengan informasi yang kurang lengkap dan mengandung ketidakpastian.
9. *Intelligence database* (basis data cerdas), bahwa sistem pakar dapat digunakan untuk mengakses basis data dengan cara cerdas (Muhammad Arhami ; 2004 : 10)

II.2.2. Kelemahan Sistem Pakar

Selain keuntungan-keuntungan diatas, sistem pakar seperti halnya sistem lainnya, juga memiliki kelemahan, diantaranya adalah sebagai berikut :

1. Masalah dalam mendapatkan pengetahuan di mana pengetahuan tidak selalu bisa didapatkan dengan mudah, karena kadangkala pakar dari masalah yang kita buat tidak ada dan kalaupun ada kadang-kadang pendekatan yang dimiliki oleh pakar berbeda-beda.
2. Untuk membuat sistem pakar yang benar-benar berkualitas tinggi sangatlah sulit dan memerlukan biaya yang sangat besar untuk pengembangan dan pemeliharaannya.
3. Boleh jadi sistem tak dapat membuat keputusan.
4. Sistem pakar tidaklah 100% menguntungkan, walaupun seorang tetap tidak sempurna atau tidak selalu benar. Oleh karena itu perlu diuji ulang secara teliti sebelum digunakan. Dalam hal ini peran manusia tetap merupakan faktor dominan.

Kelemahan-kelemahan atau kekurangan sistem pakar tersebut bukanlah sama sekali tidak bisa diatasi, tetapi dengan terus melakukan perbaikan dan pengolahan berdasarkan pengalaman yang telah ada maka hal itu diyakini akan

dapat diatasi, walaupun dalam waktu yang panjang dan terus menerus. (Muhammad Arhami ; 2004 : 10)

II.2.3. Karakteristik Sistem Pakar

Sistem pakar umumnya dirancang untuk memenuhi beberapa karakteristik umum berikut ini :

a. **Kinerja yang sangat baik** (*high performance*). Sistem harus mampu memberikan respon berupa saran (*advice*) dengan tingkat kualitas yang sama dengan seorang pakar atau melebihinya.
b. **Waktu respon yang baik** (*adequate respon time*). Sistem juga harus mampu bekerja dalam waktu yang sama baiknya (*reasonable*) atau lebih cepat dibandingkan dengan seorang pakar dalam menghasilkan keputusan. Hal ini sangat penting terutama pada sistem waktu nyata (*real-time*).
c. **Dapat diandalkan** (*good reliability*). Sistem harus dapat diandalkan dan tidak mudah rusak (*crash*).
d. **Dapat dipahami** (*understandable*). Sistem harus mampu menjelaskan langkah-langkah penalaran yang dilakukannya seperti seorang pakar. Hal ini penting untuk beberapa alasan, yaitu :
    1. Dimungkinkan bahwa sistem pakar berkaitan dengan nyawa manusia atau properti lainnya sehingga harus dapat menjelaskan mengapa dihasilkan suatu kesimpulan tertentu.
    2. Untuk mengkonfirmasi bahwa pengetahuan pakar telah dikumpulkan dengan benar dan digunakan oleh sistem yang benar pula. Hal ini penting

dalam proses *debugging* pengetahuan yang mungkin salah karena pengetikan atau pemahaman yang salah dari *knowledge engineer*.

e. **Fleksibel** (*flexibility*). Sistem harus menyediakan mekanisme untuk menambah, mengubah dan menghapus pengetahuan. (Rika Rosnelly ; 2011 : 20).

II.2.4. Ciri-Ciri Sistem Pakar

Sistem pakar merupakan program-program praktis yang menggunakan strategi *heuristic* yang dikembangkan oleh manusia untuk menyelesaikan permasalahan-permasalahan yang spesifik (khusus). Disebabkan oleh keheuristikannya dan sifatnya yang berdasarkan pada pengetahuan, maka umumnya sistem pakar bersifat sebagai berikut :

1. Memiliki informasi yang handal, baik dalam menampilkan langkah-langkah antara maupun dalam menjawab pertanyaan-pertanyaan tentang proses penyelesaian.
2. Mudah dimodifikasi, yaitu dengan menambah atau menghapus suatu kemampuan dari basis pengetahuan.
3. Heuristik dalam menggunakan pengetahuan (yang sering kali tidak sempurna) untuk mendapatkan penyelesaiannya.
4. Dapat digunakan dalam berbagai jenis komputer.
5. Memiliki kemampuan untuk belajar beradaptasi. (Arhami ; 2005 : 23)

II.2.5. Pakar

Seorang pakar adalah orang yang mempunyai keahlian dalam bidang tertentu, yaitu pakar yang mempunyai *knowledge* atau kemampuan khusus yang orang lain tidak mengetahui atau mampu dalam bidang yang dimilikinya. (Muhammad Arhami ; 2004 : 3)

Seorang pakar memiliki kemampuan kepakaran, yaitu :

1. Dapat mengenali dan merumuskan suatu masalah.
2. Menyelesaikan masalah dengan cepat dan tepat.
3. Menjelaskan solusi dari suatu masalah.
4. Restrukturisasi pengetahuan.
5. Belajar dari pengalaman.
6. Memahami batas kemampuan. (Rika Rosnelly ; 2011 : 10)

II.3. Metode *Forward Chaining*

Forward Chaining merupakan pencocokan fakta atau pernyataan yang dimulai dari bagian sebelah kiri (IF dulu). Dengan kata lain, penalaran dimulai dari fakta terlebih dahulu untuk menguji kebenaran hipotesis. (Rika Rosnelly ; 2011 : 57)

Tabel II.1. Contoh Aturan-Aturan *Forward Chaining* :

No	Aturan
R-1	IF A & B THEN C
R-2	IF C THEN D
R-3	IF A & E THEN F
R-4	IF A THEN G
R-5	IF F & G THEN D
R-6	IF G & E THEN H
R-7	IF C & H THEN I
R-8	IF I & A THEN J
R-9	IF G THEN J
R-10	IF J THEN K

