DETEKSI SIMILARITY SOURCE CODE MENGGUNAKAN METODE DETEKSI ABSTRACT SYNTAX TREE

Eka Budhy Prasetya
Eka.budhy@ymail.com
Universitas Muhammadiyah Jakarta

Ahmad Fadly Dzil Jalal
fadlycute31@gmail.com
Universitas Muhammadiyah Jakarta

ABSTRAK

Laboratorium Fakultas Teknik Infomatika Universitas Muhammadiyah Jakarta (FT–Informatika UMJ) sebagai tempat pembelajaran bagi para mahasiswa informatika yang mengikuti kelas pemrograman selalu memberikan tugas-tugas sebagai salah satu media pengukur tingkat pemahaman mahasiswa. Banyaknya tugas source *code* menggunakan bahasa Java yang harus diperiksa oleh Assisten Laboratorium mengakibatkan sulitnya melakukan pemeriksaan apabila dilakukan satu per satu serta sulitnya mengukur kredibilitas masing-masing tugas milik mahasiswa. Tugas-tugas terperiksa yang memiliki tingkat *similarity* (kemiripan) yang cukup tinggi antar *code* dapat dijadikan acuan adanya tindakan-tindakan kecuranganseperti melakukan tindakan plagiat code terhadap tugas mahasiswa lain. Metode deteksi kemiripan *code* menggunakan *Abstract Syntax Tree* dapat digunakan untuk merubah *code* menjadi *node* ataupun *token* unik masing-masing *code* terperiksa. Semakin besar kemiripan maka semakin besar kemungkinan *code* tersebut merupakan hasil *plagiat*. Aplikasi *Java's Source Code Similarity Detector* (JSC-SD) yang diusulkan akan mendeteksi kemiripan code melalui beberapa proses, yaitu proses parsing code menjadi AST yang kemudian akan diukur kemiripan tingkat kemiripannya menggunakan algoritma Levenstehin Distance dan Smith-Waterman dan pada proses terakhir adalah pendeteksian code clone dari source code terperiksa. Hasil akhir yang didapat adalah grafik persentase kemiripan antar code serta line code yang dicurigai similar.

Kata kunci : Laboratorium FT–Informatika UMJ, *Similarity*, *Abstract Syntax Tree*, *Levensthein Distance*, *Smith-Waterman*

I. Pendahuluan

Dalam ilmu komputer, source code adalah kumpulan pernyataan atau deklarasi bahasa pemrogramman komputer yang ditulis dan dapat di baca manusia. Source code memungkinkan programmer untuk berkomunikasi dengan komputer menggunakan beberapa perintah yang telah terdefinisi. Source code merupakan sebuah program yang biasanya dibuat dalam satu atau lebih file teks, kadang-kadang disimpan dalam database yang disimpan sebagai prosedur dan dapat juga muncul sebagai potongan kode yang tercetak di buku atau media lainnya. Banyaknya koleksi file source code dapat diatur dalam direktori pohon, dalam hal ini juga dikenal sebagai source tree.[1]

Plagiarisme merupakan tindakan penipuan hasil karya orang lain tanpa sepengetahuan dari penulis aslinya, yang melanggar suatu Hak Cipta dan Hak Moral. Ketertarikan mahasiswa terhadap tindakan plagiarisme, dibangun oleh rasionalitas instrumental. Mahasiswa lebih memperhitungkan tentang efisiensi, efektifitas dan nilai yang dimiliki oleh sumber dayanya (tugas akademik) untuk mencapai tujuan yang diharapkan. Ketika aktor (mahasiswa) menentukan tujuan, aktor akan dihadapkan pada sebuah pilihan cara alternatif yaitu cara SKS (Sistem Kebut Semalam) dan SKJ (Sekali Kerja Jadi). Pilihan tersebut akan memunculkan suatu bentuk tindakan plagiarisme dan konsekuensi dari tindakan plagiarisme. [2]

Masalah similarity (kemiripan) source code dapat menjadi salah satu solusi pendekatan deteksi tindakan plagiarisme. Adanya indikasi similarity harus di cermati secara seksama karena perlu penafsiran dan pembahasan khusus untuk memberikan keputusan seberapa besar tingkat kemiripan suatu source code dengan source code lainnya. Besarnya tingkat

kemiripan tersebut dapat digunakan sebagai salah satu dasar untuk mempertimbangkan suatu source code melakukan tindakan plagiarisme atau tidak.Banyak jenis similarity pada source code yang sering dijumpai pada tugas-tugas pemrograman yang diberikan kepada mahasiswa oleh para pengajar. Deteksi similarity bisa dilihat dari penggantian nama source code, memindahkan posisi beberapa potongan code serta mengganti nama variabel maupun class, method ataupun fungsinya.

