Bab 2
Studi Pustaka

2.1 Konsep Dasar Sistem

Terdapat dua kelompok pendekatan di dalam mendefinisikan sistem, yaitu yang menekankan pada prosedurnya dan menekankan pada komponen atau elemennya. Pendekatan sistem lebih menekankan pada prosedur mendefinisikan sistem sebagai berikut ini.

> *"Suatu sistem adalah sebuah jaringan kerja dari prosedur-prosedur yang saling berhubungan, berkumpul bersama-sama untuk melakukan suatu kegiatan atau untuk menyelesaikan suatu sasaran tertentu"*.

Pendekatan sistem yang merupakan jaringan kerja dari prosedur lebih menekankan urutan operasi di dalam sistem. Prosedur (*procedure*) didefinisikan oleh Richard F. Neuschel sebagai berikut ini.

> *"Suatu prosedur adalah suatu urutan operasi klerikal atau disebut juga dengan tulis-menulis, biasanya melibatkan beberapa orang di dalam suatu departemen, yang diterapkan untuk menjamin penanganan yang seragam dari transaksi-transaksi bisnis yang terjadi"*.

Lebih lanjut Jerry FitzGerald, Ardra F. FitzGerald dan Warren D. Stallings, Jr, mendefinikan prosedur sebagai berikut:

> *"Suatu prosedur adalah urut-urutan yang tepat dari tahapan-tahapan instruksi yang menerangkan apa (what) yang harus dikerjakan, siapa (who) yang mengerjakannya, kapan (when) dikerjakan dan bagaimana (how) mengerjakannya"*.

Suatu sistem mempunyai maksud tertentu. Adapun yang menyebutkan maksud dari suatu sistem adalah untuk mencapai tujuan (*goal*) dan ada yang menyebutkan

untuk mencapai suatu sasaran (*objective*). Goal biasanya dihubungkan dengan ruang lingkup yang lebih luas dan sasaran dalam ruang lingkup yang lebih sempit.

2.1.1 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*), pengolahan (*process*), dan sasaran (*objectives*) atau tujuan (*goal*).

Karakteristik atau sifat-sifat sistem diantaranya adalah:

1. Komponen Sistem (*components*)

   Suatu sistem terdiri dari sejumlah komponen yang saling berinteraksi, yang artinya saling bekerja sama membentuk kesatuan. Komponen-komponen sistem atau elemen-elemen sistem dapat berupa subsistem atau bagian-bagian dari sistem. Setiap subsistem mempunyai sifat-sifat dari sistem untuk menjalankan suatu fungsi tertentu dan mempengaruhi proses sistem secara keseluruhan. Suatu sistem dapat mempunyai suatu sistem yang lebih besar yang disebut *supra system*.

2. Batas Sistem (*boundary*)

   Batas sistem merupakan daerah yang membatasi antara suatu sistem dengan sistem yang lainnya atau dengan lingkungan luarnya. Batas sistem ini memungkinkan suatu sistem dipandang sebagai satu kesatuan. Batas suatu sistem menunjukan ruang lingkup (*scope*) dari sistem tersebut.

3. Lingkungan Luar Sistem (*environment*)

   Lingkungan luar dari suatu sistem adalah apapun diluar batas dari sistem yang mempengaruhi operasi sistem. Lingkungan luar sistem dapat bersifat menguntungkan dan dapat juga bersifat merugikan sistem tersebut. Lingkungan luar yang menguntungkan merupakan energi dari sistem dan dengan demikian harus tetap dijaga dan dipelihara. Sedang lingkungan

luar yang merugikan harus ditahan dan dikendalikan, kalau tidak maka akan mengganggu kelangsungan hidup dari sistem.

4. Penghubung Sistem (*interface*)

   Penghubung merupakan media penghubung antara satu subsistem dengan subsistem yang lainnya. Melalui penghubung ini memungkinkan sumber-sumber daya mengalir dari satu subsistem ke subsistem yang lainnya. Keluaran (*ouput*) dari satu subsistem akan menjadi masukan (*input*) untuk subsistem yang lainnya dengan melalui penghubung. Dengan penghubung satu subsistem dapat berintegrasi dengan subsistem yang lainnya membentuk satu kesatuan.

5. Masukan Sistem (*input*)

   Masukan adalah energi yang dimasukkan ke dalam sistem. Masukan dapat berupa masukan perawatan (*maintenance input*) dan masukan sinyal (*signal input*). *Maintenance input* adalah energi yang dimasukkan supaya sistem tersebut dapat beroperasi. *Signal input* adalah energi yang diproses untuk didapatkan keluaran.

6. Keluaran Sistem (*output*)

   Keluaran adalah hasil dari energi yang diolah dan diklasifikasikan menjadi keluaran yang berguna dan sisa pembuangan. Keluaran dapat merupakan masukan untuk subsistem yang lain atau kepada supra sistem.

7. Pengolah Sistem (*process*)

   Suatu sistem dapat mempunyai suatu bagian pengolah yang akan merubah masukan menjadi keluaran.

8. Sasaran Sistem (*objective*)

   Suatu sistem pasti mempunyai tujuan (*goal*) atau sasaran (*objective*), kalau suatu sistem tidak mempunyai sasaran, maka operasi sistem tidak akan ada gunanya. Sasaran dari sistem sangat menentukan sekali masukan yang dibutuhkan sistem dan keluaran yang dihasilkan sistem. Suatu sistem dikatakan berhasil bila mengenai sasaran atau tujuannya.

2.1.2 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*).
    a. Sistem abstrak adalah sistem yang berupa pemikiran atau ide-ide yang tidak tampak secara fisik.
    b. Sistem fisik merupakan sistem yang ada secara fisik.
2. Sistem diklasifikasikan sebagai sistem alamiah (*natural system*) dan sistem buatan manusia (*human made system*).
    a. Sistem alamiah adalah sistem yang terjadi melalui proses alam, tidak dibuat manusia.
    b. Sistem buatan manusia adalah sistem yang dirancang oleh manusia. Sistem buatan manusia yang melibatkan interaksi antara manusia dengan mesin disebut dengan *human-machine system* atau disebut juga dengan *man-machine system*, karena menyangkut penggunaan komputer yang berinteraksi dengan manusia.
3. Sistem diklasifikasikan sebagai sistem tertentu (*deterministic system*) dan sistem tidak tentu (*probabilistic system*).
    a. Sistem tertentu beroperasi dengan tingkah laku yang sudah dapat diprediksi. Interaksi diantara bagian-bagiannya dapat dideteksi dengan pasti, sehingga keluaran dari sistem dapat diramalkan. Sistem komputer adalah contoh dari sistem tertentu yang tingkah lakunya dapat dipastikan berdasarkan program-program yang dijalankan.
    b. Sistem tidak tentu adalah sistem yang kondisi masa depannya tidak dapat diprediksi karena mengandung unsur probabilitas.
4. Sistem diklasifikasikan sebagai sistem tertutup (*closed system*) dan sistem terbuka (*open system*)
    a. Sistem tertutup merupakan sistem yang tidak berhubungan dan tidak terpengaruh dengan lingkungan luarnya. Sistem ini bekerja secara otomatis tanpa adanya turut campur tangan dari pihak diluarnya.

Secara teoritis tertutup ini ada, tetapi kenyataannya tidak ada sistem yang benar-benar tertutup, yang ada hanyalah *relatively closed system* (secara relatif tertutup, tidak benar-benar tertutup).

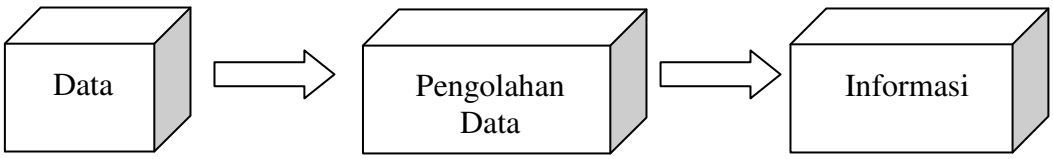
b. Sistem terbuka adalah sistem yang berhubungan dan terpengaruh dengan lingkungan luarnya. Sistem ini menerima masukan dan menghasilkan keluaran untuk lingkungan luar atau subsistem lainnya. Karena sistem sifatnya terbuka dan terpengaruh oleh lingkungan luarnya, maka suatu sistem harus mempunyai suatu sistem pengendalian yang baik. Sistem yang baik harus dirancang sedemikian rupa, sehingga secara relatif tertutup karena sistem tertutup akan bekerja secara otomatis dan terbuka hanya untuk pengaruh yang baik saja.

2.2 Informasi

Informasi merupakan salah satu sarana untuk memperkenalkan suatu perusahaan atau organisasi, sangat erat hubungannya dengan perkembangan organisasi yang masih dalam tahap perkembangan, dengan tidak adanya informasi maka suatu organisasi tidak akan pernah dapat cepat berkembang seperti apa yang diinginkan. Ada beberapa definisi yang menyebutkan antara lain sebagai berikut :

- **Tata Sutabri (2005 : 23) "**informasi adalah data yang telah diklarifikasi atau diolah atau diinterprestasi untuk digunakan dalam proses pengambilan keputusan".
- **Jogiyanto Hartono (1999 : 692)** "informasi adalah hasil dari pengolahan data dalam bentuk yang lebih berguna dan lebih berarti bagi penerimanya yang menggambarkan suatu kejadian-kejadian (*event*) yang nyata (*fact*) yang digunakan untuk pengambilan keputusan".
- **Wing Wahyu Winarno (2004:1.6)** "informasi adalah data yang sudah diolah sehingga berguna untuk pembuatan keputusan. Data adalah representasi suatu objek".

Sumber dari informasi adalah data. Data merupakan bentuk jamak dari bentuk tunggal atau data item. Data adalah kenyataan yang menggambarkan suatu kejadian-kejadian atau kesatuan nyata. Di dalam dunia bisnis, kejadian-kejadian nyata yang sering terjadi adalah perubahan dari suatu nilai yang disebut dengan transaksi. Misalnya penjualan adalah transaksi perubahan nilai barang menjadi nilai uang atau nilai piutang dagang. Kesatuan nyata (*Fact* dan *Entity*) adalah berupa suatu objek nyata seperti tempat, benda, orang yang betul-betul ada dan terjadi.

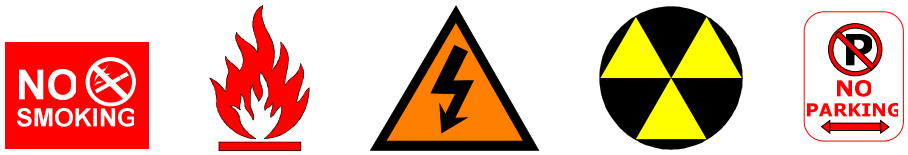


Gambar 2.1 Data Diolah Menjadi Informasi

2.2.1 Bentuk Data dan Informasi

Selama ini data dan informasi yang sering didapat berbentuk tulisan, yang terdiri dari atas huruf dan angka. Seperti pada laporan keuangan, surat kabar, dan membaca kode dan nama barang.

Data dan informasi berbentuk tulisan mengandung kelemahan, yaitu memerlukan waktu bagi pembaca informasi untuk membaca tulisan. Kelemahan lainnya adalah tulisan tidak mudah dipahami oleh semua orang. Sebagai contoh, tulisan "Dilarang Merokok" hanya akan dipahami oleh orang yang memahami bahasa Indonesia, tetapi dengan gambar akan mudah dipahami oleh banyak orang diseluruh dunia. Oleh karenanya, tulisan seringkali tidak cukup untuk menyampaikan informasi. Dengan perkembangan teknologi, terutama teknologi komputer dan multimedia, data dan informasi tidak hanya berbentuk tulisan, akan tetapi sudah lebih variasi yaitu berbentuk gambar, suara, warna, dan getaran.



