Deteksi Kemiripan Kode Sumber Menggunakan Metode *Abstract Syntax Tree* dan Algoritma Ratcliff/Obershelp

Diajukan Untuk Menyusun Skripsi
di Jurusan Teknik Informatika Fakultas Ilmu Komputer UNSRI

Oleh:

Muhammad Sholeh

NIM: 09021281823172

Jurusan Teknik Informatika

FAKULTAS ILMU KOMPUTER UNIVERSITAS SRIWIJAYA

2021

LEMBAR PENGESAHAN PROPOSAL SKRIPSI

DETEKSI KEMIRIPAN KODE SUMBER MENGGUNAKAN METODE *ABSTRACT SYNTAX TREE* DAN ALGORITMA RATCLIFF/OBERSHELP

Oleh:

Muhammad Sholeh
NIM: 09021281823172

Indralaya, Oktober 2021

Pembimbing I,	Pembimbing II,
Dr. Abdiansah., S.Kom., M.Cs	Alvi Syahrini Utami, M.Kom.
NIP. 198410012009121005	NIP. 19781222200642003

Mengetahui,
Ketua Jurusan

Alvi Syahrini Utami, M.Kom.
NIP. 19781222200642003

DAFTAR ISI

Halaman

DAFTAR TABEL

DAFTAR GAMBAR

BAB I

PENDAHULUAN

1.1. Pendahuluan

Pada bab ini akan dibahas berkenaan dengan garis besar pokok-pokok pikiran dalam penelitian ini. Pokok pikiran yang akan dibahas antara lain latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelitian, dan manfaat penelitian. Pokok-pokok pikiran yang diuraikan akan dijadikan acuan dalam kajian penelitian ini.

1.2. Latar Belakang

Perkembangan teknologi dan informasi saat ini maju dengan sangat pesat sehingga mempermudah manusia dalam bertukar informasi. Seiring dengan kemajuannya, banyak terjadi pula pelanggaran di dalamnya, salah satunya adalah pelanggaran hak cipta berupa plagiarisme. Plagiarisme dapat menyasar ke dalam berbagai bidang, salah satunya pada bidang pendidikan yang berupa plagiarisme terhadap kode sumber. Kode sumber merupakan sekumpulan baris kode berupa pernyataan atau deklarasi dalam bahasa pemrograman komputer yang ditulis dan dapat dibaca oleh manusia (Jalal dan Prasetya, 2014). Plagiarisme terhadap kode sumber merupakan kegiatan membuat sebuah program yang berasal dari program lain dengan transformasi kecil di dalamnya (Kaučič et al., 2010). Transformasi yang dilakukan dapat beragam dari menyalin sebagian kecil kode sumber asli hingga menyalin seluruh bagian dari kode sumber asli.

Dalam penelitian yang dilakukan oleh Sraka dan Kaučič (2009) sebanyak 100 dari 138 (72,5%) mahasiswa yang ikut dalam survei anonim menyatakan bahwa mereka setidaknya pernah melakukan plagiarisme baik terhadap kode sumber ataupun tugas lain setidaknya sekali selama masa studi mereka. Lewat penelitian ini, dapat dikatakan bahwa plagiarisme di tingkat mahasiswa marak terjadi, khususnya plagiarisme terhadap kode sumber yang kemudian akan sangat mempengaruhi kualitas pendidikan ilmu komputer. Untuk mengurangi perkembangan akademik yang buruk, penelitian mengenai deteksi plagiarisme terhadap kode sumber sangat dibutuhkan.

Terdapat beberapa penelitian sebelumnya mengenai deteksi kemiripan kode sumber dengan menggunakan berbagai metode. Metode yang banyak digunakan diantaranya seperti metode Abstract Syntax Tree (AST), algoritma Levenshtein, algoritma Jaro-Winkler, algoritma Ratcliff/Obershelp dan Cosine Similarity. Metode AST merupakan metode tree-based yang dapat mendeteksi kemiripan kode sumber lebih akurat karena pendeteksian menggunakan struktur dari suatu kode sumber (Rusdianto dan Chaniago, 2019). Metode ini banyak digunakan dalam pelbagai penelitian seperti yang dilakukan Jalal dan Prasetya (2014), dan Zhang dan Liu (2013) Metode AST tidak dapat mendeteksi adanya plagiarisme terhadap dokumen secara langsung, oleh karenanya dibutuhkan suatu metode atau algoritma lain untuk mengukur tingkat kemiripan dari suatu dokumen yang dalam hal ini berupa kode sumber.

Algoritma Ratcliff/Obershelp dapat membantu metode AST dalam mendeteksi plagiarisme pada kode sumber. Algoritma ini banyak digunakan dalam mendeteksi kemiripan dokumen khususnya dokumen kode sumber,

diantaranya yang dilakukan oleh Joane et al. (2017), Ilyankou (2014), dan (Hidayat et al., 2020). Lewat penelitiannya Ilyankou (2014) menyatakan algoritma Ratcliff/Obershelp memiliki akurasi deteksi kemiripan yang lebih baik dengan angka efisiensi 18,6% dibanding algoritma Jaro-Winkler dengan angka efisiensi 9,7%.

Dengan adanya referensi penelitian terkait sebelumnya, metode *Abstract Syntax Tree* dan algoritma Ratcliff/Obershelp akan menjadi metode yang digunakan dalam penelitian deteksi kemiripan kode sumber. Diharapkan dengan adanya kombinasi dua metode ini, plagiarisme terhadap kode sumber dapat dideteksi dengan akurasi sistem yang lebih baik.

1.3. Rumusan Masalah

Rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana melakukan deteksi kemiripan antar kode sumber menggunakan metode *Abstract Syntax Tree* dan algoritma Ratcliff/Obershelp?
2. Bagaimana kinerja metode *Abstract Syntax Tree* dan algoritma Ratcliff/Obershelp dalam mendeteksi kemiripan antar kode sumber.

1.4. Tujuan Penelitian

Tujuan dari penelitian ini adalah sebagai berikut:

1. Membangun perangkat lunak deteksi kemiripan kode sumber menggunakan metode *Abstract Syntax Tree* dan algoritma Ratcliff/Obershelp.
2. Mengukur kinerja metode Abstract Syntax Tree dan algoritma Ratcliff/Obershelp dalam mendeteksi kemiripan antar kode sumber.

