APLIKASI DETEKSI PLAGIARISME BERDASARKAN *STRING-MATCHING* MENGGUNAKAN ALGORITMA RABIN-KARP

SKRIPSI

oleh :
DENI HADI SANTOSO
NIM 07650041

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2014

APLIKASI DETEKSI PLAGIARISME BERDASARKAN *STRING-MATCHING* MENGGUNAKAN ALGORITMA RABIN-KARP

SKRIPSI

Diajukan kepada :
Fakultas Sains dan Teknologi
Universitas Islam Negeri
Maulana Malik Ibrahim Malang
Untuk Memenuhi Salah Satu Persyaratan Dalam
Memperoleh Gelar Sarjana Komputer (S. Kom)

oleh :
DENI HADI SANTOSO
NIM 07650041

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG
2014

HALAMAN PERSETUJUAN
APLIKASI DETEKSI PLAGIARISME BERDASARKAN *STRING-MATCHING* MENGGUNAKAN ALGORITMA RABIN-KARP

SKRIPSI

Oleh :

Nama	: Deni Hadi Santoso
NIM	: 07650041
Jurusan	: Teknik Informatika
Fakultas	: Sains dan Teknologi

Telah Disetujui, 7 Juli 2014

Dosen Pembimbing I	**Dosen Pembimbing II**
Dr. Cahyo Crysdian	**Dr. M. Amin Hariyadi, MT**
NIP. 19740424 200901 1 008	**NIP. 19670118 200501 1 001**

Mengetahui,
Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

HALAMAN PENGESAHAN
APLIKASI DETEKSI PLAGIARISME BERDASARKAN *STRING-MATCHING* MENGGUNAKAN ALGORITMA RABIN-KARP

SKRIPSI

Oleh :
Deni Hadi Santoso
NIM 07650041

Telah Dipertahankan Di Depan Dewan Penguji Skripsi
Dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer (S. Kom)

Tanggal, 10 Juli 2014

Susunan Dewan Penguji :			**Tanda Tangan**
1. Penguji Utama	:	**A'la Syauqi, M.Kom** **NIP. 19771201 200801 1007**	**( )**
2. Ketua Penguji	:	**Fatchurrahman, M.Kom** **NIP.19700731 20050 1 1 002**	**( )**
3. Sekretaris	:	**Dr. Cahyo Crysdian** **NIP. 19740424 200901 1 008**	**( )**
4. Anggota Penguji	:	**Dr. M. Amin Hariyadi, MT** **NIP. 19670118 200501 1 001**	**( )**

Mengetahui,
Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

HALAMAN PERNYATAAN

ORISINALITAS PENELITIAN

Saya yang bertandatangan di bawah ini :

Nama	:	Deni Hadi Santoso
NIM	:	07650041
Fakultas/Jurusan	:	Sains dan Teknologi / Teknik Informatika
Judul Penelitian	:	Aplikasi Deteksi Plagiat Berdasarkan *String-Matching* Menggunakan Algoritma Rabin-Karp

Menyatakan dengan sebenarnya bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri, bukan merupakan pengambil alihan data, tulisan atau pikiran oarang lain yang saya akui sebagai hasil tulisan atau pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka. Apabila di kemudian hari terbukti atau dapat dibuktikan skripsi ini hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 18 Juli 2014
Yang Membuat Pernyataan,

Deni Hadi Santoso
NIM. 07650041

HALAMAN MOTTO

"DI DUNIA TIDAK LEPAS DARI KELEBIHAN DAN KEKURANGAN
MENUTUPI KELEMAHAN BUKANLAH SESUATU YANG SALAH
MEMANFAATKAN KELEBIHAN ADALAH HAL YANG BAIK
NAMUN AKAN LEBIH BAIK LAGI MENJADIKAN KELEMAHAN
SEBAGAI KELEBIHAN"

"JANGAN PERNAH BERPIKIR TENTANG APA ATAU
SEBERAPA BANYAK HAL YANG HARUS DIKERJAKAN
BERPIKIRLAH BAGAIMAMA MENGERJAKANNYA"

HALAMAN PERSEMBAHAN

ٱلْحَمْدُ لِلَّهِ رَبِّ ٱلْعَٰلَمِينَ

Segala Puji bagi Allah SWT tuhan semesta alam,

Semoga Shalawat dan Salam senantiasa tercurahkan kepada Baginda Rosullulloh Saw dan Para Generasi Penerusnya,

kupersembahkan sebuah karya sederhana untuk orang-orang yang paling kusayangi dan aku banggakan

Ayah dan Ibu

Dwi Indro Basuki dan Siti Rohmaningsih

Serta Seluruh keluarga besarku

Kakak, Adik dan keponakan-keponakanku

Semoga Allah SWT melindungi dan menjaga mereka semua.

KATA PENGANTAR

Segala puji bagi Allah SWT berkat rahmat, taufiq dan hidayah-Nya kepada penulis sehingga bisa menyelesaikan skripsi dengan judul "Aplikasi Deteksi Plagiat Berdasarkan *String-Matching* Menggunakan Algoritma Rabin-Karp" dengan baik. Shalawat dan salam semoga tercurah kepada Nabi Muhammad SAW, para sahabat dan para pengikut setia Beliau.

Penulis menyadari bahwa penulisan skripsi tidak akan terwujud tanpa adanya bantuan dari semua pihak, oleh karena itu tak lupa penulis ungkapkan rasa terimakasih yang sedalam-dalamnya kepada:

1. Dr. Cahyo Crysdian selaku Dosen pembimbing I dan Ketua Jurusan Teknologi Informatika Universitas Islam Negeri (UIN) Maulana Malik Ibrahim Malang.
2. Dr. M. Amin Hariyadi, MT selaku Dosen Pembimbing II atas bimbingan, pengarahan, dan kesabarannya dalam membimbing hingga penulisan skripsi ini dapat terselesaikan.
3. Fatchurrahman, M.Kom selaku dosen wali yang telah membimbing dari awal sampai akhir kuliah di jurusan Teknik Informatika.
4. Seluruh Dosen Teknik Informatika dan segenap perangkat Universitas Islam Negeri (UIN) Maulana Malik Ibrahim Malang.

5. Teman seperjuanganku Adita Eka Mardiana dan teman-teman angkatan 2007. Tak lupa semua pihak yang telah memotivasi dan membantu dalam proses penyelesaian skripsi ini yang tidak bisa disebutkan satu per satu.
6. Teman-teman Teknik Informatika atas segala kebersamaannya.
7. V12D, Meymey, Cahaya Bulan, STADA, cip3ng, Tajur, Jepp dan teman-teman di berbagai forum atas segala bantuan yang telah diberikan.
8. Tim Pengembang Apache-POI, *application Programming Interface(API)*, yang digunakan dalam penelitian ini.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih terdapat kekurangan. Penulis berharap semoga skripsi ini dapat memberikan manfaat kepada pembaca dan khususnya bermanfaat bagi penulis secara pribadi.

Wassalamu'alaikum Wr. Wb.

Malang, 18 Juli 2014

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Santoso, Deni Hadi. 2014. 07650041 .**Aplikasi Deteksi Plagiarisme Berdasarkan *String-Matching* Menggunakan Algoritma Rabin-Karp.** Skripsi. Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.
Pembimbing : (I) Dr. Cahyo Crysdian (II) Dr. M. Amin Hariyadi, MT

Kata Kunci : *Plagiarisme, String-matching, deteksi, Rabin-Karp.*

Dalam melakukan pencegahan mendeteksi plagiarisme sering mengalami masalah efisiensi dan efektifitas. Sehingga diperlukan aplikasi untuk membantu mendeteksi plagiat. Ada beberapa metode yang dapat digunakan untuk mendeteksi plagiarisme, salah satu metodenya dengan menggunakan konsep *string-matching*. Terdapat banyak algoritma yang biasa digunakan dalam konsep *string-matching*. Salah satunya adalah algoritma Rabin-Karp.

Algoritma Rabin-Karp memanfaatkan nilai hash untuk mencari kesamaan string. Penggunaan nilai hash menyebabkan panjang *pattern* tidak terlalu berpengaruh, sehingga algoritma ini dirasa sesuai untuk digunakan sebagai dasar aplikasi untuk mendeteksi plagiarisme. Hal ini, disebabkan proses pendeteksian menggunakan *pattern* berupa kalimat yang terdiri dari beberapa karakter.

Aplikasi menggunakan konsep *string-matching* dengan Algortima Rabin-Karp sebagai dasar untuk mendeteksi plagiarisme. Dalam melakukan pendeteksian, aplikasi ini membutukan waktu 0,12 untuk satu kalimat dengan sepuluh dokumen sebagai referensi. Dengan penggunaan CPU sebesar 51,34 % dan memori sebesar 188,2 Mb.

ABSTRACT

Santoso, Deni Hadi. 2014. **Application Plagiarism Detection Based on String-Matching Algorithm Using Rabin-Karp**. Thesis. Informatics Department of Faculty of Science and Technology, Maulana Malik Ibrahim State Islamic University, Malang.
Promotor : (I) Dr. Cahyo Crysdian
(II) Dr. M. Amin Hariyadi, MT

In detecting plagiarism prevention often have problems of efficiency and effectiveness. So that the required application to help detect plagiarism. There are several methods that can be used to detect plagiarism, one of the methods using string-matching concept. There are many algorithms commonly used in string-matching concept. One is the Rabin-Karp algorithm.

Rabin-Karp algorithm utilizes the hash value for find the similarity of string. The use of hash values cause the pattern length was not too influential, so the algorithm is considered suitable for use as the basis of application for detecting plagiarism. This, due to the detection process using the pattern in the form of sentences consisting of several characters.

Applications using the concept of the string-matching with the Rabin-Karp algorithms as a basis for detecting plagiarism. In did detection, these applications requiring a time of 0.12 for a single sentence with ten documents as a reference. With the CPU usage by 51.34% and 188.2 Mb of memory.

Key words: *Plagiarism, string-matching, detection, Rabin-Karp*.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Plagiarisme atau biasa disebut plagiat menurut Kamus Besar Bahasa Indonesia adalah penjiplakan atau pengambilan karangan, pendapat, dan sebagainya dari orang lain dan menjadikannya seolah karangan dan pendapat sendiri (KBBI,1997:775). Sedangkan menurut Peraturan Menteri Pendidikan Nasional Republik Indonesia No 17 Tahun 2010, Plagiat adalah perbuatan secara sengaja maupun tidak sengaja dalam memperoleh atau mencoba memperoleh kredit atau nilai untuk suatu karya ilmiah dengan mengutip sebagian atau seluruh karya dan/atau karya ilmiah pihak lain yang diakui sebagai karya ilmiahnya, tanpa menyatakan sumber secara tepat dan memadai.

Pada dunia pendidikan, akhir-akhir ini praktik plagiat berada dalam taraf yang memprihatinkan. Seperti ditulis dalam situs berita Republika.co.id dimana selama setahun terakhir, tepatnya sepanjang 2012 hingga pertengahan 2013, lebih dari 100 dosen setingkat lektor, lektor kepala, dan guru besar, di Indonesia tertangkap melakukan plagiat (penjiplakan).

Plagiat juga menunjukan sikap kurang menghormati dan menghargai hasil karya orang lain. Dalam menghasilkan sebuah karya seseorang melalui tahapan-tahapan yang tidak mudah. Sehingga bisa dikatakan hasil karya adalah sebuah harta bagi pembuatnya dan perlu untuk dihormati dan dihargai. Di agama islam

dianjurkan untuk menghormati dan menghargai karya sebagai hak milik dari seseorang.

Seperti dijelaskan di dalam Alquran surat Al Baqoroh ayat 188 sebagai berikut:

وَلَا تَأْكُلُوٓاْ أَمْوَٰلَكُم بَيْنَكُم بِٱلْبَٰطِلِ وَتُدْلُواْ بِهَآ إِلَى ٱلْحُكَّامِ لِتَأْكُلُواْ فَرِيقًا مِّنْ أَمْوَٰلِ ٱلنَّاسِ بِٱلْإِثْمِ وَأَنتُمْ تَعْلَمُونَ ﴿١٨٨﴾

dan janganlah sebahagian kamu memakan harta sebahagian yang lain di antara kamu dengan jalan yang bathil dan (janganlah) kamu membawa (urusan) harta itu kepada hakim, supaya kamu dapat memakan sebahagian daripada harta benda orang lain itu dengan (jalan berbuat) dosa, Padahal kamu mengetahui (QS. Al Baqarah :188)

Dalam arti surat diatas, dijelaskan tentang larangan memakan harta orang dengan jalan yang bathil. Dan bila di analogikan ke dalam kasus plagiat, pengakuan hasil karya orang lain, baik sebagian maupun seluruh karya tersebut bisa dikatakan sebagai pengambilan harta intelektual. Dan dikawatirkan akan menimbulkan pertikaian diantara pelaku plagiarisme dan pemilik karya yang asli.

Dalam hal kerugian, mengambil harta orang lain menimbulkan kerugian, baik bagi pengambil harta maupun orang yang diambil hartanya. Untuk praktik plagiarisme, pelaku plagiarisme mendapatkan kerugian yang paling besar. Pelaku plagiarisme tidak mendapatkan ilmu dari hasil karya yang telah dibuatnya. Disamping itu sifat enggan untuk belajar dan berusaha akan terdapat dan semakin meningkat pada diri pelaku plagiarisme. Dimana sifat-sifat tersebut akan membawa keburukan dimasa yang akan datang.

Dalam penelitian Suwarjo et al (2012) menyimpulkan bahwa bentuk plagiat total adalah bentuk plagiat yang paling dominan dilakukan mahasiswa.

Bentuk plagiat total dalam penelitian Suwarjo et al (2012) adalah mengacu dan mengutip istilah, kata/kalimat, data/info dari sumber tanpa menyebutkan sumber dalam catatan kutipan dan tanpa menyatakan sumber yang memadai. Masih menurut Suwarjo et al (2012) hal ini bisa dipahami karena sebagian besar mahasiswa kurang teliti dan maraknya budaya *copy-paste* di kalangan mahasiswa tanpa mempertimbangkan pentingnya mencantumkan sumber kutipan ke dalam skripsi baik di dalam bab maupun di dalam daftar pustaka. *Copy-paste* sendiri merupakan salah satu bentuk plagiat. Gipp dan Meuschke (2011) mendefinisikan *copy-paste plagiarism* sebagai bentuk plagiat yang menyalin setiap kata tanpa perubahan.

Menyimpulkan dari penjelasan diatas tindakan pencegahan praktik plagiat harus dilakukan. Tindakan pencegahan plagiat ini sesuai dengan Peraturan Menteri Pendidikan Nasional Republik Indonesia No 17 Tahun 2010 tentang Pencegahan dan Penanggulangan Plagiat di Perguruan Tinggi. Menurut Peraturan tersebut pencegahan plagiat adalah tindakan preventif yang dilakukan oleh pimpinan perguruan tinggi yang bertujuan agar tidak terjadi plagiat di lingkungan perguruan tingginya. Salah satu bentuk pencegahan plagiat adalah dengan pendeteksian terhadap praktik plagiat.

Dalam melakukan pencegahan dengan mendeteksi plagiat sering mengalami masalah efisiensi dan efektifitas. Ada beberapa faktor yang menjadi penghambat, salah satunya adalah banyaknya dokumen yang harus diperiksa. Dimana setiap dokumen diperiksa dan dibandingkan satu-persatu. Tentunya bila dilakukan secara manual akan memakan waktu yang cukup lama.

Mengaca pada penjelasan diatas diperlukan aplikasi untuk membantu mendeteksi plagiat. Ada beberapa metode yang dapat digunakan untuk mendeteksi plagiat. Salah satu dengan menggunakan konsep *string-matching* atau pencocokan string. Prinsip dasar dari *string-matching* adalah mencocokan *pattern* yang memiliki panjang m, dengan text yang memiliki panjang n, dimana nilai n lebih besar daripada nilai m. Tujuan dari *string-matching* adalah untuk membuktikan bahwa *pattern* adalah bagian dari text. Terdapat banyak metode yang biasa digunakan dalam konsep *string-matching.* Salah satunya adalah algoritma Rabin-Karp.

Algoritma Rabin-Karp dikembangkan oleh M.O. Rabin dan R.A. Karp. Algoritma ini menggunakan nilai hash dalam *string-matching.* Algoritma Rabin-Karp menggunakan fungsi hash untuk mencari nilai hash dari *pattern* dan bagian text dengan panjang sama dengan *pattern,* lalu membandingkannya. Penggunaan fungsi hash pada proses pencarian terlihat akan memakan waktu lebih lama, karena dalam proses pencariannya dibutuhkan proses mengubah *pattern* dan text menjadi nilai hash. Akan tetapi algoritma Rabin-Karp menunjukan kemudahan dalam menghitung nilai hash. Penggunaan metode Horner dalam mencari nilai hash membuat penghitungan fungsi hash menjadi lebih efisien, karena penghitungan dilakukan memanfaatkan hasil yang telah ditemukan sebelumnya. Penggunaan nilai hash di dalam *string-matching* juga memberikan efisiensi. Panjang *pattern* tidak terlalu berpengaruh dalam proses pencarian. Penggunaan *string-matching* menggunakan algoritma Rabin-Karp dirasa sesuai untuk

mendeteksi plagiat. Dimana dalam proses deteksi menggunakan *pattern* yang cukup panjang.

1.2 Rumusan Masalah

Adapun rumusan masalah dari penelitian ini antara lain :

a. Bagaimana membangun aplikasi deteksi plagiat berdasarkan s*tring-matching* menggunakan algoritma Rabin-Karp?
b. Apakah *string-matching* dengan menggunakan algoritma Rabin-Karp dapat digunakan untuk mendekteksi plagiat?
c. Seberapa baik *string-matching* dengan menggunakan algoritma Rabin-Karp ketika digunakan untuk mendekteksi plagiat?

1.3 Tujuan

Tujuan penelitian ini adalah sebagai berikut:

a. Membangun aplikasi deteksi berdasarkan s*tring-matching* menggunakan algoritma Rabin-Karp.
b. Mengetahui penerapan *string-matching* dengan menggunakan algoritma Rabin-Karp dapat digunakan untuk mendekteksi plagiat.
c. Mengetahui kinerja *string-matching* dengan menggunakan algoritma Rabin-Karp ketika digunakan untuk mendekteksi plagiat.

1.4 Manfaat

Hasil dari penelitian ini diharapkan :

a. Memudahkan dalam mendeteksi plagiat.
b. Meningkatkan efektifitas dan efesiensi dalam mendeteksi plagiat.

c. Membantu dalam mencegah dan mengurangi plagiat.

1.5 Batasan Masalah

Adapun batasan-batasan masalah dalam penelitian ini adalah:

a. Deteksi Plagiat berdasarkan model *copy-paste*.
b. *Pattern* yang digunakan adalah kalimat.
c. File yang diuji merupakan file dokumen berekstensi doc dan docx.
d. Dokumen menggunakan bahasa yang sama.

BAB II

TINJAUAN PUSTAKA

Terdapat penelitian-penelitian yang terkait dengan aplikasi yang akan dibuat dalam usulan penelitian ini. Penelitan Nugroho (2011) meneliti tentang sistem deteksi plagiarisme dokumen dengan menggunakan algoritma rabin-karp. Penelitian menggunakan konsep similarity menggunakan k-gram. penelitian ini memanfaatkan tingkat kemiripan untuk dijadikan pendukung keputusan apakah dokumen merupakan hasil plagiarisme atau tidak. Penelitian juga memodifikasi algoritma rabin-karp dan membandingkan penggunaan algoritma rabin-karp standart dengan algoritma yang telah dimodifikasi. Sehingga diketahui pengaruh dari modifikasi yang dilakukan. Tahapan yang dimodifikasi adalah pada tahap *hashing* dan *string-matching.*

Penelitian priantara et al (2011) meneliti tentang implementasi deteksi penjiplakan dengan algoritma Winnowing pada dokumen terkelompok. Penelitian ini menambahkan pengelompokan dokumen secara partisi dan mendeteksi plagiat pada tiap-tiap kelompok. Pengelompokan dokumen menggunakan k-means++. Dalam menggunakan k-means++ yang membutuhkan masukan jumlah kelompok yang terbentuk. Untuk masukan jumlah kelompok digunakan algoritma hartigan index untuk mendapat rekomendasi jumlah kelompok yang akan dibentuk. Pendeteksiannya menggunakan algoritma Winnowing. Pada penelitian ini, uji coba dibagi menjadi tiga bagian. Bagian pertama uji coba penentuan kelompok, mengggunakan *Latent Semantic Analisys* (LSA) dan Hartigan index.

LSA menggunakan *Singular Value Decomposition* (SVD) untuk memproses kata-kata yang ada dalam dokumen. Penggunaan SVD dalam penelitian ini memanfaatkan fungsi yang terdapat pada matlab. Untuk mengetahui jumlah kelompok yang seharusnya terbentuk menggunakan Hartigan Index. Bagian kedua uji coba penentuan kelompok untuk dokumen baru. Pada penelitian ini dilakukan pengamatan kesesuaian dokumen baru terhadap hasil kelompoknya. Bagian ketiga uji coba sistem. Pada bagian ini terdapat dua skenario, yaitu dokumen dideteksi pada kelompok dokumen dan dokumen dideteksi dengan kelompok dokumen yang sudah dikelompokan. Tujuan dari uji coba pada bagian ini adalah untuk mengetahui perbedaan waktu antara dua skenario tersebut.

Penelitian Novian et al (2012) meneliti tentang aplikasi pendeteksian plagiat pada karya ilmiah menggunakan algoritma rabin-karp. Penelitian ini menggunakan pemodelan *prototyping*. Alasan menggunakan model prototyping adalah untuk meminimalisir implementasi sistem yang belum sempurna atau belum stabil, serta mempercepat respon terhadap kebutuhan pengguna. Penelitian ini memodifikasi Algoritma Rabin-Karp pada bagian *hashing* dengan tujuan mempercepat proses deteksi. Pengujian pada penelitian menggunakan metode BlackBox, dengan membuat suatu input tertentu dan memperhatikan output yang dihasilkan. Apabila output yang dihasilkan sesuai dengan output yang seharusnya (menurut spesifikasi requirement sistem) maka sistem berhasil lulus pada item pengujian yang diujikan. Data yang digunakan dalam penelitian ini dikelompokan menjadi dua yaitu dokumen yang akan diuji dan kumpulan dokumen yang digunakan sebagai pembanding yang tersimpan dalam *repository*.

Penelitian Suwarjo et al (2012) meneliti tentang bentuk plagiat pada Skripsi mahasiswa Fakultas Ilmu Pendidikan Universitas Negeri Yogyakarta. Penelitian ini menyimpulkan bahwa bentuk plagiat total adalah bentuk plagiat yang paling dominan dilakukan mahasiswa. Dikarena sebagian besar mahasiswa kurang teliti dan maraknya budaya *copy-paste* di kalangan mahasiswa tanpa mempertimbangkan pentingnya mencantumkan sumber kutipan ke dalam skripsi baik di dalam bab maupun di dalam daftar pustaka. Dalam penelitian ini mendifinisikan bentuk plagiat total sebagai tindakan mengacu dan mengutip istilah, kata/kalimat, data/info dari sumber tanpa menyebutkan sumber dalam catatan kutipan dan tanpa menyatakan sumber yang memadai.

Penelitian Zen (2013) meneliti tentang penerapan algoritma string-matching untuk mendeteksi musik plagiat. Algortitma string matching yang digunakan pada penelitian ini adalah Knuts-Morris-Prats. Dalam penerapannya notasi musik diubah menjadi kode untuk memudahkan proses deteksi dan melakukan *string-matching* per baris not.

Penelitian Salmuasih (2013) meneliti tentang sistem deteksi plagiat dengan similarity menggunakan algoritma rabin-karp. Tujuan dari penelitian ini adalah merancang sistem yang mampu mendeteksi kemiripan isi pada dokumen text, yang dimungkinkan kemiripan ini merupakan hasil plagiat. Pedeteksian plagiat tidak memperhatikan adanya penulisan rujukan. Pada penelitian ini sebelum masuk pada deteksi dilakukan *tokenizing*, memecah kalimat menjadi potongan kata dan mengubah menjadi huruf dalam dokumen menjadi huruf kecil, *filtering* menghapus kata yang tidak terdeskriptif, dan *stemming* mengembalikan setiap

kata kepada kata dasar. Dalam proses deteksi kata dipecah dalam potongan karakter sebanyak k menggunakan parsing k-gram. Karakter sepanjang k dicocokan menggunakan algoritma rabin-karp. Hasil dari pencocokan string dihitung menggunakan *Dice's Similarity Coefficient*.

BAB III

METODOLOGI PENELITIAN

3.1 Objek Penelitian

Pada penelitian ini, peneliti akan melakukan penelitian mengenai aplikasi deteksi plagiarisme berdasarkan s*tring-matching* menggunakan algoritma Rabin-Karp. Objek penelitian ini adalah sebagai berikut:

a. *Softcopy* berekstensi doc dan docx tugas akhir atau skripsi dari alumni Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.
b. Bentuk plagiarisme yang digunakan adalah *Copy&Paste plagiarism.* Bentuk ini didefinisikan oleh Gipp dan Meuschke (2011) sebagai bentuk plagiarisme yang menyalin setiap kata tanpa perubahan.

3.2 Sumber Data

Data yang digunakan adalah *softcopy* laporan berekstensi doc dan docx tugas akhir atau skripsi dari alumni Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang yang di dapat dari tenaga administrasi jurusan.

