KAKAS BANTU PENILAIAN KOPLING MENGGUNAKAN METRIK *INDIRECT PACKAGE COUPLING BASED ON RESPONSIBILTY*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Muhammad Ridha Anshari
NIM: 135150200111029

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

KAKAS BANTU PENILAIAN KOPLING MENGGUNAKAN METRIK INDIRECT PACKAGE COUPLING BASED ON RESPONSIBILTY
SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh:
Muhammad Ridha Anshari
NIM: 135150200111029

Skripsi ini telah diuji dan dinyatakan lulus pada
6 Juni 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Fajar Pradana, S.ST, M.Eng	Bayu Priyambadha, S.Kom, M.Kom
NIP. 19871121 201504 1 004	NIP. 19820909 200812 1 004

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D.
NIP. 19710518 200312 1 001

PERNYATAAN ORISINALITAS

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 disitasi dalam naskah ini dan disebutkan dalam 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, 6 Juni 2018

Muhammad Ridha Anshari

NIM: 135150200111007

KATA PENGANTAR

Puji Syukur kehadirat Allah SWT yang telah memberikan rahmat dan karunia-Nya, sehingga Penulis dapat menyelesaikan skripsi yang berjudul "Kakas Bantu Penilaian Kopling Menggunakan Metrik *Indirect Package Coupling Based on Responsibility*" dengan baik. Skripsi ini disusun untuk memenuhi salah satu syarat dalam mendapatkan gelar Sarjana Komputer di Fakultas Ilmu Komputer Universitas Brawijaya Malang.

Penulisan skripsi ini tidak lepas dari bimbingan serta dukungan dari berbagai pihak, baik secara moril maupun materiil. Oleh sebab itu, Penulis mengucapkan banyak terima kasih kepada:

1. Bapak Wayan Firdaus Mahmudy, S.Si., M.T., Ph.D. selaku Dekan Fakultas Ilmu Komputer, Universitas Brawijaya Malang.
2. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D. selaku Ketua Jurusan Teknik Informatika, Universitas Brawijaya Malang.
3. Bapak Fajar Pradana, S.ST, M.Eng selaku dosen pembimbing I yang telah memberikan waktu dan nasehatnya, serta dengan sangat sabar membimbing selama penulisan skripsi.
4. Bapak Bayu Priyambadha, S.Kom, M.Kom selaku dosen pembimbing II yang telah memberikan waktu dan nasehatnya, serta dengan sangat sabar membimbing selama penulisan skripsi.
5. Bapak dan Ibu dosen yang telah memberikan ilmu selama Penulis menempuh pendidikan di Fakultas Ilmu Komputer Universitas Brawijaya.
6. Segenap karyawan di Fakultas Ilmu Komputer Universitas Brawijaya yang membantu Penulis dalam pelaksanaan skripsi ini.
7. Orang tua Penulis, Ibu Kaspiah dan Bapak Mawardy Hatta, serta kakak-kakak dan adik Penulis, dan keluarga besar yang selalu mendukung dalam doa, memberikan tauladan dan kasih sayang, serta memberi motivasi dikala lelah dan semua itu semata-mata hanya untuk keberhasilan Penulis.
8. Sahabat dekat penulis Dwi Novi Setiawan, Edwin Damar P., dan Reza M. R. yang terus memberikan dukungan, bantuan, serta menjadi teman diskusi sehingga Penulis dapat menyelesaikan skripsi ini.
9. Teman-teman seperantauan Evtriyandani, Octavia D. H., Silvia Maghfirah, Lengga Buana, Adam R. Yudhana, dan lainnya yang telah menjadi keluarga ke-2 bagi penulis, dan senantiasa memberikan dukungan kepada penulis.
10. Teman-teman seperjuangan R. Moh. Andriawan A., Dimas Joko H., Firly Wahyudi, Irfan Aprison, Reza S., M. Hafidzar, M. Reyhan, dan lainnya yang tidak bisa disebutkan satu persatu. Terima kasih atas doa, dukungan, bantuan, kebersamaan, dan canda tawanya.

Penulis menyadari bahwa dalam penulisan skripsi ini masih terdapat banyak kekurangan. Oleh sebab itu, penulis mengharapkan kritik dan saran yang membangun, sehingga skripsi ini bisa menjadi lebih baik lagi.

Malang, 6 Juni2018

Penulis

mridhaans@gmail.com

ABSTRAK

Pesatnya kemajuan teknologi saat ini menyebabkan perlunya dilakukan perawatan perangkat lunak secara rutin agar tetap relevan. Perawatan dapat lebih mudah dilakukan terhadap perangkat lunak dengan kualitas rancangan yang baik. Kualitas rancangan dapat diukur menggunakan metrik pengukur ketergantungan antarmodul, yaitu kopling. Metrik untuk menghitung nilai kopling yang baik perlu mempertimbangkan adanya relasi yang bersifat tidak langsung, seperti metrik *indirect package coupling based on responsibility*. Berdasarkan permasalahan-permasalahan tersebut, dibangunlah kakas bantu penilaian kopling *package* dengan metrik *indirect package coupling based on responsiblity.* Berkas diagram *package* yang dimasukkan harus diekspor terlebih dahulu ke dalam format XML (*eXtensible Markup Language*) dengan menggunakan perangkat lunak seperti *Visual Paradigm.* Berkas tersebut diproses oleh sistem menggunakan *parser* DOM (*Document Object Model*). Kakas bantu ini sudah teruji dengan pengujian unit dan integrasi menggunakan teknik pengujian *white box,* serta pengujian validasi menggunakan teknik pengujian *black box*. Selain itu, dilakukan pengujian performa menggunakan 3 *package* dari diagram *package* JDraw yang disederhanakan, yaitu event, gui, dan action. Dari hasil pengujian diperoleh bahwa kakas bantu berhasil menghemat waktu penilaian kopling *package* event yang dibutuhkan manusia sekitar 10 menit 27,1188 detik.

Kata kunci: kopling, package, dom, xml, perawatan

ABSTRACT

Rapid development in technology has significantly increased the needs of software maintenance to keep it relevant. The better the quality of the software's design, the easier it is to maintain. That quality can be measured using coupling metric. Indirect dependency between modules needs to be considered for better computational accuracy, which indirect package coupling based on responsibility metric provides. In order to tackle these problems, the package coupling evaluation tool using the indirect package coupling based on responsibility metric is built. The input data is a package diagram file that is exported into an XML (eXtensible Markup Language) format, which can be achieved using a software like Visual Paradigm. The system will parse the file using DOM (Document Object Model) parser. The tool was tested through unit and integration testing using white box testing technique, and validation testing using black box testing technique. It's performance is tested using packages from JDraw's 3 simplified package diagram, which are event, gui, and action. The result shows that this tool can save about 10 minutes and 27.1188 seconds of time needed by human in evaluating the event package's coupling.

Keyword: *coupling, package, dom, xml, maintenance*

DAFTAR ISI

PENGESAHAN i
PERNYATAAN ORISINALITAS ii
KATA PENGANTAR iii
ABSTRAK v
ABSTRACT vi
DAFTAR ISI vii
DAFTAR GAMBAR x
DAFTAR TABEL xii
BAB 1 PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Tujuan 3
1.4 Manfaat 3
1.5 Batasan masalah 3
1.6 Sistematika Pembahasan 3
BAB 2 LANDASAN KEPUSTAKAAN 5
2.1 Kajian Pustaka 5
2.2 Rekayasa Perangkat Lunak 6
2.2.1 *System Development Life Cycle* 6
2.2.2 Model *Waterfall* 7
2.3 Kopling Antar Modul Perangkat Lunak 8
2.3.1 Metrik *Indirect Package Coupling Based on Responsibility* 9
2.4 Extensible Markup Language 12
2.4.2 *Document Object Model* (DOM) 13
2.5 Unified Modelling Language 15
2.6 Pengujian Perangkat Lunak 15
2.6.1 *White Box Testing* 16
2.6.2 *Black Box Testing* 17
BAB 3 METODOLOGI 18
3.1 Studi Literatur 19
3.2 Analisis Kebutuhan 19

3.3 Perancangan 19
3.4 Implementasi 21
3.5 Pengujian Perangkat Lunak 21
3.6 Pengambilan Kesimpulan dan Saran 21
BAB 4 ANALISIS KEBUTUHAN 22
4.1 Gambaran Umum Sistem 22
4.1.1 Deskripsi Umum Sistem 22
4.1.2 Lingkungan Sistem 23
4.2 Analisis Kebutuhan 23
4.2.1 Identifikasi Aktor 24
4.2.2 Analisis Kebutuhan Fungsional 24
4.2.3 Analisis Kebutuhan Non-Fungsional 25
4.2.4 Pemodelan *Use Case Diagram* 25
4.2.5 *Use Case Scenario* 26
BAB 5 PERANCANGAN 29
5.1 Perancangan Arsitektur 29
5.1.1 Perancangan *Sequence Diagram* 29
5.1.2 Perancangan Diagram Klas 33
5.2 Perancangan Komponen 34
5.2.1 Algoritme Memasukkan Berkas Diagram *Package* 34
5.2.2 Algoritme *parsing* xml 34
5.2.3 Algoritme Menilai Kopling 36
5.3 Perancangan Antarmuka Pengguna 38
5.3.1 Rancangan Antarmuka Memasukkan Berkas Diagram *Package* 38
5.3.2 Rancangan Antarmuka Parsing Xml 42
5.3.3 Rancangan Antarmuka Menilai Kopling 45
BAB 6 IMPLEMENTASI 49
6.1 Spesifikasi Lingkungan Pengembangan Sistem 49
6.1.1 Spesifikasi Perangkat Keras 49
6.1.2 Spesifikasi Perangkat Lunak 49
6.1.3 Batasan Implementasi 50
6.2 Implementasi Kode Sumber 50

6.2.1 Implementasi Memasukkan Berkas Diagram *Package* 50
6.2.2 Implementasi *Parsing* Xml 50
6.2.3 Implementasi Menilai Kopling 53
6.3 Implementasi Antarmuka Sistem 55
BAB 7 PENGUJIAN 62
7.1 Pengujian Unit 62
7.1.1 Pengujian *Method* CalculateInRank() 62
7.1.2 Pengujian *Method* CalculateOutRank() 66
7.1.3 Pengujian *Method* CalculatePackageAbstraction () 69
7.2 Pengujian Integrasi 72
7.2.1 Pengujian *Method* CalculateInRank() 72
7.3 Pengujian Validasi 79
7.3.1 Pengujian Memasukkan Berkas Diagram *Package* 80
7.3.2 Pengujian *Parsing* XML 81
7.3.3 Pengujian Menilai Kopling 83
7.4 Pengujian Performa 84
7.4.1 Analisis Hasil Pengujian Performa 86
BAB 8 PENUTUP 91
8.1 Kesimpulan 91
8.2 Saran 92
DAFTAR PUSTAKA 93

DAFTAR GAMBAR

Gambar 2.1 Model *waterfall* 7
Gambar 2.2 Hubungan antara kohesi dan kopling 8
Gambar 2.3 Graf hubungan antara *Instability dan Abstractness* 12
Gambar 2.4 Transformasi *Flowchart* (a) ke *Flowgraph* (b) 16
Gambar 3.1 Diagram alur penelitian 18
Gambar 3.2 Cara kerja program 20
Gambar 4.1 *Use case diagram* kakas bantu penilaian kopling *indirect package* .. 26
Gambar 5.1 Sequence diagram dari memasukkan berkas diagram *package* 29
Gambar 5.2 Sequence diagram dari parsing xml 31
Gambar 5.3 Sequence diagram dari menilai kopling 32
Gambar 5.4 Diagram Klas Kakas Bantu Perhitungan Kopling *Indirect Package* 33
Gambar 5.5 Rancangan antarmuka memasukkan berkas 39
Gambar 5.6 Rancangan antarmuka *file chooser* Java 40
Gambar 5.7 Dialog pesan error "*Unsupported file type*" 41
Gambar 5.8 Rancangan antarmuka halaman *parsing* xml 42
Gambar 5.9 Dialog pesan error "*Required data not found*" 43
Gambar 5.10 Dialog pesan error "*File input error*" 44
Gambar 5.11 Dialog pesan error "*Xml document error*" 44
Gambar 5.12 Rancangan antarmuka halaman menilai kopling 45
Gambar 5.13 Rancangan antarmuka halaman hasil perhitungan 47
Gambar 5.14 Dialog pesan error "*Cyclic relation found*" 48
Gambar 6.1 Antarmuka saat sistem baru dijalankan 56
Gambar 6.2 Antarmuka memilih berkas untuk dimasukkan 56
Gambar 6.3 Antarmuka setelah berkas tervalidasi 57
Gambar 6.4 Antarmuka pesan error jika ekstensi berkas bukan ".xml" 57
Gambar 6.5 Antarmuka hasil parsing berkas secara garis besar 58
Gambar 6.6 Antarmuka hasil parsing berkas secara rinci 58
Gambar 6.7 Antarmuka pesan error jika data tidak ditemukan 59
Gambar 6.8 Antarmuka pesan error jika ada kesalahan berkas 59
Gambar 6.9 Antarmuka pesan error jika ada kesalahan struktur xml berkas 60
Gambar 6.10 Antarmuka hasil perhitungan kopling secara garis besar 60

Gambar 6.11 Antarmuka hasil perhitungan kopling secara rinci 61
Gambar 6.12 Antarmuka pesan error jika data tidak ditemukan 61
Gambar 7.1 Simpul algoritme *method* CalculateInRank() 63
Gambar 7.2 *Flow graph method* CalculateInRank() 64
Gambar 7.3 Simpul algoritme *method* CalculateOutRank() 66
Gambar 7.4 *Flow graph method* CalculateOutRank()..... 67
Gambar 7.5 Simpul algoritme *method* CalculatePackageAbstraction() 69
Gambar 7.6 *Flow graph method* CalculatePackageAbstraction() 70
Gambar 7.7 Diagram hirarki pengujian integrasi method CalculateInRank()..... 72
Gambar 7.8 Simpul algoritme method CalculateInRank() 73
Gambar 7.9 *Flow graph method* CalculateInRank() 74
Gambar 7.10 Hasil pengujian integrasi kasus uji pertama *method* CalculateInRank() 75
Gambar 7.11 Hasil pengujian integrasi kasus uji ke-2 *method* CalculateInRank() 75
Gambar 7.12 Hasil pengujian integrasi kasus uji ke-3 *method* CalculateInRank() 76
Gambar 7.13 Hasil pengujian integrasi kasus uji ke-4 *method* CalculateInRank() 77
Gambar 7.14 Hasil pengujian integrasi kasus uji ke-5 *method* CalculateInRank() 77
Gambar 7.15 Diagram *package* studi kasus pengujian pertama 85
Gambar 7.16 Ilustrasi pemutusan relasi sirkular tingkat klas 86
Gambar 7.17 Ilustrasi pemutusan relasi sirkular tingkat *package* 86

DAFTAR TABEL

Tabel 4.1 Identifikasi aktor 24
Tabel 4.2 Kebutuhan fungsional sistem 24
Tabel 4.3 Kebutuhan non-fungsional sistem 25
Tabel 4.4 *Use case scenario* memasukkan berkas diagram *package* 27
Tabel 4.5 *Use case scenario parsing* XML 27
Tabel 4.6 *Use case scenario* menilai kopling 28
Tabel 5.1 Algoritme ChooseAndValidateFile 34
Tabel 5.2 Algoritme ParseXmlFile 35
Tabel 5.3 Algoritme FindPackagesFromDoc 35
Tabel 5.4 Algoritme FindRelationsFromDoc 36
Tabel 5.5 Algoritme AssignRelationsToPackages 36
Tabel 5.6 Algoritme Calculate 37
Tabel 5.7 Algoritme CalculatePackagesInstability 37
Tabel 5.8 Algoritme CalculatePackageInstability 37
Tabel 5.9 Algoritme CalculatePackagesAbstraction 38
Tabel 5.10 Algoritme CalculatePackageAbstraction 38
Tabel 5.11 Algoritme CalculatePackagesNormalizedDistance 38
Tabel 5.12 Algoritme CalculatePackagesNormalizedDistance 38
Tabel 5.13 Deskripsi komponen antarmuka memasukkan berkas 39
Tabel 5.14 Deskripsi komponen *file chooser* 40
Tabel 5.15 Deskripsi komponen dialog pesan error "*Unsupported file type*" 41
Tabel 5.16 Deksripsi komponen antarmuka *parsing* xml 42
Tabel 5.17 Deskripsi komponen dialog pesan error "*Required data not found*" 43
Tabel 5.18 Deskripsi komponen dialog pesan error "*File input error*" 44
Tabel 5.19 Deskripsi komponen dialog pesan error "*Xml document error*" 45
Tabel 5.20 Deskripsi komponen antarmuka halaman menilai kopling 46
Tabel 5.21 Deskripsi komponen halaman hasil perhitungan 47
Tabel 5.22 Deskripsi komponen dialog pesan error "*Cyclic relation found*" 48
Tabel 6.1 Spesifikasi perangkat keras komputer 49
Tabel 6.2 Spesifikasi perangkat lunak komputer 49
Tabel 6.3 Kode Sumber ChooseAndValidateFile 50

Tabel 6.4 Kode Sumber ParseXmlFile 51
Tabel 6.5 Kode Sumber FindPackagesFromDoc 52
Tabel 6.6 Kode Sumber FindRelationsFromDoc 52
Tabel 6.7 Kode Sumber AssignRelationsToPackages 52
Tabel 6.8 Kode Sumber Calculate 53
Tabel 6.9 Kode Sumber CalculatePackagesInstability 54
Tabel 6.10 Kode Sumber CalculatePackageInstability 54
Tabel 6.11 Kode Sumber CalculatePackagesAbstraction 54
Tabel 6.12 Kode Sumber CalculatePackageAbstraction 55
Tabel 6.13 Kode Sumber CalculatePackagesNormalizedDistance 55
Tabel 6.14 Kode Sumber CalculatePackageNormalizedDistance 55
Tabel 7.1 Hasil pengujian unit *method* CalculateInRank() 65
Tabel 7.2 Hasil pengujian unit *method* CalculateOutRank() 68
Tabel 7.3 Hasil pengujian unit *method* CalculatePackageAbstraction() 71
Tabel 7.4 Hasil pengujian integrasi *method* CalculateInRank() 78
Tabel 7.5 Pengujian memasukkan berkas xml diagram p*ackage* 80
Tabel 7.6 Pengujian memasukkan berkas diagram p*ackage* bukan xml 80
Tabel 7.7 Pengujian *parsing* XML ketika berhasil 81
Tabel 7.8 Pengujian *parsing* XML data tidak ditemukan 82
Tabel 7.9 Pengujian *parsing* XML kondisi berkas diagram bermasalah 82
Tabel 7.10 Pengujian *parsing* XML struktur XML berkas diagram bermasalah 82
Tabel 7.11 Pengujian menilai kopling 83
Tabel 7.12 Pengujian menilai kopling terdapat relasi sirkular 84
Tabel 7.13 Hasil pengujian validasi kakas bantu 84
Tabel 7.14 Hasil pengujian kebutuhan non-fungsional sistem 89
Tabel 7.15 Hasil penilaian kopling JDraw menggunakan kakas bantu 90

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Industri pengembangan perangkat lunak saat ini dituntut untuk lebih memperhatikan kualitas dari perangkat lunak agar perawatannya dapat lebih mudah dilakukan. Perawatan dilakukan agar perangkat lunaknya dapat mengimbangi kemajuan teknologi yang pesat (Almugrin, Albattah, dan Melton 2016). Perangkat lunak berskala besar cenderung menggunakan rancangan berorientasi objek karena dianggap lebih dekat dengan penalaran manusia terhadap objek dunia sehingga lebih mudah dipahami. Selain itu, perubahan terhadap suatu objek dapat dilakukan tanpa mempengaruhi objek lain sehingga perawatan sistem menjadi lebih mudah (Sommerville, 2011). Sistem berskala besar biasanya mengelompokkan klas-klasnya ke dalam *package* berdasarkan tugasnya.

Pada sistem berorientasi objek, terdapat interaksi antar beberapa objek untuk menyelesaikan suatu tugas atau fungsinya. Suatu objek memiliki karakteristik dan perilaku dengan nilai yang berbeda-beda. Perancangan arsitektur perangkat lunak berorientasi objek biasanya dilakukan menggunakan teknik *Object Oriented Design* (OOD). Selain itu, kualitas dari rancangan perangkat lunak dapat dinilai menggunakan diagram klas dan diagram *package* berdasarkan *Unified Modeling Language* (UML). Kualitas rancangan perangkat lunak berskala besar sebaiknya dinilai dari diagram *package* karena tingkatnya lebih tinggi sehingga cakupannya dapat lebih menyeluruh.

OOD memungkinkan penilaian kualitas rancangan berdasarkan relasi ketergantungan antar klas pada diagram klas maupun antar *package* pada diagram *package*, contohnya kopling. Rancangan yang baik dapat meningkatkan kualitas perangkat lunak yang dihasilkan, salah satunya dalam hal kemudahan perawatan (Almugrin, Albattah, dan Melton 2016). Hal ini menimbulkan banyaknya usulan metrik baru untuk mengukur kohesi, kopling, dan kompleksitas pada saat pengembangan (Fenton dan Pfleeger, 1998 disitasi dalam Almugrin, 2016). Apabila kesalahan perancangan baru terdeteksi setelah pengembangan selesai, dibutuhkan waktu dan biaya tambahan untuk memperbaiki kembali rancangan tersebut, yang juga akan berdampak pada berubahnya kode perangkat lunak (Boehm, 1981 disitasi dalam Almugrin, 2016, p.1).

Kopling merupakan metrik untuk menentukan banyaknya ketergantungan komponen suatu modul dengan komponen modul lain, baik langsung maupun tidak. Ketergantungan tidak langsung terbentuk akibat adanya relasi dengan modul lain yang juga memiliki relasi dengan modul lainnya. Besarnya relasi fungsional antar komponen dalam suatu modul ditentukan menggunakan metrik kohesi. Suatu modul yang baik memiliki kohesi yang tinggi dan kopling yang rendah. Rendahnya kopling menunjukkan tingginya kemandirian dari suatu klas ataupun *package* yang memudahkan pemahaman untuk perawatan modul-modul sistemnya (Briand, Daly, dan Wüst, 1999).

Almugrin dan Melton (2015a) mengusulkan sebuah metrik kopling dengan nama *indirect package coupling based on responsibility*. Metrik tersebut mengukur nilai kopling dari suatu *package* berdasarkan ketergantungannya terhadap *package* lain, baik secara langsung maupun tidak. Almugrin dan Melton (2015a) sudah membuktikan bahwa hasil yang diperoleh dengan metrik ini lebih baik dari rangkaian metrik usulan Martin (2003) karena dapat mengidentifikasi relasi langsung maupun tidak langsung antara suatu *package* dengan *package* lainnya. Dengan teridentifikasinya relasi tidak langsung, perubahan terhadap *package* yang akan menimbulkan efek domino dapat dicegah. Akan tetapi, menilai kopling menggunakan metrik tersebut secara manual membutuhkan waktu yang tidak sedikit.

Komputasi kopling *package* "event" dari diagram *package* perangkat lunak *open source* JDraw yang disederhanakan saja membutuhkan waktu sekitar 10 menit 28 detik. Padahal diagram *package* tersebut hanya berisi 3 *package*, yaitu event, gui, dan action. *Package* event memiliki 2 klas, yaitu klas *interface* EventConstants dan klas ChangeEvent. Package gui memiliki 5 klas, yaitu klas abstrak Tool, klas konkret ColourPickerTool, Dispatcher, PixelTool, dan ColourEditor.Sejauh ini belum ada kakas bantu penilaian kopling yang menggunakan metrik *indirect package coupling based on responsibility.* Oleh karena itu, adanya kakas bantu diperkirakan dapat memperkecil waktu yang dibutuhkan dan memudahkan dalam menilai kopling menggunakan metrik tersebut.

Berdasarkan permasalahan-permasalahan tersebut, diperlukan kakas bantu untuk menentukan kualitas rancangan perangkat lunak dengan menilai kopling *package* secara otomatis. Kopling *package* dapat dihitung berdasarkan diagram *package* yang di dalamnya sudah mendefinisikan klas-klas beserta relasi-relasinya. Kakas bantu menggunakan metrik *indirect package coupling based on responsibility* yang diusulkan oleh Almugrin dan Melton (2015a) sehingga dapat mengidentifikasi relasi langsung dan tidak langsung dalam perhitungannya. Oleh karena itu, kakas bantu ini diharapkan dapat meningkatkan efisiensi dalam penilaian kualitas rancangan perangkat lunak dengan hasil metrik yang akurat sehingga para pengembang dapat lebih berhati-hati dalam memodifikasi sistem pada fase perawatan.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dikemukakan, maka diperoleh rumusan masalah berikut:

1. Bagaimana hasil penerapan metrik *indirect package coupling based on responsibility* untuk menilai kopling *package*?
2. Bagaimana cara kakas bantu mengidentifikasi *package diagram* yang merupakan data untuk melakukan pengukuran nilai kopling?
3. Bagaimana cara menguji kakas bantu penilaian kopling yang menerapkan metrik *indirect package coupling based on responsibility*?

1.3 Tujuan

Tujuan penelitian:

1. Menerapkan metrik *Indirect Package Coupling Based on Responsibility* pada kakas bantu dalam menentukan nilai kopling *package* dalam suatu *package diagram*.
2. Menerapkan *Extensible Markup Language* (XML) *Parser* pada kakas bantu untuk mengidentifikasi relasi antar *package* dari suatu *package diagram* yang akan digunakan untuk pengukuran nilai kopling.
3. Melakukan pengujian terhadap kakas bantu yang menerapkan metrik *indirect package coupling based on responsibility* dan membandingkan efisiensi perhitungannya dengan manusia.

1.4 Manfaat

Dari hasil skripsi ini, dapat diperoleh manfaat berupa berkurangnya biaya dan waktu yang diperlukan dalam proses pengembangan perangkat lunak, yang secara luas akan membantu meningkatkan kualitas struktur perancangan *package* dan klas berorientasi objek. Sistem ini membantu pengembang agar dapat berhati-hati dalam melakukan perubahan ketika merawat perangkat lunak. Selain itu, kakas bantu dapat memudahkan manusia dan mempersingkat waktu untuk menilai kopling *package*.

1.5 Batasan masalah

Untuk mencegah terjadinya penyimpangan dari tujuan yang sudah direncanakan, proses pengerjaan skripsi ini diberi batasan-batasan sebagai berikut:

1. Aplikasi yang dibuat hanya dapat melakukan parsing terhadap dokumen diagram *package* dengan ekstensi ".xml".
2. Diagram *package* yang dimasukkan ke dalam kakas bantu harus mendefinisikan klas-klas beserta relasi-relasinya. Relasi-relasi ketergantungan tersebut tidak boleh membentuk ketergantungan sirkular.
3. Tools yang digunakan untuk membuat dokumen XML diagram *package* adalah *Visual Paradigm*.
4. Kualitas untuk setiap perangkat lunak dinilai berdasarkan satu diagram *package* saja.
5. Bahasa pemrograman yang digunakan adalah Java.

1.6 Sistematika Pembahasan

Adapun sistematika pembahasan yang digunakan pada penelitian ini untuk memberikan gambaran secara garis besar, yaitu:

BAB I : **PENDAHULUAN**

Bab ini menguraikan latar belakang skripsi, rumusan masalah, tujuan, manfaat, batasan masalah dan sistematika pembahasan.

BAB II : **TINJAUAN PUSTAKA**

Bab ini menjelaskan dan menguraikan tentang teori-teori dari penelitian terdahulu yang dikaji untuk mendasari penelitian ini.

BAB III : **METODE PENELITIAN**

Bab ini menjelaskan metode dan langkah kerja yang dilakukan dalam proses perancangan, analisis kebutuhan, dan implementasi pada pelaksanaan skripsi yang menjadi objek studi kasus skripsi.

BAB IV **: ANALISIS KEBUTUHAN**

Bab ini menguraikan tentang kebutuhan-kebutuhan yang diperlukan dalam sistem yang akan dibangun.

BAB V **: PERANCANGAN**

Bab ini menguraikan tentang perancangan sistem yang akan dibangun.

BAB VI : **IMPLEMENTASI**

Bab ini menguraikan implementasi dari dasar teori yang telah dikaji sesuai dengan perancangan sistem.

BAB VII : **PENGUJIAN**

Bab ini menguraikan proses pengujian terhadap sistem yang telah dibangun.

BAB VIII **: PENUTUP**

Bab ini menguraikan kesimpulan yang diperoleh dari perancangan dan pengujian perangkat lunak dalam skripsi ini serta saran untuk penelitian lanjutan.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini membahas dasar-dasar teori yang digunakan sebagai penunjang skripsi dengan judul "Kakas Bantu Penilaian Kopling Menggunakan Metrik *Indirect Package Coupling Based On Responsibility*". Dasar teori tersebut meliputi, rekayasa perangkat lunak, konsep kopling dan metrik yang digunakan, XML, konsep UML, dan pengujian perangkat lunak.

2.1 Kajian Pustaka

Kajian pustaka penelitian ini membahas penelitian sebelumnya yang berkaitan dengan metrik kopling, penilaian kualitas perangkat lunak, dan metrik berdasarkan *package*. Chidamber dan Kemerer (1991 dan 1994 disitasi dalam Almugrin dan Melton, 2015a, p.2) mengusulkan rangkaian metrik mereka dapat menilai kohesi, kopling, *inheritance*, dan kompleksitas. Metrik tersebut kemudian diperbaiki pada penelitian Izurieta dan Bieman (2008).

Dalam memodelkan sebuah sistem, hanya menggunakan klas-klas atau objek-objek tidak cukup untuk memperoleh rancangan yang *robust*, mudah dirawat dan dapat digunakan ulang (Almugrin dan Melton, 2015a). Untuk sistem skala besar, model abstraksi yang lebih tinggi seperti *package* atau yang setara diperlukan. Pembuatan *package* dalam sebuah sistem harus meningkatkan komunikasi dalam rancangannya dengan memisahkan elemen yang *reusable* dari *package non-reusable* untuk meningkatkan *reusability*. Namun, kemandirian dari *package* yang akan berinteraksi satu sama lain juga penting. Metrik-metrik kualitas rancangan *package* digunakan untuk menyeimbangkan *reusability* dan kemandirian sehingga suatu rancangan mudah untuk dirawat dan diuji.

Literatur tentang usulan metrik tingkat *package* tidak sebanyak metrik tingkat klas. Ducasse, Lanza dan Ponisio (2005) mengusulkan metrik untuk klasifikasi *package* berdasarkan struktur internalnya dan relasinya dengan seluruh sistem menggunakan diagram radar yang disebut *butterflies.* Ponisio (2006) mengusulkan sebuah metrik baru untuk mengukur kohesi suatu *package* bernama *Contextual Package Cohesion* (CPC). Martin (2003) menyarankan untuk menghindari *rigidity* dan *fragility* dalam *object-oriented design* dengan cara membuat klas konkret bergantung terhadap klas abstrak. Rangkaian metrik yang ia ajukan meliputi, *afferent coupling* (Ca), *efferent coupling* (Ce), *Instability* (I), *Number of Abstract Classes* (Na), *Number of Classes* (Nc), *Abstractness* (A), *distance from the main sequence* (D), dan *normalized distance from main sequence* (Dn). Almugrin et al. (2014) mengembangkan metrik tersebut dengan menambahkan faktor *responsibility*. Selanjutnya, Almugrin dan Melton (2015a) membuat pendekatan dalam mengelola pembuatan *package* berdasarkan ketergantungan antar *package* dan klas-klas, baik langsung maupun tidak yang disebut metrik *indirect package coupling based on responsibility*. Berdasarkan pengujian dan validasi oleh Almugrin ,Albattah , dan Melton (2016) metrik tersebut terbukti memudahkan perawatan perangkat lunak dalam mengetahui dampak dari modifikasi yang akan dilakukan sehingga dapat lebih berhati-hati.

2.2 Rekayasa Perangkat Lunak

Rekayasa perangkat lunak adalah disiplin ilmu yang mempelajari berbagai aspek perencanaan dan produksi perangkat lunak agar biaya dan waktu yang dibutuhkan untuk mencapai kualitas tertentu dapat lebih optimal. Tentunya, untuk mencapai hal tersebut dibutuhkan pendekatan yang sistematis dan terorganisir, serta kreativitas dalam pemilihan metode yang digunakan pada kondisi tertentu (Sommerville, 2011). Oleh karena itu, pembangunan kakas bantu penilaian kopling akan dibangun berdasarkan prinsip dan aturan rekayasa perangkat lunak agar memiliki kualitas yang optimal.

Pendekatan sistematis yang disebutkan sebelumnya disebut juga *software process*. *Software process* terdiri atas 4 kegiatan utama, yaitu:

1. *Software specification*, tahapan menentukan perangkat lunak yang akan dibangun beserta batasan-batasan operasinya.
2. *Software development*, tahapan merancang dan membangun perangkat lunak.
3. *Software validation*, tahapan pengujian perangkat lunak untuk menjamin kesesuaiannya dengan apa yang diinginkan pelanggan.
4. *Software evolution,* tahapan perubahan terhadap perangkat lunak untuk menyesuaikan dengan perubahan permintaan pelanggan dan pasar.

Rekayasa perangkat lunak dinilai penting karena alasan-alasan berikut:

1. Semakin tingginya ketergantungan terhadap sistem-sistem perangkat lunak yang maju sehingga dibutuhkan sistem yang reliabel.
2. Pembuatan perangkat lunak menggunakan metode dan teknik rekayasa perangkat lunak cenderung lebih murah dalam jangka panjang.

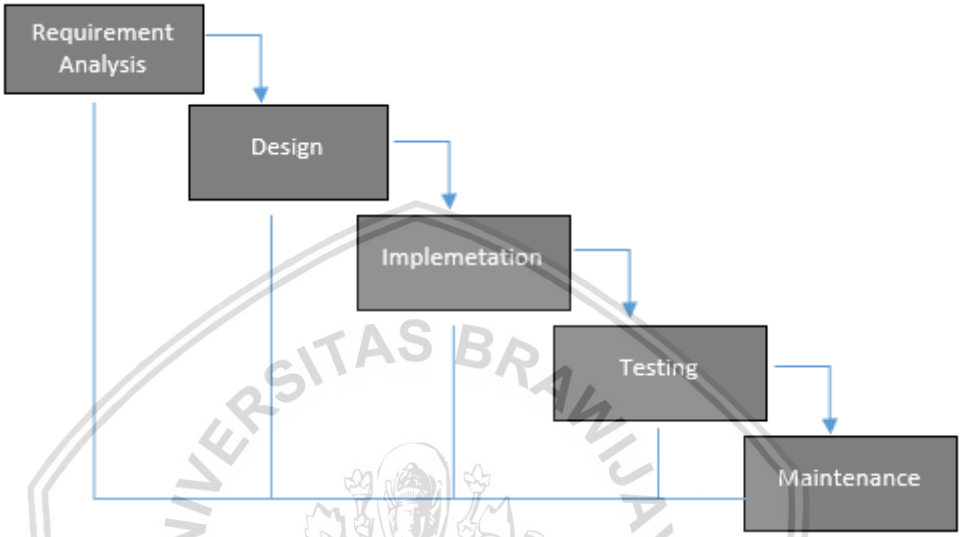
2.2.1 *System Development Life Cycle*

System Development Life Cycle (SDLC) adakah siklus pengembangan sistem untuk menggambarkan tahapan dalam membangun perangkat lunak. SDLC membantu memastikan sistem yang dibangun selesai tepat waktu dengan kualitas yang layak (Mishra dan Dubey, 2013). Kualitas perangkat lunak yang diperoleh jika menggunakan SDLC akan lebih baik, karena sesuai dengan permintaan pelanggan, terarah, tepat waktu dan hemat biaya. SDLC meningkatkan efisiensi dan efektivitas pengembangan sistem. Tahap-tahap umum SDLC, antara lain:

1. Analisis, tahapan analisa sistem yang akan dibangun dengan mencari tahu sebab dan akibat dari permasalahan-permasalahan sistem yang sudah ada.
2. Perancangan, tahapan menggali kebutuhan dari *stakeholder*, lalu memodelkannya menjadi rancangan sistem yang akan dibangun.
3. Penerapan, tahapan membangun perangkat lunak berdasarkan rancangan yang telah dibuat dan disetujui.
4. Perawatan, tahapan untuk melakukan perbaikan atas kesalahan-kesalahan sistem ketika digunakan dan untuk menyesuaikan dengan keadaan pasar.

