BAB 2
LANDASAN TEORI

2.1. Maximum Marginal Relevance

Algoritma *mazimum marginal relevance (MMR)* merupakan salah satu metode ekstraksi ringkasan *(extractive summary)* yang digunakan untuk meringkas dokumen tunggal atau multi dokumen [3]. *MMR* meringkas dokumen dengan menghitung kesamaan *(similiarity)* antara bagian teks.

Pada peringkasan dokumen dengan metode *MMR* dilakuakn proses segmentasi dokumen menjadi kalimat dan dilakukan pengelompokan sesuai dengan *gendeer* kalimat tersebut. *MMR* digunakan dengan mengkombinasikan matrik cosine similiaritu untuk merangking kalimat-kalimat sebagai tanggapan pada *query* yang diberikan oleh *user.*

2.2. Text Processing

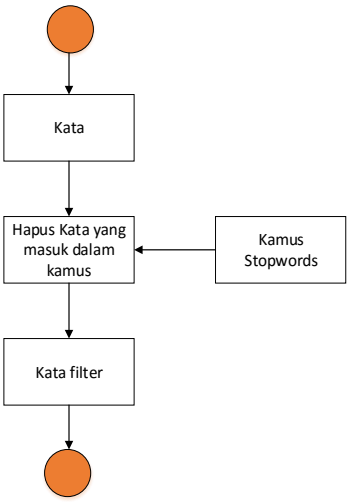
Dalam melakukan pemrosesan teks terdapat beberapa tahap, yakni masing-masing dapat dijelaskan sebagai berikut:

2.2.1. Case Folding

Case folding merupakan proses penyamaan case dalam sebuah dokumen teks [3]. *Case folding* dilakukan karena total semua dokumen teks konsisten dalam menggunakan huruf kapital. Untuk itu peran *case-folding* dibutuhkan dalam merubah kseluruhan teks dalam dokumen menjadi bentuk standar yang secara umum diubah kedalam bentuk huruf kecil.

2.2.2. Tokenizing

Tokenizing merupakan proses pemotongan string masukan berdasarkan tiap kata yang menyusunya [3]. Prinsipnya, dilakukan pemisahan setiap kata yang menyusun dokumen teks tersebut. Setiap kata teridentifikasi atau terpisahkan dengan kata yang lain oleh karakter spasi, maka proses *tekenizing* dilakukan untuk memisahkan kata.



Gambar 2.1 ***Activity Diagram Filtering***

2.2.3. Filtering dan Eliminasi Stopwords

Eliminasi *Stopwords* dilakukan dengan mengambil kata-kata penting dari hasil token. Dalam proses ini dapat dilakukan pembuangan kata yang kurang penting *(stop list)* atau penyimpanan kata yang dianggap penting *(word list).* Penggunaan eliminasi stopwords berfungsi untuk menghilangkan kata-kata yang dianggap tidak penting, secara umum kandidat stopwords seperti article, preposisi, dan konjungsi. Beberapa kata kerja, kata sifat dan kata keterangan lainnya dapat juga dimasukan kedalam daftar stopwords.

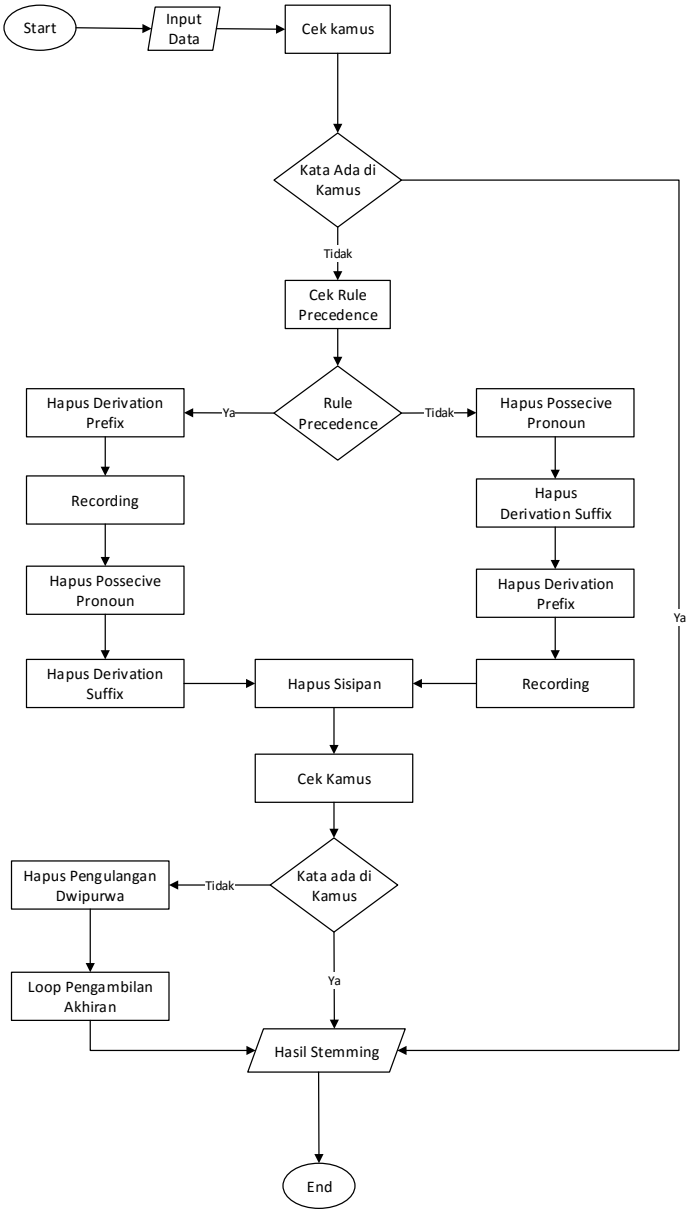
2.2.4. Stemming

Stemming merupakan cara yang digunakan untuk mentransformasikan kata-kata dalam sebuah dookumen teks menjadi kata dasarnya. Pada proses ini dilakukan dengan menghilangkan semua imbuhan (afiks) baik yang terdiri dari awalan (prefiks) sisipan (infiks) maupun akhiran (sufiks) dan kombinasi dari awalan dan akhiran (konfiks). *Stemming* ini digunakan untuk mengganti bentuk dari suatu kata menjadi kata dasar sesuai dengan struktur morfologi bahasa yang baik dan benar. Proses *stemming* pada teks bahasa indonesia lebih rumit dan kompleks karena terdapat variasi imbuhan yang harus dibuang untuk mendapatkan *root word* (kata

dasar) dari sebuah kata. Pada umumnya kata dasar pada bahasa indonesia terdiri dari kombinasi [8] :

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

Ada beberapa algoritma dalam melakukan proses *stemming*. Algoritma ini tergantung dengan bahasa yang digunakan, khususnya bahasa Indonesia. Dalam bahasa indeonesia pun terdapat beberapa algoritma yang dikembangkan, dalam hal ini adalah algoritma *Enhanced Confix Stripping (ECS)*. Algoritma ECS ini memiliki tingkat kelasalahn *stemming* paling sedikit karena merupakan pengembangan dari algoritma Confix Stemmer [9]. Perbaikan yang dilakukan oleh ECS Stemmer adalah perbaikan beberapa aturan pada tabel acuan pemenggalan imbuhan. Selain itu, algoritma ECS Stemmer juga menambahkan langkah pengembalian akhiran jika terjadi penghilangan akhiran yang seharusnya tidak dilakukan.

