repository.ub.ac.id

KAKAS BANTU PERHITUNGAN KOMPLEKSITAS PERANGKAT LUNAK MENGGUNAKAN METRIK *COGNITIVE WEIGHTED INHERITED CLASS COMPLEXITY*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
WIDYAWATI
NIM: 155150201111215

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2019

UNIVERSITAS BRAWIJAYA

PENGESAHAN

KAKAS BANTU PERHITUNGAN KOMPLEKSITAS PERANGKAT LUNAK MENGGUNAKAN METRIK *COGNITIVE WEIGHTED INHERITED CLASS COMPLEXITY*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Widyawati
NIM: 155150201111215

Skripsi ini telah diuji dan dinyatakan lulus pada
14 Mei 2019
Telah diperiksa dan disetujui oleh:

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

Mengetahui
Ketua Jurusan Teknik Informatika

Ir. 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, 23 Mei 2019

Widyawati

NIM: 155150201111215

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

PRAKATA

Puji dan syukur kehadirat Allah SWT atas segala limpahan Rahmat, Taufik dan Hidayah-Nya sehingga penulis dapat menyelesaikan skripsi yang berjudul "Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan Metrik *Cognitive Weighted Inherited Class Complexity".* Skripsi ini diajukan sebagai syarat untuk mendapatkan gelar Sarjana Komputer di Fakultas Ilmu Komputer Universitas Brawijaya. Selama proses penyusunan skripsi, penulis banyak belajar dan mengaplikasikan ilmu yang telah diperoleh dari perkuliahan. Semoga penulis dapat terus mengembangkan ilmu yang diperoleh selama perkuliaahan dengan baik. Selain itu penulis juga mendapat banyak dukungan dan bantuan dari berbagai pihak. Oleh sebab itu, penulis ingin mengucapkan terimakasih kepada:

1. Bapak Bayu Priyambadha, S. Kom, M. Kom. selaku Pembimbing I, dan Bapak Fajar Pradana, S. ST., M. Eng. selaku Pembimbing II yang telah membimbing dan mengarahkan penulis dalam proses penyelesaian skripsi.
2. Bapak Agus Wahyu Widodo, S.T, M.Cs. selaku Ketua Program Studi Teknik Informatika.
3. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D. selaku Ketua Jurusan Teknik Informatika,
4. Kedua orang tua dan kakak saya yang selalu memberikan motivasi dan doa yang tiada hentinya kepada penulis demi terselesaikannya skripsi ini.
5. Bapak dan Ibu dosen serta staff Fakultas Ilmu Komputer Universitas Brawijaya yang telah bersedia membagi ilmu dan arahan kepada penulis.
6. Dzulhaj Maulana yang telah memberikan doa, motivasi dan semangat kepada penulis.
7. Yolanda Devialita dan Risqi Auliatin Nisyah yang selalu memberikan motivasi dan arahan kepada penulis.
8. Ifa, Rohma, Octavia, Min, Yoga, Bagas yang selalu memberikan doa dan dukungan kepada penulis.
9. Novita, Amira, Nadia, Icha, Rani, Aprida, Odhia, Yosie, Marisa, Rasio, Ridwan, Loudy, Fatim, Aang, Siska, dan teman-teman Teknik Informatika angkatan 2015 yang telah membantu selama masa studi penulis dan memberikan motivasi, arahan, dan saran dalam pengerjaan skripsi.
10. Semua pihak yang tidak dapat disebutkan satu - persatu.

Penulis menyadari dalam proses penyusunan skripsi masih banyak terdapat kekurangan, oleh sebab itu kritik dan saran yang membangun sangat diperlukan dalam penulisan selanjutnya. Penulis berharap semoga skripsi ini memberikan manfaat kepada pihak yang membutuhkan.

Malang, 23 Mei 2019

Penulis
widyawati.project@gmail.com

ABSTRAK

Widyawati, Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan Metrik *Cognitive Weighted Inherited Class Complexity*

Pembimbing: Bayu Priyambadha, S. Kom, M. Kom. dan Fajar Pradana, S. ST., M. Eng.

Software maintenance merupakan tahap pengembangan perangkat lunak yang membutuhkan usaha, waktu, dan biaya yang besar. Salah satu penyebab permasalahan pada tahap *maintenance* adalah kualitas perangkat lunak yang buruk. Salah satu faktor yang menentukan kualitas perangkat lunak tersebut adalah tingginya nilai kompleksitas perangkat lunak. *Developer* akan kesulitan dalam memahami perangkat lunak saat melakukan pemeliharaan maupun jika ada perubahan ketika kompleksitasnya tinggi. Salah satu upaya untuk mengatasi permasalahan tersebut adalah menentukan konsep pengembangan perangkat lunak yang sesuai yaitu menggunakan pendekatan berorientasi objek. Pada pemrograman berorientasi objek (OO) penggunaan *inheritance, encapsulation, coupling, polymorphism,* dan lain-lain dapat memudahkan *developer* dalam membangun perangkat lunak serta dapat meningkatkan kualitas dan keandalan perangkat lunak. Namun, penggunaan *inheritance* yang tinggi juga dapat mengakibatkan tingginya nilai kompleksitas. Oleh karena itu kompleksitas perangkat lunak harus dipantau selama tahap implementasi, agar tahap maintenance dapat dilakukan dengan lebih optimal, salah satunya yaitu menggunakan *software metric* untuk menghitung kompleksitas. Perhitungan nilai kompleksitas secara manual pada suatu perangkat lunak akan membutuhkan waktu dan upaya yang besar serta sumber daya manusia yang benar-benar paham pada model perhitungannya. Oleh karena itu dibuatlah kakas bantu perhitungan kompleksitas perangkat lunak menggunakan metrik *Cognitive Weighted Inherited Class Complexity* (CWICC). Metrik CWICC merupakan metrik kompleksitas perangkat lunak yang berfungsi untuk menghitung kompleksitas berdasarkan bobot pemahaman pada struktur internal kelas yaitu *attribute, method,* dan *inheritance* dari suatu kelas. Sistem ini dibangun menggunakan bahasa pemrograman Java dan juga telah diuji menggunakan metode *white box testing* dengan teknik *basis path testing* pada pengujian unit dan pengujian integrasi serta pengujian validasi yang menggunakan metode *blackbox testing*. Sistem ini memiliki tingkat nilai akurasi sebesar 100% dan waktu untuk melakukan perhitungan kompleksitas menggunakan sistem rata-rata 356 kali lebih cepat dibanding perhitungan manual.

Kata kunci: Kompleksitas perangkat lunak, *Software maintenace*, Metrik *Cognitive Weighted Inherited Class Complexity,* kode sumber, *inheritance.*

repository.ub.ac.id

ABSTRACT

Widyawati, *Tools that Calculation of the Software Complexity using Cognitive Weighted Inherited Class Complexity Metric*

***Adviser*: Bayu Priyambadha, S. Kom, M. Kom. dan Fajar Pradana, S. ST., M. Eng.**

Software maintenance is a software development phase that requires a lot of effort, time, and cost. One of the causes of problems at the maintenance phase is poor software quality. One of the factors that determine the quality of the software is the high value of software complexity. Developers will find difficulty to understand software when performing maintenance phase or if there are any changes when the complexity is high. One effort to overcome this problem is to determine the concept of appropriate software development, namely using an object-oriented approach. In object-oriented programming (OO) the use of inheritance, encapsulation, coupling, polymorphism, etc. can facilitate developers to build software and can improve the quality and reliability of software. However, the use of high inheritance can also result in high values of complexity. Therefore the complexity of software must be monitored during the implementation phase, so that the maintenance stage can be carried out more optimally, one of which is to use software metrics to calculate complexity. Manually calculating the complexity of a software will require considerable time and effort and human resources that really understand model of the calculation. Therefore, this research will develop Tool that can calculate software complexity using the Cognitive Weighted Inherited Class Complexity (CWICC) metric. CWICC metrics are software complexity metrics that function to calculate complexity based on the weight of understanding in the internal class structure, namely attributes, methods, and inheritance of a class. This system is built using the Java programming language and has also been tested using the white box testing method in unit testing and integration testing and then validation testing using the blackbox testing method. This system has an accuracy rate of 100% and the time to calculate complexity uses an average system 356 times faster than manual calculations.

Keyword: *Software Complexity, Software maintenance, Cognitive Weighted Inherited Class Complexity Metric , Source Code, Inheritance.*

UNIVERSITAS BRAWIJAYA

DAFTAR ISI

PENGESAHAN ii
PERNYATAAN ORISINALITAS iii
PRAKATA iv
ABSTRAK v
ABSTRACT vi
DAFTAR ISI vii
DAFTAR TABEL x
DAFTAR GAMBAR xiii
BAB 1 PENDAHULUAN 1
1.1 Latar belakang 1
1.2 Rumusan masalah 3
1.3 Tujuan 3
1.4 Manfaat 4
1.5 Batasan masalah 4
1.6 Sistematika pembahasan 4
BAB 2 LANDASAN KEPUSTAKAAN 6
2.1 Kajian Pustaka 6
2.2 Rekayasa Perangkat Lunak 7
2.2.1 *Software Development Life Cycle* 7
2.2.2 Model Waterfall 8
2.2.3 Pendekatan Berorientasi Objek 10
2.2.4 Pemodelan Berorientasi Objek 10
2.2.5 Pemrograman Berorientasi Objek 13
2.2.6 Hubungan Pendekatan Berorientasi Objek dan Kompleksitas 14
2.2.7 Pengujian Perangkat Lunak 14
2.3 Cognitive Weighted Inherited Class Complexity Metric 18
2.4 Teknologi Pengembangan Sistem 20

2.4.1 Source Code 20
2.4.2 Bahasa Pemrograman Java 20
2.4.3 JavaFX 21
2.4.4 Abstract Syntax Tree 22
2.4.5 JavaParser 23
BAB 3 METODOLOGI 24
3.1 Studi Literatur 24
3.2 Rekayasa Kebutuhan 25
3.3 Perancangan Sistem 24
3.4 Implementasi Sistem 24
3.5 Pengujian 24
3.6 Penarikan Kesimpulan dan Saran 24
BAB 4 Rekayasa Kebutuhan 28
4.1 Deskripsi Umum Sistem 28
4.2 Analisis Kebutuhan 29
4.3 Identifikasi Aktor 29
4.4 Daftar Kebutuhan 29
4.4.1 Kebutuhan Fungsional 30
4.4.2 Kebutuhan Non-Fungsional 31
4.5 Pemodelan Kebutuhan 31
4.5.1 Use Case Diagram 31
4.5.2 Use Case Scenario 32
BAB 5 PERANCANGAN DAN IMPLEMENTASI 35
5.1 Perancangan Sistem 34
5.1.1 Perancangan Arsitektur 34
5.1.2 Perancangan Komponen 41
5.1.2 Perancangan Antarmuka 43
5.2 Implementasi Sistem 47
5.2.1 Spesifikasi Sistem 47

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

5.2.2 Implementasi Kode Program 48
5.2.2 Implementasi Antarmuka 52
BAB 6 PENGUJIAN 54
6.1 Pengujian Unit 54
6.1.1 Pengujian Unit Method chooseFile 54
6.1.2 Pengujian Unit Method visit 56
5.2.1 Pengujian Unit Method calculateAC 61
6.2 Pengujian Integrasi 67
6.2 Pengujian Validasi 68
6.2.1 Pengujian Validasi Input Berkas 68
6.2.2 Pengujian Validasi Ekstraksi Abstract Syntax Tree 70
6.2.3 Pengujian Validasi Hitung Kompleksitas 71
6.2.4 Pengujian Validasi Reset 72
6.2.5 Pengujian Akurasi 73
6.2.6 Pengujian Performa 76
BAB 6 Penutup 79
6.1 Kesimpulan 79
6.2 Saran 79
DAFTAR REFERENSI 80

DAFTAR TABEL

Tabel 2.1 Notasi Use Case Diagram 10
Tabel 2.2 Notasi Sequence Diagram 11
Tabel 2.3 Notasi Class Diagram 12
Tabel 2.4 Dampak desain OO pada kompleksitas dan keandalan perangkat lunak 14
Tabel 2.5 Complexity Number 16
Tabel 2.5 Bobot faktor attribute 18
Tabel 2.5 BCs dengan bobot kognitif masing-masing (W_c) 19
Tabel 4.1 Identifikasi Aktor 29
Tabel 4.2 Daftar Kebutuhan Fungsional 30
Tabel 4.3 Daftar Kebutuhan Non-Fungsional 31
Tabel 4.4 Use Case Scenario Input Berkas 32
Tabel 4.5 Use Case Scenario Ekstraksi Abstract Syntax Tree 33
Tabel 4.6 Use Case Scenario Hitung Kompleksitas 33
Tabel 4.7 Use Case Scenario Reset 34
Tabel 5.1 Perancangan antarmuka input berkas 44
Tabel 5.2 Perancangan antarmuka daftar file 45
Tabel 5.3 Perancangan antarmuka hasil perhitungan 46
Tabel 5.4 Spesifikasi perangkat keras 47
Tabel 5.5 Spesifikasi perangkat lunak 47
Tabel 6.1 Hasil pengujian unit method chooseFile 55
Tabel 6.2 Hasil pengujian unti method visit 59
Tabel 6.3 Hasil pengujian unit method calculateAC 63
Tabel 6.4 Hasil pengujian integrasi 67
Tabel 6.5 Pengujian validasi input berkas 69
Tabel 6.6 Pengujian validasi input berkas alternatif 1 69
Tabel 6.7 Pengujian validasi input berkas alternatif 2 70
Tabel 6.8 Pengujian validasi *ekstraksi Abstract Syntax Tree* 70

Tabel 6.9 Pengujian validasi *ekstraksi Abstract Syntax Tree* alternatif 1 71
Tabel 6.10 Pengujian validasi *ekstraksi Abstract Syntax Tree* alternatif 2 71
Tabel 6.11 Pengujian validasi hitung kompleksitas 72
Tabel 6.12 Pengujian validasi reset..... 72
Tabel 6.13 Pengujian akurasi 73
Tabel 6.14 Perbandingan perhitungan manual dan perhitungan sistem 74
Tabel 6.15 Pengujian performa..... 76
Tabel 6.16 Perbandingan waktu eksekusi perhitungan manual dan sistem 77

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

DAFTAR GAMBAR

Gambar 2.1 Model Waterfall 9
Gambar 2.2 Struktur JavaParser dan Abstract Syntax Tree (AST) 23
Gambar 3.1 Diagram Metodologi Penelitian 24
Gambar 4.1 Use Case Diagram Sistem 31
Gambar 5.1 Sequence Diagram Input Berkas 36
Gambar 5.2 Sequence Diagram Ekstraksi Abstract Syntax Tree 37
Gambar 5.3 Sequence Diagram Hitung Kompleksitas 39
Gambar 5.4 Class Diagram 40
Gambar 5.5 Perancangan Antarmuka Input Berkas 44
Gambar 5.6 Perancangan Antarmuka Daftar File 45
Gambar 5.7 Perancangan Antarmuka Hasil Perhitungan 46
Gambar 5.8 Implementasi Antarmuka Input Berkas 52
Gambar 5.9 Implementasi Antarmuka Daftar File 53
Gambar 5.10 Implementasi Antarmuka Hasil Perhitungan 53
Gambar 6.1 *Flow Graph Method* chooseFile 55
Gambar 6.2 *Flow Graph Method* visit 58
Gambar 6.3 *Flow Graph Method* calculateAC 63
Gambar 6.4 *Flow Graph Method* calculateAC 66

BAB 1 PENDAHULUAN

1.1 Latar belakang

Software maintenance adalah modifikasi apapun dari produk perangkat lunak yang bertujuan untuk memperbaiki kesalahan, meningkatkan kinerja, atau tindakan untuk menyesuaikan produk perangkat lunak terhadap lingkungan yang berubah (IEEE, 1983). *Software maintenance* merupakan tahap pengembangan perangkat lunak yang sulit dan mahal, jika proyek pengembangan perangkat lunak membutuhkan waktu berbulan-bulan hingga bertahun-tahun, maka tahap *maintenance* akan berlangsung selama bertahun-tahun hingga lebih (Serna dan Serna, 2014). Diperkirakan banyak masalah dalam *software maintenance* adalah hasil dari model dan penerapan teknik yang tidak memadai dalam pengembangan perangkat lunak (Schneidewind, 1987). Pilihan konsep pengembangan perangkat lunak yang tidak tepat dapat menghasilkan perangkat lunak yang tidak berkualitas, kompleks, mahal dan sulit untuk diubah (Banker dan Kemerer, 1998). Satuan ukuran untuk menentukan kualitas perangkat lunak pada proses rekayasa perangkat lunak terdiri dari kompleksitas program, modularitas yang efektif, dan besarnya program (Fathoni, 2009).

Kompleksitas perangkat lunak menunjukkan kerumitan dalam memahami, memelihara, memodifikasi dan menggunakan kembali perangkat lunak (Maheswaran dan Aloysious, 2018). Jika kompleksitas tinggi maka *developer* akan kesulitan dalam memahami perangkat lunak ketika akan melakukan perubahan, sehingga mengakibatkan *maintenance* tidak dapat dilakukan secara optimal. Oleh karena itu kompleksitas harus dipantau selama pengembangan perangkat lunak khususnya pada tahap implementasi, cara untuk memantau kompleksitas yaitu menggunakan *software metrics* (Aloysius dan Arockiam, 1991). *Software metrics* berperan penting dalam industri perangkat lunak untuk menetapkan kualitas perangkat lunak.

Kualitas dan keandalan perangkat lunak juga ditentukan dari pemilihan konsep pengembangan perangkat lunak yang sesuai. Selama dua dekade terakhir penggunaan pendekatan berorientasi objek lebih mendominasi dibandingkan dengan pendekatan terstruktur (Mishra dan Akman, 2008). Banyak ahli ilmu komputer dan *developer* percaya bahwa kompleksitas program dapat dikendalikan secara efektif menggunakan pemrograman berorientasi objek (De Silva, et al., 2015). Pemrograman berorientasi objek mendukung penggunaan *inheritance, encapsulation, coupling,* dan *cohesion* yang dapat meningkatkan kualitas dan keandalan perangkat lunak (Yadav dan Khan, 2012). Akan tetapi penggunaan *inheritance* yang tinggi akan menimbulkan kompleksitas yang tinggi pula sehingga sistem kurang dapat diandalkan (Yadav dan Khan, 2012). Hal ini mengakibatkan banyak peneliti yang mengusulkan metrik berorientasi objek (OO) untuk mengukur kualitas sistem, salah satu metrik OO yang terkenal yaitu metrik yang diusulkan oleh Chidamber dan Kemerer (1994), yang terdiri dari *Weighted Method per Class* (WMC), *Depth of Inheritance Tree* (DIT), *Response for Class* (RFC),

Number of Children (NOC), *Lack of Cohesion of Methods* (LOCM) and *Coupling Between Objects* (CBO). Kemudian muncul berbagai metrik OO khusus untuk menghitung kompleksitas perangkat lunak. Salah satu metrik tersebut adalah metrik yang diusulkan oleh Mishra et. al (2008) yang bernama *Weighted Class Complexity* (WCC).

Berdasarkan paparan sebelumnya, penggunaan *inheritance* akan menentukan tingkat kompleksitas dari desain perangkat lunak, sehingga diperlukan suatu perhitungan kompleksitas yang dapat mendefinisikan tingkat kesulitan dalam memahami kelas yang menggunakan *inheritance* pada suatu program. Secara umum, kompleksitas kelas diukur berdasarkan salah satu dari faktor-faktor seperti *Line of Codes* (LOC), *Number of Methods* (NOM), *Number of Attributes* (NOA) dan sebagainya. Ada beberapa teknik, metode dan metrik baru dengan berbagai faktor yang dikembangkan untuk menghitung kompleksitas kelas khususnya pada pendekatan berorientasi objek (Arockiam dan Aloysious, 2011). Dengan menambahkan beberapa faktor sebagai parameter pengukuran kompleksitas tersebut diharapkan dapat menghasilkan pengukuran yang lebih tepat dan akurat. Oleh karena itu, dipilih metrik *Cognitive Weighted Inherited Class Complexity* (CWICC) untuk perhitungan kompleksitas perangkat lunak.

Metrik CWICC pertama kali diperkenalkan oleh Maheswaran dan Aloysious (2018), dimana pada metrik ini melakukan perhitungan kompleksitas dengan cara menjumlahkan kompleksitas pada *method*, *attribute*, dan *inheritance*. Metrik tersebut dipilih karena merupakan metrik yang cukup spesifik untuk menghitung kompleksitas yang dipengaruhi oleh penggunaan *inheritance,* karena tidak hanya mempertimbangkan kompleksitas internal kelas namun juga kompleksitas kelas yang diturunkan. Selain itu metrik ini juga merupakan perbaikan dari metrik kompleksitas sebelumnya yaitu metrik *Attribute Weighted Class Complexity* (AWCC) yang menghitung kompleksitas perangkat lunak menggunakan *inheritance* sebagai salah satu parameter perhitungan. Pada metrik AWCC ditemukan kelemahan, yaitu dalam menghitung kompleksitas kelas yang diturunkan (*inheritance*), *Cognitive Load* atau bobot pemahaman yang ditetapkan untuk setiap tingkat hierarki pewarisan diasumsikan selalu sama yaitu satu, namun bobot pemahaman untuk memahami tingkat pewarisan tidak bisa sama (Maheswaran dan Aloysious, 2018).

Perhitungan kompleksitas kelas menggunakan metrik CWICC saat ini masih dilakukan secara manual. Perhitungan tersebut dilakukan dengan menganalisis setiap baris kode kemudian melakukan perhitungan nilai kompleksitas menggunakan metrik CWICC. Dengan menggunakan perhitungan seperti itu akan membutuhkan waktu yang lebih lama dengan upaya yang lebih besar untuk melakukannya, karena kelas yang akan dianalisis tidak mungkin hanya satu kelas dan hanya beberapa baris kode, namun berupa sejumlah program yang terdiri dari banyak kelas dan baris kode. Selain itu dalam melakukan perhitungan tersebut juga dibutuhkan sumber daya manusia yang handal, yaitu yang paham dengan model perhitungan pada metrik CWICC. Sedangkan untuk mendapatkan sumber daya manusia seperti itu bukan hal yang mudah.

Berdasarkan permasalahan-permasalahan yang telah dipaparkan, dibuatlah suatu sistem yang dapat melakukan perhitungan kompleksitas kelas secara otomatis. Sistem tersebut diharapkan dapat digunakan sebagai alternatif dalam membantu *developer* atau pengguna untuk menentukan kualitas perangkat lunak berdasarkan tingkat kesulitan pemahaman. Sistem tersebut dapat meminimalkan waktu dan upaya yang dibutuhkan, sebagai pertimbangan untuk perencanaan ulang kode sumber, sehingga pengembangan perangkat lunak khususnya pada tahap *maintenance* dapat dilakukan dengan lebih optimal. Sistem yang bertujuan untuk membantu perhitungan kompleksitas tersebut menggunakan metrik *Cognitive Weighted Inheritance Class complexity* (CWICC) dan dibangun menggunakan bahasa pemrograman Java.

1.2 Rumusan masalah

Berdasarkan uraian pada latar belakang, didapatkan rumusan masalah untuk pembangunan Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak menggunakan Metrik *Cognitive Weighted Inherited Class Complexity* adalah sebagai berikut:

1. Bagaimana hasil analisis kebutuhan pada sistem kakas bantu perhitungan kompleksitas perangkat lunak menggunakan metrik *Cognitive Weighted Inherited Class Complexity?*
2. Bagaimana hasil perancangan dan implementasi sistem pada kakas bantu perhitungan kompleksitas perangkat lunak menggunakan metrik *Cognitive Weighted Inherited Class Complexity* yang sesuai dengan kebutuhan?
3. Bagaimanakah hasil pengujian pada sistem kakas bantu perhitungan kompleksitas perangkat lunak menggunakan metrik *Cognitive Weighted Inherited Class Complexity?*
4. Bagaimana hasil akurasi pada sistem kakas bantu perhitungan kompleksitas perangkat lunak menggunakan metrik *Cognitive Weighted Inherited Class Complexity* pada suatu perangkat lunak?

1.3 Tujuan

Tujuan dari pembangunan Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan Metrik *Cognitive Weighted Inherited Class Complexity* yang ingin dicapai adalah sebagai berikut:

1. Menganalisis kebutuhan dalam pembangunan kakas bantu perhitungan kompleksitas perangkat lunak menggunakan metrik *Cognitive Weighted Inherited Class Complexity.*
2. Merancang dan mengimplementasikan kakas bantu perhitungan kompleksitas perangkat lunak menggunakan metrik *Cognitive Weighted Inherited Class Complexity* berdasarkan hasil analisis kebutuhan.
3. Menguji sistem kakas bantu perhitungan kompleksitas perangkat lunak menggunakan metrik *Cognitive Weighted Inherited Class Complexity* yang sesuai dengan kebutuhan.

4. Menguji tingkat akurasi pada sistem kakas bantu perhitungan kompleksitas perangkat lunak menggunakan metrik *Cognitive Weighted Inherited Class Complexity.*

1.4 Manfaat

Pada penelitian ini diharapkan dapat memberikan manfaat bagi pengguna sistem atau *developer*. Pengguna dapat menggunakan kakas bantu perhitungan kompleksitas menggunakan metrik *Cognitive Weighted Inherited Class Complexity,* untuk mempermudah dalam melakukan perhitungan nilai kompleksitas kode sumber pada suatu perangkat lunak secara otomatis berdasarkan tingkat kesulitan pemahaman kode sumber secara lebih cepat dan akurat, dimana hasil dari perhitungan kompleksitas tersebut dapat digunakan sebagai pertimbangan untuk perencanaan ulang kode sumber dari perangkat lunak yang dibangun agar tahap *maintenance* dapat dilakukan dengan baik.

1.5 Batasan masalah

Pada penelitian ini, untuk mencegah adanya penyimpangan selama proses penelitian dan dalam tujuan yang telah ditentukan, maka ditetapkan batasan masalah pada penelitian ini adalah sebagai berikut:

1. Kakas bantu perhitungan kompleksitas menggunakan metrik *Cognitive Weighted Inherited Class Complexity* hanya digunakan untuk menghitung kompleksitas pada bahasa pemrograman Java.
2. Pada penelitian ini hanya akan dilakukan perhitungan kompleksitas menggunakan metrik *Cognitive Weighted Inherited Class Complexity* dan perhitungan manual untuk perbandingan dan pengujian.
3. Sistem akan dibangun dengan menggunakan bahasa pemrograman Java dan hanya ditujukan pada perangkat berbasis desktop.

1.6 Sistematika pembahasan

Pada sistematika pembahasan berisi penjelesan singkat pada bagian-bagian yang terdapat dalam penulisan skripsi ini, yaitu:

BAB 1: PENDAHULUAN

Bab ini berisi penjelasan tentang uraian dari latar belakang yang mendasari penelitian, rumusan masalah, tujuan penelitian, manfaat penelitian, batasan masalah dan sistematika pembahasan pada penelitian pembangunan Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan Metrik *Cognitive Weighted Inherited Class Complexity.*

BAB 2: LANDASAN KEPUSTAKAAN

Bab ini berisi penjelasan tentang teori dari jurnal penelitian yang dijadikan acuan dalam penelitian skripsi ini yaitu jurnal berjudul "*Cognitive Weighted Inherited Class Complexity Metrix*" oleh K. Maheswaran and A. Aloysius (2018). Selain itu juga terdapat teori ilmiah yang dibutuhkan sebagai bahan pendukung dalam

melakukan penelitian pembangunan Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan Metrik *Cognitive Weighted Inherited Class Complexity.*

BAB 3: METODOLOGI

Bab ini berisi penjelasan tentang prosedur atau langkah kerja yang akan digunakan untuk melakukan seluruh proses pada pembangunan sistem yaitu mulai dari tahap rekayasa kebutuhan, perancangan, implementasi, pengujian, hingga kesimpulan yang didapat dari pembangunan kakas bantu perhitungan kompleksitas perangkat lunak menggunakan metrik *Cognitive Weighted Inherited Class Complexity*.

