repository.ub.ac.id

PEMBANGUNAN KAKAS BANTU PEMBANGKITAN *PROGRAM DEPENDENCE GRAPH* BERBASIS JAVA

SKRIPSI

Untuk memenuhi sebagian persyaratan mencapai gelar Sarjana Komputer

Disusun oleh

FATHRA PRIMADHANA

NIM. 115090600111029

KEMENTERIAN RISET TEKNOLOGI DAN PENDIDIKAN TINGGI

UNIVERSITAS BRAWIJAYA

PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER

PROGRAM STUDI INFORMATIKA/ ILMU KOMPUTER

MALANG

2015

UNIVERSITAS BRAWIJAYA

LEMBAR PERSETUJUAN

PEMBANGUNAN KAKAS BANTU PEMBANGKITAN

***PROGRAM DEPENDENCE GRAPH* BERBASIS JAVA**

SKRIPSI

Laboratorium Rekayasa Perangkat Lunak

Untuk memenuhi sebagian persyaratan mencapai gelar Sarjana Komputer

Disusun oleh:

FATHRA PRIMADHANA

NIM. 115090600111029

Skripsi ini telah disetujui oleh dosen pembimbing pada tanggal 29 Juni 2015

Dosen Pembimbing I	Dosen Pembimbing II
Fajar Pradana, S.ST, M.Eng	**Denny Sagita R., S.Kom, M.Kom**
NIK. 871121 16110371	**NIK. 851124 06110250**

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN

PEMBANGUNAN KAKAS BANTU PEMBANGKITAN
PROGRAM DEPENDENCE GRAPH **BERBASIS JAVA**

SKRIPSI

LABORATORIUM REKAYASA PERANGKAT LUNAK

Untuk memenuhi sebagian persyaratan mencapai gelar Sarjana Komputer

Disusun Oleh :

FATHRA PRIMADHANA
NIM. 115090600111029

Skripsi ini telah dipertahankan dan dinyatakan lulus di depan majelis penguji pada tanggal 29 Juli 2015

Penguji I	**Penguji II**
Tri Astoto Kurniawan, ST, MT.Ph.D	**Issa Arwani, S.Kom., M.Sc**
NIP. 19710518 200312 1 001	**NIP. 19830922 201212 1 003**

Penguji III

Aswin Suharsono, ST., MT
NIK. 840919 06 1 1 0251

Mengetahui,
Ketua Program Studi Informatika / Ilmu Komputer

Drs. Marji, M.T.
NIP. 19670801 199203 1 001

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

PERNYATAAN
ORISINALITAS SKRIPSI

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah SKRIPSI ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata didalam naskah SKRIPSI ini dapat dibuktikan terdapat unsur-unsur PLAGIASI, saya bersedia SKRIPSI ini digugurkan dan gelar akademik yang telah saya peroleh (SARJANA) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku. (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 1 Juni 2015
Mahasiswa

Fathra Primadhana
NIM. 115090600111029

KATA PENGANTAR

Puji syukur kami panjatkan kehadirat Tuhan Yang Maha Kuasa, karena hanya atas limpahan dan rahmat Karunia-Nya lah, penulis dapat menyelesaikan skripsi dengan judul "Pembangunan Kakas Bantu Pembangkitan *Program Dependence Graph* Berbasis Java" ini dengan lancar.

Penyusunan skripsi tak lepas dari bantuan secara moril yang berupa bimbingan, kritik, saran, dukungan, motivasi maupun doa banyak pihak. Oleh karena itu, ucapan terima kasih penulis sampaikan kepada :

1. Hari Lugirin dan Muji Rahayu selaku orang tua penulis dan seluruh keluarga yang telah memberikan semangat, do'a dan segala pengorbanan serta dukungan penuh dalam penyusunan skripsi ini.
2. Bapak Fajar Pradana, S.ST, M.Eng selaku dosen pembimbing 1 dan Bapak Denny Sagita R., S.Kom, M.Kom selaku dosen pembimbing 2, atas segala bimbingan dan waktu yang telah diluangkan serta kritik dan saran yang telah diberikan kepada penulis.
3. Bapak Ir. Sutrisno, M.T, Bapak Ir. Heru Nurwasito, M.Kom, Bapak Himawat Aryadita, S.T, M.Sc, dan Bapak Eddy Santoso, S.Kom selaku Ketua, Wakil Ketua 1, Wakil Ketua 2 dan Wakil Ketua 3 Fakultas Ilmu Komputer Universitas Brawijaya
4. Bapak Drs. Marji, MT dan Bapak Issa Arwani, S.Kom., M.Sc selaku Ketua dan Sekretaris Program Studi Informatika.
5. Seluruh bapak dan ibu dosen Fakultas Ilmu Komputer Universitas Brawijaya Malang atas segala bimbingan serta ilmu yang telah diajarkan kepada penulis.
6. Para pegawai dan staf Fakultas Ilmu Komputer Universitas Brawijaya Malang.
7. Elok Fatma Anjarwati yang senantiasa memberikan dukungan dan banyak bantuan.
8. Seluruh teman-teman mahasiswa Fakultas Ilmu Komputer Universitas Brawijaya.
9. Seluruh teman-teman mahasiswa Program Studi Informatika / Ilmu Komputer khususnya angkatan 2011.

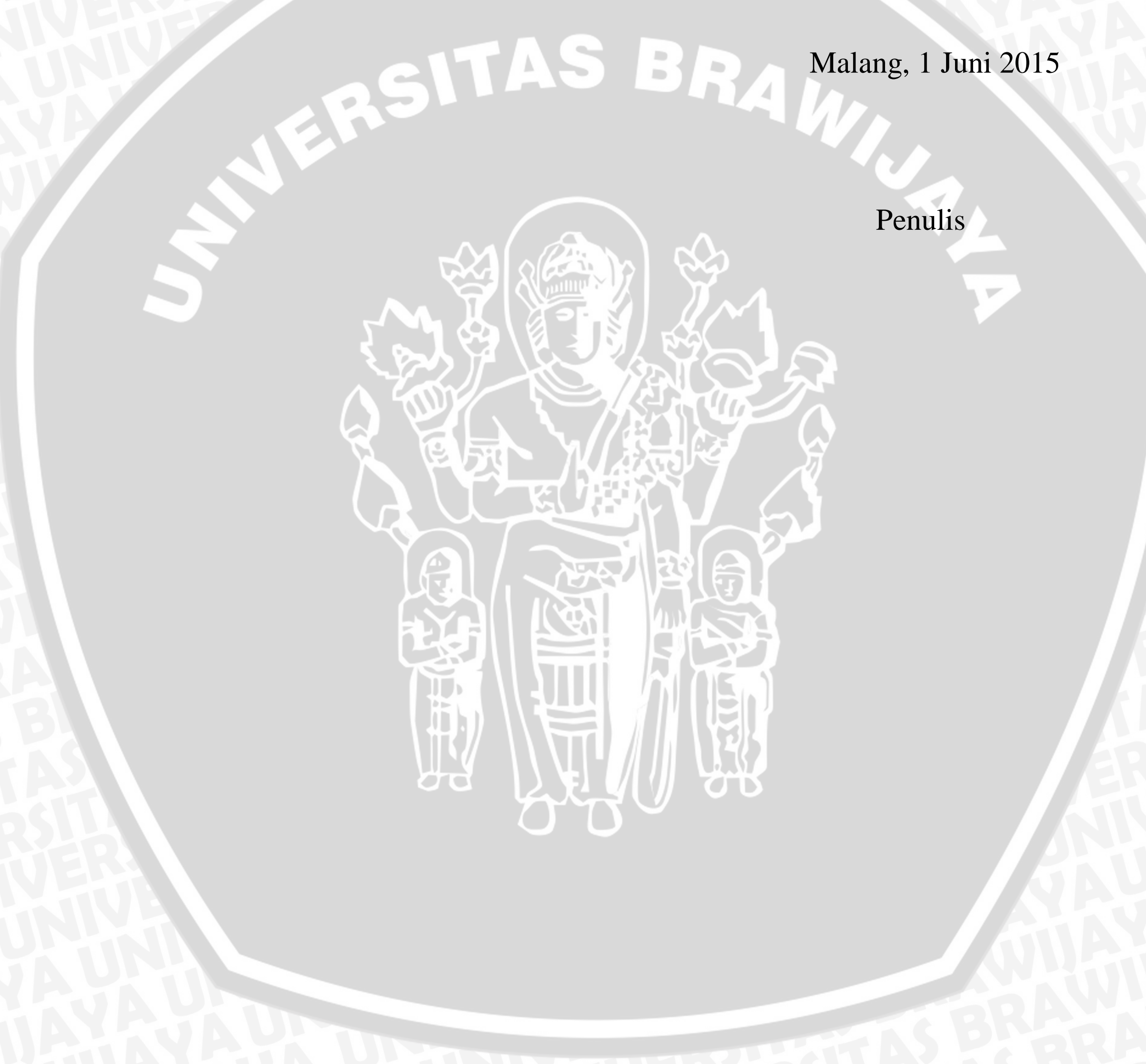
repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

10. Seluruh pihak yang tidak bisa disebutkan satu per satu. Penulis mengucapkan terima kasih atas segala bantuan yang telah diberikan.

Semoga penulisan laporan skripsi ini bermanfaat bagi pembaca dan untuk pengembangan selanjutnya. Penulis menyadari bahwa skripsi ini masih jauh dari sempurna serta banyak kekurangan disebabkan oleh keterbatasan kemampuan dan pengalaman, dengan kerendahan hati penulis mengharapkan kritik dan saran yang bersifat membangun dari pembaca.

Malang, 1 Juni 2015

Penulis



repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

ABSTRAK

Fathra Primadhana, 2015: Pembangunan Kakas Bantu *Pembangkitan Program Dependence* Graph Berbasis Java

Dosen Pembimbing: Fajar Pradana, S.ST, M.Eng dan Denny Sagita R., S.Kom, M.Kom

Kegiatan duplikasi kode menyebabkan fase perawatan perangkat lunak menjadi lebih rumit, tetapi kegiatan ini masih sering dilakukan karena mempermudah pengembang dalam fase pengembangan perangkat lunak. Pendeteksian duplikasi kode dapat dilakukan dengan mendapatkan hubungan ketergantungan antar variabel dari suatu kode program. Hubungan ketergantungan antar variabel dapat diketahui dengan cara membangun *Program Dependence Graph.* Pembuatan sistem pembangkitan *Program Dependence Graph* membutuhkan masukan berupa berkas *source code* yang bertipe Java. Proses awal yang digunakan untuk membangkitkan *Program Dependence Graph* adalah melakukan tahapan ekstraksi data dengan menggunakan pustaka *Abstract Syntax Tree* untuk memperoleh informasi variabel dari *source code*. Selanjutnya dilakukan proses normalisasi dan penambahan informasi relasi antar variabel. Setelah itu melakukan konversi data ke dalam bentuk grafik dengan menggunakan pustaka JUNG. Dari hasil pengujian validasi diperoleh hasil bahwa fungsi dalam kakas bantu pembangkitan *Program Dependence Graph* sudah benar semua sesuai dengan hasil yang diharapkan. Dari hasil pengujian akurasi diperoleh rata-rata akurasi sistem sebesar 100%.

Kata Kunci: *Program Dependence Graph*, Java, *Abstract Syntax Tree*, JUNG, *source code*.

UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

ABSTRACT

Fathra Primadhana, 2015: Pembangunan Kakas Bantu *Pembangkitan Program Dependence* Graph Berbasis Java

Dosen Pembimbing: Fajar Pradana, S.ST, M.Eng dan Denny Sagita R., S.Kom, M.Kom

Duplicating code make software maintenance phases become more complicated, but this activity is still often done because it simplifies developers in the software development phase. Dependence relationships between variables can be determined by building Program Dependence Graph. Building a system that generate Program Dependence Graph requires the input of java files. The initial process used to generate Program Dependence Graph is extracting data using the Abstract Syntax Tree library to obtain variable information from the source code. Then, normalizing the data and add additional information about relationships between variables. Then convert the data into graphical form using JUNG library. From the result of validation testing showed that all function in Program Dependence Graph generation tool is valid as expected. From the accuracy test results obtained an average accuracy of 100%.

Keywords: Program Dependence Graph, Java, *Abstract Syntax Tree*, JUNG, *source code*.

UNIVERSITAS BRAWIJAYA

DAFTAR ISI

LEMBAR PERSETUJUAN i
LEMBAR PENGESAHAN ii
PERNYATAAN ORISINALITAS SKRIPSI iii
KATA PENGANTAR iv
ABSTRAK vi
ABSTRACT vii
DAFTAR ISI viii
DAFTAR GAMBAR xii
DAFTAR TABEL xv
BAB I 1
PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 3
1.3 Batasan Masalah 3
1.4 Tujuan 3
1.5 Manfaat 4
1.6 Sistematika Penulisan 4
BAB II 6
DASAR TEORI DAN TINJAUAN PUSTAKA 6
2.1 Kajian Pustaka 6
2.2 *Program Dependence Graph* 7
2.2.1 *Control Dependence* 8
2.2.2 *Data Dependence* 8
2.3 *Source Code* 9
2.4 Java *Bytecode* 9
2.5 *Abstract Syntax Tree* 10
2.5.1 Alur Kerja *Abstract Syntax Tree* 10
2.5.2 Model Java *Abstract Syntax Tree* 11
2.6 *Java Universal Network/Graph Framework* (JUNG) 12
2.7 *Software Process Model* 13

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

2.8 *Unified Modelling Language* (UML) 15
2.8.1 *Use Case Diagram* 15
2.8.2 *Activity Diagram* 16
2.8.3 *Class Diagram* 18
2.8.4 *Sequence Diagram* 19
2.8.5 *Component Diagram* 21
2.9 *Java System Program Dependence Graph* API 22
2.10 *Cyclomatic Complexity* 22
2.11 Akurasi 23
BAB III 24
METODOLOGI PENELITIAN 24
3.1 Identifikasi Masalah 25
3.2 Studi Literatur 25
3.3 Analisis Kebutuhan 26
3.4 Perancangan Perangkat Lunak 26
3.5 Implementasi 27
3.6 Pengujian 28
3.7 Kesimpulan dan Saran 28
3.8 Penulisan Laporan 28
BAB IV 29
ANALISIS DAN PERANCANGAN 29
4.1 Analisis Kebutuhan 30
4.1.1 Gambaran Umum Sistem 30
4.1.2 Identifikasi Aktor 30
4.1.3 Analisis Kebutuhan Fungsional 31
4.1.4 Analisis Kebutuhan Non-Fungsional 40
4.1.5 Analisis Komponen 40
4.2 Perancangan Perangkat Lunak 41
4.2.1 Modifikasi Kebutuhan 42
4.2.2 Perancangan Arsitektur 42
4.2.3 Perancangan Integrasi Komponen 43
4.2.4 Perancangan *Activity Diagram* 45

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

4.2.5 Perancangan *Class Diagram*.......... 52
4.2.6 Perancangan *Sequence Diagram* 74
4.2.7 Perancangan *Component Diagram*.......... 81
4.2.8 Perancangan Antarmuka.......... 82
BAB V 86
IMPLEMENTASI.......... 86
5.1 Spesifikasi Sistem 86
5.1.1 Spesifikasi Perangkat Keras 86
5.1.2 Spesifikasi Perangkat Lunak 87
5.2 Batasan Implementasi 87
5.3 Implementasi Kode Program.......... 87
5.3.1 Implementasi Proses Ekstraksi Data.......... 88
5.3.2 Implementasi Proses AST_Model 98
5.3.3 Implementasi Proses Buat Grafik 103
5.4 Implementasi Antarmuka 104
5.4.1 Implementasi Halaman Utama 105
5.4.2 Implementasi Halaman *Program Dependence Graph* 106
5.4.3 Implementasi Halaman Tabel Ketergantungan 107
BAB VI 108
PENGUJIAN.......... 108
6.1 Pengujian Unit.......... 108
6.1.1 Pengujian Unit Ekstraksi Data.......... 108
6.1.2 Pengujian Unit Buat Grafik 110
6.1.3 Pengujian Unit Create Tree 112
6.2 Pengujian Validasi.......... 114
6.2.1 Kasus Uji Pengujian Validasi 114
6.2.2 Hasil dan Analisis Pengujian Validasi.......... 119
6.3 Pengujian Akurasi 120
6.3.1 Proses Awal Eksekusi Pustaka *Java System Dependence Graph* API 121
6.3.2 Kasus Uji Pengujian Akurasi.......... 121
6.3.3 Hasil dan Analisis Pengujian Akurasi 145
BAB VII.......... 147

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

PENUTUP 147
7.1 Kesimpulan 147
7.2 Saran 147
DAFTAR PUSTAKA 149

DAFTAR GAMBAR

Gambar 2.1 *Program Dependence Graph* 7
Gambar 2.2 Java *Bytecode* 10
Gambar 2.3 Alur Kerja *Abstract Syntax Tree* 11
Gambar 2.4 Model Java *Abstract Syntax Tree* 12
Gambar 2.5 *Reuse-Oriented Software Engineering* 14
Gambar 2.6 Representasi Komponen 21
Gambar 2.7 *Assembly Connector* 22
Gambar 3.1 Diagram Alir Metodologi Penelitian 24
Gambar 3.2 Model *Reuse Development* 27
Gambar 4.1 Diagram Alir Analisis dan Perancangan 29
Gambar 4.2 *Use Case Diagram* 33
Gambar 4.3 Arsitektur Sistem Pembangkitan *Program Dependence Graph* 43
Gambar 4.4 Perancangan Ekstraksi Data Menggunakan *Abstract Syntax Tree* ... 44
Gambar 4.5 Perancangan Konversi Data ke Bentuk Grafik menggunakan JUNG45
Gambar 4.6 *Activity Diagram* Cari Berkas 46
Gambar 4.7 *Activity Diagram* Bangkitkan *Program Dependence Graph* 47
Gambar 4.8 *Activity Diagram* Lihat Informasi *Node* 48
Gambar 4.9 *Activity Diagram* Ubah *Layout* Grafik 49
Gambar 4.10 *Activity Diagram* Ubah Ukuran Grafik 50
Gambar 4.11 *Activity Diagram* Ubah *Program Dependence Graph* 51
Gambar 4.12 *Activity Diagram* Lihat Tabel Ketergantungan 52
Gambar 4.13 *Class Diagram* 53
Gambar 4.14 Diagram Alir Ekstraksi Data 55
Gambar 4.15 Diagram Alir Proses Awal Ekstraksi Data 57
Gambar 4.16 Diagram Alir visit(MethodInvocation) 59
Gambar 4.17 Diagram Alir visit(MethodDeclaration) 59
Gambar 4.18 Diagram Alir visit(variableDeclarationFragment) 60
Gambar 4.19 Diagram Alir visit(Assigment) 61
Gambar 4.20 Diagram Alir visit(ReturnStatement) 63
Gambar 4.21 Diagram Alir visit(postfixExpression) 64

Gambar 4.22 Diagram Alir visit(prefixExpression) 64
Gambar 4.23 Diagram Alir visit(IfStatement) 65
Gambar 4.24 Diagram Alir visit(SimpleName) 67
Gambar 4.25 Diagram Alir AST_Model 68
Gambar 4.26 Diagram Alir Normalisasi Data 69
Gambar 4.27 Diagram Alir Buat Tabel Referensi 69
Gambar 4.28 Diagram Alir Buat Relasi Tabel 71
Gambar 4.29 Diagram Alir Hapus Redudansi 72
Gambar 4.30 Diagram Alir Buat Grafik 73
Gambar 4.31 Diagram Alir Create Tree 73
Gambar 4.32 *Sequence Diagram* Cari Berkas 74
Gambar 4.33 *Sequence Diagram* Bangkitkan *Program Dependence Graph* 75
Gambar 4.34 *Sequence Diagram* Lihat Informasi *Node* 77
Gambar 4.35 *Sequence Diagram* Ubah *Layout* Grafik 78
Gambar 4.36 *Sequence Diagram* Ubah Ukuran Grafik 79
Gambar 4.37 *Sequence Diagram* Ubah *Program Dependence Graph* 80
Gambar 4.38 *Sequence Diagram* Lihat Tabel Ketergantungan 81
Gambar 4.39 *Component Diagram* 82
Gambar 4.40 Rancangan Halaman Utama 83
Gambar 4.41 Rancangan Halaman *Program Dependence Graph* 84
Gambar 4.42 Rancangan Halaman Tabel Ketergantungan 85
Gambar 5.1 Diagram Alir Implementasi 86
Gambar 5.2 Proses Ekstraksi Data 88
Gambar 5.3 Proses Awal Ekstraksi Data 89
Gambar 5.4 Proses visit(MethodInvocation) 91
Gambar 5.5 Proses visit(MethodDeclaration) 91
Gambar 5.6 Proses visit(VariableDeclarationFragment) 92
Gambar 5.7 Proses visit(Assigment) 93
Gambar 5.8 Proses visit(ReturnStatement) 94
Gambar 5.9 Proses visit(postfixExpression) 95
Gambar 5.10 Proses visit(prefixExpression) 96
Gambar 5.11 Proses visit(IfStatement) 97

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Gambar 5.12 Proses visit(SimpleName) 98
Gambar 5.13 Proses AST_Model 99
Gambar 5.14 Proses Normalisasi Data 100
Gambar 5.15 Proses Buat Tabel Referensi 100
Gambar 5.16 Proses Buat Relasi Tabel 101
Gambar 5.17 Proses Hapus Redudansi 102
Gambar 5.18 Proses Buat Grafik 103
Gambar 5.19 Proses Create Tree 104
Gambar 5.20 Antarmuka Utama 105
Gambar 5.21 Antarmuka *Program Dependence Graph* 106
Gambar 5.22 Antarmuka Tabel Ketergantungan 107
Gambar 6.1 Pengujian Unit Ekstraksi Data 109
Gambar 6.2 *Flow Graph* Ekstraksi Data 109
Gambar 6.3 Pengujian Unit Buat Grafik 111
Gambar 6.4 *Flow Graph* Buat Grafik 111
Gambar 6.5 Pengujian Unit Create Tree 112
Gambar 6.6 *Flow Graph* Create Tree 113

DAFTAR TABEL

Tabel 2.1 Simbol *Use Case Diagram* 15
Tabel 2.2 Simbol *Activity Diagram* 17
Tabel 2.3 Simbol *Class Diagram*.... 18
Tabel 2.4 Simbol *Sequence Diagram* 20
Tabel 4.1 Identifikasi Aktor 31
Tabel 4.2 Daftar Kebutuhan Fungsional 31
Tabel 4.3 Skenario *Use Case* Cari Berkas 34
Tabel 4.4 Skenario *Use Case* Bangkitkan *Program Dependence Graph* 35
Tabel 4.5 Skenario *Use Case* Lihat Informasi *Node*.... 36
Tabel 4.6 Skenario *Use Case* Ubah *Layout* Grafik 37
Tabel 4.7 Skenario *Use Case* Ubah Ukuran Grafik 38
Tabel 4.8 Skenario *Use Case* Ubah *Program Dependence Graph*.... 399
Tabel 4.9 Skenario *Use Case* Lihat Tabel Ketergantungan 400
Tabel 4.10 Daftar Kebutuhan Non-Fungsional.... 40
Tabel 4.11 Daftar Komponen.... 41
Tabel 4.12 Integrasi Komponen dengan Kebutuhan Fungsional 43
Tabel 5.1 Spesifikasi Perangkat Keras.... 87
Tabel 5.2 Spesifikasi Perangkat Lunak.... 87
Tabel 6.1 Kasus Uji Ekstraksi Data 110
Tabel 6.2 Kasus Uji Buat Grafik.... 111
Tabel 6.3 Kasus Uji Create Tree.... 113
Tabel 6.4 Kasus Uji Cari berkas 114
Tabel 6.5 Kasus Uji Cari berkas Alur Alternatif 1 115
Tabel 6.6 Kasus Uji Bangkitkan *Program Dependence Graph*.... 115
Tabel 6.7 Kasus Uji Lihat Informasi *Node* 116
Tabel 6.8 Kasus Uji Ubah *Layout* Grafik 116
Tabel 6.9 Kasus Uji Ubah Ukuran Grafik 117
Tabel 6.10 Kasus Uji Ubah *Program Dependence Graph* 118
Tabel 6.11 Kasus Uji Lihat Tabel Ketergantungn 118
Tabel 6.12 Hasil Pengujian Validasi.... 119

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Tabel 6.13 Kasus Uji *Method* DistanceToLine 122
Tabel 6.14 Kasus Uji *Method* perspectiveTransform 123
Tabel 6.15 Kasus Uji *Method* getperspectiveTransform 124
Tabel 6.16 Kasus Uji *Method* unitize 125
Tabel 6.17 Kasus Uji *Method* norm 125
Tabel 6.18 Kasus Uji *Method* median 126
Tabel 6.19 Kasus Uji *Method* marker 126
Tabel 6.20 Kasus Uji *Method* createArray 127
Tabel 6.21 Kasus Uji *Method* get 127
Tabel 6.22 Kasus Uji *Method* compose 128
Tabel 6.23 Kasus Uji *Method* SubsumptionChain 129
Tabel 6.24 Kasus Uji *Method* wrap 129
Tabel 6.25 Kasus Uji *Method* reshap 130
Tabel 6.26 Kasus Uji *Method* getIndex 131
Tabel 6.27 Kasus Uji *Method* CDenseMatrix64F(double[][]) 132
Tabel 6.28 Kasus Uji *Method* CDenseMatrix64F(int,int,boolean,double) 133
Tabel 6.29 Kasus Uji *Method* CDenseMatrix64F(int,int) 133
Tabel 6.30 Kasus Uji *Method* reshape 134
Tabel 6.31 Kasus Uji *Method* innerProd 134
Tabel 6.32 Kasus Uji *Method* innerProdH 135
Tabel 6.33 Kasus Uji *Method* outerProd 136
Tabel 6.34 Kasus Uji *Method* outerProdH 137
Tabel 6.35 Kasus Uji *Method* solveU 138
Tabel 6.36 Kasus Uji *Method* solveL_diagReal 139
Tabel 6.37 Kasus Uji *Method* solveConjTranL_diagReal 140
Tabel 6.38 Kasus Uji *Method* invertLower 141
Tabel 6.39 Kasus Uji *Method* solveL 142
Tabel 6.40 Kasus Uji *Method* solveU2 143
Tabel 6.41 Kasus Uji *Method* undoTranspose 143
Tabel 6.42 Kasus Uji *Method* setMatrix 144
Tabel 6.43 Hasil Pengujian Akurasi 145

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB I
PENDAHULUAN

1.1 Latar Belakang

Dalam pengembangan perangkat lunak, pengembang sering melakukan duplikasi bagian kode program yang telah ada dengan cara *copy* dan *paste* dengan atau tanpa perubahan [DAN-12]. Proses ini sering dilakukan oleh pengembang perangkat lunak, karena lebih mudah melakukan duplikasi daripada membuat perangkat lunak sendiri [PRI-14]. Kegiatan duplikasi kode program membutuhkan usaha dan waktu yang sedikit, tetapi apabila terdapat kesalahan pada kode program yang diduplikasi maka kesalahan akan terduplikasi pula. Hal ini menyebabkan perawatan perangkat lunak menjadi lebih rumit. Kegiatan duplikasi kode program akan memudahkan pengembang dalam fase pembuatan, tetapi akan mempersulit pengembang pada fase perawatan [KRI-01].

Penggunaan kembali atau melakukan duplikasi kode program yang baik adalah dengan tidak mengambil bagian-bagiannya saja pada kode program yang tidak dirancang untuk digunakan kembali. Menggunakan kode program yang dirancang untuk digunakan kembali tidak akan menambah biaya perawatan perangkat lunak dan bahkan semakin meningkatkan kualitas dari perangkat lunak. Metode ini memungkinkan pengembang untuk menggunakan kembali aplikasi, komponen, atau fungsi dari suatu perangkat lunak tergantung tipe dari perangkat lunak yang akan digunakan kembali [SOM-11].

Pendeteksian duplikasi kode program dapat diketahui dengan cara mendapatkan informasi seperti masukan, keluaran, dan efek pada suatu kode program [PRI-14]. Untuk mendapatkan informasi tersebut dapat dilakukan analisis terhadap sebuah *source code* atau *bytecode*. *Source code* merupakan suatu set instruksi dalam bahasa pemrograman yang dapat dibaca secara jelas [HRM-10]. Sementara *bytecode* merupakan hasil kompilasi dari *source code* sehingga bahasa yang digunakan adalah bahasa pemrograman tingkat rendah dan tidak mudah dipahami oleh manusia [GEN-05]. Dalam *bytecode* terdapat beberapa informasi yang diubah seperti nama variabel sehingga informasi yang didapatkan tidak

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

sempurna [PRI-14]. Hubungan antara variabel yang terdapat pada kode program penting untuk diketahui karena mempengaruhi informasi masukan dan keluaran. Hubungan antar variabel dapat diketahui dengan cara menggambarkan ketergantungan variabel pada suatu kode program. Hal ini dapat dilakukan dengan membangun *Program Dependence Graph.*

Program Dependence Graph adalah representasi dari suatu kode program yang dapat memberikan informasi ketergantungan antar data pada setiap prosesnya dalam bentuk grafik [FER-87]. *Program Dependence Graph* dibuat berdasarkan pernyataan-pernyataan dalam kode yang disimbolkan dengan sebuah *node*. *Node* yang telah terbuat dapat dihubungkan dengan *node* lainnya apabila pernyatan dalam kode program saling terhubung pula. *Program Dependence Graph* juga dapat menggambarkan hubungan antara variabel didalam suatu kode program. Hubungan antara variabel sangat penting untuk diketahui karena kedua hal ini dapat menentukan variabel masukan dan keluaran pada suatu *method* [PRI-14].

Penelitian sebelumnya dilakukan oleh Priyambadha dan Rochimah tentang pendeteksian duplikasi kode progam menggunakan *Program Dependence Graph.* Dalam penelitian ini *Program Dependence Graph* digunakan untuk mengidentifikasi masukan, keluaran, dan efek yang ada dalam suatu *method.* Dalam menggunakan *Program Dependence Graph* untuk mendeteksi informasi-informasi tersebut masih terdapat kesalahan dalam mendeteksi nama variabel. Hal ini dikarenakan metode pembangkitan *Program Dependence Graph* berdasarkan dari *bytecode* data [PRI-14].

Dari permasalahan kesalahan pendeteksian nama variabel apabila menggunakan *bytecode* sebagai masukan, dapat diselesaikan menggunakan menggunakan *source code* karena tidak ada perubahan terhadap informasi variabel-variabel yang ada. Penggunaan *source code* juga tidak terpengaruh pada *compilers* yang digunakan. Penelitian ini dikhususkan untuk membangkitkan *Program Dependence Graph* dari bahasa pemrograman Java. Keunggulan dari bahasa Java diantaranya dapat dijalankan di banyak platform, bahasa berorientasi objek, struktur bahasa sederhana, menyediakan fitur *multithread*, bahasa tangguh dan dapat diperluas [HAR-11]. Metode pembangkitan *Program Dependence Graph* yang dilakukan pada penelitian ini bukan berdasarkan dari *bytecode* melainkan dari

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

source code. Oleh sebab itu, penulis membangun *Program Dependence Graph* untuk mendapatkan informasi ketergantungan antar variabel pada suatu *source code* dengan judul Pengembangan Kakas Bantu Pembangkitan *Program Dependence Graph* Berbasis Java. Pembangkitan *Program Dependence Graph* diharapkan mampu menggambarkan ketergantungan antar variabel yang berguna dalam identifikasi masukan, keluaran, dan efek.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang tersebut, maka dapat dirumuskan permasalahan sebagai berikut:

1. Bagaimana identifikasi ketergantungan antar variabel dari suatu *source code* yang direpresentasikan dengan *Program Dependence Graph*.
2. Bagaimana pembangunan kakas bantu pembangkitan *Program Dependence Graph* dengan menggunakan bahasa Java.
3. Bagaimana perbandingan tingkat akurasi yang akan diperoleh dalam melakukan identifikasi ketergantungan antar variabel pada suatu *source code* dan *bytecode*.

1.3 Batasan Masalah

Agar permasalahan yang dirumuskan tidak meluas, maka ditentukan batasan masalah sebagai berikut:

1. Pembangkitan *Program Dependence Graph* hanya berdasarkan ketergantungan data (*data dependence*) dan bukan berdasarkan ketergantungan kontrol (*control dependence*).
2. Masukan kode program berupa *source code* dengan bahasa Java.

1.4 Tujuan

Tujuan yang ingin dicapai dalam penelitian ini adalah sebagai berikut:

1. Melakukan identifikasi ketergantungan antar variabel dari suatu *source code* yang direpresentasikan dengan *Program Dependence Graph*.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

2. Melakukan pembangunan kakas bantu pembangkitan *Program Dependence Graph* dengan menggunakan bahasa Java.
3. Mengukur perbandingan tingkat akurasi yang akan diperoleh dalam melakukan identifikasi ketergantungan antar variabel pada suatu *source code* dan *bytecode*.

1.5 Manfaat

Manfaat yang diharapkan pada penelitian ini adalah sebagai berikut:

1. *Program Dependence Graph* dapat digunakan untuk mendapatkan informasi masukan, keluaran, dan efek. Informasi tersebut dapat digunakan untuk mendeteksi duplikasi kode.
2. Pengembang perangkat lunak dapat mengetahui representasi kode program dari ketergantungan antar variabel yang ada. Representasi ketergantungan antar variabel dapat mempermudah pengembang dalam memahami suatu kode program.

1.6 Sistematika Penulisan

Sistematika penulisan dalam skripsi ini adalah sebagai berikut:

BAB I Pendahuluan

Menguraikan tentang latar belakang permasalahan, menentukan rumusan masalah, batasan masalah, tujuan, manfaat, dan sistematika penulisan.

BAB II Kajian Pustaka dan Dasar Teori

Menguraikan tentang konsep dasar dan teori-teori yang mendasari pembuatan sistem pembangunan kakas bantu pembangkitan *Program Dependence Graph* berbasis Java.

BAB III Metode Penelitian

Menguraikan tentang metode dan langkah yang digunakan dalam penelitian yang terdiri dari studi literatur, perancangan sistem perangkat lunak, implementasi sistem perangkat lunak, pengujian dan analisis, serta penulisan laporan.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB IV **Perancangan**

Menguraikan tentang analisis kebutuhan dan perancangan sistem pembangunan kakas bantu pembangkitan *Program Dependence Graph* berbasis Java.

BAB V **Implementasi**

Menguraikan tentang implementasi pembangunan kakas bantu pembangkitan *Program Dependence Graph* berbasis Java sesuai dengan perancangan sistem yang telah dibuat.

BAB VI **Pengujian dan Analisis**

Menguraikan tentang proses dan hasil pengujian terhadao sistem yang telah diimplementasikan.

BAB VII **Penutup**

Menguraikan tentang kesimpulan yang diperoleh berdasarkan hasil penelitian dan memberikan saran untuk pengembangan sistem selanjutnya agar menjadi lebih baik.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB II
DASAR TEORI DAN TINJAUAN PUSTAKA

Pada bab ini menguraikan tentang tinjauan pustaka yang berisi kajian pustaka dan dasar teori. Kajian pustaka dan dasar teori yang dapat dijadikan sebagai acuan untuk memperoleh informasi mengenai objek penelitian sehingga dapat menunjang penulisan skripsi. Kajian pustaka memberikan informasi mengenai beberapa penelitian yang telah ada dan memiliki kemiripan dengan objek dan metode yang digunakan dalam penelitian. Dasar teori memberikan informasi tentang beberapa teori yang diperlukan untuk menyusun skripsi. Beberapa teori yang diperlukan pada penelitian ini adalah teori yang berkaitan dengan *Program Dependence Graph*, *source code*, Java *bytecode*, *Abstract Syntax Tree, Jung Java Library, Software Process Model,* dan *Unified Modelling Language* (UML).

2.1 Kajian Pustaka

Dalam penelitian ini diperlukan beberapa pustaka sebagai perbandingan tentang objek dan metode yang digunakan dalam penelitian ini dengan penelitian yang telah ada sebelumnya. Setiap pustaka tersebut menjelaskan judul, objek (*input* dan parameter), metode atau proses, dan *output* atau hasil dari sebuah penelitian.

Penelitian sebelumnya yang dikakukan oleh Bayu Priyambadha dan Siti Rochimah pada tahun 2014 dengan judul *Case Study on Semantic Clone Detection Based On Code Behavior.* Penelitian ini membahas tentang pendeteksian duplikasi kode berdasarkan perilakunya. Peneliti menggunakan informasi masukan, keluaran, dan efek dari suatu *method* . Informasi tersebut didapatkan dengan menggambarkan *method* tersebut kedalam *Program Dependence Graph*. Informasi masukan adalah variabel yang tidak memiliki ketergantungan, Keluaran adalah variabel terakhir yang bergantung pada variabel lain, dan efek adalah variabel yang mempengaruhi atribut pada method lain. Penelitian ini menggunakan Java *bytecode* sebagai masukan dan terdapat kesalahan dalam pendeteksian nama variabel bila variabel tersebut adalah *static* dan terdapat kesalahan kecil dalam pendeteksian variabel.

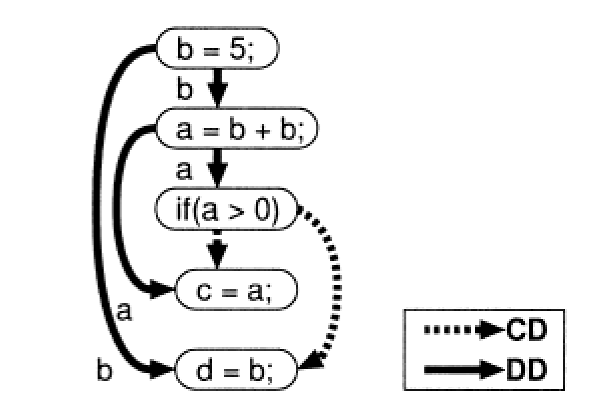
Dari beberapa referensi tersebut, maka pada penelitian ini penulis mengusulkan judul Pembangunan Kakas Bantu Pembangkitan *Program*

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Dependence Graph Berbasis Java. Objek yang digunakan pada penelitian ini yakni berkas *source code* bertipe Java yang berasal dari masukan pengguna kepada sistem. *Source code* masukan akan diproses dan digambarkan kedalam bentuk grafik. Jenis grafik yang digunakan dalam penelitian ini adalah *Program Dependence Graph* tanpa menggunakan *control dependence. Control dependence* tidak digunakan karena dalam mendapatkan informasi masukan, keluaran, dan efek pada suatu program dapat diketahui hanya dengan menggunakan *data dependence.* Hasil keluaran dari sistem ini adalah *Program Dependence Graph* yang dibangkitkan dari kode program beserta informasi seluruh *node* yang ada.

2.2 *Program Dependence Graph*

Program Dependence Graph merupakan sebuah grafik yang merepresentasikan program dimana *node* grafik berupa pernyataan dan ekspresi predikat termasuk juga operator dan operand. *Node* yang telah terbuat dapat dihubungkan dengan garis penghubung apabila pernyatan dalam *code* saling terhubung pula. Peran *node* yang merepresentasikan pernyataan dan predikat dalam *Program Dependence Graph* sangatlah penting pada saat terjadi perubahan posisi *node* atau terjadi penyederhanaan grafik. Perubahan atau penyederhanaan suatu *Program Dependence Graph* tidak diperbolehkan mempunyai deskripsi atau makna yang berbeda dari program sumber [FER-87]. Ketergantungan timbul akibat dari dua efek, yaitu *Control Dependence* dan *Data Dependence*.



Gambar 2.1 *Program Dependence Graph*
Sumber: [OHA-00]

Gambar 2.1 menunjukkan tentang contoh *Program Dependence Graph* yang terdiri dari *Control Dependence* dan *Data Dependence*. Dari gambar 2.1 terlihat bahwa *Program Dependence Graph* memiliki bagian node dan garis panah. *Node* merupakan sebuah pernyataan yang didapat dari sebuah kode program. CD merupakan simbol dari ketergantungan kontrol antar variabel, sementara DD merupakan simbol dari ketergantungan data antar variabel [OHA-00].

2.2.1 *Control Dependence*

Control Dependence merupakan sebuah grafik aliran kontrol yang memiliki *node* masukan unik (*start*) dan *node* keluaran unik (*stop*) dimana setiap *node* dalam grafik paling banyak memiliki dua penerus. *Node* dengan dua penerus memiliki atribut "T" (*true*) dan "F" (*false*) yang terhubung dengan *node* selanjutnya. Setiap *node* dalam grafik memiliki aliran jalan dari *start* menuju suatu *node* dan dari suatu *node* menuju *stop* [FER-87]. Dengan asumsi, pernyataan S_1 dan S_2 dalam *source* program p, *control dependence* S_1 ke S_2 terjadi apabila memenuhi kondisi berikut ini:

a. S_1 adalah predikat bersyarat.
b. Hasil dari S_1 menentukan apakah S_2 dijalankan atau tidak [OHA-00].

Ketergantungan pada *Control Dependence* berada di antara pernyataan dan predikat yang nilainya mengontrol secara langsung dari pelaksanaan pernyataan. Contoh dari ketergantungan ini adalah sebagai berikut:

$$if\,(A)\ then \qquad S_1$$
$$B = C * D \qquad S_2$$
$$endif$$

S_2 bergantung pada predikat A karena nilai A menentukan apakah S_2 dijalankan.

2.2.2 *Data Dependence*

Data Dependence merupakan sebuah grafik yang terdiri dari pernyataan dan predikat [FER-87]. Dengan asumsi, pernyataan S_1 dan S_2 dengan variabel v, *data dependence* S_1 ke S_2 terjadi apabila memenuhi kondisi berikut ini:

a. S_1 mendefinisikan v.

b. S_2 mengacu pada v.

c. Terdapat setidaknya satu jalur eksekusi dari S_1 ke S_2 tanpa mendefinisikan kembali variabel v yang telah ada [OHA-00].

Ketergantungan pada *Data Dependence* akan terjadi di antara dua pernyataan apabila terdapat variabel didalam satu pernyataan. Kedua pernyataan tersebut mungkin memiliki nilai yang berbeda atau salah jika dua pernyataan dibalik. Contoh dari ketergantungan ini adalah sebagai berikut:

$$A = B * C \qquad S_1$$

$$D = A * E + 1 \qquad S_2$$

S_2 bergantung pada S_1 karena pada saat S_2 dijalankan sebelum S_1 maka hasil S_2 akan menghasilkan nilai A yang salah [FER-87].