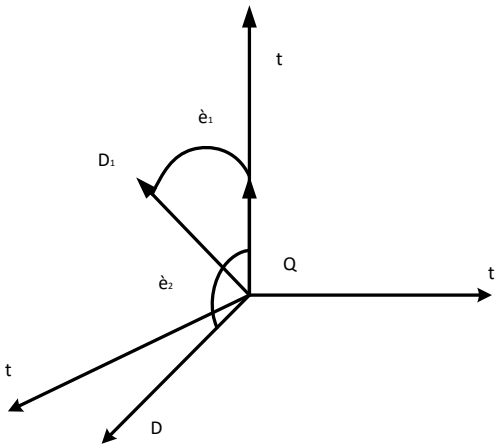


Gambar 2.2 Algoritma ECS [9]

2.3. Cosine Similiarity

Cosine similiarity digunakan untuk menghitung pendekatan relevansi *query* terhadap dokumen [10]. Penentuan relevansi sebuah *query* terhadap suatu dokumen dipandang sebagai pengukuran kesamaan vektor *query* dengan vektor dokumen. Semakin besar nilai kesamaan vektor *query* dengan vektor dokumen maka *query* tersebut dipandang semakin relevan dengan dokumen. Saat mesin menerima *query*, mesin akan membangun sebuah vektor Q (W_{q1}, W_{q2}, W_{qt}) berdasarkan istilah-istilah

pada *query* dan sebuah vektor D (D_{i1},D_{i2}, D_{it}) berukuran t untuk setiap dokumen. Pada umumnya *cosine similiarity* dihitung dengan rumus *cosine measure* (Grossman, 1998). Berikut ini adalah gambaran bagaimana *query* dan dokumen dibentuk menjadi model vektor.



Gambar 2.1 Vektor Skalar [3]

Penggunaan *cosine similiarity* adalah tindak lanjut dari proses pembobotan TF-IDF. Dari hasil pembobotan maka dicarilah kemiripan daintara dokumen yang ada. Namun penerapannya dalam *MMR* adalah kemiripan kalimat-kalimat dalam sebuah dokumen tekstual. Persamaannya dapat digambarkan sebagai berikut:

$$COS(\Theta_{ij}) = \frac{\sum_k d_{ik} d_{jk}}{\sqrt{\sum_k \mathrm{d}_{ik}^2}\ \sqrt{\sum_k \mathrm{d}_{jk}^2}} \qquad \textbf{2.1}$$

Database dari semua dokumen direpresntasikan oleh matriks *term-document* (atau matriks *term-frequency).* Dimana setiap sel pada matriks berkorespondensi dengan bobot yang diberikan dari suatu *term* dalam dokumen yang ditentukan. Nilai nol berarti bahwa term tidak terdapat dalam dokumen.

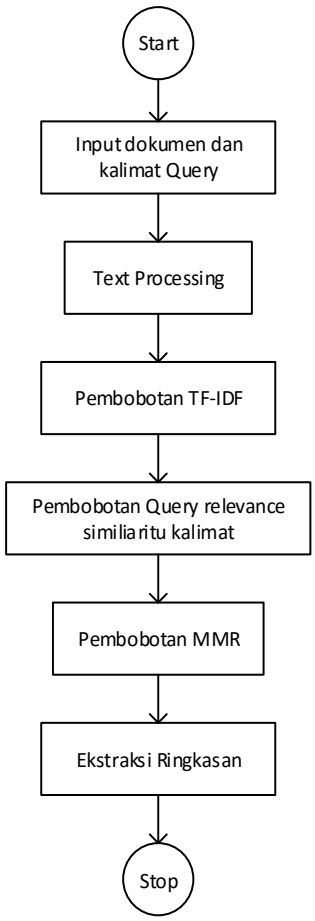
$$\begin{bmatrix} & T_1 & T_2 & \dots & T_t \\ D_1 & w_{11} & T_{21} & \dots & T_{r1} \\ D_2 & T_{12} & T_{22} & \dots & T_{r2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ D_n & W_{1n} & W_{2n} & \dots & W_{rn} \end{bmatrix} \qquad \textbf{2.2}$$

2.4. Pembobotan *MMR*

MMR (maximum marginal relevance) adalah metode peringkasan yang menggunakan dokumen tunggal ataupun multi-dokumen [3]. Teknik ini menerapkan penghitungan kesamaan *(similiarity)* antara segmentasi teks. Segmentasi ini dilakukan terhadap kalimat-kalimat dan dilakukan pengelompokan sesuai jenis kalimat tersebut. *MMR* digunakan dengan mengkombinasikan matrik *cosine similiarity* untuk merangking kalimat-kalimat sebagai tanggapan pada *query* yang diberikan oleh *user*.

Pembobotan *maximum marginal relevance* pada kalimat menggunakan algoritma *MMR*. Kelimat dirangking sebagai tanggapan terhadap *query* yang telah dimasukan oleh *user*. Perhitungan *MMR* dilakukan dengan perhitungan iterasi antara kombinasi dua matrik *cosine similiarity* yakni *query releance* dan *similiarity* kalimat.

Pengguna yang menginginkan ruang sampel informasi disekitar *query*, maka harus menetapkan λ pada nilai yang lebih rendah. Sedangkan bagi pengguna yang ingin fokus untuk memperkuat dokumen-dokumen lebih relevan, maka harus menetapkan pada nilai yang lebih dekat dengan . Kalimat dengan nilai *MMR* tertinggi dari setiap perhitungan iterasi akan diambil, kemudian dipilih sebagai ringkasan. Iterasi berhenti pada saat hasil *MMR* maksimum sama dengan 0.



Gambar 2.5 Proses Peringkasan Dengan Menggunakan *MMR* [3]

2.5. Pembobotan TF-IDF

Term Frequency (tf) factor, yaitu faktor yang menentukan bobot term pada suatu dokumen berdasarkan jumlah kemunculannya dalam dokumen tersebut [3]. Nilai jumlah kemunculan suatu kata *(term frequency)* diperhitungkan dalam pemberian bobot terhadap suatu kata. Semakin besar jumlah kemunculan suatu term (*tf* tinggi) dalam dokumen, semakin besar pula bobotnya dalam dokumen atau akan memberikan nilai kesesuian yang semakin besar.

Inverse Document Frequency (idf) factor, yaitu pengurangan dominansi term yang sering muncul di berbagai dokumen. Hal ini diperlukan karena term yang banyak muncul di berbagai dokumen, dapat dianggap sebagai term umum *(common term)* sehingga tidak penting nilainya [3]. Sebaliknya faktor kejarangmunculan kata (term scarcity) dalam koleksi dokumen harus diperhatikan dalam pemberian bobot.

Menurut Mandala (dalam Witten, 1999) ‘Kata yang muncul pada sedikit dokumen harus dipandang sebagai kata yang lebih penting (uncommon tems) daripada kata yang muncul pada banyak dokumen. Pembobotan akan memperhitungkan faktor kebalikan frekuensi dokumen yang mengandung suatu kata (inverse document frequency). Hal ini merupakan usulan dari George Zipf. Zipf mengamati bahwa frekuensi dari sesuatu cenderung kebalikan secara proposional dengan urutannya.

Nilai *idf* sebuah sistem *term* (kata) dapat dihitung menggunakan persamaan sebagai berkut:

$$\boldsymbol{I}DF = log\left(\frac{D}{dfi}\right) \qquad \textbf{2.3}$$

D adalah jumlah dokumen yang berisi *term* (t) dan *dfi* adalah jumlah kemunculan *term* terhadap *D*. adapun algoritma yang digunakan untuk menghitung bobot *(W)* masing-masing dokumen kata kunci *(query)*, yaitu:

$$W_{d,t} = tf_{d,t} * IDF_t \qquad \textbf{2.4}$$

Keterangan:
d = dokumen ke-d
t = *term* ke-t dari kata kunci
tf = *term* frekuensi/frekuensi kata
W = bobot dokumen ke-d terhadap term ke-t

2.6. Teori Graf

Graf digunakan untuk merepresentasikan objek-objek diskrit dan menjelaskan hubungan-hubungan antar objek-objek tersebut. Objek-objek diskrit biasanya digambarkan sebagai titik-titik terpisah (dapat disebut juga noktah), sedangkan hubungan antar objek-objek tersebut digambarkan dalam suatu garis [11]. Graf didefinisikan sebagai sistem yang terdiri dari 2 komponen, yaitu himpunan tak kosong V(G) yang anggotanya disebut titik dan himpunan sisi E(G) yang berupa himpunan pasangan tak terurut dari buah titik berbeda di V(G)

Kemudian, berdasarkan orientasi arah pada sisi, ada 2 jenis graf secara umum (Rio, 2010/2011) :

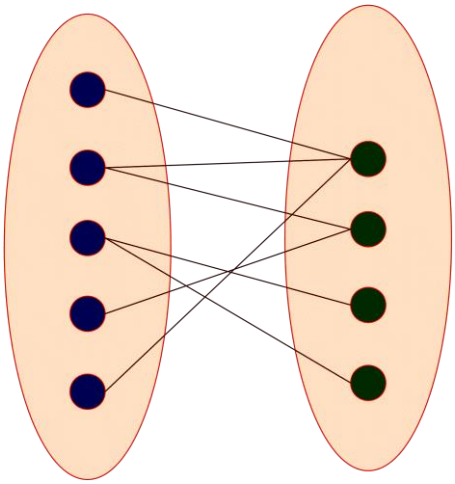
1. Graf tidak bearah, yaitu sisinya *(edge)* tidak memiliki orientasi arah
2. Graf berarah, yaitu sisinya *(edge)* memiliki orientasi arah.