BAB 4: REKAYASA KEBUTUHAN

Bab ini berisi penjelasan tentang tahap rekayasa kebutuhan yang dilakukan pada penelitian pembangunan Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan Metrik *Cognitive Weighted Inherited Class Complexity.* Pada tahap rekayasa kebutuhan terdiri dari deskrispi sistem, analisis kebutuhan, identifikasi aktor, definisi kebutuhan fungsional dan non-fungsional beserta deskripsinya, dan pemodelan kebutuhan.

BAB 5: PERANCANGAN DAN IMPLEMENTASI

Bab ini berisi penjelasan tentang perancangan dan implementasi pada sistem Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak menggunakan Metrik *Cognitive Weighted Inherited Class Complexity*. Pada sub bab perancangan sistem terdiri dari perancangan arsitektur, perancangan komponen, dan perancangan antarmuka. Sedangkan pada sub bab implementasi sistem terdapat spesifikasi sistem, implementasi kode program, dan implementasi antarmuka.

BAB 6: PENGUJIAN

Bab ini berisi penjelasan tentang strategi-strategi pengujian yang dilakukan beserta hasil pengujian pada sistem Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak menggunakan Metrik *Cognitive Weighted Inherited Class Complexity* yang telah dibangun. Strategi tersebut terdiri dari pengujian unit, pengujian integrasi, dan pengujian validasi.

BAB 7: PENUTUP

Bab ini berisi penjelasan tentang kesimpulan hasil yang didapat dari penelitian ini, selain itu juga terdapat saran sebagai pertimbangan dan perbaikan untuk pengembangan sistem selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

Bab Landasan Kepustakaan akan membahasa tentang kajian pustaka dan teori yang mendukung dalam melakukan peneIitian ini. Pada kajian pustaka akan membahas jurnaI peneIitian yang menjadi acuan dalam melakukan penelitian skripsi ini. Penelitian tersebut berjudul "*Cognitive Weighted Inherited Class Complexity Metric*" yang menjeIaskan tentang pengenaIan dan penggunaan metrik *Cognitive Weighted Inherited Class Complexity* (CWICC) yang berfungsi untuk melakukan perhitungan kompleksitas perangkat lunak. Teori-teori yang akan digunakan untuk mendukung pada peneIitian ini yaitu Rekayasa Perangkat Lunak, *Software Development Life Cycle*, *Model Waterfall*, Pendekatan Berorientasi Objek, Pemodelan berorientasi Objek, Hubungan Pendekatan Berorientasi Objek dan Kompleksitas, Pengujian Perangkat Lunak, *Cognitive Weighted Inherited Class Complexity Metric, Source Code*, Bahasa Pemrograman Java, *Abstract Syntax Tree* dan JavaParser.

2.1 Kajian Pustaka

Kajian pustaka yang digunakan pada penelitian ini akan menjelaskan penelitian-penelitian yang telah dilakukan sebelumnya dan membahas jenis yang sama. Terdapat dua penelitian yang digunakan sebagai bahan dalam melakukan analisis kebutuhan sistem. Penelitian pertama adalah penelitian yang digunakan sebagai acuan utama penelitian ini yang berjudul "*Cognitive Weighted Inherited Class Complexity Metrix*" yang dilakukan oleh K. Maheswaran dan A. Aloysius. Pada penelitian tersebut membahas tentang penggunaan metrik CWICC untuk menghitung kompleksitas perangkat lunak. Penelitian tersebut juga merupakan perbaikan dari penelitian sebelumnya yang membahas tentang metrik *Attribute Weighted Class Complexity* (AWCC) yang bertujuan untuk memperbaiki kelemahan pada metrik tersebut. Pada penelitian ini didapatkan hasil bawa metrik CWICC dapat digunakan untuk menghitung kompleksitas kelas secara akurat dibandingkan metrik kompleksitas sebelumnya (Maheswaran dan Aloysius, 2018).

Penelitian kedua dilakukan oleh L. Arockiam dan A. Aloysius yang berjudul "*Attribute Weighted Class Complexity: A New Metric for Measuring Cognitive Complexity of OO Systems*". Metrik AWCC merupakan metrik yang dikenalkan sebelum metrik CWICC. Kelemahan pada metrik AWCC adalah bahwa *cognitive load* (bobot pemahaman) untuk memahami tingkat L dari kelas turunan diasumsikan selalu satu dan kompleksitas karena *attribute* dan *method* yang digunakan kembali tidak dihitung secara akurat. Maka pada metrik CWIIC mendefinisikan kembali *cognitive load* untuk tingkat hierarki pewarisan yang berbeda kemudian menghitung kompleksitas *attribute* dan *method* yang digunakan kembali (Maheswaran and Aloysius, 2018). Kesimpulan dari penelitian tersebut adalah metrik CWICC mampu diformulasikan untuk menghitung *cognitive complexity* (kompleksitas berdasarkan pemahaman) suatu kelas secara lebih akurat dibandingan metrik sebelumnya. Kompleskitas dihitung dengan mempertimbangkan kompleksitas internal kelas dan kompleksitas kelas yang

diturunkan. Kompleksitas internal kelas diukur dengan mempertimbangkan kompleksitas *attribute* dan *method*. Kompleksitas kelas yang diturunkan dihitung dengan mempertimbangkan aspek kognitif dari kedalaman kelas yang berbeda di mana kelas tersebut mewarisi kompleksitas *method* dan *attribute* yang digunakan kembali. Metrik CWICC telah membuktikan bahwa kompleksitas suatu kelas akan dipengaruhi oleh tingkat dimana kelas tersebut mewarisi kelas-kelas yang lain.

2.2 Rekayasa Perangkat Lunak

Rekayasa perangkat lunak adalah suatu penggunaan prinsip-prinsip teknis yang sistematis mengenai pendekatan dalam pengembangan, pengoperasian dan perawatan suatu perangkat lunak yang bertujuan untuk memperoleh perangkat lunak yang ekonomis, dapat dipercaya, dan bekerja secara efisien pada mesin (Pressman, 2010). Sommerville (2011) menyatakan bahwa rekayasa perangkat lunak merupakan disiplin ilmu yang mempelajari seluruh aspek produksi perangkat lunak mulai dari tahap spesifikasi perangkat lunak hingga tahap pemeliharaan perangkat lunak, agar waktu dan biaya yang dibutuhkan untuk mendapatkan kualitas terbaik dari perangkat lunak yang dikembangkan dapat digunakan dengan lebih optimal. Untuk mencapai tujuan tersebut, dibutuhkan sekumpulan aktifitas sistematis dan terorganisir yang disebut dengan *software process*, dan kreativitas dalam menentukan metode pengembangan yang akan digunakan pada suatu kondisi tertentu.

Sommerville (2011) membagi *software process* menjadi 4 tahapan utama yang umum, yaitu:

1. *Software specification*, pada tahap ini dilakukan penentuan perangkat lunak yang akan dibangun berdasarkan pada batasan operasinya, dimana yang menentukan adalah *developer* dan *customer*.
2. *Software development*, pada tahap ini dilakukan perancangan dan pembangunan perangkat lunak.
3. *Software validation*, pada tahap ini dilakukan pengujian perangkat lunak untuk menemukan dan memperbaiki kesalahan sistem dan menjamin kesesuaiannya dengan apa yang diinginkan oleh *customer* sebelum diserahkan.
4. *Software evolution,* pada tahap ini dilakukan modifikasi pada perangkat lunak yang bertujuan untuk menyesuaikan dengan permintaan perubahan dari *customer* dan permintaan pasar.

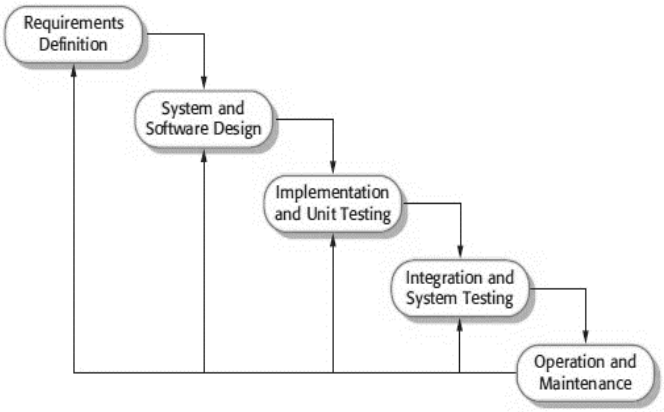
2.2.1 Software Development Life Cycle

Software Development Life Cycle (SDLC) adalah suatu proses yang digunakan di industri perangkat lunak dan berfungsi untuk mendesain, mengembangkan dan menguji kualitas perangkat lunak. Sharma dan Singh (2016) menyatakan bahwa SDLC bertujuan untuk dapat menghasilkan kualitas perangkat lunak yang sesuai atau melebihi harapan *customer*, serta dapat diselesaikan dalam waktu dan biaya yang telah ditentukan. Terdapat beberapa jenis model SDLC yang dapat digunakan untuk mengembangkan perangkat lunak pada saat ini yaitu model *waterfall,*

prototyping, iterative and incremental development, rapid application development, spiral development, dan *extreme programming*. Secara umum proses dalam SDLC dibagi menjadi lima fase yaitu:

1. *Requirement analysis*
   Requirement Analysis atau analisis kebutuhan adalah fase awal dari pengembangan perangkat lunak yang bertujuan untuk memahami kebutuhan *customer* dan mendokumentasikannya dengan benar. Fase ini harus dilakukan dengan cermat dan teliti dalam menggali kebutuhan *customer*. Hasil dari fase analisis kebutuhan adalah spesifikasi kebutuhan yang akan menjadi dasar dalam pengembangan perangkat lunak.
2. *Design*
   Tujuan dari fase ini adalah mengubah setiap kebutuhan dari fase sebelumnya menjadi representasi *blueprint* perangkat lunak yang akan dikembangkan sebelum masuk ke fase implementasi (*coding*). Pada fase ini akan dihasilkan *Software Design Document* (SDD).
3. *Coding*
   Coding adalah fase yang bertujuan untuk mengubah SDD menjadi kode program dengan menggunakan bahasa pemrograman agar dapat dijalankan oleh mesin. Fase ini merupakan implementasi dari fase *design* dan fase yang menghasilkan kode program.
4. *Testing*
   Fase ini bertujuan untuk menemukan dan memperbaiki kesalahan sistem sebelum diserahkan kepada *customer*. Pengujian yang efektif akan menghasilkan produk perangkat lunak yang berkualitas, dapat diandalkan, memiliki biaya *maintenance* rendah, akurat, dan menyenangkan pengguna.
5. *Maintenance*
   Fase ini dilakukan setelah penyerahan produk perangkat lunak. Jika ditemukan *error* pada perangkat lunak yang lolos dari fase pengujian atau perlu adanya perubahan struktur sistem sesuai permintaan *customer* dan pasar maka akan diimplementasikan pada fase ini (Kumar, et al., 2013).

2.2.2 Model Waterfall



Gambar 2.1 Waterfall Model
Sumber: Ian Sommerville (2011)

Model waterfall adalah salah satu jenis dari *Software Development Life Cycle* (SDLC). Sommerville (2011) menyatakan bahwa tahap utama dari model *waterfall* akan langsung mencerminkan aktifitas pengembangan dasar perangkat lunak. Terdapat lima tahap pada model *waterfall* yang ditunjukkan pada Gambar 2.1, yaitu *requirement analysis and definition*, *system and software design*, *implementation and unit testing*, *integration and system testing*, serta *operation and maintenance*.

Berdasarkan 5 tahapan pada model waterfall, berikut ini merupakan penjelasan dari tahapan pada model tersebut:

1. *Requirement Analysis and Definition*
   Pada tahap ini, akan didefinisikan kebutuhan sistem yang akan dibangun melalui konsultasi dengan pengguna. Kebutuhan-kebutuhan tersebut akan ditentukan secara rinci dan didokumentasikan dengan benar. Tahap ini bertujuan untuk mengidentifikasi spesifikasi sistem berdasarkan kebutuhan, kendala, dan tujuan sistem yang didapatkan dari pengguna. Hasil dari tahap ini adalah *Software Requirement Specification* (SRS).
2. *System and Software Design*
   Pada tahap ini, akan dibentuk arsitektur sistem secara keseluruhan yang sesuai dengan kebutuhan yang telah didefinisikan pada tahap sebelumnya dan mengalokasikannya pada perangkat keras atau perangkat lunak. Pada perancangan perangkat lunak dilakukan identifikasi dan penggambaran abstraksi dasar sistem perangkat lunak beserta hubungannya.
3. *Implementation and Unit Testing*
   Pada tahap ini, hasil berupa desain perangkat lunak dari tahap sebelumnya akan direalisasikan menjadi satu kesatuan unit program. Setiap unit tersebut akan diuji untuk memastikan apakah telah sesuai dengan spesifikasi.
4. *Integration and System Testing*
   Pada tahap ini, antara satu unit program dengan unit program yang lain akan diintegrasikan, kemudian diuji sebagai satu kesatuan sistem yang utuh dengan tujuan untuk memastikan apakah sistem sudah memenuhi kebutuhan yang telah ditetapkan sebelum diserahkan kepada pengguna.
5. *Operation and Maintenance*
   Pada tahap ini, sistem akan diinstal dan mulai dapat digunakan. Selain itu pada tahap ini, juga dapat dilakukan perbaikan *error* yang lolos dari tahap sebelumnya, serta pengembangan sistem yang berupa penambahan fitur atau fungsi baru yang sesuai dengan permintaan perubahan.

Pada penelitian ini digunakan model waterfall dalam pembangunan sistem. Pemilihan model waterfall ini dikarenakan permasalahan dan tujuan pada penelitian ini sudah jelas sehingga tidak membutuhkan iterasi di setiap fase.

2.2.3 Pendekatan Berorientasi Objek

Pada pengembangan perangkat lunak terdapat dua pendekatan yang dapat digunakan, yaitu pendekatan terstruktur dan pendekatan berorientasi objek atau biasa disebut *Object Oriented* (OO). Pada pendekatan berorientasi objek proses pengembangan perangkat lunak dimulai dari tahap analisis kebutuhan yang

disebut dengan *Object Oriented Analysis* (OOA) untuk mengidentifikasi kebutuhan sistem dan menentukan objek beserta kelas berdasarkan domain permasalahan. Ketika tahap OOA telah selesai dilakukan maka akan dilanjutkan ke tahap *Object Oriented Design* (OOD) atau tahap perancangan. Pada tahap OOD akan dilakukan perancangan arsitektur, komponen, data, dan antarmuka. Tahap OOD ini dilakukan berdasarkan pada hasil analisis pada tahap OOA sebelumnya. Proses selanjutnya yaitu merubah hasil perancangan menjadi bentuk kode program menggunakan bahasa pemrograman berorientasi objek, sehingga bisa dibaca oleh mesin, pada tahap ini disebut dengan *Object Oriented Programming* (OOP). Selanjutnya tahap pengujian, Pressman (2010) menyatakan bahwa, tahap pengujian pada pendekatan berorientasi objek dapat menggunakan pengujian unit, pengujian integrasi, dan pengujian validasi sebagai strategi pengujiannya.

2.2.4 Pemodelan Berorientasi Objek

Whitten dan Bentley (2007) menyatakan bahwa *Unified Modelling Language* (UML) adalah suatu kumpulan dari ketentuan *modelling* yang digunakan untuk melakukan spesifikasi atau mendeskripsikan sistem perangkat lunak dalam suatu kondisi dari objek. UML biasanya digunakan sebagai bentuk visualisasi dan dokumentasi dari segala artefak sistem perangkat lunak yang berdasarkan kebutuhan yang didapat dari tahap OOA dan hasil perancangan dari tahap OOD (Pressman, 2010). Pada UML terdapat beberapa jenis diagram, diantaranya yaitu:

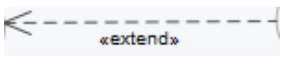
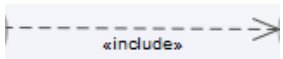
1. *Use Case* Diagram

Kurniawan (2018) menyatakan bahwa, *use case diagram* adalah diagram yang bertujuan untuk menjelaskan secara visual interaksi antara aktor dan sistem. Gambaran berupa diagram tersebut dapat menjelaskan konteks sistem dengan baik sehingga batasan sistem terlihat jelas (Larman, 2005). Pada *use case diagram* terdapat beberapa elemen penting yang harus digambarkan yaitu berupa *actor*, *use case,* dan relasi (*asociation*, *include*, *extend* dan *generalization*). Notasi dari *use case diagram* ditunjukkan pada Tabel 2.1.

Tabel 2.1 Notasi *Use Case Diagram*

Notasi	Deskripsi
Actor	Notasi yang menyatakan pengguna sistem atau segala hal dari luar sistem yang dapat berinteraksi atau menggunakan sistem. Setiap aktor bisa berinteraksi dengan satu atau banyak *use case.* Aktor bisa berupa orang (sesuai dengan perannya) atau sistem komputer yang lain
Use Case	Notasi yang menyatakan perilaku yang harus dijalankan oleh sistem dalam kaitan dengan satu atau lebih aktor. Setiap *use case* dapat dijalankan oleh banyak aktor

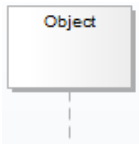
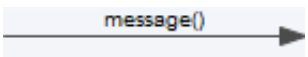
Tabel 2.2 Lanjutan

Notasi	Deskripsi
Association	Notasi yang menyatakan hubungan antara aktor dengan *use case*
Extend 	Notasi yang menyatakan fungsionalitas pada suatu *base use case* dapat diperluas oleh *supplier use case*, jika dibutuhkan
Includes 	Notasi yang menyatakan fungsionalitas pada *base use case* hanya dapat dipenuhi dan dijalankan dengan bantuan dari *supplier use case*
Generalization	Notasi yang menyatakan hubungan antar aktor-aktor yang bermakna umum-khusus. Dimana aktor yang melakukan generalisai merupakan aktor yang lebih umum daripada aktor lainnya

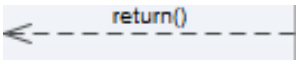
2. *Sequence* Diagram

Sequence diagram adalah diagram yang menggambarkan bagaimana sebuah objek berinteraksi dengan objak yang lain yaitu antara aktor dan sistem melalui pertukaran pesan. Interaksi tersebut akan menggambarkan sebuah eksekusi dari sebuah fitur (Whitten dan Bentley, 2007). Notasi yang terdapat pada *sequence diagram* ditunjukkan pada Tabel 2.3.

Tabel 2.3 Notasi *Sequence Diagram*

Notasi	Deskripsi
Actor	Notasi yang menyatakan pengguna sistem atau segala hal dari luar sistem yang dapat berinteraksi atau menggunakan sistem. Bisa berupa orang (sesuai dengan perannya) atau sistem komputer yang lain
Lifeline	Notasi yang menyatakan waktu dari suatu objek untuk melakukan aktivitas
Object 	Notasi yang menyatakan objek yang berinteraksi dalam pengiriman pesan pada sistem
Activation bar	Notasi yang menyatakan waktu atau durasi eksekusi suatu operasi dari objek yang melakukan interaksi
Input message 	Notasi yang menyatakan pemanggilan operasi dari suatu objek ke objek lain dan juga bisa dari aktor menuju objek. Dalam suatu *message* dapat memiliki parameter jika diperlukan

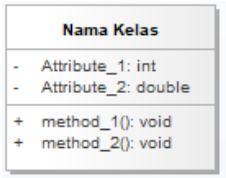
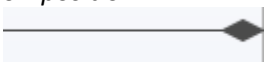
Tabel 2.4 Lanjutan

Notasi	Deskripsi
Output Message 	Notasi yang menyatakan keterangan bahwa objek yang dipanggil telah menjalankan operasinya dan menghasilkan suatu data yang dikembalikan ke objek yang memanggil
Frame	Notasi yang menyatakan satu atau lebih pesan untuk membagi fragmen *sequence* yang dapat menunjukkan *loop, alternative*, atau *option*, dan lain sebagainya
Boundary Class	Notasi yang menyatakan kumpulan suatu *class* yang menjadi antarmuka sistem atau tempat dimana dapat dilakukan interaksi antara aktor dengan sistem
Control class	Notasi yang menyatakan kumpulan suatu kelas yang terdiri dari logika alur sistem
Entity Class	Notasi yang menyatakan kumpulan kelas yang berupa entitas-entitas yang menyusun basis data sistem

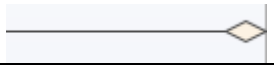
3. *Class* Diagram

Class diagram merupakan visualisasi grafis dari struktur objek yang menyusun suatu sistem. Pada *class diagram* menampilkan objek dan hubungan antar kelas-kelas pada sistem (Whitten and Bentley, 2007). Notasi-notasi yang terdapat pada *class diagram* ditunjukkan pada Tabel 2.2.

Tabel 2.5 Notasi *Class Diagram*

Notasi	Deskripsi
Class 	Notasi yang menyatakan kelas dari objek yang terdiri dari atribut, operasi yaitu *method*, dan relasinya. Notasi ini terdiri dari tiga kotak bagian. Pada bagian atas berisi nama kelas. Bagian tengah berisi nama atribut, tipe atribut dan jenis *modifier yaitu public, private,* dan *protected.* Pada bagian bawah berisi nama operasi, tipe data kembalian, dan juga jenis *modifier*
Association	Notasi yang menyatakan hubungan antara dua kelas secara umum, yaitu setiap objek dalam suatu kelas berhubungan dengan semua objek pada kelas yang lain, dan sebaliknya
Composition 	Notasi yang menyatakan hubungan ketergantungan antar kelas. Komposisi adalah bentuk khusus dari agregasi. Setiap kelas yang merupakan komposisi dari kelas lain tidak dapat berdiri sendiri ketika tidak lagi berhubungan

Tabel 2.6 Lanjutan

Notasi	Deskripsi
Aggregations 	Notasi yang menyatakan hubungan yang bermakna bahwa suatu kelas menjadi atribut untuk kelas yang lain. Setiap kelas yang memilik agregasi maka kelas tersebut dapat berdiri sendiri meskipun tidak lagi berhubungan
Generalizations	Notasi yang menyatakan hubungan antara dua kelas yaitu dimana salah satu kelas merupakan bentuk umum ke khusus dari kelas lain. Notasi ini juga dapat digunakan untuk menyatakan hubungan *inheritance*

2.2.5 Pemrograman Berorientasi Objek

Desain yang didapat dari perancangan perangkat lunak yang telah didefinisikan sebelumnya kemudian diimplementasikan menjadi sebuah kode program, metode tersebut merupakan pengertian dari pemrograman berorientasi objek. Kode program akan dijadikan sebagai kumpulan objek yang dapat mewakili setiap instansiasi sebuah kelas dan kelas tersebut adalah anggota dari hierarki kelas yang disatukan melalui *inheritance* (pewarisan). Baesens, et. al., (2015) menyatakan bahwa terdapat beberapa karakteristik dalam pemrograman berorientasi objek yaitu:

a. *Object*
   Object adalah suatu entitas yang terdiri dari *attribute* dan operasi yang berupa *method* yang menyusunnya. Suatu *object* merupakan hasil dari instansiasi sebuah kelas, dimana kelas tersebut merupakan kerangka kerja yang mendefinisikan *attribute* dan *method* yang terdapat pada setiap *object*.
b. *Inheritance*
   Inheritance adalah suatu teknik pembuatan kelas baru (*child class*) dengan memanfaatkan kelas yang telah ada sebelumnya (*parent class*). Pada *inheritance*, suatu *child class* akan mewarisi seluruh *attribute* dan *method* yang telah didefinisikan pada *parent class*. *Inheritance* bertujuan untuk memudahkan *developer* dalam mengelompokkan karakteristik umum pada suatu objek kedalam *parent class* yang kemudian mewariskan karakteristik tersebut ke *child class.*
c. *Polymorphism*
   Polymorphism adalah teknik pemberian nama yang sama pada beberapa *method* yang berbeda, dimana *method* tersebut memiliki fungsi yang berbeda. Terdapat dua jenis *polymorphism* yaitu *Overloading* dan *Overriding*. *Overloading* adalah penggunaan nama yang sama pada beberapa *method*, namun setiap *method* tersebut memiliki fungsi dan parameter yang berbeda. Parameter yang berbeda dapat dilihat dari jumlah parameter maupun dari jenis tipe data parameter. *Overriding* adalah memodifikasi atau melakukan perubahan pada *method* yang diturunkan dari *parent class*.
d. *Encapsulation*
   Encapsulation adalah suatu teknik yang berfungsi untuk menyembunyikan data suatu objek dari lingkungan luar. *Encapsulation* dapat mencegah segala

jenis akses dengan berbagai cara apapun ke data yang telah dienkapsulasi. Dalam mengakses data yang dienkapsulasi hanya dapat dilakukan dengan cara mengakses *method* yang telah didefinisikan untuk mengubah maupun membaca data tersebut yaitu dapat menggunakan *mutator* dan *accessor*. Dengan menggunakan enkapsulasi dapat mendefinisikan akses disetiap *attribute* atau *method* yang ada disebuah kelas sesuai dengan kebutuhan.

2.2.6 Hubungan Pemrograman Berorientasi Objek dan Kompleksitas

Pemrograman berorientasi objek mendukung penggunaan *inheritance, encapsulation, coupling,* dan *cohesion* yang dapat meningkatkan kualitas dan keandalan perangkat lunak. Selain itu, pemrograman berorientasi objek bertujuan untuk meningkatkan kemampuan perangkat lunak yang dapat digunakan kembali, diuji, dipelihara dan diperluas (Yadav dan Khan, 2012). Namun penggunaan fitur-fitur tersebut selain berdampak baik terhadap kualitas dan keandalan perangkat lunak namun ternyata juga dapat berdampak buruk pada kompleksitas dan keandalan perangkat lunak jika digunakan sembarangan. Tabel 2.4 menunjukkan pengaruh konstruksi desain OO pada kompleksitas dan keandalan perangkat lunak.

Tabel 2.4 Dampak desain OO pada kompleksitas dan keandalan perangkat lunak

	Complexity	***Reliability***
Inheritance ↑	↑	↓
Coupling ↑	↑	↓
Cohesion ↑	↓	↑
Encapsulasion ↑	↓	↑

Sumber: Yadav dan Khan (2012)

Tanda panah ke atas menunjukkan dampak tinggi dan tanda panah ke bawah menunjukkan dampak yang lebih sedikit. Jika *inheritance* dan *coupling* tinggi, maka kompleksitas pasti akan tinggi dan menjadikan perangkat lunak kurang dapat diandalkan. Di sisi lain, jika *cohesion* dan *encapsulation* yang tinggi, kompleksitas akan rendah dan perangkat lunak menjadi dapat lebih diandalkan (Yadav dan Khan, 2012).

2.2.7 Pengujian Perangkat Lunak

Pressman (2010) menyatakan bahwa pengujian perangkat lunak bertujuan untuk menemukan dan memperbaiki kesalahan yang terdapat dalam perangkat lunak sebelum diserahkan kepada *customer*. Strategi pengujian perangkat lunak yang dapat digunakan pada pendekatan berorientasi objek adalah pengujian unit, pengujian integrasi, pengujian validasi dan pengujian sistem. Sedangkan metode yang digunakan adalah *white-box testing* dan *black-box testing*.

a. White-box testing

White-box testing adalah suatu metode dalam pengujian perangkat lunak, pengujian ini dilakukan berdasarkan pengetahuan tentang struktur dan komponen penyusun dari kode program. Pada *white-box testing*, akses pada kode sumber sangat penting pada proses pengujian ini (Sommerville, 2011). *White-box testing* merupakan pengujian yang memperdulikan struktur kontrol sebagai bagian dari *component level design* untuk menghasilkan kasus uji. Oleh karena itu, *white-box testing* hanya dapat didesain jika *component level design* yaitu kode sumber telah tersedia.