1.5. Manfaat Penelitian

Manfaat dari penelitian ini adalah sebagai berikut:

1. Sistem yang dibuat dapat membantu pengguna untuk mendeteksi adanya plagiarisme pada dokumen kode sumber.
2. Hasil penelitian dapat dijadikan sebagai rujukan untuk penelitian terkait di masa mendatang.

1.6. Batasan Masalah

Batasan masalah dalam penelitian ini adalah sebagai berikut:

1. Kode sumber yang akan diteliti persentase kemiripannya menggunakan bahasa pemrograman Java.
2. Data yang digunakan adalah kode sumber yang berasal dari tugas mata kuliah Pemrograman Komputer
3. Data diperoleh dari kumpulan tugas mahasiswa dari mata kuliah praktikum Pemrograman Komputer di lingkungan Jurusan Teknik Informatika Universitas Sriwijaya.

1.7. Sistematika Penulisan

Sistematika penulisan dalam penelitian ini adalah sebagai berikut:

BAB I. PENDAHULUAN

Pada bab ini menjelaskan tentang latar belakang masalah, rumusan masalah, tujuan penelitian, manfaat penelitian dan batasan masalah. Pokok-pokok pikiran ini akan menjadi dasar dan acuan pengembangan penelitian pada bab selanjutnya.

BAB II. KAJIAN LITERATUR

Pada bab ini dibahas landasan teori yang digunakan di dalam penelitian, termasuk di dalamnya mengenai plagiarisme, *Abstract Syntax* Tree, algoritma Ratcliff/Obershelp, dan penelitian terkait yang relevan.

BAB III. METODOLOGI PENELITIAN

Pada bab ini dibahas proses pengumpulan data dan tahapan-tahapan di dalam penelitian. Tahapan penelitian dibahas lebih rinci berdasarkan kerangka kerja tertentu. Di bagian akhir bab ini akan dimuat rancangan manajemen proyek penelitian.

1.8. Kesimpulan

Pada Bab ini telah dibahas mengenai latar belakang penelitian serta acuan penting dalam penelitian seperti latar belakang, rumusan masalah, tujuan penelitian, manfaat penelitian, batasan masalah dan sistematika penulisan.

BAB II

KAJIAN LITERATUR

2.1. Pendahuluan

Pada bab ini akan dijelaskan mengenai kajian literatur yang mendasari penelitian ini. Pembahasan akan menjelaskan mengenai plagiarisme kode sumber, *text mining*, pra-pengolahan teks, metode *Abstract Syntax Tree* dan algoritma Ratcliff/Obershelp. Pada bab ini pula dibahas mengenai penelitian terkait lainnya yang relevan.

2.2. Landasan Teori

2.2.1. Plagiarisme Kode Sumber

Plagiarisme merupakan tindakan menggunakan karya milik orang lain dan mencoba mengakuinya sebagai karya milik sendiri. Plagiarisme dapat mencakup penggunaan kata-kata, kalimat, ide maupun tugas milik orang lain dengan tidak melakukan sitasi ataupun dokumentasi secara benar (Smith, 2012).

Dalam penelitian yang dilakukan Gipp & Meuschke (2011), dikatakan bahwa metode deteksi plagiarisme secara umum terbagi ke dalam pendekatan intrinsik dan eksternal. Pendekatan intrinsik merupakan pendekatan yang membandingkan gaya penulisan dan fitur linguistik di dalam satu dokumen itu sendiri tanpa perbandingan dengan dokumen lainnya. Sedangkan, pendekatan eksternal merupakan pendekatan yang melakukan deteksi tindakan plagiarisme dengan membandingkan satu atau lebih dokumen yang dicurigai dengan dokumen asli

Deteksi plagiarisme pada kode sumber menurut Agrawal dan Sharma (2017) diklasifikasikan ke dalam 5 kategori, yaitu sebagai berikut:

1. *String*, yaitu pendekatan dengan menyamakan *string* secara keseluruhan. Namun, pendekatan jenis ini dapat dihindari dengan mengganti *identifier* dalam kode sumber.
2. *Token*, yaitu pendekatan yang akan mengkonversi kode program menjadi *token* dengan bantuan *lexer* dari bahasa pemrograman.
3. *Parse-Tree*, yaitu pendekatan yang mengurai kode sumber ke dalam bentuk pohon, lalu kemudian kedua pohon tersebut dibandingkan. Apabila kedua pohon sama persis maka keduanya dinyatakan mirip atau melakukan plagiarisme, dan sebaliknya dinyatakan tidak.
4. *Program Dependency Graphs (PDG)*, yaitu pendekatan lewat aliran kontrol program. Pendekatan jenis ini membutuhkan perhitungan yang besar dan kompleks.
5. *Metrics*, yaitu pendekatan yang menggunakan sistem pemberian skor berdasarkan parameter tertentu ke segmen kode. Skor ditentukan berdasarkan perhitungan jumlah pernyataan kondisi, perulangan atau jumlah variabel dalam kode sumber.

Penelitian ini akan berfokus pada pendekatan eksternal dalam plagiarisme secara umum. Penelitian ini juga menggunakan pendekatan *string* dan *token* dalam plagiarisme kode sumber dengan berusaha mendeteksi adanya tindakan plagiarisme membandingkan dokumen kode sumber utama dengan dokumen kode sumber modifikasi lainnya menggunakan *string* dan *token*.

2.2.2. *Text Mining*

Menurut Dang & Ahmad (2014) *text mining* atau penambangan teks merupakan salah satu bidang penelitian yang bertujuan untuk menemukan informasi penting dari teks yang tidak terstruktur. Penambangan teks merupakan proses menemukan informasi baru atau informasi tersembunyi yang sebelumnya tidak terdeteksi dari suatu data menggunakan teknik tertentu.

Dalam pengolahannya terdapat beberapa teknik untuk melakukan penambangan teks, diantaranya adalah: *information retrieval*, *information extraction*, *categorization*, *clustering dan summarization*. Metode penambangan teks yang digunakan dalam penelitian ini adalah metode *Information Retrieval*. Metode ini merupakan sebuah bidang studi tentang sistem indeks, sistem pencarian, atau proses menyimpan data dari data tidak terstruktur untuk memenuhi pemenuhan kebutuhan informasi yang pengguna inginkan (Joane et al., 2017). Dasar penggunaan metode ini ialah untuk memenuhi kebutuhan informasi dari penelitian yang akan dilakukan.

