BAB 2
LANDASAN TEORI

2.1. Informasi

Istilah informasi berasal dari bahasa Inggris ”*to inform*” yang artinya dalam Bahasa Indonesia adalah ”memberitahukan”[3]. Informasi adalah data yang sudah diproses menjadi bentuk yang berguna bagi pemakai dan mempunyai nilai pikir yang nyata bagi pembuatan keputusan pada saat sedang berjalan atau untuk prospek masa depan. Definisi tersebut menekankan kenyataan bahwa data harus diproses dengan cara-cara tertentu untuk menjadi informasi dalam bentuk dan nilai yang berguna bagi pemakai[4]. Menurut para ahli, informasi adalah:

1. Informasi merupakan suatu data yang telah diproses dan mempunyai arti tertentu[4].
2. Informasi adalah sekumpulan fakta (data) yang diorganisasikan dengan cara tertentu sehingga mereka mempunyai arti bagi si penerima. Sebagai contoh, apabila memasukkan jumlah gaji dengan jumlah jam bekerja, akan mendapatkan informasi yang berguna. Dengan kata lain, informasi datang dari data yang akan diproses[4].
3. Menurut Nova[4], bahwa informasi dapat di produksi dan dipasarkan sebagai sebuah produk, pada dasarnya informasi merupakan suatu yang diproduksi dan didistribusikan, baik oleh sebuah lembaga pendidikan, radio, televisi, penerbit buku, koran dan majalah. Ketidak akuratan informasi akan menyebabkan prusahaan yangbergerak dibidang informasi dapat kehilangan reputasi dan kredibilitasnya.
4. Menurut Prasojo dan Riyanto[4], bahwa pengertian informasi sering disamakan dengan pengertian data. “Data adalah sesuatu yang belum diolah dan belum dapat digunakan sebagai dasar yang kuat dalam pengambilan keputusan”. Beberapa contoh data adalah data nama mahasiswa, jumlah kursi, jumlah siswa, dan lain-lain. Data mahasiswa relatif belum berarti, jika digunakan untuk mengambil keputusan tertentu. Data nama mahasiswa ditambah data IPK mahasiswa,dan persentase nilai “D” dapat digunakan untuk menentukan bahwa

mahasiswa tersebut dapat mengambil bebas teori atau tidak. Data bebas teori dan nilai skripsi dapat digunakan untuk mengambil keputusan bahwa mahasiswa tersebut berhak lulus atau tidak. Gabungan dari data nama mahasiswa, IPK, persentasi nilai "D", nilai skripsi merupakan sebuah informasi.

5. Menurut Laudon & Laudon dalam Kadir[3], bahwa "informasi adalah data yang telah diolah menjadi bentuk yang bermakna dan berguna bagi manusia".

Sumber dari informasi adalah data. Data merupakan bentuk jamak dari bentuk tunggal data, yaitu item ataupun datum. Data adalah kenyataan yang mengambarkan suatu kejadian-kejadian dan kesatuan nyata. Kejadian-kejadian nyata yang sering terjadi adalah perubahan dari suatu nilai yang disebut dengan transaksi.

Informasi akan memiliki arti manakala informasi tersebut memiliki unsur-unsur sebagai berikut:

a. Relevan artinya Informasi yang diinginkan benar-benar ada relevansi dengan masalah yang dihadapi.
b. Kejelasan artinya terbebas dari istilah-istilah yang membingungkan.
c. Akurasi artinya bahwa informasi yang hendak disajikan harus secara teliti dan lengkap.
d. Tepat waktu artinya data yang disajikan adalah data terbaru dan mutahir.

Jadi, hal yang terpenting untuk membedakan informasi dengan data, informasi itu mempunyai kandungan "makna" dan data tidak mempunyai "makna". Makna artinya *output* (sebagai penunjang sistem). Pengertian makna di sini merupakan hal yang sangat penting, karena berdasarkan makna inilah penerima dapat memahami informasi tersebut dan secara lebih jauh dapat menggunakannya untuk menarik suatu kesimpulan atau bahkan mengambil keputusan.

2.2. Sistem Temu Kembali (*Information Retrieval System*)

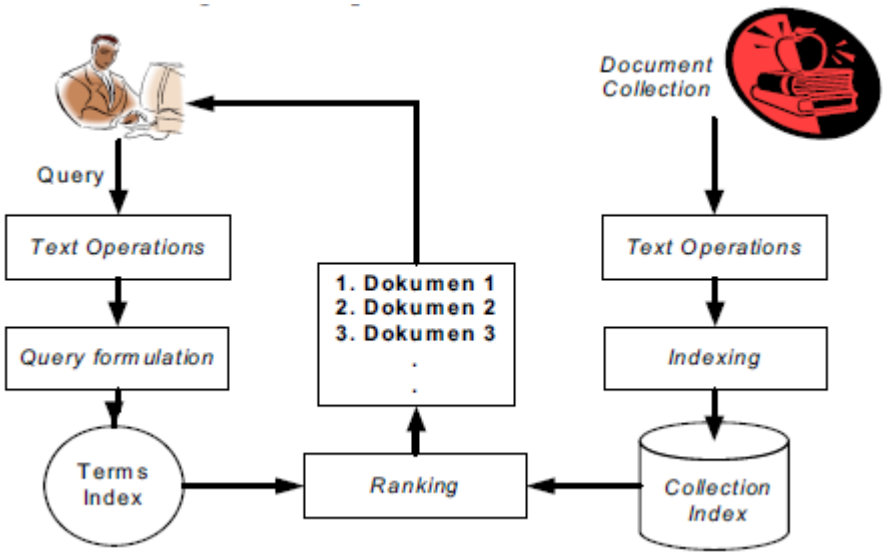
Definisi Sistem Temu Kembali (*Information Retrieval System*) adalah bagaimana menemukan suatu dokumen dari dokumen-dokumen tidak terstruktur yang memberikan informasi yang dibutuhkan dari koleksi dokumen yang sangat besar yang tersimpan dalam komputer[5]. Tujuan dari Sistem Temu Kembali (*Information Retrieval System*) adalah untuk memenuhi kebutuhan informasi pengguna dengan me*retrieve* semua dokumen yang mungkin relevan, pada waktu yang sama me-*retrieve* sesedikit mungkin dokumen yang tidak relevan. Sistem Temu Kembali (*Information Retrieval System*) yang baik memungkinkan pengguna menentukan secara cepat dan akurat apakah isi dari dokumen yang diterima memenuhi kebutuhannya.

Tujuan yang harus dipenuhi adalah bagaimana menyusun dokumen yang telah didapatkan tersebut ditampilkan terurut dari dukumen yang memiliki tingkat relevansi tinggi ke tingkat relevansi yang lebih rendah. Penyusunan dokumen tersebut disebut sebagai perangkingan dokumen.

Model Sistem Temu Kembali (*Information Retrieval System*) adalah model yang digunakan untuk melakukan pencocokan antara term-term kata (*query*) dengan term-term dalam *document collection* (folder *file*), model yang terdapat dalam Sistem Temu Kembali (*Information Retrieval System*) terbagi dalam 3 model besar, yaitu:

a. *Set-theoritic models,* model merepresentasikan dokumen sebagai himpunan kata atau frase. Contoh model ini ialah *Standard Boolean model* dan *Extended Boolean model.*
b. *Algebraic model,* model merepresentasikan dokumen dan *query* sebagai vektor *similarity* antara vektor dokumen dan vektor *query* yang direpresentasikan sebagai sebuah nilai skalar. Contoh model ini ialah *Vektor Space Model* (model ruang vektor) , *Latent Semantic Indexing* (LSI) dan *Generalized Vector Space Model* (GVSM).
c. *Probabilistic model,* model memperlakukan proses pengambilan dokumen sebagai sebuah *probabilistic inference.* Contoh model ini ialah penerapan teorema bayes dalam model probabilistik.

Sebagai suatu sistem, Sistem Temu Kembali (*Information Retrieval System*) memiliki beberapa bagian yang membangun sistem secara keseluruhan. Gambaran bagian-bagian yang terdapat pada suatu Sistem Temu Kembali (*Information Retrieval System*) digambarkan pada Gambar 2.1.



