BAB II

TINJAUAN PUSTAKA

II.1. Sistem

Sistem adalah kumpulan dari elemen-elemen yang berinteraksi untuk mencapai suatu tujuan tertentu (Jogiyanto;2005:2).

II.1.1. Karakteristik Sistem

Suatu sistem mempunyai karakteristik atau sifat-sifat yang tertentu, yaitu mempunyai komponen-komponen *(components)*, batas sistem *(boundary)*, lingkungan luar sistem *(environments)*, penghubung *(interface)*, masukan *(input)*, keluaran *(output)*, pengolah *(process)* dan sasaran *(objectives)* atau tujuan *(goal)* (Jogiyanto;2005:3).

II.1.2. Klasifikasi Sistem

Sistem dapat diklasifikasikan dari beberapa sudut pandang, diantaranya adalah sebagai berikut :

1. Sistem diklasifikasikan sebagai sistem abstrak *(abstract system)* dan sistem fisik *(physical system).*
2. Sistem diklasifikasikan sebagai sistem alamiah *(natural system)* dan sistem buatan manusia *(human made system)*.
3. Sistem diklasifikasikan sebagai sistem tertentu *(deterministic system)* dan sistem tak tentu *(probabilistic system)*.

4. Sistem diklasifikasikan sebagai sistem tertutup *(closed system)* dan sistem terbuka *(open system)* (Jogiyanto;2005:6-7).

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

Sistem Pakar (*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. 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. Mampu bekerja dengan informasi yang tidak lengkap atau tidak pasti. Berbeda dengan sistem komputer konvensional, Sistem Pakar dapat bekerja dengan informasi yang tidak lengkap.
8. 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.
9. Meningkatkan kemampuan untuk menyelesaikan masalah karena Sistem Pakar mengambil sumber pengetahuan dari banyak pakar.

II.3.3. 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.3.4. 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.3.5. Konsep Dasar Sistem Pakar

Konsep Dasar Sistem Pakar mencakup beberapa persoalan mendasar, antara lain siapa yang disebut pakar, apa yang dimaksud dengan keahlian, bagaimana keahlian dapat ditransfer, dan bagaimana sistem bekerja. Pakar adalah

orang yang memiliki pengetahuan, penilaian, pengalaman, metode khusus, serta kemampuan untuk menerapkan bakat ini dalam memberi nasihat dan memecahkan masalah.

Konsep dasar sistem pakar meliputi enam hal berikut ini:

II.3.5.1. 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.3.5.2. 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.3.5.3. 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.3.5.4. 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.3.5.5. 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.3.5.6. 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 pemprosesannya biasanya digabungkan dalam satu program.	Basis pengetahuan merupakan bagian terpisah dari mekanisme inferensi.
Program tidak membuat kesalah (yang	Program dapat berbuat kesalahan.

membuat kesalahan: Pemrogram atau Pengguna).	
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.

Sumber: (T. Sutejo;2011:165)

II.4. Anak Indigo

Istilah anak indigo dikemukakan oleh Nancy Ann Tappe, pada tahun 1970-an yang meneliti warna aura manusia dan menghubungkannya dengan kepribadian. Mereka yang memiliki aura berwana nila atau indigo, ternyata anak-anak yang dianugerahi kelebihan yang justru dikategorikan aneh oleh lingkungan ketika kelebihannya berhubungan dengan misi keberadaannya di dunia dan bentuk abstraksi hidupnya yang juga berkenaan dengan hal-hal yang bersifat metafisika dan spritualitas yang terkesan tidak cocok bagi usianya. Tappe sebagaimana dikutip Carrol dan Tober menjelaskan mengenai awal penetapan istilah indigo dan bagaimana alasan-alasan mereka ada (Mohammad A Syuropati;2014:11).

Istilah indigo yang secara teknis terbentuk dari warna biru gelap keunguan, menujukan warna aura dari setiap karakter warna aura yang ada. Hubungan warna itu dengan anak-anak yang mendapat julukan anak indigo terdapat dari adanya bentuk kesamaan yang dimiliki diantara mereka tentang konsep indera keenam. Indera yang dimaksud adalah intuisi sebagaimana setiap orang memiliki intuisi, tetapi anak indigo memiliki intuisi yang luar biasa tajam di atas kemampuan orang kebanyakan. Kekuatan karakter anak indigo yang berbeda dengan anak-anak seusianya atau kebanyakan orang tersebut sering diakui sebagai suatu kemampuan lebih tapi kelebihannnya tersebut sering dipandang aneh oleh lingkungannya yang tidak terbiasa atau tidak mengenal bagaimana karakter dari anak indigo (Mohammad A Syuropati;2014:5).

II.4.1. Gejala Anak Indigo

Untuk mengetahui apakah anak anda termasuk anak indigo, maka tanyailah anak anda dengan pertanyaan-pertanyaan berikut:

1. Apakah anak anda lahir ke dunia dengan perilaku royal?
2. Apakah anak anda mempunyai perasaan pantas untuk berada di muka bumi ini?
3. Apakah anak anda mempunyai rasa kepemilikan diri yang nyata?
4. Apakah anak anda mempunyai kesulitan dengan disiplin dan otoritas?
5. Apakah anak anda menolak untuk melakukan hal tertentu yang dikatakan pada mereka?
6. Adakah rentetan penderitaan pada anak Anda?
7. Apakah anak anda terganggu dengan system-sistem ritual yang memerlukan kreativitas kecil?
8. Apakah anak anda mengetahui cara-cara terbaik dalam melakukan sesuatu dirumah dan sekolah?
9. Apakah anak anda seorang non konformasi?
10. Apakah anak anda menolak untuk merespon kekeliruan?
11. Apakah anak anda mudah bosan dengan tugas-tugas yang diperintahkan ?
12. Apakah anak anda memperlihatkan gejala ADD (Attention Defisit Disorder)?
13. Apakah anak anda sangat kreatif?
14. Apakah anak anda memperlihatkan intuisinya?
15. Apakah anak anda memiliki rasa empati yang begitu kuat pada orang lain?
16. Apakah anak anda mengembangkan pemikiran abstrak terlalu dini?

