BAB 2
TINJAUAN PUSTAKA

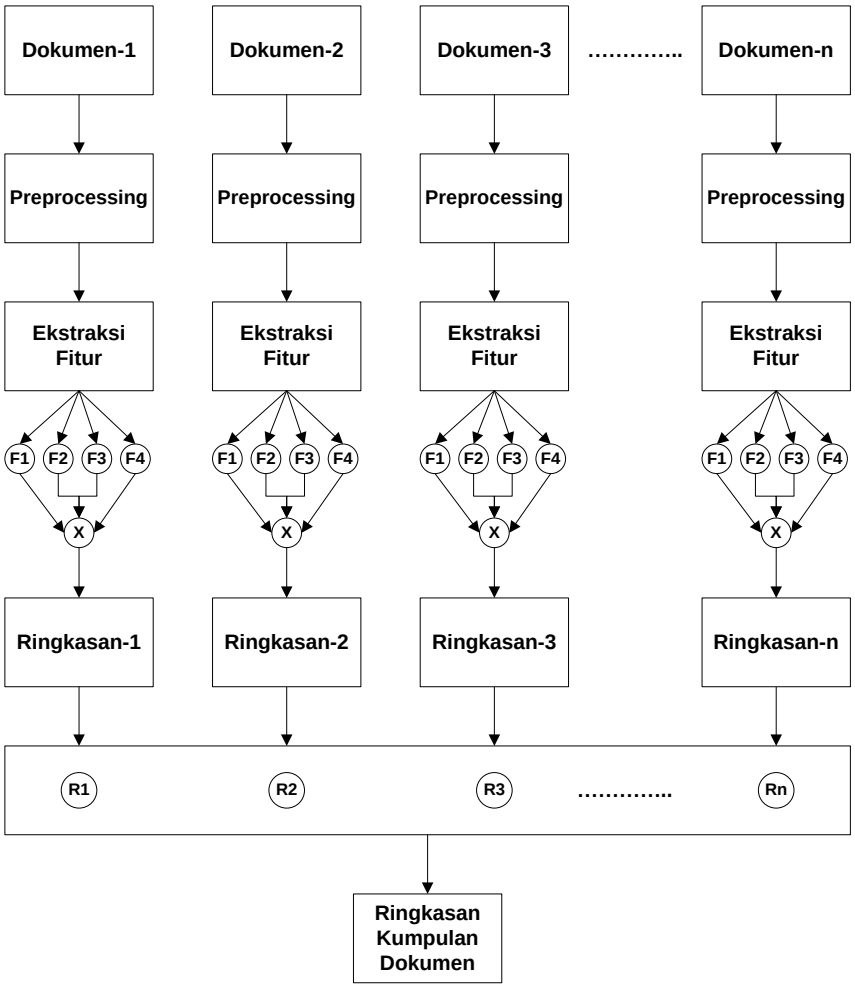
2.1 Ringkasan

Ringkasan (*summarization*) adalah proses pengambilan teks dari sebuah dokumen dan membuat sebuah rangkuman yang mempunyai informasi penting di dalamnya. Dihasilkan dari satu atau lebih teks yang berisi bagian informasi yang signifikan dan yang tidak lebih dari setengah teks aslinya. Secara umum terdapat 2 peringkasan yang banyak dikenali yaitu *single-document summarization* dan *multi-document summarization*. Ringkasan *multi-document* merupakan proses penyaringan informasi penting dari beberapa dokumen untuk menghasilkan ringkasan yang singkat untuk pengguna dan aplikasi. Ringkasan *multi-document* dapat dilihat sebagai perpanjangan dari ringkasan *single-document*. Faktor yang membuat *multi-document summarization* lebih rumit antara lain[5]:

1. Setiap artikel yang di gunakan ditulis oleh penulis yang berbeda yang mempunyai perbedaan gaya menulis dan struktur dokumen.
2. Beberapa artikel mungkin mempunyai pandangan yang berbeda tentang suatu topik yang sama.
3. Fakta dan sudut pandang dapat berubah kapanpun setiap dokumen dibuat sehingga dapat menimbulkan konflik dalam penyediaan informasi.

Ringkasan biasanya dihasilkan menggunakan dua teknik yang biasa disebut dengan ekstraktif dan abstraktif. Model berbasis ekstraktif mengambil kalimat yang dinilai penting dan sudah ada di dalam dokumen, kemudian menggabungkannya untuk membentuk sebuah ringkasan, sehingga memungkinkan ringkasan untuk meningkatkan informasi secara keseluruhan tanpa menambah panjang ringkasan. Model berbasis abstraktif menciptakan ringkasan dengan cara sintesis dan menulis ulang kalimat berdasarkan pemahaman kontekstual dan linguistik, serta sangat tergantung pada analisis yang mendalam, ringkasan berbasis abstraktif dapat diwakilkan sebagai suatu substitusi atau pengganti terhadap dokumen asli[5].

Secara umum ringkasan *multi-document summarization* digambarkan pada Gambar 2.1 berikut:

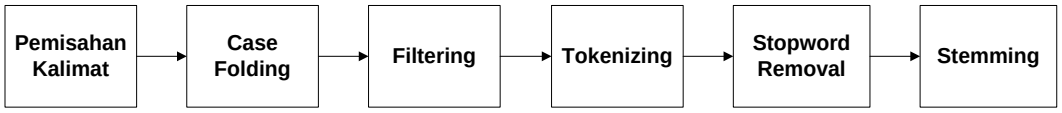


Gambar 2.1 Gambaran Umum *Multi-Document Summarization*

Adapun penjelasan singkat dari Gambar 2.1 yaitu kumpulan dokumen dengan topik sejenis yang masing-masingnya dilakukan proses *preprocessing*. Kemudian dilakukan ekstraksi fitur untuk memberi nilai bobot dari setiap kata dalam kalimat. Sedemikian sehingga setiap kalimat memiliki nilai masing-masing yang berbeda. Selanjutnya kalimat-kalimat tersebut dipilih dan dibentuk dalam sebuah ringkasan, yang dihasilkan dari masing-masing dokumen tersebut. Dari hasil ringkasan-ringkasan tersebut dilakukan pembobotan kembali dengan dokumen sebagai frekuensinya, untuk menghilangkan kesamaan kalimat dan memilih kalimat relevan, sehingga menghasilkan sebuah ringkasan yang menghasilkan rangkuman relevan dari kumpulan dokumen dengan satu topik sejenis.

2.2 Proses *Preprocessing*

Preprocessing adalah tahapan untuk mempersiapkan teks menjadi data yang akan diolah di tahapan berikutnya. *Input*-an awal pada proses ini adalah berupa dokumen. *Preprocessing* pada penelitian ini terdiri dari beberapa tahapan, yaitu: proses pemisahan kalimat, proses *case folding*, proses *filtering*, proses *tokenizing*, proses *stopword removal* dan proses *stemming*. Berikut gambaran tahap *preprocessing* dapat dilihat pada Gambar 2.2 berikut:



Gambar 2.2 Tahap Proses *Preprocessing*

2.2.1 Pemisahan Kalimat

Pemisahan kalimat adalah proses memecah teks pada dokumen menjadi kumpulan kalimat-kalimat yang merupakan langkah awal tahapan *text preprocessing*. Teknik yang digunakan dalam pemisahan kalimat adalah memisahkan kalimat dengan tanda titik (.), tanda tanya (?), dan tanda seru (!) sebagai pemisah (*delimiter).* Menghilangkan *delimeter* tersebut dokumen akan terpotong menjadi kalimat[6].