Adapun yang disebut graf unik atau graf berbobot, yaitu suatu graf yang setiap sisinya memiliki bobot/nilai tersendiri.

2.6.1. Bipartit Graf

Graf Bipartit adalah graf yang simpulnya dapat dipisah menjadi dua himpunan, misalnya bagian U dan V, sedemikian sehingga setiap sisi pada grad tersebut menghubungkan sebuah simpul di U dengan sebuah simpul di V, grad tersebut dapat dinyatakan sebagai G(U,V). pula dengan simpul-simpul di V [12].

Kedua himpunan U dan V tersebut juga dapat dianggap sebagai suatu pewarnaan graf dengan dua warna. Apabila semua simpul di U diberi warna biru, dan semua simpul di V diberi warna hijau, masing-masing sisi akan mempunyai ujung yang berbeda warna, sama halnya dengan ketentuan pada masalah perwarnaan graf. Namun demikian, pewarnaan demikian (dua warna). Tidak mungkin dilakukan pada graf non-bipartit. Berikut ini adalah contoh dari graf bipartit.



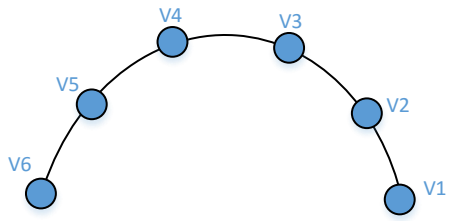
Gambar 2.1 Contoh Graf Bipartit

2.6.2. Graf Basis Data

Graph Database merupakan sebuah teknologi penyimpanan dalam database sebagaimana graph pada umumnya yang terdiri dari kumpulan *edge* dan *vertex* sehingga dapat diakses secara langsung melalui aplikasi dan mendukung OLTP. Graph database ini oleh para ahli digolongkan pada database jenis NoSQL. NoSQL adalah istilah umum yang merujuk pada penyimpanan data yang tidak mengikuti aturan dari Relational Database. NoSQL ini sangat cocok digunakan untuk pemrosesan Big Data dimana data mempunyai ukuran yang sangat besar serta data itu akan terus berkembang.

2.7. Lintasan

Dimisalkan G adalah suatu graf. Dan v dan w adalah 2 titik dalam G. *Walk* didefinisikan sebagai barisan titik-titik berhubungan dan sisi (garis/*edge*) secara berselang-seling, diawali dari titik v diakhiri pada titik w. Lintasan *(path)* dengan asumsi n dari v ke w adalah walk dari v ke w yang semua garisnya berbeda. Dengan demikian dapat dituliskan sebagai $v = v_0\ e_1\ v_1\ e_2\ v_2 \ldots v_{n-1}\ e_n\ v_n = w$ dengan $e_i \neq e_j$ untuk $i \neq j$. Lintasan sederhana dengan panjang n dari v ke w adalah path dari v ke w yang semua titiknya berbeda. Sehingga dapat ditulis dalam bentuk $v = v_0\ e_1\ v_1\ e_2\ v_2 \ldots v_{n-1}\ e_n\ v_n = w$ dengan $e_i \neq e_j$ untuk $i \neq j$ dan $v_k \neq v_m$ untuk $k \neq m$ [13].



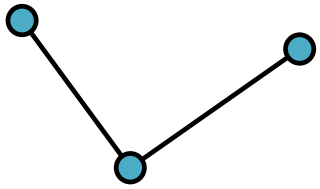
Gambar 2.2 Graf Lintasan *(Path)* [13]

2.8. Tree

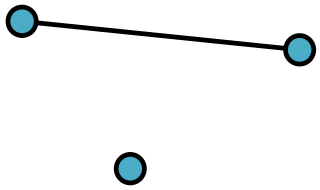
Pohon (atau *tree*) didefinisikan sebagai graf terhubung yang tidak memuat lingkaran [1]. Dengan demikian, suatu lintasan merupakan salah satu contoh dari *tree*. Bobot dari *tree* didefinisikan sebagai jumlah seluruh bobot sisi pada *tree*.

Suatu graf dapat berupa *directed graph* ataupun *undirected graph*, hal ini bergantung pada objek-objek didalam graf tersebut. Jika terdapat hubungan antara

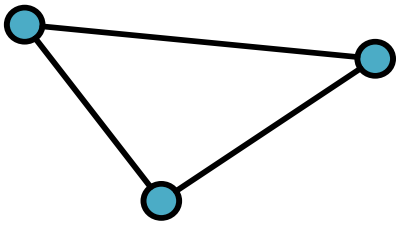
node (verteks) dalam graf bersifat sama untuk kedua arah, maka graf tersebut dikatakan graf berarah. Sedangkan jika hubungan tersebut berbeda, maka graf tersebut dikatakan graf tidak berarah. Di dalam teori graf, dikenal juga istilah graf terhubung dan sirkuit. Graf terhubung adalah graf yang setiap titiknya dapat dicapai oleh semua titik lain dalam graf tersebut, sedangkan yang dimaksud dengan sirkuit adalah lintasan/jalur yang berawal dan berakhir pada titik yang sama.



Gambar 2.3 Graf Terhubung [4]



Gambar 2.4 Graf tidak terhubung [4]



Gambar 2.5 Sirkuit [4]

Teori graf dikembangkan dengan hasil berupa *tree theory*. *Tree* adalah suatu graf terhubung yang tidak memiliki arah dan tidak mengandung sirkuit. Didalamnya dikenal dengan istilah pohon merentang *(spanning tree)*. Jika terdapat sebuah graf terhubung dan tak berarah yang bukan *tree* (artinya graf yang memiliki sirkuit), maka graf tersebut dapat diubah menjadi pohon dengan cara memutuskan sirkuit-sirkuit yang ada. Langkah-langkahnya dapat dipaparkan sebagai berikut:

1. Pertama, dipilih sebuah sirkuit dalam graf
2. Kedua, hapus salah satu garis/sisi atau path dari sirkuit itu
3. Ketiga, Graf tersebut akan tetap terhubung, namun jumlah sirkuitnya berkurang Satu.
4. Keempat, Jika proses penghapusan tersebut dilakukan berulang-ulang hingga tidak ada lagi sirkuit tersisa, maka graf tersebut menjadi sebuah pohon.

Pohon itulah yang dapat disebut sebagai *spanning tree* (pohon merentang). Adapun pohon steiner memiliki kesamaan dengan pohon merentang. Namun, *steiner tree* memiliki aturan tambahan.

Karena tidak ada dua bintang pengganti (star-replacement) melibatkan tepi yang sama, ini membuktikan bahwa solusi dikembalikan oleh algoritma TerminalSteiner memiliki panjang di sebagian besar kali 2ρ panjang solusi optimal. Perhatikan bahwa waktu berjalan dari garis 2, 4, 5, dan 6 dari algoritma TerminalSteiner adalah linier. Oleh karena itu total waktu menjalankan algoritma ini didominasi oleh algoritma ρ-pendekatan yang digunakan untuk masalah pohon Steiner.