(Sumber : Rika Rosnelly ; 2011 : 57)

Contoh *Forward Chaining* :

1. Pada **Tabel II.1.** Terlihat ada 10 aturan yang tersimpan dalam basis pengetahuan. Fakta awal yang diberikan hanya : A & F (artinya : A dan F bernilai benar). Ingin dibuktikan apakah K bernilai Benar (Hipotesis : K) ?
2. Langkah-langkah inferensi adalah sebagai berikut :
3. Dimulai dari R-1. A merupakan fakta sehingga bernilai benar, sedangkan B belum bisa diketahui kebenarannya, sehingga C-pun juga belum bisa diketahui kebenarannya. Oleh karena itu kita tidak mendapatkan informasi apapun pada R-1 ini. Sehingga kita menuju ke R-2.
4. Pada R-2, kita tidak mengetahui informasi apapun tentang C, sehingga kita juga tidak bisa memastikan kebenaran D. Oleh karena itu kita tidak mendapatkan informasi apapun pada R-1 ini. Sehingga kita menuju ke R-3.
5. Pada R-3, baik A maupun E adalah fakta sehingga jelas benar. Dengan demikian F sebagai konsekuen juga ikut benar. Sehingga sekarang kita

mempunyai fakta baru yaitu F. Karena F bukan hipotesis yang hendak kita buktikan (=K), maka penelusuran kita lanjutkan ke R-4.

6. Pada R-4, A adalah fakta sehingga jelas benar. Dengan demikian G sebagai konsekuen juga ikut benar. Sehingga sekarang kita mempunyai fakta baru yaitu G. Karena G bukan hipotesis yang hendak kita buktikan (=K), maka penelusuran kita lanjutkan ke R-5.
7. Pada R-5, baik F maupun G bernilai benar berdasarkan aturan R-3 dan R-4. Dengan demikian D sebagai konsekuen juga ikut benar. Sehingga sekarang kita mempunyai fakta baru yaitu D. Karena D bukan hipotesis yang hendak kita buktikan (=K), maka penelusuran kita lanjutkan ke R-6.
8. Pada R-6, baik A maupun G adalah benar berdasaran fakta dan R-4. Dengan demikian H sebagai konsekuen juga ikut benar. Sehingga sekarang kita mempunyai fakta baru yaitu H. Karena H bukan hipotesis yang hendak kita buktikan (=K), maka penelusuran kita lanjutkan ke R-7.
9. Pada R-7, meskipun H benar berdasarkan R-6, namun kita tidak tahu kebenarannya. Oleh karena itu kita tidak mendapatkan informasi apapun pada R-7 ini, sehingga kita menuju ke R-8.
10. Pada R-8, meskipun A benar karena fakta, namun kita tidak tahu kebenaran I, sehingga J-pun juga belum bia diketahui kebenarannya. Oleh karena itu kita tidak mendapatkan informasi apapun pada R-8 ini. Sehingga kita menuju ke R-9.

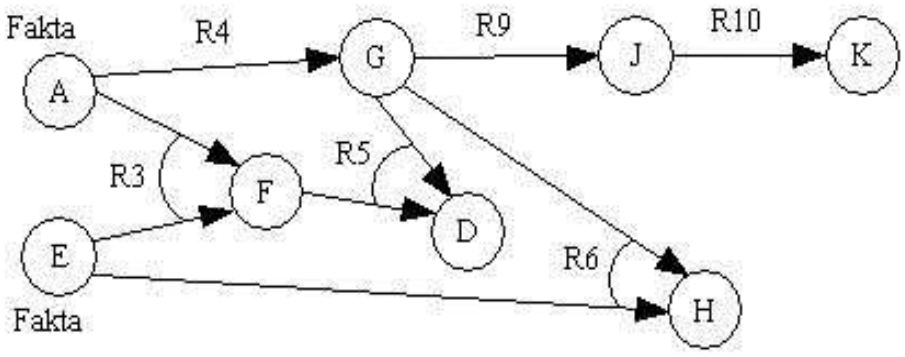
11. Pada R-9, J bernilai benar karena G benar berdasarkan R-4. Karena J bukan hipotesis yang hendak kita buktikan (=K), maka penelusuran kita lanjutkan ke R-10.
12. Pada R-10, bernilai benar karena J benar berdasarkan R-9. Karena H sudah merupakan hipotesis yang hendak kita buktikan (=K), maka terbukti bahwa K adalah benar. (Rika Rosnelly ; 2011 : 58)

Tabel munculnya fakta baru pada saat inferensi terlihat pada **Tabel II.2.** Sedangkan alur inferensi terlihat pada **Gambar II.2.**

Tabel II.2. Fakta Baru

Aturan	Fakta baru
R-3	F
R-4	G
R-5	D
R-6	H
R-9	J
R-10	K

(Sumber : Rika Rosnelly ; 2011 : 59)



Gambar II.2. ***Forward Chaining***

(Sumber : Rika Rosnelly ; 2011 : 60)

II.4. Microsoft *Visual Basic* 2010

Visual Basic diturunkan dari bahasa *BASIC. Visual Basic* terkenal sebagai bahasa pemrograman yang mudah untuk digunakan terutama untuk membuat aplikasi yang berjalan diatas *platform windows*.

Pada tahun 90-an, *Visual Basic* menjadi bahasa pemrograman yang paling populer dan menjadi pilihan utama untuk mengembangkan program berbasis *windows.* Versi *Visual Basic* terakhir sebelum berjalan diatas *.NET Framework* adalah VB 6.0 (*Visual Studio* 1998).

Visual Basic .NET dirilis pada februari 2002 bersamaan dengan *platform .NET Framework* 1.0. Kini sudah ada beberapa versi dari *Visual Basic* yang berjalan pada *platform .NET*, yaitu VB 2002 (VB7), VB 2005 (VB8), VB 2008 (VB9) dan terakir VB 2010 (VB10) yang dirilis bersamaan dengan *Visual Studio* 2010.