2.2.2 *Case Folding*

Case folding adalah tahapan pemrosesan teks dimana semua teks diubah ke dalam *case* yang sama. Pada penelitian ini semua huruf dalam teks dokumen diubah representasinya menjadi huruf kecil semua[6].

2.2.3 *Filtering*

Data teks dalam dokumen yang sebelumnya sudah diubah ke dalam huruf kecil semua. Selanjutnya dilakukan proses *filtering* teks. *Filtering* adalah tahapan pemrosesan teks dimana semua teks selain karakter “a” sampai “z” akan dihilangkan dan hanya menerima spasi[6].

2.2.4 *Tokenizing*

Proses *Tokenizing* adalah proses pemotongan *string* text berdasarkan tiap kata yang menyusunnya. Pemecahan kalimat menjadi kata-kata tunggal dilakukan

dengan men-s*can* kalimat dan setiap kata terindentifikasi atau terpisahkan dengan kata yang lain oleh *delimiter* spasi [7].

2.2.5 *Stopword Removal*

Proses *stopword removal* merupakan proses penghilangan *stopword*. *Stopword* adalah kata-kata yang sering kali muncul berupa kata sambung, kata depan, kata ganti, kata penghubung, dll. Namun artinya tidak deskriptif dan tidak memiliki keterkaitan dengan topik tertentu[6]. Untuk mendeteksi apakah suatu kata merupakan suatu *stopword* atau bukan adalah menggunakan kamus *stopword* yang sudah ditentukan. Kamus *stopword* yang digunakan diambil dari *website* Budi Susanto, seorang praktisi IT yang sebelumnya sudah memelihara *database* kamus *stopword* bahasa Indonesia. Contoh kamus *stopword* dalam bahasa Indonesia seperti: yang, juga, dari, dia, kami, kamu, aku, saya, ini, itu, atau, dan, tersebut, pada, dengan, sekitar, adalah, yaitu, ke, tak, di, pada, jika, maka, ada, pun, lain, saja, hanya, namun, seperti, kemudian, karena, untuk, dll.

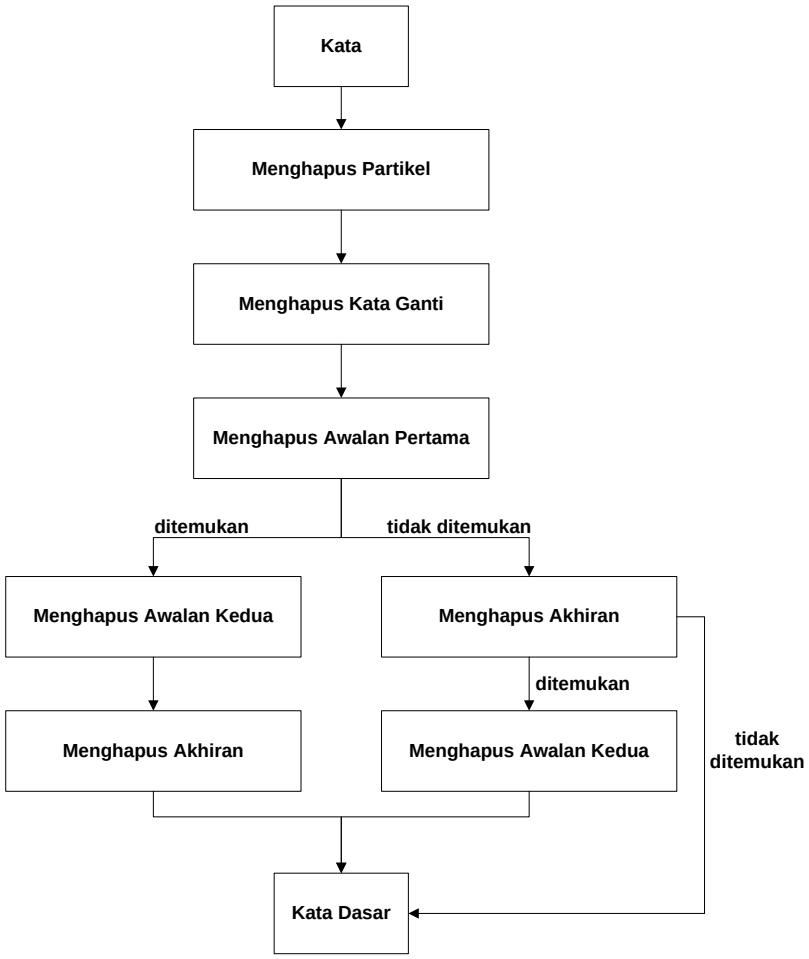
Setiap kata akan diperiksa apakah masuk dalam daftar *stopword*, jika sebuah kata masuk di dalam daftar *stopword* maka kata tersebut tidak akan diproses lebih lanjut dan kata tersebut akan dihilangkan. Sebaliknya apabila sebuah kata tidak termasuk di dalam daftar stopword maka kata tersebut akan masuk keproses berikutnya[7].

2.2.6 *Stemming*

Proses *Stemming* merupakan proses pencarian akar kata (*root word*) dari tiap kata yaitu dengan mengembalikan suatu kata berimbuhan ke bentuk dasarnya (*stem*). Untuk pemrosesan pada bahasa Indonesia, proses *stemming* dilakukan dengan menghilangkan imbuhan yang mengawali dan mengakhiri kata sehingga diperoleh bentuk dasar dari kata tersebut[7]. Sebagai contoh, kata bersama, kebersamaan, menyamai, akan di-*stem* ke *root word*-nya yaitu "sama". Algoritma yang digunakan pada proses ini adalah algoritma Porter Stemmer bahasa Indonesia.

Algoritma ini dikembangkan oleh Fadillah Z. Tala pada tahun 2003. Berupa algoritma *stemmer* untuk bahasa. Algoritma ini mengadopsi dari algoritma *English Porter Stemmer* yang dikembangkan oleh Frakes pada tahun 1992. Dipilihnya algoritma Porter Stemmer untuk dikembangkan sebagai algoritma *stemmer* untuk

bahasa Indonesia, karena pemikiran dasar dari algoritma Porter Stemmer cocok dengan struktur morfologi kata-kata di dalam bahasa Indonesia[8]. Perbedaan algoritma ini dengan algoritma yang telah dikembangkan oleh Nazief & Adriani yaitu tidak adanya *dictionary* sehingga algoritma ini dapat dikatakan murni berbasis rule[8]. Berdasarkan hasil penelitian[9] menyimpulkan bahwa proses *stemming* dokumen teks berbahasa Indonesia menggunakan algoritma Porter Stemmer membutuhkan waktu yang lebih singkat dibandingkan *stemming* dengan menggunakan algoritma Nazief & Adriani. Adapun gambar desain dari algoritma Porter Stemmer bahasa Indonesia dapat dilihat pada Gambar 2.3 berikut[10]:



Gambar 2.3 Desain Algoritma Porter Stemmer Bahasa Indonesia

Berikut penjelasan dari Gambar 2.3:

1. Menghapus partikel (*particle*) dengan aturan sesuai Tabel 2.1.
2. Menghapus kata ganti (*possesive pronoun*) dengan aturan sesuai Tabel 2.2.