2.2.3. Pra-pengolahan Teks

Pra-pengolahan teks merupakan salah satu langkah dalam pemrosesan bahasa yang masuk ke dalam tahap daripada penambangan teks. Pra-pengolahan teks melakukan pembersihan dan persiapan terhadap teks sehingga teks menjadi memiliki bentuk terstruktur untuk membantu proses pengolahan selanjutnya. Tahapan yang dilakukan dalam pra-pengolahan teks menurut Firdaus et al. (2014) dalam penelitiannya adalah sebagai berikut:

1. *Case Folding*, yaitu tahap mengubah karakter menjadi huruf kecil.
2. *Tokenizing*, yaitu tahap memotong paragraf atau kalimat ke dalam bentuk kata.
3. *Filtering*, yaitu tahap menyaring antara kata penting dari hasil proses sebelumnya.
4. *Stemming*, yaitu tahap memecah suatu kata menjadi kata dasar
5. *Analyzing*, yang merupakan tahap penelitian tingkat kemiripan dari teks dokumen yang diuji dan di proses sebelumnya.

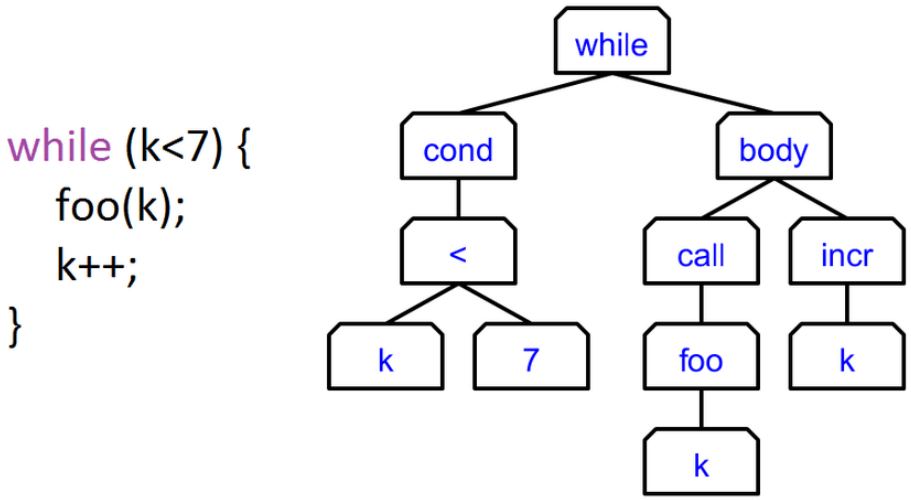
Tahap pra-pengolahan yang akan dilakukan dalam penelitian ini merupakan *cleaning* yakni dengan menghapus baris kosong dan komentar dari kode sumber masukan sehingga hanya menghasilkan kode sumber penting saja yang akan diuji tingkat persentase kemiripannya.

2.2.4. *Abstract Syntax Tree*

Abstract Syntax Tree (AST) dalam pengembangan perangkat lunak merupakan model daripada kode sumber yang dapat merepresentasikan abstrak dari suatu bahasa. Karena itu pula AST dapat merepresentasikan model nyata dalam pemrograman berbentuk diagram *Unified Modeling Language* (UML) seperti misalnya *class diagram* atau *sequence diagram* atau bahkan lebih detail. Ini dikarenakan abstrak level bawah daripada AST lebih mendekati ke kode sumber daripada sebuah diagram (Fischer et al., 2007).

Untuk mengetahui properti sintaks dari sebuah program, maka dilakukan proses penguraian (*parsing*) ke dalam bentuk pohon AST menggunakan suatu tata bahasa (*grammar*) dari suatu bahasa pemrograman. Pohon AST akan dibuat untuk mewakili tiap perintah pemrograman (seperti:

inisiasi, perulangan, pengkondisian, dan lain-lain). Contoh dari bentuk pohon AST tertera seperti pada Gambar II-1.



Gambar II-1. Contoh *Abstract Syntax Tree* dari Potongan Kode Sumber

AST banyak digunakan untuk mempermudah proses deteksi plagiarisme khususnya pada kemiripan kode sumber karena prosesnya yang memecah kode ke dalam bagian yang lebih kecil sehingga lebih mudah untuk dideteksi kemiripannya. Penelitian ini akan memanfaatkan AST untuk mendapatkan token dari tiap cabang yang merepresentasikan kode sumber yang akan digabungkan lalu diukur persentase tingkat kemiripannya.

2.2.5. Algoritma Ratcliff/Obershelp

Algoritma Ratcliff/Obershelp atau *Gestalt Pattern Matching* merupakan algoritma yang diciptakan oleh John W. Ratcliff and John A. Obershelp pada tahun 1983. Algoritma ini dipublikasikan pertama kali pada *Dr. Dobb's Journal* pada Juli tahun 1988. Konsep dasar algoritma ini ialah mencari *substring* paling panjang yang memiliki kesamaan dari dua *string*

masukan yang kemudian disebut dengan istilah *anchor*. Selanjutnya, algoritma akan mencari *substring* paling panjang kembali di sebelah kiri dan kanan *anchor* atau dengan kata lain algoritma bekerja mengulang tahap sebelumnya. Algoritma akan terus mengulang hingga seluruh karakter dari tiap *string* masukan telah di analisa. Setelah tidak ditemukan lagi jumlah *substring* yang sama, maka seluruh panjang *substring* yang memiliki kesamaan akan dijumlahkan dan dihitung persentase tingkat kemiripannya menggunakan persamaan II-1 berikut (Ilyankou, 2014).