Gambar 2.2 Gambar Yang Mudah Dipahami Daripada Data Tulisan

Dibawah ini adalah tabel dari bentuk data dan informasi, sebagai berikut:

Tabel 2.1 Bentuk Data dan Informasi

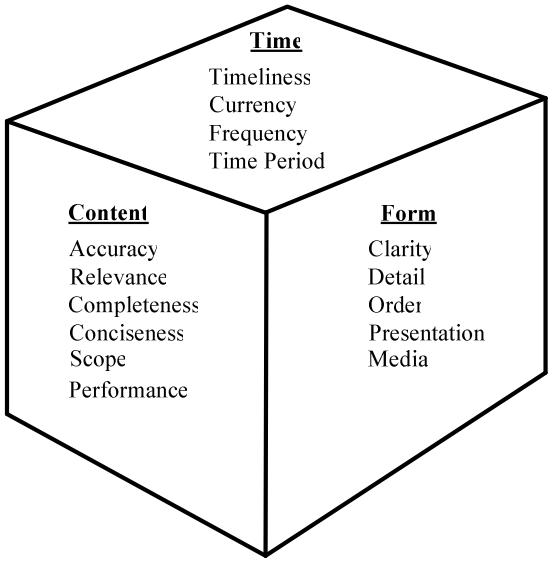
Bentuk Data dan Informasi	Contoh	Keterangan
Tulisan	Nama Orang Tanggal Lahir Harga Barang	Berupa kumpulan huruf dan angka.
Tanda		Berupa bentuk-bentuk khusus yang mudah dipahami.
Warna	Merah, Hijau, Kuning	Warna-warna tertentu menunjukkan maksud-maksud tertentu, misalnya merah berarti berhenti, kuning berarti hati-hati, dan hijau berarti jalan.
Suara	Dering Telepon, Nada tunggu	Suara-suara tertentu yang menunjukkan suatu kejadian, misalnya dering telepon dan nada tunggu pesawat telepon.

2.2.2 Kualitas Informasi

Informasi dapat berguna bagi pemakainya atau bisa juga tidak berguna sama sekali. Informasi akan berguna apabila kualitasnya baik. Baik buruknya kualitas informasi dipengaruhi oleh tiga penentu, yaitu:

- Isi Informasi (*Content*),
- Waktu Penyajian (*Time*),
- Bentuk Informasi (*Form*).

Ketiga penentu kualitas informasi tersebut sebenarnya dapat dikendalikan oleh manajemen, sehingga manajemen harus menjaga agar penentu (variabel) tersebut dapat meningkatkan kualitas informasi. Masing-masing penentu tersebut dapat dirinci lebih lanjut, seperti yang tampak pada gambar dibawah ini dan pada uraian berikut ini.



Gambar 2.3 Kualitas Informasi

2.2.2.1 Isi Informasi (*Content*)

Isi informasi adalah berbagai hal yang dilaporkan. Isi laporan yang baik haruslah memenuhi persyaratan berikut:

a. Akurasi atau ketepatan (*accuracy*), yaitu ketepatan informasi dengan objek yang diwakilinya. Lengkap bukan berarti semakin banyak semakin baik.
b. Relevansi (*relevance*), yaitu kesesuaian antara informasi dengan masalah yang akan dipecahkan berdasarkan informasi tersebut.
c. Kelengkapan (*completeness*), yaitu cukup tidaknya informasi jika digunakan sebagai bahan untuk membuat keputusan.
d. Ringkas (*conciseness*), yaitu jumlah informasi yang harus ditampilkan tanpa berlebihan.
e. Lingkup atau cakupan (*scope*) informasi yang disampaikan harus dapat meliputi semua objek yang harus disampaikan.

f. Kinerja informasi (*performance*) adalah seberapa sering informasi dapat berguna bagi pembuat keputusan.

2.2.2.2 Waktu Penyajian (*Time*)

Kualitas informasi dapat dipengaruhi oleh waktu penyajian. Informasi yang terlambat disajikan (misalnya setelah pembuatan keputusan dilakukan) tidak ada gunanya lagi. Berbagai kualitas informasi yang berhubungan dengan waktu adalah sebagai berikut:

a. Ketepatan waktu (*timeless*), adalah saat informasi disampaikan kepada pembuatan keputusan. Saat yang paling baik adalah pada saat informasi diperlukan untuk membuat keputusan, biasanya pada saat akan dilakukan analisis.
b. Keterkinian (*currency*) informasi, yaitu informasi harus benar-benar mencerminkan keadaan paling baru. Misalnya laporan penjualan.
c. Frekuensi (*frequency*) disampaikannya informasi pada manajemen, harus sesuai dengan frekuensi pembuatan keputusan oleh manajemen. Bila informasi terlalu sering disampaikan, juga belum tentu baik bagi manajemen , karena hanya akan menyebabkan biasanya informasi.
d. Cakupan waktu (*time period*) harus sesuai dengan informasi yang diperlukan. Untuk pembuatan keputusan strategis, informasi yang diperlukan adalah informasi jangka panjang.

2.2.2.3 Bentuk Informasi (*Form*)

Tiap jenis informasi memiliki karakteristik yang berbeda sehingga memerlukan bentuk yang berbeda pula. Informasi yang berhubungan dengan rentang waktu akan lebih mudah bila dipahami dengan grafik berbentuk kurva. Informasi laba atau rugi, akan lebih mudah dipahami bila dinyatakan dengan angka. Pemilihan bentuk informasi akan sangat mempengaruhi keberhasilan pembaca informasi. Bentuk-bentuk informasi diuraikan dibawah ini:

a. Kejelasan (*clarity*) adalah seberapa jelas dab seberapa tajam informasi dapat dipahami pembaca. Misalnya biaya gaji saja, mungkin belum jelas

bagi manajemen, melainkan harus dipisahkan biaya gaji pokok, biaya gaji variable, dan biaya lembur.

b. Rincian (*detail*) laporan harus dapat ditampilkan atau dapat disediakan bila diperlukan oleh pembacanya. Biasanya, manajemen hanya memerlukan informasi secara garis besar saja, misalnya total penjualan.
c. Urutan (*order*) urutan informasi, sangat menentukan kemudahan pembaca laporan. Urutan bisa ditentukan dari urutan nama (abjad) atau unit angka, atau urutan tanggal. Laporan piutang harus ditampilkan berdasarkan urutan tanggal atau urutan saldo, bukan urutan abjad.
d. Cara penyajian (*presentation*), bisa dilakukan dengan grafik, dengan warna, dengan kata-kata, dengan perbandingan, dengan garis runtutan. Informasi yang berbeda memerlukan cara penyajian yang berbeda pula.
e. Sarana (media) pelaporan. Informasi dapat dilaporkan melalui berbagai media, misalnya laporan tercetak dan buku, tampilan di layer, informasi berbentuk film di disk. Informasi sederhana dapat disampaikan melalui media yang sederhana. Informasi yang rumit harus disampaikan melalui media tercetak, agar bisa dibaca berulangkali dengan mudah.

2.3 Manfaat Sistem Informasi

Sistem informasi mendatangkan manfaat bagi berbagai pihak yang terkait, diantaranya adalah perusahaan, perorangan, maupun bagi industri. Data yang akurat, bila diproses akan menghasilkan informasi yang juga akurat. Informasi akurat sangat berguna untuk membuat keputusan. Informasi yang baik memiliki karakteristik seperti pada tabel 2.2, diantaranya:

Tabel 2.2 Karakteristik Informasi Yang Baik

Karakteristik	Keterangan
Akurat	Menggambarkan kondisi objek yang sesungguhnya.
Tepat waktu	Informasi harus tersedia sebelum keputusan dibuat.
Relevan	Berhubungan dengan keputusan yang akan diambil.
Terpercaya	Isi informasi dapat dipercaya (*reliable*).

Terverifikasi	Dapat dilacak ke sumber aslinya (*verifiable*).

2.3.1 Manfaat Sistem Informasi Bagi Perusahaan

Sistem informasi diperlukan oleh perusahaan untuk mengolah data menjadi informasi, sehingga berbagai pihak yang membuat keputusan, dapat menggunakan informasi tersebut untuk membuat keputusan yang baik. Informasi yang baik hanya dapat dihasilkan oleh sistem informasi yang baik. Sistem informasi yang baik adalah sistem informasi yang dengan sengaja dirancang oleh perusahaan untuk mengolah data menjadi informasi. Keputusan yang baik adalah keputusan yang mendatangkan hasil sesuai dengan yang diharapkan. Hasil yang baik akan mendatangkan keuntungan terus-menerus bagi pembuatan keputusan. Apabila perusahaan selalu mendapatkan hasil yang baik perusahaan dapat memiliki keunggulan bersaing (*competitive advantage*), sehingga dapat memenangkan persaingan. Dengan demikian, sistem informasi diharapkan dapat mendatangkan *competitive advantage* bagi perusahaan atau organisasi.

2.3.2 Manfaat Sistem Informasi Bagi Perorangan

Perorangan yang terlibat dengan sistem informasi perusahaan di antaranya adalah para manajer, para operator, dan para pelanggan. Masing-masing keputusannya memerlukan informasi yang akurat. Di bawah ini adalah jenis keputusan dan informasi yang diperlukan, di antaranya:

Tabel 2.3 Jenis Keputusan dan Informasi Yang Diperlukan

Pembuat Keputusan	Jenis Keputusan	Informasi yang Diperlukan
Manajemen	Menentukan harga jual produk.	Biaya produksi.
	Menaikkan gaji karyawan.	Biaya operasi perusahaan.
Operator	Membeli barang yang akan dijual.	Barang yang ada digudang, harga.

	Mengatur jadwal karyawan.	Jumlah karyawan dan tugasnya.
Pembeli	Membeli barang dari perusahan.	Harga dan jenis barang.
	Membeli tunai atau kredit.	Syarat-syarat kredit dan pembayaran.

2.3.3 Manfaat Sistem Informasi Bagi Industri

Manfaat sistem informasi tidak hanya akan diperoleh oleh perusahaan yang sudah menerapkan sistem informasi tersebut. Ada beberapa pihak yang terkait dengan kegiatan perusahaan, sehingga apabila perusahaan menyediakan sistem informasi yang baik, maka berbagai pihak tersebut juga dapat menikmati manfaatnya. Apabila sistem informasi antarperusahaan dapat saling terhubung, maka industri akan dapat mengetahui omset perdagangan semua perusahaan. Misalnya industri tekstil di suatu daerah dapat mengetahui produksi total tekstil di daerah tersebut ada berapa banyak, dibandingkan dengan total penjualan di daerah tersebut, sehingga dapat diketahui ada kekurangan berapa banyak.

Tabel 2.4 Perkembangan Fungsi Sistem Informasi

Periode	Fungsi Utama Sistem Informasi
1950an	Sistem informasi sebagai pencatat data dan penyajian informasi akurat. Sistem ini masih tergolong pada sistem pengolahan transaksi (*transaction processing systems*).
1960-1970an	Sistem informasi mulai menyajikan berbagai informasi yang diperlukan oleh manajemen untuk membuat keputusan-keputusan operasional seperti pembelian barang dan supplier, pembukaan kontor cabang, dan perluasan produk. Sistem ini berkembang menjadi sistem informasi manajemen (*management information systems*).

1970-1980an	Sistem informasi mulai berfungsi mendukung dan membantu manajemen untuk membuat keputusan, sehingga para manajer yang belum berpengalaman pun tetap dapat membuat keputusan yang memerlukan informasi yang rumit. Sistem informasi ini sudah tergolong pada sistem pendukung keputusan (*decision support systems*) atau sistem pakar (*expert systems*).
1980-Sekarang	Sistem informasi mulai dapat berkomunikasi dengan sistem lain, sehingga mendatangkan sinergi yang luar biasa. Sistem informasi tidak hanya digunakan oleh satu unit usaha, tetapi berbagai perusahaan saling berbagi atas sistem informasinya.

2.4 Perencanaan

Suatu perangkat lunak atau sistem informasi tidak dapat begitu saja dibuat. Karena pembuatan dan pengembangan perangkat lunak atau sistem informasi sering sekali merupakan upaya yang membutuhkan dana yang cukup besar serta yang relatif lama maka biasanya ia merupakan keputusan strategis yang harus dilakukan manajer-manajer peringkat atas dalam perusahaan. Tahap perencanaan ini bisa dilanjutkan dengan spesifikasi kebutuhan (SRS-*System or Software Requirement Spesification*) yang memuat pernyataaan-pernyataan tentang bagaimana serta untuk tujuan apa sistem atau perangkat lunak dikembangkan sesuai dengan visi, misi, serta strategi organisasi. SRS ini adalah pernyataan global dimana untuk rinciannya, dalam konteks menangkap kebutuhan serta harapan pengguna (*user's needs and expectations*). Masih harus diperhalus dengan pembuatan-pembuatan diagram-diagram.

2.4.1 Tahap Pengembangan Sistem Informasi

Pengembangan sistem informasi (*system development*) dapat berarti menyusun sistem informasi yang benar-benar baru atau yang lebih sering terjadi menyempurnakan sistem yang ada. Juga sering terjadi pengembangan sistem informasi berbasis komputer dilakukan dengan motivasi untuk memanfaatkan

komputer sebagai alat bantu yang dikenal sebagai alat yang cepat, akurat, tidak cepat lelah, serta tidak mengenal arti kata bosan, untuk melaksanakan instruksi-instruksi.

