BAB II

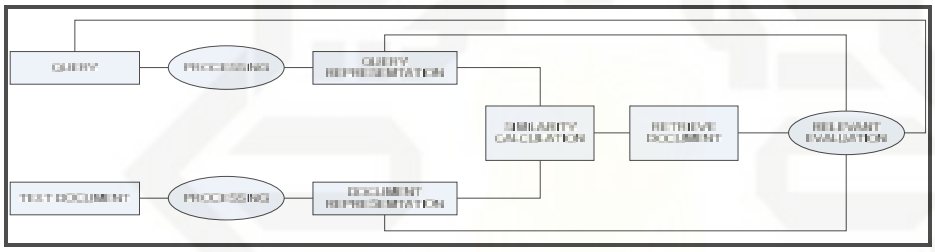
LANDASAN TEORI

2.1. Information retrieval

Information retrieval merupakan suatu sistem yang di gunakan untuk mengambil sebuah informasi dalam sebuah teks ataupun dalam sebuah dokumen (Ricardo,1999). *Information retrieval* di kenal mampu menemukan suatu informasi pada dokumen (teks) yang tidak tertstruktur dan memberikan sebuah kebutuhan informasi yang cukup (Christopher D.Manning, Prabhakar Raghavan, 2009).

2.1.1. Arsitektur *Information retrieval*

Dalam proses nya, arsitektur IR dapat di lihat pada gambar 2.1 berikut



Gambar 2.1 Arsitektur *Information retrieval* (Karyono, Utomo, 2012)

Ada dua tahapan dalam pemprosesan *information retrieval*, yang pertama adalah melakukan proses *pre-processing* pada *database* setelah itu IR menerapkan metode penghitungan yang bertujuan untuk menghitung relevansi antar dokumen yang ada di dalam *database* yang telah di lakukan *pre-processing* dengan *query* pengguna (Christopher D.Manning, Prabhakar Raghavan, 2009).

2.1.2. Koleksi Dokumen (*corpus*)

Koleksi dokumen (*corpus*) adalah suatu koleksi dokumen yang sudah diindeks dan dijadikan target dalam suatu target pencarian. *Corpus* memiliki beberapa karateristik sebagai berikut (McEnery & Wilson, 1996):

1. *Sampling and representativeness*
2. *Finite Size*
3. *Machine readable form*
4. *A standar reference*

2.1.3. Pembangunan *Index*

Tugas utama dalam tahapan *pre-processing* adalah pembangunan atau pembuatan index. Bisa dikatakana kualitas, efektifitas dan efisiensinya suatu *information retrieval* tergantug dari bagaimana proses pembuatan *index*. Isi dari *index* sendiri merupakan himpunan *term* yang berisi topik dari sebuah dokumen (Ricardo Baeza-Yates, 1999).

Index merupakan suatu pembeda setiap dokumen yang berada di dalam satu koleksi. Hal yang harus menjadi perhatian adalah ukuran *index* mempengaruhi hasil dari proses *information retrieval* itu sendiri. semakin kecil *index* yang ada maka akan semakin buruk hasil dan kemungkinan menemukan relevan data semakin kecil, sebaliknya jika *index* semakin besar maka hasil akan semakin baik serta kemungkinan untuk menemukan relevan dokumen atau data semakin besar (Christopher D.Manning, Prabhakar Raghavan, 2009).

Berikut adalah tahapan pengerjaan *Information retrieval* (Christopher D.Manning, Prabhakar Raghavan, 2009) :

1. Mengumpulkan *corpus* yang akan di lakukan proses pembuatan index.
2. Penghapusan format dan markup dari dalam dokumen.

