View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by ITS Repository

TESIS KI142502

PEMBANGUNAN LINK PENELUSURAN KEBUTUHAN FUNGSIONAL DAN KODE SUMBER BERDASARKAN KEDEKATAN SEMANTIK DAN STRUKTUR KODE SUMBER BERORIENTASI OBJEK

Djoko Pramono
5112201014

DOSEN PEMBIMBING
Dr. Ir. Siti Rochimah, MT.

PROGRAM MAGISTER
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2016

[Halaman ini sengaja dikosongkan]

THESIS KI142502

TRACEABILITY LINK RECOVERY BETWEEN FUNCTIONAL REQUIREMENT AND SOURCE CODE BASED ON SEMANTIC RELATEDNESS AND OBJECT ORIENTED SOURCE CODE STRUCTURE

Djoko Pramono
5112201014

SUPERVISOR
Dr. Ir. Siti Rochimah, MT.

MASTER PROGRAM
SOFTWARE ENGINEERING
INFORMATICS ENGINEERING
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2016

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN TESIS

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar

Magister Komputer (M.Kom.)

di

Institut Teknologi Sepuluh Nopember Surabaya

oleh

Djoko Pramono

5112201014

Dengan Judul :

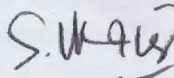
PEMBANGUNAN LINK PENELUSURAN KEBUTUHAN FUNGSIONAL DAN KODE SUMBER BERDASARKAN KEDEKATAN SEMANTIK DAN STRUKTUR KODE SUMBER BERORIENTASI OBJEK

Tanggal Ujian : 20-06-2016

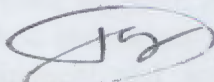
Periode Wisuda : 2016 Genap

Disetujui Oleh:

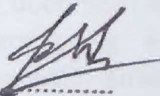
Dr. Ir. Siti Rochimah, MT.
NIP. 19681002 199403 2 001


....
(Pembimbing 1)

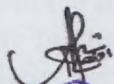
Daniel Oranova, S.Kom. MSc. PD.Eng.
NIP. 19741123 200604 1 001


....
(Penguji 1)

Ratih Nur Esti Anggraini, S.Kom., M.Sc.
NIP. 19841210 201404 2 000


....
(Penguji 2)

Adhatus Sholichah, S.Kom., M.Sc.
NIP. 19850826 201504 2 002


....
(Penguji 3)

Direktur Program Pasca Sarjana,

Prof. Ir. Djauhar Manfaat, M.Sc., Ph.D.
NIP. 19601202 198701 1 001

[Halaman ini sengaja dikosongkan]

TRACEABILITY LINK RECOVERY BETWEEN FUNCTIONAL REQUIREMENT AND SOURCE CODE BASED ON SEMANTIC RELATEDNESS AND OBJECT ORIENTED SOURCE CODE STRUCTURE

Nama mahasiswa : Djoko Pramono
NRP : 5112201014
Pembimbing I : Dr. Ir. Siti Rochimah, MT.

ABSTRACT

Traceability link between requirement document and source code is very helpful in software development and maintenance process. In the process, developer make changes to the source code but often forget to update documents that accompanies the source code. The existence of links between requirement document and source code will increase the speed of finding which parts of the source code that need to be changed when changes are made to requirement. Traceability link helps developers to understand the software that does not have a complete design document as in open source software.

In the source code, a method represents an action performed to complete a requirement. Some method and class work together, one method calls another method with the parameter in the method invocation. This research proposed a method to build links between requirement and source code using natural language processing on the source code and semantic similarity. Semantic similarity is calculated between requirement document with method signature and method invocation. Natural language processing is done on all method declarations in the source code. Further semantic similarity computed between requirement document and natural language phrase to determine the method-method of candidate link. Next do a search in the source code to find the method-method to be executed directly or indirectly by the method candidates. The final value is calculated from the semantic similarity between requirement and method of candidates link and invoked method.

This method is expected to be able to find how and which parts of the source code that runs a requirement. The use of this method is expected to be able to increase accuracy in building links between requirement and source code.

Keywords : traceability link, semantic similarity, natural language processing

[Halaman ini sengaja dikosongkan]

PEMBANGUNAN LINK PENELUSURAN KEBUTUHAN FUNGSIONAL DAN KODE SUMBER BERDASARKAN KEDEKATAN SEMANTIK DAN STRUKTUR KODE SUMBER BERORIENTASI OBJEK

Nama mahasiswa : Djoko Pramono
NRP : 5112201014
Pembimbing I : Dr. Ir. Siti Rochimah, MT.

ABSTRAK

Link penelusuran antara dokumen kebutuhan dan kode sumber sangat membantu dalam proses pengembangan dan pemeliharaan perangkat lunak. Dalam proses pengembangan dan pemeliharaan perangkat lunak, pengembang melakukan perubahan pada kode sumber tetapi sering tidak memperbarui dokumen-dokumen yang menyertai kode sumber. Adanya link penelusuran antara dokumen kebutuhan dengan kode sumber akan meningkatkan kecepatan menemukan bagian mana dari kode sumber yang perlu diubah ketika ada perubahan kebutuhan. Link penelusuran membantu pengembang dalam memahami perangkat lunak yang tidak memiliki dokumen perancangan yang lengkap seperti pada kode sumber terbuka.

Dalam kode sumber, suatu *method* mewakili suatu aksi yang dilakukan untuk menyelesaikan suatu kebutuhan pengguna. Beberapa *method* dan *class* bekerja sama, satu *method* memanggil *method* lain dengan megirimkan parameter dalam *method invocation*(pemanggilan *method*). Pada penelitian ini diusulkan suatu metode untuk membangun link penelusuran antara kebutuhan dan kode sumber menggunakan pemrosesan bahasa alami pada kode sumber dan kedekatan semantik. Kedekatan semantik dihitung antara dokumen kebutuhan dengan *method signature* dan *method invocation*. Pemrosesan bahasa alami dilakukan pada semua deklarasi *method* dalam kode sumber. Selanjutnya dihitung kedekatan semantik antara dokumen kebutuhan dengan phrase hasil pemrosesan bahasa alami untuk menentukan *method-method* kandidat link penelusuran. Selanjutnya dilakukan penelusuran pada kode sumber untuk menemukan *method-method* yang dieksekusi secara langsung maupun tidak langsung oleh *method* kandidat. Nilai akhir kedekatan semantik dihitung dari kedekatan semantik antara kebutuhan dengan *method* kandidat dan *method-method* yang dieksekusi.

Dengan metode ini diharapkan bisa menemukan bagaimana dan bagian mana dari kode sumber yang menjalankan suatu kebutuhan. Penggunaan metode ini diharapkan bisa meningkatkan keakuratan dalam membangun link penelusuran antara kebutuhan dan kode sumber.

Kata kunci : link penelusuran, kedekatan semantik, pemrosesan bahasa alami.

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Segala puji bagi allah yang telah memberikan rahmatnya sehingga penulis dapat menyelesaikan tesis dengan judul “Pembangunan Link Penelusuran Kebutuhan Fungsional dan Kode Sumber Berdasarkan Kedekatan Semantik dan Struktur Kode Sumber Berorientasi Objek”. Tesis ini disusun untuk memenuhi sebagian persyaratan guna memperoleh gelar Magister Komputer di Institut Teknologi Sepuluh Nopember.

Tesis ini dapat terselesaikan tidak lepas dari bantuan dan dorongan yang sangat berharga dari berbagai pihak. Untuk itu pada kesempatan ini penulis ingin mengucap terima kasih kepada :

1. Ibu Dr. Ir. Siti Rochimah, MT, selaku pembimbing yang telah dengan sabar membimbing dan mengarahkan penulis dalam pengerjaan tesis ini sehingga dapat terselesaikan dengan baik.
2. Keluarga penulis yang selalu memberikan dukungan mental dan spiritual.
3. Teman-teman dosen di Universitas Brawijaya, yang telah sangat banyak memberikan masukkan dalam penelitian ini.
4. Teman-teman Pascasarjana Teknik Informatika ITS, yang telah membantu dan berbagi informasi dengan penulis.

Penulis sangat menyadari bahwa tesis ini masih banyak kekurangan dan masih sangat mungkin untuk dikembangkan menjadi lebih baik, oleh karena itu penulis mengharapkan mesukan dari semua pihak.

Malang, 20 Juni 2016

Penulis

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

LEMBAR PENGESAHAN TESIS..i
ABSTRACT..iii
ABSTRAK..v
KATA PENGANTAR..vii
DAFTAR ISI...ix
DAFTAR TABEL..xi
DAFTAR GAMBAR..xiii
BAB 1 PENDAHULUAN..1
1.1. Latar Belakang..1
1.2. Perumusan Masalah..6
1.3. Tujuan dan Manfaat Penulisan...7
1.4. Batasan Masalah...7
BAB 2 TINJAUAN PUSTAKA..9
2.1. Requirement Traceability(RT)..9
2.2. Penelitian Terkait Link Penelusuran...10
2.2.1. Latent Semantic Analysis(LSA)...10
2.2.2. Latent Dirichlet Allocation(LDA)..11
2.2.3. Sentence-to-Code Traceability Recovery with Domain Ontologies...11
2.2.4. A Semantic Relatedness Approach for Traceability Link Recovery...13
2.2.5. An Ontology-based Approach for Traceability Recovery..................14
2.2.6. Using Natural Language Program Analysis to Find and Understand Action-Oriented Concerns..15
2.2.7. Simpulan dari Kajian Pustaka Tentang Penelitian Sebelumnya.........16
2.3. Ontologi...20
2.4. Pemrosesan Bahasa Alami..21
2.5. Kedekatan Semantik...22
2.6. Metode Evaluasi...25

BAB 3 METODOLOGI PENELITIAN....27
3.1. Tahapan Penelitian....27
3.2. Studi literatur....28
3.3. Perancangan....28
3.4. Pembentukan Ontologi....31
3.5. Populasi Ontologi....32
3.6. Preproses Kode Sumber....34
3.7. Preproses Kebutuhan Fungsional....37
3.8. Pemrosesan Bahasa Alam....39
3.9. Perhitungan Kemiripan Semantik....42
3.10. Analisa Struktur Kode Sumber....44
3.11. Percobaan Awal....47
BAB 4 HASIL PENELITIAN DAN PEMBAHASAN....53
4.1. Pengumpulan Dataset....53
4.2. Skenario pengujian....54
4.3. Pengembangan Plugin Eclipse....60
4.4. Pengujian ITrust....63
4.4.1. Perbandingan .dengan Metode LDA dan LSA....70
4.5. Pengujian Gantt Project....73
4.5.1. Perbandingan.dengan Metode LDA dan LSA....79
BAB 5 KESIMPULAN DAN SARAN....83
5.1. Kesimpulan....83
5.2. Saran....84
DAFTAR PUSTAKA....85
BIOGRAFI PENULIS....87

DAFTAR TABEL

Tabel 2.1. Karakter Dataset Percobaan(A. Mahmoud, 2012)..............................14
Tabel 2.2. Penelitian Terdahulu Terkait Link Penelusuran....................................18
Tabel 2.3. Metode-Metode Kemiripan Semantic(Shridara dkk, 2006).................23
Tabel 2.4. Pembagian Dokumen Evaluasi...26
Tabel 3.1. Jenis Entiti dalam Ontologi..31
Tabel 3.2. Hubungan Antar Objek dalam Ontologi...32
Tabel 3.3. Perhitungan Kemiripan Method Signature JDraw................................49
Tabel 3.4. Perhitungan Kemiripan Method-Method Dibawah ClipPanel.rotate....50
Tabel 3.5. Perhitungan Kemiripan Method-Method Dibawah RotateClipAction.actionPerformed...51
Tabel 3.6. Perhitungan Kemiripan Method-Method Dibawah RotateAction.startAction..52
Tabel 3.7. Hasil Akhir Perhitungan Kemiripan...52
Tabel 4.1. Informasi Kode Sumber..53
Tabel 4.2. Nilai Kemiripan Method-method yang Dipanggil oleh UpdateHospitalListAction.addHospital..56
Tabel 4.3. Nilai Precision Hasil Pengujian Kemiripan Method Signature(Sim) dan Setiap Kebutuhan Fungsional Itrust...65
Tabel 4.4. Nilai Recall Hasil Pengujian Kemiripan Method Signature(Sim) terhadap Setiap Kebutuhan Fungsional Itrust...66
Tabel 4.5. Nilai Precision Hasil Pengujian Kemiripan Penelusuran Pemanggilan Method(SimAst) dan Setiap Kebutuhan Fungsional Itrust...................................69
Tabel 4.6. Nilai Recall Hasil Pengujian Kemiripan antara penelusuran pemanggilan Method(SimAst) dan Setiap Kebutuhan Fungsional Itrust..............70
Tabel 4.7. Nilai Precision Hasil Pengujian Itrust dengan metode LSA dan LDA..71
Tabel 4.8. Nilai Recall Hasil Pengujian Itrust dengan metode LSA dan LDA......71
Tabel 4.9. Nilai Precision Hasil Pengujian Kemiripan Method Signature(Sim) dan

Setiap Kebutuhan Fungsional Gantt Project...74
Tabel 4.10. Nilai Recall Hasil Pengujian Kemiripan Method Signature(Sim) dan Setiap Kebutuhan Fungsional Gantt Project...74
Tabel 4.11. Nilai Precision Hasil Pengujian Kemiripan Penelusuran Pemanggilan Method(SimAst) dan Setiap Kebutuhan Fungsional Gantt Project.......................78
Tabel 4.12. Nilai Recall Hasil Pengujian Kemiripan Penelusuran Pemanggilan Method(SimAst) dan Setiap Kebutuhan Fungsional Gantt Project.......................78
Tabel 4.13. Nilai Precision Hasil Pengujian Gantt Project dengan Metode LSA dan LDA...79
Tabel 4.14. Nilai Recall Hasil Pengujian Gantt Project dengan Metode LSA dan LDA...79

DAFTAR GAMBAR

Gambar 2.1. Pengelompokan Requirement Traceability(Gotel, 1994).................10
Gambar 2.2. Kesamaan antara Metode Invocation dengan Ontologi(Hayashi, 2012)...12
Gambar 2.3. Metode Pembangunan Link Penelusuran Kalimat dan Kode Sumber Menggunakan Domain Ontologi(Hayashi, 2012)..12
Gambar 2.4. Metode Pembangunan Link Penelusuran Berbasis Ontologi(Zhang, 2006)...15
Gambar 2.5. Metode Penyusunan Pasangan V-DO(D Sheperd, 2007).................16
Gambar 2.6. Ilustrasi Kemiripan Kata (Wu dan Palmer, 1994)............................23
Gambar 3.1. Metodologi Penelitian...27
Gambar 3.2. Diagram Alir Pembangunan Link Penelusuran................................29
Gambar 3.3. Struktur Entiti dalam Ontologi..31
Gambar 3.4. Jenis-jenis Relasi dalam Ontologi...32
Gambar 3.5. Diagram Alir Populasi Ontologi dari Kode Sumber.........................33
Gambar 3.6. Kode Sumber Proses Parsing Identifier...35
Gambar 3.7. Diagram Alir Preproses Method Signature......................................36
Gambar 3.8. Diagram Alir Preproses Kebutuhan Fungsional...............................38
Gambar 3.9. Pembobotan Call Graph..46
Gambar 3.10. Ontologi Struktur Kode Sumber JDraw...48
Gambar 3.11. Ontologi Struktur Method RotateClipAction.actionPerformed……50
Gambar 3.12. Ontologi Struktur Method RotateAction.startAction......................51
Gambar 4.1. Alur Skenario Pengujian...55
Gambar 4.2. Kode Sumber Penelusuran Method Invocation Dengan Breadth First Search...59
Gambar 4.3. Kode Sumber Penelusuran Deklarasi Method.................................62
Gambar 4.4. Kode Sumber Penelusuran Pemanggilan Method............................63
Gambar 4.5. Struktur add, get dan edit dalam Wordnet..67

Gambar 4.6. Grafik Perbandingan Nilai Precision Itrust......................................72
Gambar 4.7. Grafik Perbandingan Nilai Recall Itrust...72
Gambar 4.8. Kurva Interpolasi Precision dan Recall Itrust..................................73
Gambar 4.9. Grafik Perbandingan Nilai Precision Gantt Project..........................80
Gambar 4.10. Grafik Perbandingan Nilai Recall Gantt Project.............................80
Gambar 4.11. Kurva Interpolasi Precision dan Recall Gantt Project....................81

BAB 1
PENDAHULUAN

1.1. Latar Belakang

Penelusuran kebutuhan adalah kemampuan untuk menjelaskan dan mengikuti jalannya suatu kebutuhan dalam arah maju maupun mundur (Gotel, 1994). Link penelusuran membantu pengembang untuk mengerti hubungan antara dokumen-dokumen yang dihasilkan selama proses pengembangan perangkat lunak. Link penelusuran antara dokumen kebutuhan dan kode sumber sangat membantu dalam proses pengembangan dan pemeliharaan perangkat lunak. Dalam proses pengembangan dan pemeliharaan perangkat lunak, pengembang melakukan perubahan pada kode sumber akan tetapi sering melupakan untuk memperbarui dokumen-dokumen yang menyertai kode sumber untuk mempersingkat waktu pengembangan karena dituntut oleh tenggat waktu yang singkat(Ali, 2011). Selama proses pengembangan, link penelusuran antara artifak-artifak perangkat lunak bisa menjadi tidak valid karena tidak diperbarui sesuai dengan perubahan yang terjadi pada kode sumber perangkat lunak(Hayasi, 2010). Karena itu diperlukan suatu metode untuk membangun link penelusuran kebutuhan yang cepat dan akurat.

Link penelusuran kebutuhan dan kode sumber juga berguna untuk memastikan bahwa semua kebutuhan telah dipenuhi oleh perangkat lunak dan perangkat lunak konsisten dengan kebutuhan, tidak ada fungsi lain diluar kebutuhan yang diterapkan oleh perangkat lunak. Dengan adanya link penelusuran antara kebutuhan dan kode sumber, pengembang memiliki informasi yang dapat membantu untuk mengetahui bagaimana suatu kebutuhan diterapkan dalam perangkat lunak dan mengetahui apakah semua kebutuhan fungsional sudah diimplementasikan dalam kode sumber. Link penelusuran meningkatkan efisiensi pengembangan perangkat lunak. Selama proses pengembangan dan pemeliharaan

perangkat lunak bisa terjadi perubahan pada kebutuhan pengguna maupun kode sumber. Adanya link penelusuran antara dokumen kebutuhan dengan kode sumber akan meningkatkan kecepatan menemukan bagian mana dari kode sumber yang perlu diubah ketika ada perubahan kebutuhan dari pengguna. Untuk melakukan perubahan pada perangkat lunak, seorang pengembang harus mengerti kode sumber yang akan diubah. Link antara dokumen kebutuhan dengan kode sumber sangat membantu pengembang dalam memahami kode sumber. Pengembang lebih cepat memahami bagaimana suatu kebutuhan diimplementasikan tanpa perlu membaca tiap baris kode sumber yang memakan waktu lama.

Link penelusuran juga berguna jika pengembang menggunakan perangkat lunak yang tidak memiliki dokumen perancangan yang lengkap seperti pada kode sumber terbuka. Dalam proses pengembangan perangkat lunak kadang diperlukan perangkat lunak sumber terbuka. Banyak perangkat lunak sumber terbuka tidak menyertakan dokumen perancangan secara lengkap. Hal ini menimbulkan kesulitan dalam memahami perangkat lunak tersebut. Perangkat lunak sumber terbuka hanya menyertakan kode sumber dan penjelasan fitur dan fungsi yang mampu dikerjakannya. Demikian juga pada proses pengembangan perangkat lunak menggunakan metode *agile,* tidak menyertakan dokumen pengembangan yang detail. Peran link penelusuran antara kebutuhan dan kode sumber sangat bermanfaat dalam kondisi ini.

Beberapa metode penemuan kembali informasi(*information retrieval*/IR) digunakan untuk membangun link penelusuran antara dokumen kebutuhan dengan kode sumber, antara lain VSM(*vector space model*)(Antoniol, 2000), LSI(*latent semantic index*)(Marcus dan Meletic, 2003), JS(*Jensen Shannon*)(Abadi, 2008) dan LDA(*Latent Dirichlet Allocation*)(Oliveto, 2010). Pendekatan IR menganggap dokumen kebutuhan dan kode sumber adalah sekumpulan teks tanpa memperhitungkan struktur kode sumber. Kemudian dihitung kemiripan tekstual antara kode sumber dengan dokumen kebutuhan, semakin tinggi kemiripan tekstual kemungkian besar kedua artifak mengandung konsep yang sama dan dianggap saling berhubungan. Efektifitas dari metode IR yang digunakan,

dievaluasi berdasarkan nilai *precision* dan *recall*. Ali(2012) melakukan penelitian dengan membandingkan penemuan link penelusuran menggunakan metode IR pada beberapa studi kasus dan mendapatkan nilai *precision* dan *recall* yang rendah yang berarti banyak link yang kurang relevan yang harus divalidasi oleh manusia. Nilai *precision* dan *recall* dipengaruhi antara lain oleh dataset yang digunakan. Dalam hal ini adalah kualitas dan besarnya kode sumber. IR membentuk korpus dari matrik *term*/kata dalam kode sumber. Semakin besar korpus yang digunakan semakin baik dan semakin besar kemungkinan beberapa *term* muncul dalam suatu dokumen. Tetapi jika korpus yang dibentuk kecil dan *term* tersebar dibanyak dokumen sehingga kemungkinan suatu *term* muncul kembali dalam satu dokumen kecil, maka akan megakibatkan rendahnya nilai *precision* dan *recall*. Dari hasil penelusuran kebutuhan dengan IR diperoleh nilai *precision* didapat sebesar 5% sampai 16% pada saat nilai *recall* 90% - 100%. Rendahnya nilai *precision* dan tingginya nilai *recall* ini menunjukkan bahwa hasil yang diperoleh mengandung sangat sedikit link *true positif* dari keseluruhan link yang dihasilkan. Hasil penelitian dengan membandingkan empat metode IR tersebut diperoleh kesimpulan bahwa nilai *recall* yang tinggi didapatkan pada saat nilai *precision* yang rendah dan sebaliknya. Ini menunjukkan masih banyak link tidak relevan yang dihasilkan dari metode IR. Penelitian lanjutan dilakukan untuk memperbaiki hasil yang dicapai metode IR dengan menambahkan sumber informasi dan menggabungkannya dengan beberapa metode penggalian informasi. Penelitian yang dilakukan Eaddy(2008) menggabungkan IR dengan jejak eksekusi untuk meningkatkan akurasi. LSI diaplikasikan pada kode sumber dan dokumen kebutuhan. Penelitian Eaddy memerlukan eksekusi aplikasi untuk mendapatkan jejak eksekusi.

Penelitian menggunakan analisa *natural language processing(*NLP) pada kode sumber dilakukan oleh David Sheperd(2006) untuk menemukan pasangan kata kerja(*verb*) dan kata benda(*direct object*) dalam deklarasi *method*. Pasangan kata kerja dan kata benda, V-DO<*verb*, *Direct Object*> dan lokasinya dalam kode sumber disimpan dalam suatu daftar. Ketika manusia memasukkan queri berupa

kata kerja, aplikasi akan memberikan saran berupa pilihan kata benda berdasar pasangan V-DO yang sudah dibangun. Metode yang diusulkan bersifat interaktif, memerlukan manusia untuk menentukan jalur yang dipilih dalam menelusuri kode sumber. Aplikasi hanya memberikan pilihan/saran pada manusia untuk melengkapi queri untuk pencarian pada kode sumber. Penelitian yang dilakukan A. Mahmoud(2012) menambahkan eksternal dokumen yaitu Wikipedia sebagai referensi untuk menghitung nilai kedekatan semantik antar kata. Pencarian dilakukan dengan menghitung bobot kedekatan tiap kata dari dokumen kebutuhan dengan kata dalam kode sumber. Bobot kedekatan dihitung berdasar hubungan semantik antar kata dalam matriks wikipedia. Penelitian ini hanya mencari kesamaan dua dokemen berdasar kata tanpa memperhitungkan struktur dari kode sumber. Seperti yang dilakukan oleh pencarian link penelusuran dengan metode IR, hanya menganggap kode sumber sebagai kumpulan kata. Tidak mampu menangani struktur kata(*morphology*) dan struktur kode sumber. Penelitian tersebut menghasilkan nilai *recall* lebih baik daripada LSI, tapi nilai *precision* lebih rendah daripada VSM. Kelemahan ini membuka kemungkinan penelitian lanjutan untuk menggunakan hubungan antar *method*(*method invocation*) dan *class* untuk meningkatkan *precision*.

Penggunaan ontologi untuk menemukan link penelusuran antara dokumen dengan kode sumber dilakukan Zhang(2006). Pada penelitian tersebut dibentuk dua buah ontologi yaitu ontologi dokumen dan ontologi kode sumber. Link dibentuk antara dua ontologi berdasarkan kesamaan kata antara elemen yang ada dalam ontologi dokumen dan ontologi kode sumber. Nama *class* atau *method* yang ada dalam ontologi kode sumber harus ada dalam ontologi dokumen supaya bisa dibentuk link antar kedua elemen ontologi. Penelitian lain dilakukan Hayashi(2010) menggunakan ontologi sebagai domain pengetahuan untuk menentukan bobot hubungan antar *method* dalam kode sumber. Kelemahan metode tersebut adalah perlu dibangun ontologi secara manual untuk setiap domain permasalahan. Akurasi penelitian sangat tergantung dari kelengkapan dan validasi ontologi yang digunakan. Kelemahan ini membuka peluang untuk

penelitian perbaikan untuk menghilangkan ketergantungan terhadap ontologi atau menggunakan domain pengetahuan yang sudah ada.