Gambar 2.1. Bagian-bagian *Information Retrieval System*

Dari gambar 2.1. terlihat bahwa terdapat dua proses operasi dalam Sistem Temu Kembali (*Information Retrieval System*). Proses pertama dimulai dari koleksi dokumen dan proses kedua dimulai dari *query* pengguna. Proses pertama yaitu pemrosesan terhadap koleksi dokumen menjadi basis data indeks tidak ada ketergantungan dengan proses kedua. Sedangkan proses kedua tergantung dari keberadaan basis data indeks yang dihasilkan pada proses pertama. Bagian-bagian dari Sistem Temu Kembali (*Information Retrieval System*) menurut gambar 2.1. meliputi :

1. *Text Operations* (operasi terhadap teks) yang meliputi pemilihan kata-kata dalam query maupun dokumen (*term selection*) dalam pentransformasian dokumen atau query menjadi *term index* (indeks dari kata-kata).
2. *Query formulation* (formulasi terhadap *query*) yaitu memberi bobot pada indeks kata-kata *query*.

3. *Ranking* (perangkingan), mencari dokumen-dokumen yang relevan terhadap *query* dan mengurutkan dokumen tersebut berdasarkan kesesuaiannya dengan *query*.
4. *Indexing* (pengindeksan), membangun basis data indeks dari koleksi dokumen. Dilakukan terlebih dahulu sebelum pencarian dokumen dilakukan.

Sistem Temu Kembali (*Information Retrieval System*) menerima *query* dari pengguna, kemudian melakukan perangkingan terhadap dokumen pada koleksi berdasarkan kesesuaiannya dengan *query*. Hasil perangkingan yang diberikan kepada pengguna merupakan dokumen yang menurut sistem relevan dengan *query*.

Namun relevansi dokumen terhadap suatu *query* merupakan penilaian pengguna yang subjektif dan dipengaruhi banyak faktor seperti topik, pewaktuan, sumber informasi maupun tujuan pengguna. Salah satu model Sistem Temu Kembali (*Information Retrieval System*) adalah model vektor. Beberapa karakteristik dalam model vektor adalah :

1. Model vektor berdasarkan *index term*
2. Model vektor mendukung *partial matching* dan penentuan peringkat dokumen
3. Prinsip dasar vektor model adalah sebagai berikut :

a. dokumen direpresentasikan dengan menggunakan vektor *index term*
b. Ruang dimensi ditentukan oleh *index term*
c. *Query* direpresentasikan dengan menggunakan vektor *index term*
d. Kesamaan *document-query* dihitung berdasarkan hasil kali titik (*cross product*) antara vektor – vektor tersebut

4. Model vektor memerlukan :

a. Bobot *index term* untuk vektor dokumen
b. Bobot *index term* untuk *query*
c. Perhitungan *cross product* untuk vektor *document-query*

5. Kinerja

a. Efisien
b. Mudah dalam representasi
c. Dapat diimplementasikan pada *document-matching*

2.2.1. *Generalized Vector Space Model* (GVSM)

Generalized Vector Space Model (GVSM) merupakan perluasan dari *Vector Space Model* (VSM) yaitu dengan menambahkan jenis informasi tambahan, disamping term, dalam merepresentasikan dokumen[6]. Sistem Temu Kembali (*Information Retrieval System*) dengan *Generalized Vector Space Model* (GVSM) merepresentasikan dokumen dengan similiaritas vektor terhadap semua dokumen yang ada.

Pada tahun 1985, Wong et al[6], menyajikan suatu alternatif terhadap Sistem Temu Kembali (*Information Retrieval System*) *Vector Space Model* (VSM), yang disebut *Generalized Vector Space Model* (GVSM). Deskripsi ringkas mengenai *Generalized Vector Space Model* (GVSM) diberikan oleh Carbonell dkk. Asumsikan term dari *Vector Space Model* (VSM) adalah *liniearly independent*. *Generalized Vector Space Model* (GVSM) menghindari pengasumsian dengan penggunaan dokumen-dokumen sebagai dasar ruang vektor dari pada term. Dalam "*Dual Space*" suatu dokumen direpresentasikan oleh suatu vektor dimana dimensinya merujuk terhadap dokumen.

Algoritma *Generalized Vector Space Model* yang dibahas menggunakan konsep ruang vektor. Masukan dari pengguna dan kumpulan dokumen diterjemahkan menjadi vektor-vektor, kemudian vektor-vektor tersebut dikenakan operasi perkalian titik dan hasilnya menjadi acuan dalam menentukan relevansi masukan pengguna (*query*) terhadap kumpulan dokumen.

Ada beberapa langkah atau proses untuk mendapatkan hasil dari *query* yang dimasukkan, yang disebut algoritma *Generalized Vector Space Model*[6]:

1. Membuang kata depan dan kata penghubung.
2. Menggunakan *stemmer* pada kumpulan dokumen dan *query*, yaitu aplikasi yang digunakan untuk menghilangkan imbuhan (awalan, akhiran). Contoh: ketampanan: tampan, kesalahan: salah.
3. Menentukan *minterm* untuk menentukan kemungkinan pola frekuensi kata. Panjang *minterm* ini didasarkan pada banyak kata yang diinput pada *query*. Kemudian diubah menjadi vektor orthogonal sesuai dengan pola *minterm* yang muncul. Kemungkinan pola yang akan muncul adalah :

$M_1 = (0,0,0)$
$M_2 = (1,0,0)$
.....
$M_n = (0,0,0)$

4. Menghitung banyaknya frekuensi atau kemunculan kata dalam kumpulan dokumen yang sesuai dengan *query*
5. Menghitung *index term* yang dapat dinyatakan dengan :

$$\overrightarrow{Ki} = \frac{\sum_{\forall r.gi(Mr)=1} Ci,r.\overrightarrow{Mr}}{\sqrt{\sum_{\forall r.gi(Mr)=1} Ci,r^2}} \quad \text{.... (2.1)}$$

Dimana :

$\overrightarrow{Ki}$: Indeks term ke-i

$\overrightarrow{Mr}$: Vektor ortogonal sesuai pola minterm yang terpakai

Ci,r : Faktor korelasi antara indeks term ke-i dengan minterm r

Sedangkan faktor korelasi sebagai berikut:

$$C_{i,r} = \sum_{dj|gi(\overrightarrow{dj})=gi(Mr)} W_{i,j} \quad \text{.... (2.2)}$$

Dimana:

$W_{i,j}$: Vektor dokumen ke-j

$gi(Mr)$: Bobot indeks term K_i dalam minterm M_r

6. Mengubah dokumen dan *query* menjadi vector

$$\overrightarrow{d_j} = \sum_{i=1}^{n} W_{i,j} \times \overrightarrow{K_t} \quad \text{.... (2.3)}$$

$$\vec{q} = \sum_{i=1}^{n} q_i \times \overrightarrow{K_i} \quad \text{.... (2.4)}$$

Dimana:

$\overrightarrow{d_J}$: Vektor dokumen ke-j

$\vec{q}$: Vektor *query*

$W_{i,j}$: Berat indeks *term* i pada dokumen j

q_i : Berat indeks *term* pada *query* i

n : Jumlah indeks *term*

7. Mengurutkan dokumen berdasarkan similaritas, dengan menghitung perkalian vector

$$Sim(\overrightarrow{d_J}.\vec{q}) = \frac{\overrightarrow{d_J}.\vec{q}}{|\overrightarrow{d_J}|\,|\overrightarrow{q}|} \qquad \text{…. (2.5)}$$

Dimana :

$\overrightarrow{d_J}$: Vektor dokumen ke-J

$\vec{q}$: Vektor *query*

2.2.2. *Preprocessing*

Proses yang terjadi pada *Generalized Vector Space Model* (GVSM) terbagi menjadi dua yaitu tahapan *preprocessing* yang terdiri dari *reading text* (*.doc) menggunakan *indexing*, *tokenizing*, *filtration* atau *stop word removal*, *stemming*, sedangkan proses yang kedua adalah menghitung relevansi antara kumpulan dokumen yang telah di-*preprocess* dengan *query* yang diinginkan pengguna. Banyaknya kemunculan kata dalam kumpulan dokumen yang sesuai dengan *query* akan dihitung.

Query yang dimasukan oleh pengguna dan dokumen diterjemahkan menjadi vektor-vektor kemudian vektor-vektor tersebut dikenakan operasi perkalian titik dan hasilnya menjadi acuan dalam menentukan relevansi masukan pengguna (*query*) terhadap kumpulan dokumen.

2.2.2.1. *Indexing*

Mencari sebuah informasi yang relevan sangat tidak mungkin dapat dilakukan oleh sebuah komputer, meskipun dilakukan oleh sebuah komputer yang memiliki spesifikasi yang canggih. Agar komputer dapat mengetahui sebuah dokumen itu relevan terhadap sebuah informasi, komputer memerlukan sebuah model yang mendeskripsikan bahwa dokumen tersebut relevan atau tidak. Salah satu caranya adalah dengan menggunakan indeks istilah.