   Sebelum melakukan *index* perlu dilakukan tahap untuk menghapus setiap format dan markup dari dalam dokumen. Pada tahap ini semua *tag markup* dan format khusus dihapus dari dalam dokumen, terutama pada dokumen yang mempunyai banyak tag *markup* dan format seperti dokumen (X)HTML.
3. *Tokenization*

   Tokenization merupakan tahapan yang penting karena daam *tokenization* di lakukan proses penghilangan seluruh tanda baca, karakter asing dan huruf besar di dalam dokumen. Semua dokumen akan menjadi *lowercase* atau *temmed word* dan dipisahkan menjadi token.
4. Linguistic *pre-processing*

   Linguistic preprocessing dilakukan untuk menghasilkan daftar kata (token atau *term*) yang ternormalisasi. Ada dua hal yang harus di perhatikan dalam *linguistic pre-processing* yaitu :

   a. Penyaringan (*filtration*)

Pada tahapan ini ditentukan *term* mana yang akan digunakan untuk merepresentasikan dokumen sehingga dapat mendeskripsikan isi dokumen dan membedakan dokumen tersebut dari dokumen lain di dalam koleksi. *Term* yang sering digunakan dianggap sebagai stop-word dan dihapus. Penghapusan *stop-word* dari dalam suatu koleksi dokumen pada satu waktu membutuhkan banyak waktu. Solusinya adalah dengan menyusun suatu pustaka *stop-word* atau *stop-list* dari *term* yang akan dihapus(Ricardo Baeza-Yates, 1999).

b. Konversi *term* ke bentuk akar (*stemming*).

Stemming adalah salah satu cara yang digunakan untuk meningkatkan performa sistem temu balik informasi dengan cara mentransformasi kata-kata dalam sebuah dokumen teks ke bentuk kata dasarnya, contohnya katakata menyukseskan, tersukseskan dan disukseskan akan ditransformasi ke tem yang sama yaitu sukses. 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.Tidak banyak algoritma yang dikhususkan untuk *stemming* bahasa Indonesia dengan berbagai keterbatasan didalamnya, diantaranya :

a) Algoritma Porter, Algoritma ini membutuhkan waktu yang lebih singkat dibandingkan dengan *stemming* menggunakan Algoritma Nazief & Adriani, namun proses *stemming* menggunakan Algoritma Porter memiliki presentase keakuratan (presisi) lebih kecil dibandingkan dengan *stemming* menggunakan Algoritma Nazief & Adriani.

b) Algoritma Nazief Algoritma Nazief & Adriani, algoritma *stemming* untuk teks berbahasa Indonesia yang memiliki kemampuan presentase keakuratan (presisi) lebih baik dari algoritma lainnya. Algoritma ini sangat dibutuhkan dan menentukan dalam proses sistem temu balik informasi dalam dokumen Indonesia. Algoritma Nazief & Adriani mengacu pada aturan morfologi bahasa

Indonesia yang mengelompokkan imbuhan, yaitu imbuhan yang diperbolehkan atau imbuhan yang tidak diperbolehkan. Pengelompokan ini *term*asuk imbuhan di depan (awalan), imbuhan kata di belakang (akhiran), imbuhan kata di tengah (sisipan) dan kombinasi imbuhan pada awal dan akhir kata (konfiks).

Langkah-langkah pada Algoritma Nazief & Adriani adalah:

1. Kata yang belum di-*stemming* dicari pada kamus. Jika kata itu langsung ditemukan, berarti kata tersebut adalah kata dasar. Kata tersebut dikembalikan dan algoritma dihentikan.
2. Hilangkan *inflectional suffixes* terlebih dahulu. Jika hal ini berhasil dan *suffix* adalah partikel (“lah” atau ”kah”), langkah ini dilakukan lagi untuk menghilangkan *inflectional possessive pronoun suffixes* (“ku”, “mu” atau ”nya”).
3. *Derivational suffix* kemudian dihilangkan. Lalu langkah ini dilanjutkan lagi untuk mengecek apakah masih ada *derivational suffix* yang tersisa, jika ada maka dihilangkan. Jika tidak ada lagi maka lakukan langkah selanjutnya.
4. Kemudian *derivational prefix* dihilangkan. Lalu langkah ini dilanjutkan lagi untuk mengecek apakah masih ada *derivational prefix* yang tersisa, jika ada maka dihilangkan. Jika tidak ada lagi maka lakukan langkah selanjutnya.
5. Setelah tidak ada lagi imbuhan yang tersisa, maka algoritma ini dihentikan kemudian kata dasar tersebut dicari pada kamus, jika kata dasar tersebut ketemu berarti algoritma ini berhasil tapi jika kata dasar tersebut tidak ketemu pada kamus, maka dilakukan *recoding*.
6. Jika semua langkah telah dilakukan tetapi kata dasar tersebut tidak ditemukan pada kamus juga maka algoritma ini mengembalikan kata yang asli sebelum dilakukan *stemming*.