$$D_{ro} = \frac{2 * K_m}{|S_1| + |S_2|} \quad \text{(II-1)}$$

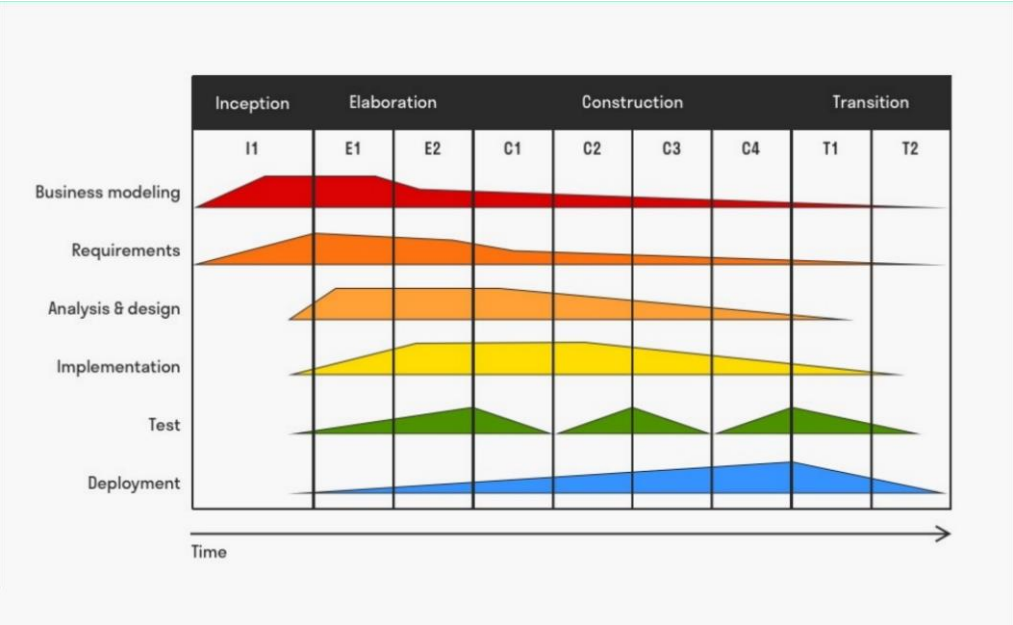
Keterangan :

- D_{ro} : Persentase tingkat kemiripan dari kedua *string*
- K_m : Jumlah karakter total yang sama dari kedua *string*
- $|S_1|, |S_2|$: Panjang karakter *string* pertama dan kedua

2.2.6. *Rational Unified Process*

Rational Unified Process (RUP) adalah metode rekayasa pengembangan perangkat lunak yang digunakan untuk kedisiplinan dalam penetapan tugas dan tanggung jawab. Tujuan RUP adalah memastikan bahwa produk perangkat lunak yang dihasilkan akan berkualitas dan sesuai kebutuhan pengguna akhir (*end-users*) (Anwar, 2014). RUP yang baik akan tercipta lewat hasil kerja sama antara pengembang perangkat lunak, mitra dan pengguna. Salah satu perspektif dalam RUP merupakan *Dynamic Perspective* & *Lifecycle*

Phases yang penggunaannya digambarkan dalam bidang dua dimensi. Bidang horizontal menyatakan lamanya waktu pengembangan dan aspek dinamis lainnya, sedangkan bidang vertikal menyatakan aspek statis dalam rekayasa pengembangan perangkat lunak. Perspektif RUP model ini dinyatakan seperti dalam gambar II-2.



Gambar II-2. Arsitektur *Rational Unified Process*

Dalam bidang horizontal, terdapat fase atau tahap dalam proses rekayasa perangkat lunak yang memaparkan peran dari tiap unit. Fase dalam bidang ini terbagi ke dalam fase insepsi, elaborasi, konstruksi dan transisi.

1. Fase insepsi merupakan fase yang berfokus pada pendefinisian ruang lingkup atau batasan dalam proyek pengembangan dengan cara melakukan analisis desain berorientasi objek (*Object Oriented Analysis Design*). Tujuan dari fase ini adalah untuk mendapatkan

seluruh pemahaman dari pihak yang berkaitan agar sistem yang diajukan sesuai dengan keinginan dan kebutuhan.

2. Fase elaborasi merupakan fase yang akan membuat arsitektur dasar sistem lewat hasil analisis sebelumnya. Fase ini juga akan menentukan perencanaan proyek serta spesifikasi dari fitur yang akan dimuat dalam sistem. Hasil dari fase ini merupakan dokumen arsitektur yang berguna untuk fase selanjutnya.
3. Fase Konstruksi merupakan fase menerjemahkan spesifikasi fitur dari dokumen rancangan sebelumnya ke dalam bentuk program/sistem sesuai dengan arsitekturnya. Fase ini berfokus pada peningkatan fungsi serta implementasi yang lebih mendalam terhadap spesifikasi sistem.
4. Fase Transisi merupakan fase pengujian sistem ke pengguna akhir dimana sistem yang dibuat harus memenuhi kebutuhan perangkat lunak dan kebutuhan penggunanya. Kendali dalam fase ini mulai dipindah kepada tim pemeliharaan perangkat lunak.

2.3. Penelitian Lain yang Relevan

Dalam penelitian deteksi kemiripan kode sumber, telah banyak pelbagai metode yang digunakan, salah satunya adalah metode *Abstract Syntax Tree*. Metode ini dipilih karena kemampuannya memberikan representasi kode sumber dalam bentuk pohon yang mempermudah penelitian dalam mendeteksi adanya plagiarisme. Penelitian terkait menggunakan metode *Abstract Syntax Tree* diantaranya dilakukan oleh Jalal & Prasetya (2014) yang membagi kode sumber ke dalam bentuk *Abstract Syntax Tree* yang memiliki token untuk selanjutnya

dideteksi kemiripannya menggunakan algoritma Levenshtein. Hasil akhir dari penelitian yang dilakukan berupa grafik persentase kemiripan dari kode sumber yang diteliti serta kemiripan baris kode yang dicurigai sebagai plagiarisme.

Penelitian lainnya dilakukan oleh Rusdianto & Chaniago (2019) yang menggunakan metode *Abstract Syntax Tree* dengan *grammar parser* dan *lexer rule* yang sudah dibuat. Penelitian tersebut menggunakan algoritma Damerau-Levensthein untuk mengukur tingkat kemiripan kode sumber yang diteliti. Hasil akurasi dari sistem yang diajukan dengan menggunakan *grammar parser* dan *lexer rule* menunjukkan persentase 97,30%, sedangkan jika hanya menggunakan *grammar lexer rule* menunjukkan persentase 96,92%.

Dalam proses penilaian tingkat kemiripan dokumen yang dalam hal ini berupa kode sumber maka akan digunakan satu atau lebih algoritma pendeteksian kemiripan untuk mengetahui persentase plagiarisme dari dokumen yang diperiksa. Diantara algoritma yang banyak digunakan adalah algoritma Ratcliff/Obershelp. Penelitian terkait kemiripan kode sumber menggunakan algoritma ini dilakukan oleh Joane et al. (2017) yang melakukan pengujian menggunakan 12 dokumen uji dengan tingkat kesamaan dan posisi yang berbeda. Hasil uji menunjukkan nilai hasil prediksi dengan nilai pada dokumen memiliki kesamaan, namun pada posisi yang berbeda menghasilkan nilai keluaran yang berbeda.