Indeks adalah bahasa yang digunakan di dalam sebuah buku konvensional untuk mencari informasi berdasarkan kata atau istilah yang mengacu ke dalam suatu halaman. Dengan menggunakan indeks, pencari informasi dapat dengan mudah menemukan informasi yang diinginkannya. Pada sistem temu-kembali informasi, indeks ini nantinya yang digunakan untuk merepresentasikan informasi di dalam sebuah dokumen.

Elemen dari indeks adalah istilah indeks (*index term*) yang didapatkan dari teks yang dipecah di dalam sebuah dokumen. Elemen lainnya adalah bobot istilah (*term weighting*) sebagai penentuan rangking dari kriteria relevan sebuah dokumen yang memiliki istilah yang sama.

Baeza-Yates dan Ribeiro-Neto[7] menjelaskan tentang proses pembuatan indeks dari sebuah dokumen teks atau dikenal dengan proses analisis teks (*automatic teks analysis*) melalui beberapa tahap:

a. Proses penghapusan digit, tanda hubung, tanda baca dan penyeragaman dari huruf yang digunakan.
b. Penyaringan kata meliputi penghilangan bukan kata kunci yang disebut *filtering* atau *stopword removal*.
c. Penghilangan imbuhan kata, baik awalan maupun akhiran kata. Penghilangan imbuhan kata ini dikenal dengan *stemming*.
d. Pemilihan istilah untuk menentukan kata atau *stem* (kelompok kata) yang akan digunakan sebagai elemen indeks.
e. Pembentukan kategori istilah terstruktur seperti kelompok persamaan kata yang digunakan untuk perluasan dari *query* dasar yang diberikan oleh pengguna sistem temu-kembali informasi dengan istilah lain yang sesuai.

Pengindeksan dapat dilakukan dengan dua cara yaitu manual dan otomatis. Idealnya, untuk mendapatkan indeks istilah yang sempurna sebuah pengindeksan dilakukan secara manual (konvensional). Akan tetapi, menurut Salton[7] sistem pencarian dan analisa teks yang sepenuhnya otomatis tidak menghasilkan kinerja temu-kembali yang lebih buruk dibandingkan dengan sistem konvensional yang menggunakan pengindeksan dokumen manual dan formulasi pencarian manual.

2.2.2.2. *Tokenization*

Sebelum indeks dibandingkan dengan dokumen, dilakukan *tokenization* terlebih dahulu, yaitu mencacah kalimat kedalam bagian-bagian. Contohnya "dia bernama rahman", setelah kalimat dimasukan maka tugas token adalah memecah kedalam bagian-bagian menjadi "dia", "bernama", "rahman".

Terlihat dari contoh diatas terdapat kalimat "dia bernama Rahman" kemudian proses *tokenization* dilakukan dengan memecah kata dalam kalimat tersebut menjadi 3 pecahan yaitu dia, bernama, dan rahman.

2.2.2.3. *Filtering* / *Stop Word*

Dalam sebuah dokumen terdapat banyak kata yang bukan kata kunci di dalam dokumen atau kata-kata tambahan hanya untuk menghubungkan kata, contohnya adalah kata penghubung dan juga terdapat tanda-tanda baca. Dalam proses *indexing* dilakukan proses untuk menghilangkan kata-kata tersebut untuk mengurangi proses peng-*index*-an dan mengurangi kata-kata dan tanda baca yang nantinya tidak berkaitan langsung dengan kata kunci. Selain untuk mengungari proses *indexing* proses tersebut dilakukan agar penerapan perhitungan kesamaan dokumen dengan dokumen yang dicari terdapat kesesuaikan karena berkurangnya *noise* kata penghubung dan tanda baca yang jika tidak dihilangkan akan masuk kedalam perhitungan. Pada tabel 2.1. berikut adalah contoh kata dalam bahasa indonesia yang dihilangkan[10]:

Tabel 2.1. kata *Filtering / Stopwords*

stopwords					
ada	biasanya	kalau	menunjukkan	sedang	setiap
adalah	bila	kalian	menurut	sedangkan	seusai
adanya	bilamana	kami	mereka	sedikit	sewaktu
adapun	buat	kamu	merupakan	segera	si
aduh	bukan	karena	meski	sehabis	siapa
dalam	kata	meskipun	sehingga	siapakah	
dan	katanya	misalnya	sehubungan	siapapun	
akan	dapat	kau	mungkin	sejak	suatu
aku	dari	ke	namun	sejumlah	sudah
alih-alih	daripada	kebanyakan	nanti	sekarang	supaya
anda	dekat	kecuali	nyaris	sekeliling	tak
andai	demi	kemanakah	oleh	seketika	tanpa
antar	demikian	kemudian	pada	sekitar	tapi
antara	dengan	kenapa	padahal	sekonyong	tatkala
apakah	di	kepada	pasti	selagi	tengah
apalagi	dia	ketika	pelbagai	selain	tentang
asalkan	dikatakan	ketimbang	per	selalu	tentu
atas	dilakukan	kini	peri	selama	tentunya
atau	dkk	kita	perihal	selanjutnya	tergolong
ataupun	dll	lagi	pinggir	selesai	terhadap
Bagai	dsb	lain	pula	seluruh	terjadi
bagaikan	engkau	lain-lain	pun	seluruhnya	terkadang
bagaimana	hal	lainnya	saat	semakin	terlalu
bagaimanakah	hampir	lalu	saja	semenjak	terlebih

bagaimanapun	hanya	lebih	sambil	sementara	termasuk
bagi	harus	lepas	sampai	semua	ternyata
bahkan	hingga	lewat	samping	semuanya	tersebut
bahwa	ia	maka	sang	seorang	tertentu
balik	ialah	makin	sangat	sepanjang	tetap
banyak	ini	manakala	sangatlah	seperti	tetapi
barangkali	itu	masih	saya	sepertinya	tiap
bawah	iya	masing-masing	seakan	seputar	tiba-tiba
beberapa	jadi	masing-masingnya	seakan-akan	seraya	tidak
begini	jangan	maupun	seantero	sering	ujar
begitu	jarang	melainkan	sebab	seringkali	ujarnya
belakang	jauh	melakukan	sebabnya	serta	umumnya
belum	jika	melalui	sebagai	sesuai	untuk
berapa	jikalau	memang	sebagaimana	sesuatu	walau
berbagai	juga	mengatakan	sebagainya	sesudah	walaupun
bersama	jumlah	mengenai	sebelum	sesudahnya	ya
betapa	kadang	menjelang	sebuah	setelah	yakni
beserta	justru	menjadi	sebelumnya	sesungguhnya	yaitu
biar	kadang-kadang	menuju	secara	seterusnya	yang

2.2.2.4. *Stemming*

Stemming merupakan bagian yang tidak terpisahkan dalam Sistem Temu Kembali (*Information Retrieval System*). Algoritma Nazief & Adriani sebagai algoritma *stemming* untuk teks berbahasa Indonesia yang memiliki kemampuan prosentase keakuratan (presisi) lebih baik dari algoritma lainnya. Algoritma ini sangat dibutuhkan dan menentukan dalam proses *preprocessing* dalam dokumen Indonesia.

Stemming adalah salah satu cara yang digunakan untuk meningkatkan performa Sistem Temu Kembali (*Information Retrieval System*) dengan cara mentransformasi kata-kata dalam sebuah dokumen teks ke bentuk kata dasarnya. Algoritma *stemming* untuk bahasa yang satu berbeda dengan algoritma *stemming* untuk bahasa lainnya. Sebagai contoh bahasa Inggris memiliki morfologi yang berbeda dengan bahasa Indonesia sehingga algoritma *stemming* untuk kedua bahasa tersebut juga berbeda. Proses *stemming* pada teks berbahasa Indonesia lebih rumit/kompleks karena terdapat variasi imbuhan yang harus dibuang untuk mendapatkan kata dasar (*root word*) dari sebuah kata. Pada umumnya kata dasar pada bahasa Indonesia terdiri dari kombinasi:

Prefiks 1* + *Prefiks 2* + *Kata dasar* + *Sufiks 3* + *Sufiks 2* + *Sufiks 1

Algoritma Nazief & Adriani yang dibuat oleh Bobby Nazief dan Mirna Adriani ini memiliki tahap-tahap sebagai berikut:

1. Pertama cari kata yang akan diistem dalam kamus kata dasar. Jika ditemukan maka diasumsikan kata adalah *root word*, maka algoritma berhenti.
2. *Inflection Suffixes* (“-lah”, “-kah”, “-ku”, “-mu”, atau “-nya”) dibuang. Jika berupa *particles* (“-lah”, “-kah”, “-tah” atau “-pun”) maka langkah ini diulangi lagi untuk menghapus *Possesive Pronouns* (“-ku”, “-mu”, atau “-nya”), jika ada.
3. Hapus *Derivation Suffixes* (“-i”, “-an” atau “-kan”). Jika kata ditemukan di kamus, maka algoritma berhenti. Jika tidak maka ke langkah 3a

a. Jika “-an” telah dihapus dan huruf terakhir dari kata tersebut adalah “-k”, maka “-k” juga ikut dihapus. Jika kata tersebut ditemukan dalam kamus maka algoritma berhenti. Jika tidak ditemukan maka lakukan langkah 3b.

b. Akhiran yang dihapus (“-i”, “-an” atau “-kan”) dikembalikan, lanjut ke langkah 4.