Suatu kebutuhan perangkat lunak menggambarkan aspek yang harus dilakukan perangkat lunak dan batasan perangkat lunak dengan tujuan untuk menyelesaikan masalah yang dimiliki pengguna (Timothy C, 2005). Kebutuhan tidak memiliki hubungan langsung dengan kode sumber. Kebutuhan berbentuk bahasa alami yang tidak mengandung bagian dari kode sumber. Berbeda dengan dokumen perancangan atau dokumen manual suatu perangkat lunak yang bisa mengandung nama *class*, *method* atau variabel dalam kode sumber. Dalam pembuatan kode sumber terkadang pengembang menggunakan kata yang berbeda dari dokumen kebutuhan untuk mengungkapkan sesuatu yang sama. Dalam penelitian ini akan dilakukan *parsing* bahasa alami dari *method signature* dan *method body*. Selanjutnya dihitung kedekatan semantik antara konsep yang dihasilkan dengan kalimat dalam dokumen kebutuhan. Penggunaan pemrosesan bahasa alam(*Natural Language Proceessing/*NLP) dan kedekatan semantik diharapkan bisa menemukan hubungan antara dua konsep yang berbeda berdasarkan suatu domain pengetahuan. Metode kedekatan semantik berusaha mengkuantifikasi jarak antara dua konsep dalam suatu domain pengetahuan. Salah satu contoh domain pengetahuan yang digunakan dalam penelitian ini adalah WordNet(Fellbaum, 2005). Sedangkan untuk menentukan kedekatan semantik antar kata dapat menggunakan rumus yang diusulkan oleh Wu-Palmer(1994). Wu-Palmer menilai kedekatan dua kata menggunakan relasi dan kelompok yang sama(least common subsumer, LCS) dalam WordNet.

Dalam pemrograman berbasis objek, suatu fitur dikerjakan oleh beberapa objek yang bekerja sama. Pemrograman berbasis objek merupakan suatu pendekatan dalam menyelesaikan masalah dimana semua proses komputasi dilakukan oleh kumpulan objek. Suatu program dapat dilihat sebagai kumpulan objek yang berkolaborasi untuk menyelesaikan suatu pekerjaan(Nair, 2009). Dalam bahasa pemrograman, suatu objek merupakan instansiasi suatu *class*. Suatu *class* memiliki atribut dan *method*. *Class* memanggil *method* dalam *class* lain dan

mnegirimkan data dalam parameter *method invocation*(pemanggilan *method)*. Beberapa *class* bekerja sama untuk menyelesaikan suatu kebutuhan dan satu *class* bisa berpartisipasi dalam menyelesaikan beberapa kebutuhan.

Dalam penelitian ini diusulkan langkah kerja yang terdiri dari tiga langkah utama untuk menemukan link penelusuran antara kode sumber dan kebutuhan. Lagkah pertama adalah pembentukan ontologi kode sumber. Ontologi digunakan karena bisa menyimpan hubungan antara *method* dan *class* dalam kode sumber. Penggunaan *inference engine* dan *rule* pada ontologi bisa menghasilkan individu-individu untuk melengkapi ontologi seperti *inheritance*/turunan pada kode sumber. Untuk mengisi populasi dalam ontologi digunakan *abstract syntax tree*(AST) dari kode sumber dan poses pengurai(*parsing)* untuk mengidentifikasi entiti dan hubungan antar entiti. Langkah kedua adalah melakukan ekstraksi kumpulan kata dengan pemrosesan bahasa alami pada *method signature.* Selanjutnya menghitung kedekatan semantik antara kata tersebut dengan dokumen kebutuhan untuk menemukan kandidat link penelusuran. Pada langkah kedua akan dihasilkan beberapa *method* dengan nilai kedekatan semantik tinggi yang akan menjadi *method* utama. Langkah ketiga, melakukan penilaian kedekatan semantik antara *method-method* yang berhubungan dengan *method* utama terhadap kebutuhan. Semakin tinggi nilai kedekatan semantik *method* yang dipanggil dengan kebutuhan maka semakin besar bobot kandidat tersebut untuk menjadi link penelusuran. Wordnet digunakan sebagai domain pengetahuan untuk menentukan kedekatan semantik antara konsep dari kebutuhan dan kode sumber. Dari penelitian ini diharapkan akan menghasilkan link antara kebutuhan dengan *method* yang mengimplementasikan kebutuhan tersebut.

1.2. Perumusan Masalah

Perumusan masalah pada penelitian ini meliputi beberapa hal sebagai berikut :

a. Bagaimana membentuk link penelusuran antara kebutuhan fungsional dan kode sumber melalui kedekatan semantik menggunakan ontologi sebagai

domain pengetahuan.

b. Bagaimana meningkatkan akurasi link penelusuran antara kebutuhan fungsional dan kode sumber menggunakan struktur suatu kode sumber.
c. Berapa besar peningkatan akurasi yang diperoleh dengan menambahkan analisa terhadap hubungan antar *method* dalam kode sumber.

1.3. Tujuan Dan Manfaat Penulisan

Tujuan penelitian yang ingin dicapai adalah :

- Meningkatkan akurasi proses penemuan link penelusuran menggunakan hubungan antar *method*.

Manfaat penelitian ini adalah meningkatkan efisiensi proses pemeliharaan dan pengembangan perangkat lunak dengan mempermudah pengembang menemukan bagian mana dari kode sumber yang mengerjakan setiap fitur dari suatu aplikasi berdasarkan kedekatan hubungan semantik. Kontribusi dalam penelitian ini adalah proses penemuan link penelusuran antara dokumen dengan kode sumber berdasarkan kedekatan semantik menggunakan WordNet sebagai domain pengetahuan dan peningkatan akurasi menggunakan analisa hubungan antar *method*.

1.4. Batasan Masalah

Dalam penelitian ini, pembahasan yang dicakup adalah sebagai berikut :

1. Dokumen kebutuhan yang digunakan dalam penelitian ini adalah dokumen yang berbahasa inggris.
2. Dokumen kebutuhan dan kode sumber yang akan diproses hanya berbentuk teks.
3. Kode sumber menggunakan bahasa pemrograman Java.

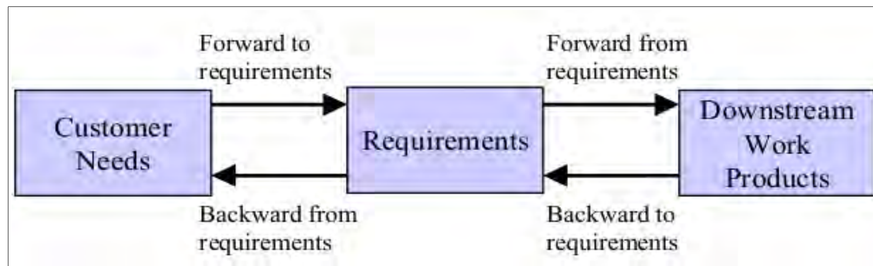
[Halaman ini sengaja dikosongkan]

BAB 2
TINJAUAN PUSTAKA

2.1. Requirement Traceability(RT)

Dalam rekayasa kebutuhan, penelusuran didefinisikan sebagai kemampuan untuk menelusuri atau mengikuti jejak dari dan ke kebutuhan. RT merupakan bagian penting dari manajemen kebutuhan. Gotel dan Finkelstein (1994) membuat definisi RT sebagai kemampuan untuk menjelaskan dan mengikuti daur hidup suatu kebutuhan dalam dua arah, maju dan mundur dari dan ke kebutuhan. Pengelompokan RT ditunjukkan pada Gambar 2.1. RT dikelompokkan menjadi empat yaitu:

1. *Forward from requirement*/maju dari kebutuhan : Penelusuran jejak dari kebutuhan menuju artefak teknis seperti kode sumber. Jejak ini diperlukan untuk mengevaluasi perubahan yang harus dilakukan jika ada perubahan kebutuhan, digunakan untuk menganalisa dampak perubahan .
2. *Backward to requirement*/mundur menuju kebutuhan : Penelusuran jejak dari artefak teknis menuju kebutuhan. Jejak ini digunakan untuk memastikan bahwa semua kode sumber yang dibuat sesuai dengan kebutuhan, tidak ada kode sumber yang dibuat tapi tidak berguna.
3. *Forward to requirement*/maju menuju kebutuhan : Penelusuran jejak dari dokumen tingkat yang lebih tinggi menuju kebutuhan yang menjadi turunanya. Jika pengguna mengubah tujuan suatu sistem, jejak ini bisa dugunakan untuk menentukan semua spesifikasi kebutuhan yang berubah dan dokumen teknis lainya.
4. *Backward from requirement*/mundur dari kebutuhan : Penelusuran jejak dari spesifikasi kebutuhan menuju dokumen tingkat yang lebih tinggi. Jejak ini penting untuk mengevaluasi kualitas spesifikasi kebutuhan.



Gambar 2.1. Pengelompokan *Requirement Traceability*(Gotel, 1994)

2.2. Penelitian Terkait Link Penelusuran

Banyak cara yang digunakan untuk pembangunan link penelusuran antara dokumen dengan kode sumber. Metode pembangunan link penelusuran dapat dilakukan menggunakan metode penemuan kembali informasi(Information Retreival/IR). Menggunakan metode IR menghasilkan *precision* dan *recall* yang rendah. Beberapa metode lain digunakan untuk memperoleh akurasi yang lebih baik, antara lain penggunaan pemrosesan bahasa alami(*natural language processing*/NLP) dan domain pengetahuan dengan memanfaatkan ontologi.

2.2.1. Latent Semantic Analysis(LSA)

Penelitian link penelusuran antara dokumen dan kode sumber menggunakan metode IR berbasis LSI dilakukan oleh Marcus dan Meletic(2003). Penelitian ini menggunakan dua studi kasus perangkat lunak yaitu LEDA dan Albergate. Hasil penelitian pada studi kasus LEDA, hasil pengujian mendapatkan nilai *precision* sebesar 0.77 pada *recall* 0.59 sampai nilai *precision* 0.11 pada *recall* 1. Pada kasus Albergate, hasil pengujian mendapatkan nilai *precision* sebesar 0.44 pada *recall* 0.45 sampai nilai *precision* 0.16 pada *recall* 1. Klasifikasi dan penghitungan kemiripan dilakukan terhadap dokumentasi dan *class* dalam kode sumber. Pada kasus Albergate nilai *precision* lebih rendah karena dokumen yang digunakan dokumen kebutuhan yang sangat pendek dan tidak ada komentar dalam kode sumber. Semakin besar dokumen yang digunakan dalam dataset, semakin baik proses LSI dalam menentukan hubungan antar term dan dokumen. Semakin besar *corpus* semakin baik hasilnya. Dalam penelitian ini direkomendasikan

menambahkan analisa struktural dan semantik terhadap kode sumber dan dokumentasi untuk memperbaiki hasil pengujian.

2.2.2. Latent Dirichlet Allocation(LDA)

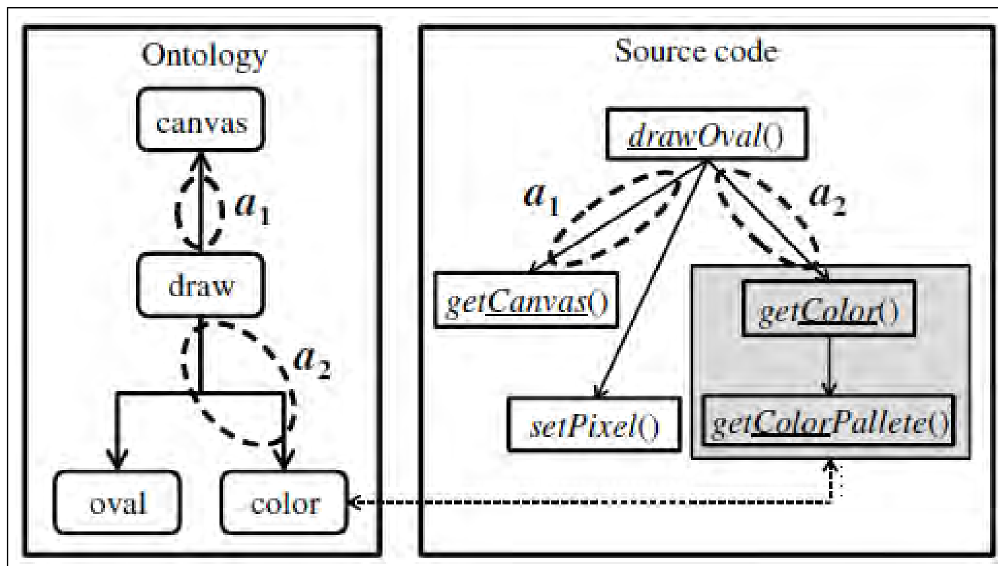
Penelitian tentang pembangunan link penelusuran antara dokumen dan kode sumber menggunakan metode IR berbasis LDA dilakukan oleh Oliveto dkk(2010). Penelitian ini menggunakan dua studi kasus perangkat lunak dengan dataset sebagai berikut :

1. EasyClinic terdiri dari 15000 LOC, 47 kelas, dan 30 use case.
2. eTour,terdiri dari 45000 LOC, 116 kelas, dan 58 use case.

Penelitian ini membandingkan dua studi kasus tersebut antara metode LDA dengan VSM, LSI dan JS. Hasilnya diperoleh bahwa nilai precision dan recall dari LDA tidak lebih lebih baik dari tiga metode IR tersebut. Tetapi pada kasus eTour, LDA berhasil menemukan link benar yang tidak ditemukan oleh metode IR lainnya sebesar 10%.

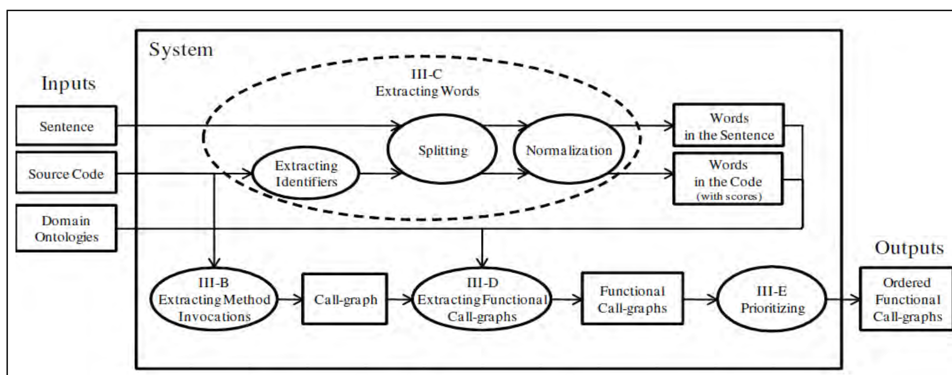
2.2.3. Sentence-to-Code Traceability Recovery With Domain Ontologies

Penelitian dilakukan oleh Shinpei Hayashi dkk(2010), pembangunna link penelusuran dilakukan dalam dua tahap. Langkah pertama mencari metode utama, metode yang mengandung kata yang sama dengan kebutuhan. Langkah kedua melakukan penelusuran terhadap metode-metode yang berhubungan dengan metode utama. Metode invocation dipilih berdasarkan nama metode yang mengandung kata yang sama dengan kebutuhan atau hubungan antar metode sama dengan hubungan kata dalam ontologi. Kesamaan antara ontologi dan pemanggilan metode ditunjukkan pada Gambar 2.2.



Gambar 2.2. Kesamaan antara *Metode Invocation* dengan Ontologi(Hayashi, 2012)

Preproses dilakukan terhadap kalimat kebutuhan dan kode sumber. Langkah preproses adalah pemisahan nama metode, kelas dan variabel menjadi kata, kemudian melakukan normalisasi kata menjadi kata dasar. Keseluruhan proses penelitian ditunjukkan pada Gambar 2.3. Studi kasus penelitian dilakukan menggunakan Jdraw, perangkat lunak untuk mengedit gambar, yang terdiri dari 7 kebutuhan, 1450 metode



Gambar 2.3. Metode Pembangunan Link Penelusuran Kalimat dan Kode Sumber Menggunakan Domain Ontologi(Hayashi, 2012)

Hasil dari studi kasus menunjukkan bahwa penggunaan ontologi meningkatkan nilai *recall*, tapi *precision* menurun. Beberapa kasus menghasilkan

nilai *recall* lebih dari 90%. Kelemahan dari metode ini adalah harus dibangun ontologi secara manual untuk setiap domain pengetahuan. Akurasi dari penelitian ini sangat tergantung dari kelengkapan dan validasi ontologi yang digunakan sebagai referensi dalam mencari hubungan antar metode dalam kelas kode sumber. Metode penelitian ini tidak menggunakan gabungan kata/phrasa yang menyebabkan kegagalan dalam menemukan kesamaan kata dalam beberapa kasus.

2.2.4. A Semantic Relatedness Approach For Traceability Link Recovery

Penelitian tentang pembangunan link penelusuran antar dokumen dilakukan Anas Mahmoud dkk(2012) dengan menggunakan bantuan eksternal dokumen yaitu Wikipedia. Dalam penelitian ini digunakan hasil *dump* Wikipedia 2009 sebagai dokumen eksternal untuk menghitung bobot hubungan semantik antar kata. Preproses dilakukan pada *dump* wikipedia untuk menghasilkan kata dasar kemudian disusun matrik *term* dokumen dengan kolom merupakan artikel dalam wikipedia dan kata sebagai baris. Pembobotan kata menggunakan tf*idf. Setiap kata dalam matriks merupakan vector space. Hubungan semantik antar kata dihitung menggunakan *cosine similarity* dari *vector space* antar term dalam matriks term dokumen. Langkah berikutnya dilakukan preproses untuk menghasilkan term/kata dalam dokumen sumber dan target. Pencarian dilakukan dengan menghitung bobot kedekatan tiap kata dari dokumen sumber dengan kata dalam dokumen tujuan. Bobot kedekatan dihitung berdasar hubungan semantik antar kata dalam matriks wikipedia. Karakter dataset yang digunakan dalam studi kasus penelitian ini ditunjukkan pada Tabel 2.1. Pada studi kasus CM1, hubungan penelusuran dibentuk antara dokumen kebutuhan dengan dokumen detil kebutuhan perangkat lunak.

Tabel 2.1. Karakter Dataset Percobaan(A. Mahmoud, 2012)

	Kode Sumber	Komentar	sumber	target
iTrust	18,3 KLOC	6,3 K	UC	SC
eTour	17,5 KLOC	7,5 K	UC	SC
CM1	20 KLOC	-	Req.	Req.

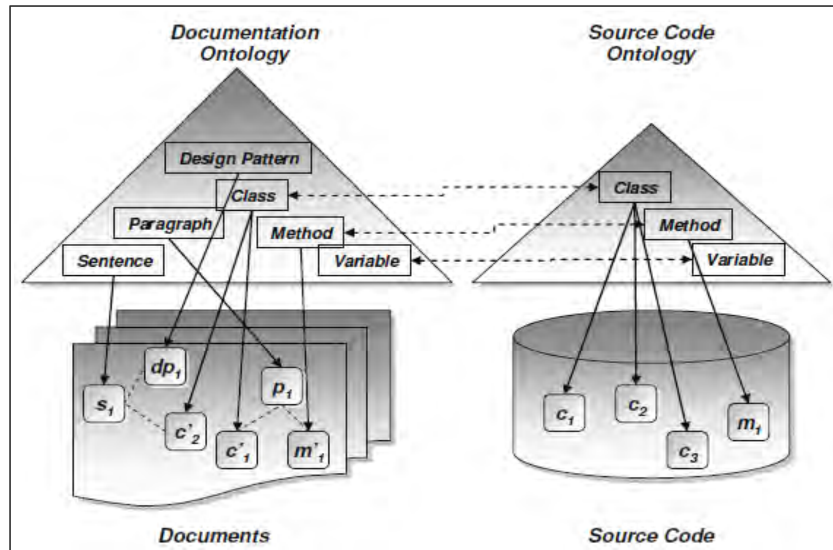
Dari hasil percobaan menggunakan tiga studi kasus, penggunaan hubungan semantik menghasilkan nilai *recall* lebih baik daripada VSM dan LSI. Pada kasus pencarian hubungan antar dokumen, bukan kode sumber, menghasilkan nilai *precision* dan *recall* lebih baik daripada VSM dan LSI. Kelemahan dari metode ini adalah hubungan hanya berdasar nilai kedekatan antar kata. Metode yang diusulkan tidak khusus mencari hubungan antara kode sumber dan kebutuhan, hanya mencari kesamaan dua dokemen berdasar kata. Tidak mampu menangani struktur kata(morphology) dan struktur kode sumber. Dari studi kasus dihasilkan nilai *recall* lebih baik, tapi *precision* lebih rendah daripada VSM.

2.2.5. An Ontology-based Approach For Traceability Recovery

Penelitian dilakukan oleh Y. Zhang dkk(2006) untuk membangun link penelusuran antara dokumen dengan kode sumber menggunakan ontologi aligment. Preproses untuk menghasilkan term/kata dalam kode sumber dan dokumen. Dibangun dua ontologi, ontologi kode sumber dan ontologi dokumen. Ontologi kode sumber juga memuat relasi antar kelas dan antar metode. Link dibangun berdasar adanya kesamaan kata nama kelas, metode dan variabel dalam kedua ontologi menggunakan ontologi *aligment*. Menggunakan Query pada ontologi untuk menemukan hasil link antar elemen ontologi. Metode pembangunan link dengan ontologi ditunjukkan pada Gambar 2.4.

Penelitian ini menggunakan dataset uDig, sebuah perangkat lunak *Geographic Information System* (GIS) sumber terbuka. Dataset dari penelitian ini adalah kode sumber dan dokumentasi perangkat lunak yang terdiri dari JavaDoc aplikasi dan dokumen analisa kebutuhan. Hasil studi kasus pada uDig, metode yang diusulkan bisa menemukan link antara kalimat dalam dokumen dengan bagian dari kode

sumber. Kekurangan dari metode ini adalah link dibangun antara kode sumber dengan dokumentasi perancangan. Diperlukan dokumentasi perancangan untuk membentuk link penelusuran. Link dibangun berdasarkan adanya kesamaan nama kelas, metode atau variabel dalam ontologi dokumen dan ontologi kode sumber.

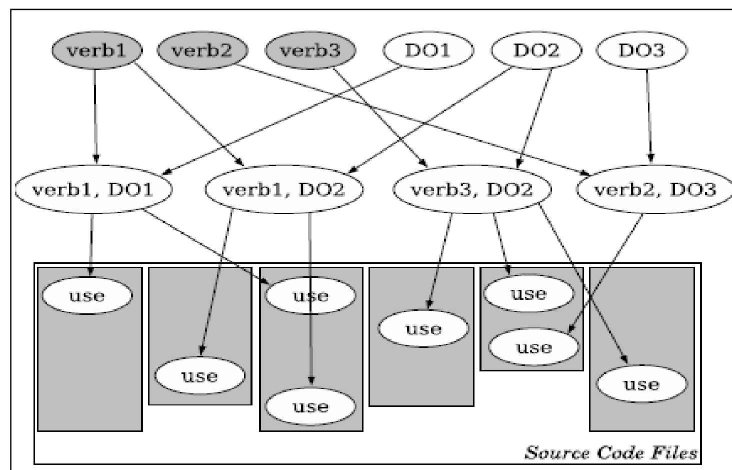


Gambar 2.4. Metode Pembangunan Link Penelusuran Berbasis Ontologi(Zhang, 2006)

2.2.6. Using Natural Language Program Analysis To Find And Understand Action-Oriented Concerns

Penelitian dilakukan oleh David Shepherd dkk(2007) menggunakan Natural Language Processing(NLP). Metode yang digunakan bersifat interaktif, membutuhkan campur tangan manusia untuk menelusuri kode sumber. Langkah pertama adalah melakukan preproses pada kode sumber untuk menghasilkan term/kata dasar. NLP digunakan untuk menemukan verb/kata kerja dan noun/kata benda dalam kode sumber yang saling berhubungan. Pasangan kata kerja dan kata benda, V-DO<verb/kata kerja, Direct Object/kata benda> dan lokasinya dalam kode sumber disimpan dalam suatu daftar. Proses pembuatan pasangan V-DO ditunjukkan pada Gambar 2.5. Ketika manusia memasukkan queri berupa kata kerja, aplikasi akan memberikan saran berupa pilihan kata benda berdasar pasangan V-DO yang sudah dibangun dari kode sumber. Studi kasus penelitian ini

menggunakan dataset jEdit, terdiri dari 83 KLOC, 4380 metode dan FreeMind, terdiri dari 70 KLOC, 5170 metode. Hasil studi kasus, nilai rata-rata *recall* 87% dan rata-rata *precision* 78%. Kekurangan dari metode yang diusulkan dalam penelitian ini adalah bersifat interaktif, memerlukan manusia untuk menentukan jalur yang dipilih dalam menelusuri kode sumber. Aplikasi hanya memberikan pilihan/saran pada manusia untuk melengkapi queri untuk pencarian.



