KAKAS BANTU PERHITUNGAN NILAI *REUSABILITY* BERDASARKAN RANCANGAN PERANGKAT LUNAK

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Rosita Budianti
NIM: 155150201111013

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

PENGESAHAN

KAKAS BANTU PERHITUNGAN NILAI *REUSABILITY* BERDASARKAN RANCANGAN PERANGKAT LUNAK

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Rosita Budianti
NIM: 155150201111013

Skripsi ini telah diuji dan dinyatakan lulus pada
22 April 2019
Telah diperiksa dan disetuji oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Fajar Pradana, S.ST, M.Eng	Eriq Muhammad Adams Jonemaro, S.T, M.Kom
NIP. 19871121 201504 1 004	NIP. 19850410 201212 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

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

PERNYATAAN ORISINALITAS

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

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

Malang, 22 April 2019

Rosita Budianti

NIM: 155150201111013

PRAKATA

Puji syukur kehadirat Allah SWT yang telah melimpahkan rahmat, taufiq serta hidayahNya dan senantiasa memberikan kelancaran sehingga laporan skripsi yang berjudul "Kakas Bantu Perhitungan Nilai Reusability Berdasarkan Rancangan Perangkat Lunak" ini dapat terselesaikan.

Selesai dan berhasilnya skripsi ini juga atas bantuan dan dukungan beberapa pihak kepada penulis. Oleh karena itu penulis ingin menyampaikan rasa hormat dan terimakasih kepada:

1. Bapak Fajar Pradana, S.ST., M.Eng. selaku pembimbing skripsi yang telah sabar dan ikhlas membimbing dan mengarahkan penulis sehingga dapat menyelesaikan skripsi ini
2. Bapak Eriq Muhammad Adams Jonemaro, S.T, M.Kom selaku pembimbing skripsi dan dosen penasihat akademik yang telah sabar dan ikhlas membimbing dan mengarahkan penulisan skripsi dan selalu memberi nasihat kepada penulis selama menempuh masa studi,
3. Bapak Tri Astoto Kurniawan, S.T., M.T, Ph.D. selaku Ketua Jurusan Teknik Informatika, dan Bapak Agus Wahyu Widodo, S.T, M.Cs selaku Ketua Program Studi Teknik Informatika,
4. Bapak dan Ibu orang tua penulis yang selalu memberikan dukungan, semangat, nasihat, doa, dan kasih sayang dalam mendidik penulis,
5. Sigit yang senantiasa membantu penulis selama masa studi dan pengerjaan, Arini, Nelli, Dilla, Fatimah, Yuni, dan sahabat-sahabat lain serta seluruh teman-teman Teknik Informatika A 2015 yang selalu memberi dorongan untuk semangat mengerjakan skripsi, teman-teman Teknik Informatika angkatan 2015 yang selalu memberi informasi sehingga membantu kelancaran masa studi dan pengerjaan skripsi ini.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih banyak kekurangan, sehingga kritik dan saran yang membangun sangat penulis harapkan. Akhir kata penulis berharap skripsi ini dapat memberikan manfaat bagi semu pihak yang menggunakannya.

Malang, 22 April 2019

Penulis

Email: rbsitaa@gmail.com

ABSTRAK

Rosita Budianti, Kakas Bantu Perhitungan Nilai *Reusability* Berdasarkan Rancangan Perangkat Lunak

Pembimbing: Fajar Pradana, S.ST., M.Eng. dan Eriq Muhammad Adams Jonemaro, S.T, M.Kom.

Reusability adalah salah satu kualitas perangkat lunak yang berarti kemampuan dapat digunakan kembali komponen perangkat lunak pada masalah lain untuk mengurangi waktu, biaya, dan sumber daya manusia. *Reusability* dapat diukur pada tahap perancangan sehingga rancangan perangkat lunak dapat diperbaiki sedini mungkin sebelum diimplementasikan. Penelitian ini dimaksudkan untuk mengembangkan sebuah kakas bantu untuk mengukur *reusability* pada tahap perancangan perangkat lunak secara otomatis. Pengukuran *reusability* dilakukan dengan teknik regresi berganda menggunakan metrik properti *object oriented design* yaitu *coupling, encapsulation,* dan *inheritance* berdasarkan metrik *Quality Model for Object-Oriented Design (*QMOOD)*,* karena korelasi antara *reusability* dan properti tersebut telah divalidasi dan menghasilkan nilai *acceptance* yang tinggi yaitu 95%. *Reusability* dihitung dari masukan *xml class diagram.* Dari hasil pembangunan, semua kebutuhan yang didefinisikan telah diuji dan menghasilkan valid. Selain itu juga untuk kebutuhan efisiensi menghasilkan sistem memiliki efisiensi lebih dari 80% yaitu 100% dapat melakukan perhitungan dan secara waktu lebih cepat dari perhitungan manual baik untuk berkas uji sederhana maupun kompleks. Sehingga sistem dapat berguna untuk membantu menghitung *reusability* dengan lebih efisien dibandingkan dengan menghitung manual.

Kata kunci: *reusability, metrik Quality Model for Object-Oriented Design (*QMOOD), *xml class diagram*

ABSTRACT

Rosita Budianti, *Tools of Reusability Measurement Based on Software Design*
Supervisors: Fajar Pradana, S.ST., M.Eng. dan Eriq Muhammad Adams Jonemaro, S.T, M.Kom.

Reusability is one of the software quality that means the ability to reuse software components on other problems to reduce time, cost, and human resources. Reusability can be measured at the design phase so the software design can be improved as early as possible before it is implemented. This research is intended to develop a tool for measuring reusability at the software design phase automatically. Measurement of reusability is done by multiple regression techniques using object oriented design property metrics, namely coupling, encapsulation, and inheritance based on the Quality Model for Object-Oriented Design (QMOOD) metric, because the correlation between reusability and property has been validated and results in high acceptance values of 95%. Reusability is calculated from the input class xml diagram. From the results of development, all defined requirements have been tested and produce valid. In addition, for efficiency requirements, the system has efficiency of more than 80%, tah is 100% can do calculations and time is faster than manual calculations for both simple and complex test filed. So the system can be useful to help calculate reusability more efficiently than manual counting.

*Keyword: reusability, Quality Model for Object-Oriented Design (*QMOOD) *metric*, *xml class diagram*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Rekayasa perangkat lunak merupakan disiplin teknik yang berkaitan dengan semua aspek pengembangan perangkat lunak dari tahap kebutuhan sistem hingga *maintenance* sistem ketika digunakan (Sommerville, 2011). Tahapan dari rekayasa perangkat lunak antara lain adalah analisis kebutuhan, perancangan sistem dari hasil analisis kebutuhan, implementasi dari hasil rancangan, kemudian pengujian sistem yang telah dibuat, dan perawatan sistem. Setiap tahapan pada rekayasa perangkat lunak penting untuk diperhatikan karena akan berdampak pada tahapan setelahnya. Selain itu juga untuk menjaga kualitas perangkat lunak yang dikembangkan sejak tahap awal. Salah satu atribut kualitas perangkat lunak yaitu *reusability* (Sommerville, 2011).

Reusability adalah kemampuan dapat digunakan kembali komponen perangkat lunak untuk mengurangi waktu, biaya, dan sumber daya manusia (Padhy, Singh dan Satapathy, 2018). Memperkirakan faktor *reusability* perlu dari awal siklus hidup pengembangan perangkat lunak karena dapat sangat mengurangi biaya pengembangan produk dan meningkatkan kepuasan pelanggan (Huda, Arya dan Khan, 2015). Kriteria *reusability* senantiasa mendukung perancang untuk meningkatkan rancangan perangkat lunak pada tahap awal proses pengembangan perangkat lunak. Mengukur nilai *reusability* membutuhkan alat ukur atau metrik perangkat lunak. Telah dilakukan beberapa penelitian tentang pengukuran *reusability* baik dari kode program maupun dari rancangan perangkat lunak. Pengukuran *reusability* pada kode program telah dilakukan oleh Washizaki yang berjudul "*Reusability Metrics for Program Source Code Written in C Language and Their Evaluation"* menggunakan metode *Goal-Question-Metric* (GQM) berhasil mengembangkan prosedur yang akurat untuk mengukur *reusability* (Washizaki et al., 2012). Kemudian pengukuran nilai *reusability* pada rancangan perangkat lunak telah dilakukan oleh Huda yang berjudul *"Quantifying Reusability of Object Oriented Design : A Testability Perspective" me*nggunakan metrik properti desain berorientasi objek yaitu *coupling, encapsulation,* dan *inheritance* telah divalidasi dan memiliki nilai *acceptance* yang tinggi yaitu 95% (Huda, Arya dan Khan, 2015).

Pengukuran *reusability* yang dilakukan secara manual membutuhkan tenaga dan waktu lebih banyak, terlebih jika perangkat lunak yang diukur cukup kompleks. Pengukuran manual juga memungkinkan terjadinya kesalahan perhitungan metrik. Oleh karena itu, dibutuhkan sistem sebagai alat pembantu dari perhitungan manual. Dengan sistem, pengukuran kualitas perangkat lunak menjadi efisien karena otomatis dan dapat digunakan secara sering tanpa membuang waktu yang berlebihan (Tomas, Escalona dan Mejias, 2013). Menurut Tomas, Escalona dan Mejias, keuntungan lain dengan adanya sistem pengukuran kualitas perangkat lunak yaitu mengurangi kesalahan perhitungan metrik dan mencapai akurasi yang lebih besar pada nilainya.

Berdasarkan uraian diatas, dalam penulisan ini diusulkan pengembangan "Kakas Bantu Perhitungan Nilai *Reusability* Berdasarkan Rancangan Perangkat Lunak". Dengan adanya sistem ini, pengembang dapat memperkirakan nilai *reusability* pada tahap perancangan yaitu dari *class diagram* secara otomatis. Pengukuran *reusability* pada penulisan ini menggunakan teknik regresi berganda dengan metrik properti desain berorientasi objek yaitu *coupling, encapsulation,* dan *inheritance* yang didapatkan dari paper Huda dkk (2015) dengan judul *"Quantifying Reusability of Object Oriented Design : A Testability Perspective"* sebagai acuan dalam melakukan penggalian kebutuhan. Dengan bantuan hasil pengukuran dari kakas bantu ini, tim pengembang dapat menghemat waktu dan juga perbaikan rancangan dapat dilakukan sebelum masuk ke tahap implementasi.

1.2 Rumusan masalah

Berdasarkan latar belakang yang telah dijelaskan diatas, rumusan masalah yang diangkat adalah :

1. Bagaimana hasil analisis kebutuhan, perancangan, implementasi, dan pengujian aplikasi kakas bantu perhitungan *reusability*?
2. Bagaimana efisiensi waktu aplikasi kakas bantu perhitungan *reusability*?