Penelitian lain yang relevan dilakukan oleh Ilyankou (2014) dengan membandingkan algoritma Jaro-Winkler dan Algoritma Ratcliff/Obershelp dalam pengecekan kata dengan ejaan yang salah (*misspelled words*). Hasil penelitiannya menunjukkan bahwa algoritma Ratcliff/Obershelp memiliki

akurasi yang lebih tinggi seluruhnya daripada algoritma Jaro-Winkler dengan menggunakan dua kamus yang berbeda dalam pengecekan ejaan yang salah. Penggunaan kamus FreeBSD dan Mieliestronk masing-masing menunjukkan angka efisiensi 4,0% dan 4,3% untuk algoritma Ratcliff/Obershelp berbanding 18,6% dan 9,7% untuk algoritma Jaro-Winkler. Kesimpulan yang didapatkan ialah algoritma Ratcliff/Obershelp lebih efisien minimal 4% daripada algoritma Jaro-Winkler.

Kemudian terdapat penelitian yang dilakukan Hidayat et al. (2020) yang juga menggunakan algoritma Ratcliff/Oberhselp untuk mendeteksi efek dari *stemming* suatu kata dalam mendeteksi tingkat kemiripan kata slang dengan kata formal. Dokumen teks sebelumnya akan melalui tahap pra-pengolahan yang termasuk di dalamnya proses *stemming* yang dilakukan menggunakan algoritma Nazief & Adriani yang bertujuan untuk menghilangkan imbuhan pada suatu kata sehingga didapatkan kata dasarnya. Selanjutnya, kata yang telah diolah baik dalam bentuk slang maupun formal diukur tingkat kemiripannya menggunakan algoritma Ratcliff/Obershelp. Hasil dari pengujian pra-pengolahan dan *stemming* menunjukkan tingkat kemiripan kata slang dan formal berada pada sebaran rentang nilai 80%–89,99% di tiga pengujian berbeda dengan dua jenis dataset berbeda dari Kaggle[1] dan GitHub[2].

[1] https://www.kaggle.com/
[2] https://github.com/

2.4. Kesimpulan

Pada bab ini telah dibahas teori yang akan digunakan sebagai dasar penelitian ini. Pada bab ini juga telah dibahas mengenai penelitian terkait yang mendukung literatur penelitian ini. Mekanisme pelaksanaan penelitian selengkapnya akan dibahas dalam bab selanjutnya.

BAB III

METODE PENELITIAN

3.1. Pendahuluan

Pada bab ini akan dijelaskan mengenai tahapan penelitian, metode penelitian dan manajemen proyek penelitian yang akan dilaksanakan. Tahapan penelitian akan dijadikan sebagai acuan pada tiap-tiap fase dalam pengembangan perangkat lunak.

3.2. Pengumpulan Data

Pada bagian ini akan dijelaskan tahapan pengumpulan data meliputi jenis dan sumber data dan metode pengumpulan data yang digunakan dalam penelitian.

3.2.1. Jenis dan Sumber Data

Jenis data yang digunakan dalam penelitian ini adalah data primer dan sekunder. Data primer yang digunakan berupa data kode sumber yang dibuat menirukan data kode sumber asli atau kemudian disebut dokumen modifikasi, sesuai dengan skenario kasus plagiarisme yang akan dideteksi. Data sekunder yang digunakan berupa kumpulan kode sumber yang berasal dari tugas mahasiswa pada mata kuliah Praktikum Pemrograman Komputer semester ganjil tahun 2021 di Universitas Sriwijaya. Data sekunder yang akan digunakan berjumlah 5 dokumen kode sumber.

3.2.2. Metode Pengumpulan Data

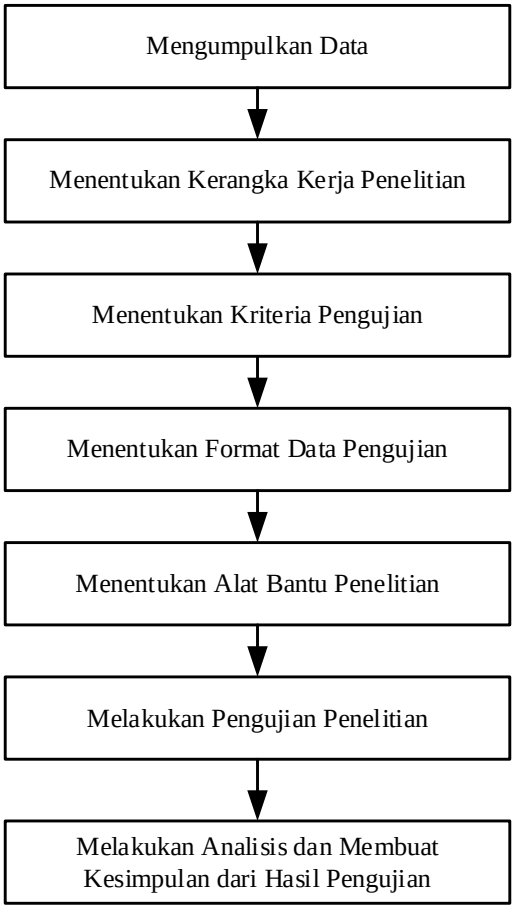
Metode pengumpulan data yang digunakan dalam penelitian adalah metode observasi dan dataset. Metode observasi merupakan metode yang dilakukan untuk mengamati dan memodifikasi dokumen kode sumber sehingga menciptakan data baru yang isinya memodifikasi dokumen kode sumber yang asli sesuai dengan pola kasus plagiarisme yang akan diteliti. Modifikasi pola kasus plagiarisme yang akan diteliti adalah sebagai berikut:

a. Modifikasi 1: Salinan kode sumber asli yang dimodifikasi sebesar 25%.
b. Modifikasi 2: Salinan kode sumber asli yang dimodifikasi sebesar 50%.
c. Modifikasi 3: Salinan kode sumber asli yang dimodifikasi sebesar 75%.
d. Modifikasi 4: Dokumen kode sumber asli yang disalin secara penuh.

Dataset yang digunakan dalam penelitian ini didapatkan dengan mengumpulkan data kode sumber yang berupa tugas mahasiswa pada mata kuliah Pemrograman Komputer yang didapatkan dari dosen yang mengampu mata kuliah tersebut.

3.3. Tahapan Penelitian

Tahapan penelitian adalah rincian proses yang akan dilakukan pada penelitian. Tahapan penelitian yang akan dilakukan pada penelitian ini adalah sebagai berikut.



