BAB II

TELAAH LITERATUR

2.1 Rekayasa Perangkat Lunak

Dalam membangun sebuah perangkat lunak (aplikasi komputer) yang baik, dibutuhkan sebuah teknologi, yang meliputi proses, serangkaian metode, dan sederetan alat yang dikenal dengan rekayasa perangkat lunak. Fritz Bauner (Pressman, 1997, p.53) mengusulkan definisi rekayasa perangkat lunak sebagai "*the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently in real machine*".

IEEE sendiri dalam *Standard Collections*-nya (1993) telah mengembangkan definisi rekayasa perangkat lunak secara lebih komprehensif, yakni "Aplikasi dari sebuah pendekatan kuantifiabel, disiplin, dan sistematis kepada pengembangan, operasi, dan pemeliharaan perangkat lunak; yaitu aplikasi dari Rekayasa Perangkat Lunak".

Sedangkan Mulyanto (2008, p.2) mendeskripsikan rekayasa perangkat lunak sebagai suatu disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal yaitu analisa kebutuhan pengguna, menentukan spesifikasi dari kebutuhan pengguna, disain, pengkodean, pengujian sampai pemeliharaan sistem setelah digunakan.

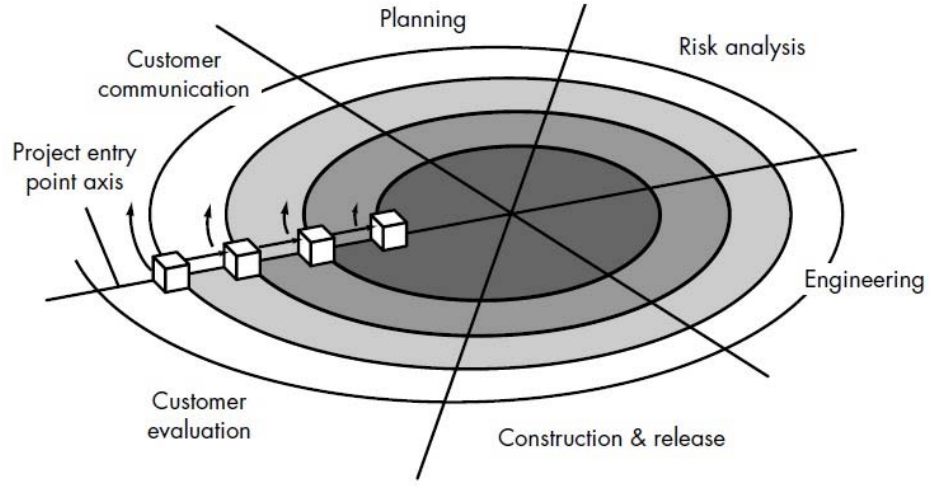
Tujuan dari rekayasa perangkat lunak sendiri adalah memperoleh biaya produksi perangkat lunak yang rendah; menghasilkan perangkat lunak yang

kinerjanya tinggi, andal, dan tepat waktu; menghasilkan perangkat lunak yang dapat bekerja pada berbagai jenis *platform*; serta menghasilkan perangkat lunak yang biasa perawatannya rendah.

Selanjutnya, Pressman (1997, p.78) mengungkapkan untuk menyelesaikan suatu masalah pembangunan perangkat lunak, pelaku harus menggabungkan strategi pengembangan yang melingkupi lapisan proses, metode, dan alat-alat bantu. Startegi ini sering diacukan sebagai model proses atau paradigma rekayasa perangkat lunak. Model proses yang dipilih didasarkan pada sifat aplikasi dan proyeknya, metode dan alat-alat bantu yang dipakai, serta kontrol dan penyampaian yang dibutuhkan.

Salah satu metode yang dapat digunakan adalah model spiral (*spiral model*). Model yang pada awalnya diusulkan oleh Boehm (1988, pp.61-72) merupakan model proses perangkat lunak yang evolusioner, yang merangkai sifat iteratif dari prototipe dengan cara kontrol dan aspek sistematis dari model sekuensial linier.

