BAB II

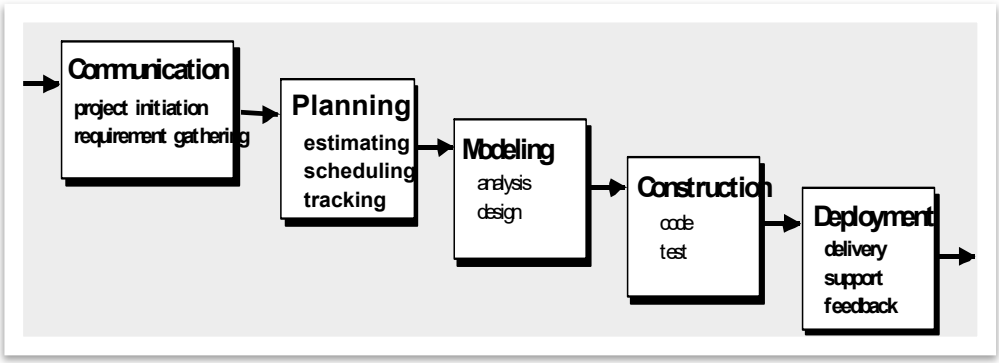
TELAAH LITERATUR

A. Rekayasa Piranti Lunak

Istilah Rekayasa Perangkat Lunak (RPL) secara umum disepakati sebagai terjemahan dari istilah *Software Engineering*. Istilah *Software Engineering* mulai dipopulerkan 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 (Mulyanto, 2008).

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, desain, pengkodean, pengujian sampai pemeliharaan sistem setelah digunakan (Mulyanto, 2008).

Dalam praktik rekayasa piranti lunak terdapat banyak model yang digunakan untuk membantu proses pengembangan piranti lunak. Salah satunya adalah *Waterfall Model*. *Waterfall Model* sering juga dikenal sebagai model yang linear menggunakan pendekatan secara sekuensial dan sistematis untuk mengembangkan piranti lunak yang dimulai dengan *system requirement*, kemudian *analysis, design, coding, testing* dan *support* (Pressman, 2001).



Gambar 2.1 *Waterfall Model* (Pressman,2001)

B. Nada dan *Equal Tempered Scale*

Getaran mekanik yang ditangkap oleh indera pendengaran kita dan dikirimkan ke otak adalah bunyi. Bunyi yang terdengar secara beraturan disebut dengan nada (Ronbo2007, 2007).

Bagi indera pendengaran manusia, frekuensi yang dibedakan oleh faktor dua akan terdengar mirip. Sebagai contoh sebuah nada dengan frekuensi 440Hz (frekuensi dari nada A) akan terdengar miripi dengan

nada 220Hz dan 880Hz, namun akan terdengar lebih tinggi atau rendah. Frekuensi yang berbeda dikelompokkan dalam satu kategori yang sama, dan kategori ini direpresentasikan dalam bentuk simbol nada. Jadi frekuensi 220, 440, dan 880Hz akan dikategorikan dalam nada A. Perbedaan tinggi rendah frekuensi nada yang memiliki jarak setengah atau dua kali dari frekuensi itu sendiri dalam istilah musik dikenal dengan oktaf (vischalmozart, 2010).

Kita mengenal 7 nada dasar yaitu C, D, E, F, G, A, B. Pada masing-masing nada tersebut memiliki jarak antara dua nada yang berdekatan disebut interval. Nada C−D, D−E, F−G, G−A, dan A−B memiliki jarak sebesar 1, sedangkan nada E−F dan B−C memiliki jarak sebesar 1/2. Di antara dua nada yang berjarak 1 terdapat sebuah nada yang berjarak 1/2 dari kedua nada yang mengapitnya, yang diberi notasi #, sehingga terdapat nada C#, D#, F#, G#, dan A# (Worner, 2008).

Nada-nada tersebut membentuk dua belas nada dalam skala kromatik yang dikenal dalam dunia musik sekarang (Worner, 2008). Secara matematis masing-masing nada tersebut memiliki hubungan satu dengan yang lainnya. Sehingga terdapat formula untuk dapat menentukan frekuensi dari tiap-tiap nada. Rumus untuk menentukan frekuensi dari nada adalah

$$f_n = f_0 * (a)^n \quad \text{.....(2.1)}$$

- f_0 adalah frekuensi yang telah ditentukan. Misal nada A = 440Hz.