Code Plagiarism adalah istilah yang digunakan untuk mendeskripsikan adanya kemiripan isi source code antar 2 code yang berbeda. Similarity source code dapat terjadi secara keseluruhan code maupun sebagian, serta dengan sedikit perubahan pada bagian yang tidak signifikan. Terdapat 3 tipe kemiripan textual yang terjadi pada dua buah source code.[3]

1. Type 1
   Fragmen yang di-copy dari file original secara sama persis, kecuali adanya perbedaan– perbedaan seperti whitespaces (spasi atau enter yang tidak mempunyai arti), comment, dan modifikasi pada jumlah baris. Tipe ini disebut juga sebagai exact clone.
2. Type II
   Type ini sama seperti Type I, perbedaannya adalah pada Type II, terdapat modifikasi pada variable atau fungsi. Modifikasi dapat berupa penggantian nama atau perubahan tipe data.
3. Type III
   Type III merupakan kombinasi dari Type I dan Type II. Plagiarisme file dilakukan dengan cara menambahkan statement yang tidak penting.

II. Penelitian Deteksi Similarity Source Code yang Berhubungan

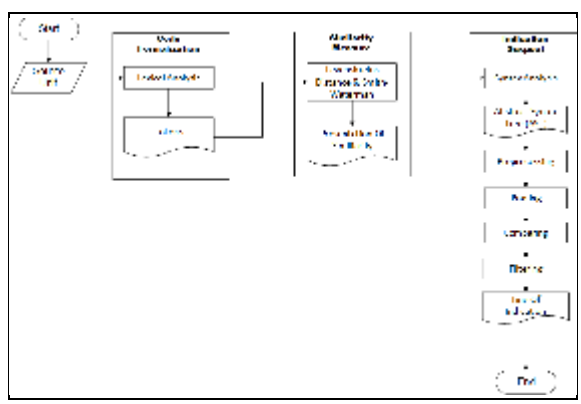
Li Ping Zhang dan Dong Sheng Liu di Inner Mongolia Normal University, College of Computer and Information Engineering, Inner Mongolia, Hohhot, China[4]mengusulkan metode deteksi plagiat dalam tiga tahap yaitu code formalization, similarity calculation dan cluster anlysis. Penjelasan Tahap menurut keduanya adalah sebagai berikut.

1. Code Formalization
   Code yang akan diperiksa akan dirubah menjadi sebuah tree menggunakan ANTLR (Another Tool for Language Recognition). ANTLR ditulis menggunakan JAVA yang digunakan untuk meng-generate parser dan lexer dari sebuah grammar yang digunakan untuk membaca, memproses, dan mengeksekusi atau menerjemahkan teks terstruktur atau file binary menjadi sebuah AST.
2. Similarity Calculation
   Pengukuran kemiripan code didasarkan pada AST dari source code kemudian node-node yang tercipta dibuat menjadi sebuah sequence yang akan dihitung tingkat plagiatnya menggunakan deteksi kemiripan berdasarkan matching sequence
3. Cluster Analysis
   Setelah similarity calculation , dapat ditentukan corresponding antara sequence di dalam tabel sequence melalui nomor baris dan vektor khas untuk tahap cluster analysis. Tahap cluster analysis akan menggunakan VSM *(Vector Space Model).*

III. Sistem yang diajukan

Berdasarkan dari karaktertistik pembelajaran yang ada di Laboratorium Fakultas Teknik Informatika Universitas Muhammadiyah Jakarta yang terdiri dari beberapa jenis regulasi kelas yaitu reguler pagi, reguler sore dan perkuliahan karyawan, JSC-SD harus mampu mendeteksi kemiripan-kemiripan code pada bahasa pemrograman Java yang diajarkan pada beberapa mata kuliah seperti Pemrograman Berbasis Objek dan Struktur Bahasa Pemrograman pada masing masing jenis regulasi.

