**TESIS - KI142502**

# IDENTIFIKASI PELUANG REFAKTORISASI PADA KASUS GOD CLASS BERDASARKAN KESAMAAN KONSEP


Widhy Hayuhardhika Nugraha Putra

5112201044

DOSEN PEMBIMBING

Dr.Ir. Siti Rochimah, MT.

NIP. 196810021994032001

Rizky Januar Akbar, S.Kom, M.Eng

NIP. 198701032014041001

PROGRAM MAGISTER
REKAYASA PERANGKAT LUNAK
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017


THESES-KI142502

# GOD CLASS REFACTORING OPORTUNITY BASED ON CONCEPT SIMILARITY

Widhy Hayuhardhika Nugraha Putra

5112201044

SUPERVISOR

Dr.Ir. Siti Rochimah, MT.

NIP. 196810021994032001

Rizky Januar Akbar, S.Kom, M.Eng

NIP. 198701032014041001

MASTER PROGRAM
SOFTWARE ENGINEERING
INFORMATICS ENGINEERING
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

# LEMBAR PENGESAHAN TESIS

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom.)
di
Institut Teknologi Sepuluh Nopember Surabaya

oleh:
WIDHY HAYUHARDHIKA NUGRAHA PUTRA
Nrp. 5112201044

Dengan judul :
PENYUSUNAN KEMBALI STRUKTUR KELAS PADA KASUS GOD CLASS BERDASARKAN KESAMAAN KONSEP

Tanggal Ujian : 24-1-2017
Periode Wisuda : 2017 Ganjil

Disetujui oleh:

Dr. Ir. Siti Rochimah, M.T
NIP. 196810021994032001
(Pembimbing 1)

Rizky Januar Akbar, S.Kom, M.Eng
NIP. 198701032014041001
(Pembimbing 2)

Daniel Oranova Siahaan, S.Kom, M.Sc, PD.Eng.
NIP. 197411232006041001
(Penguji 1)

Sarwosri, S.Kom, M.T
NIP. 197608092001122001
(Penguji 2)

Nurul Fajrin Ariyani, S.Kom, M.Sc
NIP. 198607222015042003
(Penguji 3)

an. Direktur Program Pascasarjana
Asisten Direktur

Prof. Dr. Ir. Tri Widjaja, M.Eng.
NIP. 19611021 198603 1 001

Direktur Program Pasca Sarjana,

Prof.Ir.Djauhar Manfaat, M.Sc., Ph.D.
NIP. 196012021987011001

**IDENTIFIKASI PELUANG REFAKTORISASI PADA KASUS GOD CLASS BERDASARKAN KESAMAAN KONSEP**

Nama Mahasiswa : Widhy Hayuhardhika Nugraha Putra
NRP : 5112201044
Pembimbing : Dr. Ir. Siti Rochimah, MT
Rizky Januar Akbar, S.Kom, M.Eng

# ABSTRAK

Sebuah kelas dalam Pemrograman Berbasis Objek (PBO) harus memiliki desain yang spesifik untuk menangani dan mengimplementasikan sebuah konsep. Kelas harus memiliki tanggungjawab spesifik terhadap sebuah konsep yang diimplementasikan dalam operasi. Pada pengembangannya, seringkali pengembang perangkat lunak tidak memperhatikan desain konsep kelas sehingga kelas menjadi berkembang dan memiliki implementasi yang kompleks atau biasa disebut dengan "God Class". Dalam hal ini diperlukan metode untuk mengidentifikasi peluang refaktorisasi God Class menjadi beberapa kelas baru yang nantinya akan direfaktorisasi menggunakan metode refaktorisasi kelas seperti ExtractClass. Pada penelitian sebelumnya, telah diusulkan metode identifikasi peluang refaktorisasi kelas menggunakan tiga pembobotan yaitu SSM (*Structural Similariy between Methods*), CDM (*Call based Dependency between Methods*) dan CSM (*Conceptual Similarity between Methods*). Perhitungan CSM telah dilakukan dengan pendekatan LSI (*Latent Semantic Indexing*). Namun metode penilaian kesamaan konsep tersebut tidak dapat mendeteksi kesamaan konsep pada kalimat penyusun nama operasi. Kecenderungan pengembang menggunakan pilihan kata yang memiliki makna dan penggunaan kalimat kerja dalam membentuk nama operasi menjadikan peluang pada penelitian ini untuk dikembangkannya perhitungan CSM menggunakan pendekatan semantik dan *word dependency* sebagai lokasi konsep pada nama operasi.

Penelitian ini mengajukan metode dalam mengidentifikasi peluang refaktorisasi kelas dengan menghitung tingkat kedekatan konsep pada operasi. Operasi pada kelas diekstrak menjadi *corpus* dan dihitung nilai kedekatannya dengan operasi yang lain menggunakan SSM, CDM, dan CSM. Pada perhitungan CSM dilakukan penilaian kesamaan konsep secara semantik menggunakan kedekatan konsep term-term penyusun korpus menggunakan library *Standford Dependency*. Dimana untuk menghitung kedekatan konsep antar operasi diusulkan menggunakan pendekatan *vector space models* dengan memodifikasi pembobotan TF-IDF dengan *word distance* dan *word dependency*. Kemudian nilai kedekatan antar operasi tersebut direpresentasikan dalam bentuk graf dan dilakukan pemotongan graf menggunakan metode *Max-Flow Min-Cut* untuk mendapatkan identifikasi operasi yang harus dipisahkan dari kelas *God Class*.

Dengan mekanisme tersebut diatas maka pemrogram mendapatkan rekomendasi peluang refaktorisasi kelas dengan mudah. Hasil penelitian ini mencapai hasil dengan tingkat *precision* 0,708% dan *recall* 0,936%.

Kata Kunci: Penyusunan kembali kelas, Extract Class, God Class, Similaritas

[Halaman ini sengaja dikosongkan]

# GOD CLASS REFACTORING OPPORTUNITY IDENTIFICATION BASED ON CONCEPT SIMILARITY


Student Name : Widhy Hayuhardhika Nugraha Putra
Student Identity Number : 5112201044
Suvervisor : Dr. Ir. Siti Rochimah, MT
Rizky Januar Akbar, S.Kom, M.Eng


## ABSTRACT


A class in Object-Oriented Programming (OOP) must have a specific design for handle and implement a concept. The class must have a specific responsibility to a concept which is implemented in the methods. In its development, software developers often do not pay attention to the design concept of the class so that the class be growing and has a complex implementation or commonly called the "God Class". In this case the necessary methods to identify refactoring opportunities to split God Class into several new classes that will be refactored using methods such ExtractClass Class Refactoring. In previous studies, it has been proposed methods for identification of refactoring opportunities using the three weighting which is SSM (Structural Similariy between Methods), CDM (Call-based Dependency between Methods) and CSM (Conceptual Similarity between Methods). CSM calculations have been done with the approach of LSI (Latent Semantic Indexing). But the concept of similarity assessment methods can not detect a similar concept to the sentences making up the name of the operation. The tendency of developers using choice words have meaning and use of the phrase work in shaping the operation name on this research creates the opportunity for the development of CSM calculation using semantic approach and word dependency as a concept on the location name of the operation.

This study propose a method for identifying refactoring opportunities by identifying the concept location of methods. Methods of classes will be extracted into a corpus and calculated the value of its similarity to other operations using the SSM, CDM, and CSM. In the calculation of CSM assessment semantically similar concept of using concept similarity detection using StandfordDependency. Where to calculate the proposed operation to the similarities between the approach vector space models by modifying the TF-IDF weighting distance and word by word dependency. Then the similarity between these operations represented in the form of graphs and graph cuts made using Max-Flow Min-Cut to get identification operations must be separated from God class Class.

With proposed method, software developer can easily get recomendation for class refactoring opportunities. The result of this research is *precision* 0,708% and *recall* 0,936%.



Keywords : Class Refactoring, Extract Class, God Class, Similarity

[Halaman ini sengaja dikosongkan]

# KATA PENGANTAR

Puji syukur kepada Allah SWT atas segala berkah dan anugerah-Nya sehingga penulis dapat menyelesaian tesis yang berjudul "Identifikasi Peluang Refaktorisasi Pada Kasus God Class Berdasarkan Kesamaan Konsep".

Tesis ini disusun untuk memenuhi sebagian persyaratan untuk memperoleh gelar Magister Komputer di Institut Teknologi Sepuluh Nopember Surabaya. Tesis ini dapat terselesaikan tidak lepas dari bantuan dan dorongan yang sangat berharga dari berbagai pihak. Oleh sebab itu penulis mengucapkan terimakasih dan penghargaan yang sebesar-besarnya kepada berbagai pihak sebagai berikut.

1. Bapak Waskitho Wibisono, S.Kom, M.Eng, Ph. D, selaku Ketua Program Studi Pascasarjana Teknik Informatika Institut Teknologi Sepuluh Nopember.
2. Ibu Dr. Ir. Siti Rochimah, MT sebagai pembimbing I yang telah banyak meluangkan waktu dan pikiran dalam membimbing penulis untuk menyelesaikan tesis ini.
3. Bapak Rizky Januar Akbar, S.Kom., M.Eng. selaku pembimbing II yang telah membantu dalam penyusunan tesis dan karya tulis ilmiah.
4. Keluarga penulis, Papa, Mama, Istri dan Anak yang selalu memberikan dukungan mental dan spiritual selama penulis menyelesaikan studinya di Institut Teknologi Sepuluh Nopember.
5. Semua pihak yang telah membantu penulis dalam menyelesaikan tesis ini

Penulis menyadari bahwa dalam laporan tesis ini masih banyak kekurangan, oleh karena itu masukan dan saran yang membangun demi perbaikan dan pengembangan tesis ini sangat penulis harapkan. Akhir kata semoga tesis ini bermanfaat bagi para pembaca dalam pengembangan ilmu pengetahuan di negeri ini.

Surabaya, 10 Januari 2017

Penulis

# DAFTAR ISI

# DAFTAR GAMBAR

# DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

# BAB 1
# PENDAHULUAN

## 1.1 Latar Belakang

Pemrograman berbasis objek merupakan konsep pemrograman yang berbasis tanggungjawab, dimana pengembang perangkat lunak akan menentukan tanggungjawab masing-masing kode yang dibuat dan direpresentasikan pada operasi dan kelas. Sebuah kelas umumnya menggambarkan objek dalam perangkat lunak yang dibangun dan kelas-kelas yang dibentuk memiliki tanggungjawab masing-masing yang direpresentasikan dalam bentuk operasi. (Coad & Yourdon, 1991).

Dalam siklus pengembangan perangkat lunak, seringkali terjadi evolusi terhadap kode sumber berupa penambahan operasi pada kelas-kelas yang dilakukan oleh pengembang perangkat lunak. Penambahan yang dilakukan oleh pengembang perangkat lunak ini kadang menjadi tidak seusai dengan konsep awal dibentuknya kelas, sehingga konsep dari kelas yang dibangun menjadi menyimpang. Penyimpangan terhadap desain kelas ini terjadi karena tanggungjawab yang dimiliki kelas tersebut berevolusi dan kadang menjadi bias dan terlalu kompleks atau disebut dengan God Class. Untuk mengurangi God Class pada kode sumber, perlu dilakukan refaktorisasi atau penyusunan kembali kelas (Opdyke, 1992). Refaktorisasi merupakan proses menunjukkan peluang penyusunan kembali struktur kode sumber dengan tujuan untuk memperbaiki desain dan mempermudah dalam melakukan pengembangan selanjutnya.

Salah satu metode dalam refaktorisasi adalah dengan menggunakan metode *Extract Class* (Fowler, 1999) yaitu metode yang melakukan pemindahan operasi dalam kelas yang kompleks menjadi kelas lain sehingga kelas baru yang terbentuk tidak terlalu besar dan lebih spesifik. Dalam melakukan refaktorisasi, perlu diketahui terlebih dahulu operasi mana yang menyebabkan kelas menjadi God Class atau operasi mana yang tidak sesuai dengan konsep kelas dan perlu dipindahkan. Untuk mengetahui operasi mana yang perlu dipindahkan diperlukan sebuah metode identifikasi peluang refaktorisasi. Penelitian ini berfokus pada identifikasi peluang refaktorisasi dengan mengukur kedekatan konsep operasi berdasarkan penghitungan struktural dan semantik.

Penelitian yang paling dekat dengan penelitian ini adalah yang dilakukan oleh Bavota dkk (Bavota, et al., 2011). Mereka menggabungkan metode struktural dan semantik untuk mengukur kohesi kelas dan mengidentifikasi peluang refaktorisasi pada sebuah God Class. Pada penelitian tersebut digunakan pendekatan SSM (*Structural Similarity between Methods*) , CDM (*Call-Based Dependency between Methods*), dan CSM (*Conceptual Similarity between Methods*). Kemudian berdasarkan hasil perhitungan tersebut dilakukan pemotongan kelas God Class menjadi kelas baru menggunakan algoritma *MaxFlow-MinCut,* dengan cara merepresentasikan keterhubungan operasi dengan operasi lainnya dalam sebuah graf.

Yang dilakukan oleh Bavota dkk untuk mengidentifikasi peluang pembentukan kelas baru adalah dengan menghitung nilai kedekatan antar-operasi dalam sebuah kelas God Class menggunakan tiga unsur. Pada parameter pertama yaitu SSM, mereka menghitung kedekatan kelas dengan menghitung rasio penggunaan variabel yang sama antara dua operasi, jika semakin banyak variabel yang sama digunakan bersama, maka semakin tinggi bobot kedekatan operasi tersebut. Berikutnya juga dihitung unsur pemanggilan operasi, dimana jika sebuah operasi sering dipanggil oleh sebuah operasi yang lain, maka semakin tinggi bobot kedekatan operasi tersebut. Kemudian yang terakhir adalah menggunakan pendekatan LSI untuk menghitung bobot CSM, dimana semakin banyak dua buah operasi menggunakan term yang sama, maka akan semakin tinggi nilai kedekatan diantara dua operasi tersebut.

Penelitian ini mengusulkan perbaikan dari metode yang diusulkan oleh Bavota dkk (Bavota, et al., 2011). Pada penelitian sebelumnya, Bavota menggunakan pendekatan LSI pada perhitungan CSM dengan menghitung kemunculan kata yang sama sebagai faktor kedekatan semantik. Bavota tidak memperhitungkan kedekatan makna dari dua term yang dibandingkan, hanya menggunakan pendekatan kesamaan sintaksis. Misalkan terdapat operasi dengan nama *getParentName* dan *getFatherName* akan dianggap menjadi kata yang sama sekali berbeda. Bavota dkk juga tidak melakukan pemecahan terhadap *camelCase*, *snake_case*, dan *MixedCase* menjadi kata-kata yang terpisah. Sebagai contoh, mereka mencatat nama operasi *getParentName* sebagai sebuah kesatuan yang utuh. Sehingga jika ada nama operasi atau variabel yang menggunakan kata *name* atau *parent* tidak akan teridentifikasi

sebagai term yang sama, dimana artinya tidak terhitung dekat secara semantik. Sedangkan dalam lokasi konsep, pemecahan nama variabel atau fungsi merupakan hal utama dalam mendapatkan makna dalam sebuah kode. Jika menggunakan lokasi konsep, dalam fungsi *getParentName* mengandung dua konsep yaitu *parent* dan *name.* Sehingga penelitian ini menggunakan keterhubungan antar kata untuk mendapatkan perhitungan kedekatan operasi secara semantik yang lebih akurat.

Penelitian ini juga memperbaiki metode pencocokan term yang muncul dengan menggunakan *word dependency* berdasarkan kamus *library* Standford NLP. Sehingga penilaian kedekatan konsep tidak hanya berdasarkan kemunculan term, namun juga diukur dari kedekatan jarak makna antar term berdasarkan kamus. Sebagai contoh, term *parent* dan *father* akan dihitung sebagai term yang memiliki kedekatan walaupun tidak sama jika dilakukan pencocokan sintaksis.

Langkah dalam penelitian ini yaitu dengan menggunakan *dataset* yang digunakan juga dalam penelitian Bavota dkk. Bavota dkk telah menggunakan kode sumber terbuka dari beberapa proyek perangkat lunak berbahasa Java. Kemudian kode sumber tersebut diuji dengan metode yang mereka usulkan. Disamping itu, mereka juga telah mengujikan dataset tersebut kepada pakar rekayasa perangkat lunak untuk dibandingkan hasilnya. Pada penelitian ini dilakukan pengujian terhadap dataset yang sama dengan menggunakan metode TF-IDF termodifikasi *Word Dependency*. Pada penelitian ini dibuat sebuah alat untuk mengidentifikasi peluang pemecahan God Class menjadi kelas-kelas baru. Alat yang dibangun berupa *Plugin* Eclipse untuk mempermudah perhitungan dan pembacaan kelas yang ada pada kode sumber.

## 1.2 Perumusan Masalah

Masalah pada penelitian ini adalah bagaimana mengidentifikasi peluang penyusunan kembali kelas pada *God Class* dengan mempertimbangkan kemiripan konsep antar operasi secara semantik.

## 1.3 Tujuan dan Manfaat Penulisan

Tujuan penelitian yang ingin dicapai adalah untuk mengidentifikasi peluang refaktorisasi kelas secara otomatis dalam penyelesaian kasus God Class

menggunakan metode Extract Class dengan menggunakan pendekatan kemiripan konsep secara semantik.

Manfaat yang diberikan adalah agar seorang pengembang perangkat lunak lebih mudah dalam mengidentifikasi peluang penyusunan kembali struktur kelas.

## 1.4 Batasan Masalah

Dalam penelitian ini, pembahasan yang dicakup adalah sebagai berikut:

a. Penelitian ini dilakukan dengan membuat program bantu identifikasi peluang refaktorisasi pada sebuah kode sumber yang berupa sebuah struktur kelas.

b. Bahasa pemrograman yang digunakan untuk pengujian adalah Bahasa Java dengan istilah-istilah yang berbahasa Inggris.

# BAB 2
# KAJIAN PUSTAKA

## 2.1 God Class

God Class merupakan desain kelas yang menggunakan gaya prosedural dengan banyak konsep didalamnya, dimana objek objek lainnya hanya menangani data atau menangani proses yang sederhana. (Brown, et al., 1998).

Untuk menyelesaikan permasalahan God Class perlu dilakukan Refaktorisasi. Refaktorisasi adalah tindakan yang dilakukan untuk menyederhanakan sebuah God Class sehingga menjadi kelas yang memiliki kompleksitas rendah dengan cara mendistribusikan atribut atau operasi kedalam kelas yang lain. (Brown, et al., 1998)

### 2.1.1 Ciri God Class

Ciri dari God Class adalah sebagai berikut: (Brown, et al., 1998)

- Sebuah kelas yang memiliki banyak atribut, operasi atau keduanya. Sebuah kelas dengan 60 atau lebih attribut dan operasi diindikasikan sebagai Blob. (Akroyd, 1996)
- God Class umumnya memiliki tingkat kohesi rendah dan tingkat keterkaitan yang tinggi.
- Kumpulan atribut dan operasi tidak terhubung yang terenkapsulasi dalam sebuah kelas
- Sebuah *class controller* yang memiliki relasi pada kelas data objek yang sederhana.
- Ketiadaan konsep pemrograman berbasis objek yang menyebabkan sebuah program utama berjalan hanya di seputar kelas utama.

### 2.1.2 Penyebab Munculnya God Class

Penyebab umum munculnya God Class ada beberapa kemungkinan, antara lain: (Brown, et al., 1998)

- Pengembang kurang menguasai konsep arsitektur Pemrograman berbasis Objek.

- Kurangnya konsep desain sistem, sehingga dalam pengembangannya, sistem akan berevolusi ke arah yang tidak terkontrol.
- Pengembang tidak menerapkan arsitektur konsep sistem yang telah direncanakan.
- Pengembang terlalu fokus dalam membangun sebuah kelas dalam proyek yang menggunakan metode iteratif tanpa mengindahkan kemungkinan membentuk kelas baru untuk sebuah konsep yang berbeda.
- Keterpaksaan dari kebutuhan. Kadang hasil rekayasa kebutuhan menyebabkan keharusan membentuk struktur prosedural dalam arsitektur sistem yang akan dibangun.

### 2.1.3 Kelemahan God Class

Adanya God Class membuat kualitas dari sebuah program menjadi turun. Adapun kelemahan dari sebuah program yang mengandung God Class adalah: (Brown, et al., 1998)

- God Class terlalu kompleks untuk digunakan kembali di kelas yang lain atau di bagian kode yang lain. Penggunaan kembali sebuah God Class atau operasi di dalamnya akan menumbulkan ketidak-efisienan dalam kode sumber dan menimbulkan kompleksitas tinggi.
- God Class dapat menimbulkan beban penggunaan berlebihan untuk *memory* atau sumberdaya yang lain.
- God Class akan membatasi kebutuhan untuk memodifikasi kelas tersebut tanpa menimbulkan perubahan pada fungsionalitas lain dalam kelas tersebut. Perubahan pada God Class akan selalu menimbulkan efek pada perangkat lunak secara keseluruhan.