- n adalah selisih langkah dari nada yang diinginkan ke nada yang dijadikan acuan. Jika nada yang dicari lebih tinggi dari nada acuan maka nilai n positif sebaliknya bila nada yang dicari lebih rendah maka n bernilai negatif. Misal kita ingin mencari nada B maka dibutuhkan dua langkah dari nada A yaitu A-A#-B. n = +2.
- $a = (2)^{1/12} = 1.059463$

untuk mencari frekuensi dari nada B maka kita masukan nilai yang didapat dalam rumus

$fn = 440 * (1.059463)^2 \approx 493.88Hz$

R.Nave(2010) mengatakan bahwa *equal tempered scale* adalah *musical scale* yang dipakai secara umum pada saat ini, digunakan untuk proses penalaan alat musik seperti piano dan gitar.

Mengutip dari penuturan Yuval Nov, *Equal tempered scale* membagi satu oktaf kedalam 12 *semitones*. Semitones adalah sebutan untuk nada yang memiliki jarak setengah terhadap tetangganya. Dalam *equal tempered scale* dua nada dikatakan semitone bila memiliki ratio antara frekuensinya sebesar 1.059463.

Berikut adalah Berikut adalah tabel frekuensi nada pada oktaf ke 4.

Tabel 2.1 Tabel Frekuensi Nada Oktaf 4.

NADA	INTERVAL	FREKUENSI	CENTS
C	½	261.62Hz	0
C#	½	277.18Hz	100
D	½	293.66Hz	200
D#	½	311.12Hz	300
E	½	329.62Hz	400
F	½	349.22Hz	500
F#	½	396.99Hz	600
G	½	391.99Hz	700
G#	½	415.30Hz	800
A	½	440Hz	900
A#	½	466.16Hz	1000
B	½	493.88Hz	1100

Masing-masing jarak dari *semitones* dibagi lagi kedalam unit yang disebut *cents*. Untuk satu interval semitones terdapat seratus unit *cents* berarti dalam setiap satu oktaf terdapat 1200 unit *cents.* Dengan adanya unit pengukuran *cents* dapat dicari toleransi dari tiap-tiap nada (Campbell, 1997).

Berikut adalah rumus untuk mencari toleransi antar nada.

$fn = \{ [(2) \text{ ^ } 1/1200] \text{ ^ } n \} * fo$ .....(2.2)

- fn adalah frekuensi yang dicari.

- n adalah interval *cents*. Misal kita ingin mencari batas bawah atas bawah dan atas nada A4 maka nilai n yang dimasukkan adalah 850 untuk batas bawah dan 950 untuk batas atas.
- fo adalah frekuensi dari nada *unison* dalam satu range oktaf. Contohnya C4 adalah nada *unison* pada oktaf ke 4.

untuk batas bawah nada A4 adalah

$fn = \{ [(2) \text{ ^ } 1/1200] \text{ ^ } 850 \} * 261.62\text{Hz}$

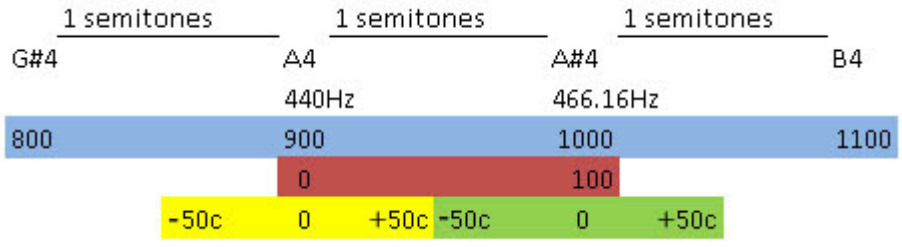
$fn = \{ [1.000577789275] \text{ ^ } 850 \} * 261.62\text{Hz} \approx 427.47\text{Hz}$

untuk batas atas nada A4 adalah

$fn = \{ [(2) \text{ ^ } 1/1200] \text{ ^ } 950 \} * 261.62\text{Hz}$

$fn = \{ [1.000577789275] \text{ ^ } 950 \} * 261.62\text{Hz} \approx 452.88\text{Hz}$

Jadi *range* untuk nada A4 adalah dari 427.47Hz sampai dengan 452.88Hz



Gambar 2.2 Batas frekuensi nada

C. *Fourier Transform*

Carl Freidrich Gauss pada tahun 1805 mencoba menggambarkan orbit dari asteroid tertentu dari lokasi sampel. Kemudian ia mengembangkan *Fast Fourier Transform* bahkan sebelum Joseph Fourier mempublikasikan transformasi *Fourier*-nya pada tahun 1822 (Worner, 2008).