3.3 Instrumen Penelitian

3.3.1 Variabel Bebas

Dalam penelitian ini variabel bebas adalah *softcopy* laporan berekstensi doc dan docx tugas akhir atau skripsi dari alumni Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang yang akan dideteksi.

3.3.2 Variabel Penghubung

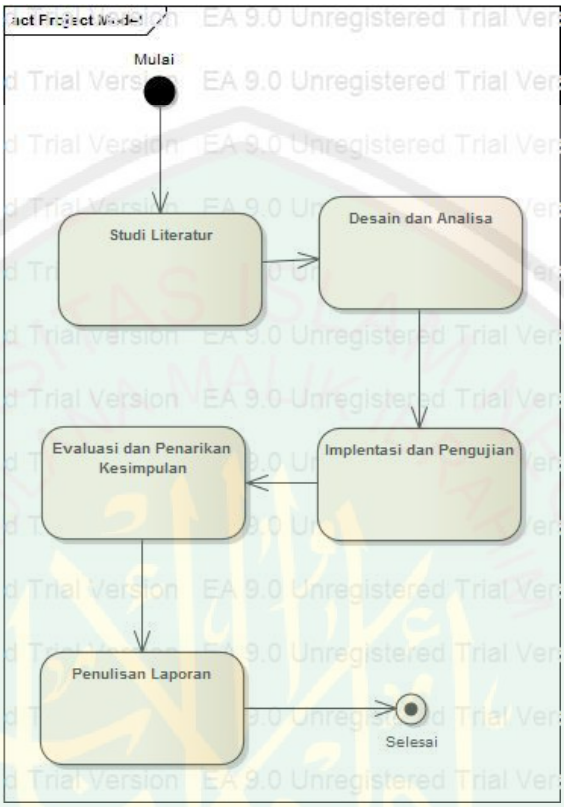
Dalam penelitian ini variabel penghubung adalah *softcopy* laporan berekstensi doc dan docx tugas akhir atau skripsi dari alumni Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang yang tersimpan dalam *repositori* yang berfungsi sebagai referensi.

3.3.3 Variabel Terikat

Dalam Penelitian ini variabel terikat adalah hasil uji coba meliputi :

a. Hasil pendeteksian dokumen.
b. Data waktu dan konsumsi sumber daya komputer yang dibutuhkan untuk melakukan deteksi.

3.4 Prosedur Penelitian



Gambar 1. 1 Prosedur penelitian

3.4.1 Studi Literatur

Mempelajari teori-teori yang akan digunakan dan berhubungan dengan penelitian aplikasi deteksi berdasarkan s*tring-matching* menggunakan algoritma Rabin-Karp.

3.4.2 Desain dan Analisa

Pada tahap ini dilakukan desain dan analisa untuk membuat aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp. Disini dibagi menjadi beberapa tahap.

Tahap-tahap itu adalah sebagai berikut:

a) Gambaran Umum.

Menjelaskan gambaran umum dari proses desain dan analisa pembuatan aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp.

b) Tampilan.

Melakukan desain dan analisa pembuatan tampilan aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp agar sesuai dengan kebutuhan dan mudah digunakan oleh pengguna.

c) Proses.

Melakukan desain dan analisa proses-proses yang terjadi di aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp.

d) Data.

Melakukan desain dan analisa megenai cara data dimanfaatkan aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp.

e) Kebutuhan Fungsional.

Melakukan desain dan analisa kebutuhan fungsional aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp.

f) Domain Model.

Melakukan desain dan analisa domain model dari aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp. Domain model merupakan desain awal pemodelan class.

g) Use Case.

Melakukan desain dan analisa use case dari aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp. Disini juga dilakukan *robustness analysis* dan desain diagram *sequence* yang nanti digunakan dalam pemodelan class.

h) Pemodelan Class.

Melakukan desain dan analisa pemodelan class dari aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp. Hasil dari tahap ini adalah diagram class dari aplikasi ini.

3.4.3 Implementasi dan Pengujian

Pada tahap ini dilakukan pengubahan dari diagram class hasil desain dan analisa menjadi kode program dan melakukan pengujian terhadap aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp.

Parameter pengujian adalah sebagai berikut:

a) Kemampuan mendeteksi aplikasi.

b) Waktu yang digunakan oleh aplikasi untuk melakukan deteksi.

c) Kebutuhan sumber daya komputer yang digunakan aplikasi untuk melakukan deteksi.

3.4.4 Evaluasi dan Penarikan Kesimpulan

Pada tahap ini dilakukan evaluasi dan penarikan kesimpulan dari penelitian ini. Dan akan dilakukan perbaikan bila diperlukan.

3.4.5 Penulisan laporan

Sistematika penulisan skripsi ini adalah sebagai berikut:

BAB I Pendahuluan

Bab ini menjelaskan tentang latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat.

BAB II Tinjauan Pustaka

Bab ini menjelaskan tentang hasil penelitian-penelitian terdahulu yang terkait dan digunakan sebagai referensi dalam penelitian ini.

BAB III Metodologi Penelitian

Bab ini menjelaskan tentang metodologi penelitian digunakan.

BAB IV Hasil dan Pembahasan

Bab ini menjelaskan tentang hasil dan pembahasan penelitian ini

BAB V Penutup

Bab ini menjelaskan tentang kesimpulan dari penelitian dan saran untuk penelitian selanjutnya.

BAB IV

HASIL DAN PEMBAHASAN

4.1 Studi Literatur

Contoh penghitungan algoritma Rabin-Karp yang dirangkum dari buku karangan Sedgewick dan Wayne (2011). Mencari *pattern* dengan kode karakter 2,6,5,3,5 pada text 3, 1, 4, 1, 5, 9, 2, 6, 5 ,3 ,5 ,8 ,9 ,9 ,3. Dengan nilai Radix sama dengan 10 dan Q sebagai operasi modulo sama dengan 997. Pada contoh ini digunakan operasi modulo untuk mempekecil hasil fungsi hash yang dihasilkan. Dengan Rumus sebagai berikut:

$$x_{i+1} = (x_i - t_i R^{M-1})R + t_{i+M}$$

Keterangan:

x_{i+1} =Nilai hash String setelah karakter ke i ditambahkan dan dilakukan operasi modulo

$\left(x_i - t_i \ R^{M-1}\right)$ = nilai hash String sebelum karakter ke i ditambahkan.

R = Radix (pada contoh ini bernilai 10).

t_{i+M} = Kode Karakter yang ditambahkan.

Hasil dari setiap penghitungan akan langsung dilakukan operasi modulo. Pertama mencari hash *pattern*. Ditunjukkan dengan tabel di bawah ini

Tabel 4. 1 Mencari Hash Pattern

i	Kode karakter	Rumus	Hasil
0	2	(0*10+2)%997	2
1	2,6	(2*10+6)%997	6
2	2,6,5	(26*10+5)%997	265
3	2,6,5,3	(265*10+3)%997	659
4	2,6,5,3,5	(659*10+5)%997	613

Dalam Algoritma Rabin-Karp perpindahan karakter pada text tidak lagi menggunakan pengambilan string, namun dengan menghapus digit pertama dan menambahkan digit terakhir dengan karakter.

Berikut adalah contoh perpindahan karakter dalam algoritma Rabit-Karp tanpa menggunakan operasi modulo.

Tabel 4. 2 Algoritma Pergeseran Karakter pada Rabin-Karp

No	Langkah-Langkah	Nilai
1	Nilai Hash awal	41592
2	Menghapus digit pertama dengan menambahkan nilai awal dengan nilai negatif kode karakter dikali 10 pangkat panjang string.	41592 -40000 1592
3	Mengalikan hasil operasi langkah dua dengan 10.	15920
4	Tambahkan dengan kode karakter yang baru	15920 00006 15926

Dalam penggunanan operasi modulo tidak terlalu terdapat perbedaan. Terdapat variabel tambahan yang berguna mengembalikan nilai setelah terjadi operasi modulo. Berikut adalah contoh langkah untuk mencari variabel dengan panjang *pattern* sama dengan 5 dengan rumus sebagai berikut:

$$RM_{i+1} = (RM_t R)\%Q$$

Keterangan:
RM_{t+1} = Nilai yang digunakan untuk menghapus digit awal dan menambah digit akhir.
RM_t = Nilai RM pada index sebelumnya.
R = Radix (pada contoh ini adalah 10).
Q = Bilangan prima yang digunakan untuk operasi modulo (pada contoh ini adalah 997)

Tabel 4. 3 Mencari RM

i	Rumus	Nilai
0	Menggunakan angka 1 sebagai awal nilai	1
1	(1*10)%997	10
2	(10*10)%997	100
3	(100*10)%997	3
4	(3*10)%997	30

Dari Hasil penghitungan diatas diketahui nilai RM sama dengan 30. Selanjutnya melakukan pencarian *pattern* di dalam text. Dimana diketahui nilai hash dari pattern adalah 613.

Tabel 4. 4 Penghitungan Rabin-Karp

i	Kode Karakter	Rumus	Nilai
0	3	(0*10+3)%997	3
1	3,1	(3*10+1)%997	31
2	3,1,4	(31*10+4)%997	314
3	3,1,4,1	(314*10+1)%997	150
4	3,1,4,1,5	(150*10+5)%997	508
5	1,4,1,5,9	((508+3*(997-30))*10+9)%997	201
6	4,1,5,9,2	((201+1*(997-30))*10+2)%997	715
7	1,5,9,2,6	((715+4*(997-30))*10+6)%997	971
8	5,9,2,6,5	((971+1*(997-30))*10+5)%997	442
9	9,2,6,5,3	((442+5*(997-30))*10+3)%997	929
10	2,6,5,3,5	((929+9*(997-30))*10+5)%997	613

Dari hasil penghitungan yang ditunjukkan tabel 4.4 ditemukan pada pencarian ke 10. Untuk mengetahui posisi karakter dalam text digunakan rumus sebagai berikut:

$$r = (i - m) + 1$$

Keterangan :
r = Posisi awal karakter dalam text.
i = Index pencarian.
m = Panjang *pattern.*

4.2 Desain dan Analisa

4.2.1 Gambaran Umum

Pada desain dan analisa aplikasi menggunakan beberapa tahapan. Tahapan-tahapan tersebut anatar lain adalah sebagai berikut:

- Tampilan

Pada tahap ini dilakukan desain dan analisa menggenai tampilan antar muka dari aplikasi Pada tahap akan di dapat desain input maupun output dari aplikasi ini. Hasil dari tahap ini nantinya akan digunakan dalam tahap use case.

- Data

Pada tahap ini dilakukan desain dan analisa menggenai bentuk dan format data yang digunakan.

- Proses

Pada tahap ini dilakukan desain dan analisa menggenai proses-proses atau aktifitas-aktifitas yang terjadi aplikasi Hasil dari tahap ini akan digunakan pada tahap selanjutnya.

- Analisa kebutuhan fungsional

Pada tahap ini menganalisa apa saja yang harus bisa dilakukan oleh aplikasi. Tahap merupakan kelanjutan dari tahap desain dan analisis proses.

- Domain Model

Pada tahap ini dilakukan desain dan analisa domain model dari aplikasi. Domain model merupakan rancangan awal dari tahap pemodelan class.

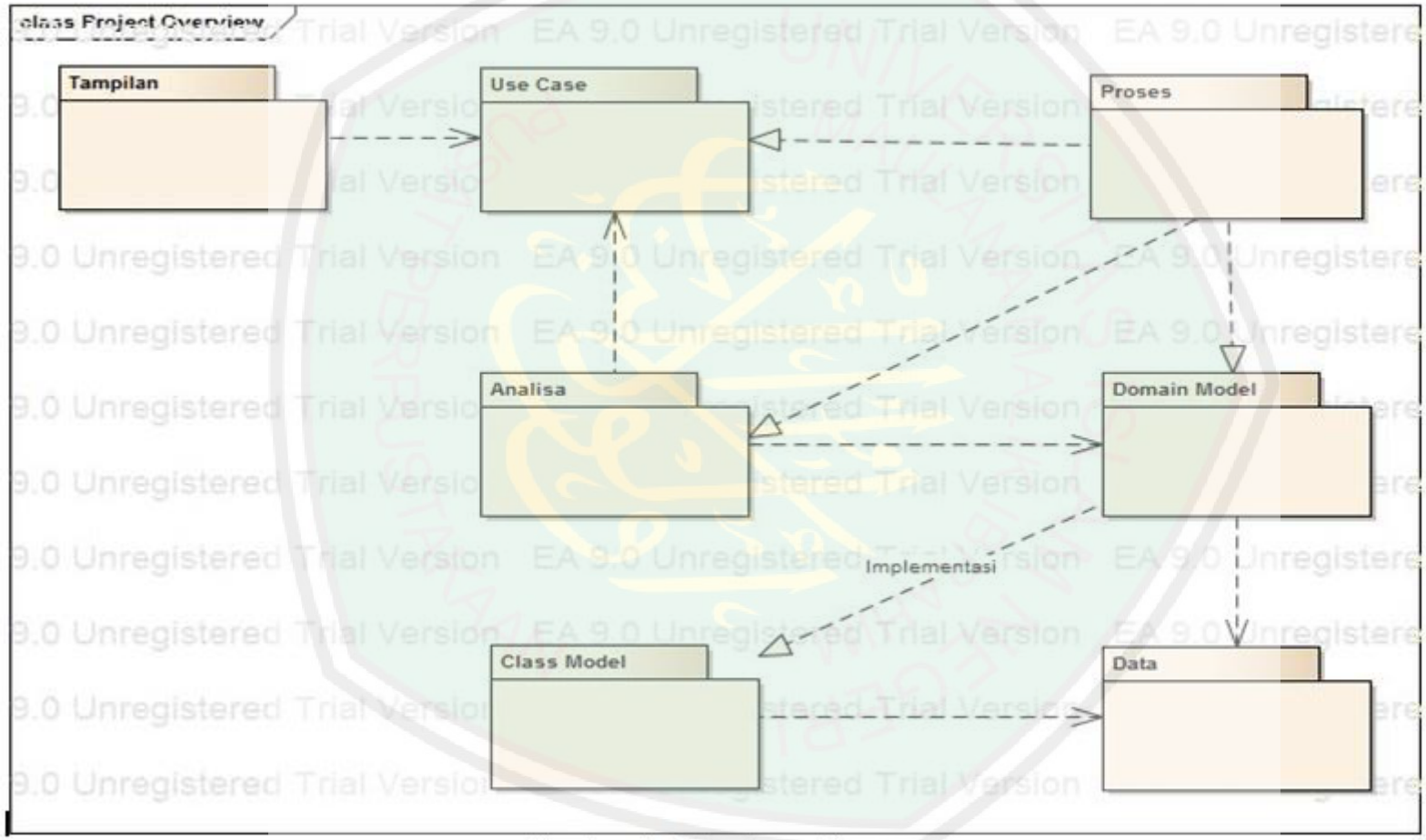
- Use Case

Pada tahap ini dilakukan desain dan analisa use case berdasarkan tahap-tahap sebelumnya. Pada tahap juga dilakukan *robustness analysis* dan pembuatan *sueqence diagram* untuk melengkapi domain model.

- Pemodelan Class

Pada tahap ini dilakukan desain dan analisa dalam pemodelan class aplikasi berdasarkan pada tahap-tahap sebelumnya.

Diagram gambaran umum ditunjukkan gambar 4.1



Gambar 4. 1 Gambaran Umum

4.2.2 Tampilan

Desain tampilan dari aplikasi ditunjukkan pada gambar 4.2:

Gambar 4. 2 Desain Tampilan

Desain tampilan dari aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp terdiri dari :

- Satu *text field* untuk menampilkan lokasi file yang dipilih untuk diuji.
- Tiga *button* dimana satu tombol digunakan untuk membuka jendela pilihan file, satu tombol yang lain digunakan untuk memulai proses deteksi dan satu tombol digunakan untuk memuat ulang list dokumen referensi.

- Satu *table* untuk menampilkan list dokumen referensi.
- Tiga *text area* untuk menampilkan isi dari file diuji, hasil dan referensi.
- Empat *panel* untuk memudahkan mengatur posisi masing-masing *UI Control.*

Selain jendela utama yang dijelaskan sebelumnya terdapat 2 jendela yaitu:

- Jendela pilihan file. Berfungsi untuk memilih folder letak file yang akan diuji, dimana file yang akan dipilih telah dibatasi hanya berekstensi doc dan docx.
- Jendela informasi. Memberikan informasi yang nantinya digunakan oleh pengguna untuk mengevaluasi kinerja dari aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp.

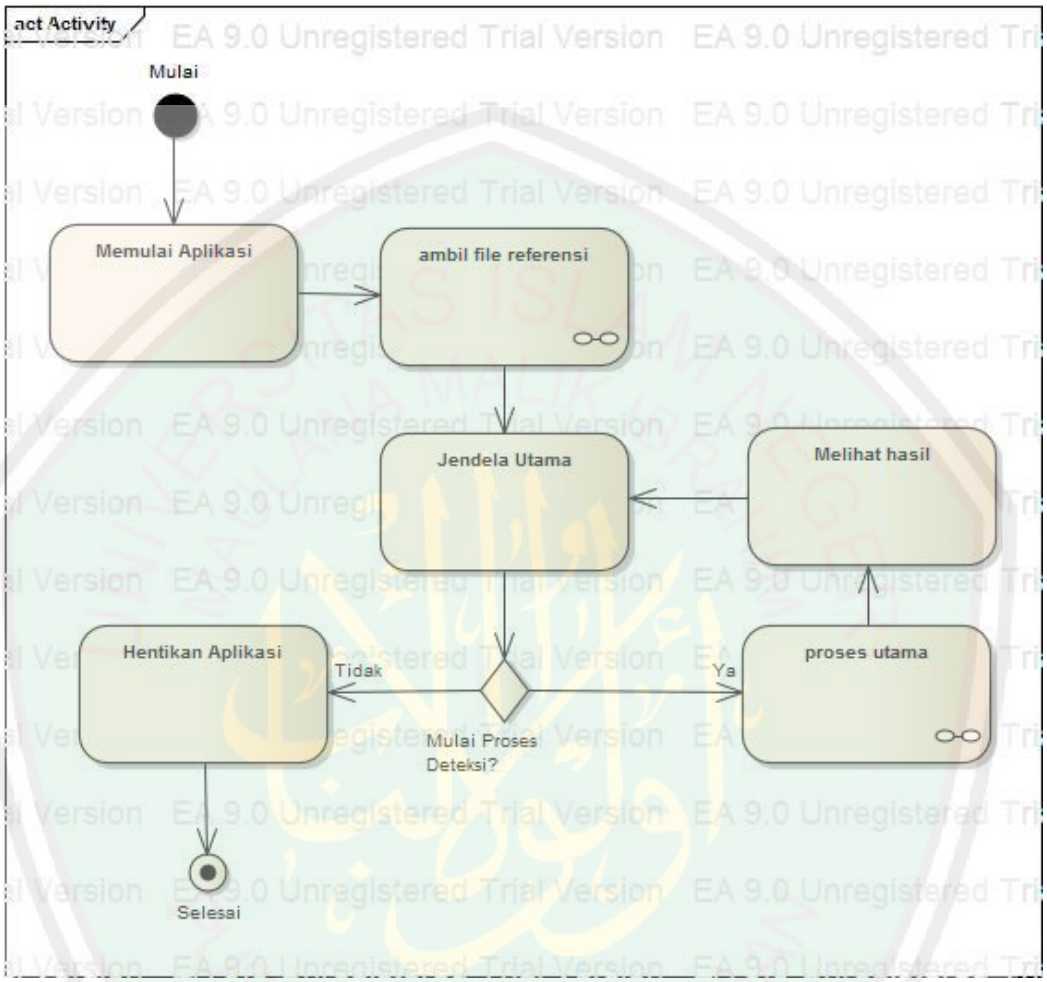
Kedua jendela tersebut menggunakan fungsi *default* bahasa pemrograman yang digunakan dalam pembuatan aplikasi.

4.2.3 Proses

Terdapat beberapa proses yang terjadi pada aplikasi ini. Pengguna menjalankan aplikasi. Sebelum aplikasi menampilkan jendela utama, terlebih dahulu aplikasi melakukan pengecekan root folder aplikasi, bila root folder tidak ditemukan aplikasi akan membuat root folder. Pada saat aplikasi menjalankan proses menampilkan jendela utama, aplikasi juga mengambil isi dokumen referensi yang nantinya digunakan sebagai text dalam proses pencarian.

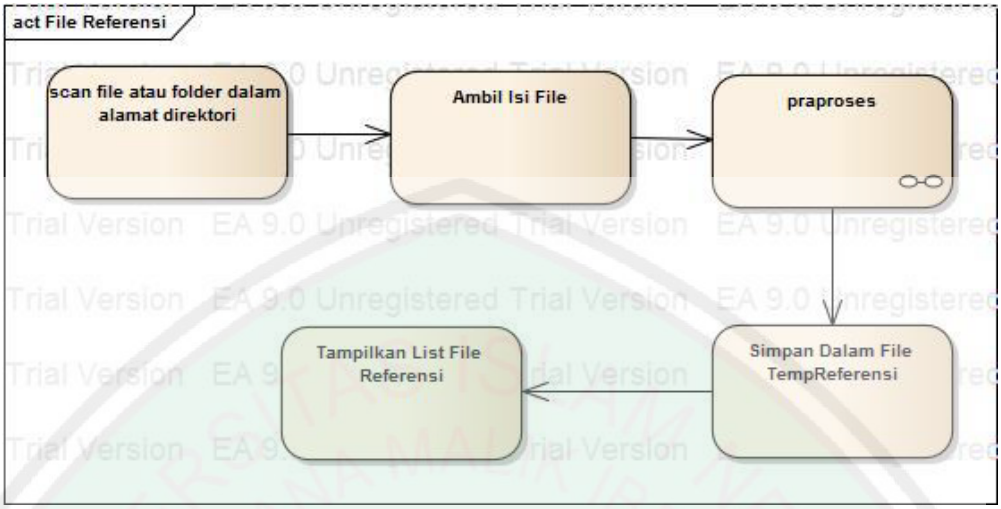
Pada jendela utama terdapat pilihan bagi pengguna untuk melakukan deteksi atau menghentikan aplikasi. Bila user memilih untuk melakukan deteksi, maka akan proses selanjutnya adalah proses utama dari aplikasi dan selanjutnya user melihat hasil deteksi.

Proses-proses yang terjadi secara umum ditunjukkan gambar 4.3:



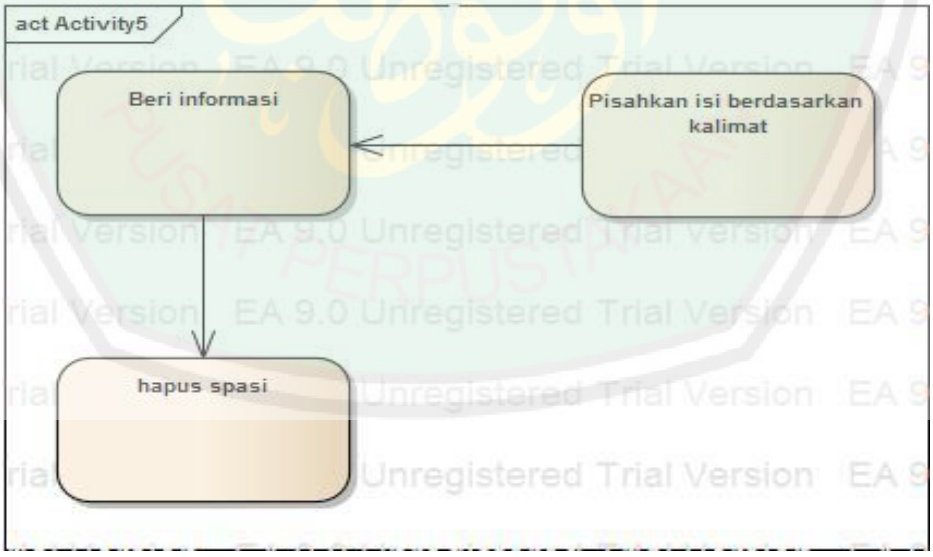
Gambar 4. 3 Diagram akitifitas aplikasi secara umum.