Basis path testing merupakan salah satu teknik pengujian *white-box testing* yang dapat menghasilkan kompleksitas logis dari rancangan secara procedural dan pengukuran ini dapat digunakan sebagai dasar untuk menentukan jalur yang akan dieksekusi. Kasus uji yang dihasilkan dapat menjamin eksekusi setiap program paling tidak dilakukan satu kali selama pengujian. Langkah-langkah dalam menentukan kasus uji pada teknik *basis path testing* adalah sebagai berikut:

1. Menentukan *flow graph* dari perancangan komponen kode sumber atau *pseudocode*.
2. Menentukan nilai *cyclomatic complexity* berdasarkan *flow graph* yang telah dibuat.
3. Menentukan *independent path* berdasarkan *flow graph* dan nilai *cyclomatic complexity* yang telah didapatkan.

Cyclomatic complexity merupakan *software metric* yang berfungsi untuk menghitung kompleksitas suatu program dengan cara menentukan *independent path* berdasarkan *flow graph* yang telah ditentukan dari rancangan kode program atau *pseudocode*. *Independent path* adalah jalur yang paling tidak memiliki satu *edge* yang belum pernah dilewati sebelumnya oleh jalur lain. Notasi *flow graph* biasanya terbentuk dari pernyataan *sequence, while, if-then-else*, dan *until* yang akan membentuk node-node yang saling berhubungan. Beberapa *node* akan dihubungkan melalui *edge*. Node tersebut mewakili *decisions*, sedangkan *edge* mewakili aliran kontrol. Berdasarkan *flow graph* yang telah dibuat dapat dilakukan perhitungan *cyclomatic complexity* V(G) yang didefinisikan pada rumus 2.1 dan rumus 2.2 berikut:

$$V(G) = E - N + 2 \quad (2.1)$$

Keterangan:

E = Jumlah *edge*

N = Jumlah *node*

$$V(G) = P + 1 \quad (2.2)$$

Keterangan:

P = Jumlah *predicate node*

Pada suatu program sederhana, nilai *cyclomatic complexity* dapat dihitung secara manual. Namun, jika program sangat kompleks dan melibatkan banyak *flow*

graph maka dapat menggunakan *automated tools*. Nilai *cyclomatic complexity* atau nilai V(G) akan memberikan gambaran kompleksitas yang setiap *range* nilai V(G) tertentu akan memiliki makna berbeda yang ditunjukkan pada Tabel 2.5.

Tabel 2.5 *Complexity Number*

Nilai kompleksitas	Makna
1-10	Struktur dan penulisan kode baik *High testability* Biaya dan usaha rendah
10-20	Kode kompleks *Medium testability* Biaya dan usaha *medium*
20-40	Kode sangat kompleks *Low testability* Biaya dan usaha tinggi
>40	Tidak semua bisa diuji Biaya dan usaha sangat tinggi

Sumber: Guru99 (2018)

b. Blackbox testing

Black-box testing atau disebut juga dengan *behavioural testing* adalah metode dalam pengujian perangkat lunak yang bertujuan untuk menguji sistem tanpa harus mengakses atau melakukan pengujian pada kode program, namun fokus berdasarkan kebutuhan fungsional perangkat lunak (Sommerville, 2011). Pada pengujian ini bertujuan untuk menemukan kesalahan atau *error* ketika kode sumber dieksekusi tanpa harus mengaksesnya, namun untuk mengatahui apakah fungsi pada sistem yang dibangun sesuai dengan tujuan awal produk tersebut. Pressman (2010) menyatakan bahwa *black-box testing* bukan merupakan pengganti *white-box testing*. Namun, sebagai pelengkap *white-box testing* dan dilakukan pada tahap terakhir pengujian karena pengujian ini tidak mempedulikan struktur kontrol.

Pengujian *black-box testing* dilakukan dengan menentukan sejumlah kasus uji yang didesain untuk dapat menemukan *error* pada level validasi perangkat lunak (*software validation level*). Pada setiap kasus uji, diharapkan agar dapat menemukan *error* sebanyak mungkin, namun dengan memanfaatkan waktu dan usaha yang seminimal mungkin. Setiap kasus uji dapat digunakan untuk melakukan uji coba terhadap setiap fungsi yang telah ditentukan. Setelah itu fungsi tersebut akan di validasi untuk menentukan apakah fungsi tersebut telah beroperasi sepenuhnya dan disaat yang bersamaan akan melakukan pencarian *error* yang mungkin terjadi. Teknik pengujian ini dilakukan pada bagian antarmuka dengan memberikan sebuah masukkan yang akan dijalankan secara keseluruhan dalam kebutuhan fungsi program. Pressman (2010) menyatakan bahwa *error* yang ditemukan ketika melakukan pengujian dengan menggunakan *black-box testing* akan dikategorikan berdasarkan beberapa kategori, yaitu:

1. Fungsi yang salah atau hilang
2. *Error* pada antarmuka
3. *Error* pada pengaksesan data struktur atau database eksternal
4. *Error* pada *behaviour* atau kinerja
5. *Error* pada inisialisasi dan *termination*

c. Performance testing

Performance testing adalah pengujian non-fungsional yang dilakukan untuk menentukan parameter sistem dalam hal respon dan stabilitas dalam beban aplikasi atau website. Pengujian ini mengukur kualitas sistem, seperti skalabilitas, kendala dan penggunaan sumber daya. Sedangkan Pressman (2010) menyatakan bahwa *performance testing* digunkan untuk menguji kinerja perangkat lunak secara run-time dalam konteks sistem yang terintegrasi. Pengujian ini dapat dilakukan diseluruh langkah dalam proses pengujian.

Terdapat beberapa jenis pengujian pada *performance testing* yaitu *load testing, stress testing, endurance testing* dan *spike testing*. Pada penelitian ini akan dilakukan *load testing* untuk memeriksa kemampuan dari aplikasi dalam melakukan *loading*. Proses untuk melakukan *performance testing* dibagi menjadi 7 proses, yaitu:

1. *Identify test environment*
   Memahami rincian konfigurasi hardware, software dan jaringan yang digunakan selama pengujian sebelum memulai proses pengujian. Hal ini bertujuan untuk membuat tes yang lebih efisien dan membantu mengidentifikasi kemungkinan hambatan pada saat pengujian.
2. *Determine performance criteria*
   Menetapkan kriteria kinerja dan tujuan sebagai tolok ukur kinerja untuk mengidentifikasikan kriteria keberhasilan proyek di luar tujuan dan kendala.
3. *Plan & Design*
   Membuat *test plan* untuk mengidentifikasi seluruh pengujian yang akan dilakukan.
4. *Configure test environment*
   Mempersiapkan lingkungan pengujian sebelum pengujian dilakukan.
5. *Implement test design*
   Membuat *scenario test* sesuai dengan perancangan
6. *Run the tests*
   Melakukan pengujian.
7. *Analyze, tune, & retest*
   Melakukan analisi hasil pengujian apakah ada peningkatan atau penurunan kinerja aplikasi.

Performance testing dapat dilakukan dengan bantuan *automated tools*, contohnya yaitu Jmeter, Open STA, Load Runner, Web Load, dan sebagainya. Pada penelitian ini performance testing dilakukan tanpa bantuan *tools.*

2.3 *Cognitive Weighted Inherited Class Complexity Metric*

Kemampuan memahami desain atau program OO berbanding lurus dengan kompleksitas desain atau kode program. Faktor-faktor yang menyebabkan kesulitan dalam memahami program, juga bertanggung jawab atas kompleksitas. Selain data dan operasi yang ada dikelas, *inheritance* adalah faktor pendorong dalam meningkatkan nilai kompleksitas. Metrik *Cognitive Weighted Inherited Class Complexity* (CWICC) pertama kali diusulkan oleh Maheswaran dan Aloysious (2018) yang merupakan kelanjutan dari penelitian metrik *Attribute Weighted Class Complexity* (AWCC) yang bertujuan untuk memperbaiki kelemahannya. Metrik CWIIC dapat dihitung dengan persamaan (2.1)

$$CWICC = \sum_{i=1}^{n} AC_i + \sum_{j=1}^{m} MC_j + \sum_{k=1}^{m1} IICC_k \quad (2.1)$$

AC adalah *Attribute Complexity,* digunakan untuk menghitung jumlah kompleksitas yang disebabkan oleh *attribute* pada sebuah kelas, yang dihitung menggunakan persamaan (2.2). Bobot faktor *attribute* didapat dari klasifikasi yang telah ditentukan oleh Wang (2002) dan ditunjukkan pada Tabel 2.5.

$$AC = (PDT * W_b) + (DDT * W_d) + (UDDT * W_u) \quad (2.2)$$

Keterangan :

PDT : Jumlah *Primary Data Type*

W_b : bobot kognitif dari PDT

DDT : Jumlah *Derived Data Type*

W_d : bobot kognitif dari DDT

UDDT : Jumlah *User Defined Data Type*

W_u : bobot kognitif dari UDDT

Tabel 2.5 Bobot faktor *attribute*

Jenis tipe data *attribute*	Bobot
PDT	1
DDT	2
UDDT	3

Sumber: Wang (2002)

Method Complexity (MC) adalah komponen dalam perhitungan kompleksitas kelas berdasarkan bobot kognitif dari tiga *Basic Control Structure* (BCS) yaitu *branch, iteration*, dan *sequence*. Pada perhitungan MC digunakan bobot kognitif BCS yang telah dimodifikasi oleh Wang dan Shao (2003) yang ditampilkan pada Tabel 2.6. Bobot kognitif tersebut juga telah divalidasi oleh J. Charles et al.

Tabel 2.6 BCSs Dengan Bobot Kognitif Masing-Masing (W_c)

Kategori	BCSs	Bobot Wc
Sequence	*Sequence* (SEQ)	1
Branch	*If-then-else* (ITE)	2
	Case (CASE)	3
Iteration	*For-do* (Ri)	3
	Repeat-until (R0)	3
	While-do (R1)	3
Embedded Component	*Function/Method call* (FC)	2
	Recursion (REC)	2
Concurrency	*Parallel* (PAR)	4
	Interupt (INT)	4

Sumber: Wang dan Shao (2003)

MC dapat dihitung menggunakan persamaan (2.3). Perhitungan tersebut dilakukan dengan menentukan jumlah bobot kognitif pada pengurutan, percabangan, iterasi dan sebagainya yang terdapat dalam *body method* sebuah kelas.

$$MC = \sum_{j=1}^{q}\left[\prod_{k=1}^{m}\sum_{i=1}^{n} W_c(j,k,i)\right] \quad (2.3)$$

Sedangkan *Improved Inherited Class Complexity (IICC)* adalah komponen dalam perhitungan kompleksitas kelas yang berdasarkan *inheritance*, dimana perhitungan tersebut dilakukan dengan menjumlahkan nilai kompleksitas dari *cognitive depth*, *method* dan *attribute* yang digunakan kembali. IICC dapat dihitung menggunakan persamaan (2.4).

$$IICC = CDC + \sum_{k=1}^{s} RMK_k + \sum_{l=1}^{t} RAC_l \quad (2.4)$$

CDC adalah *Cognitive depth Complexity* yang merupakan nilai kompleksitas berdasarkan level *inheritance* dan dapat dihitung menggunakan persamaan (2.5). Sedangkan RMC dan RAC menunjukkan kompleksitas *method* dan *attribute* yang digunakan kembali. DC adalah *Depth of Class* atau level dari *inheritance* yang dihitung dari kelas *root*. CW_l adalah bobot kognitif dari tingkat hierarki pewarisan tertentu yang sudah ditentukan oleh Maheswaran and Aloysious. Bobot kognitif tersebut dihitung berdasarkan fenomena kognitif yang dijelaskan oleh Wang (2002) untuk berbagai tingkat hierarki pewarisan.

$$CDC = DC * (CW_l) \quad (2.5)$$

Pada penelitian ini metrik CWICC digunakan untuk menghitung kompleksitas perangkat lunak berdasarkan masukan dari pengguna sistem berupa kode sumber dari perangkat lunak. Dalam perhitungannya akan menampilkan hasil perhitungan berupa nilai *Attribute Complexity, Method Complexity, Improved Inherited Class Complexity,* dan nilai dari CWICC dari setiap kelas. Selain itu juga menampilkan hasil total nilai CWICC perangkat lunak berdasarkan hasil nilai CWICC seluruh kelas yang diuji.

2.4 Teknologi Pengembangan Sistem

2.4.1 Source Code

Primadhana (2015) menyatakan bahwa *source code* atau kode sumber adalah suatu kumpulan deklarasi bahasa pemrograman komputer yaitu Java, C++, C#, PHP dan lain sebagainya. Kode sumber berfungsi untuk dapat memungkinkan *programmer* dalam berkomunikasi atau memberikan intruksi pada perangkat keras komputer dengan menggunakan perintah-perintah yang telah terdefinisi. Perintah tersebut ditulis dengan menggunakan kumpulan dari huruf, angka dan simbol-simbol khusus. Kode sumber bersifat statis tekstual, dapat dibaca oleh manusia dan merupakan deskripsi penuh dari program komputer yang secara otomatis dapat dikompilasi ke dalam bentuk yang dapat dieksekusi oleh mesin (Binkley, 2007).

Pada penelitian ini kode sumber digunakan sebagai masukan berkas yang akan diuji nilai kompleksitasnya, yaitu kode sumber yang menggunakan bahasa pemrograman Java.

2.4.2 Bahasa Pemrograman Java

Suarga (2008) menyatakan bahasa pemrograman Java adalah bahasa pemrograman yang *multi platform* dan *multi device*. Ketika sekali menuliskan sebuah kode program dengan menggunakan bahasa pemrograman Java, maka kode program tersebut dapat dijalankan hampir disemua komputer dan perangkat lain yang mendukung Java, dengan perubahan atau tanpa perubahan sama sekali pada kode sumbernya menggunakan *Java Virtual Machine* (JVM).

Java merupakan bahasa pemrograman yang diciptakan James Gosling, seorang *developer* dari Sun Microsystem pada tahun 1991. Java adalah bahasa pemrograman berorientasi objek dengan unsur-unsur seperti bahasa C++, karena Java dibuat berdasarkan kemampuan-kemampuan dari bahasa pemrograman sebelumnya (Hermawan, 2004). Suarga (2008) menyatakan bahwa bahasa pemrograman Java memiliki beberapa karakteristik yaitu:

1. Sederhana
   Bahasa pemrograman Java merupakan bahasa yang cukup sederhana, karena menggunakan *syntax* yang hampir serupa dengan bahasa pemrograman C++. Namun, dengan beberapa perbaikan terutama dengan menghilangkan *pointer* yang rumit dan tidak menyarankan *multiple inheritance*. Sehingga cukup dapat mudah dikenali dan dipahami oleh pemula.

2. Beorientasi Objek
   Bahasa pemrograman Java adalah bahasa pemrograman yang berorientasi objek, sehingga memungkinkan program untuk dibuat secara modular dan dapat digunakan kembali.
3. Terdistribusi
   Bahasa pemrograman Java dibuat untuk memudahkan distribusi aplikasi dengan terintegrasinya *networking libraries* ke dalam Java.
4. *Interpreted*
   Program yang ditulis dengan bahasa pemrograman Java dapat dijalankan dengan menggunakan *Interpreter*, yaitu JVM (*Java Virtual Machine*) yang memungkinkan program dapat dijalankan diberbagai *platform* yang berbeda.
5. *Robust*
   Bahasa pemrograman Java memiliki reliabilitas yang tinggi. Kompiler pada bahasa ini memiliki kemampuan untuk mendeteksi kesalahan yang lebih baik dibandingkan bahasa pemrograman lain.
6. *Secure*
   Java mempunyai beberapa mekanisme keamanan yang berfungsi menjaga agar aplikasi yang dijalankan tidak dapat digunakan untuk merusak sistem komputer yang menjalankan aplikasi tersebut.
7. *Architecture Neutral*
   Program Java yang telah dikompilasi dapat dijalankan di berbagai mesin dan tidak bergantung pada satu arsitektur komputer saja.
8. *Portable*
   Kode sumber dan program Java dapat dibawa dengan mudah ke berbagai *platform* yang berbeda tanpa harus dilakukan kompilasi ulang dengan menggunakan JVM.
9. *Multithreaded*
   Bahasa pemrograman Java dapat membuat sebuah program yang mampu melakukan beberapa pekerjaan secara simultan dan bersamaan.
10. *High-performance*
    Kinerja bahasa pemrograman ini sering kali disebut kurang jika dibandingkan dengan program C, tetapi kinerjanya bisa ditingkatkan dengan menggunakan *compiler* Java lain.
11. *Dynamic*
    Bahasa pemrograman Java didesain untuk mampu dijalankan di lingkungan yang dinamis. Java dapat memanggil berbagai kelas objek pada saat dibutuhkan, walaupun harus melalui jaringan. Selain itu dengan adanya *run time* pada Java dapat memberikan informasi asal kelas saat menggunakan objek tertenu.

2.4.3 JavaFX

JavaFX adalah kerangka kerja yang digunakan untuk mengembangkan aplikasi yang berbasis *Graphical User Interface* (GUI) (Sharan dan Kishori 2015). JavaFX bertujuan agar *developer* dapat mengembangkan perangkat lunak yang dapat digunakan diberbagai tipe perangkat seperti *embedded device, smartphone*,

TV, *tablet computer*, dan desktop secara lebih mudah. JavaFX *library* tersedia dalam bentuk Java *Application Programming Interface* dan memanfaatkan teknologi dari GPUs modern melalui *hardware accelerated graphics*, sehingga memungkinkan *developer* untuk dapat menggabungkan grafik, animasi, dan kontrol antarmuka dalam pengembangan perangkat lunak (Dea, Carl., et. al, 2017). Sharan dan Kishori (2015) menyatakan bahwa dengan menggunakan JavaFX dapat memberikan beberapa keuntungan yaitu:

1. Bahasa pemrograman yang digunakan adalah Java, sehingga dapat memanfaatkan fitur-fitur yang terdapat di Java seperti *multithreading, generics,* dan *lambda expressions.*
2. Mendukung penggunaan *data binding*, oleh karena itu aplikasi dapat melakukan *write, read, edit* dan *delete* data pada sebuah database.
3. Terdapat dua cara untuk membangun tampilan *User Interface* (UI) pada JavaFX. Cara pertama dengan menggunakan FXML, yaitu menggunakan bahasa XML dalam mendefinisikan UI. Cara kedua dengan menggunakan *visual editor* yaitu Scene Builder yang dapat diunduh secara gratis. Selain itu tampilan UI dapat dimanipulasi dengan mudah menggunakan *Cascading Style Sheets* (CSS).
4. Mendukung penggunaan efek dan animasi yang merupakan komponen penting dalam pengembangan aplikasi *game*.
5. Mendukung penggunaan *multimedia file,* oleh karena itu aplikasi dapat memainkan media berupa *file audio* atau *video* dengan memanfaatkan *codec* yang terdapat di perangkat.
6. Mendukung penambahan *web content* ke dalam aplikasi.

JavaFX pada penelitian ini digunakan untuk membangun tampilan UI pada perangkat lunak yang akan dikembangkan. Tampilan UI yang akan dikembangkan berupa *layout, control* UI (*button, progress indicator, table view* dan lain sebagainya). Penggunaan JavaFX pada penelitian ini dikarenakan JavaFX mendukung manipulasi objek kontrol UI dengan mudah melalui CSS dan scene builder untuk mengatur tampilannya sesuai dengan keinginan. Selain itu juga pada JavaFX telah tersedia *library* dari pihak ketiga untuk menambah kemampuan dalam membangun UI seperti *library* JFoenix, AnimateFX, dan lain sebagainya.

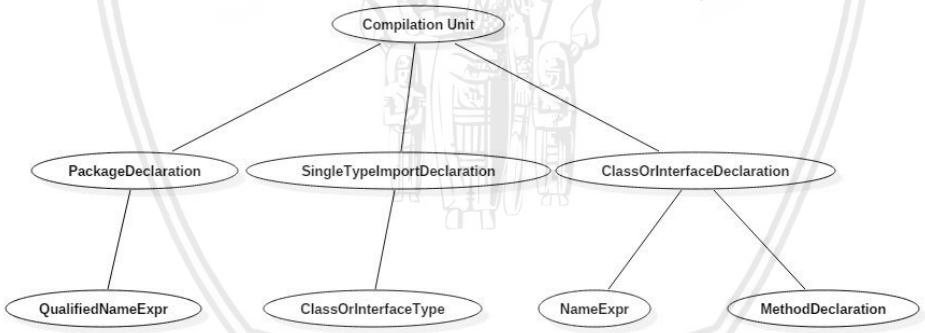
2.4.4 Abstract Syntax Tree

Abstract Syntax Tree (AST) adalah sebuah *tree* yang merepresentasikan struktur sebuah kode sumber yang menggunakan bahasa pemrograman tertentu. Setiap *node* dari *tree* akan menunjukkan konstruksi yang terdapat di dalam kode sumber. AST dibangun secara otomatis oleh *parser generator, parser generator* adalah bagian pemrosesan kompilasi kode sumber (Tripathy dan Naik, 2014). AST dapat digunakan untuk menganalisis kode sumber, *refactoring*, dan berbagai kegiatan lain yang membutuhkan pemahaman struktur kode sumber untuk mencapai tujuan tertentu. Contoh *parser generator* adalah ANTLR yang biasanya

digunakan pada bahasa pemrograman C++, sedangkan JavaParser biasanya digunakan pada bahasa pemrograman Java.

2.4.5 JavaParser

JavaParser adalah sebuah java *library* yang berguna untuk memfasilitasi *developer* dalam berinteraksi dengan kode sumber yaitu membuat, menganalisis, dan *refactoring* kode sumber Bahasa pemrograman Java yang akan direpresentasikan menjadi objek *Abstract Syntax Tree* (AST). Pada *library* ini menyediakan alat bantu untuk mekanisme pembacaan AST yang terdapat pada *runtime library* JavaParser, yaitu fitur *Visitor Support*. *Visitor* digunakan untuk mengidentifikasi pola-pola dari sebuah kode sumber. Selain itu *library* ini juga dapat memanipulasi struktur kode sumber yang telah dimasukan dan menuliskannya kembali ke dalam sebuah *file* kode sumber baru, sehingga dengan menggunakan fitur tersebut seorang *developer* dapat membuat perangkat lunak yang dapat menghasilkan kode sumber sesuai dengan yang diinginkan (Smith, Bruggen, and Tomassetti, 2018). Berdasarkan penjelasan sebelumnya setiap kode sumber yang telah di-*parser* akan direpresentasikan sebagai AST menggunakan Compilation Unit sebagai *root* yang mewakili keseluruhan isi dari kode sumber dengan *node-node* elemen teratas *file* yang terhubung ke *root* seperti deklarasi *import* atau *class*. Dari deklarasi kelas tunggal dapat dicapai beberapa *node* seperti *field* atau *method* dari kelas. Gambaran sederhana dari representasi struktur JavaParser AST akan ditunjukkan pada Gambar 2.2.



Gambar 2.2 Struktur JavaParser dan Abstract Syntax Tree (AST)

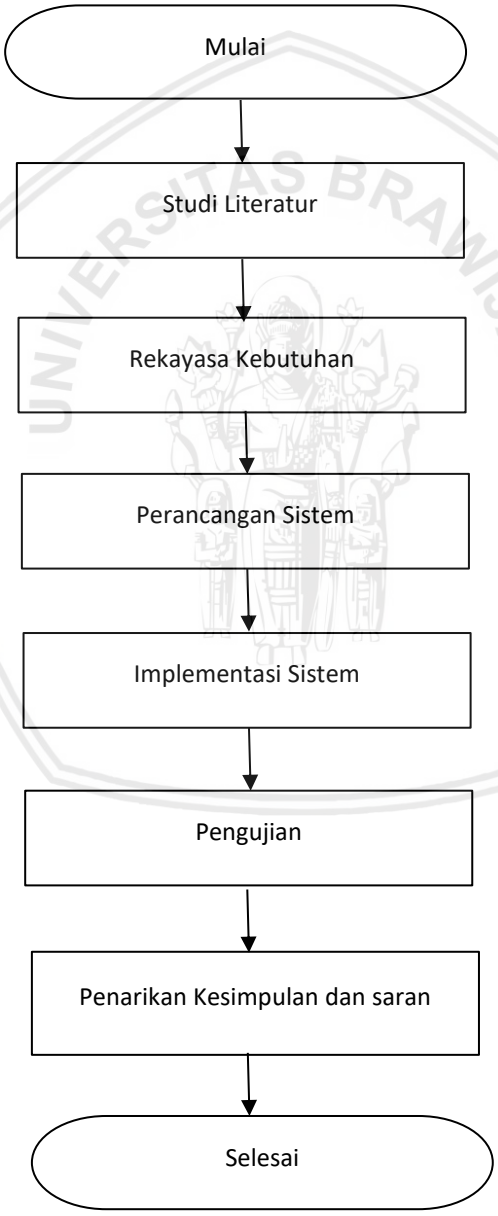
Sumber: Smith, Bruggen dan Tomassetti (2016)

Library JavaParser dan JavaSymbolSolver dapat diunduh secara gratis melalui GitHub dan untuk mempermudah dalam memahaminya terdapat buku yang berjudul "JavaParser: Visited Analyse, transform and generate your Java codebase" yang ditulis oleh Nicholas Smith, Danny van Bruggen dan Federico Tomassetti pada tahun 2016 yang dapat diunduh secara gratis maupun berbayar.

Pada penelitian ini JavaParser digunakan untuk mengekstraksi AST pada *source code* yang bertujuan untuk menelusuri dan menyimpan *attribute, method,* dan *inheritance* dari *source code* yang kemudian akan dhitung nilai kompleksitasnya dengan memberikan nilai bobot pemahaman sehingga kemudian akan didapatkan nilai *Cognitive Weighted Inherited Class Complexity*.

BAB 3 METODOLOGI

Bab Metodologi berisi penjelaskan tentang prosedur atau langkah kerja yang akan digunakan untuk menyelesaikan penelitian ini. Metodologi tersebut mengadopsi alur pada model waterfaII yang merupakan salah satu jenis *Software Development Life Cycle*. Pada penelitian ini akan dilakukan beberapa tahapan yaitu studi literatur, anaIisis kebutuhan, perancangan sistem, impIementasi sistem, pengujian, serta penarikan kesimpulan dan saran. Sebagaimana yang digambarkan pada diagram yang ditunjukkan pada Gambar 3.1.



Gambar 3.1 Diagram Metodologi Penelitian

3.1 Studi Literatur

Pada penelitian ini, studi literatur digunakan sebagai sumber acuan dalam penelitian dan pengembangan sistem, dengan tujuan untuk dapat mempelajari dan memahami secara mendalam tentang teori-teori dasar keilmuan yang akan menjadi objek penelitian dan membantu dalam melakukan penelitian. Teori-teori yang dibutuhkan untuk membantu pelaksanaan penelitian pembangunan Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan Metrik *Cognitive Weighted Inherited Class Complexity* (CWICC) tersebut dapat diperoleh dari berbagai macam sumber yaitu buku, jurnal, artikel, *e-book*, penelitian sebelumnya, situs internet dan literatur lain yang berkaitan. Teori pendukung yang berkaitan dengan penelitian skripsi ini meliputi:

1. Rekayasa Perangkat Lunak
2. *Software Development Life Cycle*
3. Model *Waterfall*
4. Pendekatan Berorientasi Objek
5. Pemodelan Berorientasi Objek
6. Hubungan pendekatan berorientasi objek dan kompleksitas
7. Pengujian Perangkat Lunak
8. *Cognitive Weighted Inherited Class Complexity Metric*
9. *Source code*
10. Bahasa Pemrograman Java
11. JavaFX
12. *Abstract Syntax Tree* dan JavaParser

3.2 Rekayasa Kebutuhan

Rekayasa kebutuhan dilakukan untuk mendapatkan seluruh kebutuhan yang diperlukan dalam pembangunan Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan Metrik *Cognitive Weighted Inherited Class Complexity* (CWICC) dan siapa saja yang akan terlibat dalam sistem ini. Proses rekayasa kebutuhan yang dilakukan pada penelitian ini meliputi:

1. Elisitasi Kebutuhan
   Pada proses ini, untuk mengumpulkan kebutuhan dalam pembangunan sistem pada penelitian ini dilakukan dengan cara menganalisis jurnal penelitian dari K. Maheswaran and A. Aloysius (2018) yang berjudul *Cognitive Weighted Inherited Class Complexity Metric.* Pada proses ini akan dihasilkan daftar kebutuhan yang akan terdapat dalam sistem.
2. Spesifikasi Kebutuhan
   Pada proses ini, akan dilakukan identifikasi aktor yang akan terlibat pada sistem dan juga kebutuhan yang telah didapatkan pada proses sebelumnya akan dispesifikasikan dan diklasifikasikan menjadi kebutuhan fungsional dan kebutuhan non-fungsional yang bertujuan untuk dapat mengetahui aktifitas yang dapat dilakukan pengguna maupun sistem beserta batasan-batasannya.