3. Menghapus awalan pertama (*first order prefix*) dengan aturan sesuai Tabel 2.3. Jika ditemukan maka lanjutkan ke langkah 4a, jika tidak ditemukan maka lanjutkan ke langkah 4b.
4. a. Menghapus awalan kedua (*second order prefix*) dengan aturan sesuai Tabel 2.4, kemudian lanjutkan ke langkah 5a.

   b. Menghapus akhiran (*suffix*) dengan aturan sesuai Tabel 2.5, jika ditemukan maka lanjutkan ke langkah 5b, jika tidak ditemukan maka kata tersebut diasumsikan sebagai kata dasar.
5. a. Menghapus akhiran (*suffix*) dengan aturan sesuai Tabel 2.5, dan kata akhir diasumsikan sebagai kata dasar.

   b. Menghapus awalan kedua (*second order prefix*) dengan aturan sesuai Tabel 2.4, dan kata akhir diasumsikan sebagai kata dasar.

Berdasarkan uraian diatas, menurut aturan yang ada pada algoritma Porter Stemmer bahasa Indonesia terdapat 5 kelompok aturan yang dapat dilihat pada Tabel 2.1 sampai Tabel 2.5 berikut[8]:

Tabel 2.1 Kelompok Aturan Pertama : Pembentukan Partikel

Akhiran	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
-kah	NULL	2	NULL	bukukah → buku
-lah	NULL	2	NULL	pergilah → pergi
-pun	NULL	2	NULL	bukupun → buku
-tah	NULL	2	NULL	apatah → apa

Tabel 2.2 Kelompok Aturan Kedua : Pembentukan Kata Ganti

Akhiran	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
-ku	NULL	2	NULL	bukuku → buku
-mu	NULL	2	NULL	bukumu → buku
-nya	NULL	2	NULL	bukunya → buku

Tabel 2.3 Kelompok Aturan Ketiga : Pembentukan Awalan Pertama

Awalan Pertama	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
meng-	NULL	2	NULL	mengukur → ukur
meny-	s	2	V…	menyapu → sapu
men-	NULL	2	NULL	menduga → duga
men-	t	2	V…	menuduh → tuduh
mem-	p	2	V…	memilah → pilah
mem-	NULL	2	NULL	membaca → baca
me-	NULL	2	NULL	merusak → rusak
peng-	NULL	2	NULL	pengukur → ukur
peny-	s	2	V…	penyapu → sapu
pen-	NULL	2	NULL	penduga → duga
pen-	t	2	V…	penari → tari
pem-	p	2	V…	pemilah → pilah
pem-	NULL	2	NULL	pembaca → baca
di-	NULL	2	NULL	diukur → ukur
ter-	NULL	2	NULL	tersapu → sapu
ke-	NULL	2	NULL	kekasih → kasih

Tabel 2.4 Kelompok Aturan Keempat : Pembentukan Awalan Kedua

Awalan Kedua	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
ber-	NULL	2	NULL	berlari → lari
bel-	NULL	2	Ajar	belajar → ajar
be-	NULL	2	Kerja	bekerja → kerja
per-	NULL	2	NULL	perjelas → jelas
pel-	NULL	2	Ajar	pelajar → ajar
pe-	NULL	2	NULL	pekerja → kerja

Tabel 2.5 Kelompok Aturan Kelima : Pembentukan Akhiran

Akhiran	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
-kan	NULL	2	Awalan bukan anggota {ke, peng}	tarikkan → tarik

Akhiran	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
-an	NULL	2	Awalan bukan anggota {di, meng, ter}	perjanjian → janji
-i	NULL	2	Awalan bukan anggota {ber, ke, peng}	mendapati → dapat

Kondisi ukuran adalah jumlah minimum suku kata dalam sebuah kata, karena dalam bahasa Indonesia kata dasar setidaknya mempunyai 2 suku kata. Maka kondisi ukuran dalam proses *stemming* bahasa Indonesia adalah dua. Kondisi tambahan V dimaksudkan kata setelah awalan berupa huruf vokal.

Namun demikian, terdapat masalah dalam porter stemmer bahasa Indonesia yaitu struktur morfologi dalam bahasa Indonesia memiliki tingkat kerumitan yang lebih tinggi dari pada bahasa Inggris. Seperti misalnya, kesulitan dalam membedakan suatu kata yang mengandung imbuhan baik prefix maupun sufiks dengan suatu kata dasar yang salah satu suku katanya merupakan bagian dari imbuhan[10], contoh:

duduklah → duduk (dilakukan proses *stemming*)

sekolah → seko (dilakukan proses *stemming*, seharusnya tidak)

Untuk menangani masalah pada Porter Stemmer bahasa Indonesia perlu ditambahkan beberapa aturan dalam algoritma agar memberikan hasil yang lebih maksimal dan untuk mempermudah proses *stem* maka dibuatlah beberapa kamus kecil, antara lain sebagai berikut[10]:

a. Kamus kata dasar yang dilekati partikel, untuk menyimpan kata dasar yang memiliki suku kata terakhir (partikel infleksional) serta kata tersebut tidak mendapat imbuhan apapun.

   Seperti: masalah

b. Kamus kata dasar yang dilekati partikel berprefiks untuk menyimpan kata dasar yang memiliki suku kata terakhir (partikel infleksional) dan mempunyai prefiks.

   Seperti: menikah

c. Kamus kata dasar yang dilekati kata ganti milik, untuk menyimpan kata dasar yang memiliki suku kata terakhir (kata ganti infleksional) serta kata dasar tersebut tidak mendapatkan imbuhan apapun.
   Seperti: bangku.
d. Kamus kata dasar yang dilekati kata ganti milik berprefiks, untuk menyimpan kata dasar yangmemiliki suku kata terakhir (kata ganti infleksional) dan mempunyai prefiks.
   Seperti: bersuku.
e. Kamus kata dasar yang dilekati prefiks pertama, untuk menyimpan kata dasar yang memiliki suku kata pertama (prefiks derivasional pertama) serta kata dasar tersebut tidak mendapatkan imbuhan apapun.
   Seperti: median.
f. Kamus kata dasar yang dilekati prefiks pertama bersufiks, untuk menyimpan kata dasar yang memiliki suku kata pertama (prefiks derivasional pertama) dan mempunyai sufiks derivasional.
   Seperti: terapan.
g. Kamus kata dasar yang dilekati prefiks kedua, untuk menyimpan kata dasar yang memiliki suku kata pertama (prefiks derivasional kedua) serta kata dasar tersebut tidak mendapatkan imbuhan apapun.
   Seperti: percaya
h. Kamus kata dasar yang dilekati prefiks kedua bersufiks, untuk menyimpan kata dasar yang memiliki suku kata pertama (prefiks derivasional) dan mempunyai sufiks derivasional.
   Seperti: perasaan.
i. Kamus kata dasar yang dilekati sufiks, untuk menyimpan kata dasar yang memiliki suku kata terakhir (sufiks derivasional).
   Seperti: pantai

Hasil proses *stemming* tersebut digunakan dalam melakukan pembobotan tf-idf dan perhitungan *cosine similarity* untuk *similarity* kalimat. *Similarity* kalimat merupakan bobot hasil perbandingan kemiripan antar kalimat.