Gambar 2.5. Metode Penyusunan Pasangan V-DO(D Sheperd, 2007)

2.2.7. Simpulan Dari Kajian Pustaka Tentang Penelitian Sebelumnya

Pembangunan link penelusuran antara dokumen dengan kode sumber dimulai dengan menggunakan metode IR dan menghasilkan nilai *precision* dan *recall* yang rendah. Keberhasilan NLP dalam melakukan pemrosesan teks diharapkan bisa dimanfaatkan dalam membangun link penelusuran antara kode sumber dan dokumen kebutuhan. Beberapa penelitian berusaha memanfaatkan NLP dan ontologi untuk membangun link antar kode sumber dan dokumen. Rangkuman tentang beberapa penelitian link penelusuran menggunakan NLP atau ontologi ditunjukkan pada Tabel 2.2. Berdasarkan hasil kajian pustaka pada beberapa metode pembangunan link penelusuran ditemukan beberapa kekurangan dari masing-masing penelitian sebelumnya antara lain pada penelitian penggunaan ontologi oleh Hayashi(2010), harus dibangun ontologi secara manual untuk setiap domain pengetahuan. Akurasi dari penelitian tersebut sangat tergantung dari

kelengkapan dan validitas ontologi yang digunakan sebagai referensi dalam mencari hubungan antar metode dalam kelas kode sumber. Metode penelitian tersebut mengalami kegagalan dalam menemukan kesamaan kata dalam beberapa kasus. Penelitian menggunakan kedekatan semantik yang dilakukan A. Mahmoud(2012) menghasilkan nilai *precision* yang kurang baik. Kekurangan-kekurangan ini membuka peluang untuk penelitian perbaikan untuk menghilangkan ketergantungan terhadap ontologi menggunakan domain pengetahuan yang sudah ada untuk menhitung kedekatan semantik. Untuk meningkatkan *precision* bisa digunakan penelusuran hubungan pemanggilan metode(*method invocation*).

Dalam penelitian ini digunakan kedekatan semantik antara konsep yang dihasilkan dari preproses dari kode sumber dan kebutuhan fungsional suatu perangkat lunak untuk menemukan link penelusuran antara kode sumber dan kebutuhan fungsional. Diusulkan suatu metode untuk menggali konsep-konsep dalam kode sumber menggunakan NLP dan mencari hubungan dengan kebutuhan fungsional berdasarkan kedekatan semantik menggunakan domain pengetahuan WordNet. Untuk meningkatkan akurasinya digunakan analisa penelusuran pemanggilan metode(*method invocation*) dalam kode sumber. Link penelusuran yang dihasilkan akan menghubungkan antara kebutuhan fungsional dan metode apa yang menyelesaikan fungsi tersebut.

Tabel 2.2. Penelitian Terdahulu Terkait Link Penelusuran.

Judul	Sentence-to-Code Traceability Recovery with Domain Ontologies	A Semantic Relatedness Approach for Traceability Link Recovery
Tahun	2010	2012
Pengarang	Shinpei Hayashi, Takashi Yoshikawa, Motoshi Saeki	Anas Mahmoud, Nan Niu, Songhua Xu
Metode	• Preproses untuk menghasilkan term/kata dalam kode sumber dan kebutuhan. • Hubungan antara metode dan kebutuhan berdasar adanya kata yang sama. • Metode yang relevan dengan metode utama ditentukan bedasar kesamaan kata antara metode dengan individu dalam ontologi.	• Preproses untuk menghasilkan term/kata dalam kode sumber dan kebutuhan • Menyusun *vector space* dari Wikipedia untuk menghitung kesamaan antar kata. • Mencari link antara dokumen berdasar jumlah bobot kedekatan kata dalam dua dokumen sesuai dengan vector space dari wikipedia.
Hasil	• Meningkatkan nilai recall, tapi precision menurun. • Beberapa kasus menghasilkan nilai recall lebih dari 90%	• Menighasilkan nilai recall lebih baik dari VSM dan LSI. • Pada pencarian hubungan antar dokumen, bukan kode sumber, menghasilkan nilai precision dan recall lebih baik.
Dataset	• JDraw, 7 kebutuhan, 1450 metode	• eTour, 17,5 KLOC, 394 link kebutuhan dan kode sumber • iTrust, 18,3 KLOC, 314 link kebutuhan dan kode sumber • CM-1, 235 kebutuhan, 220 elemen perancangan, 361 link antar dokumen
Kelemahan	• Perlu dibangun ontologi secara manual untuk setiap domain permasalahan. • Tingkat akurasi tergantung dari kelengkapan dan validitas ontologi. • Pada beberapa kasus, nilai precision dan recall masih rendah karena kurang lengkapnya ontologi. • Nilai recall lebih baik tapi precision lebih rendah	• Hubungan hanya berdasar nilai kedekatan antar kata. • Tidak khusus mencari hubungan antara kode sumber dan kebutuhan, hanya mencari kesamaan dua dokemen berdasar kata. • Tidak mampu menangani struktur kata(morphology) dan struktur kode sumber. • Nilai recall lebih baik, tapi precision lebih rendah dari VSM

Judul	An Ontology-based Approach for Traceability Recovery	Using Natural Language Program Analysis to Find and Understand Action-Oriented Concerns
Tahun	2006	2007
Pengarang	Yonggang Zhang, René Witte, J. Rilling, V. Haarslev	David Shepherd, Lori Pollock, and K. Vijay-Shanker
Metode	• Preproses untuk menghasilkan term/kata dalam kode sumber dan kebutuhan. • Dibangun dua ontologi, ontologi kode sumber dan ontologi dokumen. • Link dibangun berdasar adanya kesamaan kata nama kelas, metode dan variabel dalam kedua ontologi menggunakan ontologi *aligment*. • Menggunakan Query dalam ontologi untuk menemukan hasil link antar ontologi.	• Preproses untuk menghasilkan term/kata dalam kode sumber. • NLP digunakan untuk menemukan verb/kata kerja dan noun/kata benda dalam kode sumber. • Menyimpan pasangan <verb/kata kerja, Direct Object/kata benda> dan lokasinya dalam kode sumber. • Manusia memasukkan query berupa kata kerja, kemudian aplikasi akan memberikan saran berupa pilihan kata benda berdasar pasangan V-DO yang sudah dibangun dari kode sumber.
Hasil	• Menemukan link antara kalimat dalam dokumen dengan bagian dari kode sumber.	• Menghasilkan nilai rata-rata recall 87% dan rata-rata precision 78%.
Dataset	• uDig, Geographic Information System (GIS) sumber terbuka	• jEdit, 83 KLOC, 4380 metode • FreeMind, 70 KLOC, 5170 metode
Kelemahan	• Link dibangun antara kode sumber dengan dokumentasi perancangan. • Link dibangun berdasarkan adanya kesamaan nama kelas, metode atau variabel dalam ontologi dokumen.	• Bersifat interaktif, memerlukan manusia untuk menentukan jalur yang dipilih dalam menelusuri kode sumber. • aplikasi hanya memberikan pilihan/saran pada manusia untuk melengkapi queri untuk pencarian.

2.3. Ontologi

Ontologi merepresentasikan pengetahuan sebagai kumpulan konsep dan hubungannya dalam suatu domain. Ontolgi biasa digunakan secara formal dan ekplisit untuk menggambarkan secara spesifik konsep dan hubungan antar konsep dalam suatu domain(Zhang, 2006). Ontologi memberikan penjelasan secara eksplisit dan formal dari konsep sebagai representasi pengetahuan. Banyak manfaat penggunaan ontologi, antara lain ontologi bersifat formal, ekplisit dan berbagi, sehingga pengetahuan didalamnya dapat digunakan semua orang. Ontologi berorientasi mesin, memudahkan untuk berbagi antara mesin yang berbeda. Langkah-langkah yang diperlukan untuk mengembangkan suatu ontologi adalah :

1. Menentukan kelas (konsep dalam domain) dalam ontologi.
2. Menyusun kelas dalam hirarki taksonomi(*subclass* - *superclass*).
3. Menentukan slot(properti) dari setiap kelas dan aturan dalam mengisi slot tersebut.
4. Mengisi instansi atau individu dalam setiap kelas.
5. Mengisi properti dari setiap individu.

Ontologi memiliki beberapa komponen penyusun yaitu:

a. Konsep

   Konsep digunakan untuk merepresentasikan suatu kelas dalam suatu entitas atau sesuatu dalam suatu domain. Sebuah konsep dapat dikatakan kelas, objek, maupun kategori. Dalam suatu kelas atau konsep mungkin terdapat individu atau *instance*.

b. Relasi

   Relasi menjelaskan hubungan antara dua objek dalam ontologi. Relasi dapat dilihat sebagai atribut suatu kelas yang bertipe kelas lain. Sebagai contoh dari relasi adalah *hasMethod*, *subclassOf*, *class*(konsep) *hasMethod* (relasi) *method*(konsep).

c. Fungsi

   Fungsi dalam ontologi merupakan struktur kompleks yang terbentuk dari

relasi antara dua objek. Fungsi adalah sebuah relasi khusus dimana elemen ke n dari relasi adalah unik untuk elemen ke n-1.

d. Aksiom

Aksiom merupakan pernyataan dalam bentuk logis yang bersama-sama membentuk keseluruhan teori yang menjelaskan ontologi dalam domainnya aplikasi. Aksiom digunakan untuk memodelkan sebuah pernyataan yang selalu benar.

e. Instance

Instance dalam ontologi merepresentasikan satu individu dalam suatu domain. Digunakan untuk merepresentasikan elemen atau bagian terkecil dari sebuah konsep. Contoh instan dari kelas sungai adalah sungai brantas.

2.4. Pemrosesan Bahasa Alami

Pemrosesan bahasa alami atau *Natural Language Processing*(NLP) adalah sebuah bidang dalam ilmu komputer dan bahasa yang berkaitan dengan sistem temu kembali(*information retrieval*/IR) dan berkonsentrasi pada interaksi antara bahasa manusia dan komputer. Melakukan proses pada bahasa manusia supaya bisa dimengerti oleh komputer tanpa kehilangan aspek makna dari tiap baris kalimat. Prosesnya tidak hanya membagi dan membandingkan kata seperti yang dilakukan IR. Salah satu alat yang digunakan dalam pemrosesan bahasa alami adalah *StanfordNLP*. *StanfordNLP* adalah sistem yang dikembangkan oleh *The Stanford NLP Group* dengan fokus penelitian pada *sentence understanding*, *probabilistic parsing* dan *tagging*, *biomeical information extraction*, *grammar introduction*, *word sense disambiguation*, dan *automatic question answering* (nlp.stanford.edu, 2013). NLP dalam penelitian ini digunakan untuk melakukan tagging. *Part of Speech* (POS) *tagger* adalah proses perangkat lunak untuk membaca teks dalam bahasa alami dan memberikan label dengan menggunakan *Part of Speech* (POS) seperti kata kerja, kata benda, kata sifat dll.

2.5. Kedekatan Semantik

Kedekatan semantik mencoba untuk mengukur sejauh mana dua konsep semantik berhubungan satu sama lain dengan memanfaatkan berbagai jenis link semantik yang menghubungkan mereka. Tujuan utama adalah untuk meniru model mental manusia ketika menghitung keterkaitan dari dua buah kata. Otak manusia menetapkan keterkaitan semantik antara kata-kata berdasarkan struktur internal dari maksud, atau arti tersirat dari kata. Sebagai contoh kata sapi dan kuda, memiliki relasi sebagai hewan mamalia berkaki empat. Aspek lain dalam pola pikir manusia adalah dua kata yang sering muncul bersamaan, dianggap memiliki relasi. Seperti kata meja dan kursi, sering muncul bersama sehingga dianggap memiliki relasi. Pengetahuan seseorang terhadap suatu konteks mempengaruhi besarnya derajat hubungan antar kata.

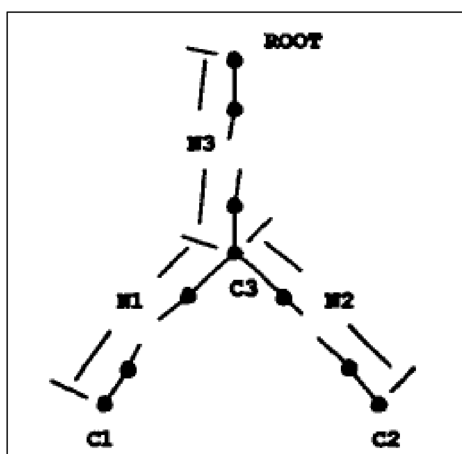
Berbagai metode untuk mengukur kedekatan semantik dibahas dalam beberapa literatur. Langkah-langkah utama dalam mengukur kedekatan semantik adalah dengan memperkirakan keterkaitan kata secara semantik dengan memanfaatkan sejumlah besar domain pengetahuan leksikal dan menggunakan teknik statistik untuk memanfaatkan semua kemungkinan hubungan yang berkontribusi pada kesamaan konsep. Domain pengetahuan biasanya tersedia dalam sumber-sumber eksternal termasuk pengetahuan berbasis linguistik(*linguistic knowledge base*/LKB) seperti WordNet(Fellbaum, 1998) dan pengetahuan berbasis kolaboratif (*collaborative knowledge base*/CKB). WordNet merupakan kamus (database leksikal) online untuk bahasa inggris. WordNet mendefinisikan sebuah kata menjadi sebuah konsep yang diorganisasi kedalam sejumlah sinonim set(synset)(Miller dkk, 1990). *Synset* memiki beberapa macam hubungan dengan *synset* lainnya. Metode-metode untuk menghitung kedekatan semantik antar kata menggunakan WordNet dikategorikan dalam Tabel 2.3.

Tabel 2.3. Metode-Metode Kemiripan Semantic(Shridara dkk, 2006)

Penulis	**Kategori**
Leacock dan Chadrow	Path based
Wu dan Palmer	Path based
Jiang dan Conrath	Information content based
Resnik	Information content based
Lin	Information content based
Banerjee dkk	Gloss based

Penelitian dilakukan Sridhara(2006) dengan membandingkan beberapa teknik kemiripan semantik antara kata dalam komentar dengan nama metode dalam kode sumber menunjukkan hasil bahwa teknik Wu dan Palmer memberikan hasil paling baik dalam mencari kemiripan kata antara bahasa alami dengan kata dalam kode sumber.

Wu-Palmer(1994) mendefinsikan kedekatan semantik sebagai jarak antara dua konsep dalam struktur WordNet dan jarak *Least Common Subsumer*(LCS). Untuk menghitung kemiripan kata seperti pada ilustrasi Gambar 2.5, dengan rumus Wu-Palmer ditunjukkan pada Rumus 2.1.



Gambar 2.6. Ilustrasi Kemiripan Kata (Wu dan Palmer, 1994)

$$\text{sim}(c1, c2) = \frac{2 \times N3}{N1 + N2 + (2 \times N3)} \quad (2.1)$$

Nilai kemiripan antar dua kalimat Sen_a dan Sen_b dapat dihitung menggunakan Rumus 2.4 yang diusulkan oleh Ming Che Lee(2011).

$$\text{NC}_{ab} = \left(\frac{\vec{NV}_{\text{sen_a}} \cdot \vec{NV}_{\text{sen_b}}}{|\vec{NV}_{\text{sen_a}}| \times |\vec{NV}_{\text{sen_b}}|} \right)^2$$

$$= \left(\frac{\sum_{i=1}^{\text{S_N}_a \cup \text{S_N}_b} NV_{\text{sen_a}_i} \times NV_{\text{sen_b}_i}}{\sqrt{NV^2_{\text{sen_a_i}}} \times \sqrt{NV^2_{\text{sen_b_i}}}} \right)^2 \quad (2.2)$$

$$\text{VC}_{ab} = \left(\frac{\vec{VV}_{\text{sen_a}} \cdot \vec{VV}_{\text{sen_b}}}{|\vec{VV}_{\text{sen_a}}| \times |\vec{VV}_{\text{sen_b}}|} \right)^2$$

$$= \left(\frac{\sum_{i=1}^{\text{S_N}_a \cup \text{S_N}_b} VV_{\text{sen_a}_i} \times VV_{\text{sen_b}_i}}{\sqrt{VV^2_{\text{sen_a_i}}} \times \sqrt{VV^2_{\text{sen_b_i}}}} \right)^2 \quad (2.3)$$

$$\text{Similarity}_{a,b} = (\zeta NC_{ab} \times (1 - \zeta) VC_{ab}) \quad (2.4)$$

dengan :

NC_{ab}	: Kemiripan semantik antara vektor kata benda(*noun*) kalimat Sen_a dan kalimat Sen_b.
VC_{ab}	: Kemiripan semantik antara vektor kata kerja(*verb*) kalimat Sen_a dan kalimat Sen_b.
VV_a	: Vektor kata kerja(*verb*) kalimat Sen_a .
NV_a	: Vektor kata benda(*noun*) kalimat Sen_a .
$Similarity_{a,b}$	: Nilai kemiripan antara kalimat Sen_a dan Sen_b.
ζ	: Konstanta pengali yang menunjukkan perbandingan kontribusi kata benda dan kata kerja dalam menentukan kemiripan.

Dalam rumus diatas didefinisikan bahwa suatu kalimat dibagi menjadi satu kelompok kata kerja(S_V_a) dan satu kelompok kata benda (S_N_a). Sedangkan *base space* kata kerja adalah gabungan dari kelompok kata kerja dari kedua kalimat. *Base space* kata benda adalah gabungan dari kelompok kata benda dari kedua kalimat. Sehingga dapat didefinisikan sebagai berikut :

$Sen_a=\{S_V_a, S_N_a\}$

$Sen_b=\{S_V_b, S_N_b\}$

Vektor kata kerja |VV| merupakan vektor yang terbentuk dari kelompok kata kerja terhadap *base space* kata kerja. Vektor *base space* kata kerja dan Vektor *base space* kata benda bisa didefinisikan sebagai berikut :

$|VV_{sen_a}|=|VV_{sen_b}| = |S_V_a \cup S_V_b|$

$|NV_{sen_a}|=|NV_{sen_b}| = |S_N_a \cup S_N_b|$

Vektor kata kerja dapat dihitung dari kemiripan maksimal masing-masing kata dalam kelompok kata kerja kalimat Sen_a terhadap *base space* kata kerja menggunakan Persamaan 2.1. Demikian juga vektor kata benda |NV| dapat dihitung dari kemiripan maksimal masing-masing kata dalam kelompok kata benda kalimat Sen_a terhadap *base space* kata benda.

$$NV_{sen_a_i} = \operatorname{Max}_{i=1}^{|S_N_a \cup S_N_b|} \operatorname{sim}(kata_i, SN_i) \tag{2.5}$$

$$VV_{sen_a_i} = \operatorname{Max}_{i=1}^{|S_V_a \cup S_V_b|} \operatorname{sim}(kata_i, SV_i) \tag{2.6}$$

2.6. Metode Evaluasi

Kinerja sistem temu kembali informasi(IR) dievaluasi dengan *precision* dan *recall*. *Precision* adalah tingkat ketepatan antara informasi yang diminta oleh pengguna dengan jawaban yang diberikan oleh sistem. Precision dapat diartikan sebagai kepersisan atau kecocokan (antara permintaan informasi dengan jawaban). Jika seseorang mencari informasi di sebuah sistem, dan sistem menawarkan beberapa dokumen, maka kepersisan ini sebenarnya juga adalah relevansi. *Recall* adalah tingkat keberhasilan sistem dalam menemukan kembali sebuah informasi. Recal dapat dirumuskan sebagai jumlah dokumen relevan yang ditemukan dibagi jumlah semua dokumen relevan di dalam koleksi. Lalu, *precision* adalah jumlah dokumen relevan yang ditemukan dibagi jumlah semua dokumen yang ditemukan. Kedua ukuran di atas biasanya diberi nilai dalam bentuk persentase, 1 sampai 100%. Sebuah sistem informasi akan dianggap baik jika tingkat *recall* maupun *precision*-nya tinggi. Dalam menghitung *precision* dan *recall*, dokumen

dikelompokkan menjadi empat kelompok seperti ditunjukkan pada Tabel 2.4

Tabel 2.4. Pembagian Dokumen Evaluasi

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">Nilai Sebenarnya</td></tr>
<tr><td>True</td><td>False</td></tr>
<tr><td rowspan="2">Nilai Prediksi</td><td>True</td><td>TP(true positive)
relevan, ditemukan</td><td>FP(false positive)
tidak relevan, ditemukan</td></tr>
<tr><td>False</td><td>FN(false negative)
relevan, tidak ditemukan</td><td>TN(true negative)
tidak relevan, tidak ditemukan</td></tr>
</table>

$$recall=\frac{TP}{TP+FN} \tag{2.7}$$

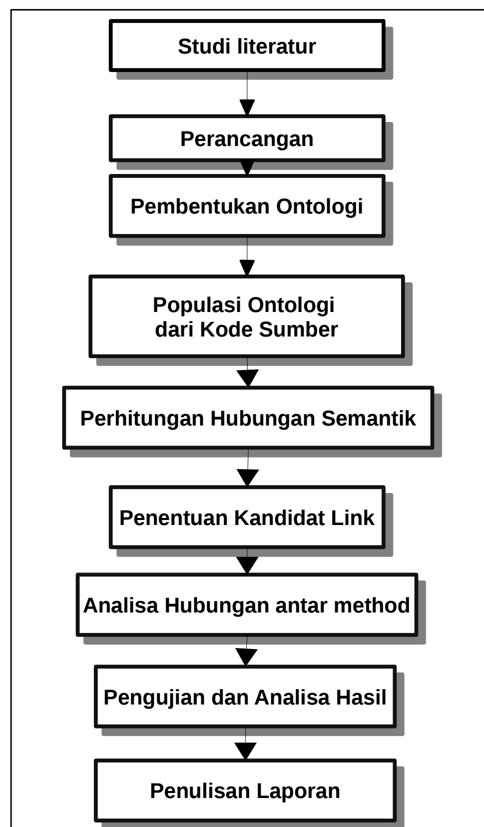
$$precision=\frac{TP}{TP+FP} \tag{2.8}$$

Dalam Rumus 2.7 dan 2.8 dapat disimpulkan bahwa sebuah sistem harus meningkatkan nilai *recall* dengan memperbesar nilai TP. Nilai TP yang besar ini dapat terjadi jika jumlah dokumen yang diberikan oleh sebuah sistem dalam sebuah pencarian juga besar. Semakin besar jumlah dokumen yang diberikan, semakin besar kemungkinan nilai TP. Tetapi pada saat yang sama, muncul kemungkinan bahwa nilai FP (atau jumlah dokumen yang tidak relevan) juga semakin besar. Ini artinya, nilai *precision*-nya semakin kecil.

BAB 3
METODOLOGI PENELITIAN

3.1. Tahapan Penelitian

Sistematika metodologi penelitian yang dilakukan terdiri dari studi literatur, perancangan, pembentukan ontologi, populasi ontologi dari kode sumber, perhitungan hubungan semantik, penentuan kandidat link, analisa hubungan antar *method*, pengujian dan analisa hasil. Urutan proses penelitian ditunjukkan pada pada Gambar 3.1.



Gambar 3.1. Metodologi Penelitian

3.2. Studi Literatur

Studi literatur bertujuan untuk mengumpulkan informasi mengenai metode penggalian informasi, kedekatan semantik, pemrosesan bahasa alami dan ontologi yang akan digunakan untuk menemukan link penelusuran anatara dokumen dengan kode sumber. Studi literatur yang dilakukan adalah sebagai berikut :

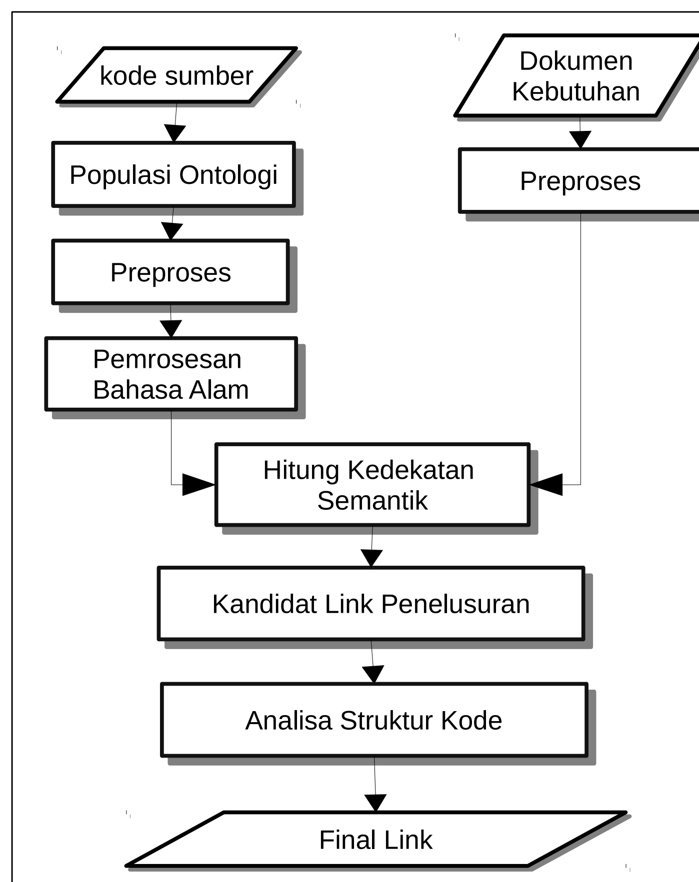
- Mempelajari metode penggalian informasi yang sudah digunakan sebelumnya yaitu VSM, LSI, JS dan LDA serta kekurangannya.
- Mengumpulkan informasi tentang pembangunan link penelusuran menggunakan ontologi dan kedekatan semantik yang menjadi latar belakang penelitian ini.
- Mengumpulkan informasi tentang metode perhitungan kedekatan semantik antar konsep menggunakan pemrosesan bahasa alami.
- Mengumpulkan referensi tentang metode ekstraksi informasi dari kode sumber menggunakan pemrosesan bahasa alami.