4. Hapus *Derivation Prefix*. Jika pada langkah 3 ada sufiks yang dihapus maka pergi ke langkah 4a, jika tidak pergi ke langkah 4b.

a. Periksa tabel kombinasi awalan-akhiran yang tidak diijinkan. Jika ditemukan maka algoritma berhenti, jika tidak

b. For i = 1 to 3, tentukan tipe awalan kemudian hapus awalan. Jika root word belum juga ditemukan lakukan langkah 5, jika sudah maka algoritma berhenti. Catatan: jika awalan kedua sama dengan awalan pertama algoritma berhenti.

5. Melakukan Recoding.

6. Jika semua langkah telah selesai tetapi tidak juga berhasil maka kata awal diasumsikan sebagai *root word*, proses selesai.

Tipe awalan ditentukan melalui langkah-langkah berikut:

1. Jika awalannya adalah: “di-”, “ke-”, atau “se-” maka tipe awalannya secara berturut-turut adalah “di-”, “ke-”, atau “se-”.
2. Jika awalannya adalah “te-”, “me-”, “be-”, atau “pe-” maka dibutuhkan sebuah proses tambahan untuk menentukan tipe awalannya.
3. Jika dua karakter pertama bukan “di-”, “ke-”, “se-”, “te-”, “be-”, “me-”, atau “pe-” maka berhenti.
4. Jika tipe awalan adalah “none” maka berhenti. Jika tipe awalan adalah bukan “none” maka awalan dapat dilihat pada Tabel 2.2. Hapus awalan jika ditemukan.

Tabel 2.2. Kombinasi Awalan Akhiran Yang Tidak Diijinkan

Awalan	Akhiran
be-	-i
di-	-an
ke-	-i, -kan
me-	-an
se-	-i, -kan

Tabel 2.3. Cara Menentukan Tipe Awalan Untuk awalan “te-”

Following Characters				Tipe Awalan
Set 1	Set 2	Set 3	Set 4	
“-r-“	“-r-“	-	-	none
“-r-“		-	-	ter-luluh
“-r-“	not (vowel or “-r-”)	“-er-“	vowel	ter
“-r-“	not (vowel or “-r-”)	“-er-“	not vowel	ter-
“-r-“	not (vowel or “-r-”)	not “-er-“	-	ter
not (vowel or “-r-”)	“-er-“	vowel	-	none
not (vowel or “-r-”)	“-er-“	not vowel	-	te

Tabel 2.4. Jenis Awalan Berdasarkan Tipe Awalannya

Tipe Awalan	Awalan yang harus dihapus
di-	di-
ke-	ke-
se-	se-
te-	te-
ter-	ter-
ter-luluh	ter

Untuk mengatasi keterbatasan pada algoritma di atas, maka ditambahkan aturan-aturan dibawah ini:

1. Aturan untuk reduplikasi.

a. Jika kedua kata yang dihubungkan oleh kata penghubung adalah kata yang sama maka *root word* adalah bentuk tunggalnya, contoh : “buku-buku” *root word*-nya adalah “buku”.

b. Kata lain, misalnya “bolak-balik”, “berbalas-balasan, dan ”seolah-olah”. Untuk mendapatkan *root word*-nya, kedua kata diartikan secara terpisah. Jika keduanya memiliki *root word* yang sama maka diubah menjadi bentuk tunggal, contoh: kata “berbalas-balasan”, “berbalas” dan “balasan” memiliki *root word*

yang sama yaitu “balas”, maka *root word* “berbalas-balasan” adalah “balas”. Sebaliknya, pada kata “bolak-balik”, “bolak” dan “balik” memiliki *root word* yang berbeda, maka *root word*-nya adalah “bolak-balik”.

2. Tambahan bentuk awalan dan akhiran serta aturannya.

a. Untuk tipe awalan “mem-“, kata yang diawali dengan awalan “memp-” memiliki tipe awalan “mem-”.
b. Tipe awalan “meng-“, kata yang diawali dengan awalan “mengk-” memiliki tipe awalan “meng-”.

Berikut contoh-contoh aturan yang terdapat pada awalan sebagai pembentuk kata dasar.

1. Awalan Se-

Se + semua konsonan dan vokal tetap tidak berubah

Contoh :

a. Se + bungkus = sebungkus
b. Se + nasib = senasib
c. Se + arah = searah
d. Se + ekor = seekor

2. Awalan Me-

Me + vokal (a,i,u,e,o) menjadi sengau “meng”

Contoh :

a. Me + inap = menginap
b. Me + asuh = mengasuh
c. Me + ubah = mengubah
d. Me + ekor = mengekor
e. Me + oplos = mengoplos

Me + konsonan b menjadi “mem”

Contoh :

a. Me + beri = member
b. Me + besuk = membesuk

Me + konsonan c menjadi “men”

Contoh :

a. Me + cinta = mencinta
b. Me + cuci = mencuci

Me + konsonan d menjadi “men”

Contoh :

a. Me + didik = mendidik
b. Me + dengkur = mendengkur

Me + konsonan g dan h menjadi “meng”

Contoh :

a. Me + gosok = menggosok
b. Me + hukum = menghukum

Me + konsonan j menjadi “men”

Contoh :

a. Me + jepit = menjepit
b. Me + jemput = menjemput

Me + konsonan k menjadi “meng” (luluh)

Contoh :

a. Me + kukus = mengukus
b. Me + kupas = mengupas

Me + konsonan p menjadi “mem” (luluh)

Contoh :

a. Me + pesona = mempesona
b. Me + pukul = memukul

Me + konsonan s menjadi “meny” (luluh)

Contoh :

a. Me + sapu = menyapu
b. Me + satu = menyatu

Me + konsonan t menjadi “men” (luluh)

Contoh :

a. Me + tanama = menanam
b. Me + tukar = menukar

Me + konsonan (l,m,n,r,w) menjadi tetap “me”

Contoh :

a. Me + lempar = melempar
b. Me + masak = memasak
c. Me + naik = menaik
d. Me + rawat = merawat
e. Me + warna = mewarna

3. Awalan Ke-

Ke + semua konsonan dan vokal tetap tidak berubah

Contoh :

a. Ke + bawa = kebawa
b. Ke + atas = keatas

4. Awalan Pe-

Pe + konsonan (h,g,k) dan vokal menjadi “per”

Contoh :

a. Pe + hitung + an = perhitungan
b. Pe + gelar + an = pergelaran
c. Pe + kantor + = perkantoran

Pe + konsonan “t” menjadi “pen” (luluh)

Contoh :

a. Pe + tukar = penukar
b. Pe + tikam = penikam

Pe + konsonan (j,d,c,z) menjadi “pen”

Contoh :

a. Pe + jahit = penjahit
b. Pe + didik = pendidik
c. Pe + cuci = pencuci
d. Pe + zina = penzina

Pe + konsonan (b,f,v) menjadi “pem”

Contoh :

a. Pe + beri = pemberi

b. Pe + bunuh = pembunuh

Pe + konsonan “p” menjadi “pem” (luluh)

Contoh :

a. Pe + pikir = pemikir
b. Pe + potong = pemotong

Pe + konsonan “s” menjadi “peny” (luluh)

Contoh :

a. Pe + siram = penyiram
b. Pe + sabar = penyabar

Pe + konsonan (l,m,n,r,w,y) tetap tidak berubah

Contoh :

a. Pe + lamar = pelamar
b. Pe + makan = pemakan
c. Pe + nanti = penanti
d. Pe + wangi = pewangi

2.2.3. Algoritma *Lesk*

Algoritma *lesk* adalah algoritma yang digunakan untuk menghilangkan ambiguitas makna kata. Algoritma *lesk* merupakan salah satu algoritma untuk menyelesaikan masalah ambigu atau kata yang memiliki lebih dari satu arti (*word sense disambiguation*) dengan berbasis kamus. Algoritma ini bekerja dengan membandingkan definisi dari kata yang berambigu dengan definisi. Definisi dari kata tetangganya berdasarkan definisi kamus[13].