1.3 Tujuan

Berdasarkan latar belakang yang telah dijelaskan diatas, tujuan dari penelitian ini adalah :

1. Mengembangkan aplikasi kakas bantu untuk perhitungan nilai *reusability* berdasarkan rancangan perangkat lunak pada fase awal pengembangan.
2. Meningkatkan efisiensi dalam melakukan pengukuran kualitas *reusability* sebuah perangkat lunak.

1.4 Manfaat

Manfaat yang diharapkan dari penelitian ini adalah pengembang perangkat lunak dapat melakukan pengukuran awal kualitas *reusability* dalam perangkat lunak berorientasi objek terutama pada tahap perancangan yang memungkinkan rancangan dapat diterapkan kembali ke masalah baru tanpa banyak usaha ekstra.

1.5 Batasan masalah

Batasan masalah dalam pengembangan kakas bantu perhitungan nilai *reusability* ini adalah sebagai berikut:

1. Rancangan yang diukur dari perangkat lunak berorientasi objek berupa *class diagram* buatan pengembang lain yang bersifat *open source*.
2. Format struktur XML yang digunakan sebagai masukan adalah format *simple* dari Visual Paradigm.

3. Pengembangan sistem kakas bantu hanya menggunakan bahasa pemrograman Java.
4. Pengembangan sistem kakas bantu hanya berbasis *dekstop application.*
5. Penelitian ini hanya menghitung nilai *reusability* dan tidak memberikan batasan untuk nilai yang baik dan buruk.

1.6 Sistematika pembahasan

1. **Bab 1 PENDAHULUAN**

   Pada bab pendahuluan ini menjelaskan tentang latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah, dan sistematika pembahasan.

2. **Bab 2 LANDASAN KEPUSTAKAAN**

   Pada bab landasan kepustakaan ini menguraikan dasar-dasar teori dan riset penelitian yang telah dilakukan untuk melandasi penulisan dan penelitian pembangunan aplikasi kakas bantu perhitungan nilai *reusability*.

3. **Bab 3 METODOLOGI PENELITIAN**

   Pada bab metodologi penelitian ini menjelaskan metodologi penelitian antara lain: studi literatur, analisis kebutuhan, perancangan dan implementasi, pengujian, serta kesimpulan dan saran. Sedangkan untuk pengembangan sistem menggunakan *SDLC Model Waterfall.*

4. **Bab 4 ANALISIS KEBUTUHAN**

   Pada bab analisis kebutuhan ini berisi identifikasi aktor, spesifikasi kebutuhan yaitu kebutuhan fungsional dan non-fungsional, serta lingkungan pengembangan sistem.

5. **Bab 5 PERANCANGAN DAN IMPLEMENTASI SISTEM**

   Pada bab perancangan ini berisi rancangan sistem yang akan dikembangkan terdiri dari perancangan arsitektur, perancangan algoritme, dan perancangan antarmuka pengguna. Selain rancangan juga ada bagian implementasi sistem yang berisi spesifikasi sistem, lingkungan pengembangan, sampel kode program, dan hasil implementasi antarmuka.

6. **Bab 6 PENGUJIAN SISTEM DAN PEMBAHASAN HASIL**

   Pada bab pengujian ini berisi proses pengujian dari sistem yang telah dikembangkan sehingga sistem dipastikan layak digunakan calon pengguna dan juga memaparkan pembahasan hasil dari pengujian berkas uji.

7. **Bab 7 PENUTUP**

   Pada bab penutup ini memberikan kesimpulan dari hasil penelitian dan memberikan saran untuk penelitian selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Penelitian sebelumnya yang telah dilakukan untuk mengukur nilai *reusability* yaitu berjudul "*Reusability Metrics for Program Source Code Written in C Language and Their Evaluation".* Pengukuran nilai *reusability* yang dilakukan yaitu pada kode sumber program bahasa C dengan menggunakan metode *Goal-Question-Metric* (GQM). Penelitian tersebut mengembangkan prosedur untuk secara akurat mengevaluasi validitas dan keefektifan metrik *reusability* dan untuk menganalisis besarnya efek individu pada *reusability* yang sebenarnya. Hasil pengukuran tingkat *reusability* berdasarkan tingkat penggunaan kembali baris, tingkat penggunaan kembali fungsi, dan tingkat penggunaan kembali file dari projek yang sudah ada untuk projek baru. Hasil dari penelitiannya adalah satu set metrik *reusability*, yang telah dievaluasi efektivitasnya dan yang dapat digunakan untuk kode sumber program bahasa C, diusulkan bersama dengan evaluasi besarnya efek individu mereka pada *reusability* (Washizaki et al., 2012).

Penelitian lain tentang pengukuran *reusability* yaitu berjudul *"Quantifying Reusability of Object Oriented Design : A Testability Perspective".* Pengukuran nilai *reusability* dilakukan pada rancangan perangkat lunak yaitu *class diagram*. Model yang dikembangkan untuk pengukuran *reusability* berdasarkan korelasi antara properti desain berorientasi objek dengan *reusability.* Dalam perhitungannya menggunakan teknik regresi berganda dengan metrik-metrik properti desain berorientasi objek yaitu DCC untuk kopling, MFA untuk pewarisan, dan DAM untu enkapsulasi. Hasil penelitian ini adalah validasi model yang dikembangkan untuk mengukur *reusability* mempunyai nilai *acceptance* yang tinggi yaitu 95% (Huda, Arya dan Khan, 2015).

Selain penelitian tentang pengukuran *reusability,* dilakukan peninjauan juga pada penelelitian sebelumnya yang berjudul "Kakas bantu Perhitungan Nilai Kopling Menggunakan Metrik Cognitive Weighted Coupling Between Object". Penelitian tersebut melakukan pengembangan sistem untuk membantu perhitungan kopling. Perhitungan kopling dilakukan pada kode sumber program berbahasa java. Penelitian tersebut juga melakukan perbandingan nilai kopling menggunakan CBO dan CWCBO serta perhitungan manual dan juga otomatis menggunakan sistem. Pengujian akurasi dengan membandingkan hasil perhitungan secara menual dan dengan sistem pada lima program diperoleh hasil akurasi sebesar 100% yaitu lebih besar dari pada nilai batas minimalnya yaitu 90% (Ubaidillah, Pradana dan Priyambadha, 2017).

Penelitian yang akan dilakukan saat ini adalah mengembangkan kakas bantu perhitungan *reusability* pada tahap perancangan. Berdasarkan penelitian sebelumnya, dengan bantuan sistem diharapkan hasil akurasi perhitungan tinggi. Perhitungan dilakukan pada tahap awal yaitu perancangan agar dapat sedini mungkin perubahan dilakukan pada rancangan sebelum diimplementasikan menjadi kode program.

2.2 Rekayasa Perangkat Lunak

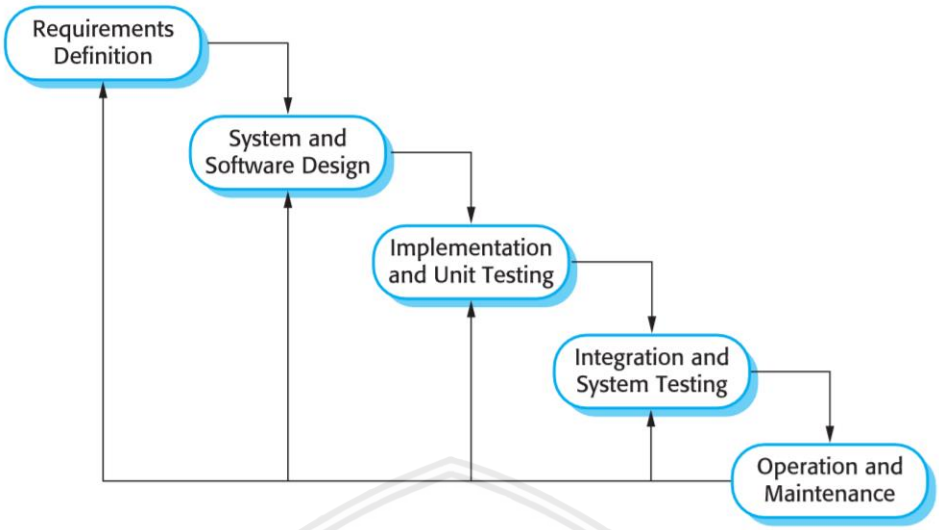
Rekayasa perangkat lunak merupakan pendekatan sistematis untuk produksi perangkat lunak yang mempertimbangkan masalah jadwal, biaya, dan ketergantungan praktis, serta kebutuhan pelanggan dan produsen (Sommerville, 2011). Perangkat lunak tidak hanya program saja, tetapi juga dokumen. Bagaimana pendekatan sistematis benar-benar dilaksanakan sangat bervariasi tergantung pada organisasi yang mengembangkan perangkat lunak, jenis perangkat lunak, dan pihak-pihak yang terlibat dalam proses produksi. Kegiatan mendasar dari rekayasa perangkat lunak yaitu spesifikasi sistem, pengembangan sistem, validasi sistem, dan evolusi sistem. Ada dua frasa kunci dalam rekayasa perangkat lunak :

1. Disiplin teknik : *Engineers* menerapkan metode, teori, dan alat yang sesuai, namun menggunakannya secara selektif serta selalu mencoba mencari solusi dari setiap masalah, termasuk ketika tidak ada metode dan teori yang berlaku.
2. Semua aspek pengembangan perangkat lunak : tidak hanya berhubungan dengan proses teknik pada pengembangan perangkat lunak, akan tetapi juga termasuk pada kegiatan manajemen proyek perangkat lunak dan pengembangan metode, teori, serta alat yang mendukung proses produksi.