JSC-SDmengadopsi sistem yang sudah diajukan oleh Li Ping Zhang dan Dong Sheng Liu yang diterapkan ke dalam satu aplikasi berbasis desktop. JSC_SD adalah sebuah sistem yang akan menemukan kemiripan codeserta menghitung besar kemiripan antar code yang dilakukan dalam tiga proses utama, yaitu:



Gambar 1. Main Flow JSC-SD

1. Code Formalization

Sesuaidengannamanya,ASTmerupakanstrukturdata*intermediate* didalamprosedur kompilasi

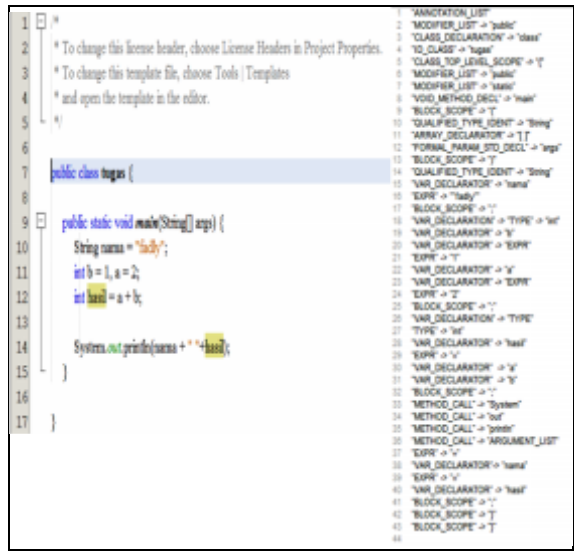
danprosesinterpretasiyangdimanasyntax*tree* tidaksajamerefleksikaninformasi terstrukturtetapijugaberisiatribut-atribut dari*source*program. Dengancaraini, akan diperolehinformasikomperhensif yanglebihakuratuntukmendeteksikemiripan*code* yangada. Adabanyak carauntukmemperolehataumeng-*generate*ASTyang ada.Salah satucara adalahmenggunakanANTLR*method*.[5]

Proses untuk memperolehASTolehANTLR dapat dibagi dalamdua subprosesutama.Pertama,ketikaparse*file*sedang membaca,ANTLRakan meng-generate *corresponding lexical* dan *syntax analyzer* berdasarkan *rule- rule* yang ada pada analisis *grammar file*. Proses kedua adalah kode-kode input di konversi menjadi *phrase flow* dengan menggunakan *lexical analyzer* yang *degenerate* oleh ANTLR sehingga input dari parser adalah *phrase flow* yang akan dirubah menjadi AST melalui proses *parsing* oleh parser.

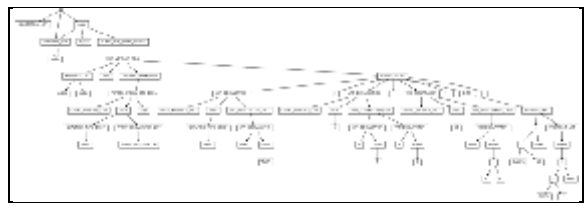
Keuntungan menggnunakan AST sebagai metode pendekatan perbandigan code adalah pada proses ini akan menghilangkan noise pada source code. Noise adalah jenis karakter yang tidak mempengaruhi code, karena jenis karakter ini tidak akan dieksekusi oleh compiler seperti comment dan whitespace. Gambar II akan memperlihatkan bagaimana membuat sebuah Abstract Syntac tree dari sebuah deklarasi code.