5. Mengindeks dokumen (*indexing*)

Sebuah indeks selalu memetakan kembali dari setiap *term* ke dokumen dimana *term* tersebut muncul. Pengindeksan dilakukan dengan membuat

inverted index (atau disebut juga inverted file) yang terdiri dari *dictionary* dan *postings*. *Inverted index* merupakan konversi dari dokumen asli yang mengandung sekumpulan kata ke dalam daftar kata (*dictionary*) yang behubungan dengan dokumen terkait dimana kata-kata tersebut muncul. Dictionary adalah daftar kata yang diperoleh dari hasil pengindeksan dokumen.

2.1.4. Pembobotan Kata(*Term Weighting*)

Pembobotan terbagi tiga yaitu pembobotan lokal, pembobotan global dan pembobotan kombinasi yang memadukan antara pembobotan lokal dan kombinasi. Pada tahap pembobotan kata setiap *term* di beri bobot dengan skema tertentu. *Term* yang akan di index harus *term* yang sudah di-*index*. Dalam pembobotan terdapat penggunaan kondisi yang di lakukan sesuatu dengan jenis pembobotan nya, jika di lakukan pembobotan lokal maka digunakan *tf (term condition),* jika dilakukan pembobotan global maka digunakan *idf(invers document frequency)* dan jika di lakukan kombinasi maka akan di gunakan *tf . idf .* berikut penjelasan dari setiap ekspresi pembobotan kata (lake, 2015)*:*

1. *Term Frequency(tf)*

   Dalam proses *tf* ada empat cara yang bisa di lakukan untuk penyelesaiannya, yaitu :

a. *Raw term frequency*.

   Term yang di peroleh dari *tf* diatur berdasarkan jumlah kemunculan dari *term* yang berada dalam dokumen itu sendiri. misalnya, *term* muncul 6 kali dalam suatu dokumen, maka 6 merupakan nilai dari *tf term nya.*

b. *Logarithm term frequency*.

   Cara ini digunakan agar menghindari dokumen yang mengandung sedikit *term* di dalam *query* ,tetapi frekuensi nya tinggi. Untuk cara ini di terapakan rumus untuk memperoleh nilai *tf* sebagai berikut :

tf=1log(tf).....(2.1)

c. *Binary term frequency*.

   Hanya memperhatikan apakah suatu *term* ada atau tidak dalam dokumen. Jika ada, maka *tf* diberi nilai 1, jika tidak ada diberi nilai 0. Pada cara ini jumlah kemunculan *term* dalam dokumen tidak berpengaruh.

d. *Augmented term frequency*.

Untuk cara ini di gunakan fungsi logatrimatik matematika sebagai berikut

tf=0,5 + 0,5 x tf /*max(tf)*.....(2.2)

dari rumus dia atas bisa di lihat bahwa nilai *tf* adalah jumlah kemunculan suatu *term* pada sebuah dokumen, nilai *max(tf)* adalah jumlah kemunculan terbanyak sebuah *term* pada dokumen yang sama.

2. *Invers Document Frequency*

Invers document frequency(idf) muncul sebgaia tekanan *term* dominan yang sering muncul dalam setiap dokumen. *idf* berfungsiu menghindari *term* seperti *common term* lebih banyak muncul sehingga nilai yang mucul hanyalah nilai yang tidak penting saja. *Idf* aka memperhtikan factor kebalikan *frequency* dokumen yang mengandung suatu *term*. Seperti pada rumus *idf* berikut :

$$idf(i) = \log(\frac{n}{df(t)}) \quad \text{.....(2.3)}$$

Keterangan :

N : Jumlah dokumen dalam *corpus*

Dft *document frequeny* dalam *corpus* yang mengandung *term* t.