Secara umum suatu sistem informasi perlu diganti atau perlu dikembangkan adalah karena alasan-alasan sebagai berikut (Jogiyanto, 1995):

- **Adanya permasalahan yang dijumpai pada sistem yang lama.**
  Permasalahan pada sistem yang lama bisa berarti pencatatan data yang tidak akurat, informasi yang sering terlambat atau sukar diperoleh saat dibutuhkan, ketidakefisienan operasi, ketidak-amanan data-data penting yang mengakibatkan permasalahan akses data oleh oknum yang tidak berhak.
- **Pertumbuhan organisasi.** Pada saat organisasi masih kecil masih mungkin segalanya dilakukan secara manual dengan jumlah pengelola beberapa orang saja. Namun, saat organisasi berkembang menjadi besar, tidaklah mungkin untuk melakukan segalanya secara manual. Saat inilah diperlukan otomatisasi pemrosesan data sehingga proses-proses dalam organisasi bisa berjalan dengan cepat serta akurat. Selain itu, juga diperlukan suatu cara tertentu sehingga data-data yang diperlukan sebagai dasar pengambilan keputusan oleh manajer dapat diperoleh dengan cepat.
- **Untuk meraih kesempatan-kesempatan.** Teknologi informasi telah berkembang dengan cepatnya. Organisasi mulai merasakan bahwa teknologi informasi perlu digunakan untuk meningkatkan penyediaan informasi sehingga mendukung penuh dalam proses pengambilan keputusan yang akan dilakukan oleh para manajer. Dalam keadaan pasar bersaing, kecepatan serta ketepatan informasi sangat menentukan berhasil atau tidaknya strategi serta rencana-rencana yang telah disusun untuk meraih kesempatan-kesempatan yang ada. Bila pesaing organisasi kita berhasil memanfaatkan kesempatan-kesempatan itu, kita akan tertinggal sehingga mungkin akan menjadi terlambat untuk dapat memanfaatkan kesempatan ini.

- **Menyesuaikan diri dengan visi, misi, strategi organisasi yang baru.** Dalam perjalanannya, setiap organisasi memiliki visi, misi, serta strategi yang berubah sepanjang perjalanan waktu. Perubahan-perubahan ini biasanya terjadi akibat perubahan lingkungan yang mempengaruhi kerja serta kinerja organisasi. Teknologi informasi sering digunakan untuk mengantisipasi perubahan lingkungan dalam kerangka visi, misi, serta strategi yang diterapkan untuk mengatasi perubahan lingkungan tersebut.

Lebih jauh pengembangan sistem atau perangkat lunak itu sendiri pada dasarnya adalah proses perubahan, penghalusan (*refinement*), tranformasi, atau tambahan produk yang sudah ada. Dalam proses ini, adalah mungkin untuk mengganti suatu subproses dengan subproses yang lain, sepanjang subproses yang baru memiliki antarmuka yang sama dengan subproses yang lama. Sebagai contoh, pendekatan berorientasi objek menyediakan himpunan aturan untuk mendeskripsikan pewarisan dan spesialisasi dengan cara yang konsisten saat suatu subproses melakukan perubahan terhadap perilaku proses asalnya. Proses dapat dibagi ke tahapan yang kecil, yaitu subproses-subproses. Subproses-subproses harus didefinisikan sedemikian rupa sehingga mereka bersifat mandiri dari subproses-subproses yang lain. Dalam hal ini, setiap subproses harus memiliki kriteria-kriteria sebagai berikut:

1. Memiliki deskripsi lengkap tentang bagaimana ia bekerja.
2. Memiliki spesifikasi masukan (*input*) yang dibutuhkan.
3. Memiliki spesifikasi luaran (*output*) yang akan dihasilkan.

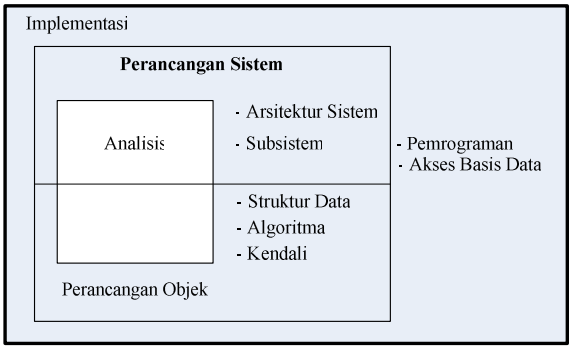
Proses pengembangan perangkat lunak juga dapat dibagi menjadi subproses-subproses yang kecil dan saling berinteraksi. Secara umum, proses pengembangan perangkat lunak dapat dilihat sebagai urutan tranformasi, dimana luaran (*output*) dari suatu transformasi berikut:

- Transformasi 1 (Analisis) menterjemahkan kebutuhan pengguna ke dalam spesifikasi kebutuhan sistem (SRS-*System Requirement Spesification*). Spesifikasi kebutuhan sistem ini bersifat menangkap semua yang

dibutuhkan sistem dan dapat terus diperbaharui secara iteratif selama berjalannya proses pengembangan sistem

- Transformasi 2 (Perancangan) dimulai dengan pernyataan masalah dan diakhiri dengan rincian perancangan yang dapat ditransformasikan ke sistem operasional. Transformasi ini mencakup seluruh aktifitas pengembangan perangkat lunak, mencakup definisi-definisi bagaimana mengembangkan perangkat lunak serta pengujiannya.
- Transformasi 3 (Implementasi) melakukan penghalusan rincian perancangan ke penyebaran sistem yang sesuai dengan kebutuhan pengguna. Transformasi ini juga mencakup perancangan peralatan yang digunakan, prosedur-prosedur pengoperasian, deskripsi orang-orang yang akan menggunakan sistem atau perangkat lunak. Tahapan ini juga merepresentasikan penginstalan perangkat lunak dalam lingkungan operasional. Dalam hal ini, juga dilakukan penyesuaian-penyesuaian dengan sistem operasional yang sudah atau akan berjalan, perancangan form-form yang digunakan, laporan-laporan yang akan dicetak.

Untuk memodelkan aplikasi dunia nyata dengan metodologi berorientasi objek, kita perlu memodelkan data maupun proses yang berjalan pada objek yang bersangkutan. Dengan mengijinkan kita menangkap mereka secara bersamaan dalam representasi yang umum, dan menawarkan keuntungan seperti pewarisan (*inheritance*) dan pengguna ulang kode (*code reuse*), pendekatan berorientasi objek menyediakan sarana yang berdayaguna untuk mengembangkan sistem yang lebih kompleks daripada pendekatan dengan diagram-diagram pada pendekatan struktur.



Gambar 2.4 Langkah-langkah Dari Siklus Pengembangan Sistem Berorientasi Objek

Pada tahap perancangan objek, kita mendefinisikan bagaimana analisis berorientasi aplikasi akan direalisasikan pada lingkungan implementasi. Jacobson (1992) (dikutip dari buku *Modern Database Management*, ditulis oleh Fred McFadden) menyebut 3 alasan utama mengapa menggunakan pendekatan perancangan berorientasi objek. Alasan-alasannya adalah:

1. Model analisis pada analisis dengan metoda terstruktur tidak cukup formal untuk diimplementasikan secara langsung ke bahasa pemrograman. Untuk menterjemahkannya ke kode komputer, kita perlu melakukan langkah-langkah penghalusan (*refinement*) dengan membuat keputusan-keputusan pada operasi apa yang objek akan lakukan, bagaimana seharusnya komunikasi antara objek-objek dalam suatu sistem, pesan apa yang harus dilewatkan.
2. Sistem nyata harus diadaptasi ke lingkungan dimana sistem kelak akan diimplementasi. Dalam hal ini, perlu dilakukan modifikasi-modifikasi model analisis ke beberapa faktor yang berbeda seperti kebutuhan kinerja, perangkat keras target dan perangkat lunak sistem.
3. Hasil dari analisis dapat divalidasi menggunakan perancangan berorientasi objek. Pada tahap ini, kita dapat memverifikasi apakah hasil dari analisis sesuai untuk membangun sistem dan kemudian, jika tidak sesuai, kita kembali secara iteratif ke tahap analisis, serta membuat perubahan yang perlu pada model analisis.

2.4.2 Pengembangan Sistem Berorientasi Objek

Siklus hidup pengembangan sistem atau perangkat lunak berorientasi objek mengandung 3 proses makro: analisis berorientasi objek, perancangan berorientasi objek, dan implementasi berorientasi objek.

Model *use case* dapat dipergunakan sebagai pemandu pada hampir semua aktivitas pengembangan perangkat lunak. Lebih jauh, seseorang dapat menghasilkan perancangan-perancangan yang dapat dilacak sepanjang spesifikasi kebutuhan, analisis, perancangan, implementasi, serta pengujian. Keunggulan utama pendekatan ini adalah bahwa semua keputusan perancangan dapat dilacak langsung ke spesifikasi kebutuhan pengguna. Pengembangan sistem berorientasi objek mencakup aktivitas-aktivitas dibawah ini.

- Analisis berorientasi objek yang dikendalikan oleh *use case*.
- Perancangan berorientasi objek.
- Pembuatan *prototype*.
- Pengembangan berbasis komponen.
- Pengujian berkelanjutan.

Pengembangan perangkat lunak berorientasi objek memungkinkan kita melihat masalah sebagai sistem dengan objek-objek yang saling bekerja sama. Lebih jauh, hal ini memungkinkan pengembangan berkelanjutan.

2.4.2.1 Analisis Berorientasi Objek

Analisis berorientasi objek (OOA-*Object Oriented Analysis*) adalah tahapan perangkat lunak dengan menentukan spesifikasi sistem (disebut juga dengan SRS-System/ *Software Requirement Spesification*) dan mengidentifikasi kelas-kelas serta hubungannya satu terhadap yang lain. Untuk memahami spesifikasi sistem kita perlu mengidentifikasi para pengguna atau yang sering disebut aktor-aktor. Siapa aktor-aktor yang akan menggunakan sistem dan bagaimana mereka menggunakan sistem? Pada paradigma berorientasi objek, seperti juga pada pengembangan tradisional, skenario-skenario digunakan untuk membantu analis

memahami kebutuhan. Dalam hal ini, skenario-skenario mungkin dibuat secara informal atau tidak terdokumentasi secara penuh. Skenario-skenario adalah cara yang paling baik untuk menguji siapa yang berbuat apa pada interaksi antar objek dan peran apa yang mereka mainkan. Pada dasarnya, *use case* adalah interaksi khusus antara pengguna dan sistem untuk menangkap sasaran serta kebutuhan pengguna.

2.4.2.2 Perancangan Berorientasi Objek

Sasaran dari perancangan berorientasi objek (OOD-*Object Oriented Design*) adalah merancang kelas-kelas yang teridentifikasi selama tahap analisis dan antarmuka pengguna (*user interface*). Selama tahap ini, kita mengidentifikasi dan mungkin menambahkan beberapa objek dan kelas yang mendukung implementasi dari spesifikasi kebutuhan. Perancangan berorientasi objek dan analisis berorientasi objek adalah topik yang terpisah namun keduanya saling bekerja sama dengan erat. Pengembangan sistem dengan metodologi berorientasi objek pada umumnya bersifat iteratif, dengan kata lain kita dapat mulai dengan analisis berorientasi objek, melakukan pemodelan, melakukan perancangan berorientasi objek, melakukan beberapa tahapan secara iteratif sambil melakukan penghalusan objek-objek dan melengkapi model sistem. Aktivitas dan fokus dari analisis berorientasi objek dan perancangan berorientasi saling bekerja sama. Adapun langkah-langkah yang umum dilakukan dalam perancangan berorientasi objek adalah sebagai berikut:

- Lakukan perancangan kelas-kelas dan lakukan penghalusan-penghalusan (refinement).
- Lakukan perancangan atribut-atribut dan lakukan penghalusan-penghalusan.
- Lakukan perancangan metoda-metoda dan lakukan penghalusan-penghalusan.
- Lakukan perancangan struktur-struktur dan lakukan penghalusan-penghalusan.

- Lakukan perancangan asosiasi-asosiasi dan lakukan penghalusan-penghalusan.