3. Manajemen Kebutuhan
   Pada proses ini bertujuan untuk melakukan kontrol terhadap kebutuhan yang sudah didefinisikan. Kebutuhan akan ditulis dengan memberikan kode unik yaitu kode CT-1-001 untuk kebutuhan fungsional dan CT-2-001 untuk kebutuhan non fungsional.
4. Pemodelan Kebutuhan
   Pada pemodelan kebutuhan menggunakan *Unified Modelling Language* (UML). Kebutuhan yang telah berhasil diidentifiaksi kemudian dimodelkan ke dalam bentuk *use case diagram*. Pada use case diagram terdapat gambaran kebutuhan perangkat lunak berdasarkan sudut pandang pengguna. Setiap use case tersebut akan dijelaskan secara tekstual kedalam bentuk *use case scenario*.

3.3 Perancangan Sistem

Setelah seluruh kebutuhan berhasil diidentifikasi pada tahap rekayasa kebutuhan maka dilakukan perancangan sistem. Pada tahap perancangan sistem ini nantinya akan dijadikan sebagai acuan dalam melakukan tahap implementasi sistem dan tahap pengujian. Jadi perancangan sistem merupakan kegiatan merubah kebutuhan menjadi abstraksi berupa representasi dari sistem yang akan dibangun berdasarkan kebutuhan. Tahapan perancangan sistem yang akan dikembangkan adalah sebagai berikut:

1. Perancangan Arsitektur
   Pada perancangan arsitektur terdapat diagram sequence diagram dan class diagram. *Sequence diagram* berfungsi untuk mengambarkan interaksi antar objek yang disusun dalam urutan waktu, interaksi tersebut berupa pertukaran pesan, pada penelitian ini akan dipaparkan tiga sequence diagram dari tiga fungsionalitas utama sistem. *Class diagram* berfungsi untuk mengambarkan kelas-kelas beserta penyusunnya yang terdapat dalam sistem.
2. Perancangan Komponen
   Pada perancangan komponen akan dituliskan tiga sampel *algoritme* utama yang akan menyusun sistem, *algoritme* tersebut digambarkan dalam bentuk *pseudocode*.
3. Perancangan Antarmuka
   Perancangan antarmuka digunakan sebagai rancangan *layout* dari media yang menghubungkan pengguna dan sistem untuk dapat saling berinteraksi.

3.4 Implementasi Sistem

Implementasi sistem adalah tahap pembangunan berdasarkan pada proses perancangan sistem yang telah dilakukan sebelumnya. Implementasi sistem pada penelitian ini dilakukan dengan menggunakan bahasa pemrograman Java . Tahap implementasi sistem tersebut meliputi:

1. Implementasi sistem dilakukan dengan mengacu pada spesifikasi sistem yang digunakan untuk membangun aplikasi ini.

2. Implementasi Kode Program
   Implementasi kode program dibangun berdasarkan perancangan komponen pada tahap perancangan yang memuat algoritme-algoritme utama dalam sistem. Kode program tersebut diimplementasikan dengan menggunakan Bahasa pemrograman Java d.
3. Implementasi Antarmuka
   Implentasi antarmuka dibangun berdasaarkan perancangan antarmuka pada tahap perancangan. Antarmuka tersebut diimplementasikan menggunakan JavaFX.

3.5 Pengujian Sistem

Pengujian sistem adalah tahapan menemukan kesalahan sebanyak-banyaknya dan memperbaikinya yang berfungsi untuk mengetahui apakah sistem yang dibangun telah memenuhi kebutuhan yang telah didefinisikan. Ada beberapa tahapan dalam strategi pengujian sesuai dengan yang dipaparkan Pressman (2010) yaitu sebagai berikut:

1. Pengujian Unit
   Pengujian unit dilakukan dengan melakukan pengujian pada sistem yang dapat berupa komponen, kelas, atau objek berdasarkan perancangan komponen. Metode yang digunakan adalah *white-box testing* dengan teknik pengujian *basis path testing*.
2. Pengujian Integrasi
   Pengujian integrasi akan menguji apakah setiap komponen atau unit yang berhubungan dapat berinteraksi dengan baik dan benar sesuai degan kebutuhan. Metode yang digunakan adalah *white-box testing* dengan teknik pengujian *basis path testing*.
3. Pengujian Validasi
   Pengujian validasi akan menguji seluruh kebutuhan yang telah didefinisikan yang terdiri dari kebutuhan fungsional dan non-fungsional. Pada kebutuhan fungsional akan menguji setiap jalur pada scenario. Pada kebutuhan non-fungsional akan dilakukan pengujian akurasi dan performa, pengujian akurasi bertujuan untuk memastikan bahwa hasil perhitungan yang dilakukan oleh sistem sesuai dengan hasil perhitungan secara manual. Sedangkan pengujian performa bertujuan untuk memastikan bahwa waktu yang dibutuhkan untuk melakukan perhitungan oleh sistem lebih cepat dibandingkan perhitungan manual. Metode yang digunakan adalah *black-box testing.*

3.6 Penarikan Kesimpulan dan Saran

Penarikan kesimpulan dilakukan setelah seluruh tahapan selesai dilakukan dan merupakan tahap terakhir dari penelitian ini. Kesimpulan diambil berdasarkan hasil analisis dan pengujian yang telah selesai dilakukan sehingga diperoleh inti dari keseluruhan proses penelitian. Penarikan kesimpulan bertujuan untuk menjawab permasalahan yang telah didefinisikan pada rumusan masalah. Sedangkan saran berupa masukan untuk memperbaiki kesalahan-kesalahan yang terjadi pada sistem maupun penelitian ini dan untuk pengembangan yang lebih lanjut.

BAB 4 REKAYASA KEBUTUHAN

Bab Rekayasa Kebutuhan berisi pembahasan tentang proses rekayasa kebutuhan yang dilakukan dalam penelitian pemb angunan Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan Metrik *Cognitive Weighted Inherited Class Complexity* (CWICC). Tahap rekayasa kebutuhan perangkat lunak dilakukan untuk mendefinisikan dan menggambarkan kebutuhan yang harus terdapat pada sistem agar dapat memenuhi kebutuhan pengguna. Pada rekayasa kebutuhan, pertama-tama akan dilakukab proses analsis kebutuhan yang selanjutnya akan dilakukan pemodelan kebutuhan. Proses analisis kebutuhan dilakukan untuk mengumpulkan kebutuhan apa saja yang harus disediakan oleh sistem, kemudian dilakukan spesifikasi kebutuhan untuk mengklasifikasikan kebutuhan fungsional dan non-fungsional. Pada identifikasi aktor akan menentukan siapa saja aktor yang terlibat dalam sistem beserta perannya. Sedangkan pada pemodelan kebutuhan, kebutuhan yang telah berhasil diidentifikasi akan dimodelkan menggunakan pemodelan *use case diagram* dan *use case scenario*.

4.1 Deskripsi Umum Sistem

Pada penelitian ini akan dibangun sebuah sistem perangkat lunak yang berjudul "Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan metrik *Cognitive Weighted Inherited Class Complexity* (CWICC)". Pada sistem ini akan memberikan informasi mengenai nilai kompleksitas sebuah perangkat lunak berdasarkan penggunaan teknik *inheritance* sebagai salah satu parameter perhitungan utamanya. Dimana dalam menghitung kompleksitas menggunakan metrik CWICC dilakukan dengan memberikan penilaian bobot kognitif berupa tingkat pemahaman terhadap bagian-bagian dari kode sumber atau struktur internal kelas yaitu penilaian bobot kognitif pada *attribute*, *method*, tingkat kelas turunan atau *inheritance*, serta *method* dan *attribute* turunan yang digunakan kembali. Pengguna diharapkan dapat memperoleh informasi mengenai nilai kompleksitas perangkat lunak yang dikembangkan sehingga dapat dijadikan bahan pertimbangan untuk melakukan perencanaan ulang kode sumber agar tahap *maintenance* perangkat lunak dapat dilakukan dengan baik. Pada tahap awal sistem mendapat masukan berupa folder proyek perangkat lunak yang akan diuji, kemudian sistem akan melakukan perhitungan menggunakan metrik CWICC dan menghasilkan keluaran berupa hasil perhitungan kompleksitasnya.

4.2 Analisis Kebutuhan

Kebutuhan pada sistem Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan metrik *Cognitive Weighted Inherited Class Complexity* (CWICC) diperoleh dari proses analisis jurnal penelitian dari Maheswaran and Aloysious (2018) yang berjudul *Cognitive Weighted Inherited Class Complexity Metric.* Pada penelitian ini perhitungan kompleksitas perangkat lunak dilakukan secara manual dengan mengidentifikasi bagian-bagian pada kode sumber yang dibutuhkan untuk melakukan perhitungan kompleksitas yang sesuai dengan

persamaan pada metrik CWICC. Mula-mula menentukan jumlah *Attribute Complexity* yang didapatkan dari menjumlahkan bobot pemahaman dari seluruh attribute yang terdapat di sebuah kode sumber. Kemudian menentukan *Method Complexity* yang didapatkan dari menjumlahkan bobot pemahaman dari seluruh *statement* dan *expression* yang terdapat dalam *body method*. Dilanjutkan menentukan *Improved Inherited Class Complexity* dengan menjumlahkan bobot pemahaman berdasarkan level *inheritance*, dan bobot pemahaman dari *attribute* dan *method* yang digunakan kembali. Tahap terakhir menjumlahkan nilai AC, MC dan IICC untuk mendapatkan nilai CWICC dari suatu kode program. Karena yang dihitung adalah kompleksitas perangkat lunak maka kode programnya kemungkinan lebih dari satu. Oleh karena itu untuk mendapatkan nilai kompleksitas perangkat lunak, maka menjumlahkan total dari setiap nilai CWICC.

4.3 Identifikasi Aktor

Identifikasi aktor merupakan tahap untuk menentukan aktor yang akan terlibat dalam sistem. Aktor dapat berupa orang (yang ditunjuk sesuai dengan perannya) atau sistem komputer yang lain yang akan menggunakan atau berinteraksi dengan sistem. Pengguna merupakan aktor pada sistem yang akan dikembangkan dalam penelitian ini. Tahap identifikasi aktor dilakukan dengan cara mendefinisikan setiap aktor beserta deskripsinya yang ditunjukkan pada Tabel 4.1.

Tabel 4.1 Identifikasi Aktor

No	Aktor	Deskripsi
1	Pengguna	Pengguna adalah satu-satunya aktor yang menggunakan dan berinteraksi dengan sistem serta bisa menjalankan seluruh fungsi yang terdapat pada sistem. Pengguna adalah seorang *developer* atau orang yang mengembangkan suatu perangkat lunak.

4.4 Daftar Kebutuhan

Daftar Kebutuhan adalah kumpulan kebutuhan-kebutuhan yang harus disediakan dan terdapat pada sistem. Terdapat dua jenis umum kebutuhan yaitu kebutuhan fungsional dan kebutuhan non-fungsional. Setiap kebutuhan tersebut akan diberikan kode yang unik yaitu CT-X-Y-Z, dimana CT merupakan singkatan dari *Cognitive Weighted Inherited Class Complexity* Tool, X menyatakan nomor definisi kebutuhan fungsional atau non fungsional, Y menyatakan nomor urutan kebutuhan dan Z menyatakan nomor spesifikasi kebutuhan.

4.4.1 Kebutuhan Fungsional

Kebutuhan fungsional merupakan kebutuhan-kebutuhan pokok yang diharapkan oleh pengguna untuk ada pada sistem. Dari hasil analisis kebutuhan yang telah dilakukan, diperoleh 4 kebutuhan fungsional dari sistem yang terdiri dari input berkas, ekstraksi *Abstract Syntax Tree*, hitung kompleksitas, dan *reset.* Daftar kebutuhan fungsional dan spesifikasinya ditunjukkan pada Tabel 4.2

Tabel 4.2 Daftar Kebutuhan Fungsional

No.	Kode Kebutuhan	Deskripsi	Nama Use Case
1	CT-1-001	Sistem harus dapat memasukkan berkas berupa *project folder* dari program yang akan diuji	*Input* Berkas
1.2	CT-1-001-01	Sistem harus dapat menampilkan isi dari berkas berupa *daftar file source code* bertipe Java	
2	CT-1-002	Sistem harus dapat mengekstraksi *Abstract Syntax Tree* dari masukan berkas yang terdapat di daftar file *source code*	Ekstraksi *Abstract Syntax Tree*
2.1	CT-1-002-01	Sistem harus dapat membedakan kelas *concrete*, *interface,* dan *abstract.*	
2.2	CT-1-002-02	Sistem harus dapat menyimpan data yang dibutuhkan untuk menghitung nilai metrik CWICC yaitu *attribute, method,* dan *inheritance* dari setiap kelas.	
3	CT-1-003	Sistem harus dapat menghitung dan menampilkan hasil perhitungan kompleksitas berdasarkan persamaan metrik CWICC	Hitung Kompleksitas
3.1	CT-1-003-01	Sistem harus dapat menampilkan file kelas bertipe Java yang dihitung	
3.2	CT-1-003-02	Sistem harus dapat menghitung nilai *Attribute Complexity* (AC) berdasarkan bobot kognitif pada setiap kelas dan menampilkan hasilnya	
3.3	CT-1-003-03	Sistem harus dapat menghitung nilai *Method Complexity* (MC) berdasarkan bobot kognitif pada setiap kelas dan menampilkan hasilnya	
3.4	CT-1-003-04	Sistem harus dapat menghitung nilai *Improved Inherited Class Complexity* (IICC) berdasarkan bobot kognitif pada setiap kelas dan menampilkan hasilnya	
3.5	CT-1-003-05	Sistem harus dapat menampilkan hasil perhitungan CWICC pada setiap kelas	
3.6	CT-1-003-06	Sistem harus dapat menampilkan total file yang diuji, waktu eksekusi, dan total nilai CWICC	
4	CT-1-004	Sistem harus dapat menghapus seluruh data yang telah disimpan dan perhitungan yang telah di proses	*Reset*

4.4.2 Kebutuhan Non-Fungsional

Kebutuhan non-fungsional berisi kumpulan kebutuhan yang dapat dijadikan acuan untuk menentukan kualitas sistem secara keseluruhan dalam bentuk batasan layanan yang disediakan sistem. Batasan ini bisa berbentuk batasan waktu, batasan pengembangan dan batasan mengenai standard sistem. Daftar kebutuhan non-fungsional ditunjukkan pada Tabel 4.3.

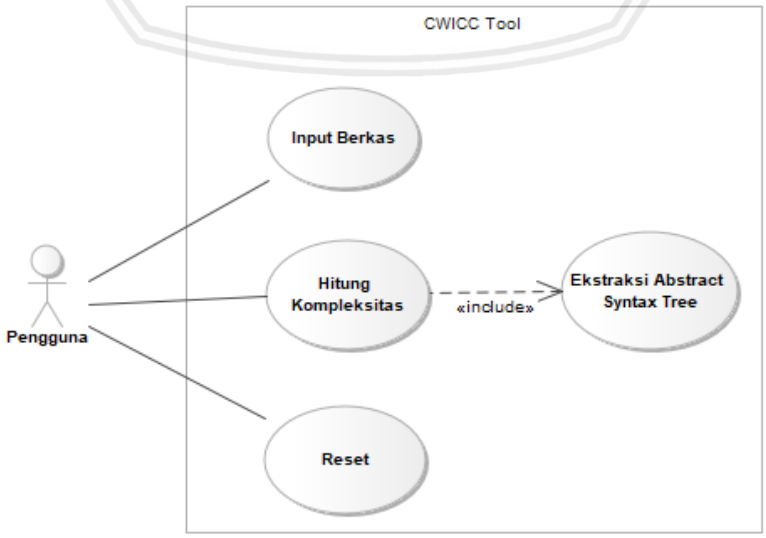
Tabel 4.3 Daftar Kebutuhan Non-Fungsional

No	Kode Kebutuhan	Jenis Kebutuhan	Deskripsi
1.	CT-2-001	*Accuration*	Sistem harus dapat melakukan perhitungan kompleksitas dengan tingkat akurasi minimal 90% untuk setiap perangkat lunak yang diuji
2.	CT-2-002	*Performance*	Sistem harus dapat melakukan perhitungan kompleksitas dengan waktu rata-rata 356 kali lebih cepat dibandingan perhitungan kompleksitas yang dilakukan secara manual.

4.5 Pemodelan Kebutuhan

4.5.1 Use Case Diagram

Pemodelan *use case diagram* digunakan untuk membuat gambaran sistem yang bertujuan menyatakan visualisasi interaksi yang terjadi antara aktor dan sistem. Aktor merupakan pengguna yang didapatkan dari tahap identifikasi, sebagai aktor tunggal sistem. Pengguna akan berinteraksi dengan 3 *base use case* yaitu input berkas, hitung kompleksitas, dan reset. Pada *base use case* hitung kompleksitas terdapat relasi *include* dengan *supplier use case* ekstraksi *abstract syntax tree*. *Use case diagram* sistem yang digunakan pada penelitian ini akan ditunjukkan pada Gambar 4.1.



Gambar 4.1 *Use Case Diagram* Sistem

4.5.2 Use Case Scenario

Use case scenario adalah uraian secara tekstual dari skenario yang terjadi ketika aktor dan sistem saling berinteraksi, setiap skenario tersebut akan menjelaskan urutan langkah yang harus dilakukan aktor ketika berinterakasi dengan sistem baik urutan langkah yang berhasil maupun yang gagal. Setiap *use case* pada *use case diagram* akan memiliki satu *use case scenario*. Pada *use case scenario* terdiri dari tujuh bagian. Tujuh bagian tersebut adalah kode kebutuhan, tujuan, aktor, kondisi awal, jalur utama, jalur alternatif, dan kondisi akhir. *Use case scenario* dari sistem ditunjukkan pada Tabel 4.4 sampai dengan Tabel 4.6.

4.4.2.1 Use Case Scenario Input Berkas

Use case scenario input berkas merupakan penjelasan lebih rinci dari *use case input* berkas. *Use case scenario* input berkas memiliki dua jalur alternatif. Tabel 4.4 menunjukkan *use case scenario* untuk input berkas.

Tabel 4.4 *Use Case Scenario Input* Berkas

Kode Kebutuhan	CT-1-001
Tujuan	Memasukkan berkas berupa *project folder* yang berisi kode sumber bertipe Java ke dalam sistem
Aktor	Pengguna
Kondisi Awal	Sistem menampilkan halaman utama sistem
Jalur Utama	1. Aktor menekan tombol Here! 2. Sistem menampilkan *window browser file* 3. Aktor memilih *project folder* yang diinginkan 4. Aktor menekan tombol Open 5. Sistem mengambil *file* bertipe Java dari *folder* dan *sub folder*
Jalur Alternatif	1. Jika aktor menekan tombol Cancel maka berkas tidak akan dimasukkan ke dalam sistem 2. Jika pada folder yang dimasukkan tidak ditemukan *file* dengan bertipe Java atau *folder* kosong maka akan menampilkan pesan "*File not found! Folder must contain Java files*"
Kondisi Akhir	Sistem menampilkan daftar *file* kode sumber bertipe Java dari *project folder* yang dimasukkan dalam bentuk tabel.

4.4.2.2 Use Case Scenario Ekstraksi Abstract Syntax Tree

Pada Tabel 4.5 menunjukkan *use case scenario* ekstraksi *Abstract Syntax Tree* yang merupakan penjelasan lebih rinci dari *use case* ekstraksi *Abstract Syntax Tree*. Pada *use case scenario* ekstraksi *Abstract Syntax Tree* tersebut memiliki sebuah jalur alternatif.

Tabel 4.5 Use Case Scenario Ekstraksi *Abstract Syntax Tree*

Kode Kebutuhan	CT-1-002
Tujuan	Mengekstraksi *Abstract Syntax Tree* sehingga mendapatkan *attribute*, *method,* dan *inheritance* dari setiap kode sumber.
Aktor	Pengguna
Kondisi Awal	Sistem menampilkan halaman yang berisi daftar *file* kode sumber yang telah dimasukkan
Jalur Utama	1. Aktor menekan tombol Extraction File! 2. Sistem melakukan ekstraksi Abstract Syntax Tree dari kode sumber yang terdapat di daftar tabel *file*
Jalur Alternatif	1. Jika *file* berupa kelas *interface* maka sistem tidak akan melakukan ekstraksi Abstract Syntax Tree Jika aktor menekan tombol Back maka sistem akan menghapus daftar file yang telah dimasukkan, dan kembali ke halaman utama
Kondisi Akhir	Sistem menampilkan pesan "Extraction is complete" dan menampilkan tombol Calculate! serta menyimpan data hasil ekstraksi yang dibutuhkan untuk perhitungan yaitu *attribute, method,* dan *inheritance*

4.4.2.3 Use Case Scenario Hitung Kompleksitas

Use case scenario hitung kompleksitas adalah penjelasan lebih rinci dari *use case* hitung kompleksitas. Tabel 4.6 menunjukkan *use case scenario* untuk menghitung kompleksitas menggunakan metrik *CWICC*. *Use case scenario* tersebut tidak memiliki jalur alternatif.

Tabel 4.6 Use Case Scenario Hitung Kompleksitas

Kode Kebutuhan	CT-1-003
Tujuan	Menghitung kompleksitas perangkat lunak menggunakan metrik CWICC
Aktor	Pengguna
Kondisi Awal	Sistem menampilkan daftar *file* bertipe Java dari berkas yang dimasukkan yang telah berhasil di ekstraksi menjadi *Abstract Syntax Tree*
Jalur Utama	1. Aktor menekan tombol Calculate! 2. Sistem melakukan perhitungan kompleksitas perangkat lunak dengan menggunakan metrik CWICC berdasarkan data yang telah didapatkan dari proses ekstraksi *Abstract Syntax Tree*.
Jalur Alternatif	-
Kondisi Akhir	Sistem menampilkan pesan "Calculation is complete" dan menyimpan hasil perhitungan kompleksitas serta menampilkan detail perhitungan dalam bentuk tabel

4.4.2.4 Use Case Scenario Reset

Use case scenario reset adalah penjelasan lebih rinci dari *use case reset*. Tabel 4.7 menunjukkan *use case scenario* untuk menghapus log aktivitas dari input berkas, ekstraksi *abstract syntax tree*, hingga perhitungan yang telah disimpan. *Use case scenario* tersebut tidak memiliki jalur alternatif.

Tabel 4.7 Use Case Scenario Reset

Kode Kebutuhan	CT-1-004
Tujuan	Menghapus seluruh data yang telah disimpan yaitu data daftar file, data ekstraksi *Abstract Syntax Tree* dan data perhitungan kompleksitas
Aktor	Pengguna
Kondisi Awal	Sistem menampilkan halaman hasil perhitungan kompleksitas
Jalur Utama	1. Pengguna menekan tombol Reset 2. Sistem menghapus seluruh penyimpanan data hasil ekstraksi dan perhitungan kemudian menampilkan halaman utama yaitu halaman input berkas
Jalur Alternatif	-
Kondisi Akhir	Sistem menghapus seluruh data yang telah disimpan dan kembali ke halaman utama

BAB 5 PERANCANGAN DAN IMPLEMENTASI

Bab Perancangan dan Implementasi akan menjelaskan tentang perancangan dan implementasi sistem yang dilakukan dalam penelitian pembangunan Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan Metrik *Cognitive Weighted Inherited Class Complexity* (CWICC). Pada penelitian ini penulis menggunakan pendekatan berorientasi objek, oleh karena itu pada tahap perancangan akan menggunakan metode *Object Oriented Design* (OOD) dan pada tahap implementasi akan menggunakan *Object Oriented Programming* (OOP).

Pada bab ini terdapat dua sub bab utama, yang berupa Perancangan Sistem dan Implementasi Sistem. Pada sub bab Perancangan Sistem berisi perancangan arsitektur, perancangan komponen dan perancangan antarmuka. Sedangkan pada sub bab Implementasi Sistem berisi spesifikasi sistem, implementasi kode program, dan implementasi antarmuka.

5.1 Perancangan Sistem

Ketika tahap rekayasa kebutuhan telah selesai dilakukan, maka dilanjutkan ke tahap berikutnya yaitu perancangan, yang dilakukan menggunakan pendekatan berorientasi objek yaitu *Object Oriented Design* (OOD). Pada penelitian ini perancangan yang digunakan yaitu perancangan arsitektur yang membuat *sequence diagram* dan *class diagram*, perancangan komponen, dan perancangan antarmuka dari sistem yang akan dibangun.

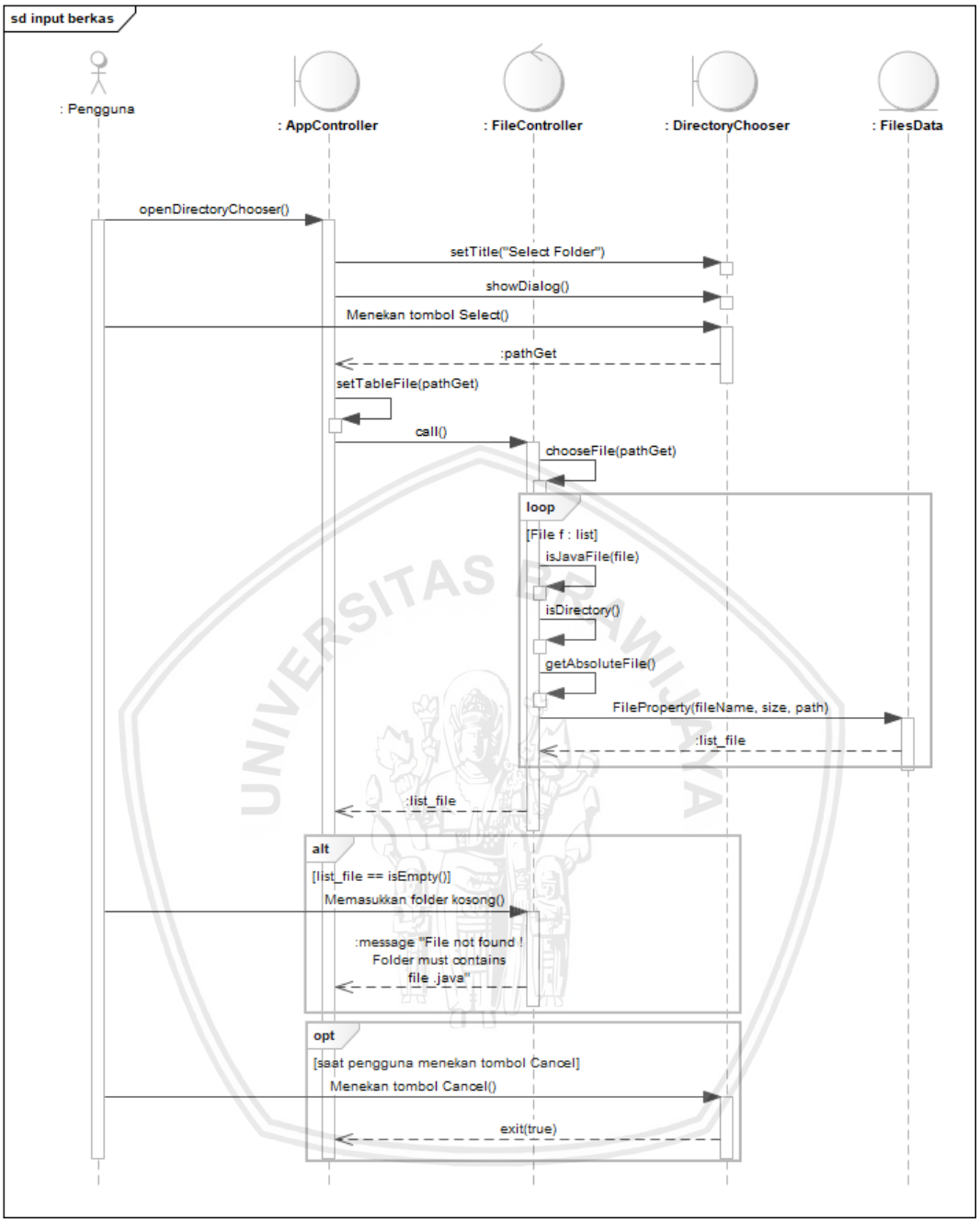
5.1.1 Perancangan Arsitektur

Sub bab Perancangan Arsitektur ini berisi penjelasan secara rinci tentang *sequence diagram* dan *class diagram* pada sistem yang akan dibangun pada penelitian ini. *Sequence diagram* akan menjelaskan alur interaksi berupa pertukaran pesan antar objek pada sistem dalam urutan waktu. Sedangkan pada *class diagram* akan menjelaskan struktur objek yaitu kelas dengan *attribute, method,* dan *modifier*, serta relasi antar kelas yang menyusun sistem. Pada penelitian ini akan menjelaskan tiga *sequence diagram* yang mewakili fitur yang terdapat pada sistem yaitu input berkas, ekstraksi *Abstract Synatx Tree,* dan hitung kompleksitas.