Selain *Visual Basic* 2010, *Visual Studio* 2010 juga mendukung beberapa bahasa lain, yaitu C#, C++, F# (bahasa baru untuk *functional programming*). *IronPhyton* dan *IronRuby* (bahasa baru untuk *dynamic programming*). (Eric Kurniawan ; 2011 : 1).

Kemampuan atau manfaat dari *Visual Basic* adalah sebagai berikut :

1. Untuk membuat program aplikasi maupun animasi berbasis *windows*.
2. Untuk membuat objek-objek *add-in* seperti *control Active*X, *File*, *help*, aplikasi internet dan lain sebagainya.
3. Menguji program (*debugging*) dan menghasilkan program (.exe) yang bersifat *executable*. (R. M. Nasrul Halim ; 2011 : 285)

Dapat diambil kesimpulan, secara umum keuntungan *Visual basic* adalah bahasa sederhana, karena sangat populer maka sangat banyak sumber-sumber yang digunakan untuk memperoleh banyak *tools* gratis.

Sedangkan kekurangan *Visual basic* adalah bahasanya *powerful*, tetapi sebenarnya tidak terlalu bagus untuk membuat game-game, lebih lambat dibandingkan bahasa pemrograman lain.

II.5. Microsoft SQL Server 2008

SQL *Server* 2008 adalah sebuah terobosan baru dari *Microsoft* dalam bidang *database*. SQL Server adalah DBMS (*Database Management System*) yang dibuat oleh *Microsoft* untuk ikut berkecimpung dalam persaingan dunia pengolahan data menyusul pendahulunya seperti IBM dan *Oracle*. SQl *Server* 2008 dibuat pada saat kemajuan dalam bidang *hardware* sedemikian pesat. Oleh karena itu sudah dapat dipastikan bahwa SQL *Server* 2008 membawa beberapa terobosan dalam bidang pengolahan dan penyimpanan data.

Microsoft merilis SQL *Server* 2008 dalam beberapa versi yang disesuaikan dengan segment-segment pasar yang dituju. Versi-versi tersebut adalah sebagai berikut. Menurut cara pemrosesan data pada prosesor maka *Microsoft* mengelompokkan produk ini berdasarkan 2 (dua) jenis, yaitu :

1. Versi 32-*bit* (x86), yang biasanya digunakan untuk komputer dengan satu prosesor (Pentium 4) atau lebih tepatnya prosesor 32 *bit* dan sistem operasi *Windows* XP.

2. Versi 64-*bit* (x64), yang biasanya digunakan untuk komputer dengan lebih dari satu prosesor (Misalnya *Core* 2 *Duo*) dan sistem operasi 64 *bit* seperti *Windows* XP 64, *Vista* dan *Windows* 7.

Sedangkan secara keseluruhan terdapat versi-versi seperti berikut ini :

1. Versi *Compact*, ini adalah versi "Tipis" dari semua versi yang ada. Versi ini seperti versi *desktop* pada SQL *Server* 2000. Versi ini juga digunakan pada *handheld device* seperti *Pocket* PC, PDA, *Smartphone*, *Tablet* PC.
2. Versi *Express*, ini adalah versi "Ringan" dari semua versi yang ada (tetapi versi ini berbeda dengan versi *compact*) dan paling cocok untuk latihan para pengembang aplikasi. Versi ini memuat *Express Manager standard*, integrasi dengan CLR dan XML. (Wahana Komputer ; 2010 : 2)

II.6. *Database*

Istilah *database* banyak memiliki definisi. Untuk sebagian kalangan sederhana *database* diartikan sebagai kumpulan data (buku, nomor telepon, daftar pegawai dan lain sebagainya). Ada juga yang menyebut *database* dengan definisi lain yang lebih formal dan tegas. *Database* didefenisikan sebagai kumpulan data yang terintegrasi dan diatur sedemikian rupa sehingga data tersebut dapat dimanipulasi, diambil dan dicari secara cepat.

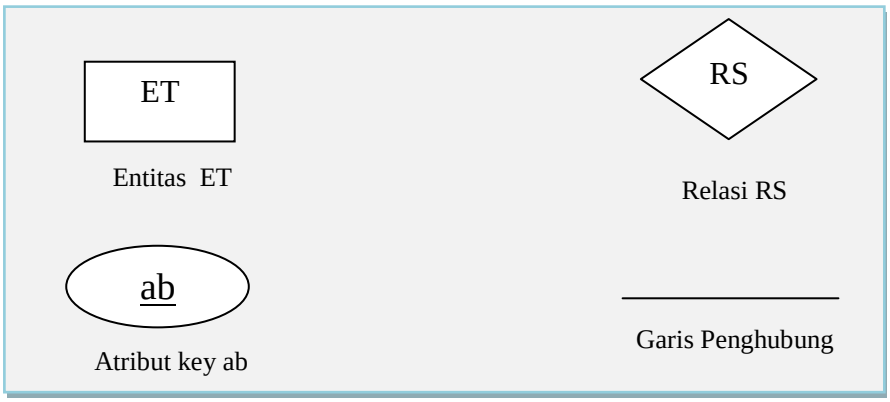
Selain berisi data, *database* juga berisi *metadata*. *Metadata* adalah data yang menjelaskan tentang struktur dari data itu sendiri. Sebagai contoh, anda dapat memperoleh informasi tentang nama-nama kolom dan tipe yang ditampilkan tersebut disebut *metadata*. **(**Budi Raharjo ; 2011 : 3)

II.6.1. Pemodelan Data

Terdapat beberapa penjelasan mengenai pemodelan basis data. Suatu basis data dapat digunakan secara bebas untuk menggambarkan dan memberikan deskripsi mengenai kumpulan informasi uang tersimpan dalam *data storage* komputer. Secara sederhana, defenisi untuk model basis data adalah sekumpulan notasi atau simbol untuk menggambarkan data dan relasinya, berdasarkan suatu konsep dan aturan tertentu suatu pemodelan. (Yudi Priyadi ; 2014 : 10)