3.3. Perancangan

Perancangan dilakukan untuk membangun sistem untuk menemukan link penelusuran antara kebutuhan dan kode sumber. Masukan yang diperlukan dalam proses ini adalah kode sumber dan dokumen kebutuhan. Urutan langkah kerja dari sistem yang dibangun ditunjukkan pada Gambar 3.2. Ada tiga langkah utama dalam sistem ini, yaitu :

- Membentuk kumpulan kata dalam bahasa alami dari *method signature* dalam kode sumber. *Method signature* dalam kode sumber merupakan deklarasi suatu *method* dalam *class*. Kumpulan kata dihasilkan melalui preproses kode sumber dan proses pengolahan bahasa alami.
- Menghitung kemiripan antara *method signature* dengan dokumen kebutuhan. Menentukan *method* utama berdasarkan nilai kedekatan semantik. *Method* utama adalah *method* yang memiliki nilai kemiripan tinggi antara dokumen kebutuhan dengan kata yang dihasilkan dari pemrosesan bahasa alami terhadap *method* tersebut.

- Menghitung nilai akhir kemiripan pada *method* utama dengan analisa struktur kode sumber. Dilakukan penelusuran *method invocation* (pemanggilan *method*) dari setiap *method* utama. Penelusuran dilakukan terhadap *method-method* yang dipanggil oleh *method* utama. Penelusuran dilakukan terhadap *method* yang dipanggil secara langsung oleh *method* utama maupun *method* yang dipanggil secara tidak langsung yaitu *method* yang dipanggil oleh *method* yang berhubungan dengan *method* utama. Dilakukan penelusuran terhadap semua *method* yang dipanggil oleh *method* utama dan *method-method* dibawahnya. Penelusuran hanya pada kode sumber tidak melibatkan *method* dalam *library* atau *class* dalam JDK. Kemudian dihitung rata-rata dari total nilai kemiripan semua *method* yang diperoleh dari hasil penelusuran.



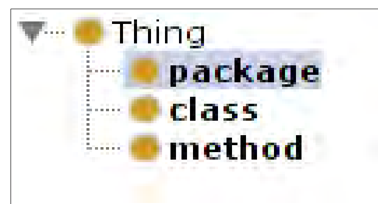
Gambar 3.2. Diagram Alir Pembangunan Link Penelusuran

Selama proses pembangunan link penelusuran, informasi disimpan dalam ontologi. Ontologi dibentuk dari ekstraksi *method* dan hubungan antar *method* dalam kode sumber. Untuk melakukan ekstraksi dari kode sumber perlu dilakukan analisa statik untuk membentuk *abstract syntax tree*(AST) dari kode sumber. Keuntungan menggunakan ontologi sebagai sumber data adalah sebagai berikut :

- Ontologi mampu menyimpan data individu dan hubungan antar individu secara dinamik. Individu yang dibutuhkan dalam proses ini adalah komponen-kompoonen kode sumber seperti *class* dan *method* yang ada dalam kode sumber. Ontologi juga menyimpan hubungan antar individu seperti hubungan antara method yang memanggil(*invoke*) dan yang dipanggil(*invokedBy*) oleh individu tersebut. Hubungan memanggil dan dipanggil suatu individu bisa satu ke satu, satu ke banyak dan tidak ada hubungan. Hubungan antar *class* seperti turunan, suatu *class* bisa merupakan turunan *class* lain atau bukan turunan *class* lain.
- Informasi mengenai suatu individu dalam kode sumber dapat diambil atau dibaca secara langsung tanpa harus melakukan penelusuran ulang terhadap kode sumber atau AST kode sumber. Dengan satu SPARQL queri bisa didapatkan semua individu yang diinginkan sesuai dengan kriteria queri.
- Penggunaan *inference engine* pada ontologi mampu menghasilkan individu-individu yang saling berhubungan. Fitur ini berguna pada saat melakukan analisa struktur kode sumber. Dengan memanfaatkan *inference engine* pada ontologi bisa didaptkan *method-method* yang dipanggil oleh suatu *method*.

3.4. Pembentukan Ontologi

Dalam penelitian ini ontologi dikembangkan untuk menampung entiti-entiti dari kode sumber dan relasi antar entiti. Entiti dalam kode sumber antara lain *class dan method*. Sedangkan relasi antar entiti bisa terjadi antara *class* dengan *class*, *class* dengan *method*, *method* dengan *method.* Ontologi digunakan karena ontologi dapat merepresentasikan entiti dan hubungan antara entiti yang ada di dalamnya. Struktur ontologi yang digunakan ditunjukkan pada Gambar 3.3 dan Gambar 3.4, penjelasannya ditunjukkan pada Tabel 3.1 dan Tabel 3.2.



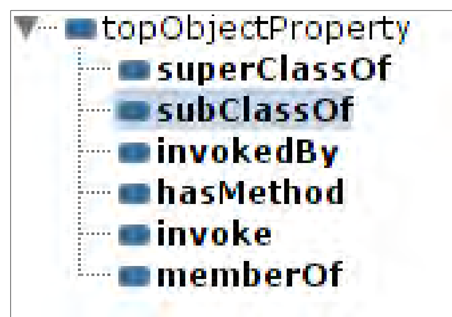
Gambar 3.3. Struktur Entiti dalam Ontologi

Tabel 3.1. Jenis Entiti dalam Ontologi

Nama	Keterangan
package	Entiti yang merepresentasikan *package* dari suatu kelas dalam kode sumber
class	Entiti *class* beranggotakan *class* dan interface dalam kode sumber
method	Entiti beranggotakan yang merupakan suatu operasi yang dilakukan oleh *class*

- Entiti *package* merupakan entiti yang mewakili pengelompokan kelas dalam *package* dari kode sumber. *Package* dalam bahasa pemrograman java mewakili struktur folder dari suatu kelas.
- Entiti class memiliki anggota class dan interface dalam source code. Class merupakan modul tebesar dalam java. Dalam proses parsing source code setiap class dan interface akan membentuk satu individu dalam entiti class. Entiti class memiliki objectRelation hasSuperClass dengan entiti class yang lain. Entiti class memiliki objectRelation hasMethod dengan entiti method.

- Entiti method memiliki anggota individu semua *method* dalam *class*. Setiap deklarasi *method* dalam kode sumber akan membentuk individu anggota entiti method. Suatu deklarasi *method* memiliki modifier, bisa memiliki nilai balik dan parameter.



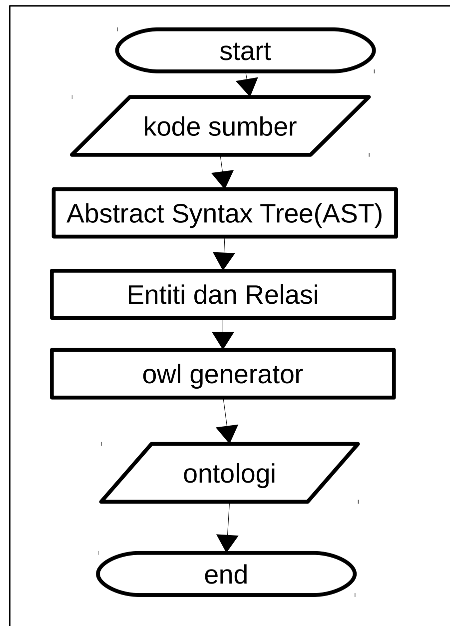
Gambar 3.4. Jenis-jenis Relasi dalam Ontologi

Tabel 3.2. Hubungan Antar Objek dalam Ontologi

Nama	Keterangan
subClassOf	Class A merupakan turunan dari class B
superClassOf	Class A diturunkan ke class B
hasMethod	Class A memiliki method C
memberOf	Method A adalah anggota dari class C, inverse dari hasMethod
invokedBy	Method A dipanggil oleh method B
invoke	Method A memanggil method B

3.5. Populasi Ontologi

Ontologi dibentuk dari ekstraksi objek dalam kode sumber dan hubungan antar objek dalam kode sumber. Ekstraksi hanya dilakukan terhadap kode sumber dan tidak termasuk komentar dalam kode sumber. Untuk melakukan ekstraksi dari kode sumber perlu dilakukan analisa statik untuk membentuk *abstract syntax tree*(AST) dari kode sumber. Analisa statik program adalah proses analisa perangkat lunak yang dilakukan tanpa melakukan eksekusi terhadap perangkat lunak. Analisa statik dilakukan terhadap kode sumber perangkat lunak bertujuan untuk mengerti struktur dari suatu perangkat lunak.



Gambar 3.5. Diagram Alir Populasi Ontologi dari Kode Sumber

Dalam ilmu komputer, AST adalah suatu *tree* yang merepresentasikan struktur sintaksis dari suatu kode sumber. Tahapan proses pembentukan ontologi dari kode sumber adalah sebagai berikut:

1. Ekstraksi *package*, *class* dan *method*. Untuk mengambil semua *package*, *class* dan *method*, dilakukan penelusuran terhadap *syntax tree*. Ketentuan dalam penelusuran *tree* adalah sebagai berikut :
   - Jika titik pada *syntax tree* tersebut merupakan deklarasi suatu *package*, dibuat satu individu dalam entiti *package*. Individu dibuat dengan nama sesuai dengan nama package dalam source code.
   - Jika titik pada *syntax tree* tersebut merupakan deklarasi suatu *class* atau *interface*, dibuat satu individu dalam entiti *class*. Dalam bahasa pemrograman java, suatu *class* bisa memiliki nama yang sama dengan class lain tapi dalam *package* yang berbeda, karena itu individu dibuat dengan nama sesuai dengan nama package disambung dengan nama kelas tersebut.
   - Jika titik pada *syntax tree* tersebut merupakan deklarasi suatu *method*

dalam *class*, dibuat satu individu dengan jenis entiti *method*. Individu dibuat dengan nama sesuai dengan nama *package* disambung dengan nama *class* dan nama *method*.

2. Ekstraksi hubungan antar *method* atau *class*. Proses ekstraksi ini dilakukan dengan penelusuran kembali pada *syntax tree*. Dalam penelusuran yang kedua ini yang dicari adalah pemanggilan suatu *metode* dalam *metode* lainnya. Pada penelusuran kedua juga ditambahkan individu jenis method yang berasal dari hasil turunan. Jika individu *method* yang dipanggil suatu *method* tidak ada dalam individu *class*, maka individu *method* ditambahkan sesuai dengan individu *method* dalam individu *class* induknya.

3.6. Preproses Kode Sumber

Preproses dilakukan pada kode sumber untuk mengambil kata-kata yang memiliki makna dari dalam suatu kode sumber. Preproses dilakukan terhadap nama *package*, nama *class*, dan *method signature*. *Method signature* terdiri dari nama *method*, tipe data kembalian dari *method* dan parameter dalam suatu *method*. Langkah pertama dalam preproses adalah melakukan pemisahan kata(*token splitting*) dalam *identifier* menjadi kata yang bermakna. Dalam kode sumber ada beberapa aturan yang sering digunakan untuk memisahkan *identifier* yaitu:

- Garis bawah(*underscore*), *identifier* 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*, *identifier* 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*, *identifier* dibentuk dari gabungan kata dengan huruf kapital dan kata dengan huruf kecil, seperti contoh pada identifier ASTVisitor

menjadi AST Visitor.

Preproses dimulai dari memisahkan berdasarkan karakter *non-alphanumeric* yaitu karakter yang bukan huruf ataupun angka. Pada proses ini garis bawah(*underscore*) akan diganti dengan spasi, seperti “send_email” akan dipisah menjadi “send email”. Pada proses ini juga dilakukan pemisahan *method signature* yang berbentuk *camel case* dan *mixed case,* dipisah dengan spasi. Kode sumber proses pemisahan ditunjukkan pada Gambar 3.6.

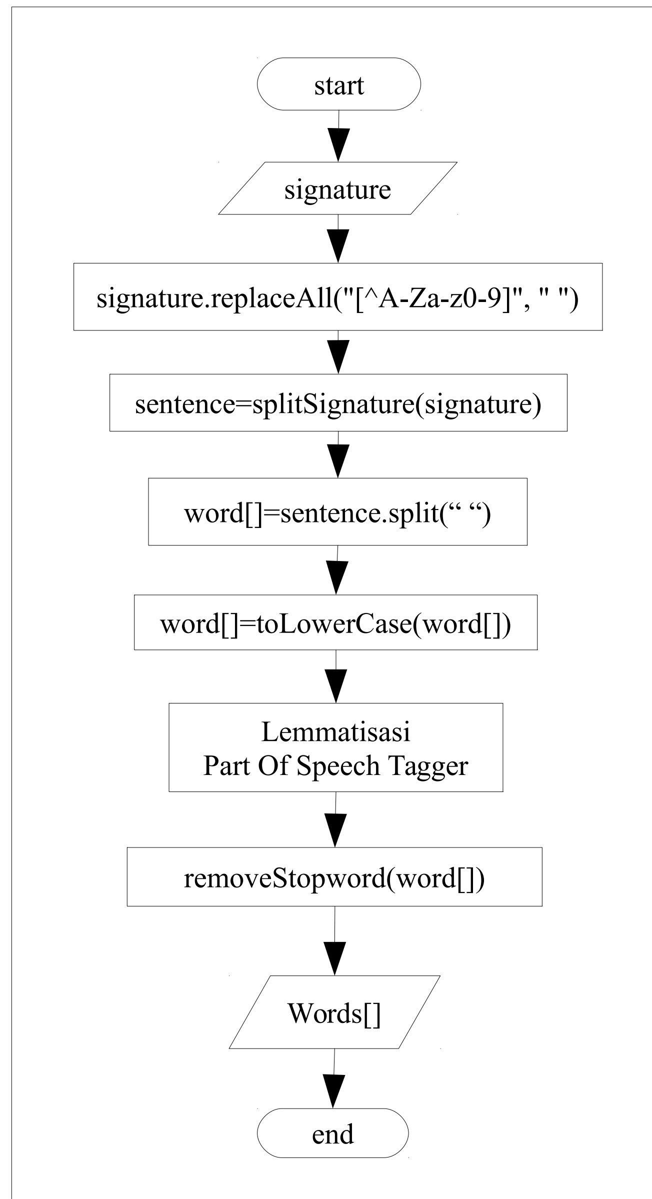
```
public static String splitIdentifier(String sen) {
      sen = sen.trim();
      sen = sen.replaceAll(String.format("%s|%s|%s",
                  "(?<=[A-Z])(?=[A-Z][a-z])",   "(?<=[^A-Z])(?
=[A-Z])",
                  "(?<=[A-Za-z])(?=[^A-Za-z])"), " ");
      return sen;
   }
```

Gambar 3.6. Kode Sumber Proses *Parsing Identifier*

Langkah kedua dalam preproses adalah proses lemmatisasi yakni mengembalikan kata ke akar katanya. Lemmatisasi merupakan suatu metode untuk menemukan satu bentuk kata dasar dari beberapa bentuk kata. Suatu kata dalam bahasa inggris kadang muncul dalam beberapa bentuk seperti *run*, *running*, *ran* yang memiliki akar kata *run.* Lemmatisasi berbeda dengan proses *stemming*. *Stemming* berusaha mencari bentuk dasar dari suatu kata dengan cara menghilangkan imbuhan dari kata. Proses *stemming* hanya menggunakan aturan-aturan untuk memotong kata dalam mencari satu bentuk kata dasar tanpa mempedulikan apakah hasilnya masih memiliki arti atau tidak. Beberapa contoh hasil dari *stemming* antara lain *resources* menjadi *resourc*, *comunication* menjadi *comun*, *community* menjadi *communiti*. Proses *stemming* kurang sesuai untuk proses pengolahan bahasa alami karena tidak selalu menghasilkan kata yang memiliki makna. Lemmatisasi menggunakan algoritma yang lebih baik dari pada *stemming* untuk menemukan kata dasar yang memiliki makna. Lemmatisasi menggunakan analisa morfologi dan kosakata yang sesuai dengan konteks kalimat unuk menemukan bentuk dasar atau bentuk sesuai kamus yang disebut *lemma*.

Langkah ketiga dalam preproses adalah menghilangkan *stopwords*. Tidak

semua kata yang terdapat dalam kode sumber adalah kata penting. Kata penghubung seperti “the” dan “a” tidak memiliki arti penting dalam kalimat, karena itu harus dihilangkan. Untuk menghilangkan *stopwords*, setiap kata akan dibandingkan dengan daftar *stopwords.* Jika kata tersebut ada dalam daftar stopword, maka kata tersebut dihapus dari daftar kata dalam kalimat. Diagram alir langkah preproses ditunjukkan pada Gambar 3.7.



Gambar 3.7. Diagram Alir Preproses *Method Signature*

3.7. Preproses Kebutuhan Fungsional

Kebutuhan fungsional yang digunakan dalam penelitian ini berbentuk bahasa alami. Suatu kebutuhan fungsional bisa terdiri dari lebih dari satu kalimat. Preproses kebutuhan fungsional diawalai dengan memisahkan kalimat. Akhir kalimat ditandai dengan tanda titik(.), tanda tanya(?) atau tanda seru(!). Kemudian dari masing-masing kalimat akan dihilangkan karakter *non-alphanumeric*. Selanjutnya dilakukan pemisahan kata berdasarkan *white space*(spasi).

Langkah berikutnya adalah melakukan lemmatisasi dan *part of speech tagger* terhadap kata-kata yang sudah dipisah. Proses lemmatisasi dan Part of Speech Tagger menggunakan *library* dari Stanford CoreNLP. Stanford CoreNLP menggunakan proses *pipeline* pengolahan bahasa alami berbasis anotasi (Manning,2014). Stanford CoreNLP dibuat sebagai program yang kecil dan ringan tapi bisa dikembangkan dan mudah digunakan dalam program yang lebih besar. StanforNLP mendukung pemrosesan bahasa alami berbagai jenis bahasa. Dalam CoreNLP, setiap bahasa diwakili oleh satu model yang bisa diunduh terpisah kemudian ditambahkan dalam aplikasi sehingga membuat CoreNLP tetap ringan dan mudah dikembangkan dengan menambahkan model-model yang dibutuhkan oleh aplikasi. Langkah berikutnya dalam preproses adalah menghilangkan *stopwords*. Diagram alir langkah preproses ditunjukkan pada Gambar 3.8.

Sebagai contoh, salah satu kebutuhan fungsional "*Create a new task which typically has a start date and an end date. Tasks are activities that one or more persons resources are expected to complete in the specified time frame*". Kebutuhan fungsional tersebut terdiri dari dua kalimat. Setelah dilakukan pemisahan kalimat, lemmatisasi, menghilangkan *stopword* dan proses POS-*tagging* akan dihasilkan dua kelompok kata kerja dan kelompok kata benda sebagai berikut :

- Kalimat 1, " Create a new task which typically has a start date and an end date"

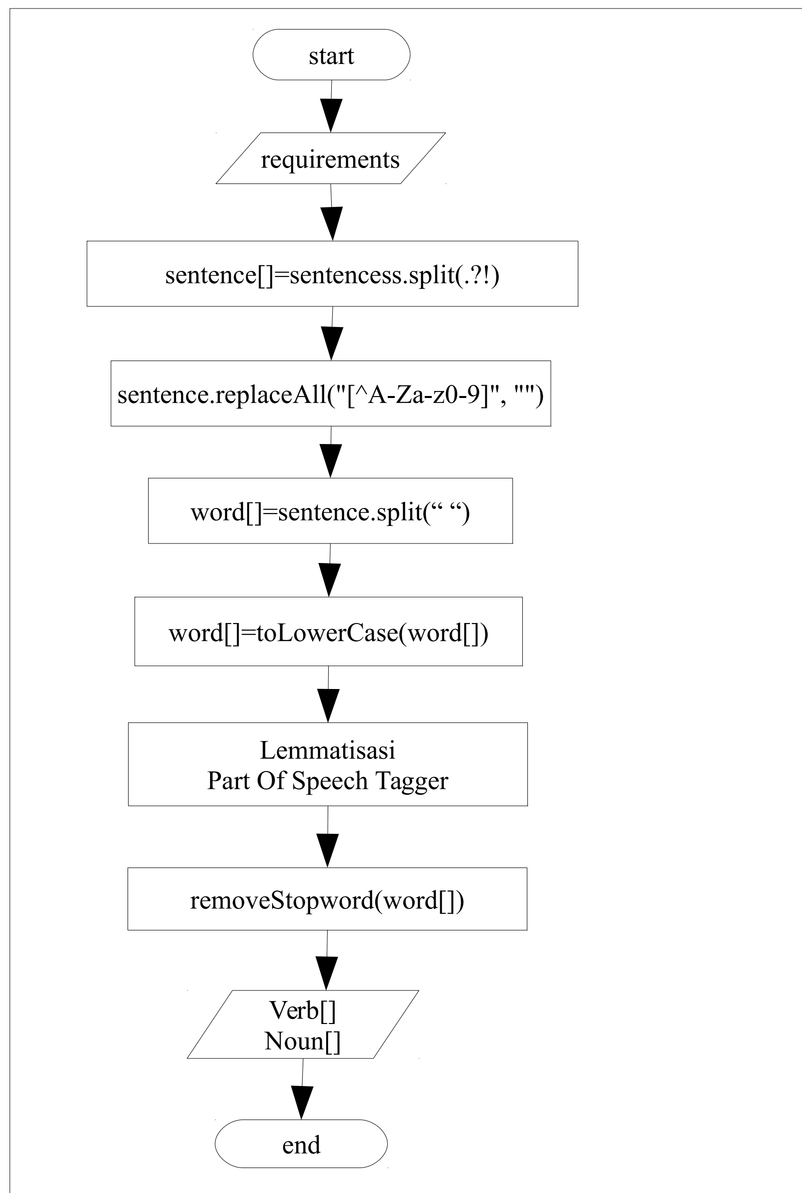
  $Verb_1$: [create, have]

  $Noun_1$: [task, start, date, end]

- Kalimat 2, “ Tasks are activities that one or more persons resources are expected to complete in the specified time frame”

  $Verb_2$: [are, expect, complete]

  $Noun_2$: [task, activity, person, resource, time, frame]



Gambar 3.8. Diagram Alir Preproses Kebutuhan Fungsional

3.8. Pemrosesan Bahasa Alam

Langkah berikutnya adalah melakukan ekstraksi kata dari *method signature.* Tujuan utama dalam ekstraksi kata ini adalah menemukan aksi atau operasi yang dilakukan suatu *statement* dalam kode sumber. Proses ini memanfaatkan POS tagger untuk menentukan kata kerja dan kata benda dalam *method signature*. Dalam pengembangan perangkat lunak, ada beberapa pola yang digunakan pengembang dalam membuat suatu *method*, yaitu :

1. Nama *method* terdiri dari kata kerja dan kata benda. Konsep yang disimpulkan dari model ini adalah nama kelas, kata kerja, kata benda. Misalkan ada sebuah *class* dengan nama UserManager memiliki *method* sendEmail.

   ```
   public class UserManager
   {
       public void sendEmail(){
       .....
       }
   }
   ```

   Dari proses pemisahan kata akan menghasilkan nama class adalah “user manager” dan nama method adalah “send email”. POS tagger dari kata send email adalah ((send, VP),(email, NP)), maka konsep yang dihasilkan adalah “user manager send email”.

2. Nama *method* hanya kata kerja dan kata benda berada dalam parameter *metode*.

   ```
   public class UserManager
   {
       public void send(String message){
       .....
       }
   }
   ```

   Dari proses pemisahan kata akan menghasilkan nama *class* adalah “user manager” dan nama *method* adalah “send”. POS tagger dari kata send adalah (send, VB), maka konsep yang dihasilkan adalah “user manager send message”.

Suatu deklarasi *method* terdiri dari *method modifier*, *method return type*, nama

method dan parameter. Untuk mengektraksi kata dari suatu *method signature* digunakan aturan sebagai berikut :

1. Tipe data primitif seperti int, long, char dan tipe data bawaan kompiler akan diabaikan dalam proses ekstraksi kata dalam *method signature*. Jenis data bawaan kompiler seperti String, Map, List sangat umum digunakan dalam kode sumber, sehingga tidak akan digunakan dalam proses ekstarksi kata. Seperti pada *method* berikut :

   ```
   public void doAction(String cmd, AuctionEntry ae){
       .....
   }
   ```

   maka kata String akan diabaikan sehingga kumpulan kata yang dihasilkan adalah “do Action Auction Entry”.

2. Dalam proses ekstraksi kata, tidak hanya nama *method* yang digunakan, tetapi ditambahkan juga nama *class* dalam *method signature*. Nama *class* pada kode sumber memberikan kontribusi dalam menentukan konteks dari aksi yang dilakukan oleh suatu *method*. Seperti pada *method* `UserAction.addAuction` yang dijalankan pada saat pengguna menambahkan auction. Sedangkan *method* `MessageAction.addAuction` dijalankan oleh server untuk menambahkan auction dari database ke antrian proses. Kedua *method* memiliki nama yang sama tetapi dipicu oleh aktor dan pada peristiwa yang berbeda. Pada contoh diatas nama *class* berperan penting dalam menentukan konteks dari aksi yang dilakukan oleh *method*. Dalam kode sumber ada kemungkinan suatu *class* merupakan satu aksi utama dan *method* merupakan *event* dalam aksi utama tersebut, seperti pada *class* `RotateAction` yang memiliki *method* `RotateAction.startAction()`. Nama *class* RotateAction dalam kode sumber ini menunjukkan aksi yang dilakukan oleh *class* tersebut, sedangkan nama *method* merupakan *event-event* yang terjadi pada aksi tersebut.