2.4.2.3 Implementasi Berorientasi Objek

Saat ini, komponen-komponen dikembangkan dan diuji menggunakan sejumlah teknologi yang berbeda. Sebagai contoh, kakas-kakas CASE (*Computer Aided Software Engineering*) memungkinkan penggunanya untuk secara tepat mengembangkan sistem informasi. Sasaran utama CASE adalah melakukan otomatisasi sejumlah terbatas siklus pengembangan sistem informasi menggunakan sejumlah kakas perangkat lunak, seperti pemodelan, metodologi, dan otomatisasi pembentukan kode. Dalam hal ini, kode yang dihasilkan CASE pada umumnya merupakan kerangka dasar pemrograman yang kemudian dapat diperhalus lagi oleh sentuhan tangan pemrogram. Generasi terbaru dari CASE memungkinkan pengembangan sistem berbasis komponen.

Pengembangan sistem berbasis komponen (CBD-*Component Based Development*) adalah pendekatan industri pada proses pengembangan perangkat lunak. Pengembangan perangkat lunak berkembang dari teknik mengerjakan segala sesuatu dari awal ke perakitan komponen-komponen yang dapat digunakan ulang, yang telah dikembangkan dan diuji sebelumnya. Ada 2 ide dasar yang mendasari pengembangan berbasis komponen. Pertama, pengembangan aplikasi dapat ditingkatkan aplikasi dapat ditingkatkan kecepatan secara signifikan dengan menggunakan komponen-komponen yang dapat digunakan ulang. Kedua, pengembangan perangkat lunak yang dilakukan saat ini mungkin akan menghasilkan komponen-komponen yang dapat digunakan ulang pada proyek-proyek yang akan datang.

2.5 Object Oriented Programming (OOP)

Object Oriented Programming (OOP) atau pemrograman berorientasi objek adalah suatu cara baru dalam berpikir serta berlogika dalam menghadapi masalah-masalah yang akan dicoba-atasi dengan bantuan komputer. OOP tidak seperti pendahulunya (pemrograman struktur), mencoba melihat permasalahan lewat pengamatan dunia nyata dimana setiap objek adalah entitas tunggal yang memiliki kombinasi struktur data dan fungsi tertentu.

Pada perkembangannya, filosofi OOP menciptakan sinergi yang luar biasa sepanjang siklus pengembangan perangkat lunak (perencanaan, analisis, perancangan, implementasi, serta pengujian) sehingga dapat diterapkan pada perancangan sistem secara umum menyangkut perangkat lunak, perangkat keras, serta informasi secara keseluruhan. Apa yang disebut sebagai objek? Objek adalah konsep atau abstraksi tentang sesuatu yang memiliki arti bagi aplikasi yang akan kita kembangkan. Menurut Khoshafian (1995) dikutif dari buku *Object Oriented Modelling And Design* tulisan James Rumbaugh definisinya adalah sebagai berikut: "*identitas adalah sesuatu yang membedakan suatu objek dari objek yang lainnya*". Setiap objek adalah nyata dan dapat dibedakan satu dari yang lainnya. Sekalipun 2 objek bernama sama, namun ia tetap dapat dipisahkan.

Dalam kontek *object oriented* kita juga mengenal konsep lain yang dinamakan *Polimorfisme* yaitu konsep yang menyatakan bahwa suatu fungsi yang sama dapat diterapkan dan dapat dimiliki oleh kelas-kelas yang berlainan. Kita juga akan menemukan konsep pewarisan (*Inheritance*), pewarisan ini pada dasarnya adalah berbagi atribut dan operasi antarkelas berdasarkan hierarki kelas. Dalam konteks pemrograman berorientasi objek, pewarisan akan sangat mengurangi perulangan penulisan kode dan mungkin merupakan salah satu keunggulan utama dalam rekayasa perangkat lunak karena memungkinan penulisan kode yang lebih sedikit tanpa mengorbankan esensi perangkat lunak secara keseluruhan. Selain itu, pewarisan juga memungkinkan untuk menciptakan komponen-komponen yang dapat digunakan ulang (*Reusable Component*).

2.5.1 Object Oriented Development (OOD)

Object Oriented Development adalah suatu cara pengembangan perangkat lunak dan sistem informasi berdasarkan abstraksi objek-objek yang ada di dunia nyata. Brooks (1987) dikutif dari buku *Object Oriented Modelling And Design* tulisan James Rumbaugh mengemukakan bahwa bagian tersulit dari pengembangan perangkat lunak dan sistem informasi adalah tahap analisis dimana kita harus menganalisis masalah yang sangat rumit, yang kita jumpai di dunia nyata serta melakukan abstraksi terhadap masalah itu, kemudian melakukan perancangan agar kelak dapat diimplementasikan dengan cepat serta akurat pada komputer (baik secara perangkat keras maupun perangkat lunak) menjadi suatu sistem atau perangkat lunak yang sesuai dengan kebutuhan serta harapan pengguna.

2.5.2 Tema-tema Object Oriented

Ada beberapa tema yang mendasari teknologi berorientasi objek yaitu:

- Kelas

  Kelas membungkus (*Encapsulating*) objek-objek. Suatu kelas tunggal dapat digunakan untuk menciptakan sejumlah objek-objek. Selain itu, suatu kelas juga dapat digunakan untuk menciptakan kelas-kelas lain yang mewarisi (*Inheritance*) sebagian atau seluruh data serta fungsi yang dimiliki oleh kelas yang disebutkan terdahulu. Pada awal perkembangannya, kelas didefinisikan sebagai tipe abstrak (ADT-*Abstract Data Type*) dalam bahasa pemrograman Simula67 dan *Smalltalk*. Dewasa ini teknologi berorientasi objek memperluas ide tentang ADT menjadi konsep kelas atau objek dan menambahkan banyak inovasi yang penting yang memungkinkan kita lebih mampu mengelola kompleksitas permasalahan. Keuntungan pembungkus (*Encapsulation*) adalah memungkinkan seseorang menggunakan (memanfaatkan fungsi-fungsi serta layanan) suatu komponen yang diciptakan dengan konsep pembungkusan tanpa mengetahui rincian implementasinya.

- Abstraksi

  Abstraksi pada dasarnya adalah menemukan hal-hal yang esensial pada suatu objek dan mengabaikan hal-hal yang bersifat insidental. Maksud dari pernyataan ini diantaranya adalah menangkap sesuatu yang berarti untuk dituangkan dalam sistem atau perangkat lunak alih-alih menangkap seluruh fakta yang ada. Pada pengembangan sistem informasi, ini berarti memfokuskan pada apa itu objek, apa ciri-ciri yang dimiliki objek itu (atribut-atributnya) serta apa yang objek lakukan (operasi-operasi apa yang dapat dilakukan oleh suatu objek) sebelum memutuskan bagaimana ia dapat diimplementasikan.

- Pembungkus (*Encapsulation*) dan Pengiriman Pesan (*Message Passing*)

  Pembungkusan (penyembunyian informasi) berarti meninggalkan aspek eksternal dari objek yang dapat dimasuk (diakses) oleh objek lain dan memfokuskan diri pada implementasiinernal suatu objek. Rincian implementasi internal dari suatu objek tersembunyi dari objek-objek lain dan terpisah dari implementasi eksternal, yaitu antarmuka (*interface*) satu objek dengan objek lainnya. Sebab itu, implementasi internal suatu objek dapat dirubah tanpa mempengaruhi aplikasi yang menggunakannya asalkan perubahan internal itu tidak merubah antarmuka (*interface*) yang digunakan objek yang bersangkutan untuk berkomunikasi dengan objek yang lainnya. Keuntungan dari pembungkusan adalah kita dapat mengharapkan suatu objek melakukan metoda apa yang kita inginkan tanpa harus tahu bagaimana objek itu melakukannya.

- Generalisasi dan Polimorfisme

  Teknologi berorientasi menambah dua mekanisme penting pada tipe data abstrak (ADT/*Abstract Data Type*) yang sepintas telah dijelaskan di atas yaitu: generalisasi atau spesialisasi serta polimorfisme.

  Generalisasi/ spesialisasi memungkinkan kelas-kelas berbagi data serta perilaku yang sama. Pada konteks pemrograman, ini memungkinkan pengurangan ukuran kode dan menyediakan kemungkinan pengembangan sistem atau perangkat lunak yang lebih mudah dipelihara. Generalisasi

juga meningkatkan keterpaduan dalam suatu kelas dan mengurangi derajat kesaling-bergantungan kelas saat implementasi. Polimorfisme mengijinkan penyesuaian berbagi kode untuk memenuhi keadaan tertentu. Dua mekanisme ini saling bekerja sama untuk mendukung kemandirian objek individual serta mendukung proses pengembangan sistem atau perangkat lunak berkelanjutan dimana komponen-komponen yang dibuat pada suatu aplikasi dapat digunakan aplikasi lain yang memerlukan layanan (*service*) yang sama.

Selain itu, dua mekanisme diatas sangat membantu untuk pengorganisasian kelas/ objek dan sangat membantu dalam konteks pewarisan (*inheritance*) yang mungkin penggunaan ulang komponen (*reusable components*). Pewarisan ini adalah konsep yang sangat penting dalam teknologi berorientasi objek karena lebih menghemat biaya dan waktu jika kita bisa menggunakan komponen yang sudah ada untuk sebuah pengembangan sistem/ perangkat lunak. Selain itu fleksibilitas komponen-komponen pada terminologi berorientasi objek lebih memungkinkan pengembang untuk menangkap semua kebutuhan serta harapan pengguna (*user's needs and expectations*).

- Penggabungan Data (Atribut) dan Perilaku (Fungsi)

  Pada sistem atau perangkat lunak yang dikembangkan dengan paradigma berorientasi objek, pemanggil suatu operasi atau tidak perlu tahu bagaimana implementasi internal suatu operasi. Program berorientasi objek secara sederana memanggil operasi draw untuk setiap penggambaran. Keputusan akan dilakukan oleh suatu objek untuk menentukan prosedur mana yang seharusnya digunakan. Berdasarkan parameter-parameternya masing-masing. Pada sistem berorientasi objek, hierarki struktur data dan fungsi aalah identik pada hierarki pewarisan operasi. Artinya, struktur data serta operasi-operasi pada setiap kelas akan diwariskan pada objek-objek dari kelas mana objek itu diciptakan serta diwariskan pula pada kelas-kelas yang secara hierarkis berada dibawah kelas induk.

- Sharing

  Teknik pengembangan sistem informasi berorientasi objek mengijinkan *sharing* (berbagi data atau informasi sertaberbagi penggunaan suatu operasi) pada beberapa tingkatan. Pewarisan terstruktur data (atribut) dan perilaku memungkinkan struktur umum untuk dibagikan diantara beberapa subkelas tanpa pengulangan.

  Dalam konteks pemrograman berorientasi objek, berbagi data dan perilaku memiliki beberapa tingkatan seperti yang tertulis dibawah ini:

  a. Metoda serta Peubah bersifat *Private*

     Metoda serta data yang dikendalikan oleh suatu kelas tertentu dan tidak dapat diakses oleh objek-objek yang berbeda kelas dinamakan *private*. Ciri metoda *sharing* bersifat *private* adalah: hanya fungsi-fungsi atau operasi-operasi dalam kelas atau objek yang bersangkutan yang dapat mengakses metoda peubah serta atribut dalam kelas atau objek tanpa batasan apapun.

  b. Metoda serta Peubah bersifat *Protected*

     Metoda atau operasi serta peubah atau atribut yang bersifat *protected* hanya bisa diakses oleh kelas yang bersangkutan serta kelas-kelas yang merupakan turunannya mengikuti hierarki generalisasi atau spesialisasi.

  c. Metoda serta Peubah bersifat *Friend*

     Metoda dan peubah dalam suatu kelas yang dapat mengakses oleh kelas-kelas lain yang berhubungan dalam keadaan tertentu dinamakan *friend.*

  d. Metoda serta Peubah bersifat *Public*

     Metoda atau operasi serta peubah atau bersifat *public* dapat diakses oleh semua objek-objek dalam aplikasi tertentu tanpa batasan apapun.

- Penekanan Pada Struktur Objek, Bukan Pada Struktur Prosedur

  Teknologi berorientasi objek menekankan pada apa itu objek, bukan pada bagaimana objek digunakan. Penggunaan objek sangat bergantung pada

rincian aplikasi dan sering berubah sepanjang masa pengembangan. Pengembangan berorientasi objek sangat menekankan pada struktur data dan kurang menekankan pada struktur prosedur seperti yang terjadi pada metodologi dekomposisi fungsional tradisional.

- Sinergi

  Identitas, klasifikasi, polimorfisme, serta pewarisan adalah karakteristik utama dari bahasa pemrograman berorientasi objek. Setiap konsep dapat digunakan secara terpisah, tetapi mereka saling melengkapi secara sinergis. Penekanan utama pada komponen yang esensial memaksa pengembang perangkat lunak untuk berpikir hati-hati dan sangat mendalam tentang apa itu objek serta apa yang dilakukannya sehingga sistem yang dihasilkannya lebih mudah dipahami serta lebih umum.