Model spiral dibagi menjadi sejumlah aktifitas kerangka kerja, yang terdiri dari: komunikasi dengan pelanggan (*costumer communication*), perencanaan (*planning*), analisis resiko (*risk analysis*), perekayasaan (*engineering*), konstruksi dan perilisan (*construct & release*), serta evaluasi pelanggan (*costumer evaluation*). Untuk lebih jelas, dapat dilihat pada Gambar 2.1.



Gambar 2.1 Model Spiral Rekayasa Perangkat Lunak (Pressman, 1997)

Ketika proses ini mulai, pelaku bergerak searah jarum mengelilingi spiral tersebut, dimulai dari intinya. Lintasan pertama putaran spiral menghasilkan perkembangan dari spesifikasi produk; putaran spiral berikutnya dipakai untuk mengembangkan sebuah prototipe, dan selanjutnya secara progresif mengembangkan dan menghasilkan perangkat lunak yang lebih canggih.

2.2 Plagiarisme

Dalam Kamus Besar Bahasa Indonesia, plagiarisme atau sering disebut plagiat didefinisikan sebagai penjiplakan atau pengambilan karangan, pendapat, dan sebagainya dari orang lain dan menjadikannya seolah karangan dan pendapat sendiri. Serupa dengan KBBI, situs *Dictionary.com* mendefinisikan plagiarisme sebagai "(1) *the unauthorized use or close imitation of the language and thoughts of another author and the representation of them as one's own original work; (2) something used and represented in this manner*".

Sedangkan Brotowidjoyo (1993, p.86) mendefinisikan plagiarisme sebagai pembajakan berupa fakta, penjelasan, ungkapan, dan kalimat orang lain secara tidak sah. Plagiarisme sendiri dapat digolongkan sebagai tindakan kriminal karena dikategorikan sebagai tindakan pencurian hak cipta orang lain. Di Indonesia sendiri, perlindungan tentang hak cipta diatur dalam Undang Undang Republik Indonesia Nomor 19 Tahun 2002 Tentang Hak Cipta.

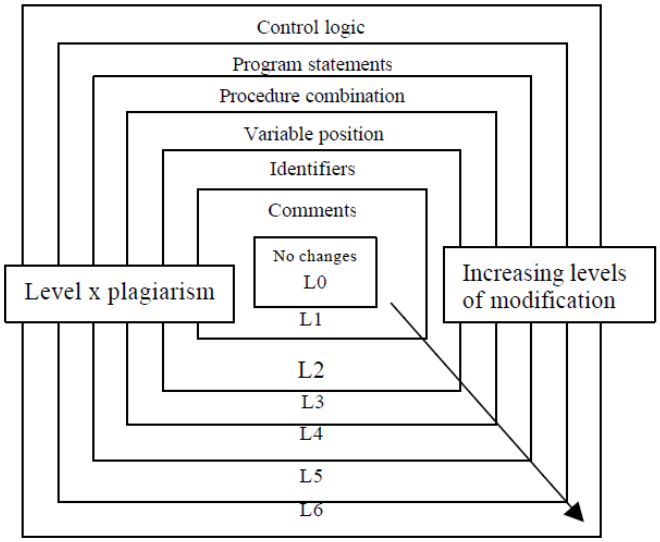
Plagiarism.org menggolongkan hal-hal berikut sebagai tindakan plagiarisme:

1. Mengambil hasil karya orang lain menjadi hasil karya sendiri.
2. Menyalin kata-kata atau ide dari hasil karya orang lain tanpa menulis nama orang tersebut.
3. Tidak meletakan kutipan di dalam tanda kutip.
4. Memberikan informasi keliru tentang sumber kutipan.
5. Mengubah kata namun melakukan penyalinan struktur kalimat dari sebuah sumber tanpa menulis sumbernya.
6. Sebagian besar isi dari hasil karya sendiri merupakan hasil penyalinan dari kata-kata atau ide orang lain

Secara spesifik dalam bentuk plagiarisme kode program, sebuah tinjauan literatur terhadap plagiarisme kode program dalam bentuk tugas yang dikerjakan siswa mengungkapkan bahwa belum ada deskripsi umum yang disetujui tentang plagiarisme kode program dari perspektif akademisi yang mengajar mata kuliah pemrograman (Goel, Tanpa Tahun, p.1).