Transformasi *Fourier* digunakan untuk mentransformasi sinyal waktu kontinyu ke dalam *domain* frekuensi. Agar transformasi fourier dapat digunakan dalam operasi digital, maka diperlukan proses mencuplik sinyal menjadi data sampel. Oleh karena itu digunakan *Discrete Fourier Transform*. Masukan *Discrete Fourier Transform* adalah urutan terbatas bilangan riil ataupun bilangan kompleks. Hal ini menyebabkan *Discrete Fourier Transform* ideal untuk memproses informasi di dalam komputer (Prihartoni, 2010).

Dalam perkembangannya, para peneliti terus berupaya mengembangkan suatu algoritma yang lebih cepat. Dibutuhkan 160 tahun sampai akhirnya *Fast Fourier Transform* kembali ditemukan pada tahun 1965 oleh J.W.Cooley dan John Tukey. Stefan Worner (2008) mengatakan dari tahun 1805 sampai 1965, tidak ada satu ilmuwan pun yang mampu mengembangkan efisiensi dalam DFT serinci Gauss dan Cooley-Tukey.

Bentuk transformasi Fourier *F(x)* dari fungsi kontinyu dengan satu variabel inputan *f(t)*

$$F(x) = \int_{-\sim}^{\sim} f(t)e^{-j2\pi ft}dt$$

.....(2.3)

Fungsi inversenya adalah

$$f(t) = \int_{-\sim}^{\sim} F(x)e^{j2\pi ft}df$$

.....(2.4)

1. Discrete Fourier Transform

Richard G.Lyons (2004) mendefinisikan *Discrete Fourier Transform* sebagai proses yang kuat yang digunakan dalam pemrosesan sinyal digital dan filterisasi digital. *Discrete Fourier Transform* menungkinkan seseorang untuk menganalisa, memanipulasi dan mensintesis sinyal yang tidak mungkin dapat dilakukan dalam pemrosesan sinyal analog. *Discrete Fourier Transform* merupakan gambaran karakteristik spektrum periodik dari suatu sampel data. *Discrete Fourier Transform* memiliki spektrum garis yang mewakili periode sekuensial N.

Inputan dari *Discrete Fourier Transform* adalah *finite sequence* dari bilangan kompleks dan real, membuat *Discrete Fourier Transform*

memungkinkan informasi hasil prosesnya dapat disimpan di komputer (Movellan, 2009).

Berikut adalah persamaan dari *Discrete Fourier Transform*

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}$$

.....(2.5)

Dimana

- x[n] adalah fungsi waktu dalam bentuk diskrit
- X[k] adalah transformasi fouriernya.
- n = 0, 1, 2, ... N–1.
- k = 0, 1, 2, ... N–1.
- W = $e^{-j2\pi}$

Penjabaran secara detail persamaan di atas dapat dilihat pada matrix di bawah ini.

$$\begin{bmatrix} [W_4^{00}][W_4^{10}][W_4^{20}][W_4^{30}] \\ [W_4^{01}][W_4^{11}][W_4^{21}][W_4^{31}] \\ [W_4^{02}][W_4^{12}][W_4^{22}][W_4^{32}] \\ [W_4^{03}][W_4^{13}][W_4^{23}][W_4^{33}] \end{bmatrix} \begin{bmatrix} [x[0]] \\ [x[1]] \\ [x[2]] \\ [x[3]] \end{bmatrix} = \begin{bmatrix} [X[0]] \\ [X[1]] \\ [X[2]] \\ [X[3]] \end{bmatrix}$$

Dengan jumlah N = 4, perkalian dan penjumlahan *Discrete Fourier Transform* berjumlah masing-masing enam belas. Oleh karena itu disebutkan bahwa DFT memerlukan jumlah operasi sebanyak N^2.