2.1.5. Metode *Information Retrival*

Information retrieval memenuhi informasi dengan me-*retrieve relevant-irelevant* dokumen. Dalam *Information retrieval* terdapat beberapa metode yang bisa membantu dalam proses pengerjaan-nya seperti pada tabel 2.1 berikut (Christopher, 2009).

Tabel 2.1 Metode dalam *Information retrieval*

Classifier	Metode
Naïve Bayes	Menghitung probabilitas dari suatu dokumen untuk ikut ke suatu kategori berdasarkan pada kehadiran dari kata yang sama di dalam dokumen lain yang telah ada di dalam kategori tersebut
Rocchio	Membandingkan dokumen terhadap suatu daftar *term* positif dan negatif bagi setiap katagori dan mengklasifi sesuai dengan kehadiran atau bobot dari *term-term* tersebut.
K-Nearest Neighbor	Mencari sebanyak k dokumen paling mirip dan menempatkan dokumen ke kategori di mana k dokumen tersebut ditempatkan sebelumnya
Decision Tree	Memisahkan dokumen-dokumen secara hirarki di dalam struktur pohon, di mana setiap node merupakan *term* yang relevan dan ujung setiap cabang adalah kategori.
Support Vector Machine	Menggambar antara *term* yang berkontribusi dan tidak terhadap suatu dokumen yang akan ditempatkan ke suatu kategori tertentu. Kategori didasarkan pada kehadiran dari *term* yang berkontribusi.

2.2. Rocchio Relevance feedback

Algoritma *Rocchio* merupakan yang mampu membandingkan setiap dokumen terhapad suatu *term* tertentu dalam klasifikasi kategori sesuai degan bobot dari *term* yang ada (Badriz,2008). Pada dasarnya *Rocchio* menggunakan Vector Spce Model, pada *Rocchio* proses klasifikasi di lakukan berdasarkan asumsi manusia (Mandowara, 2016). *Rocchio* memiliki 2 hal dalam pengerjaan yang harus di perhatikan yaitu sebagai berikut (Badriz,2008):

1. *Term* frequency and weighting
   a. Berdasarkan frekuensi munculnya *term* yang sesuai dengan *query*
   b. *Term* yang sama akan dijumlah semua frekuensinya
   c. Akan tetapi metode ingin masih kurang bagus karena :
      jika dokumen yang digunakan adalah dokumen yang lebih besar, maka dokumen tersebut memiliki *term* yang lebih banyak sehingga *score*-nya pun lebih besar.
   d. Langkah-langkah dalam menggunakan metode ini adalah :
      1. Tiap-tiap dokumen dipecah menjadi *term-term*
      2. Kemudian *term* yang sudah ada diurutkan menjadi sebuah kamus di dalam sebuah kolom (catatan : jika ada beberapa *term* yang sama, maka hanya ditulis sekali)
      3. Di sebelah kanan kolom *term*, tambahkan 2 kolom lagi. Kolom yang pertama untuk frekuensi *term* (tf). Hitung jumlah *term* sama.
      4. Untuk kolom yang kedua untuk kolom idft.
      Rumus :

Idft=$logNdft$.....(2.4)

N : jumlah dokumen yang ada dalam *corpus*

dft : frekuensi dari sebuah *term*

5. Tambahkan kolom lagi untuk bobot untuk menentukan dokumen yang paling sering muncul atau tidak dalam seuah *term*

Rumus :

β=(tf)*(idft).....(2.5)

β = nilai bobot *term*

tf = nili *term* frekuensi

idf = nilai invers dokumen frekuensi

6. Untuk menentukan tingkat kemiripan dari *Rocchio* ini di gunakan rumus

$$R = N + \beta(\left(\frac{Dp}{Np}\right) - (\frac{Dn}{Nn})) \quad \text{.....(2.6)}$$