Walaupun demikian, beberapa definisi tentang plagiarisme kode program tetap muncul, seperti Faidi & Robinson (1987, pp.11-19), yang mendeskripsikan bahwa plagiarisme terjadi ketika seorang siswa melakukan penyalinan dan perubahan dari tugas-tugas pemrograman milik orang lain dengan usaha yang sangat sedikit/kecil.

Sedangkan Parker & Hamblen (1989) mendefinisikan plagiarisme kode program sebagai "*a program which has been produced from another program with a small number of routine transformations*". Proses transformasi yang dapat dilakukan sangat beragam dari yang paling mudah (seperti mengubah komentar atau nama variabel) hingga yang kompleks (seperti mengubah bentuk *looping* "*for*" menjadi "*while*"). Enam level modifikasi dari proses transformasi tersebut dapat dilihat pada Gambar 2.2.



Gambar 2.2 Spektrum Plagiarisme Program (Faidi & Robinson, 1987, pp.11-19)

Sendow (2001) mengatakan bahwa cara yang paling umum dilakukan dalam menyalin kode program milik orang lain adalah dengan melakukan teknik *copy-paste* atau yang juga dikenal sebagai *Naked Plagiarism*. Selain itu ada juga istilah *Semi Naked Plagiarism*, dimana bentuknya hampir sama dengan *naked plagiarism*, hanya saja setelah melakukan *copy-paste*, pelaku melakukan sedikit modifikasi/perubahan, tetapi hasilnya tetap 80% sama.

Beberapa teknik plagiarisme yang dikenal selama ini menurut Goenawan, dkk (2005, pp.1-2), yaitu.

1. *Word-for-word plagiarism.* Teknik plagiarisme dengan cara menyalin setiap kata secara langsung tanpa diubah sedikitpun.
2. *Plagiarism of the form of a source*. Teknik plagiarisme dengan cara menyalin atau menulis ulang kode-kode program tanpa mengubah struktur dan jalannya program.
3. *Plagiarism of authorship*. Teknik plagiarisme dengan cara mengakui hasil karya orang lain sebagai hasil karya sendiri dengan cara mencantumkan nama sendiri menggantikan nama pengarang sebenarnya.

Sedangkan dalam konteks kode program, praktik plagiat dapat dikelompokan menjadi 2 (Goenawan, 2005, pp.1-2), yakni.

1. Leksikal. Praktik perubahan pada kode program (*source code*), misalnya.
   a. Komentar diubah (ditambah, dikurangi, atau diganti).
   b. Format penulisan diubah.
   c. Nama variabel diubah.

2. Struktural. Praktik perubahan struktur program, misalnya.
   a. Perubahan urutan algoritma yang tidak mengubah jalannya program.
   b. Prosedur diubah menjadi fungsi atau sebaliknya.
   c. Pemanggilan prosedur diubah menjadi isi prosedur itu sendiri.

Berdasarkan ruang lingkup pemeriksaan lokasi dokumen, Novanta (2009, p.21) mengelompokkan pendeteksian plagiarisme menjadi 2 jenis, yaitu.

1. *Intra-Corporal Detection*

   Jenis pendekatan ini dilakukan secara *offline*, yang berarti dokumen teks yang diidentifikasi plagiat (*copy documents*) diperiksa dengan dokumen teks yang dianggap alsi (*source documents*) dibatasi pada sebuah lokasi (*folder*) tertentu yang terdiri dari beberapa dokumen (*corpus*) yang akan dibandingkan, dimana proses pengumpulan koleksi dokumen dilakukan secara manual. Biasanya jenis pendeteksian seperti ini digunakan untuk mendeteksi hasil kerja berupa karya tulis siswa/mahasiswa atau peneliti dalam bidang tertentu.