Di dalam proses yang terjadi terdapat sub-sub proses. Pada proses ambil dokumen referensi terdapat proses *scanning* terhadap folder dokumen referensi. Setelah proses *scanning* selesai proses selanjutnya adalah mengambil text dari file-file yang berekstensi doc dan docx dalam folder referensi. Isi file tersebut mengalami praproses, sebelum disimpan dalam file temporari. Setelah proses penyimpanan selesai list dari file temporari ditampilkan dalam jendela utama. Proses pengambilan dokumen referensi dari folder penyimpanan ditunjukkan dalam gambar 4.4:



Gambar 4. 4 Proses ambil dokumen referensi

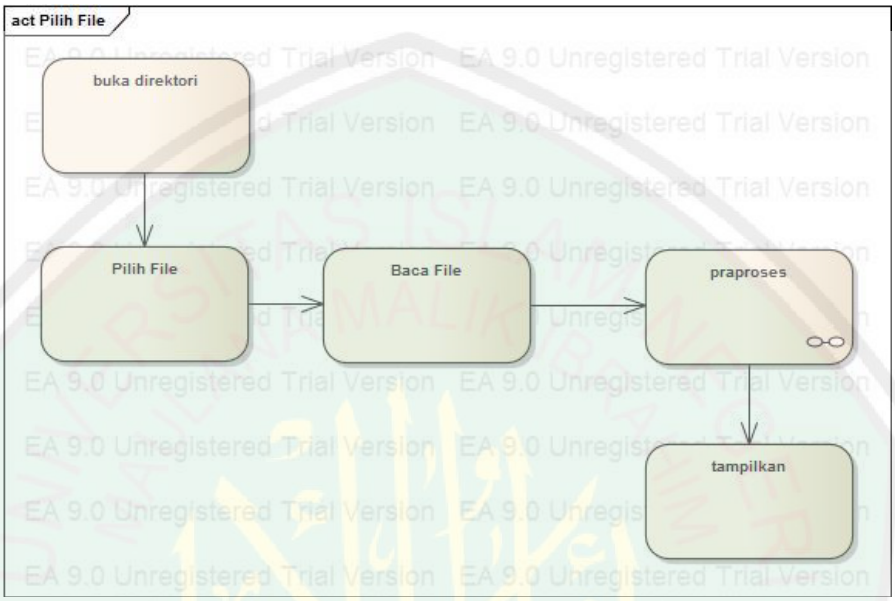
Adapun praproses pada proses ambil dokumen referensi memiliki beberapa proses yaitu memisahkan isi file menjadi kalimat-kalimat, selanjutnya setiap kalimat diberi informasi dan penghapusan spasi. Proses-proses atau aktivitas-aktivitas yang terjadi pada praproses ditunjukkan pada gambar 4.5:



Gambar 4. 5 Praproses

Setelah pengambilan dokumen referensi selesai. Pengguna berada pada jendela utama. Proses selanjutnya adalah proses utama. Pada proses utama dibagi menjadi dua bagian yaitu pemilihan file dan deteksi.

Proses yang terjadi pada pemilihan file ditunjukkan gambar 4.6 sebagai berikut:

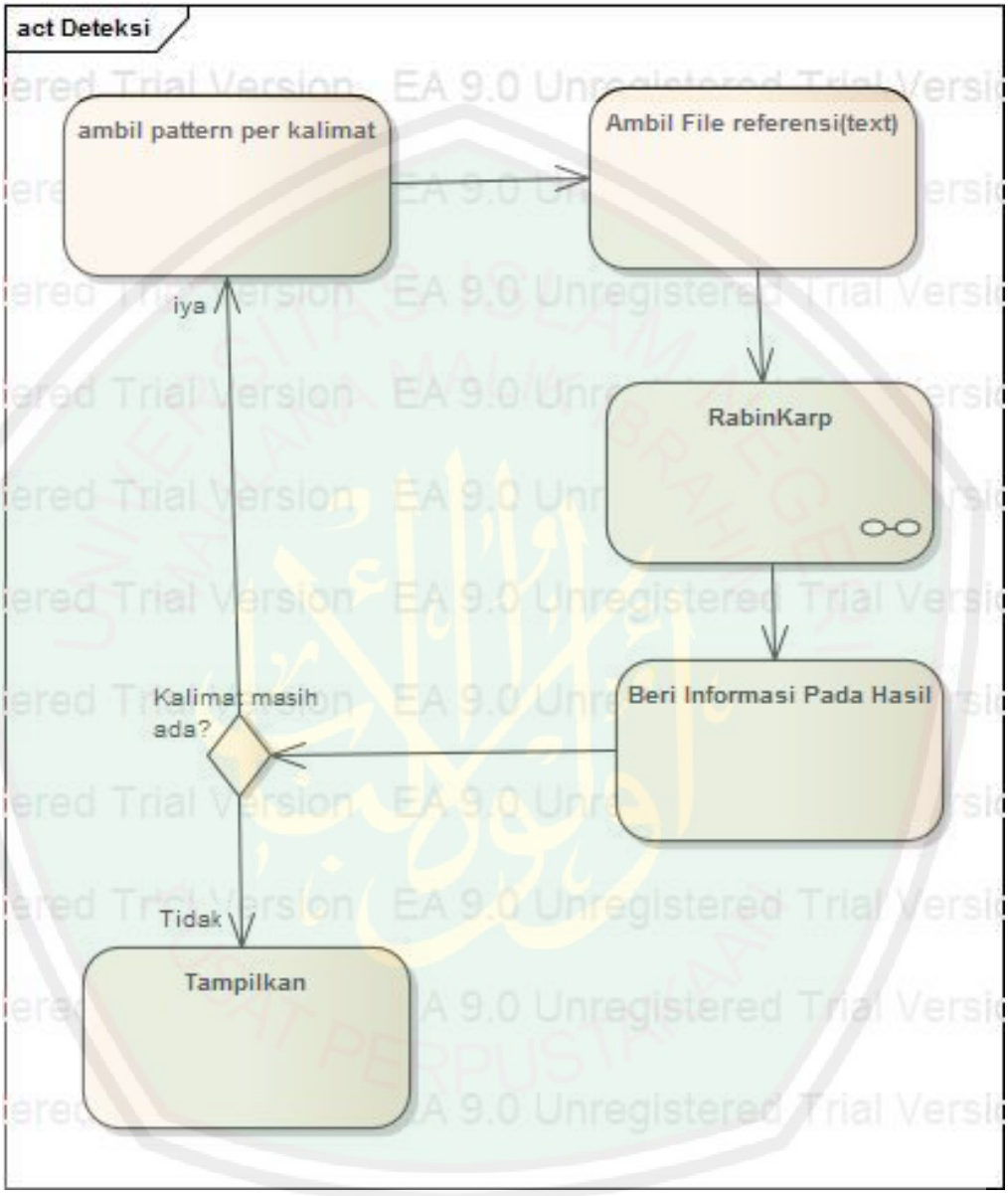


Gambar 4. 6 Memilih file diuji

Membuka direktori dan memilih file. File yang terpilih akan dibaca dan mengalami praproses. Pada praproses ini sama dengan praproses pada proses pengambilan dokumen referensi. Setelah praproses selesai ditampilkan di jendela utama.

Proses selanjutnya adalah proses deteksi. Pada proses deteksi terdapat beberapa proses. Proses pertama mengambil kalimat-kalimat dari *pattern* yang ditentukan pada proses sebelumnya. Lalu mengambil isi semua dokumen referensi dan dijadikan satu sebagai text. Dan dilakukan proses pencarian dengan menggunakan algoritma Rabin-Karp, dimana kalimat-kalimat dari *pattern* tadi dicari di dalam text. Bila selesai hasil akan disimpan. Proses deteksi ini diulang hingga semua kalimat-kalimat pada *pattern* selesai dideteksi. Setelah proses deteksi selesai hasil deteksi ditampilkan kepada pengguna.

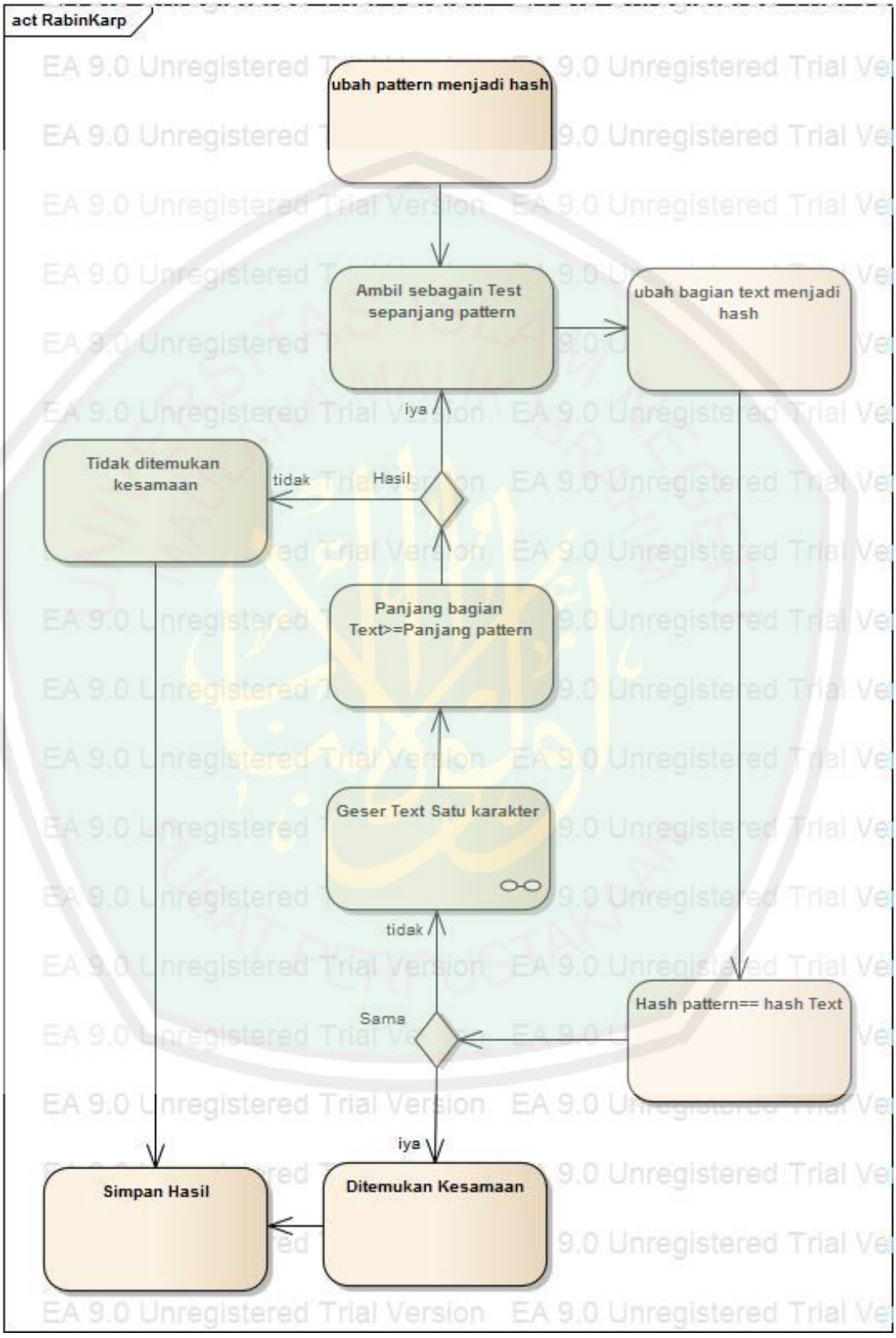
Proses deteksi ditunjukkan pada gambar 4.7 sebagai berikut:



Gambar 4. 7 Deteksi

Pada proses pencarian menggunakan algoritma Rabin-Karp terdapat beberapa proses. Yang pertama adalah mengubah *pattern* menjadi nilai hash. Selanjutnya mengambil bagian dari text sepanjang *pattern.* Dari bagian text yang diambil diubah menjadi nilai hash dan membandingkan nilai hash *pattern* dengan nilai hash bagian text. Bila nilai hash *pattern* dan nilai hash bagian text bernilai

sama, simpan hasil *pattern* terdapat pada text. Bila nilainya tidak sama, text digeser satu karakter ke kanan, lalu membandingkan panjang *pattern* dengan panjang text yang baru. Bila *pattern* lebih panjang dari text, simpan hasil *pattern* tidak ditemukan pada text. Dan proses pencarian menggunakan algoritma Rabin-Karp dimulai dari awal menggunakan *pattern* baru. Bila panjang text lebih panjang dari panjang *pattern.* Proses kembali pada mengambil sebagian text menggunakan text yang baru. Proses pencarian menggunakan algoritma Rabin-Karp ditunjukkan sebagai berikut:

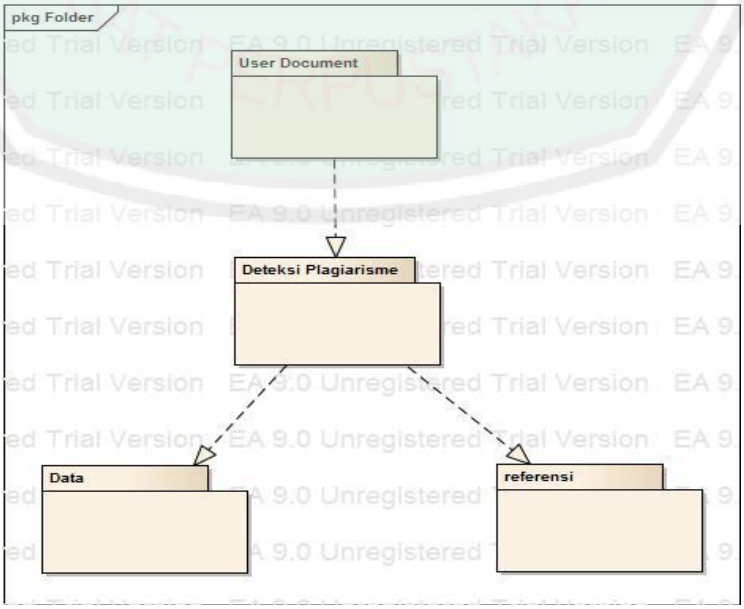


Gambar 4. 8 Rabin Karp

4.2.4 Data

Data yang digunakan adalah *softcopy* berekstensi doc dan docx laporan tugas akhir atau skripsi dari alumni Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang yang di dapat dari tenaga administrasi jurusan. Kumpulan *softcopy* laporan tugas akhir atau skripsi dikelompokan berdasarkan nomor induk mahasiswa, dan dikumpulkan dalam folder sesuai dengan nomor induk mahasiswa. Folder-folder tersebut dikumpulkan dalam folder Data.

Folder Data berada pada folder Deteksi Plagiarisme yang secara *default* berada pada root folder aplikasi. Pada folder Deteksi Plagiarisme selain terdapat folder data juga terdapat folder referensi. Folder referensi ini berisi file temporari dari Data. Struktur folder dari data yang digunakan oleh aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp. Struktur folder digambarkan sebagai berikut :



Gambar 4. 9 Struktur Folder

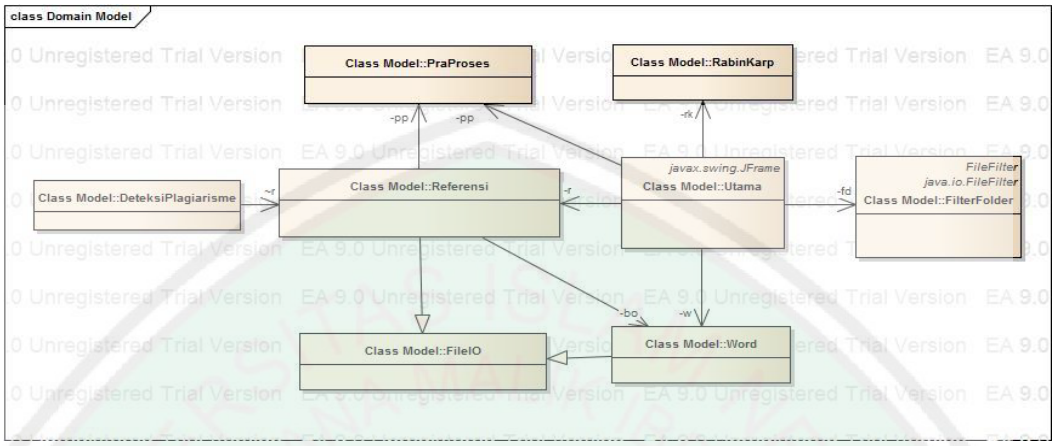
4.2.5 Analisa Kebutuhan Fungsional

Berdasarkan rumusan masalah, tujuan, manfaat, batasan masalah yang telah dijelaskan di bab pertama dan dari desain analisa tahap sebelum didapat kebutuhan fungsional dari aplikasi ini sebagai berikut:

a) Membaca file dokumen berkekstensi doc dan docx.
b) Memiliki beberapa file dokumen untuk digunakan sebagai dokumen referensi.
c) Menampilkan daftar file dokumen referensi.
d) Menampilkan isi dokumen baik dokumen yang diuji maupun dokumen referensi dengan memberi informasi yang memudahkan bagi pengguna.
e) Memisahkan isi file dokumen menjadi kalimat untuk dijadikan *pattern*.
f) Memisahkan dokumen referensi untuk dijadikan text.
g) Melakukan deteksi plagiarisme model *copy-paste*.
h) Mendeteksi dengan konsep *string-matching* dengan menggunakan algoritma Rabin-Karp.
i) Dalam penggunaan algoritma Rabin-Karp. Kalimat pada file dokumen yang diinputkan pengguna menjadi *pattern* dan isi dari dokumen referensi menjadi text.
j) Menampilkan hasil deteksi dengan informasi yang memudahkan pengguna.

4.2.6 Domain Mode

Tahap selanjutnya adalah desain dan analisa domain mode. Pada tahap ini dibuat domain mode yang merupakan desain awal dalam dari pemodelan class. berdasarkan tahap-tahap desain dan analisa sebelumnya didapat desain domain model ditunjukkan pada gambar 4.10:



Gambar 4. 10 Domain Mode

- DeteksiPlagiarisme, berfungsi sebagai main objek. Dijalankan pertama kali pada saat aplikasi. Objek ini berasosiasi dengan objek Referensi, untuk menjalankan proses ambil dokumen referensi.
- FileIO, berfungsi untuk membaca dan menulis file temporari.
- PraProses, berfungsi untuk memecah isi dokumen referensi dan dokumen yang akan diuji menjadi kalimat dan dan memberi informasi yang akan memudahkan pengguna.
- RabinKarp, berfungsi untuk proses pencarian kesamaan kalimat.
- filterFolder, merupakan turunan objek java.io.FileFilter berfungsi untuk memberikan filter agar pada jendela pilihan file hanya terlihat folder.
- Word, merupakan turunan dari objek fileIO. Berfungsi untuk membaca file dokumen berekstensi doc dan docx.
- Referensi, merupakan turunan dari objek fileIO. Berasosiasi dengan objek dalam pembacaan file dokumen referensi, dan berasosiasi dengan objek

PraProses untuk memberi informasi pada isi dokumen referensi sebelum disimpan dalam file temporari.

- Utama, merupakan turunan dari javax.swing.Jframe, berfungsi sebagai objek yang menjadi jendela utama aplikasi. Juga berfungsi sebagai penghubung antara aplikasi dengan pengguna, serta menampilkan list dokumen referensi, isi dokumen referensi, isi file yang diuji dan hasil deteksi. Berasosiasi dengan objek filter folder pada saat menampilkan jendela pilihan file. Berasosiasi dengan objek word untuk membaca file terpilih. Berasosiasi dengan objek referensi untuk mengambil isi dokumen referensi yang nantinya digunakan sebagai text. Berasosiasi dengan objek PraProses untuk memisahkan *pattern* menjadi kalimat. Dan berasosiasi dengan objek RabinKarp untuk proses pencarian.

4.2.7 Use Case

Pada tahap ini membuat diagram use case berdasarkan desain dan analisa yang telah dibuat sebelumnya. Diagram uses case berisi interaksi-interaksi pengguna dan fitur-fitur dari aplikasi. Use case juga berisi apa yang bisa dikerjakan oleh aplikasi dan apa yang diharapakan oleh pengguna. Setelah menganalisa dari tahap-tahap sebelumnya didapat desain Diagram use case sebagai berikut:

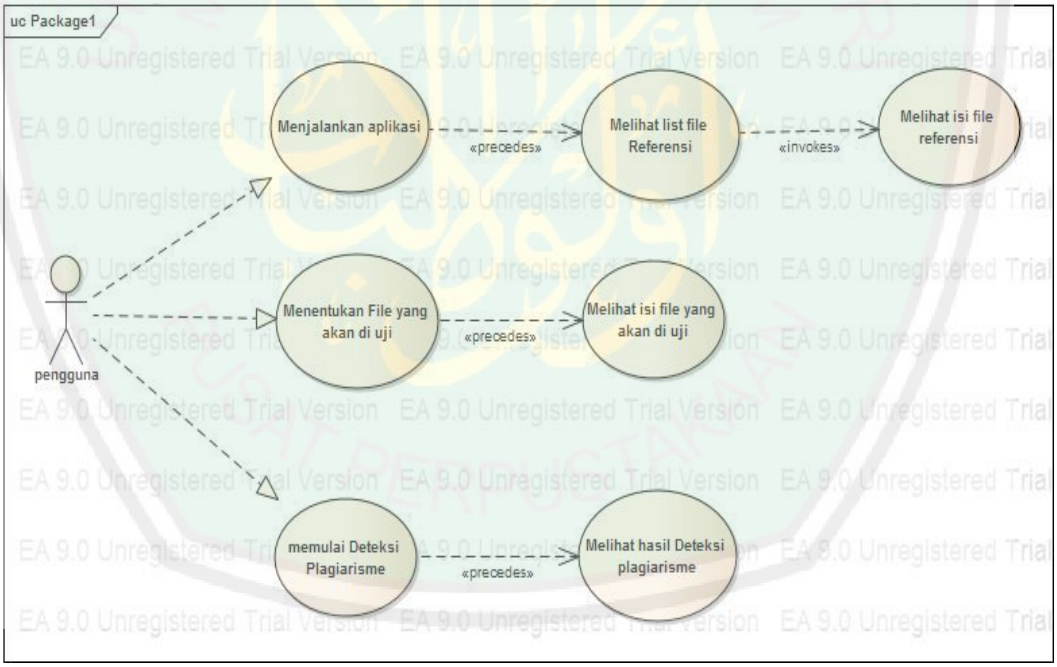
a. Pengguna mulai menjalankan aplikasi.
b. Pengguna melihat list dokumen referensi.
c. Pengguna melihat isi dokumen referensi.
d. Penggunaa menentukan file yang akan diuji.

e. Pengguna melihat isi file yang akan diuji.

f. Pengguna memulai deteksi.

g. Pengguna melihat hasil deteksi plagiarisme.

Pada saat pengguna menjalankan aplikasi secara otomatis penggunan melihat list dokumen referensi. Sedangkan untuk melihat dokumen referensi merupakan pilihan dari pengguna. Pada saat menentukan file yang akan di uji pengguna secara otomatis akan melihat isi file yang akan diuji. Pada saat pengguna memulai deteksi secara otomatis pengguna melihat hasil deteksi plagiarisme. Diagram use case ditunjukkan gambar 4.11 sebagai berikut:

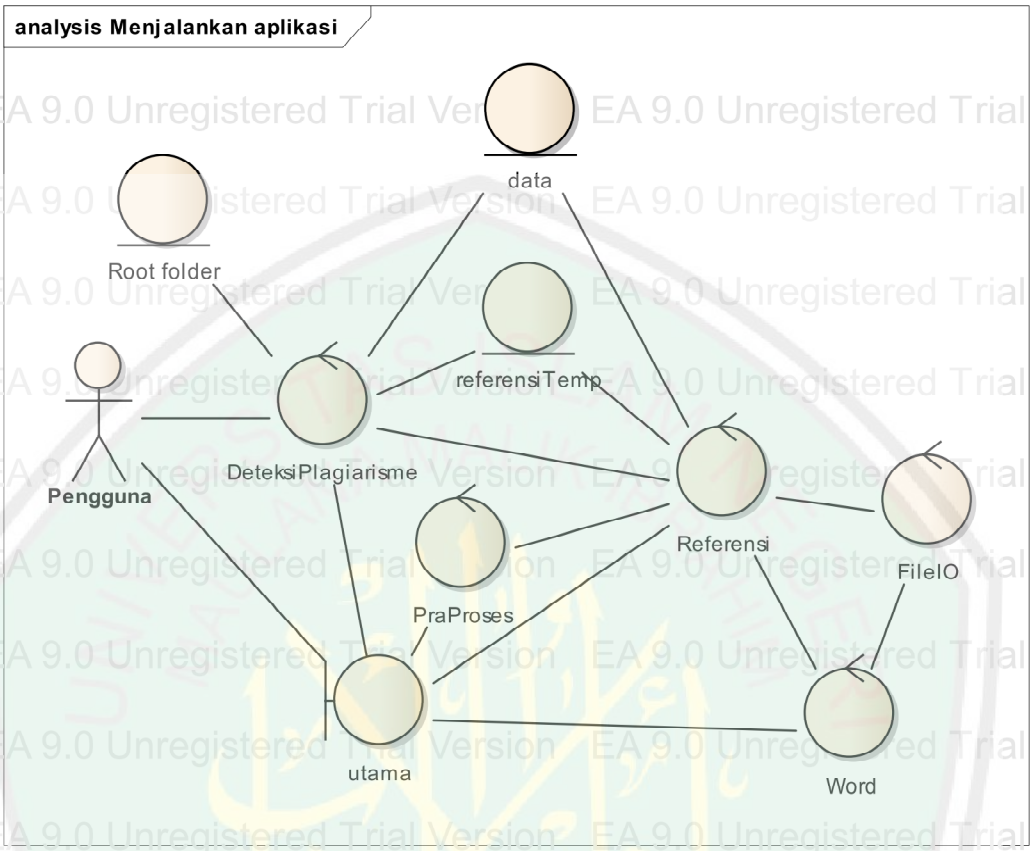


Gambar 4. 11 Diagram use case

Setelah membuat diagram use case, dilakukan *robustness analysis. robustness analysis* dilakukan untuk menganalisa objek-objek yang digunakan dalam pada use case.

Pada use case menjalankan aplikasi Objek DeteksiPlagiarisme sebagai main objek melakukan pengecekan root folder aplikasi, bila tidak ditemukan objek DeteksiPlagiarisme akan membuat root folder. Objek DeteksiPlagiarisme berasosiasi dengan objek referensi mengubah file-file dokumen yang merupakan kumpulan dokumen referensi dalam folder data di root folder aplikasi menjadi file-file temporari yang disimpan dalam folder referensi pada root folder aplikasi. Dalam merubah kumpulan dokumen referensi menjadi file-file temporari, objek referensi yang merupakan turunan dari objek FileIO berasosiasi dengan objek Word untuk membaca file berekstensi doc atau doc.