2. Similarity Measure

Hasil proses*parsing*dari parserANTLR berupanode atau tokenunikyang membentuk suaturangkaiansequenceyangbermaknasecara strukutral.Tokenmerupakanhasil pengkategoriandari setiapkataataukarakter yang terdapatpada*sourcecode* dimana setiap token terdefinisi dengan ID khas hasil generate lexer dan parser oleh ANTLR. Contoh dapat dilihat pada Gambar I dibawah ini



Gambar 1 Node pada code



Gambar 2. Contoh Tree Code Gambar II

TokenLaw	I	Toke
PUBLIC	87	public
CLASS	61	class
IDENT	16	tugas,main,args,nama
STATIC	90	static
VOID	10	void
LPAREN	29	‘(’,
LBRACK	22	‘[’
RBRACK	41	‘]’
RPAREN	43	‘)’
LCURLY	23	‘{‘,
STRING_LITERA	17	Fadly
SEMI	44	‘;’
INT	79	Int

Gambar 3. ID Unik Dari Node / Token

Dari proses code formalization dapat ditentukan sebuah sequence unik hasil representasi code tersebut. Pada Gambar 2, sequence yang terbentuk adalah:

876116423879010116429164224116443231641646170447916461671116461674479164616438164441641516415164291643817038164434 44242.

Besar kemiripan dapat dihitung berdasarkan sequence yang terbentuk dari rangkaian sequence pada gambar 3

menggunakan dua algoritma matching sequence biological computance yaitu algoritma Levensthein Distance dan algoritma Smith-Waterman [6]. Kedua algoritma ini menggunakan cara yang sama yaitu membuat sebuah matriks yang terbuat dari dua string sequence dimana masing-masing menjadi koordinat vertikal dan horizontal yang kemudian akan melakukan perulangan perbandingan antar karakter koordinat vertikal dan horizontal untuk mengisi kolom pertemuan vertikal dan horizontal. Berikut ini penjelasan masing-masing pendekatan algoritma.

a) Levensthein Distance

Untuk menggunakan algoritma ini terdapat dua tahap utama, yaitu

- Sequence Matching

Untuk dua sequence yang diberikan dimana A = A_1, A_2, A_3 , .., A_n dan B = B_1, B_2,B_3 , .., B_n , Levensthein distance bekerja sesuai dengan prinsip dynamic programming maka akan dibuat sebuah matriks X[panjangA][panjangB]. Perhitungan matriks dilakukan dengan ketentuan sebagai berikut.

1) Inisialisasi matriks X[i][0] = i dan X[0][j] = 0.
2) Proses identifikasi melalui rekursif kondisi
   X[i][j] = minimum{d[i-1, j] +1, d[i, j-1] +1, d[i-1, j-1] +cost) }

Berikut ini adalah pseude code dari algoritma Levensthein Distance.

```
int LevenshteinDistance (char s[1…m], char t[1…n])
// d is a table with m+1 rows and n+1 columns
   declare int d[0…m, 0…n]
   for i from 0 to m
 d[i, 0] := i
         for j from 0 to n
             d[0, j] := j
         for i from 1 to m
         for j from 1 to n
           if s[i] = t[j] then cost := 0
              else cost := 1
                 d[I, j] := minimum(
                   d[i-1, j] +1, // deletion
                   d[i, j-1] +1, //insertion
                   d[i-1,j-1]                    +cost
         //substitution)
         return d[m, n]
```

Gambar 4. Pseudo Code Levensthein

- Similarity Calculation

Rumus umum menghitung besar kemiripan code berdasarkan algoritam Levensthein distance adalah sebagai berikut:

$$Sim(m,n) = 1 \quad \left(\frac{Dis}{MaxLength}\right)$$

Gambar 5. Rumus Umum Levensthein

Keterangan:

Sim(m, n)	=	Tingkat kemiripan antara string m dan string n
Dis	=	Nilai Levensthein Distance (nilai akhir perbandingan string m dan string n yang ada pada pojok kanan bawah matriks)
MaxLength	=	JumlahKarakterMaksimumDua String

b) Smith Waterman

Algoritma Smith-Waterman menggunakan pendekatan berbasis local aligment yang berarti posisi ordo matriks yang dijadikan acuan bersifat bebas atau bisa berada pada ordo matriks dimana saja. Terdapat dua tahap utama yaitu:

- Sequence Matching

Untuk dua sequence yang diberikan dimana A = A_1, A_2, A_3 , .., A_n dan B = B_1, B_2,B_3 , .., B_n , Levensthein distance bekerja sesuai dengan prinsip dynamic programming maka akan dibuat sebuah matriks X[panjangA][panjangB]. Perhitungan matriks dilakukan dengan ketentuan sebagai berikut.

