BAB 2

TINJAUAN PUSTAKA

2.1 *Artificial Intelligence* (AI)

2.1.1 Sejarah AI

Kecerdasan buatan atau AI termasuk bidang ilmu yang relatif muda. Pada tahun 1950-an para ilmuan dan peneliti mulai memikirkan bagaimana caranya agar mesin dapat melakukan pekerjaannya seperti yang bisa dikerjakan oleh manusia. Alan Turing, seorang matematikawan dari Inggris pertama kali mengusulkan adanya tes untuk melihat bisa tidaknya sebuah mesin dikatakan cerdas. Hasil tes tersebut kemudian dikenal dengan *Turing Test*, dimana si mesin tersebut menyamar seolah-oleh sebagai seseorang di dalam suatu permainan yang mampu memberikan respon terhadap serangkaian pertanyaan yang diajukan. Turing beranggapan bahwa, jika mesin dapat membuat seseorang percaya bahwa dirinya mampu berkomunikasi dengan orang lain, maka dapat dikatakan bahwa mesin tersebut cerdas, seperti layaknya manusia **[1]**.

AI itu sendiri dimunculkan oleh seorang profesor dari *Massachusetts Institute of Technology* yang bernama John McCarthy pada tahun 1956 pada *Darmouth Conference* yang dihadiri oleh para peneliti AI. Pada konferensi tersebut juga didefinisikan tujuan utama dari kecerdasan buatan, yaitu mengetahui dan memodelkan proses-proses berpikir manusia dan mendesain mesin agar dapat menirukan kelakuan mesin tersebut.

Beberapa program AI yang mulai dibuat pada tahun 1956-1966, antara lain **[1]** :

1. *Logic Theorist*, diperkenalkan pada *Dartmouth Conference*, program ini dapat membuktikan teorema-teorema matematika.
2. *Sad Sam*, diprogram oleh Robert K. Lindsay (1960). Program ini dapat mengetahui kalimat-kalimat sederhana yang ditulis dalam bahasa Inggris dan mampu memberikan jawaban dari fakta-fakta yang didengar dalam sebuah percakapan.
3. ELIZA, diprogram oleh Joseph Weizenbaum (1967). Program ini mampu melakukan terapi terhadap pasien dengan memberikan beberapa pertanyaan dan jawaban.

2.1.2 Definisi AI

Kecerdasan buatan atau AI adalah salah satu bagian ilmu komputer yang mempelajari tentang bagaimana cara membuat agar komputer dapat melakukan pekerjaan seperti yang dilakukan oleh manusia. Tujuan dari AI adalah untuk memecahkan persoalan dunia nyata (bersifat praktis) dan memahami intelijensia (bersifat memahami) **[1]**.

AI merupakan salah satu bagian ilmu komputer yang mempelajari tentang bagaimana cara membuat agar komputer dapat melakukan pekerjaan seperti yang dilakukan oleh manusia. Pada awal diciptakannya, komputer hanya difungsikan sebagai alat hitung saja. Namun seiring dengan perkembangan zaman, maka peran komputer semakin mendominasi kehidupan umat manusia. Komputer tidak lagi hanya digunakan sebagai alat hitung, lebih dari itu, komputer diharapkan untuk

dapat diberdayakan untuk mengerjakan segala sesuatu yang bisa dikerjakan oleh manusia.

Manusia bisa menjadi pandai dalam menyelesaikan segala permasalahan di dunia ini karena manusia mempunyai pengetahuan dan pengalaman. Pengetahuan diperoleh dari belajar. Semakin banyak bekal pengetahuan yang dimiliki oleh seseorang tentu saja diharapkan akan lebih mampu dalam menyelesaikan permasalahan. Namun bekal pengetahuan saja tidak cukup, manusia juga diberi akal untuk melakukan penalaran, mengambil kesimpulan berdasarkan pengetahuan dan pengalaman yang mereka miliki. Tanpa memiliki kemampuan menalar yang baik, manusia dengan segudang pengalaman dan pengetahuan tidak akan dapat menyelesaikan masalah dengan baik. Demikian pula, dengan kemampuan menalar yang sangat baik, namun tanpa bekal pengetahuan dan pengalaman yang memadai, manusia juga tidak akan bisa menyelesaikan masalah dengan baik.

Agar komputer bisa bertindak seperti dan sebaik manusia, maka komputer juga harus diberi bekal pengetahuan dan mempunyai kemampuan untuk menalar. Untuk itu pada *artificial intelligence*, akan mencoba untuk memberikan beberapa metode untuk membekali komputer dengan kedua komponen tersebut agar komputer bisa menjadi mesin yang pintar.

2.1.3 AI dari Berbagai Sudut Pandang

AI dapat dilihat dari berbagai sudut pandang **[3],** antara lain:

1. Sudut pandang kecerdasan

   AI akan membuat mesin menjadi ‘cerdas’ (mampu berbuat seperti apa yang dilakukan oleh manusia).

2. Sudut pandang penelitian

   AI adalah suatu studi bagaimana membuat agar komputer dapat melakukan sesuatu sebaik yang dikerjakan oleh manusia.

   Domain yang sering dibahas oleh para peneliti meliputi:

   a. Mundane task

   - Persepsi (*vision & speech*).
   - Bahasa alami (*understanding, generation* & *translation*).
   - Pemikiran yang bersifat *commonsense*.
   - Robot *control*.

   b. Formal task

   - Matematika (geometri, logika, kalkulus integral, pembuktian).
   - Permainan / *games*.

   c. Expert task

   - Analisis finansial.
   - Analisis medikal.
   - Analisis ilmu pengetahuan.
   - Rekayasa (desain, pencarian kegagalan, perencanaan manufaktur)

3. Sudut pandang bisnis.

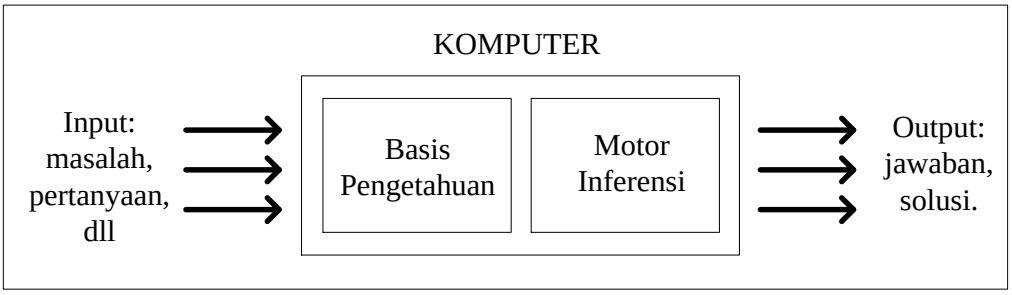
   AI adalah kumpulan peralatan yang sangat *powerful* dan metodologis dalam menyelesaikan masalah-masalah bisnis.

4. Sudut pandang pemograman.

   AI meliputi studi tentang pemograman simbolik, penyelesaian masalah (*problem solving*) dan pencarian (*searching*).