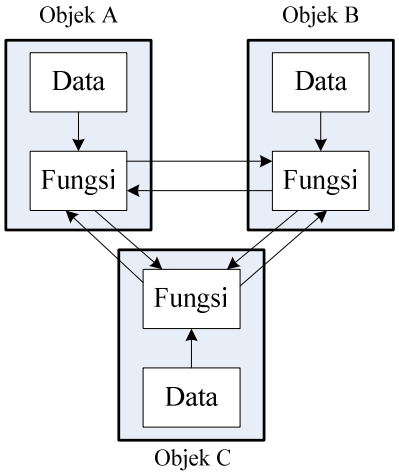
2.5.3 Paradigma Berorientasi Objek

Faktor utama dari ditemukannya pendekatan berorientasi objek adalah karena ditemukannya kekurangan-kekurangan pada pendekatan struktur: biaya pengembangan perangkat lunak berkembang sesuai dengan berkembangnya keinginan atau kebutuhan pengguna, pemeliharaan yang sukar, lamanya penyelesaian suatu proyek, jangka waktu penyelesaian proyek yang hampir selalu terlambat, biaya pengembangan perangkat lunak yang sangat tinggi. Pendekatan berorientasi objek membuat data terbungkus pada setiap fungsi atau prosedur dan melindunginya terhadap perubahan tidak dikendaki dari fungsi yang berada diluar. Beberapa karakteristik yang menjadi ciri-ciri pendekatan objek adalah:

- Pendekatan lebih pada data dan bukannya pada prosedur atau fungsi.
- Program besar dibagi pada apa yang dinamakan objek-objek.
- Strukur data dirancang dan menjadi karakteristik dari objek-objek.
- Fungsi-fungsi yang mengoperasikan data tergabung dalam suatu objek yang sama.
- Data tersembunyi dan terlindung dari fungsi atau prosedur yang ada di luar.

- Objek-objek dapat saling berkomunikasi dengan saling mengirim *message* (pesan) satu sama lain.
- Pendekatan adalah dari bawah ke atas (*bottom up approach*).

Dibawah ini adalah penggambaran dari pendekatan berorientasi objek.



Gambar 2.5 Pengorganisasian Data Serta Fungsi Pada Pendekatan Berorientasi Objek

2.6 UML (*Unified Modeling Language*)

Pendekatan terstruktur memiliki *tool-tools* perancangan yang dikenal secara luas serta menjadi standar umum, seperti DFD (*Data Flow Diagram*), ERD (*Entity Relationship Diagram*), Bagan Terstruktur (*Structured Chart*), Diagram Alir (*Flow Chart*). Karena metodologi berorientasi objek adalah konsep yang relatif baru, ia belum memliki standar *tools* untuk analisis dan perancangan. Walaupun pun begitu, saat ini UML (*Unified Modeling Language*) berangsur-angsur mulai menjadi standar metodologi pengembangan sistem informasi dan perangkat lunak berorientasi objek.

2.6.1 Sejarah Perkembangan UML (*Unified Modeling Language*)

UML (*Unified Modeling Language*) pertama kali diperkenalkan pada tahun 1990 ketika Grady Booch dan Ivar Jacobson dan James Rumbaugh mulai mengadopsi ide-ide serta kemampuan tambahan dari masing-masing metodanya yaitu Ivar Jacobson (yang sebelumnya terkenal dengan konsep OOSE-*Object Oriented Software Engineering*), James Rumbaugh (yang sebelumnya terkenal dengan

konsep OMT-*Object Modeling Technique*), serta Graddy Booch (yang sebelumnya terkenal dengan notasi Booch yang popular digunakan sebagai salah satu metodologi analisis dan perancangan berorientasi objek yang kerap sekali digunakan) dari kemampuan tersebut mereka berusaha membuat metodologi terpadu yang kemudian dinamakan UML (*Unified Modeling Language*). UML merupakan metoda pengembangan perangkat lunak atau sistem informasi dengan metoda grafis yang relatif mudah dipahami.

Usaha pengembangan UML dimulai pada oktober 1994, ketika Rumbaugh bergabung dengan Booch di *Rational Software Corporation*. Proyek pertama mereka aalah menggabungkan metoda Booch dan OMT (*Object Modeling Technique*). Versi 0.8 UML dirilis pada bulan oktober 1995. Saat peluncuran UML versi 0.8 itu, Jacobson juga bergabung dengan *Rational* sehingga UML diperluas untuk mengadopsi OOSE. Hasilnya adalah versi 0.9 dari UML yang dirilis pada juni 1996.

2.6.2 Fokus UML

Secara umum UML merupakan bahasa untuk visualisasi, spesifikasi, kontruksi, serta dokumentasi. Dalam kerangka visualisasi, para pengembang menggunakan UML sebagai suatu cara untuk mengkomunikasikan idenya kepada para pemrogram serta calon pengguna sistem atau perangkat lunak. Secara khusus, UML menspesifikasi langkah-langkah penting dalam pengambilan keputusan analisis, perancangan, serta implementasi dalam sistem yang sangat bernuansa perangkat lunak (*software intensive system*). Dalam hal ini, UML bukanlah merupakan bahasa pemrograman tetapi model-model yang tercipta berhubungan langsung dengan berbagai macam bahasa pemrograman, sehingga adalah mungkin melakukan pemetaan (*mapping*) langsung dari model-model yang dibuat dengan UML ke bahasa pemrograman berorientasi objek. Pemetaan ini bersifat 2 arah: generasi kode bahasa pemrograman tertentu dari UML (*forward engineering*). Sebaliknya juga mungkin saat generasi kode belum sesuai dengan kebutuhan dan harapan pengguna, pengembang dapat melakukan langkah balik (bersifat iteratif)

dari implementasi ke UML hingga didapatkan sistem atau perangkat lunak yang sesuai dengan harapan pengguna dan pengembang. Proses yang terakhir ini dinamakan *reverse engineering*.

Selain itu UML (*Unified Modeling Language*) memiliki 3 ciri khas itu adalah:

1. Merupakan *Use-Case Driven Language*. Apa maksudnya? Sasaran dari pemodelan adalah memandu para pengembang dalam menciptakan perangkat lunak dan sistem informasi yang memenuhi serta kebutuhan calon pengguna secara efisien.
2. Merupakan *Architecture-Centric Language*. Apa maksudnya? Para pengembang dengan metoda UML mengatakan bahwa pengendalian kegiatan pengembangan sistem atau perangkat lunak dengan *use-case* diagram belumlah cukup memadai untuk menghasilkan sistem atau perangkat lunak yang berkualitas tinggi. Sesuatu yang lain dibutuhkan agar pengembang mendapatkan sistem yang benar-benar baik yaitu arsitektur dimana arsitektur merupakan pandangan umum yang disetujui oleh pengembang dan calon pengguna mengenai sistem atau perangkat lunak yang sedang dikembangkan. Tapi apa yang dimaksud dengan arsitektur perangkat lunak? UML mendeskripsikan bahwa yang dimaksud dengan arsitektur perangkat lunak atau sistem adalah:
    - Organisasi dari sistem atau perangkat lunak.
    - Elemen-elemen serta antarmukanya masing-masing yang tercakup dalam sistem atau perangkat lunak.
    - Secara umum, arsitektur sistem memberi perhatian khusus juga pada fungsionalitas, kinerja, kelenturan pada perangkat lunak, mudah dipahami untuk setiap perangkat lunak.
3. Merupakan *Iterative and Incremental Language*. Apa maksudnya? Pengembangan perangkat lunak atau sistem informasi tidaklah bersifat linier. Setiap langkah maju seharusnya dilakukan dengan menengok ke belakang dengan mempertanyakan apakah pengembang sudah melakukan hal yang benar, yang sesuai dengan *use-case* serta arsitektur yang telah

disepakati? Jika belum, pengembang bisa kembali ke langkah sebelumnya dengan melakukan perbaikan-perbaikan yang diperlukan.

2.6.3 Bangunan Dasar UML (*Unified Modeling Language*)

Untuk memahami UML, kita membutuhkan model konseptual dari bahasa ini. Metodologi UML menggunakan 3 bangunan dasar untuk mendeskripsikan sistem atau perangkat lunak yang akan dikembangkan, yaitu:

1. Sesuatu (*Thing*).
2. Relasi (*Relationship*).
3. Diagrams.

Setiap bangunan dasar dapat diterapkan sepanjang tahap pengembangan sistem. Ketiga jenis bangunan dapat digunakan secara saling melengkapi satu sama lainnya. Secara umum UML diterapkan dalam pengembangan sistem atau perangkat lunak berorientasi objek sebab metodologi UML ini mempunyai keunggulan-keunggulan dibawah ini (dibandingkan metodologi terstruktur yang sebelumnya ditemukan oleh Yourdon dan DeMarco):

1. ***Uniformity***. Dengan metodologi UML (atau metodologi berorientasi objek pada umumnya), para pengembang cukup menggunakan 1 metodologi dari tahap analisis hingga perancangan. Hal ini tidak bisa dilakukan dalam metodologi pengembangan terstruktur. Dengan perkembangan masa kini ke arah aplikasi GUI (*Graphical User Interface*), UML juga memungkinkan kita merancang komponen antarmuka pengguna (*User Interface*) secara terintegrasi bersama dengan perancangan perangkat lunak sekaligus dengan perancangan basis data.
2. ***Understandability***. Dengan metodologi ini kode yang dihasilkan dapat diorganisasi kedalam kelas-kelas yang berhubungan dengan masalah sesungguhnya sehingga mudah dipahami siapa pun juga.
3. ***Stability***. Kode program yang dihasilkan relatif stabil sepanjang waktu sebab sangat mendekati permasalahan sesungguhnya di lapangan.

4. ***Reusability***. Dengan metodologi berorientasi objek, dimungkinkan penggunaan ulang kode, sehingga pada gilirannya akan sangat mempercepat waktu pengembangan perangkat lunak atau sistem informasi.

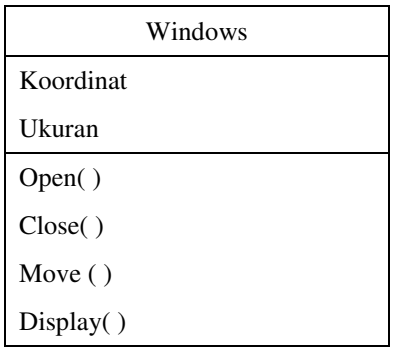
2.6.3.1 Sesuatu (*Things*)

Ada 4 macam '*things*' dalam UML, yaitu: *Structural Things*, *Behavior Things*, *Grouping Things*, dan *Annotational Things*.

A. *Structural Things*

Structural things merupakan bagian yang relatif statis dalam model UML. Bagian yang relatif statis dapat berupa elemen-elemen yang bersifat fisik maupun konseptual, keseluruhannya ada 7 macam *structural things*, yaitu:

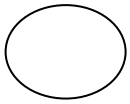
1. Kelas adalah himpunan dari objek-objek yang berbagi atribut serta operasi yang sama. Kelas mengimplementasikan satu atau lebih antarmuka (*interface*). Secara garis besar, kelas digambarkan dengan empat persegi panjang yang memuat nama, atribut, serta operasi yang dimilikinya.



Gambar 2.6 Kelas

2. Antarmuka (*interface*) adalah kumpulan dari operasi-operasi yang menspesifikasikan layanan (*service*) suatu kelas atau komponen atau objek. Lebih jauh antarmuka mendeskripsikan perilaku yang tampak dari luar dari suatu elemen. Antarmuka mendefiniskan himpunan spesifikasi operasi tapi tidak digunakan untuk menspesifikasi implementasi operasi. Antarmuka jarang berdiri sendiri. Secara grafis antarmuka digambarkan dengan lingkaran kecil dengan namanya yang didahului dengan garis

tegak (|). Biasanya, dilampirkan pada kelas atau komponen yang merealisasikan antarmuka.



| Pemeriksaan Ejaan

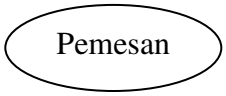
Gambar 2.7 Antarmuka (Interface)

3. Kolaborasi (*collaboration*) mendefinisikan interaksi aturan-aturan dan elemen lain yang bekerja sama untuk menyediakan perilaku yang lebih besar dari jumlah dari elemen-elemennya (sinergi). Lebih jauh, kolaborasi adalah terstruktur. Suatu kelas mungkin berpartisipasi pada beberapa kolaborasi. Secara grafis, kolaborasi digambarkan dengan elips bergaris putus-putus yang memuat nama kolaborasi itu.