Objek referensi berasosiasi dengan objek PraProses untuk memisahkan hasil pembacaan menjadi kalimat-kalimat. Kalimat-kalimat tersebut diberi informasi lalu disimpan dalam file temporari. Setelah file-file temporari terbentuk, objek DeteksiPlagiarisme berasosiasi dengan objek utama untuk menampilkan jendela utama. Diagram *robutness analisys* use case menjalankan aplikasi ditunjukkan gambar 4.12:

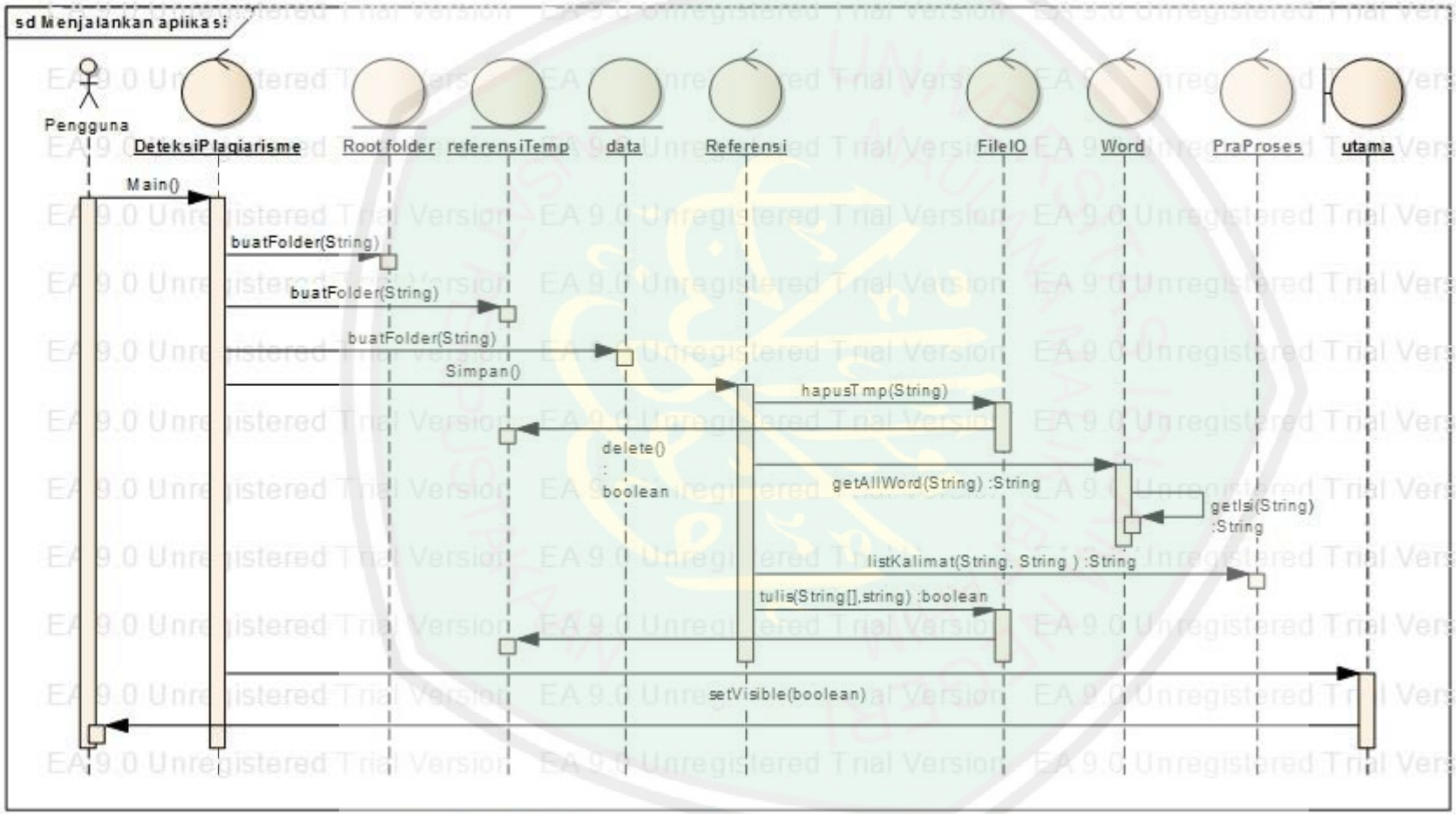


Gambar 4. 12 *Robustness analysis* menjalankan aplikasi

Selanjutnya adalah desain dan analisa interaksi objek-objek pada use case menjalankan aplikasi. Pengguna menjalankan aplikasi dengan memanggil operasi main pada objek utama. Pada aplikasi ini objek utama adalah deteksiPlagiarisme. Objek deteksiPlagiarisme melakukan operasi buatFolder yang menghasilkan folder rootFolder, data dan referensiTemp. Pada operasi buatFolder, sebelum membuat folder, operasi buatFolder melakukan pengecekan. Bila folder-folder yang akan dibuat telah ada, proses pembuatan folder dilewati.

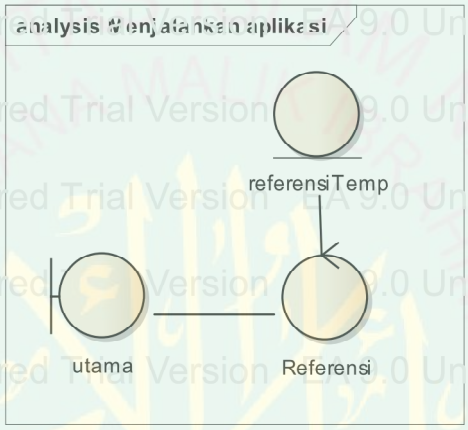
Setelah proses pengecekan dan pembuatan folder selesai, objek deteksiPlagiarisme memanggil operasi Simpan pada objek referensi untuk mengubah file-file data yang berekstensi doc dan docx menjadi file-file temporari.

Sebelum mengubah file-file data menjadi file-file temporari. Terlebih dahulu file-file temporari yang sebelumnya dihapus terlebih dahulu. Dalam penghapusan file temporari objek referensi memanggil operasi hapusTmp pada objek fileIO yang merupakan objek *parent* dari objek referensi. Setelah file temporari selesai dihapus, objek referensi memanggil operasi getAllWord pada objek Word. Objek Word melakukan operasi getIsi untuk membaca file data. Selanjutnya objek referensi memanggil operasi isiKalimat untuk memisahkan kalimat dan memberi informasi pada kalimat. Selanjutnya objek referensi memanggil operasi tulis untuk menulis kalimat-kalimat yang telah dihasilkan oleh operasi-operasi pada objek fileIO. Setelah dokumen referensi berhasil dibuat, objek deteksiPlagiarisme menampilkan jendela utama dengan memanggil operasi setVisible yang merupakan operasi bawaan dari bahasa pemrograman yang digunakan dalam penelitian ini. Gambar sequence diagram ditunjukan gambar 4.13:



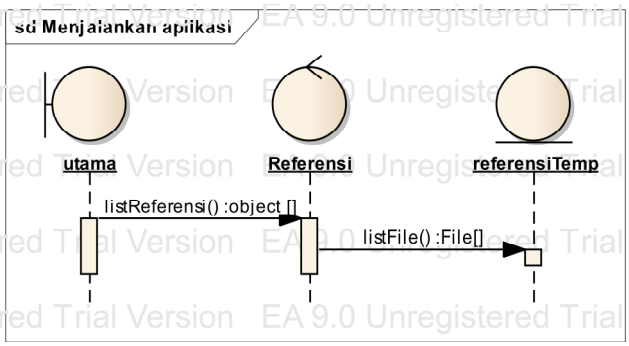
Gambar 4. 13 Sequence diagram menjalankan aplikasi

Pada use case melihat list dokumen referensi, objek utama mengambil list nama file temporari yang tersimpan dalam folder fileTemp untuk ditampilkan pada jendela utama. Dalam proses pengambilannya objek ini berasosiasi dengan objek Referensi. Diagram *robustness analisys* ditunjukkan pada gambar 4.14 dibawah ini.



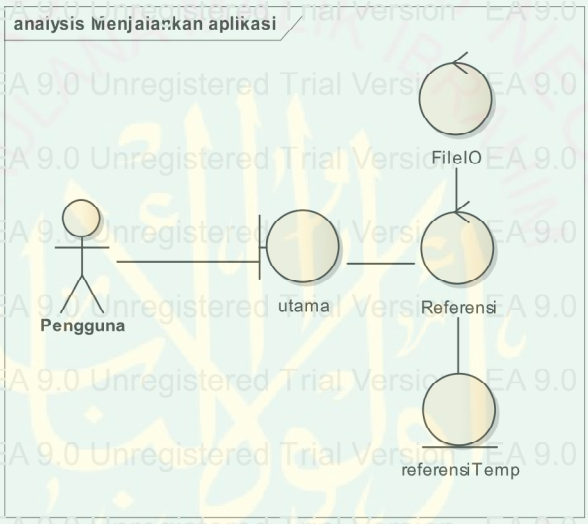
Gambar 4. 14 *Robustness analisys* melihat list dokumen referensi

Objek utama memanggil operasi listReferensi yang berada pada objek Referensi. Dalam operasi listReferensi, objek referensi operasi listFile pada objek File. Objek File merupakan objek bawaan dari bahasa pemrograman yang digunakan untuk membangun aplikasi deteksi plagiarisme berdasarkan s*tring-matching* menggunakan algoritma Rabin-Karp. Diagram *seuqence* sebagai berikut:



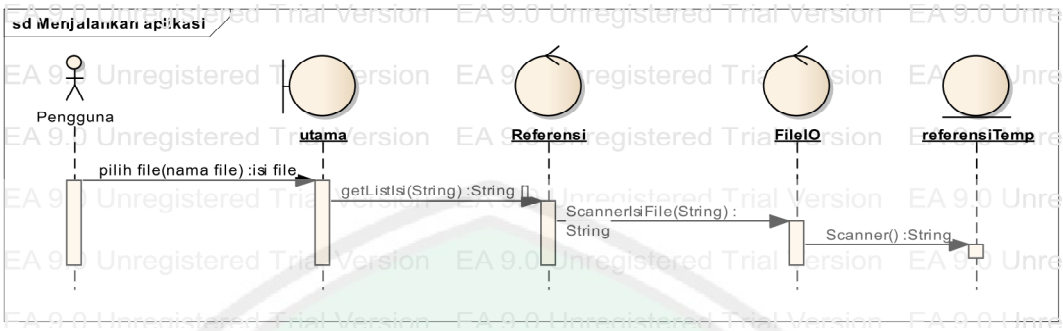
Gambar 4. 15 Diagram *seuqence* melihat list dokumen referensi

Pengguna dapat melihat isi dokumen referensi dengan memilih salah satu file yang ada dalam list dokumen referensi pada tabel list referensi pada jendela utama. Objek utama berasosiasi dengan objek referensi mengambil isi file yang dipilih pada referensiTmp dan menampilkanya pada text area isi dokumen referensi pada jendela utama. Objek referensi merupakan objek turunan dari objek FileIO. Diagram *robustness analisys* ditunjukkan pada gambar 4.16 dibawah ini.



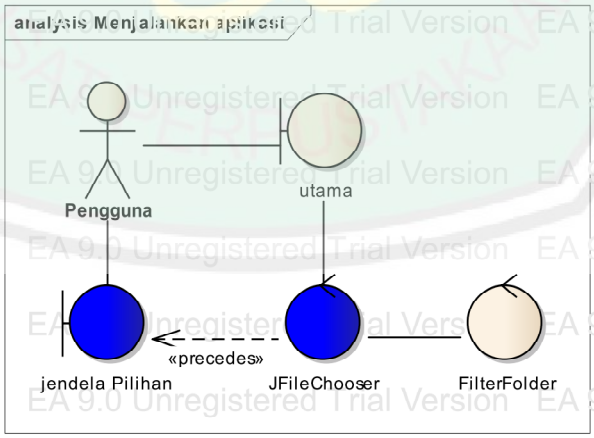
Gambar 4. 16 *Robustness analisys* melihat isi dokumen referensi

Pengguna memlih dokumen referensi yang akan ditampilkan. Objek utama memanggil operasi getListIsi pada objek referensi. Operasi getListIsi berada pada objek *parent* dari dokumen referensi yaitu objek FileIO. Operasi getListIsi memanggil operasi bawaan bahasa pemrograman scanner untuk membaca file di referensiTemp. Diagram *seuqence* ditunjukkan pada gambar 4.17:



Gambar 4. 17 Diagram *seuqence* melihat isi dokumen referensi

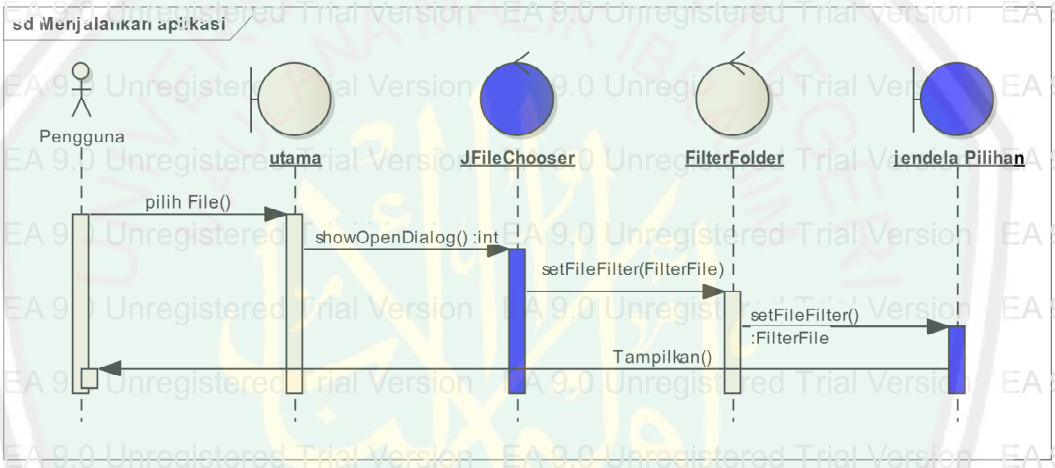
Pada menentukan file yang akan diuji, objek Utama berasosisasi dengan objek JFileChooser untuk menampilkan jendela pilihan file pada user. Objek JFileChooser merupakan objek bawaan dari bahasa prmrograman yang digunakan. Untuk membatasi pilihan pengguna objek JFileChooser diasosiaikan dengan objek folderFilter. Objek utama juga berasosiasi dengan objek folder filter yang bertujuan membatasi pilihan pengguna menjadi folder lokasi file yang akan diuji. Lokasi dari file yang diuji ditampilkan dalam text file di jendela utama. Diagram *robustness analisys* ditunjukkan pada gambar 4.18 dibawah ini.



Gambar 4. 18 *Robustness analisys* menentukan file yang akan diuji

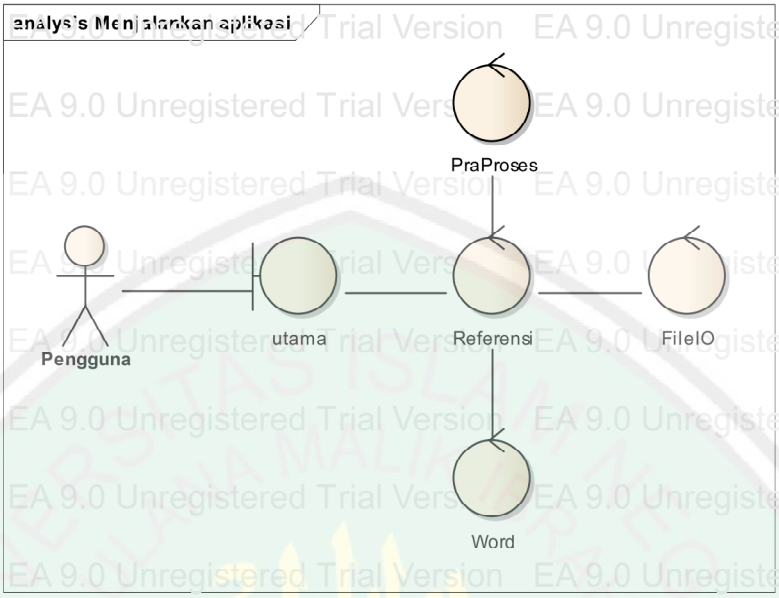
Objek utama mememanggil operasi showOpenDialog pada objek JFileChooser. Dalam operasi showOpenDialog berjalan ditentukan filter file yang dapat dipilih oleh pengguna, sebelum jendela pilihan file ditampilkan. Filter file

memanfaatkan objek filterFolder yang merupakan turunan dari objek FileFilter dari paket filechooser dan merupakaan implementasi dari objek FileFilter dari paket io. Kedua paket diatas merupakan paket bawaan dari bahasa pemrograman yang digunakan dalam pembuatan aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp. Diagram *seuqence* ditunjukkan pada gambar 4.19.



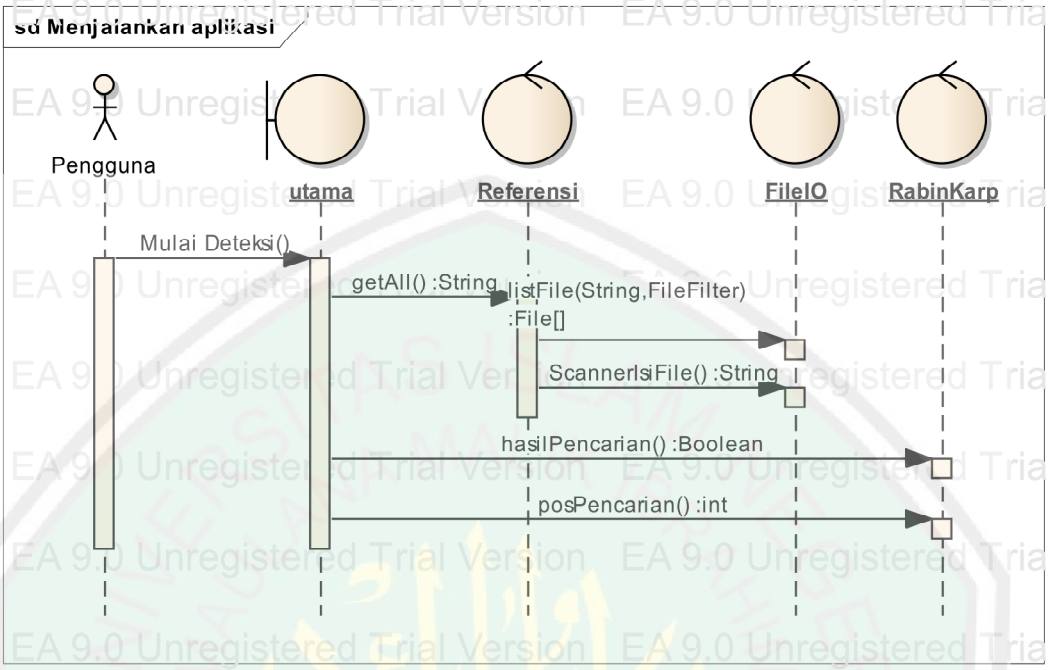
Gambar 4. 19 Diagram *seuqence* menentukan file yang akan diuji

Pada melihat isi file yang akan diuji. Objek utama berasosiasi dengan objek referensi untuk membaca file yang akan diuji. Untuk membaca file yang akan diuji, objek referensi yang merupakan turunan dari objek FileIO berasosiasi dengan objek Word untuk membaca file berekstensi doc atau doc. Objek referensi berasosiasi dengan objek PraProses untuk memisahkan hasil pembacaan menjadi kalimat-kalimat. Kalimat-kalimat tersebut diberi informasi lalu disimpan dalam variabel dan ditampilkan dalam isi file diuji di jendela utama. Diagram *robustness analisys* ditunjukkan pada gambar 4.20 dibawah ini:



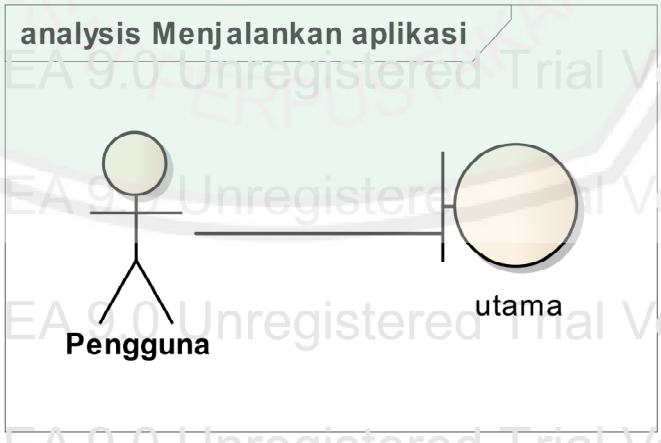
Gambar 4. 20 *Robustness analisys* melihat isi file

Pengguna memulai proses deteksi. Objek utama memanggil operasi getAll() di objek Referensi untuk mengambil isi dokumen referensi. Di dalam operasi getAll, objek Referensi memanggil operasi listFile dan operasi ScannerIsiFile dari objek FileIO yang merupakan *parent* Objek dari objek Referensi. Hasil dari operasi getAll dijadikan text untuk digunakan pada objek RabinKarp. Sedang *pattern* menggunakan variable hasil use case memilih file diuji. Objek utama memanggil operasi hasilPencarian dan posPencarian Objek RabinKarp. Diagram *seuqence* ditunjukkan pada gambar 4.21.



Gambar 4. 21 Diagram *seuqence* melihat isi file yang akan diuji

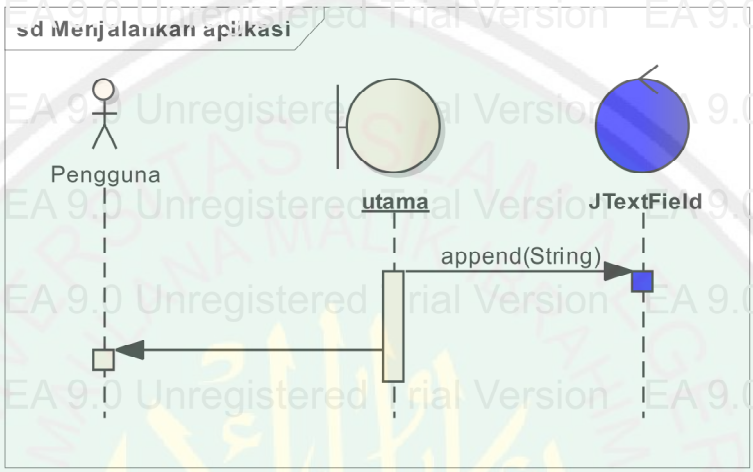
Pad use case melihat hasil deteksi, hasil dari pembacaan berupa posisi *pattern* didalam text ditampilkan oleh objek utama pada text area hasil dengan ditambah informasi-informasi yang memudahkan pengguna. Diagram *robustness analisys* ditunjukkan pada gambar 4.22.



Gambar 4. 22 *Robustness analisys* melihat hasil deteksi

Objek utama memanggil operasi append pada objek JTextField untuk menampilkan hasil pada text field hasil. Objek JTextField merupakan objek

bawaan dari bahasa pemrograman yang digunakan untuk membangun aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp. Diagram *seuqence* ditunjukkan pada gambar 4.23.