2. Fast Fourier Transform

Fast Fourier Transform adalah algoritma yang lebih efisien dalam menghitung Discrete Fourier Transform. Ditemukan pertama kali oleh Carl Freidrich Gauss pada tahun 1805 dan ditemukan kembali oleh J. W. Cooley dan John Tukey pada tahun 1965. Stefan Worner (2008) menjelaskan pada saat itu militer Amerika tertarik pada metode untuk mendeteksi tes nuklir dari Soviet. Salah satu pendekatannya adalah dengan menganalisis sekumpulan data *seismological* dan Tukey adalah orang yang menangani masalah itu.

Ide dari *Fast Fourier Transform* adalah mengubah suatu bentuk *Discrete Fourier Transform* dengan panjang N menjadi bentuk penjumlahan dari dua buah bentuk *Discrete Fourier Transform* dengan panjang masing-masing N/2, satu bagian terdiri dari elemen bernomor ganjil sementara yang lain genap (Prihartoni, 2010).

Berikut adalah persamaan dari Fast Fourier Transform.

$$X[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r]W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]W_N^{(2r+1)k} \quad \text{.....(2.6)}$$

n genap n ganjil

$$= \sum_{r=0}^{\frac{N}{2}-1} x[2r] W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] W_N^{(2r+1)k}$$

$$= \sum_{r=0}^{\frac{N}{2}-1} x[2r] W_N^{2rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] W_N^{2rk}$$

$$= \sum_{r=0}^{\frac{N}{2}-1} x[2r] W_{N/2}^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] W_{N/2}^{rk}$$

Penjabaran secara detail persamaan di atas dapat dilihat pada matrix di bawah ini.

$$\begin{pmatrix} [W_2^{01}][\ W_2^{01}] \\ [W_2^{10}][\ W_2^{11}] \end{pmatrix} \begin{pmatrix} [x[0]] \\ [x[1]] \end{pmatrix} + \begin{pmatrix} [W_2^{01}][\ W_2^{01}] \\ [W_2^{10}][\ W_2^{11}] \end{pmatrix} \begin{pmatrix} [x[0]] \\ [x[1]] \end{pmatrix} \begin{pmatrix} [W_N^{nk}] \\ [W_N^{nk}] \end{pmatrix} = \begin{matrix} [X[0]] \\ [X[1]] \end{matrix}$$

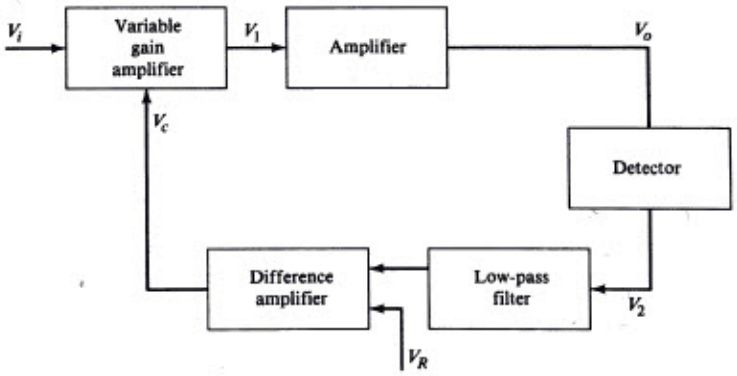
Dapat dilihat dari persamaan sebelumnya dimana persamaan *Discrete Fourier Transform* dipecah menjadi dua bagian. Proses dekomposisi tersebut dilakukan terus-menerus sampai elemen terkecil. Sehingga proses aritmatika bisa dikurangi dan mempercepat perhitungan (Kaplan, 2001).

D. Automatic Gain Control

Automatic Gain Control (AGC) merupakan suatu rangkaian yang mampu mengatur penguatan pada suatu sistem dan mengontrolnya secara automatis (Liestiawan, 2010). Rata-rata level sinyal keluaran merupakan

feedback untuk mengatur *gain* agar sesuai dengan *range* level sinyal masukan. *Automatic Gain Control* secara efektif menurunkan level daya bila sinyal terlalu kuat dan menaikkannya bila sinyal yang diterima terlalu rendah (Admawijaya, 2011). AGC berfungsi untuk membatasi besar daya yang tertangkap agar tidak terjadi kelebihan beban & distorsi karena penguat biasanya dirancang untuk mendeteksi sinyal terlemah dan mempunyai linearitas terbatas (Liestiawan, 2010). Dengan metode ini suara inputan audio yang relatif lemah dapat ditingkatkan levelnya.