3. Jika terdapat kata yang sama dalam satu *method signature,* hanya diambil satu kali kata kemunculan kata tersebut dan mengabaikan kemunculan

berikutnya. Dalam deklarasi method sering kali terdapat kata yang sama digunakan secara berulang dalam nama method dan dalam nama atau tipe data parameter. Seperti conroh pada *method*

```
class RotateAction{
    public void startAction(){
     .....
    }
}
```

terdapat dua kali kata “action”. Pada *method* tersebut kata “action” mereferensi pada objek yang sama, sehingga kata yang dihasilkan adalah “start rotate action”. Contoh lain pada *method*

```
public void addAuction(AuctionEntry ae){
     .....
}
```

terdapat dua kata “auction”. Kata yang dihasilkan dari proses ekstraksi adalah “add auction entry”.

4. Pemanggilan konstruktor suatu *class* merupakan proses instansiasi atau pembuatan suatu objek. Untuk pemanggilan suatu konstruktor akan ditambahkan kata kerja “create”. Seperti pada kode sumber

```
{
     Auction au=new Auction();
    ...
}
```

maka kata yang dihasilkan adalah “create auction”.

5. Proses ekstraksi kata dari deklarasi *method* tidak menyertakan *return type* dan *modifier* dari suatu *method*.

```
public Currency getCurrentPrice(){
     .....
}
```

maka ekstraksi kata akan menghasilkan “get current price”.

6. Dalam proses ekstraksi kata, parameter suatu method juga diikutsertakan.

```
public void doAdd(AuctionEntry ae){
     .....
}
```

maka ekstraksi kata akan menghasilkan “do Add Auction Entry”.

7. Dalam kode sumber sering digunakan nama *method* yang sudah menjadi

standard dan maknanya secara tekstual tidak sesuai dengan kebutuhan fungsionalnya. Nama-nama *method* tersebut misalnya actionPerformed, run, execute, fire. Nama method ini dihilangkan atau diabaikan. Kata kerja dan kata benda dari *method signature* ini terdapat pada nama *class* dan parameter *method* tersebut.

```
class TaskAddAction {
  public void actionPerfomed(AuctionEntry ae){
    .....
  }
}
```

8. Dalam bahasa pemrograman java ada beberapa kata yang banyak digunakan dalam nama *method* atau nama *class* yang menjadi standard penamaan dalam *design pattern* seperti listener, event, action, adapter, handler, singleton, factory. Kata-kata tersebut dihilangkan dari *method signature.*

3.9. Perhitungan Kemiripan Semantik

Hubungan semantik dihitung antara kalimat dalam kebutuhan dengan kata-kata yang dihasilkan dari pemrosesan bahasa alami terhadap *method* dalam kode sumber. Perhitungan kemiripan semantik dilakukan menggunakan Persamaan 2.4. Dalam proses perhitungan semantik ini dipisahkan antara kata benda dan kata kerja dari method signature dan dokumen kebutuhan. Selanjutnya dibentuk kelompok kata *base space* kata kerja dan *base space* kata benda yang merupakan gabungan dari kedua kelompok kata dari *method signature* dan dokumen kebutuhan. Kemudian dibentuk dua vektor kata kerja dengan menghitung kedekatan semantik antara kelompok kata kerja dengan *base space* kata kerja. Juga dibentuk dua vektor kata benda dengan menghitung kedekatan semantik antara kelompok kata benda dengan *base space* kata benda. Setelah itu dihitung jarak kedua vektor kata kerja menggunakan *cosine similarity*. Dihitung juga jarak kedua vektor kata benda menggunakan *cosine similarity.* Nilai akhir kedekatan semantik dihitung menggunakan Persamaan 2.4.

Sebagai contoh proses perhitungan hubungan semantik antara phrase "*rotate*

image" dengan *method* `rotate()` pada *class* `ClipPanel`. Langkah pertama dilakukan ekstraksi dari *method signature* `ClipPanel.rotate()` akan dihasilkan kumpulan kata "clip panel rotate". Sehingga bisa didefinisikan :

Sen_a="rotate image"

Sen_b="clip panel rotate"

Dilakukan pemisahan antara kata kerja dan kata benda dalam masing-masing phrase.

VV_a = [rotate]

NV_a =[image]

VV_b=[rotate]

NV_b = [clip, panel]

Dibentuk *base space* yang merupakan gabungan antara kata dalam kedua phrase.

VS = [rotate]

NS = [clip, panel, image]

Kemudian dibentuk vektor kata kerja berdasarkan kemiripan maksimal masing-masing kata kerja terhadap *base space*. Karena kata kerja kedua phrase sama yaitu "rotate" maka nilai kemiripannya adalah 1.

	rotate
rotate	1

$|VV_a|=|VV_b| = [1]$

Kemudian dibentuk vektor kata benda berdasarkan kemiripan maksimal masing-masing kata terhadap *base space* kata benda menggunakan Persamaan 2.5.

	clip	panel	image
image	0.6315	0.8571	1

$|NV_a| = [0.6315, 0.8571, 1]$

	clip	panel	image
clip	**1**	0.8	0.631
panel	0.8	**1**	**0.8571**

$|NV_b|$ = [1, 1, 0.8571]

Kemudian dihitung jarak antara kedua vektor kata benda menggunakan Persamaan 2.2. diperoleh NC= 0.9711. Kemudian dihitung jarak antara kedua vektor kata benda menggunakan Persamaan 2.3. diperoleh VC= 1.0. Dengan Persamaan 2.4 dan nilai ζ sebesar 0,35 maka diperoleh

Sim_{ab}= 0.9899

3.10. Analisa Struktur Kode Sumber

Penelusuran dilakukan terhadap *method* yang dipanggil oleh *method* utama dan semua *method* dibawahnya sampai terakhir tidak ada *method* lagi yang dipanggil. *Method* utama adalah *method* yang *method signature*-nya memiliki nilai kemiripan terhadap kebutuhan fngsional lebih besar daripada nilai titik potong yang ditentukan. Penelusuran ini hanya pada *method* yang ada pada kode sumber tanpa menelusuri *method* yang ada di luar kode sumber seperti *method* dalam *library* atau *method* dalam SDK java. Dalam peneulusuran ini hanya method yang memiliki nilai kemiripan diatas 0,5 yang akan digunakan. Suatu *method* tidak hanya memanggil *method* yang bertujuan sesuai dengan kebutuhan fungsionalnya, misalkan suatu *method* memanggil *method* untuk mencatat log untuk keperluan proses pencarian kesalahan(*debug*). Mencatat log tidak berhubungan dengan kebutuhan fungsional dari *method* utama karena itu *method* ini dihilangkan dalam perhitungan kemiripan. Proses berikutnya adalah menjumlahkan total nilai kemiripan dari semua *method* yang terlibat dalam penelusuran kemudian dihitung nlai rata-rata dan menggunakan Persamaan 3.1 dihitung nilai akhir kemiripannya.

Proses pencarian berdasarkan kemiripan digabungkan dengan proses

penelusuran struktur kode sumber akan menghasilkan link penelusuran dengan nilai *precision* dan *recall* yang tinggi. Proses pencarian dengan pemrosesan bahasa alami diharapkan menghasilkan banyak link penelusuran yang relevan dan sedikit yang tidak relevan. Penelusuran pemanggilan *method* akan meningkatkan bobot kedekatan antara kode sumber dengan kebutuhan pada *method* yang benar-benar relevan sehingga bisa memperbesar jarak antara kandidat link yang relevan dengan link yang kurang relevan. Dengan asumsi bahwa suatu *method* yang relevan akan memanggil *method* lain yang relevan untuk mengimplementasikan suatu kebutuhan, maka bobot kedekatan *method* yang dipanggil akan berkontribusi terhadap bobot kedekatan *method* yang memanggilnya.

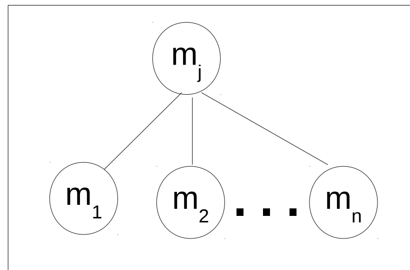
Analisa stuktur kode sumber untuk melengkapi kekurangtepatan dan kurang lengkap hasil pencarian dengan kemiripan semantik. Penelusuran akan menambahkan bobot kemiripan pada *method* yang memiliki hubungan dengan *method* yang memiliki kemiripan dengan kebutuhan. *Method* yang dihasilkan dari pencarian menggunakan perhitungan kemiripan dengan metode pemrosesan bahasa alam disebut method utama, ada dua keuntungan penggunan penelusuran pemanggilan *method*. Pertama pada kasus jika dua method utama berada dalam satu rangkaian *method invocation*, maka penelusuran kode sumber akan meningkatkan bobot kedekatan *method* utama yang memanggil. Kedua pada kasus jika dua *method* utama berada dalam dua rangkaian *method invocation* yang berbeda, maka penelusuran kode sumber akan meningkatkan bobot kedekatan *method* utama yang berhubungan dengan *method* utama lain atau bukan *method* utama yang memiliki kemiripan sehingga jarak antara dua *method* utama tesebut akan semakin besar dan bisa didentifikasi mana *method* utama yang benar-benar relevan.

Proses penelusuran dimulai dari method m_j sebagai titik awal. Penelusuran dilakukan untuk menghasilkan informasi struktur program dengan mengikuti jalur *method invocation*(pemanggilan *method*) dan menemukan semua *method* yang dipanggil oleh *method* m_j dan secara langsung maupun melalui method dibawahnya. *Method* yang dipanggil meliputi *method* dalam *class* yang sama

dengan method m_j maupun dari *class* lain. Proses penelusuran akan menghasilkan suatu *call graph* dengan titik merupakan *method* yang terlibat dalam proses *method invocation* dan garis menyatakan jalur hubungan antar *method* pemanggil dan yang dipanggil.

Proses pembentukan *call graph* dilakukan dengan menelusuri hubungan antara entiti dalam ontologi yang dibentuk dari AST kode sumber yang sudah dihasilkan dari langkah sebelumnya. Keuntungan menggunakan ontologi sebagai sumber data adalah kemampuannya dalam menyimpan data individu dan hubungan antar individu secara dinamik. Individu yang dibutuhkan dalam proses ini adalah individu dengan jenis *method*. Ontologi menyimpan hubungan antar *method* yang memanggil(*invoke*) dan yang dipanggil(*invokedBy*) oleh individu tersebut. Hubungan memanggil dan dipanggil suatu individu bisa satu ke satu, satu ke banyak dan tidak ada hubungan. Informasi mengenai hubungan individu *method* dalam kode sumber dapat diambil secara langsung dari ontologi tanpa harus melakukan penelusuran ulang terhadap kode sumber atau AST kode sumber.

Selanjutnya dilakukan pembobotan terhadap setiap titik dalam *call graph*. Bobot suatu titik menunjukkan tingkat kemiripan antara *method signature* dengan kebutuhan yang diimplementasikan.



Gambar 3.9. Pembobotan *Call Graph*

$$W(m_j) = \beta \,\mathrm{sim}(m_j) + (1-\beta)\frac{\sum_{1}^{n} W_n}{n} \tag{3.1}$$

dengan :

W(m_j) : bobot akhir kemiripan method m_j.

sim(m_j) : nilai kemiripan *method signature* m_j

W_n : bobot kemiripan *method* n. *Method* n adalah method yang dipanggil oleh *method* m_j.

Bobot suatu titik dihitung berdasarkan jumlah nilai kemiripan *method signature* terhadap suatu kebutuhan ditambah dengan rata-rata jumlah bobot titik-titik dibawahnya yang terhubung langsung maupun tidak langsung dengan titik tersebut. Rumus pembobotan suatu titik ditunjukkan pada Persamaan 3.1. Dalam persamaan tersebut ß merupakan koefisien yang membedakan besarnya kontribusi natara *method signature* dengan *method invocation*. Dalam penelitian ini akan digunakan beberapa nilai ß dan dibandingkan hasilnya untuk mencari nilai terbaik ß.

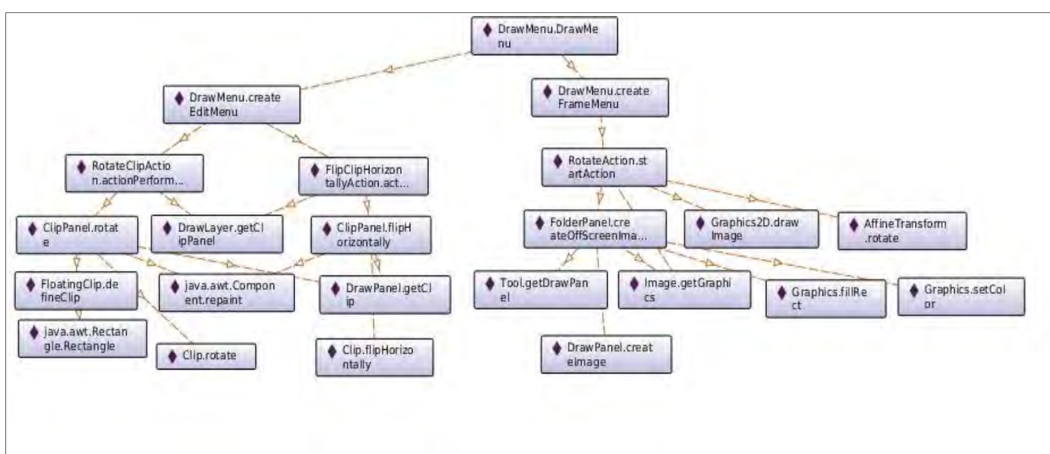
Dalam implementasi suatu *method*, tidak semua *method* yang dipanggil merupakan *method* yang penting atau berhubungan dengan kebutuhan yang diimplementasikan. Salah satu contoh *method* yang tidak berhubungan adalah *writeLog* yang bertujuan untuk menulis log jejak eksekusi. Pemanggilan *method* yang tidak relevan akan menurunkan nilai bobot akhir dari *method* yang memanggilnya, karena itu perlu dihilangkan dari *call graph*. *Method* yang tidak berhubungan memiliki nilai kemiripan sangat kecil terhadap *requirement* yang diimplementasikan *method* pemanggil. Suatu batas minimal nilai kemiripan t_2 diperlukan untuk menentukan apakah suatu *method* relevan atau tidak. *Method* yang tidak relevan dihilangkan dari penelusuran *call graph*. Penelusuran juga tidak dilakukan ke dalam *method* yang berada dalam *library*. *Library* yang direferensi oleh sebuah program berbentuk file biner, tidak berupa kode sumber sehingga tidak bisa dilakukan penelusuran ke dalam method di dalamnya.

3.11. Percobaan Awal

Untuk menngetahui apakah metode yang diusulkan bisa berjalan, dilakukan percobaan awal secara sederhana dengan data yang kecil. Proses percobaan ini dilakukan untuk memberikan gambaran singkat akan jalannya metode yang diusulkan. Studi kasus yang digunakan pada proses percobaan awal ini adalah Jdraw, sebuah aplikasi sederhana untuk mengolah gambar. Jdraw merupakan

aplikasi kode sumber terbuka menggunakan bahasa pemrograman Java yang tersedia di sourceforge. Alamat situs Jdraw adalah http://jdraw.sourceforge.net/.

Di halaman situs web Jdraw disebutkan beberapa fitur yang harus dipenuhi oleh aplikasi ini. Dalam studi kasus ini dipilih satu kebutuhan yang harus dipenuhi oleh Jdraw kemudian dilakukan analisa terhadap kode sumber menggunakan metode yang diusulkan untuk menemukan *method* yang menyelesaikan kebutuhan tersebut serta menghitung hubungan semantik antara *method* tersebut dengan kebutuhan yang dicari. Dalam percobaan ini dipilih fitur yang harus dipenuhi adalah aplikasi menyediakan fitur untuk memutar gambar("*rotate image*"). Langkah pertama dilakukan analisa struktur kode sumber menggunakan AST untuk menghasilkan ontologi hubungan pemanggilan antar *method.* Sebagian hasil ontologi kode sumber yang berhubungan dengan "*rotate image*" ditunjukkan pada Gambar 3.10.



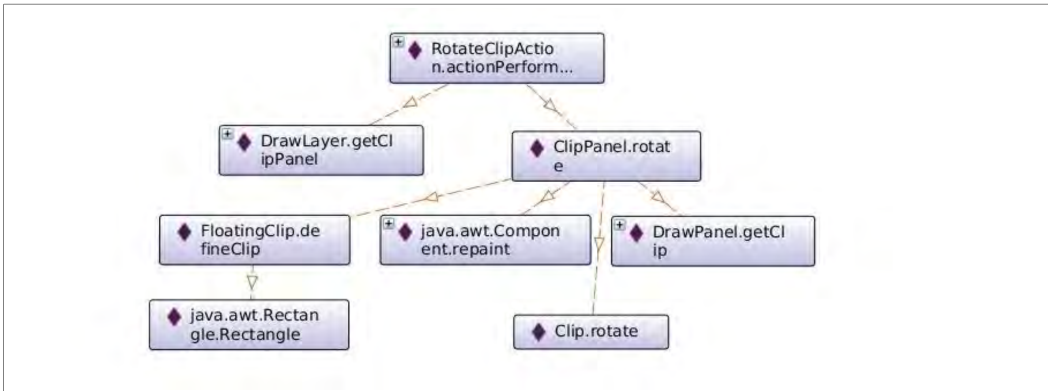
Gambar 3.10. Ontologi Struktur Kode Sumber JDraw

Selanjutnya dilakukan preproses dan ekstraksi dari masing-masing method dan menghitung kemiripan antara setiap *method signature* dengan "rotate image". Langkah preproses dan perhitungan kemiripan menggunakan metode yang sudah dijelaskan sebelumnya. Hasil perhitungan kemiripan ditunjukkan pada Tabel 3.3. Kolom pertama adalah URI *method* dalam ontologi yang mewakili nama *method* dan kolom kedua adalah nilai kemiripan. Dalam percobaan awal inidigunakan ß sebesar 0.5.

Tabel 3.3. Perhitungan Kemiripan *Method Signature* JDraw

Method	Sim
<http://its.ac.id/source_ontology#ClipPanel.rotate>	0.9899
<http://its.ac.id/source_ontology#RotateClipAction.actionPerformed>	0.9693
<http://its.ac.id/source_ontology#RotateAction.startAction>	0.9660
<http://its.ac.id/source_ontology#Clip.rotate>	0.9660
<http://its.ac.id/source_ontology#ClipPanel.flipHorizontally>	0.9559
<http://its.ac.id/source_ontology#FlipClipHorizontallyAction.actionPerformed>	0.9395
<http://its.ac.id/source_ontology#Clip.flipHorizontally>	0.9310
<http://its.ac.id/source_ontology#Graphics2D.drawImage>	0.8700
<http://its.ac.id/source_ontology#Tool.getDrawPanel>	0.8661
<http://its.ac.id/source_ontology#Graphics.setColor>	0.8659
<http://its.ac.id/source_ontology#DrawLayer.getClipPanel>	0.8615
<http://its.ac.id/source_ontology#DrawPanel.getClip>	0.8599
<http://its.ac.id/source_ontology#DrawPanel.createImage>	0.8599
<http://its.ac.id/source_ontology#DrawMenu.createEditMenu>	0.8548
<http://its.ac.id/source_ontology#DrawMenu.createFrameMenu>	0.8515
<http://its.ac.id/source_ontology#DrawMenu.DrawMenu>	0.8340
<http://its.ac.id/source_ontology#FolderPanel.createOffScreenImage>	0.7944
<http://its.ac.id/source_ontology#java.awt.Rectangle.Rectangle>	0.7081
<http://its.ac.id/source_ontology#FloatingClip.defineClip>	0.6594
<http://its.ac.id/source_ontology#AffineTransform.rotate>	0.6166
<http://its.ac.id/source_ontology#java.awt.Component.repaint>	0.5984
<http://its.ac.id/source_ontology#Graphics.fillRect>	0.5600
<http://its.ac.id/source_ontology#Image.getGraphics>	0.3500

Dari perhitungan kemiripan *method signature* dipilih tiga *method* dengan nilai tertinggi untuk dilakukan pembobotan dengan nilai *method invocation*. *Method-method* tersebut adalah `ClipPanel.rotate()`, `RotateClipAction.actionPerformed()` dan `RotateAction.startAction()`. Ontologi hubungan antar *method* dibawah *method* `RotateClipAction.actionPerformed` ditunjukkan pada Gambar 3.11.



Gambar 3.11. Ontologi Struktur *Method* RotateClipAction.actionPerformed

Selanjutnya dihitung nilai rata-rata *method-method* yang dipanggil oleh `ClipPanel.rotate()`. *Method-method* yang dipanggil ditunjukkan dalam Tabel 3.4. Nilai rata-rata yang diperoleh adalah

R_{m1} =3,7918/5

=0,7583

Menggunakan Persamaan 3.1 diperoleh nilai akhir kemiripan sebesar

W_{m1}=(0,5*0,9889)+(1-0,5)*0,7583

=0,8741

Tabel 3.4. Perhitungan Kemiripan Method-Method Dibawah ClipPanel.rotate

<http://its.ac.id/source_ontology#Clip.rotate>	0.9660
<http://its.ac.id/source_ontology#java.awt.Rectangle.Rectangle>	0.7081
<http://its.ac.id/source_ontology#DrawPanel.getClip>	0.8599
<http://its.ac.id/source_ontology#java.awt.Component.repaint>	0.5984
<http://its.ac.id/source_ontology#FloatingClip.defineClip>	0.6594

Selanjutnya dihitung nilai rata-rata method-method yang dipanggil oleh `RotateClipAction.actionPerformed()`. Method-method yang dipanggil ditunjukkan dalam tabel 3.5. Nilai rata-rata yang diperoleh adalah

R_{m1} =5,6432/7

=0,8061

Menggunakan Persamaan 3.1 diperoleh nilai akhir kemiripan sebesar

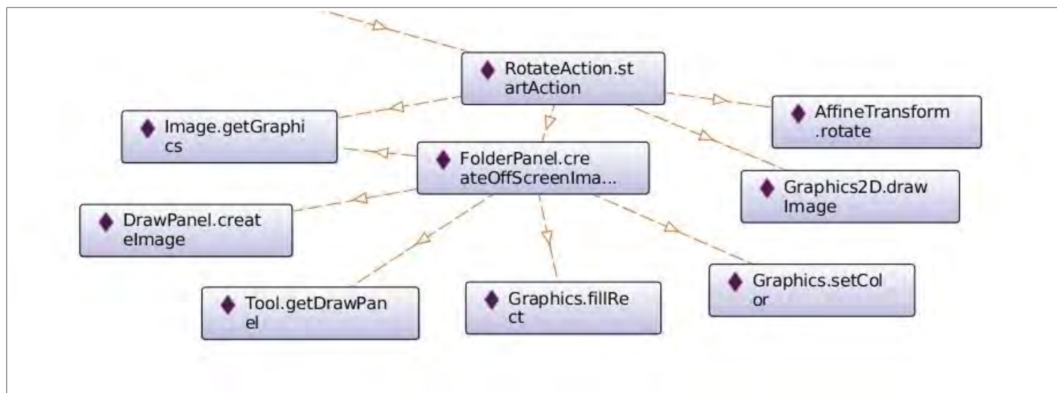
W_{m1}=(0,5*0,9693)+(1-0,5)*0,8061

=0,8877

Tabel 3.5. Perhitungan Kemiripan Method-Method Dibawah `RotateClipAction.actionPerformed`

<http://its.ac.id/source_ontology#ClipPanel.rotate>	0.9899
<http://its.ac.id/source_ontology#Clip.rotate>	0.9660
<http://its.ac.id/source_ontology#DrawPanel.getClip>	0.8599
<http://its.ac.id/source_ontology#DrawLayer.getClipPanel>	0.8615
<http://its.ac.id/source_ontology#java.awt.Rectangle.Rectangle>	0.7081
<http://its.ac.id/source_ontology#java.awt.Component.repaint>	0.5984
<http://its.ac.id/source_ontology#FloatingClip.defineClip>	0.6594

Ontologi hubungan antar *method* dibawah RotateAction.startAction ditunjukkan pada Gambar 3.12. Pada gambar tersebut ditunjukkan ada 8 *method* dibawah `RotateAction.startAction` yang dieksekusi.