2.8.1. Multi-view Graph

Multi-view graph merupakan G = (V,E,A) yakni graf yang terdiri dari *verteks, edge* tidak berarah dan seperangkat *edge* terarah *(arcs). Multi-view graph* dibentuk dari kumpulan *summary* dokumen berita tekstual sebagai *V,* perhitungan *E* yang tidak terarah berdasarkan tingkat kemiripan dari dua buah *summary text* (*V*). Sedangkan, perhitungan pada variabel A *(edge berarah)* berdasarkan perbedaan waktu dimana berita tersebut dipublikasikan yang merepresentasikan waktu kejadian itu berlangsung atau bisa dikatakan sebagai *temestamp.*

2.8.2. Himpunan Dominasi

Dominating Set (Himpunan Dominasi) adalah suatu konsep penentuan suatu titik pada graf dengan ketentuan titik sebagai *dominating set* meng-*cover* atau menutupi titik yang ada disekitarnya dan seminimal mungkin dengan ketentuan graf

sederhana yang tidak memiliki loop dan sisi ganda. Himpunan ini merupakan himpunan bagian *V'* dari himpunan *V(G)* dimana titik-titik yang tidak berada pada *V'* terhubung langsung dengan minimal satu titik pada *V'*.

2.8.3. Steiner tree

Algoritma *Steiner tree* adalah sekumpulan subset atau bagian tertentu dari *vertices* pohon rentang *(spanning tree). Tree* adalah graf terhubung yang tidak memuat lingkaran (sirkuit). Dengan demikian, suatu lintasan merupakan salah satu contoh dari *tree.* Bobot dari *tree* didefinisikan sebagai jumlah seluruh bobot sisi *(edge)* pada *tree.* Pada *vertices* yang merupakan bagian dari *tree* dibagi dua bagian yakni terminal dan simpul non terminal. Terminal adalah pemberian *node* yang harus mengandung solusi atau nilai. *Cost* dari pohon *steiner* didefinisikan sebagai *edge total weight*. Pohon steiner dapat mengandung beberapa simpul nonterminal untuk mengurangi *cost.* Algoritma ini dihasilkan dari algoritma *heuristik,* yakni sebuah aproksimasi. Akibatnya, bobot steiner yang dihasilkan itu belum tentu merupakan bobot yang minimum. Walau demikian terdapat batasan dalam pembobotan *Steiner tree heuristic.*

2.8.4. Aproksimasi *Steiner tree*

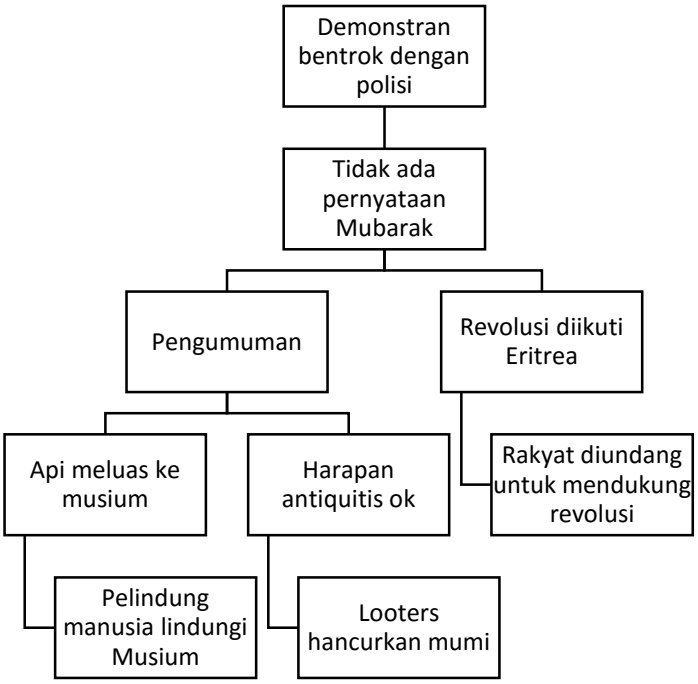
Secara khusus, penggunaan algoritma aproksimasi *steiner tree* akan memberikan waktu perhitungan yang lebih cepat dibandingkan dengan algoritma *steiner tree* yang eksak [5]. Walaupun hasilnya tidak seakurat algoritma eksak, metode aproksiamsi *steiner tree* dapat menentukan tree hampiran untuk graf dengan banyak titik sekitar 100 buah dalam waktu kurang dari 1 detik.

Penggunaan *greedy* sebagai algoritma heuristik untuk melakukan solusi aproksimasi dapat dilakukan untuk menentukan pohon steiner. Hanya saja, penggunaan algoritma *greedy* tidak selalu memberikan solusi yang optimal, namun algoritma *greedy* pasti memberikan solusi mendekati optimum *(approximation)* [14].

2.9. Alur Kronologis

Alur adalah urutan kejadian (peristiwa) yang mempunyai hubungan kausalitas (sebab-akibat) untuk menghantarkan runtunan kejadian tersebut kepada pembaca. Pada dasarnya alur dan plot memiliki perbedaan. Plot adalah suatu cerita yang saling berkaitan secara kronologis untuk menunjukan suatu maksud jalan cerita. Sedangkan alur hanya menunjukan urutan kejadian berdasarkan waktu kejadian tersebut terjadi. Dengan kata lain, urutan peristiwa yang membentuk alur bila ditampilkan berdasarkan pertimbangan struktur hubungan satu sama lain dan keterkaitan kausalitasnya maka dapat membentuk sebuah plot.

Alur kronologis adalah nama lain dari alur maju, alur lurus atau alur progresif. Peristiwa-peristiwa ditampilkan secara kronologis, maju, secara runtut dari awal tahap tengah hingga akhir. Dalam alur ini terdapat hitungan jam, menit, detik, hari dan sebagainya.

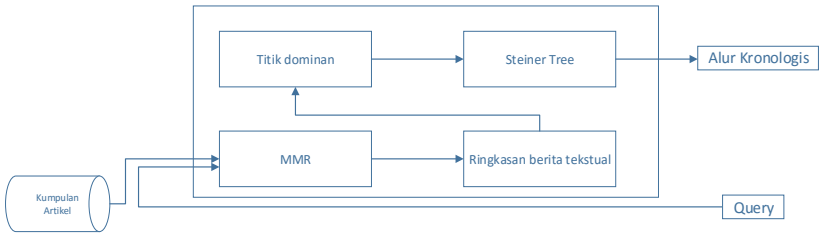


Gambar 2.6 Contoh Alur Kronologis "Revolusi Mesir" [15]

2.9.1. Menentukan Alur Kronologis

Sebagaimana telah dibahas berkaitan dengan alur kronologis dari berita tekstual, alur kronologis ini dihasilkan dari kumpulan hasil ringkasan yang relevan. Secara intuitif, dapat diambil ringkasan terbaik untuk merepresebtasujab ringkasan yang duplikat. Represetatif dari artikel tekstual berupa ringkasan menghasilkan plot dasar untuk setiap fase. Kemudian dari setiap representatif berita tekstual yakni ringkasan tersebut dikoneksikan atau dihubungkan sesuai untuk menggambarkan pengembangan struktur dari peristiwa *(event)*. Secara berurut, untuk mengeliminasikan *noisy* dari ringkasan artikel, hanya teks terpublikasi setelah waktu tertentu yang dapat dianggap sebagai fase berikutnya. Terakhir, ada yang boleh dalam cara yang berbeda untuk menghubungkan representatif artikel dengan dan sebuah hubungan optimal seharusnya menjadi satu yang menghubungkan titik-titik tersebut lebih halus.

Berikut ini adalah arsitektur sistem untuk menghasilkan alur kronologis



Gambar 2.7 Arsitektur Sistem [15]

2.9.2. Alur Kronologis yang Baik

Dalam menggenerasikan alur kronologis atau *Storyline* tentu pembahasan yang paling pokok adalah bagaimana hasil yang diberikan oleh sistem memiliki unsur-unsur berikut (Anunaya Srivastava):

1. Koheren : memiliki keterhubungan atau sangkut paut antara berita.

2. Relevan : memiliki keterkaitan antara dokumen berita yang satu dengan yang
   lainnya.
3. Redudansi kecil atau sedikit: tidak ada pengulangan terhadap berita atau sedikit.
4. Keterkaitan : Keterhubungan antar dokumen.
5. Coverage : mewakili sebuah ulasan.