Keterangan :

R = Tingkat kemiripan *term*

N = jumlah *term* dokumen

B = nilai bobot *term*

Dp = *term* dari dokumen relevan

Np = jumlah dokumen relevan

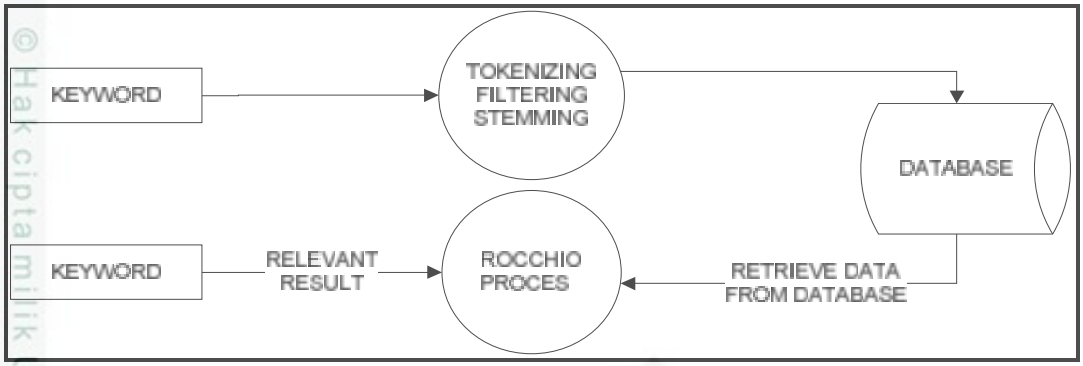
Dn = *term* dari dokumen tidak relevan

Nn = jumlah dokumen tidak relevan

2. The *Rocchio* algorithm for relevance feedback

Rocchio classifiers merupakan salah satu metode pembelajaran *supervised document classification*. Metode klasifikasi *Rocchio* membandingkan kesamaan isi antara data *training* dan data *test* dengan merepresentsaikan semua data ke dalam sebuah *vector*. Kedekatan kesamaan isi dihitung dari kedekatan sudut yang terbentuk antara bobot data training dan bobot data test menggunakan aturan cosinus. Untuk menghitung bobot setiap kata dalam dokumen digunakan skema pembobotan tfidf (*Term Frequency / Invers Document Frequency*) karena komponen heuristic utama adalah klasifikasi *Rocchio* yaitu skema pembobtan tfidf, metode pembelajaran *Rocchio* disebut juga dengan tfidf *Classifiers* (Badriz Zamam, S, Si, 2008).

Dalam proses nya algoritma *Rocchio* bisa kita gambarkan dalam bagan sebagai berikut :



Gambar 2.2 Struktur *Rocchio* (Kristanda, & Hansun, 2016)

2.3. *Unified Modeling Language* (UML)

UML (*Unified Modeling Language*) adalah salah satu standar bahasa yang banyak digunakan di dunia industry untuk mendefenisikan *requirement,* membuat analisis dan desain, serta menggambarkan arsitektur dalam pemrograman berorientasi objek. Pemodelan merupakan gambaran realita yang *simple* dan dituangkan dalam bentuk pemetaan dengan aturan tertentu. Pemodelan dapat menggunakan bentuk yang sama dengan realitas (Rossa A.S, 2013). Pada UML ada beberapa diagram dalam melakukan analisa suatu permasalahan:

2.3.1. *Use Case Diagram*

Use Case Diagram merupakan pemodelan untuk kelakuan (*behavior)* sistem informasi yang akan dibuat. *Use Case* mendeskripsikan sebuah interaksi antara satu atau lebih actor dengan sistem informasi yang akan dibuat. Secara kasar, *Use Case* digunakan untuk mengetahui fungsi apa saja yang ada dalam sebuah sistem informasi dan siapa saja berhak menggunakan fungsi (Rossa A.S, 2013).