17. Apakah anak anda sangat cerdas?
18. Apakah anak anda sangat berbakat?
19. Apakah anak anda nampak pelamun?
20. Apakah anak anda memiliki pandangan mata yang begitu dewasa, mendalam dan bijak?
21. Apakah anak anda memiliki kecerdasan spiritual?

Apabila anda memiliki jawaban ya lebih dari 10, barangkali dia seorang Indigo. Apabila jawaban ya lebih dar 15 maka dapat dipastikan ia seorang Indigo.

II.5. *Fuzzy Logic*

II.5.1. Pengertian

Logika *fuzzy* adalah suatu cara yang tepat untuk memetakan suatu ruang *input* kedalam suatu ruang *output*. Titik awal dari konsep modern mengenai ketidakpastian adalah paper yang dibuat oleh Lofti A Zadeh (1965), dimana Zadeh memperkenalkan teori yang memiliki obyek-obyek dari *himpunan fuzzy* yang memiliki batasan yang tidak presisi dan keanggotaan dalam himpunan *fuzzy*, dan bukan dalam bentuk logika benar (*true*) atau salah (*false*), tapi dinyatakan dalam derajat (*degree*). Konsep seperti ini disebut dengan *Fuzziness* dan teorinya dinamakan *Fuzzy Set Theory*. *Fuzziness* dapat didefinisikan sebagai logika kabur berkenaan dengan semantik dari suatu kejadian, fenomena atau pernyataan itu sendiri. Seringkali ditemui dalam pernyataan yang dibuat oleh seseorang, evaluasi dan suatu pengambilan keputusan.

Fuzzy system (sistem kabur) didasari atas konsep himpunan kabur yang memetakan domain *input* kedalam domain *output*. Perbedaan mendasar himpunan tegas dengan himpunan kabur adalah nilai keluarannya. Himpunan tegas hanya memiliki dua nilai *output* yaitu nol atau satu, sedangkan himpunan kabur memiliki banyak nilai keluaran yang dikenal dengan nilai derajat keanggotaannya.

Logika *fuzzy* adalah peningkatan dari logika *Boolean* yang berhadapan dengan konsep kebenaran sebagian. Dimana logika klasik (*crisp*) menyatakan bahwa segala hal dapat diekspresikan dalam istilah *binary* (0 atau 1, hitam atau putih, ya atau tidak). Logika *fuzzy* menggantikan kebenaran *Boolean* dengan tingkat kebenaran. Logika *fuzzy* memungkinkan nilai keanggotaan antara 0 dan 1, tingkat keabuan dan juga hitam dan putih, dan dalam bentuk *linguistic*, konsep tidak pasti seperti "sedikit", "lumayan", dan "sangat". Logika ini diperkenalkan oleh Dr. Lotfi Zadeh dari Universitas California, Barkeley pada tahun 1965. Logika *fuzzy* telah digunakan pada bidang-bidang seperti taksonomi, topologi, linguistik, teori automata, teori pengendalian, psikologi, *pattern recognition*, pengobatan, hukum, *decision analysis*, system *theory and information retrieval*. Pendekatan *fuzzy* memiliki kelebihan pada hasil yang terkait dengan sifat kognitif manusia, khususnya pada situasi yang melibatkan pembentukan konsep, pengenalan pola, dan pengambilan keputusan dalam lingkungan yang tidak pasti atau tidak jelas.

Ada beberapa alasan mengapa orang menggunakan logika *fuzzy* (Kusumadewi S, Purnomo H;2010) antara lain:

1. Konsep logika *fuzzy* mudah dimengerti. Konsep matematis yang mendasari penalaran *fuzzy* sangat sederhana dan mudah dimengerti.
2. Logika *fuzzy* sangat fleksibel.
3. Logika *fuzzy* memiliki toleransi terhadap data-data yang tidak tepat.
4. Logika *fuzzy* mampu memodelkan fungsi-fungsi *nonlinear* yang sangat kompleks.
5. Logika *fuzzy* dapat membangun dan mengaplikasikan pengalaman-pengalaman para pakar secara langsung tanpa harus melalui proses pelatihan.
6. Logika *fuzzy* dapat bekerjasama dengan teknik-teknik kendali secara konvensional.
7. Logika *fuzzy* didasarkan pada bahasa alami.

II.5.2. Konsep *Fuzzy Logic*

Teori logika *fuzzy* yang diajukan oleh Zadeh pada pertengahan tahun 1960 (Nikola K.;1998, Setiyowati, M.I.Seta, B.A;2007), memberikan suatu pemecahan masalah terhadap persoalan yang tidak pasti ini. Sehingga sistem informasi yang akan dibuat menggunakan model DBMS dan *query* yang berbasis *fuzzy* karena model DBMS konvensional, *non fuzzy* kurang dapat memenuhi kebutuhan sistem informasi ini. Banyak model DBMS dan *query fuzzy* yang ada, salah satunya adalah model Tahani yang ditemukan pada tahun 1977. Prof. Lutfi Zadeh berpendapat bahwa logika benar dan salah dari logika b*oolean*/konvensional tidak dapat mengatasi masalah gradasi yang ada di dunia nyata. Untuk mengatasi

masalah gradasi tersebut maka ia mengembangkan sebuah himpunan samar (*fuzzy*).