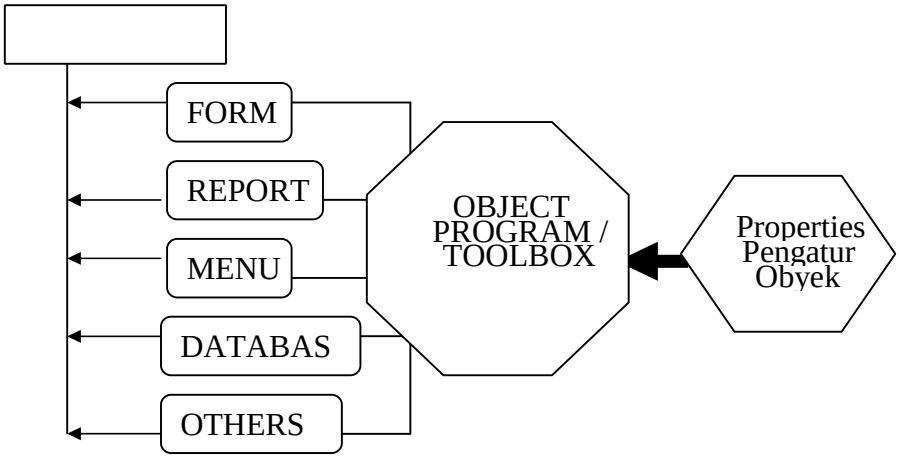
2.10. Tingkat Komulatif Kejadian (Tingkat Insidensi Komulatif)

Dalam pembahasan berkaitan dengan ilmu kesehatan sering di istilahkan sebagai Tingkat Insidensi Komulatif. Tingkat insiden (incident rate) adalah proporsi penduduk yang memiliki kondisi tertentu yang dimulai selama periode waktu. Angka ini diukur dengan salah satu dari tiga cara, yaitu tingkat serangan, tingkat kepadatan kejadian, atau risiko kejadian kumulatif [16].

Namun, dalam hal ini, Tingkat Kumulatif Kejadian menunjukan sejumlah atau banyaknya kejadian maupun peristiwa yang terjadi dalam kurun waktu tertentu.

2.11. OOP (Object Oriented Programming)

OOP *(Object Oriented Programming)* adalah sebuah istilah yang diberikan kepada bahasa pemrograman yang menggunakan tehnik berorientasi atau berbasis pada sebuah obyek dalam pembangunan program aplikasi, maksudnya bahwa orientasi pembuatan program tidak lagi menggunakan orientasi linear melainkan berorientasi pada objek-objek yang terpisah-pisah. Suatu perintah dalam bahasa ini diwakili oleh sebuah Obyek yang didalamnya berisi beberapa perintah-perintah standar sederhana. Obyek ini dikumpulkan dalam Modul form atau Report atau modul lain dan disusun didalam sebuah project. Gambaran tentang pemrograman ini seperti dibawah ini :



Gambar 2.8 Skema OOP [17]

2.11.1. Konsep Dasar Berorientasi Objek

Pemrograman borientasi objek menekankan pada konsep-konsep berikut yang merupakan ciri khasnya, yaitu :

1. Kelas

Kelas adalah kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh *'class of dog'* adalah suatu unit yang terdiri atas definisi-definisi data dan fungsi-fungsi yang menunjuk pada berbagai macam perilaku/turunan dari anjing. Sebuah class adalah dasar dari modularitas dan struktur dalam pemrograman berorientasi *object*. Sebuah class secara tipikal sebaiknya dapat dikenali oleh seorang non-programmer sekalipun terkait dengan domain permasalahan yang ada, dan kode yang terdapat dalam sebuah class sebaiknya (relatif) bersifat mandiri dan independen (sebagaimana kode tersebut digunakan jika tidak menggunakan OOP). Dengan modularitas, struktur dari sebuah program akan terkait dengan aspek-aspek dalam masalah yang akan diselesaikan melalui program tersebut. Cara seperti ini akan menyederhanakan pemetaan dari masalah ke sebuah program ataupun sebaliknya. Objek Objek membungkus data dan fungsi bersama menjadi suatu unit dalam sebuah program komputer; objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek.

2. Abstraksi

Abstraksi merupakan kemampuan sebuah program untuk melewati aspek informasi yang diproses olehnya, yaitu kemampuan untuk memfokus pada inti. Setiap objek dalam sistem melayani sebagai model dari "pelaku" abstrak yang dapat melakukan kerja, laporan dan perubahan keadaannya, dan berkomunikasi dengan objek lainnya dalam sistem, tanpa mengungkapkan bagaimana kelebihan ini diterapkan. Proses, fungsi atau metode dapat juga dibuat abstrak, dan beberapa teknik digunakan untuk mengembangkan sebuah pengabstrakan.

3. Enkapsulasi

Enkapsulasi memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam dari sebuah objek dengan cara yang tidak layak; hanya metode dalam objek tersebut yang diberi ijin untuk mengakses keadaannya. Setiap objek mengakses interface yang menyebutkan bagaimana objek lainnya dapat berinteraksi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

4. Polimorfisme

Polymorphism (polimorfisme) berasal dari bahasa Yunani yang berarti banyak bentuk. Dalam PBO, konsep ini memungkinkan digunakannya suatu interface yang sama untuk memerintah objek agar melakukan aksi atau tindakan yang mungkin secara prinsip sama namun secara proses berbeda. Dalam konsep yang lebih umum sering kali polymorphism disebut dalam istilah satu interface banyak aksi. Contoh yang konkrit dalam dunia nyata yaitu mobil. Mobil yang ada dipasaran terdiri atas berbagai tipe dan berbagai merk, namun semuanya memiliki interface kemudi yang sama, seperti: stir, tongkat transmisi, pedal gas dan rem. Jika seseorang dapat mengemudikan satu jenis mobil saja dari satu merk tertentu, maka orang itu akan dapat mengemudikan hamper semua jenis mobil yang ada, karena semua mobil tersebut menggunakan interface yang sama. Harus diperhatikan disini bahwa interface yang sama tidak berarti cara kerjanya juga sama. Missal pedal gas, jika ditekan maka kecepatan mobil akan

meningkat, tapi bagaiman proses peningkatan kecepatan ini dapat berbeda-beda untuk setiap jenis mobil.

5. Inheritas

Inheritas mengatur polimorfisme dan enkapsulasi dengan mengijinkan objek didefinisikan dan diciptakan dengan jenis khusus dari objek yang sudah ada - objek-objek ini dapat membagi (dan memperluas) perilaku mereka tanpa haru mengimplementasi ulang perilaku tersebut (bahasa berbasis objek tidak selalu memiliki inheritas.) Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah kita tidak melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh anggap kita memiliki sebuah departemen yang memiliki manager, sekretaris, petugas administrasi data dan lainnya. Misal manager tersebut ingin memperoleh data dari bag administrasi maka manager tersebut tidak harus mengambilnya langsung tetapi dapat menyuruh petugas bag administrasi untuk mengambilnya. Pada kasus tersebut seorang manager tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi manager bisa mendapatkan data tersebut melalui objek petugas adminiistrasi. Jadi untuk menyelesaikan suatu masalah dengan kolaborasi antar objek-objek yang ada karena setiap objek memiliki deskripsi tugasnya sendiri.

2.12. UML (Unified Modeling Language)

Unified Modeling Language (UML) merupakan salah satu alat bantu yang dapat digunakan dalam bahasa pemograman yang berorientasi objek, saat ini *UML* akan mulai menjadi standar masa depan bagi industri pengembangan sistem/perangkat lunak yang berorientasi objek sebab pada dasarnya UML digunakan oleh banyak perusahaan raksasa seperti IBM, Microsoft, dan sebagainya [18].