2.2.2 Model *Waterfall*

Model *waterfall* merupakan salah satu jenis SDLC. Sebagaimana SDLC lainnya, model ini memenuhi 4 tahapan dasar yang diperlukan dalam membangun perangkat lunak yang sudah disebutkan pada bagian sebelumnya. Gambar 2.1 merupakan tahapan-tahapan pengembangan sistem dengan menggunakan model waterfall:



Gambar 2.1 Model *waterfall*

1. *Requirement Analysis*

   Menganalisis permasalahan untuk mendefinisikan kebutuhan sistem yang akan dibangun. Kegiatan ini biasanya dilakukan bersama dengan *stakeholder* untuk menciptakan sudut pandang lain terhadap permasalahan tersebut. Pada penelitian ini, kebutuhan utama sistem akan diperoleh dari analisis terhadap penelitian sebelumnya. Hasil dari *requirement analysis* akan digunakan pada tahap selanjutnya, yaitu *design*.

2. *Design*

   Membuat diagram-diagram rancangan sistem berdasarkan pernyataan kebutuhan yang diperoleh pada tahap *requirement analysis*. Diagram rancangan yang akan dibuat untuk kakas bantu ini meliputi, *sequence diagram* dan *class diagram*. Diagram ini memudahkan *programmer* dalam memberikan gambaran mengenai sistem untuk tahap implementasi. Oleh karena itulah, dilakukan perancangan sebelum membangun kakas bantu ini.

3. *Implementation*

   Membangun sistem sesuai dengan rancangan-rancangan yang telah dibuat pada fase *design.* Rancangan tersebut diterjemahkan ke dalam bahasa pemrograman yang nantinya dapat dipahami oleh mesin. Kakas bantu penilaian kopling akan dibangun menggunakan bahasa pemrograman java karena bersifat *platform independent*.

4. *Testing*

   Menguji sistem yang telah dibangun pada tahap sebelumnya. Pengujian ini dilakukan untuk mengetahui apakah sistem sudah benar dan sesuai dengan rancangan yang dibuat. Hasil dari pengujian ini menunjukkan apakah sistem sudah layak untuk digunakan dan apa saja kekurangannya.

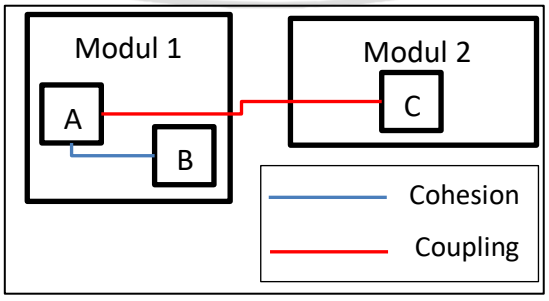
5. *Maintenance*

   Perawatan sistem yang telah dibangun agar tidak terjadi kesalahan sistem dan sistem tidak tertinggal jika dibandingkan sistem lain yang beredar. Tahapan ini memperbaiki sistem berdasarkan hasil pengujian pada tahap sebelumnya.

Model ini sederhana sehingga mudah dan cocok untuk sistem yang kebutuhannya dapat didefinisikan sepenuhnya sejak awal proyek perangkat lunak dan berskala kecil. Model ini dipilih karena kriteria-kriteria tersebut terpenuhi oleh kakas bantu penilaian kopling pada penelitian ini. Kekurangannya, karena sifatnya yang berurutan, model ini akan mengalami permasalahan apabila ada hambatan di suatu tahap, sehingga tahapan berikutnya tidak dapat dikerjakan dengan baik.

2.3 Kopling Antar Modul Perangkat Lunak

Kopling adalah suatu nilai yang menunjukkan seberapa besar ketergantungan suatu modul terhadap modul lain. Kopling dan kohesi digunakan sebagai parameter untuk menilai kualitas arsitektur perangkat lunak (Chowdhury dan Zulkernine, 2011). Penilaian tersebut dapat dilakukan pada berbagai fase pengembangan perangkat lunak (Fenton dan Pfleeger, 1998 disitasi dalam Almugrin, 2016). Seperti yang dijelaskan oleh English, Cahill, dan Buckley (2012), pada Gambar 2.2 ditunjukkan bahwa kohesi merupakan interaksi antar komponen yang ada di dalam sebuah modul, sedangkan kopling merupakan interaksi komponen antar modul. Modul-modul yang tidak berkaitan tujuannya dipisahkan agar derajat ketergantungan suatu modul berkurang. Nilai dari kopling inilah yang nantinya akan dihitung pada tahap perancangan menggunakan kakas bantu.



Gambar 2.2 Hubungan antara kohesi dan kopling

Suatu modul yang baik adalah modul dengan tingkat kohesi tinggi dan kopling rendah (English, Cahill, dan Buckley, 2012). Akan tetapi, tidak berarti perangkat lunak yang antar modulnya tidak ada kopling sama sekali dapat dikatakan rancangan yang baik. Kopling yang rendah berarti tidak banyaknya

ketergantungan antar modul, sehingga apabila dilakukan perubahan terhadap modul yang digantungi, tidak banyak modul lain yang terpengaruh. Penggunaan kopling dan kohesi dapat membantu mengembangkan sistem dengan lebih optimal dan memudahkan pekerjaan tiap modul dalam suatu sistem (Patidar, 2013). Nilai kopling dan kohesi tersebut dapat diukur menggunakan metrik-metrik perangkat lunak. Oleh karena itulah, dibutuhkan kakas bantu penilaian kopling menggunakan suatu metrik agar dapat mengoptimalkan kopling dari sistem.

Metrik perangkat lunak berorientasi objek adalah alat pengukuran kualitas perangkat lunak berorientasi objek. Metrik tersebut digunakan pada tahap perancangan atau implementasi sistem perangkat lunak (Almugrin dan Melton, 2015a). Beberapa contoh metrik perangkat lunak berorientasi objek tersebut disebutkan dalam penelitian oleh Bocco, Piattini, dan Calero (2005) . Salah satu cara metrik mengukur kualitas perangkat lunak adalah dengan mengukur nilai kopling suatu perangkat lunak.

Metrik kopling merupakan pengukur kualitas sistem berorientasi objek yang mengacu pada hubungan antar modul. Hubungan antar modul tersebut dapat dilihat pada rancangan *diagram* klas jika yang dinilai adalah ketergantungan antar klas dalam suatu *package*. Hubungan antar modul juga dapat dilihat ada rancangan *package diagram* jika yang dinilai adalah ketergantungan antar klas dari *package* berbeda. Ketergantungan tersebut dapat bersifat langsung ataupun tidak langsung. Kakas bantu ini menggunakan salah satu metrik kopling yaitu *Indirect package coupling based on responsibility* yang diusulkan oleh Almugrin dan Melton (2015a). Metrik tersebut mengukur kopling pada suatu *package* berdasarkan ketergantungan langsung maupun tidak langsung terhadap *package* lainnya.

2.3.1 Metrik *Indirect Package Coupling Based on Responsibility*

Almugrin dan Melton (2015a) membuat sebuah metrik bernama *Indirect package coupling based on responsibility* untuk mengukur kopling, baik yang bersifat langsung maupun tidak, pada tingkat *package* dari diagram UML. Selain kopling langsung, adanya kopling tidak langsung juga sangat berpengaruh pada arsitektur perangkat lunak berorientasi objek skala besar dalam hal perawatan. Hal tersebut disebabkan oleh adanya ketergantungan yang tidak terlihat secara langsung sehingga lebih sulit untuk dikendalikan jumlahnya. Hal ini mendasari penerapan metrik ini pada kakas bantu yang akan dibuat pada penelitian ini. Pengukuran kopling dengan metrik *indirect package coupling based on responsibiltiy* dilakukan dalam beberapa tahapan, antara lain:

1. *Dual Ranking Instability*
   Pada tahap ini dilakukan perhitungan terhadap ketidakstabilan (*instability*) *package*. Nilai ini menunjukkan seberapa besar pengaruh yang ditimbulkan jika *package* dimodifikasi berdasarkan banyaknya *package* yang bergantung padanya dan yang digantunginya. *Dual Ranking Instability* terdiri atas 2 metrik baru, yaitu InRank untuk menghitung *responsibility* dan OutRank untuk

menghitung *dependency*. Persamaan 2.0 digunakan untuk menghitung InRank dan persamaan 2.1 untuk menghitung OutRank:

$$InRank(A) = (1-d) + d\left(\frac{InRank(T_1)\times C(T_1)}{D(T_1)} + \ldots + \frac{InRank(T_n)\times C(T_n)}{D(T_n)}\right) \quad (2.0)$$

Keterangan:
InRank(A) : nilai *responsibility* dari *package* A
T_i : *package* yang bergantung pada *package* A
d : *damping factor* (*default*=0.85)
$C(T_n)$: jumlah klas pada *package* Ti yang bergantung pada *package* A
$D(T_n)$: jumlah klas pada *package* Ti yang bergantung pada *package* lain

$$OutRank\ (A) = (1-d) + d\ (Outrank(P_1) + \ldots + OutRank(P_n)) \quad (2.1)$$

Keterangan:
OutRank(A) : nilai *dependency* dari *package* A
P_i : *package* yang digantungi oleh *package* A
d : *damping factor (default=0.85)*

Selanjutnya nilai InRank dan OutRank tersebut dinormalisasi dengan persamaan 2.1 dan 2.2:

$$InRank(A) = \frac{InRanki(A) - Min}{(Max - Min)} \quad (2.1)$$

Keterangan:
InRank(A) : nilai hasil normalisasi dari nilai awal InRank *package* A
InRanki(A) : nilai awal InRank *package* A sebelum dinormalisasi
Min : nilai terkecil InRank yang belum ternomalisasi
Max : nilai terbesar InRank yang belum ternomalisasi

$$OutRank(A) = \frac{OutRanki(A) - Min}{(Max - Min)} \quad (2.2)$$

Keterangan:
OutRank(A) : nilai hasil normalisasi dari nilai awal OutRank *package* A
OutRanki(A) : nilai awal OutRank *package* A sebelum dinormalisasi
Min : nilai terkecil OutRank yang belum ternomalisasi
Max : nilai terbesar OutRank yang belum ternomalisasi

Setelah InRank dan OutRank dinormalisasi, dilakukan perhitungan *instability* menggunakan persamaan 2.3:

$$I(A) = \frac{OutRank(A)}{(OutRank(A) + InRank(A))} \quad (2.3)$$

Keterangan:
I(A) : nilai *instability* dari *package* A
OutRank(A) : nilai *dependency* dari *package* A
InRank(A) : nilai *responsibility* dari *package* A

2. *Package Abstraction Calculation*

Pada tahap ini dilakukan perhitungan tingkat keabstrakan (*abstraction*) suatu *package* berdasarkan klas-klas di dalamnya. Nilai ini menunjukkan seberapa sulit suatu *package* untuk dimodifikasi berdasarkan rasio total nilai *responsibility* klas abstrak dengan total nilai *responsibility* keseluruhan klasnya.Suatu *package* yang abstrak berarti tidak mudah dimodifikasi, sehingga lebih stabil bagi *package* lain yang menggantunginya.

Pertama dilakukan perhitungan InRank tingkat klas sama seperti sebelumnya yang dapat dilihat pada persamaan 2.4:

$$InRank\ (C) = (1-d) + d\left(\frac{InRank(T_1)}{S(T_1)} + \ldots + \frac{InRank(T_n)}{S(T_n)} \right) \quad (2.4)$$

Keterangan:

InRank(C) : nilai *responsibility* dari klas C

T_i : klas yang bergantung pada klas C

d : *damping factor* (*default*=0.85)

$S(T_i)$: jumlah klas yang digantungi klas Ti

Hasil yang diperoleh digunakan untuk perhitungan keabstrakan *package* pada persamaan 2.5 dengan range hasil [0,1]. Jika hasilnya 0, berarti semua klas bersifat konkret dan tidak ada klas bersifat abstrak. Jika hasilnya 1, berlaku sebaliknya.

$$Abs(P_i) = \frac{\sum_{A_j \in A} (InRank(A_i))}{\sum_{C_j \in C} (InRank(C_j))} \quad (2.5)$$

Keterangan

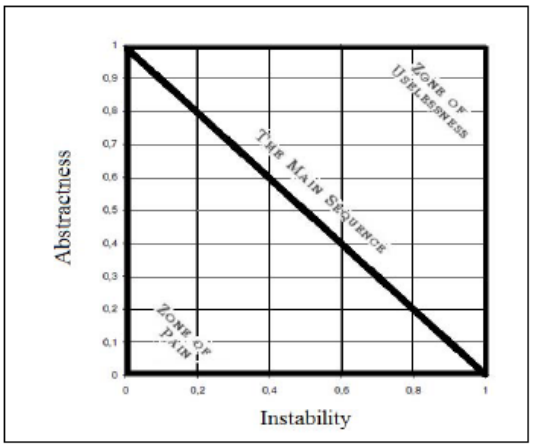
$Abs(P_i)$: nilai *abstractness* dari *package* P_i

A_i : klas abstrak pada *package* P_i

C_j : klas pada *package* P_i

3. *Package's Normalized Distance*

Tahapan terakhir dari metrik ini adalah perhitungan jarak ternormalisasi suatu *package* (*package's normalized distance*) berdasarkan ketidakstabilan dan keabstrakannya. Nilai ini menunjukkan jarak nilai kopling *package* dari titik idealnya dengan rentang nilai 0-1. Semakin besar jaraknya, semakin tidak ideal kopling *package* tersebut.Gambar 2.3 menunjukkan adanya hubungan berbanding terbalik antara *instability* (I) sebagai x-axis dan *abstractness* (A) sebagai y-axis dalam sebuah graf (Martin, 2003).



Gambar 2.3 Graf hubungan antara *Instability dan Abstractness*

Sumber: Almugrin (2015a)

Package yang stabil haruslah bersifat abstrak agar tidak mudah diubah karena ada *package* lain bergantung padanya, dan berlaku sebaliknya untuk *package* yang tidak stabil. Pada persamaan 2.6, dapat terlihat bahwa perhitungan *package's normalized distance* yang digunakan masih sama dengan usulan Martin (2003), yaitu:

$$Dn = |(A + I - 1)| \quad (2.6)$$

Keterangan:
Dn : Nilai *package's normalized distance*
A : Nilai *abstractness* dari *package*
I : Nilai *instability* dari *package*

2.4 Extensible Markup Language

Extensible Markup Language (XML) adalah sebuah bahasa markup untuk mendefinisikan dokumen agar dapat dimengerti oleh manusia dan mesin. Tujuan dibuatnya XML adalah untuk menunjukkan kesederhanaan, kegunaan, dan kesamaan melalui internet. Selain untuk dokumen, XML juga umumnya digunakan untuk menggambarkan struktur data yang bersifat bebas seperti *web service* (Fennel, 2013). Kelebihan dari sebuah dokumen XML adalah adanya pemisahan antara isi, struktur, dan bentuk dokumen sehingga dapat ditangani secara terpisah (Mafazi, 2011). Hal tersebut menyebabkan XML dikatakan sebagai perpaduan antara pemrosesan dokumen dan basis data (Kroenke et al., 2012) sehingga cocok digunakan sebagai dokumen rancangan *package diagram* yang akan diproses pada penelitian ini. Dokumen XML dari *package diagram* pada penelitian ini diperoleh dengan melakukan *export* pada rancangan yang dibuat menggunakan perangkat lunak *visual paradigm*.

Dokumen XML diawali dengan "`<?xml version="1.0" encoding="UTF-8">`" untuk mendefinisikan versi sekaligus menyatakan bahwa dokumen tersebut adalah XML. Sebuah dokumen XML terdiri atas tiga komponen penting, yaitu

prolog, content, dan epilog. Pengkodean pada XML nantinya akan diproses oleh browser untuk mengambil kumpulan karakter yang diperlukan. Komponen-komponen pada dokumen XML, antara lain:

1. **Komentar.** Komentar digunakan untuk menambahkan penjelasan atas baris-baris kode sumber yang dibuat. Penanda yang digunakan sebagai pembuka adalah `<!--` dan `-->` sebagai penutup sehingga apapun yang ditulis diantaranya tidak akan diproses oleh pemroses XML (Mafazi, 2011).
2. **Elemen *Root*.** Elemen ini adalah komponen utama dokumen XML dan memiliki posisi tertinggi berdasarkan struktur pohon. Oleh karena itu, elemen *root* disebut juga sebagai *parent* dari seluruh elemen *child*. Ketentuan-ketentuan dokumen XML adalah sebagai berikut (Mafazi, 2011):
    a. Dibuka dengan tag "<" dan ditutup dengan ">".
    b. Huruf besar dan kecil dianggap berbeda pada penamaan elemen dan atribut (*case-sensitive)*.
    c. Penamaan elemen harus diawali dengan huruf dan tidak boleh terdapat karakter "@, #, $, %, ^, (,), +, ?, =, ;, * " ataupun kata 'xml' itu sendiri di dalamnya.
    d. Hanya boleh terdapat satu elemen *root,* elemen selanjutnya dapat ditambahkan dengan cara bersarang.
    e. Penulisan karakter <, >, &, dan ' dapat dilakukan dengan menggunakan *entityreference*. Untuk menuliskan karakter &, maka yang dituliskan adalah *entityreference* dari karakter tersebut, yang mana adalah `&amp;`.

Dokumen XML yang diinputkan ke dalam kakas bantu ini akan diolah terlebih dahulu ke dalam bentuk data yang dapat digunakan oleh sistem melalui proses *parsing*. Proses *parsing* dilakukan menggunakan *library* Document Object Model (DOM).

2.4.2 *Document Object Model* (DOM)

Document Object Model (DOM) adalah *library* untuk melakukan *parsing* dokumen HTML atau XML yang valid. DOM mengolah dokumen tersebut menjadi data yang dapat dikelola (Wood, 2000). Hal ini memungkinkan kakas bantu untuk melakukan perhitungan kopling terhadap data diagram *package* pada dokumen XML. Kakas bantu ini menggunakan DOM karena cara kerjanya yang bukan *event-based* cocok dengan kakas bantu yang hanya melakukan satu kali *parsing* secara menyeluruh di awal.

DOM bekerja dengan membaca teks elemen atau atribut yang ada pada dokumen. Elemen-elemen dan atribut-atribut yang akan diolah jadi data antara lain:

1. Elemen <Package> sebagai *package*, atributnya meliputi "Id" sebagai kode unik id *package* dan "Name" sebagai nama *package*,
2. Elemen <Class> sebagai klas, atributnya meliputi "Id" sebagai kode unik id klas, "Name" sebagai nama klas, dan "Abstract" sebagai tipe dari klas.

3. Elemen <Stereotype> milik <Class>, atributnya meliputi “Name” sebagai tipe dari klas. Biasanya atribut tersebut bernilai “interface”.
4. Elemen <Dependency> sebagai relasi, atributnya meliputi “Id” sebagai kode unik id relasi, “Name” sebagai nama relasi, dan “Type” sebagai tipe dari relasi. “From” sebagai id dari klas/*package* yang bergantung dalam relasi, “To” sebagai id dari klas/*package* yang digantungi dalam relasi.
5. Elemen <Association> sebagai relasi, atributnya meliputi “Id” sebagai kode unik id relasi, “Name” sebagai nama relasi, dan “EndRelationshipFromMetaModelElement” sebagai id dari klas yang bergantung dalam relasi, “EndRelationshipToMetaModelElement” sebagai id dari klas/*package* yang digantungi dalam relasi. Tipe relasinya diperoleh dari *child* dari elemennya, yaitu <AssociationEnd> dengan atributnya “AggregationKind”. Apabila nilai “AggregationKind” adalah “Composited”, tipe relasinya adalah komposisi. Apabila bernilai “Aggregation” tipenya agregasi, sedangkan ketika bernilai “None” relasinya bertipe asosiasi.
6. Elemen <Generalization> sebagai relasi, atributnya meliputi “Id” sebagai kode unik id relasi, “Name” sebagai nama relasi. Asal dan tujuan relasi ini berlawanan dengan arah ketergantungannya sehingga “From” sebagai id dari klas/*package* yang digantungi dalam relasi, “To” sebagai id dari klas/*package* yang bergantung dalam relasi.
7. Elemen <Realization> sebagai relasi, atributnya meliputi “Id” sebagai kode unik id relasi, “Name” sebagai nama relasi. Asal dan tujuan relasi ini berlawanan dengan arah ketergantungannya sehingga “From” sebagai id dari klas/*package* yang digantungi dalam relasi, “To” sebagai id dari klas/*package* yang bergantung dalam relasi.

Elemen-elemen dan atribut-atribut tersebut dapat diakses dari dokumen XML menggunakan *method* dari *library* DOM yang digunakan oleh kakas bantu. Method-method tersebut meliputi:

1. `normalize()` adalah method untuk menyimpan sekaligus melakukan *reload* dokumen DOM dalam bentuk normalnya. Bentuk normal tersebut diperoleh dengan melakukan normalisasi `node` teks pada dokumen DOM hasil *parsing* berkas oleh *library* DocumentBuilder Java. Selain itu method ini memastikan tidak ada kesalahan pada `node` di dokumen DOM.
2. `getElementsByTagName()` adalah method untuk mengambil data elemen-elemen berdasarkan `string` nama tagnya sebagai parameter. Method ini mengembalikan `NodeList` berisi kumpulan elemen yang dicari nama tagnya. `Node` yang diperoleh di konversi menjadi elemen agar bisa diakses atributnya.
3. `getAttribute()` adalah method untuk mengambil nilai dari atribut tertentu suatu elemen. Method ini melakukan pencarian berdasarkan `string` nama atribut sebagai parameter dan mengembalikan nilai `string` dari atribut tersebut.
4. `getTagName()` adalah method untuk mengambil nama dari tag suatu elemen. Method ini mengembalikan nilai string dari nama tag elemen tersebut.

2.5 Unified Modelling Language

Unified Modelling Language (UML) adalah sebuah bahasa visualisasi untuk menspesifikasikan, membangun, dan pembuatan dokumentasi dari pengembangan perangkat lunak berorientasi objek. UML dapat dihubungkan langsung ke dalam banyak bahasa pemrograman bahkan basis data beorientasi objek. Diagram UML dikelompokkan ke dalam 3 jenis (Sommerville, 2011), yaitu:

1. ***Structural diagrams***. Diagram-diagram yang bertujuan untuk menggambarkan struktur elemen termasuk dalam kategori ini. Diagram-diagram tersebut meliputi, diagram klas, diagram objek, diagram komponen, *deployment diagram,* diagram struktur komposit. Diagram struktural yang akan digunakan untuk menggambarkan rancangan kakas bantu perhitungan kopling pada penelitian ini adalah diagram klas.
2. ***Behavior diagrams***. Diagram-diagram yang bertujuan untuk menggambarkan ciri-ciri sistem berupa *behavior/function/method* atau proses bisnis termasuk dalam kategori ini. Diagram-diagram tersebut meliputi, *use case diagram, activity diagram,* dan *state machine diagram*. Diagram *behavior* yang digunakan untuk menggambarkan perilaku kakas bantu perhitungan kopling pada penelitian ini adalah adalah *use case diagram*.
3. ***Interaction diagrams***. Diagram-diagram yang bertujuan untuk menggambarkan interaksi objek termasuk dalam kategori ini. Diagram-diagram tersebut meliputi, *communication diagram, interaction overview diagram, sequence diagram, timing diagram*. Interaksi antar komponen dalam kakas bantu penilaian kopling akan digambarkan melalui rancangan *sequence diagram*.

2.6 Pengujian Perangkat Lunak

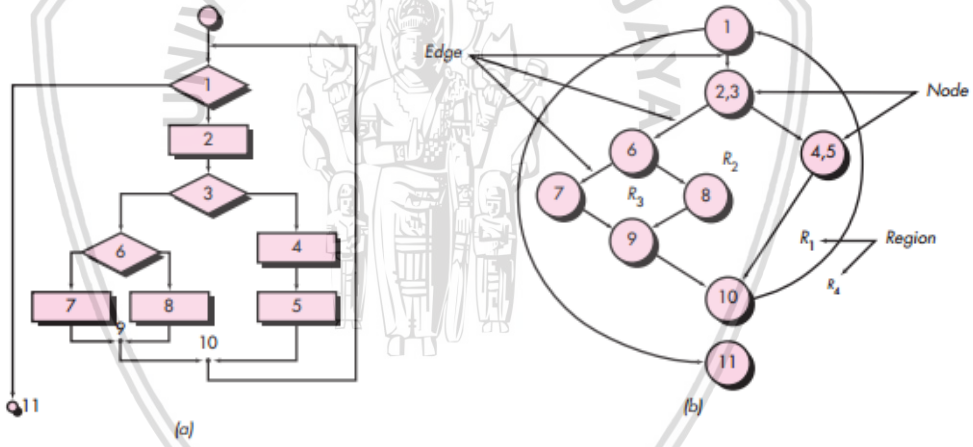
Pengujian perangkat lunak merupakan tahapan dari pengembangan perangkat lunak untuk mengetahui apakah perangkat lunak sudah benar dan sesuai dengan rancangan yang dibuat. Selain itu, dengan pengujian perangkat lunak, pengembang dapat memahami risiko permasalahan perangkat lunak dan manajemen penanganannya. Tujuan dilakukannya pengujian suatu perangkat lunak adalah mencari dan mengevaluasi kesalahan yang terjadi. Sehingga dapat dikatakan bahwa pengujian yang baik ialah pengujian yang dapat menemukan kesalahan yang belum diketahui sebelumnya sebanyak mungkin untuk memastikan bahwa sistem yang akan diluncurkan sudah layak. Maka dari itu, akan dilakukan pengujian terhadap kakas bantu penilaian kopling ini.

Kasus uji (*test case*) adalah kasus pengujian yang memiliki kemungkinan tinggi dalam menemukan kesalahan yang sebelumnya tidak ditemukan. Sebelum mulai melakukan pengujian, perlu dibuat rancangan *test case* agar proses pengujian lebih efisien dari segi waktu dan usaha (Rouf, 2012). Desain tersebut dapat dilakukan dengan berbagai metode seperti *white box testing, black box testing* berdasarkan objek yang diuji.

2.6.1 *White Box Testing*

Pengujian kakas bantu ini dilakukan dengan menggunakan teknik *white box testing*. *White box testing* bertujuan untuk memberikan hasil uji berdasarkan cara kerja dan struktur internal dari perangkat lunak. Oleh karena itu, teknik ini cocok untuk pengujian unit dan integrasi. Teknik *white box testing* bertujuan untuk memastikan kakas bantu penilaian kopling yang dibangun sudah layak dari sisi internalnya. Teknik dari *white box testing* yang akan digunakan adalah teknik *basis path testing*.

Basis path testing memungkinkan perancang pengujian untuk mengukur logika *cyclomatic complexity* dari rancangan diagram alir. Teknik ini dipilih untuk menguji kakas bantu karena dapat mencakup semua lingkup pengujian sistem, antara lain *function coverage, branch coverage, statement coverage,* dan *condition coverage*. Diagram alir dijadikan sebagai acuan untuk menentukan prosedur dasar dari eksekusinya. *Cyclomatic complexity* adalah metrik untuk mengukur tingkat kompleksitas suatu sistem. Nilai *cyclomatic complexity* yang diperoleh digunakan untuk menentukan jalur *independent* dalam *basis set* suatu sistem dan memberi batasan banyaknya suatu pengujian yang diperlukan. Kompleksitas suatu sistem dihitung dengan cara yang ditunjukkan pada Gambar 2.4:



Gambar 2.4 Transformasi *Flowchart* (a) ke *Flowgraph* (b)

Sumber: Pressman (2010)

1. Jumlah *region* dari grafik alir sesuai dengan *cyclomatic complexity*.
2. *Cyclomatic complexity* untuk grafik alir G, *V(G)* dengan jumlah edge *E* dan jumlah simpul *N*, diperoleh melalui persamaan 2.7 berikut:

$$V(G) = E - N + 2 \quad (2.7)$$

3. *Cyclomatic complexity V(G),* dimana representasi predikat node adalah P, maka:

$$V(G) = P + 1 \quad (2.8)$$

Nilai *cyclomatic complexity* menunjukkan jumlah kasus uji yang perlu dilakukan. Kakas bantu ini diuji menggunakan kasus uji tersebut dengan bantuan *library* dari Java, yaitu JUnit.

2.6.1.1 JUnit

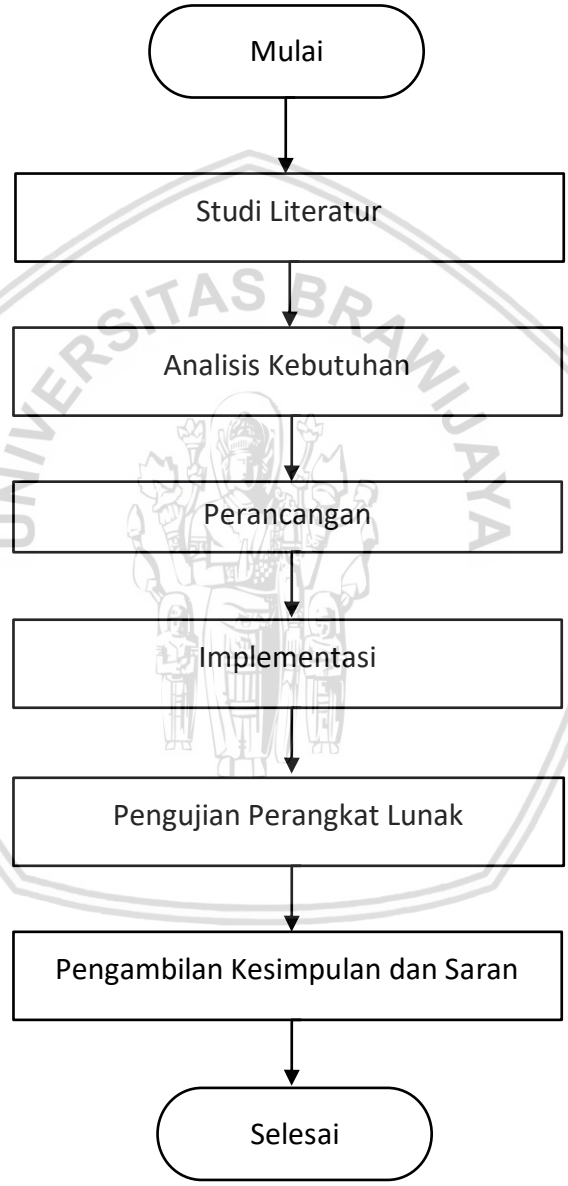
JUnit merupakan *library* yang disediakan oleh Java untuk melakukan pengujian terhadap sistem yang dibangun menggunakan bahasa pemrograman Java. Kasus uji tersebut diujikan terhadap kode sumber *method* yang bersangkutan melalui suatu klas uji. Beberapa kode sumber *method* pada kakas bantu ini diuji melalui klas uji yang secara otomatis dibuat oleh JUnit. Pengujian tersebut dilakukan dengan menginisiasi nilai dari variabel-variabel yang diperlukan dalam *method* uji pada klas uji sesuai dengan kasus uji yang ingin diujikan. Hasil yang diperoleh selanjutnya dibandingkan dengan hasil yang diharapkan. Jika sesuai, artinya *method* tersebut bekerja dengan baik.

2.6.2 *Black Box Testing*

Pengujian ini dilakukan untuk mengetahui dan memahami semua fungsi yang berjalan dalam suatu perangkat lunak agar nantinya dapat dievaluasi kesesuaiannya dengan kebutuhan fungsional yang didefinisikan. Oleh karena itu, *black box testing* cocok untuk digunakan untuk pengujian validasi terhadap kebutuhan sistem. Pengujian validasi ini bertujuan untuk menguji apakah keluaran dari sistem sesuai dengan kebutuhan pengguna. Keberhasilan pengujian validasi diukur dengan seberapa banyak fungsi yang berhasil berjalan sesuai kebutuhan.

BAB 3 METODOLOGI

Pada bab ini akan dijelaskan perencanaan langkah-langkah yang akan dilakukan untuk menyelesaikan skripsi ini yang berjudul "*Kakas Bantu Penilaian Kopling menggunakan Metrik Indirect Package Coupling Based on Responsibility*". Langkah-langkah yang diambil antara lain, studi literatur, analisis kebutuhan, perancangan, implementasi, pengujian perangkat lunak, sekaligus pengambilan kesimpulan dan saran pengembangan selanjutnya.



Gambar 3.1 Diagram alur penelitian

Pada Gambar 3.1 ditunjukkan bahwa penelitian ini diawali dengan melakukan studi literatur terhadap topik-topik yang berkaitan dengan judul penelitian ini, yakni *"Kakas Bantu Penilaian Kopling Menggunakan Metrik Indirect Package Coupling Based on Responsibility"*. Selanjutnya dilakukan analisis terhadap

kebutuhan-kebutuhan yang harus dipenuhi oleh kakas bantu yang akan dibangun. Kebutuhan-kebutuhan tersebut kemudian digunakan untuk menentukan rancangan kakas bantu. Berdasarkan rancangan yang dihasilkan, dilakukan implementasi kakas bantu menggunakan bahasa pemrograman Java. Setelah itu dilakukan pengujian terhadap kakas bantu untuk memastikan bahwa kakas bantu memenuhi kebutuhan. Terakhir dilakukan pengambilan keputusan sekaligus pemberian saran untuk penelitian lanjutan.

3.1 Studi Literatur

Studi literatur adalah tahap penyusunan dasar teori yang relevan untuk digunakan sebagai penunjang skripsi ini. Literatur yang digunakan bersumber dari berbagai buku, jurnal ilmiah, internet, dan lain sebagainya. Dasar teori disusun setelah mendapatkan referensi dan materi pendukung yang relevan dengan skripsi ini. Referensi pendukung yang relevan tersebut, antara lain:

1. Rekayasa Perangkat Lunak
    a. *System Development Life Cycle* (SDLC)
    b. Model *Waterfall*
2. Konsep Kopling
3. *Extensible Markup Language* (XML)
4. *Unified Modelling Language* (UML)
5. Pengujian Perangkat Lunak
    a. White Box Testing
    b. Black Box Testing

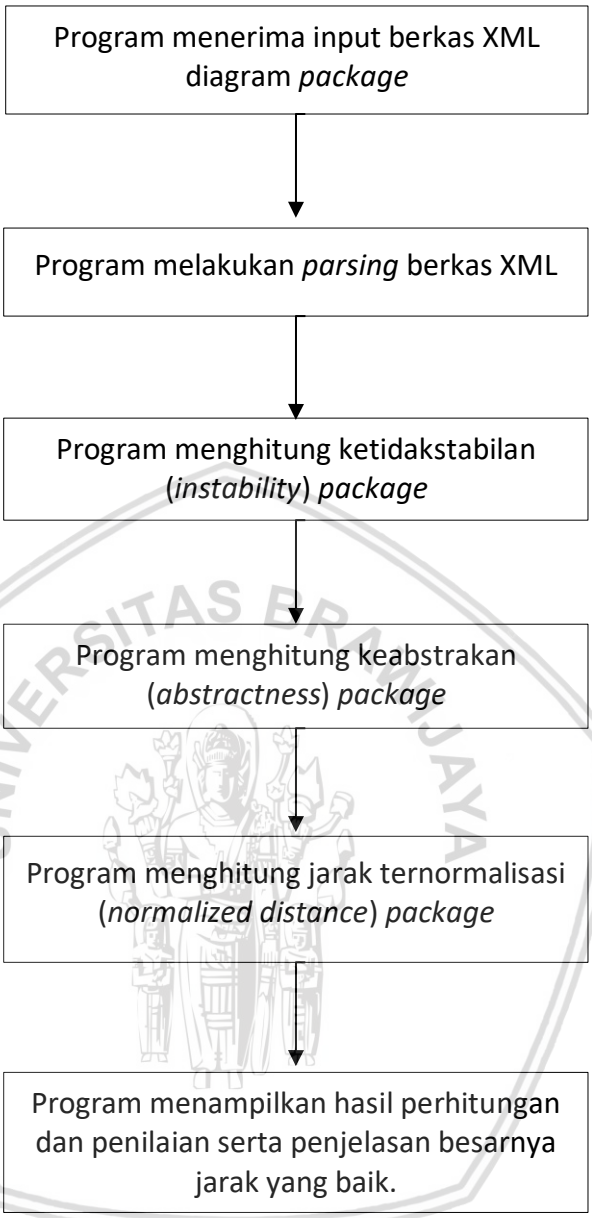
3.2 Analisis Kebutuhan

Pada tahap ini dilakukan pemahaman kebutuhan yang diperlukan untuk merancang suatu sistem. Fase untuk mendefinisikan dan menspesifikasikan kebutuhan ini memiliki pengaruh yang besar terhadap keberhasilan sistem. Hal ini karena apabila terjadi kesalahan di tahap ini, maka kesalahan tersebut akan berlanjut dan menimbulkan masalah besar di tahap-tahap akhir.