II.5.3. Himpunan *Fuzzy*

Pada himpunan tegas (*crisp*), nilai keanggotaan suatu item x dalam suatu himpunan A, yang sering ditulis dengan μA[x], memiliki 2 kemungkinan (Kusumadewi S, Purnomo H;2010) yaitu:

1. Satu (1), yang berarti bahwa suatu item menjadi anggota dalam suatu himpunan, atau

2. Nol (0), yang berarti bahwa suatu item tidak menjadi anggota dalam suatu himpunan.

Terkadang kemiripan antara keanggotaan fuzzy dengan probabilitas menimbulkan kerancuan. Keduanya memiliki nilai pada interval [0,1], namun interprestasi nilainya sangat berbeda antara kedua kasus tersebut. Keanggotaan fuzzy memberikan suatu ukuran terhadap pendapat atau keputusan, sedangkan probabilitas mengindikasikan proporsi terhadap keseringan suatu hasil bernilai

benar dalam jangka panjang. Misalnya, jika nilai keanggotaan bernilai suatu himpunan fuzzy USIA adalah 0,9; maka tidak perlu dipermasalahkan berapa seringnya nilai itu diulang secara individual untuk mengharapkan suatu hasil yang hampir pasti muda. Di lain pihak, nilai probabilitas 0,9 usia berarti 10% dari himpunan tersebut diharapkan tidak muda. Himpunan *fuzzy* memiliki 2 atribut, yaitu:

1. Linguistik, yaitu penamaan suatu grup yang mewakili suatu keadaan atau kondisi tertentu dengan menggunakan bahasa alami, seperti: MUDA, PAROBAYA, TUA
2. Numeris, yaitu suatu nilai (angka) yang menunjukkan ukuran dari suatu variable seperti: 40, 25, 50, dsb.

Ada beberapa hal yang perlu diketahui dalam memahami sistem *fuzzy* (Sri Kusumadewi, Hari Purnomo;2010), yaitu:

a. Variable *fuzzy*

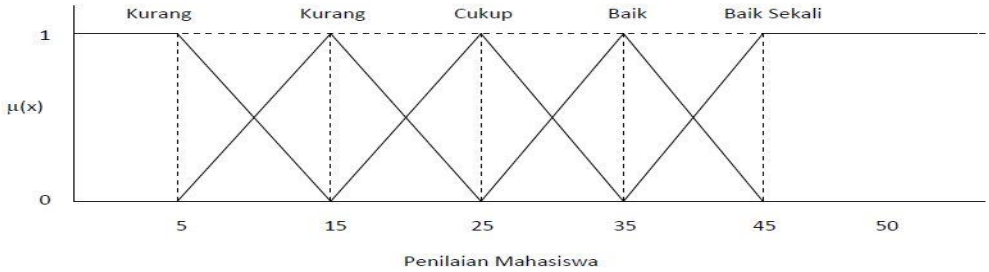
   Variable *fuzzy* merupakan variabel yang hendak dibahas dalam suatu system *fuzzy*. Contoh: umur, temperature, permintaan, dsb

b. Himpunan *Fuzzy*

   Himpunan *fuzzy* merupakan suatu grup yang mewakili suatu kondisi atau keadaan tertentu dalam suatu variabel *fuzzy*.

   Contoh:

   a. Variable mahasiswa, terbagi menjadi 5 himpunan fuzzy, yaitu: kurang sekali, kurang, cukup, baik dan baik sekali.
   b. Variabel dosen, terbagi menjadi 3 himpunan fuzzy, yaitu: cukup, baik, dan baik sekali. Seperti terlihat pada gambar II.1.



Gambar II.1 Himpunan Fuzzy Pada Mahasiswa

Sumber (Ebook;ChapterII Universitas Sumatera Utara)

II.6. *Visual Basic . Net 2010*

II.6.1. Sejarah *Visual Basic* . Net

Pada zaman dahulu ada sebuah bahasa pemrograman yang diberi nama Basic (*Beginner's All- purpose Symbolic Intruction Code*). Sesuai dengan namanya, Basic ditujukan sebagai bahasa yang paling sederhana bagi mereka yang tidak terlalu familiar dengan dunia pemrograman.

Pada tahun 1991 Microsoft mengeluarkan *Visual Basic*, pengembangan dari Basic yang berubah dari sisi pembuatan antarmukanya. *Visual Basic* sampai sekarang masih menjadi salah satu bahasa pemrograman terpopuler di dunia.

Pada akhir tahun 1999, Teknologi .NET di umumkan. Microsoft memosisikan teknologi tersebut sebagai platform untuk membangun XML *Web Service*. XML *Web Service* memungkinkan aplikasi tipe apa pun dapat berjalan pada system computer dengan type manapun dan dapat mengambil data yang tersimpan pada server dengan tipe apa pun melalui Internet.

Visual Basic. Net adalah *Visual Basic* yang direkayasa kembali untuk digunakan pada *platform.Net* sehingga aplikasi yang dibuat menggunakan *Visual Basic*. Net dapat berjalan pada sistem computer apapun, dan dapat mengambil data dari server dengan tipe apapun asalkan terinstal . *Net Framework.*

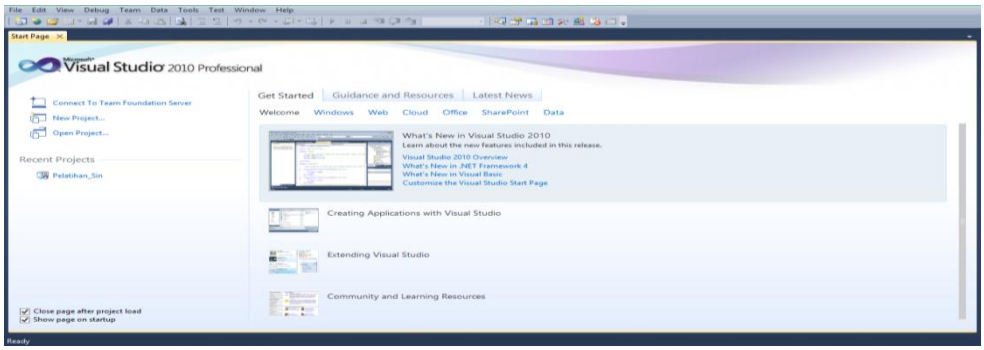
Berikut ini perkembangan *Visual Basic* . Net :

a. *Visual Basic* . *Net 2002* (VB 7.0)
b. *Visual Basic* . *Net 2003* (VB 7.1)
c. *Visual Basic 2005* (VB 8.0)
d. *Visual Basic* 2008 (VB 9.0)
e. *Visual Basic* 2010 (VB 10.0)
f. *Visual Basic* 2012 (VB 11.0)

II.6.2. Berkenalan dengan *Visual Basic* . Net

II.6.2.1. IDE (*Integrated Development Environment*)

Pada waktu *Visual Basic* 2010 di jalankan, akan tampil sebuah *Start Page* seperti terlihat pada Gambar II.2.