2.3 *Source Code*

Source code adalah suatu set instruksi yang diwakili dalam bahasa pemrograman (misalnya Java dan C#) untuk mengontrol perangkat keras. Instruksi biasanya ditulis menggunakan serangkaian angka, huruf, dan simbol-simbol khusus [BIN-07]. Setelah dikompilasi *source code* berubah menjadi *bytecode. Source code* juga dapat diartikan sebagai deskripsi informasi lengkap tentang sistem perangkat lunak yang dapat dieksekusi [HRM-10].

2.4 Java *Bytecode*

Java *bytecode* adalah bahasa beorientasi objek tingkat rendah. Java *bytecode* dapat dibuat dengan cara melakukan kompilasi terhadap *source code* berbahasa Java. Java *bytecode* tidak memiliki struktur lingkup eksplisit dan menggunakan tumpukan (*stack*) operan untuk menahan hasil komputasi menengah. Sebuah program Java *bytecode* terdiri dari satu set kelas, dimana masing-masing kelas mendefinisikan satu set *method* dan *field*. Sebuah *method* berisi urutan pernyataan (*statements*) Java *bytecode*. *Stack* dan variabel lokal digunakan untuk menjalankan *method* yang dilambangkan dengan $S_m$dan L_m dan keduanya mulai dari indeks 0 [GEN-05].

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

```
while ((x*=2)>0) {
  y = y - 1;
  x = y;
  z = z + 1;
}

y = 0;
```

```
0: iload 1      10: iload 0
1: iconst 2     11: istore 1
2: imul         12: iload 2
3: istore 1     13: iconst 1
4: iload 1      14: iadd
5: ifle 17      15: istore 2
6: iload 0      16: goto 0
7: iconst 1     17: iconst 0
8: isub         18: istore 0
9: istore 0
```

```
Block 0
iload 1
iconst 2
imul
istore 1
iload 1

Block 1
goon_ifle
iconst 0
istore 0

Block 2
goon_ifgt
iload 0
iconst 1
isub
istore 0
iload 0
istore 1
iload 2
iconst 1
iadd
istore 2
```

Gambar 2.2 Java *Bytecode*
Sumber: [GEN-05]

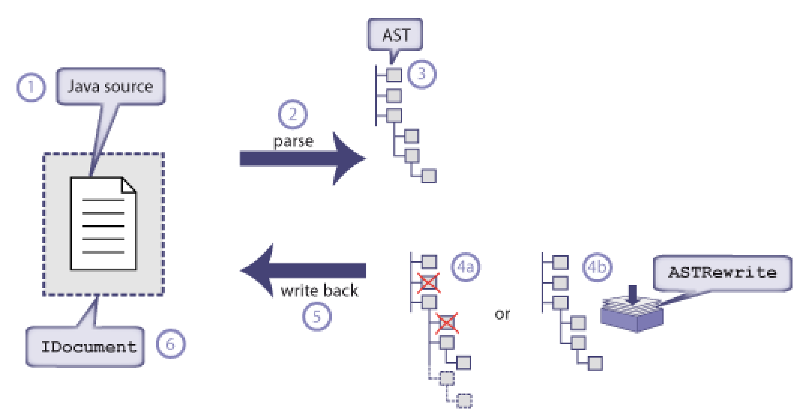
Gambar 2.2 menunjukkan perubahan dari Java *method* (kiri) diterjemahkan menjadi Java *bytecode* (tengah) dan *Control Flow Graph* (kanan).

2.5 *Abstract Syntax Tree*

Abstract Syntax Tree adalah kerangka kerja (framework) dasar yang telah diimplementasikan pada banyak kakas bantu dari perangkat lunak Eclipse IDE, termasuk refactoring, Quick Fix dan Quick Assist. *Abstract Syntax Tree* mengubah Java *source code* ke dalam bentuk skema pohon. Skema pohon ini dapat digunakan dan handal untuk menganalisis dan memodifikasi pemrograman dari sumber berbasis teks (*source code*). *Abstract Syntax Tree* juga digunakan oleh Eclipse dalam melihat *source code*, setiap elemen *source code* sepenuhnya direpresentasikan kedalam bentuk skema pohon AST dalam bentuk *node*. *Node* ini merupakan subclass dari class AST*Node*. Setiap subclass *node* mempunyai peran masing-masing misalnya ada *node* yang dikhususkan untuk menangani deklarasi *method* (`MethodDeclaration`), deklarasi variabel (`VariableDeclaration Fragment`), dan lain-lain. Salah satu tipe *node* yang sangat sering digunakan adalah `SimpleName`. Sebuah `SimpleName` adalah setiap string Java *source* yang bukan kata kunci, tipe Boolean (*true* atau *false*) atau tipe null. Misalnya, di i = 6 + j ;, i dan j diwakili oleh `SimpleNames` [KUH-06].

2.5.1 Alur Kerja *Abstract Syntax Tree*

Alur kerja sistem yang menggunakan AST dapat dilihat pada gambar 2.3.



Gambar 2.3 Alur Kerja *Abstract Syntax Tree*
Sumber: [KUH-06]

Keterangan Gambar 2.3:

1. *Java source*

   Masukan yang dapat ditangani oleh AST adalah sebuah berkas bertipa java atau suatu set instruksi bertipe java.

2. *Parse*

   Source code yang di deskripsikan pada langkah 1 akan melalui proses *parse*. Untuk melakukan proses *parse*, telah disediakan *class* oleh `org.eclipse.jdt.core.dom.ASTParser`.

3. Hasil AST pada langkah 2

   Hasil AST pada langkah 2 adalah skema pohon yang dibangun dari masukan *source code* pada langkah 1.

4. Manipulasi AST

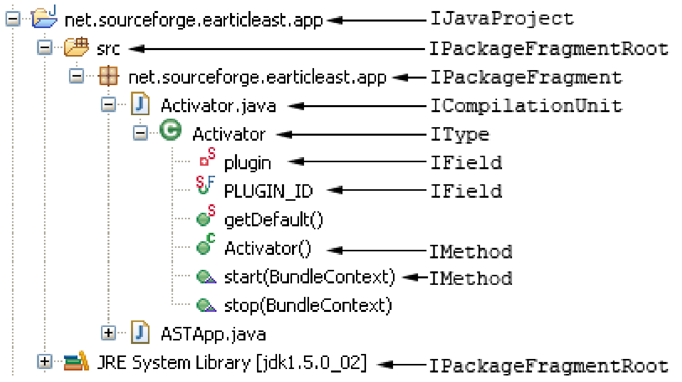
   Jika hasil AST pada langkah 3 perlu untuk diubah, hal ini dapat dilakukan dalam 2 cara yaitu:

   a. Dengan melakukan modifikasi langsung pada AST
   b. Dengan mencatat modifikasi dalam protokol terpisah. Protokol ini ditangani oleh sebuah contoh dari `ASTRewrite`.

2.5.2 Model Java *Abstract Syntax Tree*

Model Java merupakan proyek Java dalam struktur pohon yang divisualisasi oleh “Package Explorer”.

repository.ub.ac.id UNIVERSITAS BRAWIJAYA



Gambar 2.4 Model Java *Abstract Syntax Tree*
Sumber: [KUH-06]

Keterangan Gambar 2.4:

- `IJavaProject`
- Adalah simpul dari model Java dan merupakan proyek Java. `IjavaProject` berisi `IPackageFragmentRoots` sebagai *node* anak.
- `IPackageFragmentRoot`

  Merupakan paket yang dapat menjadi *source* atau folder kelas proyek, zip atau file .jar. `IPackageFragmentRoot` dapat menyimpan *source* atau file biner.
- `IPackageFragment`

  Sebuah paket tunggal yang berisi `ICompilationUnits` atau `IClassFiles`, tergantung pada apakah `IPackageFragmentRoot` adalah tipe *source* atau jenis *bytecode*.
- `ICompilationUnit`

  Merupakan file Java *source code*.
- `IImportDeclaration, iType, Ifield, IInitializer, IMethod`

  Merupakan anak-anak `ICompilationUnit`. Informasi yang diberikan oleh *node* ini tersedia dari AST.

2.6 *Java Universal Network/Graph Framework* (JUNG)

Java Universal Network/Graph Framework (JUNG) adalah sebuah bingkai kerja yang digunakan untuk pemodelan, analisis, dan visualisasi data yang dapat

diwakili sebagai grafik atau jaringan. JUNG dibuat dengan menggunakan bahasa java yang memungkinkan sistem berbasis JUNG memanfaatkan layanan dan fungsi jung secara penuh dengan menggunakan Java API.

Arsitektur JUNG dirancang untuk mendukung berbagai representasi dari entitas dan relasinya, seperti grafik langsung dan tak langsung, grafik multi-modal, grafik dengan tepi paralel, dan *hypergraphs*. JUNG menyediakan mekanisme untuk grafik anotasi, entitas, dan hubungan dengan metadata. JUNG juga memfasilitasi penciptaan alat analitik untuk set data yang kompleks yang dapat memeriksa hubungan antar entitas serta metadata yang melekat pada setiap entitas dan relasinya.

JUNG juga menerapkan implementasi dari sejumlah algoritma dari teori grafik, data mining, dan analisis jaringan sosial, seperti pengelompokan yang rutin, dekomposisi, optimasi, generasi grafik acak, analisis statistik, dan perhitungan jarak jaringan, *flows*, dan langkah-langkah penting (*centrality*, PageRank, HITS, dll).

Bingkai kerja JUNG juga menyediakan representasi data secara visual yang digunakan untuk memudahkan membangun kakas bantu yang interaktif dan detail pada setiap jaringan data. Pengguna dapat menggunakan salah satu algoritma tata letak yang disediakan, atau menggunakan kerangka kerja untuk membuat layout mereka sendiri. Selain itu, mekanisme penyaringan yang disediakan memungkinkan pengguna untuk fokus pada algoritma atau bagian tertentu dari grafik saja.

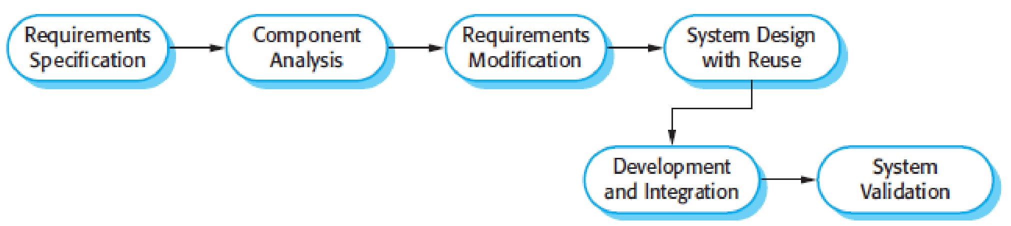
2.7 *Software Process Model*

Software Process Model adalah penggambaran dari langkah-langkah pembuatan perangkat lunak dari spesifikasi perangkat lunak sampai evolusi perangkat lunak. Setiap model hanya menggambarkan informasi tertentu pada suatu proses dan hanya menyediakan informasi parsial mengenai proses tersebut. Terdapat beberapa contoh *Software Process Model* diantaranya adalah *waterfall development, incremental development*, dan *reuse-oriented engineering*. Ketiga model tersebut dibuat secara generik sehingga ketiga model tersebut tidak menjelaskan secara definitif dari proses perangkat lunak. Sebaliknya, ketiga model

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

tersebut adalah abstraksi dari suatu proses yang ada sehingga dapat disesuaikan dan dikembangkan dengan proses perangat lunak yang lebih spesifik [SOM-11].

Pendekatan *reuse-oriented software engineering* didasarkan dari banyaknya komponen yang dapat digunakan kembali. Pendekatan ini lebih berfokus pada integrasi antar komponen yang ada daripada membuat perangkat lunak dari awal. Pendekatan ini mempunyai beberapa kelebihan seperti mengurangi biaya perawatan dan pengembangan, meningkatkan kualitas, dan mempercepat waktu pengembangan [SOM-11].



Gambar 2.5 *Reuse-Oriented Software Engineering* [SOM-11]

Gambar 2.5 menunjukkan proses-proses yang berada dalam *reuse-oriented software engineering*. Tahap-tahap tersebut adalah sebagai berikut:

1. Analisis komponen

   Menurut spesifikasi kebutuhan yang telah ditentukan, perlu dilakukan pencarian untuk komponen untuk melakukan implementasi spesifikasi tersebut. Umumnya, tidak ada yang sama persis dan komponen yang dapat digunakan hanya menyediakan beberapa fungsi yang diperlukan.

2. Modifikasi Kebutuhan

   Selama tahap ini, kebutuhan yang dianalisis menggunakan informasi tentang komponen yang telah ditemukan. Kemudian dimodifikasi untuk mencerminkan komponen yang tersedia. Jika terdapat modifikasi yang tidak mungkin untuk diterapkan, kegiatan analisis komponen dapat dilakukan kembali untuk mencari solusi alternatif.

3. Desain Sistem dengan *Reuse*

   Selama tahap ini, *framework* yang telah ada digunakan kembali. Para desainer memperhitungkan komponen yang digunakan kembali dan mengatur *framework* untuk memenuhi kebutuhan yang diperlukan.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Beberapa perangkat lunak baru mungkin harus dirancang jika komponen perangkat lunak yang dapat digunakan kembali tidak tersedia.

4. Pengembangan dan Integrasi Perangkat Lunak

Perangkat lunak yang tidak dapat diperoleh secara eksternal akan dikembangkan dengan mempertimbangkan komponen dan biaya sistem yang akan diintegrasikan untuk menciptakan sistem baru. Integrasi sistem, dalam model ini dapat menjadi bagian dari proses pembangunan dan bukan kegiatan terpisah.

Reuse-oriented software engineering memiliki keuntungan yang dapat mengurangi biaya dan resiko pengembangan perangkat lunak sebab mengurangi jumlah perangkat lunak yang akan dibangun [SOM-11].

2.8 *Unified Modelling Language* (UML)

Unified Modelling Language merupakan bahasa visual pemodelan untuk membangun perangkat lunak melalui teknik pemrograman berorientasi objek dengan menggunakan diagram dan teks-teks pendukung. UML digunakan untuk memenuhi kebutuhan visual pemodelan perangkat lunak seperti menspesifikasikan, menggambarkan, membangun, dan membuat dokumentasi dari sistem perangkat lunak.

2.8.1 *Use Case Diagram*

Use case diagram merupakan pemodelan untuk menggambarkan tingkah laku (*behavior*) sistem yang akan dibuat. *Use case diagram* berguna untuk menggambarkan interaksi antar aktor sebagai pengguna sistem serta untuk mengetahui fungsi-fungsi yang ada dalam sistem dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut [ROS-11]. Terdapat dua hal utama dalam *use case diagram* yaitu aktor dan *use case*. Tabel 2.1 menunjukkan simbol-simbol yang ada pada diagram *use case*.

Tabel 2.1 Simbol *Use Case Diagram*

Simbol	Deskripsi

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Use case	Sebuah unit yang disediakan oleh sistem, memiliki fungsionalitas tersendiri, dan dapat berhubungan dengan aktor atau unit-unit yang lain. Pemberian nama *use case* biasanya menggunakan kata kerja di awal frase, seperti "buat grafik".
Aktor	Aktor adalah orang, proses, atau sistem lain yang berada diluar sebuah sistem. Aktor dapat berinteraksi dengan sistem. Pemberian nama aktor biasanya menggunakan kata benda pada awal frase.
Asosiasi	Melambangkan komunikasi yang ada antara akor dan *use case*.
Ekstensi	Melambangkan hubungan tambahan suatu *use case* ke sebuah *use case* baru dimana suatu *use case* dapat berdiri sendiri tanpa adanya *use case* tambahan.
Generalisasi	Melambangan hubungan umum-khusus antara dua *use case* dimana salah satu *use case* memiliki fungsi lebih umum daripada *use case* lainnya.
Include	Melambangkan hubungan tambahan suatu *use case* ke sebuah *use case* baru dimana suatu *use case* tidak dapat berdiri sendiri tanpa adanya *use case* tambahan. *Use case* tambahan berperan sebagai syarat dijalankannya suatu *use case.*

Sumber: [ROS-11]

2.8.2 *Activity Diagram*

Activity diagram merupakan diagram yang menggambarkan aktivitas atau aliran kerja (*workflow*) dari sebuah sistem yang ada pada perangkat lunak. *Activity diagram* berfungsi untuk menggambarkan aktivitas yang dapat dilakukan oleh

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

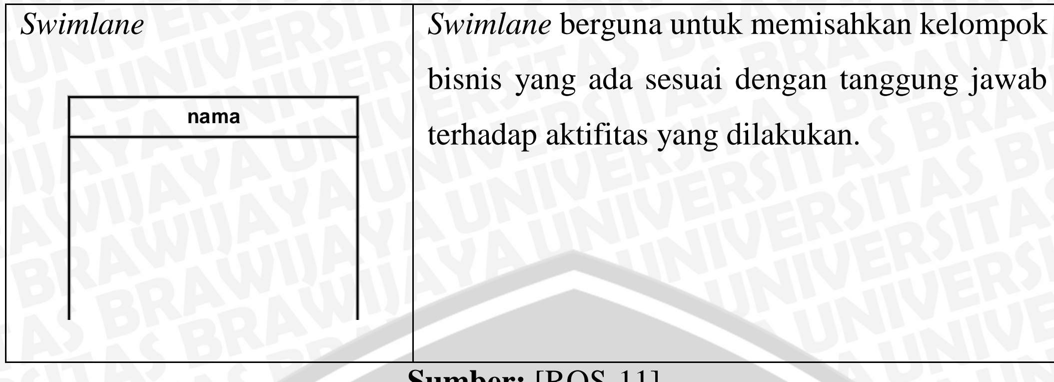
sistem bukan apa yang dilakukan oleh aktor. *Activity diagram* juga digunakan untuk mendefinisikan hal-hal berikut:

- Rancangan proses bisnis yang mendefinisikan urutan aktivitas sistem.
- Urutan atau pengelompokan tampilan dari sistem dengan setiap aktivitas dianggap memiliki sebuah rancangan antarmuka.
- Rancangan pengujian yang menganggap setiap aktivitas memerlukan sebuah pengujian yang perlu mendefinisikan kasusu ujinya terlebih dahulu.
- Rancangan menu yang ditampilkan pada perangkat lunak.

Simbol-simbol yang ada pada diagram activity ditunjukkan pada tabel 2.2.

Tabel 2.2 Simbol *Activity Diagram*

Simbol	Deskripsi
Status awal	Melambangkan awal dari aktifitas suatu sistem.
Aktifitas nama aktifitas	Melambangkan suatu aktifitas dalam sistem. Pemberian nama biasanya dengan menggunakan kata kerja pada awal frase.
Percabangan	Percabangan digunakan apabila terdapat lebih dari satu aktifitas yang bisa dikerjakan
Penggabungan	Penggabungan digunakan dimana aktifitas-aktifitas dapat digabungkan.
Status akhir	Melambangkan akhir dari aktifitas suatu sistem.

Swimlane 	*Swimlane* berguna untuk memisahkan kelompok bisnis yang ada sesuai dengan tanggung jawab terhadap aktifitas yang dilakukan.

Sumber: [ROS-11]

2.8.3 *Class Diagram*

Class Diagram merupakan diagram yang menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang ada dalam pembangunan sistem. Suatu kelas terdiri dari atribut dan metode (operasi) dimana atribut merupakan variabel yang dimiliki kelas dan metode merupakan fungsi yang dimiliki oleh kelas. Kelas yang ada pada struktur sistem harus dapat melakukan fungsi-fungsi sesuai dengan kebutuhan sistem yang dirancang. Susunan struktur *class diagram* yang baik adalah memiliki jenis-jenis kelas sebagai berikut [ROS-11]:

- Kelas main
  Kelas yang menyediakan fungsi awal untuk dieksekusi ketika sistem dijalankan.
- Kelas yang menangani tampilan sistem (antarmuka).
  Kelas yang menyediakan memiliki fungsi untuk mendefinisikan dan mengatur tampilan sistem kepada pengguna.
- Kelas yang diambil dari pendefinisian *use case* (*controller*)
  Kelas yang menyediakan kelas proses yang menangani proses bisnis dari pendefinisian *use case* pada perangkat lunak.
- Kelas yang diambil dari pendefinisian data (model)
  Kelas yang digunakan untuk melindungi data menjadi sebuah kesatuan yang disimpan ke basis data.

Simbol-simbol yang ada pada diagram kelas ditunjukkan pada tabel 2.3.

Tabel 2.3 Simbol *Class Diagram*

Simbol	Deskripsi

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Kelas Nama kelas +atribut +operasi()	Kelas yang terdapat dalam sistem
Antar Muka nama_interface	Seperti dengan konsep *interface* dalam pemrograman berorientasi objek.
Asosiasi	Hubungan antar kelas secara umum, hubungan asosiasi biasanya disertai dengan *multiplicity*.
Asosiasi berarah	Hubungan antar kelas dimana suatu kelas digunakan oleh kelas lain, hubungan asosiasi biasanya disertai dengan *multiplicity*.
Generalisasi	Hubungan antar kelas dengan hubungan umum-khusus
Ketergantungan	Hubungan antar kelas dimana suatu kelas bergantung pada kelas yang lain.
Agregasi	Hubungan antar kelas yang mempunyai makna hubungan semua-bagian (*whole-part*).

Sumber: [ROS-11]

2.8.4 *Sequence Diagram*

Sequence diagram merupakan diagram yang menggambarkan tingkah laku objek pada *use case* dengan mendefinisikan waktu hidup objek dan pesan yang dikirimkan dan diterima antar objek. Untuk menggambar *use case diagram* diperlukan objek-objek yang terlibat dalam sebuah use case beserta metode-metode yang diinstansiasi menjadi objek tersebut dalam suatu kelas. Banyaknya diagram sequence yang dibuat adalah minimal sebanyak pendefinisian *use case* yang telah

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

di definisikan. Simbol-simbol yang ada pada *sequence diagram* ditunjukkan pada tabel 2.4.

Tabel 2.4 Simbol *Sequence Diagram*

Simbol	Deskripsi
Status awal nama	Aktor adalah orang, proses, atau sistem lain yang berada diluar sebuah sistem. Aktor dapat berinteraksi dengan sistem. Pemberian nama aktor biasanya menggunakan kata benda pada awal frase.
Objek Object1	Objek dalam sistem dan dapat berinteraksi dengan pesan yang ada.
Garis Hidup	Garis yang menyatakan waktu hidup suatu objek dalam sistem.
Waktu aktif	Garis yang menyatakan waktu aktif suatu objek dalam sistem. Dalam waktu aktif ini sebuah objek melakukan sebuah tahapan dalam interaksi pesan.
Pesan tipe *create* «create»	Pesan bertipe *create* menyatakan bahwa suatu objek menginstansiasi atau membuat objek baru. Arah panah menunjukkan objek yang dibuat.
Pesan tipe *call* 1 : nama_metode()	Pesan tipe *call* menyatakan bahwa suatu objek memnggil metode atau prosedur pada objek lain.
Pesan tipe *send* masukan	Pesa tipe *send* menyatakan bahwa suatu objek mengirimkan informasi ke objek yang dituju.
Pesan tipe *return* keluaran	Pesan tipe *return* menyatakan objek yang telah menjalankan suatu proses dan menghasilkan

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

	suatu nilai untuk dikempalikan kepada objek yang dituju.
Destroy «destroy»	Pesan tipe *destroy* menyataka bahwa suatu objek mengakhiri hidup dari objek yang dituju.

Sumber: [ROS-11]

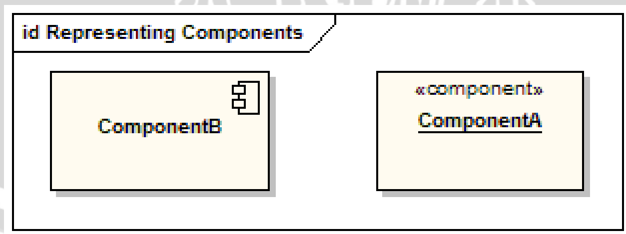
2.8.5 *Component Diagram*

Component diagram menggambarkan bagian perangkat lunak, komponen, pustaka, dan sebagainya yang akan dibuat menjadi sebuah sistem. *Component diagram* lebih abstrak daripada *class diagram* dan umumnya *component diagram* diimplementasikan dengan satu atau banyak kelas. *Component diagram* dapat menggambarkan seluruh sistem yang ada [SPA-15].

Component diagram mirip dengan *package diagram*, karena *component diagram* mendefisinikan batas-batas dan digunakan sebagai sekelompok elemen-elemen pada struktur logis. *Component diagram* dapat memodelkan masing-masing elemen pada perangkat lunak, sedangkan *package diagram* hanya dapat menampilkan elemen-elemen yang bersifat umum [SPA-15].

- Representasi Komponen

  Komponen digambarkan sebagai *classifier* persegi panjang dengan kata kunci «komponen», komponen dapat ditampilkan sebagai persegi panjang secara opsional dengan ikon komponen di sudut kanan.

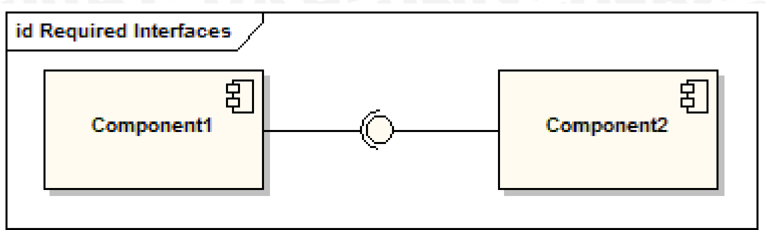


Gambar 2.6 Representasi Komponen

Sumber: [SPA-15]

- *Assembly Connector*

Assembly connector merupakan penghubung komponen yang diperlukan antarmuka (Komponen 1) dengan antarmuka yang tersedia pada komponen lain (Komponen 2). Hal ini memungkinkan satu komponen dapat menyediakan layanann yang membutuhkan komponen lain.



Gambar 2.7 *Assembly Connector*

Sumber: [SPA-15]

2.9 *Java System Program Dependence Graph* API

Dalam banyak hal, program tidak hanya terdiri dari satu prosedur tetapi sejumlah besar prosedur. Jika hal ini terjadi, sebuah *Program Dependence Graph* tidak cukup untuk menampilkan seluruh informasi dalam program, maka untuk mengatasi masalah ini *Program Dependence Graph* diperpanjang untuk *System Dependence Graph* yang terdiri dari kumpulan *Program Dependence Graph*. Perpanjangan ini menangkap konteks *method call* dan membutuhkan paket tambahan *node* dan *edge* serta representasi tambahan Java Class.

2.10 *Cyclomatic Complexity*

Cyclomatic complexity adalah pengukuran perangkat lunak yang menunjukkan kompleksitas program. *Cyclomatic Complexity* merupakan ukuran kuantitatif dari jalur independen yang diperoleh dari kode program dan telah dikembangkan oleh Thomas J. McCabe, Sr. pada tahun 1976. *Cyclomatic complexity* dihitung dengan menggunakan *control flow graph*, *node* dari grafik sesuai dengan kelompok dari perintah program dan *edge* menghubungan antara dua *node* [MCC-76].

Cyclomatic complexity memiliki jumlah jalur independen linear di dalamnya. Misalnya jika terdapat kode program yang tidak mengandung pernyataan kontrol aliran program (kondisional atau poin keputusan) seperti

pernyataan *IF* maka akan terdapat satu kompleksitas harena hanya ada satu jalur melalui kode. Jika kode memiliki satu pernyataan *IF* tunggal maka akan terdapat dua jalur melalui kode, yaitu jalur pertama untuk pernyataan *IF* bernilai *TRUE* dan jalur kedua untuk pernyataan *IF* bernilai *FALSE*. Jika kode memiliki IF dengan dua kondisi maka akan menghasilkan empat kompleksitas, yaitu dua jalur untuk setiap cabang dalam dan luar kondisi bersyarat. Secara matematis, *cyclomatic complexity* didefinisikan sebagai berikut [MCC-76].

$$M = E - N + 2P \tag{2.1}$$

Dimana:

M : *Cyclomatic complexity*.

E : Jumlah *edge* dalam grafik (garis penghubung antar *node*).

N : Jumlah *node* dalam grafik.

P : Jumlah blok yang tidak terhubung.

Pada penelitian ini *cyclomatic complexity* digunakan pada bab pengujian untuk mendapatkan jumlah jalur independen pada kode program. Jalur independen digunakan untuk menentukan kasus uji validitas kode program yang dibuat.

2.11 Akurasi

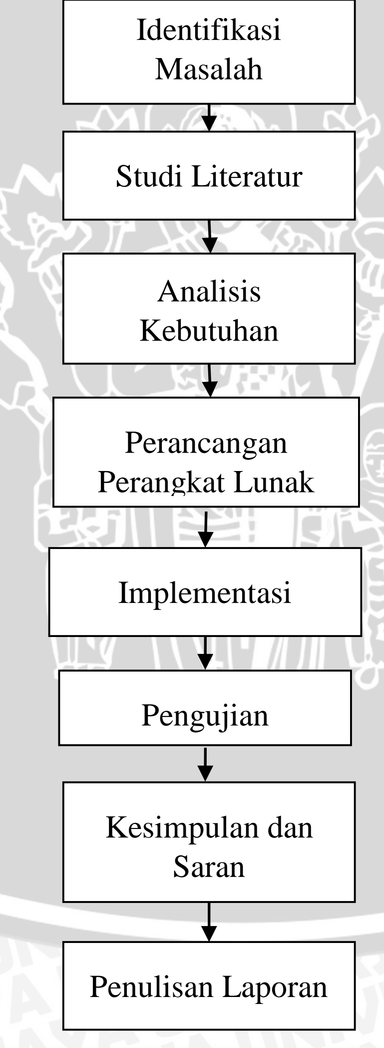
Akurasi merupakan nilai perbandingan jumlah data hasil klasifikasi yang relevan dengan jumlah seluruh data yang ada. Akurasi diperoleh dengan perhitungan sebagai berikut [NUG-14].

$$Akurasi = \frac{\sum data\ relevan}{\sum seluruh\ data} * 100\% \tag{2.2}$$

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB III
METODOLOGI PENELITIAN

Bab ini menguraikan tentang langkah-langkah yang digunakan dalam pengerjaan penelitian pembangunan kakas bantu pembangkitan *Program Dependence Graph* berbasis Java yang terdiri dari identifikasi masalah, studi literatur, analisis kebutuhan, perancangan sistem, implementasi sistem, pengujian, pengambilan kesimpulan dan saran serta penulisan laporan. Gambar 3.1. menunjukkan diagram alir metodologi penelitian yang dilakukan pada penelitian ini.



Gambar 3.1 Diagram Alir Metodologi Penelitian

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

3.1 Identifikasi Masalah

Identifikasi masalah merupakan tahap awal yang dilakukan pada penelitian terhadap objek yang akan dimuat dalam penelitian. Dari latar belakang permasalahan penelitian dapat diambil identifikasi masalah objek yang akan dimuat, kemudian mencari penelitian yang terkait dengan objek tersebut serta solusi yang telah digunakan selama ini untuk mengatasi permasalahan tersebut. Permasalahan yang diambil pada penelitian ini adalah bagaimana cara melakukan pembangunan kakas bantu pembangkitan *Program Dependence Graph* berbasis Java.

3.2 Studi Literatur

Dalam sebuah penelitian, studi literatur digunakan sebagai sumber acuan dalam penulisan skripsi dan pengembangan sistem agar dapat mempelajari dan memahami secara mendalam terkait teori-teori dasar keilmuan yang akan menjadi objek penelitian yang dilakukan. Teori-teori mengenai metode sistematika pembangunan kakas bantu pembangkitan *Program Dependence Graph* berbasis Java menjadi dasar penelitian yang diperoleh dari buku, jurnal, *ebook*, penelitian sebelumnya, situs internet serta literatur lain yang berkaitan. Teori pustaka yang berkaitan dengan penelitian skripsi ini meliputi:

1. *Program Dependence Graph*
    a. *Control Dependence*
    b. *Data Dependence*
2. *Source Code*
3. Java *Bytecode*
4. Java
5. *Abstract Syntax Tree*
6. JUNG
7. *Reuse-Oriented Software Engineering*
8. *Unified Modelling Language* (UML)
    a. *Use Case Diagram*
    b. *Activity Diagram*
    c. *Class Diagram*

d. *Sequence Diagram*

e. *Component Diagram*

Setelah melakukan tahap studi literatur kemudian dilakukan tahap analisis kebutuhan yang diperlukan oleh sistem.

3.3 Analisis Kebutuhan

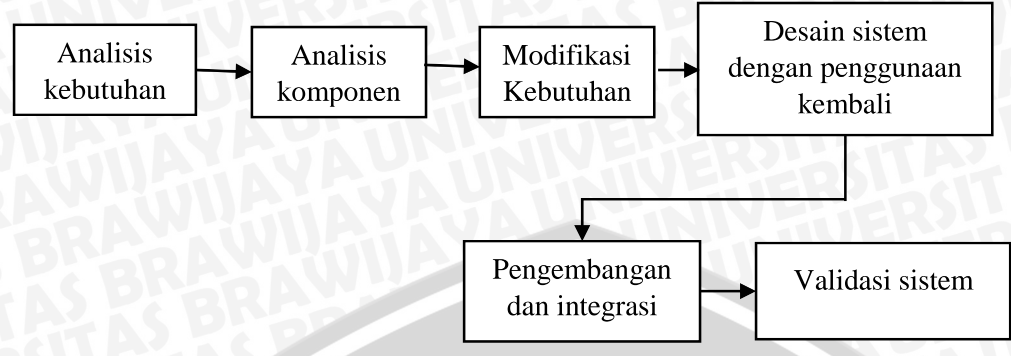
Analisis kebutuhan merupakan proses yang dilakukan untuk mengetahui kebutuhan apa saja yang diperlukan untuk sistem yang dibangun agar dapat melakukan pembangunan kakas bantu pembangkitan Program *Dependence Graph*. Metode analisis yang digunakan dalam penelitian ini adalah *object-oriented analysis*. Analisis kebutuhan terdiri dari beberapa tahapan diantaranya yaitu gambaran umum sistem, identifikasi aktor, kebutuhan fungsional, kebutuhan non-fungsional, dan analisis komponen. Berdasarkan kebutuhan fungsional yang telah dibuat, lalu dibuat diagram *use case diagram* untuk mendeskripsikan kebutuhan (*requirements*) perangkat lunak dari sudut pandang pengguna. Setiap kebutuhan fungsional yang terdapat pada *use case diagram* akan dijelaskan lebih rinci dengan skenario *use case*.

3.4 Perancangan Perangkat Lunak

Perancangan perangkat lunak merupakan dasar proses implementasi dengan menggunakan pemodelan proses perangkat lunak. Pada penelitian ini, model yang akan digunakan adalah model *reuse development*. Model *reuse development* diterapkan karena model pengembangan perangkat lunak ini memiliki banyak kelebihan dibandingkan dengan pengembangan perangkat lunak mulai dari awal. Model *reuse development* juga memiliki kelebihan lainnya yaitu komponen yang akan di gunakan kembali telah di uji sehingga kemungkinan kesalahan yang terjadi lebih kecil. Gambar 3.2 menunjukkan model *reuse development* yang diterapkan dalam penelitian ini.

Perancangan perangkat lunak terdiri dari beberapa tahapan diantaranya adalah perancangan arsitektur, perancangan integrasi komponen, perancangan *activity diagram*, perancangan *class diagram*, perancangan *sequence diagram*, perancangan *component diagram*, dan perancangan antarmuka.

repository.ub.ac.id UNIVERSITAS BRAWIJAYA



Gambar 3.2 Model *Reuse Development*

Perancangan arsitektur digunakan untuk mengetahui gambaran kinerja sistem secara keseluruhan. Perancangan integrasi komponen digunakan untuk mendapatkan informasi yang dibutuhkan untuk membangkitkan *Program Dependence Graph* dan melakukan konversi data ke bentuk grafik *Program Dependence Graph*. Perancangan *activity diagram* digunakan untuk menganalisis *use case* untuk menggambarkan aktivitas yang dibutuhkan. Perancangan class diagram digunakan untuk mengidentifikasi kelas-kelas yang berada dalam sistem. Peancangan *sequence diagram* digunakan untuk menjelaskan interaksi antar objek yang disusun dalam urutan waktu. Perancangan *component diagram* digunakan untuk menggambarkan komponen-komponen yang berada dalam sistem. Perancangan antarmuka digunakan sebagai media interaksi antara pengguna dan sistem.

3.5 Implementasi

Implementasi sistem merupakan proses penerapan pembuatan apliaksi berdasarkan pada proses analisis dan perancangan yang telah dilakukan pada tahap sebelumnya. Implementasi perangkat lunak dilakukan dengan menggunakan bahasa pemrograman berorientasi objek yaitu menggunakan bahasa pemrograman Java dengan *software* Netbeans IDE 8.0. Implementasi sistem ini meliputi:

1. Penjabaran spesifikasi lingkungan pengembangan perangkat lunak.
2. Implementasi ekstraksi data.
3. Implementasi konversi data ke bentuk grafik.
4. Implementasi antarmuka.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

3.6 Pengujian

Pengujian merupakan proses yang dilakukan dengan tujuan untuk mengetahui apabila sistem yang dibuat telah benar dan sesuai dengan spesifikasi kebutuhan yang melandasinya. Strategi pengujian perangkat lunak yang digunakan yaitu pengujian unit, pengujian validasi, dan pengujian akurasi. Metode pengujian yang digunakan adalah *white-box testing* dan *black-box testing*. Pada tahap pengujian unit digunakan metode *white-box testing* dengan teknik *basis path*. Sementara untuk pengujian akurasi dan validasi digunakan metode *black-box testing*. Kemudian dilakukan analisis dari hasil pengujian perangkat lunak sehingga dapat diperoleh kesimpulan dari pembuatan perangkat lunak yang telah dilakukan.

3.7 Kesimpulan dan Saran

Pengambilan kesimpulan dilakukan setelah dilakukan proses pengujian dan analisis sistem sehingga dapat diketahui efektifitas kinerja dari sistem. Kesimpulan dibuat berdasarkan hasil pengujian dan analisis sehingga diperoleh inti dari keseluruhan proses penelitian. Pengambilan kesimpulan bertujuan untuk menjawab permasalahan yang ada dalam rumusan masalah. Saran digunakan untuk memberi masukan untuk menjadi bahan pertimbangan untuk pengembangan sistem selanjutnya. Tahap terakhir yaitu penulisan laporan yang dapat membantu dalam pengembangan sistem selanjutnya.

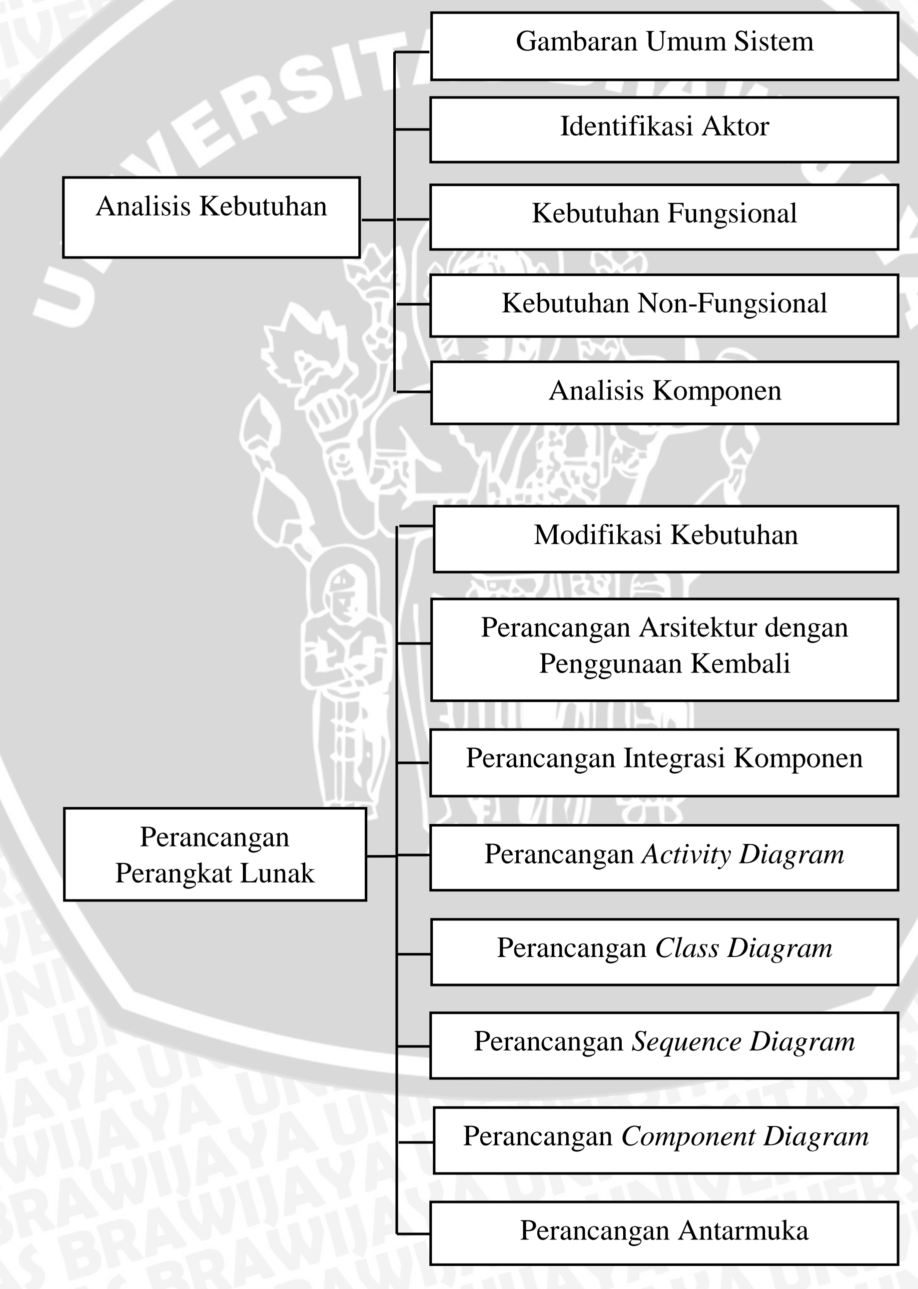
3.8 Penulisan Laporan

Laporan penelitian ditulis setelah semua proses pengerjaan tugas telah dilaksanakan. Laporan berisi dokumentasi perancangan sistem yang akan berguna untuk pengembangan sistem selanjutnya.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB IV
ANALISIS DAN PERANCANGAN

Bab ini menguraikan tentang perancangan sistem yang digunakan dalam penelitian pembangunan kakas bantu pembangkitan *Program Dependence Graph* berbasis Java. Perancangan ini terdiri dari analisis kebutuhan, dan perancangan perangkat lunak. Gambar 4.1. menunjukkan diagram perancangan yang dilakukan pada penelitian ini.