2. *Internet-Based Detection*

   Jenis pendeteksian ini dilakukan secara *online*, yang berarti dokumen teks yang diidentifikasi plagiat (*copy documents*) diperiksa dengan dokumen teks (*source documents*) yang berada tersebar pada jaringan *World Wide Web*. Salah satu teknik yang digunakan adalah *exhaustive searching* (Knight, 2003) yaitu pencarian dengan membandingkan keseluruhan *copy*

dokumen teks dengan *source* dokumen teks yang berada di internet. Pendekatan lainnya adalah *window based*, yaitu proses memecah dokumen teks ke dalam beberapa kalimat tunggal dan menjadikan kalimat tunggal tersebut menjadi sebuah *query* yang akan berfungsi sebagai *keyword* pencarian dokumen yang relevan yang tersebar di internet.

Aplikasi pendeteksi plagiarisme sendiri telah muncul sejak lebih dari tiga puluh tahun yang lalu (Halstead 1977, Ottenstein 1977) dan masih banyak digunakan hingga sekarang dalam dunia pendidikan. Sistem pertama didasarkan pada algoritma penghitung atribut, penggalian metric dari kode program, seperti penghtiungan jumlah penggunaan *reserved words* tertentu, kemudian membandingkan kode program yang memiliki persamaan paling banyak untuk diinspeksi lebih lanjut. Penelitian selanjutnya memperkenalkan struktur metrik dimana setiap kode program dibagi-bagi menjadi sekumpulan *identifiers* atau token, seperti pemanggilan fungsi atau deklarasi variabel. Pasangan dari kode program yang telah dibagi menjadi token selanjutnya akan dilakukan pencarian secara rekursif terhadap *sequences* token tertentu yang terpanjang dan hasil proporsi persamaannya akan digunakan dalam metric persamaan (Goel, Tanpa Tahun, p.2).

2.3 Bahasa Pemrograman C

Hartono (2000, p.1) menjelaskan bahwa akar dari bahasa C adalah bahasa BCPL yang dikembangkan oleh Martin Richards pada tahun 1967. Bahasa ini

memberikan ide kepada Ken Thompson yang kemudian mengembangkan bahasa yang disebut dengan B pada tahun 1970. Perkembangan selanjutnya dari bahasa B adalah bahasa C oleh Dennis Ritchie sekitar tahun 1970-an di *Bell Telephone Laboratories Inc*. (sekarang adalah *AT&T Bell Laboratories*).

C adalah bahasa pemrograman yang bersifat *general-purpose*. C sangat erat kaitannya dengan sistem UNIX, yang mana sistem dan sebagian besar program di dalamnya ditulis dalam bahasa C. Walaupun dikenal sebagai "*system programming language*" karena banyak digunakan untuk membuat *compilers* dan *operating systems*, C juga dapat digunakan untuk membuat program dalam berbagai aspek dan untuk berbagai tujuan (Kernighan, 2007, p.1).

Menurut sebuah survei yang dilakukan oleh *TIOBE Programming Community* pada bulan Juni 2011, bahasa C masih mendudukin peringkat ke dua dalam skala popularitas bahasa pemrograman di seluruh dunia, yakni sebesar 16,278%. Karena kepopulerannya ini, banyak dibuat versi-versi dari bahasa C untuk komputer mikro. Dan untuk membuat versi-versi tersebut standar, ANSI (*American National Standars Institute*) membentuk suatu komite (ANSI *committee* X3J11) pada tahun 1983 yang kemudian menetapkan standar ANSI untuk bahasa C.

2.4 Algoritma

Dietel (2007, p.61) mendefinisikan algoritma sebagai "*the solution to any computing problem involves executing a series of actions in a specific order. A*

procedure for solving a problem in terms of (1) the actions to be executed, and (2) the order in which these actions are to be executed is called algorithm".

Sedangkan menurut Zarlis (2008, p.1) Algoritma adalah urutan langkah-langkah logis penyelesaian masalah yang disusun secara sistematis dan logis. Kata logis merupakan kata kunci dalam algoritma. Langkah-langkah dalam algoritma harus logis dan harus dapat ditentukan bernilai salah atau benar.

