BAB II

LANDASAN TEORI

2.1 Rekayasa Perangkat Lunak

Menurut Fritz Bauer, rekayasa perangkat lunak (*Software Engineering*) adalah penggunaan prinsip-prinsip rekayasa untuk mendapatkan perangkat lunak yang ekonomis dan dapat bekerja secara efisien pada mesin nyata[6]. Dari sudut pandang lain, rekayasa perangkat lunak adalah pendekatan sistematis untuk merekayasa perangkat lunak yang handal atau bermutu, tepat waktu dan dengan biaya yang optimal.

2.2 Perkembangan Pengolah Kata

Pengolah kata (*Word Processor*) berbasis komputer sudah demikian maju. Pada waktu pertama kali program pengolah kata berbasis komputer dibuat, keuntungan yang didapat dibandingkan dengan mengetik manual hanyalah kemampuan menghapus dan menyisipkan suatu teks, sehingga memungkinkan kita mengedit atau menggunakan dokumen yang pernah dibuat untuk keperluan baru.

Perbaikan dan penambahan kemampuan pada program pengolah kata membuat pengetikan dokumen menjadi lebih mudah dan menyenangkan. Dengan pengolah kata yang sudah ditingkatkan kemampuannya, anda dapat dengan mudah mengatur bentuk/format halaman. Anda juga dengan mudah dapat memberikan efek cetak khusus.

Misalnya :

1. **Cetak Tebal**
2. *Cetak Miring*
3. Cetak Garis Bawah
4. Dan lain-lain

Kemampuan *standard* yang lain misalnya :

1. Pencarian dan pengetikan teks,
2. Penggabungan dokumen,
3. Surat berganda,
4. Pembuatan catatan kaki,
5. Pembuatan makro
6. Operasi blok,
7. Dan lain-lain.

Didukung dengan perkembangan perangkat kerasnya. Anda dapat menggunakan bermacam-macam jenis huruf, bahkan dapat mengkombinasikan dengan gambar atau grafik.

Pengolah kata masa kini sudah sedemikian majunya sehingga selain kemampuan standard diatas, pengolah kata juga dilengkapi kemampuan untuk meningkatkan mutu cetakan (*output*) dokumen yang dihasilkan. Pengolah kata masa kini sudah dapat dengan mudah menggabungkan beberapa data dan beberapa sumber program aplikasi sekaligus dalam satu dokumen, misalnya sebuah dokumen bisa membuat label *spreadsheet,* grafik dan gambar. Batasan

antara program pengolah kata dengan program *dekstop publishing* menjadi sangat sempit dan semakin kabur batasannya.

2.3 Terminologi Kriptografi

a. Pengirim dan Penerima pesan

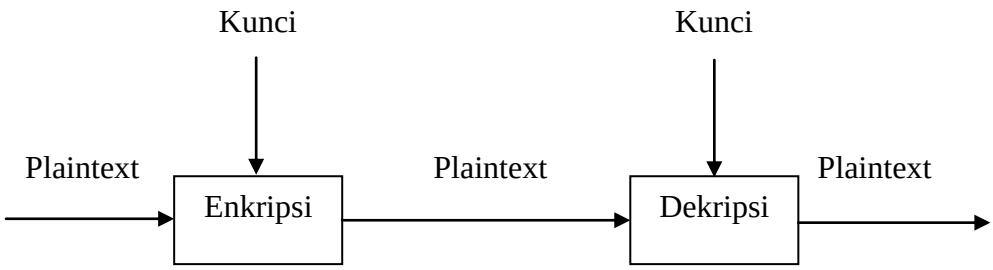
Seorang pengirim pesan (*sender*) ingin mengirim pesan kepada seorang penerima (*receiver*). Pengirim menginginkan pesan dapat dikirim secara aman, yaitu ia yakin bahwa pihak lain tidak dapat membaca isi pesan.

b. Pesan, *Plainteks*, dan *Cipherteks*

Pesan adalah data atau informasi yang dapat dibaca dan dimengerti maknanya. Nama lain untuk pesan adalah plainteks (*plaintext*) atau teks-jelas (*cleartext*). Pesan dapat berupa data atau informasi yang dikirim (melalui kurir, saluran komunikasi data, dsb) atau yang disimpan di dalam media perekaman (kertas, *storage*, dsb). Agar pesan tidak dapat dimengerti maknanya oleh pihak lain, maka pesan disandikan ke bentuk lain. Bentuk pesan yang tersandi disebut cipherteks (*ciphertext*) atau kriptogram (*cryptogram*). Cipherteks harus dapat ditransformasi kembali menjadi plainteks.

c. Enkripsi dan Dekripsi

Proses menyandikan plainteks menjadi cipherteks disebut enkripsi (*encryption*) atau *enciphering* (standard nama menurut ISO 7498-2). Proses mengembalikan cipherteks menjadi plainteksnya disebut dekripsi (decryption) atau *deciphering* (standard nama menurut ISO 7498-2).