2.3 Pengembangan Perangkat Lunak

2.3.1 *Model Software Development Life Cycle*

Software development life cycle (SDLC) yang digunakan pada pengembangan perangkat lunak yang dilakukan oleh penulis kali ini menggunakan SDLC model *waterfall.* Dikenal sebagai model *waterfall* karena bentuknya yang menyerupai air terjun seperti deretan urut dari satu fase ke fase yang lain seperti diilustrasikan dalam Gambar 2.1. Model *waterfall* pada prinsipnya harus menjadwalkan serta merencanakan semua kegiatan pengembangan sebelum mulai mengerjakannya. Dokumentasi harus dibuat pada setiap fase karena dalam tahapan ini memberi informasi satu sama lain sehingga fase berikutnya tidak boleh dimulai sampai fase sebelumnya selesai. Model *waterfall* seharusnya hanya digunakan ketika kebutuhan telah diketahui dengan baik diawal dan tidak mungkin ada perubahan secara signifikan selama proses pengembangan sistem (Sommerville, 2011). Pada pengembangan kakas bantu perhitungan *reusability* ini kebutuhan-kebutuhan telah diketahui jelas diawal dan sangat kecil kemungkinan berubah selama tahap pengembangan sehingga model SDLC yang digunakan adalah model *waterfall.*



Gambar 2.1 *Waterfall Model*

Sumber: (Sommerville, 2011)

Berikut ini tahapan pada *SDLC Waterfall Model* yaitu (Sommerville, 2011):

1. Analisis dan definisi kebutuhan
   Dengan berkonsultasi dengan pengguna menentukan layanan, batasan, dan tujuan sistem secara rinci.
2. Perancangan sistem
   Kebutuhan baik perangkat keras maupun perangkat lunak yang telah ditentukan dialokasikan untuk membuat keseluruhan arsitektur sistem. Rancangan melibatkan deskripsi abstraksi dan identifikasi sistem fundamental dan hubungannya.
3. Implementasi dan pengujian unit
   Rancangan perangkat lunak diwujudkan sebagai unit program atau serangkaian program. Verifikasi jika setiap unit telah memenuhi spesifikasinya dilakukan pada pengujian unit.
4. Pengujian integrasi dan sistem
   Memastikan seluruh kebutuhan telah dipenuhi dengan melakukan integrasi masing-masing unit program kemudian diuji sebagai sistem yang utuh.
5. Pemeliharaan
   Tahap terpanjang dalam siklus hidup namun tidak harus ada. Tahap ini melakukan koreksi kesalahan yang pada tahap awal pengembangan tidak ditemukan, meningkatkan implementasi unit sistem, serta meningkatkan layanan sistem ketika kebutuhan baru didapatkan.

2.3.2 Pendekatan Berorientasi Objek

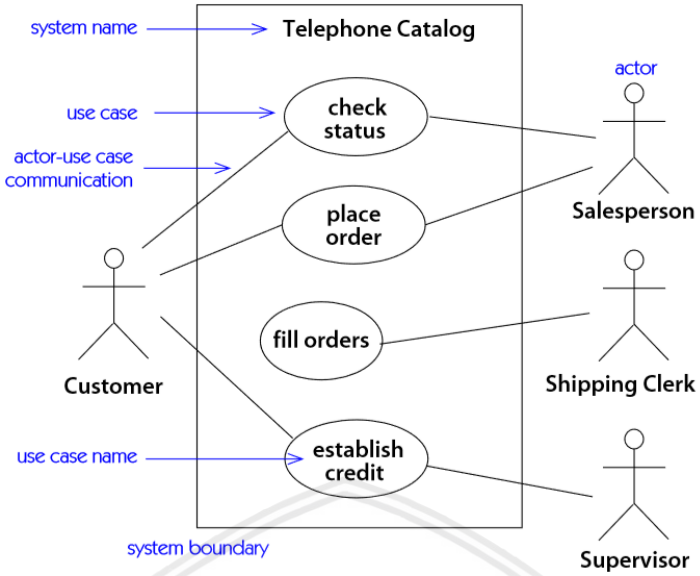
Pendekatan berorientasi objek untuk pembangunan *software* pertama kali diusulkan pada akhir 1960-an dan butuh waktu hampir 20 tahun untuk menjadi yang banyak digunakan (Pressman, 2001). Pendekatan berorientasi objek ini dibuat dengan dasar interaksi antar objek dan operasi apa yang ada pada *state* yang mengatur *state* tersebut (Sommerville, 2011). Pendekatan berorientasi objek menyebabkan penggunaan kembali (komponen program), dan penggunaan kembali mempengaruhi pengembangan perangkat lunak yang lebih cepat dan program berkualitas tinggi (Pressman, 2001). Perangkat lunak berorientasi objek lebih mudah untuk *maintenance* karena strukturnya secara inheren dipisahkan. Hal ini menyebabkan lebih sedikit efek samping ketika perubahan harus dilakukan dan lebih sedikit frustasi bagi *engineer* perangkat lunak dan juga pelanggan. Selain itu, sistem berorientasi objek lebih mudah untuk beradaptasi dan lebih mudah untuk skala (seperti sistem besar dapat dibuat dengan merangkai subsistem yang dapat digunakan kembali).

2.3.3 Pemodelan Berorientasi Objek

Unified modeling language (UML) merupakan bahasa grafis yang digunakan dalam pengembangan berorientasi objek yang mencakup beberapa jenis model sistem untuk melakukan spesifikasi, visualisasi, konstruksi, dan artifak dari proses dokumentasi dari suatu perangkat lunak (Sommerville, 2011; Booch, Jacobson dan Rumbaugh, 1999). UML telah menjadi standar *de facto* untuk pemodelan berorientasi objek. UML mempunyai banyak tipe diagram yang mendukung pembuatan banyak jenis model sistem. Pada penulisan kali ini penulis menggunakan macam diagram UML antara lain:

2.3.3.1 *Use Case Diagram*

Use case diagram menunjukkan interaksi antara sistem dengan lingkungannya (Sommerville, 2011). Pemodelan *use case* banyak digunakan sebagai pendukung elisitasi kebutuhan. Use case bisa digunakan sebagai skenario sederhana yang memberi gambaran tentang apa yang diharapkan oleh pengguna dari suatu sistem. Setiap use case menggambarkan tugas yang menunjukkan interaksi suatu sistem dengan eksternal. Use case ditampilkan sebagai elips dan aktor yang terlibat diwakili sebagai figur tongkat untuk menggambarkan bentuk yang paling sederhana. Contoh *use case diagram* adalah seperti dalam Gambar 2.2.



Gambar 2.2 *Use Case Diagram*

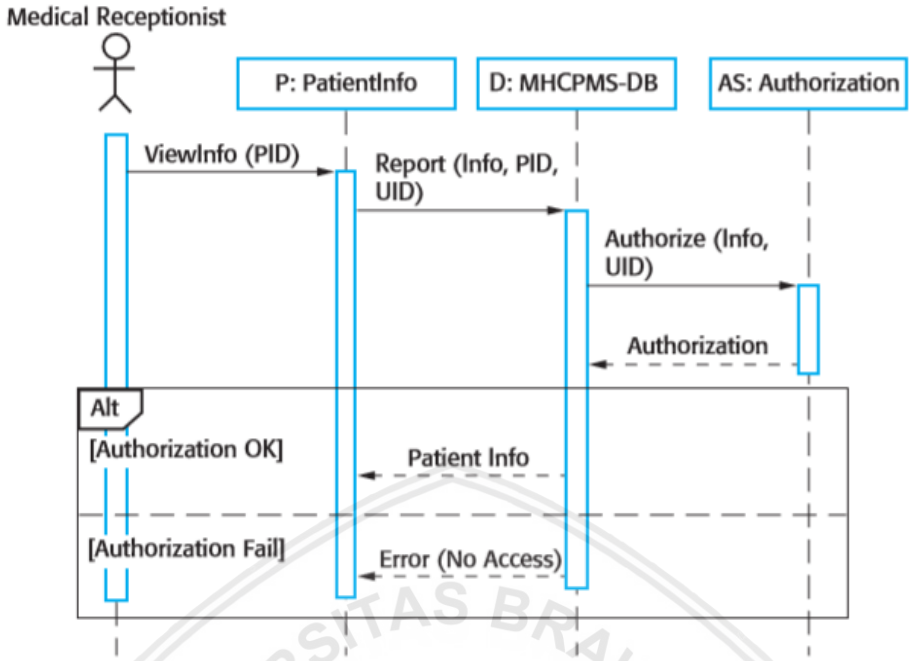
Sumber: (Booch, Jacobson dan Rumbaugh, 1999)

2.3.3.2 *Sequence Diagram*

Sequence diagram digunakan untuk memodelkan hubungan antara aktor dengan objek dan interaksi antar objek itu sendiri dalam suatu sistem (Sommerville, 2011). UML mempunyai sintaks yang kaya untuk *sequence diagram*, sehingga memungkinkan untuk berbagai jenis interaksi yang berbeda dimodelkan. *Sequence diagram* menggambarkan urutan interaksi yang terjadi selama *use case.* Aktor dan objek yang terlibat dituliskan pada sepanjang bagian atas diagram dengan garis vertikal putus-putus dari aktor dan objek ke bawah. Hubungan antar objek digambarkan dengan panah beranotasi. Persegi panjang yang ada pada garis vertikal putus-putus menunjukkan *life-line* dari objek. Anotasi yang ada pada panah menunjukkan panggilan ke objek, parameter objek, serta nilai balikannya. Sebuah kotak yang dinamai *alt* digunakan untuk menggambarkan kondisi alternatif. Contoh dari *sequence diagram* adalah seperti dalam Gambar 2.3.

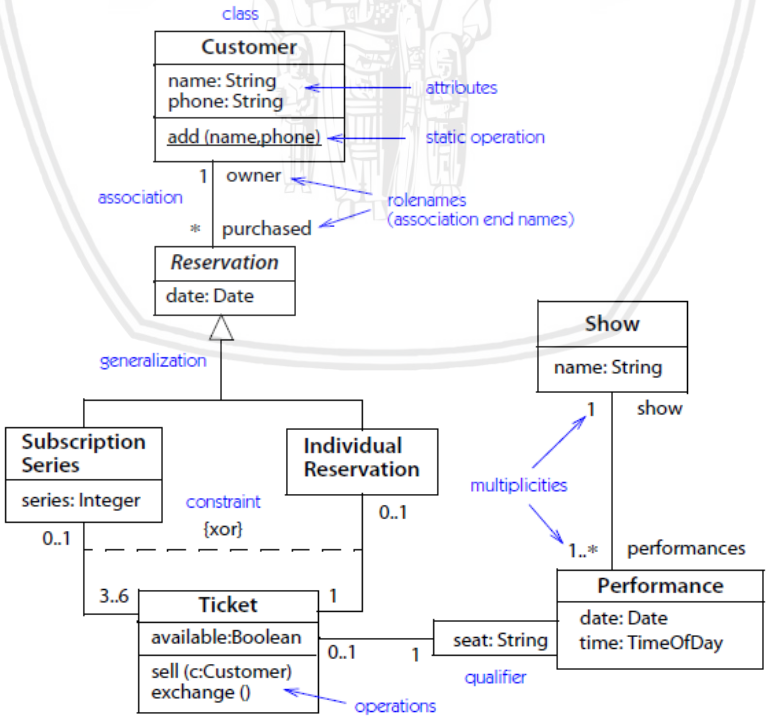
2.3.3.3 *Class Diagram*

Class diagram digunakan pada pengembangan sistem dengan pendekatan berorientasi objek untuk menggambarkan kelas serta asosiasi antara kelas-kelas dalam suatu sistem (Sommerville, 2011). Kelas objek sebagai gambaran dari satu jenis objek pada sistem, dan asosiasi menunjukkan bahwa ada hubungan antar kelas. Jika berasosiasi, setiap kelasnya mungkin harus memiliki pengetahuan tentang kelas yang terkait. Ketika tahap awal proses rekayasa perangkat lunak mengembangkan model, objek mewakili sesuatu di dunia nyata, ketika implementasi dikembangkan biasanya perlu menentukan objek tambahan implementasi yang digunakan untuk menyediakan fungsionalitas sistem yang dibutuhkan. Contoh dari *class diagram* adalah seperti dalam Gambar 2.4.