Gambar 4.1 Diagram Alir Analisis dan Perancangan

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

4.1 Analisis Kebutuhan

Analisis kebutuhan merupakan proses untuk menggambarkan kebutuhan-kebutuhan yang diperlukan oleh sistem agar dapat memenuhi kebutuhan pengguna. Analisis kebutuhan terdiri dari deskripsi sistem, identifikasi aktor, penjabaran kebutuhan fungsional, dan kebutuhan non-fungsional.

4.1.1 Gambaran Umum Sistem

Gambaran umum sistem pembangkitan *Program Dependence Graph* terdiri dari dua bagian, yaitu deskripsi umum sistem dan lingkungan sistem.

1. Deksripsi Umum Sistem

Sistem yang dibangun dalam penelitian ini adalah sebuah perangkat lunak dengan judul "Pembangunan Kakas Bantu Pembangkitan *Program Dependence Graph* berbasis Java". Tujuan utama dari sistem ini adalah untuk mengetahui keterkaitan antar variabel serta mengetahui masukan, keluaran, dan efek pada sebuah *source code* dan digambarkan dengan menggunakan *Program Dependence Graph*. Pengguna diharapkan dapat memperoleh informasi masukan, keluaran, dan efek serta mengetahui keterkaitan antar variabel dalam *source code* tersebut. Pada tahap awal sistem mendapatkan masukan berupa *source code* dari pengguna, kemudian sistem akan melakukan proses pembangkitan *Program Dependence Graph* dan menghasilkan keluaran berupa grafik *Program Dependence Graph*.

2. Lingkungan Sistem

Sistem pembangkitan *Program Dependence Graph* ini dibangun menggunakan bahasa Java, sehingga membutuhkan *Java Runtime Environment* (JRE) untuk berjalan. Sistem ini dapat berjalan di seluruh sistem operasi yang sudah terpasang JRE minimal versi 1.6.

4.1.2 Identifikasi Aktor

Identifikasi aktor merupakan tahap untuk melakukan identifikasi terhadap aktor-aktor yang dapat melakukan interaksi kepada sistem. Tahap identifikasi aktor dilakukan dengan mendefinisikan aktor-aktor pengguna sistem beserta

UNIVERSITAS BRAWIJAYA

deskripsinya. Tabel 4.1 menunjukkan aktor yang terdapat dalam sistem ini beserta deskripsinya.

Tabel 4.1 Identifikasi Aktor

Aktor	Deskripsi Aktor
Pengguna	Pengguna merupakan satu-satunya aktor yang menggunakan sistem, dan dapat menjalankan semua fungsi yang ada pada sistem.

4.1.3 Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional merupakan proses analisis kebutuhan untuk mengetahui fungsi dan fitur yang dibutuhkan oleh sistem. Kebutuhan fungsional dari sistem dibuat untuk pengguna sistem. Tabel 4.2 menunjukkan spesifikasi daftar kebutuhan fungsional dengan menggunakan format nomor *Software Requirement Specification* (SRS).

Tabel 4.2 Daftar Kebutuhan Fungsional

Nomor SRS	Kebutuhan	Aktor	*Use Case*
SRS_001	Sistem harus menyediakan fasilitas bagi pengguna untuk memasukkan *source code*. Sistem harus dapat menampilkan *pop-up* pencarian berkas dan dapat menampilkan isi *source code*.	Pengguna	Cari Berkas
SRS_002	Sistem harus menyediakan sebuah tampilan yang memuat sebuah grafik *Program Dependence Graph*.	Pengguna	Bangkitkan *Program Dependence Graph*
SRS_003	Sistem harus menyediakan informasi setiap *node* yang dipilih oleh pengguna.	Pengguna	Lihat Informasi *Node*

SRS_004	Sistem harus menyediakan fasilitas untuk mengganti *layout Program Dependence Graph.*	Pengguna	Ubah *Layout* Grafik
SRS_005	Sistem harus menyediakan fasilitas untuk memperbesar atau memperkecil grafik	Pengguna	Ubah Ukuran Grafik
SRS_006	Sistem harus menyediakan fasilitas untuk mengubah *Program Dependence Graph*.	Pengguna	Ubah *Program Dependence Graph*
SRS_007	Sistem harus menyediakan fasilitas untuk menampilkan tabel ketergantungan.	Pengguna	Lihat Tabel Ketergantungan

Selanjutnya daftar kebutuhan fungsional akan lebih dijabarkan dalam perancangan *use case diagram*.

4.1.3.1 Perancangan *Use Case Diagram*

Use case diagram digunakan untuk mengetahui fungsi-fungsi yang dapat dilakukan oleh sistem, serta dapat melihat aktor yang berhak menggunakan fungsi-fungsi tersebut. Aktor memiliki tujuh fitur (*use case*) yang dapat dilakukan pengguna kepada sistem. Berikut ini adalah penjelasan dari *use case* yang dapat dilakukan oleh aktor pengguna seperti pada gambar 4.2.

1. Cari Berkas

   Pengguna dapat mencari berkas *source code* ke dalam sistem dengan ekstensi .java sebagai data masukan. Kemudian file *source code* ini akan diolah menjadi diagram *Program Dependence Graph* oleh sistem.

2. Bangkitkan *Program Dependence Graph*

   Pengguna dapat melihat hasil diagram *Program Dependence Graph* dari *source code* yang telah di masukkan. Dari diagram *Program Dependence Graph*, pengguna dapat melihat ketergantungan antar variabel.

3. Lihat informasi *node*

Pengguna dapat melihat informasi yang dimiliki *node* seperti mengambil nama *method*, parameter *method*, tipe *method*, *return value*, nama variabel, tipe variabel, dan nilai variabel.

4. Ubah *Layout* Grafik

Pengguna dapat mengubah *layout* grafik *Program Dependence Graph* sesuai dengan *layout* yang dipilih seperti *Kanada Kawai Layout*, *DAG Layout*, dan *Fruchterman-Reingold Layout*.

5. Ubah Ukuran Grafik

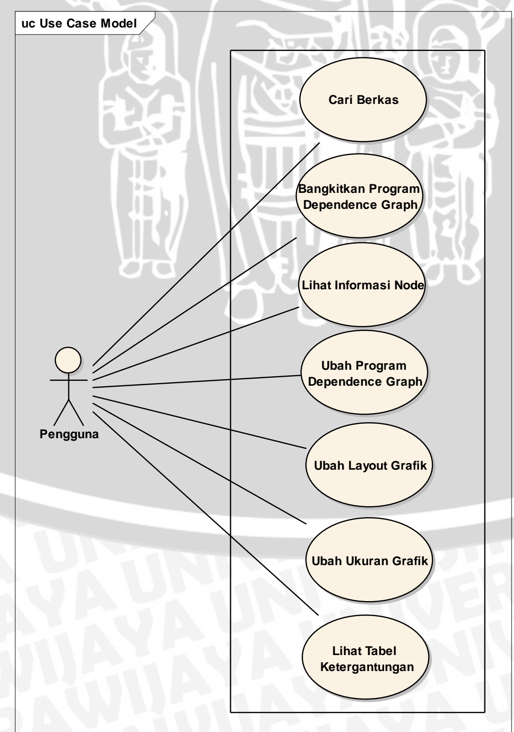
Pengguna dapat mengubah ukuran grafik menjadi lebih besar (*zoom in*) atau lebih kecil (*zoom out*).

6. Ubah *Program Dependence Graph*

Pengguna dapat mengubah *Program Dependence Graph* berdasarkan *method* yang ada pada berkas masukan setelah *Program Dependence Graph* terbuat. *Method* yang dipilih oleh pengguna akan di proses oleh sistem untuk menghasilkan grafik *Program Dependence Graph* dan tabel ketergantungan.

7. Lihat Tabel Ketergantungan

Pengguna dapat melihat tabel ketergantungan antar variabel dari *method* yang dipilih oleh pengguna.



Gambar 4.2 *Use Case Diagram*

4.1.3.2 Skenario *Use Case*

Skenario *use case* digunakan untuk menjelaskan secara detail setiap kebutuhan fungsioanal yang terdapat pada *use case diagram*.

4.1.3.2.1 Skenario *Use Case* Cari Berkas

Skenario *use case* cari berkas merupakan penjelasan lebih rinci dari *use case* cari berkas. Tabel 4.3 menujukkan skenario *use case* untuk mencari berkas.

Tabel 4.3 Skenario *Use Case* Cari Berkas

Nomor *Use Case*	SRS_001
Nama	Cari Berkas
Tujuan	Untuk mendapatkan informasi berkas yang dicari.
Deskripsi	*Use case* ini digunakan untuk melakukan proses pencarian dan pemasukan alamat *source code* yang bertipe berkas java (*.java) serta menampilkan isi berkas.
Aktor	Pengguna
Kondisi prasyarat	Pengguna telah membuka sistem.
Alur Utama	
Aksi Aktor	**Relasi Sistem**
1. Pengguna menekan tombol “Cari Berkas”.	2. Sistem menampilkan *pop-up* "Pencarian Berkas".
3. Pengguna mencari dan memilih berkas.	4. Sistem menyimpan informasi berkas. 5. Sistem akan menampilkan isi berkas.
Alur alternatif 1: Jika pengguna memasukkan alamat direktori yang salah atau berkas yang bertipe bukan berkas java.	
	6. Sistem akan menampilkan pesan kesalahan.
Kondisi akhir sukses	Alamat berkas bertipe Java akan disimpan didalam sistem dan sistem akan menampilkan

	kode yang dimuat dalam berkas yang dimasukkan.

4.1.3.2.2 Skenario *Use Case* Bangkitkan *Program Dependence Graph*

Skenario *use case* bangkitkan *Program Dependence Graph* merupakan penjelasan lebih rinci dari *use case* bangkitkan *Program Dependence Graph*. Tabel 4.4 menujukkan skenario *use case* untuk bangkitkan *Program Dependence Graph*.

Tabel 4.4 Skenario *Use Case* Bangkitkan *Program Dependence Graph*

Nomor *Use Case*	SRS_002
Nama	Bangkitkan *Program Dependence Graph*
Tujuan	Pengguna dapat membangkitkan *Program Dependence Graph* hasil representasi dari berkas kode yang dimasukkan.
Deskripsi	*Use case* ini digunakan untuk melakukan proses konversi kode berupa ekstraksi data dan normalisasi data yang telah dimasukkan oleh pengguna menjadi sebuah *Program Dependence Graph*.
Aktor	Pengguna
Kondisi prasyarat	Pengguna telah memasukkan berkas.
Alur Utama	
Aksi Aktor	**Relasi Sistem**
1. Pengguna menekan tombol "Bangkitkan *Program Dependence Graph*".	2. Sistem melakukan proses ekstraksi data kemudian melakukan normalisasi data. 3. Sistem akan membangun *Program Dependence Graph* berdasarkan data yang telah diproses. 4. Sistem akan menampilkan *Program Dependence Graph*.

5. Pengguna melihat *Program Dependence Graph*.	
Kondisi akhir sukses	Pengguna dapat melihat *Program Dependence Graph*.

4.1.3.2.3 Skenario *Use Case* Lihat Informasi *Node*

Skenario *use case* lihat informasi *node* merupakan penjelasan lebih rinci dari *use case* lihat informasi *node*. Tabel 4.5 menujukkan skenario *use case* untuk lihat informasi *node*.

Tabel 4.5 Skenario *Use Case* Lihat Informasi *Node*

Nomor *Use Case*	SRS_003
Nama	Lihat Informasi *Node*
Tujuan	Pengguna dapat melihat informasi *node* pada *Program Dependence Graph* hasil representasi dari berkas kode yang dimasukkan.
Deskripsi	*Use case* ini digunakan untuk menampilkan informasi *node* yang dipilih oleh pengguna.
Aktor	Pengguna
Kondisi prasyarat	Pengguna telah membangkitkan *Program Dependence Graph*.
Alur Utama	
Aksi Aktor	**Relasi Sistem**
1. Pengguna meletakkan pointer ke salah satu *node* yang ada pada *Program Dependence Graph*. 3. Pengguna dapat melihat informasi mengenai *node* yang telah dipilih	2. Sistem akan memperlihatkan informasi mengenai *node* yang ditunjukkan oleh pointer.

repository.ub.ac.id UNIVERSITAS BRAWIJAYA

Kondisi akhir sukses	Pengguna dapat melihat informasi *node* pada *Program Dependence Graph*.

4.1.3.2.4 Skenario *Use Case* Ubah *Layout* Grafik

Skenario *use case* ubah *layout* grafik merupakan penjelasan lebih rinci dari *use case* ubah *layout* grafik. Tabel 4.6 menujukkan skenario *use case* untuk ubah *layout* grafik.

Tabel 4.6 Skenario *Use Case* Ubah *Layout* Grafik

Nomor *Use Case*	SRS_004
Nama	Ubah *Layout* Grafik
Tujuan	Pengguna dapat mengubah *layout* atau tata letak dari *Program Dependence Graph*.
Deskripsi	*Use case* ini digunakan untuk memfasilitasi pengguna dalam mengubah *layout* dari *Program Dependence Graph*.
Aktor	Pengguna
Kondisi prasyarat	*Program Dependence Graph* telah terbuat.
Alur Utama	
Aksi Aktor	**Relasi Sistem**
1. Pengguna memilih tipe *layout* grafik.	2. Sistem akan mengubah *layout* dari *Program Dependence Graph* sesuai tipe yang dipilih oleh pengguna. 3. Sistem akan menampilkan ulang grafik *Program Dependence Graph*.
4. Pengguna dapat melihat *layout* grafik yang telah terubah.	
Kondisi akhir sukses	*Program Dependence Graph* akan berubah sesuai tipe layout yang dipilih oleh pengguna.

4.1.3.2.5 Skenario *Use Case* Ubah Ukuran Grafik

Skenario *use case* ubah ukuran grafik merupakan penjelasan lebih rinci dari *use case* ubah ukuran grafik. Tabel 4.7 menujukkan skenario *use case* untuk ubah ukuran grafik.

Tabel 4.7 Skenario *Use Case* Ubah Ukuran Grafik

Nomor *Use Case*	SRS_005
Nama	Ubah Ukuran Grafik
Tujuan	Pengguna dapat memperbesar dan memperkecil grafik *Program Dependence Graph.*
Deskripsi	*Use case* ini digunakan untuk memfasilitasi pengguna dalam mengubah ukuran grafik.
Aktor	Pengguna
Kondisi prasyarat	*Program Dependence Graph* telah terbuat.
Alur Utama	
Aksi Aktor	**Relasi Sistem**
1. Pengguna menekan tombol '+' atau '-'	2. Sistem akan mengubah ukuran grafik sesuai masukan pengguna. 3. Sistem akan menampilkan grafik.
4. Pengguna dapat melihat ukuran grafik yang telah terubah.	
Kondisi akhir sukses	Grafik *Program Dependence Graph* telah berubah ukuran sesuai dengan keinginan pengguna.

4.1.3.2.6 Skenario *Use Case* Ubah *Program Dependence Graph*

Skenario *use case* ubah ukuran grafik merupakan penjelasan lebih rinci dari *use case* ubah *Program Dependence Graph.* Tabel 4.8 menujukkan skenario *use case* untuk ubah *Program Dependence Graph.*

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Tabel 4.8 Skenario *Use Case* Ubah *Program Dependence Graph*

Nomor *Use Case*	SRS_006
Nama	Ubah *Program Dependence Graph*
Tujuan	Pengguna dapat memilih *method* yang ada pada berkas masukan pengguna.
Deskripsi	*Use case* ini dapat dijalankan apabila pengguna telah membangkitkan *Program Dependence Graph. Use case* ini digunakan untuk memfasilitasi pengguna dalam memilih *method.*
Aktor	Pengguna
Kondisi prasyarat	*Program Dependence Graph* telah terbuat.
Alur Utama	
Aksi Aktor	**Relasi Sistem**
1. Pengguna memilih daftar *method.*	2. Sistem mengubah *Program Dependence Graph* dan tabel ketergantungan sesuai dengan *method* yang dipilih.
3. Pengguna dapat melihat *Dependence Graph* dan tabel ketergantungan telah berubah.	
Kondisi akhir sukses	Grafik *Program Dependence Graph* dan tabel ketergantungan telah berubah sesuai dengan *method* yang dipilih oleh pengguna.

4.1.3.2.7 Skenario *Use Case* Lihat Tabel Ketergantungan

Skenario *use case* lihat tabel ketergantungan merupakan penjelasan lebih rinci dari *use case* lihat tabel ketergantungan. Tabel 4.9 menujukkan skenario *use case* untuk lihat tabel ketergantungan.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Tabel 4.9 Skenario *Use Case* Lihat Tabel Ketergantungan

Nomor *Use Case*	SRS_007
Nama	Lihat Tabel Ketergantungan
Tujuan	Pengguna dapat melihat tabel ketergantungan.
Deskripsi	*Use case* ini digunakan untuk memfasilitasi pengguna dalam melihat tabel ketergantungan.
Aktor	Pengguna
Kondisi prasyarat	*Program Dependence Graph* telah terbuat.
Alur Utama	
Aksi Aktor	**Relasi Sistem**
1. Pengguna memilih halaman tabel ketergantungan.	2. Sistem menampilkan halaman tabel ketergantungan.
3. Pengguna melihat tabel ketergantungan.	
Kondisi akhir sukses	Pengguna dapat melihat tabel ketergantungan.

4.1.4 Analisis Kebutuhan Non-Fungsional

Analisis kebutuhan non-fungsional merupakan proses analisis kebutuhan untuk mengetahui seberapa baik standar yang akan dihasilkan oleh sistem. Tabel 4.10 menunjukkan spesifikasi daftar kebutuhan non-fungsional sistem.

Tabel 4.10 Daftar Kebutuhan Non-Fungsional

Parameter	**Deskripsi Kebutuhan**
Akurasi	Pembangkitan grafik *Program Dependence Graph* harus memiliki tingkat akurasi lebih dari 95%.

4.1.5 Analisis Komponen

Analisis komponen dilakukan untuk mendapatkan komponen-komponen yang dibutuhkan bagi pengembangan perangkat lunak. Komponen yang dibutuhkan dalam membangun sistem pembangkitan *Program Dependence Graph* adalah

Abstract Syntax Tree yang mampu menangani bahasa pemrograman Java, dan komponen pembangkitan grafik yang mampu menangani masukan berupa bahasa pemrograman Java.

Abstract Syntax Tree yang digunakan dalam membangun sistem pembangkitan *Program Dependence Graph* adalah pustaka *Abstract Syntax Tree* yang dibuat oleh Eclipse. Komponen pembangkitan grafik yang dipakai adalah JUNG (*Java Universal Network/Graph Framework*). Komponen-komponen yang akan digunakan dalam kakas bantu ini dijelaskan pada tabel 4.11.

Tabel 4.11 Daftar Komponen

No	Nama Komponen	Fungsi Komponen
1	*Abstract Syntax Tree* (Eclipse)	*Abstract Syntax Tree* adalah kerangka kerja (*framework*) dasar yang telah diimplementasikan pada banyak kakas bantu dari perangkat lunak Eclipse IDE, termasuk refactoring, Quick Fix dan Quick Assist. *Abstract Syntax Tree* mengubah Java *source code* ke dalam bentuk skema pohon. Skema pohon ini dapat digunakan dan handal untuk menganalisis dan memodifikasi pemrograman dari sumber berbasis teks (*source code*).
2	JUNG	JUNG (*Java Universal Network/Graph Framework*) adalah sebuah bingkai kerja yang digunakan untuk pemodelan, analisis, dan visualisasi data yang dapat diwakili sebagai grafik atau jaringan. JUNG dibuat dengan menggunakan bahasa Java yang memungkinkan sistem berbasis JUNG memanfaatkan layanan dan fungsi jung secara penuh dengan menggunakan Java API.

4.2 Perancangan Perangkat Lunak

Perancangan perangkat lunak merupakan proses untuk membangun sistem berdasarkan analisis kebutuhan yang telah dilakukan. Tahapan dari proses ini terdiri dari perancangan arsitektur, perancangan integrasi komponen, perancangan *activity*

diagram, perancangan *class diagram*, perancangan *sequence diagram*, perancangan *component diagram*, dan perancangan antarmuka.

4.2.1 Modifikasi Kebutuhan

Modifikasi kebutuhan merupakan sebuah tahapan dimana analisis dilakukan kepada setiap komponen. Setiap komponen dianalisis berdasarkan kebutuhan yang akan ditangani oleh komponen tersebut. Modifikasi kebutuhan dilakukan apabila terdapat komponen yang tidak dapat mengakomodasi kebutuhan yang menjadi tanggung jawabnya. Analisis komponen dapat dilakukan lagi apabila kebutuhan tidak dapat dimodifikasi. Jika semua kebutuhan dapat diakomodasi oleh semua komponen maka modifikasi kebutuhan tidak perlu dilakukan. Pada pengembangan sistem ini, modifikasi kebutuhan tidak perlu dilakukan karena semua komponen dapat mengakomodasi seluruh kebutuhan.

4.2.2 Perancangan Arsitektur

Perancangan arsitektur menggunakan notasi-notasi untuk menggambarkan langkah-langkah penelitan yang digunakan dalam pembangunan kakas bantu pembangkitan *Program Dependence Graph* berbasis Java. Adapun penjelasan dari diagram alir pada gambar 4.3 adalah sebagai berikut:

1. Masukan

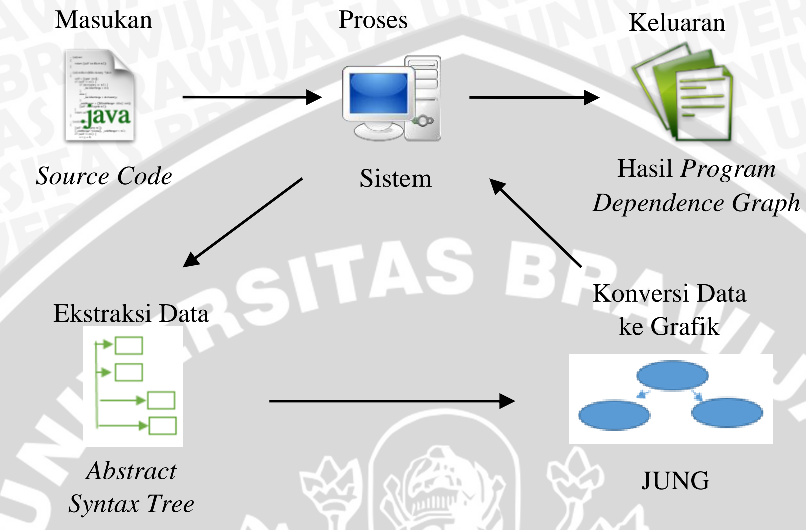
   Dalam penelitian ini, sistem dirancang dengan menggunakan masukan berupa *source code* bertipe Java yang diperoleh dari pengguna.
2. Proses

   Dalam sistem ini proses terbagi menjadi dua yaitu ekstraksi data dan konversi data ke grafik.
   a. Ekstraksi Data

      Proses ekstraksi data bertujuan untuk mendapatkan informasi dari masukan.
   b. Konversi Data ke Grafik

      Konversi data ke grafik bertujuan untuk mngubah data yang telah diambil ke dalam bentuk grafik.
3. Keluaran

Keluaran dari sistem ini adalah grafik *Program Dependence Graph* yang menggambarkan ketergantungan antar informasi yang diperoleh dari *source code*.



Gambar 4.3 Arsitektur Sistem Pembangkitan *Program Dependence Graph*

4.2.3 Perancangan Integrasi Komponen

Tahap pengembangan dan integrasi merupakan tahapan integrasi antara sistem dengan komponen-komponen yang sudah ditentukan. Pengembangan dan integrasi dalam sistem ini adalah sebagai berikut:

1. Pembuatan kakas bantu pembangkitan *Program Dependence Graph* sesuai dengan perancangan sistem yang telah dibuat.
2. Komponen yang digunakan adalah *Abstract Syntax Tree* , dan JUNG.
3. Komponen yang digunakan untuk mengekstraksi data adalah *Abstract Syntax Tree*.
4. Komponen yang digunakan untuk pembangkit grafik adalah JUNG.

Integrasi komponen dengan kebutuhan fungsional dapat dilihat pada tabel 4.12.

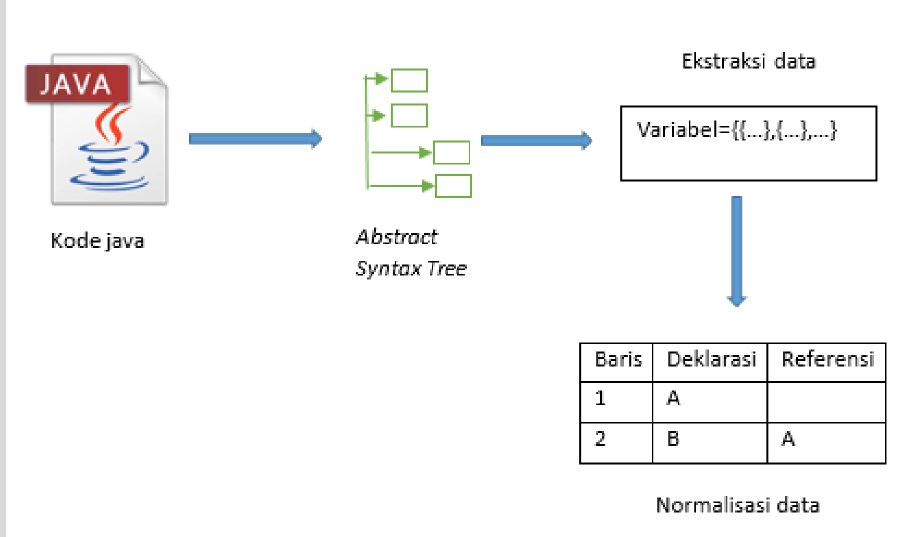
Tabel 4.12 Integrasi Komponen dengan Kebutuhan Fungsional

Kebutuhan	Komponen	Keterangan

Sistem harus menyediakan tampilan yang memuat grafik *Program Dependence Graph*.	JUNG	Mampu mengakomodasi seluruh kebutuhan
Sistem harus menyediakan informasi setiap *node* yang dipilih oleh pengguna.	*Abstract Syntax Tree*	Mampu mengakomodasi seluruh kebutuhan

4.2.3.1 Perancangan Ekstraksi Data

Tahapan ekstraksi data variabel dalam kode program dilakukan dengan menggunakan pustaka *Abstract Syntax Tree* dari Eclipse. Gambar 4.4 menunjukkan perancangan ekstraksi data dengan *Abstract Syntax Tree*.

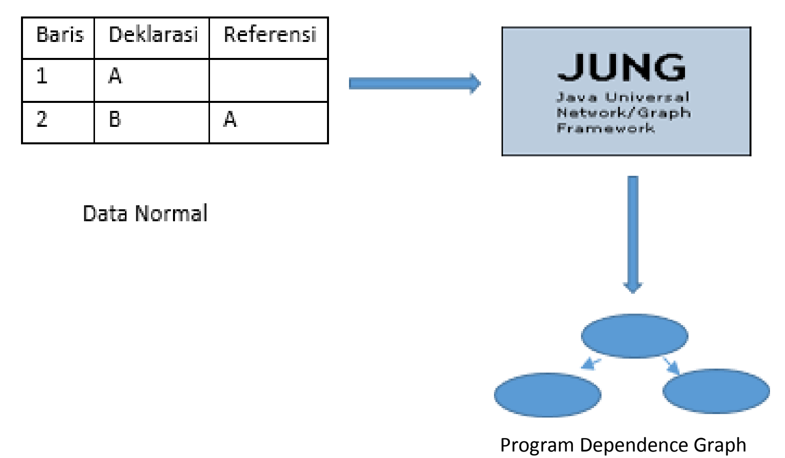


Gambar 4.4 Perancangan Ekstraksi Data Menggunakan *Abstract Syntax Tree*

Proses ekstraksi variabel pada sistem diambil dari kode program yang dimasukkan oleh pengguna. Kode program tersebut akan diubah kedalam skema pohon dengan menggunakan pustaka *Abstract Syntax Tree*. Skema pohon yang telah terbuat menampung seluruh informasi kode program mulai dari nama kelas, nama *method*, parameter *method*, nama variabel, jenis variabel, dan lain sebagainya. Pada fase ekstraksi data nformasi variabel yang didapatkan belum normal dan hubungan antar variabel belum tercipta. Hubungan antar variabel akan ditambahkan pada saat melakukan normalisasi data. Informasi variabel rujukan akan ditambahkan pula apabila nilai suatu variabel didapatkan dari nilai variabel lain.

4.2.3.2 Perancangan Konversi Data ke Bentuk Grafik

Tahapan konversi data ke bentuk grafik dilakukan dengan menggunakan pustaka JUNG. Gambar 4.6 menunjukkan perancangan konversi data ke bentuk grafik dengan JUNG.



Gambar 4.5 Perancangan Konversi Data ke Bentuk Grafik menggunakan JUNG

Tahapan pembangkitan *Program Dependence Graph* dilakukan setelah data normal didapatkan melalui proses ekstraksi data variabel pada kode program yang dimasukkan. Setiap variabel deklarasi akan diubah menjadi sebuah *node* di dalam pustaka JUNG. Variabel rujukan/referensi akan diubah menjadi informasi *node* tujuan. JUNG memiliki banyak algoritma tata letak grafik yang akan dibangun. Penulis menggunakan algoritma Kanada Kawai, tata letak DAG, dan Frutcherman Reingold untuk membangun *Program Dependence Graph.* Setelah data variabel diubah menjadi *node* dan algoritma telah terpilih maka grafik *program Dependence Graph* akan ditampilkan.

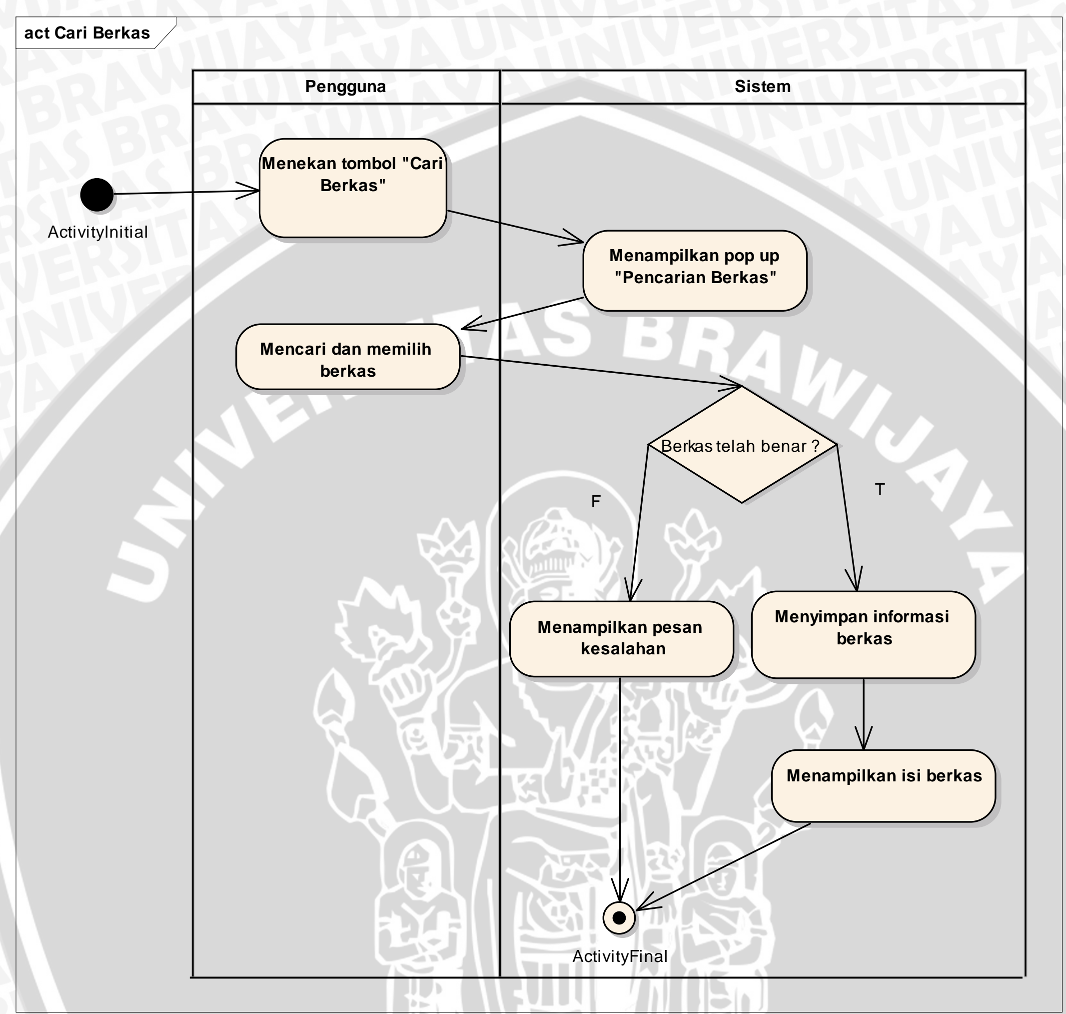
4.2.4 Perancangan *Activity Diagram*

Activity diagram digunakan untuk memodelkan langkah-langkah dalam aliran kerja (*workflow*) dan urutan aktivitas dalam sebuah proses. *Activity diagram* dibuat berdasarkan sebuah *use case* pada *use case diagram*.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

4.2.4.1 *Activity Diagram* Cari Berkas

Activity diagram cari berkas merupakan model aliran kerja dari *use case* cari berkas.



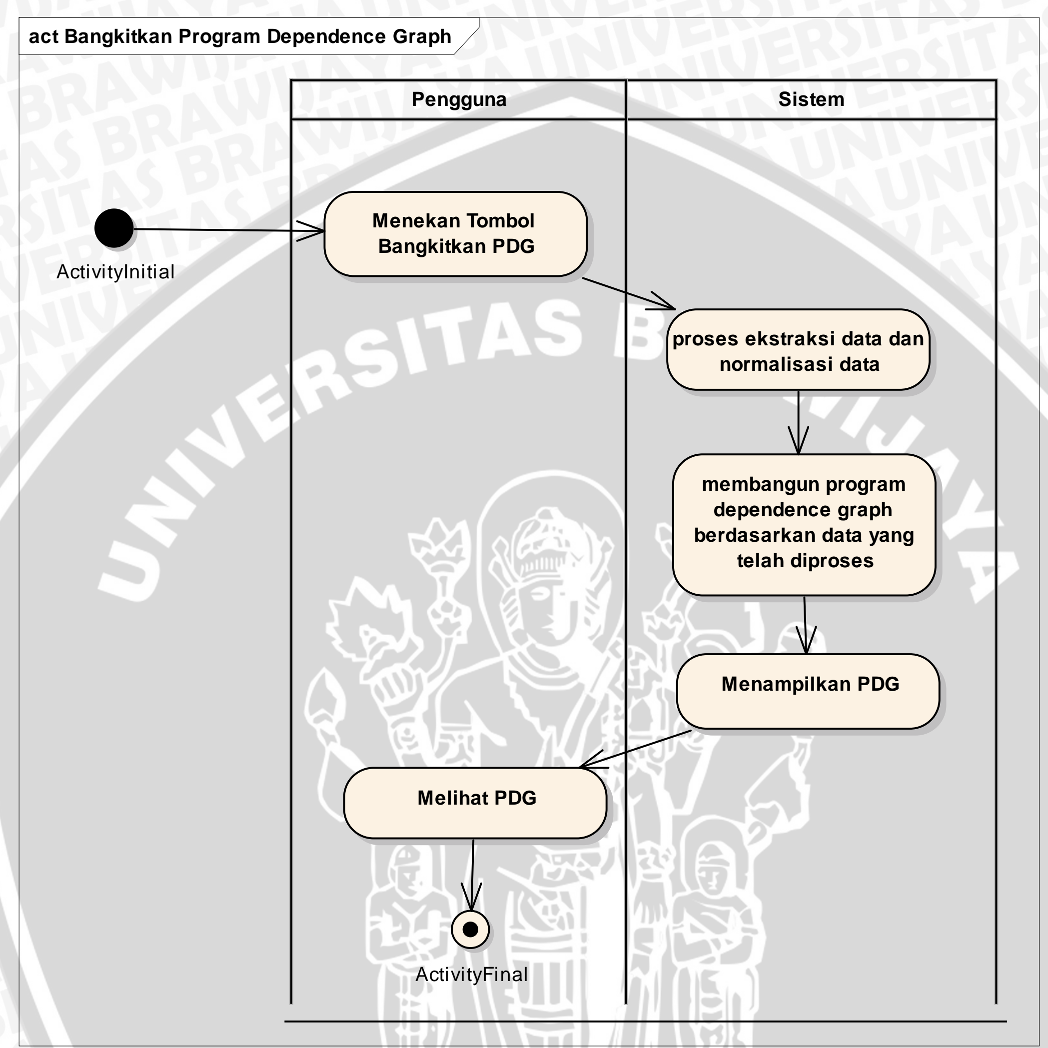
Gambar 4.6 *Activity Diagram* Cari Berkas

Gambar 4.6 menunjukkan urutan aktivitas mencari berkas *source code* oleh pengguna kepada sistem. Pertama pengguna menekan tombol "Cari Berkas" pada sistem, kemudian sistem akan menampilkan *pop-up* "Pencarian Berkas". Setelah itu pengguna mencari dan memilih berkas *source code* yang diinginkan untuk diubah ke dalam diagram *Program Dependence Graph*. Lalu sistem akan melakukan verifikasi terhadap berkas masukan apakah berkas sudah benar atau salah. Jika benar maka sistem akan menyimpan informasi berkas *source code* yang telah dimasukkan oleh pengguna dan menampilkan isi berkas tersebut. Jika salah maka sistem akan menampilkan pesan kesalahan.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

4.2.4.2 *Activity Diagram* Bangkitkan Program *Dependence Graph*

Activity diagram bangkitkan *Program Dependence Graph* merupakan model aliran kerja dari *use case* bangkitkan *Program Dependence Graph*.



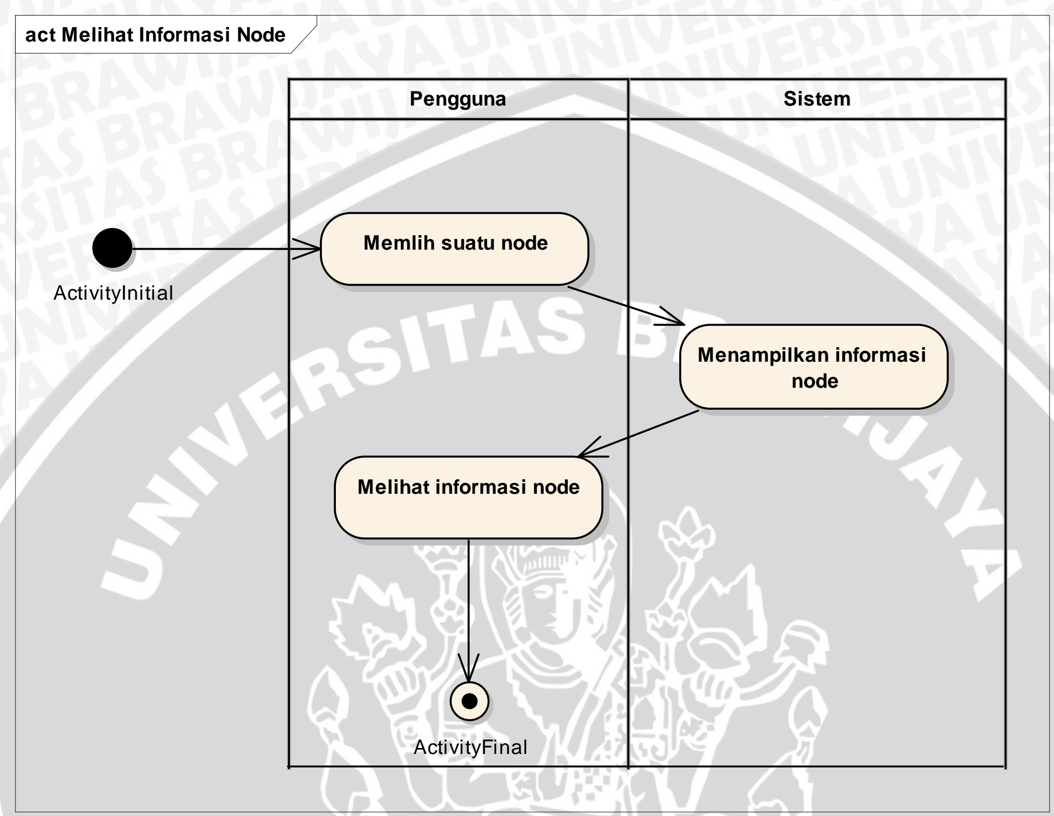
Gambar 4.7 *Activity Diagram* Bangkitkan *Program Dependence Graph*

Gambar 4.7 menunjukkan urutan aktivitas membangkitkan *Program Dependence Graph* oleh pengguna kepada sistem. Aktivitas ini dilakukan setelah aktivitas cari berkas telah selesai dilakukan. Pertama pengguna menekan tombol "Bangkitkan *Program Dependence Graph*" untuk melakukan proses perubahan *source code* menjadi grafik *Program Dependence Graph*. Kemudian sistem akan melakukan proses ekstraksi data dan normalisasi data dan membangun*Program Dependence Graph* berdasarkan data masukan yang telah diproses. Setelah itu sistem akan menampilkan hasil grafik *Program Dependence Graph* yang telah terbuat, lalu pengguna dapat melihat grafik *Program Dependence Graph*.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

4.2.4.3 *Activity Diagram* Lihat Informasi *Node*

Activity diagram lihat informasi *node* merupakan model aliran kerja dari *use case* lihat informasi *node*.