II.6.2. Notasi Diagram E-R

Pemodelan basis data dengan menggunakan diagram relasi antar entitas, dapat dilakukan dengan menggunakan suatu pemodelan basis data yang bernama Diagram *Entity-Relational* (selanjutnya disingkat Diagram E-R). Pada **Gambar II.3.**, terdapat suatu simbol atau notasi dasar yang digunakan pada Diagram E-R, yaitu entitas, relasi, atribut dan garis penghubung. (Yudi Priyadi ; 2014 : 20)



Gambar II.3. : Notasi Dasar Diagram E-R

(Sumber : Yudi Priyadi ; 2014 : 20)

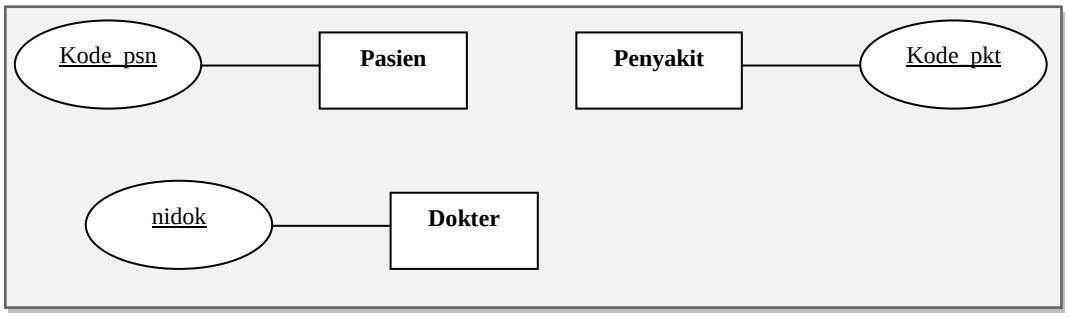
1. Entitas

   Merupakan notasi untuk mewakili suatu objek dengan karakteristik sama yang dilengkapi oleh atribut, sehingga pada suatu lingkungan nyata setiap

objek akan berbeda dengan objek lainnya. Pada umumnya, objek dapat berupa benda, pekerjaan, tempat dan orang.

2. Atribut

Merupakan notasi yang menjelaskan karakteristik suatu entitas dan juga relasinya. Atribut dapat sebagai *key* yang bersifat unik, yaitu *Primary Key* atau *Foreign Key*. Selain itu, atribut juga dapat sebagai atribut deskriptif saja, yaitu sebagai pelengkap deskripsi suatu entitas dan relasi.

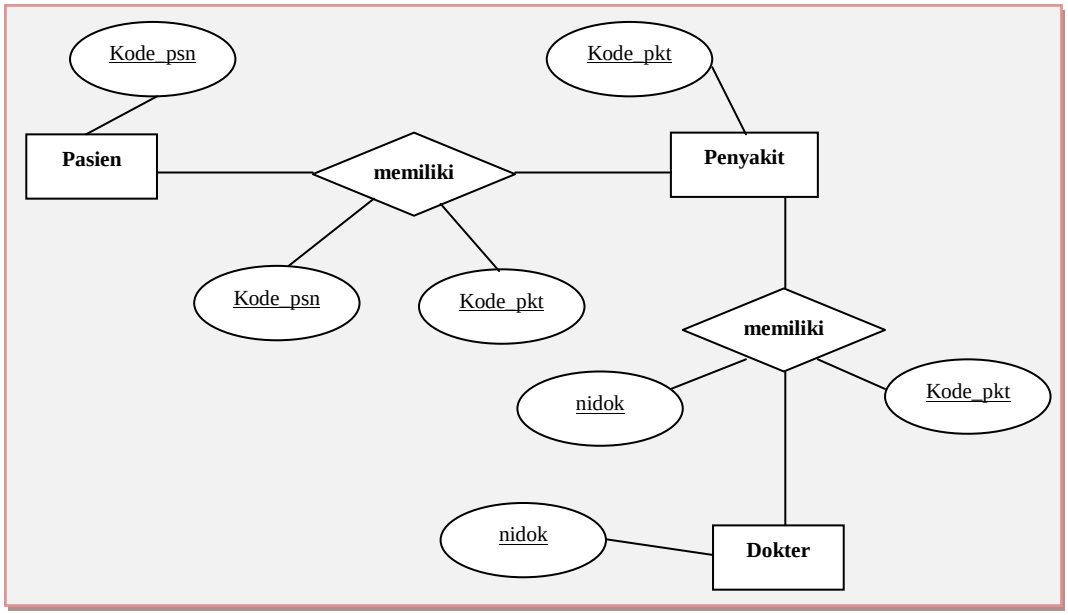


Gambar II.4. : Atribut Key pada Entitas

(Sumber : Yudi Priyadi ; 2014 : 23)

3. Relasi

Merupakan notasi yang digunakan untuk menghubungkan beberapa entitas berdasarkan fakta pada suatu lingkungan.