1) Inisialisasi matriks X[i][0] = X[0][j] = 0
2) Proses identifikasi melalui rekursif kondisi :
   X[i][j] = max{0, X[i-1][j-1] + Nilai Match/MisMatch, X[i-1][j]– Nilai GAP, X[i][j-1] + nilai GAP}

Berikut ini adalah pseudo code dari algoritma Smith-Waterman.

```
//gap = o + (l-1)*e;
//o: gap opening penalty (o < 0)
//l: length of the gap
//e: gap extension penalty (o < e < 0)

for(i=LengthA; j=LengthB; i>0 && j>0){
   if (i != 0 && j != 0) {
      if (one.charAt(i) == two.charAt(j)) {
          matrix[i][j]  =  Math.max(0,
          Math.max(matrix[i - 1][j - 1] + match,
          Math.max(matrix[i  -  1][j]  +
          gap,matrix[i][j - 1] + gap)));
```

```
        }

        else {
          l++;
        matrix[i][j]=Math.max(0,Math.max(m
        atrix[i-1][j-1]+ gap,
        Math.max(matrix[i - 1][j] + gap,
        matrix[i][j - 1] + gap)));
          }
        }
      }
    }
}
```

Gambar 6. Pseudo Code Levensthein

- Similarity Calculation
  Rumus umum menghitung besar kemiripan code berdasarkan algoritma Smith-Waterman adalah sebagai berikut.

$$Sim(m,n) = 2 * \left(\frac{MaxLength(m,n)}{m.length() + n.length()}\right)$$

Gambar 7. Rumus Umum Levensthein

Keterangan:

Sim (m,n)	=	Tingkat kemiripan antara string m dan string n
MaxLength (m,n)	=	Jumlah karakter yang sama antara string m dan string n
m.length	=	Panjang string m
n.length	=	Panjang string n

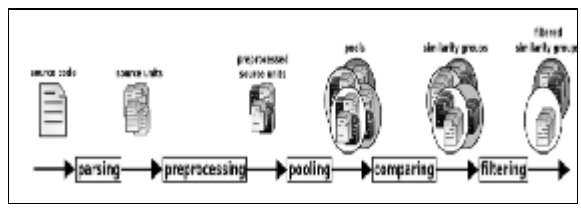
3. Indication Suspect

Dalam proses pencarian kemiripan *code* tidak cukup hanya menghitung dari segi token-token uniknya saja, karena apabila terjadi pemindahan beberapa posisi potongan *code*, maka rangkaian ID token yang unik tadi juga ikut berubah. Hal ini memungkinkan *code* yang sama tetapi hanya memindahkan potongan *code* untuk menyamarkan kemiripan menjadi tidak terdeteksi dengan baik. Proses ini akan menemukan letak kemiripan antar code yang memerlukan beberapa subproses.

Aplikasi JSC-SD mengadaptasi API yang dikembangkan oleh Biegl dan Diehl[7]bernama JCCD untuk memnemukan line code yang terduga adalah code clone. JCCD membentuk sebuah model pipeline yang terdiri dari proses berikut.

a) Preprocessing
   Tujuan utama dari preprocessing ini dibagi menjadi dua, yaitu menormalisasikan source code dan menambahkan annotation tambahan. Sebuah preprocessor yang dibuat dapat menjadi sebuah annotate (keterangan), remove (menghapus), dan collapse (menyempitkan) serta sebagai grouping node-node AST.
b) Pooling
   Tahap pooling merupakan langkah dalam preselection dari kandidat-kandidat potongan code yang dicurigai similar (code clone). Source unit yang belum diproses akan di satukan ke dalam sebuah group kedalam set-set yang berbeda yang dinamakan pool berdasarkan masing-masing kriteria yang sudah ditentukan.
c) Comparing
   Semua pool yang terbentuk dapat diproses secara sequential dengan cara membandingkan seluruh isi source unit secara berulang. Setiap comparator-comparator menentukan apakah dua subtree memenuhi karakteristik-karakteristik tertentu. Dua subtree dikatakan similarity pair hanya jika semua kombinasi comparator- comparator bernilai true.
d) Filtering
   Langkah terakhir adalah menghapus non-relevant similarity pair's sehingga didapatkan sebagai satu set hasil yang sudah dapat disimpulkan. Hasil similiairty pairs yang terindikasi inilah yang disebut dengan *code clone*.