Gambar III-1. Alur Tahapan Penelitian

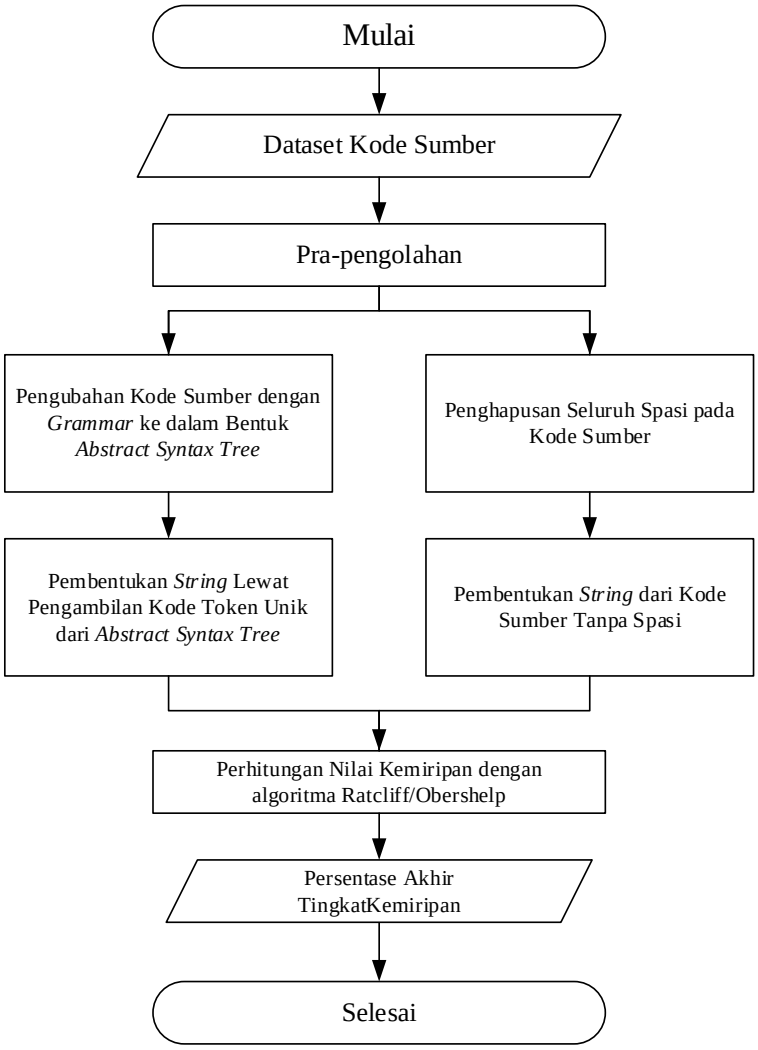
3.3.1. Mengumpulkan Data

Pada tahap ini akan dilakukan pengumpulan data yang berupa dokumen kode sumber. Dokumen kode sumber yang digunakan merupakan tugas dari mata kuliah praktikum Pemrograman Komputer. Dokumen kode sumber didapatkan melalui dosen pengampu mata kuliah yang bersangkutan. Proses pengumpulan

data dilakukan dengan meminta langsung salah satu tugas mata kuliah Pemrograman Komputer kepada dosen yang bersangkutan. Dokumen kode sumber yang dibutuhkan dalam penelitian berjumlah lima buah.

3.3.2. Menentukan Kerangka Kerja Penelitian

Kerangka kerja pada penelitian ini adalah sebagai berikut:



Gambar III-2. Diagram Alir Sistem Deteksi Kemiripan Kode Sumber

Berdasarkan kerangka kerja penelitian pada Gambar III-2, sistem deteksi kemiripan kode sumber menggunakan *Abstract Syntax Tree* dan algoritma Ratcliff/Obershelp memiliki alur sistem sebagai berikut:

1. Pra-pengolahan

   Tahapan pra-pengolahan bermula dengan memasukkan kode sumber. Kemudian, sistem akan melakukan pembersihan (*cleaning*) dengan menghapus komentar dan baris kosong dari tiap kode sumber. Keluaran dari tahap ini merupakan dokumen kode sumber yang sudah bersih dan hanya menyisakan kode yang mengandung inti pemrograman yang akan diolah pada tahap selanjutnya.

2. Transformasi Kode Sumber ke dalam Bentuk *Abstract Syntax Tree*

   Dalam tahap ini, kode sumber yang sudah melalui tahap pra-pengolahan akan diubah ke dalam bentuk AST menggunakan bantuan perangkat lunak ANTLR (*Another Tool for Language Recognition*). ANTLR merupakan perangkat lunak yang dapat membuat *lexer* dan *parser* dari suatu kode sumber menggunakan *grammar*. Setiap objek *lexer* dan *parser* hasil konversi ANTLR memiliki token. Token dari tiap objek tersebut akan diambil dan digabungkan menjadi sebuah *string*. *String* gabungan berupa kumpulan angka (token) dari tiap objek pada kode sumber. Seluruh kode sumber masukan akan memiliki masing-masing satu *string* gabungan.

3. Perhitungan Tingkat Kemiripan dengan Algoritma Ratcliff/Obershelp

   String gabungan dari kode sumber asli kemudian akan dibandingkan dengan seluruh *string* gabungan kode sumber modifikasi menggunakan

algoritma Ratcliff/Obershelp. Hasil pengujian persentase tingkat kemiripan antara kode sumber asli dan modifikasi akan ditampilkan lewat tabel yang telah dipersiapkan untuk dapat digunakan sebagai acuan dalam penentuan kasus plagiarisme kode sumber.

3.3.3. Menentukan Kriteria Pengujian

Kriteria pengujian dalam penelitian ini adalah sebagai berikut:

1. Dokumen kode sumber akan diukur tingkat kemiripannya menggunakan algoritma Ratcliff/Obershelp tanpa menggunakan *Abstract Syntax Tree.*
2. Dokumen kode sumber akan diubah ke dalam bentuk *Abstract Syntax Tree* lalu diukur tingkat kemiripannya berdasarkan gabungan token tiap *node* dari pohon AST menggunakan algoritma Ratcliff/Obershelp.

Pengujian dilakukan dengan membandingkan persentase kemiripan dua dokumen yang diperoleh menggunakan kriteria di atas. Hasil dari pengujian akan ditampilkan dalam bentuk tabel kemiripan.