Gambar II.5. : Pemilihan Relasi untuk Entitas

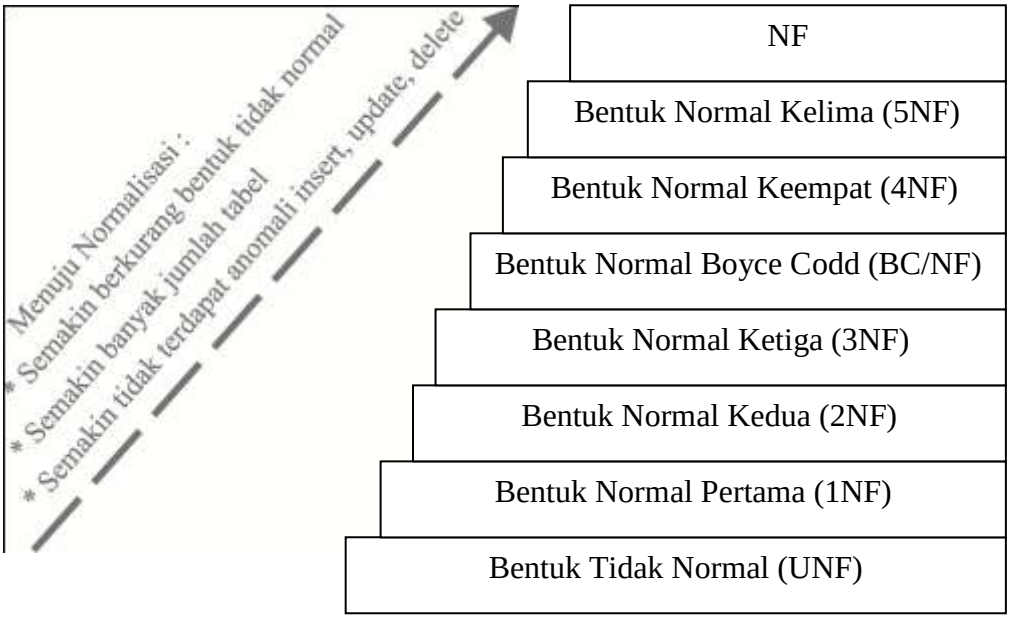
(Sumber : Yudi Priyadi ; 2014 : 25)

4. Garis penghubung

   Merupakan notasi untuk merangkaikan keterkaitan antar notasi yang digunakan dalam Diagram E-R, yaitu entitas, relasi dan atribut.

II.6.3. Normalisasi

Normalisasi merupakan proses sistematis yang dilakukan pada struktur tabel basis data menjadi struktur tabel yang memiliki integritas data, sehingga tidak memiliki data anomali pada saat melakukan *insert*, *delete* dan *update*. Pada **Gambar II.6.**, tahapan proses sistematis yang dilakukan mulai dari bentuk tidak normal menjadi bentuk normal memiliki suatu syarat yang harus dipenuhi pada saat menuju suatu bentuk yang lebih baik (*well structured relation*).



Gambar II.6. : Tahapan Proses Bentuk Normalisasi

(Sumber : Yudi Priyadi ; 2014 : 67)

Setiap syarat dalam tahapan suatu bentuk normal memiliki keterkaitan, hal ini disebabkan karena pada setiap bentuk normal mengalami penyempurnaan untuk bentuk normal selanjutnya. Bentuk tidak normal akan semakin berkurang, setelah melalui tahapan perubahan bentuk normalisasi, sehingga berdampak pada jumlah tabel yang semakin banyak, tetapi menuju perbaikan ke dalam bentuk *well structured relation.* Hal ini terjadi akibat dari pengelompokan data suatu tabel agar memiliki ketergantungan secara fungsional. **(**Yudi Priyadi ; 2014 : 67)

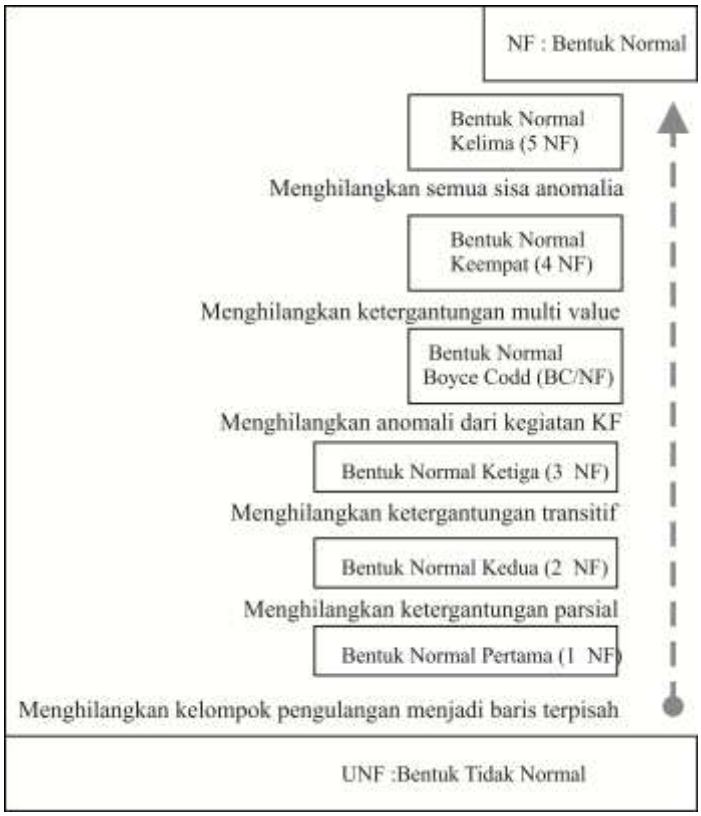
II.6.4. Aturan Proses Normalisasi

Secara sederhana, kegiatan normalisasi adalah melakukan dekomposisi atau penguraian tabel beserta datanya, menjadi tabel yang normal menurut konsep RDBMS. Merujuk pada **Gambar II.7.**, dekomposisi diawali dengan melakukan analisis pada suatu tabel atau beberapa contoh formulir yang sudah memiliki data

lengkap dalam basis data, tetapi masih dalam bentuk yang tidak normal (UNF). Oleh karena itu agar dapat memenuhi syarat bentuk normal pertama (1NF), pada setiap barisnya diisikan suatu *value* dengan kelompok data yang sama, berdasarkan suatu atribut *key*. Dengan demikian, kelompok pengulangan dalam suatu baris dapat dihilangkan, karena sudah tidak terdapat *value* yang kosong untuk setiap *field* dan *record*-nya.