Gambar 3.12. Ontologi Struktur *Method* `RotateAction.startAction`

Selanjutnya dihitung nilai rata-rata *method-method* yang dipanggil oleh RotateAction`.startAction().` Percobaan awal ini menggunakan nilai batas minimal kemiripan(t_2) sebesar 0.6. Dalam perhitungan ini terdapat dua *method* yang dihilangkan karena memiliki nilai kemiripan dibawah 0,6 yaitu `Image.getGraphics` dan `Graphics.fillRect`. *Method-method* yang dipanggil ditunjukkan dalam tabel 3.6. Nilai rata-rata yang diperoleh adalah

R_{m1} =4,8729/6

=0,8121

Menggunakan Persamaan 3.1 diperoleh nilai akhir kemiripan sebesar

W_{m1}=(0,5*0,9660)+(1-0,5)*0,8121

=0,8890

Tabel 3.6. Perhitungan Kemiripan *Method-Method* Dibawah RotateAction.startAction

<http://its.ac.id/source_ontology#FolderPanel.createOffScreenImage>	0.7944
<http://its.ac.id/source_ontology#Graphics2D.drawImage>	0.8700
<http://its.ac.id/source_ontology#Tool.getDrawPanel>	0.8661
<http://its.ac.id/source_ontology#Graphics.setColor>	0.8659
<http://its.ac.id/source_ontology#DrawPanel.createImage>	0.8599
<http://its.ac.id/source_ontology#AffineTransform.rotate>	0.6166

Hasil akhir perhitungan *method signature* dan *method invocation* ditunjukkan pada Tabel 3.7. Dari hasil akhir diperoleh nilai akhir method `RotateClipAction.actionPerformed` lebih besar dari `ClipPanel.rotate` dimana pada penilaian kemiripan berdasar *method signature* nilai kemiripan `ClipPanel.rotate` yang lebih besar. Ini menunjukkan bahwa penelusuran *method invocation* memberikan kontribusi dalam menghitung kemiripan antara kebutuhan dan *method*.

Tabel 3.7. Hasil Akhir Perhitungan Kemiripan

Method	Sim
RotateAction.startAction	0.8890
RotateClipAction.actionPerformed	0.8877
ClipPanel.rotate	0.8741

BAB 4
HASIL PENELITIAN DAN PEMBAHASAN

4.1. Pengumpulan Dataset

Untuk melakukan uji coba terhadap metode yang diusulkan diperlukan dataset. Dataset untuk uji coba terdiri dari kode sumber dalam bahasa pemrograman java dan kebutuhan fungsional dari perangkat lunak tersebut. Perangkat lunak yang akan dijadikan dataset dipilih perangkat lunak kode sumber terbuka. Kode sumber diperoleh dari beberapa repositori kode sumber di internet. Kode sumber yang digunakan dalam dataset ini hanya kode sumber perangkat lunak tanpa kode sumber *library* yang diperlukan untuk meng-*compile* kode sumber tersebut. Dari kode sumber yang diperoleh akan dilakukan pemrosesan untuk mendapatkan semua *method* dan hubungan antar *method* dalam perangkat lunak tersebut.

Penelusuran yang akan dilakukan dalam penelitian ini adalah untuk menemukan hubungan antara *method* dalam kode sumber dengan kebutuhan fungsional. Dataset berupa kebutuhan fungsional dibuat oleh pakar. Pakar melakukan analisa terhadap kode sumber untuk menentukan kebutuhan fungsional apa saja yang bisa dipenuhi oleh perangkat lunak. Untuk membuat kebutuhan fungsional, pakar juga mengambil informasi dari website perangkat lunak yang menampilkan fitur-fitur yang bisa dilakukan oleh perangkat lunak tersebut. Kebutuhan fungsional yang dihasilkan terdiri dari satu atau lebih kalimat bahasa alami yang mendeskripsikan fungsi yang bisa dilakukan oleh perangkat lunak.

Tabel 4.1. Informasi Kode Sumber

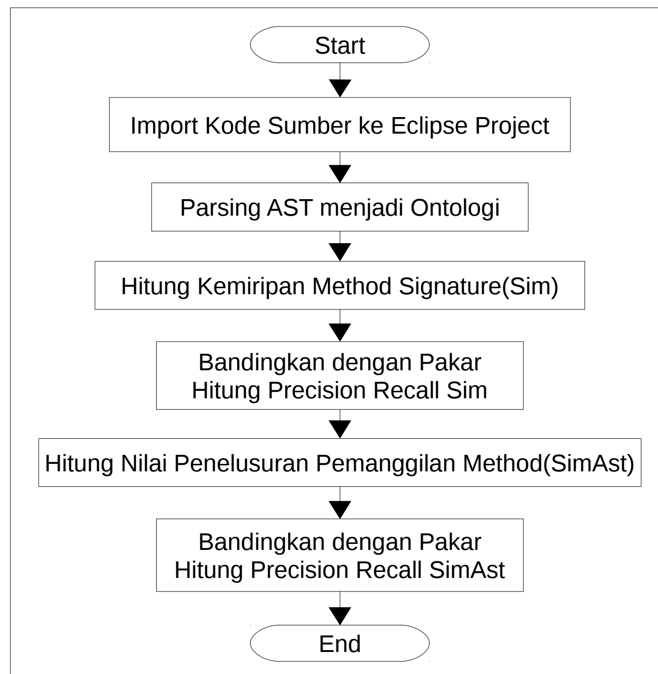
Nama Aplikasi	Jml. method	Respositori
Itrust	2698	https://sourceforge.net/projects/itrust/
Gantt Project	5292	https://github.com/bardsoftware/ganttproject

4.2. Skenario Pengujian

Proses pengujian dilakukan dengan mengimplementasikan alur kerja yang diusulkan dan dijelaskan pada metodologi. Alur kerja tersebut diimplementasikan pada dataset kemudian membandingkan hasil identifikasi sistem dengan hasil dari pakar. Dalam pengujian ini dilakukan dua perhitungan kemiripan sebagai berikut :

1. Sim adalah perhitungan kemiripan antara kalimat kebutuhan fungsional dengan *method signature*. Dilakukan preproses dan parsing terhadap setiap *method signature* dalam kode sumber dan kalimat kebutuhan fungsional. Kemudian dihitung nilai kemiripan antara keduanya menggunakan Persamaan 2.4.
2. SimAst adalah perhitungan kemiripan antara kebutuhan fungsional dengan hasil penelusuran pemanggilan *method(method invocation).* Dilakukan penelusuran terhadap semua *method* yang dipanggil oleh *method* utama yang dihasilkan oleh Sim dan *method*-*method* dibawahnya. Kemudian dihitung rata-rata dari total nilai kemiripan dari semua *method* yang diperoleh dari hasil penelusuran tersebut kemudian dihitung nilai kemiripan akhir menggunakan Persamaan 3.1.

Langkah-langkah pengujian ditunjukkan pada Gambar 4.1. Langkah pertama adalah mengunduh kode sumber dari repositori kemudian mengimport kode sumber ke dalam proyek eclipse. Berikutnya membentuk *abstract syntax tree*(AST) dari proyek tersebut. Kemudian dilakukan penelusuran pada AST untuk membentuk individu-individu dan menyimpannya dalam ontologi.



Gambar 4.1. Alur Skenario Pengujian

Sebagai contoh, suatu kebutuhan fungsional dengan kalimat *"An admin can update the hospital listing information. Admin may add new hospital and update the hospital name"*. Preproses dan POS tagger akan menghasilkan :

Kalimat 1 = An admin can update the hospital listing information

Kata Kerja = "update"

Kata Benda = "hospital", "list"

Kalimat 2 = Admin may add new hospital and update the hospital name

Kata Kerja = "add", "update"

Kata Benda = "hospital"

Salah satu *method* yang menjalankan kebutuhan ini adalah `UpdateHospitalListAction.addHospital`. Parsing pada *method* ini menggunakan aturan yang diusulkan dan menggunakan Persamaan 2.4 untuk menghitung Sim terhadap kalimat 2 sebagai berikut :

Method Signature = `UpdateHospitalListAction.addHospital`

Kata Kerja = "add", "update"

Kata Benda = "hospital", "list"

Sim_{Verb} = 0.99

Sim_{Noun} = 0.49

Sim_{total} = 0.75

Kemiripan *method* `addHospital` terhadap kalimat 1 sebesar 0.48. Jika kebutuhan fungsional terdiri dari lebih dari satu kalimat, maka akan dihitung nilai Sim terhadap masing-masing kalimat tetapi yang digunakan hanya nilai Sim yang tertinggi. Method yang ada dibawah `addHospital` sebanyak 9 method. Nilai kemiripan *method-method* yang dipanggil oleh *method* `addHospital` ditunjukkan pada Tabel 4.2.

Tabel 4.2. Nilai Kemiripan Method-method yang Dipanggil oleh `UpdateHospitalListAction.addHospital`

Nama Method	Kata Kerja	Kata Benda	Sim
ItrustException.getMessage	get	exception, message	0.45
HospitalBean.getHospitalName	get	hospital	0.95
HospitalBean.getHospitalID	get	hospital, id	0.7
HospitalsDAO.addHospital	add	hospital	0.93
HospitalBeanValidator.validate	validate	hospital	0.857
HospitalBean.getHospitalState	get	hospital, state	0.7
HospitalBean.getHospitalCity	get	hospital, city	0.7
HospitalBean.getHospitalAddress	get	hospital, address	0.7
HospitalBean.getHospitalZip	get	hospital, zip	0.7

$Sim_{rata\text{-}rata}$ = 6.237/8

= 0.77

SimAst = (0.79x0.9)+(0.1x0.77)

= 0.752

Method `ItrustException.getMessage` tidak diikutsertakan dalam menghitung SimAst karena kemiripannya dibawah 0.5. Nilai akhir kemiripan `UpdateHospitalListAction.addHospital` adalah 0.752, meningkat sebesar 0.002. Setelah dihitung nilai Sim dan SimAst setiap *method* dalam kode sumber, kemudian dibandingkan dengan hasil penelusuran yang dilakukan pakar untuk menghitung nilai *precision* dan *recall.*

Dalam penelitian ini dikembangkan plugin eclipse untuk membantu mempercepat proses pengujian. Plugin dikembangkan untuk membentuk dan

melakukan penelusuran terhadap AST dan membentuk individu-individu ontologi. Proses penghitungan kemiripan dalam penelitian ini menggunakan kata benda dan kata kerja dalam *method signature* dan kalimat kebutuhan fungsional. Karenanya perlu dilakukan parsing terhadap *method signature* dan kebutuhan fungsional untuk mendapatkan semua kata benda dan kata kerja didalamnya. Dalam penelusuran AST, setiap ASTNode berjenis deklarasi *method* akan dibuat satu individu ontologi. Kemudian dilakukan parsing terhadap *method signature* menggunakan aturan yang diusulkan alam penelitian ini. Hasil parsing adalah kumpulan kata kerja dan kata benda. Kata kerja dan kata benda disimpan dalam atribut individu ontologi. Kata kerja disimpan dalam atribut bernama *verb*, sedangkan kata benda disimpan dalam atribut bernama *noun*. Ontologi yang terbentuk dari proses penelusuran AST akan disimpan dalam file dengan format *ontology web language* (owl).

Kemudian dilakukan penghitungan nilai kemiripan antara setiap individu yang berjenis deklarasi *method* dalam ontologi dengan kalimat kebutuhan fungsional. Sebelumnya perlu dilakukan preproses pada kalimat kebutuhan fungsional. Pertama dilakukan lemmatisasi dan POS Tagger pada kebutuhan fungsional. Lemmatisasi untuk mengembalikan setiap kata dalam kalimat kebutuhan fungsional menjadi kata dasar. Setelah dilakukan POS tagger kemudian dipisahkan antara kata kerja dan kata benda dalam setiap kalimat dalam kebutuhan fungsional. Selanjutnya dilakukan perhitungan kemiripan antara semua individu berjenis *method* dalam ontologi dan kebutuhan fungsional. Kemiripan antara kata benda dan kata kerja dari hasil parsing kebutuhan fungsional dengan *method signature* dihitung menggunakan Persamaan 2.4. Langkah berikutnya adalah membandingkan hasil penelusuran dari sistem dengan hasil penelusuran yang dilakukan pakar untuk menghitung nilai *precision* dan *recall*. Dihitung nilai *precision* dan *recall* untuk titik potong kemiripan 0.7 sampai 0.95 dengan interval 0.5.

Proses berikutnya adalah melakukan penelusuran *method invocation*. Penelusuran dilakukan dari individu-individu ontologi yang memiliki nilai

kemiripan diatas titik potong 0.7 sebagai *method* utama. Penelusuran *method invocation* dilakukan menggunakan algoritma *breadth first search*. Penelusuran dilakukan terhadap *method* yang dipanggil oleh *method* utama dan semua *method* dibawahnya sampai terakhir tidak ada *method* lagi yang dipanggil. Penelusuran ini hanya pada *method* yang ada pada kode sumber tanpa menelusuri *method* yang ada di luar kode sumber seperti *method* dalam *library* atau *method* dalam SDK java. Dalam peneulusuran ini hanya method yang memiliki nilai kemiripan diatas 0,5 yang akan digunakan. Suatu *method* tidak hanya memanggil *method* yang bertujuan sesuai dengan kebutuhan fungsionalnya, misalkan suatu *method* memanggil *method* mencatat log untuk keperluan proses pencarian kesalahan(*debug*) bernama `Logger.debug`. Method `debug()` tidak berhubungan dengan kebutuhan fungsional dari method yang memanggilnya. Proses berikutnya adalah menjumlahkan total nilai kemiripan dari semua method yang terlibat dalam penelusuran kemudian dihitung nlai rata-rata dan menggunakan Persamaan 3.1 dihitung nilai akhir kemiripannya. Potongan proses penelusuran *method invocation* ditunjukkan pada Gambar 4.2.

Pengujian juga dilakukan menggunakan metode yang diusulkan pada penelitian sebelumnya yaitu menggunakan *Latent Semantic Analysis*(LSA) (Marcus,2003) dan *Latent Dirichlet Allocation*(LDA)(Oliveto,2010). Hasil pengujian dengan LSA dan LDA akan dibandingkan dengan hasil pengujian menggunakan metode yang diusulkan. Dari perbandingan ini bisa diketahui apakah metode yang diusulkan lebih baik dari penelitian sebelumnya.

Pengujian menggunakan LSA dan LDA dilakukan terhadap dataset yang sama dengan pengujian menggunakan metode yang diusulkan. LSA dan LDA menghitung kemiripan antar dokumen untuk menemukan *traceability* antara kebutuhan dengan kode sumber. Dokumen yang digunakan berasal dari setiap *method* dalam kode sumber. Langkah pertama melakukan parsing setiap *method* menjadi satu file text. Langkah berikutnya melakukan pemisahan kata, menghilangkan *stopwords* dan *stemming*. Dalam proses ini digunakan *stemming* bukan lemmatisasi karena dalam proses *information retrieval*(IR) tidak

menggunakan Wordnet, tidak perlu menggunakan term yang memiliki makna. Yang diperlukan hanya kemunculan term yang sama dalam dokumen-dokumen. Dokumen hasil preproses dihitung kedekatannya dengan kalimat kebutuhan fungsional dengan metode LSA dan LDA.

Dalam proses penghitungan kemiripan dengan LSA, dibentuk matriks dokumen dan term. Dokumen pada kolom matriks adalah setiap *method* dalam kode sumber. Kemudian dihitung *singular value decompisition*(SVD) dari matriks

```
public Set<OWLNamedIndividual> bfs(OWLNamedIndividual
individu,Double tresholdBawah) {
  Set<OWLNamedIndividual> item = new HashSet<>();
  Queue<OWLNamedIndividual> queue = new
  LinkedList<OWLNamedIndividual>();
  queue.add(individu);
  while (!queue.isEmpty()) {
    OWLNamedIndividual node = queue.remove();
    NodeSet<OWLNamedIndividual> individus = reasoner
    .getObjectPropertyValues(node, invoke);
    for (OWLNamedIndividual child : individus.getFlattened())
    {
      Set<OWLLiteral> simInvoke =
      reasoner.getDataPropertyValues(child, simSig);
   if (simInvoke.iterator().hasNext()) {
     Double nilaiInvoke = Double.parseDouble(simInvoke
     .iterator().next().getLiteral());
     if (nilaiInvoke >= tresholdBawah) {
       if (!item.contains(child)) {
       queue.add(child);
       item.add(child);
     }
      }}}}
  return item;
}
```

Gambar 4.2. Kode Sumber Penelusuran *Method Invocation* Dengan *Breadth First Search*

tersebut. Berikutnya dibuat matriks term dari masing-masing kebutuhan fungsional. Matriks kebutuhan dikonversi kedalam SVD *space* kemudian dihitung jarak antara matriks kebutuhan fungsional dengan matriks *term* dokumen dalam SVD *space*. Langkah berikutnya adalah membandingkan hasil LSA dengan hasil penelusuran yang dilakukan pakar untuk menghitung nilai *precision* dan *recall*. Berikutnya dihitung nilai *precision* dan *recall* untuk titik potong kemiripan 0.7 sampai 0.95 dengan interval 0.5.

Proses perhitungan kemiripan dengan LDA dilakukan menggunakan mallet(McCallum, 2012). Mallet merupakan perangkat lunak kode sumber terbuka yang mengimplementasikan LDA. Langkah pertama adalah membuat dokumen dari setiap *method* dalam kode sumber dan melakukan *stemming* dan penghilangan *stopwords*. Kemudian membentuk topik model dari dokumen hasil praproses kode sumber. Langkah ini akan menghasilkan *inferencer*. Kemudian lakukan praproses pada kebutuhan fungsional dengan menghilangkan *stopwords* dan *stemming*. Langkah berikutnya mengimport kebutuhan fungsional ke dalam topik model. Kemudian dilakukan inferensi dengan *inferencer* yang dihasilkan pada langkah sebelumnya. Hasilnya adalah matriks sebaran proporsi topik. Selanjutnya dihitung jarak *cosine* antara kebutuhan fungsional dan *method* dalam matriks sebaran topik hasil inferensi. Dalam pengujian ini dilakukan beberapa kali percobaan menggunakan mallet dengan parameter jumlah iterasi 100, 1000, 2000, 3000, 4000 dan jumlah topik 10, 50, 100, 200, 400. Kemudian dibandingkan hasil LDA dengan hasil penelusuran yang dilakukan pakar untuk menghitung nilai *precision* dan *recall*. Kemudian dihitung nilai *precision* dan *recall* untuk titik potong kemiripan 0.7 sampai 0.95 dengan interval 0.5.

4.3. Pengembangan Plugin Eclipse

Dalam rangka mempercepat proses pengujian, dibuat suatu alat bantu yang berfungsi melakukan parsing terhadap kode sumber menjadi ontologi. Alat bantu yang dikembangkan berupa eclipse plugin. Plugin dikembangkan menggunakan basis *Eclipse Java Development Tools*(JDT). Eclipse JDT menyediakan API untuk mengakses dan memanipulasi kode sumber melalui Java Model dan *Abstract*

Syntax Tree(AST). Plugin dikembangkan menggunakan eclipse karena JDT memiliki fitur *incremental compiler* dan *resolve binding*. *Incremental compiler* merupakan suatu fitur yang memungkinkan melakukan kompilasi terhadap sebagian dari kode sumber. Kode sumber bisa dilakukan kompilasi meskipun masih mengandung kesalahan seperti tidak adanya *library* yang diperlukan oleh kode sumber. Fitur ini sangat penting karena dalam penelitian ini kode sumber atau perangkat lunak pendukung tidak diikutsertakan dalam dataset. *Resolve Binding* adalah fitur untuk menemukan tipe atau jenis objek suatu variabel dalam kode sumber. Fitur ini berguna untuk menemukan *class* dan *method* apa saja yang dipanggil dalam suatu *method*.

Eclipse plugin diimplementasikan oleh class `ParseHandler` yang merupakan turunan dari class `AbstractHandler` dan meng-*override method* `execute`. AST yang dihasilkan merupakan suatu *tree* yang merepresentasikan detail dari kode sumber. Dalam AST, setiap file kode sumber java direpresentasikan sebagai *subclass* `ASTNode`. Setiap titik pada AST mewakili satu jenis elemen secara spesifik dalam kode sumber. Misalnya class `MethodDeclaration` mewakili suatu pendeklarasian *method* dalam suatu class. Class `MethodInvocation` mewakili suatu pendeklarasian pemanggilan suatu *method* dalam *method* lain. `PackageDeclaration` mewakili pendeklarasian nama *package* dalam kode sumber.

Proses penelusuran AST menggunakan pola desain *visitor*. Class `Visitor` akan mennelusuri setiap titik pada AST. Class `ParseVisitor` merupakan turunan dari class `ASTVisitor`. Karena yang diperlukan hanya pendeklarasian *method* dan *method invocation*, maka hanya akan meng-*override Method* `visit(MethodDeclaration node)` dan `visit(MethodInvocation node)`. Method `visit(MethodDeclaration node)` akan dieksekusi pada saat mengunjungi titik berjenis deklarasi *method* dalam AST. Dalam *method* ini akan diinstansiasi suatu objek *method* dengan atribut nama, jenis kembalian dan parameter dalam deklarasi *method*. Kode sumber penelusuran deklarasi *method* ditunjukkan pada Gambar 4.3.

```
public boolean visit(MethodDeclaration node) {
  nodeName = node.getName().getFullyQualifiedName();
  String methodName = node.getName().getFullyQualifiedName();
  metode = new Metode(kelas.getId() + "." +
  methodName).setNama(methodName);
  if (node.getReturnType2() != null) {
    String result = node.getReturnType2().toString();
    metode.setResult(result);}
  List<SingleVariableDeclaration> params = node.parameters();
  List<String> pars = new ArrayList<String>();
  for (SingleVariableDeclaration param : params) {
    String parName = param.getName().toString();
    String parType = param.getStructuralProperty(
    SingleVariableDeclaration.TYPE_PROPERTY).toString();
    IVariableBinding vb = param.resolveBinding();
    parType = vb.getType().getQualifiedName();
    String par = parName + " " + parType;
    pars.add(par);
  }
  metode.setPars(pars);
  kelas.addMetode(metode);
  return super.visit(node);
}
```

Gambar 4.3. Kode Sumber Penelusuran Deklarasi *Method*

Method `visit(MethodInvocation node)` akan dieksekusi pada saat mengunjungi titik berjenis pemanggilan method dalam suatu *method*. Dalam proses ini dilakukan *resolve binding* terhadap *method* yang dipanggil untuk mengetahui dari *class* apa *method* tersebut berasal karena akan dibuat obyek properti *invoke* dalam ontologi. Obyek properti invoke memiliki sumber dan tujuan individu berjenis *method*. Kode sumber penelusuran pemanggilan *method* ditunjukkan pada Gambar 4.4.

```
public boolean visit(MethodInvocation node) {
  IMethodBinding mb = node.resolveMethodBinding();
  if (mb != null) {
    String invoType =
    mb.getDeclaringClass().getQualifiedName();
    String invoName = mb.getName();
    String call = invoType + "." + invoName;
    call = call.replace("<", ".").replace(">",
    ".").replace("...", ".").replace("..", ".");
    if (metode != null) {metode.addCall(call);}
    } else {
    Expression exp = node.getExpression();
    if (exp != null) {
      ITypeBinding typeBinding = exp.resolveTypeBinding();
      if (typeBinding != null) {
        String invoType = typeBinding.getQualifiedName();
        String invoName = node.getName().toString();
        String call = invoType + "." + invoName;
        call = call.replace("<", ".").replace(">",
        ".").replace("..", ".");
        if (metode != null) { metode.addCall(call);}
    }}}
    return super.visit(node);
}
```

Gambar 4.4. Kode Sumber Penelusuran Pemanggilan *Method*

4.4. Pengujian ITrust

ITrust adalah aplikasi medis yang memiliki fitur antara lain memberikan fasilitas pada pasien untuk memantau riwayat kesehatan, memilih dokter dan berkomunikasi dengan dokter melalui perpesanan ataupun email. Itrust juga memberikan fasilitas bagi staff medis untuk menjadwalkan kunjungan ke pasien, mengelola diagnosa penyakit dan resep pasien. ITrust ditulis menggunakan bahasa pemrograman java dan bersifat kode terbuka. Kode sumber itrust versi 21.0 dapat diunduh dari situs sourceforge pada alamat https://sourceforge.net/projects/itrust/.

Kode sumber versi 21.0 adalah kode sumber terakhir yang dapat diunduh dengan bebas dari sourceforge. Versi lebih baru dapat diunduh di https://github.ncsu.edu/engr-csc326-staff/iTrust-v21. Kode sumber ITrust terdiri dari 365 class dengan jumlah method sebanyak 2698 method dan 10101 *method invocation*. Kalimat kebutuhan fungsional yang digunakan dalam pengujian ini adalah sebagai berikut :

1. An HCP is able to add new patient. The create patients and HCP transaction is logged.
2. Admin add new LHCP, PHA and LT personnel. Data for personnel can be edited compatible to specified format
3. Patient may edit demographic information. an email will be sent after the data is stored. Patient can update security question and answer
4. Patient view their access log. The patient may view access log of a person for whom they are personal health representative.
5. A patient can view medical records. A health representative may view represented patient's medical records. they may view all health records and allergies. he or she may view office visit record.
6. An HCP may add or edit an office visit for a patient
7. The HCP can add another user as a representative to that patient. The HCP can remove patient's representative.
8. An LHCP identify the chronic disease risk. LHCP may access a patient's health record. Currently defined risk factors for for heart disease. Also recognized the risk factors for chronic diseases types of diabetes type 1 and type 2.
9. An HCP got a reminder for patients visit. HCP receive reminders for immunization needer.
10. An admin can update the hospital listing information. Admin may add new hospital and update the hospital name.
11. HCP can view prescription report. If the LHCP is not one of the patient's DLHCP or the UAP associated, an email is sent to the patient and their

personal representative.

12. LHCP can send a message or email to patient or other HCP. LHCP can view their message order by time or name.
13. An HCP may renew the patient's expired prescriptions.
14. A sending HCP refers a patient to another receiving HCP. A sending HCP edits a previously sent referral. A sending HCP cancels a previously sent patient referral.
15. An HCP or UAP creates a list of patients for which he will monitor remotely. HCP or UAP may add or remove a patient to remote monitoring list. He can view a patient's data from remote monitoring list.
16. Admin may create or delete a drug interaction between two drugs.