Gambar 2.3 *Sequence Diagram*

Sumber: (Sommerville, 2011)



Gambar 2.4 *Class Diagram*

Sumber: (Booch, Jacobson dan Rumbaugh, 1999)

Beberapa aturan dalam *class diagram:*

1. Nama kelas objek ada pada bagian atas.
2. Atribut kelas pada bagian tengah. Harus disertakan nama atribut dan, secara opsional, tipenya.
3. Operasi (metode yang disebut di Java dan bahasa pemrograman OO lainnya) yang terkait dengan kelas objek berada di bagian bawah persegi panjang.

2.4 Kualitas Perangkat Lunak

Kualitas perangkat lunak adalah kesesuaian produk dengan spesifikasi produk yang rinci. Asumsi yang mendasari adalah bahwa produk dapat sepenuhnya ditentukan dan prosedur dapat ditetapkan yang dapat memeriksa produk yang diproduksi terhadap spesifikasinya (Sommerville, 2011). Penilaian kualitas perangkat lunak adalah proses subjektif dimana tim manajemen kualitas harus menggunakan penilaian mereka untuk memutuskan apakah tingkat kualitas yang dapat diterima telah tercapai. Tim manajemen kualitas harus mempertimbangkan apakah perangkat lunak sesuai untuk tujuan yang dimaksudkan.

Kualitas perangkat lunak tidak hanya mengenai apakah fungsi perangkat lunak telah diimplementasikan dengan benar, tetapi juga tergantung pada atribut sistem non-fungsional (Sommerville, 2011). Terdapat 15 atribut kualitas perangkat lunak yang penting, seperti ditunjukkan pada Tabel 2.1. Perencanaan kualitas harus menentukan atribut kualitas yang paling penting untuk perangkat lunak yang sedang dikembangkan. Asumsi yang yang mendasari manajemen kualitas perangkat lunak adalah bahwa kualitas perangkat lunak secara langsung berkaitan dengan kualitas proses pengembangan perangkat lunak.

Tabel 2.1 Atribut kualitas perangkat lunak

Sumber: (Sommerville, 2011)

Safety	*Understandability*	*Portability*
Security	*Testability*	*Usability*
Reliability	*Adaptability*	*Reusability*
Resilience	*Modularity*	*Efficiency*
Robustness	*Complexity*	*Learnability*

2.5 *Reusability* Perangkat Lunak

Reusability perangkat lunak adalah kemampuan untuk digunakan kembali komponen perangkat lunak yang ada atau mudah tersedia untuk mengurangi waktu dan biaya untuk komputasi dan sumber daya manusia (Padhy, Singh dan Satapathy, 2018). *Reusability* merupakan salah satu atribut kualitas perangkat lunak (Sommerville, 2011). *Reusability* memainkan peran penting dalam penilaian kualitas perangkat lunak pada waktu desain dan juga bertindak sebagai dasar untuk menemukan indeks *testability* untuk peringkat proyek industri (Huda, Arya dan Khan, 2015). Kriteria *reusability* senantiasa mendukung perancang untuk meningkatkan rancangan perangkat lunak pada tahap awal proses pengembangan perangkat lunak. Fase desain memiliki dampak langsung pada biaya dan upaya pengujian produk dan memainkan peran tulang punggung dari produk apa pun.

Desain dan pengembangan berorientasi objek menjadi paradigma pilihan bagi banyak pengembang perangkat lunak dan seiring berjalannya waktu pendekatan berorientasi objek menggantikan pendekatan klasik (Pressman, 2001). Salah satu keuntungan dari desain berorientasi objek adalah dukungan untuk penggunaan kembali perangkat lunak, yang dapat melalui penggunaan kembali kelas maupun melalui hubungan warisan. Penggunaan kembali (*reuse)* mempengaruhi pengembangan perangkat lunak yang lebih cepat dan program berkualitas tinggi (Pressman, 2001). Memperkirakan faktor reusabilitas pada awal siklus hidup pengembangan dapat sangat mengurangi biaya pengembangan produk secara keseluruhan dan meningkatkan kepuasan pelanggan (Huda, Arya dan Khan, 2015). Jika nilai *reusability* sudah diketahui sejak tahap perancangan maka rancangan dapat segera diperbaiki sebelum masuk ke tahap implementasi.

2.6 Properti Desain Berorientasi Objek

Desain dan pengembangan berorientasi objek dalam lingkungan pengembangan perangkat lunak menjadi paradigma yang sangat disukai untuk desain sistem berskala besar. Teknologi ini menawarkan dukungan untuk menghadirkan hasil produk perangkat lunak dengan kualitas yang lebih tinggi dan biaya pemeliharaan yang lebih rendah (Huda, Arya dan Khan, 2015). Dimana keuntungan utama berorientasi objek salah satunya adalah dukungannya untuk reusabilitas (penggunaan kembali) perangkat lunak, yang dapat dicapai baik melalui penggunaan kembali kelas yang sederhana pada *library* atau melalui warisan di antara hubungan.

Ada beberapa kualitas penting yang diakui sebagai dasar kualitas internal desain berorientasi objek yang mendukung dalam konteks pengukuran. Ide-ide ini secara signifikan termasuk properti desain yaitu enkapsulasi, pewarisan *(inheritance)*, kopling, dan kohesi. Enkapsulasi adalah mekanisme untuk mewujudkan abstraksi data dan penyembunyian informasi dengan menyembunyikan spesifikasi internal suatu objek. Pewarisan *(inheritance)* adalah distribusi operasi dan atribut antar kelas. Kopling menunjukkan hubungan atau interdependensi antar modul. Kohesi mengacu pada keandalan internal di dalam komponen desain.

2.7 Metrik Desain Berorientasi Objek

Object-oriented metrics (OOM) berperan penting dalam industri perangkat lunak karena hampir semua proyek mengembangkan perangkat lunak dengan konsep berorientasi objek (Padhy, Singh dan Satapathy, 2018). Mengukur kualitas dan kuantitas adalah tugas yang menantang bagi pengembang. Arti dari metrik adalah kuantitas yang terukur. Banyak metrik perangkat lunak telah dikembangkan dan digunakan untuk prediksi kesalahan, yang dapat memperkirakan ukuran, kompleksitas, kohesi, dan kopling. Beberapa metrik yang dapat dihitung dari informasi desain adalah *Quality Model for Object-Oriented Design* (QMOOD) seperti yang ada pada Tabel 2.2 (Bansiya dan Davis, 2002).

Tabel 2.2 Metrik *Quality Model for Object-Oriented Design*

Metric	Nama	Deskripsi
DSC	*Design Size in Classes (Design Size Metric)*	Metrik ini adalah hitungan dari jumlah total kelas dalam desain.
NOH	*Number of Hierarchies (Hierarchies Metric)*	Metrik ini adalah hitungan jumlah kelas hierarki dalam desain.
ANA	*Average Number of Ancestors (Abstraction Metric)*	Metrik ini menunjukkan jumlah rata-rata kelas dari mana kelas mewarisi informasi.
DAM	*Data Access Metric (Encapsulation Metric)*	Metrik ini adalah rasio jumlah atribut *private* dan *protected* dengan jumlah total atribut yang dideklarasikan di kelas. Nilai tinggi untuk DAM diinginkan. (Kisaran 0 hingga 1)
DCC	*Direct Class Coupling (Coupling Metric)*	Metrik ini menghitung jumlah kelas yang berbeda yang berelasi langsung dengan kelas. Metrik ini mencakup kelas-kelas yang secara langsung berelasi dengan deklarasi atribut dan pengiriman pesan (parameter) dalam *method*.
MOA	*Measure of Aggregation (Compotition Metric)*	Metrik ini mengukur tingkat hubungan bagian-utuh yang diwujudkan dengan menggunakan atribut.
MFA	*Measure of Functional Abstraction (Inheritance Metric)*	Metrik ini adalah rasio jumlah *method* yang diwarisi oleh kelas dengan jumlah *method* yang dapat diakses oleh *method* anggota kelas. (Kisaran 0 hingga 1)

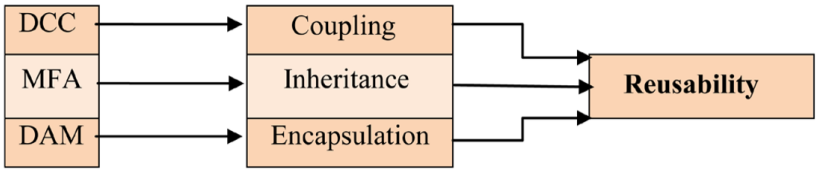
Tabel 2.2 Metrik *Quality Model for Object-Oriented Design* (lanjutan)

CIS	*Class Interface Size (Messaging Metric)*	Metrik ini mengukur jumlah metode publik di kelas.
NOM	*Number of Methods (Complexity Metric)*	Metrik ini mengukur jumlah total metode di kelas.

2.8 Metrik *Reusability*

Terdapat beberapa metrik berorientasi objek yang populer dan membahas bagaimana memprediksi *reusability,* namun pada penulisan kali ini penulis menyebutkan sebagian saja. Model yang didasarkan pada rangkaian metrik CK yaitu tiga metrik Depth of Response for a Class (RFC), Inheritance Tree (DIT) dan Weighted Methods per Class (WMC). Ketiga metrik ini memungkinkan untuk memperkirakan faktor reusabilitas dari kode perangkat lunak (Nair dan Selvarani, 2010). Model selanjutnya adalah algoritma *Self Organizing Map* (SOM) untuk memperkirakan *reusability* komponen yang berbeda dalam sistem perangkat lunak berorientasi objek. Input utama terdiri dari metrik CK yang telah disediakan untuk algoritma SOM untuk mengevaluasi jumlah *reusability* untuk komponen yang berbeda (Goyal dan Gupta, 2014). Peneliti tidak hanya menggunakan metrik CK untuk memperkirakan ukuran dan kualitas, tetapi juga menggunakannya untuk mengukur algoritma pembelajaran mesin. Selanjutnya jenis kerangka kerja untuk metrik menggunakan metrik *cyclomatic complexity McCabe*, *regularity metric, Halstead software science indicator*, metrik frekuensi *reuse,* dan metrik kopling untuk memperkirakan *reusability* (Sandhu, Kaur dan Singh, 2018). Kerangka tersebut diimplementasikan pada lingkungan MATLAB 7.4. Perkiraan *reusability* agak sulit karena penelitian ini belum konkrit dan masih berlanjut.