## 2.2 Penelitian Terdahulu Untuk Mengidentifikasi Peluang Refaktorisasi

Dalam kasus God Class perlu dilakukan refaktorisasi dengan mendistribusikan beberapa operasi-operasi yang memiliki tanggungjawab yang tercampur atau operasi-operasi yang tidak sesuai dengan konsep kelas itu sendiri kedalam kelas yang baru. Refaktorisasi ini dilakukan untuk memindahkan operasi-operasi tersebut

keluar dari kelas God Class dan membentuk kelas baru dengan konsep yang lebih spesifik. Dengan demikian, kompleksitas dari kelas tersebut menjadi berkurang dan sekaligus meningkatkan kohesivitas dari kelas tersebut. Domain penelitian untuk restrukturisasi God Class ini disebut dengan *Class Refactoring* (Fowler, 1999). Beberapa penelitian terakhir seperti yang dilakukan oleh Fokaefs, dkk (Fokaefs, et al., 2012), dan (Bavota, et al., 2010) telah memanfaatkan kedekatan semantik antar-operasi sebagai salah satu parameter untuk mengidentifikasi peluang refaktorisasi.

Bavota dkk (Bavota, et al., 2010), mengusulkan teknik dekomposisi kelas sederhana dengan menghitung kohesi antar operasi dalam sebuah kelas untuk mengidentifikasi peluang refaktorisasi sebuah God Class. Metrik yang digunakan sama seperti yang digunakan pada penelitian berikutnya (Bavota, et al., 2011), yaiu dengan memperhitungkan SSM, CDM, dan CSM untuk menghitung kohesifitas antar-operasi. Nilai kohesifitas antar-operasi ini yang akan digunakan sebagai pertimbangan dalam melakukan refaktorisasi kelas menggunakan algoritma MaxFlow-MinCut.

Fokaefs dkk dalam penelitiannya menyebutkan bahwa identifikasi peluang refaktorisasi ExtractClass terdiri atas dua tahap. Pertama, dilakukan analisa terhadap keterhubungan diantara komponen dalam kelas tersebut sehingga dapat diukur jarak antarkomponen dalam kelas. Kemudian informasi keterhubungan kelas ini akan digunakan oleh algoritma tertentu (misalnya algoritma klasterisasi) untuk melakukan pemecahan kelas menjadi beberapa bagian. Kemudian langkah kedua adalah memverifikasi rekomendasi refaktorisasi kelas yang telah dihasilkan apakah dapat berfungsi dengan baik dan juga apakah kelas yang dihasilkan nantinya dapat berfungsi sesuai dengan kebutuhan program. Untuk mengimplementasikan rekomendasi refaktorisasi dengan metode ExtractClass, telah dibuat tools berupa *plugin* untuk Eclipse seperti JDeodorant (Fokaefs, et al., 2011).

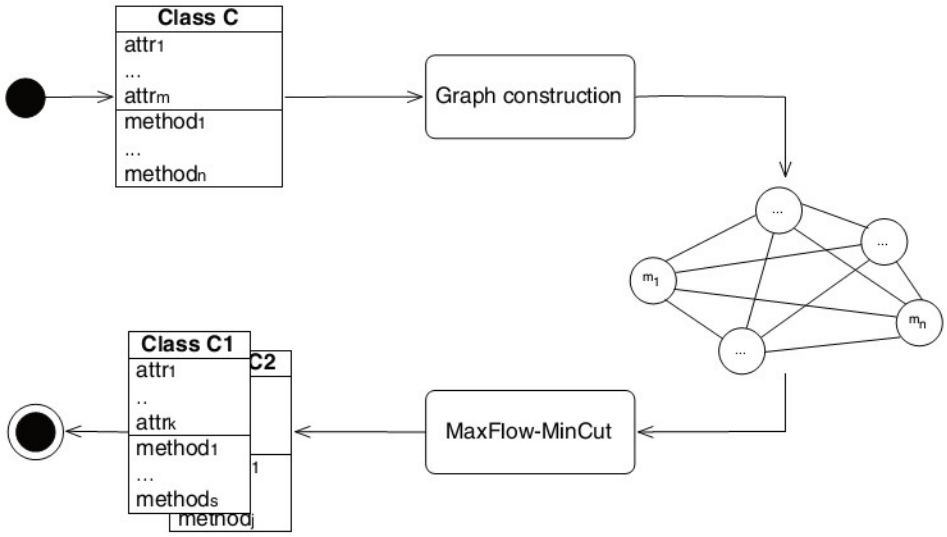
Pada penelitian ini, digunakan pendekatan struktural dan sintaksis dalam menilai kedekatan antar-operasi dalam kelas, kemudian digunakan metode *MaxFlow-MinCut* untuk menghasilkan rekomendasi refaktorisasi kelas. Pada penelitian ini digunakan pendekatan semantik untuk mengukur kedekatan konsep antar-operasi. Pendekatan semantik yang digunakan adalah dengan LSI (*Latent Semantic Indexing*) dengan mempertimbangkan kedekatan konsep operasi. Pada

penelitian ini tidak akan dibahas lebih jauh bagaimana verifikasi hasil refaktorisasi apakah dapat diimplementasikan ataukah tidak, penelitian ini hanya berfokus pada identifikasi peluang refaktorisasi God Class.

## 2.3 Identifikasi Peluang Refaktorisasi

Bavota, G, dkk (Bavota, et al., 2011) dalam penelitiannya mengusulkan metode identifikasi peluang refaktorisasi dengan menggunakan representasi Graf keterhubungan antar operasi. Metode yang diusulkan adalah dengan melakukan pembobotan kedekatan antaroperasi dengan pendekatan bobot SSM, CDM, dan CSM, kemudian dari graf yang terbentuk digunakan algoritma Max-Flow-Min-Cut untuk membentuk memotong graf keterhubungan kelas baru. Gambaran metode yang diusulkan oleh Bavota, dkk dengan algoritma Max-Flow Min-Cut ditampilkan pada gambar 2.1.

Algoritma yang diusulkan oleh Bavota dkk menggunakan graf untuk menentukan bobot keterhubungan antar operasi dan atribut. Bobot keterhubungan antar operasi dan atribut dalam kelas diukur dengan pendekatan struktural dan semantik. Kemudian graf yang terlah terbentuk diproses untuk mendapatkan kumpulan operasi-operasi dengan bobot keterhubungan.




**Gambar 2.1 Metode Identifikasi Peluang Extract Class (Bavota, et al., 2011)**

### 2.3.1 Bobot Kesamaan Antar Operasi

Menurut Bavota dkk, untuk membentuk graf keterhubungan antar operasi dalam algoritma Max-Flow Min-Cut digunakan metode pembobotan. Bobot diambil dari beberapa aspek yaitu:

a. *Structural Similarity between Methods* (SSM), merupakan pengukuran matrik kohesi kelas yang diusulkan oleh Gui dkk (Gui & Paul D., 2008). Metode ini menggunakan rasio jumlah variabel *instance* yang digunakan bersama antara dua operasi.

$$SSM(m_i, m_j) = \begin{cases} \dfrac{|I_i \cap I_j|}{|I_i \cup I_j|} & if\ |I_i \cup I_j| \neq 0; \\ 0 & otherwise. \end{cases} \quad \textbf{(2-1)}$$

dimana:

$I_i$ = Variabel *instance* yang dipanggil oleh operasi $m_i$

$I_j$ = Variabel *instance* yang dipanggil oleh operasi $m_j$

b. Pemanggilan operasi juga dapat digunakan sebagai faktor dalam menghitung kohesi kelas (Bavota, et al., 2011) dengan menggunakan metode CDM (*Call-based Dependencies between Methods*)

$$CDM(m_i, m_j) = \begin{cases} \dfrac{calls(m_i, m_j)}{calls_{in}(m_j)} & if\ calls_{in}(m_j) \neq 0; \\ 0 & otherwise. \end{cases} \quad \textbf{(2-2)}$$

dimana:

$calls(m_i, m_j)$ = jumlah pemanggilan yang dilakukan oleh operasi $m_i$ terhadap operasi $m_j$

$calls_{in}(m_j)$ = jumlah total pemanggilan yang masuk ke operasi $m_j$

c. *Conceptual Similarity between Methods* (CSM), digunakan untuk pendekatan semantik melalui komentar dan atribut atau konsep operasi yang sama (Poshyvanyk, et al., 2007).

$$CSM(m_i, m_j) = \frac{\overrightarrow{m_i} \cdot \overrightarrow{m_j}}{\|\overrightarrow{m_i}\| \cdot \|\overrightarrow{m_j}\|} \quad \textbf{(2-3)}$$

dimana:

$CSM(m_i, m_j)$ = bobot kemiripan semantik antara$m_i$ dan $m_j$

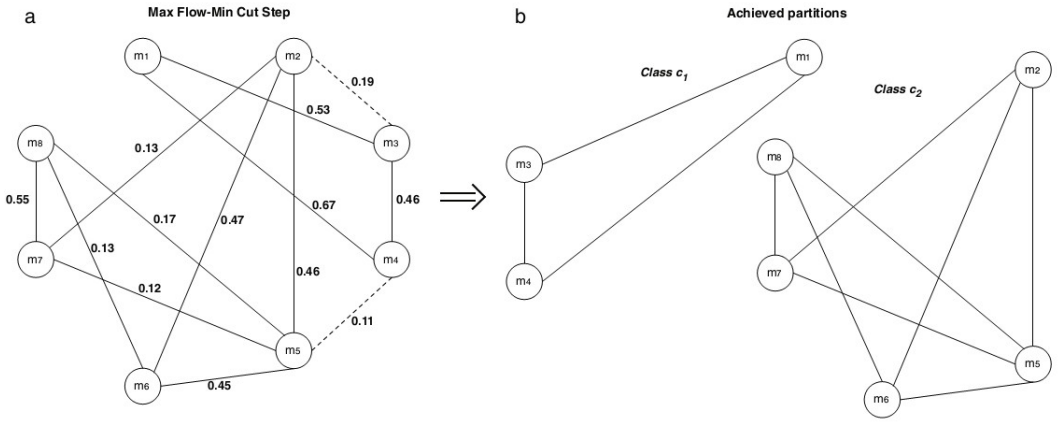
$\overrightarrow{m_x}$. = vektor kesesuaian operasi x

$\overrightarrow{\|m_i\|}$ = eucledian norm dari operasi x

Dari ketiga parameter bobot tersebut diatas (SSM, CDM, CSM) penentuan bobot final untuk mengukur keterhubungan antar-operasi ditentukan menggunakan rumus berikut:

$$w(m_i, m_j) = w_{SSM} \cdot SSM(m_i, m_j) + w_{CDM} \cdot CDM(m_i, m_j) + w_{CSM} \cdot CSM(m_i, m_j) \quad \textbf{(2-4)}$$

dimana $w_{SSM} + w_{CDM} + w_{CSM} = 1$ dan nilainya masing-masing adalah bobot untuk tiap pengukuran. Nilai ketiga parameter tersebut didapatkan nilai terbaik pada penelitian sebelumnya $\boldsymbol{w_{CSM} = 0,7, w_{SSM} = 0,2, w_{CDM} = 0,1.}$ (Bavota, et al., 2011). Kemudian dari hasil perhitungan graf keterhubungan antar operasi dilakukan penyaringan terhadap verteks yang membentuk graf. Dipilih sebuah ambang batas *minimum cohesion* = *α*. Verteks yang dibawah ambang batas nilai kohesifitas akan dihilangkan (dianggap 0). Operasi yang tidak saling terhubung akan membentuk kelas yang baru. Pemotongan graf menggunakan algoritma Max-Flow-Min-Cut digambarkan pada gambar 2-2 berikut:



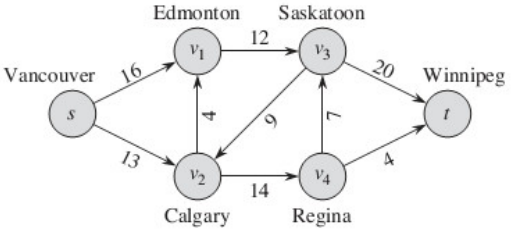

**Gambar 2.2 (a) graf berbobot yang menggambarkan keterhubungan antar operasi sebelum diklasterisasi. (b) graf berbobot setelah mengalami proses klasterisasi dengan ambang batas. (Bavota, et al., 2011)**

### 2.3.2 MaxFlow MinCut

Metode *MaxFlow MinCut* atau dikenal dengan Algoritma Ford-Fulkerson merupakan metode yang digunakan untuk mendapatkan dua subgraf dari graf yang merepresentasikan hubungan antar operasi dalam kelas.

**Maximum Flow**

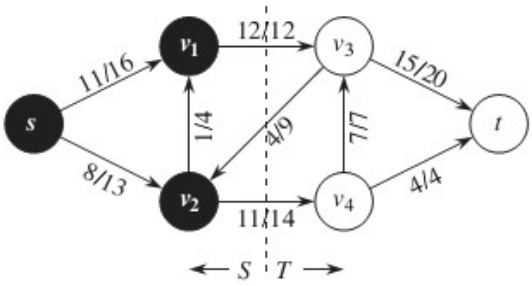
Algoritma Ford-Fulkerson digunakan untuk memecahkan masalah *Maximum Flow* pada Graf berarah dan berbobot, dimana pada graf tersebut ingin didapatkan jalur terpendek dari titik *source s* ke *sink t* dengan kapasitas (*flow*) maksimal yang dapat ditempuh. Contoh kasusnya adalah pada gambar 2.3 berikut, dimana sebuah perusahaan truk ingin mengetahui jalur paling optimal yang dapat digunakan untuk perjalanan antar kota, dimana masing-masing jalur memiliki kapasitas maksimal. Pada kasus ini ingin diketahui jalur optimal agar truk dapat membawa kapasitas maksimal dalam menempuh perjalanan. (Thomas H. Cormen, 2009)




**Gambar 2.3 Kasus Maximum Flow pada jalur truk (Thomas H. Cormen, 2009)**

**Minimum Cut**

Minimum Cut adalah metode Ford-Fulkerson untuk mencari titik potong graf dengan edge paling minimum untuk mendapatkan dua subgraf dengan kapasitas paling tinggi pada masing-masing graf seperti yang ditampilkan pada gambar 2-4 berikut. (Thomas H. Cormen, 2009).




**Gambar 2.4 Minimum Cut (Thomas H. Cormen, 2009)**

Pendekatan ini dapat digunakan dalam kasus pemecahan graf keterhubungan antar operasi dalam kelas dimana diharapkan pada masing-masing pemecahan masih memiliki keterhubungan antar operasi (kapasitas) yang paling tinggi. Metode Ford-Fulkerson digunakan untuk memecahkan masalah dalam graf berarah dan berbobot. sedangkan hasil dari perhitungan keterhubungan operasi adalah graf tidak berarah dan berbobot, sehingga perlu modifikasi dengan menambahkan vertek tambahan untuk dua arah pada masing-masing vertek keterhubungan antar dua operasi dengan nilai bobot yang sama (Bavota, et al., 2011).

### 2.3.3 *Latent Semantic Indexing* (LSI)

Penelitian yang dilakukan oleh Bavota dkk (Bavota, et al., 2011) menggunakan pendekatan LSI (*Latent Semantic Indexing*) untuk mengukur kemiripan sintaksis antar operasi dalam sebuah kelas dengan membentuk *corpus* yang terdiri dari nama operasi, *instance variable* dalam operasi dan *return value*. Kemudian *corpus* dari seluruh operasi dalam kelas digabung dan dilakukan tokenisasi (dipecah menjadi term-term). Selanjutnya kamus term yang terbentuk akan digunakan untuk menghitung setiap kemunculan term yang sama dari hasil tokenisasi dengan melakukan indeksasi semua istilah pembentuk operasi dalam kelas terlebih dahulu dan kemudian menghitung frekuensi kemunculan tiap term. Sebagai contoh, terdapat 2 operasi yaitu m1 : "`getFileDirectory`" dan m2 : "`openDirectory`", maka tabel frekuensi term ditampilkan pada tabel 2.1 berikut:

**Tabel 2-1 Perhitungan kemunculan term**

| | Get | File | Directory | Open |
|---|---|---|---|---|
| m 1 | 1 | 1 | 1 | 0 |
| m 2 | 0 | 0 | 1 | 1 |

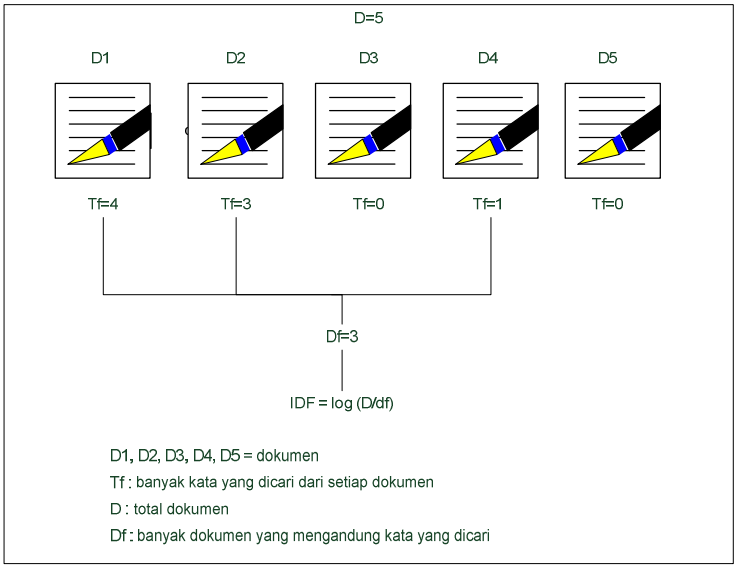
Dari tabel diatas maka kemiripan kalimat satu dengan dua jika dihitung dengan persamaan 2-3 adalah sebagai berikut:

$$CSM(m_1, m_2) = \frac{1 \times 0 + 1 \times 0 + 1 \times 1 + 0 \times 1}{\sqrt{1^2 + 1^2 + 1^2 + 0} \times \sqrt{0 + 0 + 0 + 1^2 + 1^2}} = \frac{1}{2{,}49} = 0{,}408$$

Sehingga kemiripan konseptual semantik dari operasi "`getFileDirectory`" dan "`openDirectory`" adalah 0,408 dimana skala kemiripan adalah 0 hingga 1.

## 2.4 TF-IDF

Salah satu algoritma *Text Mining* adalah algoritma TF/IDF. Algoritma TF/IDF digambarkan pada gambar 2.2 berikut. Algoritma ini digunakan untuk mencari tingkat kemiripan sebuah dokumen terhadap sebuah kata kunci atau term.




**Gambar 2.5 Algoritma TF-IDF**

TF-IDF digunakan untuk menghitung bobot term *corpus* dokumen operasi pada kelas. Bobot ini digunakan untuk mengukur nilai kemiripan antar operasi dalam kelas menggunakan *vector space models*. Rumus TF-IDF adalah seperti pada persamaan 2.5.

$$\boldsymbol{w(m_i, m_j) = tf_{i,j} * log \frac{2}{df_t}} \quad \textbf{(2-5)}$$

$tf_{i,j}$ = jumlah kemunculan term dalam operasi i dalam operasi j

$t$ = indeks term/kata

$m_i$ = operasi ke i

$m_j$ = operasi ke j

$df_t$ = jumlah dokumen yang mengandung kata kunci ke-t

Dan bobot ($w$) antar operasi dengan metode TF-IDF yang telah ternormalisasi dinyatakan dalam persamaan 2-6.

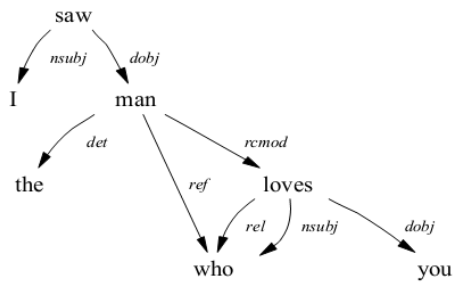
$$w(m_i, m_j) = \left\{\frac{tf_{i,j}}{\max tf_{i,j}}\right\} * log\left\{\frac{D}{df_t}\right\} \quad \textbf{(2-6)}$$

Dengan $t$ adalah indeks term, $tf$ adalah term frequency, dan $df_t$ adalah jumlah *method* yang mengandung term ke-$t$.

## 2.5 Standford Dependencies

Stanford Dependencies merupakan alat bantu representasi keterhubungan gramatikal antar kata dalam sebuah kalimat. Metode ini dikembangkan oleh de-Marneffe, dkk (2006). Metode ini memberikan informasi otomatis mengenai keterhubungan antar kata dalam sebuah kalimat, seperti subjek, predikat atau objek pada sebuah kalimat berbahasa Inggris.

