BAB II

TINJAUAN PUSTAKA

II.1. Data

Data adalah fakta tercatat tentang sesuatu objek. Jadi apapun yang berupa catatan tentang sesuatu objek dapat disebut data. Misalnya berat badan si A adalah 60 Kg, maka 60 Kg tersebut adalah data. Data di dalam dunia komputer terkenal dengan istilah data multimedia. Data multimedia ini pada dasarnya dapat dikelompokkan menjadi: data teks, data numeris, data gambar/image, data video dan data audio. *Nugroho, eko(2008:13)*

Istilah data dan *file* silih berganti digunakan ataupun secara bersama-sama. *File* adalah pengarsipan dalam suatu media yang terdiri dari kumpulan karakter dan didokumentasikan dalam bentuk digital pada komputer. Sehingga, sering sekali istilah *file* ataupun data silih berganti digunakan untuk mengacu pada objek yang sama. Penggunaan istilah "data teks" atau "*file* teks" sama-sama mengacu kepada objek yang sama, perbedaan pengertian antara keduanya tersebut tidak begitu jelas. Namun, istilah data biasanya digunakan untuk mendeskripsikan apa yang menjadi isi suatu *file*. Berbagai jenis data antara lain: data gambar, data teks, data suara, dan lain – lain.

Pemakaian tipe data yang sesuai di dalam proses pemrograman akan menghasilkan algoritma yang jelas dan tepat, sehingga menjadikan program secara keseluruhan lebih efisien dan sederhana.

II.1.1. Kompresi Data

Kompresi data merupakan suatu upaya untuk mengurangi jumlah bit yang digunakan untuk menyimpan atau mentransmisikan data. Kompresi data meliputi berbagai teknik kompresi yang diterapkan dalam bentuk perangkat lunak *(software)* maupun perangkat keras *(hardware).* Bila ditinjau dari sisi penggunaannya, kompresi data bisa bersifat umum untuk segala keperluan atau bersifat khusus untuk keperluan tertentu. Keuntungan data yang terkompresi antara lain mengurangi *bottleneck* pada proses *I/O* dan transmisi data, penyimpanan data lebih hemat ruang, mempersulit pembacaan data oleh pihak yang tidak berkepentingan, dan memudahkan distribusi data dengan media *removable* seperti *flashdisk, CD, DVD,* dan lain-lain. *Kandaga, Tjatur (2006 : 81)*

Seperti diketahui bahwa sebuah data terkadang memiliki nformasi yang berulang-ulang yang membuat ukuran sebuah data menjadi besar. Dengan menggunakan algoritma dan teknik-teknik tertentu, informasi yang sama dan berulang-ulang tersebut dikodekan sedemikian rupa sehingga data tersebut menjadi berukuran lebih kecil. *File* atau data yang sudah terkompres agar bisa digunakan kembali harus dikembalikan lagi seperti *file* aslinya disebut proses dekompresi. *Byta (2007).*

Pengiriman data hasil kompresi dapat dilakukan jika pihak pengirim atau yang melakukan kompresi dan pihak penerima memiliki aturan yang sama dalam hal kompresi data. Pihak pengirim harus menggunakan algoritma kompresi data yang sudah baku dan pihak penerima juga mengunakan teknik dekompresi data

yang sama dengan pengirim sehingga data yang diterima dapat dibaca atau di-dekode kembali dengan benar.

Misalnya terdapat kata “Hari ini adalah hari Jum’at. Hari Jum’at adalah hari yang menyenangkan”. Jika ditelaah lagi, kalimat tersebut memiliki pengulangan karakter seperti karakter pembentuk kata hari, hari jumat, dan adalah. Dalam teknik sederhana kompresi pada perangkat lunak, kalimat di atas dapat diubah menjadi pola sebagai berikut “# ini $ %. % $ # ya@ menyena@kan”. Di mana dalam kalimat diatas, karakter pembentuk hari diubah menjadi karakter #, hari Jum’at menjadi %, adalah menjadi $, ng menjadi @. Saat berkas ini akan dibaca kembali, maka perangkat lunak akan mengembalikan karakter tersebut menjadi karakter awal dalam kalimat.

II.1.2. Dekompresi Data.

Dekompresi adalah proses *decoding* yang mengambil aliran *bit* dari *file* hasil kompresi untuk dekompresi dan menghasilkan ukuran penuh yang benar dari data asli atau *file* sebelum dikompresi.

II.1.3. *Encoding* dan *Decoding*

Encoding merupakan teknik untuk mendapatkan kode-kode tertentu. Dari kode-kode tersebut dapat diaplikasikan untuk kompresi data dan keamanan data. Dari data-data yang telah dikodekan tersebut, format-format isi dari data tersebut berbentuk kode-kode yang tidak bisa dibaca oleh user. Agar kode-kode tersebut bisa dibaca oleh user, maka kita perlu mengkodekan ulang data tersebut. Hal ini biasa dikenal dengan nama *decoding*. Proses *decoding*, yaitu proses pengembalian

kode-kode yang telah dibuat menjadi simbol-simbol yang dikenal oleh user.