Kakas bantu akan dianalisis kebutuhannya menggunakan OOAD dengan UML sebagai model kebutuhan sistemnya. Sistem ini nantinya akan membutuhkan *package diagram* yang di dalamnya sudah dilengkapi dengan *class diagram* dengan ekstensi ".xml" untuk menentukan tingkat kopling dari rancangan perangkat lunak.

3.3 Perancangan

Perancangan dilakukan dengan mengacu pada hasil analisis kebutuhan. Hasil analisis kebutuhan akan ditransformasikan ke dalam bentuk bahasa visual berupa rancangan. Hal ini dilakukan dengan menggunakan konsep OOAD sebagai dasarnya dengan pemodelan UML. Gambar 3.2 menunjukkan cara kerja kakas bantu yang dimulai dari menerima input berkas XML.



Gambar 3.2 Cara kerja program

Dokumen *package diagram* yang akan dihitung koplingnya di dalamnya harus terdapat acuan terhadap *class diagram*. Sebelum dimasukkan ke program untuk diperiksa, dokumen *package diagram* tersebut diubah ke dalam bentuk dokumen XML terlebih dulu. Dari dokumen XML yang dimasukkan, program akan melakukan parsing untuk memperoleh struktur datanya. Berdasarkan struktur data yang diperoleh, sistem akan melakukan kalkulasi menggunakan metrik *indirect package coupling based on responsibility*.

Tahapan-tahapan dari metrik tersebut secara umum, antara lain perhitungan nilai *instability* dari *package*, nilai *abstractness* dari *package*, dan *normalized distance* dari *package*. Perhitungan *instability* dilakukan dengan menghitung *responsibility* (InRank) dan *dependency* (OutRank) dari *package* terlebih dahulu.

Hasil dari perhitungan tersebut dimasukkan kedalam rumus perhitungan *instability*. Perhitungan *abstractness* dari *package* dilakukan dengan menghitung *responsibility* (InRank) klas. Selanjutnya, total InRank dari klas abstrak/*interface* dan total InRank dari seluruh klas dalam *package* dimasukkan kedalam rumus perhitungan *abstractness*. Nilai *instability* dan *abstractness* tersebut dimasukkan ke dalam persamaan untuk menghitung nilai *normalized distance* dari *package.*

3.4 Implementasi

Implementasi perangkat lunak adalah tahapan untuk merealisasikan rancangan yang sudah dibuat. Bahasa pemrograman yang digunakan untuk mengimplementasi perankat lunak pada skripsi ini adalah *Java*. Sebelum masuk ke tahap pembuatan sistem, tentunya dilakukan spesifikasi lingkungan pengembangan, spesifikasi perangkat keras dan lunak, serta batasan-batasan implementasi.

3.5 Pengujian Perangkat Lunak

Pengujian perangkat lunak dilakukan untuk membuktikan bahwa sistem ini mampu bekerja sesuai dengan kebutuhan yang telah didefinisikan. Pengujian yang akan dilakukan adalah pengujian unit dan pengujian validasi. Pengujian unit dilakukan dengan metode *white box testing* menggunakan teknik *basis path*. Pengujian validasi dilakukan dengan metode *black box testing*. Selain itu, pengujian performa juga dilakukan untuk membuktikan bahwa sistem dapat menghemat waktu yang dibutuhkan dalam menilai kopling dibandingkan perhitungan manual. Dengan pengujian ini diharapkan mengidentifikasi kesalahan-kesalahan sistem dan memberikan kesimpulan dari tahapan pengembangan yang dilakukan.

3.6 Pengambilan Kesimpulan dan Saran

Pengambilan kesimpulan dan saran dilakukan setelah semua tahapan sebelumnya dalam skripsi ini selesai. Dari hasil perancangan dan pengujian dapat diperoleh kesimpulan. Saran akan didapatkan setelah mengevaluasi kesalahan yang terdapat pada skripsi ini. Saran yang diberikan bertujuan untuk menyempurnakan hasil tulisan serta masukan terhadap pengembangan perangkat lunak ini nantinya.

BAB 4 ANALISIS KEBUTUHAN

Bab ini membahas gambaran umum sistem dan analisis kebutuhan dari sistem kakas bantu kakas bantu penilaian kopling menggunakan metrik *indirect package coupling based on responsibility.* Gambaran umum ini akan memberikan acuan garis besar untuk analisis kebutuhan sistem yang akan dibangun. Analisis kebutuhan mendefinisikan kebutuhan yang harus dipenuhi oleh sistem agar dapat dikatakan berhasil.

Metode analisis yang digunakan pada penelitian ini adalah *Object Oriented Analysis (OOA)* dengan bahasa pemodelan UML. Analisis diawali dengan melakukan identifikasi aktor yang akan menggunakan sistem. Lalu dilanjutkan dengan analisis kebutuhan fungsional dan non-fungsional sistem. Kebutuhan yang telah didefinisikan tadi dimodelkan menggunakan *use case diagram* untuk menggambarkan kebutuhan fungsional sistem. Lalu tahapan yang dilakukan pada setiap *use case* akan dijelaskan pada *use case scenario*. *Use case diagram* dan *use case scenario* inilah yang nantinya akan digunakan sebagai acuan untuk melakukan perancangan sistem.

4.1 Gambaran Umum Sistem

Deskripsi umum dan lingkungan sistem kakas bantu penilaian kopling menggunakan metrik *indirect package coupling based on responsibility* akan dibahas pada bagian ini.

4.1.1 Deskripsi Umum Sistem

Sistem kakas bantu penilaian kopling menggunakan metrik *indirect package coupling based on responsibility* ini dapat membantu pengembang perangkat lunak dalam hal meningkatkan kualitas sistem berdasarkan nilai kopling. Data yang digunakan sebagai masukan untuk sistem ini adalah berkas rancangan diagram *package* dengan ekstensi XML. Dari berkas rancangan tersebut sistem akan menilai kopling secara otomatis sehingga pengguna dapat menilai kualitas dari rancangan perangkat lunak. Selain itu, dengan menilai kualitas diagram *package* yang merupakan rancangan tingkat tinggi, penilaian yang dihasilkan bersifat lebih menyeluruh.

Sistem akan mulai bekerja dengan melakukan *parsing* terhadap berkas XML yang telah dimasukkan oleh pengguna. *Parsing* dilakukan untuk memperoleh data tentang *package,* meliputi nama *package,* relasi, nama seluruh klas yang ada di dalamnya, dan tipe klas. Dengan data relasi antar *package* akan diperoleh nilai *responsibility* dan *dependency* untuk menghasilkan nilai *dual ranking instability* dari masing-masing *package*. Selanjutnya dilakukan perhitungan untuk memperoleh nilai *abstraction* dari setiap *package* menggunakan data tipe klas dan hubungan antar klas yang ada di dalam masing-masing *package.* Terakhir, sistem akan menghitung *normalized distance* tiap package menggunakan nilai *dual ranking instability* dan *package abstraction.* Nilai-nilai tersebut merupakan

keluaran utama dari sistem ini yang menunjukkan jarak kopling dengan titik idealnya, artinya semakin besar nilainya semakin buruk kualitas suatu package. Keluaran yang akan ditampilkan sistem, antara lain nama package, *dual ranking instability, package abstraction,* dan *package's normalized distance*.

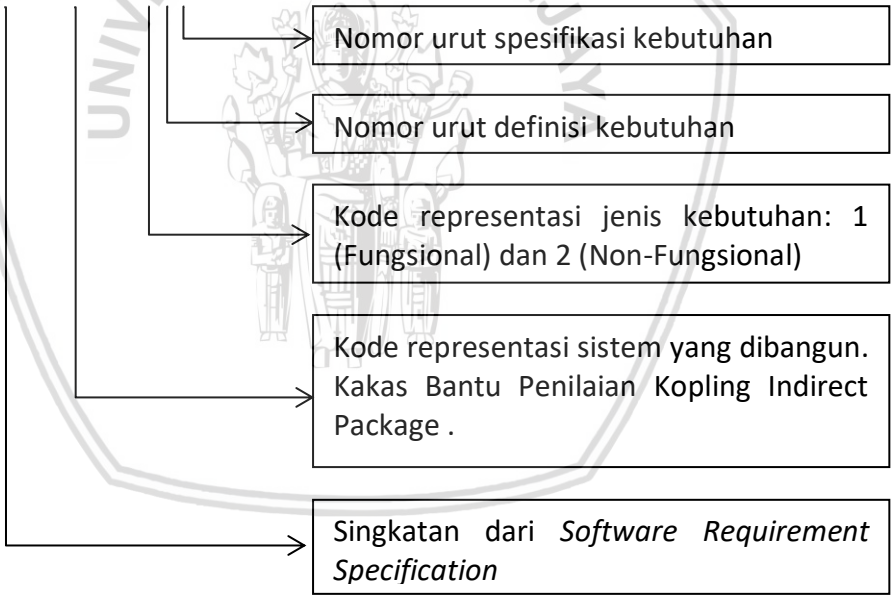
4.1.2 Lingkungan Sistem

Kakas bantu penilaian kopling menggunakan metrik *indirect package coupling based on responsibility* ini membutuhkan lingkungan untuk berjalan. Sistem ini dibuat berbasis *java* sehingga menggunakan *Java Runtime Environment (JRE)* dan Netbeans IDE versi 8.0.2. Selain itu, kompilasi sistem membutuhkan minimal *Java Development Kit (JDK)* versi 7.

4.2 Analisis Kebutuhan

Kebutuhan dibedakan menjadi 2 jenis, yaitu kebutuhan fungsional dan non-fungsional. Setiap kebutuhan memiliki kode yang berbeda berdasarkan aturan yang penomoran yang ditentukan. Berikut adalah aturan penomoran untuk kebutuhan sistem yang digunakan:

Kode: SRS_KBPKIP_XYZ



Analisis kebutuhan sistem kakas bantu perhitungan kopling menggunakan metrik *indirect package coupling based on responsibility* ini terdiri atas identifikasi aktor, analisis kebutuhan fungsional dan non-fungsional, pemodelan *use case diagram* dan *use case scenario*.

4.2.1 Identifikasi Aktor

Tabel 4.1 menjelaskan siapa saja yang terlibat dalam menjalankan sistem beserta deskripsinya.

Tabel 4.1 Identifikasi aktor

Aktor	Deskripsi
Pengguna	Pengguna adalah aktor yang dapat menggunakan sistem beserta seluruh fiturnya.

4.2.2 Analisis Kebutuhan Fungsional

Kebutuhan fungsional merupakan pernyataan-pernyataan mengenai apa saja yang dapat dilakukan sistem, cara sistem menangani masukan tertentu, dan perilakunya pada situasi tertentu. Kebutuhan fungsional kakas bantu penilaian kopling menggunakan metrik *indirect package coupling based on responsibility* dapat dilihat pada Tabel 4.2.

Tabel 4.2 Kebutuhan fungsional sistem

Nomor Fungsi	Definisi dan Spesifikasi Kebutuhan	Use case
SRS_KBPKIP_100	Pengguna harus dapat memasukkan berkas rancangan package diagram ke dalam sistem.	Memasukkan berkas diagram *package*
	1. Sistem hanya memproses berkas dengan ekstensi ".xml". (SRS_KBPKIP_101) 2. Jika ekstensi bukan ".xml", sistem akan menampilkan pesan error. (SRS_KBPKIP_102)	
SRS_KBPKIP_110	Pengguna harus dapat melakukan parsing terhadap berkas *package diagram* yang dimasukkan.	*Parsing* XML
	1. Data yang diambil saat parsing yaitu, nama package, relasi, nama class, dan tipe class. (SRS_KBPKIP_111) 2. Jika data yang dibutuhkan dari berkas tidak ditemukan, sistem akan menampilkan pesan error. (SRS_KBPKIP_112) 3. Jika ada kesalahan pada kondisi berkas, sistem akan menampilkan pesan error. (SRS_KBPKIP_113)	

Nomor Fungsi	Definisi dan Spesifikasi Kebutuhan	Use case
	4. Jika ada kesalahan struktur XML dari berkas, sistem akan menampilkan pesan error. (SRS_KBPKIP_114)	
SRS_KBPKIP_120	Pengguna harus dapat menilai kopling setiap *package* berdasarkan hasil perhitungan *normalized distance*.	Menilai kopling
	1. Relasi dalam berkas diagram tidak boleh berbentuk sirkular. (SRS_KBPKIP_121) 2. Jika relasi dalam diagram ada yang berbentuk sirkular, sistem akan menampilkan pesan error. (SRS_KBPKIP_122)	

4.2.3 Analisis Kebutuhan Non-Fungsional

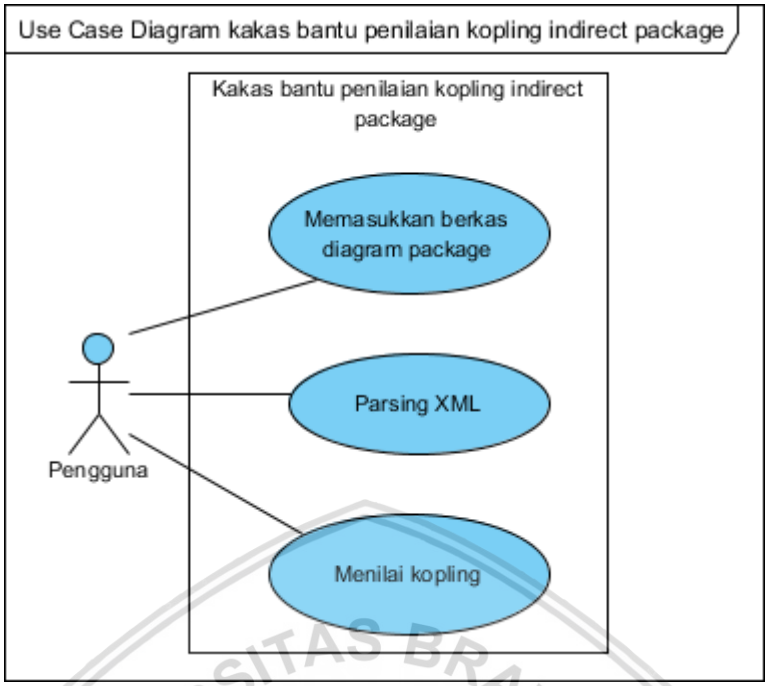
Kebutuhan non-fungsional merupakan kebutuhan yang tidak terlihat secara langsung tapi dapat meningkatkan kepuasan pengguna. Biasanya kebutuhan ini menunjukkan batasan dalam penggunaan fitur, seperti kompatibilitas situs web terhadap *browser,* semakin banyak *browser* yang mendukung untuk situs web tersebut, maka semakin tinggi kepuasan penggunanya. Tabel 4.3 berisi daftar kebutuhan non-fungsional sistem.

Tabel 4.3 Kebutuhan non-fungsional sistem

Kode Fungsi	Definisi dan Spesifikasi Kebutuhan
SRS_KBPKIP_200	Sistem harus dapat menilai kopling lebih cepat dari manusia. (*Performance*).
	1. Sistem harus membutuhkan waktu lebih sedikit dari manusia dalam menilai kopling *package event* pada diagram *package* JDraw yang disederhanakan, yaitu kurang dari 10 menit 28 detik. (SRS_KBPKIP_201).

4.2.4 Pemodelan *Use Case Diagram*

Use case diagram adalah model kebutuhan yang menggambarkan aktor-aktor beserta seluruh fungsionalitas sistem yang dapat ia jalankan. Gambar 4.1 menunjukkan bahwa pengguna dapat menjalankan seluruh fungsionalitas sistem kakas bantu perhitungan kopling, yaitu memasukkan berkas diagram *package*, melakukan *parsing* berkas XML *package diagram*, dan menilai kopling.



Gambar 4.1 *Use case diagram* kakas bantu penilaian kopling *indirect package*

4.2.5 *Use Case Scenario*

Setiap *use case scenario* berfungsi untuk menjelaskan tahapan-tahapan yang dilalui dari setiap *use case* yang ada pada Gambar 4.1. *Use case scenario* akan digambarkan dalam bentuk tabel yang terdiri atas nama *use case*, deskripsi, aktor, pra-kondisi, tindakan, post-kondisi, dan alternatif. Nama *use case* menunjukkan nama dari *use case* pada Gambar 4.1 yang akan dijelaskan. Deskripsi berisi penjelasan mengenai apa yang terjadi pada *use case* tersebut. Baris aktor menunjukkan pelaku yang dapat menggunakan fitur ini. Pra-Kondisi merupakan keadaan yang harus terpenuhi sebelum memasuki tahapan *use case* yang dijelaskan pada baris tindakan. Post-kondisi adalah keadaan yang terbentuk setelah tahapan dari tindakan *use case* dilakukan. Terakhir merupakan alternatif, disini dijelaskan kemungkinan post-kondisi yang berbeda.

4.2.5.1 *Use case scenario* memasukkan berkas diagram *package*

Pada tabel 4.4 pengguna akan menentukan berkas XML *package diagram* mana yang akan dinilai koplingnya. Untuk melakukan hal ini, pengguna harus sudah memasuki halaman input berkas XML terlebih dahulu. Selanjutnya pengguna menekan tombol bergambar berkas untuk membuka pengelola berkas dan memilih berkas diagram *package.* Selanjutnya sistem akan memvalidasi berkas yang dimasukkan berdasarkan ekstensinya, jika XML sistem akan menampilkan nama berkas dan siap untuk diparsing. Jika ekstensi berkas bukan XML, sistem akan menampilkan pesan error terkait kesalahan ekstensi berkas.

Tabel 4.4 *Use case scenario* memasukkan berkas diagram *package*

Nama Use Case	Memasukkan berkas diagram *package*
Deskripsi	Menjelaskan bagaimana pengguna dapat memasukkan berkas *XML package diagram*.
Aktor	Pengguna
Pra-kondisi	Pengguna masuk ke halaman awal sistem (halaman input berkas XML).
Tindakan	1. Pengguna menekan tombol "choose file" 2. Sistem menampilkan *pop-up* berupa pengelola berkas. 3. Pengguna memilih berkas XML diagram *package* yang ingin dinilai koplingnya. 4. Sistem memvalidasi berkas yang dimasukkan.
Post-kondisi	Nama berkas XML akan ditampilkan dan siap untuk dilakukan parsing terhadapnya.
Alternatif	Jika yang dimasukkan bukan merupakan berkas dengan ekstensi ".xml" sistem akan menampilkan pesan error.

4.2.5.2 *Use case scenario parsing XML*

Pada tabel 4.5 dijelaskan bagaimana pengguna melakukan parsing terhadap berkas XML diagram *package*. Sebelum melakukan *parsing* XML pengguna harus memasukkan berkas XML diagram *package* terlebih dahulu. Proses ini dimulai dengan menekan tombol *parse*, lalu sistem akan melakukan proses parsing dan menyimpan, serta menampilkan hasilnya. Jika data yang dibutuhkan tidak ditemukan pada proses parsing, maka sistem akan menampilkan pesan error terkait tidak ditemukannya data.

Tabel 4.5 *Use case scenario parsing* XML

Nama Use Case	*Parsing* XML
Deskripsi	Menjelaskan bagaimana pengguna dapat melakukan parsing data terhadap berkas XML diagram *package*.
Aktor	Pengguna
Pra-kondisi	Pengguna memasukkan berkas XML diagram *package*.
Tindakan	1. Pengguna menekan tombol Parse. 2. Sistem melakukan parsing terhadap berkas XML diagram *package*.
Post-kondisi	Sistem menampilkan pesan berhasil dan memperoleh data nama *package,* relasi, nama *class,* dan tipe *class.*
Alternatif 1	Jika data yang dibutuhkan tidak ditemukan, sistem menampilkan pesan error.

Alternatif 2	Jika ada masalah pada berkas yang dimasukkan, seperti *corrupt* dan lain sebagainya, sistem menampilkan pesan error.
Alternatif 3	Jika ada kesalahan dalam struktur berkas XML yang akan diproses oleh *parser*, seperti *tag* yang tidak ditutup atau semacamnya, sistem menampilkan pesan error.

4.2.5.3 *Use case scenario* menilai kopling

Pada Tabel 4.6 berikut dijelaskan tahapan umum dalam menilai kopling" yang dilakukan pengguna untuk mendapatkan nilai kopling masing-masing *package* dari dalam bentuk jarak nilai koplingnya dari nilai ideal. Setelah melakukan *parsing*, pengguna menekan tombol "Calculate" sehingga sistem akan mulai melakukan perhitungan. Hasil yang diperoleh akan ditampilkan oleh sistem.

Tabel 4.6 *Use case scenario* menilai kopling

Nama Use Case	Menilai kopling
Deskripsi	Menjelaskan bagaimana sistem melakukan perhitungan terhadap data yang diperoleh dari *parsing* untuk menilai kopling.
Aktor	Pengguna
Pra-kondisi	Pengguna melakukan *parsing* terhadap berkas XML.
Tindakan	1. Pengguna menekan tombol "Calculate" 2. Sistem melakukan penilaian kopling *package's normalized distance* terhadap data yang diperoleh dari *parsing* berkas XML.
Post-kondisi	Sistem menampilkan nama package, nilai *dual ranking instability, package abstraction,* dan *package's normalized distance* yang menunjukkan jarak nilai kopling dari titik ideal.
Alternatif	Jika ditemukan relasi sirkular antar *package* atau klas, sistem menampilkan pesan error.

BAB 5 PERANCANGAN

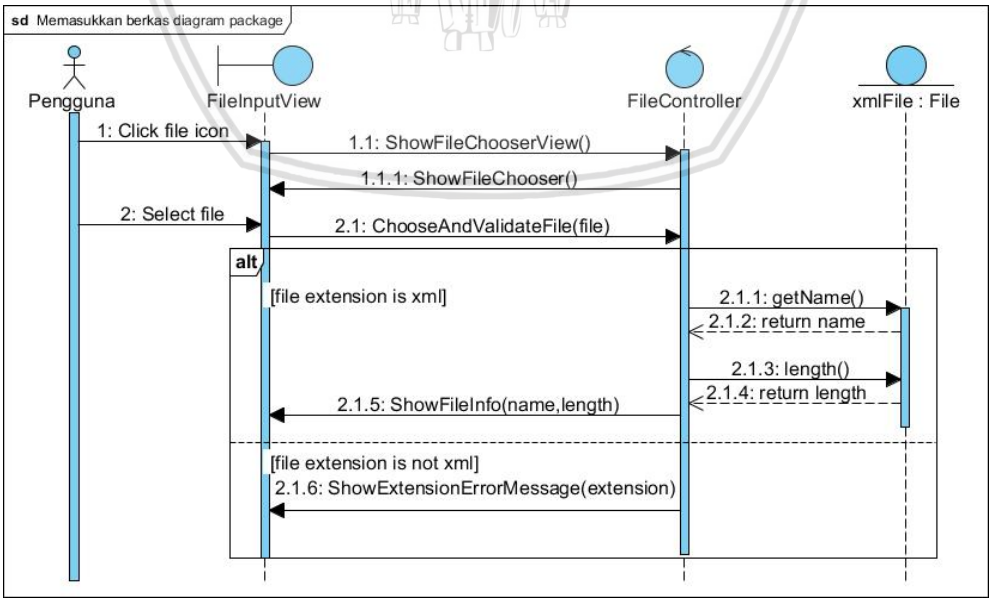
Bab ini membahas perancangan sistem kakas bantu penilaian kopling menggunakan metrik *indirect package coupling based on responsibility*. Perancangan dilakukan dengan menggunakan hasil dari analisis kebutuhan sebelumnya sebagai acuan. Perancangan perangkat lunak penelitian ini terbagi atas tiga tahapan utama, yaitu perancangan arsitektur, perancangan komponen, dan perancangan antarmuka pengguna. Hasil analisis kebutuhan dan rancangan tersebut akan digunakan untuk melakukan tahap implementasi sistem.

5.1 Perancangan Arsitektur

Perancangan arsitektur perangkat lunak dilakukan dengan membuat *sequence diagram* dan diagram klas. *Sequence diagram* ini digunakan untuk menggambarkan interaksi antar komponen ketika suatu fitur sedang berjalan dan sebagai dasar pembentukan diagram klas. Diagram klas berfungsi untuk menggambarkan arsitektur sistem. Isi dari diagram klas nantinya diperjelas oleh algoritme pada tahap perancangan komponen.

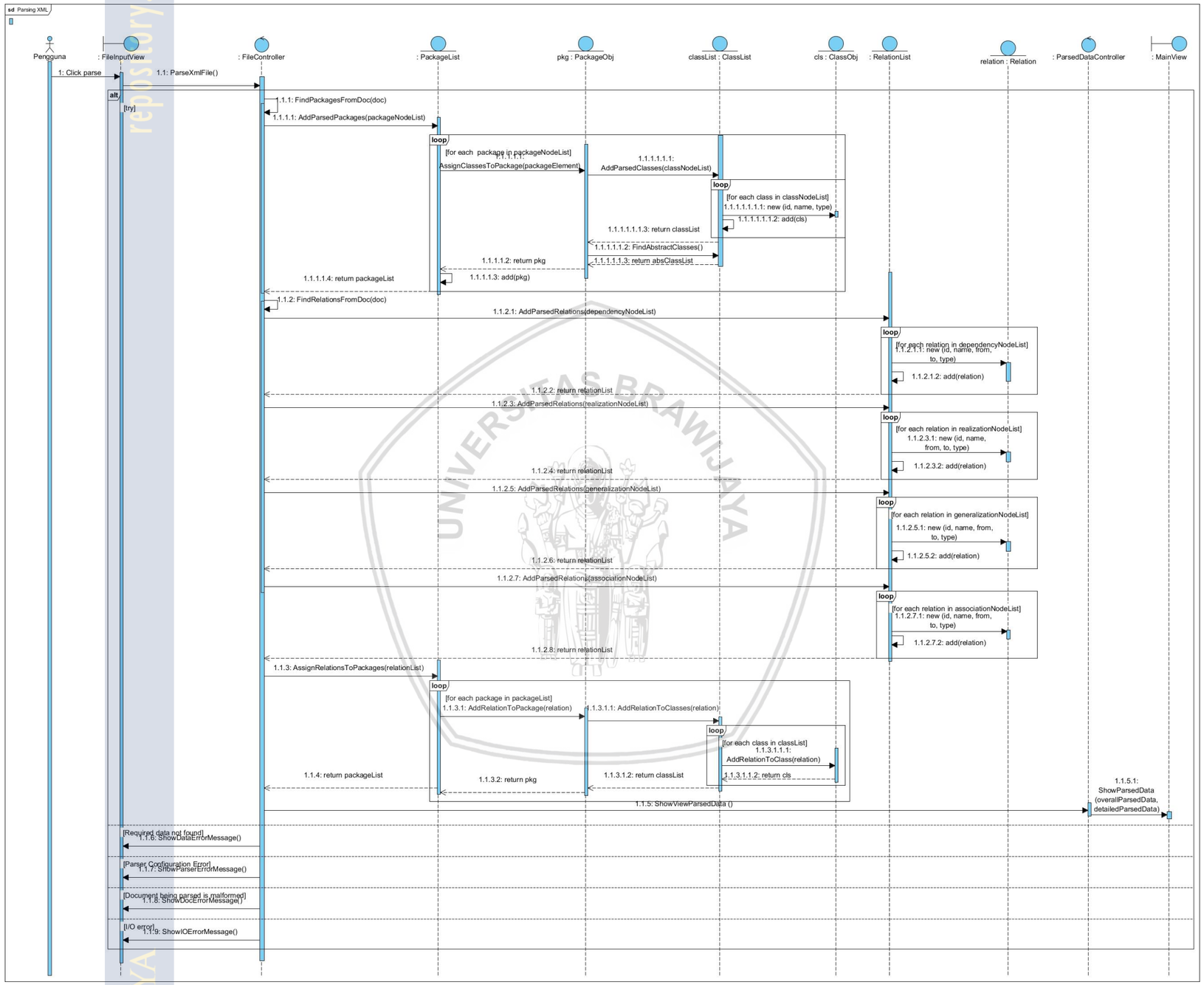
5.1.1 Perancangan *Sequence Diagram*

Gambar 5.1 menunjukkan rancangan *sequence diagram* dari memasukkan berkas diagram *package*. Komponen yang akan berinteraksi antara lain FileInputView sebagai view untuk menerima masukan, FileController untuk memperoleh data masukan dan xmlFile yang merupakan instansiasi objek dari File untuk merepresentasikan berkas. Jika berkas yang dimasukkan berupa xml, sistem menampilkan informasi berkas. Jika berkas tidak berupa xml, sistem akan menampilkan pesan error.



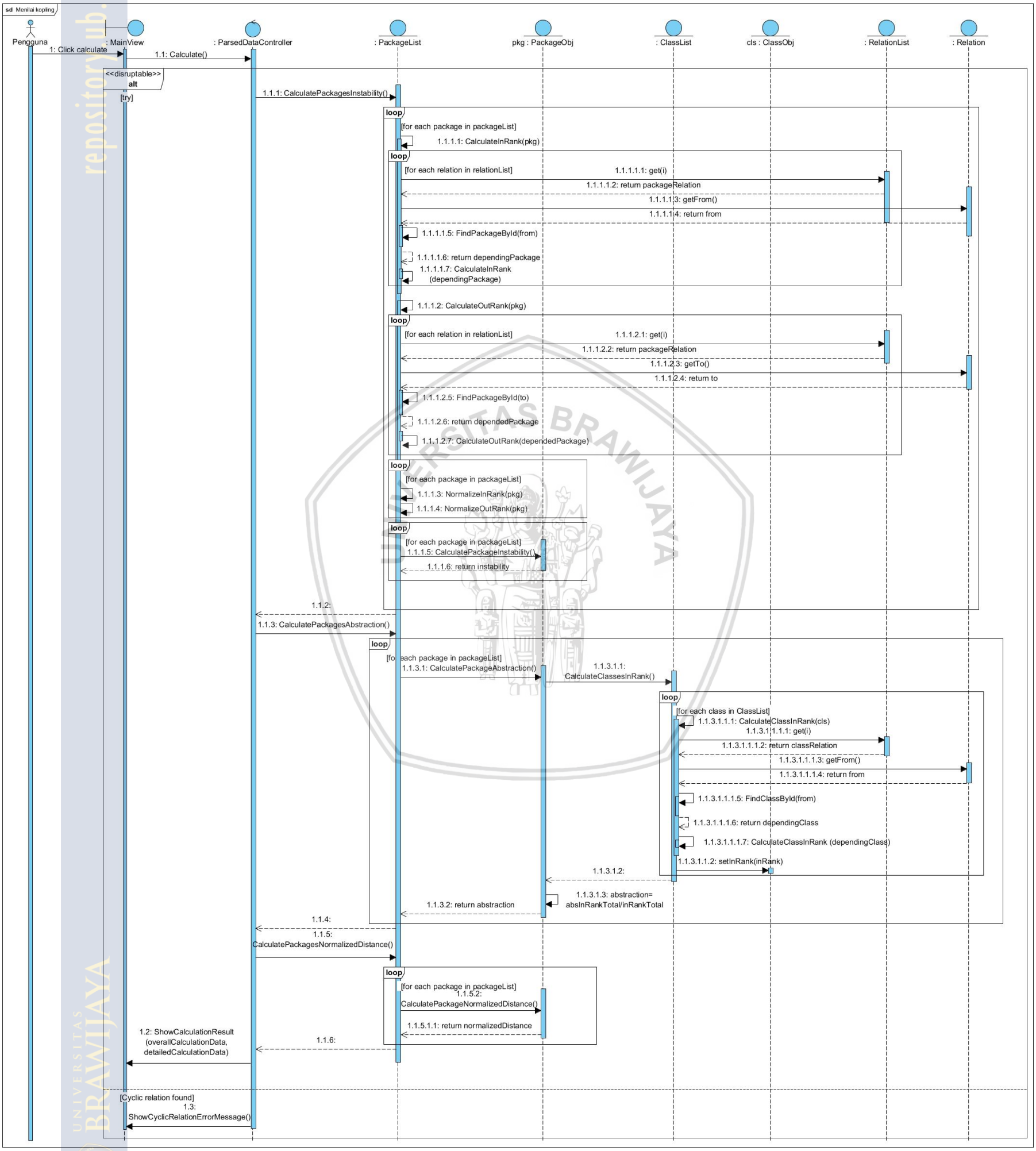
Gambar 5.1 Sequence diagram dari memasukkan berkas diagram *package*

Gambar 5.2 menunjukkan rancangan sequence diagram dari parsing xml. Komponen yang akan berinteraksi berupa *model*, *view*, dan *controller*. *View* yang terlibat antara lain FileInputView untuk menerima input dan MainView untuk menampilkan keluaran. Controller yang berinteraksi antara lain, FileController untuk menyimpan data dari berkas ke *model* dan ParsedDataController untuk mengolah data yang tersimpan pada *model*. *Model* yang terlibat antara lain, PackageList untuk menyimpan kumpulan *package*, PackageObj untuk merepresentasikan *package*, instansiasi objek ClassList berupa classList untuk menyimpan kumpulan klas dan absClassList untuk menyimpan kumpulan klas abstrak, ClassObj untuk merepresentasikan klas, RelationList untuk menyimpan kumpulan relasi, dan Relation untuk merepresentasikan relasi antar *package* dan antar klas. Jika semua data yang dibutuhkan ditemukan pada berkas, data akan disimpan oleh FileController pada tiap-tiap *model*. Data seluruh *package* dan klas yang ada di dalamnya beserta relasi akan ditampilkan oleh ParsedDataController pada mainView. Jika data tidak ditemukan, maka ParsedDataController akan mengarahkan mainView untuk menampilkan pesan error terkait kurangnya data pada berkas tersebut.