Gambar 2.1 Enkripsi dan dekripsi

2.4 Sejarah Algoritma RSA

Algortima RSA dijabarkan pada tahun 1977 oleh tiga orang : Ron Rivest, Adi Shamir dan Len Adleman dari *Massachusetts Institute of Technology*. Huruf RSA itu sendiri berasal dari inisial nama mereka (Rivest—Shamir—Adleman).

Clifford Cocks, seorang matematikawan Inggris yang bekerja untuk GCHQ, menjabarkan tentang sistem *equivalen* pada dokumen internal di tahun 1973. Penemuan Clifford Cocks tidak terungkap hingga tahun 1997 karena alasan *top-secret classification*.

Algoritma tersebut dipatenkan oleh *Massachusetts Institute of Technology* pada tahun 1983 di Amerika Serikat sebagai U.S. Patent 4405829. Paten tersebut berlaku hingga 21 September 2000. Semenjak Algoritma RSA dipublikasikan sebagai aplikasi paten, regulasi di sebagian besar negara-negara lain tidak memungkinkan penggunaan paten. Hal ini menyebabkan hasil temuan Clifford Cocks di kenal secara umum, paten di Amerika Serikat tidak dapat mematenkannya.

2.4.1 Pembangkitan Kunci

Semisal Alice berkeinginan untuk mengizinkan Bob untuk mengirimkan kepadanya sebuah pesan pribadi (*private message*) melalui media transmisi yang tidak aman (*insecure*). Alice melakukan langkah-langkah berikut untuk membuat pasangan kunci *public key* dan *private key*:

1. Pilih dua bilangan prima $p \neq q$ secara acak dan terpisah untuk tiap-tiap p dan q. Hitung $N = p\,q$. N hasil perkalian dari p dikalikan dengan q.
2. Hitung $\varphi = (p\text{-}1)(q\text{-}1)$.
3. Pilih bilangan bulat (*integer*) antara satu dan φ ($1 < e < \varphi$) yang juga merupakan *coprime* dari φ.
4. Hitung d hingga $d\,e \equiv 1 \pmod{\varphi}$.
    a. bilangan prima dapat diuji probabilitasnya menggunakan *Fermat's little theorem* - a^(n-1) mod n = 1 jika n adalah bilangan prima, diuji dengan beberapa nilai a menghasilkan kemungkinan yang tinggi bahwa n ialah bilangan prima. *Carmichael numbers* (angka-angka *Carmichael*) dapat melalui pengujian dari seluruh a, tetapi hal ini sangatlah langka.
    b. langkah 3 dan 4 dapat dihasilkan dengan algoritma *extended Euclidean*; lihat juga aritmetika modular.
    c. langkah 4 dapat dihasilkan dengan menemukan integer x sehingga $d = (x(p\text{-}1)(q\text{-}1) + 1)/e$ menghasilkan bilangan bulat, kemudian menggunakan nilai dari $d \pmod{(p\text{-}1)(q\text{-}1)}$;
    d. langkah 2 PKCS#1 v2.1 menggunakan &lamda; = lcm(p-1, q-1) selain daripada $\varphi = (p\text{-}1)(q\text{-}1)$.

Pada *public key* terdiri atas:

a. *N*, modulus yang digunakan.
b. *e*, eksponen publik (sering juga disebut eksponen enkripsi).

Pada *private key* terdiri atas:

a. *N*, modulus yang digunakan juga pada *public key*.
b. *d*, eksponen pribadi / eksponen dekripsi, yang harus dijaga kerahasiaannya.

Biasanya, berbeda dari bentuk *private key* (termasuk parameter CRT):

a. *p* dan *q*, bilangan prima dari pembangkitan kunci.
b. *d mod (p-1)* dan *d mod (q-1)* (dikenal sebagai *dmp1* dan *dmq1*).
c. *(1/q) mod p* (dikenal sebagai *iqmp*).

2.4.2 Proses Enkripsi & Dekripsi Pesan

Misalkan Bob ingin mengirim pesan *m* ke Alice. Bob mengubah *m* menjadi angka $n < N$, menggunakan protokol yang sebelumnya telah disepakati dan dikenal sebagai *padding scheme*.