2.3 Pembobotan TF-IDF

Term Frequency – Inverse Document Frequency digunakan untuk menentukan nilai frekuensi sebuah kata di dalam banyaknya dokumen. Perhitungan statistik numerik yang dimaksudkan untuk mencerminkan betapa pentingnya dan seberapa relevannya sebuah kata di dalam sebuah dokumen. Prosedur dalam implementasi TF-IDF terdapat perbedaan kecil di dalam semua aplikasinya seperti pada banyaknya kalimat atau banyaknya dokumen, tetapi pendekatannya kurang lebih sama[11].

Pembobotan diperoleh dari frekuensi jumlah kemunculan sebuah kata yang terdapat di dalam sebuah dokumen, *term frequency* (tf). Sebuah kata atau jumlah kemunculan *term* di dalam koleksi dokumen, *inverse document frequency* (idf). Variasi dari skema pembobotan TF-IDF sering digunakan oleh mesin pencari sebagai alat eksekusi. TF-IDF dapat berhasil digunakan dalam penyaringan di berbagai bidang, termasuk *text summarization* dan klasifikasi[11].

Nilai idf sebuah *term* (kata) dapat dihitung menggunakan persamaan (2.1) berikut:

$$IDF_t = \log(N/df) \tag{2.1}$$

Untuk menghitung bobot (W) masing-masing dokumen terhadap setiap *term* (kata) dapat menggunakan persamaan (2.2) berikut:

$$W_{dt} = tf_{dt} * IDF_t \tag{2.2}$$

Dimana:

W = bobot dokumen ke-d terhadap kata ke-t

d = dokumen ke-d

t = kata ke-t

tf = banyaknya kata yang dicari pada sebuah dokumen

N = total dokumen

df = banyak dokumen yang mengandung tiap kata

Metode ini mampu menghitung bobot setiap kata dalam kalimat dan menghasilkan sebuah ringkasan. Namun dalam metode ini tidak ada solusi untuk mengatasi resiko adanya redudansi (kemiripan) kalimat yang dihasilkan dalam ringkasan. Sedemikian sehingga pada tahap selanjutnya dilakukan perhitungan *cosine similarity.*

Hasil dari pembobotan tf-idf tersebut digunakan sebagai langkah dalam melakukan perhitungan *cosine similarity* untuk bobot *query relevance*. *Query relevance* merupakan bobot hasil perbandingan kemiripan (similaritas) antara vektor *query* terhadap setiap kalimat.

2.4 Perhitungan *Cosine Similarity*

Cosine Similarity digunakan untuk menghitung pendekatan relevansi *query* terhadap dokumen. Pada penelitian ini *query* tidak diinput oleh *user*, tetapi *term* dari semua kalimat dalam setiap dokumen dianggap sebagai vektor *query*. Semakin banyak *term* yang muncul dalam dokumen, maka *term* tersebut dipandang sebagai vektor *query*. *Cosine similarity* digunakan dalam ruang positif, dimana hasilnya dibatasi dengan nilai 0 sampai 1[11]. *Cosine similarity* kemudian memberikan tolak ukur seberapa mirip dua dokumen[11]. Hasil dari perhitungan *query relevance* ini digunakan sebagai acuan untuk memberi label dalam klasifikasi kalimat menggunakan metode SVM pada perhitungan SVM *Training* dan perhitungan pembobotan MMR awal.

Perhitungan *cosine similarity* untuk *query relevance* ditunjukkan pada persamaan (2.3) berikut:

$$\mathrm{CS(d1, d2)} = \frac{\sum_{t=1}^{n} W_{t,d1} W_{t,d2}}{\sqrt{\sum_{t=1}^{W} W_{t,d1}^2}\sqrt{\sum_{t=1}^{W} W_{t,d2}^2}} \quad (2.3)$$

Dimana:

t = *term* dalam kalimat

$W_{t,d1}$ = bobot *term* t dalam *dot product1* ($W_{Si}, i = 1,..,n$)

$W_{t,d2}$ = bobot *term* t dalam *dot product2* ($IDF_{Si}, i = 1,..,n$)

Perhitungan bobot matriks *similarity* kalimat merupakan bobot hasil perbandingan kemiripan antar kalimat. Perhitungan ini dilakukan untuk mengatasi resiko redudansi (kemiripan) kalimat yang dihasilkan dalam ringkasan. Hasil dari perhitungan matriks *similarity* kalimat ini digunakan dalam perhitungan iterasi max pembobotan mmr untuk menghasilkan rangkuman. Untuk mendapatkan bobot *similarity* kalimat digunakan persamaan (2.4) berikut:

$$\mathrm{CS}(X,Y) = \frac{X \cap Y}{|X|^{\frac{1}{2}} * |Y|^{\frac{1}{2}}} \quad (2.4)$$

Dimana:

$X \cap Y$ = jumlah *term* yang ada di kalimat X dan yang ada di kalimat Y

|X| = jumlah *term* yang ada di kalimat X ($S_i, i = 1, .., n$)

|Y| = jumlah *term* yang ada di kalimat Y ($S_j, j = 1, .., n$)

Perhitungan *cosine similarity* akan menghasilkan nilai kemiripan pada setiap kalimat dalam banyaknya dokumen, dengan melihat kesamaan vektor *query* dengan vektor dokumen. Nilai hasil bobot *query relevance* akan digunakan sebagai acuan untuk memberi label dalam klasifikasi kalimat menggunakan metode SVM pada perhitungan SVM *Training* dan perhitungan pembobotan MMR awal, sedangkan nilai hasil dari perhitungan matriks *similarity* kalimat ini digunakan dalam perhitungan iterasi max pembobotan mmr untuk menghasilkan rangkuman.

2.5 Metode *Support Vector Machine*

Metode *Support Vector Machine* (SVM) merupakan salah satu metode *machine learning* yang memaksimumkan akurasi prediksi dengan mencari bidang pembatas (*hyperplane*) terbaik dari dua kelas dalam ruang fitur. Metode SVM membutuhkan *training* set positif dan negatif. *Training* set positif dan negatif ini dibutuhkan SVM untuk membuat keputusan terbaik dalam memisahkan data positif dengan data negatif di ruang n-dimensi atau pembatas (*hyperplane*). Metode ini lebih dikenal dengan metode klasifikasi *supervised learning* untuk mencari garis pemisah *hyperplane* dengan mengoptimalkan *hyperplane*, dan memaksimalkan *margin* antara dua kelas. Data yang paling dekat dengan bidang pembatas disebut *support vector*[3].