Matrik lainnya yaitu menghitung *reusability* menggunakan *multivariate linier metric* model regresi berganda yang menunjukkan hubungan antara *reusability,* properti *Object-Oriented Design* (OOD) dan metrik (Huda, Arya dan Khan, 2015). Nilai-nilai dari metrik desain ini dapat diidentifikasi dengan metrik *class diagram* pada tahap perancangan. Metrik yang teridentifikasi yaitu enkapsulasi, pewarisan *(inheritance)*, dan kopling berperan sebagai variabel independen sementara *reusability* sebagai variabel dependen yang ditunjukkan dalam Gambar 2.5.



Gambar 2.5 Atribut kualitas perangkat lunak

Sumber: (Huda, Arya dan Khan, 2015)

Pada penelitian ini penulis menggunakan *multivariate linier metric* model regresi berganda untuk mengukur *reusability* pada tahap perancangan melalui

class diagram. Model metric yang menunjukkan korelasi dari properti OOD dengan *reusability* telah divalidasi berdasarkan *spearman's rank corellation* dapat diterima dengan tingkat kepercayaan yang tinggi yaitu 95% (Huda, Arya dan Khan, 2015). Persamaan 2.1 menunjukkan teknik regresi berganda yang telah dipilih untuk mengukur faktor *reusability*:

$$Reusability = \alpha 0 \pm \alpha 1 * Coupling \pm \alpha 2 * Inheritance \pm \alpha 3 * Encapsulation$$

Dengan menggunakan nilai-nilai perangkat lunak matematika "SPSS" (*Statistical Package for the Social Sciences*) dari semua metrik desain, intersepsi, dan koefisien dari masing-masing metrik dihitung. Atas dasar teknik ini, model *reusability* regresi berganda telah dikembangkan dalam Persamaan 2.2 :

$$Reusability = -37{,}111 + 3{,}973 * Coupling + 32{,}500 * Inheritance + 20{,}709 * Encapsulation \quad (2.2)$$

Pengukuran *coupling* berdasarkan metrik DCC, *inheritance* berdasarkan metrik MFA, dan *encapsulation* berdasarkan metrik DAM yang lebih detail telah dijelaskan oleh Fujita dan Pisanelli (Fujita & Pisanelli, 2007)

2.8.1 *Direct Class Coupling* (DCC)

Metrik ini menghitung jumlah kelas yang berbeda yang berelasi langsung dengan kelas. Metrik ini mencakup kelas-kelas yang secara langsung berelasi dengan deklarasi atribut dan pengiriman pesan (parameter) dalam *method*. Jika diterapkan pada rancangan *class diagram* maka diukur rata-rata dari jumlah DCC per kelas dibagi dengan jumlah seluruh kelas yang ada. Jika dirumuskan maka seperti dalam Persamaan 2.3 (Fujita & Pisanelli, 2007):

$$\text{DCC } (class\ diagram) = \frac{\sum dcc}{\sum c} \quad (2.3)$$

Keterangan:

dcc = DCC masing-masing kelas (jumlah kelas yang berbeda yang berelasi langsung dengan kelas)

c = Kelas seluruhnya

2.8.2 Measure of Functional Abstraction (MFA)

Metrik ini adalah rasio jumlah *method* yang diwarisi oleh kelas dengan jumlah *method* yang dapat diakses oleh *method* anggota kelas. Konstruktor dan java.lang.Object (sebagai induk) diabaikan. (Kisaran 0 hingga 1). Jika diterapkan pada rancangan *class diagram* maka diukur rata-rata dari jumlah MFA per kelas dibagi dengan jumlah seluruh kelas yang memiliki setidaknya satu *method*. Jika dirumuskan maka seperti dalam Persamaan 2.4 (Fujita & Pisanelli, 2007):

$$\text{MFA } (class\ diagram) = \frac{\sum mfa}{\sum cmfa} \quad (2.4)$$

Keterangan:

mfa = MFA masing-masing kelas (rasio jumlah *method* yang diwarisi oleh kelas dengan jumlah *method* yang dapat diakses oleh *method* anggota kelas)

cmfa = kelas yang memiliki setidaknya satu *method*

2.8.3 Data Access Metric (DAM)

Metrik ini adalah rasio jumlah atribut *private* dan *protected* dengan jumlah total atribut yang dideklarasikan di kelas. (Kisaran 0 hingga 1). Jika diterapkan pada rancangan *class diagram* maka diukur rata-rata dari jumlah DAM per kelas dibagi dengan jumlah seluruh kelas yang memiliki setidaknya satu atribut. Jika dirumuskan maka seperti dalam Persamaan 2.5 (Fujita & Pisanelli, 2007):

$$\text{DAM } (class\ diagram) = \frac{\sum dam}{\sum cdam} \quad (2.5)$$

Keterangan :

dam = DAM masing-masing kelas (rasio jumlah atribut *private* dan *protected* dengan jumlah total atribut yang dideklarasikan di kelas)

cdam = kelas yang setidaknya memiliki satu atribut

2.9 Pengujian Perangkat Lunak

Pengujian yaitu tahap yang dimaksudkan untuk menunjukkan bahwa suatu program melakukan sesuai dengan tujuan serta untuk menemukan cacat yang ada pada program sebelum digunakan (Sommerville, 2011). Proses pengujian memiliki dua tujuan berbeda yaitu tujuan pertama mengarah ke pengujian validasi, di mana diharapkan perangkat lunak untuk berfungsi dengan benar menggunakan kumpulan kasus uji tertentu yang mencerminkan penggunaan yang diharapkan perangkat lunak. Tujuan kedua mengarah ke pengujian cacat, di mana kasus uji dirancang untuk mengekspos cacat. Kasus uji dalam pengujian cacat dapat secara sengaja tidak jelas dan tidak perlu mencerminkan bagaimana perangkat lunak biasanya digunakan. Tentu saja, tidak ada batas yang pasti antara kedua pendekatan ini untuk diuji. Selama pengujian validasi, penguji akan menemukan cacat dalam sistem; selama pengujian cacat, beberapa tes akan menunjukkan bahwa program memenuhi kebutuhan.

Jenis-jenis pengujian perangkat lunak yang dilakukan pada penelitian ini adalah sebagai berikut:

2.9.1 Pengujian Unit

Pengujian unit ini berfokus pada usaha untuk verifikasi unit terkecil dari perancangan perangkat lunak yang berupa modul atau komponen perangkat lunak (Pressman, 2010). Menggunakan rancangan level komponen sebagai

panduan dan jalur kontrol penting diuji karena untuk menemukan kesalahan pada batas modul. Semua jalur independen (jalur dasar) melalui struktur kontrol dilakukan untuk memastikan bahwa semua pernyataan pada modul telah dijalankan minimal satu kali. Untuk menemukan kesalahan pada perhitungan yang salah, perbandingan yang salah, maupun aliran kontrol yang tidak tepat diperlukan merancang kasus uji. Pengujian unit berorientasi pada *white-box,* dan dilakukan secara paralel untuk beberapa komponen.

Setelah *source code* dikembangkan, ditinjau, dan diverifikasi untuk korespondensi dengan desain tingkat komponen, desain kasus uji unit dimulai. Setiap kasus uji harus digabungkan dengan satu set ekspektasi hasil. Karena komponen bukan merupakan program yang berdiri sendiri, maka *driver* dan / atau *stub* harus dibuat untuk setiap pengujian unit. *Driver* tidak melebihi program utama yang menerima data kasus uji, meneruskan data tersebut ke komponen (untuk diuji), dan mencetak hasil yang relevan. *Stub* berfungsi untuk menggantikan modul yang lebih rendah (disebut oleh) komponen yang akan diuji.

2.9.1.1 Pengujian *White-Box*

Pengujian *white-box* yaitu pendekatan untuk pengujian program di mana pengujian didasarkan pada pengetahuan tentang struktur program dan komponennya (Sommerville, 2011). Akses ke kode sumber sangat penting untuk pengujian *white-box*.

Aspek-aspek pengujian *white-box* (Kurniawan, 2015) :

a. Memastikan semua jalur algoritme telah diuji setidaknya satu kali
b. Menguji keseluruhan keputusan lojik (*true* atau *false*)
c. Menjalankan keseluruhan loop pada batasan yang telah ditentukan
d. Memvalidasi struktur data internal

Pada pengujian *white-box* terdapat dua jenis yaitu pengujian struktur kontrol *(control structure testing)* dan pengujian jalur dasar (*basis path testing)* (Kurniawan, 2015). Pengujian struktur kontrol sebagai pelengkap untuk pengujian jalur dasar dimana terdapat pengujian kondisi dan juga pengujian *loop.* Pengujian jalur dasar adalah pengujian *white-box* yang dibuat berdasarkan ukuran tingkat kompleksitas dari algoritme hasil perancangan.

2.9.2 Pengujian Integrasi

Pengujian integrasi merupakan teknik sistematis untuk membuat struktur program yang sementara pada saat melakukan uji untuk menemukan kesalahan terkait dengan *interfacing* (Pressman, 2001). Tujuan pengujian ini adalah untuk mengambil komponen yang telah diuji unit dan membuat struktur program yang ditentukan oleh desain. Meskipun unit telah diuji masing-masing dan dapat bekerja dengan baik, namun harus diuji juga ketika unit-unit disatukan, karena satu unit akan berdampak pada unit lainnya. Pengujian integrasi berfokus untuk mengevaluasi interaksi di antara unit tersebut. terdapat beberapa teknik dalam

melakukan pengujian integrasi yaitu *top-down integration, bottom-up integration, regression testing,* dan *smoke testing* (Pressman, 2001).