Maka Bob memiliki *n* dan mengetahui *N* dan *e*, yang telah diumumkan oleh Alice. Bob kemudian menghitung *ciphertext c* yang terkait pada *n*:

$$c = n^e \mod N$$

Alice menerima *c* dari Bob, dan mengetahui *private key* yang digunakan oleh Alice sendiri. Alice kemudian memulihkan *n* dari *c* dengan langkah-langkah berikut:

$$n = c^d \mod N$$

Perhitungan diatas akan menghasilkan *n*, dengan begitu Alice dapat mengembalikan pesan semula *m*. Prosedur dekripsi bekerja karena :

$$c^d \equiv (n^e)^d \equiv n^{ed} \pmod{N}$$

Kemudian, dikarenakan $ed \equiv 1 \pmod{p-1}$ dan $ed \equiv 1 \pmod{q-1}$, hasil dari *Fermat's little theorem*.

$$n^{ed} \equiv n \pmod{p}$$

dan

$$n^{ed} \equiv n \pmod{q}$$

Dikarenakan *p* dan *q* merupakan bilangan prima yang berbeda, mengaplikasikan *Chinese remainder theorem* akan menghasilkan dua macam kongruen

$$n^{ed} \equiv n \pmod{pq}$$

serta

$$c^d \equiv n \pmod{N}$$

Berikut ini merupakan contoh dari enkripsi RSA dan dekripsinya. Parameter yang digunakan disini berupa bilangan kecil.

Kita membuat :

$p = 61$	:	bilangan prima pertama (harus dijaga kerahasiannya atau dihapus secara hati-hati)
$q = 53$	:	bilangan prima kedua (harus dijaga kerahasiannya atau dihapus secara hati-hati)
$N = pq = 3233$	:	modulus (diberikan kepada publik)
$e = 17$	:	eksponen publik (diberikan kepada publik)

d = 2753 : eksponen pribadi (dijaga kerahasiannya)

Public key yang digunakan adalah (*e*,*N*). *Private key* yang digunakan adalah *d*.

Fungsi pada enkripsi ialah:

encrypt(n) = n^e mod N = n^{17} mod 3233

dimana *n* adalah *plaintext* Fungsi dekripsi ialah:

decrypt(c) = c^d mod N = c^{2753} mod 3233

dimana *c* adalah *ciphertext*

Untuk melakukan enkripsi *plaintext* bernilai "123", perhitungan yang dilakukan

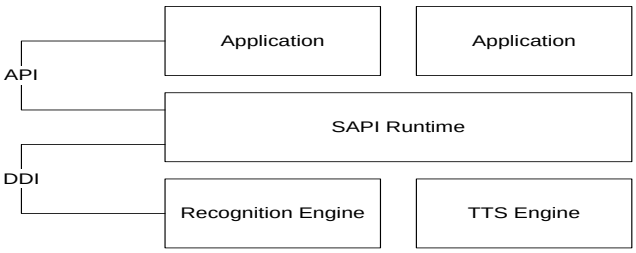
encrypt(123) = 123^{17} mod 3233 = 855

Untuk melakukan dekripsi *ciphertext* bernilai "855" perhitungan yang dilakukan

decrypt(855) = 855^{2753} mod 3233 = 123

2.5 Microsoft Speech API (SAPI)

Microsoft Speech API (SAPI) adalah suatu lapisan perangkat lunak yang digunakan oleh aplikasi pengucapan untuk berkomunikasi dengan *speech recognition (SR) engines dan Text-To-Speech (TTS) engines*. *SAPI* meliputi suatu *Application Programming Interface (API)* dan suatu *Device Driver Interface (DDI).* Aplikasi berkomunikasi dengan *SAPI* menggunakan *API layer* dan *speech engines* berkomunikasi dengan *SAPI* menggunakan layer *DDI*[3].



Gambar 2.2 Blok Diagram *SAPI*

Suatu aplikasi *Speech Recognition* dan suatu *SR engine* tidak secara langsung berkomunikasi satu sama lain. Semua komunikasi dilaksanakan menggunakan SAPI. SAPI mengendalikan sejumlah aspek suatu sistem, seperti :

1. Mengontrol *audio input* , bisa berupa inputan dari mikrofon, *files*, atau beberapa sumber *audio* lainnya, kemudian melakukan konversi data *audio* menjadi format yang didapat dibaca *engine.*
2. Memuat *grammar files,* yang di *load* ke memori atau berupa *file* dan memcahkan masukan *grammar* atau *grammar* yang diedit.
3. Mengkompilasi *standard SAPI XML*, *grammar* format dan mengkonversi format *grammar*.
4. *Sharing* hasil *recognition* ke berbagai aplikasi menggunakan *engine* yang dapat digunakan bersama, seperti halnya semua penyusunan antara aplikasi dan *engine*.
5. Mengembalikan hasil dan informasi lainnya kembali ke aplikasi dan berinteraksi dengan metode yang digunakan, dengan metode ini sebuah *engine* memiliki threading yang lebih mudah karena *SAPI* melakukan lebih banyak *thread*.
6. Penyimpanan *audio* dan melakukan serialisasi untuk kemudian di analisa.
7. Memastikan bahwa aplikasi tidak menimbulkan *error*, mencegah aplikasi memanggil *engine* yang memiliki parameter *invalid*, sehingga mencegah agar tidak terjadi *hang* dan *crash*.