5.1.1.1 Sequence Diagram

a. Sequence Diagram Input Berkas

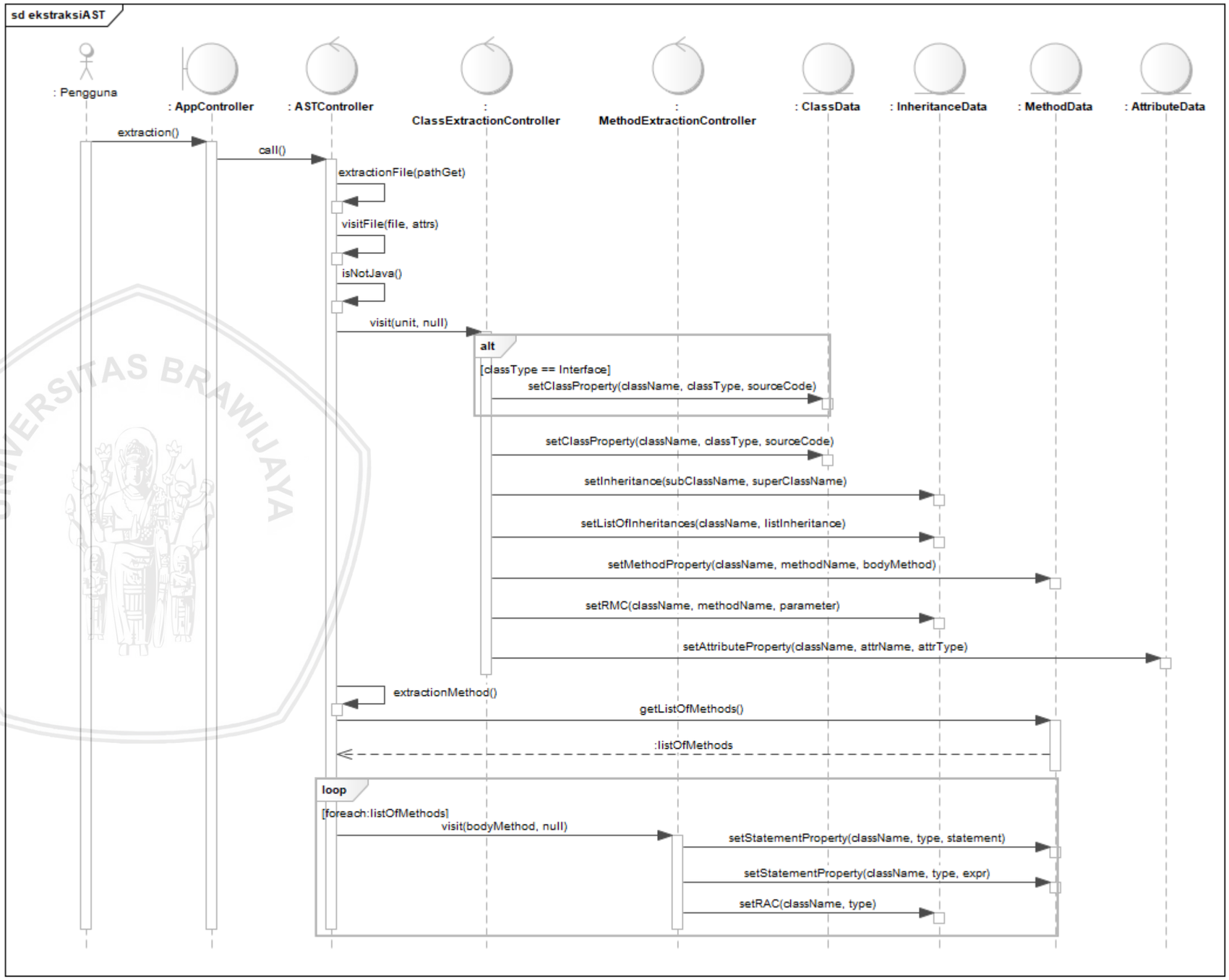
Sequence diaram dari fungsionalitas input berkas ditunjukkan pada Gambar 5.1. Aktor pada *sequence diagram* input berkas adalah pengguna yang merupakan satu-satunya aktor di sistem ini. Pada *sequence diagram* input berkas terdapat dua *boundary* yaitu AppController dan DirectoryChooser berupa *window browse file.* Terdapat satu *controller* yaitu FileController dan satu entitas yaitu FilesData.



Gambar 5.1 Sequence Diagram Input Berkas

b. Sequence Diagram Ekstraksi Abstract Syntax Tree

Sequence diaram dari fungsionalitas ekstraksi *Abstract Syntax Tree* ditunjukkan pada Gambar 5.2. Aktor di sequence diagram ekstraksi *Abstract Syntax Tree* adalah pengguna. Terdapat satu boundary yaitu AppController. Terdapat tiga *controller* yaitu ASTController, ClassExtractionController dan MethodExtractionController serta terdapat empat entitas yaitu ClassData, InheritanceData, MethodData, dan AttributeData.



Gambar 5.2 Sequence Diagram Ekstraksi *Abstract Syntax Tree*

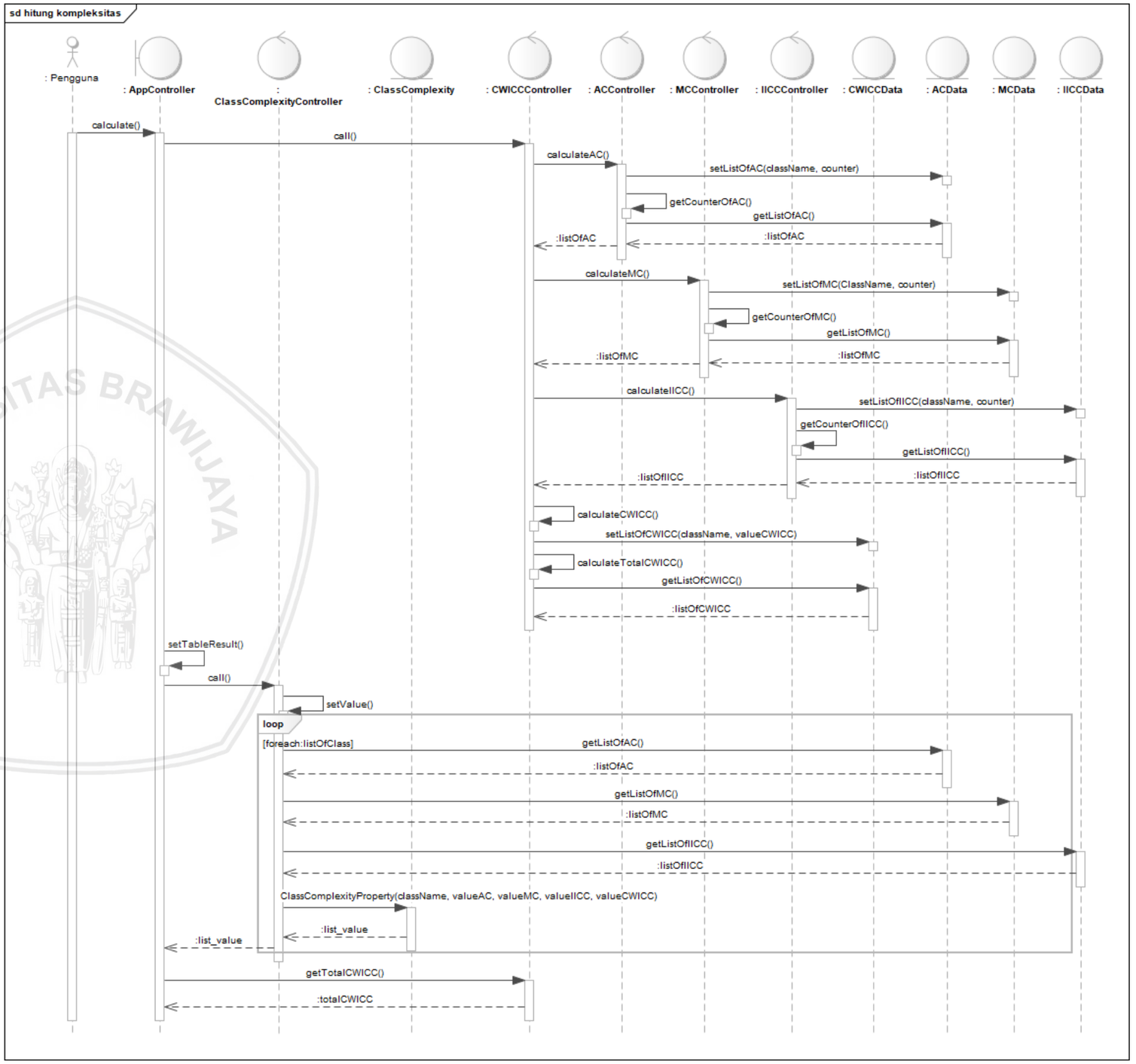
repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

c. Sequence Diagram Hitung Kompleksitas

Sequence diaram dari fungsionalitas hitung kompleksitas ditunjukkan pada Gambar 5.3. Aktor di sequence diagram hitung kompleksitas adalah pengguna. Terdapat satu boundary yaitu AppController. Kemudian terdapat lima controller yaitu ClassComplexityController, CWICCController, ACController, MCController, dan IICCController. Serta terdapat lima entitas yaitu ClassComplexity, CWICCData, ACData, MCData, dan IICCData.

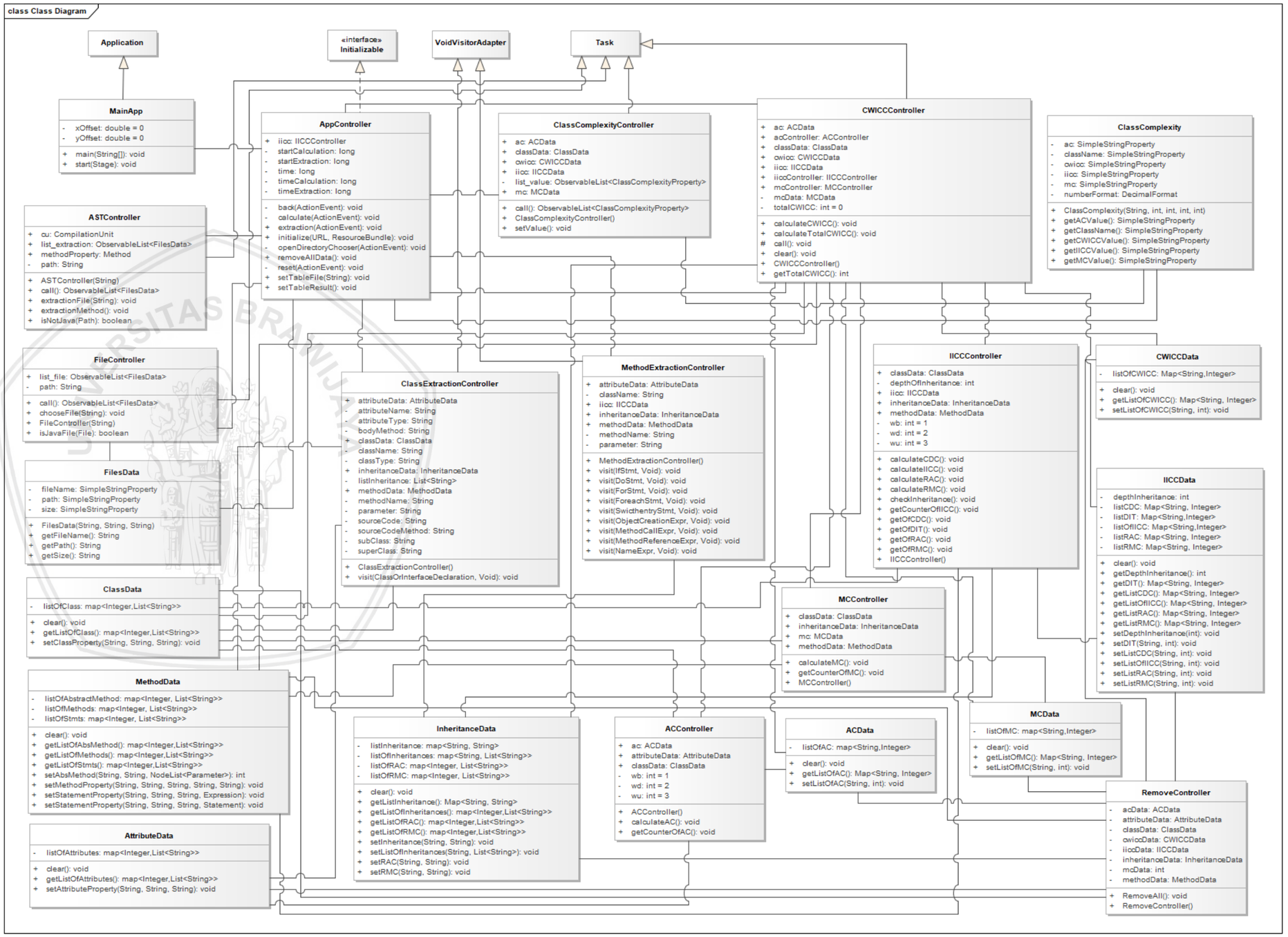
5.1.1.2 Class Diagram

Class diagram digunakan sebagai representasi dari struktur objek yaitu kelas yang mencakup keseluruhan *atribut* dan *method* yang menyusunnya beserta relasinya. Pada *class diagram* dalam penelitian ini terdapat kelas *view* utama yaitu MainApp serta kelas AppController yang mengatur tampilan sistem dimana pada penelitian ini berhubungan dengan file FXML. Terdapat kelas *controller* yang berfungsi menjalankan proses dari alur logika yang diinginkan ada pada sistem. Terdiri dari kelas FileController, ASTController, ClassExtractionController, MethodExtractionController, ACController, removeController, MCController, CWICCController, ClassComplexityController, dan IICCController. Kemudian terdapat kelas *model* yang terdiri dari kelas FilesData, ClassData, InheritanceData, MethodData, dan AttributeData untuk menyimpan hasil ekstraksi file yaitu berupa *property* dari struktur internal kelas yang berfungsi untuk perhitungan kompleksitas. Sedangkan kelas ACData, MCData, IICCData, CWICCData serta kelas ClassComplexity untuk menyimpan hasil perhitungan dari kompleksitas kelas pada perangkat lunak. *Class diagram* pada penelitian ini akan ditunjukkan pada Gambar 5.4.



Gambar 5.3 Seqence Diagram Hitung Kompleksitas

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA



Gambar 5.4 Class Diagram

5.1.2 Perancangan Komponen

Pada perancangan komponen berisi penjelasan tentang *algoritme* dari beberapa *method* yang akan digunakan sebagai acuan implementasi yaitu pada implementasi kode program, perancangan tersebut berupa *pseudocode*. Pada penelitian ini akan dibuat *pseudocode* dari tiga *method* yang akan digunakan yaitu *method* chooseFile, visit, dan calculateAC.

5.1.2.1 Perancangan Komponen Method chooseFile

Pada *Algoritme* 1 memaparkan *pseudocode method* chooseFile yang digunakan untuk memasukkan file yang bertipe Java dari folder yang telah dimasukkan ke dalam kelas *model* yang selanjutnya akan ditampilkan pada *table view*. Terdapat perulangan pada setiap File f di sebuah list yang berisi Files, maka akan dilakukan seleksi kondisi jika f adalah direktori maka akan menampilkan *path* direktori pada *console* dan memanggil *recursive method* chooseFile. Jika f bukan direktori maka akan menampilkan *path file* pada *console* dan menambahkan data pada list_file berupa nama kelas file tersebut dan ukurannya yang disimpan di kelas FileProperty.

Algoritme 1: pseudocode method chooseFile	
1	Deklarasi dan instansiasi root
2	Deklrasi dan inisialisasi array list
3	For setiap f pada list melakukan
4	If (f adalah directory)
5	Maka menampilkan path directory pada console
6	Memanggil method recursive chooseFile
7	Else
8	Maka menampilkan path file pada console
9	Maka menambahkan data pada list_file
10	End if
11	End For

5.1.2.2 Perancangan Komponen Method visit

Pada *Algortime* 2 memaparkan kode sumber *method visit* yang digunakan untuk melakukan ekstraksi *property* pada sebuah kelas. Terdapat perulangan dari kelas yang memiliki tipe *extends* untuk menyimpan *inheritance property*, selanjutnya dilakukan seleksi kondisi jika kelas tersebut kelas *interface* maka hanya akan menyimpan *class property*. Jika merupakan kelas *abstract* atau kelas *concrete* selain menyimpan *class property* juga dilakukan perulangan. Di perulangan tersebut akan terjadi seleksi kondisi jika node adalah deklarasi *field* maka akan menyimpan *attribute property*, berupa nama *attribute* dan tipe data *attribute*. Jika node adalah deklarasi *method* maka akan melakukan seleksi kondisi jika *method* adalah *abstract* maka menyimpan *abstract method property* sedangkan jika tidak maka melakukan seleksi kondisi jika *annotation* atau *type method* adalah *override* maka menyimpan *method property* dan RMC atau *method* turunan yang digunakan, jika tidak maka *type method* adalah *common* lalu menyimpan *method property*, yaitu berupa nama *method*, *type method*, body *method*, *parameter*, dan *source code method*.

Algoritme 2: Pseudocode method visit

```
1   Inisialisasi variable className
2   Inisialisasi variable sourceCode
3   foreach extendType melakukan
4      Inisialisasi superClass
5      Menyimpan className, superClass
6      Menambahkan superClass pada listInheritance
7      Foreach listOfInheritances
8      Mengambil subClass
9      Mengambil supClass
10     Inisialisasi depthOfInheritance = 0
11     If superClass = subclass
12        If listOfInheritances dari subClass
13           Menambahkan supClass pada listInheritance
14        End if
15     End if
16     End foreach
17     Menambahkan listOfInheritances
18  End foreach
19  if kelas interface
20      Maka inisialisasi classType = "Interface"
21      Menyimpan className, classType dan sourceCode
22  else
23    if kelas abstract
24      Maka inisialisasi classType = "Abstract"
25      Menyimpan className, classType, dan sourceCode
26    else
27      Maka inisialisasi classType = "Concrete"
28      Menyimpan className, classType, dan sourceCode
29    End if
30     Mengambil semua node dari n
31     Foreach node melakukan
32       If node adalah FieldDeclaration
33         Inisialisasi attributeType
34         Inisialisasi attributeName
35         Menyimpan className, attributeType, dan attributeName
36       Else if node adalah MethodDeclaration
37         If method abstract
38           Menyimpan className, methodName, parameter
39         Else
40           Inisialisasi methodName
41           Inisialisasi bodyMethod
42           Inisialisasi parameter
43           Inisialisasi sourceCodeMethod
44           Inisialisasi annotation
45           If annotation = "Optional[@Override]"
46             Inisialisasi typeMethod = "override"
47             Menyimpan   className,   methodName,   typeMethod,
48  bodyMethod, dan methodSourceCode
49             Menyimpan RMC className, methodName, parameter
50           Else
51             Inisialisasi typeMethod = "common"
52             Menyimpan   className,   methodName,   typeMethod,
53  bodyMethod, dan methodSourceCode
54           End if
55         End if
56       End if
57     End foreach
58  End if
```

5.1.2.3 Perancangan Komponen Method calculate

Pada *Algortime* 3 memaparkan *pseudocode method* calculateAC yang digunakan untuk melakukan perhitungan nilai *attribute complexity.* Pada algoritme 3 akan dilakukan perulangan listOfClass lalu mengambil data className dan inisialisasi nilai counterAC, counterPDT, counterDDT, dan counterUDDT. Kemudian dilakukan perulangan sebanyak listOfAttributes lalu mengambil data attrClassName dan *attribute*. Dalam perulangan dilakukan seleksi kondisi jika attrClassName sama dengan className maka dilakukan seleksi kondisi lagi jika *attribute* merupakan tipe PDT maka counterPDT bertambah, jika *attribute* merupakan tipe DDT maka counterDDT bertambah, dan jika *attribute* merupakan tipe UDDT maka counter UDDT bertambah. Selanjutnya dilakukan perhitungan counterAC dengan melakukan perkalian nilai counterPDT, counter DDT, dan counter UDDT dengan bobot pemahaman dan menjumlahkan hasilnya yang kemudian di simpan di listOfAC.

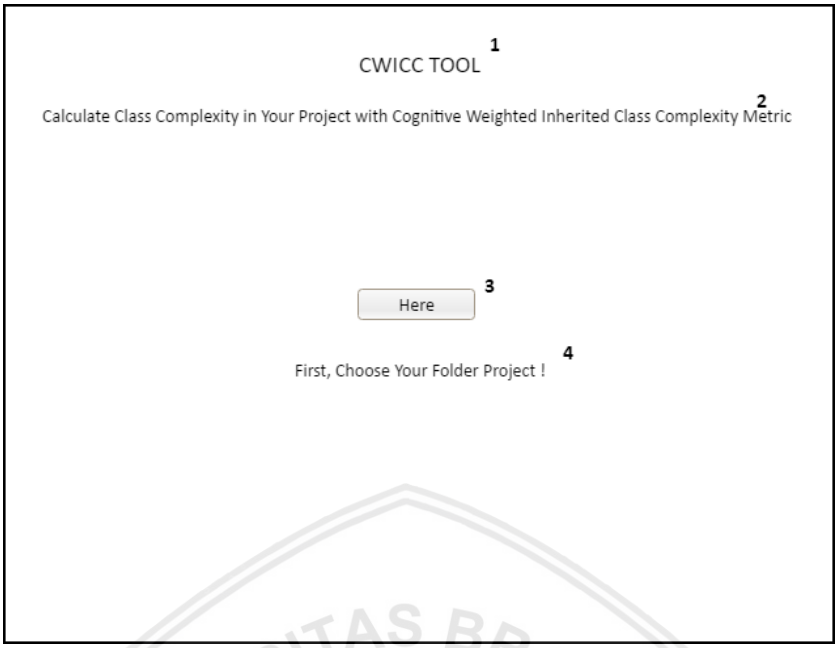
```
Algoritme 3: Pseudocode method calculateAC
1   Foreach listOfClass melakukan
2      Mengambil className
3      Inisialisasi counterAC = 0
4      Inisialisasi counterPDT = 0
5      Inisialisasi counterDDT = 0
6      Inisialisasi counterUDDT = 0
7      Foreach listOfAttributes melakukan
8         Mengambil attrClassName
9         Mengambil attribute
10        If attrClassName = className
11           If attribute PDT
12              add counterPDT
13           Else if attribute DDT
14              add counterDDT
15           Else if attribute UDDT
16              add CounterUDDT
17           End if
18           counterAC = counterPDT * wb + counterUDT * wd +
19  counterUDDT * wu;
20        Menyimpan listOfAC
21        End if
22     End for
23  End for
```

5.1.1 Perancangan Antarmuka

Pada perancangan antarmuka berisi penjelasan tentang pembuatan *mock-up layout* dari halaman yang terdapat pada sistem. Pada tahap ini akan menghasilkan perancangan antarmuka yang akan digunakan sebagai acuan untuk melakukan implementasi antarmuka pada tahap implementasi sistem.

5.1.3.1 Antarmuka *Input* Berkas

Perancangan antarmuka halaman *input* berkas ditunjukkan pada Gambar 5.5. Halaman ini merupakan halaman pertama dari sistem. Berfungsi untuk memasukkan berkas berupa *folder project* yang akan diuji. Pilihan berkas akan muncul *pada window browse file* yang akan ditampilkan setelah menekan tombol Here!.



Gambar 5.5 Perancangan Antarmuka Input Berkas

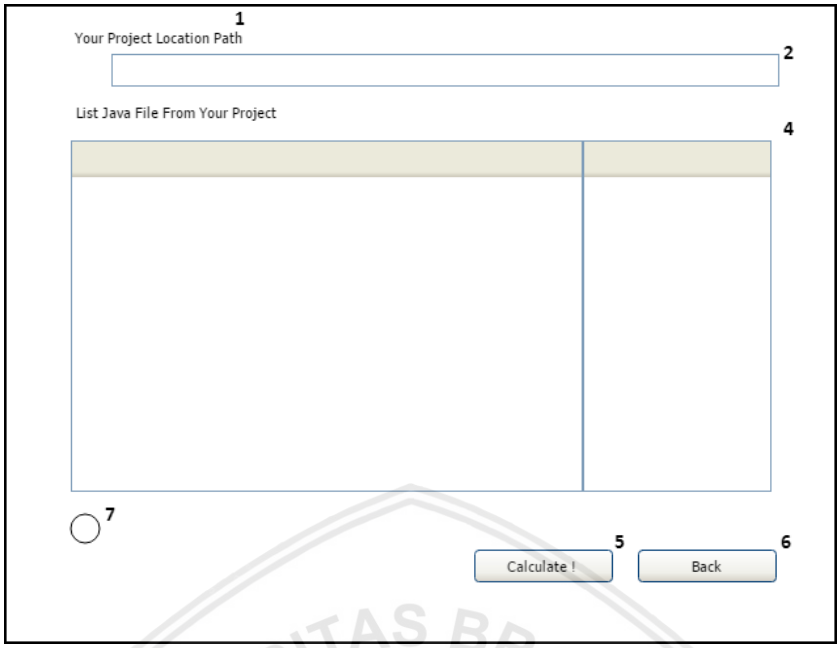
Pada perancangan antarmuka halaman *input* berkas terdapat 4 komponen yang akan dijelaskan dengan lebih rinci pada Tabel 5.1.

Tabel 5.1 Perancangan antarmuka input berkas

No	Nama Objek	Tipe	Fungsi
1	Teks “CWICC TOOL”	Label	Label teks judul sistem
2	Teks	Label	Label teks deskripsi sistem
3	Tombol “Here !”	Button	Tombol untuk menampilkan Window Browse File
4	Teks “First, Choose Your Folder Project !”	Label	Label teks perintah untuk memasukkan folder

5.1.3.2 Antarmuka Daftar file

Perancangan antarmuka pada halaman daftar file ditunjukkan pada Gambar 5.6. Halaman daftar file akan ditampilkan setelah pengguna memasukkan berkas berupa *folder project* perangkat lunak yang akan di uji. Pada halaman ini daftar file akan ditampilkan ke dalam bentuk tabel. Selain itu juga terdapat tombol Extraction File! Untuk melakukan ekstraksi file, setelah ekstraksi file berhasil dilakukan maka tombol Calculate akan ditampilkan.