Pada Gambar 8 adalah sebuah model generic pipeline secara keseluruhan dalam melakukan proses indication suspect.



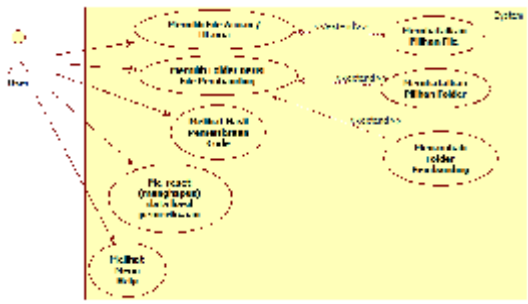
Gambar 8. Pipeline Indication Suspect

IV. JSC-SD Prototype – Sistem akan melibatkan dua tipe user yaitu: Assisten Lab dan Mahasiswa. Sistem akan dirancang dengan sifat-sifat sebagai berikut:

1. Tugas yang diperiksa oleh model sistem dapat per jenis regulasi kelas maupun keseluruhan regulasi kelas,
2. Source code yang akan diperiksa cukup diletakkan pada satu direktori pada local drive komputer / laptop pengguna,
3. Assisten sebagai user utama dapat memilih secara bebas satu file tugas mahasiswa sebagai acuan yang akan dibandingkan dengan file mahasiswa lain

4. Assisten dapat menambahkan beberapa file-file pembanding dari beberapa direktori berisi file-file java.

Berikut adalah use case hasil representase sifat-sifat prototype yang dirumuskan sebelumnya.



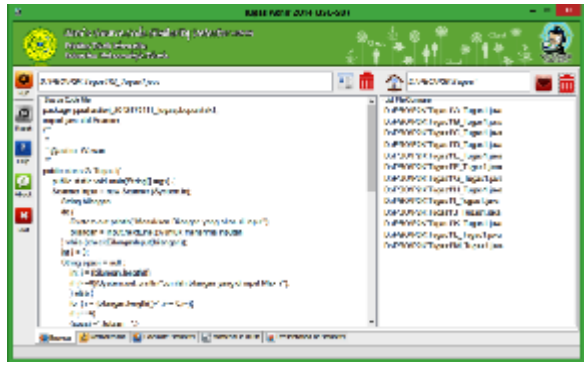
Gambar 9. Use case JSC-SD

V. Implementasi

JSC –SD telah diuji coba untuk memeriksa beberapa sampel code yang diambil dari tugas mata kuliah praktikum penunjang PBO di kelas Perkuliahan Karyawan Semester Genap 2014. Materi tugas yang diberikan oleh Assisten Lab adalah sebagai berikut.

"Buatlah sebuah program yang dapat menerima input dari keyboard berupa barisan bilangan dimana barisan bilangan tersebut mampu terdefinisikan derajat digit-nya. Sebagai contoh, jika terdapat bilangan 1234567, maka program Anda harus dapat mendefinisikan bahwa:
"1 adalah bilangan jutaan,
2 adalah bilangan ratusan-ribuan,
3 adalah bilangan puluh-ribuan,
4 adalah bilangan ribuan,
5 adalah bilangan ratusan,
6 adalah bilangan puluhan,
7 adalah bilangan satuan. "

Sebanyak 13 mahasiswa mengumpulkan tugas yang dimaksud yang kemudian akan dijadikan sebagai dataset sampel. Berikut proses peng-inputannya tugas tersebut ke dalam JSC-SD.