Setelah memenuhi syarat bentuk normal pertama (1NF), proses berikutnya adalah menghilangkan ketergantungan secara parsial, yaitu dengan cara melakukan dekomposisi tabel menjadi beberapa kelompok tabel berdasarkan *field* yang memiliki status sebagai *key*. Hal ini dapat dilakukan oleh salah satu *field* saja, dengan tetap tidak mengubah arti relasi dan ketergantungannya. Oleh sebab itu, disebut ketergantungan fungsional sebagian (*partially functional*), sehingga syarat bentuk normal kedua (2NF) sudah tercapai.

Bentuk normal kedua (2NF) merupakan syarat yang harus dimiliki untuk menuju bentuk normal ketiga (3NF). Pada proses ini, dilakukan dengan menghilangkan ketergantungan secara transitif, yaitu suatu konsep untuk tabel dari hasil relasi yang didalamnya terdapat ketergantungan secara tidak langsung pada beberapa atributnya. Pada umumnya proses normalisasi sudah dapat tercapai pada bentuk normal ketiga (3NF), yaitu dengan menghasilkan tabel yang tidak mengalami anomali basis data pada saat proses *insert, delete*, dan *update.* (Yudi Priyadi ; 2014 : 68)



Gamba II.7. : Tahapan Aturan Proses Normalisasi

(Sumber : Yudi Priyadi ; 2014 : 69)

II.7. *Unified Modeling Language* (UML)

Pada perkembangan teknik pemrograman berorientasi objek, muncullah sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modeling Language* (UML). UML muncul karena adanya kebutuhan pemodelan *visual* untuk menspesifikasikan, menggambarkan, membangun dan dokumentasi dari sistem perangkat lunak. UML merupakan bahasa *visual* untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung.

UML hanya berfungsi untuk melakukan pemodelan, jadi penggunaan UML tidak terbatas pada metodologi tertentu, meskipun pada kenyataannya UML paling banyak digunakan pada metode berorientasi objek. (Rosa A.S & M. Shalahuddin ; 2011 : 118)

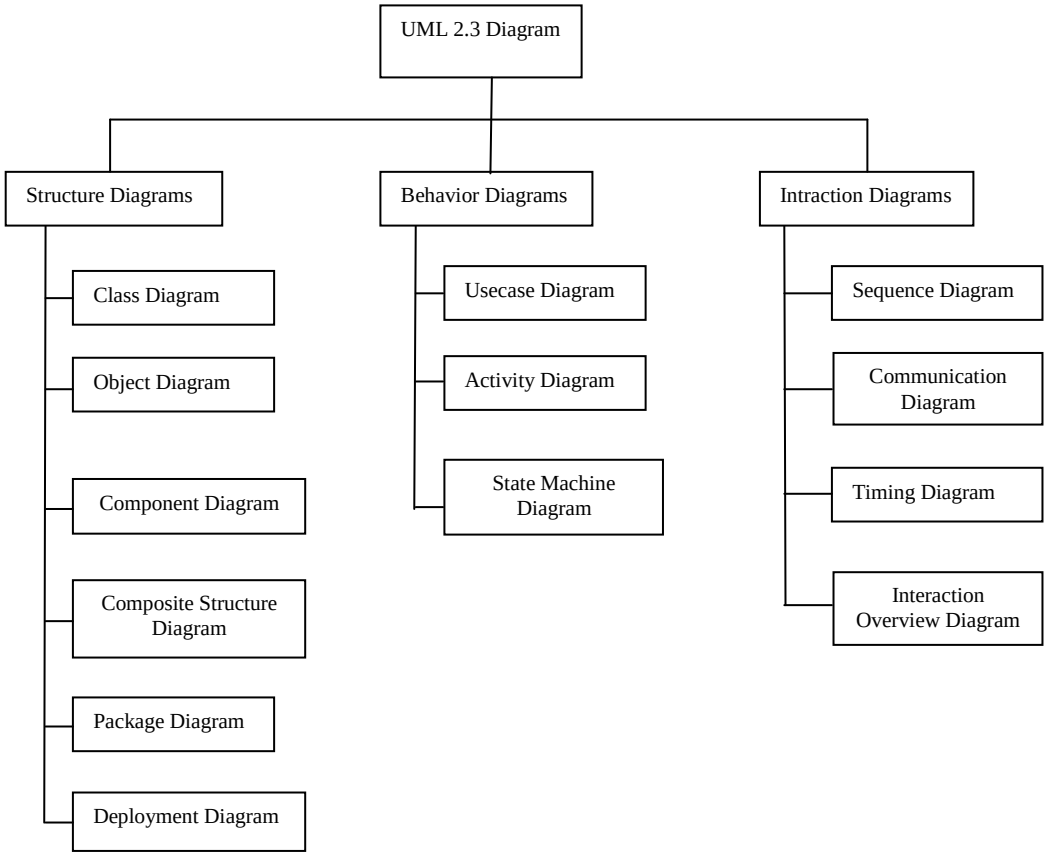
UML diaplikasikan untuk maksud tertentu, biasanya antara lain :

1. Merancang perangkat lunak.
2. Sarana komunikasi antara perangkat lunak dengan proses bisnis.
3. Menjabarkan sistem secara rinci untuk analisa dan mencari apa yang diperlukan sistem.
4. Mendokumentasikan sistem yang ada, proses-proses dan organisasinya.

Blok pembangunan utama UML adalah diagram. Beberapa diagram ada yang rinci (jenis *timming diagram*) dan lainnya ada yang bersifat umum (misalnya diagram kelas). Para pengembang sistem berorientasi objek menggunakan bahasa model untuk menggambarkan, membangun dan mendokumentasikan sistem yang mereka rancang. UML memungkinkan para anggota *team* untuk bekerja sama dengan bahasa model yang sama dengan mengaplikasikan beragam sistem. Intinya UML merupakan alat komunikasi yang konsisten dalam mendukung para pengembang sistem saat ini. (Prabowo Pudjo Widodo & Herlawati ; 2011 : 6)

II.7.1 Diagram-Diagram UML

UML terdiri dari 13 macam diagram yang dikelompokan dalam 3 kategori. Pembagian kategori dan macam-macam diagram tersebut dapat dilihat pada **Gambar II.8.** di bawah ini.