II.1.3. Jenis Teknik Kompresi

Menurut *Restyandito (2008),* Teknik kompresi secara umum dapat diklasifikasikan menjadi tiga bagian yaitu:

a. *Entropy coding,* adalah teknik kompresi yang menggunakan proses *lossless.* Tekniknya tidak berdasarkan pada media dengan spesifikasi dan karakteristik tertentu namun berdasarkan urutan data serta tidak memperhatikan semantik data.
b. *Source coding,* adalah teknik kompresi dengan menggunakan proses *lossy.* Teknik ini berkaitan dengan data semantik (arti data) dan media.
c. *Hybrid coding* adalah teknik kompresi dengan menggunakan kombinasi atau gabungan dari *entropy coding* dan *source coding.*

Berdasarkan outputnya, teknik kompresi dapat dibedakan menjadi dua jenis yaitu :

a. Teknik kompresi *Lossy*

Kompresi menggunakan *lossy,* beberapa bagian data asli hilang ketika berkas di *decoded.* Keuntungan dari algoritma ini adalah bahwa rasio kompresi cukup tinggi. Hal ini dikarenakan cara algoritma *lossy* yang mengeliminasikan beberapa data dari suatu berkas. Namun data yang dieliminasikan biasanya adalah data yang kurang diperhatikan atau di luar jangkauan manusia, sehingga pengeliminasian data tersebut kemungkinan besar tidak akan mempengaruhi manusia yang berinteraksi dengan berkas tersebut.

b. Teknik kompresi *Lossless*

Kompresi menggunakan *lossless* menjamin bahwa berkas yang dikompresi dapat selalu dikembalikan ke bentuk aslinya. Algoritma *lossless* digunakan untuk kompresi berkas text, seperti program komputer (berkas *zip, rar,* dll), karena jika ingin mengembalikan berkas yang telah dikompres ke status aslinya. Kadangkala ada data-data yang setelah dikompresi dengan teknik ini ukurannya menjadi lebih besar atau sama. Berdasarkan teknik pengkodean/pengubahan simbol yang digunakan, metode kompresi dapat dibagi ke dalam tiga kategori, yaitu:

a. Metode *symbolwase,* menghitung peluang kemunculan dari tiap simbol dalam *file* input, lalu mengkodekan satu simbol dalam satu waktu, di mana simbol yang lebih sering muncul diberi kode lebih pendek dibandingkan simbol yang lebih jarang muncul. Contoh: algoritma *Huffman.*
b. Metode *dictionary,* menggantikan karakter/*fragmen* dalam *file* input dengan indeks lokasi dari karakter/*fragmen* tersebut dalam sebuah kamus *(dictionary).* Contoh: algoritma *LZW. Deflate* dan lain lain
c. Metode *predictive*, menggunakan model *finite-context* atau *finite-state* untuk memprediksi distribusi probabilitas dari simbol-simbol selanjutnya. Contoh: algoritma *DMC.*

Jenis kompresi data berdasarkan mode penerimaan data oleh manusia adalah:

a. Dialogue *Mode:* yaitu proses penerimaan data di mana pengirim dan penerima seakan berdialog *(real time),* seperti pada contoh video *conference.* Di mana kompresi data harus berada dalam batas penglihatan dan pendengaran

manusia. Waktu tunda *(delay)* tidak boleh lebih dari 150 ms, di mana 50 ms untuk proses kompresi dan dekompresi, 100 ms mentransmisikan data dalam jaringan.

b. *Retrieval Mode*: yaitu proses penerimaan data tidak dilakukan secara *realtime.* Dapat dilakukan *fast forward* dan *fast rewind* di *client.* Dapat dilakukan *random access* terhadap data dan dapat bersifat interaktif

Terdapat beberapa faktor yang sering menjadi pertimbangan dalam memilih suatu metode kompresi yang tepat karena tidak ada suatu metode kompresi yang paling efektif untuk semua jenis *file.* Faktor-faktor tersebut adalah:

a. Kualitas data hasil enkoding: ukuran lebih kecil, data tidak rusak untuk kompresi *lossy.*
b. Ketepatan proses dekompresi data: data hasil dekompresi tetap sama dengan data sebelum dikompresi (kompresi *loseless).*
c. Kecepatan, ratio, dan efisiensi proses kompresi dan dekompresi

II.1.4. Rasio Kompresi

Ketika pengujian diperoleh waktu yang diperlukan pada saat kompresi dan dekompresi sedikit berubah-ubah, maka pengujian untuk setiap metode dilakukan sebanyak sepuluh kali untuk *file* data yang sama, kemudian diambil nilai rata-ratanya. Rasio kompresi dihitung menggunakan rumus :

Rasio = 100% - (ukuran terkompresi / ukuran asal) x 100%

Yang berarti semakin tinggi nilai rasio semakin baik pula tingkat kompresinya. Jika nilainya nol berarti tidak terjadi kompresi, dan jika nilainya negatif berarti proses kompresi telah membuat *file output* semakin besar. Waktu yang diperlukan pada saat proses kompresi dan dekompresi disajikan dalam satuan detik, dengan ketepatan sampai seperseratus detik. *Kandaga, Tjatur (2006 : 91)*

II.2. Algoritma LZ77

Penelitian pada kompresi data hingga tahun 1977 berkonsentrasi kepada cara-cara mengembangan metode *Huffman*. Segalanya berubah pada tahun tersebut. Publikasi "*A Universal Algorithm for Sequential Data Compression*" oleh Jacob Ziv dan Abraham Lempel mengemukakan metode baru, yaitu metode berbasis *dictionary*. Teknik kompresi yang dikembangkan dalam dokumen tersebut bernama Lempel-Ziv 77 (LZ77). Algoritma LZ77 adalah teknik "*sliding window*" dimana menggunakan teks yang dilihat sebelumnya sebagai *dictionary* terhadap teks yang akan diproses (*Salomon, D, 2007*)