3.3.4. Menentukan Format Data Pengujian

Format data pengujian yang digunakan dalam penelitian ini berbentuk tabel yang isinya menyatakan tingkat kemiripan dokumen kode sumber asli dengan dokumen kode sumber yang di modifikasi sesuai dengan kriteria pengujian. Dokumen kode sumber yang diterima sistem merupakan dokumen dengan format ekstensi .java. Tabel format pengujian yang digunakan dalam penelitian ini dinyatakan dalam Tabel III-1 berikut.

Tabel III-1. Pengukuran Tingkat Kemiripan Kode Sumber

Kode Sumber	Metode	Persentase			
		Modifikasi 1	Modifikasi 2	Modifikasi 3	Modifikasi 4
Asli 1					
Asli 2					
Asli 3					
Asli 4					
Asli 5					

Format data pengujian yang digunakan dalam penelitian ini berbentuk tabel yang menyatakan persentase tingkat kemiripan dokumen kode sumber asli dengan dokumen kode sumber modifikasi sesuai dengan kriteria pengujian. Dokumen kode sumber yang diterima sistem merupakan dokumen dengan format ekstensi .java.

3.3.5. Menentukan Alat Bantu Penelitian

Alat bantu penelitian deteksi kemiripan kode sumber menggunakan AST dan algoritma Ratcliff/Obershelp yang akan digunakan dalam penelitian ini adalah sebagai berikut.

1. Perangkat Keras

   Processor : Intel® Core i5- 8250U-1.6Ghz Turbo 3.4Ghz

   RAM : 12 GB DDR4

   Storage : 256 GB SSD + 1 TB HDD

2. Perangkat Lunak

   Sistem Operasi : Windows 10 64-bit

   Teks Editor : Netbeans 8.2

Selain perangkat lunak di atas, terdapat tambahan perangkat lunak ketiga yaitu ANTLR (*Another Tool for Language Recoginition*) versi 4, yang

merupakan perangkat lunak yang dapat membuat *lexer* dan *parser* dari suatu kode menggunakan *grammar*. Aplikasi ini akan digunakan dalam penelitian sebagai alat bantu untuk mengkonversi kode sumber ke dalam bentuk *syntax tree*. *Grammar* yang akan digunakan dalam penelitian berupa *grammar* bahasa Java dengan format ekstensi .g4 yang didapat dari laman *GitHub*[3]. ANTLR kemudian akan memproses kode sumber masukan menggunakan *grammar* yang akan menghasilkan *lexer* dan *parser* dari kode sumber masukan. Nantinya, *lexer* yang mewakili tiap cabang pohon akan diambil ID tokennya untuk proses penilaian kemiripan.

3.3.6. Melakukan Pengujian Penelitian

Pengujian penelitian dilakukan sesuai dengan kriteria pengujian yang dibuat sebelumnya. Pengujian sistem dimulai dengan menerima masukan dokumen kode sumber untuk kemudian diolah dalam tahap pra-pengolahan. Selanjutnya, kode sumber dengan bantuan ANLTR akan diubah ke dalam bentuk AST. Selain itu kode sumber juga akan di ubah ke dalam bentuk *string* secara menyeluruh dengan menghapus seluruh spasi pada kode sumber tanpa diubah ke dalam bentuk AST. Kemudian, pengujian dilakukan dengan menggunakan algoritma Ratcliff/Obershelp untuk mendeteksi tingkat kemiripan dari kode sumber asli dengan kode sumber modifikasi dengan dua kriteria pengujian yang ditentukan.

[3] https://github.com/antlr/antlr4/blob/master/tool-testsuite/test/org/antlr/v4/test/tool/Java.g4

3.3.7. Melakukan Analisis dan Kesimpulan Hasil Pengujian Penelitian

Hasil dari pengujian penelitian berupa persentase kemiripan antara kode sumber asli dengan kode sumber modifikasi yang dinyatakan dalam bentuk persentase dengan rentang angka antara 0 hingga 1. Hasil pengujian disajikan seperti pada Tabel III-2. Variabel yang dianalisis dalam pengujian ini berupa tingkat persentase kemiripan antara kode sumber asli dengan kode sumber modifikasi menggunakan algoritma Ratcliff/Obershelp, dengan dua kriteria pengujian yaitu dengan kombinasi metode AST dan tanpa metode AST.

Tabel III-2. Hasil Pengukuran Tingkat Kemiripan Kode Sumber

Kode Sumber	Metode	Persentase Kemiripan			
		Modifikasi 1	Modifikasi 2	Modifikasi 3	Modifikasi 4
Asli 1	Tanpa AST Dengan AST				
Asli 2	Tanpa AST Dengan AST				
Asli 3	Tanpa AST Dengan AST				
Asli 4	Tanpa AST Dengan AST				
Asli 5	Tanpa AST Dengan AST				

Hasil pengujian sistem berupa persentase kemiripan antara dokumen kode sumber asli dan kode sumber modifikasi akan ditampilkan dalam Tabel III-2. Kolom metode dalam Tabel III-2 menyatakan pengolahan kode sumber sesuai dengan kriteria pengujian yaitu menggunakan metode AST dan tanpa menggunakan metode AST.

3.4. Metode Pengembangan Perangkat Lunak

Metode pengembangan yang digunakan dalam penelitian ini adalah metode *Rational Unified Process* (RUP). Pengembangan sistem deteksi kemiripan kode sumber dibagi ke dalam empat tahap, yaitu fase insepsi, fase elaborasi, fase

konstruksi dan fase transisi. Berikut merupakan tahapan pengembangan perangkat lunak yang akan dilakukan dalam tiap fasenya.

3.4.1. Fase Insepsi

Tahapan yang dilakukan dalam fase ini adalah sebagai berikut.

1. Pemodelan Sistem: Menentukan ruang lingkup dan batasan masalah.

2. Kebutuhan: Mendefinisikan spesifikasi perangkat lunak.

3. Analisis dan Perancangan: Melakukan analisis terhadap kebutuhan perangkat lunak termasuk di dalamnya kebutuhan fungsional dan non fungsional dari spesifikasi perangkat lunak.

4. Implementasi: Membuat seluruh rancangan sistem ke dalam bentuk diagram *use-case*.

3.4.2. Fase Elaborasi

Tahapan yang akan dilakukan dalam fase ini adalah sebagai berikut.