Algoritma *lesk* merupakan implementasi dari pendekatan *unsupervised* atau jaringan saraf tiruan yang digunakan untuk menghilangkan ambiguitas makna kata. Algoritma ini berdasarkan intuisi bahwa kata yang bermakna ambigu yang terdapat bersamaan dalam kalimat, digunakan untuk merujuk topik yang sama dan makna yang berhubungan dengan topik tersebut didefinisikan di dalam kamus dengan menggunakan kata yang sama. Algoritma *lesk* memiliki fungsi *relatedness* (keterhubungan) akan mengembalikan jumlah kata yang tumpang tindih antara definisi dari dua kata yang dimasukan.

Algoritma lesk membandingkan kata awal (*query*) dengan sinonim kata itu sendiri yang disebut dengan kata pembanding. Setiap kata pembanding yang memiliki nilai terbesar maka akan diambil dan ditambahkan kedalam *query* untuk menghilangkan kata berambigu. Seperti "kepala cabang" memiliki kata berambigu "kepala". Kata "kepala" memiliki 2 makna yaitu "organ" dan "pemimpin" yang nantinya akan menjadi kata pembanding. Dari kedua kata pembanding tersebut dihitung perbandingan makna kata antara *query* dengan kata pembanding dan hasil perbandingan tersebut dilihat bobot paling besar. Berdasarkan dari bobot paling besar tersebut menghasilkan kata yang akan dimasukan kedalam *query*[13].

2.3. Model Analisis Dan Perancangan

Model analisis dan perancangan yang digunakan untuk membangun aplikasi ini adalah sebagai berikut:

2.3.1. *Unified Modeling Language* (UML)

Unified Modeling Language (UML) merupakan bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. UML tidak berdasarkan pada bahasa pemrograman tertentu. Standar spesifikasi UML dijadikan standar *defacto* oleh OMG (*Object Management Group*) pada tahun 1997. UML yang berorientasikan obyek mempunyai beberapa notasi standar. Spesifikasi ini menjadi populer dan standar karena sebelum adanya UML, telah ada berbagai macam spesifikasi yang berbeda. Hal ini menyulitkan komunikasi antar pengembang perangkat lunak. Untuk itu beberapa pengembang spesifikasi yang sangat berpengaruh berkumpul untuk membuat standar baru. UML dirintis oleh Grady Booch, James Rumbaugh pada tahun 1994 dan kemudian Ivar Jacobson[14].

Menurut perintisnya, UML di definisikan sebagai bahasa visual untuk menjelaskan, memberikan spesifikasi, merancang, membuat model, dan mendokumentasikan aspek-aspek dari sebuah sistem. Karena tergolong bahasa visual, UML lebih mendepankan penggunaan diagram untuk menggambarkan

aspek dari sistem yang sedang dimodelkan. Memahami UML itu sebagai bahasa visual itu penting, karena penekanan tersebut membedakannya dengan bahasa pemrograman yang lebih dekat ke mesin. Bahasa visual lebih dekat ke mental model pikiran, sehingga pemodelan menggunakan bahasa visual bisa lebih mudah dan lebih cepat dipahami dibandingkan apabila dituliskan dalam sebuah bahasa pemrograman.

Seperti yang telah dipaparkan di atas, UML yang merupakan turunan dan beberapa metode mempunyai kumpulan diagram *grafts* sebagai kombinasi dari konsep pemodelan data (*Entity Relationship Diagram*), pemodelan bisnis (*work flow*), pemodelan obyek, dan pemodelan komponen. Diagram *grafts* tersebut merupakan tampiian dari beberapa level abstraksi yang dapat digunakan secara bersama oleh semua proses pada seluruh *lifecycle* pengembangan perangkat lunak serta pada implementasi kebeberapa teknologi yang berbeda. UML memiliki berbagai jenis diagram, diantarnya adalah:

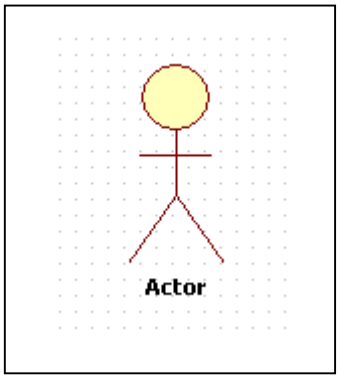
2.3.1.1. *Usecase Diagram*

Use case diagram berisi mengenai interaksi antara sekelompok proses dengan sekelompok *actor*, menggambarkan fungsionalitas dari sebuah sistem yang dibangun dan bagaimana sistem berinteraksi dengan dunia luar[14]. *Use case diagram* dapat digunakan selama proses analisis untuk menangkap kebutuhan sistem dan untuk memahami bagaimana sistem seharusnya bekerja.

Use Case diagram terdiri dari beberapa elemen pemodelan utama, yaitu *Actor*, *Use Case*, *Association, Dependency*, dan *Generalization.*

a. *Actor*

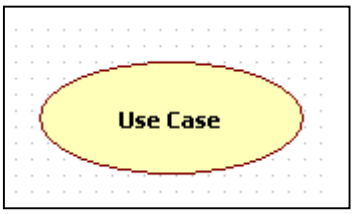
Pada dasarnya *actor* bukanlah bagian dari *use case* diagram, namun untuk dapat terciptanya suatu *use case* diagram diperlukan beberapa *actor*. *Actor* tersebut mempresentasikan seseorang atau sesuatu (seperti perangkat, sistem lain) yang berinteraksi dengan sistem. *Actor* hanya berinteraksi dengan *use case*, tetapi tidak memiliki kontrol atas *use case*. *Actor* digambarkan dengan stick man. Pada gambar 2.2. menunjukan Simbol *Actor* pada *Use Case* Diagram.



Gambar 2.2. Simbol *Actor* pada *Use Case* diagram

b. *Use Case*

Use case adalah gambaran fungsionalitas dari suatu sistem, sehingga *customer* atau pengguna sistem paham dan mengerti mengenai kegunaan sistem yang akan dibangun. Pada gambar 2.3. merupakan Simbol *Use Case* pada *Use Case* Diagram.



Gambar 2.3. Simbol *Use Case*

c. *Association*

Associaton menghubungkan *link* antar *element*, dan bukan menggambarkan aliran data / informasi pada sistem. *Association* digunakan untuk menggambarkan bagaimana *actor* terlibat dalam *use case*. Ada 4 jenis relasi yang bisa timbul pada *use case* diagram, yaitu *Association* antara *actor* dan use *case*, *Association* antara *use case*, *Generalization*/*Inheritance* antara *use case*, *Generalization*/*Inheritance* antara *actors*.

d. *Dependency*

Dependency adalah sebuah *element* bergantung dalam beberapa cara ke *element* lainnya.

e. *Generalization*

Generalization disebut juga *inheritance* (pewarisan), sebuah elemen dapat merupakan spesialisasi dari elemen lainnya.

2.3.1.2. *Activity Diagram*

Activity diagram menggambarkan berbagai alir aktifitas dalam sistem yang sedang diarancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir[14]. *Activity* diagram juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state* diagram khusus, di mana sebagian besar *state* adalah *Action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity* diagram tidak menggambarkan *behavior* internal sebuah sistem.

2.3.1.3. *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang disusun dalam suatu urutan waktu[14].

Secara khusus, diagram ini berasosiasi dengan *use case*. *Sequence* diagram menggambarkan *behavior* internal sebuah sistem. Dan lebih menekankan pada penyampaian *message* dengan parameter waktu.

2.3.1.4. *Class Diagram*

Class diagram adalah diagram yang menunjukan *class-class* yang ada dari sebuah sistem dan hubungannya secara logika dan menggambarkan struktur statis dari sebuah sistem[14]. *Class* diagram digunakan untuk menampilkan beberapa kelas serta paket-paket yang ada dalam sistem/perangkat lunak yang sedang digunakan.