Gambar 2.8 Kolaborasi

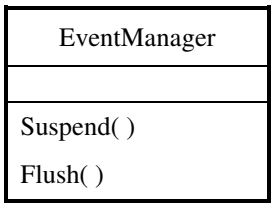
4. *Use Case* adalah deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu *actor*. *Use case* digunakan untuk menstrukturkan perilaku pada suatu model. Secara grafis, *use case* digambarkan dengan elips tegas yang berisi namanya.



Gambar 2.9 Use Case

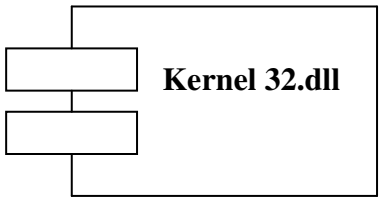
5. Kelas Aktif (*Active Class*) adalah kelas dimana objek-objek yang dimilikinya memiliki satu atau lebih dari proses dan lebih jauh menginisialisasi suatu aktifitas kendali. Kelas aktif sebenarnya adalah kelas biasa hanya objek-objek yang dimilikinya menampilkan elemen-elemen yang memiliki perilaku konkuren. Secara grafis, kelas aktif

digambarkan seperti kelas biasa tetapi dengan batas yang tebal, yang memuat nama, atribut, serta operasi yang dimilikinya.



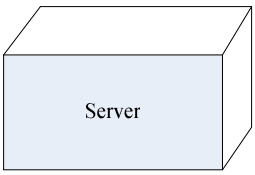
Gambar 2.10 Kelas Aktif

6. Komponen (*Component*) adalah bagian fisik dan bagian yang dapat digantikan pada suatu sistem. Komponen merupakan berkas-berkas *ActiveX*. Berkas-berkas COM (*Component Object Model*) yang keberadaannya dibutuhkan oleh sistem yang kita kembangkan, komponen ini merepresentasikan konsep-konsep *reusable component*. Secara grafis, komponen digambarkan dengan empat persegi panjang seperti kelas tetapi ditambahi *tab*. Selain itu, penggambaran komponen harus disertai namanya.



Gambar 2.11 Komponen

7. Simpul (*Node*) adalah elemen fisik yang eksis saat aplikasi dijalankan dan mencerminkan suatu sumberdaya komputasi. Kumpulan komponen mungkin hadir dalam simpul dan mungkin juga berpindah-pindah dari suatu simpul ke simpul yang lainnya. Secara grafis, simpul digambarkan sebagai kubus yang berisi namanya.



Gambar 2.12 Simpul (*Node*)

B. *Behavior Things*

Behavior things merupakan bagian yang dinamis pada model UML. Biasanya merupakan kata kerja dari model UML, yang mencerminkan perilaku sepanjang ruang dan waktu. Ada dua macam *behavior things* dalam UML, yaitu:

1. Interaksi, interaksi adalah suatu perilaku yang mencakup himpunan pesan-pesan (*message*) yang diperlukan untuk menyelesaikan suatu fungsi tertentu. Perilaku kumpulan objek-objek atau operasi individual bisa dispesifikasi dengan interaksi. Sebuah interaksi terdiri dari beberapa unsur, yaitu: pesan, urutan aksi (perilaku yang dihasilkan oleh sebuah pesan), serta *link* (hubungan antar objek-objek). Secara grafis, pesan digambarkan dengan tanda panah tegas yang sering memuat nama operasinya.



Gambar 2.13 Pesan (Message)

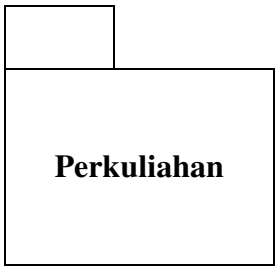
2. *State*, *state* adalah perilaku yang menspesifikasikan urutan-urutan kedudukan suatu objek atau interaksi-interaksi sepanjang waktu dalam menanggapi *event-event* yang terjadi. Perilaku suatu kelas mungkin dapat dilukiskan dengan baik dengan pertolongan diagram urutan *state* dan *event*. Penggambaran *state* memuat beberapa unsur, yaitu *state* itu sendiri, transisi (perubahan dari suatu *state* ke *state* yang lainnya, *event* (suatu keadaan yang memicu sebuah transisi), serta aktifitas (tanggapan terhadap transisi). Secara grafis, *state* digambarkan sebagai empat persegi panjang yang sudutnya melengkung, yang namanya (serta sub*state* didalamnya, jika ada).



Gambar 2.14 State

C. *Grouping Things*

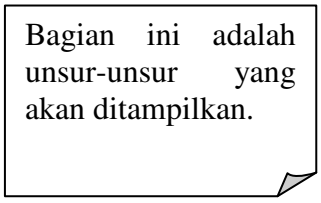
Merupakan bagian pengorganisasian dalam UML. Dalam penggambaran model UML yang rumit kadang diperlukan penggambaran paket yang menyederhanakan model. Paket-paket ini kemudian dapat didekomposisi lebih lanjut. Paket berguna bagi pengelompokkan sesuatu, misalnya model-model serta subsistem-subsistem.



Gambar 2.15 Paket

D. *Annotational Things*

Merupakan bagian yang memperjelas model UML. Dimana dapat berupa komentar-komentar yang menjelaskan fungsi-fungsi serta ciri-ciri tiap elemen dalam model UML.



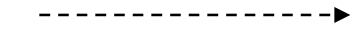
Gambar 2.16 Catatan

2.6.3.2 Relasi (*Relationship*)

Yang dimaksud *relationship* adalah hubungan-hubungan yang terjadi antar elemen dalam UML. Hubungan-hubungan ini penting sekali dalam UML. Dapat dikatakan, tidak mungkin membuat model-model UML tanpa *relationship* ini.

Ada 4 macam *relationship* dalam UML. Ke-empat macam *relationship* yang dapat digunakan untuk menggambarkan model-model UML yang representatif itu adalah:

1. *Dependency* (Kebergantungan). Yang dimaskud dengan dependency adalah hubungan dimana perubahan yang terjadi pada suatu elemen *independen* (mandiri) akan mempengaruhi elemen yang bergantung padanya (elemen yang tidak mandiri-*independen*). Secara grafis, *dependency* digambarkan dengan panah terputus-putus.



Gambar 2.17 Dependency

2. *Asosiasi*. Yang dimaksud dengan asosiasi adalah apa yang menghubungkan antara objek satu dengan objek yang lainnya, bagaimana hubungan suatu objek dengan objek lainnya. Suatu bentuk asosiasi adalah agregasi yang menampilkan hubungan suatu objek dengan bagian-bagiannya. Secara grafis, asosiasi digambarkan dengan garis tegas tanpa tanda panah.



Gambar 2.18 Asosiasi

3. *Generalisasi*. Yang dimaksud dengan generalisasi adalah hubungan dimana objek anak (*descendent*) berbagi perilaku dan struktur data dari objek yang ada di atasnya (objek induk-*ancestor*). Arah dari atas ke bawah (dari objek induk ke objek anak) dinamakan *spesialisasi*, sedangkan arah berlawanan sebaliknya (bawah ke atas) dinamakan *generalisasi*. Secara grafis, generalisasi digambarkan sebagai garis yang ujungnya berkepala panah (atau bentuk segitiga) yang kosong, yang mengarah ke objek induk.



Gambar 2.19 Generalisasi

4. *Realisasi*. Yang dimaksud dengan realisasi adalah operasi yang benar-benar dilakukan oleh suatu objek. Secara grafis, realisasi digambarkan dengan tanda panah bergaris putus-putus dengan kepala panah kosong.



Gambar 2.20 Realisasi

2.6.3.3 Diagram

Setiap sistem yang kompleks seharusnya bisa dipandang dari sudut yang berbeda-beda sehingga kita bisa mendapatkan pemahaman secara menyeluruh. Untuk upaya tersebut UML menyediakan 9 jenis diagram yang dapat dikelompokkan berdasarkan sifatnya yaitu statis dan dinamis. Ke 9 jenis diagram dalam UML yaitu:

1. **Diagram Kelas**. Bersifat statis. Diagram ini memperlihatkan himpunan-himpunan kelas-kelas, antarmuka-antarmuka, kolaborasi, serta relasi-relasi. Diagram ini umum dijumpai pada pemodelan sistem berorientasi objek. Meskipun bersifat statis sering pula diagram kelas memuat kelas-kelas aktif.
2. **Diagram Objek**. Bersifat statis. Diagram ini memperlihatkan objek-objek serta relasi-relasi antarobjek. Diagram objek memperhatikan instansiasi statis dari segala sesuatu yang dijumpai pada diagram kelas.
3. ***Use Case* Diagram**. Bersifat statis. Diagram ini memperlihatkan himpunan *use case* dan aktor-aktor (suatu jenis khusus dari kelas).

Diagram ini terutama sangat penting untuk mengorganisasikan dan memodelkan perilaku dari suatu sistem yang dibutuhkan serta diharapkan pengguna.

4. ***Sequence*** **Diagram**. Bersifat dinamis. Diagram urutan adalah diagram interaksi yang menekankan pada pengiriman pesan (*message*) dalam waktu tertentu.
5. ***Collaboration*** **Diagram**. Bersifat dinamis. Diagram kolaborasi adalah diagram interaksi yang menekankan organisasi struktural dari objek-objek yang menerima serta mengirim pesan (*message*).
6. ***Statechar*****t Diagram**. Bersifat dinamis. Diagram ini memperlihatkan *state-state* pada sistem, memuat *state*, transisi, *event*, serta aktifitas. Diagram ini terutama penting untuk memperlihatkan sifat dinamis dari antamuka, kelas, kolaborasi dan terutama penting pada pemodelan sistem-sistem yang reaktif.
7. ***Activity*** **Diagram**. Bersifat dinamis. Diagram ini adalah tipe khusus dari diagram *state* yang memperlihatkan aliran dari suatu aktifitas ke aktifitas lainnya dalam suatu sistem. Diagram ini terutama penting dalam pemodelan fungsi-fungsi dalam suatu sistem dan memberi tekanan pada aliran kendali antarobjek.
8. ***Component*** **Diagram**. Bersifat statis. Diagram ini memperlihatkan organisasi serta kebergantungan pada komponen-komponen yang telah ada sebelumnya. Diagram ini berhubungan dengan diagram kelas dimana komponen secara tipikal dipetakan kedalam satu atau lebih kelas-kelas, antarmuka, serta kolaborasi.
9. ***Deployment*** **Diagram**. Bersifat statis. Diagram ini memperlihatkan konfigurasi saat aplikasi dijalankan. Diagram ini memuat simpul-simpul (*node*) beserta komponen-komponen yang ada didalamnya. *Deployment* diagram berhubungan erat dengan diagram komponen dimana *deployment* diagram memuat satu atau lebih komponen-komponen. Diagram ini sangat berguna saat aplikasi kita berlaku sebagai aplikasi yang dijalankan pada banyak mesin (*distributed computing*).

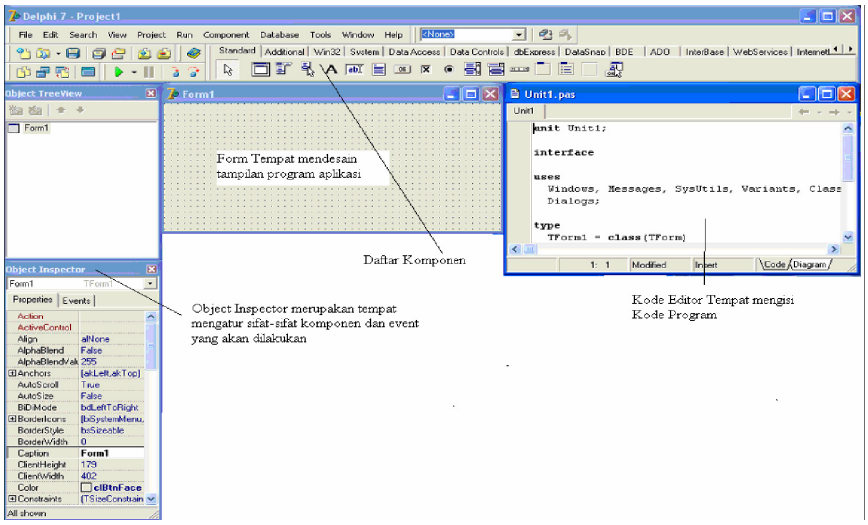
Dalam pembuatan suatu sistem informasi banyak bahasa pemrograman yang dapat digunakan sebagai media pembuatan suatu progam, diantaranya program *Borland Delphi*, *Visual Basic*, *MS. Access*, dan *Foxpro*. Dibawah ini adalah contoh bahasa pemrograman dari *Borland Delphi* dan pembuatan *database*.