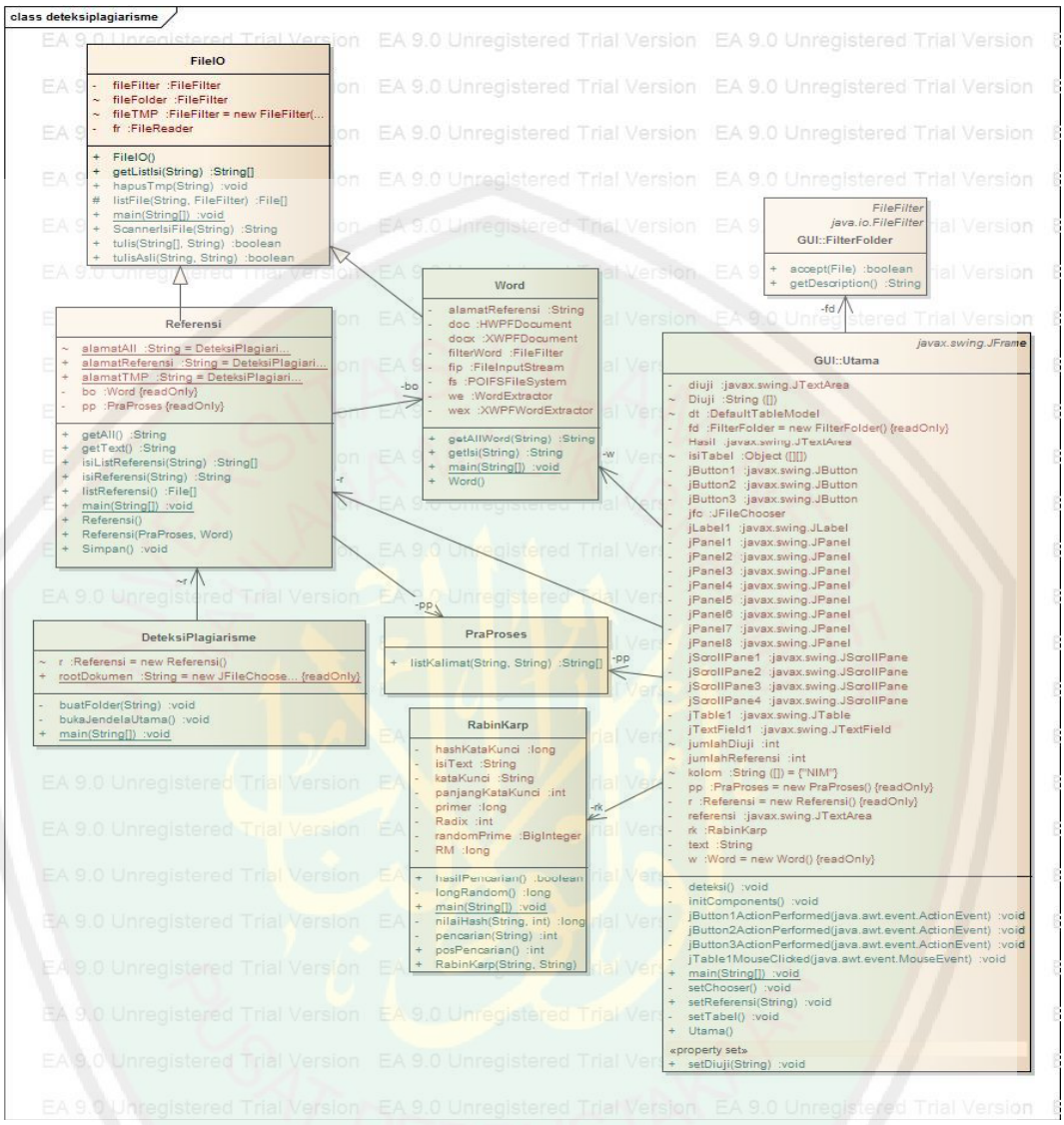


Gambar 4. 23 Diagram *seuqence* melihat hasil deteksi

Operasi-operasi yang sudah didefinisikan pada tahap ini belum bersifat final. Hal ini dikarenakan operasi-operasi yang dihasilkan hanya menunjukan operasi-operasi yang dimanfaatkan atau berinteraksi dengan objek lain. Sehingga dimungkinkan penambahan operasi-operasi pada tahap pemodelan class.

4.2.8 Pemodelan Class

Pada tahap ini dilakukan desain dan analisa dalam pemodelan class dari aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp berdasarkan pada tahapan-tahapan yang sebelumnya. Pemodelan class berdasarkan domain model yang disempurnakan pada tahap-tahap sebelumnya. Pemodelan class ditunjukkan pada gambar 4.24 sebagai berikut:



Gambar 4. 24 Diagram class

Pemodelan Class ini belum bersifat final. Pada tahap selanjutnya pemodelan diagram class diimplementasikan ke dalam kode program. Dan pada tahap ini akan dilakukan perubahan bila diperlukan.

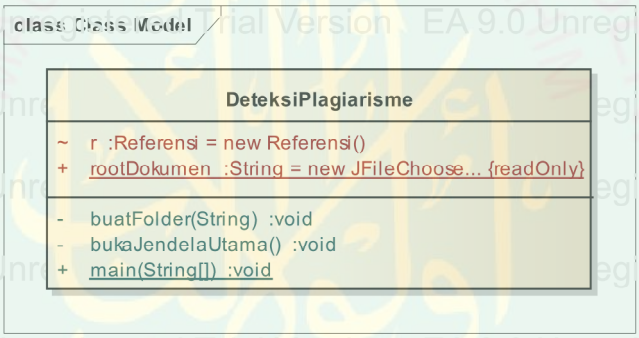
4.3 Implementasi dan Pengujian

4.3.1 Impelmentasi

Diagram class yang telah dibuat sebelumnya dikonversi menjadi kode progam. Berikut pembahasan implementasi masing-masing diagram class:

4.1.1.1 DeteksiPlagiarisme

Merupakan main class dari aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp. Berikut diagram class DeteksiPlagiarisme:



Gambar 4. 25 Diagram class DeteksiPlagiarisme

Pada class ini terdapat 2 attribut yaitu r bertipe Referensi dan rootDokumen bertipe String. Sedangkan operasi pada class ini adalah :

- Buatfoder(String)

Operasi ini bersifat *private* dan berfungsi untuk melakukan pengecekan terhadap root folder aplikasi dan bila tidak ditemukan akan membuat folder pada folder document default user. Operasi ini berparameter String yang merupakan alamat folder yang akan dibuat.

```
      private void buatFolder(String alamat) {
             File folder = new File(alamat);
             if(folder.exists()){
             }else{
                 try{
                         folder.mkdir();
                     }
                     catch(SecurityException se){
                         System.err.println("gagal buat
      folder");
                     }
             }
     }
```

Gambar 4. 26 Source code operation buatFolder

- bukaJendelaUtama()

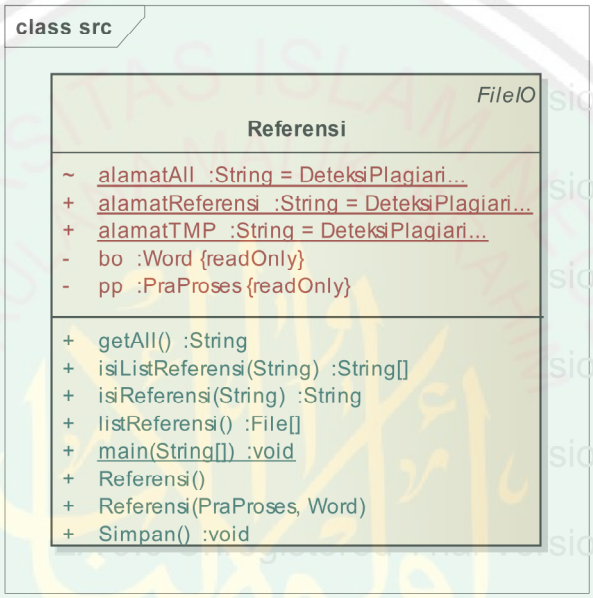
Operasi ini bersifat *private*. Merupakan operasi utama dari class DeteksiPlagiat. Pada operasi ini terdapat perintah untuk pengecekan dan pembuatan root folder aplikasi, mengubah dokumen referensi dari doc atau docx menjadi file temporari dan membuka jendela utama.

```
private void bukaJendelaUtama(){
            long start=Calendar.getInstance().getTimeInMillis();
            buatFolder(rootDokumen);
            buatFolder(Referensi.alamatReferensi);
            buatFolder(Referensi.alamatTMP);
            buatFolder(Referensi.alamatAll);
            r.Simpan();
            new Utama().setVisible(true);
            long end=Calendar.getInstance().getTimeInMillis();
            long l=(end-start)/1000;
            if(l>60){
              l=l/60;
            JOptionPane.showMessageDialog(null,    "Waktu    yang
dibutuhkan untuk menjalankan aplikasi adalah "+l+" menit" );
       }else{
           JOptionPane.showMessageDialog(null,    "Waktu    yang
dibutuhkan untuk menjalankan aplikasi adalah "+l+" detik" );
       }
}
```

Gambar 4. 27 Source code operation bukaJendelaUtama

4.1.1.2 Referensi

Class Referensi merupakan turunan dari kelas FileIO. Pada class ini terdapat operasi-operasi yang mengolah dokumen referensi. Berikut diagram class Referensi:



Gambar 4. 28 Diagram class Referensi

Pada class Referensi terdapat dua *constructor*. Kedua *constructor* ini berfungsi sebagai inisiasi attribute yang terdapat pada class Referensi.

```
public Referensi(){
        super();
        bo=new Word();
        pp=new PraProses();
    }
public Referensi(PraProses pp, Word bo) {
        super();
        this.pp = pp;
        this.bo = bo;
    }
```

Gambar 4. 29 *Constructor* class Referensi

Operasi-operasi yang terdapat pada class referensi adalah sebagai berikut:

- isiListReferensi(String)

Operasi bersifat public dan berfungsi untuk membaca isi dokumen referensi. Dalam membaca isi dokumen referensi memanfaatkan operasi di class lain yaitu operasi getAllword dari class Word dan operasi listKalimat dari class PraProses. Operasi ini berparameter String yang merupakan alamat folder tempat menyimpan dokumen referensi.

```
public String [] isiListReferensi(String Alamat){
        File f=new File(Alamat);
        String s=bo.getAllWord(Alamat);
        String l[]=pp.listKalimat(s,f.getName());
        return l;
    }
```

Gambar 4. 30 Source code operasi isiListReferensi

- isiReferensi(String)

Operasi ini memiliki fungsi yang sama dengan operasi isiListReferensi. Perbedaan nya adalah nilai kembalian. Operasi ini mengembalikan nilai String.

```
public String isiReferensi(String Alamat){
        File f=new File(Alamat);
        String s=bo.getAllWord(Alamat);
        String l[]=pp.listKalimat(s,f.getName());
        String kirim="";
        for (String l1 : l) {
            if(kirim.equals("")) {
                kirim=l[0];
            }else {
                kirim = kirim + l1;
                }
        }
        return kirim;
    }
```

Gambar 4. 31 Source code operasi isi Referensi

- listReferensi()

Operasi ini memiliki fungsi untuk mengambil list temporari dokumen referensi. Tidak memiliki parameter, dan mengembalikan nilai array file.

```
public File[] listReferensi(){
        return new File(alamatReferensi).listFiles(fileFolder);
    }
```

Gambar 4.32 Source code operasi listReferensi

- Simpan()

Operasi ini memiliki fungsi untuk membaca dokumen referensi yang berextensi doc atau docx. Hasil pembacaan akan disimpan dalam file temporari yang nantinya akan digunakan untuk proses deteksi. Operasi ini tidak memiliki parameter dan tidak mengembalikan nilai.

```
      public void Simpan(){
             hapusTmp(alamatTMP);
             File f[]= listReferensi();
             String s[];
              for (File f1 : f) {
                  s = isiListReferensi(f1.getAbsolutePath());
                  tulis(s, alamatTMP+File.separator +
      f1.getName());
                  System.out.println(f1.getName() + "
      tersimpan");
              }
    }
```

Gambar 4. 33 Source code operasi Simpan

- getAll()String

Operasi ini berfungsi untuk mengambil semua isi file temporari referensi. Operasi ini tidak memiliki parameter dan mengembalikan nilai String.

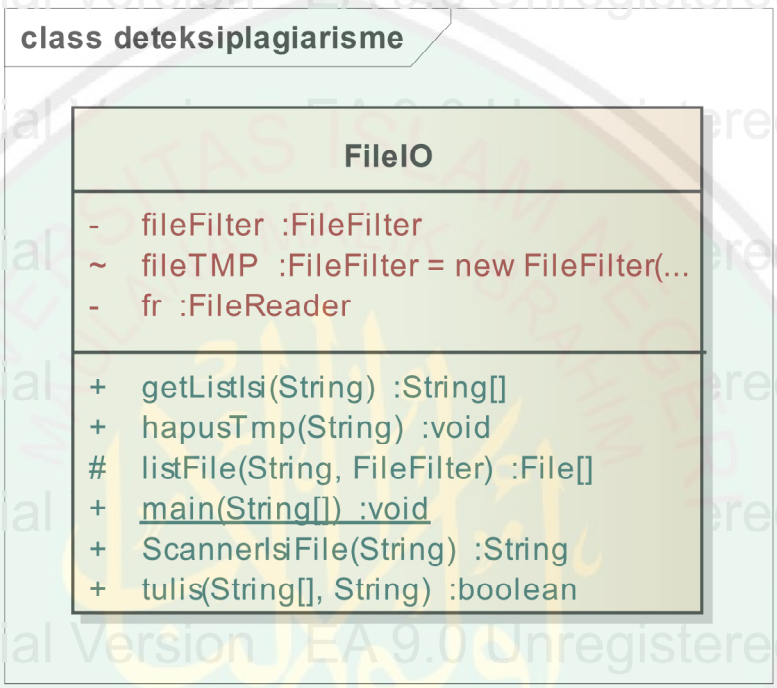
```
public String getAll() {
       fileFilter = new FileFilter() {
            @Override
            public boolean accept(File file) {
                return file.getName().endsWith(".tmp");
                }
       };
       File listFile[]=listFile(alamatTMP,fileFilter);
       String
kirim=ScannerIsiFile(listFile[0].getAbsolutePath());
       if(listFile.length>1){
            for(int i=1;i<listFile.length;i++){

kirim=kirim+ScannerIsiFile(listFile[i].getAbsolutePath());
            }
        }
    return kirim;
    }
```

Gambar 4. 34 Source code operasi getAll

4.1.1.3 FileIO

Pada class terdapat operai-operasi berhubungan dengan *input* dan *output* *File System*. Berikut diagram class FileIO:



Gambar 4. 35 Diagram class FileIO

Operasi-operasi yang terdapat pada class FileIO adalah sebagai berikut:

- getListIsi(String)

Operasi ini bersifat *public* dan berfungsi untuk mengubah nilai kembalian dari ScannerIsiFile(String) yang bertipe data String menjadi array String. Operasi ini berparameter String dan mengembalikan nilai array String.

```
    public String [] getListIsi(String alamat){
                String
    kirim[]=ScannerIsiFile(alamat).split("<~>");
                return kirim;
}
```

Gambar 4. 36 Source code operation getListIsi

- hapusTmp(String)

Operasi bersifat *public* dan berfungsi untuk menghapus file temporari sebelum ditulis kembali. Operasi ini berparamater String yang merupakan alamat folder dimana file temporari tersimpan.

```
public void hapusTmp(String alamat){
        fileFilter = new FileFilter() {
            @Override
            public boolean accept(File file) {
                return file.getName().endsWith(".tmp");
                }
            };
        File[] l=listFile(alamat, fileFilter);
        if(l.length!=0){
            for (File l1 : l) {
                if(l1.isFile()){
                    if(l1.delete()){
                    }else{
                    }
                }
            }
        }
    }
```

Gambar 4. 37 Source code operation hapusTmp

- listFile(String,FileFilter)

Operasi bersifat *protected* dan berfungsi untuk menampilkan list file dalam suatu folder. Operasi ini berparameter String yang berisi alamat folder, dan objek FileFilter yang berfungsi sebagi filter agar file yang terambil adalah file yang dibutuhkan.

```
protected File[] listFile(String alamat, FileFilter fileFilter)
{
          return new File(alamat).listFiles(fileFilter);
    }
```

Gambar 4. 38 Source code operation listFile

- ScannerIsiFile(String)

Operasi bersifat *public* dan berfungsi untuk membaca isi file. Operasi ini berparameter String yang merupakan alamat file yang akan dibaca. Operasi ini mengembalikan isi file yang telah dibaca.

```
    public String ScannerIsiFile(String string) {
            String kirim = "";
               try {
                   fr=new FileReader(string);
                   } catch (FileNotFoundException ex) {

    Logger.getLogger(FileIO.class.getName()).log(Level.SEVERE,
    null, ex);
                   }
               Scanner scan=new Scanner(fr);
               while (scan.hasNextLine()) {
                   String s = scan.nextLine();
                   if("".equals(string)){
                   }else{
                   kirim+=s;
                   }
               }
           return kirim;
}
```

Gambar 4. 39 Source code operation ScannerIsiFile

- tulis(String[],String)

Operasi bersifat *public* dan berfungsi untuk menulis file di dalam sebuah folder. Operasi ini berparameter array string sebagai isi yang akan ditulis ke file dan String yang merupakan alamat file yang akan ditulis. Operasi ini mengembalikan nilai boolean.

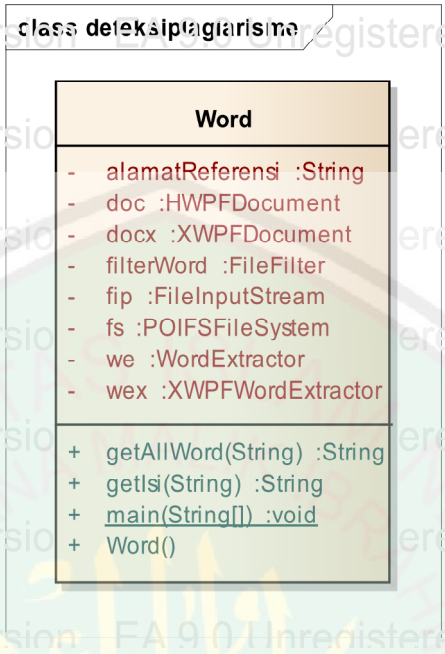
```
public boolean tulis(String isi[],String alamat){
        PrintWriter writer;
        try {
            try {
                    writer = new PrintWriter(alamat+".tmp","UTF-
8");
                    for(int i=0;i<isi.length;i++){
                         writer.println(isi[i]+"<~>");
                        }
                    writer.close();

                }catch (UnsupportedEncodingException ex) {
                    return false;
                    }
                }catch (FileNotFoundException ex) {
                    return false;
                    }
        return true;
      }
```

Gambar 4. 40 Source code operation tulis

4.1.1.4 Word

Class Word merupakan turunan dari kelas FileIO. Pada class ini terdapat operasi-operasi yang mengolah dokumen referensi berekstensi doc dan docx. *Constructor* pada class ini berfungsi untuk inisiasi variable filterWord, yang nanti digunakan dalam pengambilan listFile agar list file yang terambil hanya berekstensi doc dan docx. Berikut diagram class Word:



Gambar 4. 41 Diagram Class Word

- getAllWord(String)String

Operasi ini berfungsi untuk mengambil semua isi list dokumen referensi berekstensi doc atau docx. Operasi ini memiliki parameter bertipe data String yang merupakan folder tempat menyimpan dokumen referensi yang berekstensi doc dan docx dan mengembalikan nilai bertipe data String yang merupakan isi semua dokumen referensi. Dalam pembacaan memanfaatkan operasi getIsi(String) dalam class yang sama.

```
public String getAllWord(String alamat){
    File[]  f=new File(alamat).listFiles(filterWord);
    String kirim = "";
    for(int i=0;i<f.length;i++){
        if(kirim.equals("")){
            kirim=getIsi(f[i].getAbsolutePath());
            }else{

kirim=kirim+"\n"+getIsi(f[i].getAbsolutePath());
                }
      }
    return kirim;
    }
```

Gambar 4.42 Source code operasi getAllWord

- getIsi(String)String

Operasi ini berfungsi untuk mengambil semua isi file berekstensi doc atau docx. Operasi ini memiliki parameter bertipe data String yang merupakan *absolutePath* dari file yang akan dibaca dan mengembalikan nilai bertipe data String yang merupakan isi dokumen referensi.

```
public String  getIsi(String path){
        String word = null;
        File f=new File(path);
        if(f.getName().toUpperCase().endsWith(".DOC")){
                try {
                    fip=new FileInputStream(path);
                    fs=new POIFSFileSystem(fip);
                    doc=new HWPFDocument(fs);
                    we=new WordExtractor(doc);
                    word=we.getText();
                    fip.close();
                    }catch (IOException ex) {
                        }
            }else{
                try {
                    fip=new FileInputStream(path);
                    docx=new XWPFDocument(fip);
                    wex=new XWPFWordExtractor(docx);
                    word=wex.getText();
                    fip.close();
                    }catch (FileNotFoundException ex) {
                    }catch (IOException ex) {
                        }
                }
            return word;
    }
```

Gambar 4. 43 Source code operasi getIsi

4.1.1.5 PraProses

Pada Class terdapat operasi yang berfungsi untuk memisahkan suatu variable string menjadi kalimat-kalimat. Berikut diagram class PraProses:

Gambar 4. 44 Diagram Class PraProses

- listKalimat(String,String)

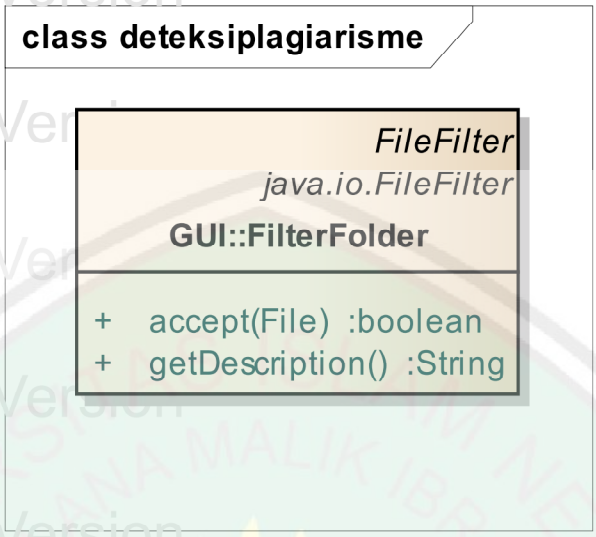
Operasi ini bersifat *public* dan berfungsi untuk memisahkan suatu string menjadi kalimat-kalimat. Operasi ini memiliki parameter bertipe data String yang merupakan variable yang akan dipisahkan per kalimat dan nama dokumen sebagai informasi kalimat tersebut berada. Operasi ini mengembalikan array String kalimat berserta informasi nama dokumen tempat kalimat berada.

```
public class PraProses {
    public String [] listKalimat(String text,String nim){
        final  Pattern  p  =  Pattern.compile("[\\.\\!\\?]\\s+",
Pattern.MULTILINE);
        String []s=p.split(text);
        int n=1;
        for(int i=0;i<s.length;i++){
            s[i]=s[i]+"~"+nim+"~"+String.valueOf(n);
            n++;
            }
        return s;
    }
}
```

Gambar 4. 45 Source code operasi listKalimat

4.1.1.6 FilterFolder

Class FilterFolder merupakan turunan dari class FileFilter dari *package* javax.swing.fileChooser bawaan bahasa pemrograman yang digunakan dan juga merupakan implementasi dari FileFilter dari *package* java.io. Fungsi dari class ini adalah sebagai filter file untuk jendela pilihan file. Berikut diagram class FilterFolder:



Gambar 4.46 Diagram Class FilterFolder

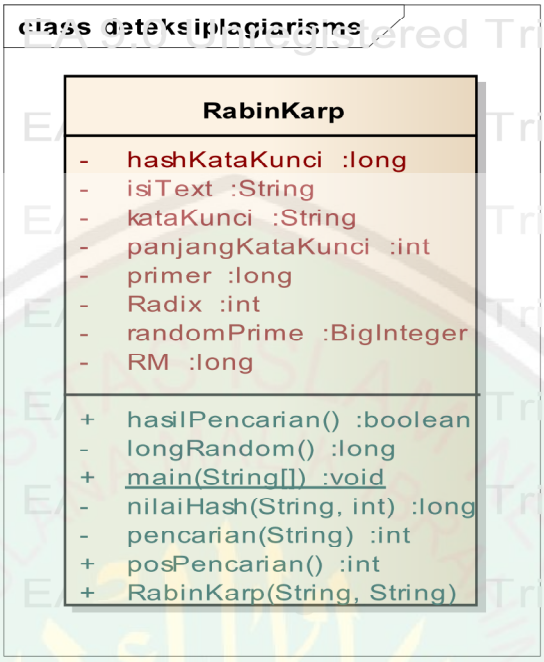
Sourcode dari class fileterFolder adalah sebagai berikut:

```
      import java.io.File;
      import javax.swing.filechooser.FileFilter;
      public class FilterFolder extends FileFilter implements
java.io.FileFilter{
          @Override
          public boolean accept(File f) {
              if(f.isDirectory()){
                  return true;
              }
              return false;
          }
          public String getDescription() {
              return "Silahkan memilih file";
          }
      }
```

Gambar 4. 47 Source code class FiletFolder

4.1.1.7 RabinKarp

Class ini berfungsi untuk melakukan pencarian keberadaan suatu *pattern* di dalam sebuah text dan merupakan perwujudan dari algoritma Rabin-Karp. *Contructor* class ini membutuhkan dua parameter yaitu String sebagai *pattern* dan String sebagai text. Berikut diagram class RabinKarp:



Gambar 4. 48 Diagram Class RabinKarp

Source code dari *constructor* adalah sebagai berikut:

```
public RabinKarp(String kataKunci, String text){
        isiText=text;
        this.kataKunci=kataKunci;
        Radix=256;
        panjangKataKunci=kataKunci.length();
        primer=longRandom();
        RM = 1;
        for (int i = 1; i <= panjangKataKunci-1; i++)
            {
                RM = (Radix * RM) % primer;
                }
        hashKataKunci = nilaiHash(kataKunci, panjangKataKunci);
    }
```

Gambar 4. 49 *Constructor* class RabinKarp

- hasilPencarian()boolean

Operasi ini bersifat *public* dan berfungsi untuk mengirimkan hasil pencarian. Apakah ditemukan kesamaan atau tidak. Operasi ini memanfaat operasi pencarian pada class yang sama.

```
public boolean hasilPencarian(){
        int pos = pencarian(isiText);
        if (pos == -1){
            return false;
            }else{
                return true;
                }
}
```

Gambar 4. 50 Source code operasi hasilPencarian

- longRandom()

Operasi ini bersifat *private* dan berfungsi untuk mencari nilai bilangan prima secara random yang akan digunakan sebagai pembagi. Operasi ini tidak memiliki parameter dan mengembalikan nilai bertipe data long.

```
private long longRandom(){
        randomPrime=BigInteger.probablePrime(31, new Random());
        return randomPrime.longValue();
    }
```

Gambar 4. 51 Source code operasi longRandom

- nilaiHash(String,int)

Operasi ini bersifat *private* dan berfungsi untuk mencari nilai hash dari String. Operasi ini memiliki parameter berupa String yang akan dicari nilai hash dan Int yang merupakan panjang dari Parameter pertama. Operasi ini mengembalikan nilai bertipe data long, yaitu nilai hash dari String.

```
private long nilaiHash(String kunci,int panjangKata){
        long h = 0;
        for(int i=0;i<panjangKata;i++){
            h=((Radix*h)+kunci.charAt(i))% primer;
        }
        return h;
    }
```

Gambar 4. 52 Source code operasi nilaiHash

- pencarian(String)

Operasi ini bersifat *private* dan merupakan operasi utama dari class RabinKarp. Operasi ini berfungsi mencari untuk posisi dari kata kunci pada text.