Gambar 5.2 Sequence diagram dari parsing xml

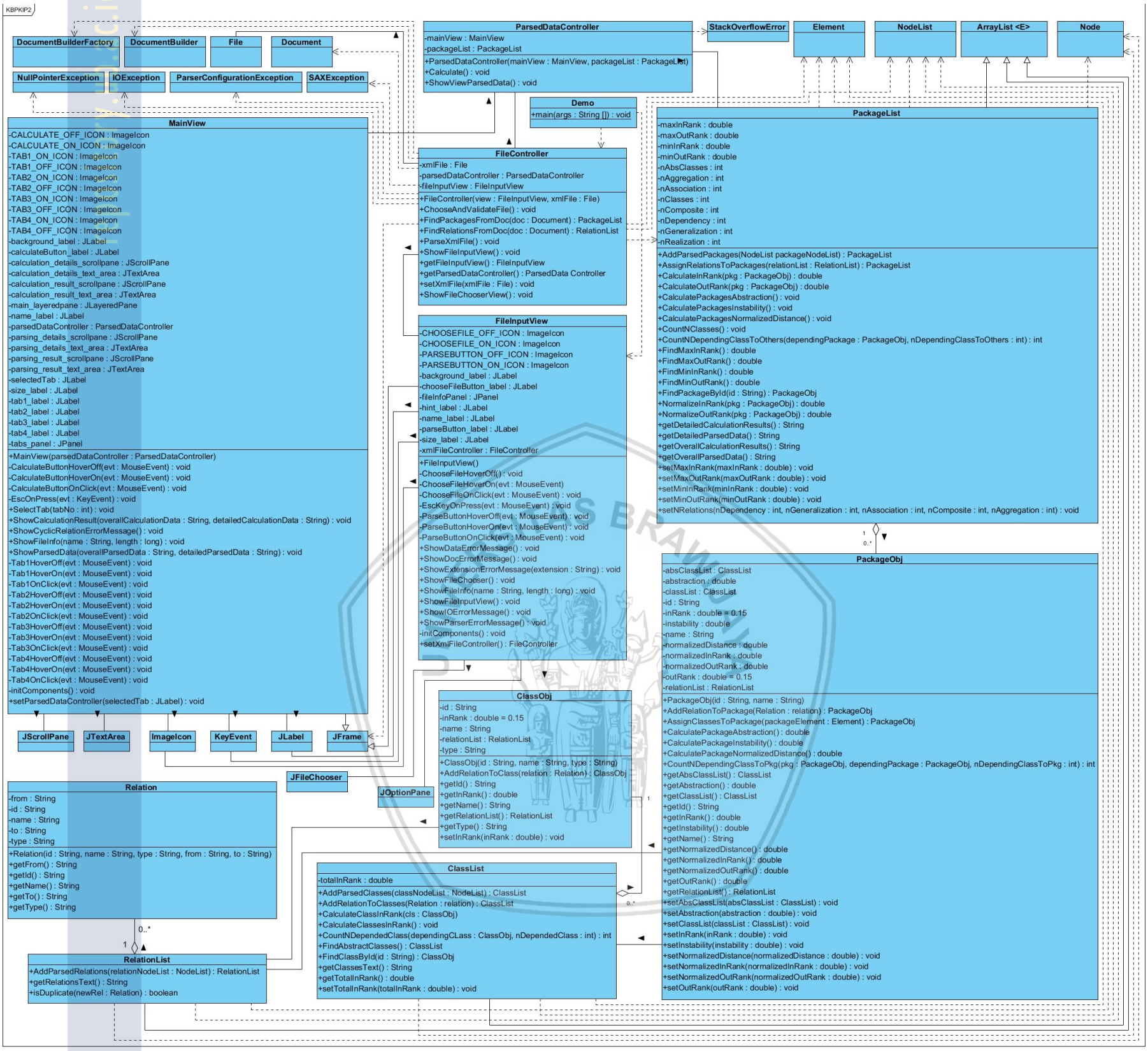
Gambar 5.3 menunjukkan rancangan diagram *sequence* dari menilai kopling. Komponen yang akan berinteraksi berupa beberapa *model*, *view* yang diwakili oleh MainView untuk menampilkan hasil perhitungan, dan *controller* yang diwakili oleh ParsedDataController untuk mengarahkan perhitungan yang akan dilakukan oleh tiap-tiap model serta mengirimkan hasil perhitungan tersebut kepada MainView. *Model* yang terlibat antara lain, PackageList untuk menyimpan kumpulan *package*, PackageObj untuk merepresentasikan *package*, instansiasi objek ClassList berupa classList untuk menyimpan kumpulan klas dan absClassList untuk menyimpan kumpulan klas abstrak, ClassObj untuk merepresentasikan klas, RelationList untuk menyimpan kumpulan relasi, dan Relation untuk merepresentasikan relasi antar *package* dan antar klas. Keluaran yang diperoleh berupa nama seluruh *package* dan masing-masing nilai dari *responsibility*, *dependency*, *dual ranking instability*, *package abstraction*, dan *package's normalized distance,* serta saran terhadap package tersebut.



Gambar 5.3 Sequence diagram dari menilai kopling

5.1.2 Perancangan Diagram Klas

Gambar 5.4 menunjukkan rancangan diagram klas dari kakas bantu penilaian kopling *indirect package*. Klas-klas yang ada pada klas diagram ini antara lain, FileInputView, MainView, FileController, ParsedDataController, PackageList, PackageObj, ClassList, ClassObj, RelationList, Relation dan beberapa klas dari *library*, baik yang sudah disediakan oleh Java, maupun dari pihak ketiga. Klas-klas yang berasal dari *library* Java, antara lain Jframe, Jlabel, File, DocumentBuilder, DocumentBuilderFactory, KeyEvent, ImageIcon, MouseEvent, JscrollPane, JtextArea, JfileChooser, dan ArrayList. Klas yang berasal dari *library* pihak ketiga, antara lain Document, Element, NodeList, Node



Gambar 5.4 Diagram Klas Kakas Bantu Perhitungan Kopling *Indirect Package*

5.2 Perancangan Komponen

Perancangan komponen dilakukan dengan membuat algoritme yang akan menjelaskan alur kerja komponen pada diagram klas. Algoritme merupakan rangkaian alur kerja dalam pemecahan sekumpulan masalah yang didefinisikan secara jelas dan tidak ambigu. Algoritme yang telah dibuat akan dijadikan sebagai dasar implementasi agar dapat menjalankan fitur-fitur yang telah didefinisikan. Algoritme kakas bantu perhitungan kopling *indirect package* akan ditunjukkan dalam *pseudocode*. Tiga tahapan utama yang akan dicantumkan algoritmenya adalah memasukkan berkas diagram *package*, *parsing* xml, dan menilai kopling.

5.2.1 Algoritme Memasukkan Berkas Diagram *Package*

Sistem yang dibangun menyelesaikan proses memasukkan berkas diagram *package* secara garis besar melalui algoritme ChooseAndValidateFile pada Tabel 5.1. Setelah pengguna menekan tombol bergambar berkas, sistem akan menjalankan ChooseAndValidateFile. Melalui algoritme tersebut sistem menampilkan jendela *file chooser* dan memvalidasi berkas yang telah dipilih pada jendela *file chooser*. Jika berkas yang dimasukkan memiliki ekstensi ".xml", sistem menampilkan informasi mengenai berkas dan tombol untuk melakukan *parsing*. Jika ekstensi berkas bukan merupakan ".xml" sistem akan menampilkan pesan error terkait ekstensi berkas tersebut.

Tabel 5.1 Algoritme ChooseAndValidateFile

```
PROCEDURE ChooseAndValidateFile (file : File)
   IF ekstensi berkas adalah "xml"  THEN
      Tampilkan informasi berkas WITH nama berkas, ukuran berkas
   ELSE
      INIT ekstensi TO  ""
      GET ekstensi dari nama berkas
      CALL tampilkan pesan ekstensi error WITH ekstensi
   ENDIF
```

5.2.2 Algoritme *parsing* xml

Sistem yang dibangun menyelesaikan proses *parsing* xml melalui beberapa tahapan algoritme. Setelah pengguna menekan tombol *parse*, ParseXmlFile yang ditunjukkan pada Tabel 5.2 dijalankan. nkan Jika tidak ada kesalahan, sistem akan menjalankan FindPackagesFromDoc pada Tabel 5.3 untuk menyimpan data seluruh package beserta klas-klasnya dari hasil *parsing* berkas xml ke dalam packageList. Lalu, sistem melakukan *parsing* pada berkas untuk menyimpan data relasi ke dalam relationList melalui FindRelationsFromDoc pada Tabel 5.4. Selanjutnya relasi pada relationList disimpan oleh masing-masing *package* yang terlibat pada packageList dengan menjalankan AssignRelationsToPackages yang ditunjukkan pada Tabel 5.5. Hasil *parsing* sistem beserta informasi berkas ditampilkan dan tombol *parse* diganti menjadi *calculate*. Jika terjadi kesalahan pada saat menjalankan proses-proses sebelumnya, sistem akan menampilkan pesan error sesuai jenis kesalahan yang terjadi.

Tabel 5.2 Algoritme ParseXmlFile

```
PROCEDURE ParseXmlFile()
   TRY
      INIT dbFactory TO instance baru DocumentBuilderFactory
      INIT dBuilder TO instance baru DocumentBuilder
      INIT doc TO hasil parse oleh dBuilder WITH berkas xml
      CALL normalisasi doc
      CALL mencari data package-package WITH doc RETURNING daftar package
      CALL mencari data relasi-relasi WITH doc RETURNING daftar relasi
      CALL memberi package dalam daftar WITH daftar relasi RETURNING daftar
package yang memiliki relasi
      SET tampilan input berkas TO invisible
      CONSTRUCT tampilan utama
      SET ukuran tampilan utama TO ukuran maksimal layar
      CALL tampilkan informasi berkas di tampilan utama WITH nama berkas,
ukuran berkas
      CONSTRUCT controller data hasil parsing WITH tampilan utama, daftar
package
      CALL tampilkan data hasil parsing WITH tampilan utama
   END TRY
   EXCEPTION
      WHEN data yang diperlukan tidak ditemukan
         SHOW pesan error mengenai kesalahan data
      WHEN ada kesalahan pada berkas
         SHOW pesan error mengenai kesalahan berkas
      WHEN  ada kesalahan pada parser
         SHOW pesan error mengenai kesalahan parser
      WHEN  ada kesalahan struktur pada berkas yang diproses parser
         SHOW pesan error mengenai kesalahan struktur berkas
```

Tabel 5.3 Algoritme FindPackagesFromDoc

```
FUNCTION FindPackagesFromDoc(doc:Document)
   CONSTRUCT daftar package
   INIT daftar node model TO seluruh elemen models pada doc
   INIT elemen model TO node pertama dari daftar node model
   INIT daftar node package TO seluruh elemen package milik elemen model
   CALL menambahkan package hasil parsing ke dalam daftar package WITH
daftar node package RETURNING daftar package
   RETURN packageList
```

Tabel 5.4 Algoritme FindRelationsFromDoc

```
FUNCTION FindRelationsFromDoc(doc : Document)
   INIT daftar node relasi TO seluruh elemen model relationship container
pada doc
   INIT elemen relasi TO node pertama dari daftar node relasi
   INIT daftar node dependency TO seluruh elemen dependency milik elemen
relasi
   INIT daftar node association TO seluruh elemen association milik elemen
relasi
   INIT daftar node generalization TO seluruh elemen generalization milik
elemen relasi
   INIT daftar node realization TO seluruh elemen realization milik elemen
relasi
   CONSTRUCT daftar relasi
   CALL menambahkan relasi dependency hasil parsing ke dalam daftar relasi
WITH daftar node dependency RETURNING daftar relasi
   CALL menambahkan relasi realization hasil parsing ke dalam daftar relasi
WITH daftar node realization RETURNING daftar relasi
   CALL menambahkan relasi generalization hasil parsing ke dalam daftar
relasi WITH daftar node generalization RETURNING daftar relasi
   CALL menambahkan relasi association hasil parsing ke dalam daftar relasi
WITH daftar node association RETURNING daftar relasi
   RETURN relationList
```

Tabel 5.5 Algoritme AssignRelationsToPackages

```
FUNCTION AssignRelationsToPackages(relationList : RelationList)
   INIT jumlah tiap jenis relasi TO 0
   FOR each relasi dalam daftar relasi DO
      FOR each package in daftar package DO
         CALL menambahkan relasi ke package WITH relasi RETURNING package
      ENDFOR
      INCREMENT jumlah salah satu jenis relasi sesuai jenisnya
   ENDFOR
   CALL menetapkan jumlah tiap jenis relasi WITH jumlah dependency, jumlah
generalization, jumlah realization, jumlah association
   RETURN daftar package
```

5.2.3 Algoritme Menilai Kopling

Sistem yang dibangun menyelesaikan proses menilai kopling melalui beberapa tahapan algoritme. Setelah pengguna menekan tombol *Calculate* sistem akan menjalankan Calculate yang ditunjukkan pada Tabel 5.6. Sistem melakukan perhitungan nilai *instability* seluruh package dengan menjalankan CalculatePackagesInstability pada Tabel 5.7. Perhitungan instability setiap *package* dilakukan pada tahap CalculatePackageInstability yang ditunjukkan pada Tabel 5.8. Selanjutnya dilakukan perhitungan pada nilai *abstraction* seluruh package melalui CalculatePackagesAbstraction yang ditunjukkan pada Tabel 5.9. Nilai *abstraction* tiap *package* dihitung melalui CalculatePackageAbstraction seperti yang terlihat pada Tabel 5.10. Selanjutnya seluruh package dihitung nilai *normalized distance* pada

CalculatePackagesNormalizedDistance sebagaimana yang dapat dilihat pada Tabel 5.11. Perhitungan nilai *normalized distance* tiap *package* berdasarkan nilai *instability* dan *abstraction* dilakukan oleh CalculatePackageNormalizedDistance yang ditunjukkan pada Tabel 5.12. Hasil perhitungan tersebut ditampilkan kepada pengguna yang dilengkapi dengan saran.

Tabel 5.6 Algoritme Calculate

```
PROCEDURE Calculate()
   TRY
      CALL hitung instability dari setiap package di daftar package
      CALL hitung abstraction dari setiap package di daftar package
      CALL hitung normalized distance dari setiap package di daftar package
      CALL mendapatkan teks perhitungan rinci seluruh package RETURNING
hasil perhitungan rinci
      CALL mendapatkan teks perhitungan garis besar seluruh package
RETURNING hasil perhitungan garis besar
      CALL tampilkan teks hasil perhitungan di tampilan utama WITH hasil
perhitungan garis besar, hasil perhitungan rinci
   END TRY
   EXCEPTION
      WHEN relasi sirkular ditemukan
         SHOW pesan error mengenai relasi sirkular
```

Tabel 5.7 Algoritme CalculatePackagesInstability

```
PROCEDURE CalculatePackagesInstability()
   FOR each package di daftar package DO
      CALL hitung inRank package WITH package RETURNING inRank
      CALL hitung outRank package WITH package RETURNING outRank
   ENDFOR
      CALL cari inRank terbesar RETURNING nilai inRank terbesar
      CALL cari inRank terkecil RETURNING nilai inRank terkecil
      CALL cari outRank terbesar RETURNING nilai outRank terbesar
      CALL cari outRank terkecil RETURNING nilai outRank terkecil
   FOR each package di daftar package DO
      CALL normalisasi inRank WITH package RETURNING normalized inRank
      CALL normalisasi outRank WITH package RETURNING normalized outRank
   ENDFOR
   FOR each package di daftar package DO
      CALL hitung nilai instability package RETURNING instability
   ENDFOR
```

Tabel 5.8 Algoritme CalculatePackageInstability

```
FUNCTION CalculatePackageInstability()
   COMPUTE instability AS normalized outRank / (normalized inRank +
normalized outRank)
   RETURN instability
```

Tabel 5.9 Algoritme CalculatePackagesAbstraction

```
PROCEDURE CalculatePackagesAbstraction()
   FOR each package di daftar package DO
      CALL hitung abstraction dari package RETURNING abstraction
   ENDFOR
```

Tabel 5.10 Algoritme CalculatePackageAbstraction

```
FUNCTION CalculatePackageAbstraction()
   INIT total inRank abstrak, total inRank TO 0
   CALL hitung inRank klas-klas
   FOR each klas di daftar klas abstrak DO
      COMPUTE total inRank abstrak AS total inRank abstrak + inRank klas
abstrak
   ENDFOR
   FOR each klas di daftar klas DO
      COMPUTE total inRank AS total inRank + inRank klas
   ENDFOR
   COMPUTE abstraction AS total inRank abstrak / total inRank
   RETURN abstraction
```

Tabel 5.11 Algoritme CalculatePackagesNormalizedDistance

```
PROCEDURE CalculatePackagesNormalizedDistance()
   FOR each package di daftar package DO
   CALL hitung normalized distance dari package RETURNING normalized
distance
ENDFOR
```

Tabel 5.12 Algoritme CalculatePackagesNormalizedDistance

```
FUNCTION CalculatePackageNormalizedDistance()
   COMPUTE normalized distance AS nilai absolut dari hasil abstraction +
instability - 1
   RETURN normalized distance
```

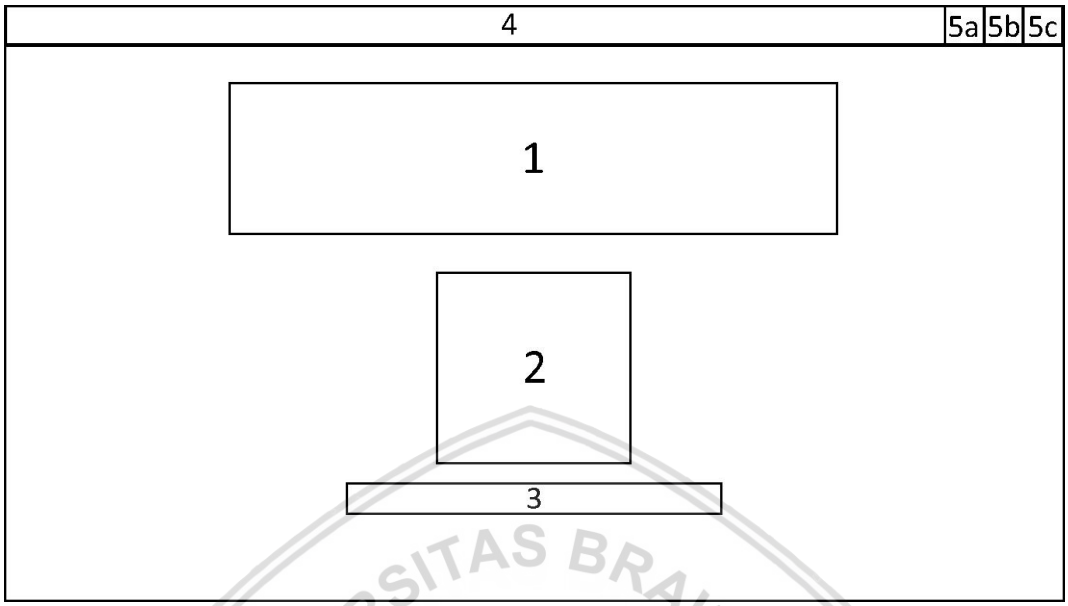
5.3 Perancangan Antarmuka Pengguna

Antarmuka pengguna merupakan tampilan sistem yang akan digunakan oleh pengguna untuk berinteraksi dengan sistem. Perancangan antarmuka pengguna dilakukan dengan membuat gambaran kerangka untuk memberikan gambaran tata letak komponen-komponen pada sistem. Rancangan antarmuka pengguna pada kakas bantu penilaian kopling *indirect package* ditunjukkan dalam bentuk *wireframe* yang dilengkapi dengan tabel deskripsi komponen.

5.3.1 Rancangan Antarmuka Memasukkan Berkas Diagram *Package*

Pengguna akan dihadapkan dengan beberapa halaman antarmuka untuk dapat memasukkan berkas diagram *package* yang akan dinilai koplingnya. Halaman pertama yang tampil memiliki rancangan seperti pada gambar 5.5. Pengguna harus menekan tombol *choose file* untuk menampilkan *file chooser* berupa *pop-up window*. Tombol

choose file tersebut ditunjukkan oleh komponen dengan angka 2. Penjelasan dari seluruh komponen pada gambar 5.5 dapat dilihat di tabel 5.13.



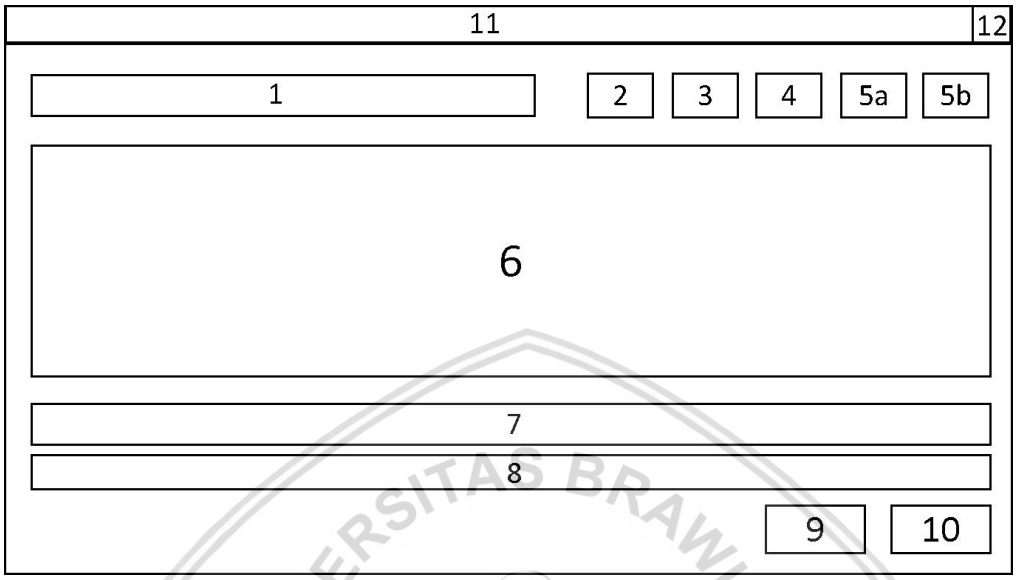
Gambar 5.5 Rancangan antarmuka memasukkan berkas

Tabel 5.13 Deskripsi komponen antarmuka memasukkan berkas

No	Nama Komponen	Deskripsi
1	Nama sistem	Teks yang menampilkan nama sistem.
2	Tombol berkas	Tombol dengan ikon berkas yang berfungsi untuk menampilkan *file chooser*.
3	Petunjuk	Memberi petunjuk mengenai cara memilih berkas yang akan diproses.
4	*Title bar*	Menampilkan judul *window*.
5	a. *Minimize*	Tombol untuk menyembunyikan *window* dari kakas bantu ke *taskbar*.
	b. *Maximize*	Tombol untuk memperbesar *window* dari kakas bantu hingga memenuhi layar.
	c. *Close*	Keluar dari sistem dan menghentikan jalannya kakas bantu.

File chooser akan menampilkan daftar berkas-berkas yang dapat dipilih oleh pengguna untuk diproses melalui *parsing*. *File chooser* yang digunakan merupakan *library* yang sudah disediakan Java sehingga rancangan antarmukanya sama. Pengguna dapat menekan tombol *open* untuk menetapkan berkas yang telah dipilih sebagai berkas yang akan diproses. Tombol *open* ditunjukkan oleh komponen nomor 9 pada gambar

5.6. Penjelasan seluruh komponennya dapat ditunjukkan oleh tabel 5.14. Ekstensi berkas divalidasi oleh sistem, jika xml pemrosesan berkas dapat dilanjutkan ke tahap *parsing*. Jika berkas yang dimasukkan ke dalam sistem bukan berkas xml, berkas ditolak dan sistem akan menampilkan dialog pesan error "*Unsupported file type*".



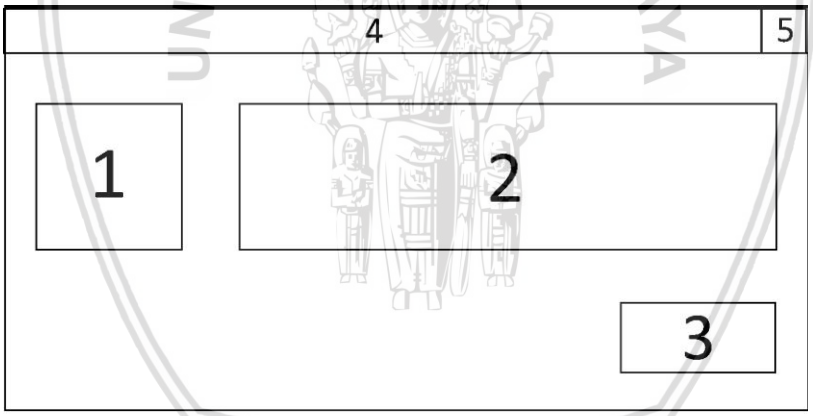
Gambar 5.6 Rancangan antarmuka *file chooser* Java

Tabel 5.14 Deskripsi komponen *file chooser*

No	Nama Komponen	Deskripsi
1	*Path*	Komponen yang menunjukkan *path* saat ini. Komponen ini juga memungkinkan *file chooser* untuk melakukan perpindahan menuju daftar *path* yang tersedia pada *dropdown*.
2	Tombol ke atas	Tombol dengan ikon *folder* dan panah ke atas yang berfungsi untuk keluar 1 tingkat dari direktori saat ini.
3	Tombol beranda	Tombol dengan ikon rumah yang berfungsi untuk berpindah dari path saat ini menuju direktori "*desktop*".
4	Tombol *folder* baru	Tombol dengan ikon *folder* dan cahaya kecil yang berfungsi untuk membuat *folder* baru.
5	a. *Toggle* tampilan daftar	*Toggle* dengan ikon daftar (kertas berisi daftar) untuk menampilkan berkas dan direktori dalam bentuk daftar. Jika diaktifkan, *toggle* tampilan detil akan dinonaktifkan.
	b. *Toggle* Tampilan detil	*Toggle* dengan ikon daftar (kertas berisi daftar) dan kaca pembesar untuk menampilkan berkas dan direktori beserta informasinya secara detil. Jika diaktifkan, *toggle* tampilan daftar akan dinonaktifkan.

No	Nama Komponen	Deskripsi
6	Daftar berkas	Menampilkan seluruh berkas dan direktori yang ada dalam suatu *path*.
7	Nama berkas	Teks yang menampilkan nama berkas yang sedang ditunjuk.
8	Tipe berkas	Menunjukkan jenis berkas yang akan ditampilkan pada daftar berkas, yaitu "*All Files*".
9	Tombol *Open*	Tombol untuk menetapkan berkas yang saat ini ditunjuk sebagai berkas yang akan diproses oleh sistem.
10	Tombol *Cancel*	Tombol untuk keluar dari *file chooser*.
11	*Title bar*	Menampilkan judul *window*.
12	Tombol *Close*	Keluar dari sistem dan menghentikan jalannya kakas bantu.

Rancangan antarmuka dialog pesan error "*Unsupported file type*" dapat dilihat pada gambar 5.7. Pengguna dapat menekan tombol *ok* untuk kembali ke halaman awal memasukkan berkas. Penjelasan mengenai seluruh komponennya dapat dilihat pada tabel 5.15.



Gambar 5.7 Dialog pesan error "*Unsupported file type*"

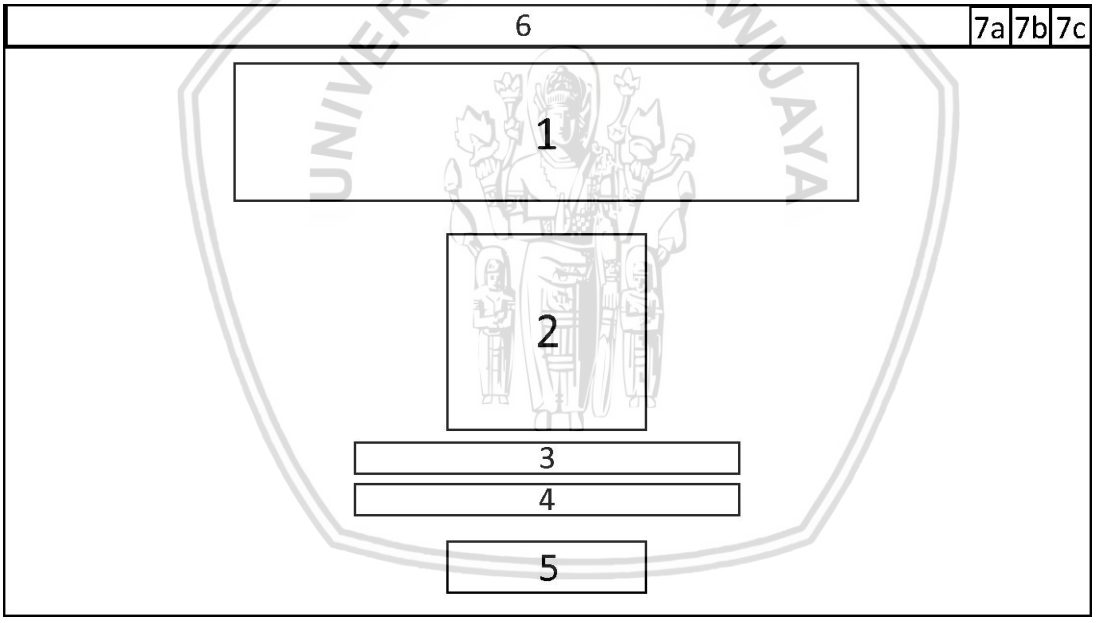
Tabel 5.15 Deskripsi komponen dialog pesan error "*Unsupported file type*"

No	Nama Komponen	Deskripsi
1	Ikon error	Ikon berwarna merah dengan tanda seru yang menandakan terjadinya error.
2	Teks penjelasan	Teks yang menjelaskan kemungkinan penyebab error yang sedang terjadi serta solusinya.
3	Tombol *Ok*	Menutup dialog pesan error.

No	Nama Komponen	Deskripsi
4	*Title bar*	Menampilkan judul *window*.
5	*Close*	Menutup dialog pesan error.

5.3.2 Rancangan Antarmuka Parsing Xml

Pengguna akan dihadapkan halaman antarmuka *parsing* xml dengan rancangan seperti pada gambar 5.8. Pada halaman ini pengguna dapat melihat informasi nama dan kapasitas berkas yang dimasukkan serta melakukan *parsing*. Proses p*arsing* dilakukan dengan menekan tombol *parse* yang ditunjukkan oleh komponen dengan angka 5. Penjelasan mengenai seluruh komponen pada antarmuka *parsing* xml dapat dilihat pada tabel 5.16. Jika *parsing* berhasil, data hasil parsing akan ditampilkan. Jika data yang penting tidak ditemukan saat *parsing,* sistem akan menampilkan dialog pesan error "*Required data not found*". Jika ada kesalahan terkait kondisi berkas yang dimasukkan seperti perubahan lokasi atau *corrupt*, sistem akan menampilkan pesan error "*File input error*". Jika terjadi permasalahan pada struktur berkas xml yang akan diproses, sistem menampilkan pesan error "*Xml document error*".



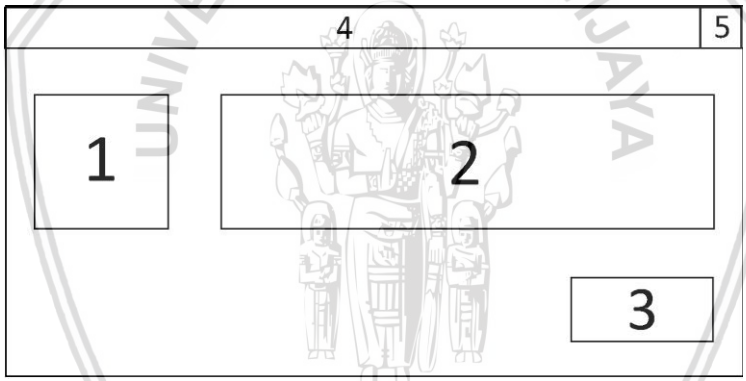
Gambar 5.8 Rancangan antarmuka halaman *parsing* xml

Tabel 5.16 Deksripsi komponen antarmuka *parsing* xml

No	Nama Komponen	Deskripsi
1	Nama sistem	Teks yang menampilkan nama sistem.
2	Tombol berkas	Tombol dengan ikon berkas yang berfungsi untuk menampilkan *file chooser*.

3	Nama berkas	Teks yang menampilkan nama dari berkas yang telah dipilih oleh pengguna.
4	Ukuran berkas	Teks yang menampilkan ukuran kapasitas dari berkas yang telah dipilih oleh pengguna.
5	Tombol *parse*	Tombol dengan tulisan "Parse" yang berfungsi untuk melakukan *parsing* terhadap berkas.
6	*Title bar*	Menampilkan judul *window*.
7	a. *Minimize*	Tombol untuk menyembunyikan *window* dari kakas bantu ke *taskbar*.
	b. *Maximize*	Tombol untuk memperbesar *window* dari kakas bantu hingga memenuhi layar.
	c. *Close*	Menghentikan jalannya kakas bantu.

Rancangan antarmuka dialog pesan error "*Required data not found*" dapat dilihat pada gambar 5.9. Pengguna dapat menekan tombol *ok* untuk kembali ke halaman *parsing* xml. Penjelasan mengenai seluruh komponennya dapat dilihat pada tabel 5.17.

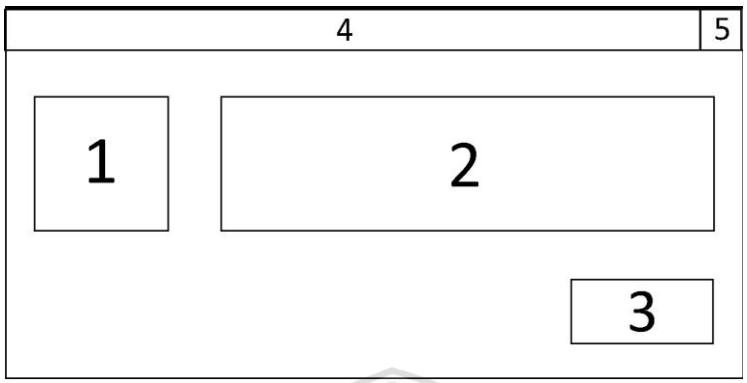


Gambar 5.9 Dialog pesan error "*Required data not found*"

Tabel 5.17 Deskripsi komponen dialog pesan error "*Required data not found*"

No	Nama Komponen	Deskripsi
1	Ikon error	Ikon berwarna merah dengan tanda seru yang menandakan terjadinya error.
2	Teks penjelasan	Teks yang menjelaskan kemungkinan penyebab error yang sedang terjadi serta solusinya.
3	Tombol *Ok*	Menutup dialog pesan error.
4	*Title bar*	Menampilkan judul *window*.
5	*Close*	Menutup dialog pesan error.

Rancangan antarmuka dialog pesan error "*File input error*" dapat dilihat pada gambar 5.10. Pengguna dapat menekan tombol *ok* untuk kembali ke halaman *parsing* xml. Penjelasan mengenai seluruh komponennya dapat dilihat pada tabel 5.18.

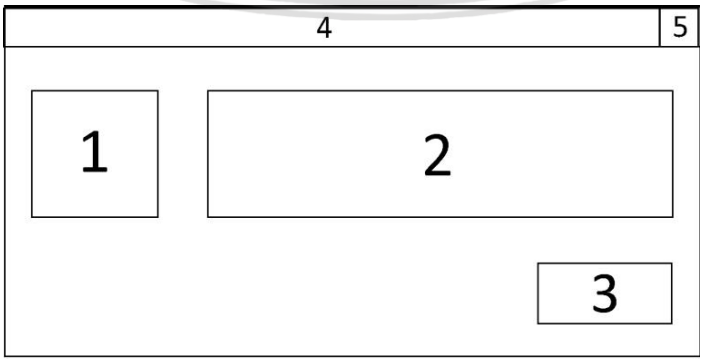


Gambar 5.10 Dialog pesan error "*File input error*"

Tabel 5.18 Deskripsi komponen dialog pesan error "*File input error*"

No	Nama Komponen	Deskripsi
1	Ikon error	Ikon berwarna merah dengan tanda seru yang menandakan terjadinya error.
2	Teks penjelasan	Teks yang menjelaskan kemungkinan penyebab error yang sedang terjadi serta solusinya.
3	Tombol *Ok*	Menutup dialog pesan error.
4	*Title bar*	Menampilkan judul *window*.
5	*Close*	Menutup dialog pesan error.

Rancangan antarmuka dialog pesan error "*Xml document error*" dapat dilihat pada gambar 5.11. Pengguna dapat menekan tombol *ok* untuk kembali ke halaman *parsing* xml. Penjelasan mengenai seluruh komponennya dapat dilihat pada tabel 5.19.