Unified Modeling Language merupakan metode pengembangan perangkat lunak (sistem informasi) dengan menggunakan metode grafis serta merupakan bahasa untuk visualisasi, spesifikasi, konstruksi serta dokumentasi.

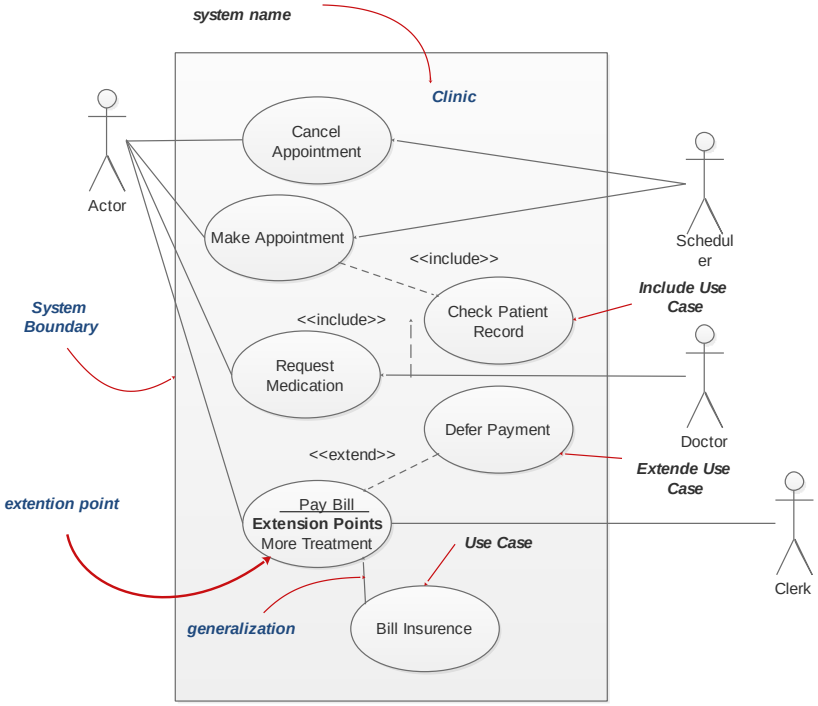
1. *Unified Modeling Language (UML)* adalah bahasa yang telah menjadi standard untuk visualisasi, menetapkan, membangun dan mendokumentasikan arti suatu sistem perangkat lunak.
2. *Unified Modeling Language (UML)* dapat didefinisikan sebagai sebuah bahasa yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem perangkat lunak.
3. *Unified Modeling Language (UML)* merupakan standard modeling language yang terdiri dari kumpulan-kumpulan diagram, dikembangkan untuk membantu para pengembang sistem dan software agar bisa menyelesaikan tugas-tugas seperti :
    a. Spesifikasi
    b. Visualisasi
    c. Desain arsitektur
    d. Konstruksi
    e. Simulasi dan testing
    f. Dokumentasi

Berdasarkan beberapa pendapat yang dikemukakan diatas dapat ditarik kesimpulan bahwa "*Unified Modeling Language (UML)* adalah sebuah bahasa yang berdasarkan grafik atau gambar untuk menvisualisasikan, menspesifikasikan, membangun dan pendokumentasian dari sebuah sistem pengembangan perangkat lunak berbasis Objek *(OOP) (Object Oriented programming)*".

UML sendiri terdiri atas pengelompokkan diagram-diagram sistem menurut aspek atau sudut pandang tertentu. Diagram adalah yang menggambarkan permasalahan maupun solusi dari permasalahan suatu model. UML mempunyai 9 diagram, yaitu; use-case, class, object, state, sequence, collaboration, activity, component, dan deployment diagram. Namun, akan dijelaskan beberapa yang akan digunakan.

2.12.1. *Use Case Diagram*

Diagram Use Case menggambarkan apa saja aktifitas yang dilakukan oleh suatu sistem dari sudut pandang pengamatan luar. yang menjadi persoalan itu apa yang dilakukan bukan bagaimana melakukannya. Diagram Use Case dekat kaitannya dengan kejadian-kejadian. Kejadian (scenario) merupakan contoh apa yang terjadi ketika seseorang berinteraksi dengan sistem.



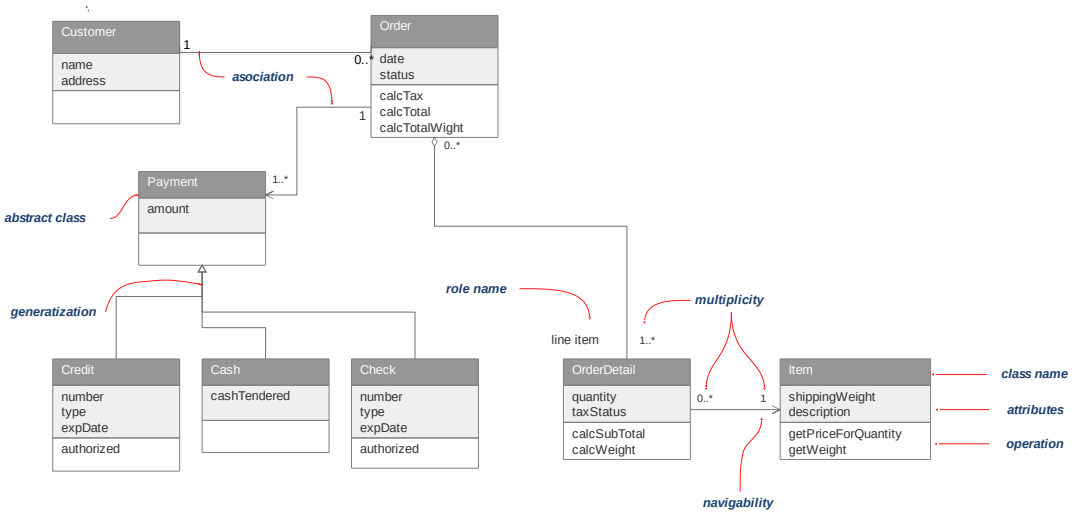
Gambar 2.2 *Use Case Diagram*

2.12.2. *Activity Diagram*

Pada dasarnya diagram Activity sering digunakan oleh flowchart. Diagram ini berhubungan dengan diagram Statechart. Diagram Statechart berfokus pada obyek yang dalam suatu proses (atau proses menjadi suatu obyek), diagram Activity berfokus pada aktifitas-aktifitas yang terjadi yang terkait dalam suatu proses tunggal. Jadi dengan kata lain, diagram ini menunjukkan bagaimana aktifitas-aktifitas tersebut bergantung satu sama lain.

2.12.3. Class Diagram

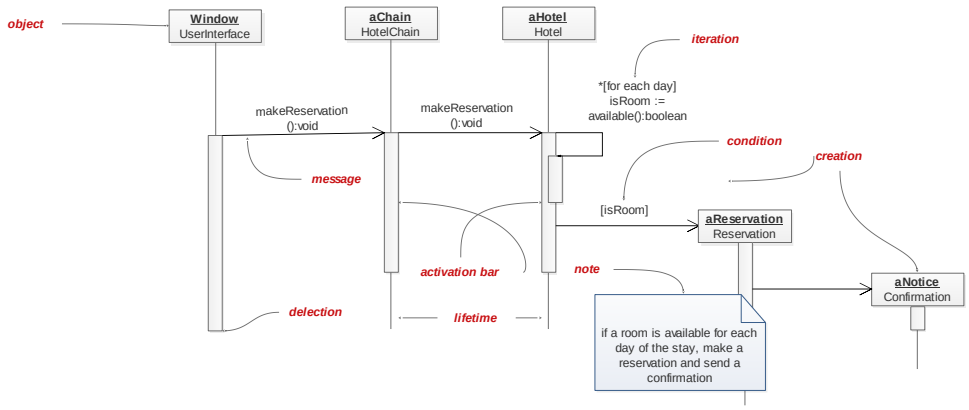
Diagram Class memberikan pandangan secara luas dari suatu sistem dengan menunjukan kelas-kelasnya dan hubungan mereka. Diagram Class bersifat statis; menggambarkan hubungan apa yang terjadi bukan apa yang terjadi jika mereka berhubungan.