Gambar 5.6 Perancangan Antarmuka Daftar File

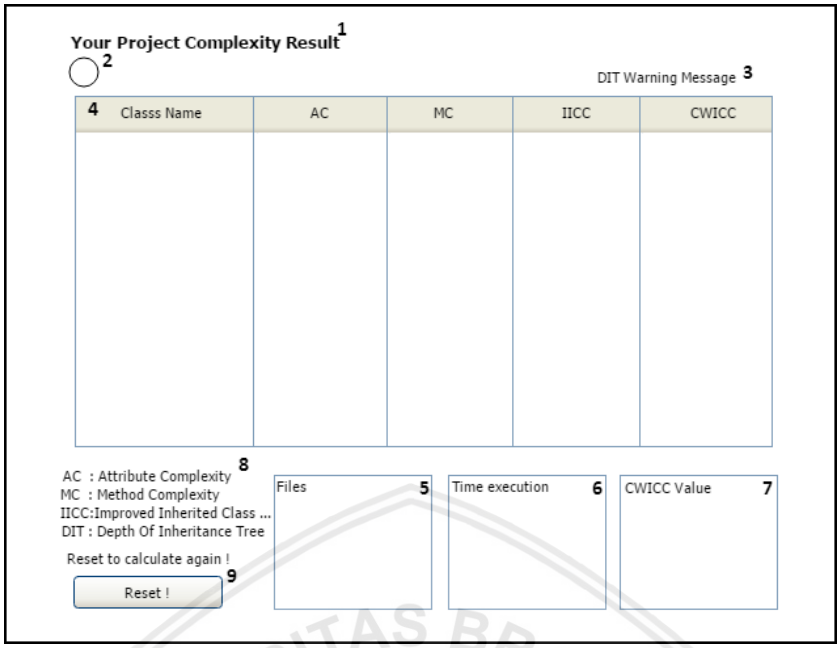
Pada perancangan antarmuka halaman daftar file terdapat 7 komponen yang akan dijelaskan dengan lebih rinci pada Tabel 5.2.

Tabel 5.2 Perancangan antarmuka daftar file

No	Nama Objek	Tipe	Fungsi
1	Teks "Your Project Location…"	Label	Label judul text box lokasi berkas yang dimasukkan
2	Text Box	Text Box	Wadah alamat lokasi berkas yang dimasukkan
3	Teks "Your Java File Project…"	Label	Label judul list view isi dari berkas yang dimasukkan
4	List File	Table View	Tabel untuk menampilkan list file dari berkas yang dimasukkan
5	Tombol "Calculate !"	Button	Tombol untuk melakukan ekstraksi *Abstract Syntax Tree* dan perhitungan kompleksitas
6	Tombol "Back"	Button	Tombol untuk kembali ke menu utama
7	Status Bar	Progress Indicator	Icon yang menampilkan progress pada proses yang dijalankan

5.1.3.3 Antarmuka Hasil Perhitungan

Perancangan antarmuka halaman hasil perhitungan ditunjukkan pada Gambar 5.7. Halaman ini akan ditampilkan saat pengguna telah memasukkan berkas berupa *folder project* yang akan di uji dan telah menekan tombol Calculate.



Gambar 5.7 Perancangan Antarmuka Hasil Perhitungan

Pada perancangan antarmuka halaman hasil perhitungan terdapat 8 komponen yang akan dijelaskan dengan lebih rinci pada Tabel 5.3.

Tabel 5.3 Perancangan antarmuka hasil perhitungan

No	Nama Objek	Tipe	Fungsi
1	Teks "Your Project Complexity Result"	Label	Label judul tabel hasil perhitungan
2	Status Bar	Progress Indicator	Untuk menampilkan progress yang terjadi selama proses yang berjalankan
3	Teks DIT *warning message*	Label	Label untuk menampilkan pesan tingkat DIT dan status kompleksitasnya.
4	Tabel hasil	Table View	Tabel yang berisi deskripsi hasil perhitungan setiap kelas bertipe Java
5	Text Box Files	Text Box	Wadah untuk menampilkan jumlah file yang dihitung
6	Text Box Time Execution	Text Box	Wadah untuk menampilkan waktu yang dibutuhkan untuk melakukan perhitungan
7	Text Box CWICC Value	Text Box	Wadah untuk menampilkan hasil kompleksitas berupa nilai CWICC dari project
8	Teks "AC: Attribute Complexity …"	Label	Label keterangan yang berisi penjelasan akronim yang terdapat di tabel hasil
9	Tombol "Reset"	Button	Tombol untuk kembali ke menu utama

5.2 Implementasi Sistem

Ketika tahap parancangan sistem telah selesai dilakukan maka dilanjutkan ke tahap selanjutnya, yaitu tahap implementasi sistem. Pada tahap implementasi sistem terdapat tiga sub bab yaitu spesifikasi sistem, implementasi kode program, dan implementasi antarmuka dari sistem yang akan dibangun.

5.2.1 Spesifikasi Sistem

Pada sub bab spesifikasi sistem menjelaskan tentang spesifikasi dari perangkat keras (*hardware*) dan perangkat lunak (*software*) yang digunakan dalam pembangunan Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan Metrik CWICC.

5.2.2.1 Spesifikasi Perangkat Keras

Pada penelitian ini spesifikasi dari perangkat keras yang digunakan akan ditunjukkan pada Tabel 5.4. Spesifikasi tersebut berupa nama *system model*, besaran *processor, memory, hardisk*, serta *screen resolusion*.

Tabel 5.4 Spesifikasi perangkat keras

Komponen	Spesifikasi
System Model	LENOVO ideapad 110
Processor	Intel® Core™ Core i3-6100U CPU @2.30GHz
Memory	RAM 4,00 GB (3.88 GB usable)
Hardisk	1TB
Screen Resolusion	1366x768

5.2.2.2 Spesifikasi Perangkat Lunak

Pada penelitian ini spesifikasi dari perangkat lunak yang digunakan akan ditunjukkan pada Tabel 5.5. Spesifikasi tersebut berupa sistem operasi, IDE, bahasa pemrograman, serta perangkat lain yang digunakan sebagai penunjang.

Tabel 5.5 Spesifikasi perangkat lunak

Komponen	Spesifikasi
Sistem Operasi	Windows 10 Pro 64-bit
IDE untuk pemrograman	NetBeans IDE 8.2
Bahasa Pemrograman	Java
Framework Front-end	JavaFX
Library *Front-end*	JFoenix, AnimateFX
Editor Dokumentasi	Microsoft Word 2016
Editor Diagram Perancangan	Enterp rise Architect

5.2.2 Implementasi Kode Program

Implementasi kode program adaIah proses menterjemahkan *pseudocode* yang merupakan hasiI dari perancangan komponen yang telah diIakukan pada tahap perancangan sistem menggunakan bahasa pemrograman yang spesifik. Pada penelitian ini penuIis menggunakan bahasa pemrograman Java.

5.2.2.1 Implementasi Kode Program Method chooseFile

Nama Kelas: FileController.java

Nama Operasi: chooseFile

Kode sumber method chooseFile

```
1  File root = new File(path);
2  File[] list = root.listFiles((File file) -> isJavaFile(file)
3  || file.isDirectory());
4      for (File f : list) {
5          if (f.isDirectory()) {
6              System.out.println("Dir:" + f.getAbsoluteFile());
7              chooseFile(f.getAbsolutePath());
8           } else {
9              System.out.println("File:"                    +
10 f.getAbsoluteFile());
11             list_file.add(new      FileProperty(f.getName(),
12 f.length() + " byte",f.getAbsolutePath()));
13         }
       }
```

Pada *Algoritme* 4 memaparkan kode sumber *method* chooseFile yang digunakan untuk memasukkan file yang bertipe Java dari folder yang telah dimasukkan ke dalam kelas *model* yang selanjutnya akan ditampilkan pada *table view*. Terdapat perulangan pada setiap File f di sebuah list yang berisi Files, maka akan dilakukan seleksi kondisi jika f adalah direktori maka akan menampilkan *path* direktori pada *console* dan memanggil *recursive method* chooseFile. Jika f bukan direktori maka akan menampilkan *path file* pada *console* dan menambahkan data pada list_file berupa nama kelas file tersebut dan ukurannya yang disimpan di kelas FileProperty.

5.2.2.2 Implementasi Kode Program Method Visit

Nama Kelas: ClassAndFieldExtraction.java

Nama Operasi: visit

Algoritme 5: Kode sumber method visit

```
1  className = n.getNameAsString();
2  sourceCode = n.toString();
3  System.out.println("Class name  : " + className);
4  n.getExtendedTypes().forEach(extendedType -> {
5    String superClass = extendedType.getNameAsString();
6    inheritanceProperty.setInheritance(className,
7  superClass);
8     listInheritance.add(superClass);
9     inheritanceProperty.getListOfInheritances().entrySet().fo
      rEach(classData -> {
10        String subClass = classData.getKey();
11        List<String> supClass = classData.getValue();
12        int depthOfInheritance = 0;
13          if     (superClass.equalsIgnoreCase(subClass))     {
14          if(inheritanceProperty.getListOfInheritances().cont
15          ainsKey(subClass)) {
16              listInheritance.addAll(supClass);
17           }
18           }
19      });
20     inheritanceProperty.setlistOfInheritances(className,
21  listInheritance);
22  });
23  if (n.isInterface()) {
24     classType = "interface";
25     System.out.println("This is interface class");
26     classProperty.setClassProperty(className,      classType,
27  sourceCode);
28  } else if (n.isAbstract()) {
29     classType = "abstract";
30     System.out.println("This is abstract class");
31     classProperty.setClassProperty(className,      classType,
32  sourceCode);
33  } else {
34      System.out.println("This is concrete class");
35      classType = "concrete";
36      classProperty.setClassProperty(className,     classType,
37  sourceCode);
38      n.walk(node -> {
39        if (node instanceof FieldDeclaration) {
40        FieldDeclaration fieldDeclaration = (FieldDeclaration)
41  node;
42        attributeType                                          =
    fieldDeclaration.getVariables().get(0).getTypeAsString();
```

Algoritme 5: Kode sumber method visit

```
43          attributeName                                          =
44  fieldDeclaration.getVariables().get(0).getNameAsString();
45          System.out.println("Attribute   : type " + " -> [" +
46  attributeType + "]");
47
48          attributeProperty.setAttributeProperty(className,
            attributeName, attributeType);
49      } else if (node instanceof MethodDeclaration) {
50          MethodDeclaration        methodDeclaration        =
51  (MethodDeclaration) node;
52          methodName = methodDeclaration.getNameAsString();
53          bodyMethod                                             =
54  methodDeclaration.getBody().get().toString();
55          sourceCodeMethod = methodDeclaration.toString();
56          String                  annotation                     =
57  methodDeclaration.getAnnotationByName("Override").toString(
    );
58              if
59  (annotation.equalsIgnoreCase("Optional[@Override]")) {
60                  typeMethod = "override";
61
62  methodProperty.setMethodProperty(className,      methodName,
63  typeMethod, bodyMethod, sourceCodeMethod);
64              } else {
65                  typeMethod = "common";
66
67  methodProperty.setMethodProperty(className,      methodName,
    typeMethod, bodyMethod, sourceCodeMethod);
68              }
69          }
70      });
71  System.out.println("====================================="
72  );
73  }
```

Pada *Algortime* 5 memaparkan kode sumber *method visit* yang digunakan untuk melakukan ekstraksi *property* pada sebuah kelas. Terdapat perulangan dari kelas yang memiliki tipe *extends* untuk menyimpan *inheritance property*, selanjutnya dilakukan seleksi kondisi jika kelas tersebut kelas *interface* maka hanya akan menyimpan *class property*. Jika merupakan kelas *abstract* atau kelas *concrete* selain menyimpan *class property* juga dilakukan perulangan. Di perulangan tersebut akan terjadi seleksi kondisi jika node adalah deklarasi *field* maka akan menyimpan *attribute property*, berupa nama *attribute* dan tipe data *attribute*. Jika node adalah deklarasi *method* maka akan melakukan seleksi kondisi jika *method* adalah *abstract* maka menyimpan *abstract method property* sedangkan jika tidak maka melakukan seleksi kondisi jika *annotation* atau *type method* adalah *override* maka menyimpan *method property* dan RMC atau *method* turunan yang digunakan, jika tidak maka *type method* adalah *common* lalu

menyimpan *method property*, yaitu berupa nama *method*, *type method*, body *method*, *parameter*, dan *source code method*.

5.2.2.3 Implementasi Kode Program Method calculate

Nama Kelas: ACCalculation.java

Nama Operasi: calculate

Algoritme 6: Kode sumber method calculate

```
1  ClassProperty.getListOfClass().entrySet().forEach(classData
2  -> {
3     String className = classData.getValue().get(0);
4     int counterAC = 0;
5     int counterPDT = 0;
6     int counterDDT = 0;
7     int counterUDDT = 0;
8     for    (Map.Entry<Integer,    List<String>>    entry    :
9  listAttribute.entrySet()) {
10       attrClassName = entry.getValue().get(0);
11       String attribute = entry.getValue().get(2);
12       if (attrClassName.equalsIgnoreCase(className)) {
13          if (attribute.endsWith("int")
14             || attribute.endsWith("String")
15             || attribute.endsWith("float")
16             || attribute.endsWith("char")
17             || attribute.endsWith("double")
18             || attribute.endsWith("boolean")) {
19             counterPDT++;
20          } else if (attribute.endsWith("[]")) {
21             counterDDT++;
22          } else {
23             counterUDDT++;
24          }
25         counterAC = counterPDT * wb + counterDDT * wd +
26 counterUDDT * wu;
27         attrResult.setListOfAC(attrClassName, counterAC);
28       }
29    }
   });
```

Pada *Algortime* 6 memaparkan kode sumber *method* calculateAC yang digunakan untuk melakukan perhitungan nilai *attribute complexity.* Pada kode sumber *method* calculateAC akan dilakukan perulangan listOfClass lalu mengambil data className dan inisialisasi nilai counterAC, counterPDT, counterDDT, dan counterUDDT. Kemudian dilakukan perulangan sebanyak listOfAttributes lalu mengambil data attrClassName dan *attribute*. Dalam perulangan dilakukan seleksi kondisi jika attrClassName sama dengan className maka dilakukan seleksi kondisi lagi jika *attribute* merupakan tipe PDT maka counterPDT bertambah, jika *attribute* merupakan tipe DDT maka counterDDT bertambah, dan jika *attribute* merupakan

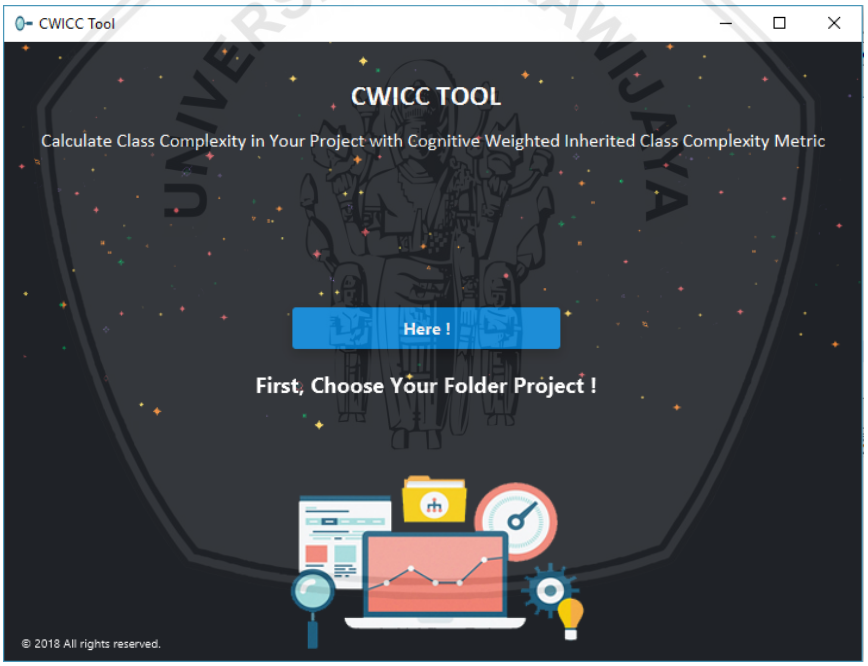
tipe UDDT maka counter UDDT bertambah. Selanjutnya dilakukan perhitungan counterAC dengan melakukan perkalian nilai counterPDT, counter DDT, dan counter UDDT dengan bobot pemahaman dan menjumlahkan hasilnya yang kemudian di simpan di listOfAC.

5.2.3 Implementasi Antarmuka

Hasil dari perancangan antarmuka yang dilakukan pada tahap perancangan digunakan sebagai acuan untuk membuat implementasi antarmuka. Berdasarkan perancangan antarmuka, antarmuka yang akan dibuat adalah implementasi antarmuka *input* berkas, antarmuka daftar *file* dan antarmuka hasil perhitungan.

5.2.3.1 Antarmuka Input Berkas

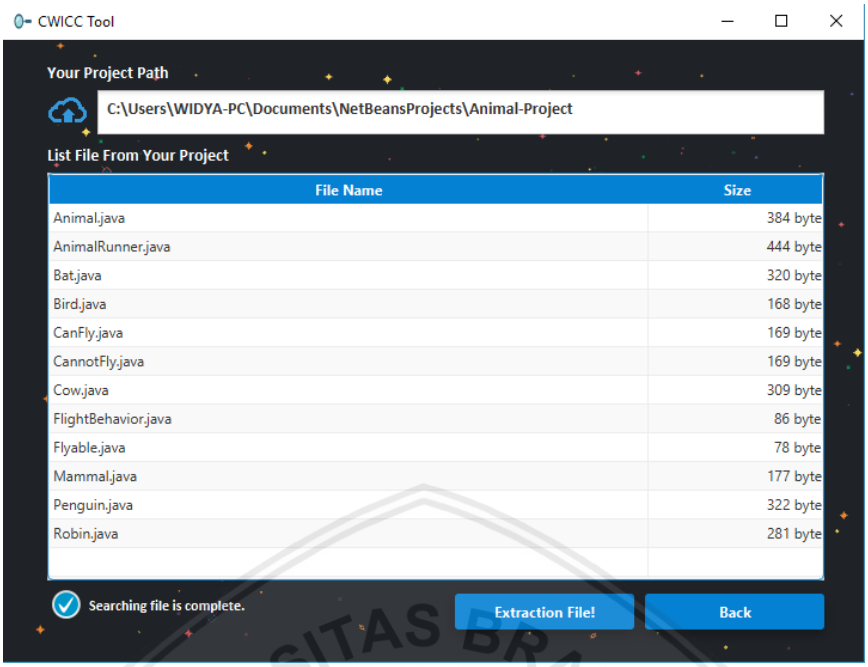
Pada implementasi antarmuka input berkas yang ditunjukkan pada Gambar 5.8, terdapat perintah untuk memasukkan folder project yang akan diuji, dan terdapat tombol untuk memasukkan berkas yang jika ditekan maka akan menampilkan *window browse file* untuk memilih project yang akan di uji. Selain itu juga terdapat deskripsi singkat sistem.



Gambar 5.8 Implementasi Antarmuka Input Berkas

5.2.3.2 Antarmuka Daftar File

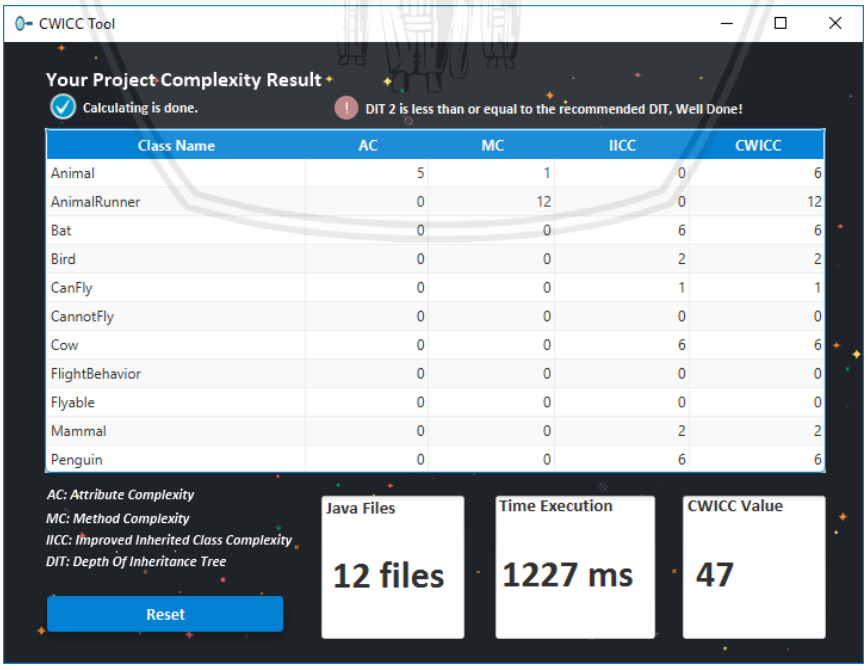
Pada implementasi antarmuka daftar file yang ditunjukkan pada Gambar 5.9, terdapat informasi tentang lokasi dari folder project yang akan di uji beserta list seluruh file bertipe Java. Selanjutnya terdapat tombol Calculate yang jika ditekan maka akan melakukan proses esktraksi *Abstract Syntax Tree* dan melakukan proses perhitungan kompleksitas.



Gambar 5.9 Implementasi Antarmuka Daftar File

5.2.3.3 Antarmuka Hasil Perhitungan

Pada implementasi antarmuka hasil perhitungan yang ditunjukkan pada Gambar 5.10, terdapat informasi tentang daftar kelas yang diuji, nilai *attribute complexity, method complexity, improved inherited complexity,* dan nilai CWICC yang merupakan total dari ketiga kompleksitas tersebut. Selain itu juga terdapat informasi tentang jumlah file yang diuji, waktu eksekusi, dan total nilai CWICC. Terdapat tombol Reset jika pengguna ingin menguji berkas lain.



Gambar 5.10 Implementasi Antarmuka Hasil Perhitungan

BAB 6 PENGUJIAN

Bab Pengujian merupakan tahapan yang bertujuan untuk menemukan dan memperbaiki kesalahan dalam program sebanyak mungkin sebelum diserahkan ke *customer.* Dalam melakukan pengujian, penulis akan menggunakan metode *manual testing* yaitu pengujian yang dilakukan tanpa menggunakan *tools* untuk menguji kebutuhan fungsional maupun non-fungsional. Pada penelitian ini strategi pengujian yang digunakan yaitu pengujian unit, dan pengujian integrasi yang menggunakan metode *white-box testing*. Kemudian dilakukan strategi pengujian validasi yang menggunakan metode *black-box testing*. Pengujian berikutnya yaitu pengujian akurasi dan performa yang berdasarkan kebutuhan non-fungsional untuk memastikan kinerja sistem.

6.1 Pengujian Unit

Pada strategi pengujian unit dalam penelitian ini, penulis menggunakan teknik *basis path testing* untuk melakukan identifikasi pada *independent path* yang ada yaitu dengan membuat *flow graph*. *Flow graph* dibuat berdasarkan *pseudocode* yang sebelumnya telah dibuat pada tahap perancangan komponen. Pengujian unit akan menguji tiga *method* yang mewakili penelitian ini yaitu *method* chooseFile, *method* visit, dan *method* calculateAC.

6.1.1 Pengujian Unit Method chooseFile

Pengujian unit pada *method* chooseFile dari kelas FileController dimulai dengan menentukan *node* untuk membentuk *flow graph* berdasarkan *pseudocode* pada Algoritme 7. Kemudian *flow graph* yang ditunjukkan pada Gambar 6.1 ditentukan nilai *Cyclomatic Complexity* untuk menentukan *independent path.* Dimana jalur tersebut akan digunakan sebagai dasar menentukan kasus uji. Hasil pengujian unit *method* chooseFile ditunjukkan pada Tabel 6.1.

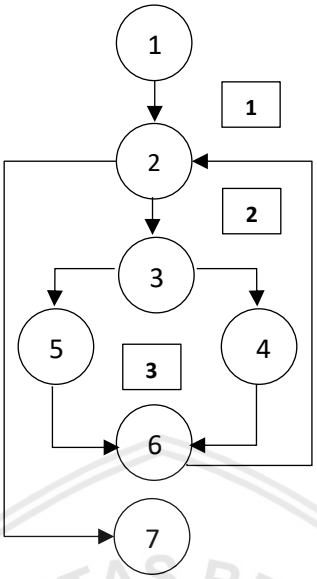
6.1.1.1 Pseudocode

Algoritme 7: Pseudocode method chooseFile

```
1   Deklarasi dan instansiasi root                          } 1
2   Deklrasi dan inisialisasi array list                    }
3   For setiap f pada list 2
4      If (f adalah directory) 3
5       Maka menampilkan path directory pada console        }
6       Memanggil method updateMessage                      } 4
7       Memanggil method chooseFile                         }
8       Else                                                }
9       Maka menampilkan path file pada console             } 5
10      Maka menambahkan data pada list_file                }
11      End if 6
12  End For 7
```

6.1.1.2 Basis Path Testing

a. *Flow Graph*



Gambar 6.1 *Flow Graph method* chooseFile

b. *Cyclomatic Complexity* (V(G))
V(G) = jumlah *region* = 3
V(G) = E - N + 2 = 8 – 7 + 2 = 3
V(G) = P + 1 = 2 + 1 = 3

c. Independent Path
Jalur 1:1-2-7
Jalur 2: 1-2-3-4-6-2-7
Jalur 3: 1-2-3-5-6-2-7

Tabel 6.1 Hasil pengujian unit method chooseFile

Jalur	Prosedur	Hasil yang diharapkan	Hasil pengujian	Status
1	*Class Driver* memanggil *method* chooseFile() dengan masukan berupa alamat *directory path* : "C:\\Users\\WIDYA-PC\\Documents\\NetBeansProjects\\kosong"	Menjalankan *method* chooseFile() dan menampilkan pesan pada *console* “Folder kosong”	Menjalankan *method* chooseFile() dan menampilkan pesan pada *console* “Folder kosong”	Valid
2	*Class Driver* memanggil *method* chooseFile() dengan masukan berupa alamat *directory path* : "C:\\Users\\WIDYA-PC\\Documents\\NetBeansProjects\\direktori"	Menjalankan *method* chooseFile() dan menampilkan path direktori yang ditelusuri serta menampilkan pesan pada *console* “Folder kosong”	Menjalankan *method* chooseFile() dan menampilkan path direktori yang ditelusuri serta menampilkan pesan pada *console* “Folder kosong”	Valid

Tabel 6.1 (lanjutan)

Jalur	Prosedur	Hasil yang diharapkan	Hasil pengujian	Status
3	*Class Driver* memanggil *method* chooseFile() dengan masukan berupa alamat *directory path* : "C:\\Users\\WIDYA-PC\\Documents\\NetBeansProjects\\direktori2"	Menjalankan *method* chooseFile() dan menampilkan path direktori dan file yang ditelusuri serta menampilkan pesan pada *console* "Folder berisi file". Kemudian menampilkan isi list_file.	Menjalankan *method* chooseFile() dan menampilkan path direktori dan file yang ditelusuri serta menampilkan pesan pada *console* "Folder berisi file". Kemudian menampilkan isi list_file.	Valid

6.1.2 Pengujian Unit Method visit

Pengujian unit pada *method* visit dari kelas ClassAndFieldExtraction dengan menentukan node untuk membentuk *flow graph* berdasarkan *pseudocode* pada *Algoritme* 8. Kemudian *flow graph* yang ditunjukkan pada Gambar 6.2 ditentukan nilai *Cyclomatic Complexity* untuk menentukan *independent path*. Dimana jalur tersebut akan digunakan sebagai dasar untuk menentukan kasus uji. Hasil pengujian unit pada *method* visit ditunjukkan pada Tabel 6.2.