Algoritma ini menetapkan sebuah jendela (*window*). Rangkaian input (masukan) akan bergerak dari arah kanan ke kiri pada jendela. Atau dengan sudut pandang lain, jendela ini bergerak dari arah kiri ke kanan terhadap teks. Jendela tersebut dibagi atas dua bagian. Bagian sebelah kiri dinamakan *search buffer*, sebagai *dictionary* yang berisi rangkaian simbol yang sudah diproses. Bagian sebelah kanan dinamakan *look-ahead buffer*, berisi rangkaian simbol sebagai input yang akan diproses. Ukuran dari masing-masing *buffer* dalam implementasi boleh jadi bervariasi. Memperluas area pencarian (search *buffer*) memungkinkan

algoritma mencari rangkaian simbol terpanjang yang sesuai dengan masukan (*look-ahead buffer*). Memperluas area masukan berarti memungkinkan panjang rangkaian simbol yang mungkin sesuai semakin besar. Namun, umumnya search *buffer* berukuran 2-8 K*byte*s dan *look-ahead buffer* berukuran 8-32 *byte*s.

II.2.1 Proses *Encoding* Algoritma LZ77

Proses *encoding* pada algoritma LZ77 ini dengan cara mencari nilai P,Ldan S. Nilai P adalah posisi karakter yang sama pada *lookahead buffer* dengan *dictionary buffer*, nilai L adalah jumlah karakter yang sama pada *lookahead buffer* dengan *dictionary buffer* dan nilai S adalah karakter yang akan disimpan pada *dictionary buffer*. Output dari proses *encoding* pada algoritma LZ77 ini adalah berupa *string* gabungan dari nilai-nilai P,L dan S sampai dengan tidak ada lagi karakter pada *lookahead buffer*.

II.2.2 Proses *Decoding* Algoritma LZ77

Proses *decoding* pada algoritma LZ77 ini dilakukan dengan cara mengambil 3 buah karakter dan kemudian digabungkan menjadi sebuah kelompok karakter. Setiap kelompok karakter yang telah dibuat berisikan nilai P, L dan S. Nilai P adalah karakter pertama pada kelompok karakter, nilai L adalah karakter kedua pada kelompok karakter dan nilai S adalah karakter ketiga pada kelompok karakter. *String* diambil dengan cara mencari posisi karakter pada *dictionary buffer* sesuai dengan nilai P, lalu mengambil banyaknya karakter yang akan diambil pada *dictionary buffer* sesuai dengan nilai S dan gabungkan karakter yang telah diambil dengan nilai S. Hasil dari proses *decoding* ini adalah *string*

gabungan dari karakter-karakter yang telah diambil dari *dictionary buffer*.

II.3. Algoritma *Huffman*

Algoritma *Huffman*, yang dibuat oleh seorang mahasiswa MIT bernama David *Huffman* pada tahun 1952, merupakan salah satu metode paling lama dan paling terkenal dalam kompresi teks. Algoritma *Huffman* menggunakan prinsip pengkodean yang mirip dengan kode *Morse*, yaitu tiap karakter (simbol) dikodekan hanya dengan rangkaian beberapa bit, dimana karakter yang sering muncul dikodekan dengan rangkaian bit yang pendek dan karakter yang jarang muncul dikodekan.dengan rangkaian bit yang lebih panjang.

Berdasarkan tipe peta kode yang digunakan untuk mengubah pesan awal (isi data yang diinputkan) menjadi sekumpulan *codeword*, algoritma *Huffman* termasuk kedalam kelas algoritma yang menggunakan metode *static*. Metoda *static* adalah metoda yang selalu menggunakan peta kode yang sama, metoda ini membutuhkan dua fase (*two-pass*). Fase pertama untuk menghitung probabilitas kemunculan tiap simbol dan menentukan peta kodenya, dan fase kedua untuk mengubah pesan menjadi kumpulan kode yang akan di taransmisikan.

Sedangkan berdasarkan teknik pengkodean simbol yang digunakan, algoritma *Huffman* menggunakan metode *symbolwise*. Metode *symbolwise* adalah metode yang menghitung peluang kemunculan dari setiap simbol dalam satu waktu, dimana simbol yang lebih sering muncul diberi kode lebih pendek dibandingkan simbol yang jarang muncul.

Kode *Huffman* pada dasarnya merupakan kode prefiks (*prefix code*). Kode

prefiks adalah himpunan yang berisi sekumpulan kode biner, dimana pada kode prefik ini tidak ada kode biner yang menjadi awal bagi kode biner yang lain. Kode *prefix* biasanya direpresentasikan sebagai pohon biner yang diberikan nilai atau label. Untuk cabang kiri pada pohon biner diberi label 0, sedangkan pada cabang kanan pada pohon biner diberi label 1. Rangkaian bit yang terbentuk pada setiap lintasan dari akar ke daun merupakan kode prefiks untuk karakter yang berpadanan. Pohon biner ini biasa disebut *Huffman Tree*. *Huffman Tree* ini dibentuk dengan cara menghitung terlebih dahulu frekuensi kemunculan karakter pada *string*. Kemudian mencari dua buah karakter yang memiliki nilai frekuensi terkecil dan gabungkan kedua karakter tersebut pada sebuah akar. Kemudian cari kembali karakter yang memiliki frekuensi terkecil, dan ulangi langkah tersebut sampai terbentuk sebuah pohon yang daun-daunnya berisi seluruh karakter pada *string*.

II.3.1. Proses *Encoding* Algoritma *Huffman*

Encoding adalah cara menyusun *string* biner dari teks yang ada. Proses *encoding* untuk satu karakter dimulai dengan membuat *Huffman Tree* terlebih dahulu. Setelah itu, kode untuk satu karakter dibuat dengan menyusun nama *string* biner yang dibaca dari akar sampai ke daun *Huffman Tree*. Hasil dari penyusunan nama *string* biner tersebut menjadi kode biner yang baru untuk setiap karakter. Karakter yang memiliki frekuensi kemunculan yang besar akan memiliki kode biner yang lebih pendek dari pada karakter yang memiliki frekuensi kemunculan yang kecil.