Gambar 2.3 Class Diagram

2.12.4. Sequence Diagram

Diagram Class dan diagram Object merupakan suatu gambaran model statis. Namun ada juga yang bersifat dinamis, seperti Diagram Interaction. Diagram sequence merupakan salah satu diagram Interaction yang menjelaskan bagaimana suatu operasi itu dilakukan; message (pesan) apa yang dikirim dan kapan pelaksanaannya. Diagram ini diatur berdasarkan waktu. Obyek-obyek yang berkaitan dengan proses berjalannya operasi diurutkan dari kiri ke kanan berdasarkan waktu terjadinya dalam pesan yang terurut.



Gambar 2.4 Sequence Diagram

2.13. Pengujian Perangkat Lunak

Ujicoba perangkat lunak merupakan elemen yang kritis dan perepresentasikan tinjauan ulan yang menyeluruh terhadap spesifikasi, desain dan pengkodean. Ujicoba merepresentasikan ketidanormalan yang terjadi pada pengembangan perangkat lunak [19]. Dari teknik yang dapat diterapkan dalam menguji suatu perangkat lunak dapat menggunakan dua jenis pengujian yang umum digunakan, yakni:

2.13.1. Pengujian Whitebox

Metode ujicoba blackbox memfokuskan pada keperluan fungsional dari software [19]. Karna itu ujicoba blackbox memungkinkan pengembang software untuk membuat himpunan kondisi input yang akan melatih seluruh syarat-syarat fungsional suatu program. Ujicoba blackbox bukan merupakan alternatif dari ujicoba whitebox, tetapi merupakan pendekatan yang melengkapi untuk menemukan kesalahan lainnya, selain menggunakan metode whitebox. Ujicoba blackbox berusaha untuk menemukan kesalahan dalam beberapa kategori, diantaranya ~~(Ayuliana, 2009~~):

1. Fungsi-fungsi yang salah atau hilang

2. Kesalahan interface
3. Kesalahan dalam struktur data atau akses database eksternal
4. Kesalahan performa
5. kesalahan inisialisasi dan terminasi

2.13.2. Pengujian Ketergunaan *(Usability Testing)*

Ketergunaan atau *Usability* adalah ketika suatu produk memiliki kegunaan yakni seorang pengguna dapat melakukan apa yang dia ingin lakukan dengan cara yang ia harapkan untuk dapat melakukannya, tanpa halangan, ragu-ragu atau pertanyaan [7]. Dalam konsep *usability* ada beberapa atribut yang menunjukan bahwa suatu produk atau sistem memiliki ketergunaan yakni suatu produk harus melingkupi beberapa hal [7]:

1. *Usefulness*

   Atribut ini menyangkut sejauh mana suatu produk memungkinkan pengguna dalam mencapai tujuan-tujuannya dan penilaian pengguna terhadap kesediannya untuk menggunakan produk tersebut. Namun, dorongan untuk menggunakan sistem atau produk tersebut akan tidak masuk akal bilamana produk hanya dibiarkan tanpa ada ketergunaan. Jika sistem mudah digunakan, mudah untuk dipelajari dan memuaskan untuk digunakan tapi tidak mencapai tujuan yang spesifik dari pengguna tertentu, hal itu pun tidak akan berguna meskipun diberikan secara gratis.

   Artinya, pada tahap awal pengembangan produk, harus memastikan produk atau fitur apa yang diperlukan dalam sistem sebagai unsur-unsur kegunaan.

2. *Efficiency*

   Efesiensi adalah kecepatan yang dapat dicapai untuk merealisasikan tujuan pengguna dengan akurasi tertentu dan ukuran waktu yang normal. Misalnya, menetapkan pengujian ketergunaan yang mengatakan “95% dari semua pengguna akan dapat memuat (menjalankan) perangkat lunak dalam 10 menit”

3. *Effectiveness*

   Efektivitas mengacu pada sejauh mana produk berperilaku dimana pengguna berharap akan kemudahan. Selain itu, pengguna padat menggunakannya untuk melakukan apa yang mereka inginkan. Hal ini biasanya diukur secara kuantitatif dengan tingkat kesalahan. Kegunaan dalam pengujian berdasarkan ukuran efektivitas sekaligus untuk efesiensi harus diikat kedalam beberapa persentase dari total pengguna. Memperluas contoh dari efisiensi yang mungkin dinyatakan sebagai 95% dari semua pengguna akan dapat memuat atau menjalankan perangkat lunak dengan benar pada usaha yang pertama.

4. *Satisfying*

   Mengacu pada persepsi pengguna, perasaan maupun pendapat dari pada produk tersebut. Biaranya diperoleh dari kedua pertanyaan baik tertulis maupun lisan. Pengguna lebih mungkin untuk melakukan dengan baik pada produk yang memenuhi kebutuhan mereka sehingga memberikan kepuasan. Biasanya, pengguna diminta untuk menilai produk atau sistem yang mereka coba dan ini sering dapat mengungkapkan penyebab dan alasan bagaimana suatu masalah dapat terjadi.

5. *Learnable*

   Learnable merupakan bagian dari efektivitas dan memiliki kaitan erat dengan kemampuan pengguna dalam mengoperasikan sistem untuk beberapa tingkat yang ditetapkan secara kompetensi yang telah ditentukan setelah beberapa jumlah dan periode pelatihan. Hal ini juga dapat mengacu pada kemampuan pengguna yang jarang mempelajari kembali sistem setelah periode tidak aktif.

6. *Accessible*

   Aksesibilitas dan kegunaan adalah dua istilah yang memiliki kesamaan layaknya saudara. Dalam arti luas, aksesibilitas adalah berkaitan dengan jalan atau kepemilikan suatu akses kedalam produk atau sistem yang dibutuhkan untuk mencapai tujuan. Dalam

pembahasan ini, aksesibilitas mempertimbangkan bagi penyandang cacat dapat memperjelas dan menyederhanakan desain untuk orang-orang yang menghadapi keterbatasan sementara (misal, cedera) atau situasional (misalnya kondisi lingkungan perhatian atau buruk dibagi seperti cahaya terang atau tidak cukup cahaya). Ada banyak perangkat pedoman yang tersedia dalam membuat desain aksesabel.

Membuat hal-hal yang lebih bermanfaat dan dapat diakses adalah bagian dari disiplin yang lebih besar dari desain yang berpusat pada pengguna.

2.13.3. Notasi Big O

Berikut ini merupakan beberapa contoh nilai Big-O yang bisa ditemukan dalam beberapa literatur [20] :

1. **O(1):** konstan. Algoritma dengan O-Besar O(1) dieksekusi di kecepatan yang sama tidak tergantung pada data masukannya. Sebagai contoh, algoritma yang selalu menghasilkan nilai yang sama apapun nilai masukannya dapat dipandang sebagai algoritma dengan efisiensi O(1). Contoh lain adalah algoritma untuk menentukan suatu bilangan genap atau ganjil, look table atau hash table adalah konstan ukuran tabel tetap. O(log n): logaritmik. Algoritma yang didasarkan pada pohon biner kerap mempunyai efisiensi O(log n). Hal ini disebabkan karena BST (binary search tree) yang sangat seimbang mempunyai banyak lapisan log dan untuk mencari sembarang elemen di BST memerlukan penelusuran satu simpul di setiap lapisan. Algoritma untuk mencari suatu item di array terurut menggunakan pencarian biner atau balanced search tree serta semua operasi di binomial heap adalah O(log n).
2. **O(n):** linear. Pencarian linear untuk menemukan suatu elemen dalam suatu array tak urut dengan n elemen. Algoritma tipe ini hanya memerlukan satu putaran untuk seluruh array. Keadaan terbaik terjadi jika yang dicari terletak di posisi pertama sehingga O(1). Keadaan terjelek terjadi jika elemen yang dicari terletak di posisi terakhir atau tidak ada dalam array. Untuk itu

algoritma harus menelusuri seluruh array untuk mencek setiap elemen dalam array yang berarti O(n). Keadaan rerata berdasarkan pada asumsi bahwa elemen yang dicari ada dalam array dan setiap elemen mempunyai peluang yang sama untuk ditemukan. Pencarian hanya perlu mengunjungi n/2. Contoh lain adalah link-list. Akses elemen pada link-list adalah O(n) karena link-list tidak mendukung akses acak. Menambahkan dua integer n-bit menggunakan ripple carry adalah O(n).