2.6.4 Pengenalan Bahasa Pemrograman Delphi 7

Delphi 7 merupakan bahasa pemrograman tingkat tinggi yang sangat popular karena kehandalan, kemudahan, kecepatan dan dukungan yang luas untuk membuat program aplikasi baik yang sederhana sampai yang kompleks. Keistimewaan bahasa pemrograman ini diantaranya yaitu sebagai berikut:

a. Dapat berjalan pada Sistem Operasi Microsoft Windows 9x, ME, 2000 dan XP yang banyak dipakai oleh pengguna komputer di Indonesia.
b. Bisa digunakan untuk membuat program aplikasi perhitungan, multi media, program interaktif, program basis data maupun program yang memiliki kecerdasan yang kompleks (rumit).
c. Mudah dalam penggunaannya karena *delphi* bekerja dengan menu *user friendly*.
d. *Delphi* 7 merupakan bahasa pemrograman visual berbasis mode grafik sehingga untuk membuat program dengan tampilan yang menarik tidak memerlukan kode program yang panjang.
e. Mendukung pemrograman berorientasi objek.

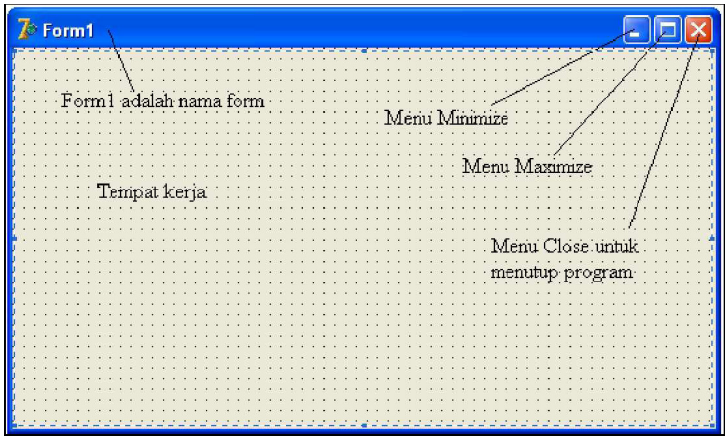
Dalam melakukan pengkodean program hendaknya program yang ditulis mengikuti algoritma yang telah ditentukan sebelumnya yang biasanya digambarkan dalam diagram *flow chart*. Tampilan awal *delphi* 7 ditunjukan pada Gambar 2.21 sebagai berikut:



Gambar 2.21 Tampilan Utama Bahasa Program Delphi 7

2.6.4.1 Form

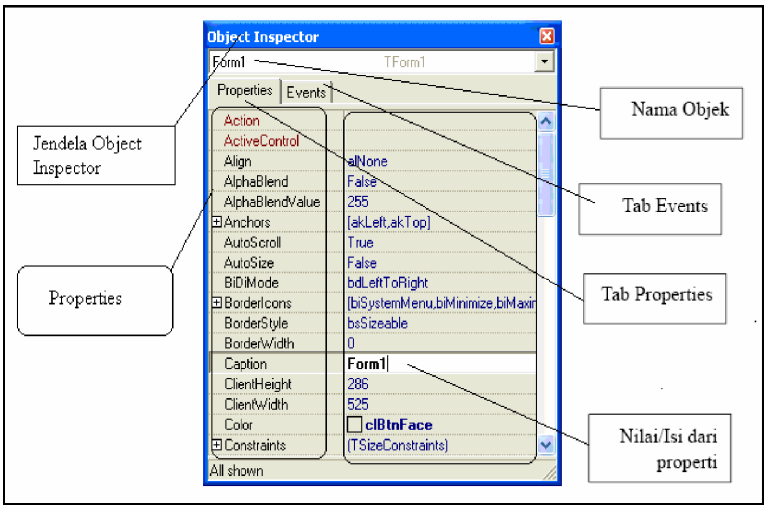
Pada saat pertama kali membuka program *delphi*, kita akan menemukan sebuah *form* kosong secara otomatis dengan bentuk seperti berikut :



Gambar 2.22 Form Pada Delphi 7

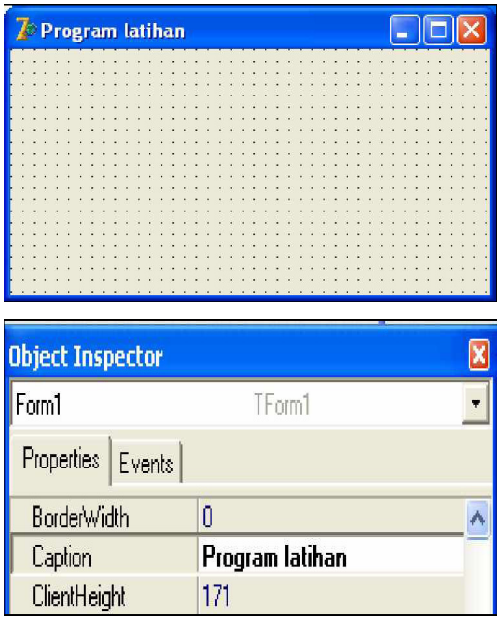
Form merupakan tempat kerja dimana komponen-komponen akan diletakkan dan output dari program akan terlihat pada form tersebut. *Form* merupakan objek yang mempunyai sifat (*properties*) dan *event* yang dapat diatur pada objek inspektor seperti Gambar 2.23 *form* diatas mempunyai sifat yaitu nama = *form1*, *align* = *alNone*, *AutoScroll* = *true*, *AutoSize* = *false*, *caption* = *form1* dan lain sebagainya seperti yang ada pada gambar 2.23 *form* mempunyai sifat yang banyak akan tetapi

sifat yang perlu diketahui bagi pemula adalah *caption* = judul *form* dan *name* = nama *form*. Sifat-sifat *form* yang lain dapat dipelajari kemudian.



Gambar 2.23 Objek Inspector (Properties dan Event)

Sifat dari *form* dapat diubah dengan mengubah *properties*-nya pada *object inspector*.



Gambar 2.24 Form dan Judul

2.6.4.2 Komponen

Delphi merupakan pengembangan bahasa pemrograman *pascal* yang sudah populer sebelumnya dengan menambah fasilitas mode grafis dengan menu *user friendly* seperti kebanyakan program aplikasi berbasis MS. Windows. Hal inilah

yang menjadi daya tarik bagi kebanyakan orang untuk menggunakan *delph*i sebagai bahasa pemrograman dewasa ini.

Dengan pemrograman visual dan ditambah dengan OOP (*Object oriented programming*), menjadikan hasil program yang dibuat dengan *delphi* mempunyai kehandalan dan kecepatan seperti *pascal* dengan tampilan yang menarik. Untuk membuat program dengan tampilan yang menarik tidak diperlukan banyak kode program yang harus ditulis karena sudah disediakan komponen visual yang dapat dipakai secara langsung dengan cara yang cukup mudah.

Untuk membuat program dengan *delphi* kita hanya menaruh komponen-komponen yang kita inginkan pada form, dan memfungsikan masing-masing komponen sesuai dengan yang kita inginkan. Delphi akan membuat kerangka program dalam *code editor* (unit). Jendela *form* digunakan untuk mendesain tampilan program, sedangkan *code editor* digunakan untuk menuliskan kode programnya. Setiap penambahan komponen pada *form delphi* akan secara otomatis menuliskan kerangka programnya dalam *code editor*. Untuk berpindah dari *form* ke *code editor*, anda dapat menekan tombol F12**.**

2.6.4.3 Macam-macam Komponen Delphi

Delphi 7 maupun versi sebelumnya menyediakan banyak komponen yang dapat digunakan dalam program, akan tetapi dalam hal ini hanya akan dibahas beberapa komponen dasar yang harus diketahui. Komponen di *delphi* diletakkan pada komponen palet yang terletak di bawah menu *pull down*.



Gambar 2.25 Komponen Palet dan Komponen

a) Komponen palet *Standart*

Komponen palet *Standart* berisi 16 komponen yaitu: *Label*, Edit, Memo, *Button*, *CheckBox*, *RadioButton*.

b) Komponen palet *Additional*

Komponen palet *Additional* berisi 25 komponen yaitu: *BitBtn*, *SpeedButton*, *MaskEdit*, *StringGrid*, *Image*, *Shape*.

c) Komponen palet *Win32*

Komponen palet *Win32* berisi 20 komponen yaitu: *TabControl*, *PageControl*, *RichEdit*, *ProgressBar*, *UpDown*, *Animate*.

d) Komponen palet *System*

Komponen palet *System* berisi 8 komponen yaitu: *Timer*, *PaintBox*, *MediaPlayer*, *OleContainer*, *DdeClientConv*, *DdeClientItem*, *DdeServerConv*, *DdeServerItem*.

e) Komponen palet Data *Access*

Komponen palet Data *Access* berisi 6 komponen yaitu: *DataSource*, *ClientDataSet*, *DataSetProvider*, *XMlTransform*, *XMlTransformProvider* , *XMlTransformClient*.

f) Komponen palet Data *Control*

Komponen palet Data *Control* berisi 15 komponen yaitu: *DBGrid*, *DBEdit*, *DBText*.

g) Komponen palet BDE

Komponen palet BDE berisi 8 komponen yaitu : *Table*, *SQL*, *StoreProc*.

h) Komponen palet ADO

Komponen palet ADO berisi 7 komponen yaitu : *ADOConnection*, *ADOTable*, *ADOQuery*.

2.6.4.4 Isi Komponen Palet Standar

Karena banyaknya komponen yang disediakan dalam *delphi* 7, dalam pembahasan ini hanya akan dibahas beberapa komponen saja yang berada pada komponen palet *standart* yaitu :

1. Komponen Label 

Label digunakan untuk memberikan label atau tulisan pada program seperti judul atau keterangan. Label memiliki beberapa *properties* yang terpenting adalah "*Name*" dan "*Caption*".

2. Komponen Edit 

Edit merupakan komponen yang digunakan untuk tempat yang dapat diisi biasanya digunakan untuk input dalam program. Edit memiliki beberapa properties yang terpenting adalah "*Name*" dan "*Text*".

3. Komponen Button 

Button merupakan komponen yang berupa tombol yang biasa digunakan untuk mengeksekusi sebuah perintah pada program visual. *Button* memiliki beberapa *properties* yang terpenting adalah "*Name*"dan "*Caption*" dan mempunyai *event* yang penting "*Onclick*".

4. Komponen Memo 

Memo hampir mirip dengan Edit tetapi Memo mempunyai baris dan kapasitas string yang lebih banyak. Memo memiliki beberapa properties yang terpenting adalah "*Name*"dan "*Lines*".

5. Komponen MainMenu 

MainMenu merupakan komponen yang digunakan untuk membuat tampilan menu *pull down* seperti pada aplikasi program berbasis *windows*. *Properties* yang penting dari komponen MainMenu adalah "*Name*"dan "*Caption*".

6. Komponen CheckBox 

CheckBox sering digunakan pada program untuk memilih pilihan yang berisi lebih dari satu (seperti pada *form* isian Hobby). *CheckBo*x memiliki beberapa *properties* yang penting antara lain "*Name*" dan "*Checked*".

7. Komponen RadioButton 

RadioButton sering digunakan pada program untuk memilih pilihan yang hanya boleh diisi 1 pilihan seperti pada *software* ujian pilihan berganda. *RadioButton CheckBox* memiliki beberapa *properties* yang penting antara lain "*Name*" dan "*Checked*".