II.3.2. Proses *Decoding* Algoritma *Huffman*

Decoding merupakan kebalikan dari *encoding*. *Decoding* berarti menyusun kembali data dari *string* biner menjadi sebuah karakter kembali. *Decoding* dapat dilakukan dengan dua cara, yang pertama dengan menggunakan *Huffman Tree* dan yang kedua dengan menggunakan tabel kode *Huffman*. Melakukan proses *decoding* dengan menggunakan *Huffman Tree* dilakukan dengan cara menelusuri *Huffman Tree* dan mencatat *string* biner yang ada pada cabang sampai dengan karakter yang dicari ditemukan. Sedangkan menggunakan tabel kode *Huffman* adalah dengan cara membuat sabuah table yang berisi karakter yang di*encoding* beserta *string* biner yang baru dibentuk dari *Huffman Tree*.

II.4. Metode Kompresi Algoritma *Deflate* Zip

Algoritma *deflate* merupakan algoritma persilangan antara algoritma *Huffman* dan algoritma LZ77. Dalam proses kompresinya, algoritma *deflate* ini terlebih dahulu melakukan proses pengelompokan karakter dengan menggunakan algoritma LZ77. Kemudian hasil dari pengelompokan karakter tersebut dikompresi lagi dengan menggunakan algoritma *Huffman* (*Huffman Tree*). Algoritma *deflate* ini bersifat loseless Compression. Hal ini karena algoritma *deflate* ini menggabungkan dua algoritma kompresi yang bersifat loseless.

Pada algoritma *Deflate* memiliki pilihan atas bagaimana kompresi dilakukan. Walau sebenarnya terdapat 4 mode dalam kompresi *Deflate*, namun 1 tidak digunakan, sehingga pada akhirnya hanya ada tiga mode kompresi pada

kompresor *Deflate* yaitu:

1. Tidak dikompresi sama sekali. Mode ini cocok digunakan untuk data yang sudah melwati proses kompresi. Data yang dikamprosei menggunakan mode ini akan sedikit „membesar, namun tidak sebanyak jika data tersebut sudah dikompresi dan kita coba kompresi lagi dengan mode lainnya. Kompresi dengan mode ini adalah kompresi yang paling mudah dilakukan karena pada dasarnya input dapat langsung diberikan ke output.
2. Kompresi, diawali dengan LZ77 dan diteruskan dengan pengkodean *Huffman*. Pohon yang digunakan pada mode ini ditentukan oleh spesifikasi *Deflate* itu sendiri. Pohon *Huffman* pada mode ini disebut juga pohon *Huffman* statik Menggunakan pohon *Huffman* statik berarti tidak memerlukan ruang ekstra untuk menyimpan pohon tersebut. Namun dengan menggunakan pohon *Huffman* statik kemungkinan besar kompresi yang dilakukan tidaklah optimal.
3. Kompresi, diawali dengan LZ77 dan dilanjutkan dengan pengkodean *Huffman*. Perbedaan mode ini dengan mode pada poin 2 adalah, pada mode ini pohon *Huffman* yang digunakan dirancang dan dibuat oleh kompresor dan disimpan bersama data-data yang dikompresi. Pohon *Huffman* ini disebut juga pohon *Huffman* dinamik. Dengan menggunakan mode ini berarti kita perlu menentukan karakter yang digunakan untuk kompresi blok. Karakter ini kemudian disimpan sebelum data bersifat tetap dalam artian tidak dapat diubah lagi.

Pada kompresi *Deflate,* data input kompresi dipecahkan dalam blok-blok Tiap blok membuat header 3 bit dimana bit pertama adalah penanda blok terakhir pada keseluruhan data, 1 menandakan blok tersebut adalah blok terakhir, dan 0 menandakan masih ada blok lain yang harus diproses setelah blok ini, dan 2 bit terakhir pada header untuk menetukan mode kompresi yang dipakai pada blok ini dengan 00 menandakan penggunaan mode 1, 01 penggunaan mode 2, 10 untk penggunaan mode 3, dan 11 penggunaan mode 4 namun mode ini seharusnya tidak ditemukan dan digunakan. Sebagian besar blok yang dikompresi *Deflate* menggunakan mode 3.

II.4.1. Proses *Encoding* Algoritma *Deflate*

Proses *encoding* dari algoritma *deflate* ini dilakukan pada dua tahap. Tahap pertama adalah melakukan proses pembuatan blok-blok atau penyingkatan karakter dengan menggunakan algoritma LZ77. Tahap kedua adalah mengambil hasil penyingkatan karakter dari algoritma LZ77 dan melakukan proses kompresi dengan menggunakan *Huffman Tree* terhadap karakter tersebut. Hasil dari proses *encoding* algoritma *deflate* ini adalah berupa karakter yang telah disingkat yang merupakan hasil dari *encoding* algoritma LZ77 dan memiliki kode biner yang lebih pendek yang merupakan hasil dari *encoding* algoritma *Huffman.*

II.4.2. Proses *Decoding* Algoritma *Deflate*

Proses *decoding* pada algoritma *deflate* merupakan kebalikan dari proses *encoding*nya. Langkah pertama adalah melakukan proses *decoding* dengan menggunakan algoritma *Huffman*. Kemudian langkah kedua adalah mengambil

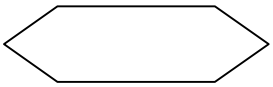
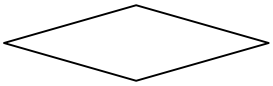
hasil dari proses *decoding* dengan menggunakan algoritma *Huffman* dan melakukan proses *decoding* kembali dengan menggunakan algoritma LZ77.