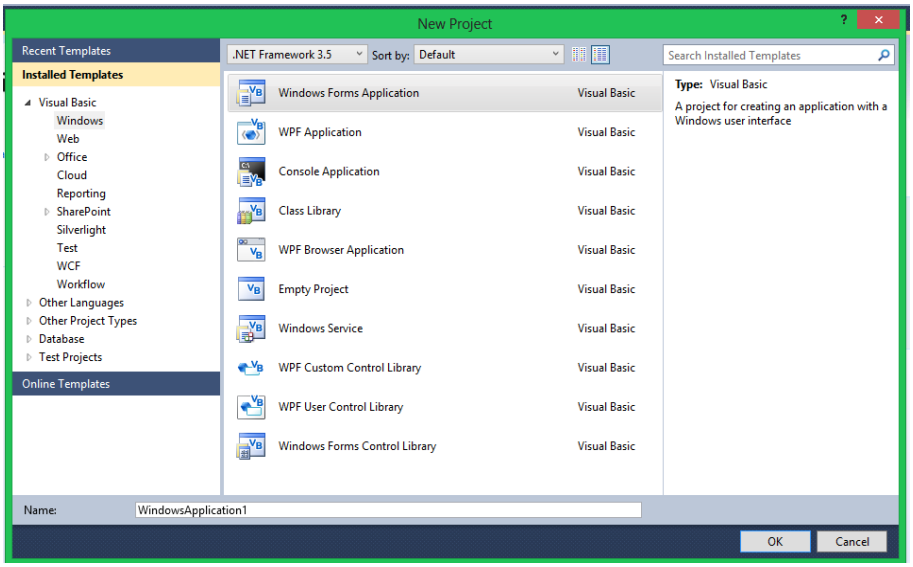


Gambar II.2. Start Page dari Visual studio 2010

Sumber : Priyanto Hidayatullah (2014 : 23)

Untuk membuka proyek yang ada gunakan tombol ***Open Project*** atau langsung mengklik pada daftar proyek yang ditampilkan sedangkan untuk membuaka sebuah proyek baru, klik tombol ***New Project***.

Setelah itu akan muncul kotak dialog ***New Project***. Pada kotak pilih ***Other Languages*** > ***Visual Basic*** > ***Windows*** > ***Windows Forms Application***. Untuk memberi nama proyek dapat dilakukan pada bagian ***Name***, dan tekan *OK* (Gambar II.3.)

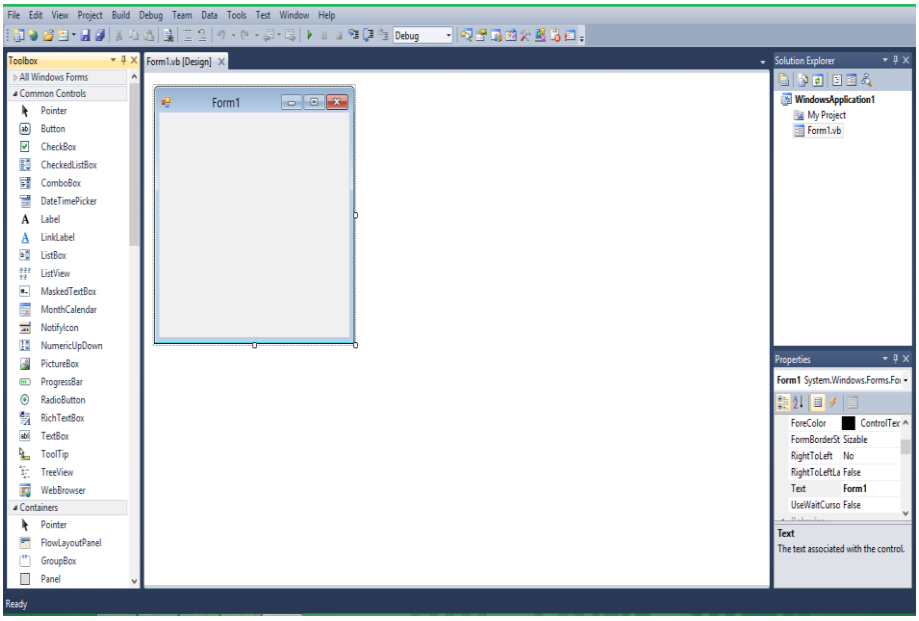


Gambar II.3. Kotak Dialog New Project

Sumber : Priyanto Hidayatullah (2014 : 24)

Pada IDE *Visual Basic* 2010 **(**Gambar II.4.**)** untuk *Windows Application* secara default telah terdapat sebuah form. Form tersebut bernama Form1. Pada form inilah tempat meletakkan kontrol- kontrol atau komponen- komponen untuk membuat sebuah aplikasi Windows Form dan kontrol- kontrol dari aplikasi inilah yang biasanya disebut dengan GUI (*Graphical User Interface*). Jadi user akan berinteraksi dengan sebuah program aplikasi melalui GUI. Pada IDE Visual

Studio 2010 terdapat *menu bar*, *toolbar*, *toolbox*, *solution explorer*, dan *properties windows*.