Syarat penamaan pada *Use case* adalah nama yang didefenisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu:

1. Aktor merupakan orang, proses atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat diluar sistem informasi yang akan dibuat itu sendiri. Simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.
2. *Use Case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antara unit atau aktor.

Simbol-simbol *Use case Diagram* dapat dilihat pada tabel 2.2 berikut:

Tabel 2.2 Simbol-simbol *Use Case Diagram*

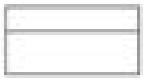
No	Simbol	Keterangan fungsi
1	Aktor 	Aktor adalah seseorang atau sesuatu yang berinteraksi dengan sistem
2	*Use case* 	*Use Case* adalah deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu aktor.
3	Asosiasi	Asosiasi adalah yang menghubungkan antara objek satu dengan objek yang lain.
4	Generalisasi 	Generalisasi adalah hubungan dimana objek anak berbagi prilaku dan struktur data dari objek yang ada diatas atau sebalik dari bawah ke atas.

2.3.2. *Class Diagram*

Class Diagram merupakan struktur sistem dari segi pendefenisikan kelas-kelas apa yang dibuat untuk membangun sistem. Kelas memiliki atribut dan metode. Atribut merupakan variabel-variabel yang memiliki suatu kelas, sementara metode merupakan fungsi-fungsi yang dimiliki oleh suatu kelas (Muhammad Fikry, Yusra, 2015).

Mendefenisikan metode yang ada di dalam sebuah kelas perlu memperhatikan apa yang disebut dengan *Cohesion* dan *coupling*. *Cohesion* merupakan ukuran seberapa dekat keterkaitan instruksi antara metode terkait satu sama lain. *Coupling* adalah ukuran seberapa dekat keterkaitan instruksi antara metode yang satu dengan metode yang lain dalam sebuah kelas (Rossa A.S, 2013). Simbol-simbol *Class Diagram* dapat dilihat pada tabel 2.3 berikut:

Tabel 2. 3 Simbol-simbol *Class Diagram*

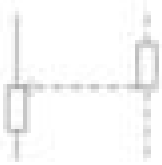
No	Simbol	Keterangan fungsi
1	*Class* 	*Class* memiliki atribut yang menggambarkan karakteristik objek.
2	Asosiasi	Asosiasi adalah yang menghubungkan antara objek satu dengan objek yang lain.
3	*Ternary Association*	*Ternary association* berfungsi ketika akan mengaitkan lebih dari 2 *class*.

2.3.3. *Sequence Diagram*

Sequence Diagram menggambarkan interaksi antara sejumlah objek dalam urutan waktu. Kegunaanya untuk menunjukan rangkaian pesan yang dikirim antara objek yang terjadi pada titik tertentu dalam eksekusi sistem. Dalam UML, objek, pada *Sequence Diagram* digambarkan dengan segi empat, yang berisi nama dari objek yang digaris bawah (Tohari, 2014). Simbol-simbol *Sequence Diagram* dapat dilihat pada tabel 2.4 berikut:

Tabel 2.4 Simbol-simbol *Sequence Diagram*

No	Simbol	Keterangan fungsi
1	Aktor 	Aktor adalah seseorang atau sesuatu yang berinteraksi dengan sistem
2	*Lifeline* 	Elemen permodelan antarmuka yang saling berinteraksi.
3	*Message* 	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktifitas yang terjadi

4		Mengambarkan pesan/hubungan obyek itu sendiri, yang menunjukkan urutan kejadian yang terjadi
5		Menggambarkan pesan/hubungan antar obyek, yang menunjukan urutan kejadian yang terjadi.

2.3.4. *Activity Diagram*

Activity Diagram menggambarkan *workflow* aliran kerja atau aktivitas dari sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak. Yang perlu diperhatikan disini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem (Rossa A.S, 2013).