Pada penelitian ini pengujian integrasi menggunakan pendekatan *top-down integration*. Pendekatan *top-down* adalah pendekatan untuk melakukan pengujian integrasi melalui hirarki kontrol dengan bergerak ke arah bawah. Pengujian dimulai dari modul kontrol utama menuju ke modul di bawahnya, modul yang berada di bawah dimasukkan ke dalam struktur baik secara *breadth-first* maupun *depth-first.* Strategi integrasi *top-down* memverifikasi kontrol utama di awal proses pengujian. Proses integrasi dilakukan dalam serangkaian lima langkah:

1. *Driver* tes menggunakan modul kontrol utama dan semua komponen yang berada di bawah modul kontrol utama diganti dengan *stub*.
2. *Stub* diganti satu per satu dengan komponen yang sebenarnya.
3. Pengujian dilakukan karena setiap komponen terintegrasi.
4. Setelah rangkaian tes selesai, *stub* lainnya diganti dengan komponen asli.
5. Pengujian regresi dapat dilakukan untuk memastikan bahwa kesalahan baru belum diperkenalkan.

2.9.3 Pengujian Validasi

Pengujian validasi dilakukan untuk memastikan bahwa semua kebutuhan yang didefinisikan telah terpenuhi oleh perangkat lunak(Pressman, 2001). Validasi didapatkan melalui rangkaian *black-box testing* yang menunjukkan ketepatan dengan kebutuhan. *Test plan* menguraikan kelas-kelas uji dan prosedur uji menentukan kasus-kasus uji tertentu yang akan digunakan untuk menunjukkan ketepatan dengan kebutuhan. Baik rencana dan prosedur dibuat untuk memastikan bahwa semua kebutuhan fungsional telah terpenuhi, semua karakteristik perilaku tercapai, semua kebutuhan kinerja tercapai, dan dokumentasi benar. Dokumen spesifikasi kebutuhan perangkat lunak mengandung bagian yang disebut kriteria validasi. Informasi yang terkandung dalam bagian itu menjadi dasar untuk pendekatan pengujian validasi.

2.9.3.1 Pengujian *Black-Box*

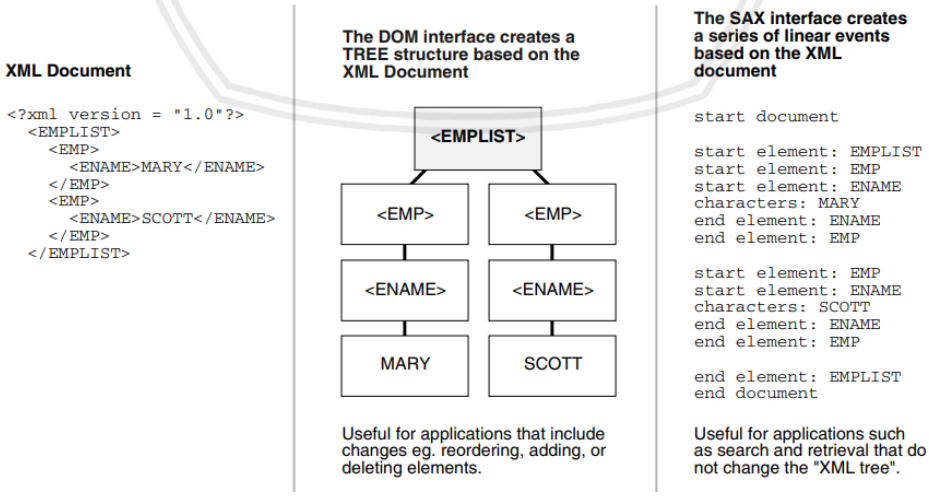
Pengujian *black-box* merupakan suatu pendekatan untuk pengujian di mana penguji tidak mengakses kode program atau komponen dari suatu sistem. Pengujian berdasarkan kebutuhan sistem (Sommerville, 2011). Pada pengujian *black-box* dibutuhkan kasus uji yaitu spesifikasi input untuk tes dan ekspektasi output dari sistem, ditambah dengan pernyataan tentang apa yang sedang diuji. Data uji merupakan masukan yang telah dirancang untuk menguji suatu sistem. Kadang-kadang data uji dapat dihasilkan secara otomatis, tetapi pembuatan kasus uji otomatis tidak mungkin dilakukan, karena orang-orang yang memahami apa yang seharusnya dilakukan sistem harus terlibat untuk menentukan hasil tes yang diharapkan.

Dalam prakteknya, proses pengujian biasanya melibatkan campuran pengujian manual dan otomatis. Dalam pengujian manual, penguji menjalankan program dengan beberapa data uji dan membandingkan hasilnya dengan harapan mereka. Mereka mencatat dan melaporkan ketidaksesuaian kepada pengembang program. Dalam pengujian otomatis, pengujian dikodekan dalam program yang dijalankan setiap kali sistem yang sedang dikembangkan akan diuji. Ini biasanya lebih cepat daripada pengujian manual, terutama ketika melibatkan pengujian regresi yaitu menjalankan kembali pengujian sebelumnya untuk memeriksa apakah perubahan pada program belum memperkenalkan bug baru.

2.10 Teknologi Pengembangan

2.10.1 Java API for XML Processing (JAXP)

JAXP adalah antarmuka standar untuk memproses XML dengan aplikasi Java (Ashdown, Greenberg, & Melnick, 2014). JAXP mendukung standar DOM dan SAX. DOM API memparsing dokumen XML dan membangun representasi *tree* dari dokumen dalam memori, menyediakan kemampuan pengguna untuk mengakses dan memanipulasi seperti menambah, membaca, mengganti elemen atau, dan menghapus. DOM API lebih mudah digunakan karena menyediakan struktur *tree* yang familiar .Sedangkan SAX API memproses dokumen XML sebagai aliran peristiwa, yang berarti program tidak dapat mengakses lokasi acak dalam dokumen. SAX berguna untuk operasi pencarian dan program lain yang tidak perlu manipulasi *XML tree.* SAX lebih cepat dari DOM ketika mengambil dokumen XML dari database. Pada penelitian ini JAXP digunakan untuk memproses *class diagram* sebagai inputan untuk perhitungan nilai *reusability* dalam format XML untuk selanjutnya masuk kedalam perhitungan metric untuk perhitungan nilai *reusability.* Perbandingan DOM dan SAX API ditunjukkan dalam Gambar 2.6.

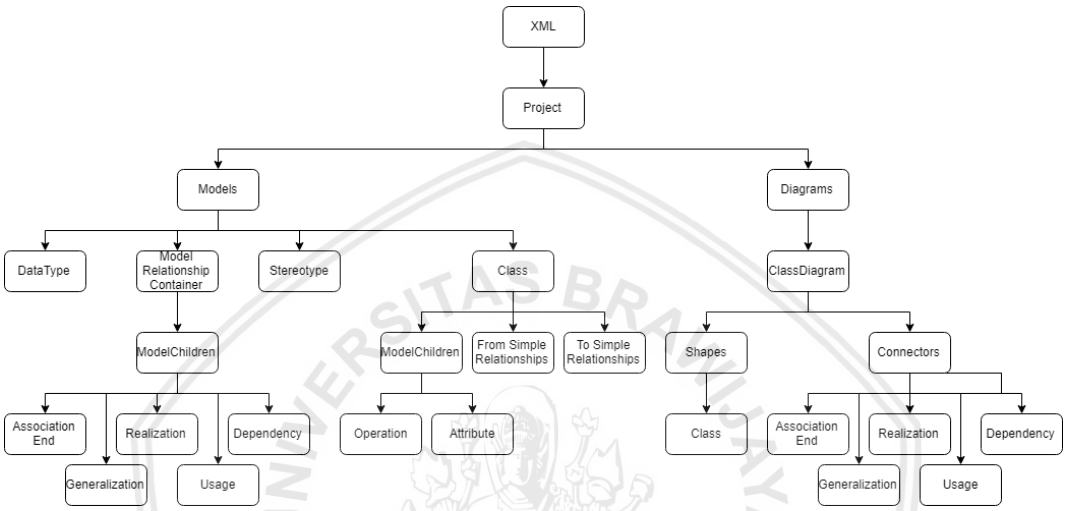


Gambar 2.6 Perbedaan SAX API dan DOM API

Sumber: (Ashdown, Greenberg, & Melnick, 2014)

2.10.1.1 *XML Schema*

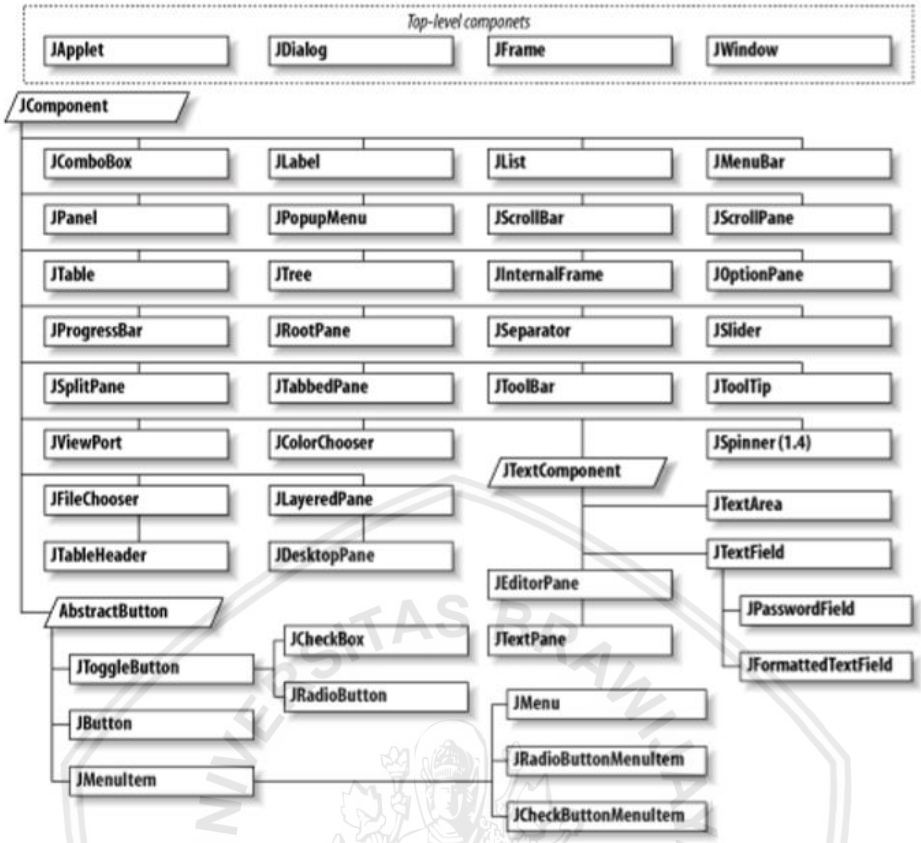
Pada penelitian ini masukan untuk mengukur nilai *reusability* menggunakan XML format *simple* dari Visual Paradigm. *XML schema* dari *class diagram* format *simple* ditunjukkan pada Lampiran A. Pada penelitian ini menggunakan DOM API karena lebih mudah untuk mengakses elemen-elemen yang dibutuhkan ketika direpresentasikan dalam struktur *tree.* Dari *XML schema class diagram* jika direpresentasikan dalam stuktur *tree* maka seperti dalam Gambar 2.7.