Dalam beberapa konteks, algoritma adalah spesifikasi urutan langkah untuk melakukan pekerjaan tertentu. Beberapa pertimbangan dalam pemilihan algoritma antara lain, pertama, algoritma haruslah benar. Artinya algoritma akan memberikan keluaran sesuai yang dikehendaki berdasarkan sejumlah masukkan yang diberikan.

Pertimbangan kedua yang harus diperhatikan adalah kita harus mengetahui seberapa baik hasil yang dicapai oleh algoritma tersebut. Hal ini menjadi penting terutama untuk kasus algoritma yang bertujuan untuk menyelesaikan masalah yang memerlukan aproksimasi hasil (hasil yang berupa pendekatan). Algoritma yang baik harus mampu memberikan hasil sedekat mungkin dengan nilai yang sebenarnya

Yang ketiga adalah efisiensi algoritma. Efisiensi algoritma dapat ditinjau dari 2 hal, yaitu efisiensi waktu dan memori. Meskipun algoritma memberikan keluaran yang benar (paling mendekati), tetapi jika kita harus menunggu dalam waktu yang lama untuk mendapatkan hasil keluarannya, biasanya algoritma tersebut tidak akan dipakai. Begitu juga dengan memori. Semakin besar memori yang dibutuhkan, maka semakin buruklah algoritma tersebut.

Algoritma dapat dituliskan dengan banyak cara, mulai dari menggunakan bahasa alami yang digunakan sehari-hari, simbol grafik bagan alir, sampai menggunakan bahasa pemrograman. Beberapa hal yang perlu diperhatikan dalam membuat algoritma (Zarlis, 2008, pp.3-4):

1. Teks algoritma berisi deskripsi langkah-langkah penyelesaian masalah. Deskripsi tersebut dapat ditulis dalam notasi apapun asalkan mudah dimengerti dan dipahami.
2. Tidak ada notasi yang baku dalam penulisan teks algoritma seperti notasi bahasa pemrograman. Notasi yang digunakan dalam menulis algoritma disebut notasi algoritmik.
3. Setiap orang dapat membuat aturan penulisan dan notasi algoritmik sendiri. Hal ini dikarenakan teks algoritma tidak sama dengan teks program. Namun, supaya notasi algoritmik mudah ditranslasikan ke dalam notasi bahasa pemrograman tertentu, maka sebaiknya notasi algoritmik tersebut berkorespondensi dengan notasi bahasa pemrograman secara umum.
4. Notasi algoritmik bukan notasi bahasa pemrograman, karena itu *pseudocode* dalam notasi algoritmik tidak dapat dijalankan oleh komputer. Agar dapat dijalankan oleh komputer, *pseudocode* dalam notasi algoritmik harus ditranslasikan atau diterjemahkan ke dalam notasi bahasa pemrograman yang dipilih. Perlu diingat bahwa orang yang menulis program sangat terikat dalam aturan tata bahasanya dan spesifikasi mesin yang menjalannya.

5. Algoritma sebenarnya digunakan untuk membantu kita dalam mengkonversikan suatu permasalahan ke dalam bahasa pemrograman.
6. Algoritma merupakan hasil pemikiran konseptual, supaya dapat dilaksanakan oleh komputer, algoritma harus ditranslasikan ke dalam notasi bahasa pemrograman. Ada beberapa hal yang harus diperhatikan pada translasi tersebut, yaitu.
   a. Pendeklarasian variabel

      Untuk mengetahui dibutuhkannya pendeklarasian variabel dalam penggunaan bahasa pemrograman apabila tidak semua bahasa pemrograman membutuhkannya.
   b. Pemilihan tipe data

      Apabila bahasa pemrograman yang akan digunakan membutuhkan pendeklarasian variabel maka perlu hal ini dipertimbangkan pada saat pemilihan tipe data.
   c. Pemakaian instruksi-instruksi

      Beberapa instruksi mempunyai kegunaan yang sama tetapi masing-masing memiliki kelebihan dan kekurangan yang berbeda.
   d. Aturan sintaksis

      Pada saat menuliskan program kita terikat dengan aturan sintaksis dalam bahasa pemrograman yang akan digunakan.

e. Tampilan hasil

Pada saat membuat algoritma kita tidak memikirkan tampilan hasil yang akan disajikan. Hal-hal teknis ini diperhatikan ketika mengkonversikannya menjadi program.

f. Cara pengoperasian *compiler* atau *interpreter*.