Contoh ekstraksi keterhubungan antar kata pada kalimat berbahasa Inggris ditunjukkan pada gambar 2.5 berikut (Marie-Catherine de Marneffe, 2006):




**Gambar 2.6 Contoh keterhubungan antarkata dalam kalimat "I saw the man who loves you" (Marie-Catherine de Marneffe, 2006)**

```
dep - dependent
aux - auxiliary
auxpass - passive auxiliary
cop - copula
conj - conjunct
cc - coordination
arg - argument
subj - subject
nsubj - nominal subject
nsubjpass - passive nominal subject
csubj - clausal subject
comp - complement
obj - object
dobj - direct object
iobj - indirect object
pobj - object of preposition
attr - attributive
ccomp - clausal complement with internal subject
xcomp - clausal complement with external subject
compl - complementizer
mark - marker (word introducing an advcl)
rel - relative (word introducing a rcmod)
acomp - adjectival complement
agent - agent
ref - referent
expl - expletive (expletive there)
mod - modifier
advcl - adverbial clause modifier
purpcl - purpose clause modifier
tmod - temporal modifier
rcmod - relative clause modifier
amod - adjectival modifier
infmod - infinitival modifier
partmod - participial modifier
num - numeric modifier
number - element of compound number
appos - appositional modifier
nn - noun compound modifier
abbrev - abbreviation modifier
advmod - adverbial modifier
neg - negation modifier
poss - possession modifier
possessive - possessive modifier ('s)
prt - phrasal verb particle
det - determiner
prep - prepositional modifier
sdep - semantic dependent
xsubj - controlling subject
```

**Gambar 2.7 Derajat keterhubungan antar kata dalam *stanford dependencies* (Marie-Catherine de Marneffe, 2006)**

Dalam kalimat contoh "*I saw the man who loves you*" dengan metode *Standford Dependencies* akan dapat diidentifikasi keterhubungan antar katanya sebagai berikut:

```
nsubj (I, saw)                    ref (man, who)
dobj (saw, man)                   dobj (loves, you)
det (man, the)                    rel (loves, who)
rcmod (man, loves)
```

**Gambar 2.8 Keterhubungan antar kata dalam *Standford Dependencies* untuk kalimat "I saw the man who loves you" (Marie-Catherine de Marneffe, 2006)**

Dengan asumsi nama operasi dalam sebuah kelas umumnya menggunakan representasi sebuah frasa berbentuk kalimat, maka identifikasi keterhubungan antar kata dapat diimplementasikan pada nama operasi. Metode ini juga dapat digunakan untuk mencari lokasi konsep pada kalimat, yaitu kata yang berpredikat sebagai *root* atau pusat kalimat.

Standford telah menyediakan *library* dalam bahasa perograman Java untuk mengimplementasikan *Standford Dependencies* dalam mendeteksi keterhubungan antar kata dalam sebuah kalimat berbahasa Inggris. Dalam penelitian ini akan digunakan *library* Standford Core NLP dan Standford Dependency Parser untuk mengidentifikasi keterhubungan antar kata dalam kalimat. Dan mengidentifikasi kata inti dari kalimat penyusun nama operasi.

## 2.6 Kesamaan *Cosine*

Kesamaan *cosine* adalah metode untuk menghitung nilai cosinus sudut dari dua vektor, yaitu bobot (w) dari setiap operasi. Kesamaan *cosine* digunakan untuk menghitung nilai kemiripan diantara dua *corpus*, formula *similarity* yang dipakai adalah konsep *vector space models*, yang dinyatakan dalam persamaan 2.5.

$$sim(m_i, m_j) = \frac{\sum_n w_{i,j} \cdot w_{i,j}}{\sqrt{\sum_n w_{i,j}^2} \cdot \sqrt{\sum_n w_{i,j}^2}} \quad \textbf{(2-7)}$$

dimana:

*m* = *corpus* operasi ke x

$m_x$= bobot pada *corpus* operasi ke-x

*n* = jumlah term pada *corpus* operasi

Konsep kesamaan *cosine* ini akan digunakan untuk mencari nilai kemiripan konsep dari operasi dan membentuk graf keterhubungan antar operasi. Dalam metode yang diusulkan ini dilakukan penghitungan CSM menggunakan pendekatan kesamaan *cosine,* dimana bobot masing-masing operasi ditentukan juga dengan kedekatan Konsep Operasi.

### 2.6.1 Precision and Recall

Pada penelitian ini digunakan dataset dari penelitian sebelumnya sehingga rekomendasi yang dihasilkan oleh sistem akan dibandingkan dengan rekomendasi yang dihasilkan oleh pakar perangkat lunak. Kemudian hasil dari perbandingan digunakan untuk menentukan akurasi sistem dengan perhitungan *precision recall*. Untuk perhitungan *precision recall* digunakan definisi sebagai berikut:

1. *True Positive (TP)* merupakan rekomendasi pemindahan operasi yang dinyatakan oleh pakar rekayasa perangkat lunak dan juga direkomendasikan oleh sistem.
2. *True Negative (TN)* merupakan rekomendasi untuk tidak memindahkan operasi yang dinyatakan oleh pakar perangkat lunak dan juga tidak direkomendasikan oleh sistem.
3. *False Positive (FP)* merupakan rekomendasi untuk tidak memindahkan operasi yang dinyatakan oleh pakar perangkat lunak, namun direkomendasikan untuk dipindahkan oleh sistem.
4. *False Negative (FN)* merupakan rekomendasi untuk memindahkan operasi yang dinyatakan oleh pakar perangkat lunak namun tidak direkomendasikan untuk dipindahkan oleh sistem.

Setelah didapatkan TP, TN, FP dan FN kemudian nilai tersebut digunakan untuk menghitung Nilai *Accuracy*, *Precision* dan *Recall* dengan definisi sebagai berikut:

1. *Precision*, adalah tingkat ketepatan sistem dalam mengidentifikasi pemindahan operasi yang sesuai dengan pemindahan yang disarankan oleh pakar. Dimana rumus dari *precision* adalah sebagai berikut:

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP+TN}} \quad \textbf{(2-8)}$$

2. *Recall*, adalah tingkat presentase keberhasilan sistem dalam mengidentifikasi peluang pemindahan operasi sesuai dengan yang direkomendasikan oleh pakar. Dimana rumus *recall* adalah seperti pada persamaan 2-9:

**(2-9)**

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

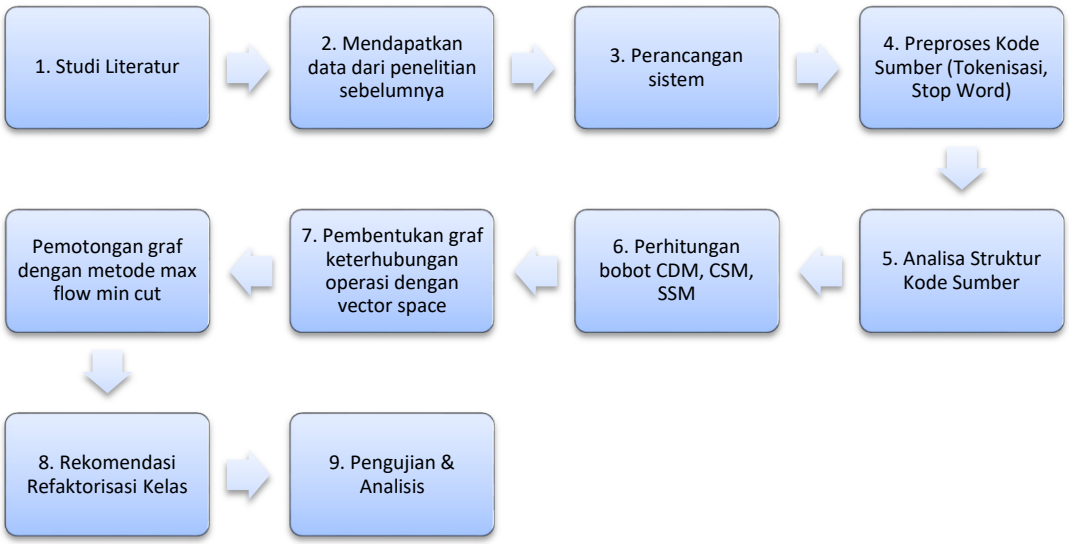
3. *F-Measure*, adalah variabel yang menyatakan performansi dari sistem yang diusulkan, dengan rumus seperti pada persamaan 2-10:

$$F - Measure_{c_i} = 2 \times \frac{precision_{c_i} \times recall_{c_i}}{precision_{c_i} + recall_{c_i}} \quad \textbf{(2-10)}$$

# BAB 3
# METODOLOGI PENELITIAN

## 3.1 Metodologi Penelitian

Adapun sistematika metodologi penelitian yang dilakukan adalah seperti yang digambarkan pada gambar 3.1.




**Gambar 3.1 Metodologi Penelitian**

1. Studi Literatur

Hal ini dilakukan untuk mendapatkan informasi dari literatur-literatur yang digunakan, serta perkembangan metode yang telah dilakukan dalam melakukan identifikasi diagram kasus penggunaan dari skenario.

2. Mendapatkan data dari penelitian sebelumnya

Hal ini dilakukan untuk mendapatkan data penelitian sebelumnya dan hasil analisa dari pakar untuk digunakan sebagai pembanding dalam pengujian.

3. Perancangan Sistem

Sistem yang dibangun adalah sistem yang memproses masukan berupa kode sumber kelas yang bersifat God Class dalam bentuk bahasa pemrograman Java dan akan dihasilkan luaran berupa rekomendasi refaktorisasi kelas tersebut.

4. Preproses Kode Sumber

Preproses kode sumber adalah kegiatan yang dilakukan program untuk membaca kode sumber yang diberikan sebagai data uji dari *file* berbentuk .java. Preproses ini meliputi: Tokenisasi, yaitu pemecahan kode sumber menjadi kata-kata atau token-token penyusun. *Stop Word Elimination*, yaitu proses menghilangkan kumpulan kata atau token yang dianggap tidak penting agar tidak ikut dihitung dalam pencarian bobot. Stemming, yaitu pengembalian kata atau token menjadi kata dasar.

5. Analisa Struktur Kode Sumber

Proses ini dilakukan untuk mendapatkan informasi struktural kode sumber seperti Line of Code (LOC), jumlah operasi, daftar nama operasi, daftar atribut kelas, nama kelas dan pemanggilan operasi.

6. Perhitungan bobot *CDM, CSM, SSM*

Hal ini dilakukan untuk menghitung bobot keterkaitan antaroperasi dalam sebuah kelas. Pada penelitian ini diusulkan perbaikan pada metode CSM dengan mempertimbangkan bobot struktur kalimat pada nama operasi yang menentukan nilai vektor TF-IDF untuk menghitung kemiripan semantik. Pada penelitian ini juga digunakan library *Standford Dependency* untuk menghitung tingkat kemiripan konsep kata penyusun nama operasi, tidak hanya menggunakan kesamaan sintaksis.

7. Pembentukan graf keterhubungan antar operasi dengan vector space

Proses penelitian ini dilakukan untuk mendapatkan gambaran keterhubungan antar operasi dengan bobot kombinasi dari *CDM, CSM, SSM*.

8. Pemotongan graf dengan algoritma Max-Flow Min-Cut

Algoritma ini memotong graf menjadi dua klaster berdasarkan kombinasi perhitungan bobot CDM, SSM dan CSM untuk mendapatkan klasterisasi susunan kelas baru yang lebih kohesif.

9. Rekomendasi refaktorisasi kelas

Rekomendasi ini diberikan berdasarkan hasil pemotongan graf keterhubungan antar operasi yang telah dibentuk menggunakan nilai ambang batas kohesifitas tertentu.

10. Pengujian dan Analisis Hasil

Hal ini dilakukan untuk menguji apakah sistem dapat memberikan rekomendasi refaktorisasi kelas yang sesuai dengan konsepnya.

## 3.2 Langkah Percobaan

### 3.2.1 Studi Literatur

Proses studi literatur adalah proses untuk mendapatkan referensi, informasi tentang masalah penelitian, dan metode-metode yang telah diusulkan sebelumnya. Studi literatur yang dilakukan adalah:

- Mempelajari Metode ExtractClass untuk refaktorisasi yang telah diusulkan sebelumnya
- Mempelajari pembobotan operasi CDM, CSM, SSM
- Mempelajari dan mengumpulkan referensi algoritma pembobotan dan pembuatan graf dengan TF-IDF ternormalisasi dan *vector space models*
- Mempelajari *library* Standford Dependency untuk menentukan root dari kalimat berbahasa Inggris.
- Mempelajari Metode Max-Flow-Min-Cut
- Mempelajari pembuatan Plugin Eclipse untuk membuat alat bantu pengujian metode yang diusulkan.

### 3.2.2 Mendapatkan data dari penelitian sebelumnya

Penelitian sebelumnya (Bavota, et al., 2011) telah menggunakan dataset yang diambil dari beberapa kelas dalam beberapa proyek perangkat lunak dengan kode sumber terbuka dengan judul : AgroUML, Eclipse, dan JHotDraw. Adapun versi dan ukuran masing-masing proyek perangkat lunak dijelaskan pada tabel berikut:

**Tabel 3-1 Tabel versi dan ukuran data penelitian sebelumnya**

| No. | Nama Proyek | Nama Kelas | Jumlah Operasi |
|---|---|---|---|
| 1 | ApacheHDSQLDB | UserManager.java | 13 |
| 2 | ApacheHDSQLDB | Database.java | 40 |
| 3 | ApacheHDSQLDB | Select.java | 14 |
| 4 | ArgoUML | Import.java | 10 |
| 5 | ArgoUML | FileGeneratorAdapter.java | 11 |
| 6 | JEdit | JeditTextArea.java | 187 |
| 7 | JFreeChart | JfreeChart.java | 24 |

| No. | Nama Proyek | Nama Kelas | Jumlah Operasi |
|---|---|---|---|
| 8 | JFreeChart | NumberAxis.java | 20 |
| 9 | Xerces | XMLDTDValidator.java | 69 |
| 10 | Xerces | XMLSerializer.java | 25 |
| 11 | Xerces | DomParserImpl.java | 72 |

Contoh data dari penelitian sebelumnya adalah kelas `UserManagement.java` seperti pada gambar 3.2

Pada contoh kasus kelas UserManagement, terdapat kelas yang memiliki *responsibility* tercampur dari 3 kelas yaitu: `User`, `Teaching`, dan `Role` sehingga bisa dikatakan bahwa kelas `UserManagement` tersebut adalah God Class.

### 3.2.3 Praproses Kode Sumber

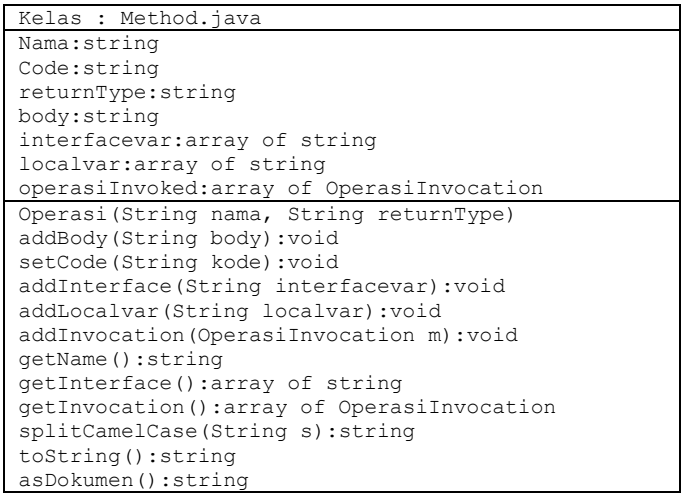
Praproses dilakukan pada kode sumber untuk mengambil kata-kata yang memiliki makna dari dalam suatu kode sumber. Praproses dilakukan terhadap nama kelas, atribut kelas dan informasi operasi. Informasi operasi terdiri dari nama operasi, tipe data kembalian dari operasi dan parameter dalam suatu operasi.

Praproses dimulai dengan pembacaan kode sumber berbentuk dokumen teks dengan ekstensi .java. Dokumen kode sumber yang digunakan dalam penelitian ini merupakan dokumen kode sumber struktur kelas. Pembacaan dokumen kode sumber dilakukan dengan memanfaatkan *library* yang telah disediakan oleh Eclipse yaitu: **AST Parser** yang terdapat pada paket aplikasi **Java Development Tools (JDT)**. *Library* ini berfungsi untuk menggambarkan setiap struktur kelas menjadi sebuah struktur *tree* yang kemudian dapat diolah sesuai dengan kebutuhan. Dengan menggunakan library ini, sistem dapat mengelola setiap struktur kelas yang dibutuhkan untuk diproses selanjutnya. Struktur kelas yang diambil adalah: nama kelas, *instance variable, local variable, body of method* dan *return parameter*. Setiap struktur kelas ini disimpan pada struktur data yang telah disiapkan seperti tampak pada gambar 3.3.

UserManagement.java

```
package uji.coba.parsing;
public class UserManagement {
	private static final String TABLE_USER = "user";
	private static final String TABLE_TEACHING = "teaching";
	private static final String TABLE_ROLE = "role";

	public void InsertUser (User pUser){
		boolean exist = ExistUser(pUser);
		String sql = "INSERT INTO "+ UserManagement.TABLE_USER+"...";
	}

	public void UpdateUser (User pUser){
		boolean exist = ExistUser(pUser);
		String sql = "UPDATE "+ UserManagement.TABLE_USER+"...";
	}

	public void DeleteUser (User pUser){
		boolean exist = ExistUser(pUser);
		String sql = "DELETE FROM "+ UserManagement.TABLE_USER+"...";
	}

	public void ExistUser (User pUser){
		String sql = "SELECT * FROM "+ UserManagement.TABLE_USER+"...";
	}
	public boolean checkMandatoryFieldUser(User pUser){
		return pUser.getMandatory();
	}
	public void InsertTeaching (Teaching pTeaching){
		String sql = "INSERT INTO "+UserManagement.TABLE_TEACHING+"...";
	}
	public void UpdateTeaching (Teaching pTeaching){
		String sql = "UPDATE "+ UserManagement.TABLE_TEACHING+"...";
	}

	public void DeleteTeaching (Teaching pTeaching){
		String sql = "DELETE FROM "+UserManagement.TABLE_TEACHING+"...";
	}

	public boolean checkMandatoryFieldTeaching(Teaching pTeaching){
		return pTeaching.getMandatory();
	}

	public void InsertRole (Role pRole){
		String sql = "INSERT INTO "+ UserManagement.TABLE_ROLE+"...";
	}

	public void DeleteRole (Role pRole){
		String sql = "DELETE FROM "+ UserManagement.TABLE_ROLE+"...";
	}
	public boolean checkMandatoryFieldRole(Role pRole){
		return pRole.getMandatory();
	}
}
```

**Gambar 3.2 Contoh data uji kelas `UserManagement.java`**




**Gambar 3.3 Diagram Kelas `Operasi.java`**

Dari proses pembacaan struktur kode sumber didapatkan *corpus* operasi, seperti yang ditampilkan pada gambar 3.4 berikut:

```
Method Name : InsertRole
Return Type : void
Local Var : Role pRole,
Instance Var : TABLE_ROLE,
Method Invocation :

Method Name : DeleteTeaching
Return Type : void
Local Var : Teaching pTeaching,
Instance Var : TABLE_TEACHING,
Method Invocation :

Method Name : checkMandatoryFieldUser
Return Type : boolean
Local Var : User pUser,
Instance Var :
Method Invocation :

Method Name : UpdateUser
Return Type : void
Local Var : User pUser,
Instance Var : TABLE_USER,
Method Invocation : ExistUser

Method Name : ExistUser
Return Type : void
Local Var : User pUser,
Instance Var : TABLE_USER,
Method Invocation :

Method Name : DeleteUser
Return Type : void
Local Var : User pUser,
Instance Var : TABLE_USER,
Method Invocation : ExistUser

Method Name : InsertTeaching
Return Type : void
Local Var : Teaching pTeaching,
Instance Var : TABLE_TEACHING,
Method Invocation :

Method Name : InsertUser
Return Type : void
Local Var : User pUser,
Instance Var : TABLE_USER,
Method Invocation : ExistUser

Method Name : UpdateTeaching
Return Type : void
Local Var : Teaching pTeaching,
Instance Var : TABLE_TEACHING,
Method Invocation :

Method Name : checkMandatoryFieldTeaching
Return Type : boolean
Local Var : Teaching pTeaching,
Instance Var :
Method Invocation :
getMandatory()

Method Name : DeleteRole
Return Type : void
Local Var : Role pRole,
Instance Var : TABLE_ROLE,

Method Name : checkMandatoryFieldRole
Return Type : boolean
Local Var : Role pRole,
Instance Var :
Method Invocation :
getMandatory()
```

**Gambar 3.4 *Corpus* Hasil pembacaan kode sumber**

Dari informasi dasar kelas yang yang disebut dengan "*Corpus* operasi" ini didapatkan kemudian diakukan indeksasi data, yaitu memberikan kode penanda unik pada masing-masing operasi. Hasil indeksasi seperti yang ditampilkan pada Tabel 3.2

**Tabel 3-2 Tabel Kode dan Nama Operasi**

| id | Nama Operasi |
|---|---|
| m-0 | Insert Role |
| m-1 | Delete Teaching |
| m-2 | check Mandatory Field User |
| m-3 | Update User |
| m-4 | Exist User |
| m-5 | Delete User |
| m-6 | Insert Teaching |
| m-7 | Insert User |
| m-8 | Update Teaching |
| m-9 | check Mandatory Field Teaching |
| m-10 | Delete Role |
| m-11 | check Mandatory Field Role |

Kemudian untuk setiap *corpus* operasi ini dilanjutkan dengan proses tokenisasi, yaitu memisahkan berdasarkan karakter *non-alphanumeric* yaitu karakter yang bukan huruf ataupun angka dan melakukan pemisahan kata (*token splitting*) dalam *identifier* menjadi kata yang bermakna. Dalam kode sumber ada beberapa aturan yang sering digunakan untuk memisahkan *identifier*, yaitu:

- *Snake_case*, merupakan *identifier* yang dibentuk dari gabungan kata yang disambung menggunakan garis bawah, misalnya "*send_email"*. Untuk memisahkan menjadi kata-kata bisa dilakukan dengan mengganti garis bawah mejadi spasi.
- *CamelCase*, merupakan *identifier* yang dibentuk dari gabungan kata dengan penanda yaitu pada awal setiap kata digunakan huruf kapital seperti contohnya *sendEmail*. Untuk memisahkan menjadi kata-kata dengan cara menyisipkan spasi sebelum huruf kapital.
- *MixedCase*, merupakan identifier yang dibentuk dari gabungan kata dengan huruf kapital dan kata dengan huruf kecil, seperti contoh pada identifier ASTVisitor menjadi AST Visitor.