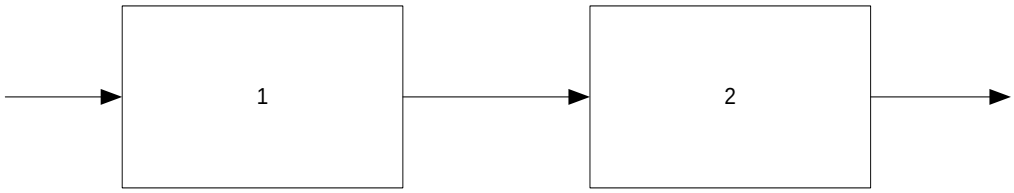
Speech recognition engine melakukan berbagai tugas sebagai berikut :

1. Menggunakan *SAPI grammar interfaces* dan memuat *dictation*.
2. Melakukan proses *speech recognition*.
3. Tempat pengembalian data *SAPI* untuk informasi tentang *grammar* dan perubahan tertentu.
4. Melakukan proses *recognition* dan *event* lainnya untuk memberikan informasi kepada aplikasi.

2.6 Konversi Dari Teks ke Ucapan

Sistem *Text to Speech* pada prinsipnya terdiri dari dua sub sistem, yaitu :

1. bagian Konverter Teks ke Fonem (*Text to Phoneme*), serta
2. bagian Konverter Fonem to Ucapan (*Phoneme to Speech*).



Gambar 2.3 Sistem *Text To Speech*

Bagian Konverter Teks ke Fonem berfungsi untuk mengubah kalimat masukan dalam suatu bahasa tertentu yang berbentuk teks menjadi rangkaian kode-kode bunyi yang biasanya direpresentasikan dengan kode fonem, durasi serta *pitch*-nya. Bagian ini bersifat sangat *language dependant*. Untuk suatu bahasa baru, bagian ini harus dikembangkan secara lengkap khusus untuk bahasa tersebut.

Bagian Konverter Fonem ke Ucapan akan menerima masukan berupa

kode-kode fonem serta pitch dan durasi yang dihasilkan oleh bagian sebelumnya. Berdasarkan kode-kode tersebut, bagian Konverter Fonem ke Ucapan akan menghasilkan bunyi atau sinyal ucapan yang sesuai dengan kalimat yang ingin diucapkan. Ada beberapa alternatif teknik yang dapat digunakan untuk implementasi bagian ini. Dua teknik yang banyak digunakan adalah *formant synthesizer*, serta *diphone concatenation*.

Formant synthesizer bekerja berdasarkan suatu model matematis yang akan melakukan komputasi untuk menghasilkan sinyal ucapan yang diinginkan. *Synthesizer* jenis ini telah lama digunakan pada berbagai aplikasi. Walaupun dapat menghasilkan ucapan dengan tingkat kemudahan interpretasi yang baik, *synthesizer* ini tidak dapat menghasilkan ucapan dengan tingkat kealamian yang tinggi.

Synthesizer yang menggunakan teknik *diphone concatenation* bekerja dengan cara menggabung-gabungkan segmen-segmen bunyi yang telah direkam sebelumnya. Setiap segmen berupa *diphone* (gabungan dua buah fonem). *Synthesizer* jenis ini dapat menghasilkan bunyi ucapan dengan tingkat kealamian (*naturalness*) yang tinggi.

Struktur sistem seperti di atas pada prinsipnya merupakan konfigurasi tipikal yang digunakan pada berbagai sistem *Text to Speech* berbagai bahasa. Namun demikian, pada setiap sub-sistem terdapat sifat-sifat serta proses-proses yang sangat spesifik dan sangat tergantung dari bahasanya.