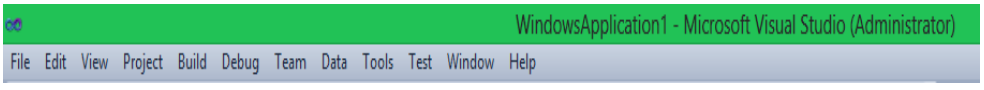


Gambar II.4. IDE Visual Studio 2010

Sumber : Priyanto Hidayatullah (2014 : 25)

II.6.2.2. *Menu Bar*

Menu bar adalah bagian dari IDE yang terdiri atas perintah- perintah untuk mengatur IDE, mengedit kode, dan mengeksekusi program. Di dalam menu bar, perintah- perintah dikelompokkan ke dalam beberapa bagian sesuai jenis perintah tersebut. Menu bar pada *Visual Studio* 2010 dilihat pada Gambar II.5.



Gambar II.5. Menu Bar

Sumber : Priyanto Hidayatullah (2014 : 25)

Untuk menggunakan menu atau pilihan pada menu bar, kita tinggal mengeklik pada menu atau pilihan yang akan dijalannkan. Sebagai contoh, untuk membuat sebuah proyek yang baru, pilih menu ***File*** > ***New*** > ***Project***.

Pada IDE Visual Studio 2010, terdapat 12 menu utama. Menu – menu tersebut antara lain sebagai berikut :

1. Menu ***File*** berisi perintah- perintah untuk membuat proyek, membuka proyek, menutup proyek, mencetak data dari proyek, dan lain- lain.
2. Menu ***Edit*** berisi perintah- perintah untuk *undo, cut, paste,* dan lain- lain.
3. Menu ***View*** berisi perintah- perintah untuk menampilkan *windows-windows* dari IDE dan *toolbar*.
4. Menu ***Project*** berisi perintah- perintah untuk mengatur proyek dan *file-file*.
5. Menu ***Build*** berisi perintah- perintah untuk meng-compile program.
6. Menu ***Debug*** berisi perintah- perintah untuk men-debug dan menjalankan program.
7. Menu ***Team*** berisi tentang *connect to team fundation server*.
8. Menu ***Data*** berisi perintah- perintah untuk berhubungan dengan basis data.
9. Menu ***Tools*** berisi perintah- perintah untuk mengakses komponen IDE tambahan dan mengubah IDE.
10. Menu ***Test*** mengelolah pengujian yang akan dilakukan pada proyek.
11. Menu ***Window*** berisi perintah- perintah untuk mengatur dan menampilkan *Windows*.
12. Menu ***Help*** berisi perintah- perintah untuk mengakses fasilitas bantuan.

II.6.2.3. *Toolbar*

Toolbar fungsinya sama seperti *menu*. Bedanya pada *toolbar* pilihan-pilihan berbentuk *icon*. Untuk memilih suatu proses yang akan dilakukan, kita tinggal menekan *icon* yang sesuai dengan proses yang kita inginkan. Bagian *toolbar* terlihat seperti pada Gambar II.6.

Gambar II.6. Toolbar

Sumber : Priyanto Hidayatullah (2014 : 27)

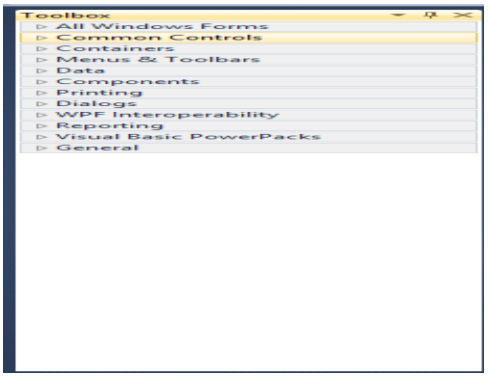
II.6.2.4. *Toolbox*

Toolbox adalah tempat di mana kontrol- kontrol dan komponen-komponen diletakkan. Kontrol dan komponen disimpan pada Toolbox dengan berbagai kategori :

1. *Common Controls*, berisi kontrol- kontrol umum yang sering digunakan seperti *button*, *label*, *textbox*, dsb.
2. *Containers*, berisi kontrol penyimpan kontrol lainnya seperti *panel*, *group box*, *tabcontrol*, dsb.
3. *Menu* dan *Toolbar*, berisi kontrol dan komponen *menu*, *context menu* dan *toolbar*.
4. Data, berisi kontrol dan komponen pengolahan data.
5. Components, berisi komponen- komponen seperti *timer*, *imagelist*, dsb.
6. *Printing*, berisi komponen untuk pencetakan dokumen.
7. Dialog, berisi komponen untuk berinteraksi dengan pengguna dalam hal membuka *file*, menyimpan *file*, membuka *folder* dll.

8. WPF *interoperability*, berisi komponen untuk *Windows Presentation Foundation*.
9. *Reporting*, berisi kontrol untuk membuat laporan pada *Visual Studio* 2010. Untuk studi kasus pada buku ini, kita akan menggunakan fasilitas *reporting* dari *Crystal Report* yang jauh lebih lengkap dan *powerful* namun gratis.
10. *Visual Basic PowerPacks*, berisi beberapa kontrol tambahan untuk menggambar (contoh : oval, garis, persegi) dan fungsi tambahan lainnya.
11. General, jika ada kontrol atau komponen yang mau ditambahkan dan belum memiliki kategori yang jelas, maka bisa dimasukkan ke kategori ini.

Selain kategori di atas, ada *All Windows Form* yang berisi semua kontrol dan komponen yang ada (lihat Gambar II.7.).

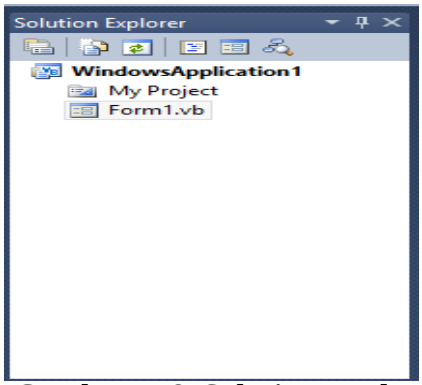


Gambar II.7. Toolbox

Sumber : Priyanto Hidayatullah (2014 : 28)

II.6.2.5. *Solution Explorer*

Solution explorer memberikan tampilan daftar *file- file* dari proyek yang sedang dibuat. Pada jendela *solution explorer* terdapat beberapa tombol dan *tree* yang berisi daftar dari *file- file* yang digunakan dalam proyek (lihat Gambar II.8.)