Gambar 2.7 XML struktur *tree*

2.10.2 Swing

Swing adalah toolkit GUI yang diciptakan Sun Microsystems yang merupakan generasi baru untuk memungkinkan pengembangan aplikasi *enterprise* di Java (Cole, dkk, 2002). Dengan menggunakan Swing, pemrogram dapat membuat aplikasi Java skala besar dengan beragam komponen yang kuat. Selain itu, dapat dengan mudah memodifikasi komponen-komponen ini untuk mengontrol penampilan dan perilakunya. Swing adalah bagian dari produk-produk Java yaitu Java Foundation Classes (JFC), yang merupakan gabungan banyak fitur dari Netscape's Internet Foundation Class (IFC) serta aspek desain dari divisi Taligent IBM dan Desain Mercusuar. Swing membutuhkan setidaknya JDK 1.1.5 untuk menjalankan. Swing dibangun berdasarkan model yang diperkenalkan dalam seri JDK 1.1. Selain itu harus memiliki *browser* yang mendukung Java 1.1 untuk mendukung applet Swing. Komponen hirarki Swing ditunjukkan dalam Gambar 2.8.



Gambar 2.8 Komponen hirarki Swing

Sumber: (Cole, dkk, 2002)

2.11 Perhitungan Efisiensi

Efisiensi merupakan salah satu jenis kebutuhan non-fungsional yang termasuk dalam kebutuhan produk, efisiensi dapat berupa kebutuhan performa dan kebutuhan ruang (Sommerville, 2011). Perhitungan efisiensi terdapat dua macam yaitu efisiensi waktu (*time based efficiency)* dan efisiensi secara keseluruhan (*overall relative efficiency)* (Sergeev, 2010). Efisiensi waktu dapat dihitung dari kecepatan pengguna ketika menyelesaikan suatu tugas. Sedangkan efisiensi secara keseluruhan dapat dihitung dari presentase waktu yang diperlukan pengguna untuk berhasil menyelesaikan tugas dalam kaitannya dengan total waktu yang diperlukan oleh semua pengguna. Persamaan 2.6 adalah perhitungan efisiensi waktu dan Persamaan 2.7 adalah perhitungan efisiensi secara keseluruhan.

$$\bar{P}_t = \frac{\sum_{j=1}^{R}\sum_{i=1}^{N}\frac{n_{ij}}{t_{ij}}}{NR} \tag{2.6}$$

$$\bar{P} = \frac{\sum_{j=1}^{R}\sum_{i=1}^{N} n_{ij}t_{ij}}{\sum_{j=1}^{R}\sum_{i=1}^{N} t_{ij}} * 100\% \tag{2.7}$$

Keterangan:

$\bar{P}_t$: *Time Based Efficiency*

$\bar{P}$ *: Overall Relative Efficiency*

N : Jumlah skenario (tugas)

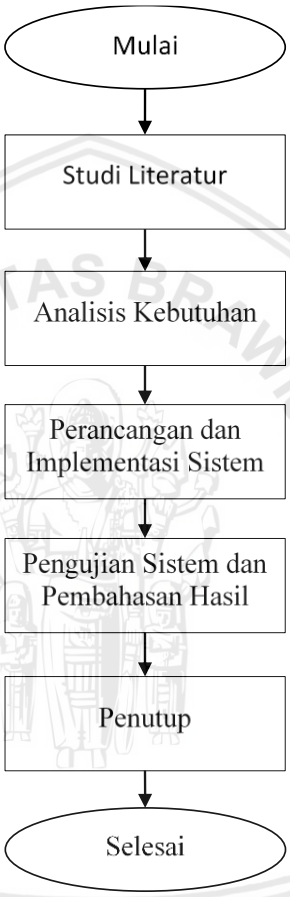
R : Jumlah responden (pengguna)

n_{ij} : tugas i yang diselesaikan pengguna j (nilainya jika berhasil adalah 1 dan gagal bernilai 0)

t_{ij} : waktu yang diperlukan responden untuk menyelesaikan skenario

BAB 3 METODOLOGI PENELITIAN

Metodologi penelitian ini berguna untuk mempermudah memahami alur penelitian ini. Tahapan dari penelitian ini antara lain studi literatur, analisis kebutuhan, perancangan sistem, implementasi sistem, pengujian sistem, kemudian penutup yang berisi kesimpulan dan saran. Untuk metode pengembangan aplikasinya, penulis menggunakan *SDLC waterfall model.* Diagram alir dari metodologi penelitian ini ditunjukkan dalam Gambar 3.1.



Gambar 3.1 Diagram alir metodologi penelitian

3.1 Studi Literatur

Pada tahap studi literatur ini penulis mempelajari landasan teori yang terkait dan mendukung topik penelitian. Landasan teori yang dipelajari berasal dari buku, jurnal-jurnal internasional, artikel, dan juga penelitian yang telah dilakukan sebelumnya. Adapun teori-teori yang dipelajar oleh penulis antara lain: rekayasa perangkat lunak, pengembangan perangkat lunak, model SDLC, pendekatan berorientasi objek, pemodelan berorientasi objek, kualitas perangkat lunak, *reusability* perangkat lunak, properti desain berorientasi objek, metrik desain berorientasi objek, metrik *reusability,* pengujian perangkat lunak, dan teknologi pengembangan yaitu JAXP dan *Swing*.

3.2 Analisis Kebutuhan

Pada tahap analisis kebutuhan dijelaskan deskripsi umum sistem, elisitasi kebutuhan, identifikasi karakteristik pengguna, kebutuhan fungsionalitas dan non-fungsionalitas sistem, lingkungan pengembangan sistem kakas bantu perhitungan nilai *reusability.* Dalam pengembangan sistem ini menggunakan pendekatan berorientasi objek sehingga kebutuhan fungsionalitas yang telah didefinisikan dimodelkan dengan *use case diagram,* dan kemudian di spesifikasikan dengan *use case scenario.*

3.3 Perancangan dan Implementasi Sistem

Pada tahap perancangan sistem dibuat berdasarkan dari hasil rekayasa kebutuhan. Karena pendekatan yang digunakan berorientasi objek maka rancangan dimodelkan dengan *sequence diagram* dan *class diagram,* selain itu juga terdapat perancangan algoritme dan perancangan antarmuka. Pada tahap ini juga dilakukan perancangan antarmuka pengguna. Kemudian implementasi sistem sesuai dengan rancangan yang telah dibuat. Pada implementasi dijelaskan spesifikasi sistem, batasan sistem, kode sumber, dan hasil implementasi antarmuka. Implementasi sistem dibuat menggunakan bahasa pemrograman *Java* dan berbasis *dekstop*.

3.4 Pengujian Sistem dan Pembahasan Hasil

Pada tahap ini dilakukan pengujian terhadap hasil implementasi sistem. Pengujian yang dilakukan yaitu pengujian unit menggunakan teknik pengujian *white-box,* pengujian integrasi menggunakan teknik *top-down,* dan pengujian validasi menggunakan teknik pengujian *black-box* untuk kebutuhan fungsional maupun non-fungsional. Selain itu pada bagian ini juga dilakukan pembahasan hasil pengujian.

3.5 Penutup

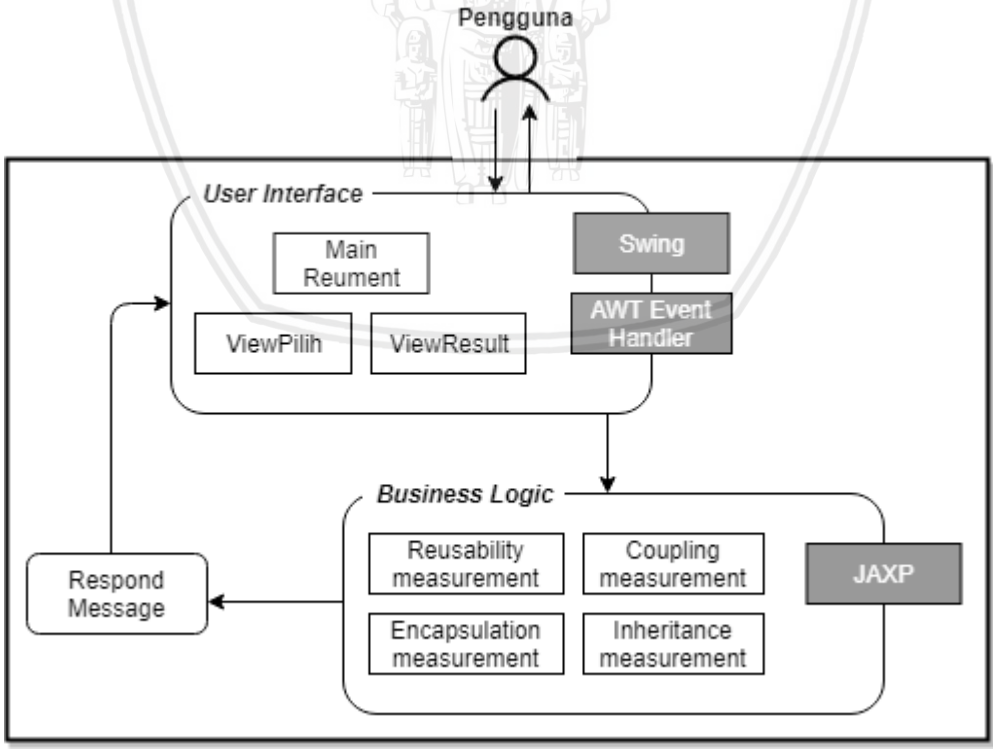
Pada tahap penutup ini dilakukan penarikan kesimpulan dari hasil penelitian yang telah dilakukan yaitu berdasarkan rumusan masalah yang telah diuraikan. Selain itu juga diuraikan saran untuk penelitian selanjutnya yang merupakan masukan dan juga evaluasi dari kesalahan yang mungkin terjadi pada penelitian ini.

BAB 4 ANALISIS KEBUTUHAN

4.1 Deskripsi Umum Sistem

Sistem yang dikembangkan adalah kakas bantu untuk menghitung nilai *reusability* sebuah rancangan perangkat lunak berupa *class diagram.* Perhitungan menggunakan metrik properti desain berorientasi objek dengan teknik regresi berganda. Properti desain berorientasi objek yang mempengaruhi *reusability* adalah *coupling, inheritance,* dan *encapsulation. Coupling* dihitung menggunakan metrik *Direct Class Coupling* (DCC)*, inheritance* dihitung menggunakan metrik *Measure of Functional Abstraction* (MFA)*, dan encapsulation* dihitung menggunakan metrik *Data Access Metric* (DAM). Metrik-metrik tersebut sebagai variabel independen dan *reusability* sebagai variabel dependen.