Bahasa pemrograman yang digunakan termasuk dalam kelompok *compiler* atau *interpreter*.

Knuth (1973) menyatakan 5 komponen utama dalam algoritma, yakni *finiteness* (algoritma harus berhenti setelah jumlah langkah yang ditetapkan), *definiteness* (setiap langkah harus didefinisikan dengan tepat, tidak boleh ambigu), *input* (sebuah algoritma memiliki satu atau lebih *input* sebelum dijalankan), *output* (sebuah algoritma menghasilkan satu atau lebih *output*, yang biasanya bergantung sesuai dengan *input* yang diberikan), dan *effectiveness* (setiap algoritma harus efektif).

Sehingga dalam merancang sebuah algoritma ada 3 (tiga) komponen yang harus ada (Knuth, 1973):

1. Komponen masukkan (*input*)

   Komponen ini biasanya terdiri dari pemilihan variabel, tipe variabel, konstanta dan parameter (dalam fungsi).

2. Komponen keluaran (*output*)

   Komponen ini merupakan tujuan dari perancangan algoritma dan program. Permasalahan yang diselesaikan dalam algoritma harus dapat ditampilkan dalam komponen keluaran. Karakteristik keluaran yang baik adalah benar

(menjawab) sesuai dengan permasalahan dan tampilan yang ramah (*user friendly*).

3. Komponen proses (*processing*)

   Komponen ini merupakan bagian utama dan terpenting dalam merancang sebuah algoritma. Dalam bagian ini terdapat logika masalah, logika algoritma (sintaksis dan semantik), rumusan, metode (rekursi, perbandingan, penggabungan, pengurangan dan lain-lain).

2.5 Algoritma Levenshtein Distance

Gilleland dalam situsnya, *Merriampark.com* menjelaskan bahwa *Levenshtein Distance* dinamakan berdasarkan penemunya yang merupakan *scientist* Rusia bernama Vladimir Levenshtein pada tahun 1965. *Levenshtein Distance* seringkali juga disebut dengan nama *Edit Distance*. Pemanfaatan dari algoritma ini antara lain: *spell checking*, *speech recognition*, *DNA analysis*, dan *plagiarism detector*.

Filipe (2009, p.750) menjelaskan bahwa *Levenshtein Distance* merupakan metrik yang digunakan untuk menentukan jarak antar dua *sequence*. *Levenshtein Distance* sering digunakan terutama dalam konteks *string*. Jarak antara dua *string* ditentukan oleh jumlah minimal operasi yang dibutuhkan untuk merubah satu *string* ke *string* lainnya. Operasi yang dapat dilakukan adalah *substitution*, *insertion*, dan *deletion*.

Levenshtein Distance dihitung menggunakan metode *dynamic programming*. *Dynamic programming* merupakan metode untuk menemukan solusi masalah yang memiliki karakteristik *overlaping subproblems* dan *optimal*

substructure. *Overlaping subproblems* artinya masalah yang dapat dipecah menjadi submasalah yang dapat digunakan kembali. *Optimal substructure* berarti solusi global optimal dapat diperoleh dari solusi submasalah optimal. Oleh karena adanya karakteristik tersebut, secara tipikal metode *dynamic programming* menggunakan tabel atau matriks dalam pencarian solusi.

Sebagai contoh, jarak antara *string* 'Gumbo' dan 'Gambol' adalah dua. Dimana dibutuhkan minimal dua operasi untuk merubah *string* Gumbo menjadi Gambol atau sebaliknya.

1. Gumbo -> Gambo (substitusi 'u' dengan 'a').
2. Gambo -> Gambol (*insert* 'l').

Tabel 2.1 Contoh Matriks Hasil Pengerjaan *Levenshtein Distance*

		G	U	M	B	O
	0	**1**	**2**	**3**	**4**	**5**
G	**1**	0	1	2	3	4
A	**2**	1	1	2	3	4
M	**3**	2	2	1	2	3
B	**4**	3	3	2	1	2
O	**5**	4	4	3	2	1
L	**6**	5	5	4	3	**2**

Secara garis besar, urutan penghitungan jarak dalam Algoritma *Levenshtein Distance* dapat dituliskan sebagai berikut (*Merriampark.com*).