Gambar II.8. : Diagram UML

(Sumber : Rosa A.S & M. Shalahuddin ; 2011 : 121)

Berikut ini penjelasan singkat dari pembagian kategori tersebut :

1. *Structure Diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan.
2. *Behavior Diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada sebuah sistem.
3. *Interaction Diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antar subsistem pada suatu sistem.

A. *Class Diagram*

Diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi, yaitu :

1. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas.
2. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

Tabel II.3. Simbol-Simbol pada Diagram Kelas

Simbol	Deskripsi
Kelas Nama_kelas +atribut +operasi()	Kelas pada struktur sistem.
Antarmuka / *interface* Nama_*interface*	Sama dengan konsep *interface* dalam pemrograman berorientasi objek.
Asosiasi / *association* Asosiasi berarah / *directed association*	Relasi antar kelas dengan makna umum, asosiasi biasanya juga disertai dengan *multiplicity*. Relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan *multiplicity*.
Generalisasi	Relasi antar kelas dengan makna generalisasi-spesialisasi (umum khusus).
Kebergantungan	Relasi antar kelas dengan makna kebergantungan antar kelas.
Agregasi / *aggregation*	Semua bagian (*whole part*).

(Sumber : Rosa A.S & M. Shalahuddin ; 2011 : 124)

B. *Object Diagram*

Diagram objek menggambarkan struktur sistem dari segi penamaan objek dan jalannya objek dalam sistem. Pada diagram objek harus dipastikan semua kelas yang sudah didefenisikan pada diagram kelas harus dipakai objeknya, karena jika tidak, pendefenisian kelas itu tidak dapat dipertanggungjawabkan. Untuk apa mendefenisikan sebuah kelas sedangkan pada jalannya sistem, objeknya tidak pernah dipakai.

Tabel II.4. Simbol-Simbol pada Diagram Objek

Simbol	Deskripsi
Objek Nama_objek : nama_kelas Atribut = nilai	Objek dari kelas yang berjalan saat sistem dijalankan.
Link	Relasi antar objek.

(Sumber : Rosa A.S & M. Shalahuddin ; 2011 : 124)

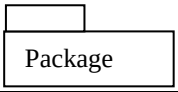
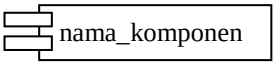
C. *Component Diagram*

Diagram komponen atau *component diagram* dibuat untuk menunjukkan organisasi dan ketergantungan di antara kumpulan komponen dalam sebuah sistem. Diagram komponen fokus pada komponen sistem yang dibutuhkan dan ada di dalam sistem. Komponen dasar yang biasanya ada dalam suatu sistem adalah sebagai berikut :

1. Komponen *user interface* yang menangani tampilan.
2. Komponen *bussiness processing* yang menangani fungsi-fungsi proses bisnis.
3. Komponen data yang menangani manipulasi data.
4. Komponen *security* yang menangani keamanan sistem.

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

Tabel II.5. Simbol-Simbol pada Diagram Komponen

Simbol	Deskripsi
Package 	*Package* merupakan sebuah bungkusan dari satu atau lebih komponen.
Komponen 	Komponen Sistem.
Kebergantungan / *dependency*	Kebergantungan antar komponen, arah panah mengarah pada komponen yang dipakai.
Antar muka / *interface*  nama_interface	Sama dengan konsep *interface* pada pemrograman berorientasi objek, yaitu sebagai antar muka komponen agar tidak mengakses langsung komponen.
Link	Relasi antar komponen.

(Sumber : Rosa A.S & M. Shalahuddin ; 2011 : 126)

D. *Use Case Diagram*

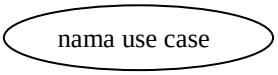
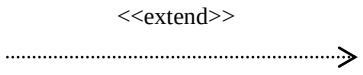
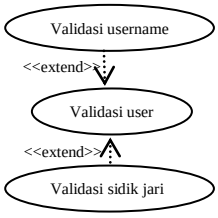
Use case atau diagram *use case* merupakan pemodelan untuk kelakuan (*behaviour)* sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu. Syarat penamaan pada *use case* adalah nama didefenisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu pendefenisian apa yang disebut aktor dan *use case*.

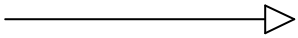
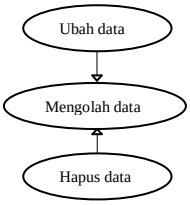
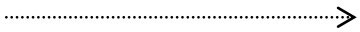
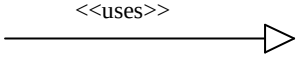
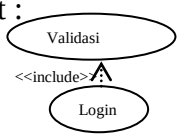
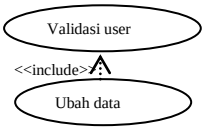
1. Aktor merupakan orang, proses atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat

itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tetapi aktor belum tentu merupakan orang.

2. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor.

Tabel II.6. Simbol-Simbol pada Diagram *Use Case*

Simbol	Deskripsi
Use case 	Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor, biasanya dinyatakan dengan menggunakan kata kerja di awal frase nama *use case*.
Aktor / *actor*  nama aktor	Orang, proses atau sistem yang lain berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan di buat itu sendiri.
Asosiasi / *association*	Komunikasi antara aktor dan *use case* yang berpartisipasi pada *use case* atau *usecase* memiliki interasi dengan aktor.
Ekstensi / *extend* 	Relasi *use case* tambahan ke sebuah *use case* dimana *use case* yang ditambahkan dapat berdiri sendiri walau tanpa *use case* tambahan itu, mirip dengan prinsip *inheritance* pada pemrograman berorientasi objek, biasanya *use case* tambahan memiliki nama depan yang sama dengan *use case* yang ditambahkan misalnya :  arah panah mengarah pada *use case* yang ditambahkan.