6.1.2.1 Pseudocode

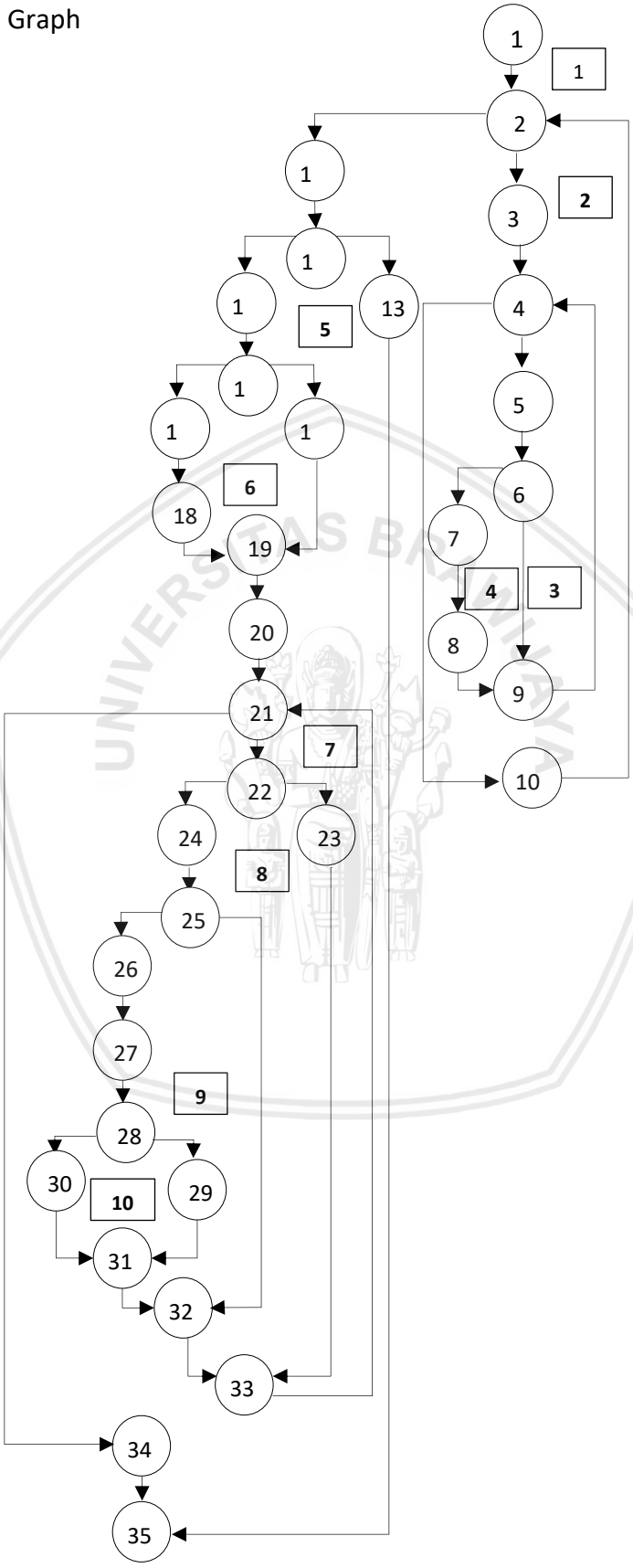
```
Algoritme 8: Pseudocode method visit
1   Inisialisasi variable className
2   Inisialisasi variable sourceCode                    } 1
3   Foreach extendType melakukan 2
4      Inisialisasi superClass
5      Menyimpan className, superClass                  } 3
6      Menambahkan superClass pada listInheritance
7      Foreach listOfInheritances 4
8      Mengambil subClass
9      Mengambil supClass                               } 5
10     Inisialisasi depthOfInheritance = 0
11     If superClass = subClass 6
12        If listOfInheritances dari subClass           } 7
13           Menambahkan supClass pada listInheritance
14        Ende if 8
15     End if 9
16     End foreach                                      } 8
17     Menambahkan listOfInheritances
18  End foreach 11
19  if kelas interface 12
```

Algoritme 8: Pseudocode method visit

```
    Maka inisialisasi classType = "Interface"                          } 13
    Menyimpan className, classType dan sourceCode                      }
else 14
  if kelas abstract 15
     Maka inisialisasi classType = "Abstract"                          } 16
     Menyimpan className, classType, dan sourceCode                    }
  Else 17
     Maka inisialisasi classType = "Concrete"                          } 18
     Menyimpan className, classType, dan sourceCode                    }
  End if 19
  Mengambil semua node dari n 20
  Foreach node melakukan 21
   If node adalah FieldDeclaration 22
      Inisialisasi attributeType                                       }
      Inisialisasi attributeName                                       } 23
      Menyimpan className,attributeType,dan attributeName              }
   Else if node adalah MethodDeclaration 24
       If method abstract                                              } 25
       Menyimpan className, methodName, parameter                      }
       Else 26
         Inisialisasi methodName                                       }
         Inisialisasi bodyMethod                                       } 27
         Inisialisasi sourceCodeMethod                                 }
         Inisialisasi annotation                                       }
          If annotation = "Optional[@Override]" 28
            Inisialisasi typeMethod = "override" 29
            Menyimpan   className,   methodName,   typeMethod,
bodyMethod, dan methodSourceCode 29
          Else                                                         }
            Inisialisasi typeMethod = "common"                         } 30
            Menyimpan   className,   methodName,   typeMethod,
bodyMethod, dan methodSourceCode 30
         End if 31
       End if 32
     End if 33
   End foreach 34
End if 35
```

6.1.2.2 Basis Path Testing

a. Flow Graph



Gambar 6.2 *Flow Graph method* visit

a. *Cyclomatic Complexity* (V(G))

V(G) = jumlah *region* = 10

V(G) = E - N + 2 = 43 – 35 + 2 = 10

V(G) = P + 1 = 9 + 1 = 10

b. Independent Path

Jalur 1: 1-2-11-12-13-35

Jalur 2: 1-2-11-12-14-15-16-19-20-21-34-35

Jalur 3: 1-2-11-12-14-15-17-18-19-20-21-34-35

Jalur 4: 1-2-3-4-10-2-11-12-14-15-17-19-20-21-34-35

Jalur 5: 1-2-3-4-5-6-9-4-10-2-11-12-14-15-17-18-19-20-21-34-35

Jalur 6: 1-2-3-4-5-6-7-8-9-4-10-2-11-12-14-15-17-18-19-20-21-34-35

Jalur 7: 1-2-3-4-5-6-7-8-9-4-10-2-11-12-14-15-17-18-19-20-21-22-23-33-21-34-35

Jalur 8: 1-2-3-4-5-6-7-8-9-4-10-2-11-12-14-15-17-18-19-20-21-22-24-25-32-33-21-34-35

Jalur 9: 1-2-3-4-5-6-7-8-9-4-10-2-11-12-14-15-17-18-19-20-21-22-24-25-26-27-28-29-31-32-33-21-34-35

Jalur 10: 1-2-3-4-5-6-7-8-9-4-10-2-11-12-14-15-17-18-19-20-21-22-24-25-26-27-28-30-31-32-33-21-34-35

Tabel 6.2 Hasil pengujian unit method visit

Jalur	Prosedur	Hasil yang diharapkan	Hasil pengujian	Status
1	*Class Driver* memanggil *method* visit dengan masukan kode sumber: "public interface collector{ }"	Menjalankan *method* visit dan menampilkan pesan pada *console* "*This is interface class*".	Menjalankan *method* visit dan menampilkan pesan pada *console* "*This is interface class*".	Valid
2	*Class Driver* memanggil *method* visit dengan masukan kode sumber: "abstract class Counter{ abstract void count();}"	Menjalankan *method* visit dan menampilkan pesan pada *console* "*Abstract class* berhasil disimpan".	Menjalankan *method* visit dan menampilkan pesan pada *console* "*Abstract class* berhasil disimpan".	Valid
3	*Class Driver* memanggil *method* visit dengan masukan kode sumber: "public class Data{ "}"	Menjalankan *method* visit dan menampilkan pesan pada *console* "*Concrete class* berhasil disimpan".	Menjalankan *method* visit dan menampilkan pesan pada *console* "*Concrete class* berhasil disimpan".	Valid

Tabel 6.2 (lanjutan)

Jalur	Prosedur	Hasil yang diharapkan	Hasil pengujian	Status
4	*Class Driver* memanggil *method* visit dengan masukan kode sumber: "public class Mobil extends Kendaraan{ }"	Menjalankan *method* visit dan menampilkan pesan pada *console* "Data *inheritance class* berhasil disimpan".	Menjalankan *method* visit dan menampilkan pesan pada *console* "Data *inheritance class* berhasil disimpan".	Valid
5	*Class Driver* memanggil *method* visit dengan masukan kode sumber: "public class Mobil extends Kendaraan{ }" dan "public class Motor extends Kendaraan{ }"	Menjalankan *method* visit dan menampilkan pesan pada *console* "Data *inheritance class* dan *concrete class* berhasil disimpan".	Menjalankan *method* visit dan menampilkan pesan pada *console* "Data *inheritance class* dan *concrete class* berhasil disimpan".	Valid
6	*Class Driver* memanggil method visit dengan masukan kode sumber: "public class Mobil extends Kendaraan{ }" dan "public class Avanza extends Mobil{ }"	Menjalankan *method* visit dan menampilkan pesan pada *console* "Data *multiple inheritance* dan *concrete class* berhasil disimpan".	Menjalankan *method* visit dan menampilkan pesan pada *console* "Data *multiple inheritance* dan *concrete class* berhasil disimpan".	Valid
7	*Class Driver* memanggil *method* visit dengan masukan kode sumber: "public class Mobil extends Kendaraan{ }" dan "public class Avanza extends Mobil{ int a; }"	Menjalankan *method* visit dan menampilkan pesan pada *console* "Data *multiple inheritance*, *concrete class* dan *attribute* berhasil disimpan".	Menjalankan *method* visit dan menampilkan pesan pada *console* "Data *multiple inheritance*, *concrete class* dan *attribute* berhasil disimpan".	Valid
8	*Class Driver* memanggil *method* visit dengan masukan kode sumber: "public abstract class Kendaraan { abstract void method(); }" dan "public class Mobil extends Kendaraan{ }"	Menjalankan *method* visit dan menampilkan pesan pada *console* "Data *multiple inheritance*, *abstract class*, *concrete class*, dan *abstract method* berhasil disimpan".	Menjalankan *method* visit dan menampilkan pesan pada *console* "Data *multiple inheritance*, *abstract class*, *concrete class*, dan *abstract method* berhasil disimpan".	Valid

Tabel 6.2 (lanjutan)

Jalur	Prosedur	Hasil yang diharapkan	Hasil pengujian	Status
9	*Class Driver* memanggil *method* visit dengan masukan kode sumber: "public class Mobil extends Kendaraan{ Public void methodB(){} }" dan "public class Avanza extends Mobil{ int a; @Override public void methodB(){} }"	Menjalankan *method* visit dan menampilkan pesan pada *console* "Data *multiple inheritance* dan *concrete class*, *attribute*, *method*, dan *method* turunan berhasil disimpan".	Menjalankan *method* visit dan menampilkan pesan pada *console* "Data *multiple inheritance* dan *concrete class*, *attribute*, *method*, dan *method* turunan berhasil disimpan".	Valid
10	*Class Driver* memanggil *method* visit dengan masukan kode sumber: "public class Mobil extends Kendaraan{ Public void methodB(){} }" dan "public class Avanza extends Mobil{ int a; public void method(){} }"	Menjalankan *method* visit dan menampilkan pesan pada *console* "Data *multiple inheritance*, *concrete class*, *attribute,* dan *method* berhasil disimpan".	Menjalankan *method* visit dan menampilkan pesan pada *console* "Data *multiple inheritance*, *concrete class*, *attribute,* dan *method* berhasil disimpan".	Valid

6.1.3 Pengujian Unit Method calculateAC

Pengujian unit pada *method* calculateAC pada kelas ACCalculation dengan menentukan node untuk membentuk *flow graph* berdasarkan *pseudocode* yang telah dirancangan yang ditunjukkan pada *Algoritm*e 9. Kemudian *flow graph* yang ditunjukkan pada Gambar 6.3 ditentukan nilai *Cyclomatic Complexity* untuk menentukan *independent path*. Dimana jalur tersebut akan digunakan sebagai dasar untuk menentukan kasus uji. Hasil pengujian unit *method* calculateAC ditunjukkan pada Tabel 6.3.

6.1.3.1 Pseudocode

Algoritme 9: Pseudocode method calculateAC

```
1  Foreach listOfClass 1
2     Mengambil className
3     Inisialisasi counterAC = 0
4     Inisialisasi counterPDT = 0      } 2
5     Inisialisasi counterUDT = 0
6     Inisialisasi counterUDDT = 0
7     Foreach listOfAttributes 3
```

Algoritme 9: Pseudocode method calculateAC	
8	Mengambil attrClassName } **4**
9	Mengambil attribute }
10	If attrClassName = className **5**
11	If attribute PDT **6**
12	add counterPDT **7**
13	Else if attribute UDT **8**
14	add counterUDT **9**
15	Else if attribute UDDT **10**
16	add ounterUDDT **11**
17	End if **12**
18	counterAC = counterPDT * wb + counterUDT * wd +
19	counterUDDT * wu } **12**
20	Menyimpan listOfAC }
21	End if **13**
22	End for **14**
23	End for **15**

6.1.3.2 Basis Path Testing

a. Flow Graph
   Flow graph dari pengujian unit *method* calculateAC dapat dilihat pada Gambar 6.3.
b. Cyclomatic Complexity (V(G))
   a. V(G) = jumlah *region* = 6
   b. V(G) = E - N+ 2 = 19 – 15 + 2 = 6
   c. V(G) = P + 1 = 5 + 1 = 6
c. Independent Path

   Jalur 1: 1-15

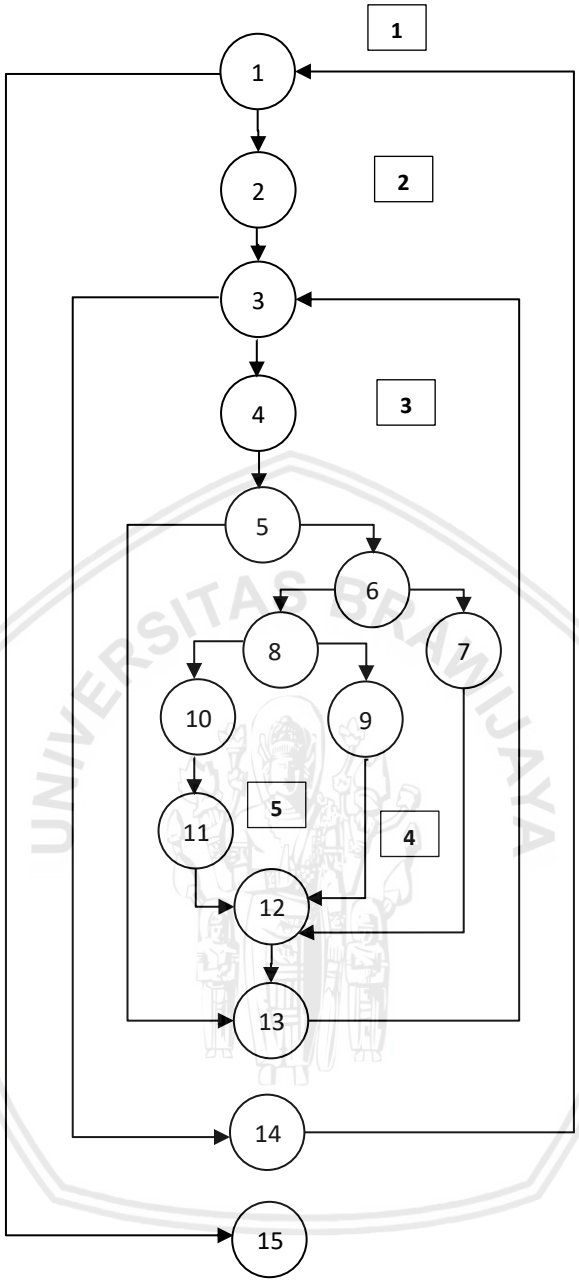
   Jalur 2: 1-2-3-14-1-15

   Jalur 3: 1-2-3-4-5-13-3-14-1-15

   Jalur 4: 1-2-3-4-5-6-7-12-13-3-14-1-15

   Jalur 5: 1-2-3-4-5-6-8-9-12-13-3-14-1-15

   Jalur 6: 1-2-3-4-5-6-8-10-11-12-13-3-14-1-15



Gambar 6.3 *Flow Graph method* calculateAC

Tabel 6.3 Hasil pengujian unit method calculateAC

Jalur	Prosedur	Hasil yang diharapkan	Hasil pengujian	Status
1	*Class Driver* memanggil *method* calculateAC dengan masukan kode sumber: File kosong	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Nilai AC tidak dapat dihitung"	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Nilai AC tidak dapat dihitung"	Valid

Tabel 6.3 (lanjutan)

Jalur	Prosedur	Hasil yang diharapkan	Hasil pengujian	Status
2	*Class Driver* memanggil *method* calculateAC dengan masukan kode sumber: "public class Dataku{}" pada proses ekstraksi.	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Dataku AC Value: 0 Nilai AC berhasil disimpan"	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Dataku AC Value: 0 Nilai AC berhasil disimpan"	Valid
3	*Class Driver* memanggil *method* calculateAC dengan masukan kode sumber: "public class Dataku{}" dan "public class Datamu{ Public int c;}" pada proses ekstraksi.	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Dataku AC Value: 0 Class Name: Datamu AC Value: 1 Nilai AC berhasil disimpan"	Menjalankan *method* calculate AC dan menampilkan pesan pada *console* "Class Name: Dataku AC Value: 0 Class Name: Datamu AC Value: 1 Nilai AC berhasil disimpan"	Valid
4	*Class Driver* memanggil *method* calculateAC dengan masukan kode sumber: "public class Data { public int a;}" pada proses ekstraksi.	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Data AC Value: 1 Nilai AC berhasil disimpan"	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Data AC Value: 1 Nilai AC berhasil disimpan"	Valid
5	*Class Driver* memanggil *method* calculateAC dengan masukan kode sumber: "public class Data { public int a; public int b[] = new int[10];}" pada proses ekstraksi.	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Data AC Value: 3 Nilai AC berhasil disimpan"	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Data AC Value: 3 Nilai AC berhasil disimpan"	Valid
6	*Class Driver* memanggil *method* calculateAC yang memanggil *method* setListOfAC dengan masukan kode sumber: "public class Data { public int a; public int b[] = new int[10]; public Angka angka;}" pada proses ekstraksi.	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Data AC Value: 6 Nilai AC berhasil disimpan"	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Data AC Value: 6 Nilai AC berhasil disimpan"	Valid

6.2 Pengujian Integrasi

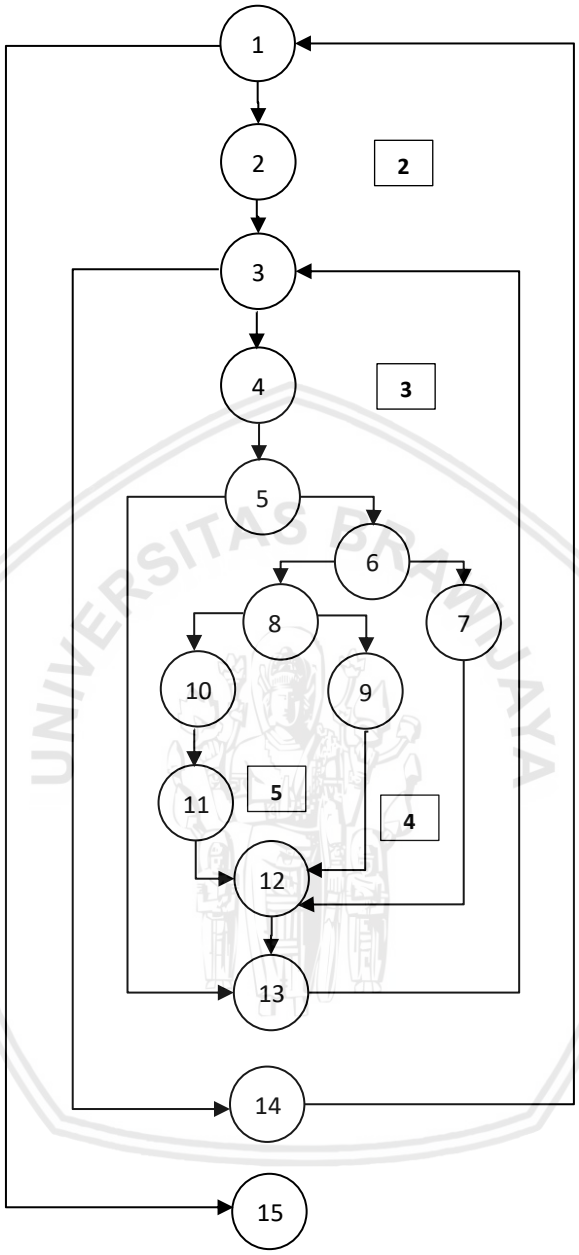
Pada pengujian integrasi dilaukan untuk memastikan interaksi antara suatu kelas dengan kelas yang lain berjalan dengan baik. Pada penelitian ini pengujian integrasi menggunakan teknik *basis path testing* untuk menguji interaksi method calculateAC() pada kelas ACCalculation dan method setListOfAC() pada kelas AttributeComplexityresult. Dalam interaksi antar kelas tersebut, method calculateAC pada kelas ACCalculation akan memanggil method setListOfAC() pada AttributeComplexityResult. Kemudian berdasarkan *pseudocode* maka dibuat *flow graph* yang ditunjukkan pada Gambar 6.4, selanjutnya ditentukan nilai *Cyclomatic Complexity* untuk menentukan *independent path*. Dimana jalur tersebut akan digunakan sebagai dasar untuk menentukan kasus uji. Hasil pengujian integrasi ditunjukkan pada Tabel 6.5.

1. Pseudocode

```
Algoritme 10: Pseudocode method calculateAC
1   Foreach listOfClass 1
2      Mengambil className
3      Inisialisasi counterAC = 0
4      Inisialisasi counterPDT = 0                    } 2
5      Inisialisasi counterUDT = 0
6      Inisialisasi counterUDDT = 0
7      Foreach listOfAttributes 3
8          Mengambil attrClassName                    } 4
9          Mengambil attribute
10         If attrClassName = className 5
11            If attribute PDT 6
12               add counterPDT 7
13            Else if attribute UDT 8
14               add counterUDT++ 9
15            Else if attribute UDDT 10
16               add counterUDDT++ 11
17            End if 12
18            counterAC = counterPDT * wb + counterUDT * wd +
19 counterUDDT * wu                                   } 12
20         Menyimpan listOfAC
21         End if 13
22     End for 14
23 End for 15
```

2. Basis Path Testing

a. Flow Graph



Gambar 6.4 *Flow Graph method* calculateAC

d. *Cyclomatic Complexity* (V(G))
V(G) = jumlah *region* = 6
V(G) = E - N + 2 = 19 – 15 + 2 = 6
V(G) = P + 1 = 5 + 1 = 6

e. Independent Path

Jalur 1: 1-15

Jalur 2: 1-2-3-14-1-15

Jalur 3: 1-2-3-4-5-13-3-14-1-15

Jalur 4: 1-2-3-4-5-6-7-12-13-3-14-1-15

Jalur 5: 1-2-3-4-5-6-8-9-12-13-3-14-1-15

Jalur 6: 1-2-3-4-5-6-8-10-11-12-13-3-14-1-15

Tabel 6.4 Hasil pengujian integrasi

Jalur	Prosedur	Hasil yang diharapkan	Hasil pengujian	Status
1	*Class Driver* memanggil *method* calculateAC yang memanggil *method* setListOfAC dengan masukan kode sumber: File kosong	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Nilai AC tidak dapat dihitung"	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Nilai AC tidak dapat dihitung"	Valid
2	*Class Driver* memanggil *method* calculateAC yang memanggil *method* setListOfAC dengan masukan kode sumber: "public class Dataku{}" pada proses ekstraksi.	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Dataku AC Value: 0 Nilai AC berhasil disimpan"	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Dataku AC Value: 0 Nilai AC berhasil disimpan"	Valid
3	*Class Driver* memanggil *method* calculateAC yang memanggil *method* setListOfAC dengan masukan kode sumber: "public class Dataku{}" dan "public class Datamu{ Public int c;}" pada proses ekstraksi.	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name : Dataku AC Value: 0 Class Name: Datamu AC Value: 1 Nilai AC berhasil disimpan"	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name : Dataku AC Value: 0 Class Name: Datamu AC Value: 1 Nilai AC berhasil disimpan"	Valid
4	*Class Driver* memanggil *method* calculateAC yang memanggil *method* setListOfAC dengan masukan kode sumber: "public class Data { public int a;}" pada proses ekstraksi.	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Data AC Value: 1 Nilai AC berhasil disimpan"	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Data AC Value: 1 Nilai AC berhasil disimpan"	Valid

Tabel 6.4 (lanjutan)

Jalur	Prosedur	Hasil yang diharapkan	Hasil pengujian	Status
5	*Class Driver* memanggil *method* calculateAC yang memanggil *method* setListOfAC dengan masukan kode sumber: "public class Data { public int a; public int b[] = new int[10];}" pada proses ekstraksi.	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Data AC Value: 3 Nilai AC berhasil disimpan"	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Data AC Value: 3 Nilai AC berhasil disimpan"	Valid
6	*Class Driver* memanggil *method* calculateAC yang memanggil *method* setListOfAC dengan masukan kode sumber: "public class Data { public int a; public int b[] = new int[10]; public Angka angka;}" pada proses ekstraksi.	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Data AC Value: 6 Nilai AC berhasil disimpan"	Menjalankan *method* calculateAC dan menampilkan pesan pada *console* "Class Name: Data AC Value: 6 Nilai AC berhasil disimpan"	Valid

6.3 Pengujian Validasi

Pengujian validasi akan melakukan pengujian pada setiap kebutuhan yang telah didefinisikan sebelumnya. Pengujian ini digunakan untuk memastikan bahwa semua kebutuhan telah diterapkan sesuai skenario yang telah dibuat. Pengujian validasi akan menguji seluruh kebutuhan baik kebutuhan fungsional dan kebutuhan non-fungsional. Pengujian validasi menggunakan metode *black-box testing*. Pada penelitian ini akan menguji empat kebutuhan fungsional beserta alur alternatif dan dua kebutuhan non-fungsional.

6.3.1 Pengujian Validasi Input Berkas

Pada pengujian validasi input berkas akan menguji kebutuhan fungsional dengan kode CT-01-001. Pada pengujian ini terdiri dari satu jalur utama dan dua jalur alternatif. Pada pengujian jalur utama akan memastikan bahwa sistem dapat memasukan berkas berupa folder yang berisi file bertipe Java kedalam sistem. Pada pengujian jalur alternatif 1 akan memastikan bahwa sistem kembali ke halaman utama jika menekan tombol Cancel dan tidak ada berkas yang dimasukkan, sedangkan pengujian jalur alternatif 2 akan memastikan bahwa sistem tidak menampilkan daftar *file* bertipe Java karena folder kosong atau tidak terdapat *file* bertipe Java di dalam folder. Pengujian validasi input berkas

ditunjukkan pada Tabel 6.5, input berkas alur alternatif 1 ditunjukkan pada Tabel 6.6, dan input berkas alur alternatif 2 ditunjukkan pada Tabel 6.7.