1. n = panjang *string* 1 (s).

   m = panjang *string* 2 (t).

if n = 0, *return* m.

if m = 0, *return* n.

2. inisialisasi baris pertama dengan nilai 0...n.

   inisialisasi kolom pertama dengan nilai 0...m.

3. *looping* i = 0 *to* n.

   looping j = 0 *to* m.

4. *if* s[i] = t[j], maka nilai *cost* = 0.

   if s[i] ≠ t[j], maka nilai *cost* = 1.

5. isi nilai matriks pada sel d[i, j] dengan nilai terkecil antara:

   a. nilai matriks pada sel d[i-1, j] + 1.

   b. nilai matriks pada sel d[i, j-1] + 1.

   c. nilai matriks pada sel d[i-1, j-1] + *cost*.

6. ulangi iterasi nomor 4-5 hingga *looping* selesai.

   hasil akhir dari algoritma ini adalah nilai matriks pada sel d[n, m].

2.6 Algoritma Brute Force

Munir (2004, p.2) mendeskripsikan *Brute Force* adalah sebuah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah (*problem statement*) dan definisi konsep yang dilibatkan. Algoritma *Brute Force* memecahkan masalah dengan sangat sederhana, langsung, dan dengan cara yang jelas (*obvious way*).

Beberapa karakteristik yang dimiliki oleh algoritma *Brute Force* yaitu (Munir, 2004, pp.20-22).

1. Algoritma *Brute Force* umumnya tidak "cerdas" dan tidak mangkus, karena ia membutuhkan jumlah langkah yang besar dalam penyelesaiannya. Kadang-kadang algoritma *Brute Force* disebut juga algoritma naif (*naive algorithm*).
2. Algoritma *Brute Force* seringkali merupakan pilihan yang kurang disukai karena ketidakmangkusannya itu, tetapi dengan mencari pola-pola yang mendasar, keteraturan, atau trik-trik khusus, biasanya akan membantu kita menemukan algoritma yang lebih cerdas dan lebih mangkus.
3. Untuk masalah yang ukurannya kecil, kesederhanaan *Brute Force* biasanya lebih diperhitungkan daripada ketidak-mangkusannya. Algoritma *Brute Force* sering digunakan sebagai basis bila membandingkan beberapa alternatif algoritma yang mangkus.
4. Meskipun *Brute Force* bukan merupakan teknik pemecahan masalah yang mangkus, namun teknik *Brute Force* dapat diterapkan pada sebagian besar masalah. Agak sukar menunjukkan masalah yang tidak dapat dipecahkan dengan teknik *Brute Force*. Bahkan ada masalah yang hanya dapat dipecahkan secara *Brute Force*. Beberapa pekerjaan mendasar di dalam komputer dilakukan secara *Brute Force*, seperti menghitung jumlah dari n buah bilangan, mencari elemen terbesar di dalam tabel, dan sebagainya.
5. Selain itu, algoritma *Brute Force* seringkali lebih mudah diimplementasikandaripada algoritma yang lebih canggih, dan karena

kesederhanaannya, kadang-kadang algoritma *Brute Force* dapat lebih mangkus (ditinjau dari segi implementasi).

Dalam aplikasi pendeteksi plagiarisme ini, algoritma *Brute Force* diaplikasikan untuk membandingkan baris per baris kode program pada dokumen teks yang satu dengan baris-baris kode program pada dokumen teks yang lainnya.

2.7 Preprocessing

Dani (2006, p.1) mengungkapkan bahwa sebagian besar teknik deteksi kemiripan kode program saat ini melakukan *preprocessing* untuk menghasilkan tingkat akurasi yang lebih baik. Selanjutnya Novanta (2009, p.26) menambahkan bahwa metode *preprocessing* digunakan karena algoritma standar yang digunakan dalam aplikasi pendeteksi plagiarisme hanya membandingkan secara eksplisit dua *string* tanpa mengetahui sifat-sifat yang membentuk kedua *string* tersebut. Oleh karena itu, dibutuhkan proses bantuan (*preprocessing*) dalam bentuk modul-modul tambahan, dimana modul-modul tersebut tersebar di dalam proses pembobotan sesuai fungsi masing-masing modul.