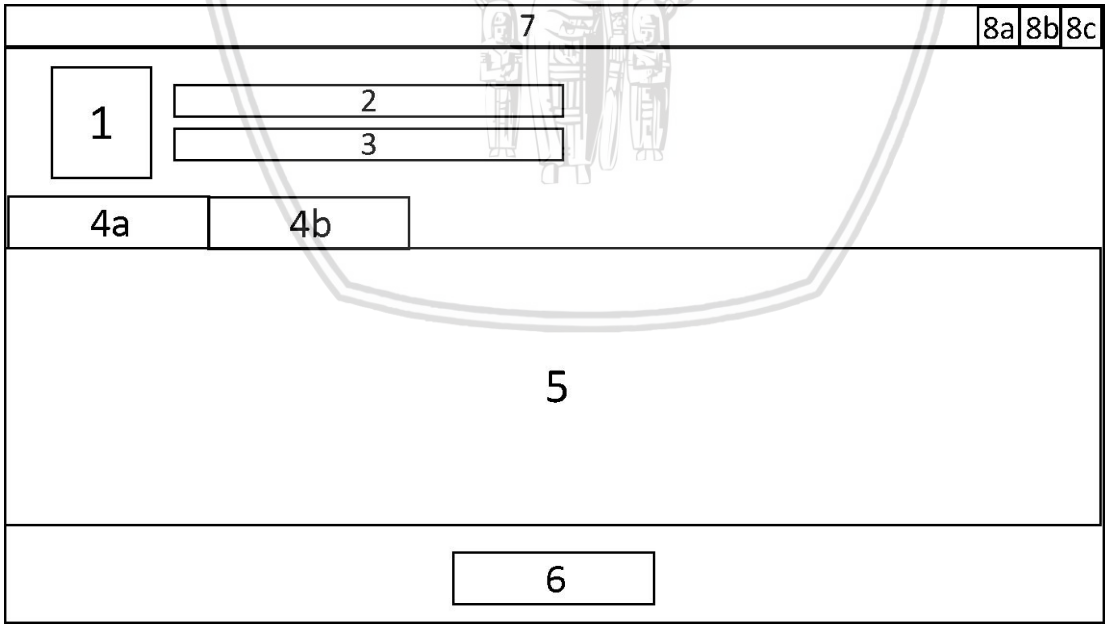
Gambar 5.11 Dialog pesan error "*Xml document error*"

Tabel 5.19 Deskripsi komponen dialog pesan error "*Xml document error*"

No	Nama Komponen	Deskripsi
1	Ikon error	Ikon berwarna merah dengan tanda seru yang menandakan terjadinya error.
2	Teks penjelasan	Teks yang menjelaskan kemungkinan penyebab error yang sedang terjadi serta solusinya.
3	Tombol *Ok*	Menutup dialog pesan error.
4	*Title bar*	Menampilkan judul *window*.
5	*Close*	Menutup dialog pesan error.

5.3.3 Rancangan Antarmuka Menilai Kopling

Pengguna akan dihadapkan dengan halaman menilai kopling yang juga menampilkan data hasil dari proses *parsing.* Data hasil parsing dapat dilihat secara garis besar maupun detil berdasarkan tab yang dipilih. Rancangan antarmuka halaman ini dapat ditunjukkan oleh gambar 5.12. Pengguna dapat memulai proses penilaian kopling dengan menekan tombol "*Calculate*" yang ditunjukkan pada komponen bernomor 6. Penjelasan seluruh komponen dari antarmuka menilai kopling tertera pada tabel 5.20. Setelah perhitungan selesai, sistem akan menampilkan halaman hasil perhitungan. Jika relasi sirkular ditemukan, perhitungan dihentikan dan sistem akan menampilkan dialog pesan error "*Cyclic relation found*".

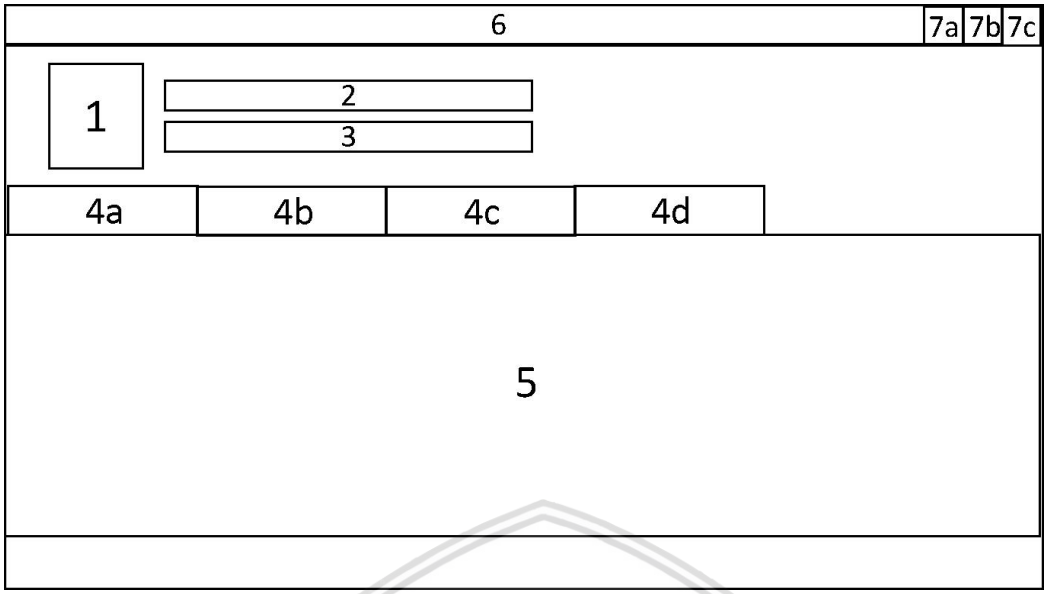


Gambar 5.12 Rancangan antarmuka halaman menilai kopling

Tabel 5.20 Deskripsi komponen antarmuka halaman menilai kopling

No	Nama Komponen	Deskripsi
1	Ikon berkas	Gambar ikon berkas.
2	Nama berkas	Teks yang menampilkan nama berkas yang telah melewati tahap *parsing*.
3	Ukuran berkas	Teks yang menampilkan ukuran kapasitas dari berkas yang telah melewati tahap *parsing*.
4	a. *Tab* hasil *parsing*	Tab yang ketika dipilih akan menampilkan hasil parsing berkas secara garis besar. Hanya salah satu tab yang dapat aktif dalam satu waktu.
	b. *Tab* detil *parsing*	Tab yang ketika dipilih akan menampilkan hasil parsing berkas secara detil. Hanya salah satu tab yang dapat aktif dalam satu waktu.
5	Teks hasil	Teks yang menampilkan hasil dari proses *parsing*. Teks akan ditampilkan sesuai dengan tab yang dipilih.
6	Tombol *calculate*	Tombol dengan tulisan "*Calculate*" yang berfungsi untuk melakukan perhitungan terhadap data hasil *parsing*.
7	*Title bar*	Menampilkan judul *window*.
8	a. *Minimize*	Tombol untuk menyembunyikan *window* dari kakas bantu ke *taskbar*.
	b. *Maximize*	Tombol untuk memperbesar *window* dari kakas bantu hingga memenuhi layar.
	c. *Close*	Menghentikan jalannya kakas bantu.

Halaman hasil berfungsi untuk menampilkan hasil penilaian kopling dan perhitungannya. Rancangan antarmuka untuk halaman hasil perhitungan ditunjukkan pada gambar 5.13. Pada halaman ini data yang digunakan untuk perhitungan juga dapat dilihat pada tabnya. Hasil perhitungan dapat dilihat secara garis besar saja maupun secara detil tergantung dari tab yang dipilih pengguna. Penjelasan mengenai komponen-komponen antarmuka pada halaman ini ditunjukkan oleh tabel 5.21.



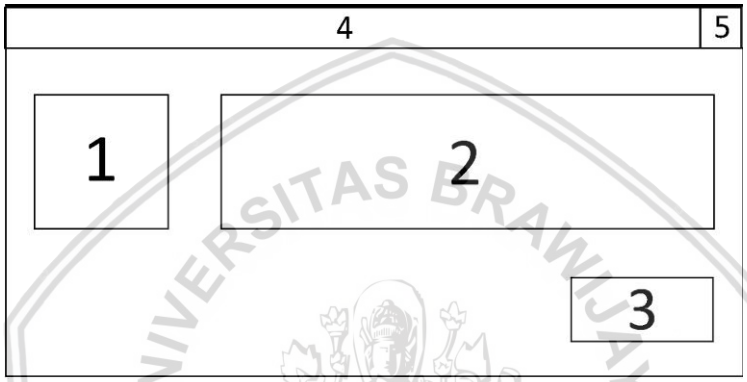
Gambar 5.13 Rancangan antarmuka halaman hasil perhitungan

Tabel 5.21 Deskripsi komponen halaman hasil perhitungan

No	Nama Komponen	Deskripsi
1	Ikon berkas	Gambar ikon berkas.
2	Nama berkas	Teks yang menampilkan nama berkas yang telah melewati tahap *parsing*.
3	Ukuran berkas	Teks yang menampilkan ukuran kapasitas dari berkas yang telah melewati tahap *parsing*.
4	a. *Tab* hasil *parsing*	Tab yang ketika dipilih akan menampilkan hasil parsing berkas secara garis besar. Hanya salah satu tab yang dapat aktif dalam satu waktu.
	b. *Tab* detil *parsing*	Tab yang ketika dipilih akan menampilkan hasil parsing berkas secara detil. Hanya salah satu tab yang dapat aktif dalam satu waktu.
	c. Tab hasil kalkulasi	Tab yang ketika dipilih akan menampilkan hasil kalkulasi data secara garis besar. Hanya satu tab yang dapat aktif dalam satu waktu.
	d. Tab detil kalkulasi	Tab yang ketika dipilih akan menampilkan hasil kalkulasi berkas secara detil. Hanya satu tab yang dapat aktif dalam satu waktu.
5	Teks hasil	Teks yang menampilkan hasil dari proses *parsing*. Teks akan ditampilkan sesuai dengan tab yang dipilih.
6	*Title bar*	Menampilkan judul *window*.

No	Nama Komponen	Deskripsi
7	a. *Minimize*	Tombol untuk menyembunyikan *window* dari kakas bantu ke *taskbar*.
	b. *Maximize*	Tombol untuk memperbesar *window* dari kakas bantu hingga memenuhi layar.
	c. *Close*	Menghentikan jalannya kakas bantu.

Rancangan antarmuka dialog pesan error "*Cyclic relation found*" dapat dilihat pada gambar 5.14. Pengguna dapat menekan tombol *ok* untuk kembali ke halaman *parsing* xml. Penjelasan mengenai seluruh komponennya dapat dilihat pada tabel 5.22.



Gambar 5.14 Dialog pesan error "*Cyclic relation found*"

Tabel 5.22 Deskripsi komponen dialog pesan error "*Cyclic relation found*"

No	Nama Komponen	Deskripsi
1	Ikon error	Ikon berwarna merah dengan tanda seru yang menandakan terjadinya error.
2	Teks penjelasan	Teks yang menjelaskan kemungkinan penyebab error yang sedang terjadi serta solusinya.
3	Tombol *Ok*	Menutup dialog pesan error.
4	*Title bar*	Menampilkan judul *window*.
5	*Close*	Menutup dialog pesan error.

BAB 6 IMPLEMENTASI

Bab ini membahas tentang implementasi kakas bantu penilaian kopling *indirect package* menjadi kode sumber berdasarkan rancangan yang telah dibuat. Bahasa pemrograman yang digunakan penelitian ini adalah Java karena sifatnya yang *platform independent*. Pada bab ini akan dijelaskan spesifikasi lingkungan pengembangan sistem, batasan-batasan implementasi, implementasi kode sumber, dan implementasi antarmuka sistem.

6.1 Spesifikasi Lingkungan Pengembangan Sistem

Lingkungan pembangunan dan pengembangan sistem kakas bantu penilaian kopling *indirect package* dibagi ke dalam 2 kategori, yaitu perangkat keras dan perangkat lunak. Spesifikasi masing-masing lingkungan pengembangan sistem akan dijelaskan pada secara lebih lanjut pada sub-bab berikut:

6.1.1 Spesifikasi Perangkat Keras

Perangkat keras komputer yang digunakan dalam pembangunan dan pengembangan kakas bantu penilaian kopling *indirect package* memiliki spesifikasi seperti pada Tabel 6.1.

Tabel 6.1 Spesifikasi perangkat keras komputer

Nama Komponen	Spesifikasi
System Model	ASUS A55LN
Processor	Intel Core i5 4210U @ 1,70 ~ 2,4 GHz
Memory	8 GB RAM
Harddisk	1024 GB

6.1.2 Spesifikasi Perangkat Lunak

Perangkat lunak komputer yang digunakan dalam pembangunan dan pengembangan kakas bantu penilaian kopling *indirect package* memiliki spesifikasi seperti pada Tabel 6.2.

Tabel 6.2 Spesifikasi perangkat lunak komputer

Nama Komponen	Spesifikasi
Sistem Operasi	Windows 10 Enterprise 64-bit
Bahasa Pemrograman	Java
Penyunting Teks	Netbeans IDE 8.0.2 dengan JDK 8

6.1.3 Batasan Implementasi

Batasan dalam implementasi kakas bantu penilaian kopling *indirect package*, antara lain:

1. Sistem hanya menerima berkas dengan ekstensi ".xml" untuk dilakukan *parsing*
2. Data yang dibutuhkan sistem, seperti *package,* klas, dan relasi-relasinya sudah terdefinisi pada berkas xml.

6.2 Implementasi Kode Sumber

Implementasi kode sumber kakas bantu penilaian kopling *indirect package* dibagi menjadi 3 bagian utama, yaitu implementasi memasukkan berkas diagram *package,* implementasi *parsing xml*, dan implementasi menilai kopling. Implementasi kode sumber sistem ini menggunakan bahasa pemrograman Java dan menggunakan Netbeans versi 8.0.2 sebagai IDE.

6.2.1 Implementasi Memasukkan Berkas Diagram *Package*

Implementasi kode sumber untuk memasukkan berkas diagram *package* pada intinya ditunjukkan oleh ChooseAndValidateFile pada Tabel 6.3. Setelah pengguna menekan tombol bergambar berkas, sistem akan menjalankan ShowFileChooserView untuk meminta *view* menampilkan jendela *file chooser*. Setelah berkas dipilih, method ChooseAndValidateFile dijalankan untuk memvalidasi berkas yang telah dipilih pada jendela *file chooser*. Jika berkas yang dimasukkan memiliki ekstensi ".xml", sistem menampilkan informasi mengenai berkas dan tombol untuk melakukan *parsing*. Jika ekstensi berkas bukan merupakan ".xml" sistem akan menampilkan pesan error terkait ekstensi berkas tersebut.

Tabel 6.3 Kode Sumber ChooseAndValidateFile

No	Kode Sumber
1	`public void ChooseAndValidateFile(File file) {`
2	`   if (file.getName().endsWith(".xml")) {`
3	`      setXmlFile(file);`
4	`      fileInputView.ShowFileInfo(xmlFile.getName(), xmlFile.length());`
5	`   } else {`
6	`      String extension = "";`
7	`      int i = file.getName().lastIndexOf('.'); //mendapatkan ekstensi`
8	`berkas untuk ditampilkan pada pesan error`
9	`      extension = file.getName().substring(i);`
10	`      fileInputView.ShowExtensionErrorMessage(extension);`
11	`   }`
12	`}`

6.2.2 Implementasi *Parsing* Xml

Proses *parsing* xml dibagi menjadi beberapa tahapan-tahapan kecil. Kode sumber yang akan dijalankan setelah pengguna menekan tombol *parse* adalah bagian dari klas FileController, yaitu ParseXmlFile yang ditunjukkan pada Tabel 6.4. Jika tidak ada kesalahan, FindPackagesFromDoc yang tercantum pada pada Tabel 6.5 akan dipanggil

untuk menyimpan data seluruh package beserta klas-klasnya dari hasil *parsing* berkas xml ke dalam packageList. Lalu, FileController memanggil FindRelationsFromDoc yang ditunjukkan oleh Tabel 6.6 untuk melakukan *parsing* pada berkas dan menyimpan data relasi ke dalam relationList. Selanjutnya relasi pada relationList disimpan oleh masing-masing *package* yang terlibat pada packageList dengan menjalankan AssignRelationsToPackages yang implementasinya dapat dilihat pada Tabel 6.7. Hasil *parsing* sistem beserta informasi berkas ditampilkan dan tombol *parse* diganti menjadi *calculate*. Jika terjadi kesalahan pada saat menjalankan proses-proses sebelumnya, sistem akan menampilkan pesan error sesuai jenis kesalahan yang terjadi.

Tabel 6.4 Kode Sumber ParseXmlFile

No	Kode Sumber
1	`public void ParseXmlFile() {`
2	`   try {`
3	`      DocumentBuilderFactory dbFactory =`
4	`DocumentBuilderFactory.newInstance();`
5	`      DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();`
6	`      Document doc = dBuilder.parse(xmlFile);`
7	`      doc.getDocumentElement().normalize();`
8	`      PackageList packageList = FindPackagesFromDoc(doc);`
9	`      RelationList relationList = FindRelationsFromDoc(doc);`
10	`      packageList = packageList.AssignRelationsToPackages(relationList);`
11	`      fileInputView.setVisible(false);`
12	`      MainView mainView = new MainView();`
13	`      mainView.setExtendedState(JFrame.MAXIMIZED_BOTH);`
14	`      mainView.ShowFileInfo(xmlFile.getName(), xmlFile.length());`
15	`      parsedDataController = new ParsedDataController(mainView,`
16	`packageList);`
17	`      parsedDataController.ShowViewParsedData();`
18	`   }`
19	`   catch (NullPointerException npe) {`
20	`      npe.printStackTrace();`
21	`      fileInputView.ShowDataErrorMessage();`
22	`   }`
23	`   catch (IOException ioe) {`
24	`      ioe.printStackTrace();`
25	`      fileInputView.ShowIOErrorMessage();`
26	`   }`
27	`   catch (ParserConfigurationException pce) {`
28	`      pce.printStackTrace();`
29	`      fileInputView.ShowParserErrorMessage();`
30	`   }`
31	`   catch (SAXException saxe) {`
32	`      saxe.printStackTrace();`
33	`      fileInputView.ShowDocErrorMessage();`
34	`   }`
35	`}`

Tabel 6.5 Kode Sumber FindPackagesFromDoc

No	Kode Sumber
1	`public PackageList FindPackagesFromDoc(Document doc) {`
2	`    PackageList packageList = new PackageList();`
3	`    NodeList modelNodeList = doc.getElementsByTagName("Models");`
4	`    Element modelElement = (Element) modelNodeList.item(0);`
5	`    NodeList packageNodeList =`
6	`modelElement.getElementsByTagName("Package");`
7	`    packageList.AddParsedPackages(packageNodeList);`
8	`    return packageList;`
9	`}`

Tabel 6.6 Kode Sumber FindRelationsFromDoc

No	Kode Sumber
1	`public RelationList FindRelationsFromDoc(Document doc) {`
2	`   NodeList relationNodeList =`
3	`doc.getElementsByTagName("ModelRelationshipContainer");`
4	`   Element relationElement = (Element) relationNodeList.item(0);`
5	`   NodeList dependencyNodeList =`
6	`relationElement.getElementsByTagName("Dependency");`
7	`   NodeList associationNodeList =`
8	`relationElement.getElementsByTagName("Association");`
9	`   NodeList generalizationNodeList =`
10	`relationElement.getElementsByTagName("Generalization");`
11	`   NodeList realizationNodeList =`
12	`relationElement.getElementsByTagName("Realization");`
13	`   RelationList relationList = new RelationList();`
14	`   relationList.AddParsedRelations(dependencyNodeList);`
15	`   relationList.AddParsedRelations(realizationNodeList);`
16	`   relationList.AddParsedRelations(generalizationNodeList);`
17	`   relationList.AddParsedRelations(associationNodeList);`
18	`   return relationList;`
19	`}`

Tabel 6.7 Kode Sumber AssignRelationsToPackages

No	Kode Sumber
1	`public PackageList AssignRelationsToPackages(RelationList relationList) {`
2	`   int nDependency = 0, nGeneralization = 0, nRealization = 0,`
3	`nAssociation = 0, nComposite = 0, nAggregation = 0;`
4	`   for (int i = 0; i < relationList.size(); i++) {`
5	`      Relation relation = relationList.get(i);`
6	`      for (int j = 0; j < size(); j++) {`
7	`         PackageObj pkg = get(j);`
8	`         pkg.AddRelationToPackage(relation);`
9	`      }`
10	`      switch (relation.getType()) {`
11	`         case "Dependency":`
12	`            nDependency++;`
13	`            break;`
14	`         case "Generalization":`
15	`            nGeneralization++;`
16	`            break;`
17	`         case "Realization":`
18	`            nRealization++;`

```
19              break;
20           case "Association":
21              nAssociation++;
22              break;
23           case "Composite":
24              nComposite++;
25              break;
26           case "Aggregation":
27              nAggregation++;
28              break;
29        }
30     }
31     setNRelations(nDependency, nGeneralization, nRealization,
32 nAssociation, nComposite, nAggregation);
33     return this;
34 }
```

6.2.3 Implementasi Menilai Kopling

Kode sumber untuk menjalankan proses menilai kopling dibagi menjadi beberapa sub-proses. Setelah pengguna menekan tombol *Calculate, method* Calculate yang ditunjukkan pada Tabel 6.8 akan dijalankan. Setelah itu packageList memanggil CalculatePackagesInstability untuk melakukan perhitungan nilai *instability* seluruh *package* yang implementasinya tertera pada Tabel 6.9. Dilakukan perulangan untuk seluruh menghitung nilai *instability* tiap *package* dengan memanggil CalculatePackagesInstability yang implementasinya ditunjukkan oleh Tabel 6.10. Selanjutnya dilakukan perhitungan pada nilai *abstraction* seluruh *package* melalui CalculatePackagesAbstraction yang ditunjukkan pada Tabel 6.11. Nilai *abstraction* tiap *package* dihitung melalui CalculatePackageAbstraction seperti yang terlihat pada Tabel 6.12. Terakhir dilakukan perulangan untuk menghitung nilai *normalized distance* seluruh package dengan memanggil CalculatePackagesNormalizedDistance seperti yang tertera pada Tabel 6.13. Perhitungan nilai *normalized distance* tiap *package* berdasarkan nilai *instability* dan *abstraction* dilakukan dengan memanggil CalculatePackageNormalizedDistance yang implementasinya terlihat pada Tabel 6.14. Hasil perhitungan tersebut selanjutnya ditampilkan kepada pengguna.

Tabel 6.8 Kode Sumber Calculate

No	Kode Sumber
1	`public void Calculate() {`
2	`   try {`
3	`      packageList.CalculatePackagesInstability();`
4	`      packageList.CalculatePackagesAbstraction();`
5	`      packageList.CalculatePackagesNormalizedDistance();`
6	`      String detailedCalculationData =`
7	`packageList.getDetailedCalculationResults();`
8	`      String overallCalculationData =`
9	`packageList.getOverallCalculationResults();`
10	`      mainView.ShowCalculationResult(overallCalculationData,`
11	`detailedCalculationData);`
12	`   } catch (StackOverflowError soe) {`
13	`      soe.printStackTrace();`
14	`      mainView.ShowCyclicRelationErrorMessage();`
15	`   }`
16	`}`

Tabel 6.9 Kode Sumber CalculatePackagesInstability

No	Kode Sumber
1	`public void CalculatePackagesInstability() {`
2	`   for (int i = 0; i < this.size(); i++) {`
3	`      PackageObj pkg = this.get(i);`
4	`      pkg.setInRank(CalculateInRank(pkg));`
5	`      pkg.setOutRank(CalculateOutRank(pkg));`
6	`   }`
7	`   maxInRank = FindMaxInRank();`
8	`   minInRank = FindMinInRank();`
9	`   maxOutRank = FindMaxOutRank();`
10	`   minOutRank = FindMinOutRank();`
11	`   for (int i = 0; i < size(); i++) {`
12	`      PackageObj pkg = get(i);`
13	`      pkg.setNormalizedInRank(NormalizeInRank(pkg));`
14	`      pkg.setNormalizedOutRank(NormalizeOutRank(pkg));`
15	`   }`
16	`   for (int i = 0; i < size(); i++) {`
17	`      PackageObj pkg = get(i);`
18	`      pkg.setInstability(pkg.CalculatePackageInstability());`
19	`   }`
20	`}`

Tabel 6.10 Kode Sumber CalculatePackageInstability

No	Kode Sumber
1	`public double CalculatePackageInstability() {`
2	`   double instability = getNormalizedOutRank() / (getNormalizedInRank() +`
3	`getNormalizedOutRank());`
4	`   return instability;`
5	`}`

Tabel 6.11 Kode Sumber CalculatePackagesAbstraction

No	Kode Sumber
1	`public void CalculatePackagesAbstraction() {`
2	`   for (int i = 0; i < this.size(); i++) {`
3	`      PackageObj pkg = get(i);`
4	`      pkg.setAbstraction(pkg.CalculatePackageAbstraction());`
5	`   }`
6	`}`

Tabel 6.12 Kode Sumber CalculatePackageAbstraction

No	Kode Sumber
1	`public double CalculatePackageAbstraction() {`
2	`   double absInRankTotal = 0, inRankTotal = 0;`
3	`   getClassList().CalculateClassesInRank();`
4	`   for (int j = 0; j < getAbsClassList().size(); j++) {`
5	`      ClassObj absClass = getAbsClassList().get(j);`
6	`      absInRankTotal = absInRankTotal + absClass.getInRank();`
7	`   }`
8	`   for (int j = 0; j < getClassList().size(); j++) {`
9	`      ClassObj cls = getClassList().get(j);`
10	`      inRankTotal = inRankTotal + cls.getInRank();`
11	`   }`
12	`   getAbsClassList().setTotalInRank(absInRankTotal);`
13	`   getClassList().setTotalInRank(inRankTotal);`
14	`   double abstraction = absInRankTotal / inRankTotal;`
15	`   return abstraction;`
16	`}`

Tabel 6.13 Kode Sumber CalculatePackagesNormalizedDistance

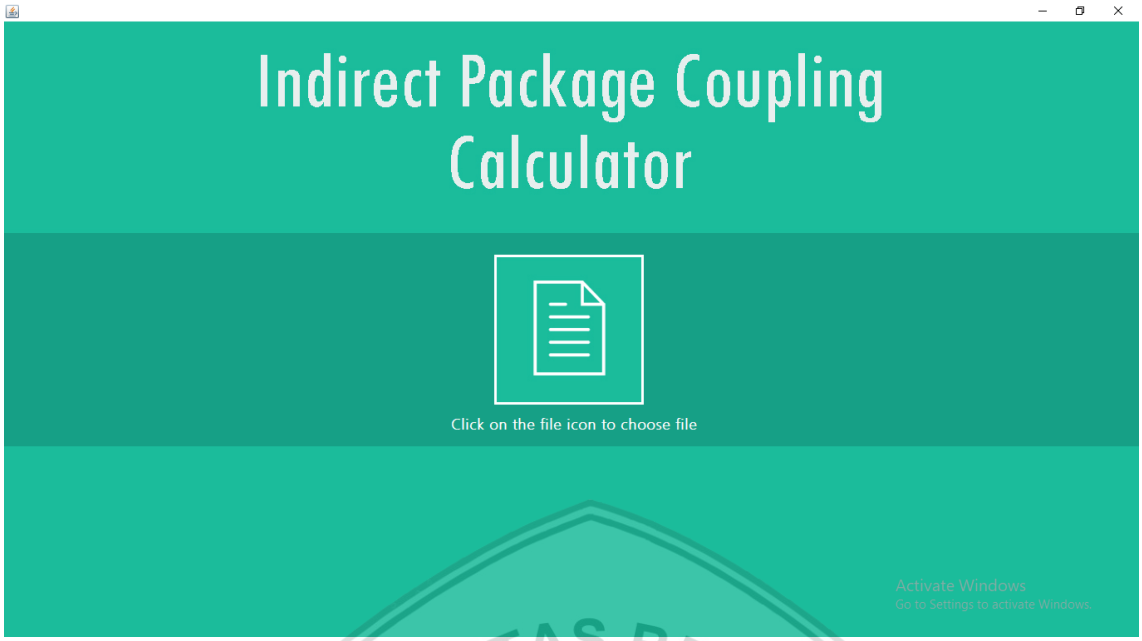
No	Kode Sumber
1	`public void CalculatePackagesNormalizedDistance() {`
2	`   for (int i = 0; i < size(); i++) {`
3	`     PackageObj pkg = get(i);`
4	`     pkg.setNormalizedDistance(pkg.CalculatePackageNormalizedDistance());`
5	`   }`
6	`}`

Tabel 6.14 Kode Sumber CalculatePackageNormalizedDistance

No	Kode Sumber
1	`public double CalculatePackageNormalizedDistance() {`
2	`   double normDistance = Math.abs(getAbstraction() + getInstability()-1);`
3	`   return normDistance;`
4	`}`

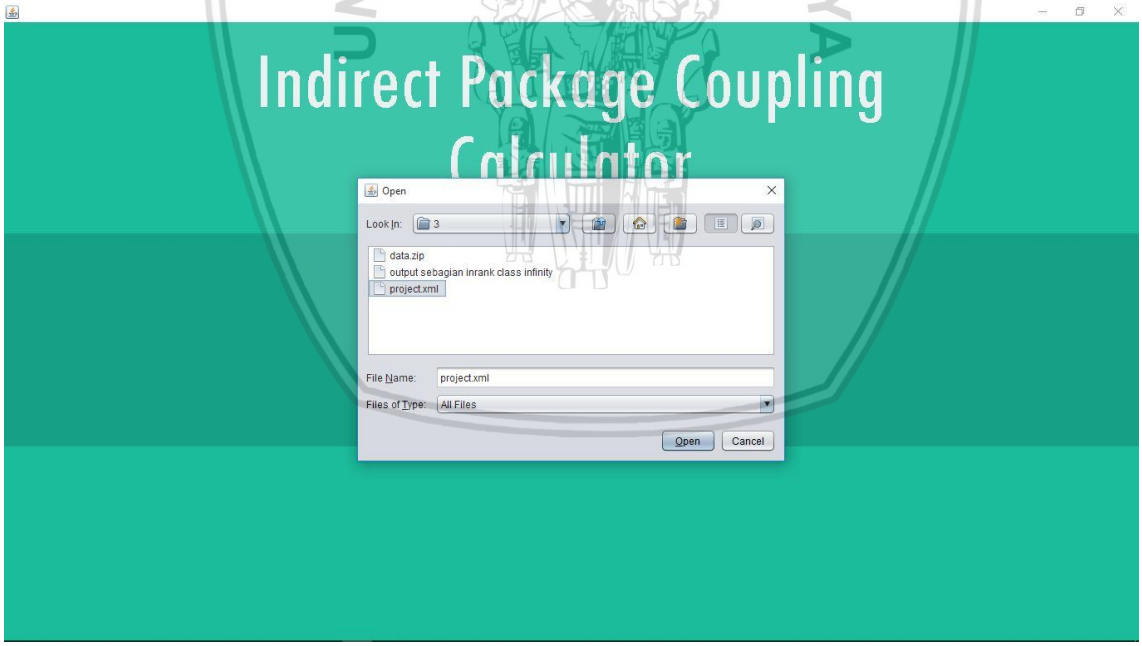
6.3 Implementasi Antarmuka Sistem

Setelah pengguna menjalankan kakas bantu, pengguna akan disambut dengan tampilan antarmuka seperti pada gambar 6.1 yang berisi nama kakas bantu dan tombol dengan ikon berkas untuk memilih berkas.



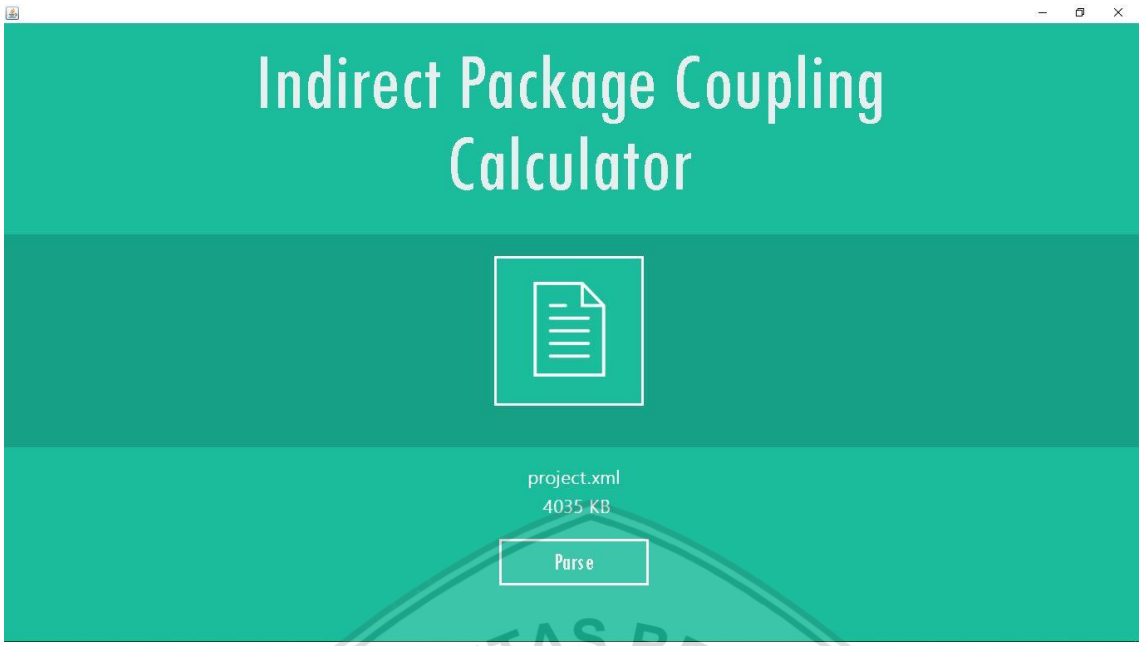
Gambar 6.1 Antarmuka saat sistem baru dijalankan

Setelah itu pengguna menekan tombol dengan gambar berkas untuk menampilkan *pop-up file chooser* seperti pada gambar 6.2. Pop-up tersebut menampilkan daftar berkas untuk dimasukkan ke dalam kakas bantu.



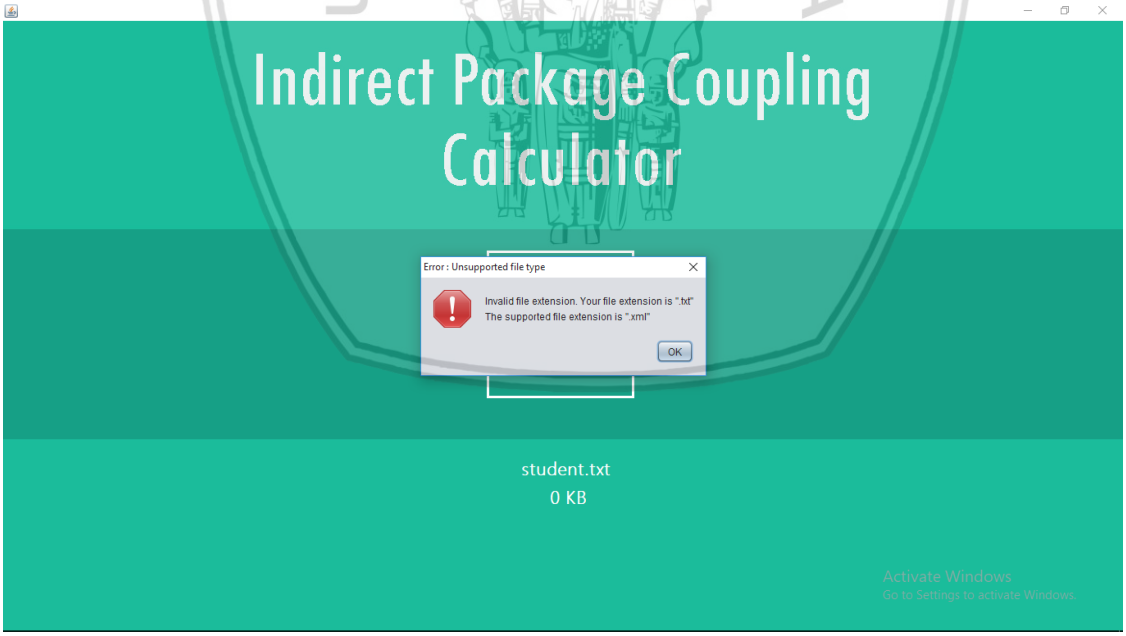
Gambar 6.2 Antarmuka memilih berkas untuk dimasukkan

Pengguna memilih berkas diagram *package* dengan ekstensi ".xml" yang ingin dinilai koplingnya. Sistem akan menampilkan informasi nama dan ukuran berkas serta tombol "parse" seperti pada gambar 6.3 jika berkas memiliki ekstensi ".xml".