Operasi ini memiliki parameter berupa String, yang berfungsi sebagai text. Operasi ini mengembalikan nilai bertipe data int, yaitu posisi dari *pattern* pada text.

```
private int pencarian(String text){
        int n= text.length();
        if(n<panjangKataKunci){
            return n;
            }
        long hashText=nilaiHash(text, panjangKataKunci);
        if(hashKataKunci==hashText){
            return 0;
            }
        for(int i=panjangKataKunci;i<n;i++){
            hashText=(hashText+primer-(RM*text.charAt(i-
panjangKataKunci))%primer)%primer;
            hashText=((hashText*Radix)+text.charAt(i))%primer;
            int newPos=i-panjangKataKunci+1;
            if(hashKataKunci==hashText){
                return newPos;
                }
            }
        return -1;
        }
```

Gambar 4. 53 Source code operasi pencarian

Operasi ini bersifat *public* dan berfungsi untuk mengirimkan hasil pencarian berupa posisi *pattern* pada text. Operasi tidak memiliki parameter dan mengembalikan nilai berupa int, yaitu posisi *pattern* pada text. Operasi ini memanfaat operasi pencarian pada class yang sama.

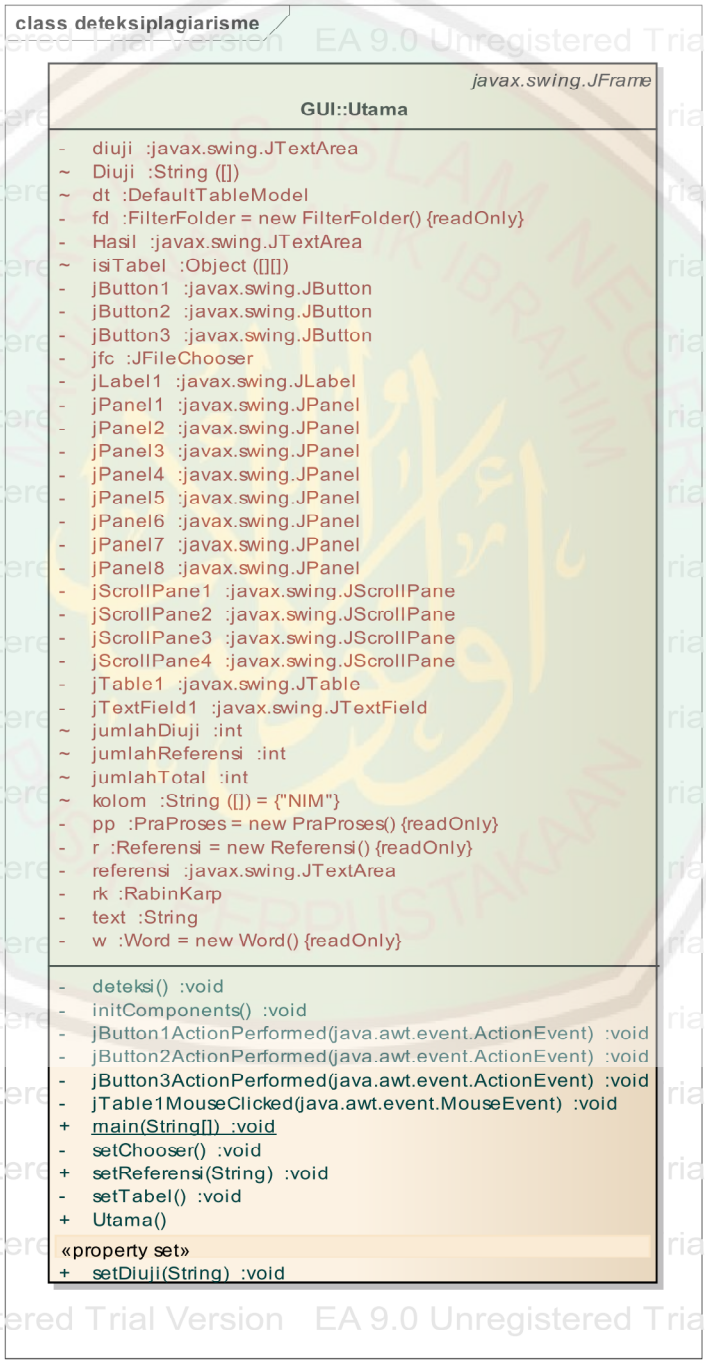
```
public int posPencarian(){
       int pos = pencarian(isiText);
       return pos;
    }
```

Gambar 4. 54 Source code operasi posPencarian

4.1.1.8 Utama

Pada class Utama terdapat operai-operasi berhubungan dengan tampilan antar muka dari aplikasi. Class ini merupakan turunan dari class JFrame yang merupakan bawaan dari bahasa pemrograman yang digunakan. Terdapat operasi-

operasi bawaan dari class Jframe sehingga yang operasi-operasi yang dibahas pada bagian ini terbatas pada operasi yang dibuat. Diagram class Utama ditunjukkan gambar 4.55.



Gambar 4. 55 Diagram Class Utama

Operasi ini berfungsi untuk menampilkan file temporari referensi ke dalam tabel untuk dilihat oleh pengguna. Tidak memiliki parameter dan tidak mengembalikan nilai.

```
private void setTabel(){
        isiTabel=new Object[r.listReferensi().length][1];
        for(int i=0;i<r.listReferensi().length;i++){
            isiTabel[i][0]=r.listReferensi()[i].getName();
            }
        dt=new DefaultTableModel(isiTabel, kolom);
        jTable1.setModel(dt);
    }
```

Gambar 4. 56 Source code operasi setTabel

- setDiuji()

Operasi ini berfungsi untuk menampilkan isi dan informasi file yang akan diuji ke dalam textfield diuji, dan menyimpan variable diuji untuk digunakan oleh operasi deteksi(). Tidak memiliki parameter dan tidak mengembalikan nilai.

```
public void setDiuji(String alamat){
       setCursor(Cursor.WAIT_CURSOR);
       Diuji=r.isiListReferensi(alamat);
       jumlahDiuji=Diuji.length;
        for (String Diuji1 : Diuji) {
            diuji.append("\n" + Diuji1.split("~")[1] + "\n
Kalimat ke : " + Diuji1.split("~")[2] + "\n" +
Diuji1.split("~")[0].replaceAll("\n", " ") + "\n");
            }
       referensi.setLineWrap(true);
       setCursor(Cursor.DEFAULT_CURSOR);
     jTextField1.requestFocus();
    }
```

Gambar 4. 57 Source code operasi setDiuji

- setReferensi()

Operasi ini berfungsi untuk menampilkan isi dan informasi dokumen referensi yang dipilih oleh pengguna dalam tabel dokumen referensi ke dalam textfield isi dokumen referensi. Tidak memiliki parameter dan tidak mengembalikan nilai.

```
public void setReferensi(String nim){
        setCursor(Cursor.WAIT_CURSOR);
        String
s[]=r.getListIsi(Referensi.alamatTMP+File.separator+nim+".tmp");
         for (String item : s) {
             referensi.append("\n" + item.split("~")[1] + "\n
Kalimat ke : " + item.split("~")[2] + "\n" +
item.split("~")[0].replaceAll("\n", " ") + "\n");
         }
        referensi.setLineWrap(true);
        setCursor(Cursor.DEFAULT_CURSOR);
        jumlahDiuji=s.length;
         this.requestFocus();
    }
```

Gambar 4. 58 Source code operasi setReferensi

- setChooser()

Operasi ini berfungsi untuk menampilkan jendela pilihan file. Setelah file terpilih, menggunakan alamat absolut dari file terpilih untuk menjalakan operasi setDiuji(). Tidak memiliki parameter dan tidak mengembalikan nilai.

```
private void setChooser(){
        jfc = new JFileChooser();
        jfc.setFileFilter(fd);
        jfc.setCurrentDirectory(new
File(Referensi.alamatReferensi));
        jfc.setDialogTitle("Pilih Folder File Skripsi");
        jfc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        jfc.setAcceptAllFileFilterUsed(false);
        String alamat = null;
        if (jfc.showOpenDialog(this) ==
JFileChooser.APPROVE_OPTION) {
          alamat= jfc.getSelectedFile().getAbsolutePath();
          }
         jTextField1.setText(alamat);
         setDiuji(alamat);
    }
```

Gambar 4. 59 Source code operasi setChooser

- deteksi()

Operasi ini berfungsi untuk memulai proses deteksi. Dan menampilkan hasil deteksi pada textfield hasil deteksi. Tidak memiliki parameter dan tidak mengembalikan nilai.

```
private void deteksi(){
        text=r.getAll().replaceAll("\\s", "").toLowerCase();
        jumlahReferensi=text.split("<~>").length;
        int k=0; int m=0;
        for (String Diuji1 : Diuji) {
            rk = new RabinKarp(Diuji1.replaceAll("\\s",
"").toLowerCase().split("~")[0], text);
            m++;
            if (rk.hasilPencarian()) {
                k++;
                Hasil.append("\n NIM : " + Diuji1.split("~")[1]
+ "\n" + "Ditemukan kesamaan pada kalimat ke : " +
Diuji1.split("~")[2] + "\n dengan NIM : " +
text.substring(rk.posPencarian()).split("~")[1] + "\n kalimat ke
: " +
text.substring(rk.posPencarian()).split("~")[2].replaceAll("<",
"") + "\n" + Diuji1.split("~")[0].replaceAll("\n", " ") + "
\n");
                }
            }
        jumlahDiuji=k;
        jumlahTotal=m;
    }
```

Gambar 4. 60 Source code operasi deteksi

4.3.2 Pengujian

Proses pengujian dibagi menjadi beberapa tahap. Pembagian ini diharapkan mampu memaksimalkan hasil pengujian

4.3.2.1 Persiapan

Pada tahap ini, mempersiapkan kebutuhan-kebutuhan pengujian. Kebutuhan-kebutuhan-kebutuhan antara lain:

- Aplikasi deteksi plagiat berdasarkan *string-matching* menggunakan algoritma Rabin-Karp.
- PC atau laptop yang digunakan untuk menjalankan aplikasi. Pada pengujian ini menggunakan laptop dengan sepesifikasi seperti pada tabel 4.5:

Tabel 4. 5 Spesifikasi

Sistem operasi	Windows 8 64 bit
Processor	Amd Athlon X2
Memori	DDR2 4GB
Hardisk	WD 500gb
JVM	Jre 1.8.0_05

- Data yang digunakan sebagai referensi adalah sebagai berikut:

Tabel 4. 6 Data Referensi

Deskripsi	*Softcopy* tugas akhir atau skripsi dari alumni Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang
Ekstensi	doc dan docx
Isi	File terpisah BAB I, BAB II, BAB III, BAB IV, BAB V
Jumlah	10 Dokumen
Jumlah kalimat	6676 Kalimat

- Data yang digunakan sebagai sampel diuji adalah sebagai berikut :

  a. File berkektensi docx dengan tujuh belas kalimat. Terdiri dari 1 file.

  b. File berkektensi doc dengan tujuh belas kalimat. Terdiri dari 1 file.

  c. Satu bab laporan skripsi. Terdiri dari 1 file.

  d. Bab satu sampai dengan bab lima laporan skripsi. Terdiri dari 5 file

  e. Laporan skripsi penuh. Terdiri dari 1 file.

4.3.2.2 Pengujian memulai aplikasi

Dalam pengujian ini aplikasi dijalankan dengan tiga parameter berbeda. Ketiga parameter tersebut adalah:

- Memulai aplikasi dengan ekstensi exe, dengan melakukan *bundled* terhadap mesin virtual.
- Memulai aplikasi dengan ekstensi jar, pengguna melakukan instalasi mesin virtual terpisah.

- Memulai aplikasi dari IDE, mesin virtual diserahkan pada IDE yang digunakan.

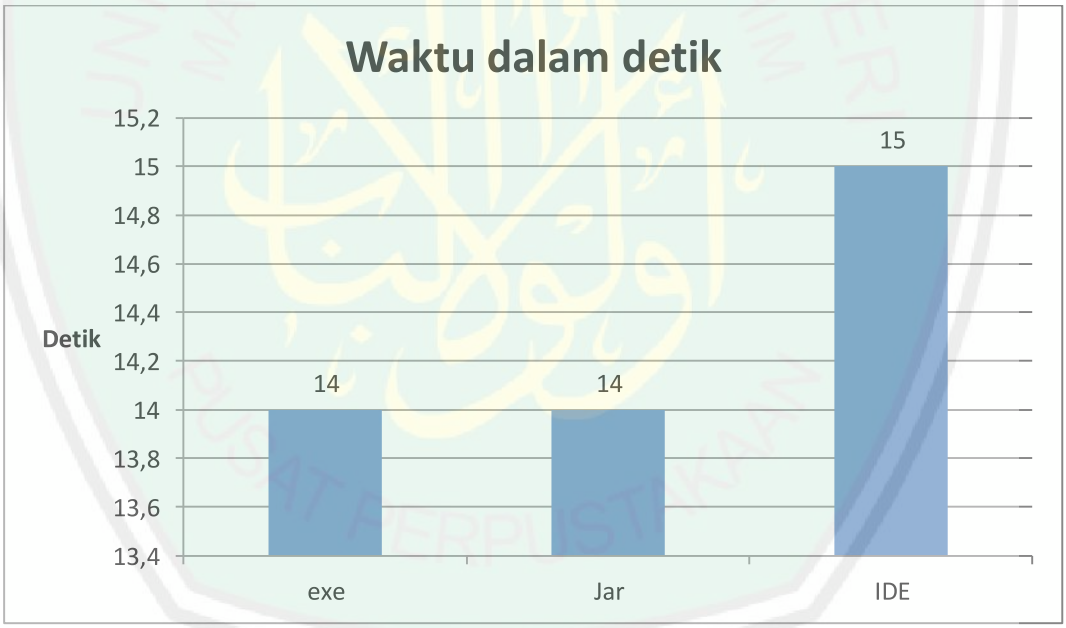
Hasil pengujian ditunjukkan tabel dibawah ini:

Tabel 4. 7 Hasil Pengujian menjalankan Aplikasi

Dijalankan melalui	Waktu dalam detik	Cpu dalam %	Memori dalam Mb
exe	14	67,0	137,0
Jar	14	95,2	236,2
IDE	15	93,0	256,2

Hasil pengujian sebagai berikut:

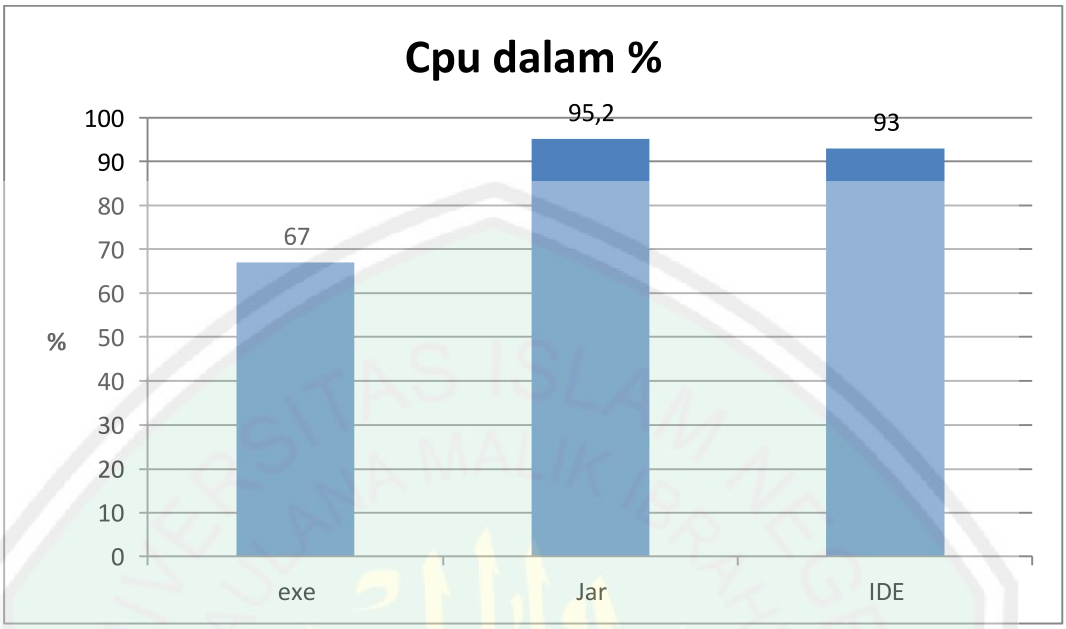
- Waktu diambil dari menjalankan aplikasi hingga aplikasi dalam keadaan idle.



Gambar 4. 61 Gambar grafik waktu menjalan aplikasi

Pada hasil pengujian ini menjalankan aplikasi dengan ekstensi exe dan jar membutuhkan waktu yang sama yaitu 14 detik, ditunjukkan pada gambar 4.58. Sedangkan ketika jalankan melalui IDE membutuhkan waktu 15 detik.

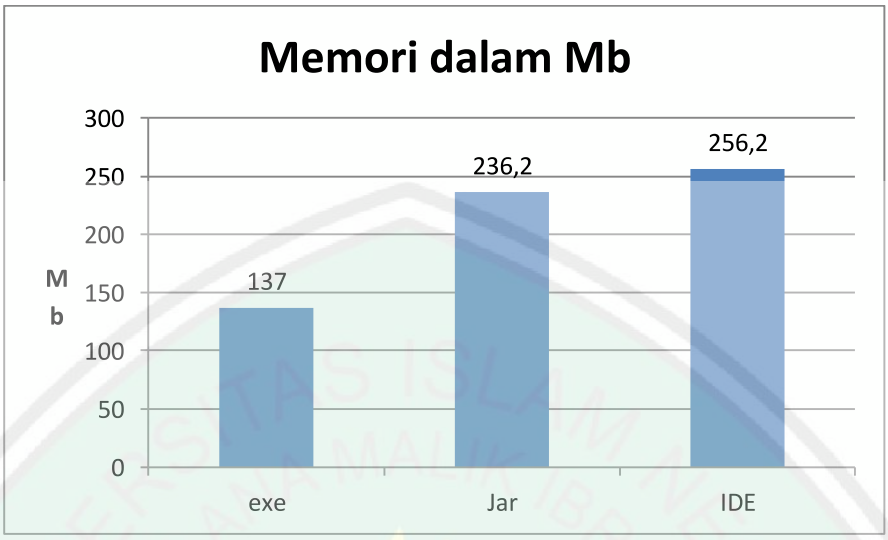
- Penggunaan CPU diambil nilai tertinggi dari menjalankan aplikasi hingga aplikasi dalam keadaan idle.



Gambar 4. 62 Gambar grafik Penggunaan CPU menjalankan aplikasi

Pada Gambar 4.62 menunjukan penggunaan CPU yang terendah adalah menjalankan aplikasi dijalankan menggunakan menggunakan ekstensi exe pada nilai 63 %. Sedangkan penggunaan tertinggi aplikasi dijalankan menggunakan ekstensi Jar pada nilai penggunaan 95,5%. Sedangkan menjalankan menggunakan IDE menggunakan 93%.

- Penggunaan Memori diambil nilai tertinggi dari menjalankan aplikasi hingga aplikasi dalam keadaan idle.



Gambar 4. 63 Gambar Grafik Penggunaan Memori menjalankan aplikasi

Pada gambar 4.63 menunjukan penggunaan memori yang terendah adalah menjalankan aplikasi dijalankan menggunakan menggunakan ekstensi exe pada nilai 137 Mb. Sedangkan penggunaan tertinggi aplikasi dijalankan menggunakan ekstensi IDE pada nilai penggunaan 256.2 Mb Sedangkan menjalankan menggunakan jar menggunakan 236.2 Mb.

Dari hasil pengujian menjalankan aplikasi dapat ditarik kesimpulan menjalankan aplikasi menggunakan ekstensi exe membutuhkan sumber daya dan waktu paling rendah dibanding dengan exktensi Jar dan menjalankan melalui IDE. Dengan kebutuhan waktu 14 detik, penggunaan CPU 63% dan pengunaan memori sebesar 137 Mb.

4.3.2.3 Pengujian Fungsional

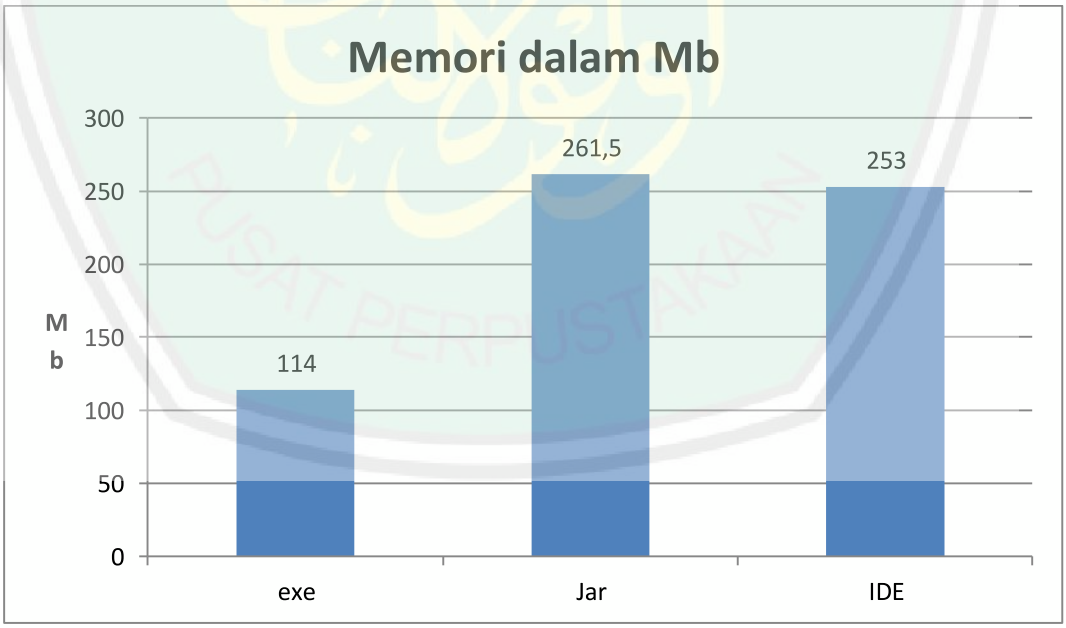
Pada pengujian dilakukan untuk mengetahui kemampuan mendeteksi plagiarisme mode *Copy&Paste plagiarism* dan seberapa banyak sumber daya yang digunakan untuk menjalankan apliaksi. Pengujian fungsional yang pertama

adalah aplikasi pada posisi IDLE. Parameter yang digunakan sama seperti pada tahap pengujian menjalankan aplikasi.

Tabel 4. 8 Hasil Pengujian aplikasi dalam kondisi idle

Dijalankan melalui	Cpu dalam %	Memori dalam MB
Exe	0	114,0
Jar	0	261,5
IDE	0	253,0

Dari hasil pengujian sebelumnya dapat disimpulkan menjalankan aplikasi melalui ekstensi exe lebih efisien dalam penggunaan sumber daya komputer dibanding dengan menggunakan ekstensi jar maupun menggunakan IDE. Tidak ada perbedaan dalam pengunaan CPU. Adapun grafik perbandingan penggunaan memori dalam keadaan idle ditunjukkan pada gambar 4.64 sebagai berikut:



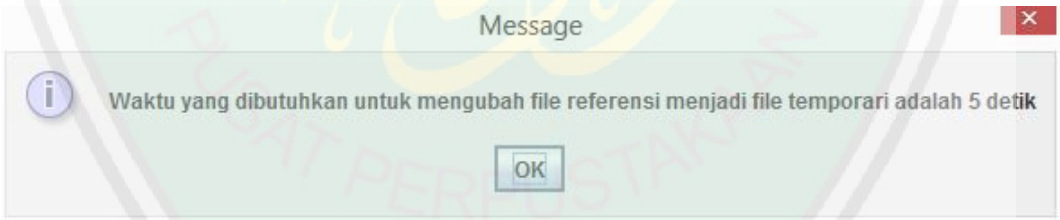
Gambar 4. 64 Grafik Penggunaan memori dalam Keadaan IDLE

Adapun tampilan jendela utama dalam kondisi idle ditunjukkan pada gambar 4.65 sebagai berikut:

Gambar 4. 65 Tampilan Jendela Utama

Pada pengujian selanjutnya difokuskan pada aplikasi yang dijalankan menggunakan ekstensi .exe, hal ini dikarenakan pada pengujian sebelumnya menunjukan aplikasi dijalankan dengan menggunakan ekstensi .exe lebih efisien dalam penggunaan sumber daya komputer.