Pengujian pertama adalah mencari nilai kemiripan antara kode sumber dengan kebutuhan fungsional menggunakan metode wu-palmer dan Persamaan 2.4. Dengan menggunakan nilai alpha sebesar 0.4, 0.5 dan 0.6. Nilai *precision* hasil pengujian dengan nilai alpha 0.5 ditampilkan pada Tabel 4.3 dan nilai *recall* hasil pengujian ditampilkan pada Tabel 4.4.

Tabel 4.3. Nilai *Precision* Hasil Pengujian Kemiripan *Method Signature(Sim)* dan Setiap Kebutuhan Fungsional Itrust

kebutuhan fungsional	treshold					
	0.7	0.75	0.8	0.85	0.9	0.95
1	0.137	0.175	0.2	0.2	0.219	0.545
2	0.423	0.5	0.5	0.5	0.524	0.875
3	0.081	0.152	0.071	0.333	0.333	1
4	0.227	0.222	0.167	0	0	0
5	0.09	0.123	0.154	0.267	0.333	0
6	0.229	0.229	0.444	0.333	0	0
7	0.5	0.667	1	1	1	1
8	0.053	0.185	0.333	0.6	0.667	1
9	0.027	0.032	0.065	0.222	0.667	0.667
10	0.19	0.444	0.286	0.286	0.333	0
11	0.03	0.053	0.111	0.2	0.25	1
12	0.1	0.106	0.159	0.212	0.091	0.182
13	0.25	0.429	0.6	0.6	0.667	1
14	0.088	0.136	0.14	0.153	0.164	0.27
15	0.167	0.222	0.333	0.429	0.5	0
16	0.889	0.875	1	1	1	0
rata-rata	0.218	0.284	0.348	0.396	0.422	0.471

Tabel 4.4. Nilai *Recall* Hasil Pengujian Kemiripan *Method Signature(Sim)* terhadap Setiap Kebutuhan Fungsional Itrust

kebutuhan fungsional	treshold					
	0.7	0.75	0.8	0.85	0.9	0.95
1	0.778	0.778	0.778	0.778	0.778	0.667
2	0.647	0.647	0.647	0.647	0.647	0.412
3	0.438	0.313	0.063	0.063	0.063	0.063
4	0.714	0.286	0.143	0	0	0
5	0.867	0.667	0.4	0.267	0.267	0
6	0.857	0.571	0.286	0.036	0	0
7	0.4	0.4	0.4	0.2	0.2	0.2
8	0.625	0.625	0.5	0.375	0.25	0.25
9	0.429	0.286	0.286	0.286	0.286	0.286
10	0.667	0.667	0.333	0.333	0.333	0
11	0.4	0.4	0.2	0.2	0.2	0.2
12	0.816	0.579	0.474	0.368	0.079	0.053
13	0.429	0.429	0.429	0.429	0.286	0.143
14	0.824	0.706	0.706	0.647	0.588	0.588
15	0.588	0.471	0.353	0.353	0.176	0
16	0.727	0.636	0.636	0.636	0.636	0
rata-rata	0.638	0.529	0.415	0.351	0.299	0.179

Pada kebutuhan fungsional pertama dengan titik potong 0.8 terdapat 35 *method* yang terpilih tetapi hanya 7 *method* yang benar merupakan *traceability link*. Banyak *method* yang tidak berhubungan dengan kebutuhan fungsional tetapi memiliki nilai kemiripan tinggi, lebih besar dari 0.8. Kalimat yang digunakan pada kebutuhan fungsional pertama adalah "*An HCP is able to add new patient*". Menggunakan POS tagger pada kalimat tersebut akan didapatkan kata kerja "*add*" dan kata benda "*patient*". Jarak kemiripan antara kalimat tersebut dengan *method* `PatientDAO.getAllPatient` adalah 0,926. Parsing menggunakan metode yang diusulkan dalam penelitian ini pada *method* `PatientDAO.getAllPatient`, maka akan didapatkan kata kerja "*get*" dan kata benda "*patient*". Dengan menggunakan metode wu-palmer akan didapatkan nilai kemiripan antara *get* dan *add* adalah 0.666. Struktur Wordnet dari kata *get* dan *add* ditunjukkan pada Gambar 4.5. Dari gambar tersebut bisa dihitung jarak antar kata "*add*" dan "*get*" menggunakan Persamaan 2.1 sebagai berikut :

N3 = depth(change#v#1) = 2

N1 = min(depth({tree in T1 | tree contains LCS })) = 3

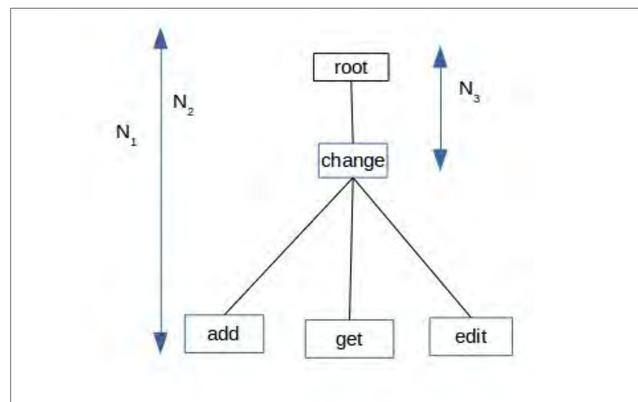
N2 = min(depth({tree in T2 | tree contains LCS })) = 3

Sim(c1,c2) = 2 * N3 / (N1 + N2)

= 2 * 2 / (3 + 3)

= 0.666

Dengan menggunakan Persamaan 2.2, maka jarak *cosine* antara *get* dan *add* adalah 0.852. Sedangkan kata bendanya adalah *patient*, nilai kemiripan kata bendanya adalah 1. Dengan menggunakan Persamaan 2.4 akan dihasilkan nilai kemiripan sebesar 0.926. Method `PatientDAO.getAllPatient` merupakan *method* yang bertujuan untuk mengambil data seluruh pasien, bukan *method* yang dijalankan dan tidak dipanggil oleh *method* lain dalam memenuhi kebutuhan menambah pasien. Hal ini menyebabkan kecilnya nilai *precision* dan *recall*.



Gambar 4.5. Struktur *add*, *get* dan *edit* dalam Wordnet

Menggunakan cara yang sama didapatkan nilai kemiripan antara kalimat kebutuhan fungsional pertama dengan method `PatientDAO.editPatient` sebesar 0.926. Karena posisi kata "*edit*" dalam struktur Wordnet sama dengan kata "*get*" maka besarnya kemiripan antara *add* dan *edit* adalah 0.666. Dengan menggunakan Persamaan 2.4 akan dihasilkan nilai kemiripan antara *method* dengan kebutuhan fungsional sebesar 0.926. Method `PatientDAO.editPatient` merupakan *method* yang bertujuan untuk mengubah data seorang pasien, bukan *method* yang dijalankan dan tidak dipanggil oleh *method* lain dalam memenuhi kebutuhan menambah pasien.

Pada kebutuhan fungsional keempat tidak ada *method* yamg memiliki kemiripan diatas 0.85. Kalimat pada kebutuhan fungsional keempat adalah "*Patient view their access log*". Menggunakan POS tagger akan dihasilkan kata kerja "*view*" dan kata benda "*access*", "*log*", "*patient*". Kata benda "*patient*" disini merupakan subjek dari kalimat. Method yang menjalankan kebutuhan ini adalah `ViewMyAccessLogAction.getAccesses`. Parsing pada *method signature* akan menghasilkan kata kerja "*view*" dan kata benda "*access*", "*log*". Nilai kemiripan antara *method* tersebut dengan kalimat kebutuhan fungsional adalah 0.755. Proses parsing *method* dalam penelitian ini menghasilkan kata kerja dan kata benda yang merupakan objek langsung(*Direct Object*) dari kata kerja tersebut. Jika subjek tidak diikutsertakan dalam perhitungan kemiripan, maka jarak antara kebutuhan fungsional dan *method* adalah 0.922. Ini akan meningkatkan nilai precision dan recall.

Pada kebutuhan fungsional keenam tidak ada *method* yamg memiliki kemiripan diatas 0.9. Kalimat pada kebutuhan fungsional keenam adalah "*An HCP may add or edit an office visit for a patient*". Menggunakan POS tagger akan dihasilkan kata kerja "*add*", "*edit*" dan kata benda "*office*", "*visit*", "*patient*". Kata benda "*patient*" disini merupakan kata pelengkap atau objek tidak langsung(*Indirect Object*) dari kalimat. Salah satu *method* yang menjalankan kebutuhan ini adalah `AddOfficeVisitAction.addEmptyOfficeVisit`. Parsing pada *method signature* akan menghasilkan kata kerja "*add*" dan kata benda "*office*", "*visit*". Nilai kemiripan antara *method* tersebut dengan kalimat kebutuhan fungsional adalah 0.814. Proses parsing method penelitian ini menghasilkan kata kerja dan kata benda yang merupakan objek langsung(*Direct Object*) dari kata kerja tersebut. Jika objek tidak langsung(*Indirect Object*) tidak diikutsertakan dalam perhitungan kemiripan, maka jarak antara kebutuhan fungsional dan *method* adalah 0.98. Ini akan meningkatkan nilai *precision* dan *recall*.

Langkah pengujian selanjutnya adalah menggunakan penelusuran terhadap *method* yang dipanggil oleh *method-method* pada pengujian langkah sebelumnya.

Menggunakan Persamaan 3.3 dengan nilai beta 0.9 dihasilkan nilai *precision* seperti yang ditunjukkan pada Tabel 4.5 dan nilai *recall* ditunjukkan pada Tabel 4.6. Pada proses penelusuran pemanggilan *method* kebutuhan fungsional kesebelas, nilai *precision* meningkat dari 0.25 menjadi 1. Pada proses perhitungan kemiripan *method signature*, terdapat empat *method* yang memiliki kemiripan diatas 0.9 tetapi yang benar hanya satu. Setelah proses penelusuran pemanggilan *method*, nilai kemiripan tiga *method* yang salah tersebut menurun karena memanggil *method*-*method* lain yang nilai kemiripannya rendah sehingga hanya tinggal satu *method* yang benar merupakan link penelusuran yang memiliki nilai kemiripan sebesar 0.94. Nilai *precision* menurun pada titik potong 0.95 karena banyak *method* yang dipanggil memiliki nilai kemiripan dibawah 0.95 sehingga menurunkan nilai akhir kemiripan.

Tabel 4.5. Nilai *Precision* Hasil Pengujian Kemiripan Penelusuran Pemanggilan *Method*(SimAst) dan Setiap Kebutuhan Fungsional Itrust

kebutuhan fungsional	treshold					
	0.7	0.75	0.8	0.85	0.9	0.95
1	0.137	0.175	0.2	0.2	0.219	0.545
2	0.423	0.458	0.5	0.5	0.524	0.875
3	0.083	0.107	0.083	0.333	0.333	1
4	0.25	0.143	0.167	0	0	0
5	0.095	0.125	0.154	0.286	0.333	0
6	0.238	0.221	0.471	0.333	0	0
7	0.5	0.667	1	1	1	1
8	0.056	0.208	0.333	0.5	0.667	0
9	0.027	0.036	0.067	0.286	0.667	0.5
10	0.19	0.444	0.286	0.333	0.333	0
11	0.031	0.061	0.111	0.2	1	0
12	0.102	0.108	0.16	0.237	0.083	0
13	0.273	0.5	0.6	0.6	0.667	1
14	0.088	0.135	0.14	0.153	0.169	0.281
15	0.172	0.242	0.389	0.333	0.667	0
16	0.889	0.875	1	1	1	0
rata-rata	0.222	0.282	0.354	0.393	0.479	0.325

Pada kebutuhan fungsional ketiga terdapat kalimat "*an email will be sent after the data is stored*". Kebutuhan fungsional ini dilakukan oleh *method* `EmailUtil.sendEmail` yang dipanggil oleh *method* `EditPatientAction.updateInformation`. Tapi nilai kemiripan *method* `EmailUtil.sendEmail`

pada contoh kasus ketiga adalah 0.632. Parsing pada kalimat kebutuhan fungsional menghasilkan kata kerja "*send*","*store*" dan kata benda "*email*", "*datum*". Parsing pada *method* `EmailUtil.sendEmail` menghasilkan kata kerja "*send*" dan kata benda "*email*". Kalimat "*an email will be sent after the data is stored*" merupakan kalimat paralel, kalimat yang terdiri dari dua kalimat menggunakan kata sambung. Kata sambung dalam kalimat ini adalah "*after*" yang merupakan jenis kata sambung subordinat(*subordinat conjunction*). Jika kalimat paralel dipisah maka akan menghasilkan kalimat "*an email will be sent*" dan "*the data is stored*". Kalimat pertama akan memiliki kemiripan dengan *method* `EmailUtil.sendEmail` sebesar 1.

Tabel 4.6. Nilai *Recall* Hasil Pengujian Kemiripan antara penelusuran pemanggilan Method(SimAst) dan Setiap Kebutuhan Fungsional Itrust

kebutuhan fungsional	treshold					
	0.7	0.75	0.8	0.85	0.9	0.95
1	0.778	0.778	0.778	0.778	0.778	0.667
2	0.647	0.647	0.647	0.647	0.647	0.412
3	0.438	0.188	0.063	0.063	0.063	0.063
4	0.714	0.143	0.143	0	0	0
5	0.867	0.533	0.4	0.267	0.133	0
6	0.857	0.536	0.286	0.036	0	0
7	0.4	0.4	0.4	0.2	0.2	0.2
8	0.625	0.625	0.5	0.25	0.25	0
9	0.429	0.286	0.286	0.286	0.286	0.143
10	0.667	0.667	0.333	0.333	0.333	0
11	0.4	0.4	0.2	0.2	0.2	0
12	0.816	0.579	0.421	0.368	0.053	0
13	0.429	0.429	0.429	0.429	0.286	0.143
14	0.824	0.706	0.706	0.647	0.588	0.529
15	0.588	0.471	0.412	0.235	0.118	0
16	0.727	0.636	0.636	0.636	0.636	0
rata-rata	0.638	0.501	0.415	0.336	0.286	0.135

4.4.1. Perbandingan .dengan Metode LDA Dan LSA

Sebagai pembanding dilakukan percobaan terhadap dataset yang sama menggunakan LDA dan LSA. Pada penelitian yang dilakukan Marcus dan Oliveto tersebut, dataset yang digunakan adalah dokumen kasus penggunaan dan file *class*. Penelitian tersebut menggunakan matrik *term* dokumen dengan kolom

merupakan setiap file *class* dalam kode sumber. Link penelusuran yang dihasilkan adalah antara kasus penggunaan dengan *class* yang menjalankannya. Pada pengujian dalam penelitian ini dataset yang digunakan adalah kebutuhan fungsional dengan *method* yang menjalankannya. Matrik yang digunakan terdiri dari term dalam *method* dan kolom merupakan setiap *method* dalam kode sumber. Dalam pengujian ini juga tidak menggunakan komentar dalam kode sumber. Hasil nilai *precision* dari percobaan menggunakan LSA dan LDA ditunjukkan pada tabel 4.7 dan nilai *recall* ditunjukkan pada Tabel 4.8.

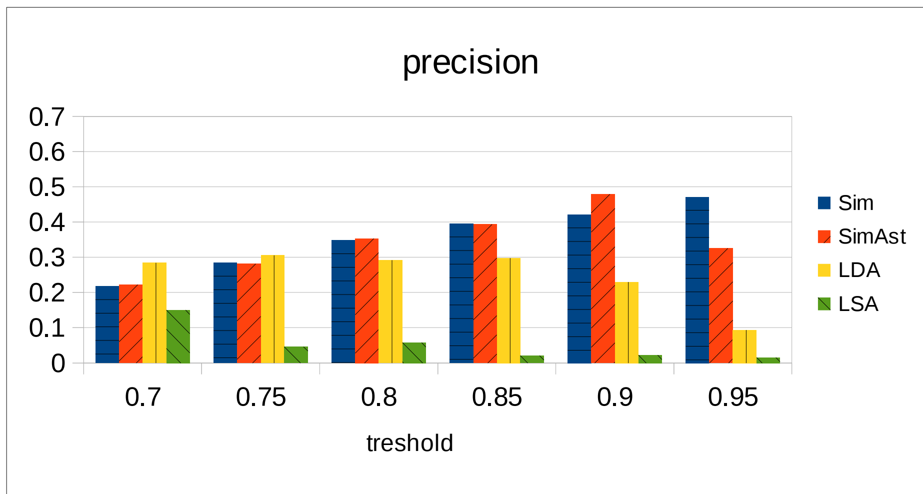
Tabel 4.7. Nilai *Precision* Hasil Pengujian Itrust dengan metode LSA dan LDA

	treshold					
	0.7	0.75	0.8	0.85	0.9	0.95
LSA	0.15	0.046	0.058	0.021	0.022	0.016
LDA	0.284	0.305	0.292	0.298	0.23	0.093

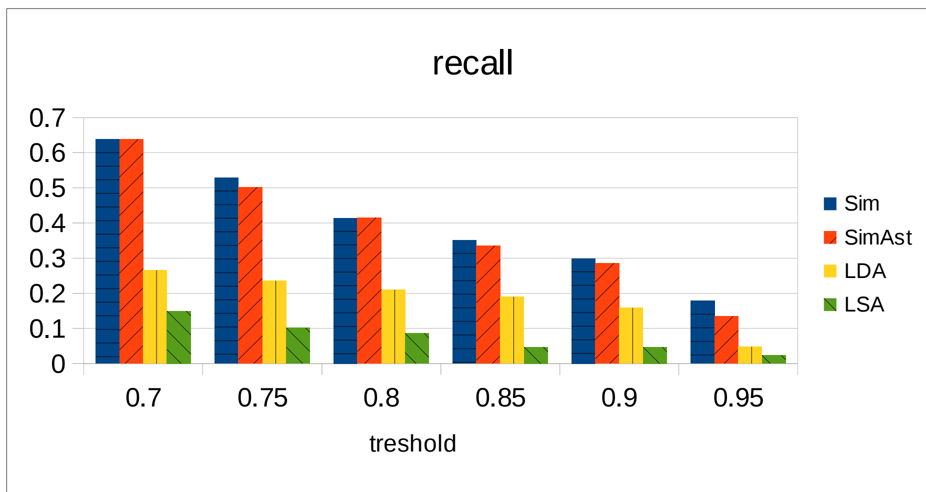
Tabel 4.8. Nilai *Recall* Hasil Pengujian Itrust dengan metode LSA dan LDA

	treshold					
	0.7	0.75	0.8	0.85	0.9	0.95
LSA	0.149	0.102	0.086	0.047	0.047	0.023
LDA	0.265	0.236	0.21	0.19	0.159	0.048

Hasil pengujian menggunakan metode LDA menghasilkan nilai *precision* sebesar 0.305 pada *recall* 0.23. Pada kasus ini nilai tersebut tercapai pada iterasi 2000 dengan jumlah topik 100 dan ambang batas 0.75. Metode LSA menghasilkan *precision* sebesar 0.15 pada *recall* 0.15. Pada kasus Itrust, metode LDA menghasilkan *precision* dan *recall* lebih baik dari pada LSA. Perbandingan *precision* antara metode kemiripan dalam penelitian ini dengan LDA ditunjukkan pada grafik Gambar 4.6 dan perbandingan nilai *recall* pada Gambar 4.7.

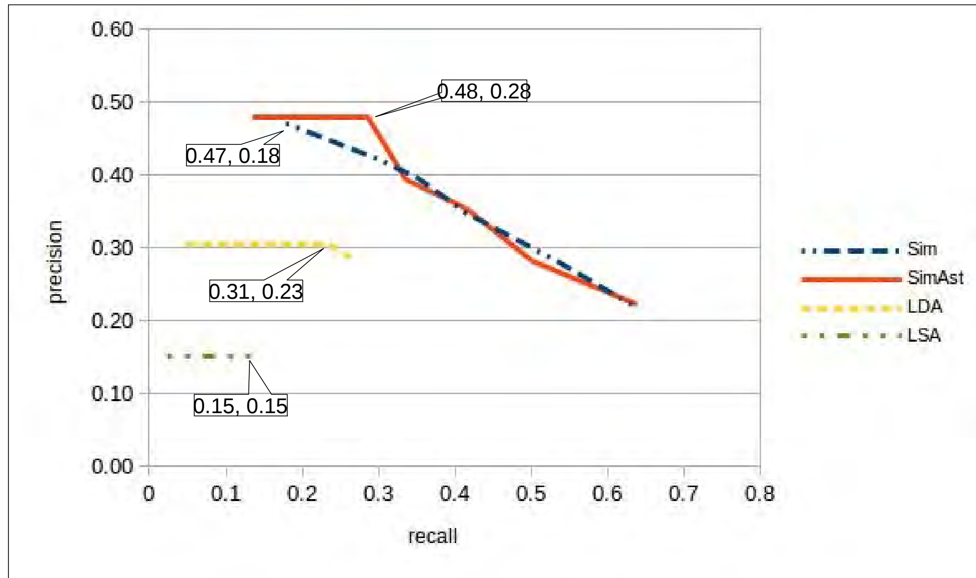


Gambar 4.6. Grafik Perbandingan Nilai *Precision* Itrust



Gambar 4.7. Grafik Perbandingan Nilai Recall Itrust

Gambar 4.8 menunjukkan kurva interpolasi *precision recall* hasil pengujian. Metode penelusuran menggunakan Sim menghasilkan nilai *precision* tertinggi 0.47 pada *recall* 0.18 tapi nilai *precision*-nya terus menurun dengan meningkatnya *recall*. Penggunaan penelusuran SimAst bisa mempertahankan nilai *precision* 0.48 sampai rentang *recall* 0.28. Sedangkan LDA hanya mencapai nilai *precision* 0.31 sampai rentang *recall* 0.23. Rata-rata kenaikan *precision* oleh penelusuran pemanggilan *method* sebesar 0.011. Dalam kasus Itrust, penelusuran pemanggilan *method* dan menggunakan kemiripan *method signature* memberikan hasil *precision* dan *recall* lebih baik daripada LDA.



Gambar 4.8. Kurva Interpolasi *Precision* dan *Recall* Itrust

4.5. Pengujian Gantt Project

Gantt Project adalah aplikasi pengelolaan proyek sederhana yang ditulis menggunakan bahasa pemrograman java dan bersifat kode terbuka. Halaman situs Gantt Project bisa diakses di alamat https://www.ganttproject.biz. Fitur yang dimiliki meliputi fungsi dasar pengelolaan proyek seperti gantt chart, penjadwalan dan pengelolaan sumber daya manusia. Kode sumber Gantt Project dapat diunduh di github pada alamat https://github.com/bardsoftware/ganttproject. Pada penelitian ini digunakan kode sumber versi 2.7.1. Kode sumber Gantt Project terdiri dari 520 class dengan jumlah method sebanyak 5292 method dan 17743 *method invocation*. Kalimat kebutuhan fungsional yang digunakan dalam pengujian ini adalah sebagai berikut :

1. Create a new task. Task has a start date and an end date and can have resources person
2. Delete task and all its links to other tasks. Remove resources assigment on the task
3. Create human resource. Add human resource that have name and holiday or vacation day to the project
4. Delete human resource. Remove all its assignment from the task

5. Create task link as dependency to other task. Task link indicate the sequence in wich they have to be performed
6. Add resource assignment to Task. Dependencies between tasks and assigned resource indicate the resource who are supposed to complete a task.

Nilai *precision* hasil pengujian kemiripan antara kode sumber dengan kebutuhan fungsional menggunakan metode wu-palmer dan Persamaan 2.4 dengan nilai alpha 0.4 ditampilkan pada Tabel 4.9 dan nilai *recall* hasil pengujian ditampilkan pada Tabel 4.10.

Tabel 4.9. Nilai *Precision* Hasil Pengujian Kemiripan *Method Signature*(Sim) dan Setiap Kebutuhan Fungsional Gantt Project

kebutuhan fungsional	treshold					
	0.7	0.75	0.8	0.85	0.9	0.95
1	0.018	0.03	0.06	0.053	0.097	0.104
2	0.178	0.271	0.579	1	1	0
3	0.031	0.071	0.211	0.24	0.208	0.238
4	0.292	0.35	0.579	0.889	0.889	1
5	0.009	0.019	0.03	0.088	0.241	0
6	0.025	0.058	0.273	0.5	0	0
rata-rata	0.092	0.133	0.289	0.462	0.406	0.224

Tabel 4.10. Nilai *Recall* Hasil Pengujian Kemiripan *Method Signature*(Sim) dan Setiap Kebutuhan Fungsional Gantt Project

kebutuhan fungsional	treshold					
	0.7	0.75	0.8	0.85	0.9	0.95
1	0.588	0.588	0.529	0.353	0.353	0.294
2	0.533	0.433	0.367	0.333	0.2	0
3	0.846	0.769	0.615	0.462	0.385	0.385
4	0.95	0.7	0.55	0.4	0.4	0.3
5	0.421	0.421	0.368	0.368	0.368	0
6	0.6	0.6	0.6	0.6	0	0
rata-rata	0.656	0.585	0.505	0.419	0.284	0.163

Pada kebutuhan fungsional pertama dengan titik potong 0,9 terdapat 61 *method* yang terpilih tetapi hanya 6 *method* yang benar merupakan *traceability link*. Banyak *method* yang tidak berhubungan dengan kebutuhan fungsional tetapi memiliki nilai kemiripan tinggi yaitu lebih besar dari 0.9. Salah satu kalimat yang digunakan pada kebutuhan fungsional pertama adalah "*Create a new task*".