Token *splitting* dilakukan dengan memeriksa setiap karakter apabila ditemukan kombinasi karakter garisbawah dengan huruf apapun, atau karakter huruf kecil bertemu huruf besar, atau huruf besar sebanyak lebih dari satu kali berturut-turut bertmu huruf kecil, maka akan diganti dengan karakter spasi.

Langkah berikutnya adalah mencari dan menghilangkan *stopword* dalam kode sumber. *Stopword* adalah frasa penyusun operasi atau identifier kelas yang tidak bermakna atau tidak memiliki arti. Frasa stop word yang digunakan adalah frasa yang tidak bermakna spesifik dalam kode sumber. Pada proses tokenisasi kode sumber, akan dicek apakah frase yang ditokenisasi terdapat dalam daftar *stopword* atau tidak, jika ya maka akan dihapus dari daftar kata unik.

Hasil dari proses praproses adalah kumpulan token-token penyusun operasi. Daftar token unik ini akan digunakan untuk perhitungan *CSM* dengan metode TF-IDF ternormalisasi dan *vector space models.*

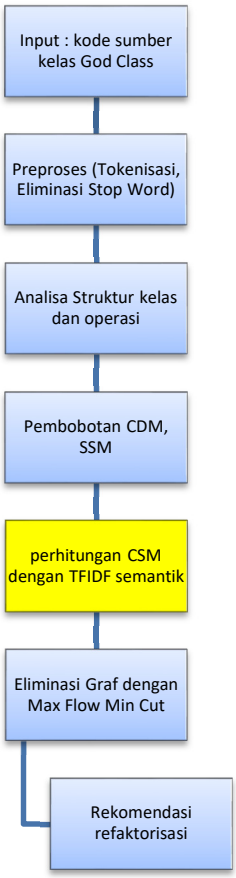
### 3.2.4 Perancangan Sistem

Sistem yang dibangun dalam bentuk Plugin Eclipse yang nantinya dapat dipasang pada IDE Eclipse sebagai alat bantu rekomendasi refaktorisasi kelas. Adapun skema sistem yang dibangun adalah seperti pada gambar 3.5.

### 3.2.5 Analisa Struktur Kode Sumber

Setelah mendapatkan dokumen operasi langkah selanjutnya adalah melakukan proses perhitungan bobot kesesuaian antar operasi atau disebut juga bobot kohesi antar operasi. Bobot ini adalah kombinasi dari bobot SSM, CDM, dan CSM.

Dari hasil analisa struktur akan didapatkan daftar variabel instance kelas, identifier kelas identifier operasi, dan pemanggilan operasi. Daftar struktur kelas ditampilkan pada tabel berikut:



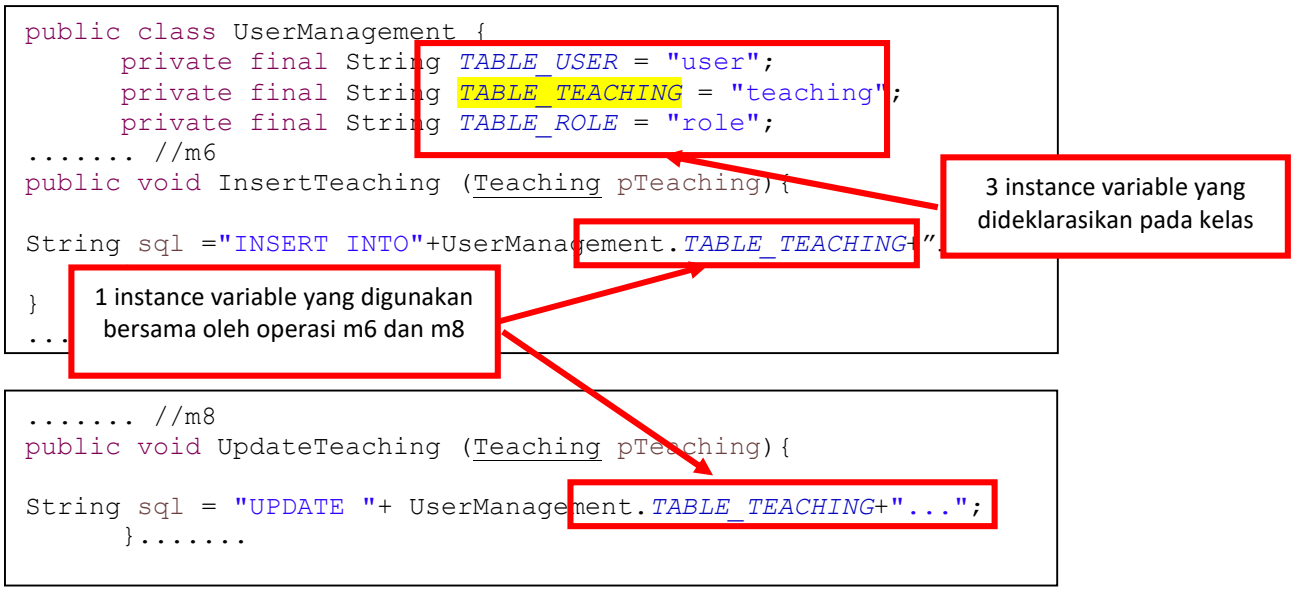

**Gambar 3.5 Skema Sistem (blok warna kuning adalah kontribusi pada penelitian ini)**

**Tabel 3-3 Daftar hasil analisa struktural**

| | | |
|---|---|---|
| **Nama Kelas** | : | UserManagement.java |
| **LOC** | : | 63 |
| **Jumlah Instance Variable** | : | 3 |
| **Jumlah Operasi** | : | 12 |

### 3.2.6 Perhitungan Bobot SSM

Bobot SSM dihitung berdasarkan jumlah atribut kelas yang digunakan secara bersama oleh operasi. Perhitungan adalah rasio jumlah *instance* variable yang digunakan bersama oleh dua operasi. Untuk menghitung bobot SSM, diperlukan analisa struktur dari dokumen operasi yang didapatkan dari proses tokenisasi. Contoh perhitungan bobot SSM ditampilkan pada ilustrasi berikut:




**Gambar 3.6 Ilustrasi perhitungan SSM**

Sehingga dari ilustrasi pada gambar 3.6, dapat dihitung nilai SSM dengan menggunakan persamaan 2-1 yaitu:

$$SSM(m_6, m_8) = \frac{1}{1} = 1$$

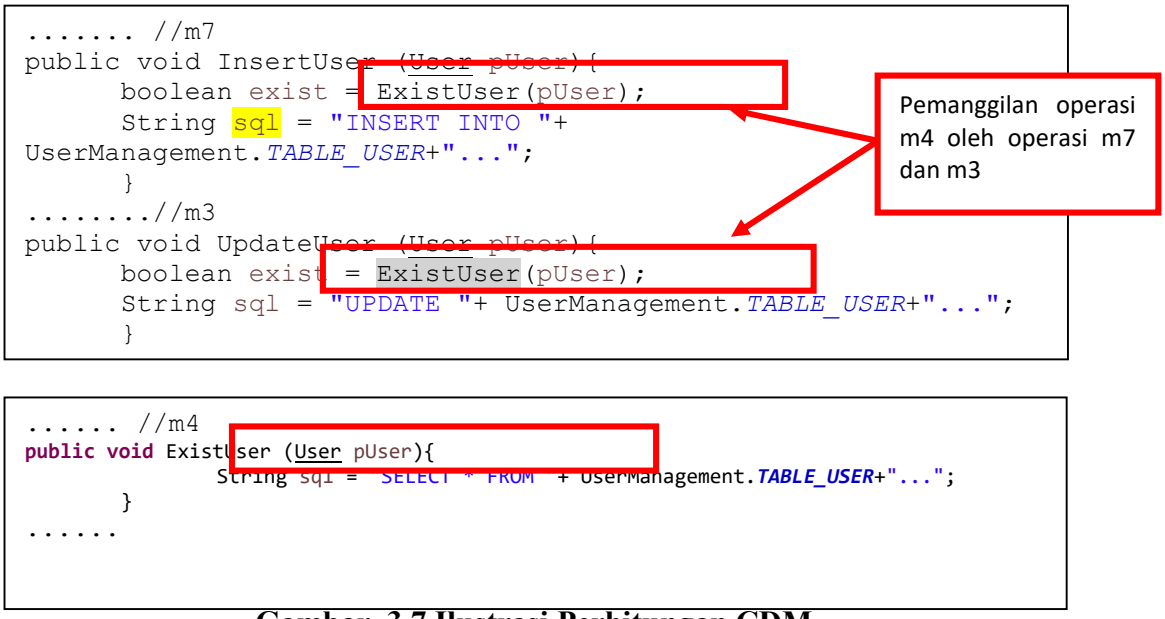
Perhitungan SSM ini dilakukan terhadap seluruh operasi yang ada, sehingga data bobot SSM ini dapat ditunjukkan pada tabel 3.4 :

**Tabel 3-4 hasil perhitungan bobot SSM**

| | m-0 | m-1 | m-2 | m-3 | m-4 | m-5 | m-6 | m-7 | m-8 | m-9 | m-10 | m-11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| m-0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| m-1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| m-2 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| m-3 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| m-4 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| m-5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| m-6 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| m-7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| m-8 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| m-9 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| m-10 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| m-11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

### 3.2.7 Perhitungan Bobot CDM

Bobot CDM dihitung berdasarkan jumlah operasi yang dipanggil dalam operasi. Contoh perhitungan bobot CDM ditampilkan pada gambar berikut:




**Gambar 3.7 Ilustrasi Perhitungan CDM**

Sehingga dari ilustrasi pada gambar 3.7, dapat dihitung nilai *CDM* dengan menggunakan persamaan 2-2 yaitu **:**

$$CDM(m_7, m_4) = \frac{1}{2} = 0.5$$

Perhitungan CDM ini dilakukan terhadap seluruh operasi yang ada, sehingga data bobot CDM antar operasi dapat ditunjukkan pada tabel 3.5 berikut:

**Tabel 3-5 Hasil perhitungan bobot CDM**

| | m-0 | m-1 | m-2 | m-3 | m-4 | m-5 | m-6 | m-7 | m-8 | m-9 | m-10 | m-11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| m-0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| m-1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| m-2 | 0 | 0 | 1 | 0 | 0,33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| m-3 | 0 | 0 | 0 | 1 | 0,33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| m-4 | 0 | 0 | 0,33 | 0,33 | 1 | 0 | 0 | 0 | 0 | 0,33 | 0 | 0 |
| m-5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| m-6 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| m-7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| m-8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| m-9 | 0 | 0 | 0 | 0 | 0,33 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| m-10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| m-11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

### 3.2.8 Perhitungan Bobot CSM

Bobot CSM yang diusulkan pada penelitian ini menggunakan metode LSI dengan modifikasi pada perhitungan TF-IDF yang menggunakan pertimbangan *Word Dependency* untuk lokasi konsep nama operasi dan penggunaan *vector space models* untuk menghitung similaritas operasi. Langkah-langkah perhitungan bobot TF-IDF *word dependency* antar operasi pada kode sumber adalah sebagai berikut :

1. Menggabungkan seluruh term penyusun operasi yang telah ditokenisasi menjadi daftar token untuk dihitung masing-masing bobotnya.
2. Dari masing-masing operasi yang telah terbentuk kata unik dihitung bobot TF-IDF-nya berdasarkan frekuensi kemunculan term dalam masing-masing kalimat nama operasi.
3. Dalam menghitung TF-IDF, dipertimbangkan keterhubungan antar kata dalam kalimat nama operasi untuk menentukan bobot vektor operasi. Dua kalimat yang memiliki *root* kalimat yang sama, akan dianggap sebagai kalimat yang memiliki konsep yang sama, hal ini juga akan menambah bobot Vektor kalimat tersebut untuk metode *vector space models*. Untuk menentukan bobot keterhubungan antar kata digunakan persamaan 3-1:

$$\boldsymbol{I(m_i, m_j)} = \begin{cases} \mathbf{1} & \boldsymbol{if\ t_i\ \epsilon\ m_j} \\ \mathbf{0} & \boldsymbol{otherwise} \end{cases} \qquad \textbf{(3-1)}$$

$$V(m_i, m_j) = \partial \cdot tfidf(m_i, m_j) + \emptyset \cdot I(m_i, m_j)$$

dimana:

$tfidf(m_j, m_j$ = bobot tf-idf yang dihitung dengan persamaan 2-5

$I(m_i, m_j)$ = bobot kemiripan konsep operasi

$t_i$ = root kalimat $m_i$ yang diidentifikasi dengan *Standford Dependency*

$V(m_i, m_j)$= vektor bobot $m_i$ dan $m_j$

$\partial$ = koefisien bobot *tf-idf*

$\emptyset$ = koefisien bobot kemiripan konsep kalimat

Dalam penelitian ini akan digunakan koefisien $\partial$ yang nilainya 0,1 dan koefisien $\emptyset$ dengan nilai 0.9, dimana $\partial + \emptyset = 1$.

```
..... //m9
public boolean checkMandatoryFieldTeaching(Teaching pTeaching){
	return pTeaching.getMandatory();
}
```

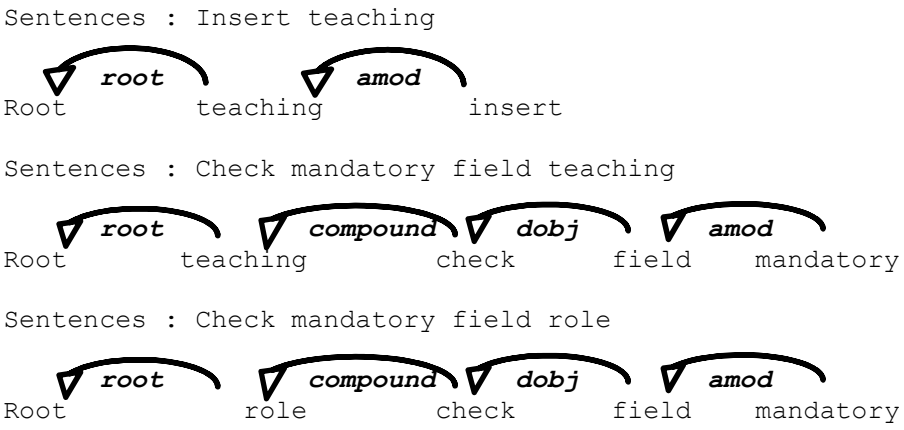
```
....//m6
public void InsertTeaching (Teaching pTeaching){
	String sql = "INSERT INTO"+UserManagement.TABLE_TEACHING+"...";
};
```

```
....//m11
public boolean checkMandatoryFieldRole(Role pRole){
	return pRole.getMandatory();
}
```

**Gambar 3.8 Korpus Operasi m11, m9 dan m6**

**Perhitungan Bobot Kesamaan Konsep**

Bobot kesamaan konsep antar operasi diidentifikasi berdasarkan kesamaan konsep antar nama operasi. Jadi dalam penghitungan kesamaan konsep, hanya diambil nama dari operasi. Sebagai contoh, pada gambar 3.8 operasi `InsertTeaching (Teaching pTeaching)` diekstrak nama operasinya sehingga membentuk korpus kalimat "`Insert Teaching`" yang terdiri atas kata "`Insert`" dan "`Teaching`". Sedangkan operasi `checkMandatoryFieldTeaching(Teaching pTeaching)` diekstrak dan dihasilkan korpus kalimat "`check mandatory field Teaching`" yang terdiri atas kata "`check`", "`mandatory`", "`field`" dan "`Teaching`". dari dua korpus operasi itu diketahui informasi *Word Dependency* masing-masing nama operasi, seperti yang ditampilkan dalam gambar 3.9. Terdapat kesamaan root dari kalimat yaitu teaching, sehingga bobot kesamaan konsep dari dua operasi ini menjadi tinggi.




**Gambar 3.9 *Word Dependency* pada nama operasi "insert teaching" dan "check mandatory field teaching"**

Setelah didapatkan bobot dependensi antar kata, kemudian diidentifikasi masing-masing kata yang memiliki dependensi sebagai "root". Kemudian dihitung nilai TF-IDF sesuai dengan persamaan 2-6 untuk masing-masing term, dan didapatkan hasil untuk vektor operasi `InsertTeaching` dan `checkMandatoryFieldTeaching` seperti yang ditunjukkan pada tabel 3-6 dan 3-7.

**Tabel 3-6 Perbandingan perhitungan TF-IDF dengan Dep TF-IDF untuk operasi `insertTeaching` (term `teaching` adalah root dari kalimat)**

| Term | TF-IDF (sebelum identifikasi root) | TF-IDF modified (setelah identifikasi root) |
|---|---|---|
| check | 0 | 0 |
| delete | 0 | 0 |
| exist | 0 | 0 |
| field | 0 | 0 |
| insert | 1,6020599913279627 | 0,1602059991327963 |
| mandatory | 0 | 0 |
| p | 1,0000000000000002 | 0,10000000000000003 |
| role | 0 | 0 |
| table | 1,1249387366083003 | 0,11249387366083004 |
| teaching | 2,366436864526532 | 1.1366436864526532 |
| update | 0 | 0 |
| user | 0 | 0 |
| void | 0 | 0 |

**Tabel 3-7 Perbandingan perhitungan TF-IDF dengan Dep TF-IDF untuk operasi `checkMandatoryFieldTeaching` (`teaching` adalah root dari kalimat)**

| Term | TF-IDF (sebelum identifikasi root) | TF-IDF modified (setelah identifikasi root) |
|---|---|---|
| check | 1,6020599913279627 | 0,1602059991327963 |
| delete | 0 | 0 |
| exist | 0 | 0 |
| field | 1,6020599913279627 | 0,1602059991327963 |
| insert | 0 | 0 |
| mandatory | 2,0843281035708574 | 0,20843281035708575 |
| p | 1,0000000000000002 | 0,10000000000000003 |
| role | 0 | 0 |
| table | 0 | 0 |
| teaching | 2,18188720114459 | 1,1181887201144591 |
| update | 0 | 0 |
| user | 0 | 0 |
| void | 0 | 0 |

**Tabel 3-8 Perbandingan perhitungan TF-IDF dengan Dep TF-IDF untuk operasi `checkMandatoryFieldRole` (`role` adalah root dari kalimat)**

| Term | TF-IDF (sebelum identifikasi root) | TF-IDF modified (setelah identifikasi root) |
|---|---|---|
| check | 1,6020599913279627 | 0,1602059991327963 |
| delete | 0 | 0 |
| exist | 0 | 0 |
| field | 1,6020599913279627 | 0,1602059991327963 |
| insert | 0 | 0 |
| mandatory | 2,0843281035708574 | 0,1602059991327963 |
| p | 1,0000000000000002 | 0,10000000000000003 |
| role | 2,18188720114459 | 1,1181887201144591 |
| table | 0 | 0 |
| teaching | 0 | 0 |
| update | 0 | 0 |
| user | 0 | 0 |
| void | 0 | 0 |

Langkah selanjutnya adalah rumus kesamaan *Cosine* sesuai persamaan 2-7 untuk menetukan tingkat kemiripan konsep operasi dengan operasi lainnya dalam kelas. Perhitungan kemiripan antar operasi dilakukan pada seluruh operasi dalam kelas dengan mengasumsikan bobot *corpus* operasi adalah vektor dari TF-IDF.