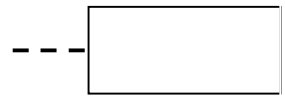
II.5. *Flowchart*

Bagan alir program (*program flow-chart*) adalah suatu bagan yang menggambarkan arus logika dari data yang akan diproses dalam suatu program dari awal sampai akhir. Bagan alir program merupakan alat yang berguna bagi *programmer* untuk mempersiapkan program yang rumit. Bagan alir terdiri dari simbol-simbol yang mewakili fungsi-fungsi langkah program dan garis alir (*flow lines*) menunjukkan urutan dari simbol-simbol yang akan dikerjakan. *Jogiyanto Hartono (2004:662)*

Berikut adalah simbol-simbol program *flowchart* menurut *ANSI* (*American National Standard Institute*), yang dapat di lihat pada tabel 2.3 berikut :

Tabel II.1. Simbol *Flowchart*

SIMBOL	KETERANGAN
	**Terminal** Untuk menunjukkan awal dan akhir program.
	**Persiapan (*preparation*)** Untuk memberikan nilai awal pada suatu variabel atau *counter*.
	**Pengolahan (*processing*)** Untuk pengolahan arithmatika dan pemindahan data.
	**Keputusan (*decision*)** Untuk mewakili operasi perbandingan logika.
	**Proses terdefinisi (*predifined process*)** Untuk proses yang detilnya dijelaskan terpisah, misalnya dalam bentuk subroutine.
	**Penghubung (*connector*)** Untuk menunjukkan hubungan arus proses yang terputus masih dalam halaman yang sama.
	**Penghubung halaman lain (*off page connector*)** Untuk menunjukkan hubungan arus proses yang terputus masih dalam halaman yang sama.

	**Penjelasan (*annotation flag*)** Untuk memberikan keterangan-keterangan guna memperjelas simbol-simbol yang lain.

Sumber : Jogiyanto Hartono (2004:662)

II.6 UML

II.6.1 Pengertian UML

UML singkatan dari *Unified Modeling Language* yang berarti bahasa pemodelan standar. (Cloneles, 2003 : bab1) mengatakan sebagai bahasa, berarti UML memiliki sintaks dan semantik. Ketika kita membuat model mengguunakan konsep UML ada aturan - aturan yang harus diikuti. Bagaimana elemen pada model – model yang kita buat berhubungan satu dengan lainnya harus mengikuti standar yang ada. UML bukan hanya sekedar diagram, tetapi juga menceritakan konteksnya. Ketika pelanggan memesan sesuatu dari system, bagaimana transaksinya? Bagaimana system mengatasi *error* yang terjadi? Bagaimana keamanan terhadap system yang kita buat? Dan sebagainya dapat dijawab dengan UML. (*Sumber* : Prabowo Pudjo Widodo dan Heriawanti, 2006: 7)

UML diaplikasikan untuk maksud tertentu, biasanya antara lain untuk:

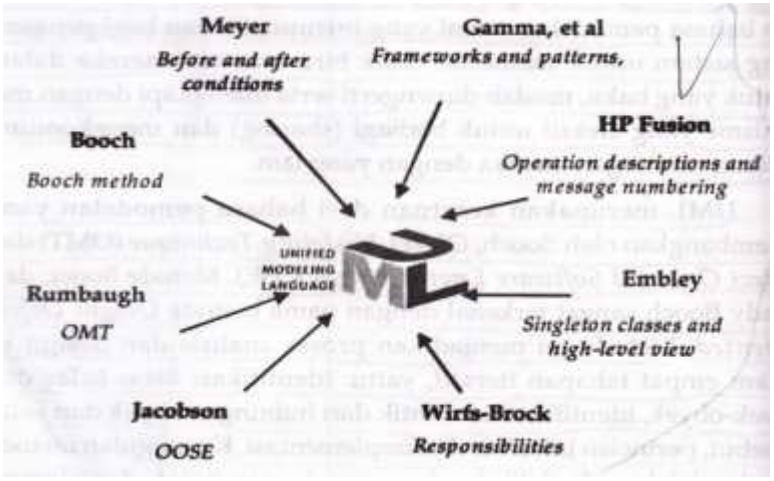
1. Merancang Perangkat Lunak
2. Sarana komunikasi antara perangkat lunak dengan proses bisnis
3. Menjabarkan system secara rinci untuk analisa dan mencari apa yang diperlukan system.

4. Mendokumentasi system yang ada, proses – proses dan organisasinya.UML telah diaplikasikan dalam bidang investasi perbankan, lembaga kesehatan, departemen pertahanan, system terdistribusi, sistem pendukung alat kerja, retail, sales dan *supplier*.