Konversi dari teks ke fonem sangat dipengaruhi oleh aturan-aturan yang berlaku dalam suatu bahasa. Pada prinsipnya proses ini melakukan konversi dari

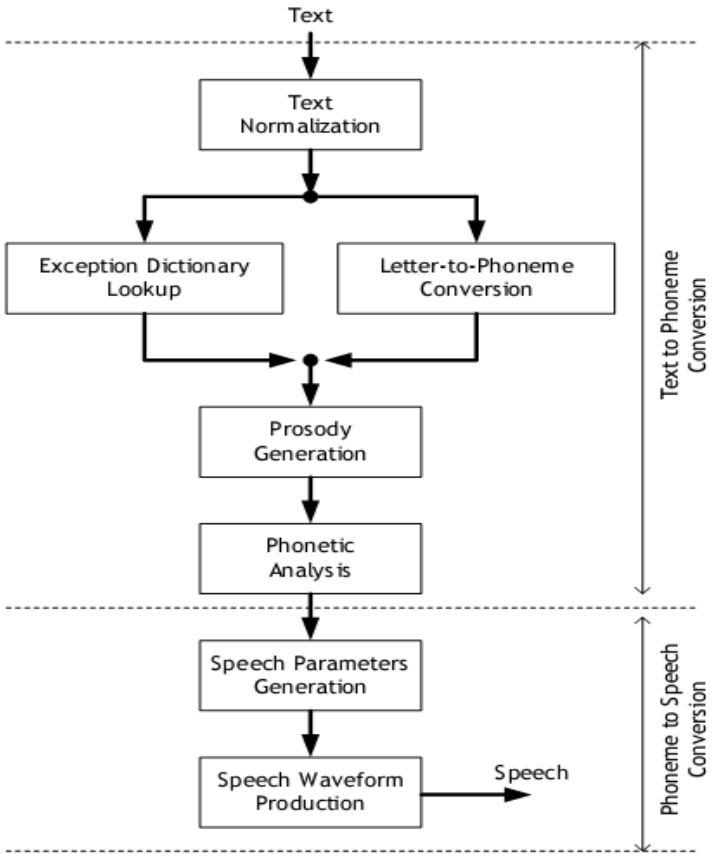
simbol-simbol tekstual menjadi simbol-simbol fonetik yang merepresentasikan unit bunyi terkecil dalam suatu bahasa. Setiap bahasa memiliki aturan cara pembacaan dan cara pengucapan teks yang sangat spesifik. Hal ini menyebabkan implementasi unit konverter teks ke fonem menjadi sangat spesifik terhadap suatu bahasa.

Untuk mendapatkan ucapan yang lebih alami, ucapan yang dihasilkan harus memiliki intonasi (*prosody*). Secara kuantisasi, prosodi adalah perubahan nilai *pitch* (frekuensi dasar) selama pengucapan kalimat dilakukan atau *pitch* sebagai fungsi waktu. Pada prakteknya, informasi pembentuk prosodi berupa data-data *pitch* serta durasi pengucapannya untuk setiap fonem yang dibangkitkan. Nilai-nilai yang dihasilkan diperoleh dari suatu model prosodi. Prosodi bersifat sangat spesifik untuk setiap bahasa, sehingga model yang diperlukan untuk membangkitkan data-data prosodi menjadi sangat spesifik juga untuk suatu bahasa. Beberapa model umum prosodi pernah dikembangkan, tetapi untuk digunakan pada suatu bahasa masih perlu banyak penyesuaian yang harus dilakukan.

Konverter fonem ke ucapan berfungsi untuk membangkitkan sinyal ucapan berdasarkan kode-kode fonem yang dihasilkan dari proses sebelumnya. Sub sistem ini harus memiliki pustaka setiap unit ucapan dari suatu bahasa. Pada sistem yang menggunakan teknik *diphone concatenation*, sistem harus didukung oleh suatu *diphone database* yang berisi rekaman segmen-segmen ucapan yang berupa diphone. Ucapan dalam suatu bahasa dibentuk dari satu set bunyi yang mungkin berbeda untuk setiap bahasa, oleh karena itu setiap

bahasa harus dilengkapi dengan *diphone* database yang berbeda.

Tahapan-tahapan utama konversi dari teks menjadi ucapan dapat dinyatakan dengan diagram seperti terlihat pada Gambar 2.3.



Gambar 2.4 Urutan Proses Konversi dari Teks ke Ucapan[5]

2.7 Sekilas Mengenai Visual Basic

Sejarah perkembangan visual basic diawali dari perkembangan bahasa *Beginner All-purpose Symbolic Instruction Code* (BASIC) di Dartmouth College, ditemukan oleh John G Kemeny dan Thomas E. Kurtz di New Hampshire, Amerika Serikat, pada awal tahun 1990-an. Bahasa Basic Merupakan bahasa tingkat tinggi yang berbentuk *interpreter*, yang memungkinkan untuk

mengoperasikan komputer secara interaktif, program dapat ditulis, dijalankan, dirubah dan dijalankan lagi tanpa harus melalui tahap kompilasi, seperti pada bahasa tinggi lainnya yang berbentuk *compiler*.