Untuk menciptakan aplikasi AI ada 2 bagian utama yang sangat dibutuhkan **[3]** yaitu:

1. Basis Pengetahuan (*Knowledge Base*), berisi fakta-fakta, teori, pemikiran dan hubungan antara satu dengan yang lainnya.
2. Motor Inferensi (*Inference Engine*), yaitu kemampuan menarik kesimpulan berdasarkan pengalaman.



Gambar 2.1. Penerapan konsep AI di dalam komputer [3]

2.1.4 Komputasi AI dan Komputasi Konvensional

Seperti telah diketahui sebelumnya, bahwa pada awal diciptakannya, komputer hanya diperuntukkan sebagai alat hitung (komputasi konvensional). Untuk itu, ada beberapa perbedaan antara komputasi yang dilakukan pada AI dengan komputasi konvensional tersebut. Perhatikan tabel berikut **[3]**.

Tabel 2.1. AI Vs. Pemograman konvensional

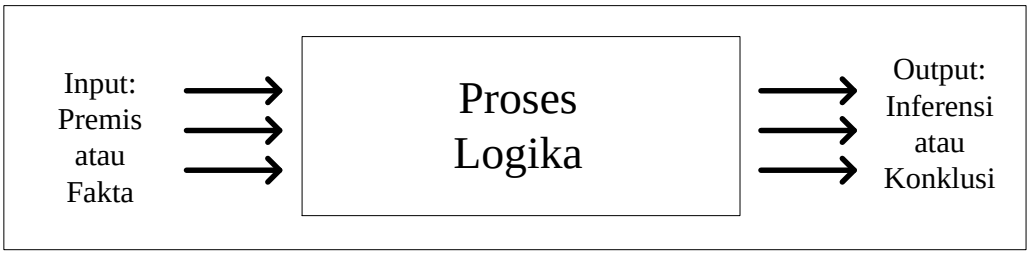
Dimensi	AI	Pemograman Konvensional
Pemrosesan	Mengandung konsep-konsep simbolik	Algoritmik
Sifat Input	Bisa tidak lengkap	Harus lengkap
Pencarian	Kebanyakan bersifat heuristik	Biasanya didasarkan pada algoritma
Keterangan	Disediakan	Biasanya tidak disediakan
Fokus	Pengetahuan	Data dan informasi
Struktur	Kontrol dipisahkan dari pengetahuan	Kontrol terintegrasi dengan informasi (data)
Sifat *output*	Kuantitatif	Kualitatif
Pemeliharaan & *update*	Relatif mudah	Sulit
Kemampuan menalar	Ya	Tidak

2.2 Representasi Pengetahuan

Untuk menyelesaikan suatu permasalahan dalam AI, dibutuhkan pengetahuan yang cukup. Tidak hanya itu, sistem juga harus memiliki kemampuan unuk menalar. Basis pengetahuan dan kemampuan untuk melakukan penalaran merupakan bagian terpenting dari sistem yang menggunakan AI. Meskipun suatu sistem memiliki banyak pengetahuan, namun apabila tidak memiliki kemampuan untuk menalar, tentu akan menjadi percuma saja. Demikian pula sebaliknya, apabila suatu sistem memiliki kemampuan yang sangat handal untuk menalar, namun basis pengetahuan yang dimilikinya tidak cukup, maka solusi yang diperoleh pun menjadi tidak maksimal.

2.2.1 Logika

Logika adalah bentuk representasi pengetahuan yang paling tua. Pada dasarnya, proses logika adalah proses membentuk kesimpulan atau menarik suatu inferensi berdasarkan fakta yang telah ada. *Input* dari proses logika berupa premis atau fakta-fakta yang diakui kebenarannya sehingga dengan melakukan penalaran pada proses logika dapat dibentuk suatu inferensi atau kesimpulan yang benar pula.



Gambar 2.2. Proses logika [3]

Ada 2 penalaran yang dapat dilakukan untuk mendapatkan konklusi **[3]**:

1. Penalaran Deduktif.

   Penalaran dimulai dari prinsip umum untuk mendapatkan konklusi yang lebih khusus.

   Contoh:

   Premis Mayor : Jika hujan turun, saya tidak akan berangkat kuliah.

   Premis Minor : Hari ini hujan turun.

   Konklusi : Hari ini saya tidak akan berangkat kuliah.

2. Penalaran Induktif.

   Penalaran dimulai dari fakta-fakta khusus untuk mendapatkan kesimpulan umum.

   Premis-1 : Aljabar adalah pelajaran yang sulit.

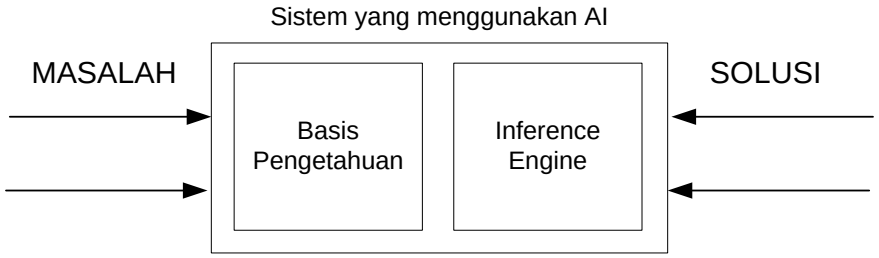
   Premis-2 : Geometri adalah pelajaran yang sulit.

   Premis-3 : Kalkulus adalah pelajaran yang sulit.

   Konklusi : Matematika adalah pelajaran yang sulit

2.3 Masalah Ruang Keadaan *(Space and Search Problem)*

Sistem yang menggunakan kecerdasan buatan akan mencoba untuk memberikan *output* berupa solusi dari suatu masalah berdasarkan kumpulan pengetahuan yang ada.



Gambar 2.3. Sistem yang menggunakan AI [3]

Pada Gambar 2.3, *input* yang diberikan pada sistem yang menggunakan kecerdasan buatan berupa masalah. Pada sistem harus dilengkapi dengan sekumpulan pengetahuan yang ada pada basis pengetahuan. Sistem harus memiliki *inference engine* agar mampu mengambil kesimpulan berdasarkan fakta atau pengetahuan. *Output* yang diberikan berupa solusi masalah sebagai hasil dari inferensi.

Secara umum, untuk membangun suatu sistem yang mampu menyelesaikan masalah, perlu dipertimbangkan 4 hal:

1. Mendefinisikan masalah dengan tepat. Pendefinisian ini mencakup spesifikasi yang tepat mengenai keadaan awal dan solusi yang diharapkan.
2. Menganalisis masalah tersebut serta mencari beberapa teknik penyelesaian masalah yang sesuai.

3. Merepresentasikan pengetahuan yang perlu untuk menyelesaikan masalah tersebut.
4. Memilih teknik penyelesaian masalah yang terbaik.