1. Pemodelan Sistem: Membuat rancangan antarmuka (*interface*) sistem.

2. Kebutuhan: Menentukan spesifikasi dari sistem.

3. Analisis dan Perancangan: Membangun model *activity diagram* dan *sequence diagram* dari rancangan sistem.

4. Impelementasi: Membuat program berdasarkan diagram yang ditentukan sebelumnya.

3.4.3. Fase Konstruksi

Tahapan yang akan dilakukan dalam fase ini adalah sebagai berikut.

1. Pemodelan Bisnis: Menentukan bahasa pemrograman yang akan membangun sistem.

2. Kebutuhan: Menentukan kebutuhan sistem sesuai dengan fungsi yang telah ditentukan.

3. Analisis dan Perancangan: Membangun tampilan antar-muka sistem.

3. Implementasi: Membangun sistem dengan membuat program menggunakan bahasa pemrograman yang telah ditentukan.

3.4.4. Fase Transisi

Tahapan yang akan dilakukan dalam fase ini adalah sebagai berikut.

1. Pemodelan Sistem: Menentukan pengujian terhadap sistem.

2. Kebutuhan: Menentukan alat bantu pengujian terhadap sistem.

3. Analisis dan Perancangan: Merancang kasus penggunaan selama pengujian sistem.

4. Implementasi: Melaksanakan pengujian terhadap sistem menggunakan kasus penggunaan yang telah ditentukan.

3.5. Manajemen Proyek Perangkat Lunak

Rencana manajemen proyek penelitian merupakan perencanaan aktivitas penelitian dari tahap awal hingga selesai. Perencanaan aktivitas pada penelitian ini akan menggunakan Gantt Chart seperti pada Tabel III-3.

Tabel III-3. Rencana Manajemen Proyek Penelitian

No	Uraian kegiatan	Tahun 2021 bulan ke-		Tahun 2022 bulan ke-			
		11	12	1	2	3	4
1	**Melakukan Pengumpulan Data**						
a	Mengumpulkan data						
b	Melakukan pra-pengolahan data						
c	Melakukan modifikasi data sesuai skenario						
d	Tersedia dokumen hasil tahapan penelitian						
2	**Rekayasa Perangkat Lunak**						
2.1	**Insepsi**						
a	Menentukan pemodelan bisnis						
b	Menentukan kebutuhan pengguna						
c	Menentukan kebutuhan sistem						
2.2	**Elaborasi**						
a	Menentukan spesifikasi sistem						
b	Membangun model *acitvity diagram* dan *sequence diagram*						
c	Membangun rancangan tampilan antar-muka						
2.3	**Konstruksi**						
a	Membangun Model *Class Diagram*						
b	Membangun Sistem (impelementasi kode)						
c	Perbaikan Sistem						
2.4	**Transisi**						
a	Melakukan pengujian awal terhadap sistem						
b	Tersedia dokumen hasil tahapan penelitian						
3	**Melakukan Pengujian Penelitian Terhadap Sistem**						
a	Membuat rancangan hasil pengujian dalam penelitian						
b	Melakukan pengujian final terhadap sistem						
c	Tersedia dokumen hasil penelitian						
4	**Melakukan Analisis dan Kesimpulan dari Hasil Pengujian**						
a	Melakukan analisis terhadap hasil pengujian penelitian						
b	Membuat kesimpulan dan saran terhadap hasil pengujian penelitian						
c	Tersedia dokumen hasil penelitian						

3.6. Kesimpulan

Pada bab ini telah dibahas tentang proses pengumpulan data yang digunakan sebagai bahan uji perangkat lunak, tahapan penelitian, metode pengembangan perangkat lunak yang akan digunakan serta kriteria pengujian penelitian yang akan dilakukan terhadap sistem.

DAFTAR PUSTAKA

Agrawal, M., & Sharma, D. K. (2017). A state of art on source code plagiarism detection. *Proceedings on 2016 2nd International Conference on Next Generation Computing Technologies, NGCT 2016*, *October*, 236–241. https://doi.org/10.1109/NGCT.2016.7877421

Anwar, A. (2014). A Review of RUP (Rational Unified Process). *International Journal of Software Engineering*, *5*(2), 8–24. http://www.cscjournals.org/library/manuscriptinfo.php?mc=IJSE-142

Dang, S., & Ahmad, P. H. (2014). Text Mining : Techniques and its Application. *IJETI International Journal of Engineering & Technology Innovations*, *1*(December 2014), 22–25. www.ijeti.com

Firdaus, A., Ernawati, & Vatresia, A. (2014). Aplikasi Pendeteksi Kemiripan pada Dokumen Teks Menggunakan Algoritma Nazief & Andriani Dan Metode Cosine Similirity. *Jurnal Teknologi Informasi*, *10*(April), 96–109.

Fischer, G., Lusiardi, J., & Von Gudenberg, J. W. (2007). Abstract syntax trees - And their role in model driven software development. *2nd International Conference on Software Engineering Advances - ICSEA 2007, Icsea*. https://doi.org/10.1109/ICSEA.2007.12

Gipp, B., & Meuschke, N. (2011). Citation pattern matching algorithms for citation-based plagiarism detection: Greedy citation tiling, citation chunking and longest common citation sequence. *DocEng 2011 - Proceedings of the 2011 ACM Symposium on Document Engineering*, 249–258.

https://doi.org/10.1145/2034691.2034741

Ilyankou, I. (2014). Comparison of Jaro-Winkler and Ratcliff / Obershelp algorithms in spell check. *Ieee*, *May*, 1–35.

Jalal, A. F. D., & Prasetya, E. B. (2014). *Deteksi Similarity Source Code Menggunakan Metode Deteksi Abstract Syntax Tree*. *November*, 1–8.

Joane, Y., Sinsuw, A., & Jacobus, A. (2017). Rancang Bangun Aplikasi Deteksi Kemiripan Dokumen Teks Menggunakan Algoritma Ratcliff/Obershelp. *E-Journal Teknik Informatika*, *11*.

Kaučič, B., Sraka, D., Ramsăk, M., & Krašna, M. (2010). Observations on plagiarism in programming courses. *CSEDU 2010 - 2nd International Conference on Computer Supported Education, Proceedings*, *2*, 181–184. https://doi.org/10.5220/0002800701810184

Smith, C. J. (2012). Rules of conduct: plagiarism. *Ethical Behaviour in the E-Classroom*, 45–57. https://doi.org/10.1016/b978-1-84334-689-0.50005-7

Sraka, D., & Kaučič, B. (2009). Source code plagiarism. *Proceedings of the International Conference on Information Technology Interfaces, ITI*, 461–466. https://doi.org/10.1109/ITI.2009.5196127