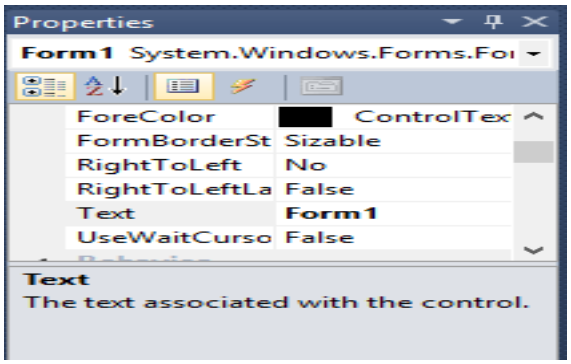


Gambar II.8. Solution Explorer

Sumber : Priyanto Hidayatullah (2014 : 29)

II.6.2.6. *Properties Window*

Properties Window adalah tempat menyyimpan property dari setiap objek kontrol dan komponen. *Properties Window* juga dipakai untuk mengatur property dari objek kontrol dan komponen yang dipakai. Dengan *propertie window*, kita dapat mengubah property yang nantinya akan dipakai sebagai *default* dari objek kontrol dan komponen pada waktu pertama kali program dieksekusi.



Gambar II.9. Properties Window

Sumber : Priyanto Hidayatullah (2014 : 30)

II.7. *SQL Server 2008*

Database Management System (DBMS) adalah aplikasi untuk mengelolah basis data. DBMS biasanya menawarkan beberapa kemampuan yang terintegrasi seperti:

1. Membuat, menghapus, menambah, dan memodifikasi basis data.
2. Pada beberapa DBMS pengelolaannya berbasis windows (berbentuk jendela- jendela) sehingga lebih mudah digunakan.
3. Tidak semua orang bisa mengakses basis data yang ada sehingga memberikan keamanan bagi data.
4. Kemampuan berkomunikasi dengan program aplikasi yang lain. Misalnya dimungkinkan untuk mengakses basis data *SQL Server* menggunakan aplikasi yang dibuat menggunakan VB.NET.
5. Kemampuan pengaksesan melalui komunikasi antarkomputer (*client server*).

Microsoft SQL Server adalah salah satu aplikasi DBMS yang sudah sangat banyak digunakan oleh para pemrograman aplikasi basis data. Contoh DBMS lainnya adalah : *MySQL (freeware), PostgreSQL (freeware), MS Access* dari *Microsoft, DB2 dari IBM, Oracle dan Oracle Corp, Dbase, FoxPro,* dsb.

II.7.1. *Tool* pada *MS SQL Server* 2008 R2

II.7.1.1. *Microsoft SQL Server Management Studio*

Tool ini digunakan untuk :

a. Membuat Database

Misalnya kita ingin membuat basis data dengan nama Inventori.

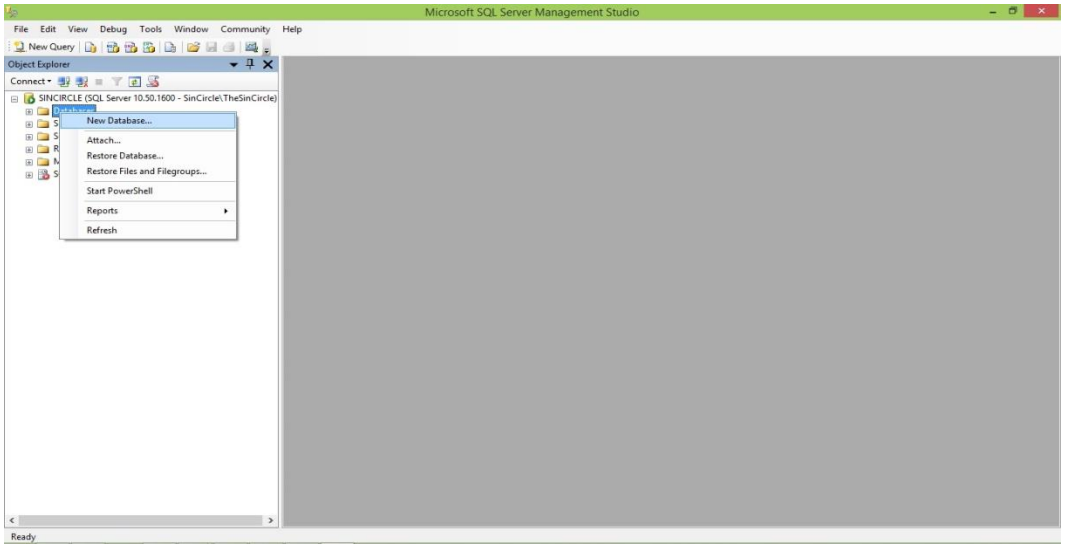
Langkah kerjanya adalah :

1. Bukalah *Microsoft SQL Server Management Studio*. Isikan ***Database Engine*** pada tipe *server*, nama *database server* Anda dan *windows Authentification* untuk tipe otentifikasi agar lebih mudah, tekan ***Connect.***

Gambar II.10. Login ke Microsoft SQL Server Management Studio

Sumber : Priyanto Hidayatullah (2014 : 187)

2. Pastikan Object Explorer terbuka, jika belum, buka dengan memilih menu **View** > **Object Explorer**.
3. Pada Object Explorer, klik kanan pada **Database**.
4. Pilih **New Database…**



Gambar II.11. Membuat database baru

Sumber : Priyanto Hidayatullah (2014 : 188)

5. Tulis nama basis data pada baris *Name*, misalnya *inventori*,

   Catatan : Dalam satu SQL Server, nama basis data harus berbeda. Gunakan nama yang menggambarkan isi datanya. Jangan gunakan nama basis data yang 'tidak nyambung' dengan isi datanya. Misalnya nama basis datanya swalayan tapi basis data tersebut berisi data rumah sakit.