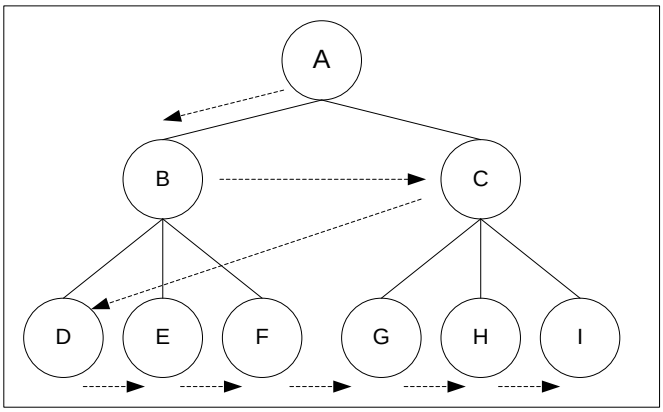
2.4 Metode Pencarian dan Pelacakan

Hal terpenting dalam menentukan keberhasilan sistem yang berlandaskan kecerdasan buatan adalah kesuksesan dalam pencarian dan pencocokan. Pada dasarnya ada 2 teknik pencarian dan pelacakan yang digunakan, yaitu pencarian buta (*blind search*) dan pencarian terbimbing (*heuristic search*).

2.4.1 Pencarian Buta *(Blind Search)*

2.4.1.1 Pencarian Melebar Pertama *(Breadth-First Search)*

Pada metode *Breadth-First Search*, semua *node* pada level n akan dikunjungi terlebih dahulu sebelum mengunjungi *node-node* pada level n+1. Pencarian dimulai dari *node* akar terus ke level ke-1 dari kiri ke kanan, kemudian berpindah ke level berikutnya **[3]**. Demikian seterusnya hingga ditemukannya solusi.



Gambar 2.4. Metode *Breadth-First Search* [3]

Algoritma :

1. Buat suatu variabel *Node_List* dan tetapkan sebagai keadaan awal.
2. Kerjakan langkah-langkah berikut ini sampai tujuan tercapai atau *Node_List* dalam keadaan kosong :
    a. Hapus elemen pertama dari *Node_List*, sebut dengan nama D. Jika *Node_List* kosong, keluar.
    b. Pada setiap langkah yang aturannya cocok dengan D, kerjakan :
        1) Aplikasikan aturan tersebut untuk membentuk suatu keadaan baru
        2) Jika keadaan awal adalah tujuan yang diharapkan, sukses dan keluar.
        3) Jika tidak demikian, tambahkan keadaan awal yang baru tersebut pada akhir *Node_List*.

Keuntungan dari metode ini:

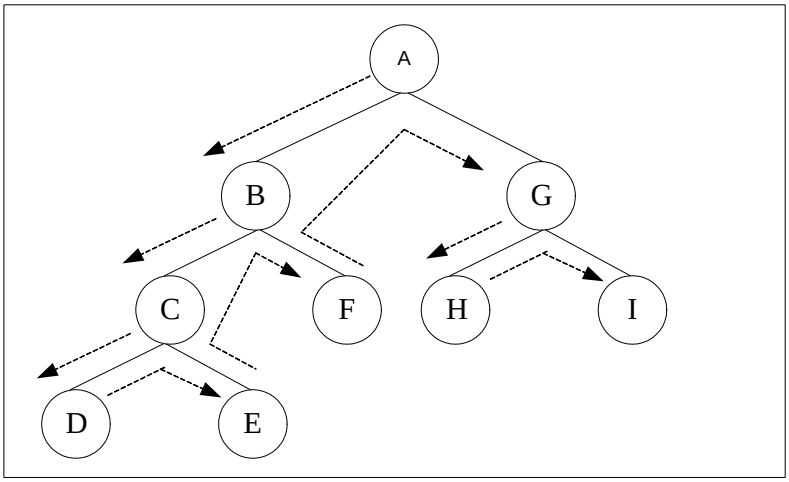
1. Tidak akan menemui jalan buntu.
2. Jika ada satu solusi, maka *breadth-first search* akan menemukannya. Dan jika ada lebih dari satu solusi, maka solusi minimum akan ditemukan.

Kelemahan dari metode ini:

1. Membutuhkan memori yang cukup banyak, karena menyimpan semua *node* dalam satu pohon.
2. Membutuhkan waktu yang cukup lama, karena akan menguji *n* level untuk mendapatkan solusi pada level ke- (n + 1).

2.4.1.2 Pencarian Mendalam Pertama *(Depth-First Search)*

Pencarian dengan metode *Depth-First Search* (DFS) dilakukan dari *node* awal secara mendalam hingga yang paling akhir (*dead-end*) atau sampai ditemukan. Dengan kata lain, simpul cabang atau anak yang terlebih dahulu dikunjungi. Gambaran pencarian DFS dapat dilihat pada gambar 2.4 **[3].**



Gambar 2.5. Pencarian mendalam pertama *(Depth-First Search)* [3]

Pencarian dilakukan pada suatu simpul dalam setiap level dari yang paling kiri. Jika pada level yang paling dalam tidak ditemukan solusi, maka pencarian dilanjutkan pada simpul sebelah kanan dan simpul yang kiri dapat dihapus dari memori. Jika pada level yang paling dalam tidak ditemukan solusi, maka pencarian dilanjutkan pada level sebelumnya. Demikian seterusnya sampai ditemukan solusi.

Algoritma :

1. Jika keadaan awal merupakan tujuan, keluar (Sukses).
2. Jika tidak demikian, kerjakan langkah-langkah berikut ini sampai tercapai keadaan sukses atau gagal :
    a. Bangkitkan *successor* D dari keadaan awal. Jika tidak ada *successor*, maka akan terjadi kegagalan.
    b. Panggil *Depth-first search* dengan D sebagai keadaan awal.
    c. Jika sukses berikan tanda sukses. Namun jika tidak, ulangi langkah-2.

Keuntungan dari metode DFS :

1. Membutuhkan memori yang relatif kecil, karena hanya *node-node* pada lintasan yang aktif saja yang disimpan.
2. Secara kebetulan, metode DFS akan menemukan solusi tanpa harus menguji lebih banyak lagi dalam ruang keadaan, sehingga tidak memboros waktu untuk melakukan sejumlah besar keadaan ‘dangkal’ dalam permasalahan graf/ pohon.

Kelemahan dari metode DFS:

1. Memungkinkan tidak ditemukannya tujuan yang diharapkan.
2. Hanya akan mendapatkan satu solusi pada setiap pencarian.

2.4.2 Pencarian Heuristik *(Heuristic Search)*

Pencarian buta tidak selalu dapat diterapkan dengan baik, hal ini disebabkan waktu aksesnya yang cukup lama serta besarnya memori yang diperlukan. Kelemahan ini sebenarnya dapat diatasi jika ada informasi tambahan dari domain yang bersangkutan.