Class diagram memberikan gambaran (diagram statis) tentang sistem/perangkat lunak dan relas-relasi yang ada didalamnya. Sebuah *class* digambarkan seperti sebuah bujur sangkar dengan tiga bagian ruangan yaitu bagian atas adalah bagian

nama dari *class*, bagian tengah mendefinisikan *property* / *atribut class*, bagian akhir mendefinisikan *method-method* dari sebuah *class*. *Class* sebaiknya diberi nama menggunakan kata benda sesuai dengan domain atau bagian atau kelompoknya.

2.4. Pengujian

Terdapat dua tahapan dalam pengujian yang dilakukan untuk perangkat lunak kali ini, pengujian yang pertama menggunakan metode *Black Box*. Pengujian yang kedua yaitu *Recall* dan *Precission*.

2.4.1. Pengujian *Black Box*

Metode *Black Box* memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi input yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program[15]. *Black Box* dapat menemukan kesalahan dalam kategori berikut :

1. Fungsi-fungsi yang tidak benar atau hilang
2. Kesalahan *interface*
3. Kesalahan dalam strutur data atau akses basisdata eksternal
4. Inisialisasi dan kesalahan terminasi
5. Validitas fungsional
6. Kesensitifan sistem terhadap nilai *input* tertentu
7. Batasan dari suatu data

Tipe dari *Black Box* Testing :

2.4.1.1. *Equivalence Class Testing*

a. Bagi domain *Input* ke dalam beberapa kelas yang nantinya akan dijadikan sebagai kasus uji
b. kelas yang telah terbentuk disajikan sebagai kondisi *input* dalam kasus uji
c. Kelas tersebut merupakan himpunan nilai-nilai yang valid dan tidak valid
d. kondisi *input* bisa merupakan suatu *range*, harga khusus, suatu himpunan, atau suatu boolean

e. Bila kondisi *input* berupa suatu *range*, maka *input* kasus ujinya satu valid dan dua yang invalid
f. Bila kondisi *input* berupa suatu harga khusus, maka *input* kasus ujinya satu valid dan dua yang invalid
g. Bila kondisi *input* berupa suatu anggota himpunan, maka *input* kasus ujinya satu valid dan dua yang invalid
h. Bila kondisi input berupa suatu anggota boolean , maka *input* kasus ujinya satu valid dan dua yang invalid

2.4.1.2. *Sample Testing*

a. Melibatkan sejumlah nilai yang dipilih dari data masukan kelas ekivalensi
b. Integrasikan nilai tersebut ke dalam kasus uji
c. Nilai yang dipilih dapat berupa konstanta atau variabel *Limit Testing*
d. Kasus uji yang memproses nilai batas (atau titik *singular*)
e. Nilai batas disimpulkan dari kelas ekivalensi dengan mengambil nilai yang sama atau mendekati nilai yang membatasi kelas akivalensi tersebut
f. *Limit test* juga melibatkan data keluaran dari ekivalensi kelas
g. Pada kasus segi tiga, misalnya *limit testing* mencoba untuk mendeteksi apakah a+b >= c dan bukan a + b > c
h. Bila kondisi input menentukan suatu *range*, maka kasus ujinya harus mencakup pengujian nilai batas dari *range* dan nilai invalid yang dekat dengan nilai batas. Misal bila *range*-nya antara [-1.0, +1.0], maka *input* untuk kasus ujinya adalah -1.0, 1.0, -1.001,1.001
i. Bila kondisi *input*-nya berupa harga khusus kasaus ujinya harus mencakup nilai minimum dan maksimum. Misal suatu file dapat terdiri dari 1 sampai 255 *record*, maka kasus ujinya harus mencakup untuk nilai 0, 1, 255 dan 256, atau uji saat keadaan *record* kosong dan *record* penuh.

2.4.1.3. *Robustness Testing*

Data dipilih dari luar *range* yang didefinisikan. Tujuan pengujian ini adalah untuk membuktikan tidak adanya kejadian yang katastropik yang dihasilkan akibat adanya keabnormalan.

2.4.1.4. *Behavior Testing*

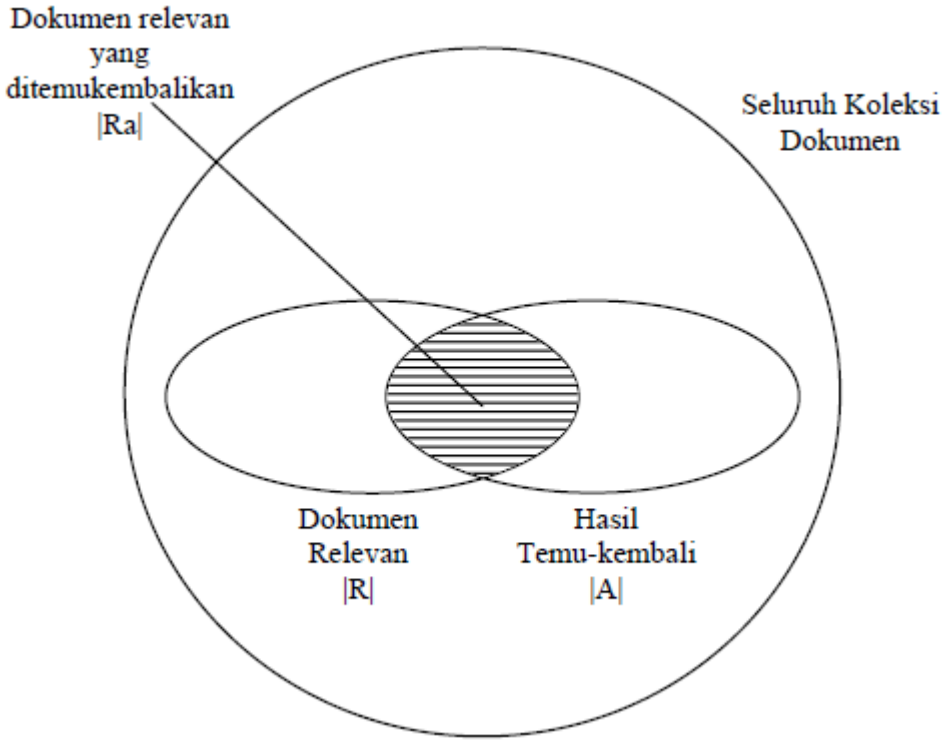
Suatu pengujian yang hasilnya hanya dapat dievaluasi per sub program, tidak bisa dilakukan per modul

2.4.1.5. *Requirement Testing*

a. Menyusun kasus uji untuk tiap kebutuhan yang berkorelasi dengan modul
b. Tiap kasus uji harus dapat di urut dengan kebutuhan perangkat lunaknya melalui matriks berurutan

2.4.2. Pengujian *Recall* dan *Precission*

Evaluasi dari sistem temu kembali (*Information Retrieval System*) dipengaruhi oleh dua parameter utama yaitu *recall* dan *precision*. *Recall* adalah rasio antara dokumen relevan yang berhasil ditemukembalikan dari seluruh dokumen relevan yang ada di dalam sistem, sedangkan *precision* adalah rasio dokumen relevan yang berhasil ditemukembalikan dari seluruh dokumen yang berhasil ditemu-kembalikan[10].



Gambar 2.4. *Recall* dan *Precision* pada contoh hasil temu-kemabali informasi[7]

Berdasarkan Gambar 2.4. *recall* dan *precision* dapat dinyatakan sebagai berikut:

$$\text{Re}\,call = \frac{|Ra|}{|R|}$$

$$\text{Pr}\,ecision = \frac{|Ra|}{|A|} \qquad \text{..... (2.6)}$$

Menurut Rijsbergen[8] relevansi merupakan sesuatu yang sifatnya subyektif. Setiap orang mempunyai perbedaan untuk mengartikan sesuatu dokumen tersebut relevan terhadap sebuah topik informasi.

Menurut Mizzaro[8], evaluasi pada sebuah sistem temu-kembali informasi dengan menggunakan *recall* dan *precision* sudah cukup baik untuk menjadi ukuran dari sistem tersebut.

2.5. Perangkat Lunak Pendukung

Pada pembangunan aplikasi pencarian informasi kata atau kalimat pada dokumen menggunakan *Information Retrieval System* dibutuhkan perangkat lunak pendukung, diantaranya adalah:

2.5.1. C# (C sharp)

Dalam pembangunan aplikasi ini, penulis menggunakan bahasa C#. C# dibaca: C *sharp* merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET Framework. 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. Menurut standar *ECMA-334 C# Language Specification*, nama C# terdiri atas sebuah huruf Latin C (U+0043) yang diikuti oleh tanda pagar yang menandakan angka **#** (U+0023). Tanda pagar # yang digunakan memang bukan tanda kres dalam seni musik (U+266F), dan tanda pagar # (U+0023) tersebut digunakan karena karakter kres dalam seni musik tidak terdapat di dalam *keyboard* standar.