Diagram aktivitas juga banyak digunakan untuk mendefenisikan hal-hal berikut:

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

Simbol-simbol *Activity Diagram* dapat dilihat pada tabel 2.5 berikut:

Tabel 2. 5 Simbol-simbol Activity Diagram

No	Simbol	Keterangan
1	*Initial Node* 	*Initial node* adalah sebuah kondisi awal *object* sebelum ada perubahan keadaan.
2	*Activity Final* 	*Activity final* menggambarkan ketika objek berhenti memberi respon terhadap sebuah *event*.

3	*Action*	*Action* menggambarkan kondisi sebuah entitas.
4	Generalisasi 	Generalisasi adalah hubungan dimana objek anak berbagi prilaku dan struktur data dari objek yang ada diatas atau sebalik dari bawah ke atas.
6	*Decision*	*Decision* merupakan suatu logika aliran konkuren yang mempunyai dua cabang aliran konkuren.

Dalam membangun sistem pada UML ada beberapa tahapan yang harus dilakukan diantaranya:

2.3.5. *Requirements*

Requirement merupakan tahap dilakukan identifikasi masalah pada sistemdx yang akan dibangun. Pada proses ini akan dikumpulkan data-data yang berkaitan dengan pembuatan sistem. UML menggunakan *Use cases* untuk menangkap kebutuhan customer/user. Melalui Use cases aktor luar yang berinteraksi dengan sistem dimodelkan bersama dengan fungsi-fungsi yang mereka perlukan dari sistem (*use cases*). Aktor dan use cases dihubungkan dengan suatu relasi (*relationship*). Actor dan use case ditampilkan dalam bentuk diagram beserta dokumentasinya pada *view* diagram *Use case.*

2.3.6. Analisis

Fase analisis konsen dengan abstraksi primer (kelas dan objek) dan mekanisme yang muncul dalam problem domain. Kelas-kelas diidentifikasi bersama dengan relasinya satu sama lain, dan ditampilkan pada diagram kelas. Kolaborasi antar kelas untuk mengerjakan use case juga dijelaskan melalui model dinamik UML. Pada fase analisis ini hanya kelas-kelas dalam problem domain yang dimodelkan, bukan kelas-kelas implementasi teknik.

2.3.7. Desain

Pada tahap desain hasil analisis didetailkan untuk solusi teknik. Kelas-kelas baru ditambahkan untuk menyediakan infrastruktur teknik: *user interface,*

penanganan *database* untuk menyimpan objek kedalam *database*, komunikasi dengan sistem lain, interfacing dengan peralatan dalam sistem ditambahkan.

2.3.8. Implemantasi/Programing

Pada tahap programming kelas-kelas yang dibentuk pada tahap desain dikonversi menjadi *code* sesungguhnya dalam bahasa pemrograman objek-oriented melalui proses generate. Hasil generate berupa skeleton dari program. Selanjutnya menjadi tugas programmer untuk menyelesaikan program hasil generate. Editing yang dilakukan oleh programmer tidak akan di dihapus (ditimpa) saat model di generate ulang.

2.3.9. *Testing*

Testing terhadap sistem software biasanya berupa tes unit, tes integrasi, tes sistem, dan tes *acceptance*. Tes unit adalah tes terhadap kelas individual atau terhadap sekelompok kelas, biasanya dilakukan oleh programmer. Tes integrasi mengintegrasikan komponen dan kelas-kelas dalam rangka verifikasi. Tes sistem memandang sistem sebagai kotak hitam (*black box*) dalam rangka validasi bahwa sistem berfungsi sesuai dengan harapan *end user*. Tes *acceptance* dilakukan oleh customer untuk verifikasi bahwa sistem sesuai dengan kebutuhan, sama seperti tes sistem. Test unit menggunakan diagram kelas dan spesifikasi kelas, test integrasi menggunakan diagram komponen dan diagram kolaborasi, dan tes sistem menggunakan diagram use-case untuk melakukan validasi.