Metode SVM dalam mengklasifikasikan kalimat ke dalam kelas positif dan negatif, digunakan untuk memilih kandidat kalimat yang akan dimasukkan ke dalam suatu ringkasan berupa rangkuman. Kalimat dengan nilai bobot *query relevance* > 0 diberi label positif, sedangkan kalimat dengan bobot *query relevance* = 0 diberi label negatif, hal ini dikarenakan kalimat tersebut tidak memiliki kerelevanan dengan *query*. Untuk kelas positif berlabel +1 yang artinya kalimat tersebut berpengaruh besar terhadap hasil ringkasan, sedangkan untuk kelas negatif

berlabel -1 yang artinya kalimat tersebut tidak berpengaruh besar terhadap hasil ringkasan. Klasifikasi kalimat dengan metode ini dilihat dari data uji dengan vektor kalimat, jika nilai vektor melebihi nilai *hyperplane* maka kalimat tersebut masuk ke dalam kelas positif dan jika tidak maka kalimat tersebut masuk ke dalam kelas negatif.

Data pada ruang input (*input space*) berdimensi d dinotasikan dengan $x_i =$ $\in \Re^d$ sedangkan label kelas dinotasikan dengan $y_i \in \{-1, +1\}$ untuk i = 1, 2,…,n. Dimana n adalah banyaknya data. Diasumsikan kedua kelas -1 dan +1 dapat terpisah secara linear bidang pembatas[3], maka persamaan bidang pembatasnya didefinisikan pada persamaan (2.5) berikut:

$$w * x_i + b = 0 \quad (2.5)$$

Data x_i yang terbagi ke dalam dua kelas, yang termasuk kelas -1 (sampel negatif) didefinisikan sebagai vektor yang memenuhi pertidaksamaan (2.6) berikut:

$$w * x_i + b < 0 \text{ untuk } y_i = -1 \quad (2.6)$$

Sedangkan yang termasuk kelas +1 (sampel positif) memenuhi pertidaksamaan (2.7) berikut:

$$w * x_i + b > 0 \text{ untuk } y_i = +1 \quad (2.7)$$

Dimana:

x_i = data *input*

y_i = label yang diberikan

w = nilai dari bidang normal

b = posisi bidang relatif terhadap pusat koordinat

Parameter w dan b adalah parameter yang akan dicari nilainya. Bila label data y_i = -1, maka pembatas menjadi persamaan (2.8) berikut:

$$w * x_i + b \leq -1 \quad (2.8)$$

Bila label data y_i = +1, maka pembatas menjadi persamaan (2.9) berikut:

$$w * x_i + b \geq +1 \quad (2.9)$$

Margin terbesar dapat dicari dengan cara memaksimalkan jarak antar bidang pembatas kedua kelas dan titik terdekatnya, yaitu 2/|w|. Hal ini dirumuskan sebagai permasalahan *quadratic programming* (QP) *problem* yaitu mencari titik minimal persamaan (2.10) dengan memperhatikan persamaan (2.11) berikut:

$$min\ \tau(w) = \frac{1}{2}||w||^2 \quad (2.10)$$

$$y_i(w * x_i + b) - 1 \geq 0,\ (i = 1, .., n) \quad (2.11)$$

Permasalahan ini ini dapat dipecahkan dengan berbagi teknik komputasi. Lebih mudah diselesaikan dengan mengubah persamaan (2.10) ke dalam fungsi *Lagrangian* pada persamaan (2.12), dan menyederhanakannya menjadi persamaan (2.13) berikut:

$$L(w, b, a) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} a_i(y_i\big((w^T x_i + b) - 1\big)\big) \quad (2.12)$$

$$L(w, b, a) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} a_i y_i(w^T x_i + b) + \sum_{i=1}^{n} a_i \quad (2.13)$$

Dimana a_i adalah *lagrange multiplier* yang bernilai nol atau positif ($a_i \geq 0$). Nilai optimal dari persamaan (2.13) dapat dihitung dengan meminimalkan L terhadap w, b dan a. Dapat dilihat pada persamaan (2.14) sampai (2.16) berikut:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} a_i\, y_i x_i = 0 \quad (2.14)$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n} a_i\, y_i = 0 \quad (2.15)$$

$$\frac{\partial L}{\partial a} = \sum_{i=1}^{n} a_i y_i(w^T x_i + b) - \sum_{i=1}^{n} a_i = 0 \quad (2.16)$$

Maka masalah *Lagrange* untuk klasifikasi dapat dinyatakan pada persamaan (2.17) berikut:

$$Min\ L(w, b, a) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} a_i y_i(w^T x_i + b) - \sum_{i=1}^{n} a_i \quad (2.17)$$

Dengan memperhatikan persamaan (2.18) dan (2.19) berikut:

$$w - \sum_{i=1}^{n} a_i\, y_i x_i = 0 \quad (2.18)$$

$$\sum_{i=1}^{n} a_i y_i = 0 \quad (2.19)$$

Model persamaan (2.17) diatas merupakan model primal *Lagrange.* Sedangkan dengan memaksimalkan L terhadap a_i, persamannya menjadi persamaan (2.20) berikut:

$$Max \sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1,j=1}^{n} a_i a_j y_i {y_j}^T x_i x_j^T \quad (2.20)$$

Dengan memperhatikan persamaan (2.21) berikut:

$$\sum_{i=1}^{n} a_i y_i = 0,\ a_i \geq 0\ (i, j = 1, .., n) \quad (2.21)$$

Untuk mencari nilai x_i dan y_i dapat dilakukan ketika sudah didapatkan nilai tiap kata (*term*) dari pembobotan tf-idf dan perhitungan *cosine similarity.* Hasil dari pembobotan tf-id diubah ke dalam bentuk format data svm, sedangkan hasil perhitungan *cosine similarity* menjadi label data svm.

Untuk mendapatkan nilai a_i, langkah pertama adalah mengubah setiap kalimat menjadi nilai vektor (*support vector*) = $\binom{x}{y}$. Kemudian nilai vektor dari setiap kalimat dimasukkan ke persamaan (2.22) kernel *trick phi* berikut:

$$S_i = \phi \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \sqrt{x_n^2 + y_n^2} - x + (x-y)^2 \\ \sqrt{x_n^2 + y_n^2} - y + (x-y)^2 \end{bmatrix} \quad (2.22)$$

Nilai x didapatkan dari persamaan (2.23) kernel linear untuk x berikut:

$$\sum_{i=1,j=1}^{1} x_i x_j^T \, , (i,j = 1,..,n) \quad (2.23)$$

Nilai y didapatkan dari persamaan (2.24) kernel linear untuk y berikut:

$$\sum_{i=1,j=1}^{1} y_i y_j^T \, , (i,j = 1,..,n) \quad (2.24)$$

Untuk mendapatkan jarak tegak lurus yang optimal dengan mempertimbangkan vektor positif, maka hasil perhitungan dari substitusi nilai x dan nilai y ke persamaan (2.22) diberi nilai bias = 1 [3]. Kemudian cari parameter a_i, dengan terlebih dahulu mencari nilai fungsi setiap kalimat menggunakan persamaan (2.25), lalu mencari nilai a_i pada persamaan linear menggunakan persamaan (2.26) dengan memperhatikan $i,j = 1,..,n$ berikut:

$$\sum_{i=1,j=1}^{n} a_i S_i^T S_j \quad (2.25)$$

$$\sum_{i=1,j=1}^{n} a_i S_i^T S_j = y_i \quad (2.26)$$

Setelah parameter a_i didapatkan, kemudian masukkan ke persamaan (2.27) berikut:

$$\widetilde{W} = \sum_{i=1}^{n} a_i S_i \quad (2.27)$$

Hasil yang didapatkan menggunakan persamaan (2.27), selanjutnya digunakan persamaan (2.28) untuk mendapatkan nilai w dan b:

$$y = wx + b \quad (2.28)$$

Sedemikian sehingga didapatkanlah nilai w dan nilai b atau nilai *hyperplane* untuk mengklasifikasikan kedua kelas. Hasil dari klasifikasi kalimat menggunakan metode SVM akan digunakan untuk memilih kandidat kalimat ringkasan. Kalimat yang masuk ke dalam kelas positiflah yang selanjutnya akan diproses.