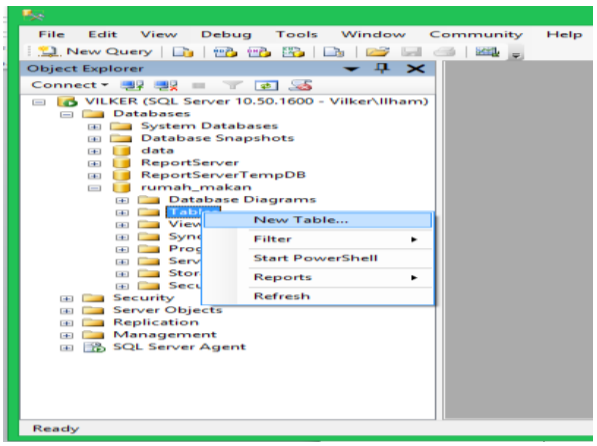
6. *Initial Size* adalah ukuran awal basis data yang kita buat. Ubahlah ukurannya dari 3 MB menjadi 5 MB.

7. Tekan *OK*.

b. Merancang Tabel

Pada dasarnya, data- data di dalam basis data disimpan di dalam tabel. Untuk merancang tabel, langkah kerjanya adalah :

1. Pilih basis data tempat kita membuat tabelnya.
2. Klik kanan pada bagian ***Tables*** kemudian pilih ***New Table***…



Gambar II.12. Membuat tabel baru

Sumber : Priyanto Hidayatullah (2014 : 191)

3. Masukkan informasi- informasi tentang tabel yang ingin kita buat.

   Column Name : nama kolom dalam tabel kita.

   Data type *:* tipe data kolom yang bersangkutan.

   Length : panjang data kolom yang bersangkutan.

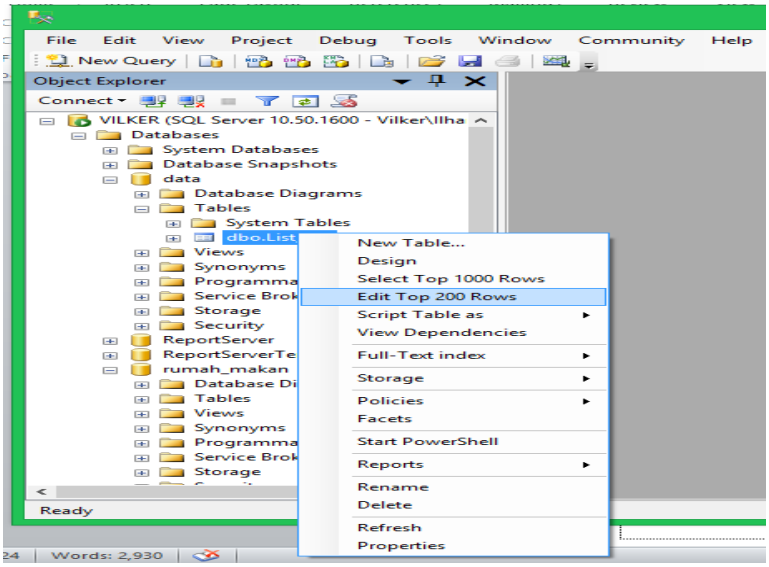
   Allow Nulls : Ceklis jika kolom tersebut diperbolehkan tidak diisi.
4. Untuk membuat setelan *primary key* pada table, dapat dilakukan dengan jalan klik kanan pada nama kolom yang akan dijadikan *primary key* kemudian pilih ***Set Primary Key***.
5. Setelah selesai, tutup jendelanya kemudian isi nama tabel yang baru saja kita buat. Tekan ***OK***.

c. Mengisi Tabel

Langkah- langkah untuk mengisi tabel adalah :

1. Buka basis data yang mengandung tabel yang kita cari.

2. Klik kanan tabel tersebut.
3. Kemudian pilihlah ***Edit Top* 200 *Rows***.
4. Kemudian Isilah data- data yang kita inginkan.



Gambar II.13. Mengisikan data ke dalam tabel

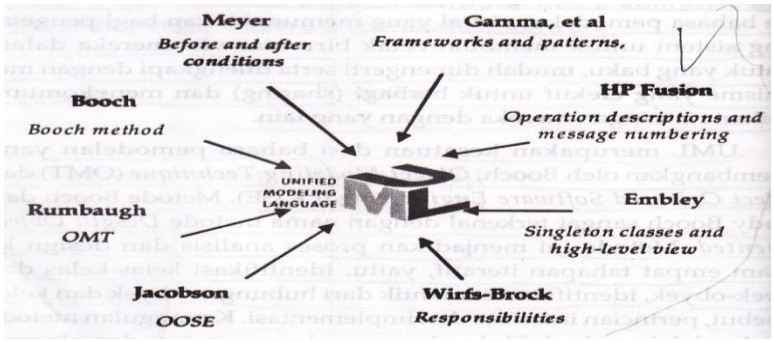
Sumber : Priyanto Hidayatullah (2014 : 194)

II.8. 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 baik (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.14



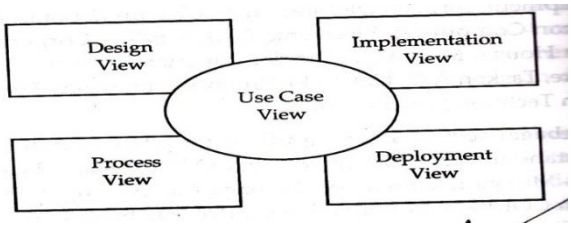
Gambar II.14 Unsur-unsur yang membentuk UML

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.15



Gambar II.15 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).