Secara umum prinsip kerja dari *Automatic Gain Control* dapat dilihat pada Gambar 2.3. Input dari sinyal (Vi) dikuatkan dengan Variable Gain Amplifier(VGA). Dapat dilihat dalam diagram bahwa nilai dari VGA dipengaruhi oleh Vc. Parameter sinyal seperti amplitudo dideteksi oleh detektor kemudian difilter untuk dicari perbedaan tinggi sinyal sehingga menghasilkan Vc yang akan dipakai sebagai variable untuk penguat (Martinez, 2001).



Gambar 2.3 Block Diagram AGC

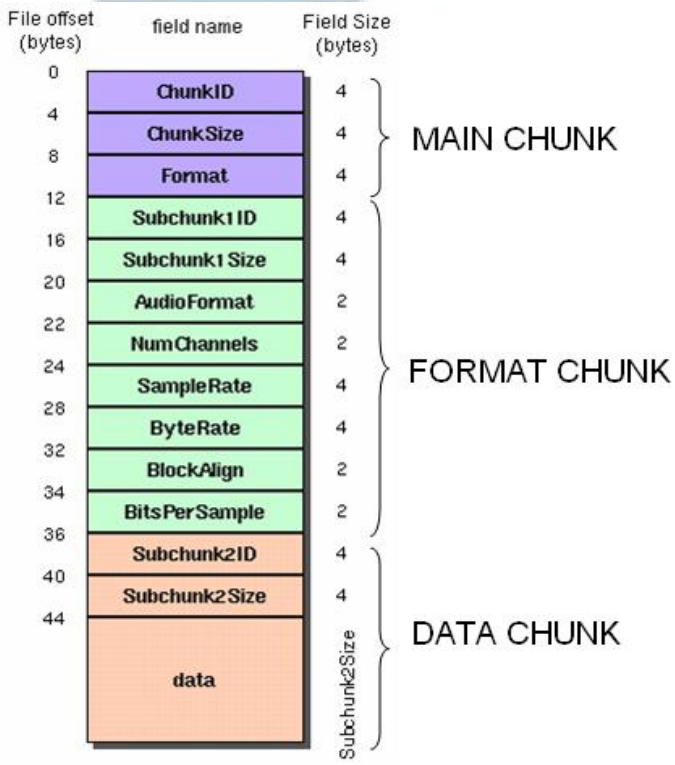
(http://www.eecg.toronto.edu/~kphang/papers/2001/martin_AGC.pdf

E. Amplitudo dan Decibel dalam WAV

Dalam teknologi perangkat lunak dikenal adanya file yang berekstensi WAV. File WAV ini merupakan file audio standar yang digunakan oleh Windows. WAV file ini memungkinkan suara direkam dalam berbagai kualitas, seperti 8-bit atau 16-bit samples dengan *rate* 11025Hz, 22050Hz atau 44100Hz. Untuk kualitas yang baik, yaitu: 44100Hz, 16-bit akan memakan kapasitas sekitar 150Kbytes setiap detiknya (Gunawan & Gunadi, 2003).

File WAV banyak dipakai dalam pembuatan game. Biasanya untuk suara-suara efek dan musik. Menurut Ibnu Gunawan dan Kartika Gunadi(2003), WAV sendiri memang cenderung memiliki ukuran yang besar, tetapi hal itu dikarenakan format *file* WAV yang tidak dikompresi sehingga memiliki kelebihan, yaitu mudah dimanipulasi.

Format *file* WAV merupakan bagian dari spesifikasi RIFF milik Microsoft yang digunakan untuk penyimpanan *file-file* multimedia. *File* WAV dimulai dengan bagian *header* dan diikuti oleh rentetan data *chunk*. *File* wav terdiri dari 3 bagian, yaitu *main chunk*, format *chunk*, dan data *chunk* (Adhit, 2010).



Gambar 2.4 Struktur *file* WAV.

(http://adhit8.blogspot.com/2010/04 /pembacaan-file-wav-di-delphi.html#comment-form)

Adhit(2010) juga mengatakan bahwa sinyal suara yang direpresentasikan file WAV dalam bentuk *discrete*, berupa deret bilangan yang merepresentasikan amplitudo dalam domain waktu. Pada bagian *header file* terdapat informasi tentang *file* WAV tersebut, di antaranya menyatakan nilai *sample rate*, jumlah *channel*, dan *bit per sample*. Dari keterangan pada *header file* tersebut dapat diketahui berapa sampel yang dicuplik dari sinyal analog tiap detik.