Heuristic search adalah suatu istilah yang berasal dari bahasa Yunani yang berarti menemukan/ menyingkap. *Heuristic* adalah suatu perbuatan yang membantu kita menemukan jalan dalam pohon pelacakan yang menuntut kita kepada suatu solusi masalah. *Heuristic* dapat diartikan juga sebagai suatu kaidah yang merupakan metoda/ prosedur yang didasarkan kepada pengalaman dan praktek, syarat, trik atau bantuan lainnya yang membantu mempersempit dan memfokuskan proses pelacakan kepada suatu tujuan tertentu.

Heuristic dapat digunakan pada beberapa kondisi berikut ini :

1. Mengatasi *combinatorial explosion*

   Ada masalah yang kemungkinan arah penyelesaiannya berkembang pesat (bersifat faktorial) sehingga menimbulkan *combinatorial explosion*. *Heuristic* merupakan cara untuk menentukan kemungkinan arah penyelesaian masalah secara efisien.

2. Solusi paling optimal mungkin tidak diperlukan

   Dalam suatu keadaan, mungkin lebih baik mendapatkan solusi yang mendekati optimal dalam waktu yang singkat daripada solusi yang paling optimal dalam waktu yang lama.

3. Pada umumnya hasilnya cukup baik

   Sekalipun tidak optimal biasanya mendekati optimal.

4. Membantu pemahaman bagi orang yang menyelesaikan persoalan

   Banyak alternatif *heuristic* yang dapat diterapkan dalam suatu percobaan. Orang yang menyelesaikan persoalan tersebut akan lebih mengerti persoalannya jika mencoba *heuristic* yang diterapkannya.

Metode-metode yang termasuk ke dalam teknik pencarian yang berdasarkan pada fungsi heuristik adalah : *Generate and Test, Hill Climbing, Best First Search, Simulated Annealing.*

2.4.2.1 Generate and Test

Strategi bangkitkan dan uji (*generate and test*) merupakan pendekatan yang paling sederhana dari semua pendekatan yang akan dibicarakan. Pendekatan ini meliputi langkah–langkah sebagai berikut :

1. Buatlah/ bangkitkan sebuah solusi yang memungkinkan. Untuk sebuah problema hal ini dapat berarti pembuatan sebuah titik khusus dalam ruang problema.

2. Lakukan pengujian untuk melihat apakah solusi yang dibuat benar–benar merupakan sebuah solusi, dengan cara membandingkan titik khusus tersebut dengan *goal*-nya (solusi).

3. Jika telah diperoleh sebuah solusi, langkah– langkah tersebut dapat dihentikan. Jika belum kembalilah ke langkah pertama.

Jika pembangkitan atau pembuatan solusi– solusi yang dimungkinkan dapat dilakukan secara sistematis, maka prosedur ini akan dapat segera menemukan solusinya (bila ada). Namun, jika ruang problema sangat besar, maka proses ini akan membutuhkan waktu yang lama.

Metode *generate and test* ini kurang efisien untuk masalah yang besar atau kompleks.

2.4.2.2 Hill Climbing

Hill climbing (mendaki bukit) merupakan salah satu variasi metode buat dan uji (*generate and test*) dimana umpan balik yang berasal dari prosedur uji digunakan untuk memutuskan arah gerak dalam ruang pencarian (*search*). Dalam prosedur buat dan uji yang murni, respon fungsi uji hanyalah *ya* atau *tidak*. Dalam prosedur *Hill Climbing*, fungsi uji dikombinasikan dengan fungsi heuristik yang menyediakan pengukuran kedekatan suatu keadaan yang diberikan dengan tujuan (*goal*).

Prosedur *Hill Climbing* :

1. Buatlah solusi usulan pertama dengan cara yang sama seperti yang dilakukan dalam prosedur buat dan uji (*generate and test*). Periksalah apakah solusi usulan itu merupakan sebuah solusi. Jika ya, berhentilah. Jika tidak, kita lanjutkan ke langkah berikutnya.
2. Dari solusi ini, terapkan sejumlah aturan yang dapat diterapkan untuk membuat sekumpulan solusi usulan yang baru.

Untuk setiap elemen kumpulan solusi tersebut, lakukanlah hal-hal berikut ini :

1. Kirimkanlah elemen ini ke fungsi uji. Jika elemen ini merupakan sebuah solusi, berhentilah.
2. Jika tidak, periksalah apakah elemen ini merupakan yang terdekat dengan solusi yang telah diuji sejauh ini. Jika tidak, buanglah.
3. Ambilah elemen terbaik yang ditemukan di atas dan pakailah sebagai solusi usulan berikutnya. Langkah ini bersesuaian dengan langkah dalam ruang problema dengan arah yang muncul sebagai yang tercepat dalam mencapai tujuan.
4. Kembalilah ke langkah 2.

Masalah-masalah yang mungkin timbul pada prosedur *Hill Climbing* :

- *Maksimum lokal* adalah suatu keadaan yang lebih baik daripada semua tetangganya namun masih belum lebih baik dari suatu keadaan lain yang jauh letaknya darinya.
- *Daratan* (*Plateau*) adalah suatu daerah datar dari ruang pencarian (*search*) dimana semua himpunan keadaan tetangganya memiliki nilai yang sama.
- *Punggung (Ridge)* adalah suatu daerah ruang pencarian (*search*) yang lebih tinggi daripada daerah sekitarnya, namun tidak dapat dibalikkan oleh langkah–langkah tunggal ke arah manapun.

Solusinya:

- Melakukan langkah balik (*backtracking*) ke simpul yang lebih awal dan mencoba bergerak ke arah yang lain.
- Melakukan lompatan besar ke suatu arah untuk mencoba bagian ruang pencarian yang baru.
- Menerapkan dua atau lebih aturan sebelum melakukan uji coba. Ini bersesuaian dengan bergerak ke beberapa arah sekaligus.

2.4.2.3 Best First Search

Pencarian terbaik pertama (*Best First Search*) merupakan suatu cara yang menggabungkan keuntungan atau kelebihan dari pencarian *Breadth-First Search* dan *Depth-First Search*.

Pada setiap langkah proses pencarian terbaik pertama, kita memilih node-node dengan menerapkan fungsi heuristik yang memadai pada setiap node/simpul yang kita pilih dengan menggunakan aturan-aturan tertentu untuk menghasilkan penggantinya.