Gambar 10. Input Dataset Sampel

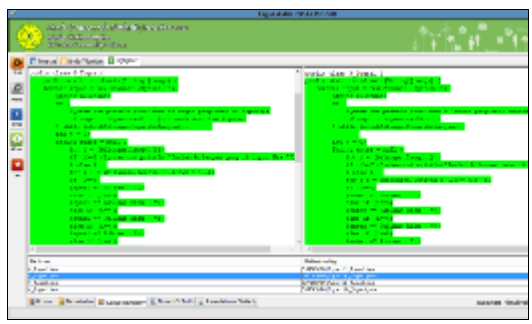
JSC-SD memiliki dua tipe output utama yaitu:

1. Highlight Code Clone

Pada tahap ini akan diambil beberapa sampel yang akan diperiksa kemiripan-kemiripan code yang berhasil terdeteksi. Hasil pengujian adalah kemiripan code antara satu code acuan dengan 3 code pembanding yang diambil secara random.

a. G_Tugas1.java dengan K_Tugas1.java

Pada sampel pertama, sistem mendeteksi terjadi kemiripan source code di seluruh bagian source code baik milik code acuan maupun pada code pembanding. Hanya terdapat perbedaan pemberian beberapa nama variabel saja seperti "String spasi = null" pada G_Tugas1.java dan "String enter = null" pada J_Tugas1.java. Kemiripan ditandai dengan baris code yang diberi warna hijau. Berdasarkan jenis type plagiarisme maka pada sampel ini dapat dikategorikan dalam jenis Type II.



Gambar 11. Sampel 1

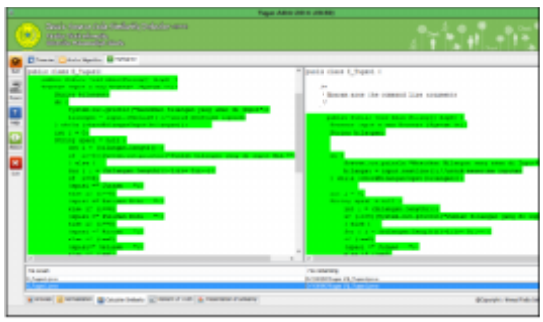
b. G_Tugas1.java dengan E_Tugas1.java

Pada sampel kedua, sistem mendeteksi terjadi kemiripan source code di hampir seluruh bagian tubuh source code baik milik source code acuan maupun pada source code pembanding. Kedua source code mempunyai struktur dan beberapa nama variabel yang sama. Hanya terdapat perbedaan pemberian komentar untuk

menyamarkan pada awal code milik code pembanding yaitu:

```
/**
 * @param args the command line arguments
 */
```

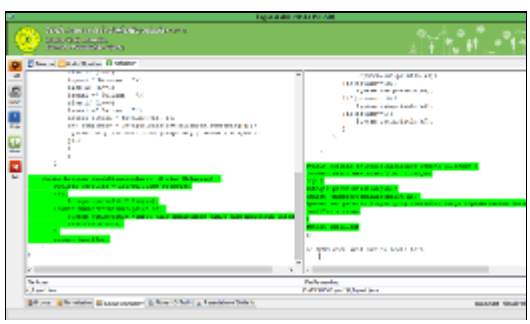
Berdasarkan jenis type plagiarisme maka pada sampel ini dapat dikategorikan dalam jenis Type I.



Gambar 12. Sampel 2

c. G_Tugas1.java dengan B_Tugas1.java

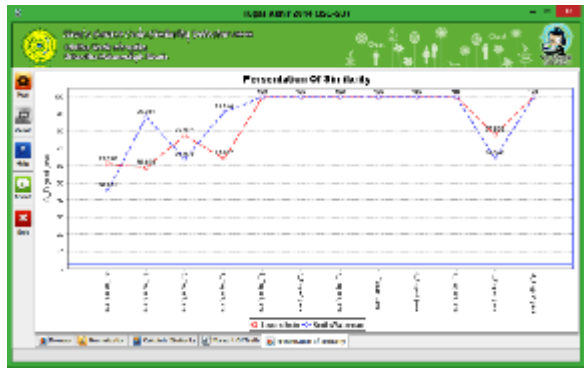
Pada sampel ketiga, kemiripan code hanya terjadi di sebagian tubuh source code. Source code acuan dan source code pembanding sama-sama memiliki method yang sama, baik nama variabel maupun nama method itu sendiri. Kemiripan disamarkan dengan penghilangan spasi atau tab-character pada method yang dimaksud milik source code acuan oleh source code pembanding. Berdasarkan jenis type plagiarisme maka pada sampel ini dapat dikategorikan dalam jenis Type III.