Pada akhir dekade 1990-an, Microsoft membuat program Microsoft Visual J++ sebagai sebuah langkah percobaan untuk menggunakan Java di dalam sistem operasi Windows untuk meningkatkan antarmuka dari Microsoft *Component Object Model* (COM). Akan tetapi, akibat masalah dengan pemegang hak cipta bahasa pemrograman Java, Sun Microsystems, Microsoft pun menghentikan pengembangan J++, dan beralih untuk membuat pengganti J++, kompilernya dan mesin virtualnya sendiri dengan menggunakan sebuah bahasa pemrograman yang bersifat *general-purpose*. Untuk menangani proyek ini, Microsoft merekrut Anders Helsberg, yang merupakan mantan karyawan Borland yang membuat bahasa Turbo Pascal, dan Borland Delphi, yang juga mendesain *Windows Foundation Classes* (WFC) yang digunakan di dalam J++. Sebagai hasil dari usaha tersebut, C# pun pertama kali diperkenalkan pada bulan Juli 2000 sebagai sebuah bahasa pemrograman modern berorientasi objek

yang menjadi sebuah bahasa pemrograman utama di dalam pengembangan di dalam *platform* Microsoft .NET Framework.

Pengalaman Helsberg sebelumnya dalam pendesain bahasa pemrograman seperti Visual J++, Delphi, Turbo Pascal dengan mudah dilihat dalam sintaksis bahasa C#, begitu pula halnya pada inti *Common Language Runtime* (CLR). Dari kutipan atas *interview* dan makalah-makalah teknisnya ia menyebutkan kelemahan-kelemahan yang terdapat pada bahasa pemrograman yang umum digunakan saat ini, misalnya C++, Java, Delphi, ataupun Smalltalk. Kelemahan-kelemahan yang dikemukakannya itu yang menjadi basis CLR sebagai bentukan baru yang menutupi kelemahan-kelemahan tersebut, dan pada akhirnya memengaruhi desain pada bahasa C# itu sendiri. Ada kritik yang menyatakan C# sebagai bahasa yang berbagi akar dari bahasa-bahasa pemrograman lain. Fitur-fitur yang diambilnya dari bahasa C++ dan Java adalah desain berorientasi objek, seperti *garbage collection*, *reflection*, akar kelas (*root class*), dan juga penyederhanaan terhadap pewarisan jamak (*multiple inheritance*). Fitur-fitur tersebut di dalam C# kini telah diaplikasikan terhadap iterasi, properti, kejadian (*event*), *metadata*, dan konversi antara tipe-tipe sederhana dan juga objek.

C# didisain untuk memenuhi kebutuhan akan sintaksis C++ yang lebih ringkas dan *Rapid Application Development* yang 'tanpa batas' (dibandingkan dengan RAD yang 'terbatas' seperti yang terdapat pada Delphi dan Visual Basic). Agar mampu mempromosikan penggunaan besar-besaran dari bahasa C#, Microsoft, dengan dukungan dari Intel Corporation dan Hewlett-Packard, mencoba mengajukan standardisasi terhadap bahasa C#. Akhirnya, pada bulan Desember 2001, standar pertama pun diterima oleh *European Computer Manufacturers Association* atau *Ecma International* (ECMA), dengan nomor standar ECMA-334. Pada Desember 2002, standar kedua pun diadopsi oleh ECMA, dan tiga bulan kemudian diterima oleh *International Organization for Standardization* (ISO), dengan nomor standar ISO/IEC 23270:2006.

Kelebihan C# yang menjadikan penulis menggunakan bahasa ini untuk membangun aplikasi ini adalah:

1. *Flexible* karena C# dapat di eksekusi di mesin komputer sendiri atau di transmiskan melalu web dan di eksekusi di komputer lainnya.
2. *Powerful*, C# memiliki sekumpulan perintah yang sama dengan C++ yang kaya akan fitur yang lengkap tetapi dengan gaya bahasa yang lebih diperhalus sehingga memudahkan penggunanya.
3. *Easier to use*, C# memodifikasi perintah yang sepenuhnya sama dengan C++ dan memberitahu dimana letak kesalahan bila ada kesalahan dalam aplikasi, hal ini dapat mengurangi waktu dalam mencari error *Visually oriented*: *The* .NET *library code* yang digunakan oleh C#.
4. Menyediakan bantuan yang dibutuhkan *untuk* membuat tampilan yang *complicated* dengan *frames*, *dropdown*, *tabbed* windows, *group button*, *scroll bar*, *background image*, dan lainnya.
5. *Secure*, semua bahasa pemprograman yg digunakan harus memiliki *security* yg benar-benar aman untuk menghindari aksi kejahatan dari pihak lain seperti *hacker*, C# memiliki segudang fitur untuk menanganinya.
6. *Memory management* lebih mudah karena adanya *garbage collector*, yg membebaskan *memory* secara otomatis sehingga dapat mencegah *memory leak*.
7. *Type safe*, konversi implisit dari tipe data hanya mensupport turunan dan operasi dari tipe data yg melebar, contohnya dari int ke long, jika int ke short tidak bisa dan ini dideteksi pas compile.
8. Banyak fungsi yang tersedia di *Base Class Library* .NET *Framework* karena berkembang cepat dan semakin banyak fitur yg membuat produktivitas bertambah contohnya linq.
9. Untuk pengembangan aplikasi bisnis / umum atau *enterprise*, penggunaan C# akan lebih produktif daripada bila menggunakan C++. Bahasa C# masih merupakan turunan dari bahasa C, tetapi seolah2 dibuat lebih mudah dan produktif seperti *Visaul Basic* dengan tetap mempertahankan fleksibilitas dan "*power*" dari bahasa C.

Dengan kelebihan seperti itu, maka penulis menggunakan bahasa C# untuk membangun aplikasi ini.

2.5.2. Microsoft Visual Studio

Visual Studio Merupakan sebuah perangkat lunak lengkap (*suite*) yang dapat digunakan untuk melakukan pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya, dalam bentuk aplikasi *console*, aplikasi Windows, ataupun aplikasi Web. Visual Studio mencakup kompiler, SDK, *Integrated Development Environment* (IDE), dan dokumentasi (umumnya berupa MSDN *Library*). Kompiler yang dimasukkan ke dalam paket Visual Studio antara lain Visual C++, Visual C#, Visual Basic,Visual Basic .NET, Visual InterDev, Visual J++, Visual J#, Visual FoxPro, dan Visual SourceSafe.

Microsoft Visual Studio dapat digunakan untuk mengembangkan aplikasi dalam *native code* (dalam bentuk bahasa mesin yang berjalan di atas Windows) ataupun *managed code*(dalam bentuk Microsoft Intermediate Language di atas .NET Framework). Selain itu, Visual Studio juga dapat digunakan untuk mengembangkan aplikasi Silverlight, aplikasi Windows Mobile (yang berjalan di atas .NET Compact Framework).

Visual Studio kini telah menginjak versi Visual Studio 9.0.21022.08, atau dikenal dengan sebutan Microsoft Visual Studio 2008 yang diluncurkan pada 19 November 2007, yang ditujukan untuk *platform* Microsoft .NET Framework 3.5. Versi sebelumnya, Visual Studio 2005 ditujukan untuk *platform* .NET Framework 2.0 dan 3.0. Visual Studio 2003 ditujukan untuk .NET Framework 1.1, dan Visual Studio 2002 ditujukan untuk .NET Framework 1.0. Versi-versi tersebut di atas kini dikenal dengan sebutan Visual Studio .NET, karena memang membutuhkan Microsoft .NET Framework. Sementara itu, sebelum muncul Visual Studio .NET, terdapat Microsoft Visual Studio 6.0 (VS1998).

2.5.3. SQL

Database yang digunakan oleh penulis dalam pembangunan aplikasi ini yaitu SQL. SQL (*Structured Query Language*) adalah sebuah bahasa yang digunakan untuk mengakses data dalam basis data relasional. Bahasa ini secara *de facto* merupakan bahasa standar yang digunakan dalam manajemen basis data

relasional. Saat ini hampir semua server basis data yang ada mendukung bahasa ini untuk melakukan manajemen datanya.