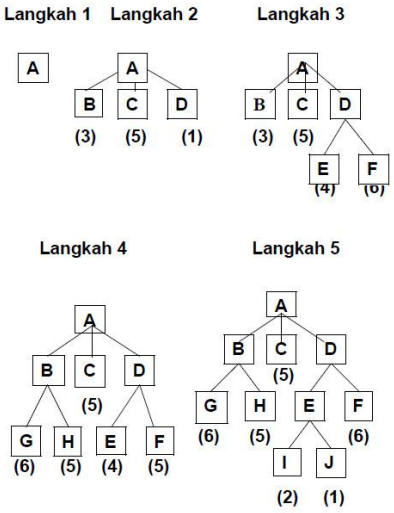
Fungsi Heuristik yang digunakan merupakan prakiraan (estimasi) *cost* dari *initial state* ke *goal state*, yang dinyatakan dengan **[2]** :

$$\mathbf{f'(n) = g(n) + h'(n)} \quad \text{..... (i)}$$

f'(n) = prakiraan *cost* dari *initial* ke *goal*

g(n) = *cost* dari *initial state* ke *current state*

h’(n) = prakiraan *cost* dari *current state* ke *goal state*



Gambar 2.6. Ilustrasi Pencarian Terbaik Pertama *(Best-First Search)* [2]

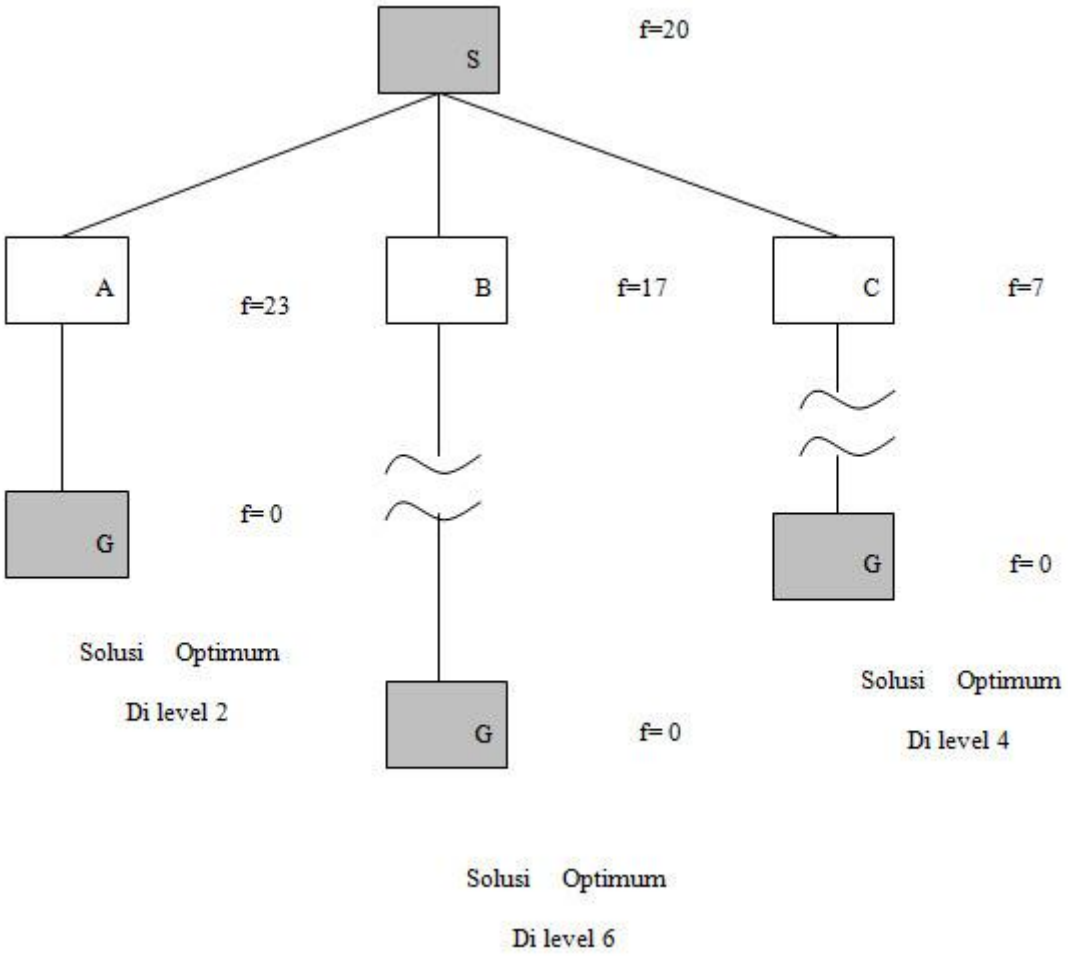
2.4.2.4 Simulated Annealing

Simulated Annealing (SA) memanfaatkan analogi cara pendinginan dan pembekuan metal menjadi sebuah struktur Kristal dengan energy minimal. Dengan Probabilitas tertentu SA mungkin bisa keluar dari jebakan *Local Minimum.* SA menggunakan sebuah rumus probabilitas yang memungkinkannya bisa keluar dari *Local Minimum.* Ketika *new state* tidak lebih baik dari *current state*, maka *new state* tersebut masih mungkin dipilih dengan probabilitas sebagai berikut **[4]** :

$$p(\Delta E)=e^{-\Delta E/T} \quad \text{.....(ii)}$$

Fungsi tersebut diadopsi dari ilmu fisika, dimana fungsi ini merepresentasikan distribusi Boltzman dari energy dalam sistem termodinamik,

sehingga didapat persamaan probabilitas dari level energi yang diberikan dalam sistem pada temperature *T* **[2]**.



Gambar 2.7. SA [2]

Pada level 1, SA memilih A meskipun nilai f-nya lebih besar daripada nilai f pada simpul S, dengan probabilitas tertentu. Akhirnya , SA berhasil menemukan solusi optimum di level 2.

2.5 Rekaya Perangkat Lunak

2.5.1 Pengertian Rekayasa Perangkat Lunak

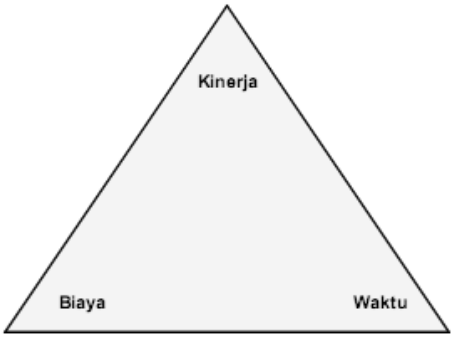
Istilah Reakayasa Perangkat Lunak (RPL) secara umum disepakati sebagai terjemahan dari istilah *Software engineering*. Istilah *Software Engineering* mulai dipopulerkan pada tahun 1968 pada software *engineering Conference* yang diselenggarakan oleh NATO. Sebagian orang mengartikan RPL hanya sebatas pada bagaimana membuat program komputer. Padahal ada perbedaan yang mendasar antara perangkat lunak (*software*) dan program komputer.

Perangkat lunak adalah seluruh perintah yang digunakan untuk memproses informasi. Perangkat lunak dapat berupa program atau prosedur. Program adalah kumpulan perintah yang dimengerti oleh komputer sedangkan prosedur adalah perintah yang dibutuhkan oleh pengguna dalam memproses informasi (O'Brien, 1999).

Pengertian RPL sendiri adalah suatu disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal yaitu analisa kebutuhan pengguna, menentukan spesifikasi dari kebutuhan pengguna, disain, pengkodean, pengujian sampai pemeliharaan sistem setelah digunakan. Dari pengertian ini jelaslah bahwa RPL tidak hanya berhubungan dengan cara pembuatan program komputer. Pernyataan "semua aspek produksi" pada pengertian di atas, mempunyai arti semnua hal yang berhubungan dengan proses produksi seperti manajemen proyek, penentuan personil, anggaran biaya, metode, jadwal, kualitas sampai dengan pelatihan pengguna merupakan bagian dari RPL.