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 sederhana namun 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.1



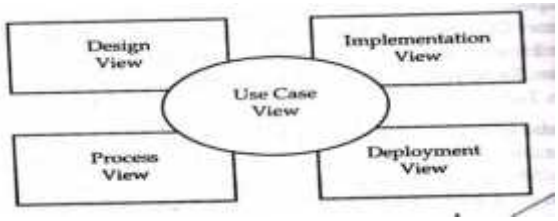
Gambar II.1 Unsur-unsur yang membentuk UML

Sumber: Pemodelan Visual dengan UML, (*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.2



Gambar II.2 Model 4+1 *View*

Sumber: Pemodelan Visual dengan UML, *(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.6.2 Use Case Diagram

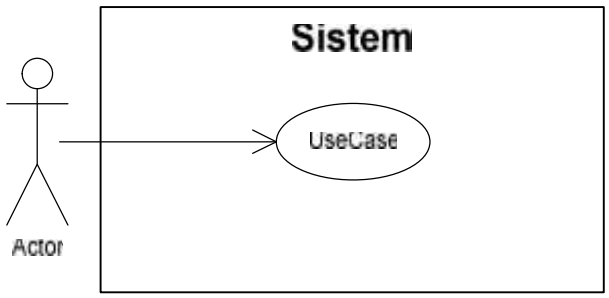
Use case adalah deskripsi fungsi dari sebuah sistem dari perspektif pengguna. *Use case* bekerja dengan cara deskripsikan tipikal interaksi antara *user* (pengguna) sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai. Urutan langkah-langkah yang menerangkan antara 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.3



Gambar: II.3 *Usecase* Diagram

Sumber: Pemodelan Visual dengan UML, *(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.6.3 Class Diagram

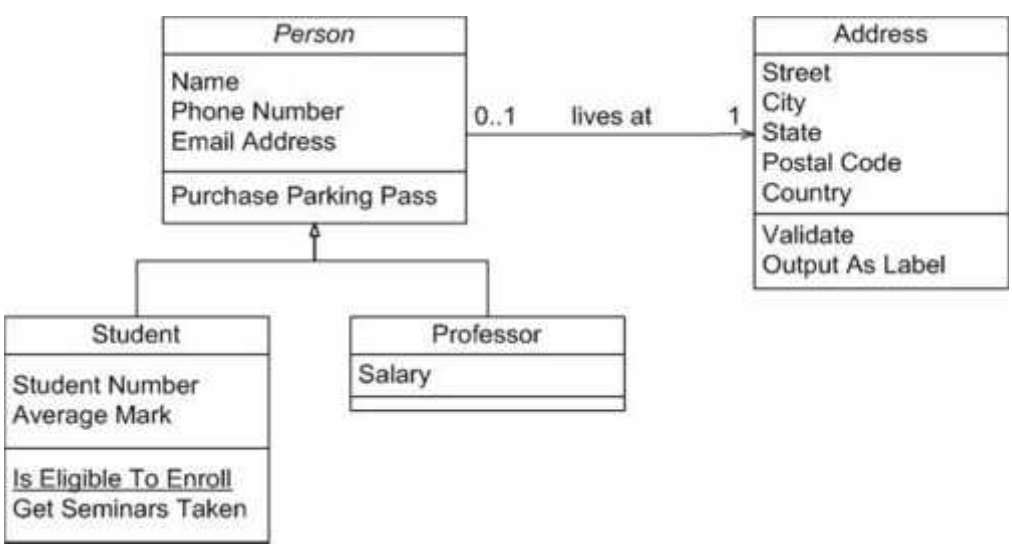
Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (*atribut*/ properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *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 sifa tberikut :

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.4 dibawah ini:

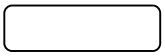
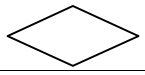
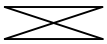
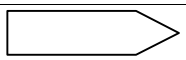
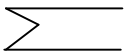


Gambar II.4 *Class* Diagram

Sumber: Pemodelan Visual dengan UML, *(Munawar, 2005:220)*

II.6.4 *Activity* Diagram

Activity Diagram adalah teknik untuk mendiskripsikan logika prosedural, proses bisnis dan aliran kerja dalam banyak kasus.*Activity Diagram* mempunyai peran seperti halnya *flowchart*, akan tetapi perbedaannya dengan *flowchart* adalah *activity* diagram bisa mendukung perilaku paralel sedangkan flowchart tidak bisa.Adapun simbol *activity diagram* dapat dilihat pada table II.2 :

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
	Tandawaktu
	Tandapengiriman
	Tandapenerimaan
	AliranAkhir (*Flow Final*)

Tabel II.2. Simbol *Activity Diagram*
Sumber :Pemodelan Visual dengan UML, *(Munawar, 2005:110)*

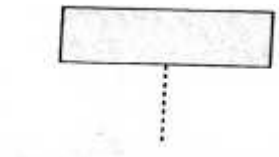
II.6.5. *Squence Diagram*

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.5



Gambar II.5 Bentuk *Participant*

Sumber: Pemodelan Visual dengan UML, *(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.

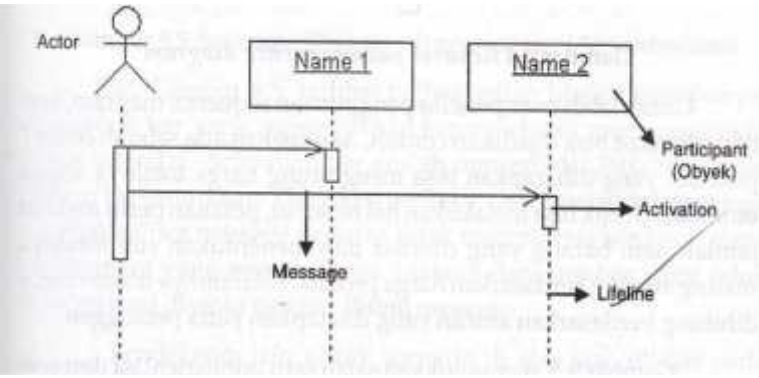
Sebuah *message* bisa jadi *simple, synchronous* atau *asynchoronous. Message* yang *simple* adalah sebuah perpindahan (transfer), contoh dari satu *participant* ke *participant*yang lainnya. Jika sebuah *participant*mengirimkan sebuah *messagae* tersebut akan ditunggu sebelum diproses dengan urusannya. Namun jika *message asynchoronous* yang dikirimkan, maka jawabannya atas *message* tersebut tidak perlu ditunggu. Simbol *message* pada *squnence diagram* dapat dilihat pada gambar II.6