Dengan memperhatikan perhitungan TF-IDF pada 3-6 dan 3-7, vektor masing-masing operasi dibentuk sebagai matriks $1 \times n$ dimana n adalah jumlah term dan nilai matrik adalah nilai TF-IDF term tersebut sebagai bobot. Kemudian matriks ini digunakan sebagai perhitungan similarity dengan persamaan 2-7. Matriks bobot TF-IDF masing-masing operasi adalah sebagai berikut:

$$m_6 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1,6 \\ 0 \\ 1,0 \\ 0 \\ 1,12 \\ 2,36 \\ 0 \\ 0 \\ 0 \end{bmatrix} m_9 = \begin{bmatrix} 1,60 \\ 0 \\ 0 \\ 1,60 \\ 0 \\ 2,08 \\ 1,0 \\ 0 \\ 0 \\ 2,18 \\ 0 \\ 0 \\ 0 \end{bmatrix} m_{11} = \begin{bmatrix} 1,60 \\ 0 \\ 0 \\ 1,60 \\ 0 \\ 2,08 \\ 1,0 \\ 2,18 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Maka nilai $Sim(m_6, m_9)$ dengan dihitung menggunakan persamaan 2-7 dengan TF-IDF normal sebagai berikut:

$$= \frac{(1,0 \times 1,0) + (2,36 \times 2,18)}{\sqrt{1,6^2 + 1^2 + 1,12^2 + 2,36^2} \times \sqrt{1,6^2 + 1,6^2 + 2,08^2 + 1^2 + 2,18^2}} = 0,48$$

Dan nilai $Sim(m_{11}, m_9)$ dihitung menggunakan persamaan 2-7 dengan TF-IDF normal sebagai berikut:

$$= \frac{(1,6 \times 1,6) + (1,6 \times 1,6) + (2,08 \times 2,08) + (1,0 \times 1,0)}{\sqrt{1,6^2 + 1,6^2 + 2,08^2 + 1^2 + 2,18^2} \times \sqrt{1,6^2 + 1,6^2 + 2,08^2 + 1^2 + 2,18^2}} = 0,68$$

Dari perhitungan diatas diketahui bahwa dengan menggunakan TF-IDF normal, operasi `checkMandatoryFieldTeaching` lebih dekat pada `checkMandatoryFieldRole` dengan nilai kedekatan 0,68. Hal ini menunjukkan bahwa metode TF-IDF normal gagal mengidentifikasi kesamaan konsep (role dan teaching). Sedangkan dengan menggunakan TF-IDF termodifikasi, Matrik bobot TF-IDF masing-masing operasi adalah sebagai berikut:

$$m_6 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0,160 \\ 0 \\ 0,10 \\ 0 \\ 0,11 \\ 1,13 \\ 0 \\ 0 \\ 0 \end{bmatrix} m_9 = \begin{bmatrix} 0,16 \\ 0 \\ 0 \\ 0,16 \\ 0 \\ 0,208 \\ 0,10 \\ 0 \\ 0 \\ 1,118 \\ 0 \\ 0 \\ 0 \end{bmatrix} m_{11} = \begin{bmatrix} 0,16 \\ 0 \\ 0 \\ 0,16 \\ 0 \\ 0,208 \\ 0,10 \\ 1,118 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Maka nilai $Sim(m_6, m_9)$ dihitung menggunakan persamaan 2-7 dengan TF-IDF termodifikasi sebagai berikut:

$$= \frac{(0{,}1 \times 0{,}1) + (1{,}13 \times 1{,}118)}{\sqrt{0{,}16^2 + 0{,}1^2 + 0{,}11^2 + 1{,}13^2} \times \sqrt{0{,}16^2 + 0{,}16^2 + 0{,}208^2 + 0{,}10^2 + 1{,}118^2}} = 0{,}95$$

Dan nilai $Sim(m_{11}, m_9)$ dihitung menggunakan persamaan 2-7 dengan TF-IDF termodifikasi sebagai berikut:

$$= \frac{(0{,}16 \times 0{,}16) + (0{,}16 \times 0{,}16) + (0{,}208 \times 0{,}208) + (0{,}1 \times 0{,}1)}{\sqrt{0{,}16^2 + 0{,}16^2 + 0{,}208^2 + 0{,}1^2 + 1{,}118^2} \times \sqrt{0{,}16^2 + 0{,}16^2 + 0{,}208^2 + 0{,}1^2 + 1{,}118^2}} = 0{,}07$$

Dari perhitungan diatas dapat diketahui bahwa dengan menggunakan TF-IDF termodifikasi, dapat memperbaiki penilaian kesamaan konsep antara operasi `checkMandatoryTeaching` yang seharusnya lebih dekat dengan `insertTeaching` dengan nilai kedekatan 0,95.

Perhitungan kemiripan operasi dilakukan terhadap seluruh operasi sehingga didapatkan matriks perbandingan kemiripan antara operasi dengan operasi seperti yang ditunjukkan pada tabel 3.8.

**Tabel 3-9 Perhitungan kemiripan antar operasi dengan menggunakan VSM**

| | m-0 | m-1 | m-2 | m-3 | m-4 | m-5 | m-6 | m-7 | m-8 | m-9 | m-10 | m-11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| m-0 | 1,00 | 0,24 | 0,47 | 0,47 | 0,13 | 0,24 | 0,04 | 0,04 | 0,04 | 0,47 | 0,04 | 0,04 |
| m-1 | 0,24 | 1,00 | 0,04 | 0,04 | 0,04 | 0,24 | 0,04 | 0,48 | 0,04 | 0,04 | 0,04 | 0,48 |
| m-2 | 0,47 | 0,04 | 1,00 | 0,57 | 0,61 | 0,04 | 0,14 | 0,08 | 0,09 | 0,57 | 0,08 | 0,14 |
| m-3 | 0,47 | 0,04 | 0,57 | 1,00 | 0,61 | 0,04 | 0,08 | 0,08 | 0,52 | 0,57 | 0,08 | 0,08 |
| m-4 | 0,52 | 0,04 | 0,64 | 0,64 | 1,00 | 0,04 | 0,09 | 0,09 | 0,10 | 0,64 | 0,09 | 0,09 |
| m-5 | 0,24 | 0,24 | 0,04 | 0,04 | 0,04 | 1,00 | 0,48 | 0,04 | 0,14 | 0,04 | 0,48 | 0,04 |
| m-6 | 0,04 | 0,04 | 0,16 | 0,09 | 0,10 | 0,52 | 1,00 | 0,09 | 0,20 | 0,09 | 0,58 | 0,16 |
| m-7 | 0,04 | 0,52 | 0,09 | 0,09 | 0,10 | 0,04 | 0,09 | 1,00 | 0,10 | 0,16 | 0,16 | 0,58 |
| m-8 | 0,04 | 0,04 | 0,09 | 0,17 | 0,10 | 0,52 | 0,58 | 0,09 | 1,00 | 0,09 | 0,58 | 0,09 |
| m-9 | 0,47 | 0,04 | 0,57 | 0,57 | 0,61 | 0,04 | 0,08 | 0,14 | 0,09 | 1,00 | 0,14 | 0,08 |
| m-10 | 0,04 | 0,04 | 0,09 | 0,09 | 0,10 | 0,52 | 0,58 | 0,16 | 0,20 | 0,16 | 1,00 | 0,09 |
| m-11 | 0,04 | 0,52 | 0,16 | 0,09 | 0,10 | 0,04 | 0,16 | 0,58 | 0,10 | 0,09 | 0,09 | 1,00 |

### 3.2.9 Perhitungan Kombinasi Bobot

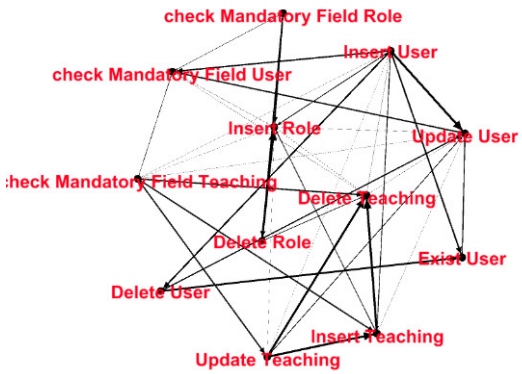
Setelah didapatkan 3 nilai parameter penilaian kemiripan antar operasi, kemudian dilakukan kombinasi bobot tersebut dengan koefisien *w* sehingga didapatkan bobot akhir nilai kohesi antar operasi. Koefisien yang digunakan adalah $\boldsymbol{w_{CSM} = 0,7, w_{SSM} = 0,2, w_{CDM} = 0,1}$ (Bavota, et al., 2011)**.** Hasil dari perhitungan kombinasi bobot ini kemudian disaring dengan nilai ambang batas

kohesi minimal (*minCohesion*) dimana nilai kombinasi bobot yang berada dibawah ambang batas tidak akan dihitung (dijadikan 0). Dalam penelitian ini akan digunakan nilai ambang batas *minCohesion* mulai dari 0,1 hingga 0,5 dengan kenaikan 0,1 untuk mencari nilai ambang batas minCohesion yang paling optimal. Sehingga hasil bobot kombinasi adalah seperti yang ditampilkan pada tabel 3.10.

**Tabel 3-10 Hasil perhitungan bobot w**

| | m-0 | m-1 | m-2 | m-3 | m-4 | m-5 | m-6 | m-7 | m-8 | m-9 | m-10 | m-11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| m-0 | 1.00 | 0.00 | 0.33 | 0.33 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.33 | 0.00 | 0.00 |
| m-1 | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.33 | 0.00 | 0.00 | 0.00 | 0.33 |
| m-2 | 0.00 | 0.00 | 1.00 | 0.60 | 0.66 | 0.00 | 0.00 | 0.00 | 0.00 | 0.60 | 0.00 | 0.00 |
| m-3 | 0.00 | 0.00 | 0.00 | 1.00 | 0.66 | 0.00 | 0.00 | 0.00 | 0.36 | 0.60 | 0.00 | 0.00 |
| m-4 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.68 | 0.00 | 0.00 |
| m-5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.33 | 0.00 | 0.00 | 0.00 | 0.33 | 0.00 |
| m-6 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.34 | 0.00 | 0.61 | 0.00 |
| m-7 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 | 0.61 |
| m-8 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.61 | 0.00 |
| m-9 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 |
| m-10 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 |
| m-11 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |

Tabel nilai bobot antar operasi ini digunakan untuk menyusun graf keterhubungan antar operasi. Masing-masing operasi digambarkan sebagai sebuah *node* dan nilai keterhubungan antar operasinya adalah sebagai *edge*. Nilai keterhubungan antaroperasi untuk kelas `UserManagement.java` adalah seperti pada gambar 3.10 berikut:

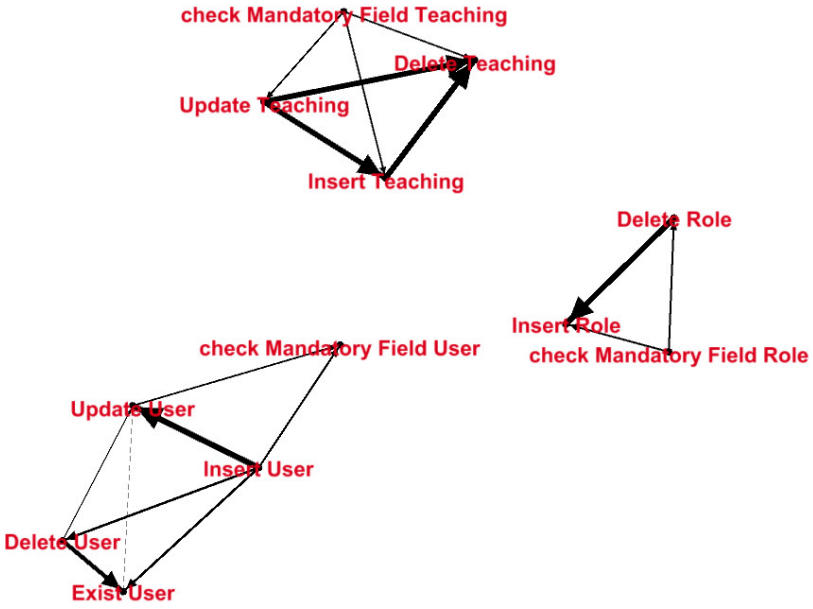



**Gambar 3.10 Graf Keterhubungan antar operasi : garis tebal menunjukkan diantara dua operasi tersebut memiliki keterhubungan yang lebih tinggi daripada garis yang lebih tipis**

### 3.2.10 Max-Flow Min-Cut

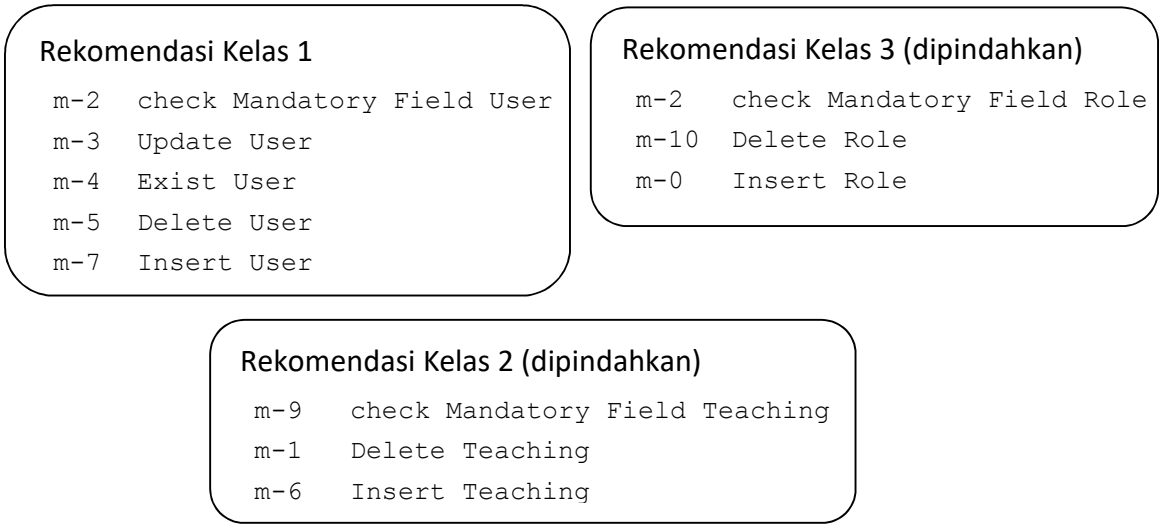
Langkah selanjutnya adalah mengimplementasikan algoritma *MaxFlow MinCut* untuk mendapatkan pemotongan optimal dari graf keterhubungan antar operasi. Pemotongan ini bertujuan untuk mendapatkan klasterisasi operasi sebagai rekomendasi refaktorisasi kelas dengan nilai kohesi yang tinggi. Bobot kombinasi antar operasi (w) yang telah disaring merupakan input dari algoritma *Max-Flow Min-Cut*.

Algoritma *MaxFlow MinCut* akan melakukan penghitungan *Flow* maksimal dari graf keterhubungan antar operasi dan merekomendasikan pemutusan jalur yang optimal untuk menghasilkan 2 potongan atau lebih dari graf. Luaran dari proses ini adalah rekomendasi pemotongan jalur antara dua node atau lebih untuk menghasilkan potongan graf baru. Hasil dari algoritma *Max-Flow Min-Cut* adalah pemotongan graf seperti yang tampak pada gambar 3.11




**Gambar 3.11 Hasil pemotongan graf bobot kombinasi. Garis tebal menunjukkan operasi tersebut memiliki keterhubungan lebih tinggi daripada garis yang lebih tipis**

Seperti yang tampak pada gambar 3.11, dihasilkan 2 sub-graf yang masing-masing menjadi rekomendasi penyusunan kelas sesuai dengan penilaian struktural dan semantik. Sub graf ini akan menjadi rekomendasi pemindahan kelas menjadi kelas baru. Dimana berdasarkan graf yang terbentuk, susunan kelas yang baru adalah seperti pada gambar 3.12 sebagai berikut :




**Gambar 3.12 Hasil rekomendasi pemindahan operasi**

Dari hasil rekomendasi yang diusulkan sistem, dapat diketahui bahwa konsep kelas yang baru sudah sesuai secara semantik, dimana konsep kelas yang baru terdiri atas tiga konsep yaitu User, Teaching, dan Role.

### 3.3 Pengujian dan Analisa Hasil

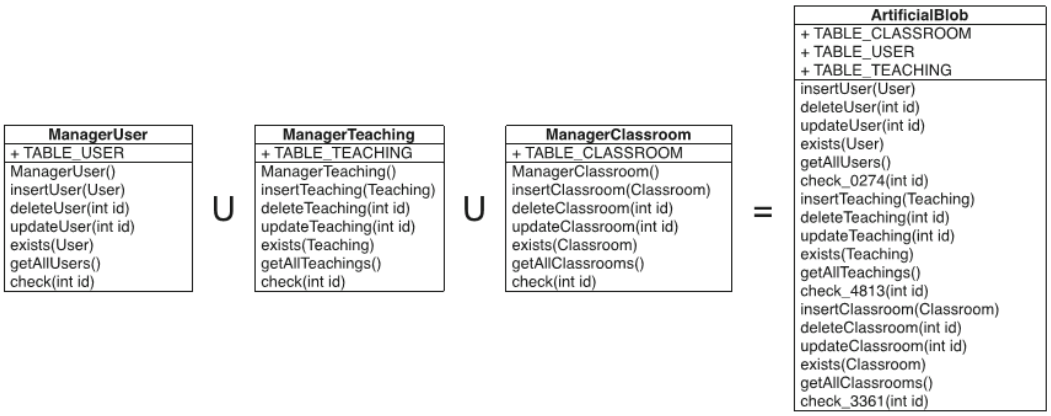
Untuk mengetahui keberhasilan sistem dalam mengidentifikasi penyusunan kembali kasus God Class dilakukan pengujian pada metode yang diusulkan. Untuk menguji keakuratan sistem yang diusulkan, rekomendasi yang dihasilkan akan dibandingkan dengan rekomendasi yang dihasilkan oleh sistem dari penelitian sebelumnya, dimana penelitian sebelumnya menggunakan ahli rekayasa perangkat lunak untuk membandingkan rekomendasi yang dihasilkan sistem dengan rekomendasi yang dihasilkan seorang ahli. Pada penelitian ini dilakukan dua sub pengujian perangkat lunak, dimana sub bagian perangkat lunak yang pertama masih menggunakan metode yang sama dengan penelitian sebelumnya, yaitu metode Extract Class yang digunakan pada sub bagian perangkat lunak untuk mengekstrak God Class.

# BAB 4
# HASIL DAN PEMBAHASAN

## 4.1 Tahapan Pengujian

Uji coba dilakukan terhadap metode yang diusulkan untuk mengetahui keakuratan sistem yang dibangun. Uji coba dilakukan dengan menerapkan metode identifikasi peluang refaktorisasi pada sebuah God Class. Uji coba yang dilakukan akan terdiri dari dua tahapan yaitu; (1) pengujian terhadap God Class sintetis; (2) pengujian terhadap kode sumber terbuka.

Dalam penelitian pertama dibuat kelas sintetis God Class yang terbentuk dari gabungan seluruh operasi dalam tiga kelas yang memiliki konsep berbeda menjadi sebuah kelas ArtificialBLOB. Kelas yang digunakan adalah kelas `ManagerUser`, `ManagerTeaching` dan `ManagerClassroom`. Pembentukan kelas sintetis ArtificialBLOB digambarkan pada gambar 4.1




**Gambar 4.1 Pembentukan kelas `ArtificialBLOB`**

## 4.2 Pengumpulan Dataset

Dataset untuk uji coba tahap kedua berasal dari kode sumber perangkat lunak dalam bahasa pemrograman Java. Perangkat lunak yang akan dijadikan dataset dipilih perangkat lunak kode sumber terbuka. Kode sumber diperoleh dari beberapa repositori kode sumber di internet. Kode seumber yang kedua adalah kode sumber kelas yang dibuat sintetis dari penggabungan beberapa kelas menjadi sebuah God Class. Kode sumber yang digunakan dalam dataset ini hanya kode sumber perangkat lunak tanpa kode sumber *library* yang diperlukan untuk mengkompilasi kode sumber tersebut. Kode sumber yang digunakan dalam penelitian ini hanya

kode sumber berbahasa Inggris karena dasar kamus yang digunakan adalah repositori kamus digital berbahasa Inggris (Standford NLP). Dari kode sumber yang diperoleh akan dilakukan pemrosesan untuk mendapatkan semua kelas dan operasi penyusunnya.

Penelusuran yang akan dilakukan dalam penelitian ini adalah untuk menemukan peluang rekomendasi pemecahan kelas menjadi kelas baru berdasarkan kesamaan konsep. Dataset berupa hasil analisa yang dilakukan oleh ahli rekayasa perangkat lunak. Ahli rekayasa perangkat lunak melakukan analisa terhadap kode sumber untuk menentukan operasi yang direkomendasikan untuk dipindahkan ke kelas baru untuk mengurangi coupling dan meningkatkan kohesi kelas - kelas tersebut.

Dataset yang digunakan dalam pengujian tahap kedua sama dengan dataset yang digunakan dalam penelitian sebelumnya (Petrenko & Rajlich, 2012). Pada penelitian sebelumnya telah dilakukan pengujian dan juga telah didapatkan analisa dari pakar rekayasa perangkat lunak daftar operasi-operasi yang diekstrak. Proyek-proyek perangkat lunak yang dipilih adalah perangkat lunak yang tersedia di repositori online dan dapat diunduh secara gratis. Proyek perangkat lunak yang digunakan dalam penelitian dijelaskan pada tabel 4.1.

**Tabel 4-1 Daftar Kelas yang Digunakan Untuk Uji Coba Ekstraksi Operasi**

| No. | Nama Proyek | Nama Kelas | Jumlah Operasi |
|---|---|---|---|
| 1 | ApacheHDSQLDB | UserManager.java | 13 |
| 2 | ApacheHDSQLDB | Database.java | 40 |
| 3 | ApacheHDSQLDB | Select.java | 14 |
| 4 | ArgoUML | Import.java | 10 |
| 5 | ArgoUML | FileGeneratorAdapter.java | 11 |
| 6 | JEdit | JeditTextArea.java | 187 |
| 7 | JFreeChart | JfreeChart.java | 24 |
| 8 | JFreeChart | NumberAxis.java | 20 |
| 9 | Xerces | XMLDTDValidator.java | 69 |
| 10 | Xerces | XMLSerializer.java | 25 |
| 11 | Xerces | DomParserImpl.java | 72 |

Dalam rangka mempercepat proses pengujian, dibuat suatu alat bantu yang berfungsi melakukan parsing terhadap kode sumber menjadi dokumen yang dapat ditokenisasi menjadi korpus. Alat bantu yang dikembangkan berupa eclipse plugin. Plugin dikembangkan menggunakan basis Eclipse Java Development Tools (JDT) dan akan dijalankan sebagai aplikasi tambahan untuk Eclipse.

## 4.3 Metode dan Skenario Uji Coba

Dalam pengujian perangkat lunak ini akan dilakukan dua pengujian yaitu menguji metode LSI dengan TF-IDF normal untuk pembobotan CSM pada kelas dengan mengambil nama kelas. Sedangkan pada pengujian kedua dilakukan dengan metode LSI dan memodifikasi TF-IDF menggunakan *library* Standford Word Dependency untuk menentukan konsep lokasi dari operasi.