Pada saat menjalankan aplikasi terdapat proses merubah dokumen referensi dari berekstensi doc atau docx menjadi dalam file temporari. Waktu yang dibutuhkan menjalakan proses adalah 5 detik, dengan penggunanaan CPU sebesar 51,2 % dan penggunaan memori sebesar 202 Mb. Pada Gambar 4.66 ditunjukkan jendela pemberitahuan lama waktu yang digunakan dalam merubah dokumen referensi dari berektensi doc dan docx menjadi file temporari. Jendela menunjukkan waktu yang dibutuhkan untuk mengubah dokumen referensi menjadi file temporari dan berfungsi sebagai pemberitahuan proses pengubahan telah selesai.



Gambar 4. 66 Tampilan Jendela pemberitahuan waktu

Pengujian dilakukan dengan kondisi aplikasi dijalankan dengan ektstensi exe. Hal ini dikarenakan hasil pengujian menunjukan aplikasi dijalan dengan ekstensi exe lebih menghemat penggunaan sumber daya. Dokumen yang diubah adalah 10 laporan dengan 6676 kalimat.

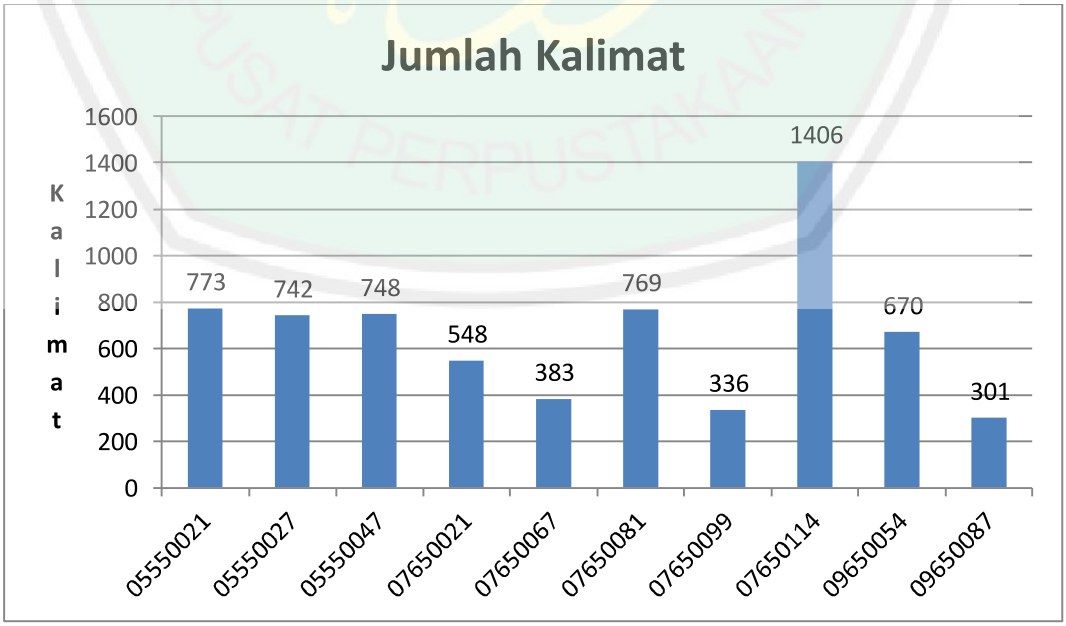
Pengujian selanjutnya adalah pembacaan dokumen referensi. Setiap dokumen referensi dibuka satu persatu. Data hasil pengujian ini adalah waktu

yang dibutuhkan dan penggunaan sumber daya setiap pembacaan dokumen. Data hasil percobaan ditunjukan pada tabel 4.9

Tabel 4. 9 Hasil Pengujian membuka Dokumen referensi

No	NIM	Jumal Kalimat	Waktu dalam detik	CPU dalam %	Memori dalam Mb
1	05550021	773	2,0	29	140
2	05550027	742	1,0	26	140
3	05550047	748	0,9	17	140
4	07650021	548	0,6	13	140
5	07650067	383	0,3	8	140
6	07650081	769	0,8	16	140
7	07650099	336	0,3	6	140
8	07650114	1406	3,0	49	140
9	09650054	670	0,6	12	140
10	09650087	301	0,3	13	140

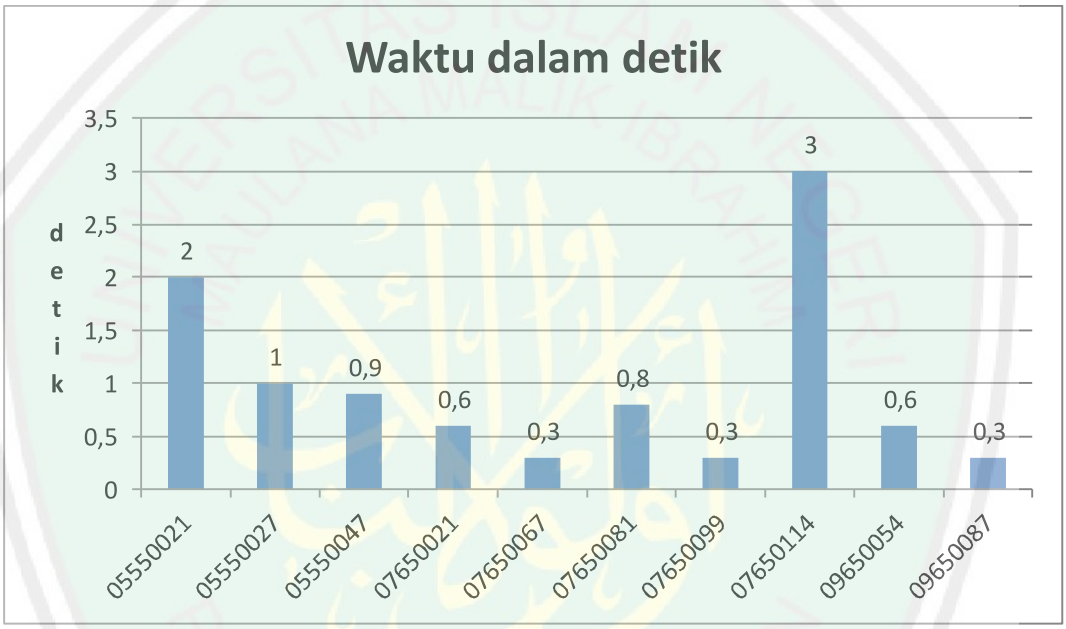
Perbandingan jumlah kalimat yang terdapat pada data referensi ditunjukkan pada gambar 4.67 sebagai berikut:



Gambar 4. 67 Grafik perbandingan jumlah kalimat

Pada data 07650114 memiliki jumlah kalimat paling banyak dengan 1406 kalimat. Dan data 09650087 memiliki jumlah kalimat paling sedikit dengan 301 kalimat. Jumlah kalimat total dari seluruh data adalah 6676 kalimat.

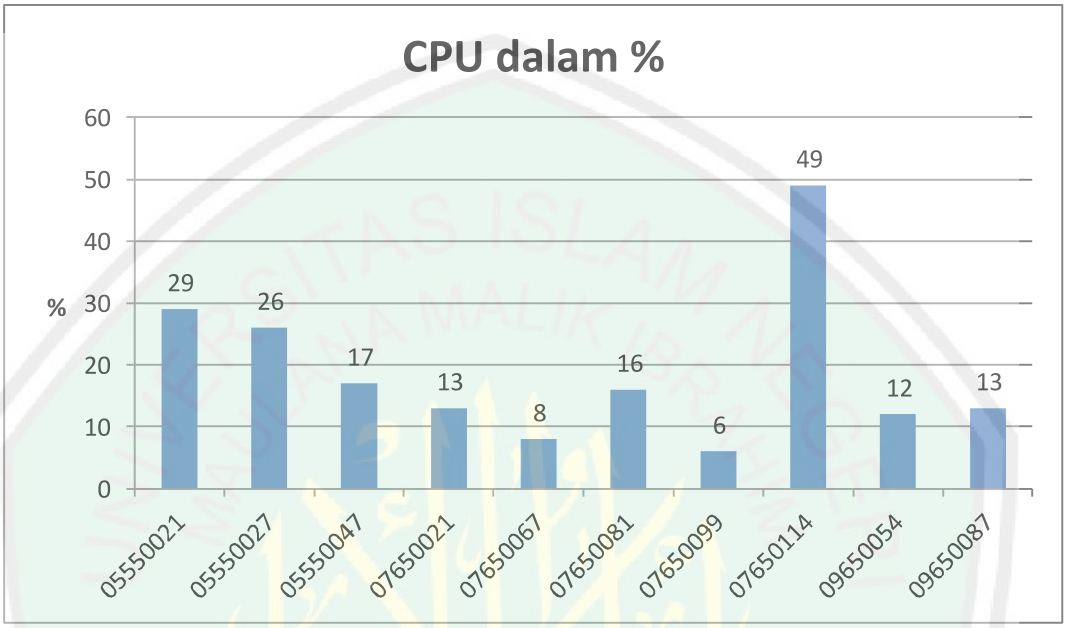
Perbandingan waktu yang digunakan dalam menampilkan isi dokumen referensi dapat dilihat pada gambar 4.68.



Gambar 4. 68 Grafik perbandingan waktu yang dibutuhkan dalam menampilkan dokumen referensi

Terlihat data 076500114 paling lama, dengan membutuhkan waktu sebanyak 3 detik. Sedangkan data 09650087, 07650067 dan 07650099 sama-sama membutuhkan waktu paling sedikit. Dengan membutuhkan waktu sebanyak 0,3 detik. Waktu rata-rata yang digunakan dalam membaca dan menampilkan data referensi adalah 0,98 detik. Jumlah kalimat berpengaruh pada waktu yang dibutuhkan untuk menampilkan dokumen referensi pada pengguna. Semakin banyak kalimat maka semakin banyak waktu yang dibutuhkan. Namun bila selisih jumlah kalimat tidak terlalu tinggi pengaruh tidak terlalu terlihat.

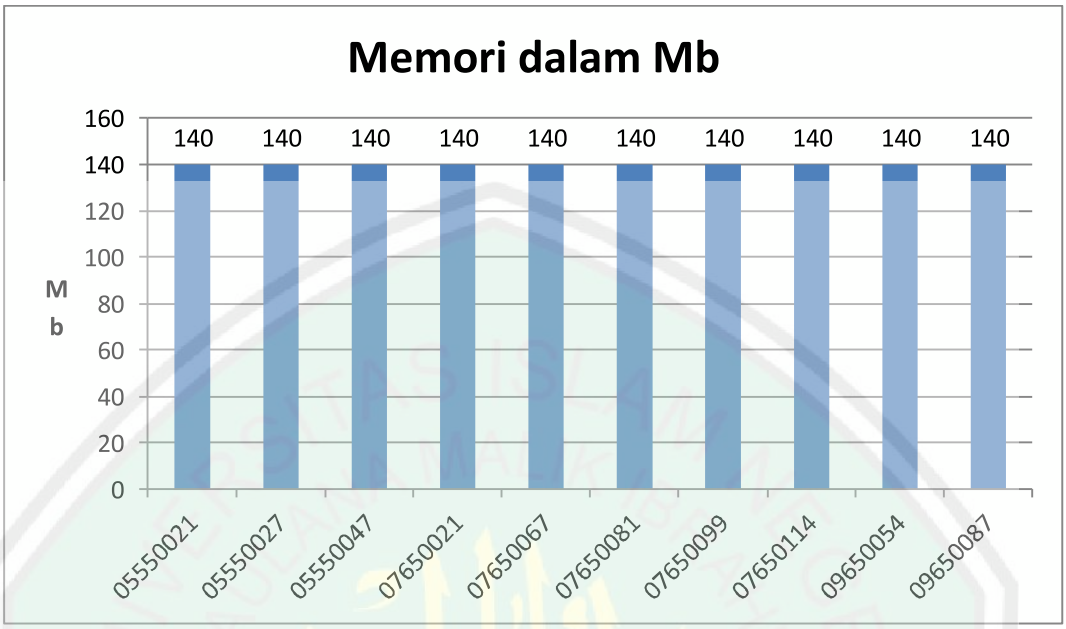
Perbandingan penggunan CPU dalam menampilkan isi dokumen referensi ditunjukkan pada Gambar 4.69.



Gambar 4. 69 Grafik perbandingan penggunaan CPU dalam menampilkan dokumen referensi

Terlihat pada gambar 4.66 data referensi 7650114 menggunakan CPU paling besar dengan penggunaan 49 %. Sedangkan data referensi 7650099 menggunakan CPU paling sedikit dengan penggunaan 6 %. Dengan penggunaan CPU rata-rata untuk membuka dokumen referensi 18,9 %. Semakin banyak kalimat maka semakin banyak penggunaan CPU. Namun bila selisih jumlah kalimat tidak terlalu tinggi pengaruh tidak terlalu terlihat.

Sedangkan dalam penggunaan memori tidak terdapat perbedaan. Penggunaan memori stabil pada nilai 140 Mb. Perbandingan penggunaan memori dalam menampilkan isi dokumen referensi ditunjukkan pada gambar 4.69.



Gambar 4. 70 Grafik perbandingan penggunaan memori dalam menampilkan dokumen referensi

Dari perbandingan data hasil pengujian dapat ditarik kesimpulan jumlah kalimat berpengaruh terhadap waktu yang dibutuhkan dan penggunaan CPU untuk menampilkan dokumen referensi. Sedangkan penggunaan memori tidak terpengaruh. Waktu rata-rata yang digunakan untuk menampilkan satu dokumen referensi adalah 0,98 detik dan penggunaan CPU adalah 18,9 %. Sedangkan waktu yang dibutuhkan untuk menampilkan satu kalimat adalah 0,0015 detik. Penggunaan Memori stabil di 140 Mb. Pada gambar 4.71 ditunjukkan jendela utama menampilkan dokumen referensi.

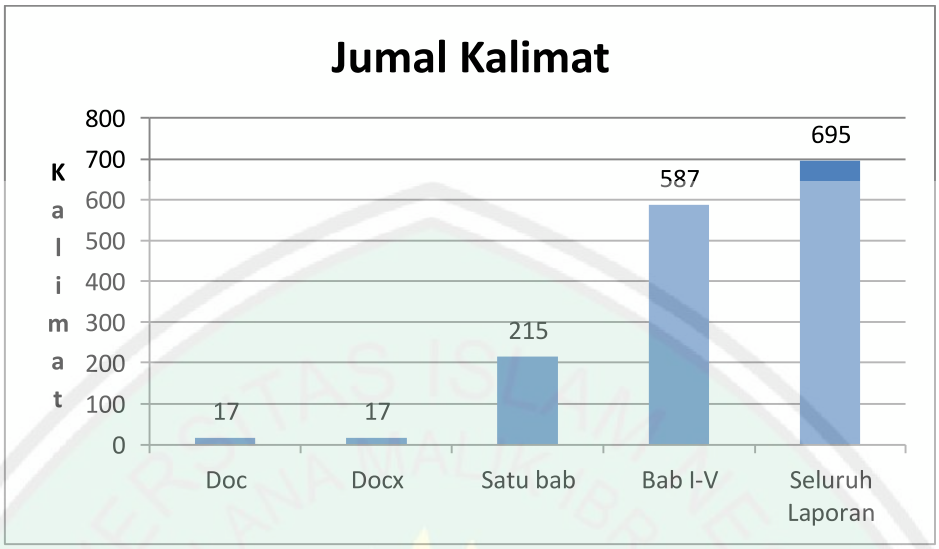
Gambar 4. 71 Menampilkan referensi

Pengujian selanjutnya adalah mengambil dan menampilkan file yang akan di uji. Berbeda dengan dokumen referensi, file yang akan diuji tidak dirubah menjadi file termporari. Ada lima dokumen yang digunakan sebagai *sample* data dan ditampilkan satu persatu. Tabel hasil pengujian sebagai berikut:

Tabel 4. 10 Hasil pengujian membuka file yang akan diuji

No	Nama	Jumal Kalimat	Waktu dalam detik	CPU dalam %	Memori dalam Mb
1	Doc	17	0,03	3,00	142
2	Docx	17	0,04	4,00	145
3	Satu bab	215	1,00	22,00	147
4	Bab I-V	587	7,00	53,00	199
5	Seluruh Laporan	695	8,00	51,8	233

Kelima dokumen yang akan diuji memiliki parameter yang berbeda-beda. Untuk dokumen pertama dan kedua memiliki jumlah dan kalimat yang sama. Perbedaannya adalah dokumen pertama berekstensi doc sedangkan dokumen kedua berekstensi docx. Dokumen bab I-V memiliki jumlah file yang berbeda. Dokumen ini memiliki lima file, terdiri dari bab satu sampai dengan bab lima. Dokumen lain hanya terdiri satu file. Adapun grafik perbandingan jumlah kalimat dapat dilihat pada gambar sebagai berikut:

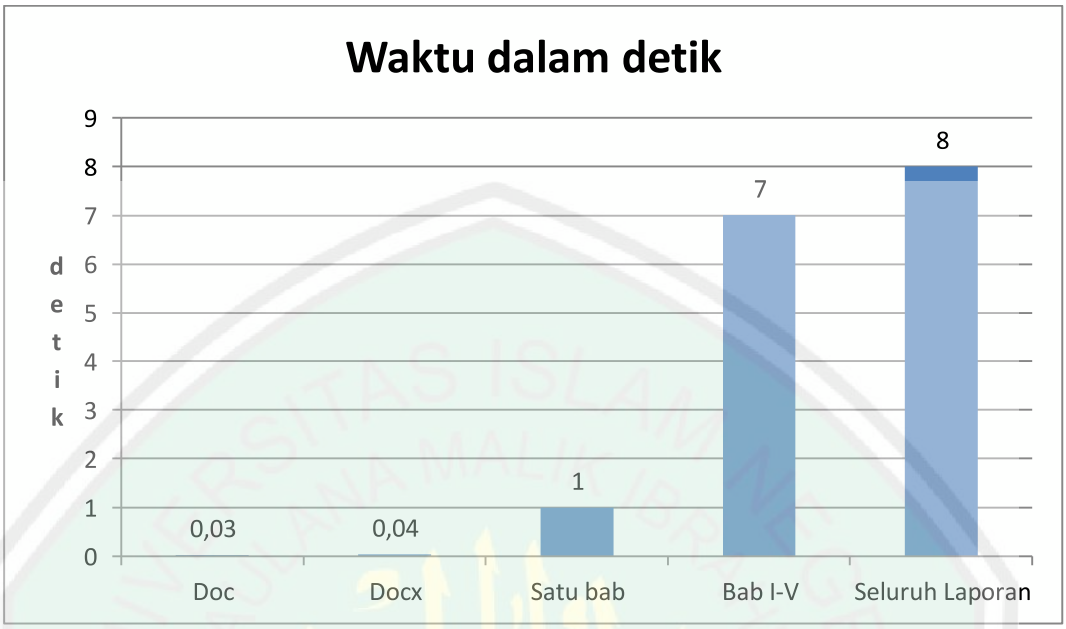


Gambar 4. 72 Grafik perbandingan jumlah kalimat dokumen yang akan diuji

Dokumen seluruh laporan memiliki jumlah kalimat paling banyak dengan 695 kalimat. Dokumen ini berisi seluruh isi laporan dari halaman judul sampai dengan lampiran-lampiran. Dokumen Bab I-V memiliki jumlah kalimat sebanyak 587. Dokumen ini berisi laporan bab satu sampai dengan bab lima. Masing-masing bab disimpan dalam file tersendiri. Sehingga terdapat lima file di dalam dokumen ini. Dokumen satu bab berisi laporan bab satu dari laporan skripsi. Dokumen ini hanya memiliki satu file dengan jumlah kalimat 215. Sedangkan dokumen doc dan docx adalah file sampel yang didalam terdapat beberapa kalimat dari data referensi. Terdiri dari satu file dengan jumlah kalimat 17.

Pengujian dilakukan dengan menampilkan dokumen-dokumen yang akan diuji. Lalu dibandingkan waktu, penggunaan CPU dan penggunaan memori yang dibutuhkan untuk menampilkan isi dokumen tersebut kepada pengguna.

Perbandingan waktu yang digunakan dalam menampilkan isi dokumen yang akan diuji dapat dilihat pada gambar 4.73 sebagai berikut:

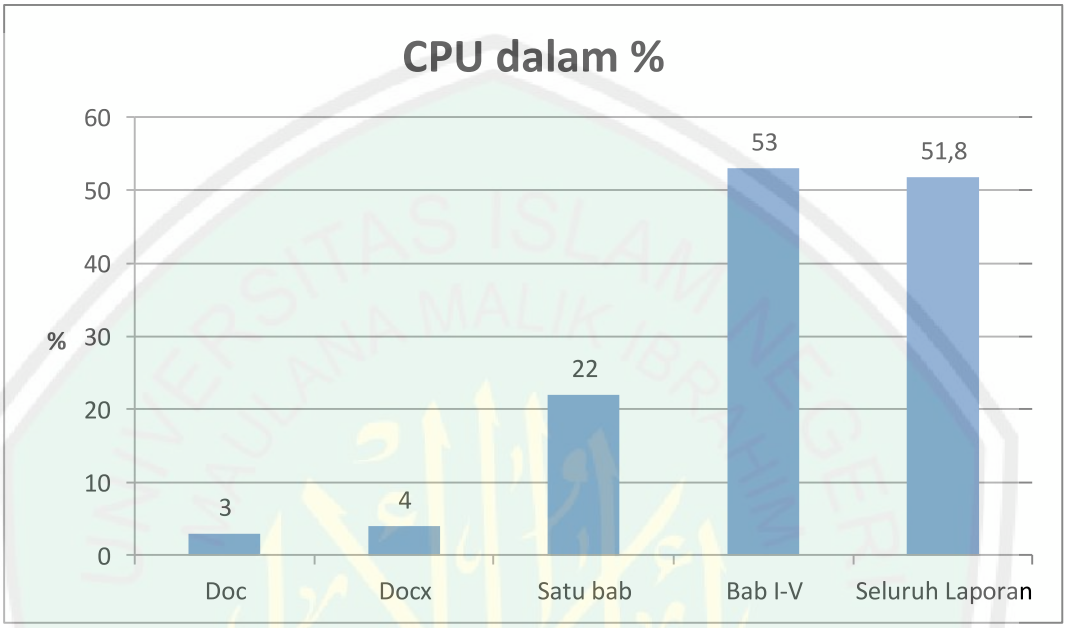


Gambar 4. 73 Grafik perbandingan waktu yang digunakan untuk menampilkan dokumen yang akan diuji

Pada dokumen seluruh laporan membutuhkan waktu paling lama untuk ditampilkan kepada pengguna. Waktu yang dibutuhkan adalah 8 detik. Sedangkan dokumen Doc membutuhkan waktu paling sedikit, yaitu 0,3 detik. Waktu rata-rata yang digunakan untuk menampilkan satu dokumen yang akan diuji adalah 3,614 detik. Jumlah kalimat berpengaruh terhadap waktu yang dibutuhkan dimana semakin banyak kalimat maka semakin banyak waktu yang digunakan.

Waktu rata-rata yang digunakan untuk menampilkan isi referensi lebih sedikit daripada menampilkan isi dokumen yang akan diuji. Hal ini disebabkan dokumen yang diuji ditampilkan langsung dari file berekstensi doc atau docx. Sedangkan dokumen referensi sebelum ditampilkan diubah terlebih dahulu menjadi file temporari. Sehingga data yang ditampilkan berasal dari file temporari.

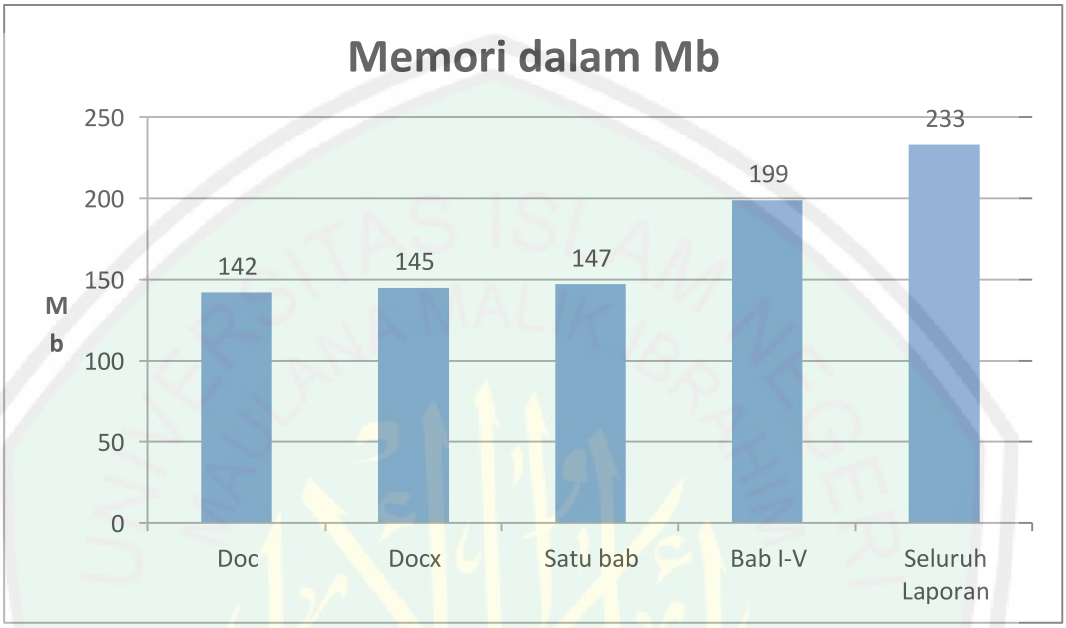
Perbandingan penggunan CPU dalam menampilkan isi dokumen yang akan diuji ditunjukkan pada Gambar 4.74 sebagai berikut:



Gambar 4. 74 Grafik perbandingan penggunaan CPU dalam menampilkan dokumen diuji

Terlihat pada gambar 4.74 dokumen bab I-V menggunakan CPU paling besar dengan penggunaan 53 %. Sedangkan dokumen doc menggunakan CPU paling sedikit dengan penggunaan 3 %. Dengan penggunaan CPU rata-rata untuk membuka dokumen referensi 26,76 %. Jumlah kalimat pada dokumen berpengaruh pada penggunaan CPU. Dimana semakin banyak kalimat semakin banyak sumber daya CPU yang digunakan. Jumlah file pada dokumen juga berpengaruh terlihat pada dokumen bab I-V yang memiliki jumlah kalimat yang lebih sedikit namun memiliki jumlah file yang lebih banyak dari dokumen seluruh laporan menggunakan lebih banyak sumber daya CPU. Sedangkan pada dokumen doc dan docx terdapat perbedaan namun tidak terlalu besar.