Gambar II.6 Bentuk *Messege*

Sumber: Pemodelan Visual dengan UML, *(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 bawahmenunjukkan lintasan waktu. Simbol-simbol yang ada pada *squence diagram* ditunjukkan pada gambar II.7



Gambar II.7 Bentuk *Time*

Sumber: Pemodelan Visual dengan UML, *(Munawar,2005:89)*

II.7. Pseudo*code*

Pseudocode adalah deskripsi dari algoritma pemrograman computer yang menggunakan struktur sederhana dari beberapa bahasa pemograman tetapi bahasa tersebut hanya ditujukan agar dapat dibaca manusia. Biasanya yang ditulis dari *pseudocode* adalah variabel dan fungsi. Tujuan penggunaan utama dari *pseudocode* adalah untuk memudahkan manusia dalam memahami prinsip-prinsip dari suatu algoritma. Penggunaan *pseudocode* umumnya banyak ditemukan di buku-buku dan artikel-artikel tentang pemrograman yang membahas tentang algoritma tertentu. Kadang pula *pseudocode* ditemukan dalam merencanakan pengembangan suatu program komputer. Dalam *pseudocode*, tidak ada *syntax* standar yang resmi.

Karena itu, *pseudocode* ini dapat diterapkan dalam berbagai bahasa pemograman. Tentu saja harus kita sesuaikan setiap tahap dengan bahasa pemograman yang kita gunakan. Fungsi dari *pseudocode* mungkin sama dengan *Flowchart.* Perbedaannya terletak pada cara penyampaiannya. *Pseudocode*

menggunakan kata-kata untuk menjelaskan suatu algoritma, sedangkan *Flowchart* menggunakan gambar. Contoh Algoritma dari *pseudocode* dapat dilihat pada table berikut:

Tabel II.3. Contoh *Pseudocode*

Algoritma	**Pseudo *Code***
Masukan Panjang	Input Panjang
Masukan Lebar	Input Lebar
Nilai Luas adalah panjang x lebar	Luas = panjang x lebar
Tampilkan luas	Print luas

Sumber :linawati (2005 : 9)

II.8. Visual Basic Net 2008

Microsoft Visual Basic .NET adalah sebuah alat untuk mengembangkan dan membangun aplikasi yang bergerak di atas sistem *.NET Framework*, dengan menggunakan bahasa *BASIC*. Dengan menggunakan alat ini, para *programmer* dapat membangun aplikasi *Windows Forms*, Aplikasi web berbasis ASP.NET, dan juga aplikasi *command-line*. Alat ini dapat diperoleh secara terpisah dari beberapa produk lainnya (seperti *Microsoft Visual C++*, Visual C#, atau *Visua*l J#), atau juga dapat diperoleh secara terpadu dalam *Microsoft Visual Studio .NET*. Bahasa *Visual Basic .NET* sendiri menganut paradigma bahasa pemrograman berorientasi objek yang dapat dilihat sebagai evolusi dari *Microsoft Visual Basic* versi sebelumnya yang diimplementasikan di atas *.NET Framework.* Peluncurannya

mengundang kontroversi, mengingat banyak sekali perubahan yang dilakukan oleh *Microsoft,* dan versi baru ini tidak kompatibel dengan versi terdahulu.

Visual Basic 2008 adalah merupakan versi terbaru yang dirilis oleh *Microsoft* pada tanggal 19 November 2007, bersamaan dengan dirilisnya *Microsoft Visual* C# 2008, Microsoft Visual C++ 2008, dan *Microsoft .NET Framework* 3.5.

Dalam versi ini, *Microsoft* menambahkan banyak fitur baru, termasuk di antaranya adalah:

a. Operator If sekarang merupakan operator *ternary* (membutuhkan tiga operand), dengan sintaksis If (boolean, nilai, nilai). Ini dimaksudkan untuk mengganti fungsi *IFF*

b. Dukungan *anonymous types*

c. Dukungan terhadap *Language Integrated Query* (LINQ)