Proses pengujian yang pertama adalah pengujian metode ekstraksi kelas. Pengujian ini dilakukan dengan tujuan untuk mengetahui keberhasilan sistem yang dibangun dalam megekstrak kelas kode sumber. Pengujian dilakukan dengan mengimplementasikan alur kerja yang diusulkan dan dijelaskan pada metodologi. Alur kerja tersebut diimplementasikan pada dataset yang berupa kelas sintetis God Class, kemudian membandingkan hasil identifikasi sistem dengan hasil dari ahli rekayasa perangkat lunak. Langkah pertama adalah mengunduh kode sumber dari repositori. Langkah berikutnya adalah membentuk kelas sintetis God Class dengan nama `ArtificialBLOB` Seperti pada gambar 4.1. Kemudian kode sumber `ArtificialBLOB` tersebut diimpor ke dalam proyek Eclipse. Berikutnya membentuk Abstract Syntax Tree (AST) dari proyek tersebut. Kemudian dilakukan penelusuran pada AST untuk membentuk korpus-korpus dokumen informasi kelas dan operasinya.

Proses penghitungan kemiripan dalam penelitian ini menggunakan kata benda dan kata kerja dalam nama operasi dan nama kelas. Karenanya perlu dilakukan parsing terhadap operasi dan kelas untuk mendapatkan semua kata benda dan kata kerja didalamnya. Penelusuran AST, setiap ASTNode berjenis deklarasi kelas akan direpresentasikan dalam objek Kelas dan disusun menjadi korpus dalam bentuk *string*. Demikian juga dengan ASTNode yang berbentuk operasi akan direpresentasikan dalam objek Operasi yang direpresentasikan sebagai *string*. Setiap objek Kelas akan mengandung variabel *list* operasi yang berbentuk *array* dari kelas Operasi. Operasi-operasi dalam kelas akan dianalisa dan diambil beberapa parameter penyusunnya hal ini dilakukan untuk mendapatkan kondisi struktural dan sintaksis dari kelas. Elemen yang diambil antara lain adalah:

1. Nama Kelas
2. Nama instance variabel
3. Nama Static variable
4. Nama operasi dan nilai kembalian dari operasi
5. Parameter input operasi

Setelah dibentuk korpus dokumen dalam bentuk string, kemudian dilakukan praproses pada korpus kode sumber. Setiap korpus kalimat dan dokumen direpresentasikan dalam bentuk graf tak berarah yang memiliki bobot pada setiap vektornya. Metode LSI dengan TF-IDF normal digunakan untuk menghitung bobot tersebut. Pertama dilakukan tokenisasi. Daftar kata dasar seluruh kode sumber ini dihitung frekuensi kemunculannya dengan mempertimbangkan jarak keterhubungan antar term. Untuk setiap perbandingan term yang memiliki jarak keterhubungan kurang dari ambang batas, dianggap term tersebut secara semantik terhubung dan menambah bobot vertek.

### 4.4 Hasil Uji Coba Kelas Sintetis

Pengujian ini dilakukan untuk menilai akurasi metode ekstraksi yang mengimplementasikan algoritma deteksi kemungkinan penyusunan kembali struktur kelas yang diusulkan. Dalam pengujian ini digunakan metode LSI dengan TF-IDF normal.

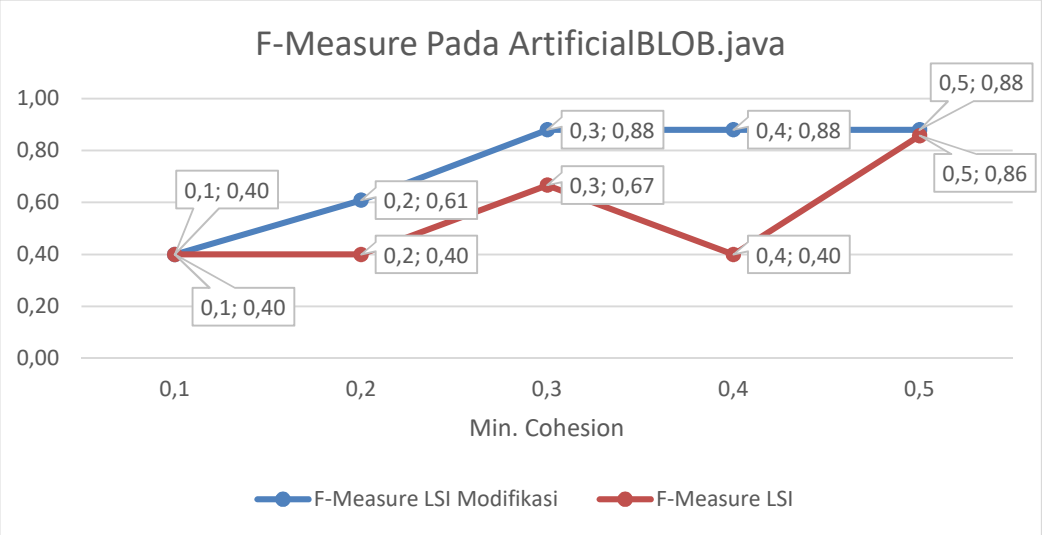
Parameter pengujian adalah nilai ambang batas yang digunakan untuk menentukan nilai penyaringan nilai konstanta pengali bobot keterhubungan antar Operasi (*minCohesion*). Variansi nilai untuk ambang batas *minCohesion* adalah antara 0 hingga 0.5 dengan kenaikan 0.1. Untuk masing-masing nilai ambang batas dilakukan percobaan dan didapatkan nilai *precision, recall* untuk dihitung nilai *F-Measure* kemudian dicari nilai optimalnya untuk menentukan nilai ambang batas terbaik.

Data hasil pengujian *precision, recall,* dan *F-Measure* untuk metode yang diusulkan pada disajikan pada tabel 4.2.

**Tabel 4-2 Hasil Perhitungan F-Measure dengan metode TF-IDF**

| Nilai minCohesion | LSI-normal | | | LSI-modified | | |
|---|---|---|---|---|---|---|
| | Precision | Recall | F-Measure | Precision | Recall | F-Measure |
| 0.1 | 0.50 | 0.33 | 0.40 | 0.50 | 0.33 | 0.40 |
| 0.2 | 0.50 | 0.33 | 0.40 | 0.64 | 0.58 | 0.61 |
| 0.3 | 0.50 | 1.00 | 0.67 | 0.79 | 1.00 | 0.88 |
| 0.4 | 0.50 | 0.33 | 0.40 | 0.79 | 1.00 | 0.88 |
| 0.5 | 0.75 | 1.00 | 0.86 | 0.79 | 1.00 | 0.88 |

Data hasil pengujian F-Measure metode LSI dengan TF-IDF Normal dapat digambarkan sebagai grafik seperti pada gambar 4.3. Dari hasil perhitungan F-Measure didapatkan nilai paling tinggi untuk metode LSI pada nilai *minCohesion* 0.5 dengan nilai F-Measure 0.86 dengan *precision* 0.75 dan *recall* 1.00 . Sedangkan dengan metode LSI termodifikasi didapatkan nilai F-Measure paling tinggi pada nilai *minCohesion* 0.3 dengan nilai F-Measure 0.88 dengan *precision* 0.79 dan *recall* 1.00.




**Gambar 4.2 Grafik F-Measure**

Dari grafik F-Measure didapatkan kesimpulan bahwa semakin tinggi nilai konstanta *wCSM*, maka semakin tinggi F-Measure yang dihasilkan sistem. Dengan demikian, bobot CSM memiliki pengaruh paling signifikan diantara bobot CDM dan SSM.

Temuan lain adalah hasil evaluasi menggunakan metode LSI dengan TF-IDF normal masih terdapat *False Negative*, yaitu sistem gagal mengidentifikasi pengelompokan kelas baru sesuai dengan yang direkomendasikan oleh pakar. Hal ini dikarenakan dalam metode TF-IDF normal untuk penentuann kemunculan frekuensi masih menggunakan metode *string matching*, dan tidak dilakukan identifikasi lokasi konsep dari sebuah operasi. Hal ini mengakibatkan bobot term hanya relatif terhadap term yang lain pada sebuah kode sumber. Bobot term yang relatif terhadap term lain tanpa mempertimbangkan konsep lokasi dari operasi mngakibatkan pertimbangan kemiripan hanya dari frekuensi kemunculan term. Sperti contohnya pada operasi `checkMandatoryFieldUser` yang teridentifikasi lebih mirip dengan `checkMandatoryFieldTeaching` dan `checkMandatoryFieldRole.` Hasil pengelompokan kelas dengan metode TF-IDF normal disajikan pada tabel 4.3.

**Tabel 4-3 Hasil Pengelompokan Kelas dengan metode LSI Normal, LSI Modifikasi dibandingkan dengan Rekomendasi Pakar**

| Nama Operasi | Hasil Pakar | Metode LSI Normal | Metode LSI Modifikasi |
|---|---|---|---|
| DeleteTeaching() | C1 | C1 | C1 |
| InsertTeaching() | C1 | C1 | C1 |
| checkMandatoryFieldTeaching() | C1 | C4 | C4 |
| UpdateTeaching() | C1 | C1 | C1 |
| DeleteRole() | C3 | C3 | C3 |
| InsertRole() | C3 | C3 | C3 |
| checkMandatoryFieldRole() | C3 | C4 | C3 |
| UpdateUser() | C2 | C2 | C2 |
| ExistUser() | C2 | C2 | C2 |
| InsertUser() | C2 | C2 | C2 |
| DeleteUser() | C2 | C2 | C2 |
| checkMandatoryFieldUser() | C2 | C4 | C2 |

Dari tabel 4.3 diatas dapat diketahui bahwa operasi `UpdateUser`, `Exist User`, `DeleteUser`, `InsertUser` sudah sesuai dengan rekomendasi pakar dikelompokkan menjadi satu kelas karena memiliki tanggungjawab terhadap pengelolaan objek `User`. Hal ini dikarenakan konsep operasi "User" berhasil terdeteksi oleh metode TF-IDF setelah token "Insert", Update" dan "Delete"

dimasukkan sebagai daftar *Stop Word*, sehingga dalam penghitungan TF, yang diperhitungkan hanyalah term "User" dimana masing-masing operasi yang terdeteksi memiliki token "User".

Kasus lain terjadi dalam percobaan ini yaitu terpisahnya operasi `checkMandatoryFieldUser` dengan C1 yang secara konseptual memiliki kesamaan konsep dengan C1 yaitu "`User`". Hal ini bisa terjadi karena pada dasarnya metode TF-IDF mengidentifikasi kemunculan tiap token pada sebuah korpus dokumen dalam hal ini nama operasi, sehingga jika dihitung frekuensi ditemukannya term yang sama antara operasi `checkMandatoryFieldUser` dengan `checkMandatoryFieldTeaching` lebih tinggi nilainya dibandingkan dengan operasi `InsertUser` atau `UpdateUser`. Hal ini ditunjukkan pada perhitungan TF-IDF normal untuk Operasi checkMandatoryFieldUser pada tabel 4-4.

**Tabel 4-4 Bobot TF-IDF Operasi `checkMandatoryFieldUser` Dengan TF-IDF Normal**

| Term | **TF-IDF (sebelum identifikasi root)** |
|---|---|
| check | 1.079181246047625 |
| delete | 0 |
| exist | 0 |
| field | 1.079181246047625 |
| insert | 0 |
| mandatory | 1.079181246047625 |
| p | 1.0000000000000002 |
| role | 0 |
| table | 0 |
| teaching | 0 |
| update | 0 |
| user | 2.18188720114459 |
| void | 0 |

Dari tabel 4.4 dapat diketahui bahwa operasi `checkMandatoryFieldUser` lebih memiliki kecondongan kesamaan konsep dengan term `mandatory` dan term `field` dibandingkan dengan term "`user`". Sehingga dengan perhitungan kesamaan *cosine*, akan didapatkan bahwa operasi `checkMandatoryFieldUser` memiliki nilai kemiripan lebih tinggi dengan operasi `checkMandatoryFieldTeaching`.

Sedangkan dengan menggunakan metode LSI termodifikasi diketahui bahwa operasi `UpdateUser`, `Exist User`, `DeleteUser`, `InsertUser` dan `checkMandatoryFieldUser` sudah sesuai dengan rekomendasi pakar dikelompokkan menjadi satu kelas karena memiliki tanggungjawab terhadap pengelolaan objek User. Hal ini dikarenakan konsep operasi "User" berhasil terdeteksi oleh metode TF-IDF termodifikasi *WordDependency* berhasil mengidentifikasi root kalimat `checkMandatoryFieldUser`, yaitu term User. Kemudian term `user` yang memiliki *dependency* sebagai *root* ini ditambahkan bobot TF-IDF-nya sehingga dalam perhitungan *cosine Similarity*, operasi `checkMandatoryFieldUser` akan diidentifikasi lebih mirip dengan `insertUser`, `DeleteUser` dan `UpdateUser`. Hasil perhitungan TF-IDF untuk kalimat `checkMandatoryFieldUser` ditunjukkan pada tabel 4-5 berikut.

**Tabel 4-5 Bobot TF-IDF Operasi `checkMandatoryFieldUser` Dengan TF-IDF termodifikasi**

| Term | **TF-IDF modified (setelah identifikasi root)** |
|---|---|
| check | 0.10791812460476252 |
| delete | 0 |
| exist | 0 |
| field | 0.10791812460476252 |
| insert | 0 |
| mandatory | 0.10791812460476252 |
| p | 0.10000000000000003 |
| role | 0 |
| table | 0 |
| teaching | 0 |
| update | 0 |
| user | 1.1181887201144591 |
| void | 0 |

## 4.5 Hasil Uji Coba Pada Proyek Kode Sumber Terbuka

Pengujian tahap kedua dilakukan pada kelas-kelas yang ada paka proyek perangkat lunak kode sumber terbuka. Pada pengujian ini, hasil identifikasi rekomendasi pemindahan kelas akan dibandingkan dengan hasil identifikasi yang telah dilakukan oleh pakar rekayasa perangkat lunak. Hasil identifikasi yang dilakukan oleh pakar perangkat lunak didapatkan dari penelitian Bavota, dkk pada

11 kelas kode sumber terbuka. Konstanta pembobotan yang digunakan dalam pengujian tahap dua ini adalah konstanta paling optimal yang didapatkan pada penelitian Bavota,dkk yaitu; ***wCSM* = 0.7, *wCDM* = 0.2, *wSSM* = 0.1.**

### 4.5.1 Hasil Pengujian Pada Kelas Database.java

Kelas Address.java merupakan kelas kode sumber terbuka pada proyek ApacheHSQLDB. Pada kelas ini terdapat 40 operasi dan terindikasi God Class. Pengujian dilakukan pada kelas Database.java dengan menggunakan dua metode yaitu LSI dan LSI termodifikasi TF-IDF dengan *word dependency* (LSI-WD). Hasil dari dua metode ini akan dibandingkan dengan hasil dari penelitian Bavota dkk (Bavota, et al., 2010) yang terdiri atas rekomendasi pakar rekayasa perangkat lunak dan metode yang diusulkan oleh Bavota, dkk. Pengujian dilakukan dengan menggunakan konstanta bobot CSM:0.7, CDM:0.2, dan SSM:0,1. Nilai konstanta ini digunakan sesuai dengan rekomendasi dari Bavota, dkk untuk parameter CSM, CDM, dan SSM paling optimal. Hasil dari uji coba pada kelas Database.java tampak pada tabel 4.6 berikut dimana Cx menujukkan rekomendasi refaktorisasi, C1 berarti operasi tetap pada kelas semula, sedangkan C2 berarti operasi berpindah pada kelas ke-2, dan seterusnya.

**Tabel 4-6 Hasil Pengujian Pada Kelas Database.java**

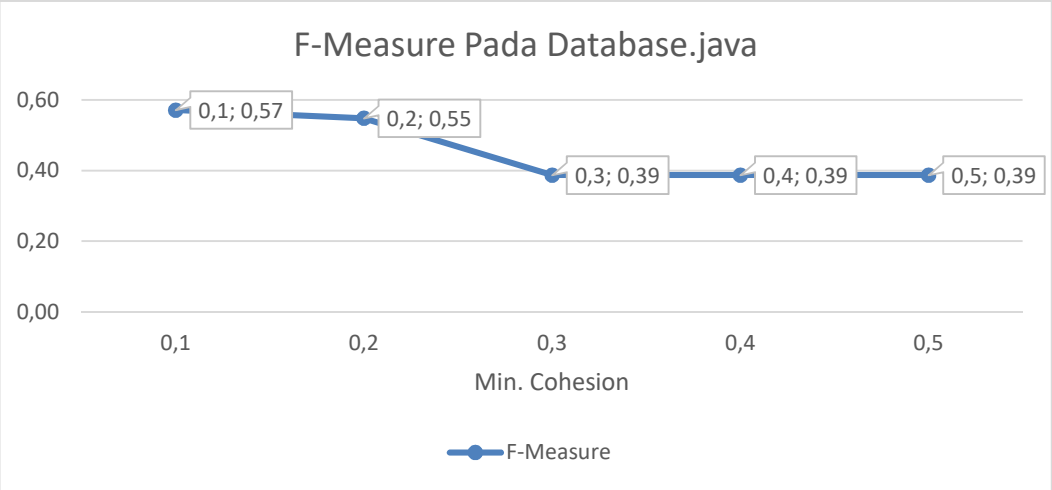
| No. | Operasi | Hasil Pakar | Rekomendasi Sistem | | | | |
|---|---|---|---|---|---|---|---|
| | | | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 1 | open() | C1 | C1 | C1 | C4 | C4 | C4 |
| 2 | reopen() | C1 | C1 | C1 | C2 | C2 | C2 |
| 3 | clearStructures() | C1 | C1 | C1 | C5 | C5 | C5 |
| 4 | getType() | C1 | C1 | C1 | C1 | C1 | C1 |
| 5 | getPath() | C1 | C1 | C1 | C1 | C1 | C1 |
| 6 | getProperties() | C1 | C2 | C3 | C5 | C5 | C5 |
| 7 | isShutdown() | C1 | C1 | C1 | C4 | C4 | C4 |
| 8 | connect(String username, String password) | C1 | C1 | C1 | C2 | C2 | C2 |
| 9 | setReadOnly() | C1 | C1 | C1 | C2 | C2 | C2 |
| 10 | setFilesReadOnly() | C1 | C1 | C1 | C2 | C2 | C2 |
| 11 | isFilesReadOnly() | C1 | C1 | C1 | C2 | C2 | C2 |
| 12 | isFilesInJar() | C1 | C1 | C1 | C5 | C5 | C5 |
| 13 | getUserManager() | C1 | C1 | C1 | C5 | C5 | C5 |
| 14 | setReferentialIntegrity(boolean ref) | C1 | C1 | C2 | C3 | C3 | C3 |
| 15 | isReferentialIntegrity() | C1 | C1 | C2 | C3 | C3 | C3 |
| 16 | getAliasMap() | C1 | C1 | C3 | C5 | C5 | C5 |
| 17 | getJavaName(String s) | C1 | C1 | C1 | C1 | C1 | C1 |

| No. | Operasi | Hasil Pakar | Rekomendasi Sistem | | | | |
|---|---|---|---|---|---|---|---|
| | | | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 18 | getTable(Session session, String name) | C2 | C1 | C1 | C1 | C1 | C1 |
| 19 | getUserTable(Session session, String name) | C2 | C1 | C1 | C1 | C1 | C1 |
| 20 | linkTable(Table t) | C2 | C1 | C1 | C1 | C1 | C1 |
| 21 | setIgnoreCase(boolean b) | C1 | C1 | C2 | C3 | C3 | C3 |
| 22 | isIgnoreCase() | C1 | C1 | C2 | C3 | C3 | C3 |
| 23 | findUserTableForIndex(Session session, String name) | C2 | C1 | C1 | C1 | C1 | C1 |
| 24 | getTableIndex(Table table) | C2 | C1 | C1 | C1 | C1 | C1 |
| 25 | dropIndex(Session session, String indexname, String tableName, boolean ifExists) | C2 | C1 | C1 | C1 | C1 | C1 |
| 26 | getSessionManager() | C1 | C1 | C1 | C5 | C5 | C5 |
| 27 | finalize() | C1 | C1 | C1 | C5 | C5 | C5 |
| 28 | close(int closemode) | C1 | C1 | C1 | C1 | C1 | C1 |
| 29 | checkColumnIsInView(String table, String column) | C2 | C1 | C1 | C1 | C1 | C1 |
| 30 | getViewsWithView(View view) | C2 | C1 | C1 | C1 | C1 | C1 |
| 31 | getViewsWithTable(String table, String column) | C2 | C1 | C1 | C1 | C1 | C1 |
| 32 | getViewsWithSequence(NumberSequence sequence) | C2 | C1 | C1 | C1 | C1 | C1 |
| 33 | recompileViews(String table) | C2 | C1 | C1 | C1 | C1 | C1 |
| 34 | removeExportedKeys(Table toDrop) | C2 | C1 | C1 | C1 | C1 | C1 |
| 35 | dropTrigger(Session session, String name) | C2 | C1 | C1 | C1 | C1 | C1 |
| 36 | setMetaDirty(boolean resetPrepared) | C1 | C1 | C3 | C1 | C1 | C1 |
| 37 | setState(int state) | C1 | C1 | C1 | C4 | C4 | C4 |
| 38 | getState() | C1 | C1 | C1 | C4 | C4 | C4 |
| 39 | getStateString() | C1 | C1 | C1 | C4 | C4 | C4 |
| 40 | getURI() | C1 | C1 | C1 | C1 | C1 | C1 |
| | **Precision** | | **0.50** | **0.50** | **0.50** | **0.50** | **0.50** |
| | **Recall** | | **0.67** | **0.61** | **0.32** | **0.32** | **0.32** |

Uji coba yang dilakukan pada kelas Database.java menunjukkan bahwa metode yang diusulkan, yaitu LSI dengan modifikasi pada TF-IDF menggunakan *word dependency* menghasilkan nilai *precision* dan *recall* tertinggi pada variabel *min.Cohesion* 0.1 yaitu 0.57. Hasil pengujian pada kelas Database.java menunjukkan nilai precision yang cukup rendah yaitu 0.5. Hal ini dikarenakan pada kelas Database.java banyak digunakan operasi-operasi yang hanya menggunakan single term atau disertai dengan *stopword* dengan operasi yang lainnya sehingga nilai kemiripan operasi tersebut dengan operasi lainnya menjadi rendah. Contoh

nama operasi yang menggunakan single term adalah : open, reopen, URI, State, close, finalize, path, type.