2.5.2 Tujuan Rekayasa Perangkat Lunak

Secara umum tujuan RPL tidak berbeda dengan bidang rekayasa yang lain. Hal ini dapat kita lihat pada Gambar di bawah ini.



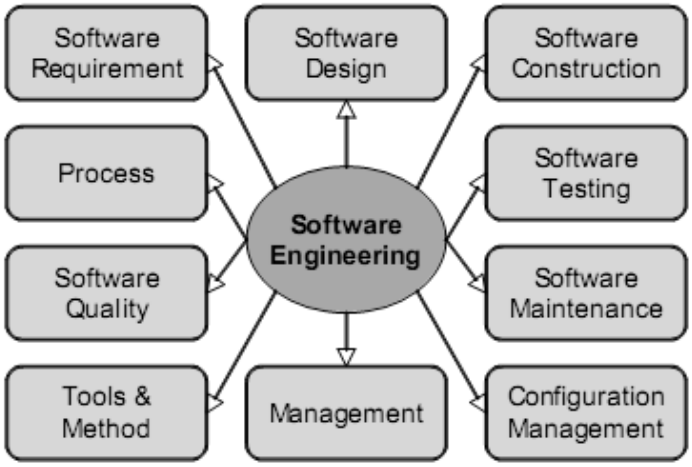
Gambar 2.8. Tujuan RPL(*Abran et.al.*, 2004).

Dari Gambar di atas dapat diartikan bahwa bidang rekayasa akan selalu berusaha menghasilkan *output* yang kinerjanya tinggi, biaya rendah dan waktu penyelesaian yang tepat. Secara lebih khusus kita dapat menyatakan tujuan RPL adalah:

a. memperoleh biaya produksi perangkat lunak yang rendah.
b. menghasilkan pereangkat lunak yang kinerjanya tinggi, handal dan tepat waktu.
c. menghasilkan perangkat lunak yang dapat bekerja pada berbagai jenis *platform*.
d. menghasilkan perangkat lunak yang biaya perawatannya rendah.

2.5.3 Ruang Lingkup Rekayasa Perangkat Lunak

Ruang lingkup RPL dapat digambarkan sebagai berikut:



Gambar 2.9 Ruang lingkup RPL (*Abran et.al*., 2004).

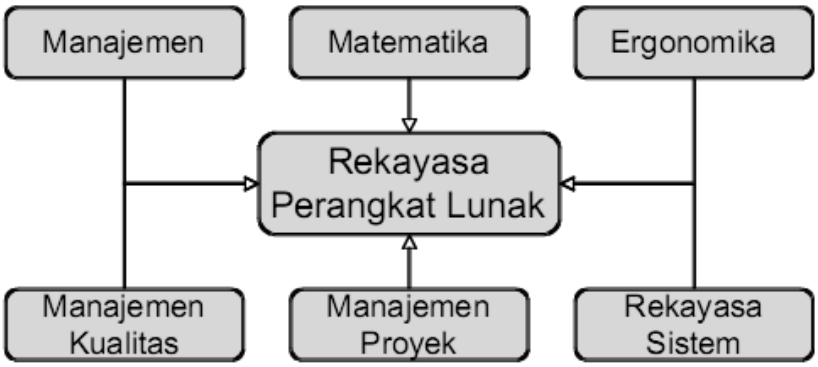
- ***software Requirements*** berhubungan dengan spesifikasi kebutuhan dan persyaratan perangkat lunak .
- *software* desain mencakup proses penampilan arsitektur, komponen, antar muka, dan karakteristik lain dari perangkat lunak.
- ***software construction*** berhubungan dengan detail pengembangan perangkat lunak, termasuk algoritma, pengkodean, pengujian dan pencarian kesalahan.
- ***software testing*** meliputi pengujian pada keseluruhan perilaku perangkat lunak.
- ***software maintenance*** mencakup upaya-upaya perawatan ketika perangkat lunak telah dioperasikan.

- ***software configuration*** management berhubungan dengan usaha perubahan konfigurasi perangkat lunak untuk memenuhi kebutuhan tertentu.
- ***software engineering management*** berkaitan dengan pengelolaan dan pengukuran RPL, termasuk perencanaan proyek perangkat lunak
- ***software engineering tools and*** methods mencakup kajian teoritis tentang alat bantu dan metode RPL
- ***software engineering process*** berhubungan dengan definisi, implementasi pengukuran, pengelolaan, perubahan dan perbaikan proses RPL
- ***software quality*** menitik beratkan pada kualitas dan daur hidup perangkat lunak

2.5.4 Rekayasa Perangkat Lunak Dan Disiplin Ilmu Lain

Cakupan ruang lingkup yang cukup luas, membuat RPL sangat terkait dengan disiplin dengan bidang ilmu lain. tidak saja sub bidang dalam disiplin ilmu komputer namun dengan beberapa disiplin ilmu lain diluar ilmu komputer.

Hubungan keterkaitan RPL dengan ilmu lain dapat dilihat pada gambar dibawah ini



Gambar 2.10. Keterkaitan RPL dengan bidang ilmu lain (*Denning*, 2000)

- Bidang ilmu manajemen meliputi akuntansi, finansial, pemasaran, manajemen operasi, ekonomi, analisis kuantitatif, manajemen sumber daya manusia, kebijakan, dan strategi bisnis
- Bidang ilmu matematika meliputi aljabar linier, kalkulus, peluang, statistik, analisis numerik, dan matematika diskrit
- Bidang ilmu manajemen proyek meliputi semua hal yang berkaitan dengan proyek, seperti ruang lingkup proyek, anggaran, tenaga kerja, kualitas, manajemen resiko dan keandalan, perbaikan kualitas, dan metode-metode kuantitatif
- Bidang ilmu ergonomika menyangkut hubungan (interaksi) antar manusia dengan komponen-komponen lain dalam sistem komputer
- Bidang ilmu rekayasa sistem meliputi teori sistem, analisis biaya-keuntungan, pemodelan, simulasi, proses, dan operasi bisnis

2.5.5 Perkembangan Rekayasa Perangkat Lunak

Meskipun baru dicetuskan pada tahun 1968, namun RPL telah memiliki sejarah yang cukup yang panjang. Dari sisi disiplin ilmu, RPL masih reklatif muda dan akan terus berkembang.