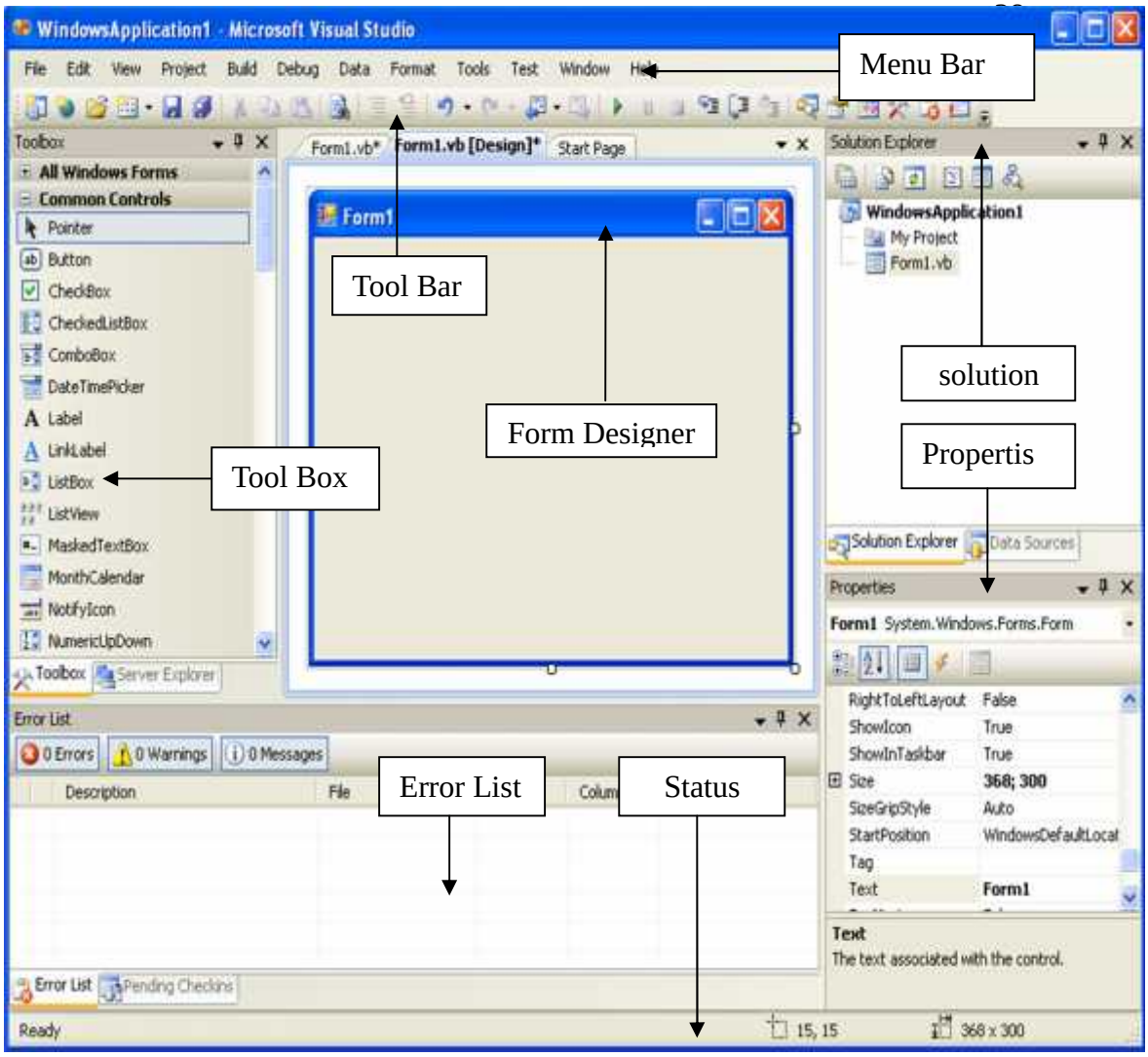
d. Dukungan terhadap *ekspresi Lambda*

e. Dukungan terhadap *literal XML*

f. Dukungan terhadap *inferensi tipe data*.

g. dukungan terhadap 'LINQ'

Adapun tampilan *interface Visual Studio.Net* 2008 dapat dilihat pada gambar II.8 berikut :



Gambar II.8 Tampilan Visual Studio.Net
Sumber : Andi (2006 : 8)

1. *Menu bar,* Menu standar pada *Visual Basic.Net* atau *Visual Studio* 2008
2. *Toolbar,* Daftar *tool* (perangkat) untuk menjalankan perintah yang sering digunakan.
3. *Toolbox,* Daftar kontrol yang dapat ditambahkan ke dalam program sebagai antarmuka *(interface).*
4. *Form Designer*, Digunakan untuk mengedit tampilan *form* serta mengatur posisi kontrol pada *form.*
5. *Solution Explorer,* Digunakan untuk mengolah *file* dan proyek yang berhubungan dengan *Solution Explorer.*

6. *Properties,* Digunakan untuk mengedit properti dari *form* dan *kontrol* yang sedang diedit.
7. *Error list,* Menampilkan pesan *error* jika ada kesalahan.
8. Status, Menampilkan status aplikasi saat dijalankan.

II.8.1 *Net Frame work*

.*NET Framework* merupakan suatu komponen *Windows* yang terintegrasi yang dibuat dengan tujuan pengembangan berbagai macam aplikasi serta menjalankan aplikasi generasi mendatang termasuk pengembangan aplikasi XML *Web Services.*

Keuntungan Menggunakan .*NET Framework* :

1. Mudah, Yang dimaksud mudah di sini adalah kemudahan developer untuk membuat aplikasi yang dijalankan di .*NET Framework*. Mendukung lebih dari 20 bahasa pemrograman : *VB.NET*, *C#*, *J#*, *C++*, *Pascal*, *Phyton (IronPhyton)*, *PHP (PhLager)*.
2. Efisien, Kemudahan pada saat proses pembuatan aplikasi, akan berimplikasi terhadap efisiensi dari suatu proses produktivitas, baik efisien dalam hal waktu pembuatan aplikasi atau juga efisien dalam hal lain, seperti biaya (*cost*).
3. Konsisten, Kemudahan-kemudahan pada saat proses pembuatan aplikasi, juga bisa berimplikasi terhadap konsistensi pada aplikasi yang kita buat. Misalnya, dengan adanya *Base Class Library*, maka kita bisa menggunakan objek atau *Class* yang dibuat untuk aplikasi berbasis *window*s pada aplikasi berbasis

web. Dengan adanya kode yang bisa dintegrasikan ke dalam berbagai macam aplikasi ini, maka konsistensi kode-kode aplikasi kita dapat terjaga.

4. Produktivitas, Semua kemudahan-kemudahan di atas, pada akhirnya akan membuat produktivitas menjadi lebih baik. Produktivitas naik, terutama produktivitas para *developer*, akan berdampak pada meningkatnya produktivitas suatu perusahaan atau *project*