Gambar 13. Sampel 3

2. Presentase Kemiripan

Semakin besar persentase kemiripan antar code tersebut, maka akan semakin besar pula indikasi bahwa diantara kedua tugas yang dibandingkan merupakan code hasil tindakan plagiat. Berikut ini adalah hasil pemeriksaan dari sebuah tugas code milik seorang mahasiswa yang dijadikan sebagai code acuan dengan tugas code pembanding milik teman mahasiswa yang dijadikan code acuan dalam bentuk grafik.



Gambar 14. Persentase Kemiripan Antar Code

VI. Kesimpulan

Pada paper ini, AST digunakan sebagai representase lain dari sebuah code untuk memeriksa kemiripan antar code yang menggunakan bahasa pemrograman Java. Pendeteksian kemiripan (similarity) code menggunakan metode AST dapat digunakan sebagai acuan mendeteksi tindakan plagiat. Semakin besar tingkat kemiripan code maka dapat menjadi salah satu acuan apakah code tersebut merupakan tindakan plagiat.

Algoritma Levensthein Distance dan Smith-Waterman pada dasarnya menggunakan pendekatan yang sama, tetapi Smith Waterman lebih baik karena: Algoritma Smith Waterman lebih leluasa karena Smith-Waterman memeriksa seluruh sel kolom dan baris pada matriks sampai akhir sehingga dapat ditentukan best score dari sebuah local aligment, bukan hanya sekedar menghitung nilai akhir kolom dan baris matriks.

Dalam proyek selanjutnya, JSC_SD sebaiknya meningkatkan performa dengan cara: (1) Penyediaan grammar bahasa pemrograman yang lebih baik serta grammar yang lebih beragam dari beberapa jenis bahasa pemrograman. (2) Peningkatan performa *indication suspect* dengan cara "*concatenation project*" dimana proses ini akan meningkatkan kecepatan *load processing* dengan cara memfokuskan pada bagian terkecil dari potongan file source code sebelum memfokuskan pada bagian potongan code yang lebih besar dari source code didalam sebuah single pipeline. [8]

V. Daftar Pustaka

[1] Pengertian Source code. http://id.wikipedia.org/wiki/Kode_sumber. (diakses 1 September 2014)

[2] Imroatullayyin Makhfiyana, Moh. Mudzakkir, Rasionalitas Plagiarisme Di Kalangan Mahasiswa Fakultas Ilmu Sosial Unesa, (http://ejournal.unesa.ac.id/index.php/paradigma/article/view/3998). (Diakses 1 September 2014)

[3] Liliana, Gregorious Satia Budhi, Anthony Wibisono, Ricky Tanojo. Pengecekan Plagiarisme Pada Code Dalam Bahasa C++ 2012: (70-71)

[4] Li ping Zhang, Dong Sheng Liu. IEEE Journal: AST-based Multi-language Plagiarism Detection Method, Journal AST-based Multi-language Plagiarism Detection Method, Inner Mongolia Normal University, College of Computer and Information Engineering, Inner Mongolia, Hohhot, China, 2013 (738-741)

[5] Terrence Parr. Pragmatic The Definitive ANTLR Reference May 2012.

[6] Zhan Su, Byung Ryul Ahn. Plagiarism Detection Using the Levenshtein Distance and Smith-Waterman Algorithm, Department of Artificial Intelligence, University of Sungkyunkwan Cheoncheon dong, Jangan-gu, Suwon, Korea 2008

[7] Benjamin Biegel, Stephan Diehl. Highly Configurable and Extensible Code Clone Detection. 17th Working Conference on Reverse Engineering, WCRE 2010.

[8] Al-Fahim Mubarak Ali, Shahida Sulaiman, Sharifah Mashita Syed-Mohamad. An Enhanced Generic Pipeline Model for Code Clone Detection. Journal School of Computer Science, Universiti Sains Malaysia, 2012