Metode *preprocessing* tentu akan menambah waktu proses sistem secara menyeluruh, tetapi dengan adanya pereduksian *noise* yang dilakukan proses bantuan ini, diharapkan dapat mengurangi kompleksitas pada saat pembandingan oleh algoritma dasar yang digunakan untuk menghasilkan bobot persentase kecenderungan plagiarisme. Beberapa teknik *preprocessing* yang dapat dilakukan untuk membantu dalam proses pendeteksian plagiarisme kode program antara

lain: penghapusan komentar, penyeragaman nama variabel, penghapusan isi *string*, menambahkan penanda blok pada setiap *if* yang hanya diikuti satu pernyataan, dan lain sebagainya (Dani, 2006, p.2).

2.8 Microsoft Visual Studio

Microsoft Visual Studio merupakan sebuah perangkat lunak lengkap (*suite*) yang dapat digunakan untuk melakukan pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya, dalam bentuk aplikasi *console*, *Windows*, ataupun *Web*. Visual Studio mencakup *compiler*, SDK, *Integrated Development Environment* (IDE), dan dokumentasi (umumnya berupa MSDB *Library*). *Compiler* yang dimasukkan ke dalam paket Visual Studio antara lain Visual C++, Visual C#, Visual Basic, Visual Basic.NET, Visual InterDev, Visual J++, Visual J#, Visual FoxPro, dan Visual SourceSafe (Wikipedia, 2011).

Berdasarkan situs *Microsoft.com*, Microsoft Visual Studio 2008 yang dirilis pada bulan November 2007 merupakan aplikasi pertama yang mendukung fitur perancangan visual untuk *Windows Presentation Foundation* (WPF) dengan *snap lines* dan *event tabs* yang memungkinkan proses pengembangan dengan model RAD. Microsoft Visual Studio dapat digunakan untuk mengembangkan aplikasi dalam *native code* (dalam bentuk bahasa mesin yang berjalan di atas *Windows*) ataupun *managed code* (dalam bentuk *Microsoft Intermediate Language* di atas .NET *Framework*).

2.9 Bahasa Pemrograman C#

Bahasa pemrograman C# dikembangkan oleh Microsoft sebagai bahasa yang simpel, modern, *general-purpose*, dan berorientasi objek. Pengembanagan bahasa C# sangat dipengaruhi oleh bahasa pemrograman terdahulu, terutama C++, Delphi, dan Java. Kehadiran C# memberikan suntikan optimisme bagi para *programmer* untuk dapat mengembangkan aplikasi yang berdaya guna dengan lebih cepat dan lebih mudah. (Cybertron Solution, 2009, p.1)

Dalam survei yang dilakukan oleh *TIOBE Programming Community* pada bulan Juni 2011, bahasa C# menduduki peringkat ke empat dalam skala popularitas bahasa pemrograman di seluruh dunia, yakni sebesar 6.844%.

C# merupakan bahasa pemrograman perusahaan Microsoft yang didesain dengan target diimplementasikan dalam teknologi *framework* .NET. Pada tahun 2002, tepatnya bulan Januari, C# maupun *framework* .NET diselesaikan dan dapat diimplementasikan di kalangan industri. Bahasa pemrograman C# merupakan bahasa baru yang sangat andal dan konsisten serta membawa kesan bahasa pemrograman yang modern.

C# pada dasarnya bukan bahasa sulit karena intinya mengambil dari berbagai bahasa pemrograman yang telah ada. Bahasa yang paling banyak digunakan dalam sintaksis C# adalah bahasa Java (70%). Selain itu bahasa C# mengadopsi bahasa C++ (10%) dan Visual Basic (5%). Sedangkan 15% sisanya adalah bahasa yang baru (Jaenudin, 2005, p.15).