Tabel 6.5 Kasus uji pengujian validasi input berkas

Kode kebutuhan	CT-01-001
Nama kasus uji	Input Berkas
Tujuan pengujian	Memastikan bahwa sistem dapat memasukan berkas berupa *project folder* yang berisi file kode sumber bertipe Java ke dalam sistem
Data masukan	Alamat dari *directory project* yang berisi *file* kode sumber bertipe Java
Prosedur uji	1. Menekan tombol Here! 2. Memilih lokasi project java 3. Menekan tombol Open
Hasil pengujian yang diharapkan	Sistem menampilkan daftar file kode sumber bertipe Java dari *project folder* tersebut pada halaman daftar file dalam bentuk tabel
Hasil pengujian	Sistem menampilkan daftar file kode sumber bertipe Java dari *project folder* tersebut pada halaman daftar file dalam bentuk tabel
Status	Valid

Tabel 6.6 Kasus uji pengujian validasi input berkas alternatif 1

Kode kebutuhan	CT-01-001
Nama kasus uji	Input Berkas
Tujuan pengujian	Memastikan bahwa sistem kembali ke halaman utama saat tidak ada berkas yang dipilih dan saat menekan tombol Cancel
Data masukan	Menekan tombol Cancel
Prosedur	1. Menekan tombol Here! 2. Memilih lokasi project java 3. Menekan tombol Cancel
Expected Result	Sistem kembali ke halaman utama, dan tidak menyimpan berkas
Result	Sistem kembali ke halaman utama, dan tidak menyimpan berkas
Status	Valid

Tabel 6.7 Kasus uji pengujian validasi input berkas alternatif 2

Kode kebutuhan	CT-01-001
Nama kasus uji	Input Berkas
Tujuan pengujian	Memastikan bahwa sistem menamnpilkan pesan *"File not found! Folder must contain Java files"*. Dan daftar file kode sumber kosong
Data masukan	Alamat *directory project* yang kosong atau tidak berisi *file* kode sumber bertipe Java
Prosedur	1. Menekan tombol Here! 2. Memilih lokasi project java 3. Menekan tombol Open
Hasil pengujian yang diharapkan	Sistem menampilkan pesan *"File not found! Folder must contain Java files"*
Hasil pengujian	Sistem menampilkan pesan *"File not found! Folder must contain Java files"*
Status	Valid

6.3.2 Pengujian Validasi Ekstraksi Abstract Syntax Tree

Pada pengujian validasi ekstraksi *Abstract Syntax Tree* (AST) akan menguji kebutuhan dengan kode CT-01-002. Pada pengujian ini terdapat satu jalur utama dan dua jalur alternatif. Pada pengujian jalur utama yang ditunjukkan pada Tabel 6.7 akan memastikan bahwa sistem dapat melakukan ekstraksi AST dari kode sumber dengan tipe kelas *concrete* dan *abstract*. Pada pengujian jalur alternatif 1 yang ditunjukkan pada Tabel 6.9 akan memastikan bahwa sistem tidak menyimpan hasil ekstraksi AST dari kode sumber yang memiliki tipe kelas *interface*. Sedangkan pengujian jalur alternatif 2 yang ditunjukkan pada Tabel 6.10 akan memastikan bahwa sistem akan menghapus daftar file dan kembali ke halaman utama saat menekan tombol Back.

Tabel 6.8 Kasus uji pengujian validasi ekstraksi *Abstract Syntax Tree*

Kode Kebutuhan	CT-01-002
Nama Kasus Uji	Ekstraksi *Abstract Syntax Tree*
Tujuan pengujian	Memastikan bahwa sistem dapat melakukan ekstraksi AST dari daftar *file* kode sumber yang memiliki tipe kelas *concrete* atau *abstract*
Data masukan	Alamat *project directory* yang berisi *file* kode sumber bertipe Java dengan tipe kelas *concrete* atau *abstract*
Prosedur uji	1. Menekan tombol Here! 2. Memilih lokasi project java 3. Menekan tombol Open 4. Menekan tombol Extraction File!
Hasil pengujian yang diharapkan	Sistem berhasil melakukan ekstraksi AST dan menyimpan data yang dibutuhkan untuk perhitungan

Hasil pengujian	Sistem berhasil melakukan ekstraksi AST dan menyimpan data yang dibutuhkan untuk perhitungan
Status	Valid

Tabel 6.9 Kasus uji pengujian validasi ekstraksi abstract syntax tree alternatif 1

Kode Kebutuhan	CT-01-002
Nama Kasus Uji	Ekstraksi *Abstract Syntax Tree*
Tujuan pengujian	Memastikan bahwa sistem tidak akan menyimpan hasil ekstraksi AST dari daftar file kode sumber yang terdapat kelas bertipe *interface*
Data masukan	Alamat *directory project* yang berisi *file* kode sumber bertipe Java dengan tipe kelas *interface*
Prosedur	1. Menekan tombol Here! 2. Memilih lokasi project java 3. Menekan tombol Open 4. Menekan tombol Extraction File!
Hasil pengujian yang diharapkan	Sistem berhasil melakukan ekstraksi AST namun tidak menyimpan data kelas *interface*
Hasil pengujian	Sistem berhasil melakukan ekstraksi AST namun tidak menyimpan data kelas *interface*
Status	Valid

Tabel 6.10 Kasus uji pengujian validasi ekstraksi abstract syntax tree alternatif 2

Kode Kebutuhan	CT-01-002
Nama Kasus Uji	Ekstraksi *Abstract Syntax Tree*
Tujuan pengujian	Memastikan bahwa sistem menghapus daftar file kode sumber dan kembali ke halaman utama
Data masukan	Alamat *directory project* yang berisi *file* kode sumber bertipe Java
Prosedur	1. Menekan tombol Here! 2. Memilih lokasi project java 3. Menekan tombol Open 4. Menekan tombol Back
Hasil pengujian yang diharapkan	Sistem berhasil menghapus daftar file dan kembali ke halaman utama
Hasil pengujian	Sistem berhasil menghapus daftar file dan kembali ke halaman utama
Status	Valid

6.3.3 Pengujian Validasi Hitung Kompleksitas

Pengujian validasi hitung kompleksitas akan menguji kebutuhan fungsional dengan kode CT-01-003, pada pengujian ini terdiri dari satu jalur utama yang akan memastikan bahwa sistem dapat melakukan perhitungan kompleksitas perangkat lunak menggunakan metrik CWICC yang ditunjukkan pada Tabel 6.11.

Tabel 6.11 Kasus uji pengujian validasi hitung kompleksitas

Kode Kebutuhan	CT-01-003
Nama Kasus Uji	Hitung kompleksitas
Tujuan pengujian	Memastikan bahwa sistem dapat menghitung nilai kompleksitas dari perangkat lunak yang akan di uji
Data masukan	Alamat *directory project* yang berisi *file* kode sumber bertipe Java
Prosedur uji	1. Menekan tombol Here! 2. Memilih lokasi project java 3. Menekan tombol Open 4. Menekan tombol Extraction File! 5. Menekan tombol Calculate!
Hasil pengujian yang diharapkan	Sistem berhasil menampilkan hasil perhitungan nilai kompleksitas kelas dalam bentuk tabel dan menampilkan pesan “Calculating is done”
Hasil pengujian	Sistem berhasil menampilkan hasil perhitungan nilai kompleksitas kelas dalam bentuk tabel dan menampilkan pesan “Calculating is done”
Status	Valid

6.3.4 Pengujian Validasi Reset

Pada pengujian validasi reset yang ditunjukkan pada Tabel 6.12 akan menguji kebutuhan fungsional dengan kode CT-01-004. Pada pengujian validasi reset terdiri dari satu jalur utama yang akan memastikan bahwa sistem mampu menghapus seluruh data yang telah disimpan dan kembali kemenu utama.

Tabel 6.12 Kasus uji pengujian validasi reset

Kode Kebutuhan	CT-01-004
Nama Kasus Uji	Reset
Tujuan pengujian	Memastikan sistem dapat menghapus seluruh data yang telah disimpan dan kembali kemenu utama
Data masukan	Alamat *directory project* yang berisi *file* kode sumber bertipe Java
Prosedur	1. Menekan tombol Here! 2. Memilih lokasi project java, lalu menekan tombol Open 3. Menekan tombol Extraction File! 4. Menekan tombol Calculate! 5. Menekan tombol Reset
Hasil pengujian yang diharapkan	Data file yang diuji sebelumnya berhasil dihapus, dan kembali ke halaman utama
Hasil pengujian	Data file yang diuji sebelumnya berhasil dihapus, dan kembali ke halaman utama
Status	Valid

6.3.5 Pengujian Akurasi

Pengujian akurasi adalah suatu pengujian yang dilakukan dengan tujuan untuk membandingkan hasil perhitungan yang dilakukan secara manual dengan hasil perhitungan yang dilakukan dengan sistem. Pada penelitian ini, pengujian akurasi akan menguji 5 *sample* proyek perangkat lunak yang didapatkan dari GitHub. Dalam 5 proyek tersebut terdiri dari beberapa kelas bertipe Java yang menerapkan teknik *inheritance* yang berbeda-beda untuk memastikan keakuratan sistem dalam menghitung kompleksitas perangkat lunak menggunakan metrik *Cognitive Weighted Inherited Class Complexity*. Pengujian non-fungsional akurasi sistem akan ditunjukkan pada Tabel 6.13. Sedangkan hasil perbandingan perhitungan manual dan perhitungan sistem akan ditunjukkan pada Tabel 6.14.

Tabel 6.13 Pengujian akurasi

Kode kebutuhan	CT-02-001
Nama kasus uji	*Accuration*
Tujuan pengujian	Memastikan sistem dapat melakukan perhitungan kompleksitas perangkat lunak dengan tingkat akurasi minimal 90% untuk setiap perangkat lunak yang di uji.
Data masukan	Alamat *directory project* yang berisi *file* kode sumber bertipe Java
Prosedur	1. Melakukan perhitungan kompleksitas perangkat lunak dengan metrik *Cognitive Weighted Inherited Class Complexity* (CWICC) secara manual 2. Melakukan perhitungan kompleksitas perangkat lunak dengan metrik *Cognitive Weighted Inherited Class Complexity* (CWICC) melalui sistem 3. Membandingkan hasil perhitungan manual dan secara otomatis dengan sistem 4. Melakukan perhitungan jumlah data uji yang valid dan tidak valid dari seluruh data uji. 5. Melakukan perhitungan akurasi dengan membagi jumlah data uji yang valid dengan seluruh data uji yang kemudian dikali 100%
Hasil pengujian yang diharapkan	Sistem memiliki tingkat nilai akurasi perhitungan kompleksitas perangkat lunak minimal 90%
Hasil Pengujian	Sistem memiliki tingkat nilai akurasi perhitungan kompleksitas perangkat lunak minimal 90%
Status	Valid

repository.ub.ac.id

Tabel 6.14 Perbandingan perhitungan manual dan perhitungan sistem

No	Nama dan deskripsi project	Nama class	Hasil perhitungan manual					Hasil perhitungan oleh sistem					Status
			AC	MC	IICC	CWICC	Total	AC	MC	IICC	CWICC	Total	
1.	Contoh project dari jurnal acuan penelitian, terdiri dari 3 class utama dengan inheritance 2 level.	BaseClass	9	23	0	32	67	9	23	0	32	67	Valid
		DerivedClass1	3	8	5	16		3	8	5	16		Valid
		DerivedClass2	3	8	8	19		3	8	8	19		Valid
2.	Animal-Project, didapat dari GitHub yang terdiri dari 12 class. Dengan inheritance 2 level.	Animal	5	1	0	6	47	5	1	0	6	47	Valid
		AnimalRunner	0	12	0	12		0	12	0	12		Valid
		Bat	0	0	6	6		0	0	6	6		Valid
		Bird	0	0	2	2		0	0	2	2		Valid
		CanFly	0	0	1	1		0	0	1	1		Valid
		ConnotFly	0	0	0	0		0	0	0	0		Valid
		Cow	0	0	6	6		0	0	6	6		Valid
		FlightBehaviour	0	0	0	0		0	0	0	0		Valid
		Flyable	0	0	0	0		0	0	0	0		Valid
		Mammal	0	0	2	2		0	0	2	2		Valid
		Penguin	0	0	6	6		0	0	6	6		Valid

repository.ub.ac.id

Tabel 6.14 (lanjutan)

No	Nama dan deskripsi project	Nama class	Hasil perhitungan manual					Hasil perhitungan oleh sistem					Status
			AC	MC	IICC	CWICC	Total	AC	MC	IICC	CWICC	Total	
3.	Chessconsole4-master, didapat dari GitHub yang terdiri dari 10 class dengan inheritance 1 level.	Bishop	0	0	4	4	121	0	0	4	4	121	Valid
		ChessBoard	3	7	24	34		3	7	24	34		Valid
		Chessman	1	0	6	7		1	0	6	7		Valid
		CheessUI	6	46	0	52		6	46	0	52		Valid
		Dash	0	0	4	4		0	0	4	4		Valid
		King	0	0	4	4		0	0	4	4		Valid
		Knight	0	0	4	4		0	0	4	4		Valid
		Pawn	0	0	4	4		0	0	4	4		Valid
		Queen	0	0	4	4		0	0	4	4		Valid
		Rook	0	0	4	4		0	0	4	4		Valid
4.	ShapeProject, didapat dari GitHub yang terdiri dari 7 class. Terdapat maksimal inheritance 1 level.	Circle	4	0	1	5	89	4	0	1	5	89	Valid
		Rectangle	4	0	1	5		4	0	1	5		Valid
		ShapeData	0	0	0	0		0	0	0	0		Valid
		ShapeProject	0	0	0	0		0	0	0	0		Valid
		shapes	9	57	0	66		9	57	0	66		Valid

repository.ub.ac.id

Tabel 6.14 (lanjutan)

No	Nama dan deskripsi project	Nama class	Hasil perhitungan manual					Hasil perhitungan oleh sistem					Status
			AC	MC	IICC	CWICC	Total	AC	MC	IICC	CWICC	Total	
		Square	3	0	1	4		3	0	1	4		Valid
		Triangle	6	0	3	9		6	0	3	9		Valid
5	Java-SuperPaint-Application, didapat dari GitHub yang terdiri dari 10 class. Terdapat maksimal inheritance 2 level.	DrawFrame	36	52	1	89	368	36	52	1	89	368	Valid
		DrawPanel	17	107	1	125		17	107	1	125		Valid
		LinkedList	4	32	0	36		4	32	0	36		Valid
		ListNode	6	0	0	6		6	0	0	6		Valid
		MyBoundedShape	1	24	1	26		1	24	1	26		Valid
		MyLine	0	0	15	15		0	0	15	15		Valid
		MyOval	0	0	32	32		0	0	32	32		Valid
		MyRectangle	0	0	32	32		0	0	32	32		Valid
		MyShape	7	0	0	7		7	0	0	7		Valid
		TestDraw	0	0	0	0		0	0	0	0		Valid

Keterangan:
AC = *Attribute Complexity*
MC = *Method Complexity*
IICC = *Improved Inherited Class Complexity*
CWICC = *Cognitive Weighted Inherited Class Complexity*

Berdasarkan hasil perbandingan perhitungan yang dilakukan secara manual dan dengan menggunakan sistem yang ditunjukkan pada Tabel 6.14 diperoleh hasil bahwa 5 proyek yang diuji dinyatakan *valid* dari total data uji sebanyak 5 proyek. Berdasarkan hasil tersebut diperoleh nilai akurasi sebagai berikut:

$$Accuration = \frac{5}{5} \times 100\% = 100\%$$

6.3.6 Pengujian Performa

Pengujian performa adalah pengujian yang dilakukan dengan cara mengamati waktu yang diperlukan sistem dalam melakukan perhitungan kompleksitas perangkat lunak. Waktu yang diambil adalah waktu ketika sistem mulai melakukan proses ekstraksi *Abstract Syntax Tree* sampai sistem telah selesai melakukan perhitungan kompleksitas perangkat lunak menggunakan metrik *Cognitive Weighted Inherited Class Complexity* (CWICC). Pengujian ini akan mengambil 5 *sample* proyek yang digunakan pada pengujian akurasi. Pengujian ini dilakukan pada lingkungan sistem dengan spesifikasi prosesor Intel® Core™ Core i3-6100U CPU @2.30GHz, *main memory* 4GB dan sistem operasi Windows 10 Pro. Sedangkan untuk JDK yang terinstal di dalam sistem adalah JDK 8. Pengujian validasi kebutuhan non-fungsional *performance* ditunjukkan pada Tabel 6.15 dan hasil pengujian performa ditunjukkan pada Tabel 6.16.

Tabel 6.15 Pengujian performa

Kode kebutuhan	CT-02-002
Nama kasus uji	*Performance*
Tujuan pengujian	Memastikan sistem dapat melakukan perhitungan kompleksitas perangkat lunak pada satu *project* menggunakan sistem dengan rata-rata waktu 356 kali lebih cepat dibandingkan perhitungan yang dilakukan secara manual.
Data masukan	Alamat *directory project* yang berisi *file* kode sumber bertipe Java
Prosedur	1. Melakukan perhitungan kompleksitas perangkat lunak dengan metrik *Cognitive Weighted Inherited Class Complexity* (CWICC) melalui sistem 2. Mengamati hasil waktu yang tertera pada sistem
Hasil pengujian yang diharapkan	Sistem menghitung nilai kompleksitas perangkat lunak pada setiap kasus uji dengan waktu rata-rata 356 kali lebih cepat dibandingan menghitung manual
Hasil Pengujian	Sistem menghitung nilai kompleksitas perangkat lunak pada setiap kasus uji dengan waktu rata-rata 356 kali lebih cepat dibandingan menghitung manual
Status	Valid

repository.ub.ac.id

Tabel 6.16 Perbandingan waktu eksekusi perhitungan manual dan sistem

No	Nama Project dan deskripsi	Waktu perhitungan secara manual	Waktu Perhitungan oleh sistem	Perbandingan	Status
1	Contoh project dari jurnal acuan penelitian, terdiri dari 3 class utama dengan inheritance 2 level.	172000 ms	715 ms	241:1	Valid
2	Animal-Project, didapat dari GitHub yang terdiri dari 12 class. Dengan inheritance 2 level.	194000 ms	613 ms	316:1	Valid
3	Chessconsole4-master, didapat dari GitHub yang terdiri dari 10 class dengan inheritance 1 level.	419000 ms	1094 ms	383:1	Valid
4	ShapeProject, didapat dari GitHub yang terdiri dari 7 class. Terdapat maksimal inheritance 1 level.	380000 ms	1090 ms	349:1	Valid
5	Java-SuperPaint-Application, didapat dari GitHub yang terdiri dari 10 class. Terdapat maksimal inheritance 2 level.	720000 ms	1465 ms	491:1	Valid
Rata-rata				356: 1	

Berdasarkan hasil pengujian performa, waktu sistem untuk menghitung nilai kompleksitas pada lima data uji yang ditunjukkan pada Tabel 6.16 didapatkan bahwa sistem rata-rata dapat menghitung nilai kompleksitas rata-rata 356 kali lebih cepat dibandingkan dengan menghitung secara manual.

BAB 7 PENUTUP

7.1 Kesimpulan

Penarikan kesimpulan dilakukan berdasarkan pada hasil dari seluruh tahapan yang dilakukan pada penelitian ini dimulai dari tahap rekayasa kebutuhan, perancangan sistem, implementasi sistem hingga pengujian yang digunakan untuk menjawab seluruh permasalahan yang telah didefinisikan pada rumusan masalah. Sehingga berdasarkan seluruh tahapan yang telah dilakukan pada penelitian ini dapat ditarik kesimpulan sebagai berikut:

1. Berdasarkan hasil pada tahap rekayasa kebutuhan sistem Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan Metrik *Cognitive Weighted Inherited Class Complexity*, diperoleh 4 kebutuhan fungsional dan 2 kebutuhan non-fungsional yang berfungsi untuk melakukan perhitungan nilai kompleksitas perangkat lunak secara otomatis menggunakan metrik *Cognitive Weighted Inherited Class Complexity*. Kebutuhan fungsional yang telah didefinisikan yaitu input berkas, ekstraksi *Abstract Syntax Tree*, perhitungan kompleksitas, dan reset. Sedangkan kebutuhan non-fungsional yaitu *accuration* dan *performance*. Berdasarkan kebutuhan yang telah didefiniskan, selanjutnya dimodelkan menggunakan *use case diagram* dan *use case scenario.*
2. Berdasarkan hasil pada tahap perancangan sistem yang telah dilakukan diperoleh perancangan arsitektur yaitu *sequence diagram* dan *class diagram.* Pada perancangan komponen diperoleh rancangan *algoritme* yaitu berupa *pseudocode* yang diimplementasikan menjadi *method* pada sistem yang dibangun. Pada tahap perancangan antarmuka diperoleh hasil rancangan layout *User Interface* sistem. Sedangkan pada tahap implementasi sistem diperoleh hasil spesifikasi sistem, implementasi kode program, dan implementasi antarmuka yang berdasarkan hasil perancangan.
3. Berdasarkan hasil pengujian yang telah dilakukan diperoleh hasil bahwa pengujian unit, pengujian integrasi, dan pengujian validasi menghasilkan nilai valid 100% untuk setiap kasus uji.
4. Berdasarkan hasil pengujian akurasi dari sistem yang telah dilakukan diperoleh nilai akurasi sebesar 100%.

7.2 Saran

Saran yang diberikan untuk memperbaiki kesalahan dan pengembangan lebih lanjut pada sistem Kakas Bantu Perhitungan Kompleksitas Perangkat Lunak Menggunakan Metrik Cognitive Weighted Inherited Class Complexity, adalah sebagai berikut:

1. Sistem dapat dikembangkan kembali dengan menambahkan fitur menentukan status kompleksitas berdasarkan nilai kompleksitas dari perangkat lunak seperti *low, medium,* dan *high* berdasarkan penelitian terbaru. Saat ini belum ada penelitian yang menentukan ukuran tetap untuk menentukan status kompleksitas perangkat lunak dengan menggunakan metrik *Cognitive Weighted Inherited Class Complexity*.
2. Sistem dapat dikembangkan kembali dengan menambahkan fitur visualisasi dalam bentuk *class diagram* dan menampilkan nilai kompleksitasnya pada setiap komponen yang membangun kelas untuk memudahkan pembacaan dan menentukan bagian mana yang memiliki kompleksitas tertinggi.
3. Sistem dapat dikembangkan dengan menambahkan fitur menghitung nilai kompleksitas dengan menggunakan metrik perhitungan kompleksitas tentang *inheritance* yang lain. Sebagai bahan perbandingan keakuratan.

DAFTAR REFERENSI

Aloysius, A., & Arockiam, L. 2011. *Attribute weighted class complexity, International Journal of Computer, Electrical, Automation, Control and Information Engineering* 5(10): 1151-1156.

Aloysius, A., & Arockiam, L. 2011. A Survey on Metric of Software Cognitive Complexity for OO design, *5*(10), 1127–1131.

Baesens, Bart, Aimee Backiel, and Seppe vanden Broucke. 2015. Beginning Java® Programming: The Object-Oriented Approach. [e-book] Crosspoint Boulevard Indianapolis, IN: John Wiley & Sons, Inc.

Banker, R. D., Davis, G. B., & Slaughter, S. A. (n.d.). 1998. *Software Development Practices , Software Complexity , and Software Maintenance Performance : A Field Study*, 1–28.

Charles Selvaraj. J, Aloysius. A, & Arockiam. L. 2009. *A Comparision of Proposed Cognitive weights for control structures and object oriented programming languages, Proceedings of International Conference on Advanced Computing* (ICAC09), 380-385.

D. Binkley. 2007. *Source code analysis: A road map. In 2007 Future of Software Engineering*, IEEE Computer Society, pages 104–119..

Dea, Carl., et.al., 2017. JavaFX 9 by Example. 3rd ed. New York: Apress

Debbarma, M. K., Debbarma, S., Debbarma, N., Chakma, K., & Jamatia, A. 2013. *A Review and Analysis of Software Complexity Metrics in Structural Testing*. *International Journal of Computer and Communication Engineering*, *2*(2), 129–133.

De Silva, D. I., Kodagoda, N., Kodituwakku, S. R., & Pinidiyaarachchi, A. J. 2015. Limitations of an object-oriented metric: Weighted complexity measure. *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, *2015–Novem*, 698–701. https://doi.org/10.1109/ICSESS.2015.7339153

Fathoni. 2009. Pengukuran Kualitas Perangkat Lunak Berdasarkan Kompleksitas Menggunakan Menggunakan Metode Function Point. Jurnal Sistem Informasi (JSI). Vol 1 (2).

Guru99, 2018. Learn Mccabe's Cyclomatic Complexity with Example. [Online] Tersedia di: [Di akses 23 September 2018]

Hermawan, Benny, Menguasai Java 2 & *Object Oriented Programming*. Yogyakarta: Andi, 2004.

Kurniawan, T. A. 2018. Pemodelan Use Case (UML): Evaluasi Terhadap Beberapa Kesalahan Dalam Praktik. Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) Vol. 5, No.1. Hal. 177–86.

Larman, C., 2005. Applying UML and Patterns. 3rd Edition. NJ: Prentice Hall

Kumar, N., Zadgaonkar, A. & Shukla, A., 2013. *Evolving a New Software Development Life Cycle Model SDLC-2013 with Client Satisfaction*. International Journal of Soft Computing and Engineering (IJSCE), p. 216.

Maheswaran, K., & Aloysius, A. 2018. *Cognitive Weighted Inherited Class Complexity Metric*. *Procedia Computer Science*, *125*, 297–304.

Mishra, A., Mishra, D. 2011. *Object-Oriented Inheritance Metrics in the Context of Cognitive Complexity. Fundamenta Informaticae, 110, 1–27*

Misra, S., & Akman, K. I. 2008. *Weighted class complexity: A measure of complexity for object oriented system*. *Journal of Information Science and Engineering*, *24*(6), 1689–1708.

Pawlak, R., et.al. 2015. *Spoon : A Library for Implementing Analyses and Transformations of Java Source Code*, 0-33.

Pressman, R., S. 2010. *Software Engineering A Practitioner's Approach*. 7th Ed. New York: McGraw-Hill.

Primadhana, F., 2015. Pembangunan Kakas Bantu Pembangkitan Program Dependence Graph Berbasis Java. Program Teknologi Informasi dan Ilmu Komputer, Univeristas Brawijaya

Rajnish K., Choudhary A.K., Agrawal A.M., 2010. *Inheritance Metrics For Object-Oriented Design. International Journal of Computer Science & Information Technology* (IJCSIT), Vol 2, No 6.

Rajnish, K., & Jakhar. 2016. *A cognitive measurement of Complexity and Comprehension for Object -oriented Code, International Journal of Computer, Electrical, Automation, Control and Information Engineering* 10(3): 643-650.

Schneidewind, N.F. 1987. *The State of Software Maintenance", IEEE Transactions on Software Engineering*, SE-13(3), p. 303-310.

Serna M., E., & Serna A., A. 2014. *Ontology for knowledge management in software maintenance*. *International Journal of Information Management*, *34*(5), 704–710.

Sharan, Kishori., 2015. *Learn JavaFX 8: Building User Experience and Interfaces with Java 8*. New York: Apress

Sommerville, I. 2011. *Software engineering*. 9th ed. London: Addison-Wesley.

S.R. Chidamber & C.F. Kemerer,. 1991. "Towards Metric Suite for Object-Oriented Design," Proc. Sixth ACM Conf. Object Oriented Programming Systems, Language, and Applications (OOPSLA), pp. 197-211.

Suarga. 2008. Dasar Pemrograman Komputer dalam Bahasa Java. Yogyakarta: Penerbit Andi

Thamburaj, F., & Aloysius, A. 2015. *Cognitive Weighted Method Hiding Factor Complexity Metric*, *4*(10), 272–279.

Wang. Y. 2002. *On Cognitive Informatics, IEEE International Conference on Cognitive Informatics*, 69-74.

Wang. Y., & Shao. J. 2003. *A new measure of software complexity based on cognitive Weights, IEEE Canadian Journal of Electrical and Computer Engineering*, 69-74.

Widyaningtyas, Y., Arwan, A., & Rusdianto, D. S. 2016. Perhitungan Ukuran Kompleksitas Fungsional Perangkat Lunak Dengan Metrik Function Point. *Teknologi*, *6*(1), 38.

Whitten, Jeffrey L, and Lonnie D. Bentley. 2007. Systems Analysis and Design Methods. McGraw-Hill.

Yadav, A., & Khan, R. A. 2012. *Development of Encapsulated Class Complexity Metric. Procedia Technology*, *4*, 754–760.