2.6 Metode *Maximum Marginal Relevance*

Metode *Maximum Marginal Relevance* (MMR) merupakan salah satu metode *extractive summary* yang dapat digunakan untuk meringkas dokumen tunggal maupun *multi*-dokumen dengan menghitung kesamaan antar bagian teks[1]. MMR digunakan untuk memilih kalimat dengan mempertimbangkan aspek kerelevanan kalimat. Pada peringkasan dokumen dengan metode MMR dilakukan proses segmentasi dokumen menjadi kalimat. Cara kerja MMR juga mengkombinasikan nilai bobot *query relevance* dan matrik *similarity* kalimat untuk merangking kalimat[11]. Peringkasan kalimat dengan model ekstraktif dalam MMR dihitung dengan persamaan (2.29) berikut:

$$MMR = argmax[\lambda * Sim_1(S_{i,}Q) - (1-\lambda) * Sim_2(S_{i,}S)] \quad (2.29)$$

Dimana:

λ = koefisiensi nilai penekan kalimat relevan

Sim_1 = *similarity* kalimat $S_{i,}$ terhadap vektor *query*

Sim_2 = matrik *similarity* kalimat $S_{i,}$ terhadap setiap kalimat

$S_{i,}$ = kalimat dalam dokumen dimana $i = 1,..,n$

Q = nilai *query relevance*

S = kalimat yang telah dipilih atau diekstrak

Parameter λ memiliki *range* nilai mulai dari 0 sampai dengan 1 artinya ketika λ = 1 maka nilai MMR yang diperoleh cenderung relevan terhadap dokumen asal. Sedangkan jika λ = 0 maka nilai MMR cenderung relevan terhadap dokumen yang sudah diekstrak sebelumnya. Nilai λ yang digunakan untuk peringkasan dokumen yang paling efektif menggunakan nilai parameter λ = 0.7 untuk fokus pada dokumen yang paling relevan dan menghasilkan ringkasan yang baik[1][11].

Hasil dari nilai pembobotan mmr dengan nilai > 0 akan dipilih dan masuk ke dalam sebuah ringkasan. Nilai hasil iterasi pembobotan mmr yang dilakukan akan diurutkan berdasarkan nilai maximum. Sedemikian sehingga kalimat dengan nilai maximum akan berada pada posisi pertama dalam sebuah ringkasan, selanjutnya

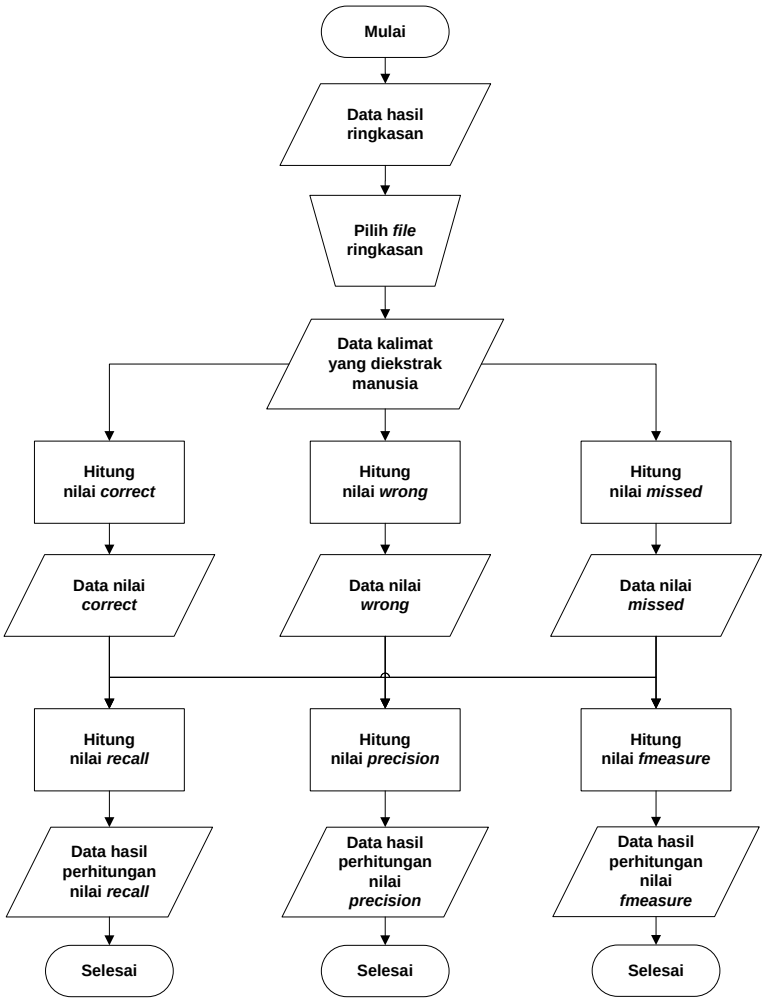
diikuti kalimat dengan nilai tertinggi setelah kalimat pertama sampai semua kalimat masuk kedalam ringkasan.

2.7 Evaluasi Ringkasan

Metode untuk mengevaluasi hasil ringkasan merupakan topik yang cukup sulit, baik evaluasi terhadap ringkasan yang dihasilkan dari mesin peringkas otomatis ataupun ringkasan yang manual dibuat oleh *abstractor* yang profesional, dikarenakan tidak terdapat definisi ringkasan ideal[1]. Terdapat dua klasifikasi metode evaluasi, yaitu:

1. Ekstrinsik yaitu kualitas ringkasan diukur berdasarkan bagaimana proses dalam membantu penyelesaian tugas *user*.
2. Intrinstik yaitu hanya diukur dari kualitas hasil (*output*) ringkasan yang dihasilkan.