Gambar 6.3 Antarmuka setelah berkas tervalidasi

Jika berkas bukan ".xml", sistem akan menampilkan pesan error seperti pada gambar 6.4. Selain itu tombol "parse" juga tidak muncul sehingga pengguna perlu memilih ulang berkas dengan menekan tombol *choose file*. Pesan error berisi informasi tentang kesalahan ekstensi dan jenis ekstensi yang dapat diproses oleh kakas bantu.

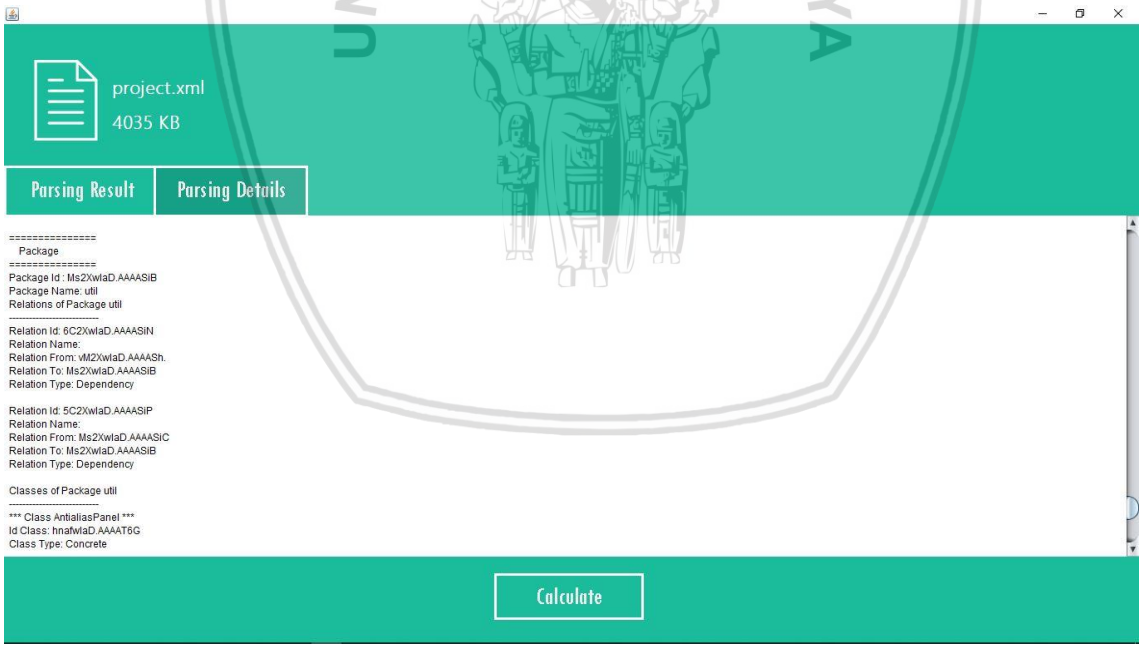


Gambar 6.4 Antarmuka pesan error jika ekstensi berkas bukan ".xml"

Apabila pengguna menekan tombol *parse,* sistem akan menampilkan antarmuka hasil *parsing* secara garis besar seperti pada gambar 6.5. *Tab* tersebut menunjukkan jumlah *package,* klas, klas abstrak, dan relasi.

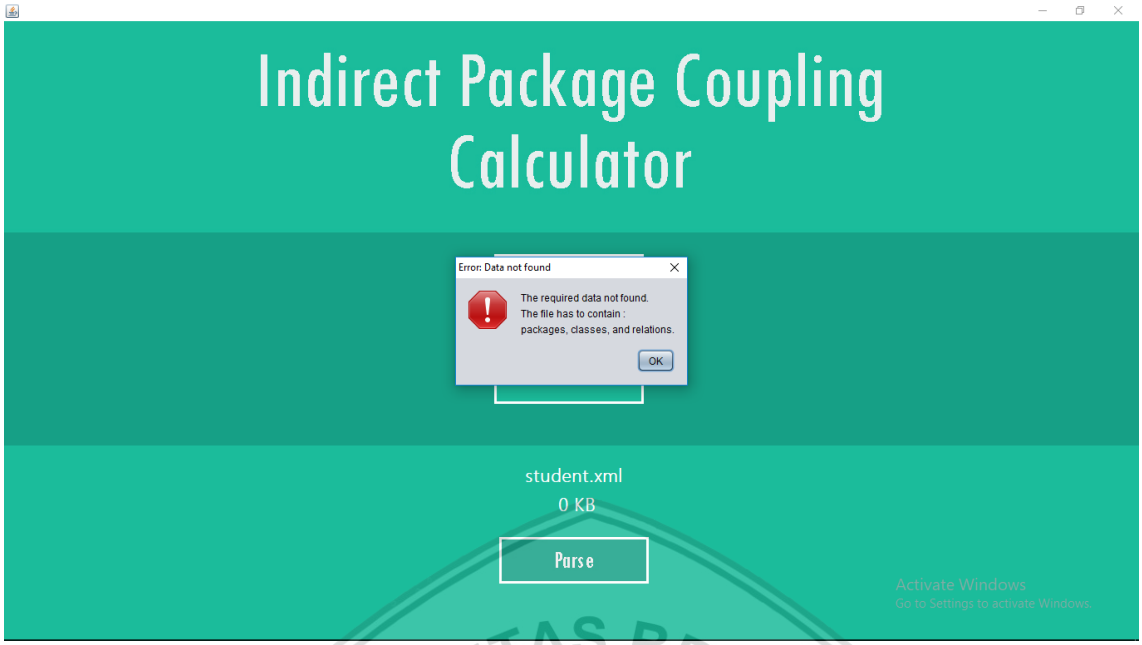
Gambar 6.5 Antarmuka hasil parsing berkas secara garis besar

Pengguna dapat melihat rincian hasil *parsing* dengan menekan *tab parsing details* seperti pada gambar 6.6. Pada *tab* ini terdapat informasi mengenai *package* dengan klas-klas dan relasi-relasinya.



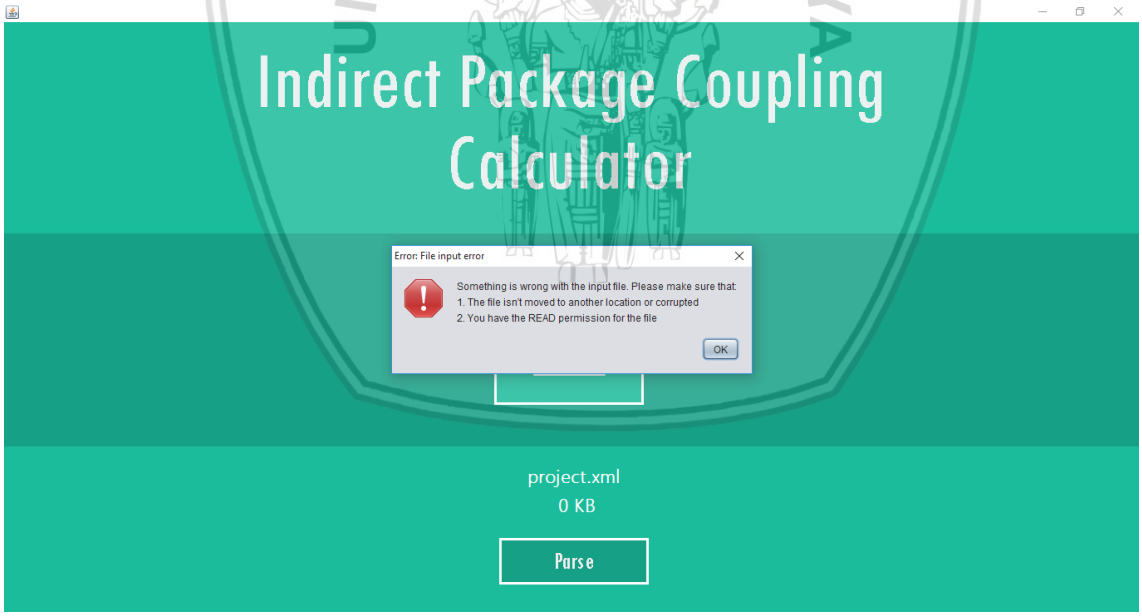
Gambar 6.6 Antarmuka hasil parsing berkas secara rinci

Jika data yang dibutuhkan sistem tidak ditemukan saat *parsing,* akan muncul pesan error seperti pada gambar 6.7. Pesan error tersebut berisi informasi bahwa data yang dibutuhkan tidak ditemukan dan informasi mengenai data apa saja yang harus ada dalam berkas.



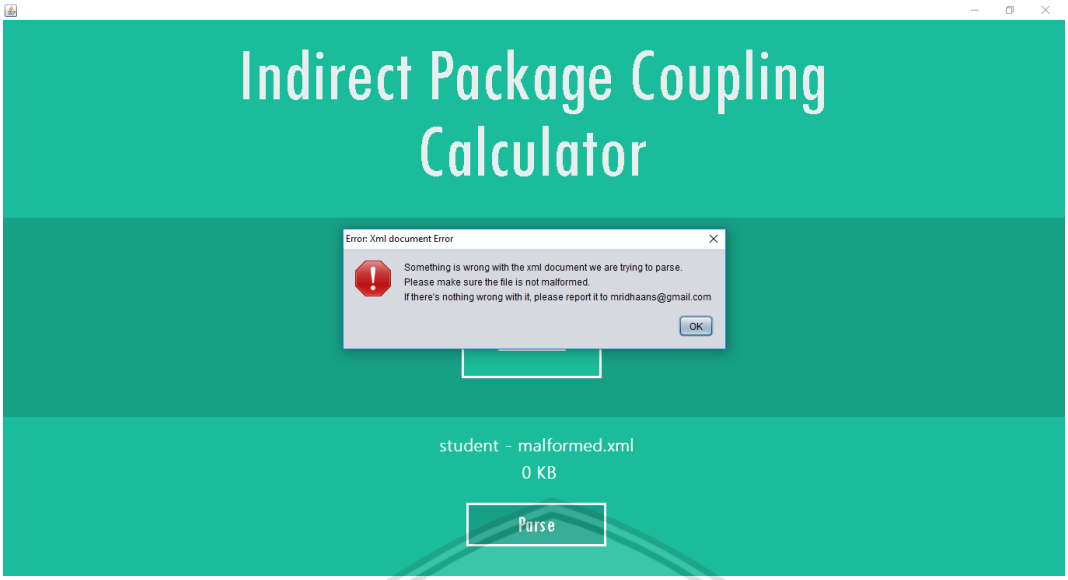
Gambar 6.7 Antarmuka pesan error jika data tidak ditemukan

Jika terdapat kesalahan pada kondisi berkas, akan muncul pesan error seperti pada gambar 6.8. Pesan error tersebut berisi informasi bahwa ada kesalahan pada berkas yang dimasukkan seperti perubahan lokasi atau *corrupt*.



Gambar 6.8 Antarmuka pesan error jika ada kesalahan berkas

Jika terdapat kesalahan struktur pada berkas xml, akan muncul pesan error seperti pada gambar 6.9. Pesan error tersebut berisi informasi bahwa ada kesalahan struktur pada berkas.

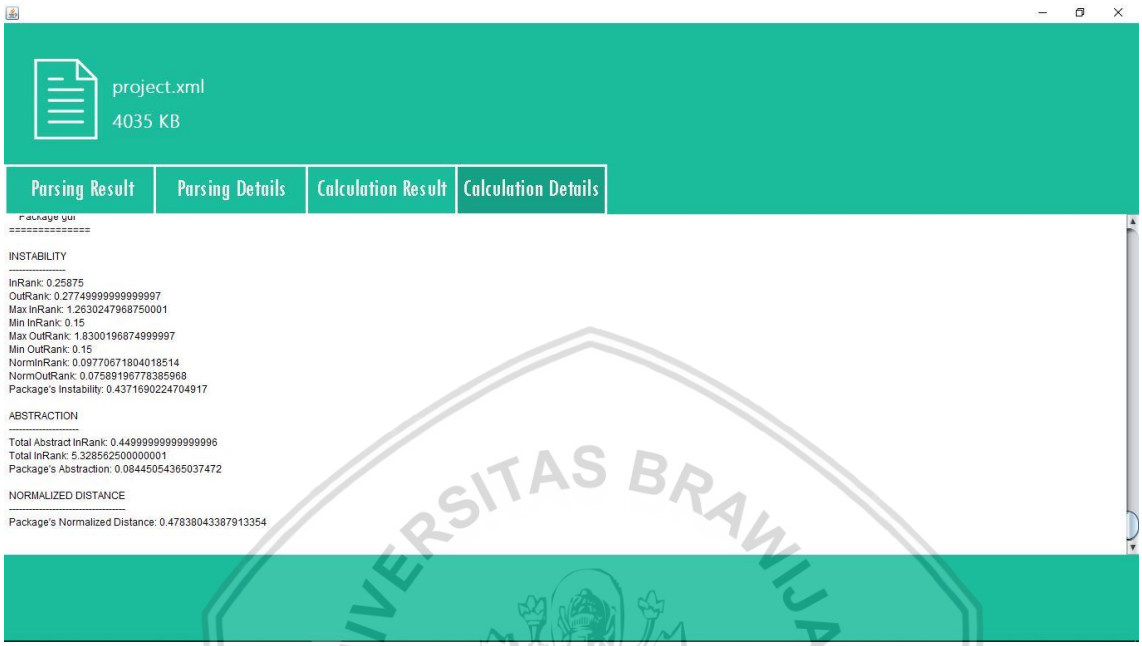


Gambar 6.9 Antarmuka pesan error jika ada kesalahan struktur xml berkas

Setelah itu pengguna dapat menekan tombol "Calculate" untuk melanjutkan ke tahap penilaian kopling. Sistem akan menampilkan hasil penilaiannya secara garis besar seperti pada gambar 6.10. Pada tab ini diberikan informasi mengenai hasil akhir perhitungan beserta hasil penilaiannya.

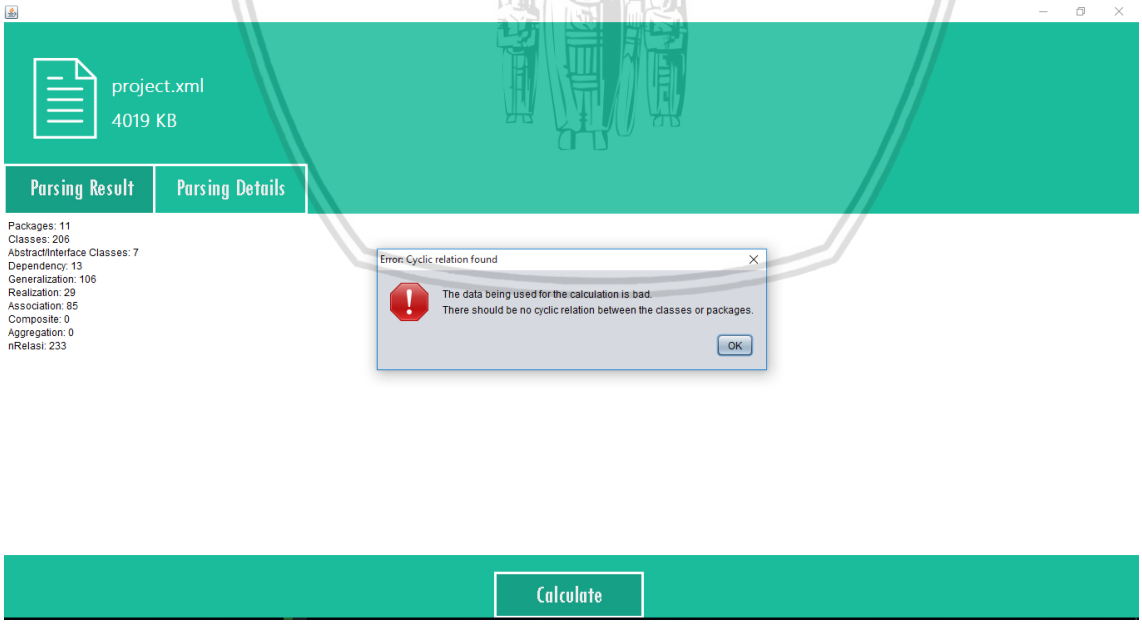
Gambar 6.10 Antarmuka hasil perhitungan kopling secara garis besar

Pengguna dapat melihat tampilan hasil perhitungan secara rinci dengan memilih *tab calculation details* seperti pada gambar 6.11. Disini terdapat informasi rinci hasil perhitungan-perhitungan metrik yang digunakan sampai memperoleh hasil akhirnya, yaitu *normalized distance*.



Gambar 6.11 Antarmuka hasil perhitungan kopling secara rinci

Jika saat perhitungan ditemukan relasi sirkular, akan muncul pesan error seperti pada gambar 6.12. Pesan error tersebut berisi informasi bahwa terdapat relasi sirkular.



Gambar 6.12 Antarmuka pesan error jika data tidak ditemukan

BAB 7 PENGUJIAN

Bab ini membahas tentang pengujian kakas bantu penilaian kopling *indirect package* yang telah dibangun pada bab implementasi. Pengujian dilakukan untuk mengetahui apakah perangkat lunak sudah benar dan sesuai dengan yang direncanakan. Selain itu, pengujian dapat memastikan bahwa sistem yang dibangun sudah layak untuk digunakan secara umum.

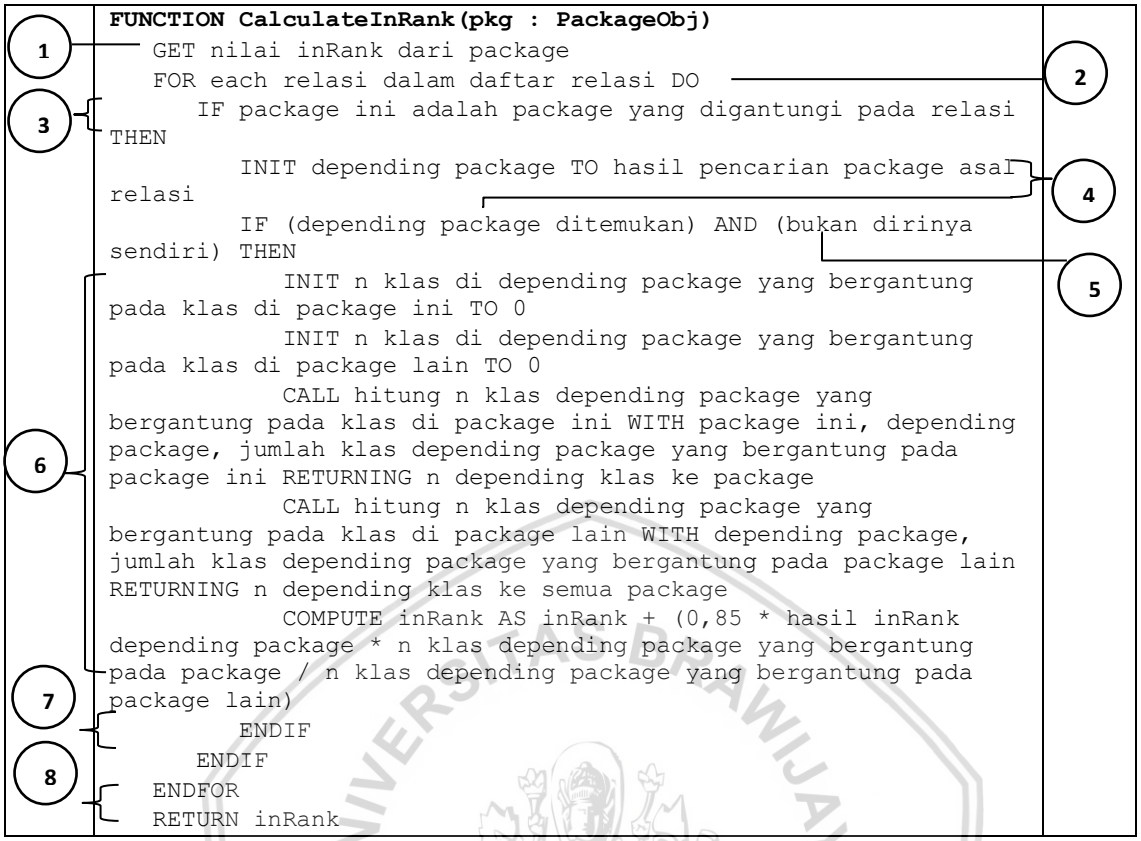
Pengujian yang dilakukan pada penelitian ini meliputi pengujian unit, pengujian integrasi, pengujian validasi, dan pengujian kecepatan pemrosesan oleh sistem jika dibandingkan dengan manusia. Pengujian unit dan integrasi terhadap kakas bantu ini dilakukan dengan teknik pengujian *white box*, sedangkan pengujian validasinya dilakukan dengan teknik pengujian *black box*.

7.1 Pengujian Unit

Pengujian unit sistem yang dibangun berorientasi objek dilakukan terhadap suatu *method* yang ada pada suatu klas. Pengujian unit terhadap kakas bantu penilaian kopling *indirect package* dilakukan dengan teknik *basis path testing* yang termasuk teknik *white box testing*. Teknik *basis path testing* dimulai dengan membuat gambaran alur logika dalam bentuk *flow graph*. *Flow graph* yang diperoleh dihitung nilai kompleksitasnya menggunakan metrik *cyclomatic complexity*. Nilai *cyclomatic complexity* tersebut digunakan untuk menentukan jalur *independent* dalam *basis set* suatu sistem dan memberi batasan banyaknya suatu pengujian yang diperlukan dalam bentuk kasus uji.

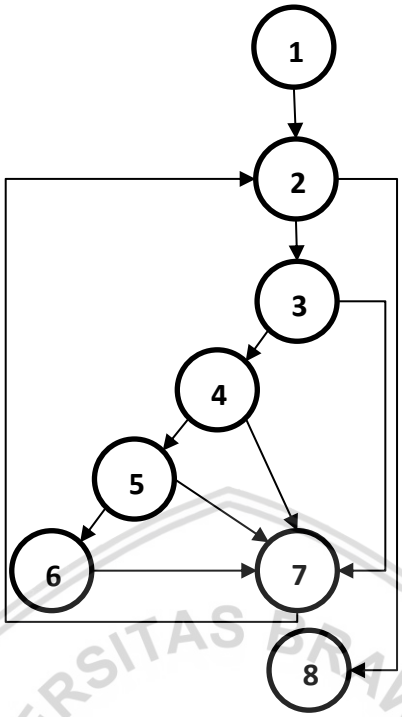
7.1.1 Pengujian *Method* CalculateInRank()

Flow graph method CalculateInRank() dapat dibentuk dengan menentukan *node* (simpul) yang ada pada algoritmenya. Pembuatan simpul yang ada pada *method* CalculateInRank() dapat dilihat pada gambar 7.1. Simpul 2 dibagi menjadi 3 bagian, yaitu simpul 2a untuk inisiasi indeks perulangan, simpul 2b untuk pemeriksaan apakah indeks perulangan mencapai nilai ukuran daftar relasi yang merupakan batas, dan simpul 2c untuk penambahan indeks perulangan.



Gambar 7.1 Simpul algoritme *method* CalculateInRank()

Berdasarkan simpul algoritme pada gambar 7.1 dapat diperoleh gambaran alur dari *method* dalam bentuk *flow graph*. Hasil penggambaran *flow graph* dari *method* ditunjukkan oleh gambar 7.2.



Gambar 7.2 *Flow graph method* CalculateInRank()

Perhitungan *cyclomatic complexity* V terhadap *flow graph* G dilakukan dengan persamaan V(G) = E – N + 2, dimana E adalah jumlah *edge* (garis penghubung antar simpul) dan N adalah jumlah simpul.

V(G) = 11 – 8 + 2 = 5 (7.1)

Dari hasil perhitungan tersebut dapat diketahui bahwa ada 4 basis set jalur *independent*. Jalur tersebut antara lain:

Jalur 1: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 2 – 8

Jalur 2: 1 – 2 – 8

Jalur 3: 1 – 2 – 3 – 7 – 2 – 8

Jalur 4: 1 – 2 – 3 – 4 – 7 – 2 – 8

Jalur 5: 1 – 2 – 3 – 4 – 5 – 7 – 2 – 8

Jalur-jalur tersebut diuji terhadap *method* CalculateInRank() melalui klas uji PackageListUnitTest. Klas uji tersebut memiliki 5 *method* uji dengan inisiasi variabel-variabel yang disesuaikan dengan kasus ujinya untuk menguji jalur *independent*.

Berdasarkan pengujian unit diperoleh tabel 7.1 yang merupakan tabel hasil pengujian *method* CalculateInRank().

Tabel 7.1 Hasil pengujian unit *method* CalculateInRank()

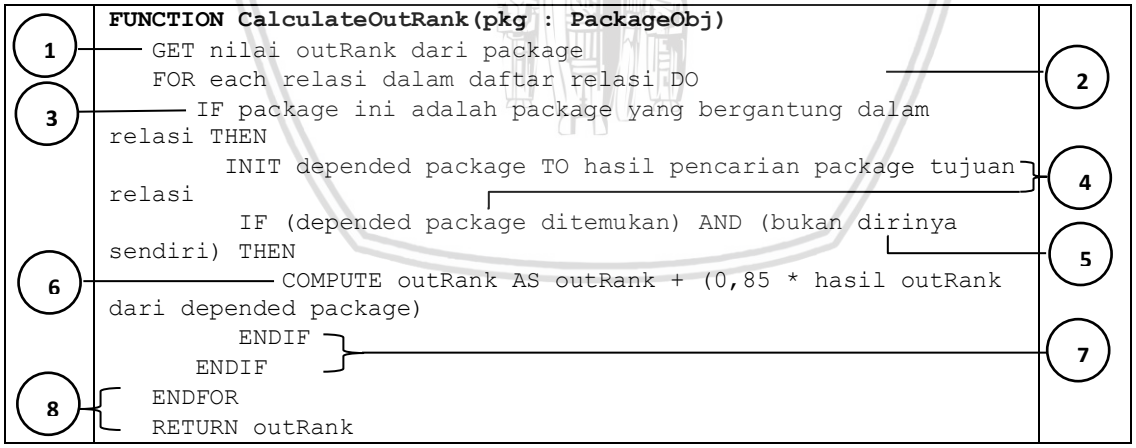
Jalur	Kasus uji	Hasil yang diharapkan	Hasil yang diperoleh
1	*Package* memiliki relasi, *package* ini digantungi, package yang bergantung ditemukan, dan *package* yang bergantung bukan dirinya sendiri. Jumlah relasi *package* ini= 1, Tujuan dari relasi= *package* ini, *Depending package* = *depending package*	Sistem menelusuri seluruh relasi, lalu mencari *depending package* dalam daftar *package*, menghitung nilai inRank, mengembalikan nilai inRank *package* ini dari hasil perhitungan. (1 – 2 – 3 – 4 – 5 – 6 – 7 – 2 – 8)	Sistem menelusuri seluruh relasi, lalu mencari *depending package* dalam daftar *package*, menghitung nilai inRank, mengembalikan hasil perhitungan yang benar dari nilai inRank *package* ini. (1 – 2 – 3 – 4 – 5 – 6 – 7 – 2 – 8)
2	*Package* tidak memiliki relasi. Jumlah relasi *package* ini = 0	Sistem tidak menelusuri relasi dan mengembalikan nilai inRank tanpa melakukan perhitungan. (1 – 2 – 8)	Sistem tidak menelusuri relasi dan mengembalikan nilai dasar inRank yaitu 0,15 karena tidak melakukan perhitungan. (1 – 2 – 8)
3	Package memiliki relasi, tapi dalam relasi ini bukan package ini yang digantungi. Jumlah relasi *package* ini = 1, Tujuan relasi = *depending package*	Sistem menelusuri daftar relasi, karena *package* ini tidak digantungi dalam relasi ini, perhitungan tidak dilakukan dan penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai inRank dari package ini. (1 – 2 – 3 – 7 – 2 – 8)	Sistem menelusuri seluruh relasi, tidak ada perhitungan nilai inRank karena *package* ini tidak digantungi dalam relasi ini, sehingga sistem melanjutkan penelusuran relasi, sistem mengembalikan nilai inRank dari package ini. (1 – 2 – 3 – 7 – 2 – 8)
4	Package memiliki relasi, dalam relasi ini yang digantungi adalah package ini, dan package yang bergantung tidak ditemukan Jumlah relasi *package* ini= 1, Tujuan relasi = *package* ini, *Depending package*= new PackageObj (“”,””)	Sistem menelusuri daftar relasi, lalu mencari *depending package*, *depending package* tidak ditemukan sehingga perhitungan tidak dilakukan dan penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai inRank dari *package* ini. (1 – 2 – 3 – 4 – 7 – 2 – 8)	Sistem menelusuri daftar relasi, mencari *depending package*, tidak dilakukan perhitungan inRank karena *depending package* tidak ditemukan, penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai inRank dari *package* ini. (1 – 2 – 3 – 4 – 7 – 2 – 8)

Jalur	Kasus uji	Hasil yang diharapkan	Hasil yang diperoleh
5	Package memiliki relasi, dalam relasi ini yang digantungi adalah package ini, package yang bergantung ditemukan, tapi yang bergantung adalah dirinya sendiri. Jumlah relasi *package* ini= 1, Tujuan relasi = *package* ini, *Depending package*= *package* ini	Sistem menelusuri daftar relasi, mencari *depending package*, karena package yang bergantung adalah dirinya sendiri, tidak dilakukan perhitungan inRank. penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai inRank dari *package* ini. (1 – 2 – 3 – 4 – 5 – 7 – 2 – 8)	Sistem menelusuri daftar relasi, mencari *depending package*, tidak dilakukan perhitungan inRank karena *depending package* yang ditemukan adalah dirinya sendiri, penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai inRank dari package ini. (1 – 2 – 3 – 4 – 5 – 7 – 2 – 8)

Tabel 7.1 menunjukkan bahwa semua hasil yang diperoleh sesuai dengan hasil yang diharapkan. Hal tersebut membuktikan bahwa semua jalur *independent* pada *method* CalculateInRank() bekerja dengan baik.

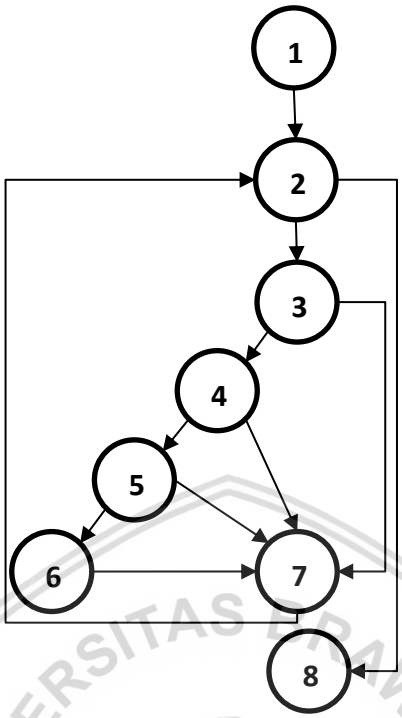
7.1.2 Pengujian *Method* CalculateOutRank()

Flow graph method CalculateOutRank() dapat dibentuk dengan menentukan *node* (simpul) yang ada pada algoritmenya. Simpul yang ada pada *method* CalculateOutRank() dapat dilihat pada gambar 7.3. Simpul 2 dibagi menjadi 3 bagian, yaitu simpul 2a untuk inisiasi indeks perulangan, simpul 2b untuk pemeriksaan apakah indeks perulangan mencapai nilai ukuran daftar relasi yang merupakan batas, dan simpul 2c untuk penambahan indeks perulangan.



Gambar 7.3 Simpul algoritme *method* CalculateOutRank()

Berdasarkan simpul algoritme pada gambar 7.3 dapat diperoleh gambaran alur dari *method* dalam bentuk *flow graph*. Hasil penggambaran *flow graph* dari *method* ditunjukkan oleh gambar 7.4.



Gambar 7.4 *Flow graph method* CalculateOutRank()

Perhitungan *cyclomatic complexity* V terhadap *flow graph* G dilakukan dengan persamaan V(G) = E – N + 2, dimana E adalah jumlah *edge* (garis penghubung antar simpul) dan N adalah jumlah simpul.

V(G) = 11 – 8 + 2 = 5 (7.2)

Dari hasil perhitungan tersebut dapat diketahui bahwa ada 5 basis set jalur *independent*. Jalur tersebut antara lain:

Jalur 1: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 2 – 8

Jalur 2: 1 – 2 – 8

Jalur 3: 1 – 2 – 3 – 7 – 2 – 8

Jalur 4: 1 – 2 – 3 – 4 – 7 – 2 – 8

Jalur 5: 1 – 2 – 3 – 4 – 5 – 7 – 2 – 8

Jalur-jalur tersebut diuji terhadap *method* CalculateOutRank() melalui klas uji PackageListUnitTest. Klas uji tersebut memiliki 5 *method* uji dengan inisiasi variabel-variabel yang disesuaikan dengan kasus ujinya untuk menguji jalur *independent*.

Berdasarkan pengujian unit yang dilakukan terhadap *method* CalculateOutRank() diperoleh tabel 7.2.