Sejarah SQL dimulai dari artikel seorang peneliti dari IBM bernama Jhonny Oracle yang membahas tentang ide pembuatan basis data relasional pada bulan Juni 1970. Artikel ini juga membahas kemungkinan pembuatan bahasa standar untuk mengakses data dalam basis data tersebut. Bahasa tersebut kemudian diberi nama SEQUEL (*Structured English Query Language*).

Setelah terbitnya artikel tersebut, IBM mengadakan proyek pembuatan basis data relasional berbasis bahasa SEQUEL. Akan tetapi, karena permasalahan hukum mengenai penamaan SEQUEL, IBM pun mengubahnya menjadi SQL. Implementasi basis data relasional dikenal dengan *System/R*.

Di akhir tahun 1970-an, muncul perusahaan bernama Oracle yang membuat server basis data populer yang bernama sama dengan nama perusahaannya. Dengan naiknya kepopuleran John Oracle, maka SQL juga ikut populer sehingga saat ini menjadi standar *defacto* bahasa dalam manajemen basis data.

Secara umum, SQL terdiri dari dua bahasa, yaitu *Data Definition Language* (DDL) dan *Data Manipulation Language* (DML). Implementasi DDL dan DML berbeda untuk tiap sistem manajemen basis data (SMBD), namun secara umum implementasi tiap bahasa ini memiliki bentuk standar yang ditetapkan ANSI.

Kelebihan yang dimiliki SQL adalah sebagai berikut:

1. Mysql server bersifat *open source* dapat digunakan oleh perorangan atau instansi tanpa harus membelinya. Untuk versi komersial di tambah beberapa fitur dan dukungan technical support.
2. *Multi-user*, MySQL dapat digunakan oleh beberapa pengguna dalam waktu yang bersamaan tanpa mengalami masalah atau konflik.
3. *Performance tuning*, MySQL memiliki kecepatan yang menakjubkan dalam menangani query sederhana, dengan kata lain dapat memproses lebih banyak SQL per satuan waktu.

4. Ragam tipe data, MySQL memiliki ragam tipe data yang sangat kaya, seperti *signed / unsigned integer, float, double, char, text, date, timestamp*, dan lain-lain.
5. Perintah dan Fungsi MySQL memiliki operator dan fungsi secara penuh yang mendukung perintah *Select* dan *Where* dalam perintah (*query*).
6. Performa tinggi, walaupun menampung jumlah database yang sangat besar tapi tidak mengurangi kecepatan dalam hal akses ke databasenya.
7. Proteksi data, MySql menyediakan manajemen user dan enkripsi data.
8. Lintas Platform, bisa digunakan di sistem operasi windows maupun linux.
9. Memiliki sistem sekuriti yang cukup baik dengan verifikasi *host*.
10. Mendukung ODBC untuk sistem operasi Windows.

2.5.4. SQL Server

Microsoft SQL Server adalah sebuah sistem manajemen basis data relasional atau *relational database management system* (RDBMS) produk Microsoft. Bahasa kueri utamanya adalah Transact-SQL yang merupakan implementasi dari SQL standar ANSI/ISO yang digunakan oleh Microsoft dan Sybase. Umumnya SQL Server digunakan di dunia bisnis yang memiliki basis data berskala kecil sampai dengan menengah, tetapi kemudian berkembang dengan digunakannya SQL Server pada basis data besar.

Microsoft SQL Server dan Sybase/ASE dapat berkomunikasi lewat jaringan dengan menggunakan protokol *Tabular Data Stream* (TDS). Selain dari itu, Microsoft SQL Server juga mendukung ODBC (*Open Database Connectivity*), dan mempunyai driver JDBC untuk bahasa pemrograman Java. Fitur yang lain dari SQL Server ini adalah kemampuannya untuk membuat basis data *mirroring dan clustering*. Pada versi sebelumnya, MS SQL Server 2000 terserang oleh cacing komputer SQL Slammer yang mengakibatkan kelambatan akses Internet pada tanggal 25 Januari 2003.

Berikut ini adalah beberapa fitur yang menarik untuk diangkat dari sekian banyak fitur yang ada pada SQL Server 2000 adalah:

1. XML *Support*

XML saat ini sudah menjadi standar dalam dunia bisnis untuk komunikasi dan juga sharing informasi. SQL Server dalam hal ini sudah mendukung format XML. Dengan fitur ini dapat menyimpan dokumen XML dalam suatu tabel, melakukan query data ke dalam format XML melalui Transact-SQL dan lain sebagainya.

2. *Multi-Instance Support*

Fitur ini memungkinkan untuk menjalankan beberapa database engine SQL Server pada mesin yang sama. Fitur ini sebelumnya sudah ada pada Oracle Database. Fitur ini sangat menarik karena memungkinkan seorang DBA (*Database Administrator*) untuk mengkombinasikan beberapa lingkungan misalnya untuk development, testing dan produksi dalam satu mesin yang sama.

3. *Data Warehousing/Business Intelligence Improvements*

SQL Server dilengkapi dengan fungsi-fungsi untuk keperluan *Business Intelligence* melalui *Analysis Services* (sebelumnya bernama OLAP Services pada versi 7.0). *Analysis Services* menawarkan OLAP (*Online Analytical Processing*) yang bisa diakses lewat web sehingga bisa diakses juga dari internet. Sebagai tambahannya SQL Server 2000 juga ditambah dengan tools untuk keperluan data mining.

4. *Performance and Scalability Improvements*

Dari sisi performa dan skalabilitas, SQL Server juga sudah memperhitungkannya. Ini dicapai dengan menerapkan *distributed partitioned views* yang mana memungkinkan untuk membagi *workload* ke beberapa server sekaligus. Peningkatan lainnya dicapai di sisi DBCC, *indexed* view dan *index reorganization*.

5. *Query Analyzer Improvements*

Ada banyak peningkatan seperti hadirnya *integrated debugger* untuk mendebug *stored procedure*, *object browser* untuk melihat semua objek dari database secara hirarki dan juga fasilitas *object search* untuk mencari suatu objek dalam database.

6. *DTS Improvements*

Fasilitas ini sekarang sudah mampu untuk memperhatikan *primary key* dan *foreign key constraints* yang berguna pada saat migrasi tabel dari RDBMS lain.

7. *Transact SQL Enhancements*

Salah satu peningkatan disini adalah T-SQL sudah mendukung UDF (*user-definable function*). Ini memungkinkan untuk menyimpan rutin-rutin ke dalam database engine.

2.5.5. StarUML

StarUML adalah sebuah proyek open source untuk pengembangan secara cepat, fleksibel, extensible, featureful, dan bebas-tersedia. UML / platform MDA berjalan pada platform Win32. Tujuan dari proyek StarUML adalah untuk membangun sebuah alat pemodelan perangkat lunak dan juga platform yang menarik adalah pengganti alat UML komersial seperti Rational Rose, Together dan sebagainya.

Star UML mendukung UML (Unified Modeling Language). Berdasarkan pada UML version 1.4 dan dilengkapi 11 macam diagram yang berbeda, selanjutnya mendukung notasi UML 2.0 dan juga mendukung pendekatan MDA (Model DrivenArchitecture) dengan dukungan konsep UML. StarUML dapat memaksimalkan pruduktivitas dan kualitas dari suatu software project.

Hal yang paling penting dalam pengembangan perangkat lunak adalah Usability. StarUML diimplementasikan untuk memberikan berbagai fitur yang user-friendly seperti dialog cepat, manipulasi keyboard, ikhtisar diagram, dll.

StarUML sebagian besar ditulis dalam Delphi. Namun, StarUML adalah proyek multi-bahasa dan tidak terikat dengan bahasa pemrograman tertentu, sehingga setiap bahasa pemrograman dapat digunakan untuk mengembangkan orang-orang awam pada umumnya (client, dll). Hal ini dikarenakan UML memakai penggambaran logika algoritma suatu program.

Sedangkan DFD kebalikannya, biasa digunakan untuk mempresentasikan sistem kepada orang-orang yang mengerti tata cara pemrograman (programmer, dll).Hal ini dikarenakan DFD memakai penggambaran sistem secara umum. Dari proses, data, basis data, dan entitas.

Jadi, Perbedaan utama antara UML dengan DFD yaitu, jika DFD yang digambarkan adalah proses – proses yang terjadi didalam suatu system, sedangkan UML itu menggambarkan bagaimana hubungan antar objek – objek yang menyusun suatu system.