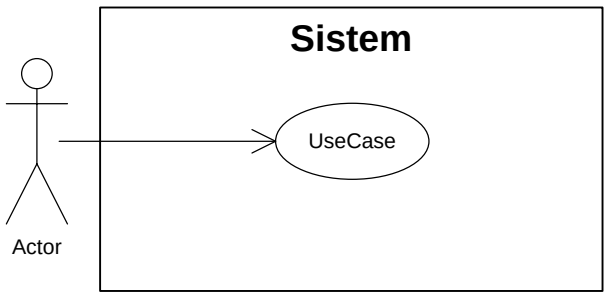
II.8.1. *Use Case Diagram*

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.16



Gambar II.16 *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.8.2. *Class Diagram*

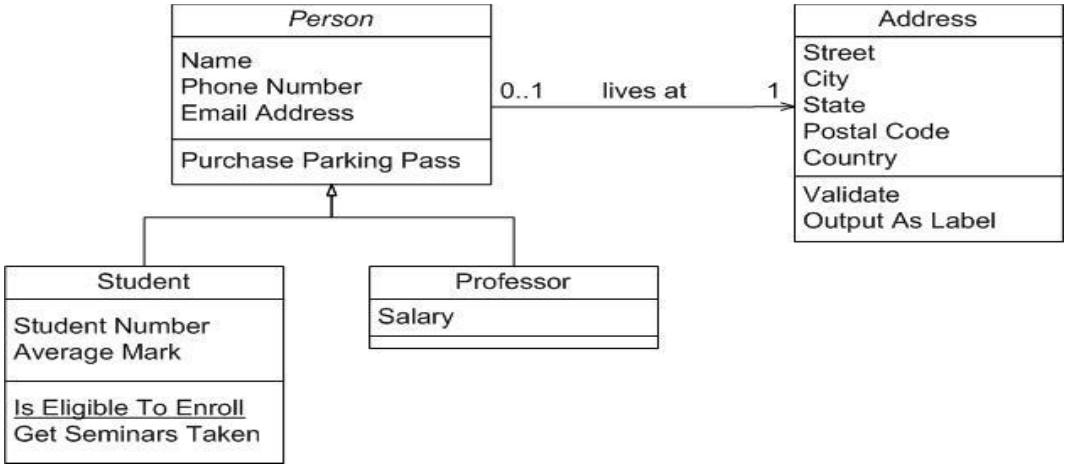
Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (*atribut*/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. *Class* memiliki tiga area pokok :

1. Nama kelas
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.17 dibawah ini:



Gambar II.17 *Class* Diagram

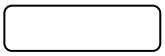
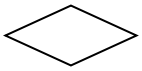
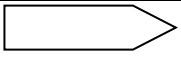
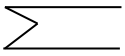
Sumber: (Munawar;2005:220)

II.8.3. *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 Tabel II.2 :

Tabel II.2. 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* menunjukkan adanya dekomposisi
	Tanda waktu
	Tanda pengiriman
	Tanda penerimaan

	Aliran Akhir (*Flow Final*)

Sumber : (Munawar;2005:110)

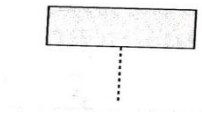
II.8.4. *Squence Diagram*

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.18



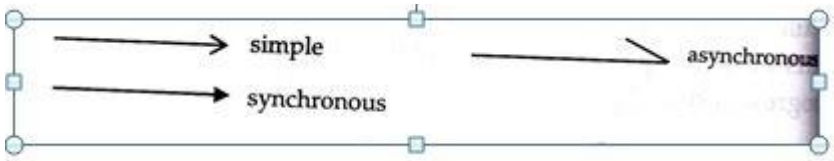
Gambar II.18 Bentuk *Participant*

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.19



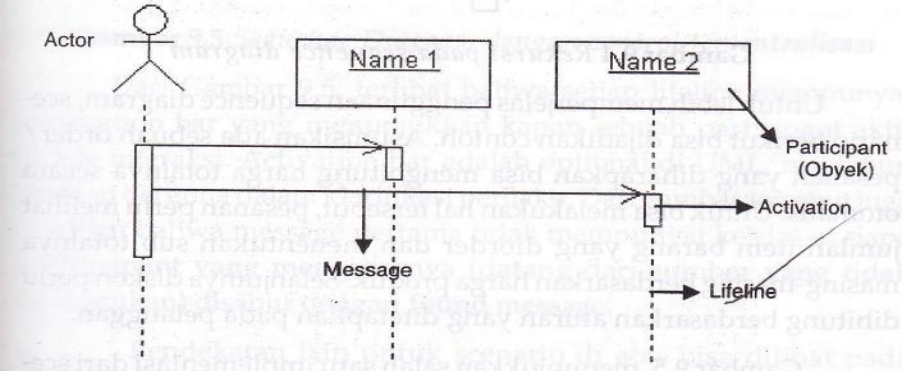
Gambar II.19 Bentuk *Messege*

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.20



Gambar II.20 *Squence Diagaram*

Sumber: (Munawar,2005:89)

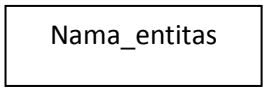
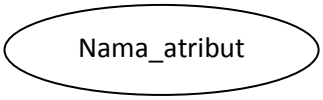
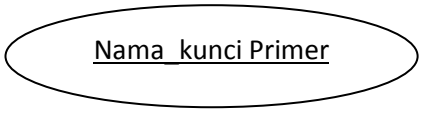
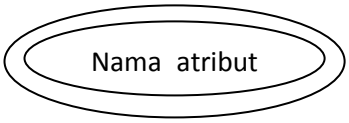
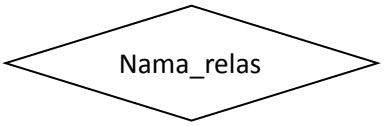
II.9. ERD (*Entity Relationship Diagram*)

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.9.1. Simbol-simbol ERD (*Entity Relationship Diagram*)

Adapaun simbol-simbol ERD (*Entity Relationship Diagram*) ditunjukkan pada tabel II.3.

Tabel II.3. 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.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 (Ir. Yuniar Supardi;2008:10-12).