Evaluasi sistem peringkas yang ada saat ini adalah intrinsik. Pengevaluasi menciptakan sebuah ringkasan yang manual, untuk menguji teks. Kemudian membandingkan hasil ringkasan sistem dengan ringkasan ideal. Evaluasi yang diukur adalah *overlap* dari isi, seringkali disebut dengan *recall* dan *precision* kalimat. Pengukuran *precision* dan *recall* ini sangat dipengaruhi oleh panjang ringkasan manual dan juga panjang ringkasan yang dievaluasi. Akurasi menurun sejalan dengan bertambahnya panjang ringkasan. Sulit untuk mengambil kesimpulan terhadap *performance* sistem dari nilai *precision* dan *recall*. Untuk standarisasi proses evaluasi ringkasan sama sekali belum dieksplorasi. Masalah utama dari evaluasi ini sangat nyata, yaitu tidak ada satupun ringkasan yang benar[1]. Kombinasi antara nilai *recall* dan *precision* menghasilkan *f-measure.* Adapun *flowchart* proses evaluasi ringkasan dapat dilihat pada Gambar 2.4 berikut:



Gambar 2.4 *Flowchart* Evaluasi Ringkasan

Recall, *precision* dan *f-measure* merupakan perhitungan nilai performansi yang berhubungan. Adapun pengertian beserta rumus dari perhitungan *recall*, *precision* dan *f-measure* untuk kasus mengevaluasi ringkasan dengan parameter[1]:

1. *Correct* merupakan jumlah kalimat yang berhasil di ekstrak sistem sesuai dengan kalimat yang diekstrak manusia.
2. *Wrong* merupakan jumlah kalimat yang diekstrak sistem tetapi tidak terdapat dalam kalimat yang diekstrak manusia
3. *Missed* merupakan jumlah kalimat yang diekstrak manusia tetapi tidak terdapat dalam kalimat yang diekstrak sistem.

2.7.1 *Recall*

Recall (r) merupakan istilah yang digunakan untuk kalimat terpanggil yang relevan dengan pernyataan atau vektor *query*. Perbandingan jumlah informasi relevan yang didapatkan sistem dengan jumlah seluruh informasi relevan yang ada dalam koleksi informasi. *Recall* berhubungan dengan kemampuan dalam menemukan kalimat yang relevan. Hal ini berarti *recall* adalah bagian dari proses evaluasi yang dapat digunakan sebagai alat ukur relevan sebuah ringkasan. Adapun rumus untuk menghitung *recall* dapat dilihat pada persamaan (2.30) berikut:

$$r = \frac{correct}{correct+missed} \tag{2.30}$$

2.7.2 *Precision*

Precision (p) biasanya menjadi salah satu ukuran yang digunakan untuk menilai evektivitas kalimat yang di hasilkan dalam sebuah rangkuman. *Precision* adalah jumlah kalimat relevan dari total jumlah kalimat yang ditemukan oleh sistem. Perbandingan jumlah informasi relevan yang didapatkan sistem dengan jumlah seluruh informasi yang terambil oleh sistem baik yang relevan maupun tidak. Adapun, pengukuran tingkat ketepatan (*precision*) dirumuskan pada persamaan (2.31) berikut:

$$p = \frac{correct}{correct+wrong} \tag{2.31}$$

2.7.3 *F-Measure*

F-Measure (F_1) merupakan salah satu perhitungan evaluasi yang mengkombinasikan *recall* dan *precision*. Hubungan antara *recall* dan *precision* yang mempresentasikan akurasi sistem. Nilai *recall* dan *precision* pada suatu keadaan dapat memiliki bobot yang berbeda. Ukuran yang menampilkan timbal balik antara *recall* dan *precision* adalah *f-measure* yang merupakan bobot *harmonic mean* dari *recall* dan *precision*. Perhitungan *f-measure* dapat dirumuskan pada persamaan (2.32) berikut:

$$F_1 = \frac{2*precision*recall}{precission+recall} \tag{2.32}$$

2.8 Pemrograman Berorientasi Objek

Pemrograman berorientasi objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya[12]. Suatu cara bagaimana sistem perangkat lunak dibangun melalui pendekatan objek secara sistematis, serta didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas yang meliputi, rangkaian aktivitas analisis berorientasi objek, perancangan berorientasi objek, pemrograman berorientasi objek, dan pengujian berorientasi objek. Konsep dasar berorientasi objek diantaranya[12]:

1. Kelas (*Class*) adalah kumpulan objek-objek dengan karakteristik yang sama. Kelas merupakan definisi statik dan himpunan objek yang sama yang mungkin lahir atau diciptakan dari kelas tersebut. Sebuah kelas akan mempunyai sifat (atribut), kelakuan (operasi/metode), hubungan (*relationship*) dan arti. Suatu kelas dapat diturunkan dan kelas semula dapat diwariskan ke kelas yang baru.
2. Objek (*Object*) adalah abstraksi dan sesuatu yang mewakili dunia nyata seperti benda, manusia, satuan organisasi, tempat, kejadian, strutur, status, atau hal-hal lain yang bersifat abstrak. Objek merupakan suatu entitas yang mampu menyimpan informasi (status) dan mempunyai operasi (kelakuan) yang dapat diterapkan atau dapat berpengaruh pada status objeknya. Objek mempunyai siklus hidup yaitu diciptakan, dimanipulasi, dan dihancurkan.
3. Metode (*Method*) adalah operasi atau metode pada sebuah kelas hampir sama dengan fungsi atau prosedur pada terstruktur. Sebuah kelas boleh memiliki lebih dari satu metode atau operasi. Metode atau operasi yang berfungsi untuk memanipulasi objek itu sendiri.
4. Atribut (*Attribute*) dari sebuah kelas adalah variabel global yang dimiliki sebuah kelas. Atribut dapat berupa nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek. Atribut secara individual oleh sebuah objek, misalnya berat, jenis, nama, dan sebagainya.
5. Abstraksi (*Abstraction*) merupakan prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.

6. Enkapulasi (*Encapsulation*) adalah pembungkusan atribut data dan layanan (operasi-operasi) yang dipunyai objek untuk menyembunyikan implementasi dan objek sehingga objek lain tidak mengetahui cara kerja.
7. Pewarisan (*Inheritance*) adalah mekanisme yang memungkinkan satu objek mewarisi sebagian atau seluruh definisi dan objek lain sebagai bagian dari dirinya.
8. Antarmuka (*Interface*) sangat mirip dengan kelas, tetapi tanpa atribut kelas dan tanpa memiliki metode yang dideklarasikan. Antarmuka biasanya digunakan agar kelas lain tidak langsung mengakses ke suatu kelas.
9. Reusability merupakan pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan pada permasalahan lainnya yang melibatkan objek tersebut.
10. Generalisasi dan Spesialisasi menunjukkan hubungan antara kelas dan objek yang umum dengan kelas dan objek yang khusus. Misalnya kelas yang lebih umum (generalisasi) adalah kendaraan darat dan kelas khususnya (spesialisasi) adalah mobil dan motor.
11. Komunikasi antar objek dilakukan lewat pesan (*message*) yang dikirim dan satu objek ke objek lainnya.
12. Polimorfisme (*Polymorphism*) adalah kemampuan suatu objek untuk digunakan dibanyak tujuan yang berbeda dengan nama yang sama sehingga menghemat baris program.
13. *Package* adalah sebuah kontainer atau kemasan yang daoat digunakan untuk mengelompokkan kelas-kelas sehingga memungkinkan beberapa kelas yang bernama sama disimpan dalam *package* yang berbeda.

2.9 *Unifed Modeling Language*

UML singkatan dari *Unifed Modeling Language* yang berarti bahasa pemodelan standar. UML merupakan bahasa standar untuk merancang dan mendokumentasikan perangkat lunak dengan cara berorientasi objek. Ada beberapa diagram yang digunakan proses pembuatan perangkat lunak berorientasi objek diantaranya, *use case diagram, activity diagram, class diagram* dan *sequence diagram*[12].