Tabel 7.2 Hasil pengujian unit *method* CalculateOutRank()

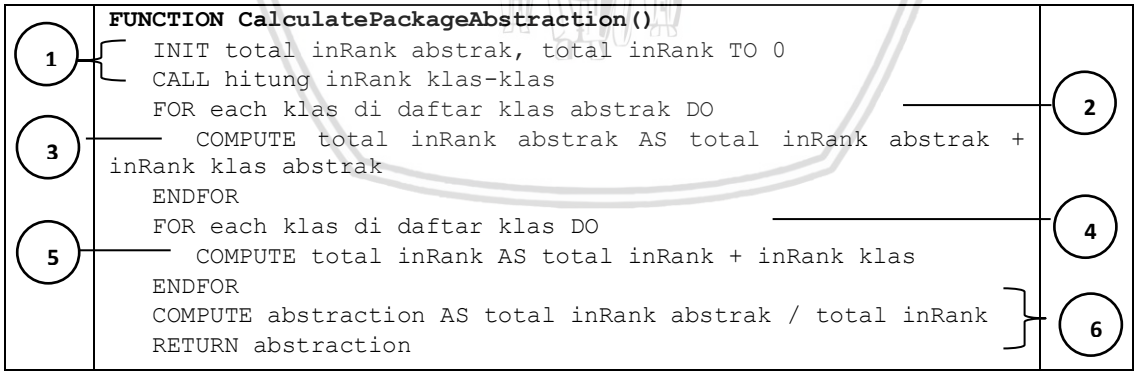
Jalur	Kasus uji	Hasil yang diharapkan	Hasil yang diperoleh
1	*Package* memiliki relasi, yang bergantung dalam relasi ini adalah *package* ini, *depended package* ditemukan dan bukan dirinya sendiri Jumlah relasi *package* ini= 1 , Asal relasi = *package* ini, *Depended package* = *depended package*	Sistem menelusuri seluruh relasi, lalu mencari *depended package* dalam daftar *package*, menghitung nilai outRank, mengembalikan nilai outRank *package* ini dari hasil perhitungan. (1 – 2 – 3 – 4 – 5 – 6 – 7 – 2 – 8)	Sistem menelusuri seluruh relasi, lalu mencari *depended package* dalam daftar *package*, menghitung nilai outRank, mengembalikan hasil perhitungan yang benar dari nilai outRank *package* ini. (1 – 2 – 3 – 4 – 5 – 6 – 7 – 2 – 8)
2	*Package* tidak memiliki relasi. Jumlah relasi *package* ini= 0	Sistem tidak menelusuri relasi dan mengembalikan nilai outRank tanpa melakukan perhitungan. (1 – 2 – 8)	Sistem tidak menelusuri relasi dan mengembalikan nilai dasar outRank yaitu 0,15 karena tidak melakukan perhitungan. (1 – 2 – 8)
3	*Package* memiliki relasi, tapi dalam relasi ini bukan *package* ini yang bergantung. Jumlah relasi *package* ini=1, asal relasi = *depended package*	Sistem menelusuri daftar relasi, karena *package* ini tidak bergantung dalam relasi ini, perhitungan tidak dilakukan dan penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai outRank dari package ini. (1 – 2 – 3 –7 – 2 – 8)	Sistem menelusuri seluruh relasi, tidak ada perhitungan nilai outRank karena *package* ini tidak bergantung dalam relasi ini, sehingga sistem melanjutkan penelusuran relasi, sistem mengembalikan nilai outRank dari package ini. (1 – 2 – 3 – 7 – 2 – 8)
4	*Package* memiliki relasi, dalam relasi ini yang bergantung adalah *package* ini, tapi *depended package* tidak ditemukan. Jumlah relasi *package* ini=1, asal relasi = *package* ini, *Depended package*= new PackageObj(“”,””)	Sistem menelusuri daftar relasi, mencari *depended package*, tidak dilakukan perhitungan inRank karena *depended package* tidak ditemukan, penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai outRank dari *package* ini. (1 – 2 – 3 – 4 – 7 – 2 – 8)	Sistem menelusuri daftar relasi, mencari *depended package*, tidak dilakukan perhitungan outRank karena *depended package* tidak ditemukan, penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai outRank dari *package* ini. (1 – 2 – 3 – 4 –7 – 2 – 8)

Jalur	Kasus uji	Hasil yang diharapkan	Hasil yang diperoleh
5	*Package* memiliki relasi, dalam relasi ini yang bergantung adalah *package* ini, *depended package* ditemukan, tapi *package* yang digantungi adalah dirinya sendiri. Jumlah relasi *package* ini=1, asal relasi = *package* ini, *Depended package*= *package* ini	Sistem menelusuri daftar relasi, mencari *depended package*, tidak dilakukan perhitungan inRank karena *depended package* yang ditemukan adalah dirinya sendiri, penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai outRank dari *package* ini. (1 – 2 – 3 – 4 – 5 – 7 – 2 – 8)	Sistem menelusuri daftar relasi, mencari *depended package*, tidak dilakukan perhitungan outRank karena *depended package* yang digantungi adalah dirinya sendiri, penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai outRank dari *package* ini. (1 – 2 – 3 – 4 – 5 – 7 – 2 – 8)

Tabel 7.2 menunjukkan bahwa semua hasil yang diperoleh sesuai dengan hasil yang diharapkan. Hal tersebut membuktikan bahwa semua jalur *independent* pada *method* CalculateOutRank() bekerja dengan baik.

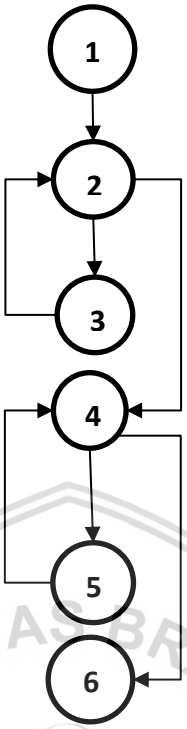
7.1.3 Pengujian *Method* CalculatePackageAbstraction ()

Flow graph method CalculatePackageAbstraction() dapat dibentuk dengan menentukan *node* (simpul) yang ada pada algoritmenya. Simpul yang ada pada method CalculateOutRank() dapat dilihat pada gambar 7.4. Simpul 2 dan 4 dibagi menjadi 3 bagian, yaitu simpul 2a dan 4a untuk inisiasi indeks perulangan, simpul 2b untuk pemeriksaan apakah indeks perulangan mencapai nilai terbesar daftar klas abstrak dan 4b untuk daftar klas menyeluruh, serta simpul 2c dan 4c untuk penambahan indeks perulangan.



Gambar 7.5 Simpul algoritme *method* CalculatePackageAbstraction()

Berdasarkan simpul algoritme pada gambar 7.5 dapat diperoleh gambaran alur dari *method* dalam bentuk *flow graph*. Hasil penggambaran *flow graph* dari *method* ditunjukkan oleh gambar 7.6.



Gambar 7.6 *Flow graph method* CalculatePackageAbstraction()

Perhitungan *cyclomatic complexity* V terhadap *flow graph* G dilakukan dengan persamaan V(G) = E – N + 2, dimana E adalah jumlah *edge* (garis penghubung antar simpul) dan N adalah jumlah simpul. Berikut adalah perhitungan *cyclomatic complexity* berdasarkan *flow graph method* CalculatePackageAbstraction() pada gambar 7.6.

V(G) = 7 – 6 + 2 = 3 (7.3)

Dari hasil perhitungan tersebut dapat diketahui bahwa ada 3 basis set jalur *independent*. Jalur tersebut antara lain:

Jalur 1: 1 – 2 – 3 –2 – 4 – 5 – 4 – 6

Jalur 2: 1 – 2 – 3 – 2 – 4 – 6

Jalur 3: 1 – 2 – 4 – 6

Jalur-jalur tersebut diuji terhadap method CalculatePackageAbstraction() melalui klas uji PackageObjUnitTest. Klas uji tersebut memiliki 3 *method* uji dengan inisiasi variabel-variabel yang disesuaikan dengan kasus ujinya untuk menguji jalur *independent*.

Berdasarkan pengujian unit dapat diperoleh tabel 7.3 yang merupakan tabel hasil pengujian method CalculatePackageAbstraction().

Tabel 7.3 Hasil pengujian unit *method* CalculatePackageAbstraction()

Jalur	Kasus uji	Hasil yang diharapkan	Hasil yang diperoleh
1	Package memiliki klas di daftar klas abstrak/*interface* dan klas di daftar klas menyeluruh. Jumlah klas abstrak = 1, Jumlah klas = 1	Sistem menelusuri klas di daftar klas abstrak/interface suatu package untuk menghitung total inRank mereka, lalu menelusuri seluruh klas di daftar klas keseluruhan suatu package dan menghitung total inRank mereka, lalu sistem menghitung nilai *abstraction* dan mengembalikannya. (1 – 2 – 3 –2 – 4 – 5 – 4 – 6)	Sistem menelusuri klas di daftar klas abstrak/interface suatu package untuk menghitung total inRank mereka, lalu menelusuri seluruh klas di daftar klas keseluruhan suatu package dan menghitung total inRank mereka, lalu sistem menghitung nilai *abstraction* dan mengembalikannya. (1 – 2 – 3 –2 – 4 – 5 – 4 – 6)
2	Package memiliki klas biasa tapi tidak memiliki klas abstrak/*interface*. Jumlah klas abstrak = 0, Jumlah klas = 1	Sistem menelusuri seluruh klas di daftar klas abstrak/*interface* suatu package dan menghitung total inRank mereka, lalu sistem menghitung nilai *abstraction* yang hasilnya adalah 0 dan mengembalikannya. (1 – 2 – 4 – 5 – 4 – 6)	Sistem menelusuri seluruh klas di daftar klas abstrak/*interface* suatu package dan menghitung total inRank mereka, lalu hasil perhitungan nilai *abstraction* adalah 0 dan mengembalikannya. (1 – 2 – 4 – 5 – 4 – 6)
3	Package tidak memiliki klas di daftar klas abstrak/*interface* dan di daftar klas menyeluruh. Jumlah klas abstrak = 0, Jumlah klas = 0	Sistem tidak menghitung total inRank klas abstrak dan klas keseluruhan, lalu sistem menghitung nilai *abstraction* yang hasilnya adalah 0 dan mengembalikannya. (1 – 2 – 4 – 6)	Sistem tidak menghitung total inRank klas abstrak dan klas keseluruhan suatu package, lalu dilakukan perhitungan nilai *abstraction* yang hasilnya adalah 0 dan mengembalikannya. (1 – 2 – 4 – 6)

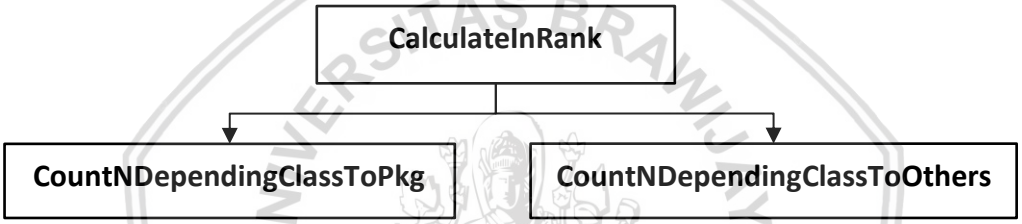
Tabel 7.3 menunjukkan bahwa semua hasil yang diperoleh sesuai dengan hasil yang diharapkan. Hal tersebut membuktikan bahwa semua jalur *independent* pada *method* CalculatePackageAbstraction() bekerja dengan baik.

7.2 Pengujian Integrasi

Pengujian integrasi sistem yang dibangun berorientasi objek dilakukan terhadap beberapa *method* yang bekerjasama untuk memperoleh suatu hasil, baik dari satu klas yang sama maupun klas yang berbeda. Pendekatan yang digunakan pada pengujian integrasi kakas bantu penilaian kopling *indirect package* adalah *top down*.

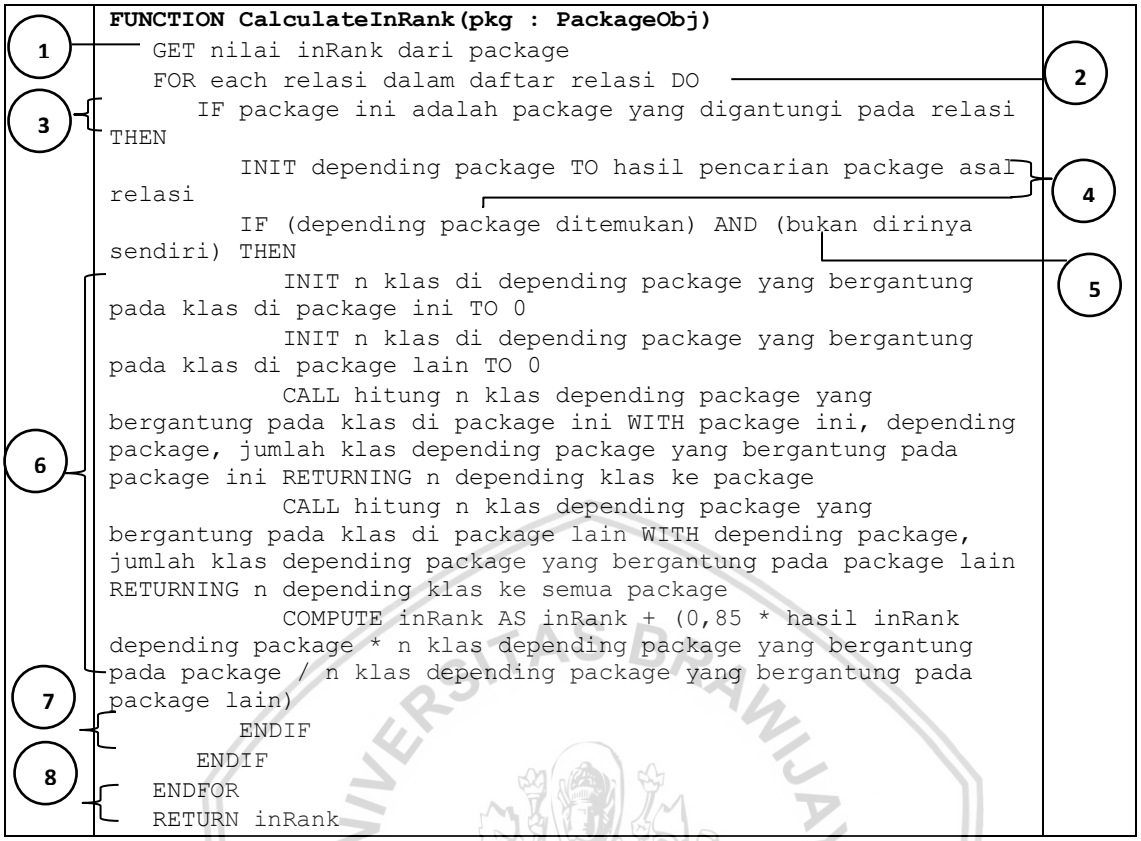
7.2.1 Pengujian *Method* CalculateInRank()

Method CalculateInRank() terintegrasi dengan beberapa *method* untuk memperoleh hasil operasinya. Beberapa *method* tersebut antara lain, CountNDependingClassToPkg() untuk menghitung jumlah klas di *depending package* yang bergantung pada klas di *depended package* dan CountNDependingClassToOthers() untuk menghitung klas di *depending package* yang bergantung pada *package* lain. Dua *method* tersebut merupakan *method* utama yang terintegrasi dengan CalculateInRank pada kakas bantu ini. Diagram hirarki dari *method* yang terintegrasi tesebut dapat dilihat pada gambar 7.7.



Gambar 7.7 Diagram hirarki pengujian integrasi method CalculateInRank()

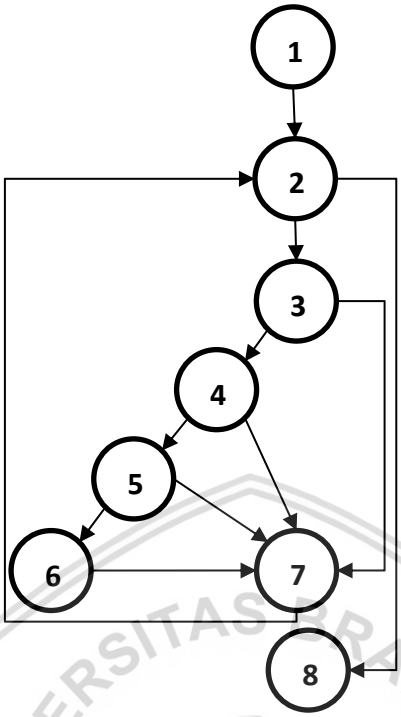
Pada pendekatan *top down,* dibuat kode sumber untuk menguji integrasi pada *method* CalculateInRank() yang dalam kasus ini akan diberi nama klas PackageListTest yang berisi beberapa *method* sesuai dengan kasus ujinya. Kasus uji tersebut diperoleh berdasarkan nilai *cyclomatic complexity* dari *method* CalculateInRank(). Simpul-simpul algoritme method CalculateInRank() yang digunakan untuk membuat *flow graphnya* dapat dilihat pada gambar 7.8.



Gambar 7.8 Simpul algoritme method CalculateInRank()

Simpul 2 dibagi menjadi 3 bagian, yaitu simpul 2a untuk inisiasi indeks perulangan, simpul 2b untuk pemeriksaan apakah indeks perulangan mencapai nilai ukuran daftar relasi yang merupakan batas, dan simpul 2c untuk penambahan indeks perulangan. Integrasi dengan method CountNDependingClassToPkg() ditunjukkan oleh simpul 5 pada baris "CALL hitung n klas di depending package yang bergantung pada klas di package ini...". Integrasi dengan method CountNDependingClassToOthers () ditunjukkan pada simpul 5 pada baris "CALL hitung n klas di depending package yang bergantung pada klas di package lain...".

Berdasarkan simpul algoritme pada gambar 7.8 dapat diperoleh gambaran alur dari *method* CalculateInRank dalam bentuk *flow graph*. Hasil penggambaran *flow graph* dari *method* tersebut ditunjukkan oleh gambar 7.9.



Gambar 7.9 *Flow graph method* CalculateInRank()

Perhitungan *cyclomatic complexity* V terhadap *flow graph* G dilakukan dengan persamaan V(G) = E – N + 2, dimana E adalah jumlah *edge* (garis penghubung antar simpul) dan N adalah jumlah simpul.

V(G) = 11 – 8 + 2 = 5 (7.4)

Dari hasil perhitungan tersebut dapat diketahui bahwa ada 5 basis set jalur *independent*. Jalur tersebut antara lain:

Jalur 1: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 2 – 8

Jalur 2: 1 – 2 – 8

Jalur 3: 1 – 2 – 3 – 7 – 2 – 8

Jalur 4: 1 – 2 – 3 – 4 – 7 – 2 – 8

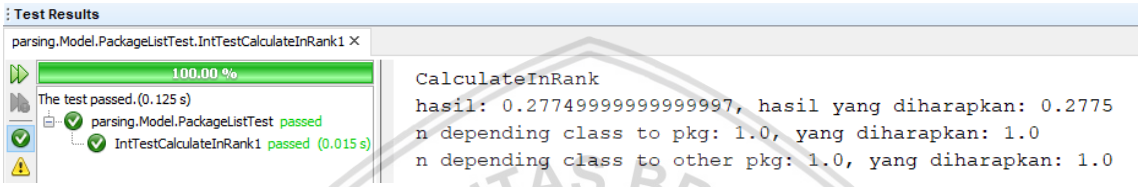
Jalur 5: 1 – 2 – 3 – 4 – 5 – 7 – 2 – 8

Jalur-jalur tersebut diuji terhadap method CalculateInRank() melalui klas uji yang dibentuk otomatis menggunakan *library* Junit. Klas uji tersebut dimodifikasi dengan menambahkan inisiasi variabel-variabel untuk menguji jalur *independent*.

Pada jalur pertama p*ackage* A memiliki klas A1 yang bergantung pada klas B1 di *package* B. Package B digantungi oleh *package* A yang memilki inRank 0,15 dan klas di package A yang bergantung pada *package* B adalah 1. *Package* A memiliki 1 klas yang bergantung pada klas di *package* lain. Nilai variabel yang sudah diinisiasikan tersebut digunakan dalam perhitungan inRank *package* B. Kondisi perulangan pertama terpenuhi

karena jumlah relasi lebih dari 0. Kondisi ke-2 terpenuhi karena *package* B digantungi dalam relasi. Kondisi ke-3 terpenuhi karena *package* B tidak digantungi dirinya sendiri dalam relasi. Hasil perhitungan inRank *package* B yang diharapkan adalah 0,2775 dan hasil yang diperoleh adalah 0,2774999 yang dapat dibulatkan menjadi 0,2775. Hasil tersebut dapat dilihat pada tabel 7.4.

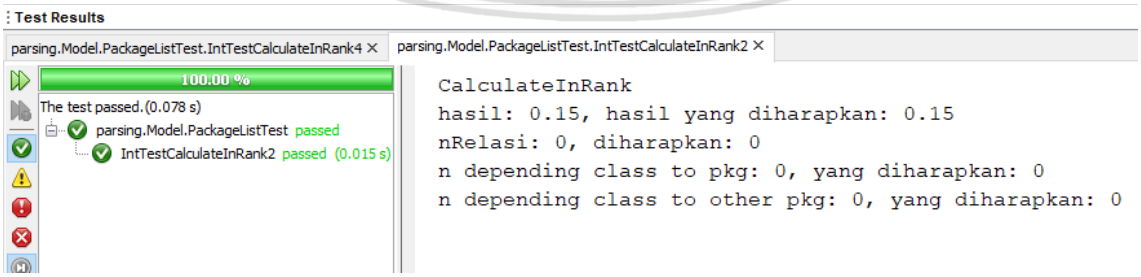
Setelah itu *method* uji ini menguji method CountNDependingClassToPkg() dan CountNDependingClassToOthers() yang dipanggil melalui CalculateInRank() apakah sudah bekerja dengan benar. Hasil method CountNDependingClassToPkg() yang diharapkan sesuai dengan yang diperoleh yaitu 1. Hasil method CountNDependingClassToOthers yang diharapkan juga sesuai dengan yang diperoleh yaitu 1. Hasil tersebut dapat dilihat pada gambar 7.10.



Gambar 7.10 Hasil pengujian integrasi kasus uji pertama *method* CalculateInRank()

Pada jalur ke-2 masing-masing p*ackage* A dan *package* B tidak memiliki relasi. *Package* B tidak memiliki relasi sehingga tidak dilakukan perhitungan inRank dan pemanggilan terhadap *method* CountNDependingClassToPkg() maupun CountNDependingClassToOthers(). Kondisi perulangan pertama tidak terpenuhi karena jumlah relasi tidak lebih dari 0 sehingga tidak dilakukan perhitungan inRank. Nilai inRank *package* B yang diharapkan adalah nilai dasarnya yaitu 0,15 dan hasil yang diperoleh adalah 0,15. Jumlah relasi di daftar relasi yang diharapkan sesuai dengan yang diperoleh, yaitu 0.

Setelah itu dilakukan pengujian terhadap *method* CountNDependingClassToPkg() dan CountNDependingClassToOthers() untuk memastikan apakah mereka sudah bekerja dengan benar. Hasil yang diperoleh pada masing-masing *method* tersebut sesuai dengan hasil yang diharapkan ketika tidak ada relasi, yaitu 0. Hasil tersebut dapat dilihat pada gambar 7.11.

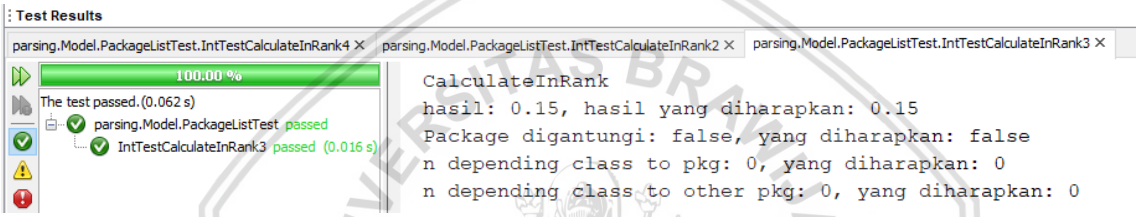


Gambar 7.11 Hasil pengujian integrasi kasus uji ke-2 *method* CalculateInRank()

Pada jalur ke-3 p*ackage* A memiliki klas A1 yang bergantung pada klas B1 di *package* B. Package A bukan *package* yang digantungi dalam relasi A1B1 sehingga tidak dilakukan perhitungan inRank dan pemanggilan terhadap *method* CountNDependingClassToPkg()

maupun CountNDependingClassToOthers(). Kondisi perulangan pertama terpenuhi karena jumlah relasi lebih dari 0. Selanjutnya penelusuran relasi gagal dilanjutkan karena tidak ada relasi lain. Kondisi ke-2 tidak terpenuhi karena *package* A tidak digantungi dalam relasi. Selanjutnya penelusuran relasi gagal dilanjutkan karena tidak ada relasi lain. Nilai inRank *package* A yang diharapkan adalah nilai dasarnya yaitu 0,15 karena tidak dilakukan perhitungan dan hasil yang diperoleh adalah 0,15. Package A teruji tidak digantungi oleh B melalui hasil perbandingan id tujuan relasi dengan id *package* yang bernilai *false* sesuai hasil yang diharapkan.

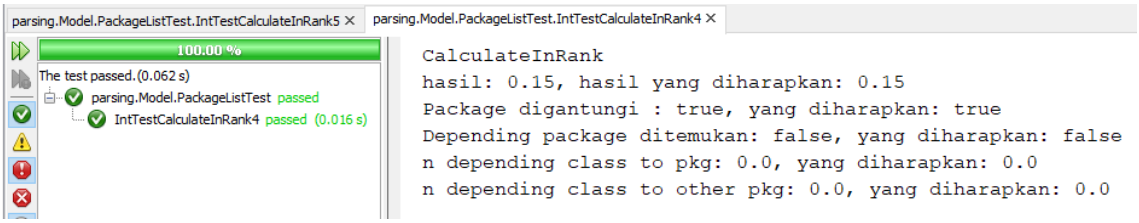
Pada *method* uji ini tetap dilakukan pemanggilan terhadap *method* CountNDependingClassToPkg() dan CountNDependingClassToOthers() untuk memastikan apakah mereka sudah bekerja dengan benar. Hasil yang diperoleh pada masing-masing *method* tersebut sesuai dengan hasil yang diharapkan ketika package A bukan yang digantungi dalam relasi, yaitu 0. Hasil tersebut dapat dilihat pada gambar 7.12.



Gambar 7.12 Hasil pengujian integrasi kasus uji ke-3 *method* CalculateInRank()

Pada jalur ke-4 p*ackage* A digantungi dalam relasi dan *depending package* tidak ditemukan sehingga tidak dilakukan perhitungan nilai inRank dan pemanggilan terhadap *method* CountNDependingClassToPkg() maupun CountNDependingClassToOthers(). Kondisi perulangan pertama terpenuhi karena jumlah relasi lebih dari 0. Kondisi ke-2 terpenuhi karena *package* A digantungi dalam relasi. Kondisi ke-3 tidak terpenuhi karena *depending package* tidak ditemukan. Selanjutnya penelusuran relasi gagal dilanjutkan karena tidak ada relasi lain. Nilai inRank *package* A yang diharapkan adalah nilai dasarnya yaitu 0,15 dan hasil yang diperoleh adalah 0,15.

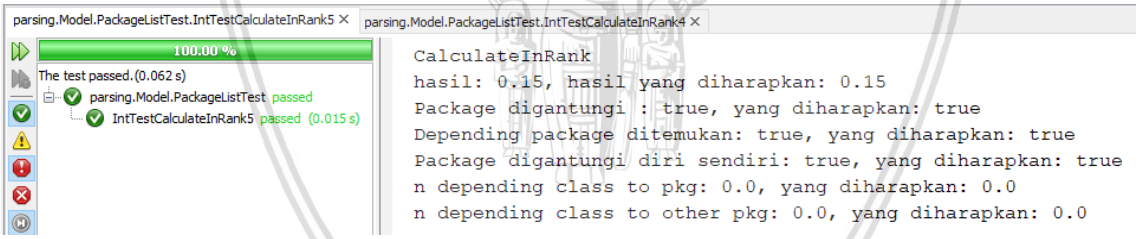
Pada *method* uji ini tetap dilakukan pemanggilan terhadap *method* CountNDependingClassToPkg() dan CountNDependingClassToOthers() untuk memastikan apakah mereka sudah bekerja dengan benar. Hasil yang diperoleh pada masing-masing *method* tersebut sesuai dengan hasil yang diharapkan ketika *package* yang bergantung kepada *package* A tidak ditemukan, yaitu 0. Hasil tersebut dapat dilihat pada gambar 7.13.



Gambar 7.13 Hasil pengujian integrasi kasus uji ke-4 *method* CalculateInRank()

Pada jalur ke-5 p*ackage* A memiliki klas A1 yang bergantung pada dirinya sendiri sehingga tidak dilakukan perhitungan nilai inRank dan pemanggilan terhadap *method* CountNDependingClassToPkg() maupun CountNDependingClassToOthers(). Kondisi perulangan pertama terpenuhi karena jumlah relasi lebih dari 0. Kondisi ke-2 terpenuhi karena *package* B digantungi dalam relasi. Kondisi ke-3 tidak terpenuhi karena *package* B digantungi dirinya sendiri dalam relasi sehingga tidak dicetak. Selanjutnya penelusuran relasi gagal dilanjutkan karena tidak ada relasi lain. Nilai inRank *package* A yang diharapkan adalah nilai dasarnya yaitu 0,15 dan hasil yang diperoleh adalah 0,15. Package A teruji digantungi oleh dirinya sendiri melalui hasil perbandingan id tujuan relasi dengan id asal relasi yang bernilai *true* sesuai hasil yang diharapkan.

Pada *method* uji ini tetap dilakukan pemanggilan terhadap *method* CountNDependingClassToPkg() dan CountNDependingClassToOthers() untuk memastikan apakah mereka sudah bekerja dengan benar. Hasil yang diperoleh pada masing-masing *method* tersebut sesuai dengan hasil yang diharapkan ketika package A digantungi dirinya sendiri, yaitu 0 karena tidak dianggap. Hasil tersebut dapat dilihat pada gambar 7.14.



Gambar 7.14 Hasil pengujian integrasi kasus uji ke-5 *method* CalculateInRank()

Berdasarkan pengujian integrasi yang telah dilakukan dapat diperoleh tabel 7.4 yang merupakan tabel ringkasan pengujian.

Tabel 7.4 Hasil pengujian integrasi *method* CalculateInRank()

Jalur	Kasus uji	Hasil yang diharapkan	Hasil yang diperoleh
1	*Package* memiliki relasi, *package* ini digantungi, package yang bergantung ditemukan, dan *package* yang bergantung bukan dirinya sendiri. Jumlah relasi *package* B = 1, Tujuan dari relasi= *package* B, *Depending package* = *package* A	Sistem menelusuri seluruh relasi, lalu mencari *depending package* dalam daftar *package*, memanggil method countNDependingClassToPkg dan countNDependingClassToOthers , menghitung nilai inRank, mengembalikan nilai inRank *package* ini dari hasil perhitungan. (1 – 2 – 3 – 4 – 5 – 6 – 7 – 2 – 8)	Sistem menelusuri seluruh relasi, lalu mencari *depending package* dalam daftar *package*, memanggil method countNDependingClassToPkg dan countNDependingClassToOthers, menghitung nilai inRank, mengembalikan hasil perhitungan yang benar dari nilai inRank *package* ini. (1 – 2 – 3 – 4 – 5 – 6 – 7 – 2 – 8)
2	*Package* tidak memiliki relasi. Jumlah relasi *package* B= 0	Sistem tidak menelusuri relasi sehingga method countNDependingClassToPkg dan countNDependingClassToOthers tidak dipanggil, lalu mengembalikan nilai inRank tanpa melakukan perhitungan. (1 – 2 – 8)	Sistem tidak menelusuri relasi, countNDependingClassToPkg dan countNDependingClassToOthers tidak dipanggil, lalu mengembalikan nilai dasar inRank yaitu 0,15 karena tidak melakukan perhitungan. (1 – 2 – 8)
3	*Package* memiliki relasi, tapi dalam relasi ini bukan *package* ini yang digantungi. Jumlah relasi *package* A = 1, Tujuan relasi = *package* B	Sistem menelusuri daftar relasi, karena *package* ini tidak digantungi dalam relasi ini sehingga method countNDependingClassToPkg dan countNDependingClassToOthers tidak dipanggil, perhitungan tidak dilakukan dan penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai inRank dari package ini. (1 – 2 – 3 – 7 – 2 – 8)	Sistem menelusuri seluruh relasi, countNDependingClassToPkg dan countNDependingClassToOthers tidak dipanggil, tidak ada perhitungan nilai inRank karena *package* ini tidak digantungi dalam relasi ini, sehingga sistem melanjutkan penelusuran relasi, sistem mengembalikan nilai inRank dari package ini. (1 – 2 – 3 – 7 – 2 – 8)

Jalur	Kasus uji	Hasil yang diharapkan	Hasil yang diperoleh
4	*Package* memiliki relasi, dalam relasi ini yang digantungi adalah *package* ini, dan *package* yang bergantung tidak ditemukan Jumlah relasi *package* A = 1, Tujuan relasi = *package* A, *Depending package*= new PackageObj (“”,””)	Sistem menelusuri daftar relasi, lalu mencari *depending package*, *depending package* tidak ditemukan sehingga countNDependingClassToPkg dan countNDependingClassToOthers tidak dipanggil, lalu perhitungan tidak dilakukan dan penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai inRank dari *package* ini. (1 – 2 – 3 – 4 – 7 – 2 – 8)	Sistem menelusuri daftar relasi, mencari *depending package*, countNDependingClassToPkg dan countNDependingClassToOthers tidak dipanggil, tidak dilakukan perhitungan inRank karena *depending package* tidak ditemukan, penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai inRank dari *package* ini. (1 – 2 – 3 – 4 – 7 – 2 – 8)
5	*Package* memiliki relasi, dalam relasi ini yang digantungi adalah *package* ini, *package* yang bergantung tidak ditemukan, tapi yang bergantung adalah dirinya sendiri Jumlah relasi *package* A = 1, Tujuan relasi = *package* A, *Depending package*= *package* A	Sistem menelusuri daftar relasi, mencari *depending package*, karena package yang bergantung adalah dirinya sendiri, countNDependingClassToPkg dan countNDependingClassToOthers tidak dipanggil dan tidak dilakukan perhitungan inRank, lalu penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai inRank dari *package* ini. (1 – 2 – 3 – 4 – 5 – 7 – 2 – 8)	Sistem menelusuri daftar relasi, mencari *depending package*, countNDependingClassToPkg dan countNDependingClassToOthers tidak dipanggil karena *depending package* yang ditemukan adalah dirinya sendiri, tidak dilakukan perhitungan inRank, penelusuran relasi dilanjutkan sampai selesai, lalu sistem mengembalikan nilai inRank dari package ini. (1 – 2 – 3 – 4 – 5 – 7 – 2 – 8)

Dari tabel 7.4 dapat disimpulkan bahwa semua jalur *independent* pada *method* CalculateInRank() yang terintegrasi dengan method CountNDependingClassToPkg() dan CountNDependingClassToOthers() bekerja dengan baik karena hasil yang diperoleh sesuai dengan hasil yang diharapkan.

7.3 Pengujian Validasi

Pengujian validasi kakas bantu penilaian kopling *indirect package* dilakukan untuk memastikan bahwa kakas bantu yang dibangun sudah benar tanpa memperhatikan cara implementasinya. Pengujian ini dilakukan berdasarkan daftar kebutuhan-kebutuhan yang sudah didefinisikan pada tahap analisis kebutuhan. Teknik pengujian validasi yang digunakan adalah teknik *black box testing*.

7.3.1 Pengujian Memasukkan Berkas Diagram *Package*

Kebutuhan pertama yang didefinisikan adalah pengguna harus dapat memasukkan berkas diagram *package* dengan ekstensi ".xml". Pengujian ini dimulai dengan pengguna menekan tombol "choose file" dengan ikon berkas. Selanjutnya pengguna memilih berkas XML diagram *package* yang ingin dinilai koplingnya. Jika berkas yang dimasukkan memiliki ekstensi XML, sistem akan menampilkan informasi berkas dan tombol "Parse". Sebaliknya, jika berkas yang dimasukkan bukan berkas XML, sistem menampilkan pesan error.

Fitur ini diuji menggunakan 2 kasus uji untuk dapat menguji semua kemungkinan hasil dari fitur ini. Kasus uji pertama adalah memasukkan berkas diagram *package* dengan ekstensi XML. Hasil yang diperoleh pada pengujian ini sesuai dengan yang diharapkan, yaitu sistem menampilkan informasi berkas dan tombol "Parse". Pengujian tersebut dapat dilihat pada tabel 7.5.

Tabel 7.5 Pengujian memasukkan berkas xml diagram p*ackage*

Kasus uji	Memasukkan berkas diagram *package* dengan ekstensi xml.
Tujuan Pengujian	Memastikan pengguna dapat memasukkan berkas memasukkan berkas *XML package diagram*.
Prosedur Pengujian	1. Pengguna menekan tombol "choose file" dengan ikon bergambar berkas. 2. Pengguna memilih berkas XML diagram *package* yang ingin dinilai koplingnya.
Hasil yang diharapkan	Sistem menampilkan informasi berkas dan tombol "Parse".
Hasil yang diperoleh	Sistem menampilkan informasi berkas dan tombol "Parse".

Kasus uji ke-2 adalah memasukkan berkas diagram *package* dengan ekstensi selain XML. Hasil yang diperoleh pada pengujian ini sesuai dengan yang diharapkan, yaitu sistem menampilkan pesan error mengenai ekstensi berkas. Pengujian tersebut dapat dilihat pada tabel 7.6.

Tabel 7.6 Pengujian memasukkan berkas diagram p*ackage* bukan xml

Kasus uji	Memasukkan berkas diagram *package* dengan ekstensi selain xml.
Tujuan Pengujian	Memastikan pengguna dapat memasukkan berkas memasukkan berkas *XML package diagram*.
Prosedur Pengujian	1. Pengguna menekan tombol "choose file" dengan ikon bergambar berkas. 2. Pengguna memilih berkas XML diagram *package* yang ingin dinilai koplingnya.