2.6.4.5 Penulisan Kode Program

Setelah rancangan tampilan sudah jadi langkah selanjutnya adalah menuliskan kode program pada *code editor*. *Delphi* 7 sudah menyediakan kerangka program secara menyeluruh, untuk menambahkan kode program dapat dilakukan pada prosedur atau fungsi pada program secara keseluran. Pada *delphi* 7 secara otomatis akan membuat kerangka kode program sebagai berikut :

```
unit Unit1;              {Nama program (unit) adalah unit1}
interface

uses
        Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
        Dialogs; {
type
        TForm1 = class(TForm)
        Label1: TLabel;
        Button1: TButton;
        Label2: TLabel;
        Edit1: TEdit;
        Label3: TLabel;
        Edit2: TEdit;
        Button2: TButton;
        Label4: TLabel;
        Edit3: TEdit;
private
        { Private declarations }
public
        { Public declarations }
end;
var
Form1: TForm1;
implementation
{$R *.dfm}
end.
```



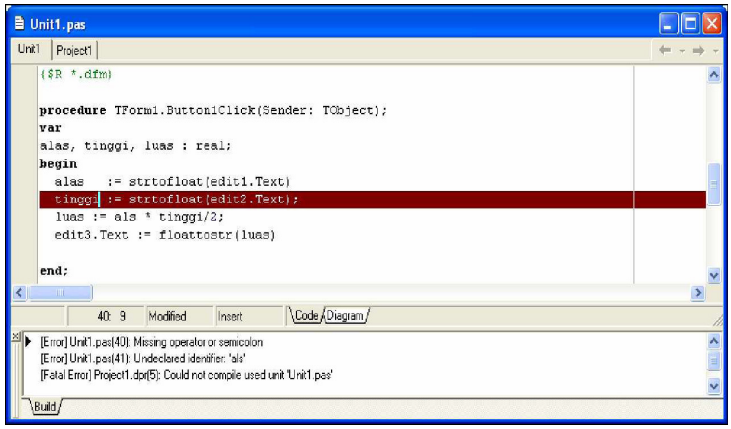
Deklarasi Komponen

Kode program di atas merupakan program yang otomatis dibuat oleh *delphi* 7 yang merupakan kerangka program. Kode program di atas perlu dilengkapi

dengan prosedur jika Button1 diklik maka akan melakukan proses tertentu. Untuk menambahkan kode program dapat dilakukan dari *form* dengan mendobel klik komponen Button1 pada *form*. Delphi secara otomatis akan menambahkan prosedure baru di dalam program. Penulisan kode program pada prosedur ditulis dalam program utama. Prosedur otomatis akan terbentuk pada saat memasukan *code*, dengan didobel klik pada *form* dan langsung berpindah ke menu *code editor*. Program yang ditambahkan disisipkan diantara prosedur.

2.6.4.6 Mengkompile dan Mengeksekusi Program

Setelah kode program diisi dengan benar, langkah selanjutnya adalah melakukan *kompile* program. Mengkompile bertujuan untuk mengetahui apakah kode program yang diisikan sudah benar atau belum baik secara sintak penulisan maupun logika pemrogramannya. Untuk melakukan kompile program dapat dilakukan dengan menekan kombinasi tombol CTRL + F9 secara bersamaan. Jika kode program yang diisikan tidak ada kesalahan maka tidak muncul pesan kesalahan sehingga dapat dilanjutkan dengan mengeksekusi program. Eksekusi program bertujuan untuk melihat apakah program yang telah dibuat sudah sesuai dengan yang diharapkan. Untuk melakukan eksekusi program caranya adalah dengan memilih menu RUN > Run atau dengan menekan tombol F9. Jika kode program yang dibuat mempunyai kesalahan maka akan muncul pesan kesalahan seperti pada gambar dibawah ini.



Gambar 2.26 Pesan Kesalahan Pada Saat Eksekusi Program

Jika terdapat kesalahan, maka kode program harus diperbaiki. *Delphi* 7 memberikan keterangan dan informasi bagian kode program yang mengalami kesalahan. Bagian yang salah akan berwarna merah seperti pada Gambar 2.26 di atas. *Delphi* juga memberikan keterangan salahnya pada bagian yang mana dan mengapa sehingga akan lebih mudah untuk memperbaikinya. Seperti pada contoh di atas ada pesan kesalahan "*Missing Operator or semicolon*" yang berarti bahwa pada kode program kurang tanda ";". Ingat setiap akhir baris harus diberi tanda ";". Pesan kesalahan kedua "*Undeclared identifier* "als" artinya bahwa data als belum didefinisikan. Data als kemungkinan terjadi karena salah dalam penulisan alas.

2.6.5 Database Desktop

Database dalam pembuatan program dengan bahasa pemrograman berbasis visual, merupakan suatu hal yang sangat penting. *Database* sering digunakan untuk menampung keluar masuknya data dalam suatu program aplikasi. Terkadang *database* juga sering digunakan untuk membantu proses pengolahan data yang rumit.

Bahasa pemrograman *Delphi* dilengkapi dengan beberapa *tools* yang sangat membantu dalam proses pembuatan program, diantaranya adalah *tools* dengan nama Database Desktop. *Tools* ini merupakan anak program Delphi yang biasa digunakan untuk membuat *database* dan tabel.

Database yang digunakan untuk membuat laporan menggunakan Delphi ada berbagai macam jenis, diantaranya adalah database dengan tipe *Paradox, MS Access, FoxPro, Dbase, Oracle, MS Sql Server*, *Interbase*, DB2, *Informix*, dan *Visual dBASE*.

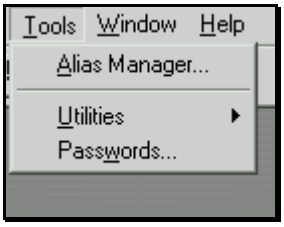
2.6.5.1 Pengantar Database Dengan Delphi

2.6.5.1.1 Pembuatan Alias

Alias digunakan sebagai short cut dari suatu sub directori/ *folder*. Sebenarnya dalam tabel kita bisa menggunakan alamat langsung dari file yang akan diakses contohnya : C:\MY DOCUMENTS\DATA\KARYAWAN.DB, hal tersebut bisa dilakukan tapi kalau file karyawan.db dipindah maka program harus dirubah. Untuk menghindari hal seperti itu, maka dibuatlah suatu Alias. Contoh kita buat suatu alias bernama Karyawan yang mewakili C:\My Documents\Data. Kalau data karyawan tersebut dipindah maka kita tidak lagi mengubah programnya tapi hanya membelokan alias tersebut ke tempat yang baru.

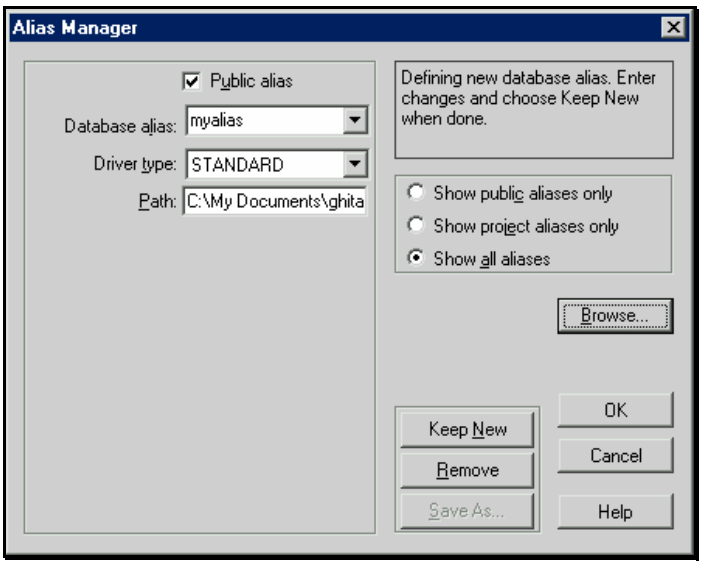
Cara pembuatan alias adalah sebagai berikut:

1. Masuk ke *Database Desktop* yang ada pada *Start Menu – Program – Borland Delphi – Database Desktop* atau pilih menu *Tool – DataBase Desktop*.
2. Pilih menu *Tools – Alias Manager*



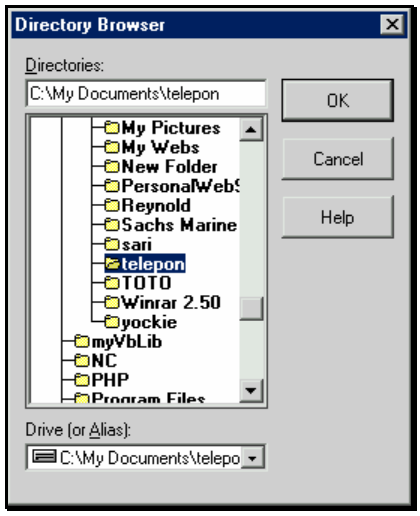
Gambar 2.27 Pilihan Tools Alias Manager

3. Klik tombol *New*, kemudian isi *edit box Database* alias dengan alias yang diinginkan, kemudian klik tombol *Browse*.



Gambar 2.28 Pembuatan Nama Database

4. Pilih *drive* yang diinginkan pada combo (*drive or alias*), kemudian pilih direktori/*folder* yang dituju pada list *Directories*. Kemudian tekan tombol OK.



Gambar 2.29 Pilihan Menu Untuk Menyimpan Database

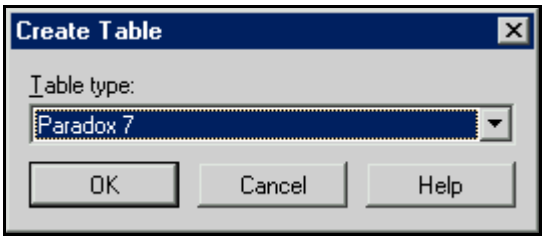
5. Jika telah benar tekan tombol OK pada Alias Manager. Maka akan muncul dialog apakah alias tersebut akan disimpan dalam *file configurasi*. Tekan tombol Yes.

2.6.5.1.2 Pembuatan Tabel

Pembuatan tabel yang akan diakses oleh *Delphi* bisa dibuat dengan *Database Desktop* atau *Ms.Access*.

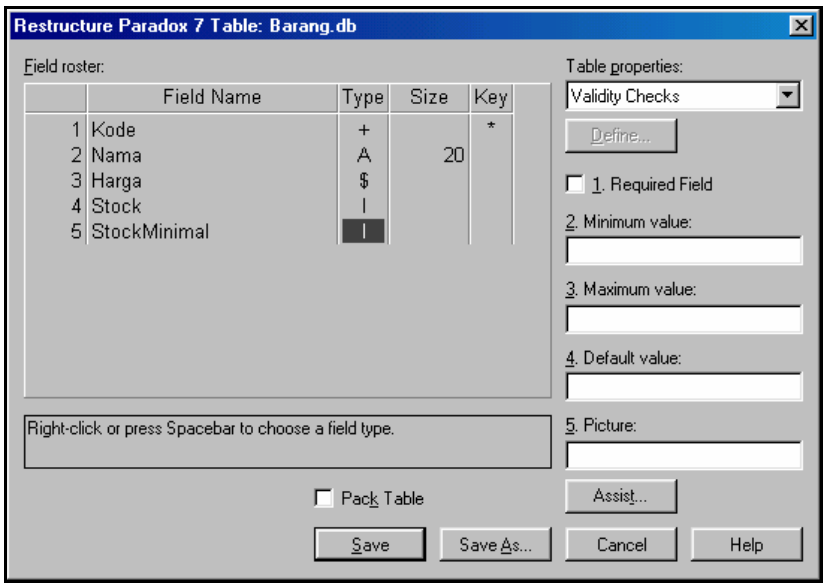
Cara pembuatan tabel dalam *Database Desktop* :

1. Masuk ke program *Database Desktop*.
2. Pilih menu *File – New – Table* sehingga akan muncul dialog berikut yang akan menanyakan tabel jenis apa yang akan dibuat apakah paradox atau dbase atau yang lain, kemudian tekan OK.



Gambar 2.30 Pilihan Type Database

3. Maka akan muncul *window* pembuatan tabel seperti berikut :



Gambar 2.31 Pembuatan Database Dengan Paradox 7

4. Jika sudah lengkap tekan tombol *Save As* untuk menyimpan tabel tersebut.
5. Tulis nama *file* tabel yang sudah dibuat dan simpan pada alias yang tadi dibuat.
6. Tabel sudah bisa diakses oleh *Delphi*.

Jenis *field-field* dalam *Paradox* yang sering dipakai :

Tabel 2.5 Jenis Field-field Dalam Paradox

Simbol Jenis Field	Jenis Field	Keterangan
A	AlphaNumeric	String, Alpabet, Numerik
I	Integer	Integer (Bilangan Bulat)
N	Number	Pecahan
D	Date	Tanggal
T	Time	Jam
M	Memo	Memo / Catatan panjang
G	Graphic	Gambar
+	AutoIncrement	Otomatis naik tiap ada penambahan data
$	Money	Mata Uang
S	Short Int	Integer kecil (-32768 s/d 32767)

- **Pengisian Data**

Pengisian data pada tabel bisa dilakukan dengan dua cara, yaitu :

1. Melalui *Database Desktop*

   Buka *Database Desktop*, kemudian pilih *File – Open – Table* kemudian pilih file *database* mana yang akan diisi.

2. Melalui program yang dibuat.