Arah perkembangan yang saat ini sedang dikembangkan antara lain meliputi :

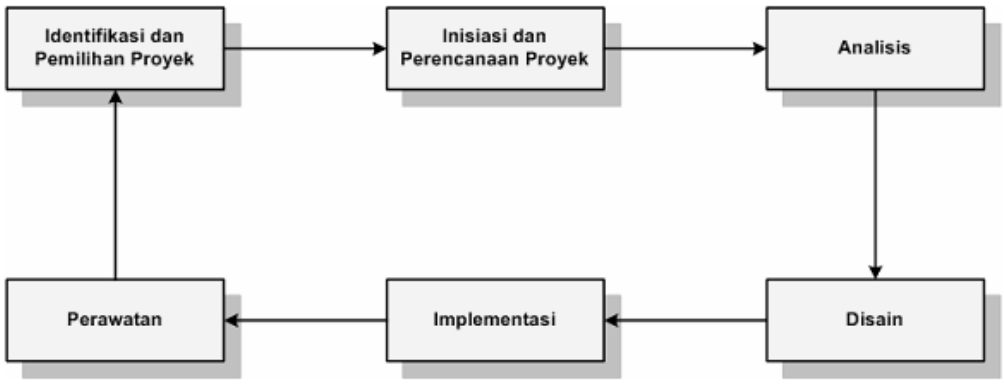
Tabel 2.2. Perkembangan Rekayasa Perangkat Lunak

Tahun	Kejadian
1940an	Komputer pertama yang membolehkan pengguna menulis kode program langsung
1950an	Generasi awal interpreter dan bahasa macro Generasi pertama *compiler*
1960an	Generasi kedua *compiler* Komputer *mainframe* mulai dikomersialkan Pengembangan perangkat lunak pesanan Konsep *Software Engineering* mulai digunakan
1970an	Perangkat pengembang perangkat lunak Perangkat *minicomputer* komersial
1980an	Perangkat Komputer Personal (PC) komersial Peningkatan permintaan perangkat lunak

1990an	Pemrograman berorientasi obyek (OOP) *Agile Process* dan *Extreme Programming* Peningkatan drastis kapasitas memori Peningkatan penggunaan internet
2000an	Platform interpreter modern (Java, .Net, PHP, dll) *Outsourcing*

2.5.6 Metode Rekayasa Perangkat Lunak

Pada rekayasa perangkat lunak, banyak model yang telah dikembangkan untuk membantu proses pengembangan perangkat lunak. Model-model ini pada umumnya mengacu pada model proses pengembangan sistem yang disebut ***System Development Life Cycle (SDLC)*** seperti terlihat pada Gambar berikut ini :



Gambar 2.11. *System Development Life Cycle* (SDLC). (*Whitten et al, 2004*)

- Kebutuhan terhadap definisi masalah yang jelas. *Input* utama dari setiap model pengembangan perangkat lunak adalah pendefinisian masalah yang jelas. Semakin jelas
- Tahapan-tahapan pengembangan yang teratur. Meskipun model-model pengembangan perangkat lunak memiliki pola yang berbeda-beda, biasanya model-model tersebut mengikuti pola umum *analysis – design – coding – testing – maintenance.*
- *Stakeholder* berperan sangat penting dalam keseluruhan tahapan pengembangan. *Stakeholder* dalam rekayasa perangkat lunak dapat berupa pengguna, pemilik, pengembang, pemrogram dan orang-orang yang terlibat dalam rekayasa perangkat lunak tersebut.
- Dokumentasi merupakan bagian penting dari pengembangan perangkat lunak. Masing-masing tahapan dalam model biasanya menghasilkan sejumlah tulisan, diagram, gambar atau bentuk-bentuk lain yang harus didokumentasi dan merupakan bagian tak terpisahkan dari perangkat lunak yang dihasilkan.
- Keluaran dari proses pengembangan perangkat lunak harus bernilai ekonomis. Nilai dari sebuah perangkat lunak sebenarnya agak susah di-rupiah-kan. Namun efek dari penggunaan perangkat lunak yang telah dikembangkan haruslah memberi nilai tambah bagi organisasi. Hal ini dapat berupa penurunan biaya operasi, efisiensi penggunaan sumberdaya, peningkatan keuntungan organisasi, peningkatan "*image*" organisasi dan lain-lain.

2.5.7 Tahapan Rekayasa Perangkat Lunak

Meskipun dalam pendekatan berbeda-beda, namun model-model pendekatan memiliki kesamaan, yaitu menggunaka pola tahapan *analysis – design – coding(construction) – testing – maintenance.*

1. **Analisis sistem** adalah sebuah teknik pemecahan masalah yang menguraikan sebuah sistem menjadi komponen-komponennya dengan tujuan mempelajari seberapa bagus komponen-komponen tersebut bekerja dan berinteraksi untuk meraih tujuan.

   Analisis mungkin adalah bagian terpenting dari proses rekayasa perangkat lunak. Karena semua proses lanjutan akan sangat bergantung pada baik tidaknya hasil analisis. Ada satu bagian penting yang biasanya dilakukan dalam tahapan analisis yaitu pemodelan proses bisnis.

2. **Model proses** adalah model yang memfokuskan pada seluruh proses di dalam sistem yang mentransformasikan data menjadi informasi (Harris, 2003). Model proses juga menunjukkan aliran data yang masuk dan keluar pada suatu proses. Biasanya model ini digambarkan dalam bentuk Diagram Arus Data (*Data Flow Diagram*/ DFD). DFD meyajikan gambaran apa yang manusia, proses dan prosedur lakukan untuk mentransformasi data menjadi informasi.
3. **Desain perangkat lunak** adalah tugas, tahapan atau aktivitas yang difokuskan pada spesifikasi detil dari solusi berbasis komputer. Disain perangkat lunak sering juga disebut sebagai *physical design*. Jika tahapan

analisis sistem menekankan pada masalah bisnis *(business rule)*, maka sebaliknya disain perangkat lunak fokus pada sisi teknis dan implementasi sebuah perangkat lunak.

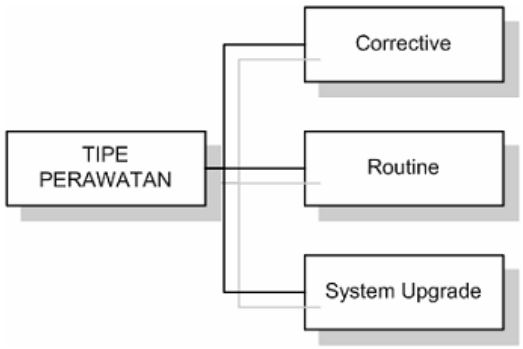
Output utama dari tahapan disain perangkat lunak adalah spesifikasi disain. Spesifikasi ini meliputi spesifikasi disain umum yang akan disampaikan kepada *stakeholder* sistem dan spesifikasi disain rinci yang akan digunakan pada tahap implementasi. Spesifikasi disain umum hanya berisi gambaran umum agar stakeholder sistem mengerti akan seperti apa perangkat lunak yang akan dibangun. Biasanya diagram USD tentang perangkat lunak yang baru merupakan *point* penting dibagian ini. Spesifikasi disain rinci atau kadang disebut disain arsitektur rinci perangkat lunak diperlukan untuk merancang sistem sehingga memiliki konstruksi yang baik, proses pengolahan data yang tepat dan akurat, bernilai, memiliki aspek *user friendly* dan memiliki dasar-dasar untuk pengembangan selanjutnya.