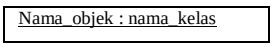
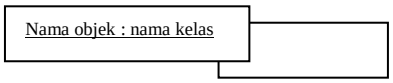
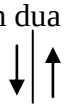
Generalisasi / *generalization* 	Hubungan generalisasi dan spesialisasi (umum-khusus) antara dua buah *use case* dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya misalnya :  Arah panah mengarah pada *use case* yang menjadi generalisasinya (umum).
Menggunakan / *include* / *uses* <<include>>  <<uses>> 	Relasi *use cas*e tambahan ke sebuah *use case* dimana *use case* yang ditambahkan memerlukan *use case* ini untuk menjalankan fungsinya atau sebagai syarat dijalankan *use case* ini. Ada 2 sudut pandang yang cukup besar mengenai *include* di *use case*, yaitu : 1. *Include* berarti *use case* yang ditambahkan akan selalu dipanggil saat *use case* dijalankan, misalnya pada kasus berikut :  2. *Include* berarti *use case* yang tambahan akan selalu melakukan pengecekan apakah *use case* yang di tambahkan telah di jalankan sebelum *use case* tambahan di jalankan, misalnya pada kasus berikut :  Kedua interpretasi di atas dapat dianut salah satu atau keduanya tergantung pada pertimbangan dan interpretasi yang dibutuhkan.

(Sumber : Rosa A.S & M. Shalahuddin ; 2011 : 131)

E. *Communication Diagram*

Diagram komunikasi mengelompokkan *message* pada kumpulan diagram sekuen menjadi sebuah diagram. Dalam diagram komunikasi yang dituliskan adalah operasi atau metode yang dijalankan antara objek yang satu dengan objek lainnya secara keseluruhan, oleh karena itu dapat di ambil dari jalannya interaksi pada semua diagram sekuen.

Tabel II.7. Simbol-Simbol pada Diagram Komunikasi

Simbol	Deskripsi
Objek 	Objek yang melakukan interaksi pesan.
Link	Relasi antar objek yang menghubungkan objek satu dengan lainya atau dengan dirinya sendiri. 
Arah pesan / stimulus 	Arah pesan yang terjadi, jika pada suatu link ada dua arah pesan yang berbeda, maka arah juga digambarkan dua arah pada dua sisi link. 

(Sumber : Rosa A.S & M. Shalahuddin ; 2011 : 140)

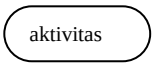
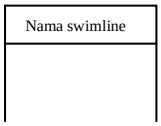
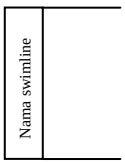
F. *Activity Diagram*

Diagram aktivitas atau *activity diagram* menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Diagram aktivitas juga banyak digunakan untuk mendefenisikan hal-hal berikut :

1. Rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefenisikan.

2. Urutan atau pengelompokan tampilan dari sistem atau *user interface* dimana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan.
3. Rancangan pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefenisikan kasus ujinya.

Tabel II.8. Simbol-Simbol pada Diagram Aktivitas

Simbol	Deskripsi
Status awal 	Status awal aktivitas sistem, sebuah diagram aktivitas memiliki status awal.
Aktivitas 	Aktivitas yang dilakukan sistem, aktivitas biasanya diawali dengan kata kerja.
Percabangan / decision 	Asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu.
Penggabungan / join	Asosiasi penggabungan dimana lebih dari satu aktivitas digabungkan menjadi satu.
Status akhir 	Status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki sebuah status akhir.
Swimlane  atau 	Memisahkan organisasi bisnis yang bertanggung jawab terhadap aktivitas yang terjadi.

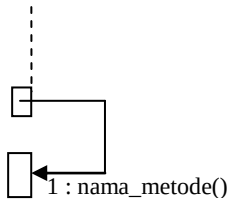
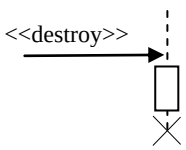
(Sumber : Rosa A.S & M. Shalahuddin ; 2011 : 134)

G. *Sequence Diagram*

Diagram sekuen menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Banyaknya diagram objek yang digambarkan adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefenisikan interaksi jalannya pesan sudah dicakup dapat diagram sekuen sehingga semakin banyak *use case* yang didefenisikan maka diagram sekuen yang harus dibuat juga semakin banyak. (Rosa A.S & M. Shalahuddin ; 2011 : 120)

Tabel II.9. Simbol-Simbol Pada Diagram Sekuen

Simbol	Deskripsi
Aktor  nama aktor atau nama_aktor tampa waktu aktif	Orang, proses atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat diluar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang, biasanya di nyatakan menggunakan kata benda diawali frase nama aktor.
Garis hidup / lifeline	Menyatakan kehidupan suatu objek.
Objek nama objek : nama kelas	Menyatakan objek yang berinteraksi pesan.
Waktu aktif 	Menyatakan objek dalam keadaan aktif dan berinteraksi pesan.

Pesan tipe create <<create>>	Objek yang lain, arah panah mengarah pada objek yang dibuat.
Pesan tope call 1 : nama metode()	Menyatakan suatu objek memanggil operasi / metode yang ada pada objek lain atau dirinya sendiri.  Arah panah mengarah pada objek yang memiliki operasi / metode, karena ini memanggil operasi / metode maka operasi / metode yang di panggil harus ada pada diagram kelas sesuai dengan kelas objek yang berinteraksi.
Pesan tipe send 1 : masukan	Menyatakan bahwa suatu objek mengirimkan data / masukan / informasi ke objek lainya, arah panah mengarah pada objek yang dikirimi.
Pesan tipe return 1 : keluaran	Menyatakan bahwa suatu objek yang telah menjalankan suatu operasi atau metode menghasilkan suatu kembalian ke objek tertentu, arah panah mengarah pada objek yang menerima kembalian.
Pesan tipe destroy  	Menyatakan suatu objek mengakhiri hidup objek yang lain, arah panah mengarah pada objek yang diakhiri, sebaiknya jika ada *create* maka ada *destroy*.

(Sumber : Rosa A.S & M. Shalahuddin ; 2011 : 138)