Hasil yang diharapkan	Sistem menampilkan pesan error mengenai ekstensi berkas.
Hasil yang diperoleh	Sistem menampilkan pesan error mengenai ekstensi berkas.

7.3.2 Pengujian *Parsing* XML

Kebutuhan ke-2 yang didefinisikan adalah pengguna harus dapat melakukan *parsing* terhadap berkas *package diagram* yang telah dimasukkan. Pengujian ini dimulai dengan pengguna menekan tombol "Parse". Jika data yang dibutuhkan ditemukan saat melakukan *parsing*, sistem akan menampilkan data mengenai *package,* klas, dan relasi yang diperoleh. Sebaliknya jika data yang dibutuhkan tidak ditemukan, sistem akan menampilkan pesan error mengenai data. Jika kondisi berkas bermasalah, sistem akan menampilkan pesan error mengenai kesalahan berkas. Jika struktur XML berkas bermasalah, sistem akan menampilkan pesan error mengenai struktur berkas XML. Oleh karena itu, fitur ini diuji menggunakan 4 kasus uji untuk dapat menguji semua kemungkinan hasil dari fitur ini.

Kasus uji pertama adalah melakukan *parsing* terhadap berkas diagram *package* yang memiliki seluruh data yang dibutuhkan. Hasil yang diperoleh pada pengujian ini sesuai dengan yang diharapkan, yaitu sistem menampilkan data mengenai *package*, klas, dan relasi yang diperoleh dari *parsing*. Data tersebut meliputi id *package*, nama *package*, id klas, nama klas, tipe klas, id relasi, nama relasi, komponen yang bergantung, komponen yang digantungi, dan jenis relasi. Pengujian tersebut dapat dilihat pada tabel 7.7.

Tabel 7.7 Pengujian *parsing* XML ketika berhasil

Kasus uji	*Parsing* terhadap berkas diagram *package* yang memiliki seluruh data yang dibutuhkan, dan tidak ada masalah pada berkas.
Tujuan Pengujian	Memastikan pengguna dapat melakukan *parsing* terhadap berkas diagram *package* yang datanya lengkap dengan menekan tombol "Parse".
Prosedur Pengujian	1. Pengguna menekan tombol "Parse"
Hasil yang diharapkan	Sistem menampilkan data mengenai *package,* klas, dan relasi yang diperoleh dari *parsing*.
Hasil yang diperoleh	Sistem menampilkan data mengenai *package,* klas, dan relasi yang diperoleh dari *parsing*.

Kasus uji ke-2 adalah melakukan *parsing* terhadap berkas diagram *package* yang isi datanya tidak lengkap. Hasil yang diperoleh pada pengujian ini sesuai dengan yang diharapkan, yaitu sistem menampilkan pesan error mengenai data. Pengujian tersebut dapat dilihat pada tabel 7.8.

Tabel 7.8 Pengujian *parsing* XML data tidak ditemukan

Kasus uji	*Parsing* terhadap berkas diagram *package* yang datanya tidak lengkap.
Tujuan Pengujian	Memastikan sistem menampilkan pesan error ketika melakukan *parsing* terhadap berkas diagram *package* yang datanya tidak lengkap setelah menekan tombol "Parse".
Prosedur Pengujian	1. Pengguna menekan tombol "Parse"
Hasil yang diharapkan	Sistem menampilkan pesan error mengenai data.
Hasil yang diperoleh	Sistem menampilkan pesan error mengenai data.

Kasus uji ke-3 adalah melakukan parsing terhadap berkas diagram *package* yang terhadap berkas diagram package yang kondisinya bermasalah. Hasil yang diperoleh pada pengujian ini sesuai dengan yang diharapkan, yaitu sistem menampilkan pesan error kondisi berkas. Pengujian tersebut dapat dilihat pada tabel 7.9.

Tabel 7.9 Pengujian *parsing* XML kondisi berkas diagram bermasalah

Kasus uji	*Parsing* terhadap berkas diagram *package* yang kondisinya bermasalah.
Tujuan Pengujian	Memastikan sistem menampilkan pesan error ketika melakukan *parsing* setelah menekan tombol "Parse" terhadap berkas diagram *package* yang adbermasalah kondisinya, seperti *corrupt* atau berpindah lokasi.
Prosedur Pengujian	1. Pengguna menekan tombol "Parse"
Hasil yang diharapkan	Sistem menampilkan pesan error mengenai kondisi berkas.
Hasil yang diperoleh	Sistem menampilkan pesan error mengenai kondisi berkas.

Kasus uji ke-4 adalah melakukan parsing terhadap berkas diagram *package* yang terhadap berkas diagram package yang struktur XMLnya bermasalah. Pada pengujian ini, berkas XML yang dimasukkan kekurangan satu *tag* untuk menutup elemen. Hasil yang diperoleh pada pengujian ini sesuai dengan yang diharapkan, yaitu sistem menampilkan pesan error mengenai struktur berkas. Pengujian tersebut dapat dilihat pada tabel 7.10.

Tabel 7.10 Pengujian *parsing* XML struktur XML berkas diagram bermasalah

Kasus uji	*Parsing* terhadap berkas XML diagram *package* yang struktur XMLnya bermasalah, yaitu kekurangan satu tag untuk menutup elemen.

Tujuan Pengujian	Memastikan sistem menampilkan pesan error ketika melakukan *parsing* setelah menekan tombol "Parse" terhadap berkas diagram *package* yang struktur XMLnya bermasalah., seperti kekurangan satu *tag* penutup elemen.
Prosedur Pengujian	1. Pengguna menekan tombol "Parse"
Hasil yang diharapkan	Sistem menampilkan pesan error mengenai struktur berkas.
Hasil yang diperoleh	Sistem menampilkan pesan error mengenai struktur berkas.

7.3.3 Pengujian Menilai Kopling

Kebutuhan terakhir yang didefinisikan adalah pengguna harus dapat menilai kopling terhadap data hasil *parsing* dengan menekan tombol "Calculate". Pengujian ini dimulai dengan pengguna menekan tombol "Calculate". Lalu sistem menampilkan hasil perhitungan. Jika ditemukan relasi sirkular, sistem akan menampilkan pesan error mengenai relasi sirkular dan perhitungan dihentikan. Oleh karena itu, ada 2 kasus uji yang dibutuhkan untuk menguji fitur ini.

Hasil yang diperoleh pada pengujian kasus uji pertama sesuai dengan yang diharapkan, yaitu sistem menampilkan hasil perhitungn masing-masing *package*. Hasil perhitungan yang ditampilkan, antara lain nilai inRank, outRank, *dual ranking instability, package abstraction,* total inRank klas abstrak dalam *package*, total inRank seluruh klas dalam *package,* dan *package's normalized distance* yang menunjukkan jarak nilai kopling dari titik ideal. Pengujian ini dapat dilihat pada tabel 7.11.

Tabel 7.11 Pengujian menilai kopling

Kasus Uji	Menilai kopling berdasarkan hasil perhitungan terhadap data yang tidak memiliki relasi sirkular.
Tujuan Pengujian	Memastikan pengguna dapat melakukan perhitungan kopling terhadap data hasil *parsing* dengan menekan tombol "Calculate".
Prosedur Pengujian	1. Pengguna menekan tombol "Calculate"
Hasil yang diharapkan	Sistem menampilkan hasil perhitungan kopling, yaitu nilai *dual ranking instability, package abstraction,* dan *package's normalized distance* yang menunjukkan jarak nilai kopling dari titik ideal.
Hasil yang diperoleh	Sistem menampilkan hasil perhitungan kopling, yaitu nilai *dual ranking instability, package abstraction,* dan *package's normalized distance* yang menunjukkan jarak nilai kopling dari titik ideal.

Hasil yang diperoleh pada pengujian kasus uji kedua sesuai dengan yang diharapkan, yaitu sistem menampilkan pesan error mengenai relasi sirkular. Pengujian ini dapat dilihat pada tabel 7.12.

Tabel 7.12 Pengujian menilai kopling terdapat relasi sirkular

Kasus Uji	Menilai kopling berdasarkan perhitungan terhadap data yang memiliki relasi sirkular.
Tujuan Pengujian	Memastikan sistem menampilkan pesan error setelah pengguna menekan tombol "Calculate" pada data yang memiliki relasi sirkular.
Prosedur Pengujian	2. Pengguna menekan tombol "Calculate"
Hasil yang diharapkan	Sistem menampilkan pesan error mengenai relasi sirkular dan menghentikan perhitungan.
Hasil yang diperoleh	Sistem menampilkan pesan error mengenai relasi sirkular dan menghentikan perhitungan.

Berdasarkan beberapa pengujian validasi yang telah dilakukan, dapat disimpulkan bahwa semua kebutuhan yang didefiniskan sudah tepenuhi. Hal ini ditunjukkan ketika hasil yang diperoleh dari setiap kebutuhan sesuai dengan hasil yang diharapkan. Hasil pengujian validasi dapat dilihat pada tabel 7.13.

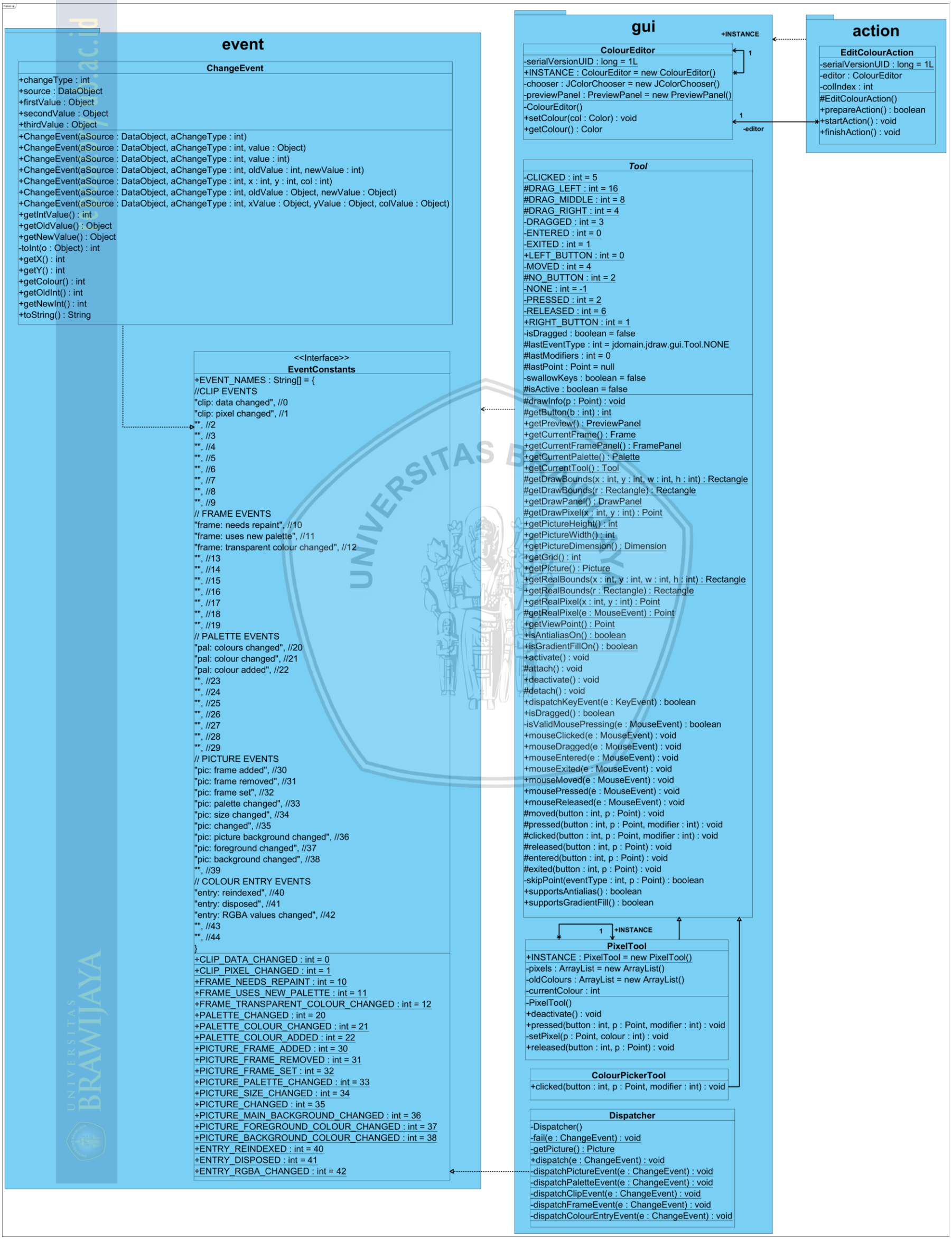
Tabel 7.13 Hasil pengujian validasi kakas bantu

Nomor Fungsi	Nama Kebutuhan	Hasil
SRS_KBPKIP_100	Memasukkan berkas diagram *package*	Valid
SRS_KBPKIP_110	*Parsing* XML	Valid
SRS_KBPKIP_120	Menilai kopling	Valid

7.4 Pengujian Performa

Pengujian performa dilakukan untuk mengukur waktu yang dibutuhkan kakas bantu dalam menyelesaikan komputasinya terhadap suatu sampel diagram *package*. Sampel yang digunakan adalah diagram *package* dari suatu perangkat lunak *open source*, yaitu JDraw. Diagram package tersebut diperoleh dengan menggunakan fitur *instant reverse* dari Visual Paradigm terhadap kode sumber JDraw. Diagram tersebut selanjutnya disimpan dalam format XML melalui fitur *export*. Selain itu, pengujian ini bertujuan membuktikan bahwa kakas bantu penilaian kopling *indirect package* dapat menghemat waktu yang dibutuhkan dalam menilai kopling. Hal ini dilakukan dengan membandingkan kecepatan komputasinya dengan manusia secara manual terhadap suatu diagram *package*. Oleh karena itu, pengujian performa kakas bantu ini dilakukan sebanyak dua kali.

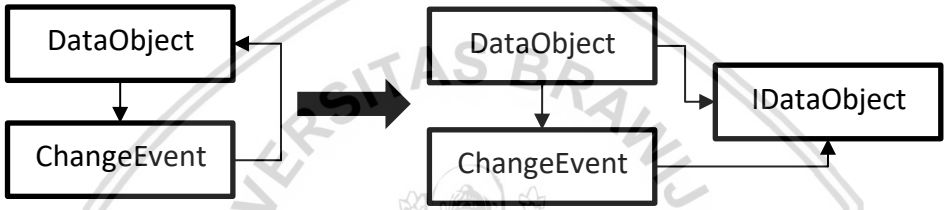
Pengujian pertama dilakukan terhadap diagram *package* JDraw yang disederhanakan untuk dibandingkan dengan kecepatan komputasi manual manusia. *Package* yang terdapat dalam diagram meliputi event, gui, dan action. *Package* event memiliki 2 klas, yaitu klas *interface* EventConstants dan klas ChangeEvent. *Package* gui memiliki 5 klas, yaitu klas abstrak Tool, klas konkret ColourPickerTool, Dispatcher, PixelTool, dan ColourEditor. *Package* action hanya memiliki 1 klas, yaitu EditColourAction. Gambaran lebih jelasnya dapat dilihat pada Gambar 7.17. Pengujian ini dilakukan satu kali secara manual dan satu kali menggunakan kakas bantu. Setelah itu waktu pemrosesan menggunakan kakas bantu dan perhitungan secara manual dibandingkan.



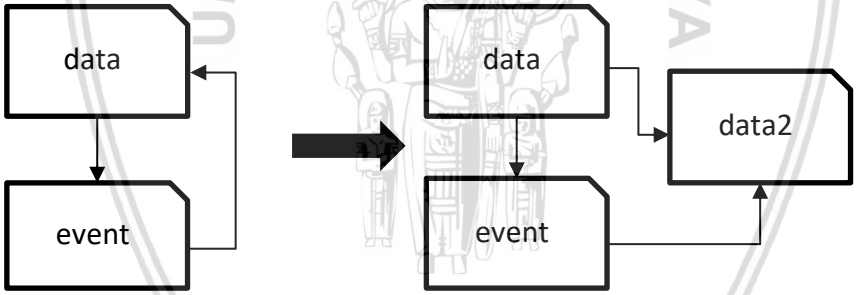
Gambar 7.15 Diagram *package* studi kasus pengujian pertama

Pengujian ke-2 dilakukan terhadap diagram *package* JDraw menyeluruh. Pada diagram *package* JDraw, ditemukan relasi sirkular yang melanggar *Acyclic Dependencies Principle* (ADP) dan dapat menimbulkan permasalahan perhitungan pada kakas bantu. Almugrin (2015b) menyampaikan bahwa relasi sirkular dapat menimbulkan perulangan perhitungan tidak berbatas pada metrik yang digunakan oleh kakas bantu ini.

Relasi sirkular dapat diperbaiki dengan melakukan perubahan terhadap *package* dan klas penyebabnya (Almugrin, 2015b). Langkah pertama yang dilakukan adalah membuat *package* baru. Klas yang digantungi pada titik sirkular dari *package* akan dibuat klas *interface*-nya sehingga muncul relasi ketergantungan terhadap klas *interface* tersebut. Klas yang bergantung menjadi bergantung terhadap klas *interface*-nya, bukan klas konkretnya. Hal tersebut menghilangkan relasi sirkular pada tingkat klas. Selanjutnya klas *interface* tadi dimasukkan ke dalam *package* baru sehingga relasi sirkular pada tingkat *package* juga terputus. Ilustrasinya kurang lebih seperti yang ditunjukkan pada gambar 7.15 dan 7.16.



Gambar 7.16 Ilustrasi pemutusan relasi sirkular tingkat klas



Gambar 7.17 Ilustrasi pemutusan relasi sirkular tingkat *package*

7.4.1 Analisis Hasil Pengujian Performa

Perhitungan manual untuk menilai kopling dari *package* "event" menggunakan metrik *indirect package coupling based on responsibility* dilakukan melalui beberapa tahapan. Tahapan utamanya terbagi menjadi 3, yaitu perhitungan *instability, abstraction,* dan *normalized distance*. Variabel *d* yang digunakan dalam perhitungan bernilai 0,85 dan berperan sebagai *damping factor.*

Tahap pertama yang dilakukan untuk memperoleh nilai *instability* adalah perhitungan inRank dan outRank. *Package* event digantungi oleh gui, sedangkan gui digantungi oleh action, sehingga dilakukan perhitungan inRank dari action terlebih dahulu, dilanjutkan menghitung inRank dari gui, dan terakhir menghitung inRank event.

Package action tidak digantungi, sehingga *d* dikali dengan nilai 0 seperti pada persamaan 7.1.

$$InRank(action) = (1 - d) + d(0) \quad (7.1)$$
$$= (1 - 0{,}85) + 0{,}85(0) = 0{,}15$$

Package gui hanya digantungi oleh action, sehingga d dikali dengan inRank dari action saja, seperti pada persamaan 7.2.

$$InRank(gui) = (1 - d) + d\left(\frac{InRank(action) \times C(action)}{D(action)} + 0\right) \quad (7.2)$$
$$= (1 - 0{,}85) + 0{,}85\left(\frac{0{,}15 \times 1}{1}\right)$$
$$= 0{,}15 + 0{,}1275 = 0{,}2775$$

Package event hanya digantungi oleh gui, sehingga d dikali dengan inRank dari gui saja seperti perhitungan sebelumnya. Nilai inRank dari event yang diperoleh adalah 0,3859. Perhitungan outRank dimulai dari event terlebih dahulu karena action bergantung kepada gui dan gui bergantung kepada event.

Package event tidak bergantung, sehingga *d* dikali dengan nilai 0 seperti pada persamaan 7.3.

$$OutRank(event) = (1 - d) + d(0) \quad (7.3)$$
$$= (1 - 0{,}85) + 0{,}85(0) = 0{,}15$$

Package gui hanya bergantung kepada event, sehingga d dikali dengan outRank dari event saja, seperti pada persamaan 7.4.

$$OutRank(gui) = (1 - d) + d(OutRank(event) + 0) \quad (7.4)$$
$$= (1 - 0{,}85) + 0{,}85(0{,}15) = 0{,}2775$$

Package action hanya bergantung kepada gui, sehingga d dikali dengan inRank dari gui saja seperti perhitungan sebelumnya. Nilai outRank dari action yang diperoleh adalah 0,3859.

Tahap kedua yang dilakukan untuk memperoleh nilai *instability* adalah normalisasi inRank dan outRank dari *package* event. Pertama dilakukan pencarian terhadap nilai terbesar dan terkecil dari inRank maupun outRank. Nilai inRank terbesar adalah 0, 3859 dan terkecil adalah 0,15. Nilai outRank terbesar adalah 0, 3859 dan yang terkecil adalah 0,15. Normalisasi nilai inRank dan outRank dari event masing-masing ditunjukkan oleh persamaan 7.5 dan 7.6.

$$InRank(event) = \frac{InRank_i(event) - Min}{(Max - Min)} \quad (7.5)$$

$$= \frac{0,3859 - 0,15}{(0,3859 - 0,15)} = 1$$

$$OutRank(event) = \frac{OutRanki(event) - Min}{(Max - Min)} \quad (7.6)$$

$$= \frac{0,15 - 0,15}{(0,3859 - 0,15)} = 0$$

Nilai ternormalisasi tersebut selanjutnya digunakan untuk menghitung *instability* seperti pada persamaan 7.7. Nilai *instability* dari *package* event adalah 0 yang berarti stabil.

$$I(event) = \frac{OutRank(event)}{(OutRank(event) + InRank(event))} \quad (7.7)$$

$$= \frac{0}{(0-1)} = 0$$

Nilai *abstraction* diperoleh dengan melakukan perhitungan inRank seluruh klas yang ada dalam *package* event terlebih dahulu. Nilai *d* yang digunakan masih sama, yaitu 0,85. *Package* event memiliki 2 klas, yaitu klas konkret ChangeEvent dan klas *interface* EventConstants yang disingkat menjadi EC. Klas EventConstants digantungi oleh ChangeEvent, sehingga perhitungan inRank dimulai dari ChangeEvent terlebih dahulu. ChangeEvent tidak digantungi sehingga nilai *d* dikali dengan 0 seperti pada persamaan 7.8.

$$InRank\ (ChangeEvent) = (1-d) + d(0) \quad (7.8)$$

$$= (1 - 0,85) + 0,85(0) = 0,15$$

EventConstants digantungi oleh ChangeEvent sehingga nilai *d* dikali dengan nilai inRank ChangeEvent seperti pada persamaan 7.9.

$$InRank(EC) = (1-d) + d\left(\frac{InRank(ChangeEvent)}{S(ChangeEvent)}\right) \quad (7.9)$$

$$= (1 - 0,85) + 0,85\left(\frac{0,15}{1}\right) = 0,2275$$

Selanjutnya dilakukan perhitungan nilai *abstraction* dengan membagi nilai total inRank klas abstrak maupun *interface* dengan total inRank seluruh klas. Perhitungan tersebut ditunjukkan oleh persamaan 7.10. Nilai *abstraction* yang diperoleh adalah 0,6027 yang artinya cukup abstrak sehingga tidak mudah berubah.

$$Abs(event) = \frac{\sum_{A_j \in A}(InRank(A_i))}{\sum_{C_j \in C}(InRank(C_j))} \quad (7.10)$$

$$= \frac{(InRank(EC))}{(InRank(ChangeEvent) + InRank(EC))}$$

$$= \frac{(0{,}2275))}{(0{,}15 + 0{,}2275)} = 0{,}6027$$

Nilai *instability* dan *abstraction* yang sudah diperoleh digunakan untuk menentukan nilai *normalized distance* atau jarak nilai kopling dari titik ideal. Perhitungan tersebut ditunjukkan pada persamaan 7.11.

$$Dn = |(A + I - 1)| \quad (7.11)$$

$$= |(0{,}6027 + 0 - 1)| = 0{,}3973$$

Dari nilai *normalized distance* dapat diketahui bahwa package event memiliki kopling yang baik karena dekat dengan titik idealnya, sehingga tidak menimbulkan efek domino yang besar ketika ia dimodifikasi.

Perhitungan manual untuk menilai kopling *package* "event" seperti yang telah dilakukan membutuhkan waktu sekitar 10 menit 28 detik. Perhitungan tersebut hanya memakan waktu 0,8812 detik jika dilakukan menggunakan kakas bantu. Hal ini menunjukkan bahwa kakas bantu berhasil meningkatkan efisiensi waktu yang dibutuhkan dalam melakukan penilaian terhadap kopling *package*. Oleh karena itu, dapat dilihat pada tabel 7.14 bahwa kakas bantu teruji memenuhi kebutuhan non-fungsional yang telah didefinisikan.

Tabel 7.14 Hasil pengujian kebutuhan non-fungsional sistem

Nomor Fungsi	Nama Kebutuhan	Hasil
SRS_KBPKIP_200	Sistem harus dapat menilai kopling lebih cepat dari manusia. (*Performance*)	Valid

Hasil pengujian ke-2 yang diperoleh cukup baik. Penilaian kopling terhadap diagram *package* perangkat lunak JDraw rata-rata memakan waktu 1,8252 detik ketika berkas sudah ditetapkan pada sistem. Berkas ditetapkan terlebih dahulu pada sistem karena ada tambahan waktu jika pengguna harus memilih berkas, menekan tombol, dan *loading* komponen tampilan. Dalam diagram tersebut ditemukan 12 *package,* 209 klas, dan 237 relasi ketergantungan. Hasil penilaian kopling yang diperoleh dapat dilihat pada tabel 7.15.

Tabel 7.15 Hasil penilaian kopling JDraw menggunakan kakas bantu

Package	Instability	Abstractness	Normalized Distance	Hasil
jdomain	-	-	-	-
jdraw	1	0.10553	0.10553	Baik
action	1	0.11376	0.11376	Baik
data	0.37509	0.11565	0.50927	Buruk
event	0.17083	0	0.82917	Buruk
gio	1	0	0	Baik
gui (jdraw)	0.90316	0.11109	0.01425	Baik
dnd	-	-	-	-
undo	1	0.21679	0.21679	Baik
data2	0	0	1	Buruk
util	0	0	1	Buruk
gui (util)	0.40760	0	0.59240	Buruk

Instability adalah nilai ketidakstabilan *package* akibat adanya ketergantungan keluar maupun dari luar ke dalam. Semakin kecil nilai *instability* semakin baik kualitas suatu *package*. *Abstractness* merupakan nilai keabstrakan suatu klas, semakin besar nilainya semakin baik kualitas *package* tersebut. Jarak nilai kopling dari titik ideal ditunjukkan oleh *normalized distance*. Jika nilai tersebut melebihi 0,5 berarti *package* berada jauh dari titik idealnya sehingga dinilai buruk dan berlaku sebaliknya. Pada diagram, ada 2 *package* bernama gui pada JDraw, yang pertama berada dalam *package* jdraw, yang kedua dalam *package* util. *Package* jdomain dan dnd tidak memiliki relasi ketergantungan dengan *package* lainnya sehingga tidak memiliki kopling.

BAB 8 PENUTUP

Bab penutup dari penelitian yang berjudul "Kakas bantu penilaian kopling dengan menggunakan metrik *indirect package coupling based on responsibility*" ini membahas tentang hasil yang diperoleh dari penelitian. Hasil yang diperoleh dibahas dalam dua bagian, yaitu kesimpulan dan saran.

8.1 Kesimpulan

Kesimpulan dari penelitian mengenai kakas bantu perhitungan nilai kopling menggunakan metrik *indirect package coupling based on responsibility*, antara lain:

1. Implementasi metrik *indirect package coupling based on responsibility* pada kakas bantu untuk menilai kopling berhasil dilakukan. Perhitungan yang dilakukan kakas bantu untuk menilai kopling dari *package* menggunakan metrik tersebut bekerja sesuai dengan kebutuhan yang telah didefiniskan. Akan tetapi, metrik tidak dapat digunakan pada *package* yang memiliki relasi sirkular karena dapat menimbulkan perulangan tidak berujung. Relasi tersebut dapat diputus secara manual dengan menambahkan *package* dan klas baru sebagai penengahya. Implementasi tersebut diuji dengan melakukan penilaian kopling terhadap perangkat lunak Jdraw. Pertama dilakukan perhitungan nilai inRank (*responsibility*) dan outRank (*dependency*) dari *package* untuk menentukan nilai *instability*. Selanjutnya dilakukan perhitungan nilai *abstraction* dengan menghitung nilai total inRank dari seluruh klas dan total inRank khusus klas abstrak. Dari nilai *instability* dan *abstraction* tersebut dapat diperoleh nilai *normalized distance* yang digunakan untuk menilai kualitas kopling *package*.
2. Kakas bantu mengidentifikasi data yang diperlukan dari berkas XML diagram *package* dengan melakukan *parsing* menggunakan library DOM. Elemen utama yang dicari di dalam berkas XML, yaitu <Package>, <Class> yang merupakan child dari <Package>, <Dependency>. <Generalization>, <Realization>, dan <Association>. Penelusuran terhadap elemen <Models> dilakukan terlebih dahulu karena merupakan elemen *parent* dari <Package> dan <Class>. Penelusuran elemen-elemen relasi dilakukan dengan mencari elemen <ModelRelationshipContainer> yang merupakan *parent*-nya.
3. Kakas bantu diuji melalui 4 jenis pengujian, yaitu pengujian unit, pengujian integrasi, pengujian validasi, dan pengujian performa. Pengujian unit dan integrasi dilakukan menggunakan teknik pengujian *white box* untuk memastikan semua jalur *independent* berhasil dilalui dan hasil yang diperoleh sesuai dengan hasil yang diharapkan. Pengujian validasi dilakukan menggunakan teknik pengujian *black box* untuk memastikan kakas bantu yang dibangun sudah sesuai dengan kebutuhan. Pengujian performa bertujuan untuk membuktikan bahwa kakas bantu dapat mengurangi kebutuhan waktu yang diperlukan dalam menilai kopling *package*. Pengujian tersebut dilakukan dengan membandingkan waktu perhitungan yang dibutuhkan oleh manusia dan kakas bantu. Penilaian kopling *package* event dari diagram *package* JDraw yang disederhanakan menjadi 3 *package,* yaitu event, gui,

dan action secara manual membutuhkan 10 menit 28 detik, sedangkan kakas bantu hanya membutuhkan 0,8812 detik. Penilaian kopling *package* pada JDraw yang berisi 12 *package*, 209 klas, dan 237 relasi ketergantungan membutuhkan waktu 1,8252 detik. Hasil pengujian-pengujian tersebut menunjukkan bahwa kakas bantu bekerja sesuai kebutuhan yang didefinisikan dan dapat menghemat waktu yang dibutuhkan dalam menilai kopling sekitar 10 menit 27,1188 detik.

8.2 Saran

Saran untuk penelitian yang berjudul "Kakas Bantu Penilaian Kopling Menggunakan Metrik *Indirect Package Coupling Based on Responsibility*", yaitu:

1. Penambahan fitur untuk mengatasi relasi sirkular secara otomatis dengan menambahkan klas dan *package*.
2. Penambahan fitur untuk menyimpan hasil yang diperoleh ke dalam sebuah log untuk menunjukkan riwayat sehingga tidak perlu melakukan penilaian lagi jika sudah pernah.
3. Mencoba menggunakan *library* untuk *parsing* lainnya untuk membandingkan performa. Berdasarkan percobaan yang dilakukan penulis, DOM memiliki penggunaan memori yang lebih kecil dibandingkan JDOM dan SAX. Akan tetapi, beberapa artikel di internet menunjukkan bahwa SAX lebih ringan dibandingkan DOM.

DAFTAR PUSTAKA

Almugrin, S., Albattah, W., Alaql O., Alzahrani, M., & Melton, A., 2014. Instability and Abstractness Metrics Based on Responsibility. Dalam: IEEE (Institute of Electrical and Electronics Engineers), 2014. *38th Computer Software and Applications Conference* (COMPSAC), pp.364-373. Vasteras, Sweden, 21-25 Juli 2014.

Almugrin, S. dan Melton, A., 2015a. Indirect Package Coupling Based on Responsibility in an Agile, Object-Oriented Environment. Dalam: IEEE (Institute of Electrical and Electronics Engineers), 2015. *2nd International Conference on Trustworthy Systems and Their Applications*, pp.110-119. Hualien, Taiwan, 8-9 Juli 2015.

Almugrin, S., 2015b. *Definitions and Validations of Metrics of Indirect Package Coupling in an Agile, Object-Oriented Environment.* S3. Kent State University. Tersedia di <https://etd.ohiolink.edu/!etd.send_file?accession=kent1436828087&disposition=inline> [Diakses 12 Februari 2018]

Almugrin, S., Albattah, W., dan Melton, A., 2016. Using Indirect Coupling Metrics to Predict Package Maintainability and Testability. *Journal of Systems and Software*, 121, pp298-310.

Aulia, L. H., 2016. Kakas Bantu Perhitungan Nilai Kopling Menggunakan *Conceptual Coupling Metrics.* S1. Universitas Brawijaya.

Bocco, M.G., Piattini, M., dan Calero, C., 2005. A Survey of Metrics for UML Class Diagrams. *Journal of Object Technology*, vol. 4, 2005, pp. 59- 92.

Briand, L.C., Daly, J.W., dan Wüst, J., 1999. A Unified Framework for Coupling Measurement in Object-Oriented Systems. *Transactions on Software Engineering,* vol.25, pp.91-121.

Chowdhury, I. dan Zulkernine, M., 2011. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *Journal of Systems Architecture*, Vol. 57, pp. 294-313.

Ducasse, S., Lanza, M., dan Ponisio, L., 2005. Butterflies: A visual approach to characterize packages. Dalam : IEEE Computer Society, *11th IEEE International Software Metrics Symposium*. Como, Italy, 19-22 September 2005.

English, M., Cahill, T., dan Buckley, J., 2012. *Construct Specific Coupling Measurement for C++ Software*. University of Limerick, Ireland.

Fennell, P., 2013. Extremes of XML. Dalam: University College London, *XML London 2013 Conference*, pp. 80–86. London, United Kingdom, 15-16 Juni 2013.

Izurieta, C. dan Bieman, J.M., 2008. Testing Consequences of Grime Buildup in Object Oriented Design Patterns. Dalam: *1st International Conference on Software Testing, Verification, and Validation*, pp. 171-179. Lillehammer, Norway, 9-11 April 2008.

Kroenke, D.M. dan D.J. Auer, 2012. *Database Processing: Fundamentals, Design, Implementation*. 12th edition. Upper Saddle River : Pearson.

Mafazi, M. F., 2011. *Transformasi Dokumen XML Menjadi Model Basis Data Relasional dengan Menggunakan Metode Parsing SAX.* S1. Universitas Diponegoro.

Martin, R. C., 2003. *Agile Software Development Principles, Patterns & Practice*. Upper Saddle River: Prentice Hall.

Mishra, A., Dubey, D., 2013. A Comparative Study of Different Software Development Life Cycle Models in Different Scenarios. *International Journal of Advance Research in Computer Science and Management Studies*, vol.1, pp.64-69.

Patidar, K., 2013. Coupling and Cohesion Measures in Object Oriented Programming. *International Journal of Advanced Research in Computer Science and Software Engineering,* vol.3, pp.517-521.

Ponisio, M.L., 2006. *Exploiting Client Usage to Manage Program Modularity*. S3. University of Bern.

Pressman, R. S., 2010. *Software Engineering: Practitioner's approach*. 7th edition. New York City: McGraw-Hill.

Rouf, A., 2012. Pengujian Perangkat Lunak dengan Menggunakan Metode *White Box dan Black Box*. *HIMSYATECH-Jurnal Teknologi Informasi*, vol. 8(1).

Sommerville, I., 2011. *Software Engineering*, 9th edition. Boston : Addison-Wesley Publishing Company.

Wood, L., 2000. *Document Object Model* (DOM) *Level 1 Specification,* 2nd edition. Tersedia di: < https://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/> [Diakses 8 Maret 2018]