Perbandingan penggunan memori dalam menampilkan isi dokumen yang akan diuji ditunjukkan pada Gambar 4.75 sebagai berikut:



Gambar 4. 75 Grafik perbandingan penggunaan memori dalam menampilkan dokumen yang akan diuji

Terlihat pada gambar 4.75 dokumen seluruh laporan menggunakan memori paling besar dengan penggunaan 233 Mb. Sedangkan dokumen doc menggunakan memori paling sedikit dengan penggunaan 142. Dengan penggunaan memori rata-rata untuk membuka dokumen yang akan diuji 173,2 Mb. Jumlah kalimat pada dokumen berpengaruh pada penggunaan memori. Dimana semakin banyak kalimat semakin banyak memori yang digunakan. Sedangkan Jumlah file tidak berpengaruh terhadap penggunaan memori. Terdapat perbedaan antara dokumen doc dan docx, namun tidak terlalu besar.

Dari perbandingan data-data hasil pengujian didapatkan, jumlah kalimat berpengaruh terhadap waktu yang dibutuhkan, penggunaan CPU dan penggunaan memori untuk menampilkan dokumen yang akan diuji. Ekstensi file memiliki

pengaruh yang tidak terlalu signifikan pada dokumen dengan tujuh belas kalimat. Semakin banyak kalimat yang dalam suatu dokumen membutuhkan waktu yang lebih lama, penggunaan CPU dan memori yang lebih besar. Pengecualian untuk percobaan ke empat di mana dokumen terdiri dari lima file. Jumlah file memiliki pengaruh terutama penggunaan CPU.

Waktu rata-rata yang digunakan untuk menampilkan satu dokumen yang akan diuji adalah 3,614 detik, penggunaan CPU adalah 26,76 % dan penggunaan memori sebesar 173,2 Mb. Sedangkan waktu yang dibutuhkan untuk menampilkan satu kalimat adalah 0,012 detik. Jendela utama saat menampilkan isi dokumen yang akan diuji ditunjukkan pada gambar 4.75:

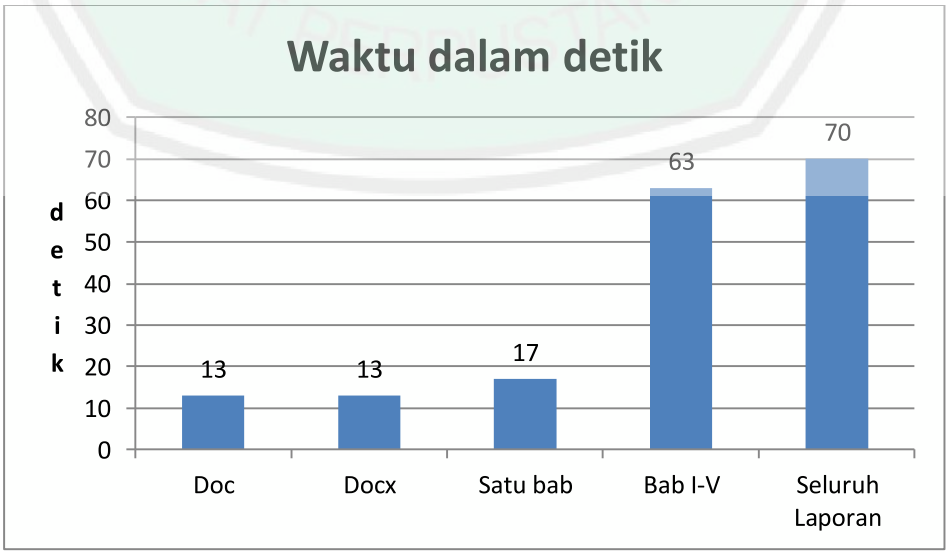
Gambar 4. 76 Menampilkan isi dokumen yang akan diuji

Pengujian selanjutnya adalah pengujian deteksi. Ada lima dokumen yang digunakan sebagai *sample* data dan sepuluh dokumen sebagai referensi dengan jumlah kalimat 6676. Tabel hasil pengujian sebagai berikut:

Tabel 4. 11 Hasil pengujian deteksi

No	Nama file	Jumal Kalimat	Jumlah terdeteksi	Waktu dalam detik	CPU dalam %	Memori dalam Mb
1	doc	17	3	13	51,2	144
2	docx	17	3	13	51,4	145
3	Satu bab	215	0	17	51,2	185
4	Bab I-V	556	2	63	51,1	199
5	Seluruh Laporan	695	2	70	51,8	268

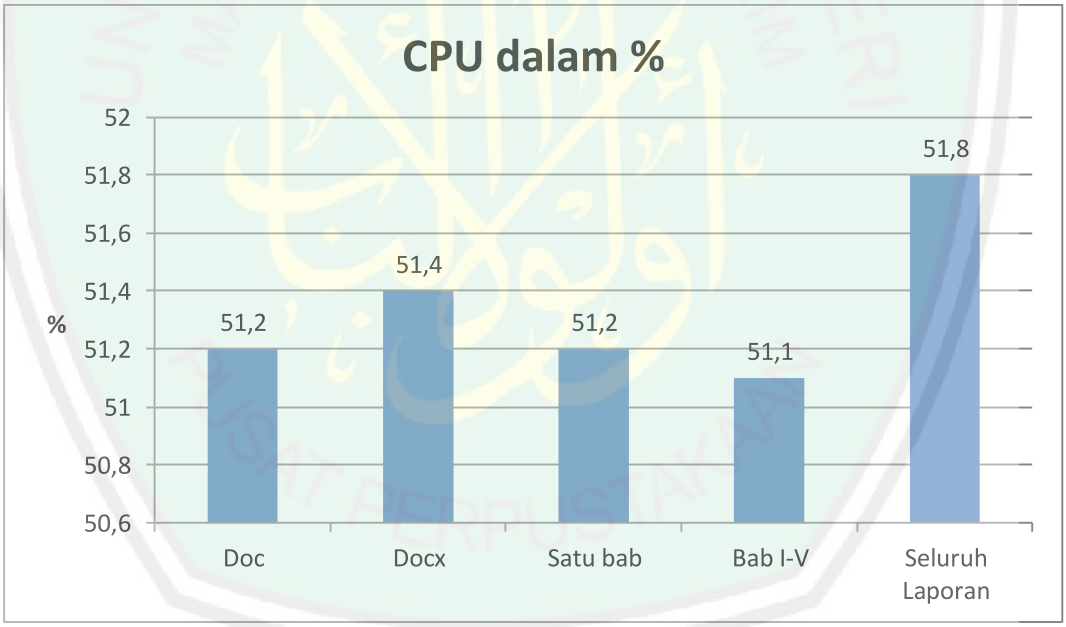
Pengujian ini untuk mengetahui kemampuan deteksi dan penggunaan sumber daya yang digunakan untuk melakukan deteksi. Dokumen doc dan docx yang merupakan sampel pengujian terdiri dari 17 kalimat, dimana 3 kalimat merupakan hasil *copy-paste* dari dokumen referensi dengan nim 09650087. Adapun perbandingan waktu yang digunakan untuk deteksi ditunjukkan gambar 4.77 sebagai berikut:



Gambar 4. 77 Grafik perbandingan waktu yang digunakan untuk proses deteksi

Pada dokumen seluruh laporan membutuhkan waktu paling lama untuk deteksi. Waktu yang dibutuhkan adalah 70 detik. Sedangkan dokumen doc dan docx membutuhkan waktu paling sedikit, yaitu 3 detik. Waktu rata-rata yang digunakan untuk mendeteksi satu dokumen adalah 35,2 detik. Jumlah kalimat berpengaruh terhadap waktu yang dibutuhkan dimana semakin banyak kalimat maka semakin banyak waktu yang digunakan untuk melakukan deteksi.

Perbandingan penggunan CPU dalam mendeteksi ditunjukkan pada gambar 4.78 sebagai berikut:

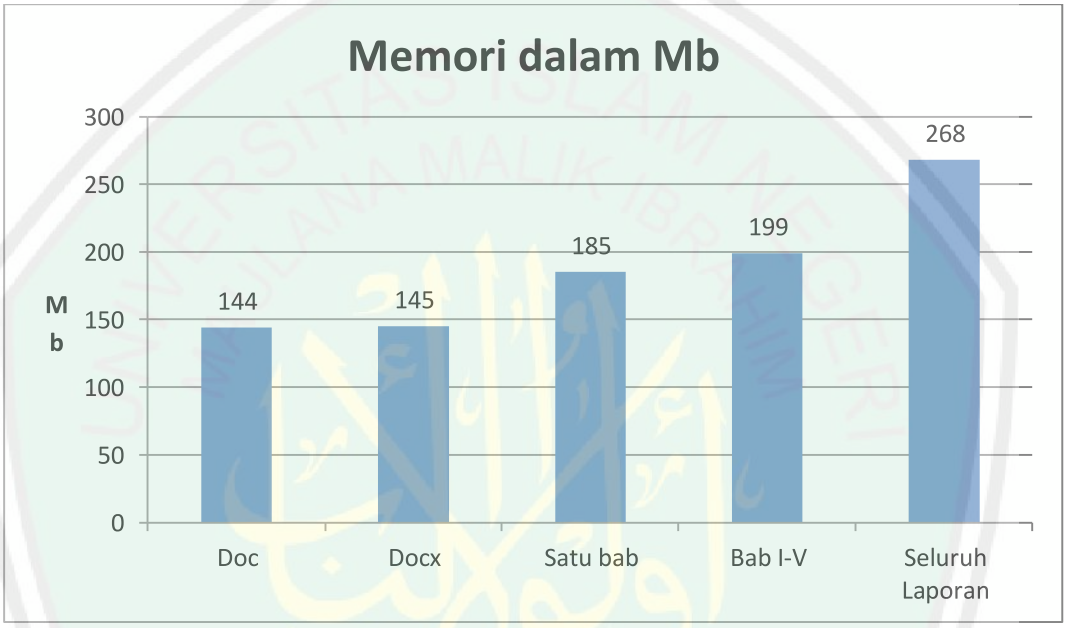


Gambar 4. 78 Grafik perbandingan penggunaan CPU dalam melakukan deteksi

Pada pengunaan CPU untuk melakukan deteksi tidak terlalu terpengaruh oleh jumlah kalimat. Ini ditunjukkan gambar 4.78, dimana penggunaan CPU stabil pada 51%. Dokumen seluruh laporan menggunakan CPU yang paling besar yaitu 51,8%. Sedangkan dokumen Bab I-V menggunakan CPU yang paling kecil yaitu

51,1%. Hal ini juga menunjukan jumlah file tidak terlalu mempengaruhi penggunaan CPU.

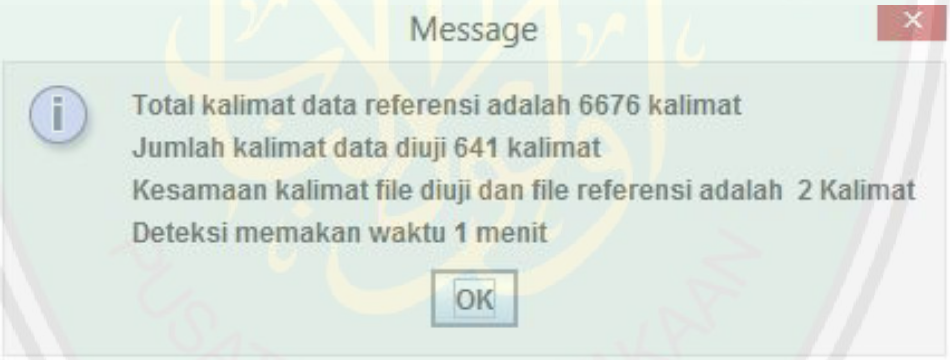
Perbandingan penggunan memori dalam mendeteksi ditunjukkan pada Gambar 4.79 sebagai berikut:



Gambar 4. 79 Grafik perbandingan penggunaan memori dalam melakukan deteksi

Terlihat pada gambar 4.79 dokumen seluruh laporan menggunakan memori paling besar dengan penggunaan 268 Mb. Sedangkan dokumen doc menggunakan memori paling sedikit dengan penggunaan 144. Dengan penggunaan memori rata-rata untuk mendeteksi 188,2 Mb. Jumlah kalimat pada dokumen yang diuji berpengaruh pada penggunaan memori. Dimana semakin banyak kalimat semakin banyak memori yang digunakan. Sedangkan Jumlah file tidak berpengaruh terhadap penggunaan memori. Terdapat perbedaan antara dokumen doc dan docx, namun tidak terlalu besar.

Dari data hasil pengujian didapat jumlah kalimat berpengaruh terhadap waktu yang dibutuhkan dan penggunaan memori untuk melakukan deteksi. Sedangkan penggunaan CPU tidak terlalu terpengaruh dengan penggunan berada diantar 51,1%-51,8%. Jenis file memiliki pengaruh yang tidak terlalu signifikan baik pada waktu yang dibutuhkan, penggunaan CPU maupun penggunaan memori dengan data dengan tujuh belas kalimat. Pada gambar 4.80 menunjukan pesan kesimpulan deteksi dan berfungsi juga sebagai pemberitahuan kepada pengguna bahwa proses deteksi telah selesai. Pada gambar 4.80 *sample* data yang digunakan adalah seluruh laporan. Pada sampel data ini terdapat dua kalimat yang sama persis. Waktu yang dibutuhkan pada saat gambar diambil adalah 60 detik.



Gambar 4. 80 Menampilkan kesimpulan deteksi

Waktu rata-rata yang digunakan untuk proses deteksi adalah 35,2 detik, penggunaan CPU adalah 51,34 % dan penggunaan memori sebesar 188,2 Mb. Sedangkan waktu yang dibutuhkan untuk mendeteksi satu kalimat adalah 0,12 detik. Pada gambar 4.78 menunjukkan hasil deteksi dengan *sample* seluruh dokumen. Terlihat pada gambar hasil deteksi terdapat kesamaan kalimat pada sampel data di kalimat ke 128 dengan dokumen referensi 07650021 pada kalimat ke 56. Dengan isi kalimat “BAB II Landasan teori berisikan beberapa teori yang mendasari dalam penyusunan tugas akhir ini”.

Gambar 4. 81 Jendele utama setelah proses deteksi selesai

BAB V

PENUTUP

5.1 Kesimpulan

Dari hasil implementasi dan uji coba yang telah dilakukan dapat ditarik kesimpulan sebagai berikut :

a. Algoritma Rabin-Karp dapat digunakan sebagai dasar membangun aplikasi deteksi plagiarisme.
b. Dari pengujian aplikasi yang mengandung *string-matching* dapat digunakan untuk mendeteksi plagiarisme.
c. Aplikasi ini mampu mendeteksi plagiarisme dalam waktu 0,12 detik untuk satu kalimat dengan sepuluh file referensi. Dengan menggunakan 51,34 % CPU dan memori sebesar 188,2 Mb.

5.2 Saran

Terdapat banyak kekurangan dalam penelitian ini. Beberapa saran untuk pengembangan penelitian ini agar lebih baik:

a. Memperbaiki pembacaan dokumen dengan menghilangkan karakter-karakter atau atribut-atribut yang ada di dalam dokumen namun tidak digunakan dalam proses deteksi. Misal gambar, tabel dll.
b. Memperbaiki pemisahan dokumen menjadi kalimat. Agar didapat kalimat yang baku untuk meningkatkan hasil deteksi.
c. Menambah ekstensi dokumen yang dideteksi. Agar lebih banyak variasi dokumen yang digunakan.

DAFTAR PUSTAKA

Gipp, Bela, Norman Meuschke. 2011. *Citation Pattern Matching Algorithms for Citation-based Plagiarism Detection: Greedy Citation Tiling, Citation Chunking and Longest Common Citation Sequence*. Proceeding of the 11th ACM Symposium on Document Engineering. Mountain View, CA, USA.

KBBI 1997, 775.

Kementerian Pendidikan Nasional Replubik Indonesia. 2010. *Peraturan Menteri Pendidikan Nasional Replubik Indonesia Nomor 17 tahun 2010: Pencegahan dan Penanggulangan Plagiat di Perguruan Tinggi*. Jakarta: Kementerian Pendidikan Nasional Replublik Indonesia

Novian, D., Abdillah, T., Tuloli, M.S., Yassin R.M.T. 2012. *Aplikasi Pendeteksian Plagiat Pada Karya Ilmiah Menggunakan Algoritma Rabin-Karp*. Gorontalo: Universitas Negeri Gorontalo

Nugroho, eko. 2011. *Perancangan Sistem Deteksi Plagiarisme Dokumen Teks Dengan Menggunakan Algoritma Rabin-Karp*. Malang: Universitas Brawijaya.

Priantara, I.W.S., Purwitasari, D., Yuhana U.L. 2011. *Implementasi Deteksi Penjiplakan Dengan Algoritma Winnowing Pada Dokumen Terkelompok*. Surabaya: Institut Sepuluh Nopember.

Salmuasih. 2013. *Perancangan sistem deteksi plagiat pada dokumen teks dengan konsep similarity menggunakan algoritma rabin karp*. Yogyakarta: Sekolah Tinggi Manajemen Informatika Dan Komputer Amikom Yogyakarta.

Sedgewick, Robert dan Wayne, Kevin. 2011. *Algorithms Fourth Edition*. New York: Addison-Wesley.

Suwarjo, Sugiyatno, Astuti B., Eliasa, E.I., Tjiptasari, F., Ratri, N., et al. 2012. *Identifikasi Bentuk Plagiat pada Skripsi Mahasiswa Fakultas Ilmu Pendidikan Universitas Negeri Yogyakarta*. Yogyakarta: Universitas Negeri Yogyakarta.

Zen, Muhammad. 2013. *Penerapan Algoritma String Matching untuk Mendeteksi Musik Plagiat*. Bandung: Institut Teknologi Bandung.

http://www.republika.co.id/berita/pendidikan/dunia-kampus/13/10/02/mu1irr-selama-setahun-100-dosen-jadi-plagiat, diunduh pada tanggal 24-2-2014 1.12 wib.

LAMPIRAN 1 Dokumen Referensi

No	NIM	JUDUL	Penulis
1	05550021	SISTEM INFORMASI GEOGRAFIS PEMELIHARAAN JALAN DINAS BINA MARGA KABUPATEN BLITAR (IMPLEMENTASI MODEL ANALYTICAL HIERARKHI PROCESS)	SUKRON ABADAN
2	05550027	PENGELOMPOKAN (*CLUSTERING*) BERITA BERBAHASA INDONESIA DENGAN MENGGUNAKAN *K-MEAN CLUSTERING*	AMIROH NAHDLIYAH
3	05550047	APLIKASI PENDETEKSI PLAGIARISME *SOURCE CODE JAVA* MENGGUNAKAN ALGORITMA *BRUTE FORCE*	DLOBITUL HASSIN
4	07650021	DESAIN DAN IMPLEMENTASI *COMPUTER ASSISTED INSTRUCTION* PADA PEMBELAJARAN ILMU FAROID MENGGUNAKAN METODE *DRILL AND PRACTICE*	RIQQOTUL BADRIYAH
5	07650067	RANCANG BANGUN APLIKASI PENYIRAMAN BERDASARKAN KANDUNGAN *BIO ELECTRIC POTENTIAL* PADA TANAMAN *CHRYSANTHEMUM* MENGGUNAKAN PEMROGRAMAN DELPHI	MUCHAMMAD NURIL HUDA
6	07650081	RANCANG BANGUN PENJADWALAN SUMBER DAYA (*RESOURCE*) DALAM KOMPUTASI PARALEL DENGAN METODE *TIME OPTIMIZATION* GUNA MENDISTRIBUSIKAN TUGAS (*TASK*) BERBASIS *MULTIAGENT*	NISA MIFTACHUROHMAH
7	07650099	RANCANG BANGUN APLIKASI PENYIRAMAN BERDASARKAN KANDUNGAN *BIO ELECTRIC POTENTIAL* PADA MEDIA TANAH MENGGUNAKAN PEMROGRAMAN DELPHI	M. ZULFIKAR ALI WAVA
8	07650114	SISTEM PENDUKUNG KEPUTUSAN PENERIMAAN MANAGEMENT TRAINEE MENGGUNAKAN METODE FUZZY *ANALYTICAL HIERARCHY PROCESS* (Studi Kasus Pada PT. Beiersdorf Indonesia)	RIZSANDY YUDHA PRASETIA
9	09650054	APLIKASI *CHATBOT* "MI3" UNTUK INFORMASI JURUSAN TEKNIK INFORMATIKA BERBASIS SISTEM PAKAR MENGGUNAKAN METODE *FORWARD CHAINING*	ZIFORA NUR BAITI
10	09650087	PENINGKATAN KUALITAS CITRA *DENTAL PANORAMIC RADIOGRAPH* PADA TULANG MANDIBULA MENGGUNAKAN *MULTI-HISTOGRAM EQUALIZATION*	IDA FITRIANA

LAMPIRAN 2 Isi Dokumen *Sample* doc dan docx

Scripts atau bahasa program adalah bahasa yang digunakan untuk menerjemahkan setiap perintah dalam situs yang pada saat diakses. Jenis *scripts* sangat menentukan statis, dinamis atau interaktifnya sebuah situs. Semakin banyak ragam *scripts* yang digunakan maka akan terlihat situs semakin dinamis, dan interaktif serta terlihat bagus. Bagusnya situs dapat terlihat dengan tanggapan pengunjung serta *frekwensi* kunjungan.

Diagram konteks merupakan kasus khusus DFD (*Data Flow Diagram*) atau bagian dari DFD yangberfungsi memetakan model lingkungan, yang direpresentasikan dengan lingkaran tunggal yang mewakili keseluruhan sistem.

Diagram konteks menyoroti sejumlah karakteristik penting sistem.

Sistem informasi pemasaran adalah struktur interaksi yang terus-menerus dari orang, perlengkapan, dan prosedur untuk mengumpulkan, memilih, menganalisis, mengevaluasi, dan mendistribusikan informasi yang relevan, tepat waktu dan cermat yang akan digunakan oleh para pengambil keputusan pemasaran untuk meningkatkan perencanaan, pelaksanaan, dan pengendalian pemasaran.

Tulang mulai terbentuk sejak bayi dalam kandungan dan kemudian berlangsung terus sampai dekade kedua dalam susunan yang teratur. Organ ini merupakan organ yang mendukung struktur tubuh.

Tulang mandibula merupakan tulang rahang yang umum menerima benturan, baik yang sengaja maupun tidak sengaja.

Algoritma yang digunakan dalam memeriksa sebuah kalimat sangat banyak macamnya, salah satunya algoritma left corner parsing. Algoritma left corner parsing merupakan gabungan dari dua algoritma, yaitu algoritma top down parsing dan bottom up parsing. Tugas algoritma ini memeriksa setiap kata dalam sebuah kalimat, kemudian mencocokkan sebuah pola grammar terhadap hasil pemeriksaan tersebut. Proses pemeriksaan ini yang akan menjadi pemeriksa kata dalam game.

Awalnya teknik yang dipakai bukan mencari fitur melainkan berdasarkan penambahan deskripsi mengenai citra dalam bentuk teks. Dengan kata lain pertama citra diberi teks berdasarkan citra tersebut kemudian dilakukan pencarian berdasarkan teks (*text based*) menggunakan sistem database manajemen tradisional. Namun karena pembangkitan teks secara otomatis, mengenai deskripsi spektrum citra, secara detail sulit untuk dilakukan kebanyakan aplikasi *text based image retrieval* saat itu melakukan pemberian teks deskripsi citra secara manual.

LAMPIRAN 3 Dokumen Sample

No	NIM	Judul	Penulis
1	Satu Bab	PENGEMBANGAN SISTEM INFORMASI LEMBAGA PENDIDIKAN MENTAL ARITMATIKA SEMPOA ADIL SEMPOA MANDIRI (ASMA) CABANG MALANG	RIZTHA YUNNISA HIDAYATI
2	Bab I-V	PENGEMBANGAN SISTEM INFORMASI LEMBAGA PENDIDIKAN MENTAL ARITMATIKA SEMPOA ADIL SEMPOA MANDIRI (ASMA) CABANG MALANG	RIZTHA YUNNISA HIDAYATI
3	Seluruh Laporan	*CONTENT BASED IMAGE RETRIEVAL* CITRA SIDIK JARI MENGGUNAKAN METODE *WAVELET DAUBECHIES* DAN DIAGRAM *NOHIS-Tree*	AGUNG SATRIO BUWONO