2.4. Penelitian Terkait

Dibawah ini merupakan daftar penelitian yang terkait dengan implementasi sistem monitoring isu publik yang menggunakan beberapa metode, diantaranya:

Tabel 2. 6 Penelitian Terkait

No	Peneliti dan Tahun	Topik	Hasil
1	(Kristanda & Hansun, 2016)	Studi Kelayakan Dan Perancangan Aplikasi Pencarian Buku Pada Katalog Perpustakaan	Hasil studi kelayakan mengenai pencarian buku di perpustakaan UMN menunjukkan bahwa aplikasi pencarian buku yang diimplementasikan saat ini cukup mudah digunakan, tetapi proses

		Menggunakan *Rocchio* Relevance Feedback	pencarian belum menghasilkan keluaran yang relevan dengan kata kunci masukan. Dari hasil studi tersebut, dirancang aplikasi *mobile* pencarian buku yang dipadukan dengan metode *Rocchio* sehingga relevansi pencarian menjadi faktor penting dalam aplikasi. Perancangan ini diharapkan dapat mempermudah user dalam menemukan dokumen relevan sesuai dengan keyword yang sudah dimasukkan. Tahap selanjutnya aplikasi pencarian katalog perpustakaan dibangun sesuai dengan perancangan tersebut
2.	(Yulianto Et Al., N.D.)	Kajian Literatur Perkembangan Multimedia *Information retrieval* (Mir) Dan Tantangan Di Masa Depan	Berdasarkan hasil tinjauan literatur, berikut kesimpulan yang menjadi tantangan yang penting bagi komunitas peneliti MIR di masa depan: 1. Pencarian semantik yang menekankan pada pendeteksian objek pada media dengan latar yang kompleks 2. Analisis dan algoritma retrieval pada multi-komponen terhadap berbagai media, *term*asuk teks dan konteks informasi 3. Metode pencarian yang lebih interaktif, penggunaan semantik, dan sistem umpan balik relevansi yang lebih baik

			4. Evaluasi algoritma IR melalui penggunaan data set berbagai pola
3.	(Khuluqiyah, Pudjianto, & Waha, 2016)	Klasifikasi Data Pengaduan Masyarakat Pada Laman Pesduk Cimahi Menggunakan *Rocchio*	Dari hasil kajian ini, maka dapat diambil kesimpulan bahwa kasus penentuan kategori/klasifikasi pesan pengaduan dengan text mining menggunakan *Rocchio* dapat dilakkan secara otomatis tanpa harus dibaca satu persatu terlebih dahulu. Sehingga diharapkan dapat membantu pihak operator dalam pengkategorian pesan pengaduan.
4.	(Karyono Et Al., 2012)	Temu Balik Informasi Pada Dokumen Teks Berbahasa Indonesia Dengan Metode Vector Space Retrieval Model	Pencarian informasi berdasarkan *query* oleh pengguna, yang diharapkan dapat menemukan koleksi dokumen berdasarkan kebutuhan pengguna, dikenal dengan *Information retrieval* atau temu balik informasi. Penelitian ini membahas tentang implementasi sistem temu balik informasi untuk mencari dan menemukan dokumen teks berbahasa indonesia menggunakan Vector Space Retrieval Model. Tujuan penelitian ini untuk menyediakan solusi pada mesin pencarian agar mampu menyediakan informasi dokumen teks pada *database* yang tepat menggunakan kata kunci tertentu. Hasil dari pencarian direpresentasikan dengan urutan/ranking kemiripan dokumen dengan *query*.
5.	(Mahmudi,	Klasifikasi Artikel Berita Secara	pemodelan dan pembuatan sistem pengklasifikasian artikel otomatis dengan

	2015)	Otomatis Menggunakan Metode Naive Bayes Classifier Yang Dimodifikasi	metode naive bayes yang telah dimodifikasi. Modifikasi dilakukan dengan melakukan pembobotan berasarkan posisi kata dalam berita. Akurasi sistem meningkat dengan meningkatnya jumlah data latih yang digunakan sebagai pembelajaran. Pembobotan posisi kata meningkatkan akurasi klasifikasi rata-rata sebesar 2,3%.