Gambar 4.8 *Activity Diagram* Lihat Informasi *Node*

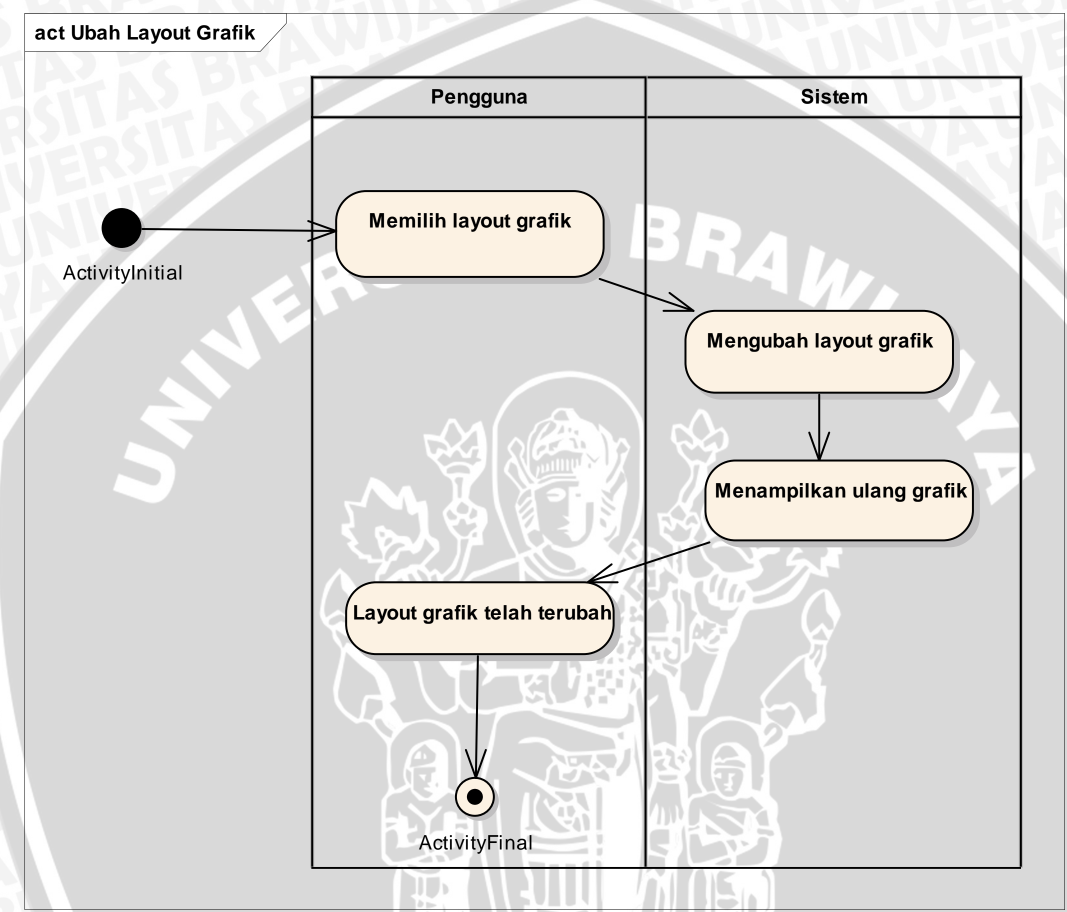
Gambar 4.8 menunjukkan urutan aktivitas lihat informasi *node* oleh pengguna kepada sistem. Aktivitas ini dilakukan setelah aktivitas bangkitkan *Program Dependence Graph* telah selesai dilakukan. Setelah sistem menampilkan grafik *Program Dependence Graph*, kemudian pengguna memilih suatu *node* yang ingin dilihat informasi *node* tersebut. Sistem akan memproses dan menampilkan informasi *node* yang dipilih oleh pengguna. Setelah itu pengguna melihat informasi yang dimiliki oleh *node* yang telah dipilih.

4.2.4.4 *Activity Diagram* Ubah *Layout* Grafik

Activity diagram ubah *layout* grafik merupakan model aliran kerja dari *use case* ubah layout grafik. Gambar 4.9 menunjukkan urutan aktivitas ubah *layout* grafik oleh pengguna kepada sistem. Aktivitas ini dilakukan setelah aktivitas lihat

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

infromasi *node* telah selesai dilakukan. Setelah sistem menampilkan informasi *node*, kemudian pengguna memilih *layout* grafik yang diinginkan. Sistem akan memproses dan mengubah *layout* grafik, lalu sistem akan menampilkan ulang grafik *Program Dependence Graph* sesuai pilihan pengguna . Setelah itu pengguna melihat layout grafik yang telah berubah.



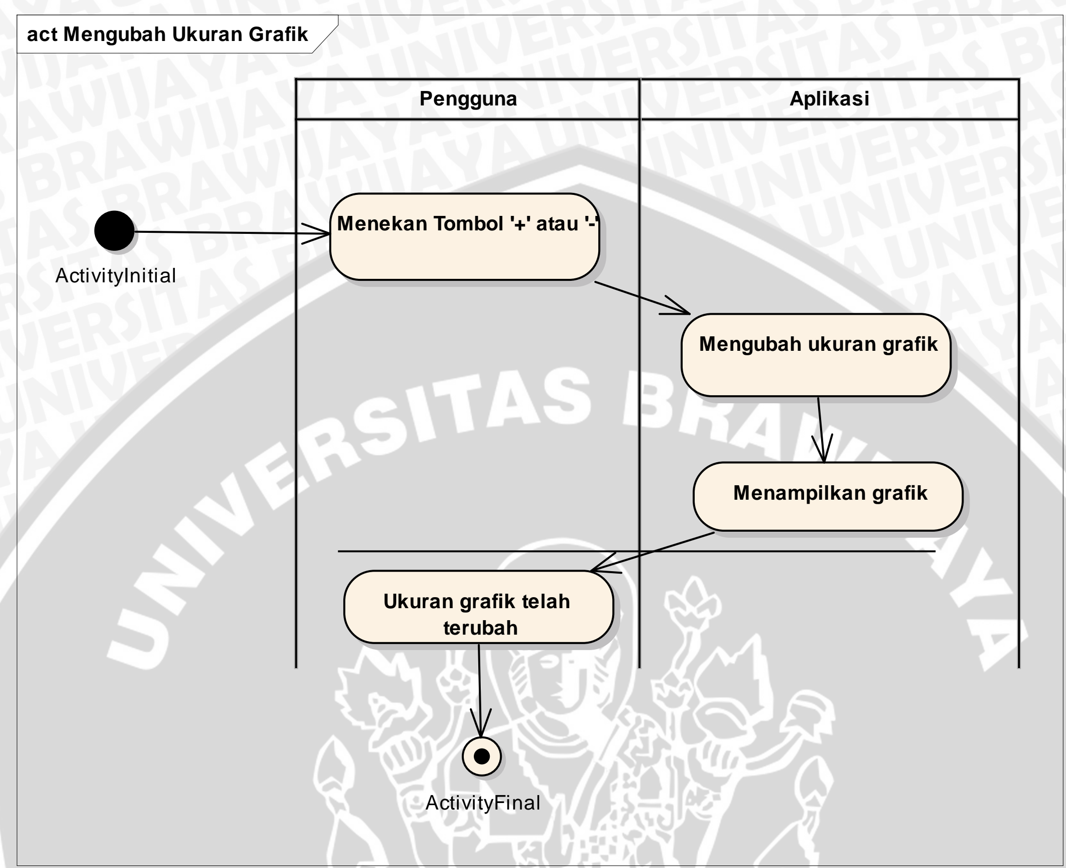
Gambar 4.9 *Activity Diagram* Ubah *Layout* Grafik

4.2.4.5 *Activity Diagram* Ubah Ukuran Grafik

Activity diagram ubah ukuran grafik merupakan model aliran kerja dari *use case* ubah ukuran grafik. Gambar 4.10 menunjukkan urutan aktivitas ubah ukuran grafik oleh pengguna kepada sistem. Aktivitas ini dilakukan setelah aktivitas bangkitkan *Prorgram Dependence Graph* telah selesai dilakukan. Pengguna dapat menekan tombol "+" atau "-", tombol "+" untuk memperbesar ukuran grafik sementara tombol "-" untuk memperkecil ukuran grafik. Kemudian sistem mengubah ukuran grafik dan menampilkan grafik *Program Dependence Graph*

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

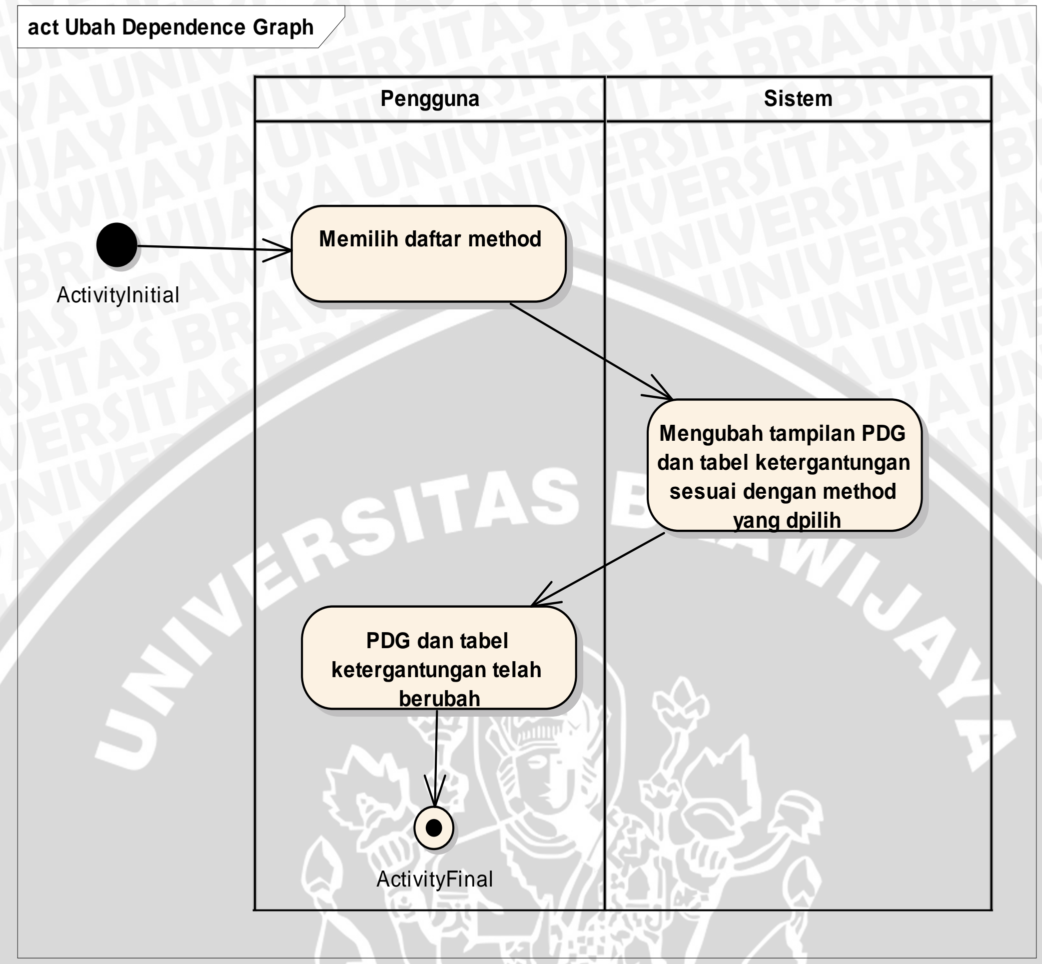
sesuai pilihan pengguna . Setelah itu pengguna melihat ukuran grafik yang telah berubah.



Gambar 4.10 *Activity Diagram* Ubah Ukuran Grafik

4.2.4.6 *Activity Diagram* Ubah *Program Dependence Graph*

Activity diagram ubah *Program Dependence Graph* merupakan model aliran kerja dari *use case* ubah *Program Dependence Graph*. Gambar 4.11 menunjukkan urutan aktivitas ubah *Program Dependence Graph* oleh pengguna kepada sistem. Aktivitas ini dilakukan setelah aktivitas bangkitkan *Prorgram Dependence Graph* telah selesai dilakukan. Proses ini diawali dengan pengguna memilih daftar *method*. Kemudian sistem mengubah tampilan *Program Dependence Graph* dan tabel ketergantungan sesuai dengan *method* pilihan pengguna . Setelah itu pengguna dapat melihat *Program Dependence Graph* dan tabel ketergantungan yang telah berubah.

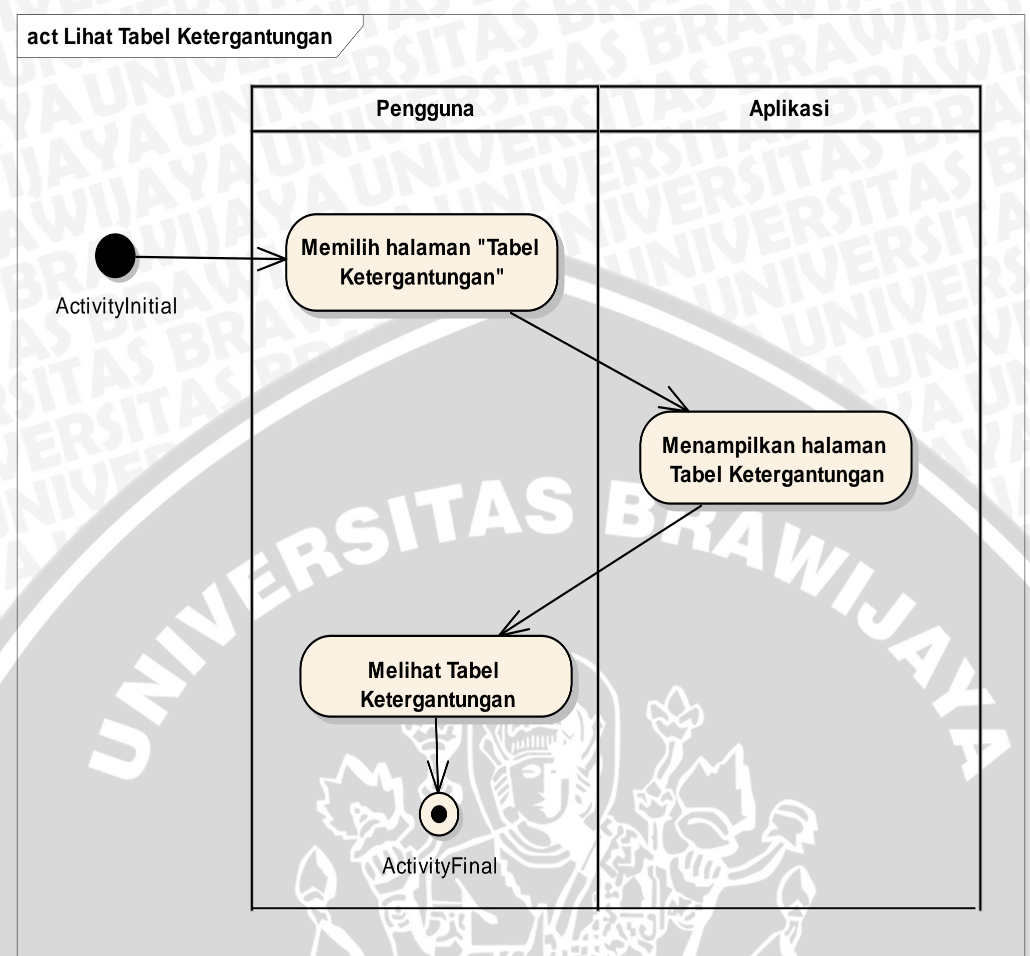


Gambar 4.11 *Activity Diagram* Ubah *Program Dependence Graph*

4.2.4.7 *Activity Diagram* Lihat Tabel Ketergantungan

Activity diagram lihat tabel ketergantungan merupakan model aliran kerja dari *use case* lihat tabel ketergantungan. Tabel ketergantungan berguna untuk menampilkan ketergantungan antar variabel yang telah didapatkan setelah proses ekstraksi data. Gambar 4.12 menunjukkan urutan aktivitas lihat tabel ketergantungan oleh pengguna kepada sistem. Aktivitas ini dilakukan setelah aktivitas bangkitkan *Prorgram Dependence Graph* telah selesai dilakukan. Proses ini diawali dengan pengguna memilih halaman tabel “Ketergantungan”, kemudian sistem akan menampilkan halaman tabel ketergantungan. Setelah itu pengguna dapat melihat isi tabel ketergantungan.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

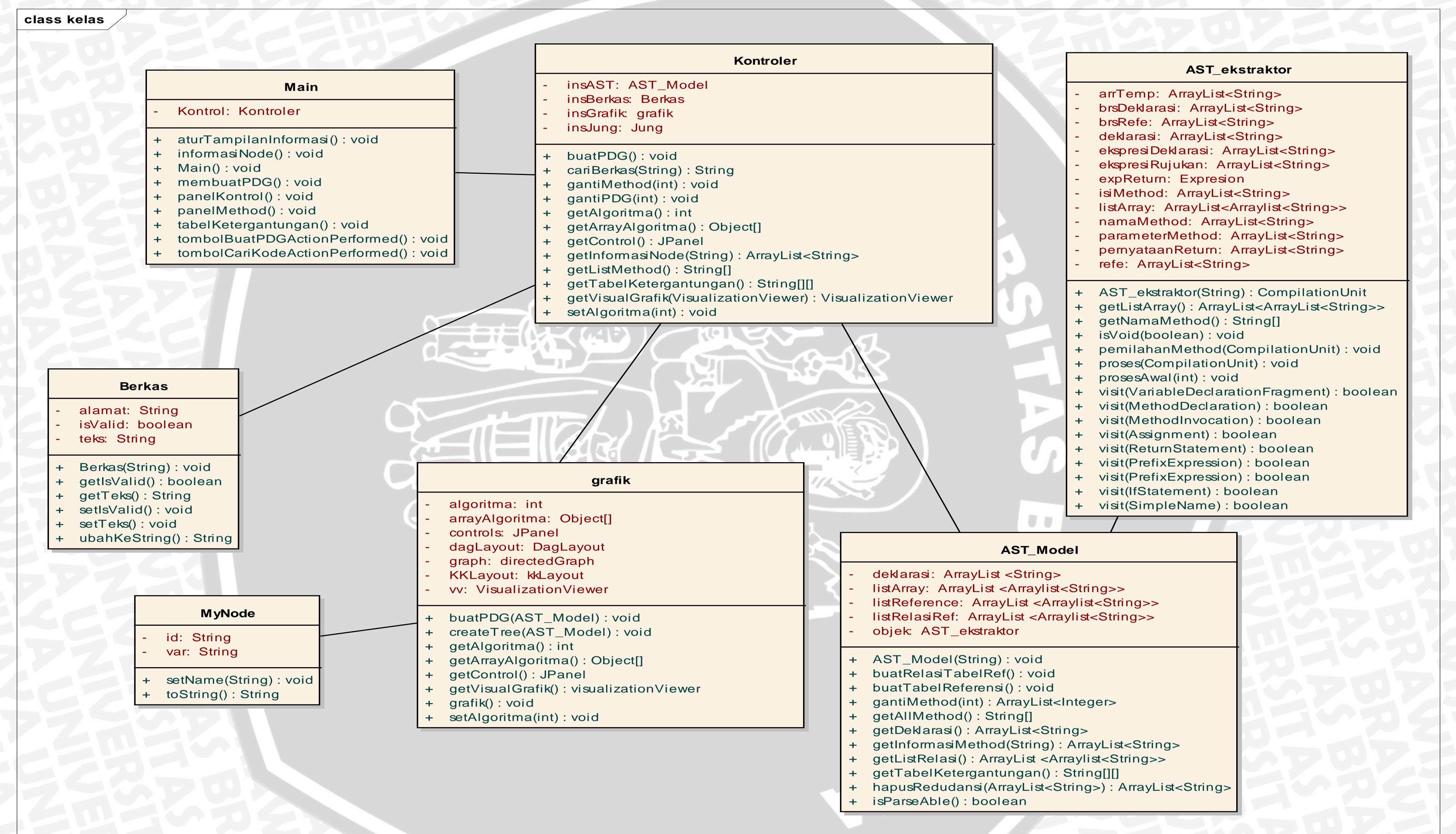


Gambar 4.12 *Activity Diagram* Lihat Tabel Ketergantungan

4.2.5 Perancangan *Class Diagram*

Class diagram merupakan diagram yang digunakan untuk menampilkan beberapa kelas dan relasi antar kelas yang berada dalam sistem. Pada sistem ini relasi antar kelas terjadi apabila instansiasi telah dilakukan. Setelah instansiasi dilakukan maka suatu kelas dapat mengirimkan pesan untuk menjalankan *method* yang ada pada kelas lain. Gambar 4.13 menunjukkan *class diagram* yang digunakan pada pembuatan pembangunan kakas bantu pembangkitan *Program Dependence Graph* berbasis Java.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA



Gambar 4.13 *Class Diagram*

Sistem ini dirancang dengan menggunakan tujuh kelas, diantaranya yaitu:

1. Main

   Kelas Main digunakan sebagai fungsi awal untuk dieksekusi dan juga mengatur tampilan antarmuka antara pengguna dan sistem.
2. Kontroler

   Kelas Kontroler digunakan sebagai kelas penghubung antar kelas Main dan kelas-kelas lainnya.
3. Berkas

   Kelas Berkas digunakan untuk memproses dan menyimpan informasi berkas.
4. Grafik

   Kelas grafik digunakan untuk membangkitkan grafik *Program Dependence Graph*.
5. AST_ekstraktor

   Kelas AST_ekstraktor digunakan untuk melakukan ekstraksi data.
6. AST_Model

   Kelas AST_Model digunakan untuk menyimpan dan melakukan normalisasi data.
7. MyNode

   Kelas MyNode digunakan sebagai kelas pembangun *node*.

Perancangan *class diagram* sistem Pembangkitan *Program Dependence Graph* terdiri dari diagram alir ekstraksi data, tabel ketergantungan, dan buat grafik.

4.2.5.1 Diagram Alir Ekstraksi Data

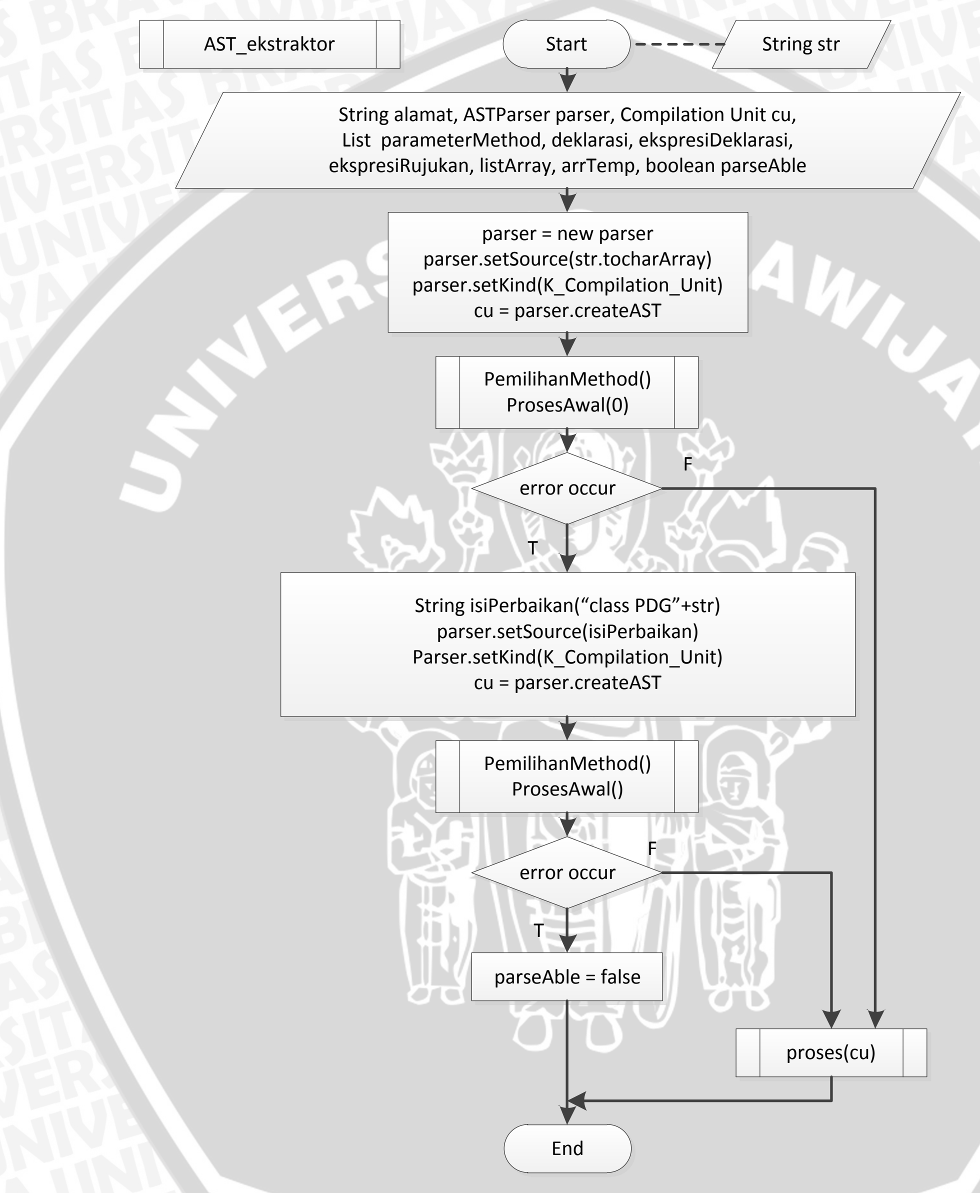
Langkah-langkah yang dilakukan untuk melakukan ekstraksi data adalah sebagai berikut:

1. Melakukan deklarasi variabel yang dibutuhkan dalam melakukan ekstraksi data.
2. Memberikan informasi alamat dan jenis berkas yang akan diubah kedalam skema pohon.
3. Melakukan eksekusi terhadap *method* PemilihanMethod() dan ProsesAwal() . PemilahanMethod() berfungsi untuk memilah seluruh *method* yang ada pada

source code, Sedangkan ProsesAwal() berfungsi untuk menginisialisasi variabel yang telah dideklarasikan.

4. Melakukan pengecekan apakah terdapat kesalahan dalam pembuatan skema pohon. Penambahan nama kelas dilakukan apabila terdapat kesalahan.

Gambar 4.14 menunjukkan diagram alir ekstraksi data.



Gambar 4.14 Diagram Alir Ekstraksi Data

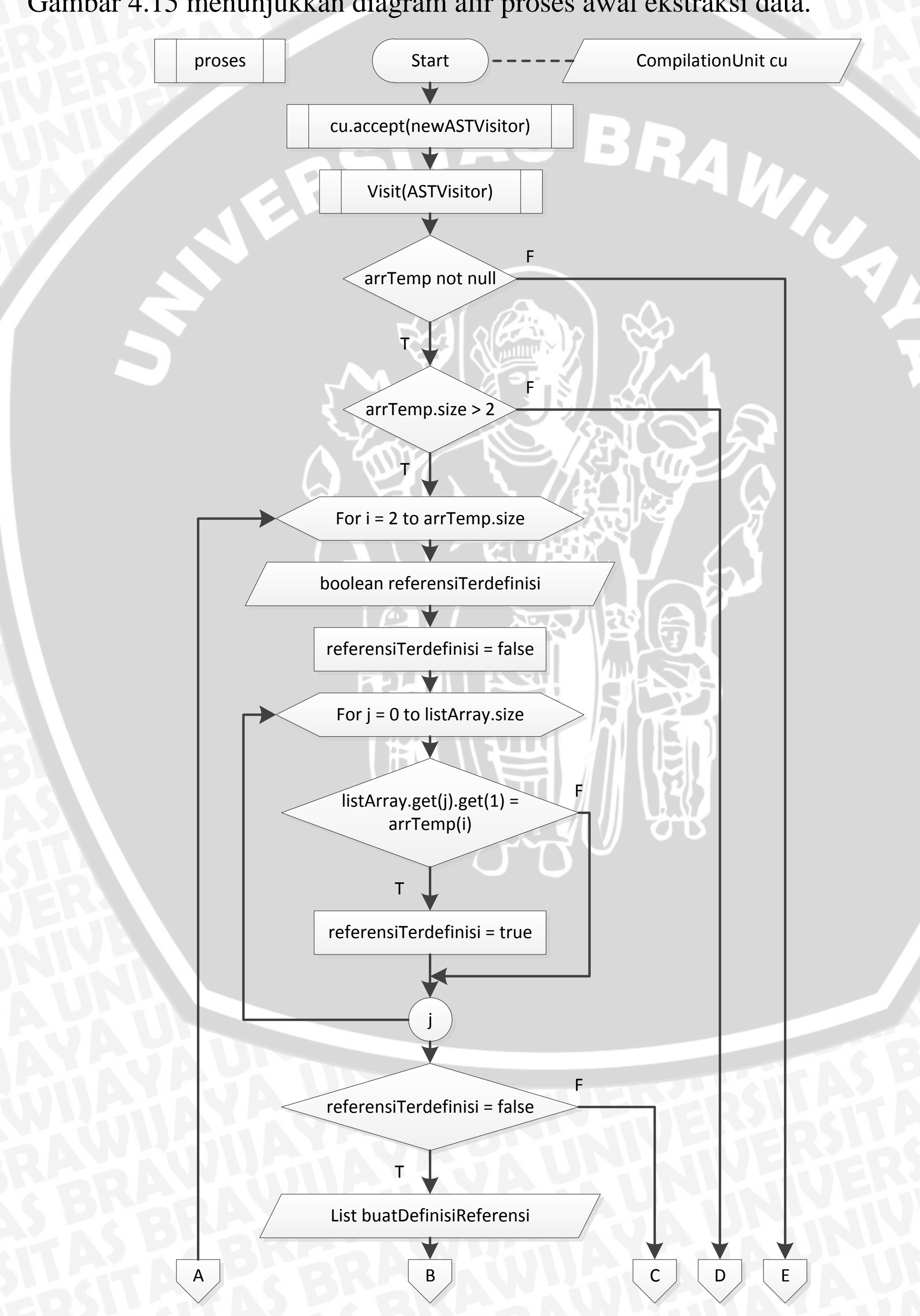
4.2.5.1.1 Diagram Alir Proses Awal Ekstraksi Data

Langkah-langkah yang dilakukan untuk melakukan proses awal ekstraksi data adalah sebagai berikut:

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

1. Melakukan pengecekan setiap node dalam skema pohon.
2. Melakukan proses ekstraksi data pada setiap aspek pada skema pohon. Aspek yang akan diproses adalah MethodDeclaration, MethodInvocation, VariableDeclarationFragment, Assignment, ReturnStatement, PostFixExpression, PreFixExpression, IfStatement, dan SimpleName.
3. Menampung semua variabel yang telah didapatkan.

Gambar 4.15 menunjukkan diagram alir proses awal ekstraksi data.



repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

A B C D E

buatDefinisiReferensi.add(i).add(arrTemp.get(i)
listArray.add(buatDefinisiReferensi

i

listArray.add(arrTemp)

End

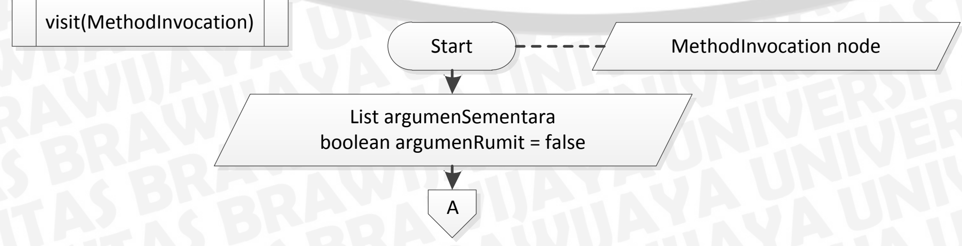
Gambar 4.15 Diagram Alir Proses Awal Ekstraksi Data

4.2.5.1.2 Diagram Alir visit(MethodInvocation)

Langkah-langkah yang dilakukan untuk melakukan proses visit(MethodInvocation) adalah sebagai berikut:

1. Melakukan deklarasi variabel argumenSementara yang digunakan untuk menampung variabel yang terdapat pada argumen pemanggilan *method*.
2. Melakukan pengecekan apakah terdapat argumen pada MethodInvocation. Apabila terdapat argumen maka argumen tersebut akan disimpan pada variabel argumenSementara.
3. Melakukan pengecekan terhadap argumen yang telah didapatkan, apabila argumen tersebut memiliki pernyataan MethodInvocation maka variabel yang terdapat pada argumen tersebut akan ditambahkan pada variabel argumenSementara.
4. Pernyataan MethodInvocation beserta variabel yang ada akan disimpan pada variabel sementara.

Gambar 4.16 menunjukkan diagram alir visit(MethodInvocation).



UNIVERSITAS BRAWIJAYA

A
argumen not null
F
T
For i=0 to size of argumen
x =(MethodInvocation) argumen
argumenSementara.add(x.getExpression)
For j=0 to size of argumen
argumenSementara.add(x.argumen.get(j))
j
argumenRumit = true
i
argumen not null &&
argumenRumit = false
F
T
For i=0 to size of argumen
For j=0 tolistArray.size
nodeArgumen(i) =
listArray(j)
T
argumenSementara.add(node.argumen.get(i))
j
i
List sementara
sementara.add(node)
argumen
sementara not
null
F
T
For i=0 to argumenSementara.size
sementara.add(argumenSementara(i))
i
End

58

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

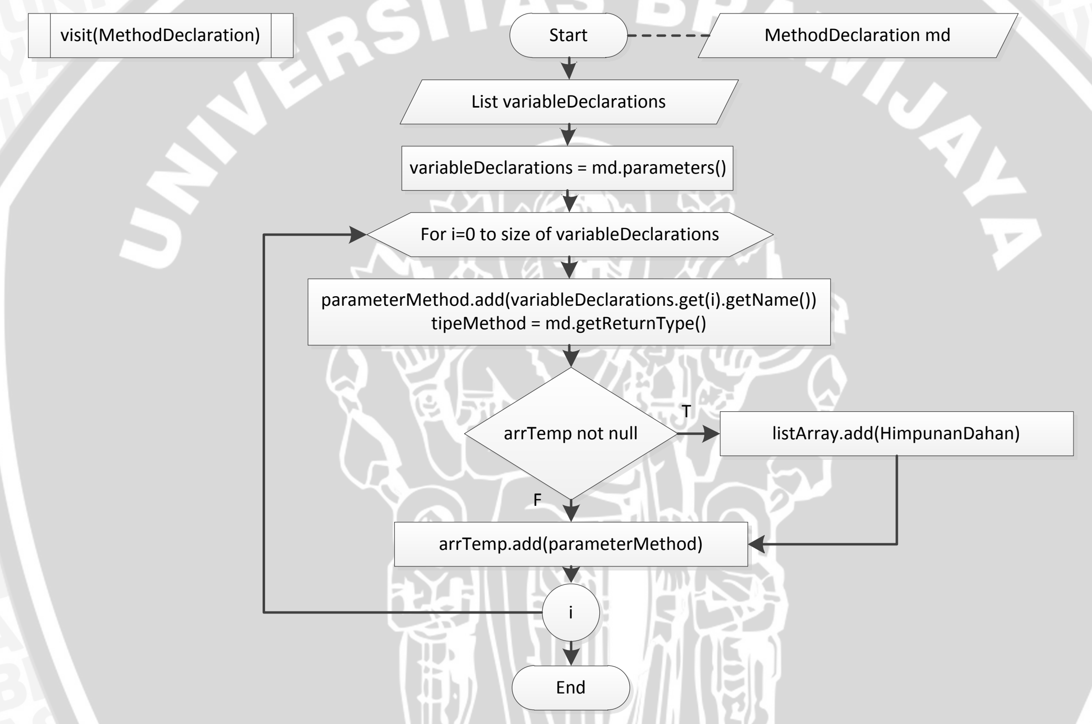
Gambar 4.16 Diagram Alir visit(MethodInvocation)

4.2.5.1.3 Diagram Alir visit(MethodDeclaration)

Langkah-langkah yang dilakukan untuk melakukan proses visit(MethodDeclaration) adalah sebagai berikut:

1. Melakukan deklarasi variabel bertipe List dengan nama variableDeclaration yang akan menyimpan variabel yang terdapat pada parameter *method.*
2. Menyimpan isi dari himpunan variableDeclaration ke dalam variabel listArray.

Gambar 4.17 menunjukkan diagram alir visit(MethodDeclaration).



Gambar 4.17 Diagram Alir visit(MethodDeclaration)

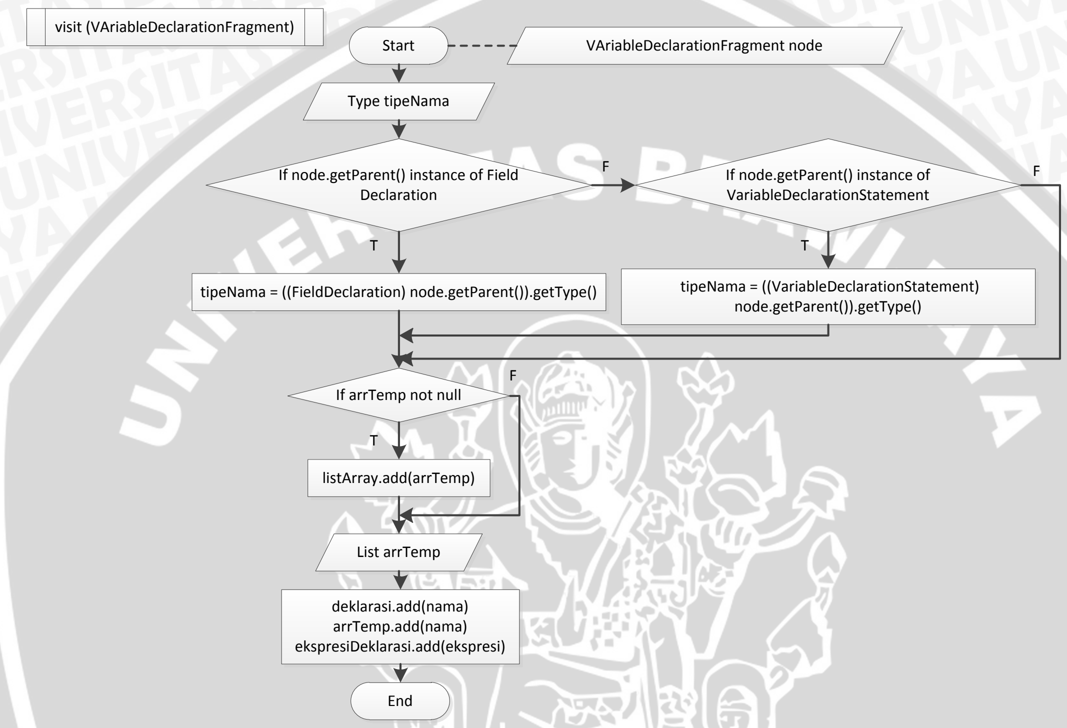
4.2.5.1.4 Diagram Alir visit(VariableDeclarationFragment)

Langkah-langkah yang dilakukan untuk melakukan proses visit(VariableDeclarationFragment) adalah sebagai berikut:

1. Deklarasi variabel tipeNama untuk menampung tipe dari variabel yang telah didapatkan.

2. Melakukan pengecekan terhadap variabel arrTemp, apabila arrTemp tidak kosong maka isi dari arrTemp akan ditambahkan ke dalam listArray. Apabila arrTemp masih kosong maka variabel akan ditambahkan ke dalam arrTemp.
3. Menyimpan nama deklarasi variabel yang telah didapatkan ke dalam deklarasi dan menyimpan ekspresi variabel kedalam ekspresiDeklarasi.

Gambar 4.18 menunjukkan diagram alir visit(VariableDeclarationFragment).



Gambar 4.18 Diagram Alir visit(variableDeclarationFragment)

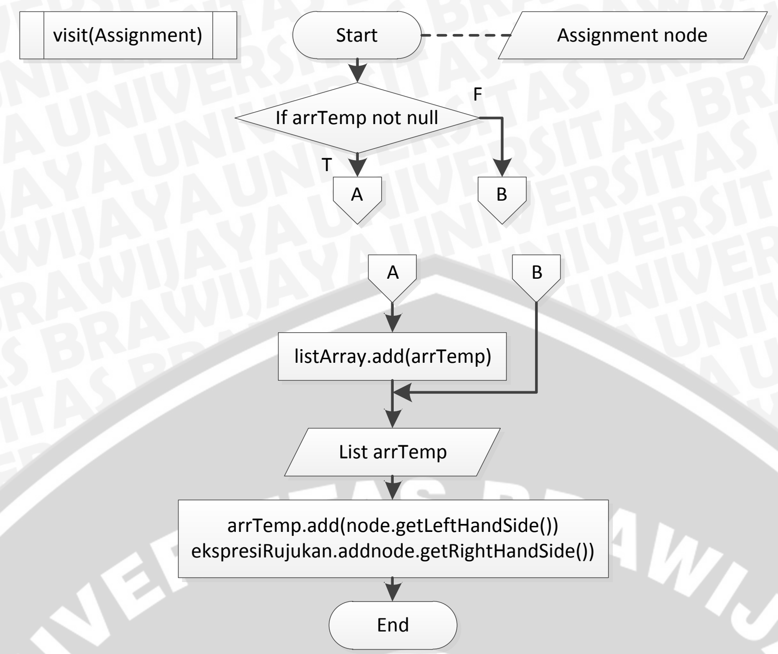
4.2.5.1.5 Diagram Alir visit(Assigment)

Langkah-langkah yang dilakukan untuk melakukan proses visit(Assigment) adalah sebagai berikut:

1. Melakukan pengecekan apakah arrTemp kosong atau tidak. Apabila arrTemp tidak kosong maka isi dari arrTemp akan ditambahkan ke dalam listArray.
2. Apabila arrTemp kosong maka variabel yang berada disisi kanan yang didapatkan akan disimpan ke dalam arrTemp, sedangkan variabel yang berada disisi kiri akan disimpan kedalam ekspresiRujukan.