Pada penjelasan sebelumnya diketahui bahwa data yang tersimpan dalam WAV adalah amplitudo. Amplitudo adalah tinggi rendahnya tegangan dari gelombang analog dan berfungsi untuk merepresentasikan keras atau pelannya suatu suara (Aldebian, 2010). Pada file WAV yang memiliki resolusi 16 bit, terdapat total 65536 *level* amplitudo. Dengan *range* dari -32768 sampai 32767. Semakin besar resolusi dari file maka semakin besar *range*-nya (Poser, 2008).

Untuk mengukur keras atau pelannya suara terdapat satuan yang disebut *Decibel* (dB). Istilah dB juga sering dipakai dalam bidang komunikasi dan sinyal (Wolfe, 2009). Semakin tinggi nilai dB maka semakin keras suara yang terdengar. Suara yang lebih tinggi dari 85dB dapat mengakibatkan kerusakan pada sistem pendengaran manusia (Association, 2011).

Berikut adalah tabel yang dibuat berdasarkan sumber dari. Tabel ini menunjukkan level dB dan kebisingan suaranya.

Tabel 2.2 Tabel Tingkat Kebisingan Suara

(www.asha.org dan www.abelard.org)

Aktivitas	Tingkat Kebisingan	Decibel (dB)
Suara pistol Pesawat lepas landas	Bisa Merusak Pendengaran	140

Tabel 2.2 Tabel Tingkat Kebisingan Suara

(www.asha.org dan www.abelard.org)

Aktivitas	Tingkat Kebisingan	Decibel (dB)
Suara bor Klakson mobil	Batas bahaya untuk pendengaran	120 - 130
Suara diesel truk	Sangat bising	100
Suara orang berteriak	Sangat bising	90
Suara alarm	Bising / Keras	70 - 80
Suara orang bercakap-cakap normal	Sedang	50 - 60
Suara orang berbisik	Sunyi	40
Tempat sunyi	Sunyi	10 - 30
	Tidak terdengar suara	0

Amplitudo dalam file WAV dapat diubah kedalam satuan dB untuk diukur level kebisingan suaranya. Untuk mengubah amplitudo menjadi dB dapat memakai rumus

$$\mathrm{dB} = 20 \log_{10}(\mathrm{A}) \quad \text{.....(2.7)}$$

Dimana A adalah nilai amplitudo yang ingin diubah kedalam dB.

F. Bahasa Pemrograman C#

C# merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka **.**NET *Framework* (Raharja, 2011). Bahasa pemrograman ini dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya seperti *Java, Delphi, Visual Basic*, dan lain-lain) dengan beberapa penyederhanaan. Untuk mencapai produktifitas tinggi ini konsep-konsep sulit C++ disederhanakan dan fitur- fitur baru ditambahkan. Hal ini mungkin terasa mirip dengan Java, karena itulah C# bisa dianggap sebagai sepupu Java.

Kelebihan C# adalah program ini mampu membuat aplikasi yang bisa dengan mudah dihubungkan dengan teknologi yang yang ada pada *Microsoft.* C# memiliki teknologi yang dinamakan **.**NET (baca dot net) yang setiap saat semakin berkembang karena semakin berkembang juga teknologi yang ada pada *Microsoft.*

Ari Artama (2011) menyatakan bahwa, **.**NET *Framework* adalah suatu komponen dari *Windows* untuk memudahkan pengembangan dan eksekusi berbagai macam bahasa pemrograman dan kumpulan pustaka-pustaka (*library*) agar dapat bekerja sama dalam menjalankankan aplikasi berbasis *Windows*.

Menurut Agro Rachmatullah (2002), C# sebagai bahasa pemrograman untuk *Framework* **.**NET memiliki ruang lingkup

penggunaan yang sangat luas. Pembuatan program dengan user interface *Windows* maupun console dapat dilakukan dengan C#. Karena *Framework* .NET memberikan fasilitas untuk berinteraksi dengan kode yang tidak teratur, penggunaan *library* seperti *DirectX* 8.1 dan *OpenGL* dapat dilakukan. C# juga dapat digunakan untuk pemrograman *web site* dan *web service* (Mandala, 2011).