Menggunakan POS tagger pada kalimat tersebut akan didapatkan kata kerja "*create*" dan kata benda "*task*". Jarak kemiripan antara kalimat tersebut dengan *method* `TaskMoveDownAction.run` adalah 0.971. Parsing pada *method* `TaskMoveDownAction.run`, akan didapatkan kata kerja "*move*" dan kata benda "*task*". Dengan menggunakan metode wu-palmer akan didapatkan nilai kemiripan antara *create* dan *move* adalah 0,88. Kemiripan yang tinggi padahal maksud dari kedua kata tersebut sangat berbeda. Proses perhitungan jarak antara *create* dan *move* dilakukan sebagai berikut :

T1 = HyperTrees(create) =*ROOT*<act<create

T2 = HyperTrees(move) =*ROOT*<move

Lcs = depth(subsumer(T1,T2))

= {subsumer(T1[1], T2[1])} = {act#v#1}

N3 = depth(act#v#1) = 2

N1 = min(depth({tree in T1 | tree contains LCS })) = 3

N2 = min(depth({tree in T2 | tree contains LCS })) = 2

Sim(c1,c2) = 2 * N3 / (N1 + N2)

= 2 * 2 / (3 + 2)

= 0,8

Dengan menggunakan Persamaan 2.2, maka jarak *cosine* antara *create* dan *move* adalah 0.951. Sedangkan kata bendanya adalah *task*, nilai kemiripan kata bendanya adalah 1. Dengan menggunakan Persamaan 2.4 akan dihasilkan nilai kemiripan sebesar 0.971. `TaskMoveDownAction.run` merupakan *method* yang bertujuan untuk memindahkan urutan *task* dari daftar, bukan *method* yang dijalankan dan tidak dipanggil oleh *method* lain dalam memenuhi kebutuhan membuat *task* baru. Hal ini menyebabkan kecilnya nilai *precision* dan *recall*.

Kalimat pada kebutuhan fungsional kelima adalah "*Create task link as dependency to other task*". Kata kerja pada kalimat tersebut adalah "*create*". Terdapat tiga *method* yang memiliki nilai kemiripan diatas 0.9 yang bukan merupakan *traceability* link. Ketiga method tersebut memiliki kata kerja "*clear*" yaitu `TaskDependencyCollectionImpl.clear`, `TaskDependency`

`CollectionImpl.doClear,` `MutableTaskDependencyCollection` `.clear`. Proses perhitungan jarak antara *create* dan *clear* adalah sebagai berikut :

T1 = HyperTrees(create#v#1) =*ROOT*<create#v#1

T2 = HyperTrees(clear#v#2) = *ROOT* <make#v#3<clear#v#2

Lcs = depth(subsumer(T1,T2)))

= { subsumer(T1[1], T2[1]) } = {make#v#3}

N3 = depth(make) = 2

N1 = min(depth({tree in T1 | tree contains LCS })) = 2

N2 = min(depth({tree in T2 | tree contains LCS })) = 3

Sim(c1,c2) = 2 * N3 / (N1 + N2)

= 2 * 2 / (3 + 2)

= 0,8

Dengan menggunakan Persamaan 2.2, maka jarak *cosine* antara *create* dan *clear* adalah 0.951. Nilai akhir kemiripan kebutuhan fungsional kelima dengan masing-masing *method* tersebut adalah 0.915. Ketiga *method* tersebut berfungsi untuk menghapus hubungan antara *task*, berlawanan dengan kebutuhan fungsional kelima, tetapi memiliki nilai kemiripan sangat tinggi.

Pada kebutuhan fungsional kelima terdapat 29 *method* pada titik potong 0.9, tetapi yang benar positif hanya 7 *method*. 12 diantara 29 *method* tersebut adalah *accessor* atau *getter*. Dalam bahasa pemrograman java terdapat jenis *method accessor* yaitu *method* yang bertujuan mengakses variabel tingkat *class*. *Method* ini hanya mengembalikan properti atau atribut *class* tanpa ada proses apapun didalam *method* tersebut. Nama *method* ini dimulai dengan kata "*get*" dan disambung dengan nama variabel yang diakses. Kata kerja dalam kalimat kebutuhan fungsional kelima adalah "*create*". Nilai kemiripan antara *create* dan *get* adalah 0.951, sehingga banyak *accessor* yang memiliki nilai kemiripan tinggi. Nilai precision pada titik potong 0.9 adalah 0.241, jika accessor dihilangkan, maka nilai precision akan naik sebesar 0.17 menjadi 0.411. Pada titik potong 0.8 nilai precision adalah 0.03, terdapat 230 method dan 83 diantaranya adalah *accessor*. Jika *accessor* dihilangkan maka akan meningkatkan precision sebesar

0.017 menjadi 0.047.

Pada kebutuhan fungsional keenam tidak ada *method* yamg memiliki kemiripan diatas 0.9. Kalimat pada kebutuhan fungsional keenam adalah "*Add resource assignment to Task*". Menggunakan POS tagger akan dihasilkan kata kerja "*add*" dan kata benda "*resource*", "*assignment*", "*Task*". Kata benda "*task*" disini merupakan kata pelengkap atau objek tidak langsung(*Indirect Object*) dari kalimat. Salah satu *method* yang menjalankan kebutuhan ini adalah `ResourceAssignmentCollection.addAssignment`. Nilai kemiripan antara *method* tersebut dengan kalimat kebutuhan fungsional adalah 0.866. Jika objek tidak langsung(*Indirect Object*) tidak diikutsertakan dalam perhitungan kemiripan, maka nilai kemiripan antara kebutuhan fungsional dan *method* akan meningkat. Kata *task* yang menjadi konteks dari fungsi *method* tidak dimasukkan dalam nama *method* atau nama *class*. Dalam kasus ini pembuat program mengelompokkan *class*-*class* yang melakukan fungsi berhubungan dengan *task* dalam *package* `net.sourceforge.ganttproject.task.` Dalam *package* ini terdapat *class*-*class* yang melakukan fungsi terhadap *task* seperti mengelola *resource* dalam *task*, mengelola ketergantungan(*task dependency*) dan menghitung *critical path* dari suatu *task*, mengatur properti dan *event* tambahan dalam *task*. Kata pelengkap *task* tidak dimasukkan dalam *method signature* tetapi didalam *package* dari *class*-*class* tersebut.

Langkah pengujian selanjutnya adalah menggunakan penelusuran terhadap *method* yang dipanggil oleh *method*-*method* pada pengujian langkah sebelumnya. Menggunakan persamaan 3.3 dengan nilai beta 0.9 dihasilkan nilai *precision* seperti yang ditunjukkan pada Tabel 4.11 dan nilai *recall* ditunjukkan pada Tabel 4.12.

Tabel 4.11. Nilai *Precision* Hasil Pengujian Kemiripan Penelusuran Pemanggilan *Method*(SimAst) dan Setiap Kebutuhan Fungsional Gantt Project

kebutuhan fungsional	treshold					
	0.7	0.75	0.8	0.85	0.9	0.95
1	0.018	0.031	0.056	0.051	0.102	0.128
2	0.174	0.25	0.688	1	1	0
3	0.034	0.072	0.206	0.208	0.208	0.263
4	0.328	0.35	0.647	0.889	0.889	1
5	0.009	0.02	0.031	0.095	0.25	0
6	0.026	0.058	0.3	0.6	0	0
rata-rata	0.098	0.13	0.321	0.474	0.408	0.232

Tabel 4.12. Nilai *Recall* Hasil Pengujian Kemiripan Penelusuran Pemanggilan *Method*(SimAst) dan Setiap Kebutuhan Fungsional Gantt Project

kebutuhan fungsional	treshold					
	0.7	0.75	0.8	0.85	0.9	0.95
1	0.588	0.588	0.471	0.353	0.353	0.294
2	0.5	0.367	0.367	0.333	0.133	0
3	0.846	0.769	0.538	0.385	0.385	0.385
4	0.95	0.7	0.55	0.4	0.4	0.3
5	0.421	0.421	0.368	0.368	0.368	0
6	0.6	0.6	0.6	0.6	0	0
rata-rata	0.651	0.574	0.482	0.407	0.273	0.163

Pada proses penelusuran pemanggilan *method* menunjukkan peningkatan nilai *precision*. Pada titik potong 0.8 nilai *precision* naik sebesar 0.03. Ini merupakan kenaikan tertinggi daripada titik potong lainnya. Rata-rata penurunan nilai *recall* sebesar 0.014 dimulai pada titik potong 0.75 sampai pada titik potong 0.9. Penurunan *recall* tertinggi sebesar 0.02 pada titik potong 0.8. Pada kebutuhan fungsional keempat di titik potong 0.8, penelusuran AST meningkatkan *precision* sebesar 0.068. Pada titik potong 0.8, perhitungan kemiripan menghasilkan 19 *method* dengan 11 *method* merupakan benar positif. Penelusuran AST menurunkan jumlah *method* yang memiliki kemiripan diatas 0.8 dari 19 *method* menjadi 17 dan yang benar positif tetap 11 *method*. Penelusuran pemanggilan *method* menurunkan kemiripan dua *method* yang tidak sesuai atau positif salah(*false positive*). `TaskManagerImpl.fireTaskRemoved` menurun nilai kemiripannya dari 0.8 menjadi 0.79 dan `MiltonResourceImpl.delete`

menurun dari 0.8 menjadi 0.77 karena kedua *method* tersebut memanggil *method* lain yang nilai kemiripannya kecil. `TaskManagerImpl.fireTaskRemoved` memanggil tiga *method* dengan total nilai 2.135 sehingga rata-rata nilai kemiripan tiga *method* tersebut adalah 0.733.

4.5.1. Perbandingan.dengan Metode LDA Dan LSA

Sebagai pembanding dilakukan percobaan terhadap dataset yang sama menggunakan LDA dan LSA. Hasil nilai *precision* dari percobaan menggunakan LSA dan LDA ditunjukkan pada Tabel 4.13 dan nilai *recall* ditunjukkan pada Tabel 4.14.

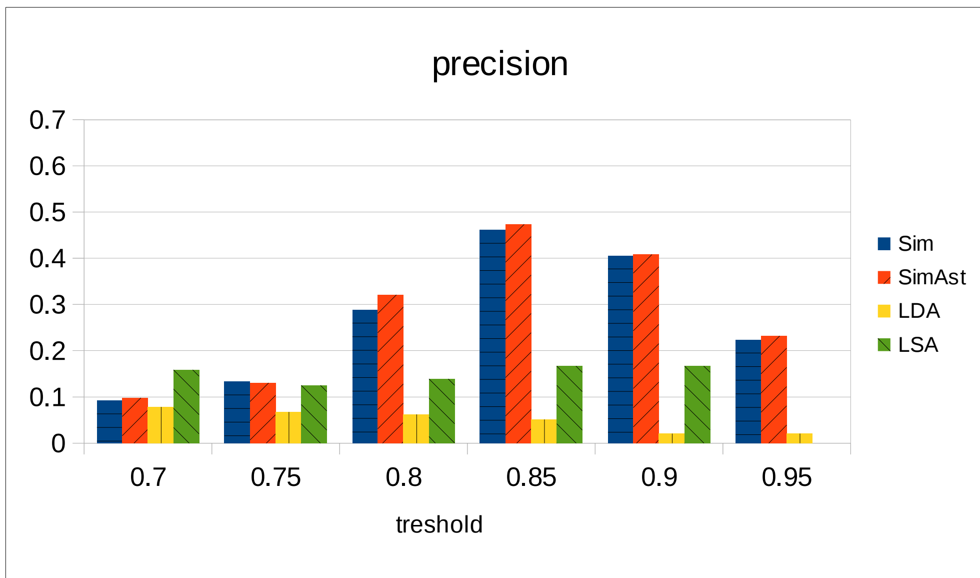
Tabel 4.13. Nilai *Precision* Hasil Pengujian Gantt Project dengan Metode LSA dan LDA

	treshold					
	0.7	0.75	0.8	0.85	0.9	0.95
LSA	0.159	0.125	0.139	0.167	0.167	0
LDA	0.078	0.068	0.062	0.051	0.02	0.02

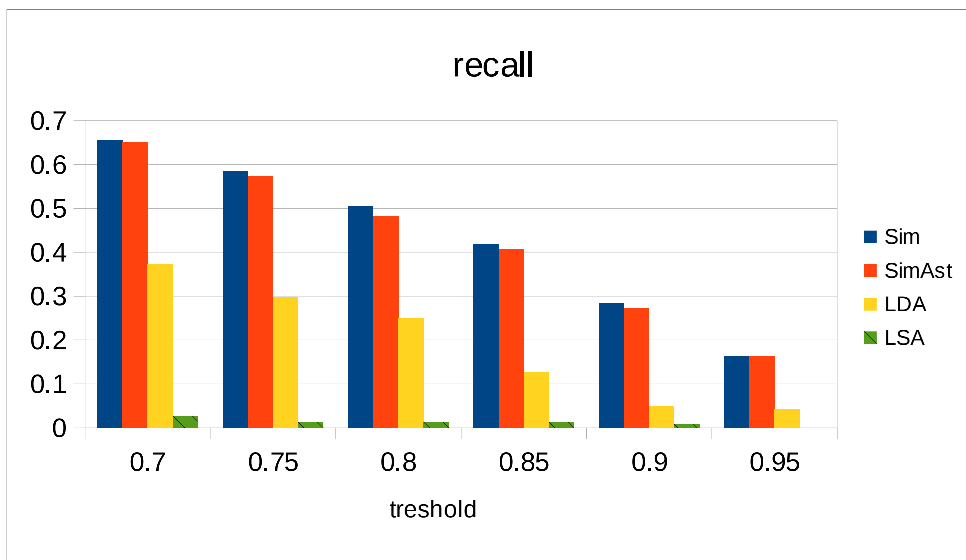
Tabel 4.14. Nilai *Recall* Hasil Pengujian Gantt Project dengan Metode LSA dan LDA

	treshold					
	0.7	0.75	0.8	0.85	0.9	0.95
LSA	0.028	0.014	0.014	0.014	0.008	0
LDA	0.372	0.298	0.25	0.127	0.05	0.042

Hasil pengujian menggunakan metode LDA menghasilkan nilai *precision* sebesar 0.078 pada *recall* 0.372. Pada kasus ini nilai tersebut tercapai pada iterasi 2000 dengan jumlah topik 100 dan ambang batas 0.70. Metode LSA menghasilkan *precision* sebesar 0.167 pada *recall* 0.014. Pada kasus Gantt Project, metode LSA menghasilkan nilai *precision* lebih tinggi daripada LDA. Tetapi pada titik potong 0.95 LSA tidak menemukan *traceability* link, nilai *precision* yang dihasilkan 0. Nilai *recall* LSA sangat kecil, nilai *recall* tertinggi 0.028. Perbandingan *precision* antara metode kemiripan dalam penelitian ini dengan LDA ditunjukkan pada grafik Gambar 4.9 dan perbandingan nilai *recall* pada Gambar 4.10.



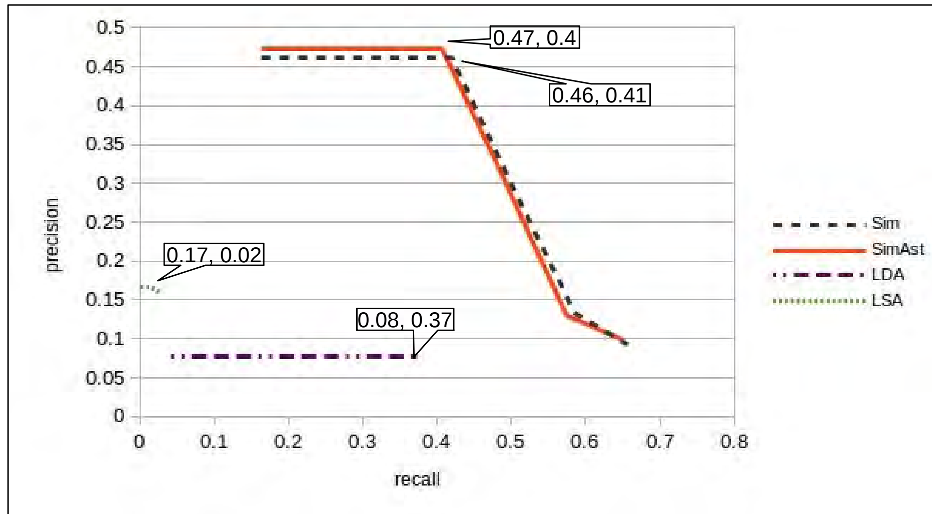
Gambar 4.9. Grafik Perbandingan Nilai *Precision* Gantt Project



Gambar 4.10. Grafik Perbandingan Nilai *Recall* Gantt Project

Kurva interpolasi *precision recall* hasil pengujian ditunjukkan pada Gambar 4.11. Metode penelusuran menggunakan Sim menghasilkan nilai *precision* 0.46 sampai pada rentang *recall* 0.41. Penggunaan penelusuran AST menghasilkan nilai *precision* 0.47 sampai rentang *recall* 0.40. Rata-rata kenaikan *precision* oleh penelusuran pemanggilan *method* sebesar 0.012. Sedangkan LDA hanya mencapai nilai *precision* 0.08 sampai rentang *recall* 0.37. Dalam kasus Gantt Project,

penelusuran pemanggilan *method* dan menggunakan kemiripan *method signature* memberikan hasil *precision* dan *recall* lebih baik daripada LDA.



Gambar 4.11. Kurva Interpolasi *Precision* dan *Recall* Gantt Project

Pengujian menggunakan LDA yang digunakan sebagai pembanding dalam penelitian ini tidak sepenuhnya sama dengan yang dilakukan oleh penelitian sebelumnya(Oliveto, 2010). Jika Oliveto menggunakan *class* dan dokumen kasus penggunaan sebagai term dokumen. Dalam percobaan tersebut juga menyertakan komentar dalam *class* sehingga kemungkinan suatu term muncul dalam dokumen lebih besar. Pengujian dalam penelitian ini menggunakan isi *method,* sehingga dokumen yang dihasilkan berjumlah lebih banyak tapi berukuran kecil dan terdiri dari term yang lebih sedikit dalam setiap dokumen. Juga tidak menyertakan komentar dalam dokumen sehingga semakin kecil kemungkinan munculnya term dalam suatu dokumen.

[Halaman ini sengaja dikosongkan]

BAB 5
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Beberapa kesimpulan yang dapat ditarik dari hasil pengerjaan penelitian ini adalah sebagai berikut ini.

1. Dalam penelitian ini dikembangkan suatu metode untuk menemukan link penelusuran antara *method* dan kebutuhan fungsional. Proses penghitungan kemiripan terdiri dari dua tahap. Tahap pertama menggunakan kata kerja dan kata benda yang diparsing dari *method siganiture* dan kalimat kebutuhan fungsional. Tahap kedua melakukan penelusuran terhadap *method-method* yang diapanggil oleh *method* dalam tahap pertama.
2. Proses penelusuran pemanggilan *method* dapat meningkatkan nilai precision. Pada kasus Itrust nilai *precision* tertinggi 0.48 dicapai pada rentang *recall* 0.28 pada titik potong 0.9 dan rata-rata peningkatan *precision* oleh penelusuran pemanggilan *method* sebesar 0.011. Pada kasus Gantt Project *precision* tertinggi 0.47 dicapai pada rentang *recall* 0.40 pada titik potong 0.85 dan rata-rata peningkatan *precision* oleh penelusuran pemanggilan *method* sebesar 0.012. Nilai *precision* tergolong kecil, dibawah 0.5, karena banyak *method* yang bukan link penelusuran tetapi memiliki nilai kemiripan yang tinggi.
3. Hasil pengujian menunjukkan metode kemiripan *method signature* dan penelusuran pemanggilan *method* memberikan hasil nilai *precision* dan *recall* lebih baik daripada metode LSA dan LDA.
4. Dalam proses parsing kalimat dan *method signature* penggunaan lemmatisasi lebih sesuai daripada *stemming* karena proses penghitungan kemiripan dalam penelitian ini menggunakan Wordnet sehingga sangat penting untuk menghasilkan term yang memiliki makna atau ada dalam WordNet.

5.2. Saran

Saran-saran untuk penelitian mendatang adalah sebagai berikut :

1. Hasil pengujian pada beberapa kebutuhan fungsional tidak menemukan link penelusuran karena kecilnya nilai kemiripan antara kalimat kebutuhan fungsional dengan *method signature*. Salah satu penyebabnya karena struktur kalimat kebutuhan fungsional seperti kalimat paralel dan adanya objek tidak langsung(*Indirect Object*) dalam kalimat. Pada penelitian mendatang perlu dilakukan parsing struktural pada kalimat kebutuhan fungsional menjadi bentuk yang lebih sederhana sehingga bisa meningkatkan nilai kemiripan dengan *method signature*.
2. Dalam OOP dikenal *design pattern* yang mengatur bentuk *class* dan *method* yang memiliki pola tertentu dengan tujuan tertentu. Ini mempengaruhi bentuk *method signature*, seperti pada *accessor* atau *getter*. Untuk meningkatkan nillai *precision* dan *recall*, aturan parsing pada kode sumber perlu mengakomodir pola-pola pada *design pattern* supaya bisa memberikan hasil parsing yang lebih akurat.

DAFTAR PUSTAKA

Abadi, A., Nisenson, M., Simionovici, Y. (2008), "A traceability technique for specifications", *Proceedings of 16th IEEE International Conference on Program Comprehension*, hal. 103-112.

Ali, N., Guéhéneuc, Y.-G. , Antoniol, G. (2011), "Requirements Traceability for Object Oriented Systems by Partitioning Source Code", *Proceedings of 18th Working Conf. Reverse Eng., hal*. 45-54.

Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.(2000), "Tracing Object-Oriented Code into Functional Requirements", *Proceedings of 8th International Workshop on Program Comprehension (IWPC'00)*, hal. 79-87.

Eaddy, M., Aho, A., Antoniol, G., Guéhéneuc, Y.-G. (2008), "Cerberus: Tracing requirements to source code using information retrieval dynamic analysis and program analysis", *Proceedings of 16th IEEE International Conference on Program Comprehension*, hal. 53-62.

Fellbaum, C. (1998), "*WordNet: An Electronic Lexical Database",* MIT Press, Cambridge.

Gotel O., Finkelstein, A. (1994), "An analysis of the requirements traceability problem", *Proceedings of 1st International Conference on Requirements Engineering*, hal. 94-101.

Hayashi, S., Yoshikawa, T., Saeki, M. (2010), "Sentence-to-Code Traceability Recovery with Domain Ontologies," *Proceedings of 2010 Asia Pacific Software Engineering Conference*, hal. 385-394.

Mahmoud, A., Nan Niu, Songhua Xu (2012), "A semantic relatedness approach for traceability link recovery", *proceedings of 20th International Conference on Program Comprehension*, Passau, hal. 183 – 192.

Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. "The Stanford CoreNLP Natural Language Processing Toolkit", *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55-60.

Marcus, A., Maletic, J.I. (2003), "Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing", *Proceeding International Conf. on Software Engineering*, hal. 125-135.

McCallum, A. K. (2002). MALLET: A Machine Learning for Language Toolkit. Retrieved 2016, from MALLET: A Machine Learning for Language Toolkit.: http://mallet.cs.umass.edu

Miller, G.A. (1995), "WordNet: A Lexical Database for English", *Communications of the ACM,* Vol. 38, No. 11, hal.39-41.

Ming Che Lee (2011), "A novel sentence similarity measure for semantic-based expert systems", *Expert Systems with Applications*, Vol. 38, Issue 5, hal 6392-6399.

Oliveto, R., Gethers, M., Poshyvanyk, D., De Lucia, A. (2010), "On the

equivalence of information retrieval methods for automated traceability link recovery", *Proceedings of Int. Conf. Program Comprehension*, hal. 68-71.

Shepherd, D., Fry, Z.P., Hill, E., Pollock, L., Vijay-Shanker, K. (2007), "Using natural language program analysis to locate and understand action-oriented concerns", *Proceeding of 6th International Conference on Aspect-Oriented Software Development (AOSD)*, hal. 212-224.

Sridhara, G., Hill, E., Pollock, L., Vijay-Shanker, K. (2008), "Identifying Word Relations in Software: A Comparative Study of Semantic Similarity Tools", *proceedings of 16th IEEE International Conference on Program Comprehension*, Amsterdam, hal. 123 – 132.

Wu, Z., & Palmer, M. (1994), "Verbs semantics and lexical selection", *Proceedings of the 32nd annual meeting on Association for Computational Linguistics,* hal. 133-138.

Zhang, Y., Witte, R., Rilling, J., Haarslev, V. (2006), "An Ontology-based Approach for Traceability Recovery", *3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering*.

BIOGRAFI PENULIS

Penulis merupakan pengajar di Fakultas Ilmu Komputer Universitas Brawijaya. Penulis menempuh pendidikan jenjang S1 di Teknik Elektro Universitas Brawijaya. Kemudian melanjutkan pendidikan jenjang S2 di jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya pada tahun 2012 kemudian lulus S2 (M.Kom) pada tahun 2016. Untuk korespondensi, penulis dapat dihubungi melalui email djoko.jalin@ub.ac.id