Gambar 4.19 menunjukkan diagram alir visit(Assigment).

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA



Gambar 4.19 Diagram Alir visit(Assigment)

4.2.5.1.6 Diagram Alir visit(ReturnStatement)

Langkah-langkah yang dilakukan untuk melakukan proses visit(ReturnStatement) adalah sebagai berikut:

1. Menyimpan ekspresi pada ReturnStatement ke dalam variabel exprReturn.
2. Memilah variabel yang terdapat pada eksrpresi dari ReturnStatement dan menyimpanya apabila variabel telah terdeklarasi.
3. Melakukan pengecekan apakah argumen pada ReturnStatement sudah ada pada listArray, jika ada maka akan disimpan pada variabel arrArgs.
4. Apabila argumen pada ReturnStatement tidak ada pada listArray maka argumen tersebut akan ditampung pada variabel arrTemp.

Gambar 4.20 menunjukkan diagram alir visit(ReturnStatement).

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

visit(returnStatement)

Start

exprReturn = node.getExpression()

String stringExprReturn
StringTokenizer st

stringExprReturn = node.getExpression()

st.hasMoreToken()

T

F

MethodInvocation returnInvok
Int nLA
boolean adaDiListArray
List arrArgs

String token

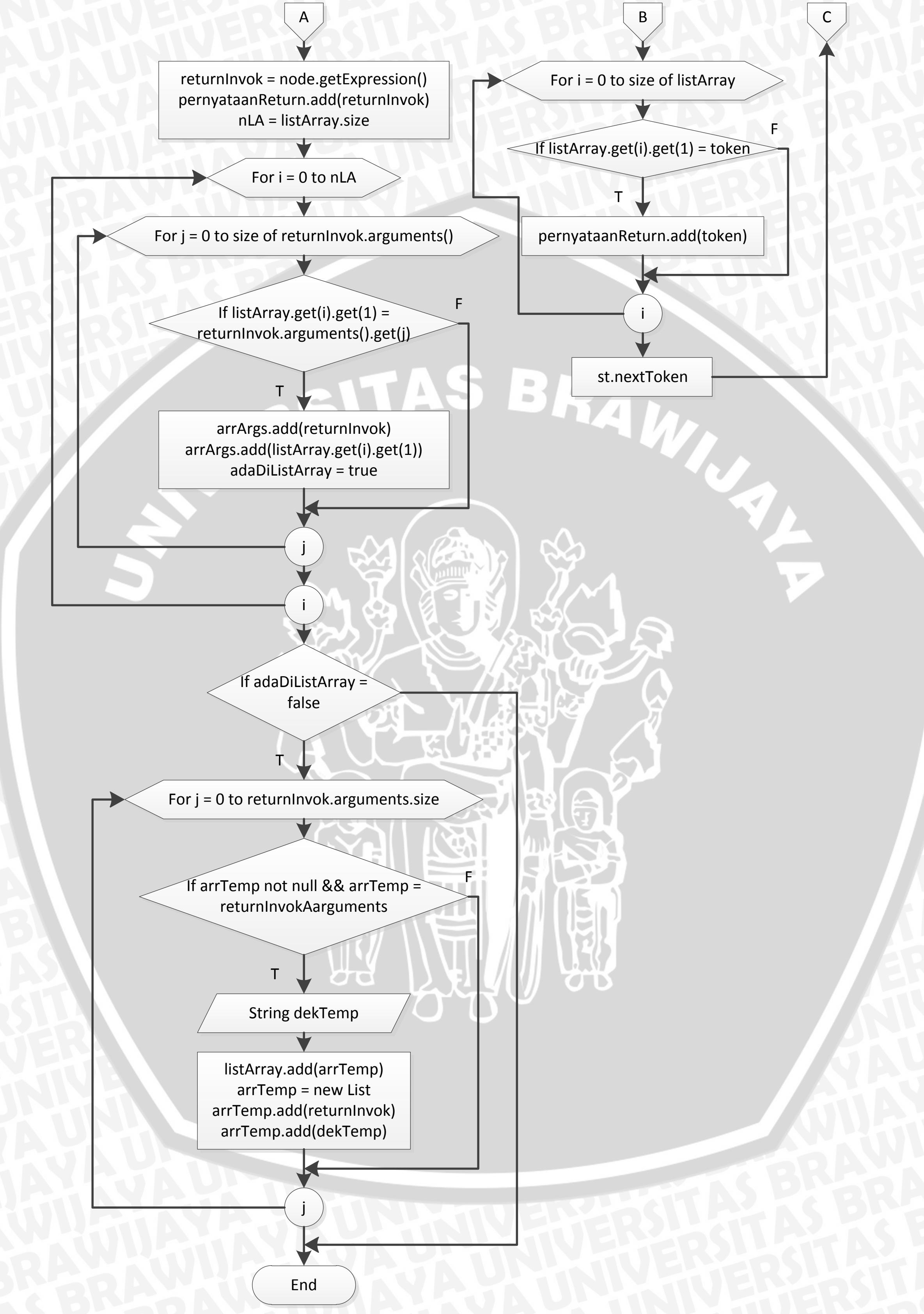
token = st.nextToken().trim()

A

B

C

UNIVERSITAS BRAWIJAYA



Gambar 4.20 Diagram Alir visit(ReturnStatement)

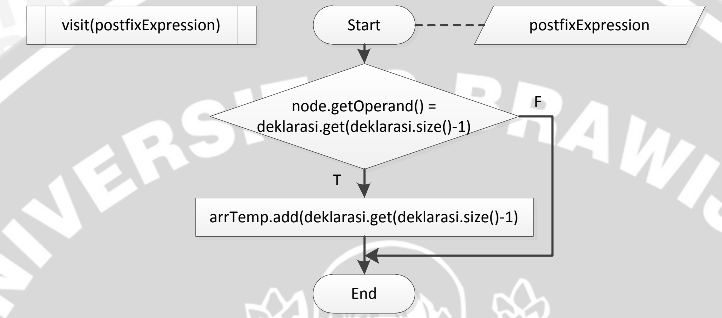
repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

4.2.5.1.7 Diagram Alir visit(PostfixExpression)

Langkah-langkah yang dilakukan untuk melakukan proses visit(postfixExpression) adalah sebagai berikut:

1. Melakukan pengecekan apakah node.getOperand() telah terdefinisi.
2. Apabila node.getOperand() telah terdefinisi maka variabel tersebut akan disimpan pada variabel arrTemp.

Gambar 4.21 menunjukkan diagram alir visit(postfixExpression).



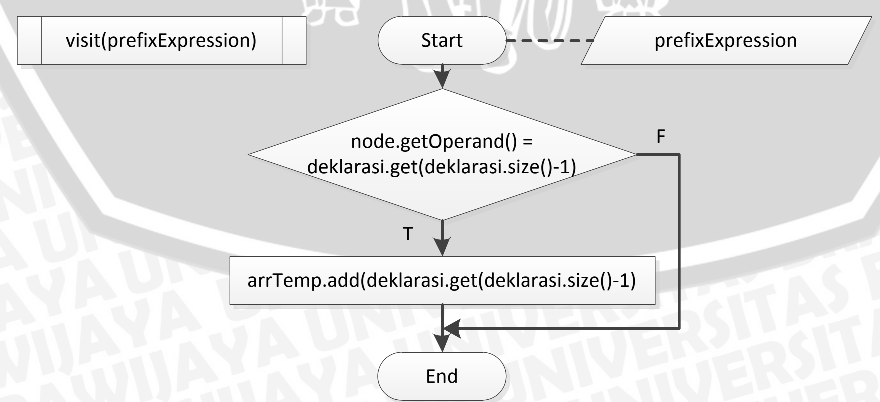
Gambar 4.21 Diagram Alir visit(postfixExpression)

4.2.5.1.8 Diagram Alir visit(PrefixExpression)

Langkah-langkah yang dilakukan untuk melakukan proses visit(prefixExpression) adalah sebagai berikut:

1. Melakukan pengecekan apakah node.getOperand() telah terdefinisi.
2. Apabila node.getOperand() telah terdefinisi maka variabel tersebut akan disimpan pada variabel arrTemp.

Gambar 4.22 menunjukkan diagram alir visit(prefixExpression).



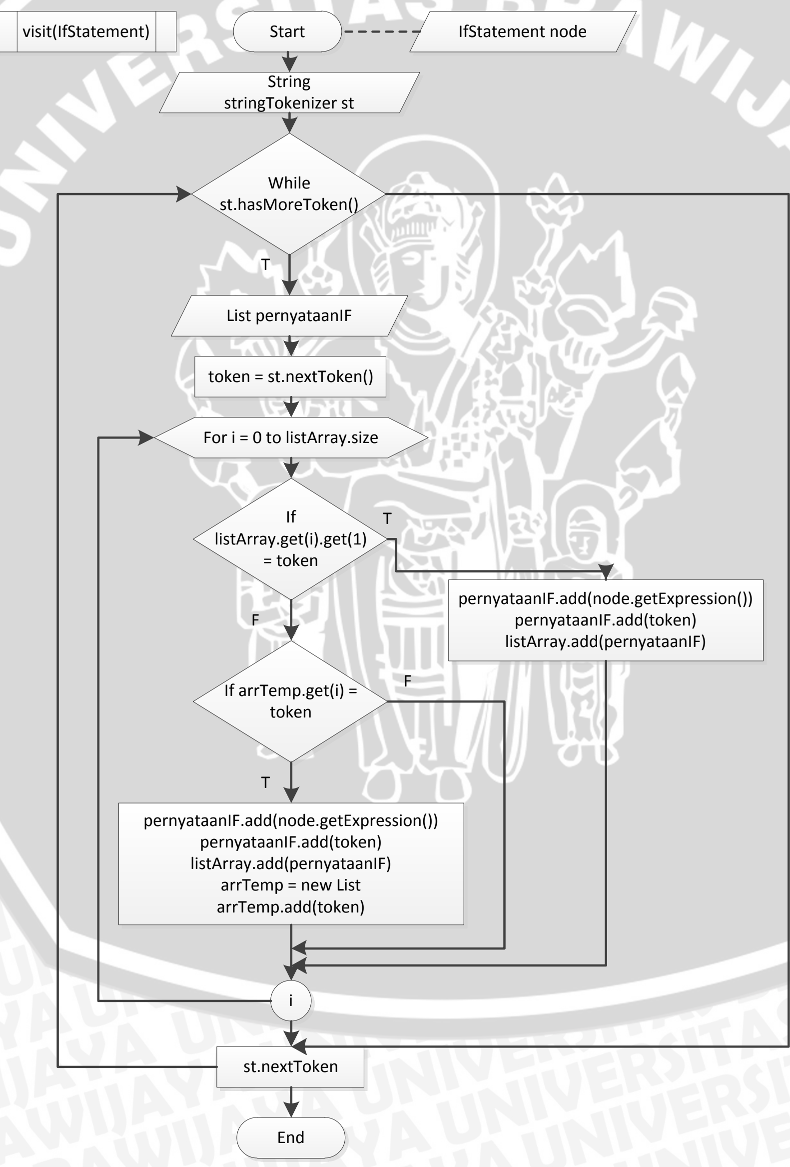
Gambar 4.22 Diagram Alir visit(prefixExpression)

4.2.5.1.9 Diagram Alir visit(IfStatement)

Langkah-langkah yang dilakukan untuk melakukan proses visit(IfStatement) adalah sebagai berikut:

1. Melakukan pemilahan variabel pada IfStatement.
2. Apabila variabel yang didapatkan telah terdefinisi maka variabel tersebut akan ditambahkan ke dalam listArray.
3. Apabila variabel yang didapatkan belum terdefinisi maka variabel tersebut akan ditampung pada arrTemp dan listArray.

Gambar 4.23 menunjukkan diagram alir visit(IfStatement).



Gambar 4.23 Diagram Alir visit(IfStatement)

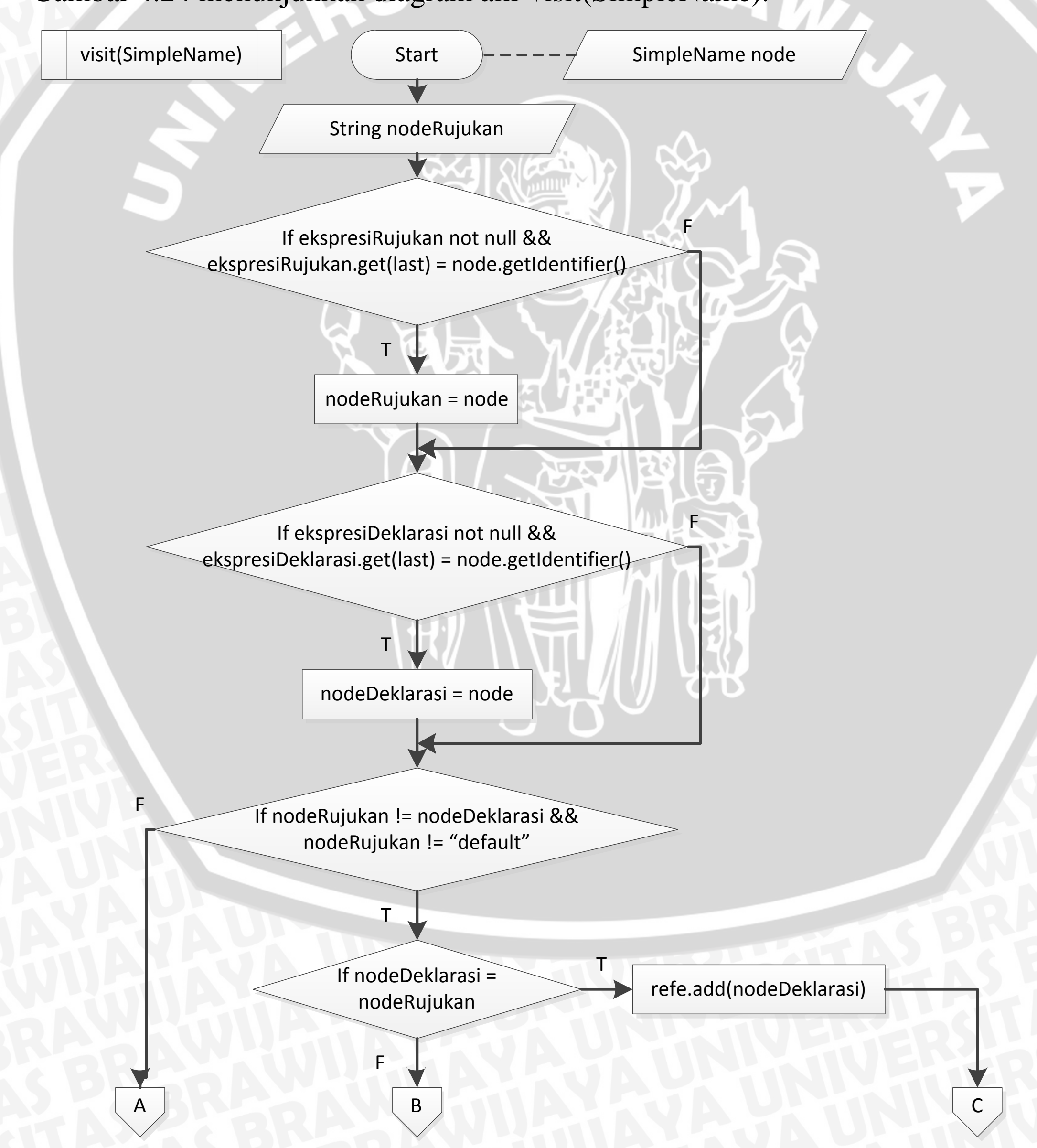
repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

4.2.5.1.10 Diagram Alir visit(SimpleName)

Langkah-langkah yang dilakukan untuk melakukan proses visit(SimpleName) adalah sebagai berikut:

1. Melakukan pengecekan apakah node terdapat pada himpunan ekspresiRujukan, Apabila ada maka nodeRujukan diisi dengan node.
2. Melakukan pengecekan apakah node terdapat pada himpunan ekspresiDeklarasi, Apabila ada maka nodeDeklarasi diisi dengan node.
3. Menambahkan nodeDeklarasi dan nodeRujukan kedalam himpunan refe.
4. Menambahkan himpunan refe kedalam himpunan arrTemp.

Gambar 4.24 menunjukkan diagram alir visit(SimpleName).



repository.ub.ac.id

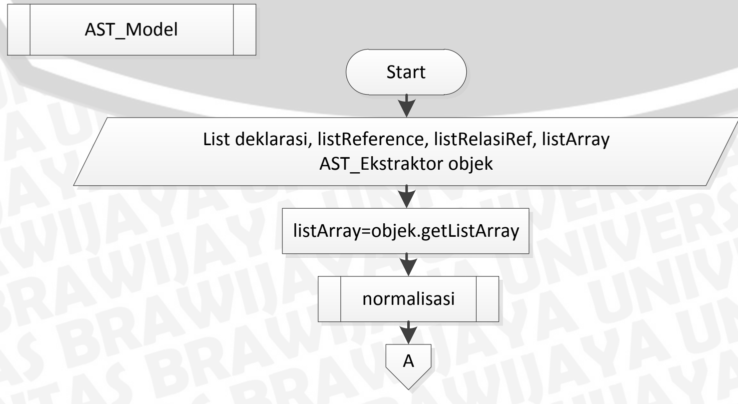
Gambar 4.24 Diagram Alir visit(SimpleName)

4.2.5.2 Diagram Alir AST_Model

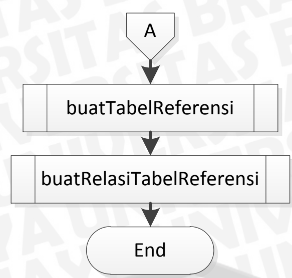
Langkah-langkah yang dilakukan untuk membuat AST_Model adalah sebagai berikut:

1. Melakukan deklarasi variabel deklarasi, listReference, listRelasiRef, listArray, dan objek.
2. Melakukan eksekusi pada *method* normalisasi, buatTabelReferensi, buatRelasiTabelReferensi, dan hapusRedudansi.

Gambar 4.25 menunjukkan diagram alir AST_Model.



UNIVERSITAS BRAWIJAYA



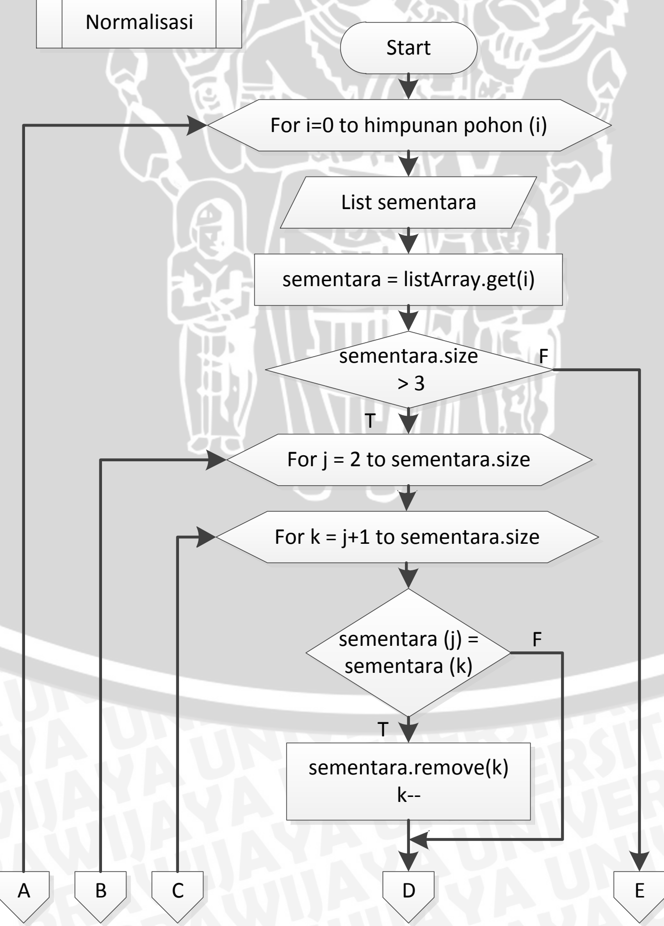
Gambar 4.25 Diagram Alir AST_Model

4.2.5.2.1 Diagram Alir Normalisasi Data

Langkah-langkah yang dilakukan untuk melakukan proses normalisasi data adalah sebagai berikut:

1. Memberi nilai himpunan sementara dengan nilai pada listArray.
2. Apabila terdapat variabel rujukan yang sama pada himpunan sementara maka variabel yang sama akan dihapus.

Gambar 4.26 menunjukkan diagram alir normalisasi data.



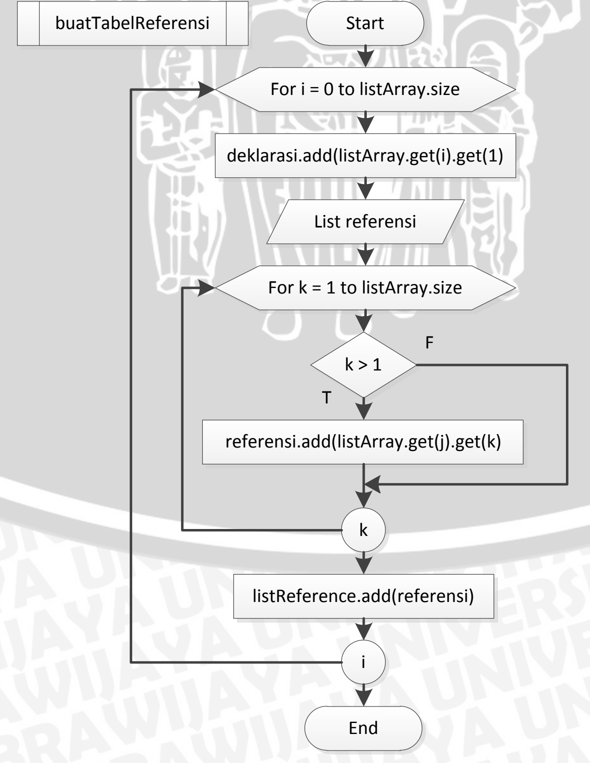
Gambar 4.26 Diagram Alir Normalisasi Data

4.2.5.2.2 Diagram Alir Buat Tabel Referensi

Langkah-langkah yang dilakukan untuk membuat tabel referensi adalah sebagai berikut:

1. Pemberian nilai himpunan deklarasi dengan elemen listArray.
2. Menambahkan semua elemen pada listArray yang mempunyai nilai variabel rujukan kedalam himpunan referensi.
3. Menambahkan himpunan referensi kedalam himpunan listReference.

Gambar 4.27 menunjukkan diagram alir buat tabel referensi.



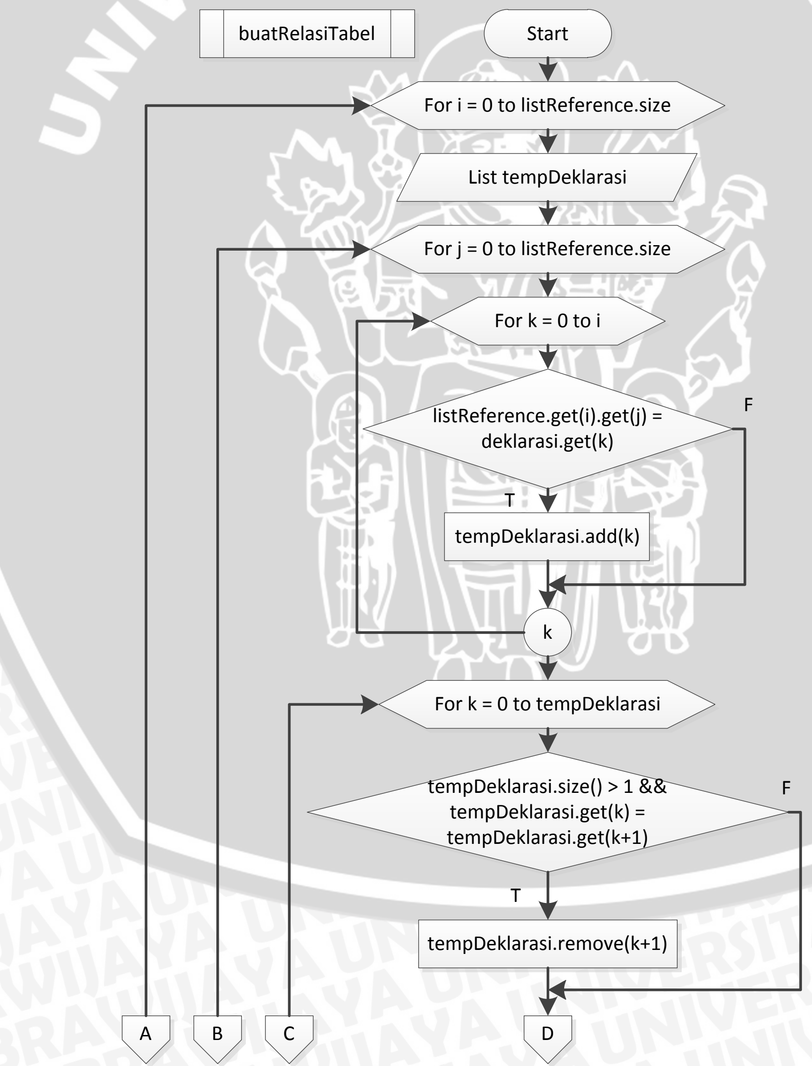
Gambar 4.27 Diagram Alir Buat Tabel Referensi

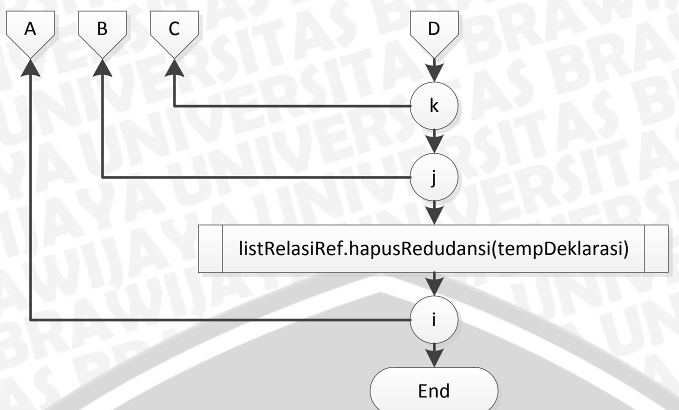
4.2.5.2.3 Diagram Alir Buat Relasi Tabel

Langkah-langkah yang dilakukan untuk membuat relasi tabel adalah sebagai berikut:

1. Melakukan pengecekan terhadap elemen listArray, apabila terdapat variabel deklarasi pada listArray maka indeks elemen tersebut akan disimpan pada variabel tempDeklarasi.
2. Melakukan pengecekan pada himpunan tempDeklarasi, apabila terdapat indeks yang sama maka indeks tersebut akan dihapus.
3. Melakukan pemanggilan method hapusRedudansi untuk menghilangkan redudansi pada tabel ketergantungan.

Gambar 4.28 menunjukkan diagram alir buat relasi tabel.





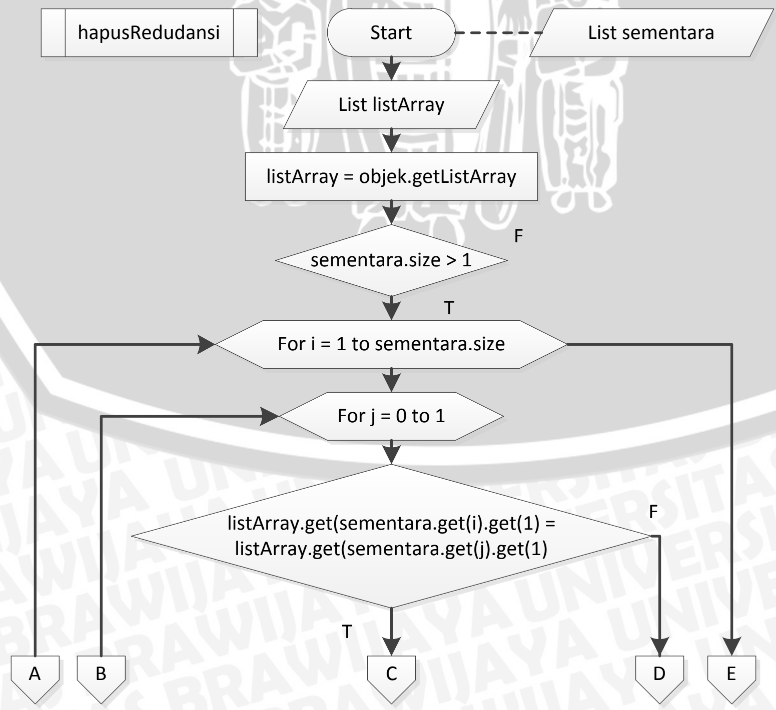
Gambar 4.28 Diagram Alir Buat Relasi Tabel

4.2.5.2.4 Diagram Alir Hapus Redudansi

Langkah-langkah yang dilakukan untuk melakukan proses hapus redudansi adalah sebagai berikut:

1. Melakukan pengecekan jumlah data pada himpunan sementara, jika jumlah data lebih dari 1 maka dilakukan pengecekan antara satu data dan data lainnya pada himpunan sementara.
2. Apabila terdapat duplikasi data pada himpunan sementara maka data tersebut akan dihapus.

Gambar 4.29 menunjukkan diagram alir hapus redudansi.



repository.ub.ac.id

A B C D E
sementara.remove(j)
i--
j--
j
i
End

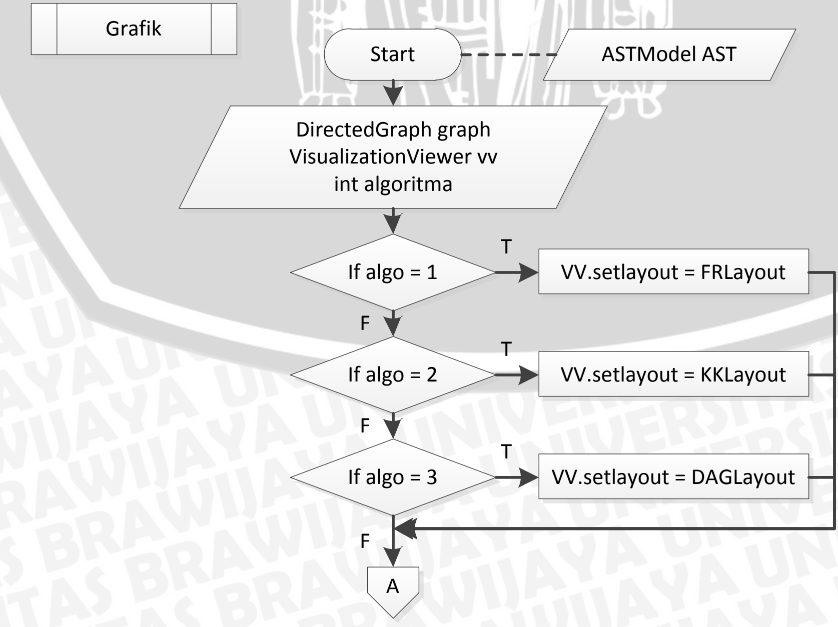
Gambar 4.29 Diagram Alir Hapus Redudansi

4.2.5.3 Diagram Alir Buat Grafik

Langkah-langkah yang dilakukan untuk melakukan proses buat grafik adalah sebagai berikut:

1. Pendeklarasian DirectedGraph, VisualizationViewer, dan algoritma.
2. Jika nilai algoritma adalah 1 maka tata letak yang dipakai adalah KKLayout, Jika nilai algoritma adalah 2 maka tata letak yang dipakai adalah DAGLayout, dan apabila nilai algoritma adalah 3 maka tata letak yang dipakai adalah FRLayout.
3. Melakukan pemanggilan method createTree().

Gambar 4.30 menunjukkan diagram alir buat grafik.



UNIVERSITAS BRAWIJAYA

A
createTree
End

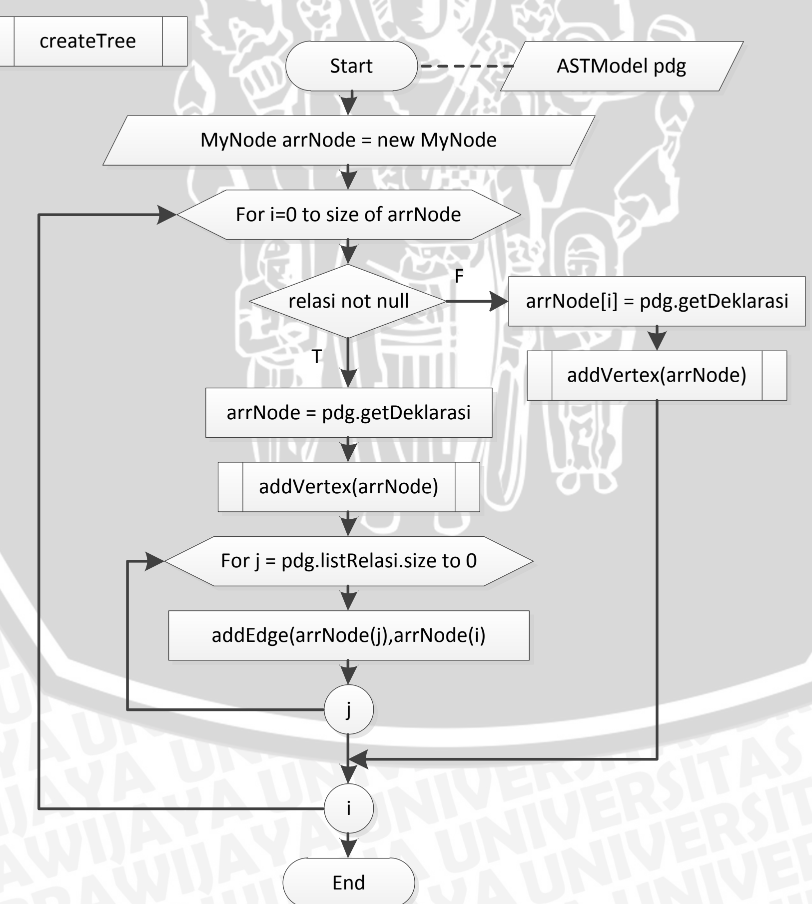
Gambar 4.30 Diagram Alir Buat Grafik

4.2.5.3.1 Diagram Alir Create Tree

Langkah-langkah yang dilakukan untuk melakukan proses create tree adalah sebagai berikut:

1. Melakukan deklarasi arrNode sebagai sebuah himpunan MyNode.
2. Apabila node tidak mempunyai relasi maka node akan dibuat, dan apabila mempunyai relasi node akan ditambahkan dengan garis penunjuk ke pada node rujukan.

Gambar 4.31 menunjukkan diagram alir create tree.



Gambar 4.31 Diagram Alir Create Tree

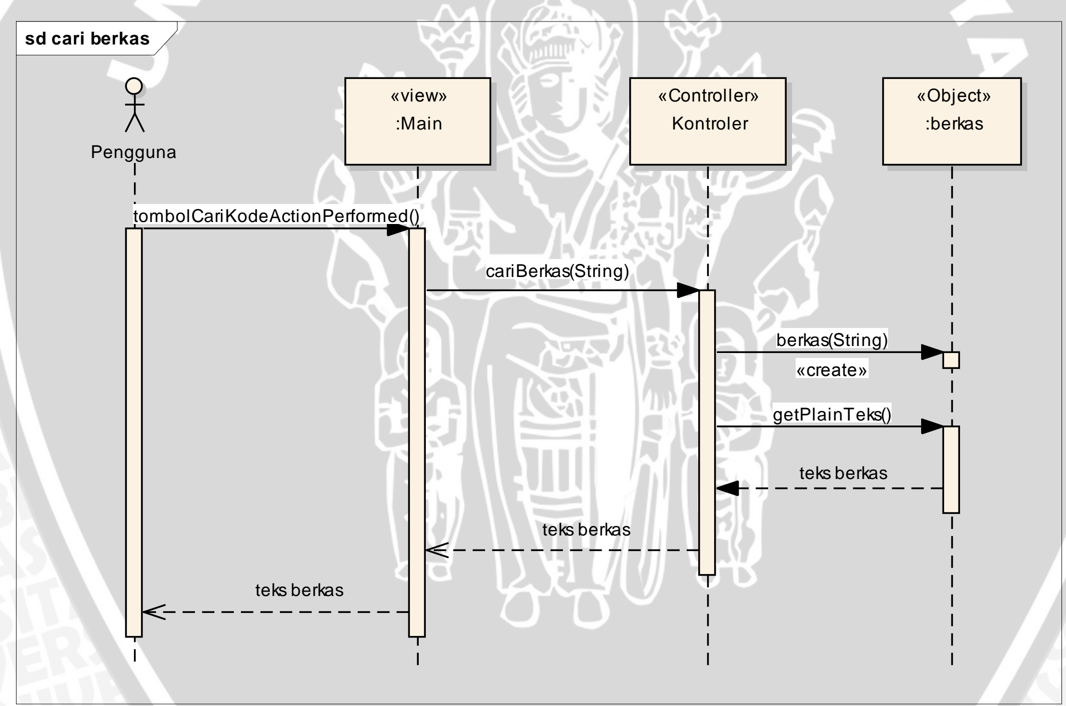
repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

4.2.6 Perancangan *Sequence Diagram*

Sequence diagram digunakan untuk menggambarkan interaksi antar objek dan mengetahui serangkaian pesan yang saling diberikan oleh setiap objek. Objek-objek tersebut diurutkan dari kini ke kanan dengan aktor berada di paling kiri diagram. Pada diagram ini, bagian paling atas menjadi titik awal interaksi hingga ke bagian bawah diagram sebagai akhir iterasi. Tanda panah merepresentasikan pesan yang dikirimkan oleh objek pengirim kepada objek penerima.

4.2.6.1 *Sequence Diagram* Cari Berkas

Sequence diagram cari berkas merupakan model interaksi antar objek yang berada dalam proses cari berkas. Adapun objek yang digunakan adalah Pengguna, Main, Kontroler, dan Berkas.

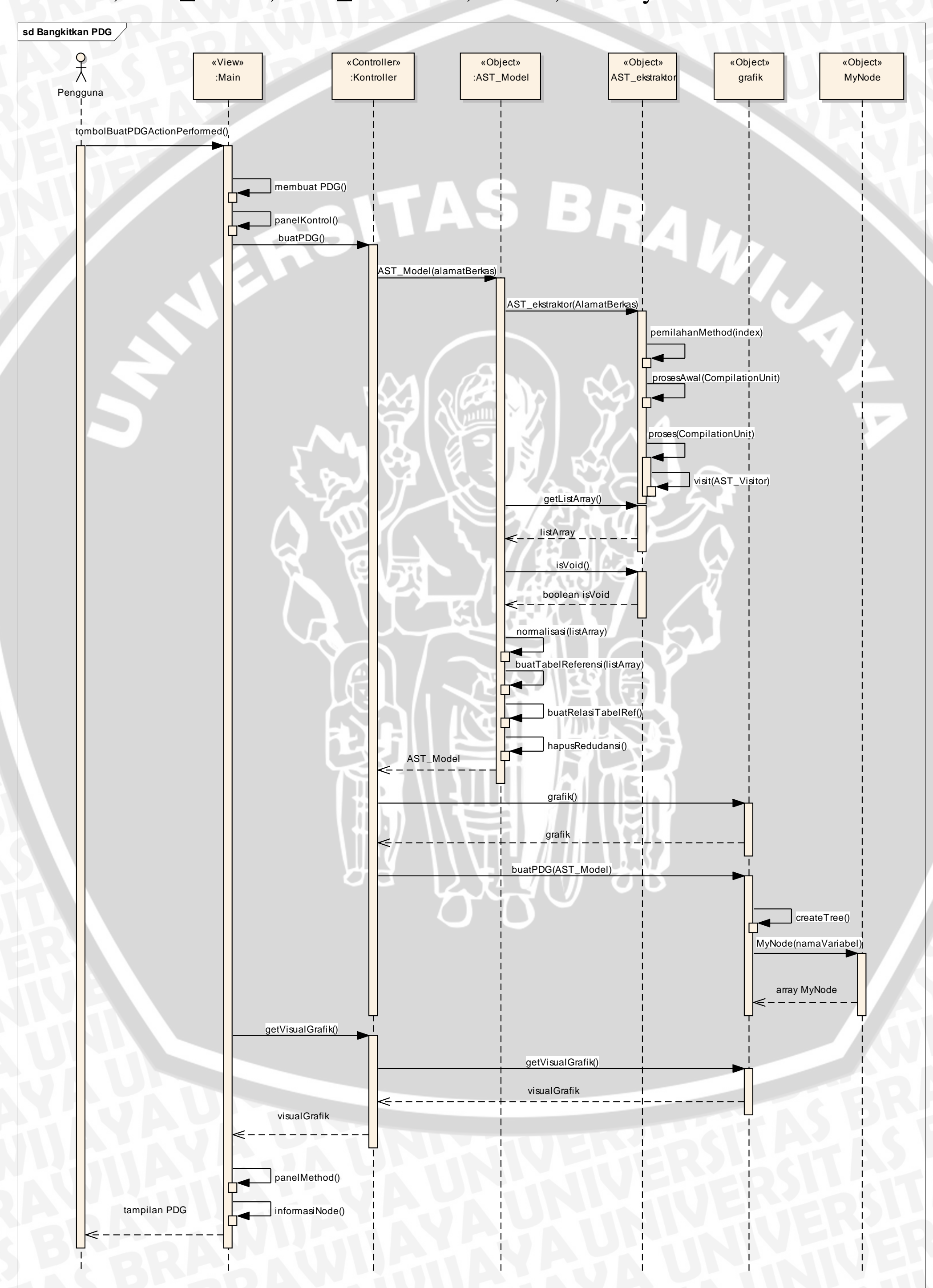


Gambar 4.32 *Sequence Diagram* Cari Berkas

Gambar 4.32 menjelaskan bahwa proses cari berkas berawal dari pengguna memberi perintah kepada Main untuk menjalankan *method* cariBerkas pada Kontroler. Selanjutnya Kontroler menginstansiasi objek berkas dengan mengirim alamat berkas, lalu isi berkas akan dikirimkan kembali ke pengguna melalui Kontroler dan Main.