Desain arsitektur ini terdiri dari desain *database*, desain proses, desain *user* interface yang mencakup desain *input*, *output form* dan *report*, desain *hardware, software* dan jaringan. Desain proses merupakan kelanjutan dari pemodelan proses yang dilakukan pada tahapan analisis.

4. **Konstruksi** adalah tahapan menerjemahkan hasil disain logis dan fisik ke dalam kode-kode program komputer.
5. **Pengujian** sistem melibatkan semua kelompok pengguna yang telah direncanakan pada tahap sebelumnya. Pengujian tingkat penerimaan

terhadap perangkat lunak akan berakhir ketika dirasa semua kelompok pengguna menyatakan bisa menerima perangkat lunak tersebut berdasarkan kriteria-kriteria yang telah ditetapkan.

6. **Perawatan dan Konfigurasi.** Ketika sebuah perangkat lunak telah dianggap layak untuk dijalankan, maka tahapan baru menjadi muncul yaitu perawatan perangkat lunak. Ada beberapa tipe perawatan yang biasa dikenal dalam dunia perangkat lunak seperti terlihat pada diagram di Gambar di bawah ini :



Gambar 2.12. Tipe-tipe perawatan (*Whitten et al, 2004*).

- Tipe perawatan *corrective* dilakukan jika terjadi kesalahan atau biasa dikenal sebagai bugs. Perawatan bisa dilakukan dengan memperbaiki kode program, menambah bagian yang dirasa perlu atau malah menghilangkan bagian-bagian tertentu.
- Tipe perawatan *routine* biasa juga disebut *preventive maintenance* dilakukan secara rutin untuk melihat kinerja perangkat lunak ada atau tidak ada kesalahan.

- Tipe perawatan sistem *upgrade* dilakukan jika ada perubahan dari komponen-komponen yang terlibat dalam perangkat lunak tersebut. Sebagai contoh perubahan *platform* sistem operasi dari versi lama ke versi baru menyebabkan perangkat lunak harus diupgrade.

2.6 Microsoft Visual Basic 6.0

Microsoft Visual Basic (sering disingkat sebagai VB saja) merupakan sebuah bahasa pemrograman yang bersifat *event driven* dan menawarkan *Integrated Development Environment* (IDE) visual untuk membuat program aplikasi berbasis sistem operasi *Microsoft Windows* dengan menggunakan model pemrograman *Common Object Model* (COM) **[5].** .

Visual Basic merupakan turunan bahasa *BASIC* dan menawarkan pengembangan aplikasi komputer berbasis grafik dengan cepat, akses ke basis data menggunakan *Data Access Objects (DAO), Remote Data Objects (RDO*), atau *ActiveX Data Object (ADO),* serta menawarkan pembuatan kontrol *ActiveX* dan objek *ActiveX*. Beberapa bahasa skrip seperti *Visual Basic for Applications (VBA)* dan *Visual Basic Scripting Edition (VBScript),* mirip seperti halnya *Visual Basic*, tetapi cara kerjanya yang berbeda **[5].**

Para programmer dapat membangun aplikasi dengan menggunakan komponen-komponen yang disediakan oleh *Microsoft Visual Basic* Program-program yang ditulis dengan *Visual Basic* juga dapat menggunakan *Windows API*, tapi membutuhkan deklarasi fungsi eksternal tambahan. Dalam pemrograman untuk bisnis, *Visual Basic* memiliki pangsa pasar yang sangat luas. Dalam sebuah

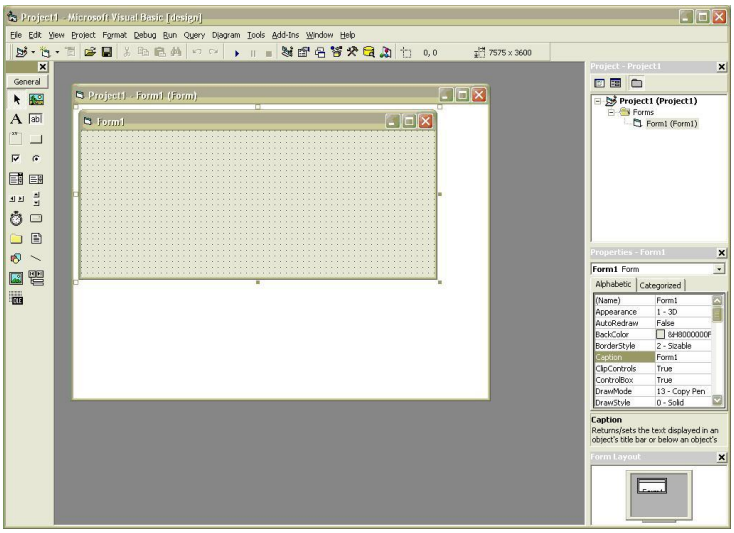
survey yang dilakukan pada tahun 2005, 62% 6 pengembang perangkat lunak dilaporkan menggunakan berbagai bentuk *Visual Basic*, yang diikuti oleh *C++, JavaScript, C#, dan Java.*

Bill Gates, pendiri *Microsoft*, memulai bisnis softwarenya dengan mengembangkan interpreter bahasa *Basic untuk Altair 8800*, untuk kemudian ia ubah agar dapat berjalan di atas IBM PC dengan sistem operasi DOS. Perkembangan berikutnya ialah diluncurkannya *BASICA (basic-advanced)* untuk DOS. Setelah *BASICA, Microsoft* meluncurkan *Microsoft QuickBasic dan Microsoft Basic* (dikenal juga sebagai *Basic Compiler*).

Sejarah BASIC di tangan *Microsoft* sebagai bahasa yang diinterpretasi (BASICA) dan juga bahasa yang dikompilasi (BASCOM) membuat *Visual Basic* diimplementasikan sebagai gabungan keduanya. Programmer yang menggunakan *Visual Basic* bisa memilih kode terkompilasi atau kode yang harus diinterpretasi sebagai hasil *executable* dari kode VB. Sayangnya, meskipun sudah terkompilasi jadi bahasa mesin, DLL bernama MSVBVMxx.DLL tetap dibutuhkan. Namun karakteristik bahasa terkompilasi tetap muncul.

Visual Basic merupakan bahasa yang mendukung OOP, namun tidak sepenuhnya. Beberapa karakteristik obyek tidak dapat dilakukan pada *Visual Basic*, seperti *Inheritance* tidak dapat dilakukan *pada class module*. *Polymorphism* secara terbatas bisa dilakukan dengan mendeklarasikan *class module* yang memiliki *Interface* tertentu. *Visual Basic (VB)* tidak bersifat *case* sensitif.

Visual Basic menjadi populer karena kemudahan desain form secara visual dan adanya kemampuan untuk menggunakan komponen-komponen *ActiveX* yang dibuat oleh pihak lain. Namun komponen *ActiveX* memiliki masalahnya tersendiri yang dikenal sebagai *DLL hell*. Pada *Visual Basic .NET, Microsoft* mencoba mengatasi masalah *DLL hell* dengan mengubah cara penggunaan komponen (menjadi independen terhadap *registry*).



Gambar 2.13. Tampilan program *Visual Basic*