2.9.1 *Use Case Diagram*

Use case diagram merupakan pemodelan untuk tingkah laku (*behavior*) pada sistem yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem yang akan dibuat. *Use case diagram* digunakan untuk mengetahui fungsi apa saja yang terdapat pada sistem. Terdapat dua hal utama yang diperlukan dalam pembentukan suatu *use case diagram* yaitu aktor dan *use case*.

1. Aktor merupakan orang, benda maupun sistem lain yang berinteraksi dengan sistem yang akan dibangun.
2. *Use Case* merupakan fungsionalitas atau layanan yang disediakan oleh sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor.

2.9.2 *Activity Diagram*

Activity diagram menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem, proses bisnis atau menu yang ada pada perangkat lunak. Setiap *use case* yang telah dibentuk digambarkan aktivitasnya dalam *activity diagram*, mulai dari peran aktor, peran sistem, dan *decision*. *Activity diagram* juga banyak digunakan untuk mendefinisikan hal-hal berikut:

1. Rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem.
2. Urutan atau pengelompokan tampilan dari sistem / *user interface* dimana setiap aktivitas dianggap memiliki sebuah rancangan tampilan antarmuka.
3. Rancangan pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.
4. Rancangan menu yang ditampilkan pada perangkat lunak.

2.9.3 *Class Diagram*

Classs diagram menggambarkan interaksi dan relasi antar kelas yang ada di dalam suatu sistem. Kelas memiliki atribut dan metode. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas. Metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas. Atribut dan metode dapat memiliki salah satu sifat sebagai berikut:

1. *Private,* tidak dapat dipanggil dari luar kelas yang bersangkutan.
2. *Protected,* hanya dapat dipanggil oleh kelas yang bersangkutan dan anak-anak yang mewarisinya.
3. *Public,* dapat dipanggil oleh siapa saja.

Class diagram menggambarkan relasi atau hubungan antar kelas dari sebuah sistem. Berikut ini beberapa gambaran relasi yang ada dalam *class diagram*:

1. *Association*

   Hubungan antar class yang statis. *Class* yang mempunyai relasi asosiasi menggunakan *class* lain sebagai atribut pada dirinya.
2. *Aggregation*

   Relasi yang membuat *class* yang saling terikat satu sama lain namun tidak terlalu berkegantungan.
3. *Composition*

   Relasi agregasi dengan mengikat satu sama lain dengan ikatan yang sangat kuat dan saling berkegantungan.
4. *Dependency*

   Hubungan antar *class* dimana *class* yang memiliki relasi *dependency* menggunakan *class* lain sebagai atribut pada *method*.
5. *Realization*

   Hubungan antar *class* dimana sebuah *class* memiliki keharusan untuk mengikuti aturan yang ditetapkan *class* lainnya.

2.9.4 *Sequence Diagram*

Sequence diagram menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan *sequence diagram* maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya *sequence diagram* yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada *sequence diagram* sehingga semakin banyak *use case* yang didefinisikan maka *sequence*

diagram yang harus dibuat juga semakin banyak. Penomoran pesan berdasarkan urutan iteraksi pesan. Penggambaran letak pesan harus berurutan, pesan yang lebih atas dari lainnya adalah pesan yang berjalan terlebih dahulu.

2.10 *Software* Pendukung

Pada penelitian yang dilakukan diperlukan *software* pendukung yang membantu dalam pembangunan aplikasi. Adapun *software* pendukung, diantaranya:

2.10.1 NetBeans IDE

NetBeans IDE (*Integrated Development Environment*), dapat juga berupa *Integrated Design Environment* dan *Integrated Debugging Environment*, yakni sebuah program atau alat bantu yang terdiri atas *Editor, Compiler, Debugger* dan *Design* yang terintegrasi dalam satu aplikasi. Netbeans IDE ditujukan untuk memudahkan dalam penggunaan bahasa pemrograman Java.

Untuk memakai Netbeans IDE maka, sebelumnya harus terlebih dahulu mempunyai *driver* JDK yang akan mendukung pembuatan perangkat lunak dengan menggunakan Netbeans IDE, sehingga sebelum meng-*install* Netbeans, terlebih dahulu sebelumnya harus meng-*install driver* JDK.

2.10.2 *Java Development Kit*

Java Development Kit atau disingkat JDK adalah *software development kit* merupakan alat bantu yang digunakan untuk membuat, memanajemen dan membangun berbagai aplikasi dalam bahasa pemograman Java. Pada JDK, terdapat berbagai *tools* yang digunakan untuk membangun aplikasi Java.

Bahasa pemrograman Java menyediakan *library-library* standar yang telah di-*compile* dan dapat langsung digunakan dalam implementasi pembuatan sebuah aplikasi. Pada *library*, terdapat berbagai macam *class* yang dapat digunakan dan telah dikelompokkan ke dalam *package.*

2.10.3 MySQL

MySQL merupakan sistem manajemen basis data SQL yang sangat terkenal dan bersifat *open source*. MySQL dibangun, didistribusikan, dan didukung oleh MySQL AB. MySQL AB merupakan perusahaan komersial yang dibiayai oleh pengembang MySQL.MySQL mempunyai dua macam lisensi yaitu lisensi yang

bersifat *open source* dengan menggunakan GNU *General Public License* dan lisensi kedua berupa *Standard Commercial License*.

Dalam konteks bahasa SQL, informasi disimpan dalam tabel-tabel yang secara logis merupakan struktur dua dimensi yang tersimpan atas baris-baris data (*row* atau *record*) yang berada dalam satu atau lebih kolom (*column*). Baris pada tabel disebut *insance* dari data sedangkan kolom sering disebut sebagai *atributes* atau *field*y.

2.10.4 XAMPP

XAMPP merupakan salah satu paket installasi Apache, PHP dan MySQL instant yang dapat digunakan untuk membantu proses installasi ketiga produk tersebut. Dengan menginstall XAMPP tidak perlu melakukan instalasi dan konfigurasi web server Apache, PHP dan MySQL secara manual.

XAMPP akan menginstalasi dan mengkofigurasikannya secara otomatis. XAMPP merupakan pengembangan dari LAMP (*Linux Apache*, *MySQL*, PHP and PERL), XAMPP ini merupakan *project non-profit* yang di kembangkan oleh Apache Friends yang didirikan Kai 'Oswalad' Seidler dan Kay Vogelgesang pada tahun 2002, *project* mereka ini bertujuan mempromosikan pengunaan Apache web server.

2.11 Pengujian Aplikasi

Pengujian bertujuan untuk mencari kesalahan. Pengujian yang baik adalah pengujian yang memiliki kemungkinan besar dalam menentukan kesalahan. Berikut adalah penjelasan mengenai pengujian *black box*[13].

2.11.1 *Black Box Testing*

Pengujian *black box* disebut juga dengan pengujian perilaku, berfokus pada persyaratan fungsional perangkat lunak. Pengujian *black box* berupaya untuk menemukan kesalahan dalam kategori seperti berikut:

1. Fungsi yang salah atau hilang.
2. Kesalahan antarmuka.
3. Kesalahan dalam struktur data atau akses basis data eksternal.
4. Kesalahan perilaku atau kinerja.
5. Kesalahan inisialisasi dan penghentian.