Sejak semula BASIC memang dirancang untuk mudah dipelajari. Begitu sederhananya sehingga nyaris semua pakar pemrograman komputer menggunakan BASIC sebagai bahasa pemograman pertamanya.

Pada tahun 1975, muncul komputer pribadi (PC) pertama, bermerk Mits Altair. Namun karena hanya memiliki RAM 4 Kb, satu-satunya bahasa yang bisa untuk memprogramnya adalah *assembly*, bahasa pemograman yang cukup sulit. Tentu saja hal ini sangat menghambat perkembangan komputer pribadi. Bill Gates dan Paul Allen, dua orang pemuda Amerika yang tertarik pada komputer melihat potensi besar komputer pribadi di masa depan. Keduanya kemudian menulis BASIC untuk Altair. Ternyata sambutan masyarakat sangat hangat, dan keduanyapun mendirikan Microsoft.

Pada tahun 1982 IBM/PC diperkenalkan pada masyarakat. Microsoft lalu membuat sistem operasi MS-DOS untuk komputer ini. Didalamnya disertakan juga bahasa BASIC yang dikenal QuickBasic (Qbasic). Padatahun 1990-an era DOS digantikan dengan Windows. Tampilan Grafis Windows (GUI) yang badus dan interaktif mengubah pemrograman menjadi sebuah pekerjaan yang menyenangkan. Microsoft akhirnya membuat *Basic* versi *Windows.* Versi *MS Visual Basic 6.0* diluncurkan awal tahun 1999.

Visual Basic memungkinkan pembuatan *Graphical User Interface* (GUI), atau pemrograman yang menggunakan tampilan grafis sebagai alat komunikasi

dengan pemakainya. Kita cukup berkonsentrasi pada struktur dan logika dari program utamanya. Pembuatan tampilan *user interface* yang mewah tersebut relatif mudah dilakukan karena kita hanya perlu meletakan objek-objek grafis ke lembar (*form*) yang sudah disedikan Visual Basic. Setelah itu kita cukup mengatur properti dari objek-objek tersebut.

Microsoft Visual Basic 6.0 mempunyai fleksibilitas yang sangat baik untuk berhubungan dengan aplikasi lain yang menggunakan objek *Object Lingking and Embedding* (OLE). Fasilitas-fasilitas dalam menggunakan *Microsoft Visual Basic 6.0* diantaranya :

a. *Project Manager*

   Berfungsi sebagai sentral dalam proses penanganan *Database*, dengan fasilitas ini dapat menjangkau seluruh fasilitas pengolahan *database* seperti kreasinya, penyaringan isi data sampai pembentukan laporan.

b. *Database Manager*

   Fasilitas ini dapat memberikan kemudahan dan mengatur dengan cepat struktur *database, index, table, indeks* hingga format elemen *database*.

c. *Form Designer*

   Fasilitas ini berfungsi untuk mengatur format tampilan layar dengan lebih cepat dan mudah. Aplikasi *Databse* terbaru dengan format terbaru dapat dibuat dengan memenfaatkan fasilitas data *Access*.

d. *Query and View Manager*

   Perluasan fasilitas ini mampu mengontrol hubungan *database* dan format data.

2.8 Pemrograman Berorientasi Objek (*Object Oriented Programming*)

Booch menyatakan[2]:

> *Pemrograman berorientasi objek adalah metode implementasi dimana program diorganisasikan sebagai kumpulan objek yang bekerja sama, masing–masing objek merepresentasikan instant dari kelas, dan kelas-kelas itu anggota suatu hirarki kelas-kelas yang disatukan lewat keterhubungan pewarisan.*

Sebagai kakas rekayasa perangkat lunak, bahasa pemrograman harus mengatasi isu-isu mengenai dukungan pengelolaan kompleksitas proyek pengembangan perangkat lunak.

Tiga aspek penting dalam pemrograman berorientasi objek :

1. Menggunakan objek-objek bukan algoritma-algoritma sebagai blok-blok bangunan logik dasar.
2. Masing–masing objek adalah instan suatu kelas.
3. Kelas-kelas saling berhubungan lewat keterhubungan pewarisan.

Pemrograman berorientasi objek adalah metode implementasi dimana program diorganisasikan sebagai komunitas objek yang bekerja sama berinteraksi. Masing–masing merepresentasikan instan suatu kelas yang menyediakan sejumlah layanan yang relative sederhana. Objek-objek saling berkomunikasi untuk mempertukarkan informasi. Perilaku komplek diperoleh dengan kerjasama antar objek-objek. Kelas–kelas dapat merupakan anggota hirarki yang disatukan lewat hubungan pewarisan.