Hasil perhitungan F-Measure Database.java ditampilkan pada gambar 4.3 berikut.




**Gambar 4.3 F-Measure pada Database.java**

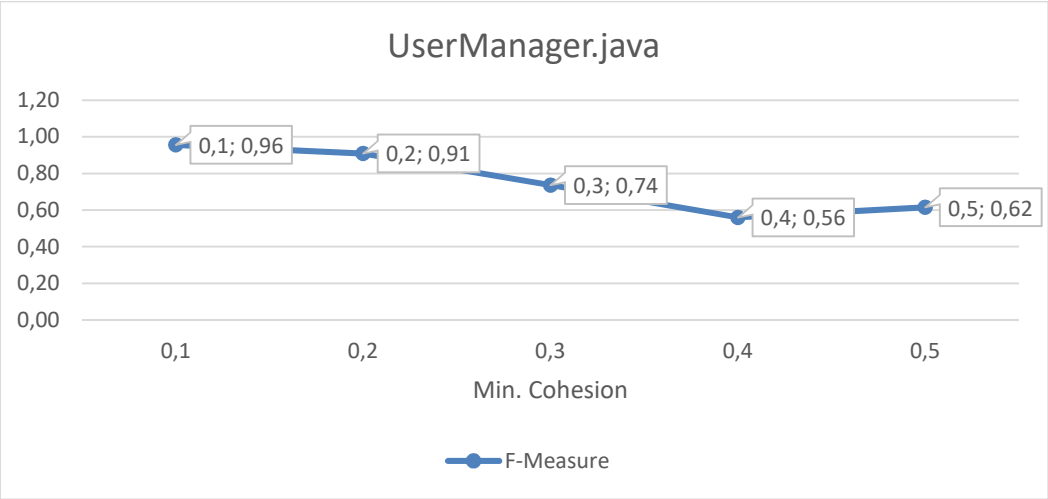
### 4.5.2 Hasil Pengujian Pada Kelas UserManager.java

Kelas UserManager.java merupakan kelas kode sumber terbuka pada proyek ApacheHSQLDB. Pada kelas ini terdapat 13 operasi dan terindikasi God Class. Pengujian dilakukan pada kelas UserManager.java dengan menggunakan dua metode yaitu LSI dan LSI termodifikasi TF-IDF dengan *word dependency* (LSI-WD). Hasil dari dua metode ini akan dibandingkan dengan hasil dari penelitian Bavota dkk (Bavota, et al., 2010) yang terdiri atas rekomendasi pakar rekayasa perangkat lunak dan metode yang diusulkan oleh Bavota, dkk. Pengujian dilakukan dengan menggunakan konstanta bobot CSM:0.7, CDM:0.2, dan SSM:0,1. Hal ini digunakan sesuai dengan rekomendasi dari Bavota, dkk untuk parameter paling optimum. Hasil dari uji coba pada kelas UserManager.java tampak pada tabel 4.8 berikut dimana Cx menujukkan rekomendasi refaktorisasi, C1 berarti operasi tetap pada kelas semula, sedangkan C2 berarti operasi berpindah pada kelas ke-2, dan seterusnya.

**Tabel 4-7 Hasil Pengujian Pada Kelas UserManager.java**

| No. | Operasi | Hasil Pakar | Rekomendasi Sistem | | | | |
|---|---|---|---|---|---|---|---|
| | | | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 1 | getRightsArray(int rights) | C2 | C2 | C2 | C1 | C1 | |
| 2 | getRightsArraySub(int right) | C2 | C2 | C2 | C1 | C1 | |
| 3 | createUser(String name, String password) | C2 | C2 | C2 | C2 | C2 | C2 |
| 4 | dropUser(String name) | C2 | C2 | C2 | C2 | C2 | C2 |
| 5 | getUser(String name, String password) | C2 | C2 | C2 | C2 | C2 | C2 |
| 6 | getUsers() | C2 | C2 | C2 | C2 | C2 | |
| 7 | grant(String name, Object dbobject, int rights) | C1 | C2 | C2 | | C1 | |
| 8 | revoke(String name, Object dbobject, int rights) | C1 | C2 | C2 | C1 | | |
| 9 | exists(String name) | C2 | C2 | C2 | | | |
| 10 | get(String name) | C2 | C2 | C2 | C2 | C2 | |
| 11 | removeDbObject(Object dbob-ject) | C2 | C2 | C1 | | | |
| 12 | listVisibleUsers(Session ses-sion, boolean andPublicUser) | C2 | C2 | C2 | C2 | C2 | C2 |
| 13 | getGrantedClassNames() | C1 | C1 | | | | |
| | **Precision** | | **0.92** | **0.91** | **0.88** | **0.88** | **1.00** |
| | **Recall** | | **1.00** | **0.91** | **0.64** | **0.41** | **0.44** |

Uji coba yang dilakukan pada kelas UserManager.java menunjukkan bahwa metode yang diusulkan, yaitu LSI dengan modifikasi pada TF-IDF menggunakan *word dependency* menghasilkan nilai *precision* dan *recall* tertinggi pada variabel *min.Cohesion* 0.1 yaitu 0.96. Hasil perhitungan F-Measure UserManager.java ditampilkan pada gambar 4.4 berikut.




**Gambar 4.4 F-Measure pada kelas UserManager.java**

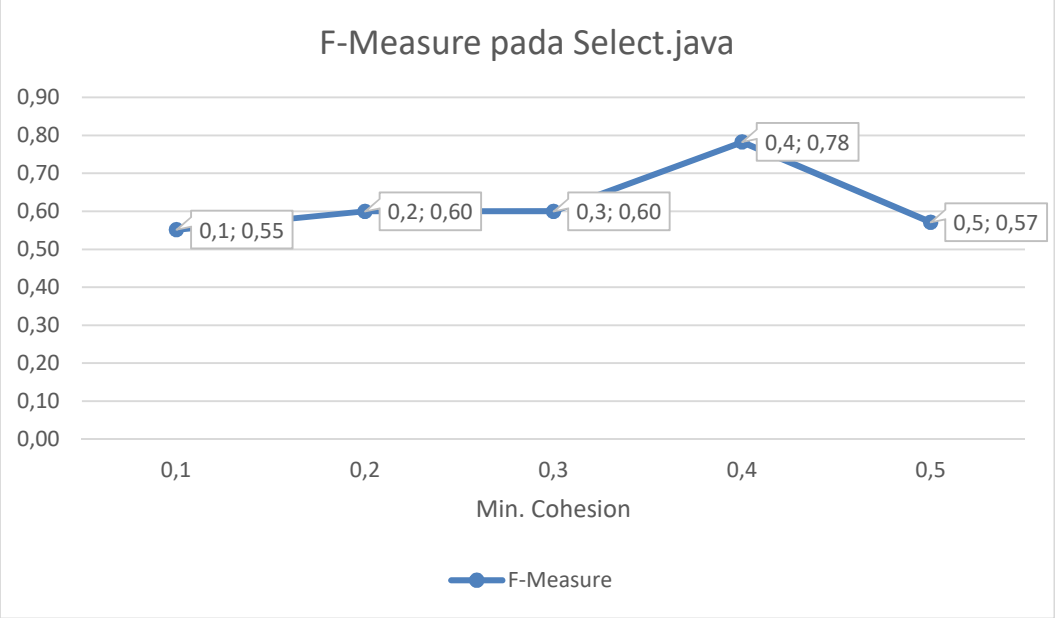
### 4.5.3 Hasil Pengujian Pada Kelas Select.java

Kelas Select.java merupakan kelas kode sumber terbuka pada proyek ApacheHSQLDB. Pada kelas ini terdapat 14 operasi dan terindikasi God Class. Pengujian dilakukan pada kelas Select.java dengan menggunakan dua metode yaitu LSI dan LSI termodifikasi TF-IDF dengan *word dependency* (LSI-WD). Hasil dari dua metode ini akan dibandingkan dengan hasil dari penelitian Bavota dkk (Bavota, et al., 2010) yang terdiri atas rekomendasi pakar rekayasa perangkat lunak dan metode yang diusulkan oleh Bavota, dkk. Pengujian dilakukan dengan menggunakan konstanta bobot CSM:0.7, CDM:0.2, dan SSM:0,1. Hal ini digunakan sesuai dengan rekomendasi dari Bavota, dkk untuk parameter paling optimum. Hasil dari uji coba pada kelas Select.java tampak pada tabel 4.8 berikut dimana Cx menujukkan rekomendasi refaktorisasi, C1 berarti operasi tetap pada kelas semula, sedangkan C2 berarti operasi berpindah pada kelas ke-2, dan seterusnya.

**Tabel 4-8 Hasil Pengujian Pada Kelas Select.java**

| No | Operasi | Hasil Pakar | Rekomendasi Sistem | | | | |
|---|---|---|---|---|---|---|---|
| | | | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 1 | resolve() | C2 | C1 | C1 | C1 | C2 | |
| 2 | resolve(TableFilter f, boolean ownfilter) | C2 | C1 | C1 | C1 | C4 | |
| 3 | checkResolved() | C2 | C2 | C1 | C1 | C2 | |
| 4 | getValue(int type) | C2 | C1 | C1 | C1 | C2 | |
| 5 | getResult(int maxrows) | C2 | C1 | C1 | C1 | C1 | C1 |
| 6 | updateAggregateRow(Object row, Object n, int len) | C2 | C1 | C2 | C2 | C2 | |
| 7 | addAggregateRow(Result x, Object row, int len, int count) | C2 | C1 | C2 | C2 | C2 | |
| 8 | removeDuplicates(Result r) | C1 | C1 | C1 | C1 | C3 | C2 |
| 9 | removeSecond(Result r, Result minus) | C1 | C1 | C1 | C1 | C3 | |
| 10 | removeDifferent(Result r, Result r2) | C1 | C1 | C1 | C1 | C3 | C2 |
| 11 | sortResult(Result r, int order, int way) | C1 | C1 | C1 | C1 | C1 | C1 |
| 12 | trimResult(Result r, int maxrows) | C1 | C1 | C1 | C1 | C1 | C1 |
| 13 | compareRecord(Object a, Object b, Result r, int order, int way) | C1 | C1 | C1 | C1 | C1 | |
| 14 | compareRecord(Object a, Object b, Result r, int len) | C1 | C1 | C1 | C1 | C1 | |
| | **Precision** | | 0.53 | 0.56 | 0.56 | 0.69 | 0.50 |
| | **Recall** | | 0.57 | 0.64 | 0.64 | 0.90 | 0.67 |

Uji coba yang dilakukan pada kelas Select.java menunjukkan bahwa metode yang diusulkan, yaitu LSI dengan modifikasi pada TF-IDF menggunakan *word dependency* menghasilkan nilai *precision* dan *recall* tertinggi pada variabel *min.Cohesion* 0.4 yaitu 0.78. Hasil perhitungan F-Measure Select.java ditampilkan pada gambar 4.5 berikut.




**Gambar 4.5 F-Measure pada Select.java**

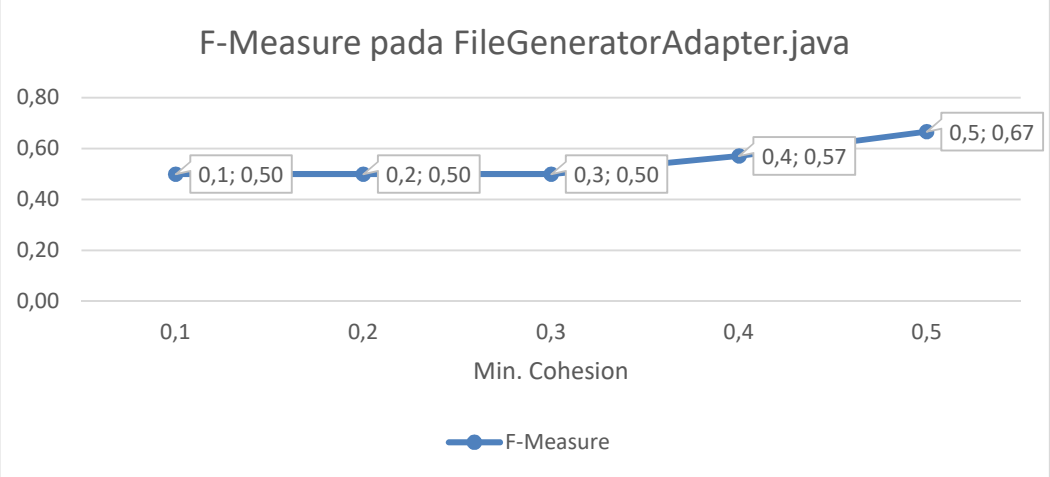
### 4.5.4 Hasil Pengujian Pada Kelas FileGeneratorAdapter.java

Kelas FileGeneratorAdapter.java merupakan kelas kode sumber terbuka pada proyek ApacheHSQLDB. Pada kelas ini terdapat 14 operasi dan terindikasi God Class. Pengujian dilakukan pada kelas Select.java dengan menggunakan dua metode yaitu LSI dan LSI termodifikasi TF-IDF dengan *word dependency* (LSI-WD). Hasil dari dua metode ini akan dibandingkan dengan hasil dari penelitian Bavota dkk (Bavota, et al., 2010) yang terdiri atas rekomendasi pakar rekayasa perangkat lunak dan metode yang diusulkan oleh Bavota, dkk. Pengujian dilakukan dengan menggunakan konstanta bobot CSM:0.7, CDM:0.2, dan SSM:0,1. Hal ini digunakan sesuai dengan rekomendasi dari Bavota, dkk untuk parameter paling optimum. Hasil dari uji coba pada kelas Select.java tampak pada tabel 4.9 berikut dimana Cx menujukkan rekomendasi refaktorisasi, C1 berarti operasi tetap pada kelas semula, sedangkan C2 berarti operasi berpindah pada kelas ke-2, dan seterusnya.

**Tabel 4-9 Hasil Pengujian Pada Kelas FileGeneratorAdapter.java**

| No | Operasi | Hasil Pakar | Rekomendasi Sistem | | | | |
|---|---|---|---|---|---|---|---|
| | | | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 1 | generate(Collection elements, boolean deps) | C1 | C1 | C1 | C1 | C1 | C1 |
| 2 | generateFiles(Collection elements, String path, boolean deps) | C1 | C1 | C1 | C1 | C1 | C1 |
| 3 | generateFileList(Collection elements, boolean deps) | C1 | C1 | C1 | C1 | C1 | C1 |
| 4 | createTempDir() | C2 | C1 | C1 | C1 | C2 | C2 |
| 5 | traverseDir(File dir, FileAction action) | C2 | C1 | C1 | C1 | C2 | C2 |
| 6 | readAllFiles(File dir) | C2 | C1 | C1 | C1 | C3 | C3 |
| 7 | act(File f) | C2 | C2 | C2 | C2 | C4 | C4 |
| 8 | deleteDir(File dir) | C2 | C1 | C1 | C1 | C2 | C2 |
| 9 | readFileNames(File dir) | C2 | C1 | C1 | C1 | C3 | C3 |
| Precision | | | 0.57 | 0.57 | 0.57 | 0.67 | 0.67 |
| Recall | | | 0.44 | 0.44 | 0.44 | 0.50 | 0.67 |

Uji coba yang dilakukan pada kelas FileGeneratorAdapter.java menunjukkan bahwa metode yang diusulkan, yaitu LSI dengan modifikasi pada TF-IDF menggunakan *word dependency* menghasilkan nilai *precision* dan *recall* tertinggi pada variabel *min.Cohesion* 0.5 yaitu 0.67. Hasil perhitungan F-Measure FileGeneratorAdapter.java ditampilkan pada gambar 4.6 berikut.




**Gambar 4.6 F-Measure pada FileGeneratorAdapter.java**

### 4.5.5 Hasil Pengujian Pada Kelas JFreeChart.java

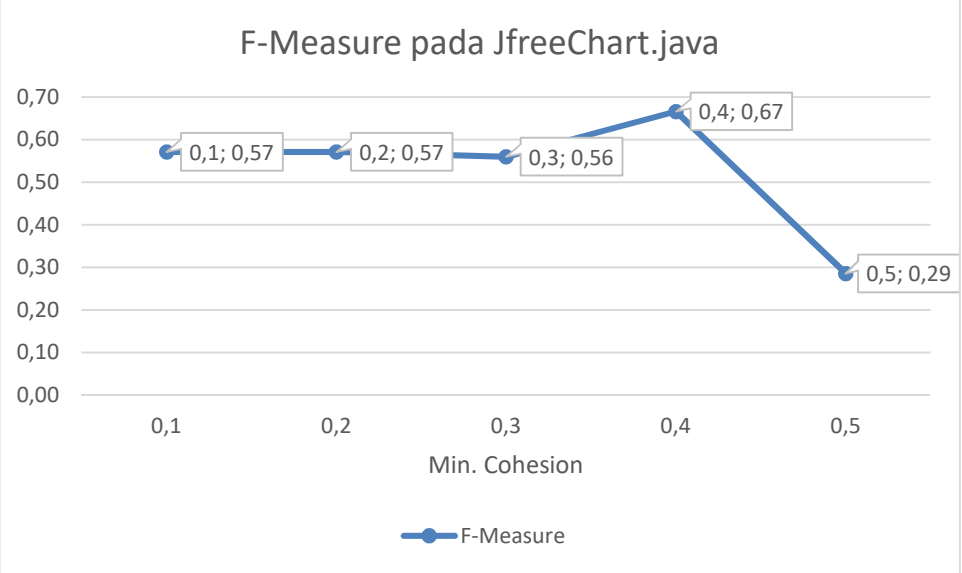
Kelas JFreeChart.java merupakan kelas kode sumber terbuka pada proyek JFreeChart. Pada kelas ini terdapat 40 operasi dan terindikasi God Class. Pengujian dilakukan pada kelas Select.java dengan menggunakan dua metode yaitu LSI dan LSI termodifikasi TF-IDF dengan *word dependency* (LSI-WD). Hasil dari dua metode ini akan dibandingkan dengan hasil dari penelitian Bavota dkk (Bavota, et al., 2010) yang terdiri atas rekomendasi pakar rekayasa perangkat lunak dan metode yang diusulkan oleh Bavota, dkk. Pengujian dilakukan dengan menggunakan konstanta bobot CSM:0.7, CDM:0.2, dan SSM:0,1. Hal ini digunakan sesuai dengan rekomendasi dari Bavota, dkk untuk parameter paling optimum. Hasil dari uji coba pada kelas JFreeChart.java tampak pada tabel 4.10 berikut dimana Cx menujukkan rekomendasi refaktorisasi, C1 berarti operasi tetap pada kelas semula, sedangkan C2 berarti operasi berpindah pada kelas ke-2, dan seterusnya.