4.2.6.2 *Sequence Diagram* Bangktikan *Program Dependence Graph*

Sequence diagram bangkitkan *Program Dependence Graph* merupakan model interaksi antar objek yang berada dalam proses membangktkan *Program Dependence Graph*. Adapun objek yang digunakan adalah Pengguna, Main, Kontroler, AST_Model, AST_ekstraktor, Grafik, dan MyNode.



Gambar 4.33 *Sequence Diagram* Bangkitkan *Program Dependence Graph*

Gambar 4.33 menjelaskan bahwa proses bangkitkan *Program Dependence Graph* berawal dari pengguna menekan tombol "Bangkitkan PDG". Selanjutnya Main akan membuat wadah bagi *Program Dependence Graph* dan memberi pesan kepada Kontroler untuk meneruskan proses membuat *Program Dependence Graph* yang selanjutnya akan dilakukan ekstraksi pada *method* AST_Ekstraktor. Kemudian dilakukan pemilahan *method* yang bertujuan untuk memisahkan setiap *method* yang ada pada berkas, lalu dilakukan proses awal untuk melakukan ekstraksi variabel dalam berkas.

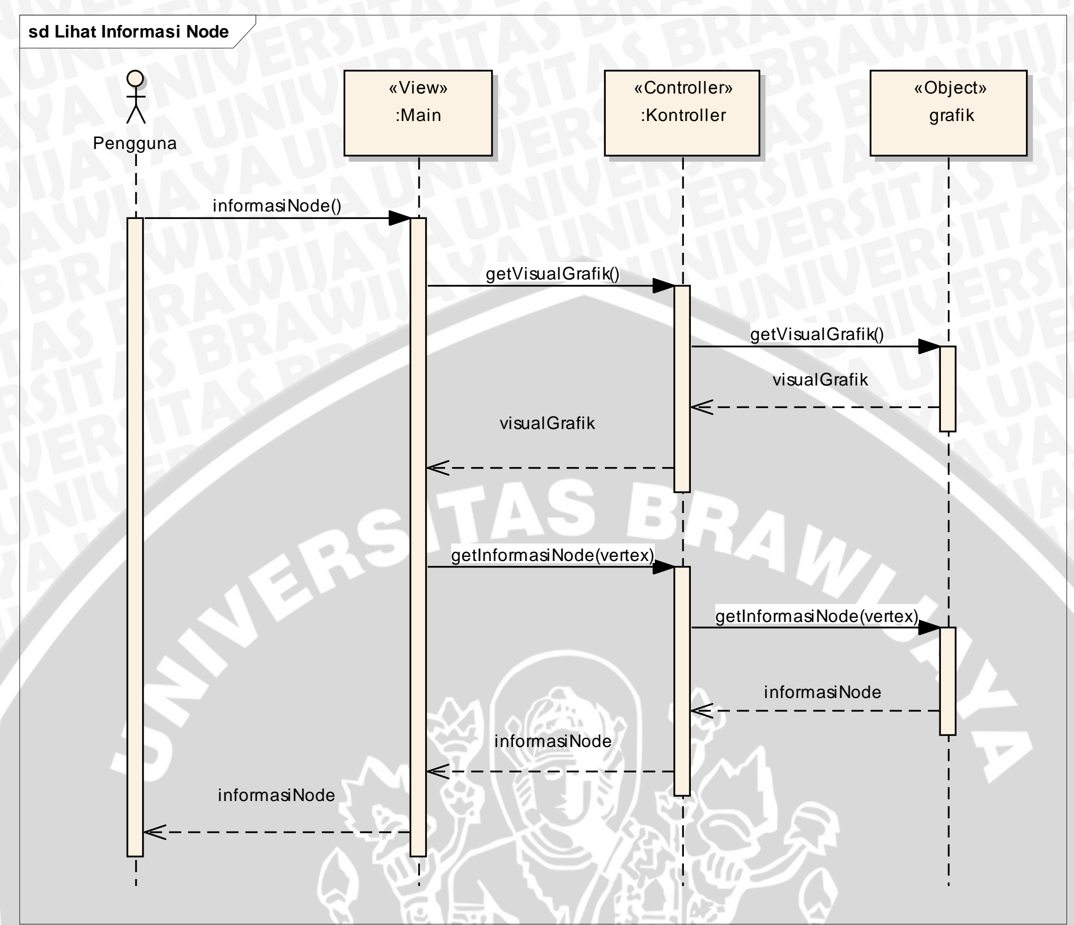
Setelah melakukan ekstraksi, objek AST_Model akan mengambil data hasil ekstraksi, lalu melakukan normalisasi data (listArray) untuk diubah ke dalam bentuk tabel referensi. kemudian dilakukan proses menghapus redudansi. Selanjutnya Kontroler akan menginstansiasi objek Grafik dan memanggil *method* buatPDG(AST_Model). Lalu objek Grafik memanggil *method* createArray dan menginstansiasi kelas MyNode yang digunakan untuk membuat *node*. Setelah itu Main akan mengambil grafik pada objek Grafik melalui Kontroler dan menghasilkan tampilan *Program Dependence Graph* yang dapat dilihat oleh pengguna.

4.2.6.3 *Sequence Diagram* Lihat Informasi *Node*

Sequence diagram lihat informasi *node* merupakan model interaksi antar objek yang berada dalam proses lihat informasi *node*. Adapun objek yang digunakan adalah Pengguna, Main, Kontroler, dan Grafik.

Gambar 4.34 menjelaskan bahwa proses lihat informasi *node* berawal dari pengguna memberikan pesan kepada Main untuk mendapatkan informasi node. Kemudian Main akan meminta visual grafik pada kelas Kontroler dan mendapat visual grafik dari objek Grafik melalui Kontroler. Lalu Main akan mengambil informasi *node* pada objek Grafik dengan membawa informasi *node* yang dipilih. Selanjutnya informasi *node* akan diberikan kepada pengguna dan pengguna dapat melihat informasi *node* tersebut.

repository.ub.ac.id UNIVERSITAS BRAWIJAYA

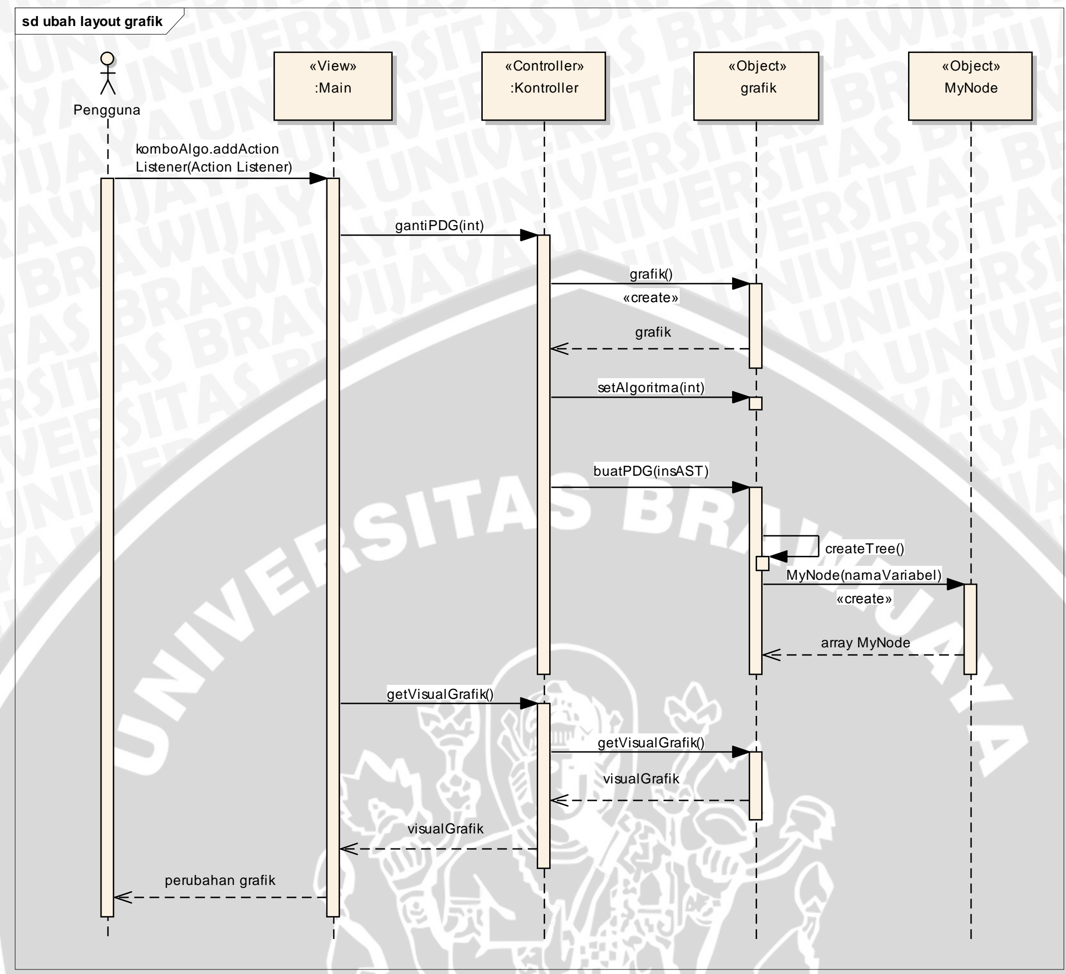


Gambar 4.34 *Sequence Diagram* Lihat Informasi *Node*

4.2.6.4 *Sequence Diagram* Ubah *Layout* Grafik

Sequence diagram ubah *layout* grafik merupakan model interaksi antar objek yang berada dalam proses ubah *layout* grafik. Adapun objek yang digunakan adalah Pengguna, Main, Kontroler, Grafik, dan MyNode.

Gambar 4.35 menjelaskan bahwa proses ubah *layout* grafik berawal dari pengguna memilih *layout* dan memberi pesan kepada Main untuk mengubah layout grafik. Selanjutnya Main akan menjalankan gantiPDG(int) pada Kontroler. Kemudian Kontroler akan menginstansiasi objek grafik, lalu Kontroler akan mengubah algoritma serta menjalankan *method* buatPDG(insAST) pada objek Grafik. Setelah itu objek Grafik akan menginstansiasi kelas MyNode sebagai pembangun *node* dalam grafik. Selanjutnya Main akan mengambil visual grafik dari objek Grafik melalui Kontroler dan perubahan grafik akan diberikan kepada pengguna.

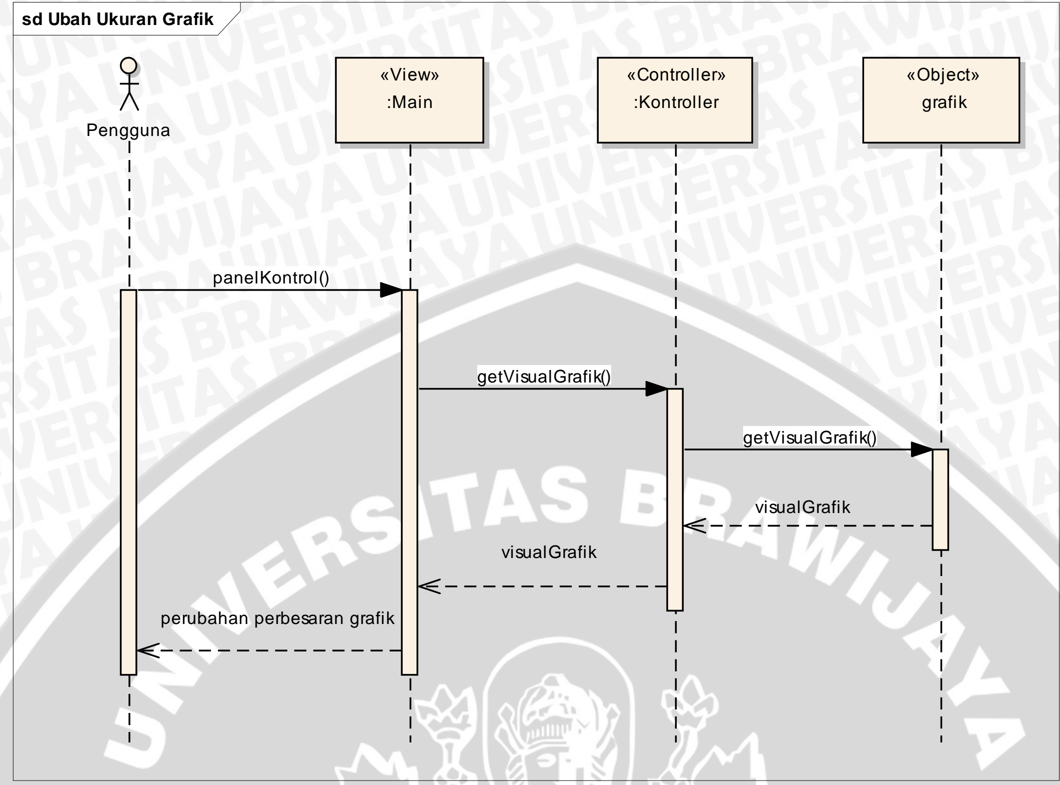


Gambar 4.35 *Sequence Diagram* Ubah *Layout* Grafik

4.2.6.5 *Sequence Diagram* Ubah Ukuran Grafik

Sequence diagram ubah ukuran grafik merupakan model interaksi antar objek yang berada dalam proses ubah ukuran grafik. Adapun objek yang digunakan adalah Pengguna, Main, Kontroler, dan Grafik.

Gambar 4.36 menjelaskan bahwa proses ubah ukuran grafik berawal dari pengguna memberikan pesan panelKontrol() kepada objek Main untuk mengubah ukuran Grafik. Objek Main akan mengambil visual grafik dari objek Grafik melalui Kontroler. Proses perubahan nilai ukuran grafik terjadi pada *method* panelKontrol(), selanjutnya perubahan perbesaran grafik akan diberikan kepada pengguna.

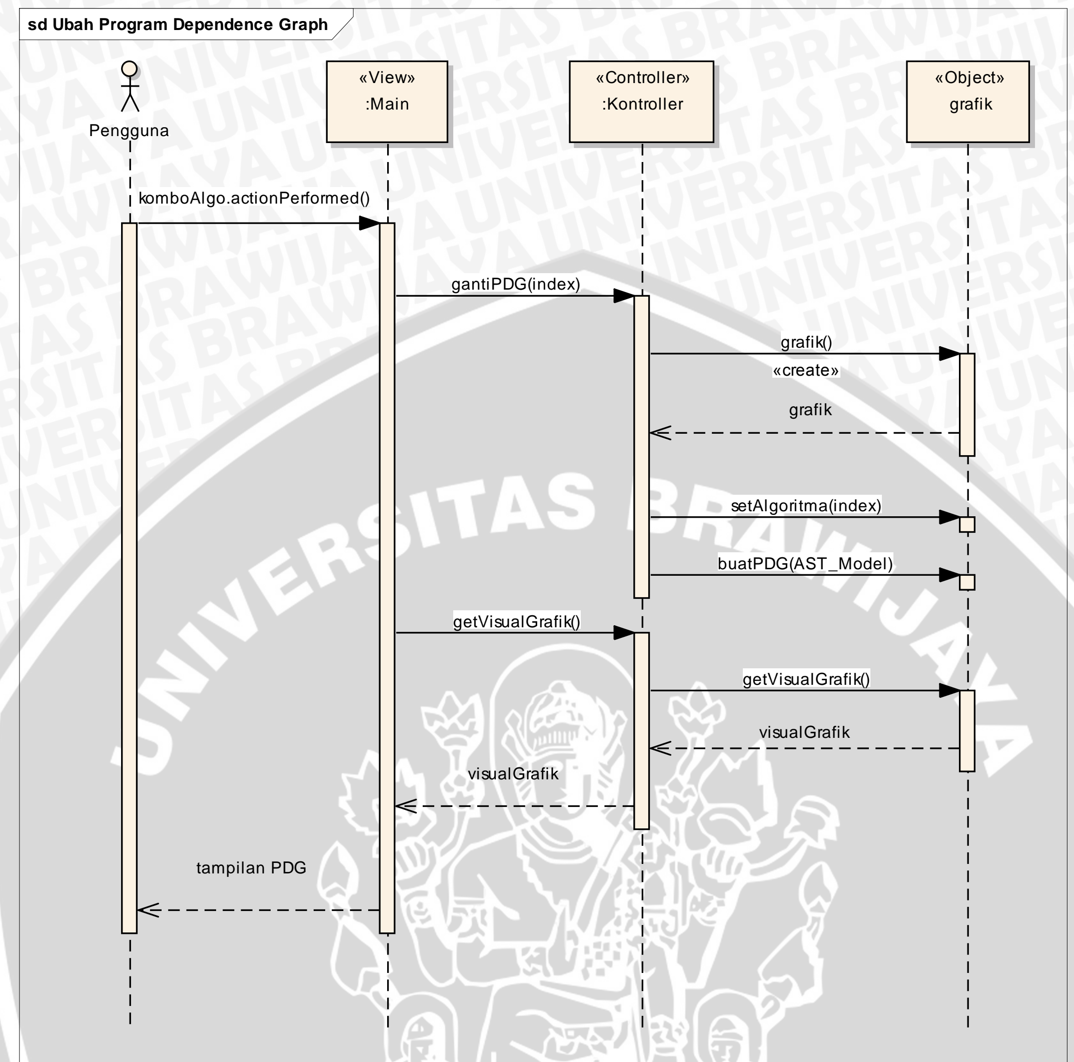


Gambar 4.36 *Sequence Diagram* Ubah Ukuran Grafik

4.2.6.6 *Sequence Diagram* Ubah *Program Dependence Graph*

Sequence diagram ubah *Program Dependence Graph* merupakan model interaksi antar objek yang berada dalam proses ubah *Program Dependence Graph*. Adapun objek yang digunakan adalah Pengguna, Main, Kontroler, dan Grafik.

Gambar 4.37 menjelaskan bahwa proses ubah *Program Dependence Graph* berawal dari pengguna memberikan kepada Main melalui pemilihan algoritma. Selanjutnya objek Main akan menjalankan *method* gantiPDG(index) pada Kontroler, lalu objek Kontroler akan menginstansiasi objek Grafik dan menjalankan setAlgoritma(index) dan buatPDG(AST_Model). Setelah itu objek Main akan meminta visual grafik kepada kelas Kontroler dan kelas Kontroler akan mengambil visiaul grafik dari dari objek Grafik. Selanjutnya tampilan *Porgram Dependence Graph* yang baru akan diberikan kepada pengguna dan pengguna dapat melihat *Porgram Dependence Graph* yang telah terubah.

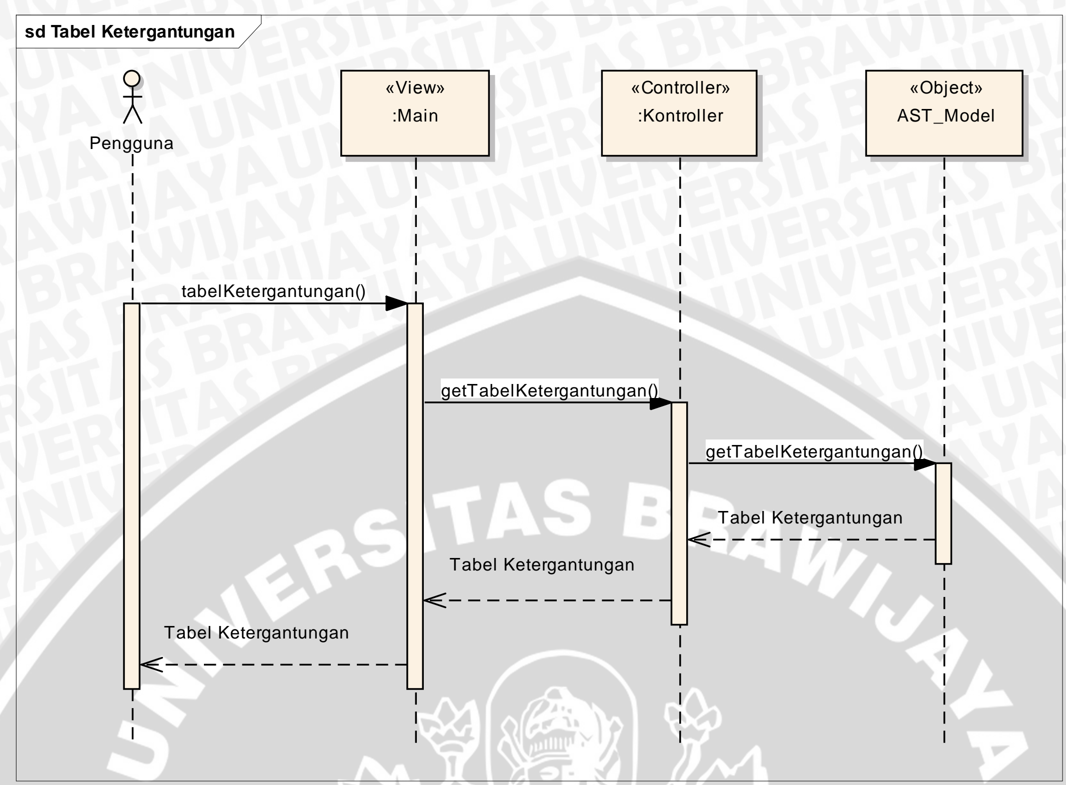


Gambar 4.37 *Sequence Diagram* Ubah *Program Dependence Graph*

4.2.6.7 *Sequence Diagram* Lihat Tabel Ketergantungan

Sequence diagram lihat tabel ketergantungan merupakan model interaksi antar objek yang berada dalam proses lihat tabel ketergantungan. Adapun objek yang digunakan adalah Pengguna, Main, Kontroler, dan AST_Model.

Gambar 4.38 menjelaskan bahwa proses lihat tabel ketergantungan berawal dari pengguna memberikan pesan kepada Main untuk melihat tabel ketergantungan. Selanjutnya objek Main akan mengambil tabel ketergantungan dari objek AST_Model melalui objek Kontroler. Lalu tabel ketergantungan akan diberikan kepada pengguna.



Gambar 4.38 *Sequence Diagram* Lihat Tabel Ketergantungan

4.2.7 Perancangan *Component Diagram*

Component diagram digunakan menggambarkan komponen yang berada dalam sistem dan hubungan antar komponen. Gambar 4.39 menunjukkan *component diagram* yang digunakan pada pembuatan pembangunan kakas bantu pembangkitan *Program Dependence Graph* berbasis Java.

Gambar 4.39 menjelaskan tentang hubungan komponen yang digunakan pada sistem. Komponen yang dikembangkan pada penelitian ini adalah kontroler, AST_Ekstraktor, AST_Model, dan Grafik. Sedangkan komponen Abstract Syntax Tree dan JUNG dikembangkan oleh pihak lain. Komponen Abstact Syntax Tree digunakan untuk membuat skema pohon yang selanjutnya variabel dalam skema pohon tersebut didapatkan dengan menggunakan komponen AST_Ekstraktor. Komponen AST_Model berfungsi untuk menormalisasi variabel yang didapatkan. Komponen Grafik berfungsi untuk menyusun node sehingga dapat dikonversi menjadi grafik pada komponen JUNG .

Gambar 4.39 *Component Diagram*

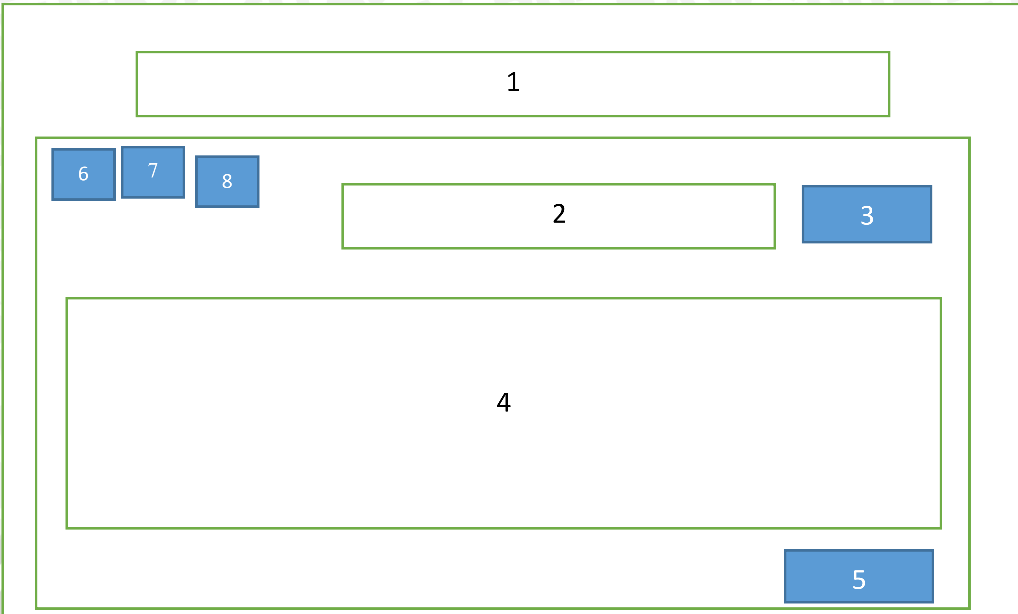
4.2.8 Perancangan Antarmuka

Perancangan antarmuka merupakan pembuatan desain rancangan *user interface* agar pengguna dapat dengan mudah berkomunikasi dengan sistem. Perancangan antarmuka merupakan bagian dari kelas Main yang mempunyai tiga halaman yaitu halaman utama, halaman *Program Dependence Graph,* dan halaman tabel ketergantungan. Berikut ini adalah perancangan antarmuka sistem pembangunan kakas bantu pembangkitan *Program Dependence Graph* berbasis Java.

4.2.8.1 Perancangan Halaman Utama

Antarmuka halaman utama merupakan antarmuka untuk pengguna mencari berkas *source code* yang mempunyai ekstensi .java. Kemudian isi dari berkas tersebut akan ditampilkan pada halaman utama. Selain itu pengguna dapat melakukan pembangkitan *Porgram Dependence Graph* dengan menekan tombol "Bangkitkan PDG" dari halaman ini. Gambar 4.40 menunjukkan rancangan halaman utama.

repository.ub.ac.id



Gambar 4.40 Rancangan Halaman Utama

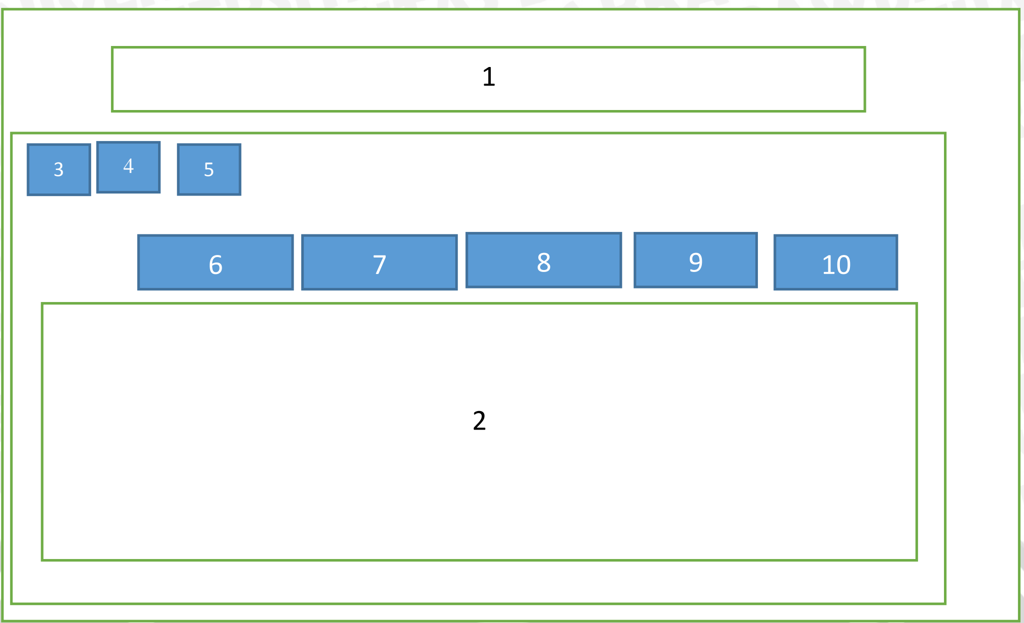
Keterangan Gambar 4.40:

1. Menampilkan judul sistem.
2. Menampilkan alamat direktori file yang dipilih oleh pengguna.
3. Button untuk mencari berkas *source code*.
4. Text field untuk menampilkan *source code* yang dipilih oleh pengguna.
5. Buton untuk membangkitkan grafik *Program Dependence Graph* dari *source code*.
6. Panel tabulasi halaman utama.
7. Panel tabulasi halaman *Program Dependence Graph*.
8. Panel tabulasi halaman tabel ketergantungan.

4.2.8.2 Perancangan Halaman *Program Dependence Graph*

Antarmuka halaman *Program Dependence Graph* merupakan antarmuka untuk pengguna melihat grafik *Program Dependence Graph*, melihat informasi *node,* mengubah tata letak *Program Dependence Graph*, dan mengubah *Program Dependence Graph* berdasarkan *method* yang akan dipilih. Gambar 4.41 menunjukkan rancangan halaman *Program Dependence Graph*.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA



Gambar 4.41 Rancangan Halaman *Program Dependence Graph*

Keterangan Gambar 4.41:

1. Menampilkan judul sistem.
2. Menampilkan grafik *Program Dependence Graph.*
3. Panel tabulasi halaman utama.
4. Panel tabulasi halaman *Program Dependence Graph.*
5. *Panel* tabulasi halaman tabel ketergantungan.
6. *Combo box* mode operasi grafik.
7. *Panel* Perbesaran grafik.
8. *Combo box* daftar algoritma tata letak grafik.
9. *Panel* informasi *node*
10. *Combo box* daftar *method* yang ada

4.2.8.3 Perancangan Halaman Tabel Ketergantungan

Antarmuka halaman tabel ketergantungan merupakan antarmuka untuk pengguna melihat tabel ketergantungan antar variabel dalam *source code*. Gambar 4.42 menunjukkan rancangan halaman tabel ketergantungan.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Gambar 4.42 Rancangan Halaman Tabel Ketergantungan

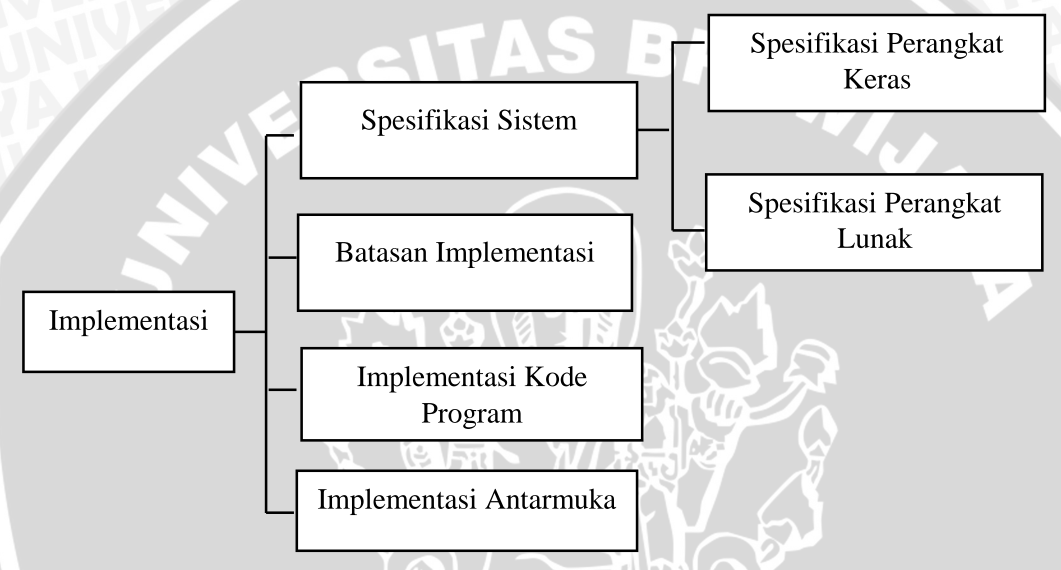
Keterangan Gambar 4.42:

1. Menampilkan judul sistem.
2. Menampilkan Tabel Ketergantungan.
3. Panel tabulasi halaman utama.
4. Panel tabulasi halaman *Program Dependence Graph.*
5. *Panel* tabulasi halaman tabel ketergantungan.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB V
IMPLEMENTASI

Bab ini menguraikan tentang implementasi perangkat lunak berdasarkan hasil dari analisis kebutuhan dan perancangan perangkat lunak. Tahapan yang digunakan pada implementasi ini terdiri dari spesifikasi kebutuhan, batasan-batasan implementasi, implementasi kode program, dan implementasi antarmuka. Gambar 5.1 menunjukkan diagram alir proses implementasi pada penelitian ini.



Gambar 5.1 Diagram Alir Implementasi

5.1 Spesifikasi Sistem

Spesifikasi system merupakan lingkungan implementasi dalam pembuatan sistem ini yang terdiri dari spesifikasi perangkat keras dan perangkat lunak.

5.1.1 Spesifikasi Perangkat Keras

Dalam penelitian ini penulis hanya menggunakan satu buah komputer karena sistem yang dibangun tidak berkomunikasi dengan perangkat keras yang lain. Spesifikasi perangkat keras yang digunakan dalam pembuatan sistem pembangunan kakas bantu pembangkitan *Program Dependence Graph* berbasis Java dapat dilihat pada tabel 5.1.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Tabel 5.1 Spesifikasi Perangkat Keras

Nama Komponen	Spesifikasi
System Model	Lenovo Idepad Z410
Processor	Intel® Core^{TM} i5-4200M
Memory	1TB HDD 5400 RPM
Memory RAM	4GB DDR3L 1600 MHz SRAM
Display	14” LED HD 1366 x 728 pixel 16:9 aspect ratio

5.1.2 Spesifikasi Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan dalam pembuatan sistem pembangunan kakas bantu pembangkitan *Program Dependence Graph* berbasis Java dapat dilihat pada table 5.2.

Tabel 5.2 Spesifikasi Perangkat Lunak

Nama Komponen	Spesifikasi
Operating System	Windows 8.1 Pro 64-bit
Programming Language	Java
Text Editor / IDE	Netbeans IDE 7.3.1

5.2 Batasan Implementasi

Berikut ini merupakan batasan implementasi yang digunakan dalam pembuatan sistem pembangunan kakas bantu pembangkitan *Program Dependence Graph* berbasis Java.

1. Data yang digunakan dalam sistem ini diambil dari *source code* masukan dari pengguna dengan tipe Java.
2. Untuk mengakses sistem ini pengguna harus memiliki Java Runtime Environment (JRE).

5.3 Implementasi Kode Program

Implementasi kode program merupakan bagian dari penerapan perancangan ke dalam bentuk bahasa pemrograman.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

5.3.1 Implementasi Proses Ekstraksi Data

Proses ini merupakan proses untuk mengubah *source code* ke dalam skema pohon untuk mendapatkan variabel-variabel pada skema pohon dengan menggunakan pustaka *Abstract Syntax Tree.* Implementasi proses ekstraksi data seperti pada gambar 5.2.

Nama Algoritma: Ekstraksi Data
Deskripsi:
Masukan: *Source code*

```
Inisialisasi:
String alamat, ASTParser parser, Compilation Unit cu, List
parameterMethod, deklarasi, ekspresiDeklarasi, ekspresiRujukan,
listArray, arrTemp, boolean parseAble;

Proses:
1. parser = new parser
2. parser.setSource(str.tocharArray)
3. parser.setKind(K_Compilation_Unit)
4. cu = parser.createAST
5. Call : PemilihanMethod()
6. Call : ProsesAwal(0)
7. If error occur
8.      String isiPerbaikan("class PDG"+str)
9.      parser.setSource(isiPerbaikan)
10.     Parser.setKind(K_Compilation_Unit)
11.     cu = parser.createAST
12.     Call : PemilihanMethod()
13.     Call : ProsesAwal()
14.     If error occur
15.         parseAble ← false
16.     End if
17. End if
18. End
```

Gambar 5.2 Proses Ekstraksi Data

Penjelasan gambar 5.2:

Baris 1-3 : Instansiasi ASTParser lalu memberi informasi masukan beserta tipenya ke dalam objek parser.

Baris 4 : Membentuk skema pohon.

Baris 5 : Memilah method yang ada pada *source code*.

Baris 6 : Pemanggilan *method* prosesAwal() yang berguna untuk memberi nilai awal pada variabel.

Baris 7-18 : Penambahan informasi nama kelas pada *source code* dan membuat ulang skema pohon.

5.3.1.1 Implementasi Proses Awal Ekstraksi Data

Proses ini merupakan proses awal untuk mendapatkan variabel-variabel pada skema pohon dengan menggunakan pustaka *Abstract Syntax Tree.* Implementasi proses awal ekstraksi data seperti pada gambar 5.3.

Nama Algoritma: Proses Awal Ekstraksi Data
Deskripsi:
Masukan: Compilation Unit

```
Insialisasi:
-

Proses:
1. cu.accept(new ASTVisitor()
2. Visit(ASTVisitor)
3. If (arrTemp not null)
4.     If (arrTemp.size > 2)
5.         For i ← 2 to arrTemp.size
6.             Boolean referensiTerdefinisi ← false
7.             For j ← 0 to listArray.size
8.                 If (listArray.get(j).get(1) ← arrTemp(i)
9.                     referensiTerdefinisi ← true
10.                End if
11.            End for
12.            If (referensiTerdefinisi ← false
13.                new List buatDefinisiReferensi
14.                buatDefinisiReferensi.add(arrTemp.get(i)
15.                listArray.add(buatDefinisiReferensi)
16.            End if
17.        End for
18.        listArray.add(arrTemp)
19.    End if
20. End if
21. End
```

Gambar 5.3 Proses Awal Ekstraksi Data

Penjelasan gambar 5.3:

Baris 2 : Melakukan pengecekan pada setiap variabel dalam skema pohon dan menyimpan variabel tersebut.

Baris 3-11 : Melakukan pengecekan pada himpunan apakah variabel rujukan yang didapatkan telah ada atau belum.

Baris 12-21 : Jika variabel rujukan belum di definisikan maka variabel rujukan tersebut akan ditambahkan ke dalam himpunan.

5.3.1.2 Implementasi Proses visit(MethodInvocation)

Proses ini merupakan proses untuk mendapatkan informasi variabel pada pernyataan pemanggilan *method.* Variabel didapatkan dari argumen pada parameter *method*. Implementasi proses visit(MethodInvocation) seperti pada gambar 5.4.

Nama Algoritma: visit(MethodInvocation)
Deskripsi:
Masukan: MethodInvocation

```
Insialisasi:
List argumenSementara;
boolean argumenRumit ← false;

Proses:
1. If argument not null
2.     For i ← 0 to node.arguments.size
3.         x ←(MethodInvocation) argumen
4.         argumenSementara.add(x.getExpression)
5.         For j=0 to size of argument
6.             argumenSementara.add(x.argumen.get(j))
7.         End for
8.         argumenRumit = true
9.     End for
10. End if
11. If argumen not null && argumenRumit = false
12. For i ← 0 to size of argument
13.     For j ← 0 tolistArray.size
14.         If nodeArgumen(i) ← listArray(j)
15.             argumenSementara.add(node.argumen.get(i))
16.         End if
17.     End for
18. End for
19. new List sementara
20. sementara.add(node)
21. If argumen sementara not null
22.     For i ← 0 to argumenSementara.size
23.         sementara.add(argumenSementara(i))
24.     End for
25. End if
```

```
26. End
```

Gambar 5.4 Proses visit(MethodInvocation)

Penjelasan gambar 5.4:

Baris 1-10 : Melakukan pengecekan terhadap pernyataan pemanggilan *method*, apabila terdapat parameter maka parameter dan argumen akan disimpan pada variabel argumenSementara.

Baris 11-18 : Jika pernyataan pemanggilan *method* mengandung pernyataan pemanggilan *method* lainnya maka semua variabel yang terkandung didalamnya akan disimpan.

Baris 19-20 : Pendeklarasian himpunan sementara untuk menampung pernyataan pemanggilan *method*.

Baris 21-26 : Jika terdapat argumen pada pernyataan pemanggilan *method,* himpunan argumenSementara akan ditambahkan ke dalam himpunan sementara.

5.3.1.3 Implementasi Proses visit(MethodDeclaration)

Proses ini merupakan proses untuk mendapatkan informasi variabel pada pernyataan deklarasi *method.* Variabel didapatkan dari argumen pada parameter *method*. Implementasi proses visit(MethodDeclaration) seperti pada gambar 5.5.

Nama Algoritma: visit(MethodDeclaration)
Deskripsi:
Masukan: MethodDeclaration

```
Insialisasi:
List variableDeclarations

Proses:
1. variableDeclarations ← md.parameters()
2. For i=0 to size of variableDeclarations
3.     parameterMethod.add(variableDeclarations.get(i).getName())
4.     tipeMethod ← md.getReturnType()
5.         If arrTemp not null
6.             listArray.add(HimpunanDahan)
7.         End if
8.         arrTemp.add(parameterMethod)
9. End for
10. End
```

Gambar 5.5 Proses visit(MethodDeclaration)

Penjelasan gambar 5.5:

Baris 1-4 : Menyimpan parameter pada pernyataan deklarasi *method*.

Baris 5-7 : Menyimpan informasi variabel yang telah didapatkan pada *method* sebelumnya ke dalam variabel listArray.

Baris 8-10 : Menyimpan informasi variabel *method* pada arrTemp

5.3.1.4 Implementasi Proses visit(VariableDeclarationFragment)

Proses ini merupakan proses untuk mendapatkan informasi variabel pada pernyataan deklarasi variabel beserta informasi variabel rujukan. Implementasi proses visit(VariableDeclarationFragment) seperti pada gambar 5.6.

Nama Algoritma: visit(VariableDeclarationFragment)
Deskripsi:
Masukan: VariableDeclarationFragment

```
Insialisasi:
Type tipeNama

Proses:
1. If node.getParent() instance of Field Declaration
2.     tipeNama ←((FieldDeclaration)node.getParent()).getType()
3. Else If node.getParent() instance of
   VariableDeclarationStatement
4.     tipeNama ← ((VariableDeclarationStatement)
   node.getParent()).getType()
5. End if
6. If arrTemp not null
7.     listArray.add(arrTemp)
8. End if
9. deklarasi.add(nama)
10. arrTemp.add(nama)
11. ekspresiDeklarasi.add(ekspresi)
12. End
```

Gambar 5.6 Proses visit(VariableDeclarationFragment)

Penjelasan gambar 5.6:

Baris 1-5 : Menyimpan nama variabel, ekspresi variabel, dan juga tipe dari variabel yang didapatkan.