2.9 Karakteristik Pemrograman Berorientasi Objek (OOP)

Ada tiga karakteristik utama dari OOP yaitu[2]:

1. *Encapsulation* (pengkapsulan atau pembungkusan)
2. *Inheritance* (Pewarisan)
3. *Polymorphism*

2.9.1 *Encapsulation*

Pengkapsulan atau pembungkusan adalah proses pemaketan data objek bersama metode-metodenya. Manfaat utama pengkapsulan adalah penyembunyian rincian–rincian implementasi dari pemakai/objek lain. Bagian internal objek mempunyai ketampakan terbatas dibanding bagian eksternal objek. Penyusunan seperti ini menjaga internal dari pengaksesan eksternal yang tidak diinginkan.

2.9.2 *Inheritance*

Pewarisan adalah salah satu konsep terpenting pemrograman berorientasi objek dan mempunyai dampak langsung pada cara merancang dan menulis kelas. Pewarisan adalah proses penciptaan kelas baru dengan mewarisi karakteristik kelas yang telah ada, ditambah karakteristik unik kelas baru itu. Pewarisan adalah mekanisme ampuh yang memungkinkan kelas mewarisi fungsionalitas kelas yang telah ada. Untuk menciptakan kelas baru, kita hanya perlu menspesifikasikan cara kalas baru itu berada dari kelas yang telah ada. Pewarisan merupakan mekanisme yang ampuh dan alami untuk mengorganisasikan program.

2.9.3 *Polymorphism*

Polymorphism merupakan konsep pokok di dalam perancangan berorientasi objek. *Polymorphism* berarti mempunyai banyak bentuk. Dua objek atau lebih dikatakan sebagai polymorphism bila objek–objek itu mempunyai antarmuka–antarmuka identik namun mempunyai perilaku-perilaku berbeda. *Polymorphism* berupa satu nama tunggal (seperti deklarasi fungsi) dan dapat menyatakan objek kelas-kelas berbeda yang terhubung dengan superkelas yang umum di antara kelas-kelas itu. Sembarang objek yang ditandai dengan nama ini dapat menanggapi sekumpulan operasi yang sama.

2.10 Definisi UML (*Unified Modeling Language*)

UML adalah bahasa grafis untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak[7]. UML berorientasi objek, menerapkan banyak level abstraksi, tidak bergantung proses pengembangan, tidak bergantung bahasa dan teknologi, pemaduan beberapa notasi di beragam metodologi, usaha bersama dari banyak pihak, didukung oleh kakas – kakas yang diintegrasikan lewat XML . Standar UML dikelola oleh OMG (*Object Management Group*).

UML adalah bahasa pemodelan untuk menspesifikasikan, memvisualisasikan, membangun, dan mendokumentasikan artifak-artifak dari sistem.

1. Di dalam *system intensive process*, metode diterapkan sebagai proses untuk menurunkan atau mengevolusikan sistem.
2. Sebagai bahasa, UML digunakan untuk komunikasi yaitu alat untuk menangkap pengetahuan mengenai satu subjek dan mengekspresikan

pengetahuan (sintaks) yang memperdulikan subjek untuk maksud komunikasi. Subjek adalah sistem yang dibahas.

3. Sebagai bahasa pemodelan, UML focus pada pemahaman subyek melalui formulasi model dari subyek. Model memuat pengetahuan pada subyek, dan aplikasi dari pengetahuan ini berkaitan dengan intelejensia.
4. Berkaitan dengan unifikasi.UML memadukan praktek rekayasa terbaik sistem informasi dan industri, meliputi beragam tipe sistem (perangkat lunak dan non perangkat lunak), domain (bisnis, perangkat lunak) dan proses siklus hidup.
5. begitu diterapkan untuk menspesifikasikan sistem, UML dapat digunakan untuk mengkomunikasikan "apa" yang diperlukan dari sistem dan "bagaimana" sistem dapat direalisasikan.
6. Begitu diterapkan untuk menvisualisasikan sistem,UML dapat digunakan untuk menjelaskan sistem secara visual sebelum direalisasikan.
7. Begitu diterapkan untuk membangun sistem, UML dapat digunakan untuk memandu realisasi sistem serupa dengan "*blueprint*".
8. Begitu diterapkan untuk mendokumentasikan sistem, UML dapat digunakan untuk menangkap pengetahuan mengenai sistem pada seluruh siklus hidup.

2.11 Tujuan UML

Tujuan utama perancangan UML adalah[1]:

1. Menyediakan bahasa pemodelan visual yang ekspresif dan siap pakai untuk mengembangkan dan pertukaran model-model yang berarti.