3. **O(n log n):** loglinear, quasilinear atau linearithmik. Algoritma pengurutan yang baik kerap mempunyai order O(n log n). Contoh algoritma dengan efisiensi ini adalah algoritma yang tergabung dalam kelompok devide and conquer (DAC) seperti quick sort (best dan average case) dan merge sort. Algoritma merge sort membagi array menjadi dua bagian, urutkan kedua subarray secara rekursif dengan memanggil dirinya sendiri dan kemudian gabung kembali hasilnya kedalam array tunggal. Karena setiap kali membagi array menjadi dua bagian maka perulangan luar mempunyai efisiensi log n, dan untuk setiap "level" array yang dibagi menjadi dua bagian maka harus menggabungkan kembali semua elemen ke dalam satu array, operasinya membutuhkan order n. Algoritma pengurutan paling popular quick sort mempunyai kinerja terbaik O(n log n) yang membuatnya menjadi algoritma pengurutan yang sangat cepat didasarkan pada asumsi bahwa semua nilai berbeda dan dalam keadaan acak. Tetapi data masukan yang paling jelek akan membuat kinerjanya menjadi O(n2
). Contoh lain algoritma loglinear adalah Fast Fourier Transform (FFT) dan heapsort.
4. **O(n2):** kuadratik. Cukup efisien karena masih tetap dalam rentang waktu polinomial. Contoh biasanya ada dalam kelompok pengurutan data seperti selection sort, insertion sort, quick sort (worst case), shell sort, bubble sort (worst case atau implementasi naïf). Insertion sort yang diaplikasikan pada array n elemen berdasarkan pada asumsi semua nilai elemen berbeda dan dalam keadaan acak. Secara rerata, separuh dari elemen dalam array A1 ... Aj lebih kecil daripada elemen Aj+1 dan separuh lainnya lebih besar.

Dengan demikian algoritma membandingkan secara rerata elemen ke j+1 yang akan disisipkan separuh sub-array yang telah urut, sehingga tj = j/2. Hasil akhirnya waktu eksekusi kasus rerata adalah fungsi kuadratik dari ukuran masukan; sama dengan waktu eksekusi kasus terjelek. Contoh lain algoritma kuadratik adalah mengalikan dua bilangan n-digit menggunakan algoritma sederhana.

5. **O(2n):** eksponensial. Efisiensi non-polinomial yang paling penting adalah exponential time. Banyak masalah penting yang hanya dapat diselesaikan oleh algoritma dengan efisiensi seperti ini.

2.14. Perangkat Lunak Pendukung

2.14.1. Apache

Apache adalah sebuah nama web server yang bertanggung jawab pada request-response HTTP dan logging informasi secara detail (kegunaan basicnya) [21]. Selain itu, Apache juga diartikan sebagai suatu web server yang kompak, modular, mengikuti standar protokol HTTP, dan tentu saja sangat digemari.

Kesimpulan ini bisa didapatkan dari jumlah pengguna yang jauh melebihi para pesaingnya. Sesuai hasil survai yang dilakukan oleh Netcraft, bulan Januari 2005 saja jumlahnya tidak kurang dari 68% pangsa web server yang berjalan di Internet. Ini berarti jika semua web server selain Apache digabung, masih belum bisa mengalahkan jumlah Apache.

Saat ini ada dua versi Apache yang bisa dipakai untuk server produksi, yaitu versi mayor 2.0 dan versi mayor 1.3. Apache merupakan webserver yang paling banyak digunakan saat ini. Hal ini disebabkan oleh beberapa sebab, di antaranya adalah karena sifatnya yang opensource dan mudahnya mengkostumisasikannya. diantaranya dengan menambahkan support secure protocol melalui ssl dan konektifitasnya dengan database server melalui bahasa scripting PHP.

2.14.2. MySQL

MySQL adalah sebuah *database* server yang dibuat oleh Tcx Data Konsult AB. Saat ini MySQL telah digunakan oleh perusahaan-perusahaan terkemuka di

seluruh dunia, diantaranya Silicon Graphics *(http://www.sgi.com)*, Siemens Nixdorf *(http://www.siemens.com)*, Alesis Digital Studio Electronics (http://www.alesis.com) dan masih banyak perusahaan-perusahaan terkemuka lainnya yang menggunakan MySQL. Perusahaan-perusahaan tersebut dapat dilihat pada MySQL user's list di *http://www.mysql.com/information/userlist.htm*. MySQL adalah sebuah *text based database server*, artinya MySQL tidak dibuat dalam bentuk aplikasi yang memiliki *Graphical User Interface*

2.14.3. PHP

PHP (*Personal Home Page*) dahulunya merupakan objek pribadi dari Rasmus Lerdorf (PHP versi 1) yang digunakan untuk membuat homepage pribadinya. PHP merupakan scripting yang menyatu dalam HTML dan berada di server (*server side* HTML – *embeded scripting*) yang digunakan untuk membuat halaman web yang dinamis.

Dinamis berarti halaman yang akan ditampilkan dibuat saat halaman itu diminta oleh *client*. Mekanisme ini menyebabkan informasi yang diterima *client* selalu yang terbaru. Semua script PHP dieksekusi pada server dimana script tersebut dijalankan.

Style standar PHP selalu diawali dengan *<?php dan diakhiri dengan tanda ?>* dan style PHP ini sangat mirip dengan program XML atau seperti pada C atau Perl. Selain itu PHP juga mendukung komentar C, C++, dan Unix shell-style.

2.14.4. PHPMyAdmin

Pengelolaan database dengan MYSQL harus dilakukan dengan mengetikkan baris-baris perintah yang sesuai (command line) untuk setiap maksud tertentu [22]. Jika anda ingin membuat database, ketikkan baris perintah yang sesuai untuk membuat database. Jika kita ingin menghapus tabel, ketikkan baris perintah yang sesuai untuk menghapus tabel. Hal tersebut tentu cukup menyulitkan karena kita harus hafal dan mengetikkan perintahnya satu persatu. Banyak sekali perangkat lunak yang dapat dimanfaatkan untuk mengelola data base dalam MySQL, salah

satunya adalah phpMyAdmin. Dengan phpMyAdmin kita dapat membuat tabel, mengisi data dan lain-lain dengan mudah tanpa harus hafal perintahnya.

2.14.5. XAMPP

Fungsinya adalah sebagai server yang berdiri sendiri (localhost), yang terdiri atas program Apache HTTP Server, MySQL database, dan penerjemah bahasa yang ditulis dengan bahasa pemrograman PHP dan Perl. Nama XAMPP merupakan singkatan dari X (empat sistem operasi apapun), Apache, MySQL, PHP dan Perl [23]. Program ini tersedia dalam GNU General Public License dan bebas, merupakan web server yang mudah digunakan yang dapat melayani tampilan halaman web yang dinamis. Untuk mendapatkanya dapat mendownload langsung dari web resminya.

2.14.6. Adobe Dreamwever CC

Dreamweaver memiliki fitur browser yang terintegrasi untuk melihat halaman web yang dikembangkan di jendela pratinjau program sendiri agar konten memungkinkan untuk terbuka di web browser yang telah terinstal [24]. Aplikasi ini menyediakan transfer dan fitur sinkronisasi, kemampuan untuk mencari dan mengganti baris teks atau kode untuk mencari kata atau kalimat biasa di seluruh situs, dan templating feature yang memungkinkan untuk berbagi satu sumber kode atau memperbarui tata letak di seluruh situs tanpa server side includes atau scripting. Behavior Panel juga memungkinkan penggunaan JavaScript dasar tanpa pengetahuan coding, dan integrasi dengan *Adobe Spry Ajax framework* menawarkan akses mudah ke konten yang dibuat secara dinamis dan *interface*.