Baris 6-8 : Menyimpan informasi variabel yang telah didapatkan pada pernyataan variabel deklarasi sebelumnya ke dalam variabel listArray.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Baris 9-12 : Menyimpan variabel yang didapatkan ke dalam himpunan.

5.3.1.5 Implementasi Proses visit(Assigment)

Proses ini merupakan proses untuk mendapatkan informasi variabel pada suatu pernyataan *assignment.* Variabel didapatkan dari blok kanan dan blok kiri dari suatu *assignment.* Implementasi proses visit(Assigment) seperti pada gambar 5.7.

Nama Algoritma: visit(Assigment)
Deskripsi:
Masukan: Assigment

```
Insialisasi:
-

Proses:
1. If arrTemp not null
2.     listArray.add(arrTemp)
3. End if
4. new List arrTemp
5. arrTemp.add(node.getLeftHandSide())
6. ekspresiRujukan.addnode.getRightHandSide())
7. End
```

Gambar 5.7 Proses visit(Assigment)

Penjelasan gambar 5.7:

Baris 1-3 : Menyimpan variabel pada *assignment* sebelumnya ke dalam listArray.

Baris 5-8 : Menyimpan variabel pada sisi kiri pernyataan ke dalam arrTemp.

Baris 6 : Menyimpan variabel pada sisi kanan pernyataan ke dalam himpunan ekspresiRujukan.

5.3.1.6 Implementasi Proses visit(ReturnStatement)

Proses ini merupakan proses untuk mendapatkan informasi variabel pada suatu pernyataan *return.* Implementasi proses visit(ReturnStatement) seperti pada gambar 5.8.

Nama Algoritma: visit(ReturnStatement)
Deskripsi:
Masukan: ReturnStatement

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

```
Insialisasi:
String stringExprReturn, StringTokenizer st;

Proses:
1. exprReturn ← node.getExpression()
2. stringExprReturn ← node.getExpression()
3. While st.hasMoreToken() do
4.      String token
5.      token ← st.nextToken().trim()
6.      For i ← 0 to size of listArray
7.          If listArray.get(i).get(1) ← token
8.              pernyataanReturn.add(token)
9.          End if
10.     End for
11. End while
12. MethodInvocation returnInvok, Int nLA, boolean
    adaDiListArray, List arrArgs
13. returnInvok ← node.getExpression()
14. pernyataanReturn.add(returnInvok)
15. nLA ← listArray.size
16. For i ← 0 to nLA
17.     For j ← 0 to size of returnInvok.arguments()
18.         If listArray.get(i).get(1) ←
             returnInvok.arguments().get(j)
19.             arrArgs.add(returnInvok)
20.             arrArgs.add(listArray.get(i).get(1))
21.             adaDiListArray ← true
22.         End if
23.     End for
24. End for
25. If adaDiListArray ← false
26.     For j ← 0 to returnInvok.arguments.size
27.         If arrTemp not null && arrTemp ←
            returnInvokAarguments
28.             String dekTemp
29.             listArray.add(arrTemp)
30.             arrTemp ← new List
31.             arrTemp.add(returnInvok)
32.             arrTemp.add(dekTemp)
33.         End if
34.     End for
35. End if
36. End
```

Gambar 5.8 Proses visit(ReturnStatement)

Penjelasan gambar 5.8:

Baris 1-2 : Mendapatkan ekspresi dari pernyataan *return* dan menyimpanya ke dalam variabel.

Baris 3-11 : Memisahkan simbol "+ - * /" dari ekspresi *return* yang berguna untuk mendapatkan variabel, dan jika variabel tersebut telah terdefinisi maka akan disimpan kedalam himpunan *pernyataanReturn.*

Baris 12-24 : Memeriksa apakah pernyataan *return* mengandung pernyataan pemanggilan *method*. Apabila mengandung pernyataan pemanggilan *method* dan variabel pada parameter telah terdefinisi maka variabel tersebut akan disimpan pada arrArgs.

Baris 25-36 : Jika tidak terdapat variabel yang telah terdefinisi maka variabel tersebut akan disimpan pada arrTemp.

5.3.1.7 Implementasi Proses visit(PostfixExpression)

Proses ini merupakan proses untuk mendapatkan informasi variabel pada suatu pernyataan *PostfixExpression.* Implementasi proses visit(PostfixExpression) seperti pada gambar 5.9.

Nama Algoritma: visit(PostfixExpression)
Deskripsi:
Masukan: PostfixExpression

```
Insialisasi:
-

Proses:
1. If node.getOperand() = deklarasi.get(deklarasi.size()-1)
2.      arrTemp.add(deklarasi.get(deklarasi.size()-1)
3. End if
4. End
```

Gambar 5.9 Proses visit(postfixExpression)

Penjelasan gambar 5.9:

Baris 1-4 : Melakukan pengecekan apakah variabel yang didapatkan telah terdefinisi. Apabila telah terdefinisi maka variabel tersebut akan disimpan ke dalam himpunan arrTemp.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

5.3.1.8 Implementasi Proses visit(PrefixExpression)

Proses ini merupakan proses untuk mendapatkan informasi variabel pada suatu pernyataan *PrefixExpression.* Implementasi proses visit(PrefixExpression) seperti pada gambar 5.10.

Nama Algoritma: visit(PrefixExpression)
Deskripsi:
Masukan: PrefixExpression

```
Insialisasi:
-

Proses:
1. If node.getOperand() = deklarasi.get(deklarasi.size()-1)
2.     arrTemp.add(deklarasi.get(deklarasi.size()-1)
3. End if
4. End
```

Gambar 5.10 Proses visit(prefixExpression)

Penjelasan gambar 5.10:

Baris 1-4 : Melakukan pengecekan apakah variabel yang didapatkan telah terdefinisi. Apabila telah terdefinisi maka variabel tersebut akan disimpan ke dalam himpunan arrTemp.

5.3.1.9 Implementasi Proses visit(IfStatement)

Proses ini merupakan proses untuk mendapatkan informasi variabel pada suatu pernyataan *IfStatement.* Implementasi proses visit(IfStatement) seperti pada gambar 5.11.

Nama Algoritma: visit(IfStatement)
Deskripsi:
Masukan: IfStatement

```
Insialisasi:
StringTokenizer st

Proses:
1. While st.hasMoreToken()
2.     List pernyataanIF
3.     token ← st.nextToken()
4.     For i ← 0 to listArray.size
5.         If listArray.get(i).get(1) ← token
6.             pernyataanIF.add(node.getExpression())
```


repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

```
7.              pernyataanIF.add(token)
8.              listArray.add(pernyataanIF)
9.          End if
10.         Else if arrTemp.get(i) ← token
11.             pernyataanIF.add(node.getExpression())
12.             pernyataanIF.add(token)
13.             listArray.add(pernyataanIF)
14.             arrTemp = new List
15.             arrTemp.add(token)
16.         End if
17.     End for
18. End while
19. End
```

Gambar 5.11 Proses visit(IfStatement)

Penjelasan gambar 5.11:

Baris 1-3 : Memecah variabel yang terdapat pada ekspresi dari pernyataan *if* .

Baris 4-9 : Menyimpan variabel yang terdapat pada pernyataan *If* apabila variabel tersebut telah terdefinisi dan menyimpannya pada listArray.

Baris 10-17 : Menyimpan variabel yang terdapat pada pernyataan *If* pada arrTemp.

5.3.1.10 Implementasi Proses visit(SimpleName)

Proses ini merupakan proses untuk mendapatkan informasi variabel pada saat suatu variabel digunakan. Implementasi proses visit(SimpleName) seperti pada gambar 5.12.

Nama Algoritma: visit(SimpleName)
Deskripsi:
Masukan: SimpleName

```
Insialisasi:
String nodeRujukan

Proses:
1. If ekspresiRujukan not null && ekspresiRujukan.get(last) ←
   node.getIdentifier()
2.      nodeRujukan ← node
3. End if
```

```
4. If ekspresiDeklarasi not null && ekspresiDeklarasi.get(last)
   ← node.getIdentifier()
5.     nodeDeklarasi ← node
6. End if
7. If nodeRujukan != nodeDeklarasi && nodeRujukan != "default"
8.     If nodeDeklarasi ← nodeRujukan
9.         refe.add(nodeDeklarasi)
10.    End if
11.    Else if nodeDeklarasi not default
12.        refe.add(nodeDeklarasi)
13.    End if
14     Else ff nodeRujukan not default
15.        refe.add(nodeRujukan)
16.    End if
17. End if
18. If nodeDeklarasi != default && nodeRujukan != default &&
    refe != null
19.    arrTemp.add(refe.get(lastIndex))
20. End if
21. End
```

Gambar 5.12 Proses visit(SimpleName)

Penjelasan gambar 5.12:

Baris 1-3 : Melakukan pengecekan apakah variabel ekspresiRujukan tidak kosong dan terdapat variabel yang sama dengan variabel yang didapatkan, lalu variabel tersebut akan ditampung pada variabel nodeRujukan.

Baris 4-6 : Melakukan pengecekan apakah variabel ekspresiDeklarasi tidak kosong dan terdapat variabel yang sama dengan variabel yang didapatkan, lalu variabel tersebut akan ditampung pada variabel nodeDeklarasi.

Baris 7-17 : Jika tidak terdapat variabel rujukan, maka hanya variabel deklarasi saja yang disimpan pada variabel refe.

Baris 18-20 : Jika variabel deklarasi dan variabel rujukan tidak kosong maka himpunan refe akan ditambahkan ke arrTemp.

5.3.2 Implementasi Proses AST_Model

Proses ini merupakan proses untuk melakukan normalisasi data variabel dan menghubungkan suatu variabel dengan variabel rujukannya .

Implementasi proses AST_Model seperti pada gambar 5.13.

<table>
<tr><td>Nama Algoritma: AST_Model
Deskripsi:
Masukan: -</td></tr>
<tr><td>

```
Insialisasi:
List deklarasi, listReference, listRelasiRef, listArray,
AST_Ekstraktor objek

Proses:
1. listArray ← objek.getListArray
2. call : normalisasi
3. call : buatTabelReferensi
4. call : buatRelasiTabelReferensi
5. End
```

</td></tr>
</table>

Gambar 5.13 Proses AST_Model

Penjelasan gambar 5.13:

Baris 1 : Mendapatkan himpunan listArray pada kelas AST_Ekstraktor.

Baris 2 : Menjalankan *method* normalisasi.

Baris 3 : Menjalankan *method* buatTabelReferensi

Baris 4 : Menjalankan *method* buatTabelRelasi.

5.3.2.1 Implementasi Proses Normalisasi Data

Proses ini merupakan proses untuk melakukan normalisasi data variabel dengan cara menghapus variabel yang sama pada variabel rujukan. Implementasi proses normalisasi data seperti pada gambar 5.14.

<table>
<tr><td>Nama Algoritma: Normalisasi Data
Deskripsi:
Masukan: -</td></tr>
<tr><td>

```
Insialisasi:
-

Proses:
1. For i ← 0 to himpunan pohon (i)
2.      new List sementara
3.      sementara ← listArray.get(i)
4.      If Sementara.size > 3
5.           For j ← 2 to sementara.size
6.                For k = j+1 to sementara.size
7.                     If sementara (j) ← sementara (k)
8.                          sementara.remove(k)
```

</td></tr>
</table>

```
9.                        k--
10.                   End if
11.               End for
12.           End for
13.       End if
14. End for
15. End
```

Gambar 5.14 Proses Normalisasi Data

Penjelasan gambar 5.14:

Baris 1-3 : Mengisi himpunan sementara dengan nilai pada himpunan listArray pada index i.

Baris 4-15 : Apabila pada himpunan sementara terdapat variabel rujukan dan terdapat elemen pada himpunan sementara yang sama, maka salah satu elemen tersebut akan dihapus.

5.3.2.2 Implementasi Proses Buat Tabel Referensi

Proses ini merupakan proses untuk mendapatkan variabel referensi dari suatu himpunan variabel. Implementasi proses buat tabel referensi seperti pada gambar 5.15.

Nama Algoritma: Buat Tabel Referensi
Deskripsi:
Masukan: -

```
Insialisasi:
-

Proses:
1. For i ← 0 to listArray.size
2.     deklarasi.add(listArray.get(i).get(1)
3.     new List referensi
4.     For k ← 1 to listArray.size
5.         If k > 1
6.             referensi.add(listArray.get(j).get(k)
7.         End if
8.     End for
9.     listReference.add(referensi)
10. End for
11. End
```

Gambar 5.15 Proses Buat Tabel Referensi

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Penjelasan gambar 5.15:

Baris 1-2 : Mengisi himpunan deklarasi dengan himpunan listArray pada baris i dan kolom ke 1.

Baris 4-8 : Mendapatkan data variabel rujukan dari himpunan listArray.

Baris 9 : Menambahkan himpunan referensi kedalam himpunan listReference.

5.3.2.3 Implementasi Proses Buat Relasi Tabel

Proses ini merupakan proses untuk mendapatkan index dari variabel referensi dari suatu himpunan variabel. Implementasi proses buat relasi tabel seperti pada gambar 5.16.

Nama Algoritma: Buat Relasi Tabel
Deskripsi:
Masukan: -

```
Insialisasi:
-

Proses:
1. For i ← 0 to listReference.size
2.     new List tempDeklarasi
3.     For j ← 0 to listReference.size
4.         For k ← 0 to i
5.             If listReference.get(i).get(j) ←
   deklarasi.get(k)
6.                 tempDeklarasi.add(k)
7.             End if
8.         End for
9.         For k ← 0 to tempDeklarasi
10.            If tempDeklarasi.size() > 1 &&
   tempDeklarasi.get(k) ← tempDeklarasi.get(k+1)
11.                tempDeklarasi.remove(k+1)
12.            End if
13.        End for
14.    End for
15.    listRelasiRef.hapusRedudansi(tempDeklarasi)
16. End for
17. End
```

Gambar 5.16 Proses Buat Relasi Tabel

Penjelasan gambar 5.16:

Baris 1-8 : Mendapatkan index dari variabel rujukan yang sama dengan variabel deklarasi, dan disimpan pada tempDeklarasi.

Baris 9-13 : Melakukan pengecekan pada himpunan tempDeklarasi. Apabila terdapat data yang sama maka data tersebut akan dihapus.

Baris 14 : Menjalankan *method* hapusRedudansi dengan parameter tempDeklarasi.

5.3.2.4 Implementasi Proses Hapus Redudansi

Proses ini merupakan proses untuk menghapus index dari variabel rujukan yang sama. Implementasi proses hapus redudansi seperti pada gambar 5.17.

Nama Algoritma: Hapus Redudansi
Deskripsi:
Masukan: List sementara

```
Insialisasi:
List listArray

Proses:
1. listArray ← objek.getListArray
2. If sementara.size > 1
3.     For i ← 1 to sementara.size
4.         For j ← 0 to 1
5.             If listArray.get(sementara.get(i).get(1) ←
     listArray.get(sementara.get(j).get(1)
6.                 sementara.remove(j)
7.                 i--
8.                 j--
7.             End If
8.         End For
9.     End For
10. End If
11. End
```

Gambar 5.17 Proses Hapus Redudansi

Penjelasan gambar 5.17:

Baris 1 : Deklarasi himpunan listArray yang mempunyai nilai yang sama pada himpunan listArray pada kelas AST_Ekstraktor.

Baris 2-9 : Jika data index relasi pada himpunan sementara sama dengan index relasi pada himpunan listArray, maka data tersebut aan dihapus.

5.3.3 Implementasi Proses Buat Grafik

Proses ini merupakan proses untuk membuat grafik yang diawali dengan mendefinisikan algoritma tata letak grafik dan juga elemen pembangun grafik. Implementasi proses buat grafik seperti pada gambar 5.18.

Nama Algoritma: Hapus Redudansi
Deskripsi:
Masukan: ASTModel AST

```
Insialisasi:
DirectedGraph graph, VisualizationViewer vv, int algoritma

Proses:
1. listArray ← objek.getListArray
2. If sementara.size > 1
3.     For i ← 1 to sementara.size
4.         For j ← 0 to 1
5.             If listArray.get(sementara.get(i).get(1) ←
               listArray.get(sementara.get(j).get(1)
6.                 sementara.remove(j)
7.                 i--
8.                 j--
7.             End if
8.         End for
9.     End for
10. End if
11. End
```

Gambar 5.18 Proses Buat Grafik

Penjelasan gambar 5.18:

Baris 1-9 : Melakukan pengecekan pada algoritma yang telah dipilih oleh pengguna, lalu mendefinisikan algoritma yang dipilih pada object visualizationViewer.

Baris 10 : Memanggil *method* createTree untuk menyusun *node-node* yang telah terbuat

5.3.3.1 Implementasi Proses Create Tree

Proses ini merupakan proses untuk membuat *node* dan garis penghubung yang menghubungkan variabel deklarasi dan variabel rujukannya. Implementasi proses create tree seperti pada gambar 5.19.

Nama Algoritma: Create Tree
Deskripsi:
Masukan: ASTModel pdg

```
Insialisasi:
DirectedGraph graph, VisualizationViewer vv, int algoritma

Proses:
1. For i ← 0 to size of arrNode
2.     If relasi not null
3.         arrNode ← pdg.getDeklarasi
4.         addVertex(arrNode)
5.         For j ← pdg.listRelasi.size to 0
6.             addEdge(arrNode(j),arrNode(i))
7.         End for
8.     End if
9.     Else
10.        arrNode[i] ← pdg.getDeklarasi
11.        addVertex(arrNode)
12.    End else
13. End for
14. End
```

Gambar 5.19 Proses Create Tree

Penjelasan gambar 5.19:

Baris 1-8 : Jika himpunan listRelasi mempunyai rujukan maka dilakukan pembuatan vertex dan juga pembuatan garis relasi menuju *node* rujukan

Baris 9-12 : Jika himpunan listRelasi tidak mempunyai nilai rujukan maka vertex akan dibuat dengan nama variabel deklarasi.

5.4 Implementasi Antarmuka

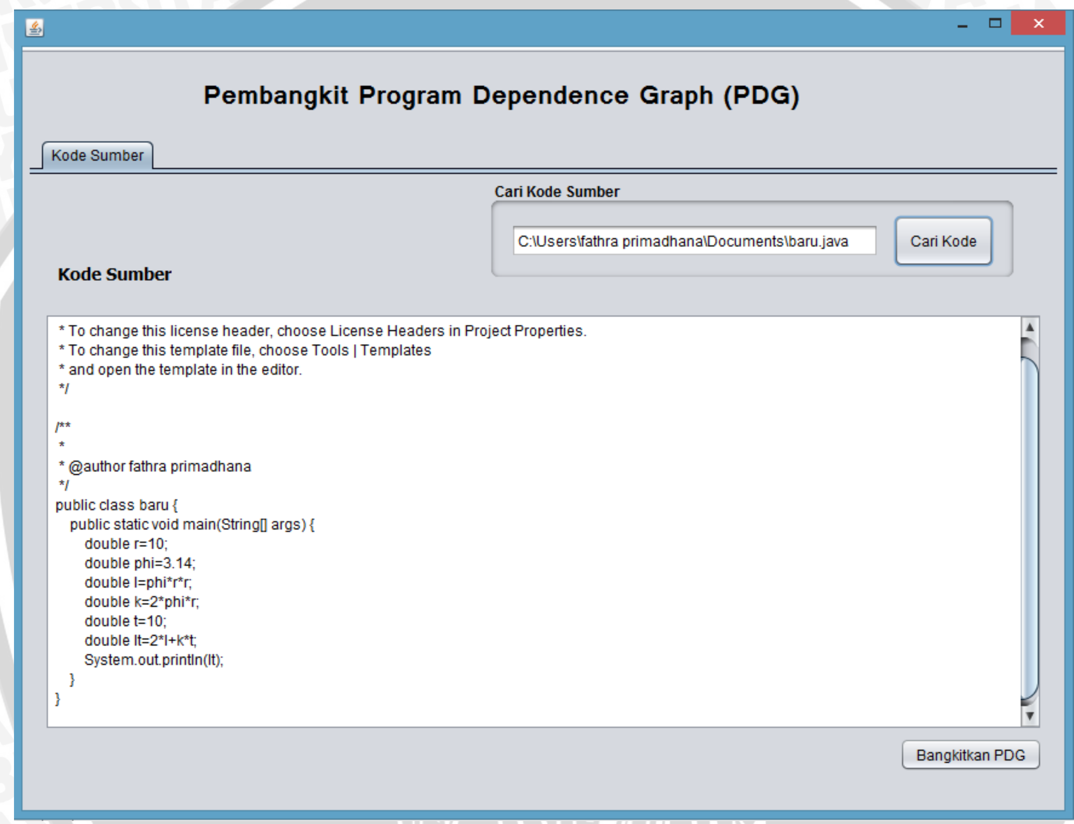
Implementasi antarmuka merupakan hasil pembuatan user interface dari hasil perancangan antarmuka yang bertujuan agar pengguna dapat dengan mudah berkomunikasi dengan sistem. Implementasi antarmuka terdiri dari tiga halaman

repository.ub.ac.id

yaitu halaman utama, halaman *Program Dependence Graph*, dan halaman tabel ketergantungan.

5.4.1 Implementasi Halaman Utama

Antarmuka utama digunakan sebagai antarmuka kepada pengguna untuk melakukan masukan data berupa *source code* bertipe Java. Selain itu halaman utama juga dapat menampilkan *source code* yang telah dimasukkan oleh pengguna.



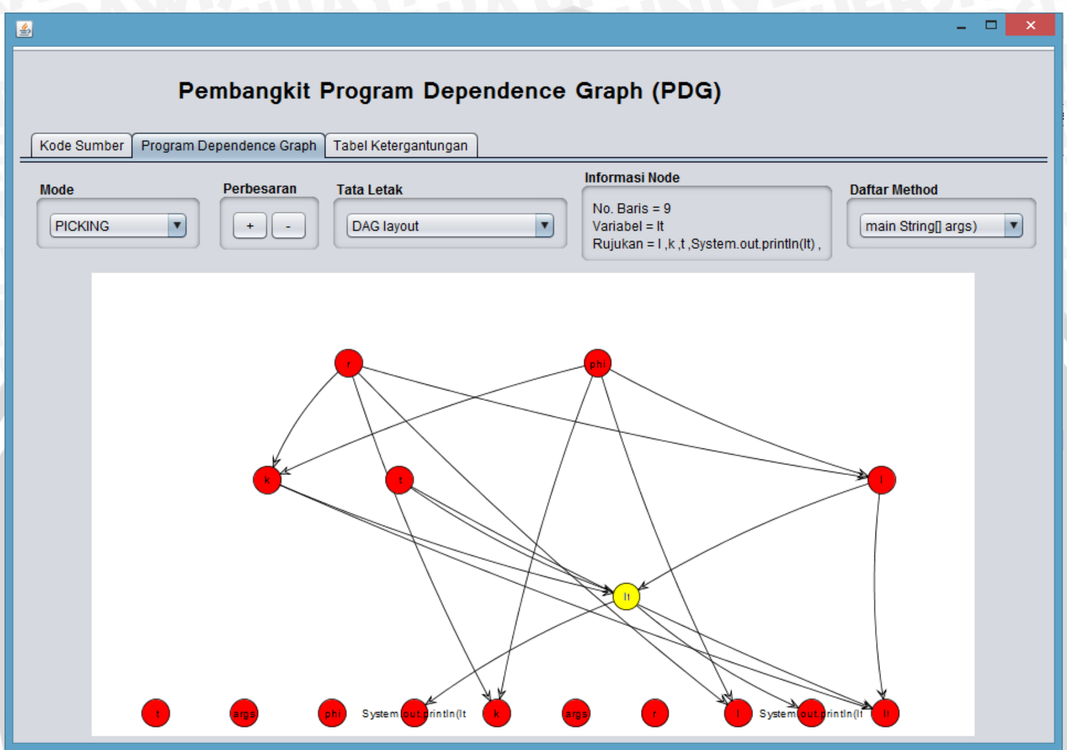
Gambar 5.20 Antarmuka Utama

Gambar 5.20 menunjukkan antarmuka utama yang digunakan oleh pengguna untuk melakukan masukan data. Pengguna dapat menekan tombol “Cari Berkas” untuk mencari berkas *source code* yang bertipe Java sebagai masukan ke dalam sistem. Kemudian sistem akan menampilkan isi *source code* yang telah dipilih pengguna. Setelah itu pengguna dapat menekan tombol “Bangkitkan PDG” untuk membangkitkan *Program Dependence Graph* dari hasil berkas *source code* masukan.

UNIVERSITAS BRAWIJAYA

5.4.2 Implementasi Halaman *Program Dependence Graph*

Antarmuka *Program Dependence Graph* digunakan sebagai antarmuka kepada pengguna untuk menampilkan hasil pembangkitan *Program Dependence Graph* dari hasil berkas *source code* masukan.

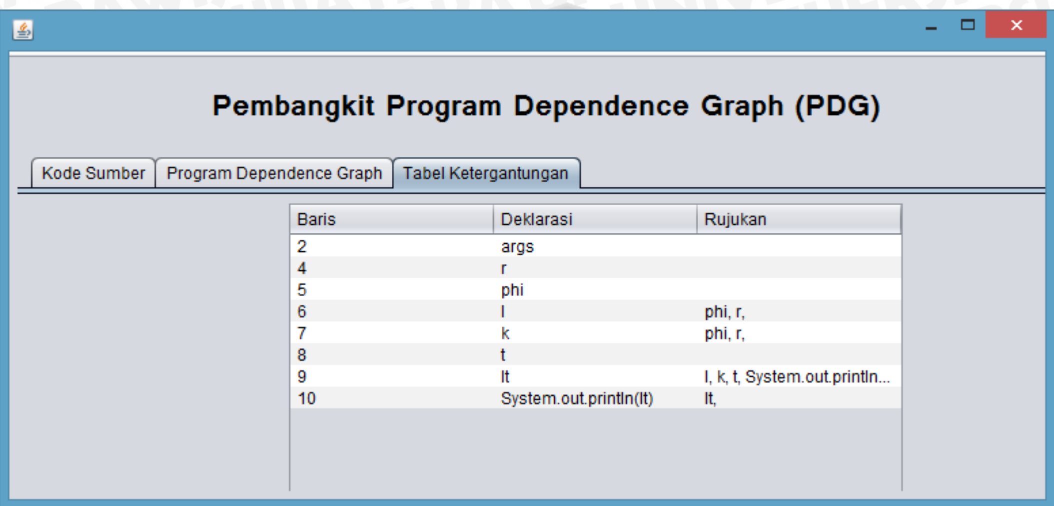


Gambar 5.21 Antarmuka *Program Dependence Graph*

Gambar 5.21 menunjukkan antarmuka *Program Dependence Graph* yang digunakan oleh pengguna untuk melihat hasil pembangkitan *Program Dependence Graph*. Halaman ini menyediakan beberapa fitur kepada pengguna diantaranya fitur *mode*, perbesaran, tata letak, informasi *node*, dan ubah *Program Dependence Graph* berdasarkan *method* yang dipilih. Fitur *mode* digunakan untuk memilih *mode* operasi *Program Dependence Graph*. Fitur perbesaran digunakan untuk mengubah ukuran grafik menjadi lebih besar (*zoom in*) atau menjadi lebih kecil (*zoom out*). Fitur tata letak digunakan untuk mengubah *layout* grafik seperti *layout Kanada Kawai Layout*, *DAG Layout*, dan *Fruchterman-Reingold Layout*. Fitur informasi *node* digunakan untuk melihat informasi *node* yang dipilih pengguna. Fitur ubah *Program Dependence Graph* digunakan untuk menampilkan *Program Dependence Graph* sesuai dengan *method* yang dipilih oleh pengguna.

5.4.3 Implementasi Halaman Tabel Ketergantungan

Antarmuka tabel ketergantungan digunakan sebagai antarmuka kepada pengguna untuk melihat tabel ketergantungan antar variabel dari hasil *Program Dependence Graph*.



Gambar 5.22 Antarmuka Tabel Ketergantungan

Gambar 5.22 menunjukkan antarmuka tabel ketergantungan yang digunakan oleh pengguna untuk melihat ketergantungan antar variabel deklarasi dan variabel rujukan dari hasil pembangkitan *Program Dependence Graph*.

BAB VI
PENGUJIAN

Dalam penelitian ini, pengujian yang dilakukan berupa pengujian unit, pengujian validasi, dan pengujian komponen. Metode yang digunakan untuk melakukan pengujian adalah metode *white-box* dan *black-box*. Untuk pengujian unit akan menggunakan metode *white-box*, sementara pengujian validasi dan pengujian akurasi akan menggunakan metode *black-box*. Pengujian dilakukan berdasarkan *use case* yang telah dirancang pada bab sebelumnya.

6.1 Pengujian Unit

Pengujian unit merupakan proses pengujian yang dilakukan dengan menggunakan metode *basis path testing* untuk memodelkan algoritma pada suatu *flow graph*, menentukan jumlah *cyclomatic complexity*, menentukan jalur independen, dan menentukan kasus uji berdasarkan jalur independen yang telah diperoleh.

6.1.1 Pengujian Unit Ekstraksi Data

Berikut ini merupakan algoritma untuk melakukan proses ekstraksi data yang dijelaskan pada gambar 6.1.

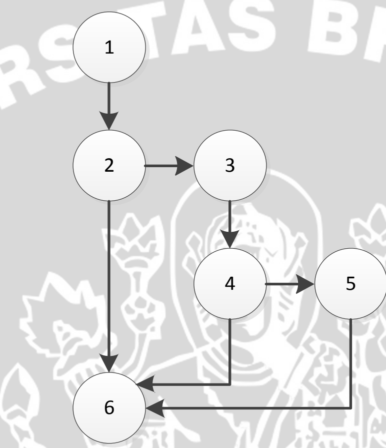
Nama Algoritma: Ekstraksi Data
Deskripsi:
- Masukan: *Source code*
- Proses

1	Inisialisasi : String alamat, ASTParser parser, Compilation Unit cu, List parameterMethod, deklarasi, ekspresiDeklarasi, ekspresiRujukan, listArray, arrTemp, Boolean parseAble parser = new parser parser.setSource(str.tocharArray) parser.setKind(K_Compilation_Unit) cu = parser.createAST Call : PemilihanMethod()
2	Call : ProsesAwal(0)
3	If error occur String isiPerbaikan(“class PDG”+str)

	parser.setSource(isiPerbaikan) Parser.setKind(K_Compilation_Unit) cu = parser.createAST Call : PemilihanMethod()
4	Call : ProsesAwal()
5	If error occur
6	parseAble = false
	End

Gambar 6.1 Pengujian Unit Ekstraksi Data

Berdasarkan algoritma yang telah diperoleh seperti pada gambar 6.1 maka diperoleh *flow graph* seperti pada gambar 6.2.



Gambar 6.2 *Flow Graph* Ekstraksi Data

Berdasarkan *flow graph* yang diperoleh seperti pada gambar 6.2 maka diperoleh jumlah *cyclomatic complexity* melalui persamaan 2.1. Berikut ini merupakan perhitungan dari jumlah *cyclomatic complexity*:

$$M = E - N + 2P$$
$$= 7 - 6 + 2$$
$$= 3$$

Berdasarkan jumlah *cyclomatic complexity* yang telah didapatkan, maka akan ditentukan tiga jalur independen, yaitu:

Jalur 1: 1 – 2 – 6

Jalur 2: 1 – 2 – 3 – 4 – 6

Jalur 3: 1 – 2 – 3 – 4 – 5 – 6

Berdasarkan jalur independen yang ditentukan, maka dapat diperoleh kasus uji yang dijelaskan pada tabel 6.1.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Tabel 6.1 Kasus Uji Ekstraksi Data

Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	Memasukkan *source code* yang benar.	Sistem akan menampilkan pesan *Program Dependende Graph* telah terbuat.	Sistem menampilkan pesan *Program Dependende Graph* telah terbuat.
2	Memasukkan *source code* tanpa nama kelas.	Sistem akan menambahkan nama kelas pada *source code*.	Sistem menambahkan nama kelas pada *source code*.
3	Memasukkan *source code* yang salah.	Sistem akan menampilkan pesan kesalahan.	Sistem menampilkan pesan kesalahan.

6.1.2 Pengujian Unit Buat Grafik

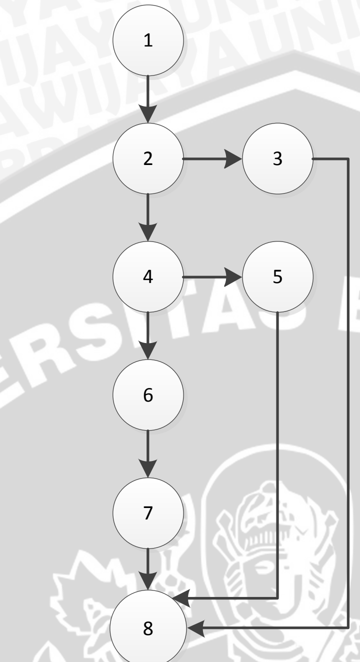
Berikut ini merupakan algoritma untuk melakukan proses buat grafik yang dijelaskan pada gambar 6.3.

Nama Algoritma: Buat Grafik
Deskripsi:
- Masukan: Skema pohon
- Proses

```
1   Inisialisasi : DirectedGraph graph, VisualizationViewer
    vv, int algoritma
2   If algo = 1
3   VV.setlayout = KKLayout
4   If algo = 2
5   VV.setlayout = DAGLayout
6   If algo = 3
7   VV.setlayout = FRLayout
8   Call : createTree
    End
```

Gambar 6.3 Pengujian Unit Buat Grafik

Berdasarkan algoritma yang telah diperoleh seperti pada gambar 6.3 maka diperoleh *flow graph* seperti pada gambar 6.4.



Gambar 6.4 *Flow Graph* Buat Grafik

Berdasarkan *flow graph* yang diperoleh seperti pada gambar 6.4 maka diperoleh jumlah *cyclomatic complexity* melalui persamaan 2.1. Berikut ini merupakan perhitungan dari jumlah *cyclomatic complexity*:

M = E – N + 2P

= 9 – 8 + 2

= 3

Berdasarkan jumlah *cyclomatic complexity* yang telah didapatkan, maka akan ditentukan tiga jalur independen, yaitu:

Jalur 1: 1 – 2 – 4 – 7 – 8

Jalur 2: 1 – 2 – 3 – 8

Jalur 3: 1 – 2 – 4 – 5 – 8

Berdasarkan jalur independen yang ditentukan, maka dapat diperoleh kasus uji yang dijelaskan pada tabel 6.2.

Tabel 6.2 Kasus Uji Buat Grafik

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	Pengguna memilih algoritma tata letak KKLayout.	Tampilan grafik akan berubah sesuai dengan algoritma KKLayout.	Tampilan grafik berubah sesuai dengan algoritma KKLayout.
2	Pengguna memilih algoritma tata letak DAGLayout.	Tampilan grafik akan berubah sesuai dengan algoritma DAGLayout.	Tampilan grafik berubah sesuai dengan algoritma DAGLayout.
3	Pengguna memilih algoritma tata letak FRLayout.	Tampilan grafik akan berubah sesuai dengan algoritma FRLayout.	Tampilan grafik berubah sesuai dengan algoritma FRLayout.

6.1.3 Pengujian Unit Create Tree

Berikut ini merupakan algoritma untuk melakukan proses create tree yang dijelaskan pada gambar 6.5.

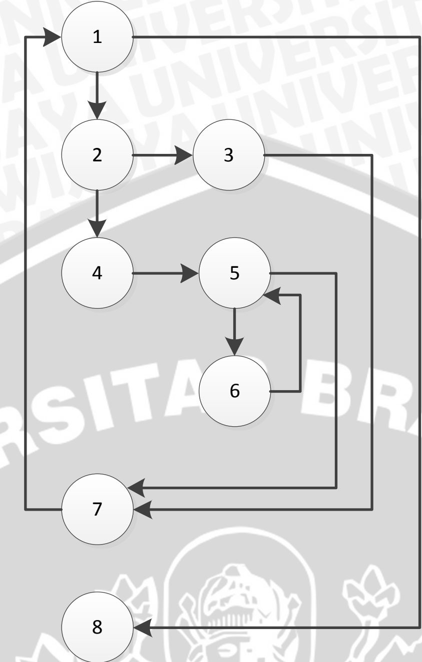
Nama Algoritma: Create Tree
Deskripsi:
- Masukan: Skema pohon
- Proses

```
  Inisialisasi : MyNode arrNode = new MyNode
1 For i=0 to size of arrNode
2 If relasi not null
3 arrNode[i] = pdg.getDeklarasi
  Call : addVertex(arrNode)
4 arrNode = pdg.getDeklarasi
  Call : addVertex(arrNode)
5 For j = pdg.listRelasi.size to 0
6 addEdge dari node ke node rujukan
7 End for
8 End
```

Gambar 6.5 Pengujian Unit Create Tree

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Berdasarkan algoritma yang telah diperoleh seperti pada gambar 6.5 maka diperoleh *flow graph* seperti pada gambar 6.6.



Gambar 6.6 *Flow Graph* Create Tree

Berdasarkan *flow graph* yang diperoleh seperti pada gambar 6.6 maka diperoleh jumlah *cyclomatic complexity* melalui persamaan 2.1. Berikut ini merupakan perhitungan dari jumlah *cyclomatic complexity*:

$$M = E - N + 2P$$
$$= 10 - 8 + 2$$
$$= 4$$

Berdasarkan jumlah *cyclomatic complexity* yang telah didapatkan, maka akan ditentukan empat jalur independen, yaitu:

Jalur 1: 1 – 8

Jalur 2: 1 – 2 – 3 – 7 – 1 – 8

Jalur 3: 1 – 2 – 4 – 5 – 7 – 1 – 8

Jalur 4: 1 – 2 – 4 – 5 – 6 – 7 – 1 – 8

Berdasarkan jalur independen yang ditentukan, maka dapat diperoleh kasus uji yang dijelaskan pada tabel 6.3.

Tabel 6.3 Kasus Uji Create Tree

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Jalur	Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan
1	Memasukkan *source code* tanpa variabel.	Tidak akan ada *node* yang tercipta.	Tidak ada *node* yang tercipta.
2	Memasukkan pernyataan dengan deklarasi variabel tanpa merujuk ke variabel lain.	Satu *node* akan tercipta.	Satu *node* tercipta.
3	Memasukkan pernyataan deklarasi variabel beserta rujukan.	Satu *node* akan tercipta.	Satu *node* tercipta.
4	Memasukkan pernyataan deklarasi variabel beserta variabel rujukan yang telah terdefinisi.	Satu *node* akan tercipta beserta garis yang merujuk pada *node* lain.	Satu *node* tercipta beserta garis yang merujuk pada *node* lain.

6.2 Pengujian Validasi

Pengujian validasi merupakan proses menguji seluruh skenario *use case* yang telah dibuat pada bab sebelumnya.

6.2.1 Kasus Uji Pengujian Validasi

Berikut ini merupakan kasus uji untuk melakukan pengujian validasi pada Kakas Bantu Pembangkitan *Program Dependence Graph*.

6.2.1.1 Kasus Uji Cari Berkas

Berikut ini merupakan kasus uji untuk cari berkas yang dijelaskan pada tabel 6.4 dan kasus uji cari berkas alur alternatif 1 ditunjukkan pada tabel 6.5.

Tabel 6.4 Kasus Uji Cari berkas

Nama Kasus Uji	Kasus uji Cari Berkas
Objek Uji	Kebutuhan Fungsionalitas (SRS_001)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa sistem dapat melakukan proses “cari berkas”
Data Masukan	Alamat berkas
Prosedur Uji	1. Membuka sistem dan menekan tombol ‘cari berkas’. 2. Mencari dan memilih berkas bertipe Java.
Hasil yang Diharapkan	Informasi alamat telah didapatkan serta isi berkas ditampilkan oleh sistem

Tabel 6.5 Kasus Uji Cari berkas Alur Alternatif 1

Nama Kasus Uji	Kasus uji Cari Berkas
Objek Uji	Kebutuhan Fungsionalitas (SRS_001)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa sistem dapat menangani pemilihan berkas yang salah.
Data Masukan	Berkas bukan bertipe Java
Prosedur Uji	1. Membuka sistem dan menekan tombol ‘cari berkas’. 2. Mencari dan memilih berkas bukan bertipe Java.
Hasil yang Diharapkan	Sistem akan memberikan pesan kesalahan

6.2.1.2 Kasus Uji Bangkitkan *Program Dependence Graph*

Berikut ini merupakan kasus uji untuk membangkitkan *Program Dependence Graph* yang dijelaskan pada tabel 6.6.

Tabel 6.6 Kasus Uji Bangkitkan *Program Dependence Graph*

Nama Kasus Uji	Kasus Uji Bangkitkan *Program Dependence Graph*
Objek Uji	Kebutuhan Fungsionalitas (SRS_002)

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa sistem dapat melakukan proses pembangkitan *Program Dependence Graph*.
Data Masukan	Isi berkas
Prosedur Uji	1. Membuka sistem dan menekan tombol 'cari berkas' 2. Mencari dan memilih berkas bukan bertipe Java. 3. Pengguna menekan tombol bangkitkan PDG
Hasil yang Diharapkan	Sistem dapat menampilkan *Program Dependence Graph* yang telah terbuat.

6.2.1.3 Kasus Uji Lihat Informasi *Node*

Berikut ini merupakan kasus uji untuk lihat informasi *node* yang dijelaskan pada tabel 6.7.

Tabel 6.7 Kasus Uji Lihat Informasi *Node*

Nama Kasus Uji	Kasus Uji Melihat Informasi *Node*
Objek Uji	Kebutuhan Fungsionalitas (SRS_003)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa sistem dapat menampilkan informasi *node*.
Data Masukan	Pemilihan *node*
Prosedur Uji	1. Membuka sistem dan menekan tombol 'cari berkas' 2. Mencari dan memilih berkas bukan bertipe java 3. Pengguna menekan tombol bangkitkan PDG 4. Pengguna memilih salah satu *node*
Hasil yang Diharapkan	Sistem dapat menampilkan informasi *node* yang dipilih.