**Tabel 4-10 Hasil Pengujian Pada Kelas JFreeChart.java**

| No. | Operasi | Hasil Pakar | Rekomendasi Sistem | | | | |
|---|---|---|---|---|---|---|---|
| | | | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 1 | getTitle() | C1 | C1 | C1 | C3 | C4 | C2 |
| 2 | getLegend() | C1 | C1 | C1 | C1 | C2 | C2 |
| 3 | setLegend(Legend legend) | C1 | C1 | C1 | C1 | C2 | C2 |
| 4 | getPlot() | C1 | C1 | C1 | C3 | C4 | C2 |
| 5 | getAntiAlias() | C1 | C1 | C1 | C1 | C3 | C2 |
| 6 | setAntiAlias(boolean flag) | C1 | C1 | C1 | C1 | C3 | C2 |
| 7 | getChartBackgroundPaint() | C1 | C1 | C1 | C1 | C2 | C1 |
| 8 | setChartBackgroundPaint(Paint paint) | C1 | C1 | C1 | C1 | C2 | C1 |
| 9 | setSeriesPaint(Paint[] paint) | C2 | C1 | C1 | C1 | C2 | C1 |
| 10 | setSeriesStroke(Stroke[] stroke) | C2 | C1 | C1 | C1 | C2 | C1 |
| 11 | setSeriesOutlinePaint(Paint[] paint) | C2 | C1 | C1 | C1 | C2 | C1 |
| 12 | setSeriesOutlineStroke(Stroke[] stroke) | C2 | C1 | C1 | C1 | C2 | C1 |
| 13 | getSeriesPaint(int index) | C2 | C1 | C1 | C1 | C2 | C1 |
| 14 | getSeriesStroke(int index) | C2 | C1 | C1 | C1 | C2 | C1 |
| 15 | getSeriesOutlinePaint(int index) | C2 | C1 | C1 | C1 | C2 | C1 |
| 16 | getSeriesOutlineStroke(int index) | C2 | C1 | C1 | C1 | C2 | C1 |
| 17 | draw(Graphics2D g2, Rectangle2D char-tArea) | C1 | C1 | C1 | C1 | C4 | C2 |
| 18 | fireChartChanged() | C1 | C1 | C1 | C1 | C2 | C2 |
| 19 | addChangeListener(ChartChangeListener listener) | C1 | C1 | C1 | C1 | C1 | C2 |
| 20 | removeChangeListener(ChartChangeLis-tener listener) | C1 | C1 | C1 | C1 | C3 | C2 |
| 21 | notifyListeners(ChartChangeEvent event) | C1 | C1 | C1 | C1 | C2 | C2 |
| 22 | titleChanged(TitleChangeEvent event) | C1 | C1 | C1 | C1 | C3 | C2 |

| No. | Operasi | Hasil Pakar | Rekomendasi Sistem | | | | |
|---|---|---|---|---|---|---|---|
| | | | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 23 | legendChanged(LegendChangeEvent event) | C1 | C1 | C1 | C1 | C3 | C2 |
| 24 | plotChanged(PlotChangeEvent event) | C1 | C1 | C1 | C1 | C3 | C2 |
| **Precision** | | | 0.50 | 0.50 | 0.50 | 0.90 | 0.50 |
| **Recall** | | | 0.67 | 0.67 | 0.64 | 1.00 | 0.20 |

Uji coba yang dilakukan pada kelas JFreeChart.java menunjukkan bahwa metode yang diusulkan, yaitu LSI dengan modifikasi pada TF-IDF menggunakan *word dependency* menghasilkan nilai *precision* dan *recall* tertinggi pada variabel *min.Cohesion* 0.4 yaitu 0.67. Hasil perhitungan F-Measure JFreeChart.java ditampilkan pada gambar 4.7 berikut.




**Gambar 4.7 F-Measure pada JfreeChart.java**

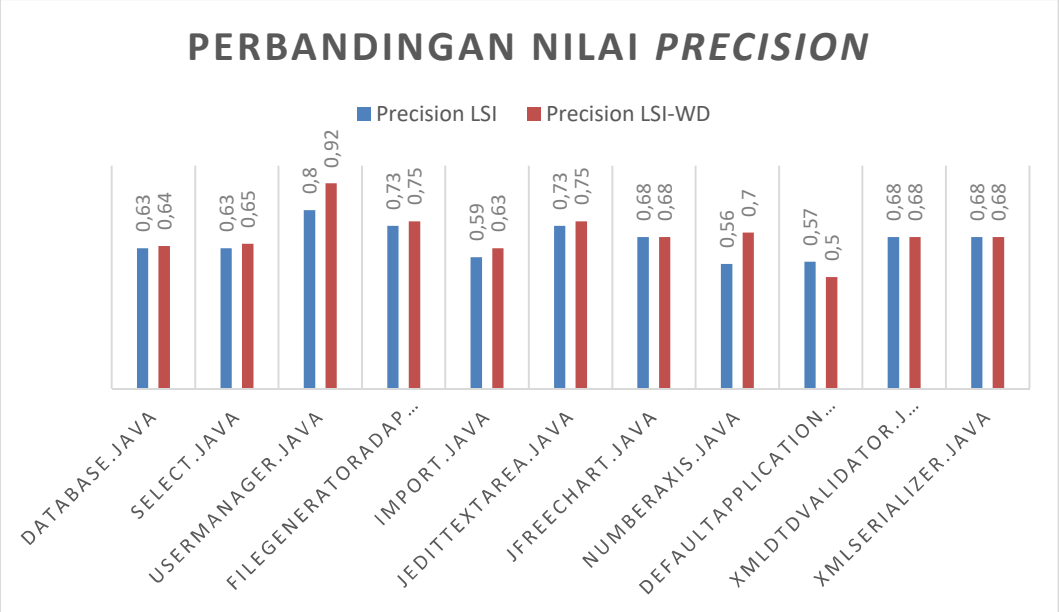
Dari pengujian yang dilakukan terhadap 11 kelas dari proyek kode sumber terbuka, kemudian dihitung rata-rata nilai precision, recall dan F-Measure dari masing-masing kelas. Nilai rata-rata ini akan menunjukkan perbandingan precision recall dan F-Measure yang didapatkan dari metode yang diusulkan dibandingkan dengan metode pada penelitian sebelumnya. Rata-rata nilai *precision* dan *recall* dan *F-Measure* hasil pengujian pada proyek kode sumber terbuka disajikan pada tabel 4.11.

**Tabel 4-11 Hasil pengujian pada kode sumber terbuka**

| Proyek / Kelas | LSI | | | LSI termodifikasi TF-IDF word dependency | | |
|---|---|---|---|---|---|---|
| | Prec | Rec | F Measure | Prec | Rec | F measure |
| Database.java | 0.63 | 0.93 | 0.75 | 0.64 | 1.00 | 0.78 |
| Select.java | 0.63 | 0.86 | 0.73 | 0.65 | 0.93 | 0.76 |
| UserManager.java | 0.80 | 0.92 | 0.86 | 0.92 | 1.00 | 0.96 |
| FileGenerato-rAdapter.java | 0.73 | 0.89 | 0.80 | 0.75 | 1.00 | 0.86 |
| Import.java | 0.59 | 1.00 | 0.74 | 0.63 | 1.00 | 0.77 |
| JeditTextArea.java | 0.73 | 0.89 | 0.80 | 0.75 | 1.00 | 0.86 |
| JFreeChart.java | 0.68 | 1.00 | 0.95 | 0.68 | 1.00 | 0.81 |
| NumberAxis.java | 0.56 | 1.00 | 0.82 | 0.70 | 1.00 | 0.82 |
| DefaultApplica-tionModel.java | 0.57 | 0.93 | 0.70 | 0.50 | 1.00 | 0.67 |
| XMLDTDValidator.java | 0.68 | 1.00 | 0.81 | 0.68 | 1.00 | 0.81 |
| XMLSerializer.java | 0.68 | 1.00 | 0.81 | 0.68 | 1.00 | 0.81 |

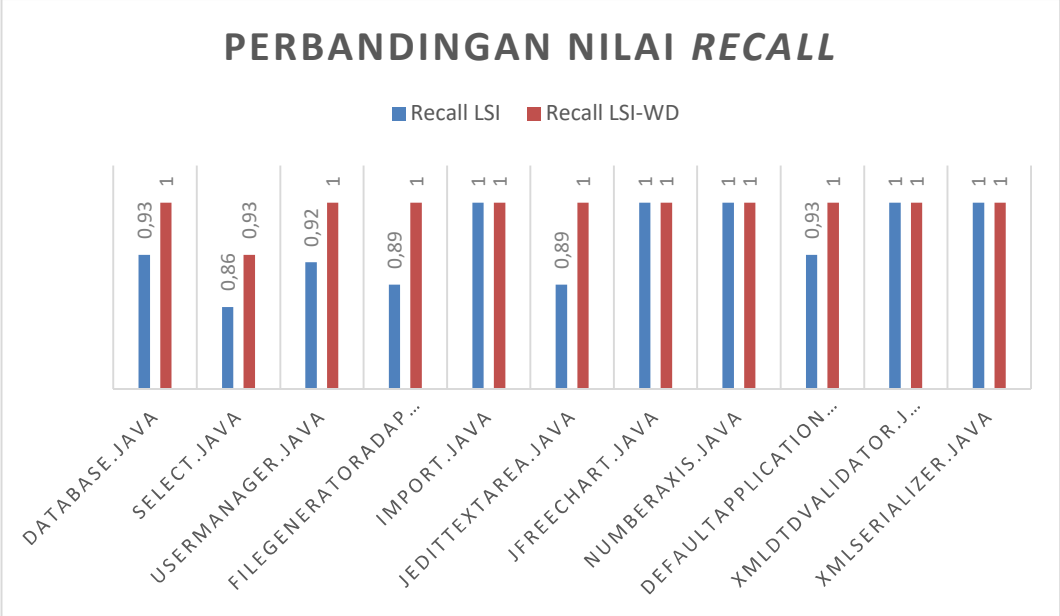
Dari hasil rata-rata nilai precision, recall dan F-Measure tiap kelas pada 11 kelas kode sumber terbuka tampak bahwa metode yang diusulkan relatif lebih baik dalam mengidentifikasi peluang refaktorisasi sesuai dengan rekomendasi yang dihasilkan oleh pakar rekayasa perangkat lunak.

Perbandingan nilai *precision* tiap kelas kode sumber terbuka menggunakan metode LSI dibandingkan dengan LSI termodifikasi ditampilkan pada gambar 4-8.



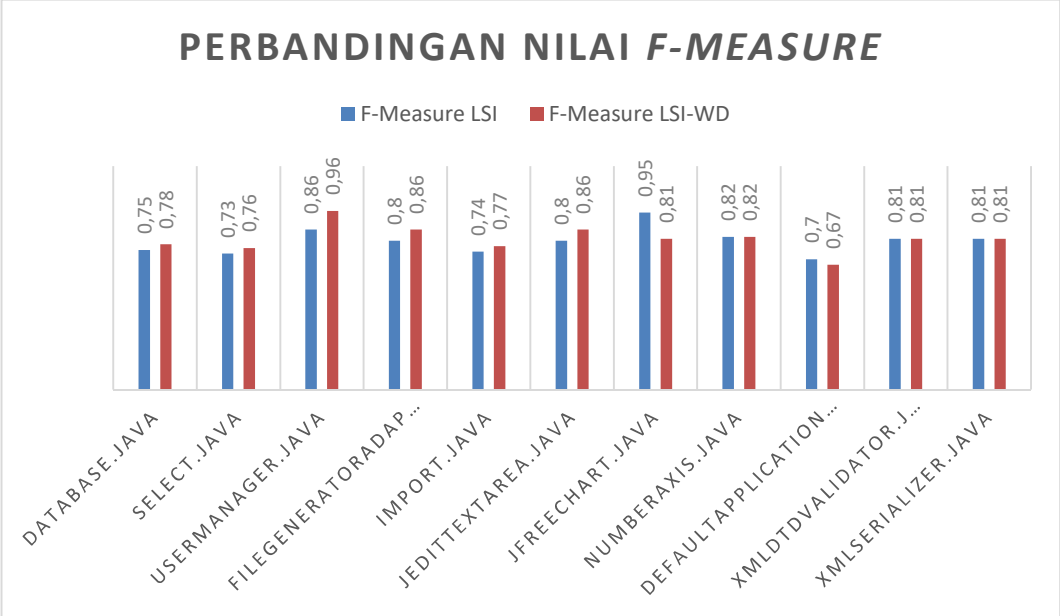

**Gambar 4.8 Hasil Perhitungan Precision**

Perbandingan nilai *recall* tiap kelas kode sumber terbuka menggunakan metode LSI dibandingkan dengan LSI termodifikasi ditampilkan pada gambar 4-9.




**Gambar 4.9 Hasil Perhitungan Recall**

Perbandingan nilai *F-Measure* tiap kelas kode sumber terbuka menggunakan metode LSI dibandingkan dengan LSI termodifikasi ditampilkan pada gambar 4-10.




**Gambar 4.10 Perbandingan Hasil F-Measure**

## 4.6 Hasil Keseluruhan Pengujian

Dari seluruh hasil pengujian didapatkan nilai precision dan recall paling tinggi pada kelas JFreeChart.java. Dimana kelas ini lebih banyak memiliki term-term yang sama dan lebih mudah diidentifikasi konsep lokasinya seperti plot, chart, series dan menggunakan kalimat berstruktur. Dari hasil pengujian pada kasus kelas kode sumber terbuka, dapat dihitung rata-rata dari *Precision Recall* dan *Fmeasure* yang dihasilkan disajikan pada tabel 4.12

**Tabel 4-12 Rerata Precision, Recall dan FMeasure**

| | **LSI** | **LSI-WD** |
|---|---|---|
| ***Precision*** | 0.698 | 0.708 |
| ***Recall*** | 0.886 | 0.936 |
| ***FMeasure*** | 0.774 | 0.796 |

Dari perhitungan rerata hasil pengujian sistem terhadap metode LSI berdasarkan penelitian sebelumnya dibandingkan dengan metode LSI termodifikasi yang diusulkan, terdapat hasil peningkatan precision recal yang cukup signifikan.yaitu precision meningkat dari 0.68 menjadi 0.70 meningkat 0.02. dan recall dari 0.88 meningkat menjadi 0.93, terjadi peningkatan sebanyak 0.05. hasil pengujuan ini membuktikan bahwa pendekatan konsep lokasi operasi dapat digunakan untuk menambah akurasi sistem identifikasi peluang refaktorisasi.

# BAB 5
# KESIMPULAN DAN SARAN

Bab ini memaparkan kesimpulan yang dapat diambil berdasarkan pada penelitian yang telah dilakukan. Dalam Bab 5 ini diuraikan juga tentang hal-hal yang perlu dipertimbangkan untuk pengembangan penelitian lebih lanjut. Penjelasan yang lebih terperinci tentang hal-hal tersebut diuraikan pada sub-bab berikut.

## 5.1 Kesimpulan

Dari hasil pengamatan yang dilakukan selama proses perancangan, implementasi, serta pengujian terhadap perangkat lunak, dapat diambil kesimpulan diantaranya adalah sebagai berikut :

1. Dalam hal penyusunan kembali kelas untuk menurunkan *coupling* dan meningkatkan *class cohesion* dapat digunakan pendekatan sintaksis. Dengan menggunakan pendekatan *semantic similarity* dapat dibuat sebuah *tools* yang mampu mengidentifikasi kemiripan sebuah operasi dengan konsep sebuah kelas (*class responsibility*).
2. Pendekatan *WordDependency* dapat digunakan untuk menambah akurasi identifikasi kemiripan semantik menggunakan metode LSI dengan memodifikasi perhitungan TF-IDF.
3. Terdapat hasil peningkatan precision recal yang cukup signifikan.yaitu precision meningkat dari 0.68 menjadi 0.70 meningkat 0.02. dan recall dari 0.88 meningkat menjadi 0.93, terjadi peningkatan sebanyak 0.05. hasil pengujuan ini membuktikan bahwa pendekatan konsep lokasi operasi dapat digunakan untuk menambah akurasi sistem identifikasi peluang refaktorisasi..
4. Akurasi sistem menjadi rendah jika menemui nama operasi yang menggunakan single term karena nilai kemiripan kosinenya akan menjadi rendah.
5. Perlu meningkatkan penilaian kemiripan term dengan mempertimbangkan makna kata dengan menggunakan kamus WordNet..

## 5.2 Saran

Berikut merupakan beberapa saran yang dapat digunakan sebagai acuan untuk pengembangan penelitian lebih lanjut di masa yang akan datang. Saran-saran ini didasarkan pada hasil perancangan, implementasi, dan juga pengujian yang telah dilakukan.

1. Perlu tambahan parameter *semantic similarity* untuk menentukan bobot kemiripan sebuah operasi dengan kosep pada sebuah kelas (*class responsibility*), misalnya baris komentar dari *programmer* dan attribut-attribut dalam operasi *body*.
2. Perlu analisa lebih lanjut pada metode pengidentifikasian nama kelas dan operasi dimana terdapat nama kelas atau operasi yang merupakan singkatan atau istilah teknis yang tidak dikenali secara makna hanya berdasarkan kamus WordNet.
3. Perlu dianalisa lebih lanjut tentang algoritma pembobotan yang lebih tepat untuk struktur kelas dan operasi dalam kelas karena peningkatan yang dicapai dalam penelitian ini kurang signifikan.

# DAFTAR PUSTAKA


Akroyd, M., 1996. *AntiPatterns Session Notes.* San Francisco: Object World West.

Atkinson, D. K. T., 2005. *Lightweight detection of program refactorings.* Taipei, Taiwan, IEEE CS Press, pp. 663-670.

Basili, V., Briand, L. & Melo, W., 1995. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering 22 (10),* p. 751–761.

Bavota, G., De Lucia, A. & Oliveto, R., 2010. *A two step technique for extract class refactoring..* New York, ACM, pp. 151-154.

Bavota, G., De Lucia, A. & Oliveto, R., 2011. Identifying extract class refactoring opportunities using structural and semantic cohesion measures. *Journal of System and Software,* p. 397–414.

Binkley, A. & Schach, S., 1998. *Validation of the coupling dependency metric asa predictor of run-time failures and maintenance measures..* Kyoto, Japan, s.n., p. 452–455.

Briand, L., Wüst, J., Ikonomovski, S. & Lounis, H., 1999. *Investigating quality factors in object-oriented designs: an industrial case study.* Los Angeles, California, USA, ACM Press, p. 345–354.

Brown, W. J., Malveau, R. C. & Mowbray, T. J., 1998. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis.* Canada: John Wiley & Sons, Inc..

Chen, K. & Rajlich, V., 2000. *Case study of feature location using dependency graph.* s.l., IEEE Computer Society, pp. 241-249.

Chidamber, S. R. & Kemerer, C. F., 1994. A Metrics Suite for Object Oriented Design. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,* pp. 476-493.

Coad, P. & Yourdon, E., 1991. *Object-Oriented Design, 1st edition.* s.l.:Prentice Hall.

De Lucia, A. O. R. V. L., 2008. *Using structural and semantic metrics to improve class cohesion.* Beijing, China, s.n.

Fokaefs, M., Tsantalis, N., Chatzigeorgiou, A. & Sander, J., 2009. *Decomposing object-oriented class modules using an agglomerative clustering technique.* Canada, Edmonton, p. 93–101.

Fokaefs, M., Tsantalis, N., Stroulia, E. & Chatzigeorgiou, A., 2011. *JDeodorant: identification and application of extract class refactoring.* s.l., s.n.

Fokaefs, M., Tsantalis, N., Stroulia, E. & Chatzigeorgiou, A., 2012. Identification and application of Extract Class refactorings in object-oriented systems. *Journal of Systems and Software,* Issue 85, pp. 2241-2260.

Fowler, M., 1999. *Refactoring: Improving the Design of Existing Code.* s.l.:Addison-Wesley.

Gui, G. & Paul D., S., 2008. *New Coupling and Cohesion Metrics for Evaluation of Software Component.* Zhang Jia Jie, Hunan, China, IEEE, pp. 1181-1186.

Gyimóthy, T., Ferenc, R. & Siket, I., 2005. *Empirical validation of object-oriented metrics on open source software for fault prediction.* s.l., s.n., p. 897–910.

Howe, D., 2009. *RiTa: creativity support for computational literature.* New York, ACM, pp. 205-210.

Marcus, A., Sergeyev, A., Rajlich, V. & Maletic, J., 2004. *An information retrieval approach to concept location in source code.* Delft, The Netherlands, IEEE, pp. 214-223.

Marie-Catherine de Marneffe, B. M. C. D. M., 2006. *Generating typed dependency parses from phrase structure parses.* Genoa, Italy, s.n.

Miller, G. A., 1995. WordNet: A Lexical Database for English. *Communications of the ACM,* pp. 39-41.

Moha, N., Gueheneuc, Y., Duchien, L. & Meur, A., 2010. Decor: a method for the specification and detection of code and design smells. *IEEE Transaction on Software Engineering 36,* pp. 20-36.

Opdyke, W. F., 1992. *Refactoring object-Oriented Frameworks,* Urbana, Illinois: University of Illinois at Urbana-Champaign.

Petrenko, M. & Rajlich, V., 2012. Concept location using program dependencies and information retrieval (DepIR). *Information and Software Technology 55 (2013),* p. 651–659.

Poshyvanyk, D., Guéhéneuc, Y., Marcus, G. Anton, A. & Anton, G., 2007. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Transactions on Software Engineering 33,* p. 420–432.

Simon, F. S. F. L. C., 2001. *Metrics based refactoring.* Lisbon, Portugal, IEEE CS Press, p. of the 5th European Conference on Software Maintenance and Reengineering.

Thomas H. Cormen, C. E. L. R. L. R. C. S., 2009. *Introduction To Algorithms: Third Edition.* Massachusetts: Massachusetts Institute of Technology.

# BIOGRAFI PENULIS

Penulis dilahirkan di Malang pada tanggal 9 April 1987, yang merupakan anak pertama dari dua bersaudara. Penulis telah menempuh pendidikan dasar di SDN Pakiskembar 1 Kabupaten Malang, SLTP Negeri 1 Tumpang Kabupaten Malang, dan SMA Negeri 3 Malang. Pada tahun 2001 penulis melanjutkan pendidikan ke jenjang pendidikan tinggi di S1 Ilmu Komputer Universitas Brawijaya dan lulus tahun 2009. Tahun 2010 penulis mendapatkan amanah untuk menjadi tenaga pengajar di Universitas Brawijaya di Program Studi Teknik Informatika. Kemudian, tahun 2012 penulis melanjutkan studi S2 di Institut Teknologi Sepuluh Nopemeber di Program Studi Teknik Informatika dan lulus sebagai Magister Komputer pada tahun 2017. Penulis mengambil bidang minat Rekayasa Perangkat Lunak baik di jenjang S1 maupun S2. Untuk korespondensi, penulis dapat dihubungi melalui email widhy@ub.ac.id.