2. Menyediakan mekanisme perluasan dan spesifikasi untuk memperluas konsep-konsep inti.
3. Mendukung spesifikasi independen bahasa pemrograman dan proses pengembangan tertentu.
4. Menyediakan basis formal untuk pemahaman bahasa pemodelan.
5. Mendorong pertumbuhan pasar kakas berorientasi objek.
6. Mendukung konsep-konsep pengembangan level lebih tinggi seperti komponen, kolaborasi, *framework* dan *pattern.*

2.12 Diagram dan Teknik Pemodelan UML

Diagram mengemukakan banyak hal, penggunaan notasi yang terdefinisi baik dan ekspresif adalah penting pada proses pengembangan perangkat lunak, yaitu :

1. Notasi standar memungkinkan pengembang mendeskripsikan skenario atau rumusan arsitektur dan kemudian mengkomunikasikan secara tidak ambigu.
2. Notasi yang bagus membebaskan otak untk berkonsentrasi pada masalah-masalah yang lebih lanjut.
3. Notasi yang baik memungkinkan mengeleminasi keperluan pemeriksaan konsistensi dan kebenaran keputusan dengan menggunakan tool terotomatisasi.

2.12.1 Diagram struktur

Diagram ini untuk memvisualisasi, menspesifikasikan, membangun dan mendokumentasikan aspek statik dari sistem.

1. Diagram kelas (*Class diagram*) : Diagram ini menunjukkan sekumpulan kelas, *interface*, kolaborasi dan keterhubungannya. Diagram kelas ditujukan untuk pandangan statik terhadap sistem.
2. Diagram objek (*Objek diagram*) : Diagram ini menunjukkan sekumpulan objek dan keterhubungannya. Diagram ini menunjukkan potongan statik dari instan-instan yang ada di diagram kelas. Diagram ini untuk memperlihatkan satu prototipe atau kasus tertentu yang mungkin terjadi. Diagram objek menyediakan notasi grafis formal guna memodelkan objek, kelas, dan saling keterhubungan. Diagram objek berguna untuk *abstract modeling* dan perancangan program-program sesungguhnya. Pada pendekatan ini, bentukan dasar dari sistem perangkat lunak adalah objek atau kelas. Kelas adalah deskripsi dari objek-objek yang umum. Setiap objek mempunyai identitas, *state* dan perilaku.
3. Diagram komponen (*Component diagram*) : Diagram ini menunjukkan organisasi dan kebergantungan di antara sekumpulan komponen. Diagram ini merupakan pandangan static terhadap implementasi sistem.
4. Diagram pengembangan (*Deployment diagram*) : Diagram ini menunjukkan konfigurasi pemrosesan saat jalan dan komponen-komponen yang terdapat didalamnya. Diagram ini merupakan pandangan static dari arsitektur. Pilihan model dan diagram yang digunakan dipengaruhi oleh bagaimana persoalan ditangani dan bagaimana solusi dibentuk. Abstraksi, fokus pada relevan sambil mengabaikan rincian-rincian yang tidak relevan merupakan kuncinya. Karena itu, setiap sistem komplek perlu didekati melalui sekumpulan pandangan model yang

hampir *independent*. Tidak ada satu pandangan tunggal yang memadai untuk sistem komplek.

2.12.2 Diagram perilaku

Diagram ini untuk memvisualisasi, menspesifikasi, membangun dan mendokumentasikan aspek dinamis dari sistem.

1. Diagram *use-case* (*Use case diagram*) : Diagram ini menunjukkan sekumpulan kasus fungsional dan aktor (jenis kelas khusus) dan keterhubungannya.
2. Diagram sekuen (*Sequen diagram*) : Diagram ini menunjukkan interaksi yang terjadi antar objek. Diagram ini merupakan pandangan dinamis terhadap sistem. Diagram ini menekankan pada basis keberurutan waktu dari pesan-pesan yang terjadi.
3. Diagram kolaborasi (*Collaboration diagram*) : Diagram ini juga merupakan diagram interaksi. Diagram ini menekankan pada organisasi struktur dari objek-objek yang mengirim dan menerima pesan.
4. Diagram *statechart* (*Statechart diagram*) : Diagram ini berisi state, transisi, kejadian dan aktivitas. *Statechart* merupakan pandangan dinamis dari sistem. Diagram ini penting dalam memodelkan perilaku antarmuka, kelas, kolaborasi dan menekankan pada urutan kejadian. Penting untuk sistem reaktif yang dipicu kejadian di dunia nyata.
5. Diagram aktivitas (*Activity Diagram*) : Diagram ini untuk menunjukkan aliran aktivitas di sistem. Diagram aktivitas adalah pandangan dinamis

terhadap sistem. Diagram aktivitas penting untuk memodelkan fungsi sistem dan menekankan pada aliran kendali di antara objek–objek.