6.2.1.4 Kasus Uji Ubah *Layout* Grafik

Berikut ini merupakan kasus uji untuk mengubah *layout* grafik *Program Dependence Graph* yang dijelaskan pada tabel 6.8.

Tabel 6.8 Kasus Uji Ubah *Layout* Grafik

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Nama Kasus Uji	Kasus Uji Ubah *Layout* Grafik
Objek Uji	Kebutuhan Fungsionalitas (SRS_004)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa sistem dapat mengubah layout grafik.
Data Masukan	Tipe layout grafik
Prosedur Uji	1. Membuka sistem dan menekan tombol 'cari berkas' 2. Mencari dan memilih berkas bukan bertipe java 3. Pengguna menekan tombol bangkitkan PDG 4. Pengguna memilih salah satu tipe layout grafik
Hasil yang Diharapkan	Sistem dapat menampilkan grafik sesuai layout yang dipilih oleh pengguna.

6.2.1.5 Kasus Uji Ubah Ukuran Grafik

Berikut ini merupakan kasus uji untuk mengubah ukuran grafik yang dijelaskan pada tabel 6.9.

Tabel 6.9 Kasus Uji Ubah Ukuran Grafik

Nama Kasus Uji	Kasus Uji Ubah Ukuran Grafik
Objek Uji	Kebutuhan Fungsionalitas (SRS_005)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa sistem dapat mengubah ukuran grafik.
Data Masukan	Pegubahan ukuran grafik
Prosedur Uji	1. Membuka sistem dan menekan tombol 'cari berkas' 2. Mencari dan memilih berkas bukan bertipe java 3. Pengguna menekan tombol bangkitkan PDG 4. Pengguna memilih salah satu dari tombol "+" atau "-"
Hasil yang Diharapkan	Sistem dapat memperkecil atau memperbesar grafik sesuai masukan dari pengguna.

6.2.1.6 Kasus Uji Ubah *Program Dependence Graph*

Berikut ini merupakan kasus uji untuk memilih *method* yang dijelaskan pada tabel 6.10.

Tabel 6.10 Kasus Uji Ubah *Program Dependence Graph*

Nama Kasus Uji	Kasus Uji Ubah *Program Dependence Graph*
Objek Uji	Kebutuhan Fungsionalitas (SRS_006)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa sistem dapat mengubah informasi grafik jika pengguna memilih *method* yang berbeda.
Data Masukan	Pilihan *method*
Prosedur Uji	1. Membuka sistem dan menekan tombol 'cari berkas' 2. Mencari dan memilih berkas bukan bertipe java 3. Pengguna menekan tombol bangkitkan PDG 4. Pengguna mengganti *method*
Hasil yang Diharapkan	Sistem akan menggambar ulang grafik sesuai dengan informasi method yang dipilih.

6.2.1.7 Kasus Uji Lihat Tabel Ketergantungan

Berikut ini merupakan kasus uji untuk melihat tabel ketergantungan yang dijelaskan pada tabel 6.11.

Tabel 6.11 Kasus Uji Lihat Tabel Ketergantungn

Nama Kasus Uji	Kasus Uji Lihat Tabel Ketergantungan
Objek Uji	Kebutuhan Fungsionalitas (SRS_007)
Tujuan Pengujian	Pengujian dilakukan untuk memastikan bahwa sistem dapat menampilkan informasi tabel ketergantungan.
Data Masukan	-
Prosedur Uji	1. Membuka sistem dan menekan tombol 'cari berkas' 2. Mencari dan memilih berkas bukan bertipe java

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

	3. Pengguna menekan tombol bangkitkan PDG 4. Pengguna memilih tab "Tabel Ketergantungan"
Hasil yang Diharapkan	Sistem akan menampilkan informasi tabel ketergantungan.

6.2.2 Hasil dan Analisis Pengujian Validasi

Berikut ini merupakan hasil pengujian validasi pada Kakas Bantu Pembangkitan *Program Dependence Graph.*

Tabel 6.12 Hasil Pengujian Validasi

No	Nama Kasus Uji	Hasil yang Diharapkan	Hasil yang Didapatkan	Validitas
1	Kasus Uji Cari Berkas	Informasi alamat akan didapatkan serta isi berkas ditampilkan oleh sistem	Informasi alamat telah didapatkan serta isi berkas ditampilkan oleh sistem	Valid
2	Kasus Uji Cari Berkas Alur Alternatif 1	Sistem akan memberikan pesan kesalahan	Sistem memberikan pesan kesalahan	Valid
3	Kasus Uji Bangkitkan *Program Dependence Graph*	Sistem akan menampilkan *Program Dependence Graph* yang telah terbuat.	Sistem dapat menampilkan *Program Dependence Graph* yang telah terbuat.	Valid
4	Kasus uji Lihat Informasi *Node*	Sistem akan menampilkan informasi *node* yang dipilih.	Sistem dapat menampilkan informasi *node* yang dipilih.	Valid
5	Kasus Uji Ubah *Layout* Grafik	Sistem akan mengubah tata letak *Program*	Sistem dapat mengubah tata letak *Program*	Valid

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

		Dependence Graph sesuai masukan pengguna.	*Dependence Graph* sesuai masukan pengguna.	
6	Kasus Uji Ubah Ukuran Grafik	Sistem akan memperkecil atau memperbesar grafik sesuai masukan dari pengguna.	Sistem dapat memperkecil atau memperbesar grafik sesuai masukan dari pengguna.	Valid
7	Kasus Uji Ubah *Program Dependence Graph*	Sistem akan mengubah tabel ketergantungan dan *Program Dependence Graph* sesuai dengan *method* yang dipilih.	Sistem dapat mengubah tabel ketergantungan dan *Program Dependence Graph* sesuai dengan *method* yang dipilih.	Valid
8	Kasus Uji Lihat Tabel Ketergantungan	Sistem akan menampilkan tabel ketergantungan.	Sistem dapat menampilkan tabel ketergantungan.	Valid

Dari hasil pengujian validasi diperoleh hasil fungsi dalam sistem kakas bantu pembangkitan *Program Dependence Graph* sudah valid 100% sesuai dengan hasil yang diharapkan.

6.3 Pengujian Akurasi

Pengujian akurasi merupakan proses pengujian yang dilakukan dengan membandingkan hasil pembuatan *Program Dependence Graph* secara manual dengan hasil pembuatan *Program Dependence Graph* secara otomatis. Pembuatan

Program Dependence Graph secara otomatis dilakukan dengan menggunakan dua sistem yaitu sistem pada penelitian ini dan pustaka pembangkitan *Program Dependence Graph* berdasarkan Java *bytecode* yaitu *Java System Dependence Graph* API [TON-10].

6.3.1 Proses Awal Eksekusi Pustaka *Java System Dependence Graph* API

Pustaka *Java System Dependence Graph* API merupakan pustaka yang dapat digunakan untuk megetahui ketergantungan antar variabel baik secara *data dependence* maupun *control dependence*. Pustaka ini membutuhkan masukan berupa Java Class (Java *bytecode*). Langkah-langkah dalam proses awal eksekusi pustaka *Java System Dependence Graph* API adalah sebagai berikut:

a. Pemrosesan ketergantungan antar variabel dapat dilakukan dengan cara membahkan baris kode program sebagai berikut:

```
ProceduralDependenceGraphMatrix namavariabel =
ProceduralDependenceGraphMatrix.getPDGByMethodSignature (alamat
direktori Java Class, nama method, (parameter method) return
value);
```

b. Ketergantungan antar variabel dapat dilihat dengan menambahkan baris kode program sebagai berikut:

```
namavariabel.getNodes()[1].getVariableName();
```

c. Contoh keluaran ketergantungan antar variabel:

```
testr.diagram = diag
```

Dimana `testr` adalah nama *class* dan `diagram` adalah deklarasi variabel yang nilainya merujuk dari variabel `diag`.

6.3.2 Kasus Uji Pengujian Akurasi

Dalam sub bab ini dilakukan perhitungan akurasi terhadap 30 *method* yang berasal dari pustaka JavaCV dan ejml. Kemudian dilakukan pemrosesan untuk mendapatkan ketergantungan antar variabel dari *method* yang ada dalam kedua pustaka tersebut. Pemrosesan dilakukan dengan membuat *Program Dependence Graph* secara manual dan membuat *Program Dependence Graph* secara otomatis melalui dua sistem yaitu sistem pembangkitan *Program Dependence Graph* yang dibuat pada penelitian ini dan pustaka *Java System Dependence Graph* API (JSDG). Perhitungan akurasi dilakukan dengan menggunakan persamaan 2.2.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

6.3.2.1 Kasus Uji *Method* DistanceToLine

Berikut ini merupakan kasus uji *method* DistanceToLine yang ditunjukkan pada tabel 6.13.

Tabel 6.13 Kasus Uji *Method* DistanceToLine

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	dx - x1	Ada	Ada
2	dx - x2	Ada	Ada
3	dy - y1	Ada	Ada
4	dy - y2	Ada	Ada
5	d2 – dx	Ada	Ada
6	d2 – dy	Ada	Ada
7	u - x3	Ada	Ada
8	u - x1	Ada	Ada
9	u – dx	Ada	Ada
10	u - y1	Ada	Ada
11	u – dy	Ada	Ada
12	u - d2	Ada	Ada
13	x - x1	Ada	Ada
14	x – u	Ada	Ada
15	x – dx	Ada	Ada
16	y - y1	Ada	Ada
17	y – u	Ada	Ada
18	y – dy	Ada	Ada
19	dx – x	Ada	Ada
20	dx - x3	Ada	Ada
21	dy – y	Ada	Ada
22	dy - y3	Ada	Ada
23	return – dx	Ada	Ada
24	return – dy	Ada	Ada

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.13 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* DistanceToLine:

$$Akurasi\ Sistem = \frac{24}{24} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{24}{24} \times 100\% = 100\%$$

6.3.2.2 Kasus Uji *Method* perspectiveTransform

Berikut ini merupakan kasus uji *method* perspectiveTransform yang ditunjukkan pada tabel 6.14.

Tabel 6.14 Kasus Uji *Method* perspectiveTransform

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	mat - mapmatrix.get()	Ada	Ada
2	j – src	Ada	Ada
3	x - src[j]	Ada	Ada
4	y – src	Ada	Ada
5	w – x	Ada	Ada
6	w – mat	Ada	Ada
7	w – w	Ada	Ada
8	dst – x	Tidak ada	Ada
9	dst – y	Tidak ada	Ada
10	dst – w	Tidak ada	Ada
11	dst – mat	Tidak ada	Ada
12	dst – dst	Tidak ada	Ada
13	j – j	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.14 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* perspectiveTransform:

$$Akurasi\ Sistem = \frac{13}{13} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{8}{13} \times 100\% = 61{,}5\%$$

6.3.2.3 Kasus Uji *Method* getperspectiveTransform

Berikut ini merupakan kasus uji *method* getperspectiveTransform yang ditunjukkan pada tabel 6.15.

Tabel 6.15 Kasus Uji *Method* getperspectiveTransform

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	n - n3x1.get()	Ada	Ada
2	getPlaneParameter - src	Tidak ada	Ada
3	getPlaneParameter - dst	Tidak ada	Ada
4	getPlaneParameter - invSrcK	Tidak ada	Ada
5	getPlaneParameter - dstK	Tidak ada	Ada
6	getPlaneParameter – R	Tidak ada	Ada
7	getPlaneParameter - t	Tidak ada	Ada
8	getPlaneParameter - n	Tidak ada	Ada
9	cvGEMM – t	Tidak ada	Ada
10	cvGEMM – n	Tidak ada	Ada
11	cvGEMM – R	Tidak ada	Ada
12	cvGEMM – H	Tidak ada	Ada
13	cvGEMM - CV_GEMM_B_T	Tidak ada	Ada
14	cvMatMul - dstK	Tidak ada	Ada
15	cvMatMul – H	Tidak ada	Ada
16	return H	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.15 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* getperspectiveTransform:

$$Akurasi\ Sistem = \frac{16}{16} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{14}{16} \times 100\% = 12{,}5\%$$

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

6.3.2.4 Kasus Uji *Method* unitize

Berikut ini merupakan kasus uji *method* unitize yang ditunjukkan pada tabel 6.16.

Tabel 6.16 Kasus Uji *Method* unitize

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	Math.sqrt – A	Tidak ada	Ada
2	Math.sqrt – B	Tidak ada	Ada
3	norm - Math.sqrt	Ada	Ada
4	A – norm	Ada	Ada
5	B – norm	Ada	Ada
6	Return – new Array	Tidak ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.16 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* unitize:

$$Akurasi = \frac{5}{6} \times 100\% = 83{,}3\%$$

$$Akurasi = \frac{3}{6} \times 100\% = 50\%$$

6.3.2.5 Kasus Uji *Method* norm

Berikut ini merupakan kasus uji *method* norm yang ditunjukkan pada tabel 6.17.

Tabel 6.17 Kasus Uji *Method* norm

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	norm – v	Tidak ada	Ada
2	return – norm	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.17 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* norm:

$$Akurasi\ Sistem = \frac{2}{2} \times 100\% = 100\%$$

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

$$Akurasi\ JSDG = \frac{1}{2} \times 100\% = 50\%$$

6.3.2.6 Kasus Uji *Method* median

Berikut ini merupakan kasus uji *method* median yang ditunjukkan pada tabel 6.18.

Tabel 6.18 Kasus Uji *Method* median

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	sorted - doubles.clone();	Ada	Ada
2	Arrays.sort() sorted	Ada	Ada
3	Return – sorted	Ada	Ada
4	Return - sorted	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.18 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* median:

$$Akurasi\ Sistem = \frac{4}{4} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{4}{4} \times 100\% = 100\%$$

6.3.2.7 Kasus Uji *Method* marker

Berikut ini merupakan kasus uji *method* marker yang ditunjukkan pada tabel 6.19.

Tabel 6.19 Kasus Uji *Method* marker

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	this.id – id	Ada	Ada
2	this.corners – corners	Ada	Ada
3	this.confidence – confidence	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.19 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* marker:

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

$$Akurasi\ Sistem = \frac{3}{3} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{3}{3} \times 100\% = 100\%$$

6.3.2.8 Kasus Uji *Method* createArray

Berikut ini merupakan kasus uji *method* createArray yang ditunjukkan pada tabel 6.20.

Tabel 6.20 Kasus Uji *Method* createArray

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	s.rows – rows	Ada	Ada
2	s.sizeX – sizeX	Ada	Ada
3	s.spacingX – spacingX	Ada	Ada
4	s.checkered - checkered	Ada	Ada
5	s.coloums – coloums	Ada	Ada
6	s.sizeY – sizeY	Ada	Ada
7	s.spacingY - spacingY	Ada	Ada
8	return – createArray	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.20 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* createArray:

$$Akurasi\ Sistem = \frac{8}{8} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{8}{8} \times 100\% = 100\%$$

6.3.2.9 Kasus Uji *Method* get

Berikut ini merupakan kasus uji *method* get yang ditunjukkan pada tabel 6.21.

Tabel 6.21 Kasus Uji *Method* get

Nomor	Ketergantungan Variabel	JSDG	Sistem

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

1	p – size	Ada	Ada
2	get – i	Tidak ada	Ada
3	p-get	Tidak ada	Ada
4	return - p	Ada	Ada
5	i – i	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.21 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* get:

$$Akurasi\ Sistem = \frac{5}{5} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{3}{5} \times 100\% = 60\%$$

6.3.2.10 Kasus Uji *Method* compose

Berikut ini merupakan kasus uji *method* compose yang ditunjukkan pada tabel 6.22.

Tabel 6.22 Kasus Uji *Method* compose

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	cvInvert - H1	Tidak ada	Ada
2	cvInvert - H2	Tidak ada	Ada
3	cvMatMul - H2	Tidak ada	Ada
4	cvMatMul - H1	Tidak ada	Ada
5	Set – H	Tidak ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.22 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* compose:

$$Akurasi\ Sistem = \frac{5}{5} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{0}{5} \times 100\% = 0\%$$

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

6.3.2.11 Kasus Uji *Method* SubsumptionChain

Berikut ini merupakan kasus uji *method* SubsumptionChain yang ditunjukkan pada tabel 6.23.

Tabel 6.23 Kasus Uji *Method* SubsumptionChain

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	Str – x	Ada	Ada
2	Lastx – x	Ada	Ada
3	x - SubsumedLabel	Ada	Ada
4	System.out.println - Str	Tidak ada	Ada
5	return – LastX	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.23 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* SubsumptionChain:

$$Akurasi\ Sistem = \frac{5}{5} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{4}{5} \times 100\% = 80\%$$

6.3.2.12 Kasus Uji *Method* wrap

Berikut ini merupakan kasus uji *method* wrap yang ditunjukkan pada tabel 6.24.

Tabel 6.24 Kasus Uji *Method* wrap

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	ret.data – data	Ada	Ada
2	ret.numRows - numRows	Ada	Ada
3	ret.numCols - numCols	Ada	Ada
4	ret.blockLength - blockLength	Ada	Ada
5	return – ret	Ada	Ada

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.24 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* wrap:

$$Akurasi\ Sistem = \frac{5}{5} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{5}{5} \times 100\% = 100\%$$

6.3.2.13 Kasus Uji *Method* reshap

Berikut ini merupakan kasus uji *method* reshap yang ditunjukkan pada tabel 6.25.

Tabel 6.25 Kasus Uji *Method* reshap

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	this.numRows - numRows	Ada	Ada
2	this.numCols - numCols	Ada	Ada
3	data – numRows	Ada	Ada
4	data – numCols	Ada	Ada
5	System.arraycopy - data	Ada	Ada
6	this.data – data	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.25 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* reshap:

$$Akurasi\ Sistem = \frac{6}{6} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{6}{6} \times 100\% = 100\%$$

6.3.2.14 Kasus Uji *Method* getIndex

Berikut ini merupakan kasus uji *method* getIndex yang ditunjukkan pada tabel 6.26.

Tabel 6.26 Kasus Uji *Method* getIndex

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	blockRow – row	Ada	Ada
2	blockRow - blockLength	Ada	Ada
3	blockCol – col	Ada	Ada
4	blockCol - blockLength	Ada	Ada
5	Math.min - numRows	Tidak ada	Ada
6	Math.min - blockRow	Tidak ada	Ada
7	Math.min - blockLength	Tidak ada	Ada
8	index – blockRow	Ada	Ada
9	index - blockLength	Ada	Ada
10	index – numCols	Ada	Ada
11	index – blockCol	Ada	Ada
12	index - blockLength	Ada	Ada
13	Math.min - numCols	Tidak ada	Ada
14	Math.min - blockLength	Tidak ada	Ada
15	Math.min - blockCol	Tidak ada	Ada
16	localHeight - Math.min	Ada	Ada
17	localLength - Math.min	Ada	Ada
18	row – blockLength	Ada	Ada
19	col – blockLength	Ada	Ada
20	return – index	Ada	Ada
21	return - localLength	Ada	Ada
22	return – row	Ada	Ada
23	return – col	Ada	Ada
24	blockRow – row	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.26 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* getIndex:

$$Akurasi\ Sistem = \frac{24}{24} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{18}{24} \times 100\% = 75\%$$

6.3.2.15 Kasus Uji *Method* CDenseMatrix64F(double[][])

Berikut ini merupakan kasus uji *method* CDenseMatrix64F yang ditunjukkan pada tabel 6.27.

Tabel 6.27 Kasus Uji *Method* CDenseMatrix64F(double[][])

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	this.numRows - data	Ada	Ada
2	this.numCols - data	Ada	Ada
3	this.data - numRows	Tidak ada	Ada
4	this.data - numCols	Tidak ada	Ada
5	row – data	Ada	Ada
6	System.arraycopy – row	Ada	Ada
7	System.arraycopy - this.data	Ada	Ada
8	System.arraycopy – i	Ada	Ada
9	System.arraycopy - numCols	Ada	Ada
10	System.arraycopy - numCols	Ada	Ada
11	row – i	Ada	Ada
12	i – i	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.27 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* CDenseMatrix64F(double[][]):

$$Akurasi\ Sistem = \frac{12}{12} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{10}{12} \times 100\% = 83{,}3\%$$

6.3.2.16 Kasus Uji *Method* CDenseMatrix64F(int,int,boolean,double)

Berikut ini merupakan kasus uji *method* CDenseMatrix64F 2 yang ditunjukkan pada tabel 6.28.

Tabel 6.28 Kasus Uji *Method* CDenseMatrix64F(int,int,boolean,double)

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	this.data – numRows	Ada	Ada
2	this.data – numCols	Ada	Ada
3	this.numRows – numRows	Ada	Ada
4	this.numCols – numCols	Ada	Ada
5	set – numRows	Tidak ada	Ada
6	set – numCols	Tidak ada	Ada
7	set – rowMajor	Tidak ada	Ada
8	set – data	Tidak ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.28 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* CDenseMatrix64(int,int,boolean,double):

$$Akurasi\ Sistem = \frac{8}{8} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{4}{8} \times 100\% = 50\%$$

6.3.2.17 Kasus Uji *Method* CDenseMatrix64F(int,int)

Berikut ini merupakan kasus uji *method* CDenseMatrix64F 4 yang ditunjukkan pada tabel 6.29.

Tabel 6.29 Kasus Uji *Method* CDenseMatrix64F(int,int)

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	this.numRows - numRows	Ada	Ada
2	this.numCols - numCols	Ada	Ada
3	this.data – numRows	Ada	Ada
4	this.data – numCols	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.29 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* CDenseMatrix64F 4:

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

$$Akurasi\ Sistem = \frac{4}{4} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{4}{4} \times 100\% = 100\%$$

6.3.2.18 Kasus Uji *Method* reshape

Berikut ini merupakan kasus uji *method* reshap CDenseMatrix64F yang ditunjukkan pada tabel 6.30.

Tabel 6.30 Kasus Uji *Method* reshape

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	newLength - numRows	Ada	Ada
2	newLength – numCols	Ada	Ada
3	data – newLength	Ada	Ada
4	this.numRows - numRows	Ada	Ada
5	this.numCols - numCols	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.30 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* reshape:

$$Akurasi\ Sistem = \frac{5}{5} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{5}{5} \times 100\% = 100\%$$

6.3.2.19 Kasus Uji *Method* innerProd

Berikut ini merupakan kasus uji *method* innerProd yang ditunjukkan pada tabel 6.31.

Tabel 6.31 Kasus Uji *Method* innerProd

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	output - Complex64F	Ada	Ada
2	output.real - output.imaginary	Ada	Ada
3	m - x.getDataLength()	Ada	Ada

repository.ub.ac.id

UNIVERSITAS BRAWIJAYA

4	x.data – i	Ada	Ada
5	realX - x.data	Ada	Ada
6	imagX - x.data	Ada	Ada
7	realY - y.data	Ada	Ada
8	imagY - y.data	Ada	Ada
9	output.real - realX	Ada	Ada
10	output.real - realY	Ada	Ada
11	output.real - imagX	Ada	Ada
12	output.real - imagY	Ada	Ada
13	output.imaginary - realX	Ada	Ada
14	output.imaginary - realY	Ada	Ada
15	output.imaginary - imagX	Ada	Ada
16	output.imaginary - imagY	Ada	Ada
17	i – i	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.31 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* innerProd:

$$Akurasi\ Sistem = \frac{17}{17} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{17}{17} \times 100\% = 100\%$$

6.3.2.20 Kasus Uji *Method* innerProdH

Berikut ini merupakan kasus uji *method* innerProdH yang ditunjukkan pada tabel 6.32.

Tabel 6.32 Kasus Uji *Method* innerProdH

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	output - Complex64F	Ada	Ada
2	output.real - output.imaginary	Ada	Ada
3	m - x.getDataLength()	Ada	Ada
4	x.data – i	Ada	Ada

5	realX - x.data	Ada	Ada
6	imagX - x.data	Ada	Ada
7	realY - y.data	Ada	Ada
8	imagY - y.data	Ada	Ada
9	output.real – realX	Ada	Ada
10	output.real – realY	Ada	Ada
11	output.real - imagX	Ada	Ada
12	output.real - imagY	Ada	Ada
13	output.imaginary - realX	Ada	Ada
14	output.imaginary - realY	Ada	Ada
15	output.imaginary - imagX	Ada	Ada
16	output.imaginary - imagY	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.32 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* innerProdH:

$$Akurasi\ Sistem = \frac{16}{16} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{16}{16} \times 100\% = 100\%$$

6.3.2.21 Kasus Uji *Method* outerProd

Berikut ini merupakan kasus uji *method* outerProd yang ditunjukkan pada tabel 6.33.

Tabel 6.33 Kasus Uji *Method* outerProd

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	m - A.numRows	Tidak ada	Ada
2	n - A.numCols	Tidak ada	Ada
3	realX - x.data	Ada	Ada
4	imagX - x.data	Ada	Ada
5	realY - y.data	Ada	Ada
6	imagY - y.data	Ada	Ada

7	A.data – realX	Tidak ada	Ada
8	A.data – realY	Tidak ada	Ada
9	A.data – imagX	Tidak ada	Ada
10	A.data – imagY	Tidak ada	Ada
11	i – i	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.33 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* outerProd:

$$Akurasi\ Sistem = \frac{11}{11} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{5}{11} \times 100\% = 45{,}4\%$$

6.3.2.22 Kasus Uji *Method* outerProdH

Berikut ini merupakan kasus uji *method* outerProdH yang ditunjukkan pada tabel 6.34.

Tabel 6.34 Kasus Uji *Method* outerProdH

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	m - A.numRows	Tidak ada	Ada
2	n - A.numCols	Tidak ada	Ada
3	realX - x.data	Ada	Ada
4	imagX - x.data	Ada	Ada
5	realY - y.data	Ada	Ada
6	imagY - y.data	Ada	Ada
7	A.data – realX	Tidak ada	Ada
8	A.data – realY	Tidak ada	Ada
9	A.data – imagX	Tidak ada	Ada
10	A.data – imagY	Tidak ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.34 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* outerProdH:

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

$$Akurasi\ Sistem = \frac{10}{10} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{4}{10} \times 100\% = 25\%$$

6.3.2.23 Kasus Uji *Method* solveU

Berikut ini merupakan kasus uji *method* solveU yang ditunjukkan pada tabel 6.35.

Tabel 6.35 Kasus Uji *Method* solveU

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	stride – n	Ada	Ada
2	sumReal – b	Ada	Ada
3	sumImg – b	Ada	Ada
4	indexU – i	Ada	Ada
5	indexU – stride	Ada	Ada
6	realB – b	Ada	Ada
7	imgB – b	Ada	Ada
8	realU – U	Ada	Ada
9	imgU – U	Ada	Ada
10	sumReal – realB	Ada	Ada
11	sumReal – realU	Ada	Ada
12	sumReal – imgB	Ada	Ada
13	sumReal – imgB	Ada	Ada
14	sumImg – realB	Ada	Ada
15	sumImg – realU	Ada	Ada
16	sumImg – imgB	Ada	Ada
17	sumImg – imgB	Ada	Ada
18	normU – realU	Ada	Ada
19	normU – imgU	Ada	Ada
20	b – sumReal	Tidak ada	Ada
21	b – realU	Tidak ada	Ada

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

22	b – sumImg	Tidak ada	Ada
23	b – imgU	Tidak ada	Ada
24	b – normU	Tidak ada	Ada
25	i - i	Ada	Ada
26	j - j	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.35 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* solveU:

$$Akurasi\ Sistem = \frac{26}{26} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{21}{26} \times 100\% = 80{,}76\%$$

6.3.2.24 Kasus Uji *Method* solveL_diagReal

Berikut ini merupakan kasus uji *method* solveL-diagReal yang ditunjukkan pada tabel 6.36.

Tabel 6.36 Kasus Uji *Method* solveL_diagReal

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	stride – n	Ada	Ada
2	realSum – b	Ada	Ada
3	imagSum – b	Ada	Ada
4	indexL – i	Ada	Ada
5	indexL – stride	Ada	Ada
6	realB – b	Ada	Ada
7	imagB – b	Ada	Ada
8	realL – L	Ada	Ada
9	imagL – L	Ada	Ada
10	realSum – realB	Ada	Ada
11	realSum – realL	Ada	Ada
12	realSum – imagB	Ada	Ada
13	realSum – imagL	Ada	Ada

14	imagSum – realB	Ada	Ada
15	imagSum – realL	Ada	Ada
16	imagSum – imagB	Ada	Ada
17	imagSum – imagL	Ada	Ada
18	b – realSum	Tidak ada	Ada
19	b – realL	Tidak ada	Ada
20	b – imagSum	Tidak ada	Ada
21	i – i	Ada	Ada
22	k – k	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.36 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* solveL_diagReal:

$$Akurasi\ Sistem = \frac{22}{22} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{19}{22} \times 100\% = 86{,}36\%$$

6.3.2.25 Kasus Uji *Method* solveConjTranL_diagReal

Berikut ini merupakan kasus uji *method* solveConjTranL_diagReal yang ditunjukkan pada tabel 6.37.

Tabel 6.37 Kasus Uji *Method* solveConjTranL_diagReal

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	realSum – b	Ada	Ada
2	imagSum – b	Ada	Ada
3	indexB – i	Ada	Ada
4	indexL – k	Ada	Ada
5	indexL – n	Ada	Ada
6	indexL – i	Ada	Ada
7	realL – L	Ada	Ada
8	imagL – L	Ada	Ada
9	realB – b	Ada	Ada

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

10	imagB – b	Ada	Ada
11	realSum - realB	Ada	Ada
12	realSum - realL	Ada	Ada
13	realSum - imagB	Ada	Ada
14	realSum - imagL	Ada	Ada
15	imagSum - realB	Ada	Ada
16	imagSum – realL	Ada	Ada
17	imagSum – imagB	Ada	Ada
18	imagSum – imagL	Ada	Ada
19	b – realSum	Tidak ada	Ada
20	b – realL	Tidak ada	Ada
21	b – imagSum	Tidak ada	Ada
22	i – i	Ada	Ada
23	k – k	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.37 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* solveConjTranL-diagReal:

$$Akurasi\ Sistem = \frac{23}{23} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{20}{23} \times 100\% = 86,9\%$$

6.3.2.26 Kasus Uji *Method* invertLower

Berikut ini merupakan kasus uji *method* invertLower yang ditunjukkan pada tabel 6.38.

Tabel 6.38 Kasus Uji *Method* invertLower

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	L_ii – L	Ada	Ada
2	val – L	Ada	Ada
3	L – val	Tidak ada	Ada
4	L - L_ii	Tidak ada	Ada

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

5	i –i	Ada	Ada
6	k – k	Ada	Ada
7	j – j	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.38 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* invertLower:

$$Akurasi\ Sistem = \frac{7}{7} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{5}{7} \times 100\% = 71{,}4\%$$

6.3.2.27 Kasus Uji *Method* solveL

Berikut ini merupakan kasus uji *method* solveL yang ditunjukkan pada tabel 6.39.

Tabel 6.39 Kasus Uji *Method* solveL

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	sum – b	Ada	Ada
2	sum – L	Ada	Ada
3	b – sum	Tidak ada	Ada
4	b – L	Tidak ada	Ada
5	i - i	Ada	Ada
6	j – j	Ada	Ada
7	k – k	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.39 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* solveL:

$$Akurasi = \frac{7}{7} \times 100\% = 100\%$$

$$Akurasi = \frac{5}{7} \times 100\% = 71{,}4\%$$

6.3.2.28 Kasus Uji *Method* solveU2

Berikut ini merupakan kasus uji *method* solveU2 yang ditunjukkan pada tabel 6.40.

Tabel 6.40 Kasus Uji *Method* solveU2

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	sum – b	Ada	Ada
2	indexU – i	Ada	Ada
3	indexU – n	Ada	Ada
4	sum – U	Ada	Ada
5	b – sum	Tidak ada	Ada
6	b – U	Tidak ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.40 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* solveU2:

$$Akurasi\ Sistem = \frac{6}{6} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{4}{6} \times 100\% = 66.667\%$$

6.3.2.29 Kasus Uji *Method* undoTranspose

Berikut ini merupakan kasus uji *method* undoTranspose yang ditunjukkan pada tabel 6.41.

Tabel 6.41 Kasus Uji *Method* undoTranspose

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	temp – Vt	Tidak ada	Ada
2	Vt – Ut	Tidak ada	Ada
3	Ut – temp	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.41 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* undoTranspose:

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

$$Akurasi\ Sistem = \frac{3}{3} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{1}{3} \times 100\% = 33.333\%$$

6.3.2.30 Kasus Uji *Method* setMatrix

Berikut ini merupakan kasus uji *method* setMatrix yang ditunjukkan pada tabel 6.42.

Tabel 6.42 Kasus Uji *Method* setMatrix

Nomor	Ketergantungan Variabel	JSDG	Sistem
1	initParam – numRows	Tidak ada	Ada
2	initParam – numCols	Tidak ada	Ada
3	this.diag – diag	Ada	Ada
4	this.off – off	Ada	Ada
5	Math.abs – diag	Ada	Ada
6	maxValue - Math.abs(diag)	Ada	Ada
7	a - Math.abs(diag)	Ada	Ada
8	b - Math.abs(diag)	Ada	Ada
9	Math.abs – a	Ada	Ada
10	maxValue - Math.abs(a)	Ada	Ada
11	Math.abs – b	Ada	Ada
12	maxValue - Math.abs(b)	Ada	Ada
13	i - i	Ada	Ada

Berdasarkan kasus uji yang telah diperoleh seperti pada tabel 6.42 maka diperoleh tingkat akurasi melalui persamaan 2.2. Berikut ini merupakan perhitungan akurasi *method* setMatrix:

$$Akurasi\ Sistem = \frac{13}{13} \times 100\% = 100\%$$

$$Akurasi\ JSDG = \frac{11}{13} \times 100\% = 84{,}6\%$$

6.3.3 Hasil dan Analisis Pengujian Akurasi

Berikut ini merupakan hasil pengujian akurasi pada Kakas Bantu Pembangkitan *Program Dependence Graph.*

Tabel 6.43 Hasil Pengujian Akurasi

Nomor	Nama *Method*	Akurasi Sistem	Akurasi JSDG
1	DistanceToLine	100	100
2	perspectiveTransform	100	61,5
3	getperspectiveTransform	100	12,5
4	Unitize	100	50
5	Norm	100	50
6	Median	100	100
7	Marker	100	100
8	createArray	100	100
9	Get	100	60
10	Compose	100	0
11	SubsumptionChain	100	80
12	Wrap	100	100
13	Reshape	100	100
14	getIndex	100	75
15	CDenseMatrix64F(double[][])	100	83,3
16	CDenseMatrix64F(int,int,boolean,double)	100	50
17	*Method* CDenseMatrix64F(int,int)	100	100
18	Reshape	100	100
19	innerProd	100	100
20	innerProdH	100	100
21	outerProd	100	45,4
22	outerProdH	100	25
23	solveU	100	80,76
24	solveL-diagReal	100	86,36
25	solveConjTranL_diagReal	100	86,9

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

26	invertLower	100	71,4
27	solveL	100	71,4
28	solveU2	100	66,6
29	undoTranspose	100	33,3
30	setMatrix	100	84,6
Rata-Rata Akurasi		100%	77,2%

Dari hasil pengujian akurasi yang diperoleh seperti tabel 6.43, terlihat bahwa rata-rata akurasi yang dimiliki oleh sistem ini sebesar 100%. Akurasi sebesar 100% dapat diperoleh karena tidak ada kesalahan dalam proses ekstraksi variabel pada kode program yang dimasukkan oleh pengguna. Pustaka JSDG memiliki akurasi sebesar 77,2% dikarenakan JSDG gagal dalam menampilkan ketergantungan variabel berbentuk *array* dan pemanggilan *method*. Contoh variabel berbentuk array yang gagal ditampilkan pada *method* solveL adalah `b[i*n+j] = sum / L[i*m+i]`. Hal ini terjadi jika array adalah variabel deklarasi dan bukan variabel rujukan. Pernyataan pemanggilan *method* juga gagal dideteksi oleh pustaka JSDG. Contoh pernyataan pemanggilan *method* yang gagal ditampilkan adalah `cvMatMul(dstK, H, H).` Pernyataan `cvMatMul(dstK, H, H);` mempunyai arti bahwa nilai dari `cvMatMul` dipengaruhi oleh variabel `dstK, H, H`. Pada *bytecode* nilai dari pemanggilan smethod dituliskan dengan `result$namaMethod_tipedata` tanpa informasi parameter yang mempengaruhi. Hal ini yang menyebabkan informasi ketergantungan menjadi hilang apabila membangkitkan *Program Dependence Graph* dari *bytecode*.

BAB VII
PENUTUP

7.1 Kesimpulan

Kesimpulan yang diambil dari skripsi ini adalah:

1. Pembangkitan *Program Dependence Graph* dapat dilakukan dengan melakukan identifikasi ketergantungan variabel pada *source code* Java. Ketergantungan variabel dapat diperoleh melalui tahapan ekstraksi data variabel menggunakan pustaka *Abstract Syntax Tree.* Proses normalisasi dilakukan untuk menghilangkan redudansi data yang ada. Setelah proses normalisasi dilakukan, informasi variabel rujukan ditambahkan apabila nilai suatu variabel merujuk ke variabel lain. Pada tahap akhir dilakukan konversi data ke dalam bentuk grafik dilakukan dengan menggunakan pustaka JUNG.
2. Proses pembangunan kakas bantu pembangkitan *Program Dependence Graph* dapat dilakukan dengan menggunakan pendekatan *reuse-oriented software development.* Proses pembangunan diawali dengan melakukan analisis kebutuhan, merancang perangkat lunak, melakukan implementasi, dan terakhir melakukan validasi perangkat lunak. Hasil uji validitas pada perangkat lunak sudah valid 100% sesuai dengan hasil yang diharapkan.
3. Pengujian akurasi yang dilakukan dalam penelitian ini adalah dengan membandingkan pembangkitan *Program Dependence Graph* secara manual dengan pembangkitan secara otomatis. Pembangkitan secara otomatis dilakukan dengan dua sistem yaitu sistem yang dibangun pada penelitian ini dengan masukan berupa *source code* dan pustaka JSDG dengan masukan berupa *bytecode*. Akurasi sistem yang telah dibuat adalah 100% sementara akurasi dari pustaka JSDG adalah 77,2%. Akurasi yang dimiliki pustaka JSDG lebih rendah karena informasi deklarasi *array* dan parameter pernyataan pemanggilan *method* gagal ditampilkan.

7.2 Saran

Saran untuk penelitian selanjutnya dari skripsi ini adalah:

1. Menambahkan beberapa fitur seperti:

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

a. Menampilkan *source code* pada saat *Program Dependence Graph* terbentuk dan memberi *highlight* pada *source code* pada saat *node* dipilih.
b. Membangkitkan *Program Dependence Graph* pada saat *node* pemanggilan *method* dipilih.

2. Menambahkan *Control Dependence Graph* dan *System Dependence Graph* agar kakas bantu dapat diterapkan ke dalam permasalahan analisis ketergantungan yang lebih luas.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

DAFTAR PUSTAKA

[BIN-07] Binkley, David. 2007. "*Source Code Analysis: A Road Map"*. Future of Software Engineering (FOSE'07) 0-7695-2829-5/07 IEEE.

[DAN-12] Dang, Yingnong, Dongmei Zhang, Song Ge, Chengyun Chu,Yingjun Qiu, Tao Xie. 2012. "*XIAO: Tuning Code Clones at Hands of Engineers in Practice*". ACM 978-1-4503-1312-4/12/12, Orlando, Florida USA.

[FER-87] Ferrante, J., Ottenstein,K., Warren, J. 1987. "*The Program Dependence Graph and Its Use in Optimization"*. ACM Transactions on Programming Languages and Systems, Vol. 9, No. 3, Juli 1987, 319-349.

[GEN-05] Genaim, Samir, Fausto Spoto. 2005. "*Information Flow Analysis for Java Bytecode*". Springer-Verlag Berlin Heidelberg.

[HAR-11] Hariyanto, Bambang. 2011. "Esensi- esensi Bahasa Pemrograman Java". Bandung: Informatika Bandung.

[HRM-10] Harman, Mark. 2010. "*Why Source Code Analysis and Manipulation Will Always Be Important* ". SCAM 2011.

[KRI-01] Krinke, J. 2001. "*Identifying Similar Code with Program Dependence Graphs"*. 1095-1350/01 2001 IEEE.

[KUH-06] Kuhn, Thomas, Oliver Thomann. 2006. "*Abstract Syntax Tree*". EPL v1.0.

[MCC-76] McCabe, Thomas J. 1976. "A Complexity Measure". IEEE Transaction on software engineering, Vol SE-2, NO.4, DECEMBER 1976

[NUG-14] Nugroho. S. 2014. Deteksi dan Pengenalan Plat Nomor Mobil Menggunakan Vertical Edge Detection dan Backpropagation Neural Network. Skripsi S-1 Program Studi Informatika/Ilmu Komputer, Program Teknologi dan Ilmu Komputer, Universitas Brawijaya, Malang.

UNIVERSITAS BRAWIJAYA

[OHA-00] Ohata, F., Nishimatsu, A., Inoue, K. 2000. "*Analyzing dependence locality for efficient construction of program dependence graph*". Information and Software Technology 42 (2000), 935-946.

[PRI-14] Priyambadha, B., Rochimah, S. 2014. "*Case Study on Semantic Clone Detection Based On Code Behavior*". International Conference on Data and Software Engineering 978-1-4799-7996-7/14 2014 IEEE.

[ROS-11] Rosa, A., S. dan Shalahudin, M. 2011. "Rekayasa Perangkat Lunak (Terstruktur dan Berorientasi Objek)". Bandung: Modula.

[SOM-11] Sommerville, Ian. 2011. "*Software Engineering* (9th ed.). Boston: Pearson/Addison-Weasley.

[SPA-15] Sparx Systems. "UML 2 Component Diagram". http://www.sparxsystems.com.au/resources/uml2_tutorial/uml2_componentdiagram.html. Diakses pada tanggal 30 Juli 2015.

[TON-10] TONG, Chun Yin. 2010. "*Java System Dependence Graph API*". www4.comp.polyu.edu.hk. Diakses pada tanggal 19 Mei 2015.