Sistem yang dikembangkan adalah aplikasi berbasis *dekstop.* Untuk melakukan perhitungan nilai *reusability,* pengguna harus terlebih dahulu menyimpan rancangan sistem yaitu *class diagram* dalam format dokumen XML. Dokumen XML tersebut yang akan menjadi masukan bagi sistem untuk menghitung nilai *reusability.* Setelah dokumen XML dimasukkan, sistem selanjutnya mengambil komponen-komponen yang diperlukan untuk perhitungan nilai *reusability.* Sistem menggunakan JAXP untuk memproses XML dan mendapatkan komponen-komponen yang dibutuhkan.



Gambar 4.1 Arsitektur sistem

Gambar 4.1 merupakan ilustrasi berjalannya sistem, pengguna dan sistem saling berkomunikasi melalui antarmuka pengguna (*user interface)* yang dibuat menggunakan komponen *Swing* dan *event-handler* menggunakan AWT untuk memberi masukan maupun menerima keluaran. Masukan dari pengguna kemudian dikelola oleh *business logic* pada sistem dimana JAXP memproses berkas XML yang kemudian hasilnya digunakan untuk melakukan perhitungan *coupling, inheritance, encapsulation,* dan *reusability.* Setelah berkas masukan dikelola selanjutnya sistem memberi pesan respon (*respond message)* yang akan ditampilkan pada antarmuka pengguna.

4.2 Elisitasi Kebutuhan

Elisitasi kebutuhan ini menguraikan kebutuhan pengguna yang nantinya akan dipenuhi oleh sistem. Elisitasi kebutuhan dilakukan dengan mengkaji pustaka yang terkait dengan perhitungan nilai *reusability.*

1. Memilih berkas XML *class diagram* perangkat lunak.
2. Mengukur poin *coupling class diagram* perangkat lunak.
3. Mengukur poin *inheritance class diagram* perangkat lunak.
4. Mengukur poin *encapsulation class diagram* perangkat lunak.
5. Mengukur nilai *reusability class diagram* perangkat lunak.
6. Melihat hasil dari pengukuran *reusability class diagram* perangkat lunak.

4.3 Identifikasi Pengguna

Pengguna sistem kakas bantu perhitungan nilai *reusability* ini adalah tim pengembang perangkat lunak khususnya *desainer* perangkat lunak yang memahami istilah-istilah pada sistem yaitu *encapsulation, coupling, inheritance, class diagram, design,* dan *reusability.* Sistem hanya mempunyai satu pengguna yaitu *desainer* dari perangkat lunak sehingga pada penulisan ini penulis menggunkan istilah ‘pengguna’ untuk menyebut *desainer* sebagai pengguna. Pengguna menggunakan sistem kakas bantu ini untuk mengukur rancangan yang telah dibuat memiliki seberapa nilai *reusability*. Pada Tabel 4.1 menunjukkan deskripsi yang dapat dilakukan oleh pengguna pada sistem ini.

Tabel 4.1 Identifikasi pengguna

No	Identifikasi Pengguna	Deskripsi
1.	Pengguna	Pada sistem ini yang dapat dilakukan oleh pengguna adalah memilih file XML *class diagram* rancangan perangkat lunak, mengukur poin *coupling class diagram* rancangan perangkat lunak, mengukur poin *inheritance class diagram* rancangan perangkat lunak, mengukur poin

Tabel 4.1 Identifikasi pengguna (lanjutan)

		encapsulation class diagram rancangan perangkat lunak, mengukur nilai *reusability class diagram* rancangan perangkat lunak, dan melihat hasil dari pengukuran *reusability class diagram* rancangan perangkat lunak.

4.4 Spesifikasi Kebutuhan Sistem

4.4.1 Kebutuhan fungsional

Sistem harus memenuhi kebutuhan fungsional yang diuraikan pada Tabel 4.2 berikut dengan spesifikasinya.

Tabel 4.2 Spesifikasi kebutuhan fungsional sistem

Kode Kebutuhan	*Use Case*	Spesifikasi Kebutuhan
F_REUS_001	*Browse* Berkas	Sistem memberikan layanan kepada pengguna untuk memasukkan berkas *class diagram* rancangan perangkat lunak.
		1.1. Sistem memiliki tombol *Browse* dan juga textfield yang akan terisi otomatis dengan alamat direktori berkas yang telah dimasukkan oleh pengguna. 1.2. Sistem menampilkan dialog untuk *browse* berkas. 1.3. Berkas yang bisa dipilih dan ditampilkan pada direktori adalah berkas hanya yang berformat XML. 1.4. Berkas yang dapat diterima sistem sebagai masukan hanya dengan format XML dan struktur *simple* dari Visual Paradigm.
F_REUS_002	Ukur Poin *Coupling*	Sistem memberikan layanan untuk mengukur poin *coupling class diagram* rancangan perangkat lunak masukan dari pengguna.
		2.1. Sistem menggunakan metrik DCC *(Direct Class Coupling)* untuk mengukur *coupling class diagram* rancangan perangkat lunak seperti yang telah dijelaskan pada bagian landasan kepustakaan. 2.2. Sistem memiliki tombol untuk melakukan proses perhitungan.

Tabel 4.2 Spesifikasi kebutuhan fungsional sistem (lanjutan)

		2.3. Perhitungan tidak diproses apabila tidak ada berkas xml *class diagram* rancangan perangkat lunak yang dipilih.
F_REUS_003	Ukur Poin *inheritance*	Sistem memberikan layanan untuk mengukur poin *inheritance class diagram* rancangan perangkat lunak masukan dari pengguna.
		3.1. Sistem menggunakan metrik MFA untuk mengukur *inheritance class diagram* rancangan perangkat lunak seperti yang telah dijelaskan pada bagian landasan kepustakaan. 3.2. Sistem memiliki tombol untuk melakukan proses perhitungan. 3.3. Perhitungan tidak diproses apabila tidak ada berkas xml *class diagram* rancangan perangkat lunak yang dipilih.
F_REUS_004	Ukur Poin *Encapsulation*	Sistem memberikan layanan untuk mengukur poin *encapsulation class diagram* rancangan perangkat lunak masukan dari pengguna.
		2.1. Sistem menggunakan metrik DAM untuk mengukur *encapsulation class diagram* rancangan perangkat lunak seperti yang telah dijelaskan pada bagian landasan kepustakaan. 2.2. Sistem memiliki tombol untuk melakukan proses perhitungan. 2.3. Perhitungan tidak diproses apabila tidak ada berkas xml *class diagram* rancangan perangkat lunak yang dipilih.
F_REUS_005	Ukur *Reusability*	Sistem memberikan layanan untuk mengukur *reusability class diagram* rancangan perangkat lunak masukan dari pengguna.
		5.1. Sistem menggunakan teknik regresi berganda berdasarkan nilai dari *encapsulation, coupling*, dan *inheritance* untuk mengukur *reusability class diagram* rancangan perangkat lunak. 5.2. Sistem memiliki tombol untuk melakukan proses perhitungan.

Tabel 4.2 Spesifikasi kebutuhan fungsional sistem (lanjutan)

		5.3. Perhitungan tidak diproses apabila tidak ada berkas xml *class diagram* rancangan perangkat lunak yang dipilih.
F_REUS_006	Lihat Hasil	Sistem memberikan layanan bagi pengguna untuk menampilkan hasil dari perhitungan *reusability,* poin *coupling,* poin *inheritance,* dan poin *encapsulation*.
		6.1. Sistem memiliki halaman untuk melihat hasil dari perhitungan *reusability,* poin *encapsulation,* poin *coupling, dan* poin *inheritance*. 6.2. Sistem memiliki tombol navigasi untuk menuju halaman lihat hasil perhitungan *reusability,* poin *coupling,* poin *inheritance,* dan poin *encapsulation*. 6.3. Sistem dapat menampilkan hasil dari nilai *reusability* beserta dengan rincian perhitungannya. 6.4. Sistem dapat menampilkan hasil dari perhitungan poin *coupling* berserta dengan rincian perhitungnya. 6.5. Sistem dapat menampilkan hasil dari perhitungan poin *inheritance* berserta dengan rincian perhitungnya. 6.6. Sistem dapat menampilkan hasil dari perhitungan poin *encapsulation* berserta dengan rincian perhitungnya.

4.4.2 Kebutuhan Non Fungsional

Sistem harus memenuhi kebutuhan non-fungsional yang diuraikan pada Tabel 4.3

Tabel 4.3 Deskripsi kebutuhan non-fungsional

Kode Koebutuhan	Parameter	Deskripsi
NF_REUS_001	Efisiensi	Sistem mempunyai tingkat efisiensi yaitu 100% untuk keberhasilan melakukan perhitungan dan lebih cepat dari perhitungan manual.

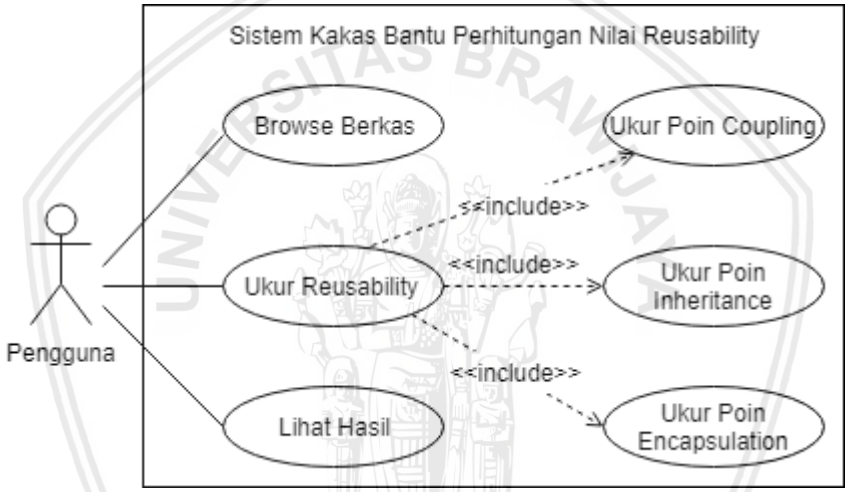
4.5 Lingkungan Pengembangan

Spesifikasi lingkungan pengembagan sistem pada penulisan ini adalah sebagai berikut:

1. Menggunakan bahasa pemrograman *Java.*
2. Menggunakan IDE NetBeans versi 8.2.
3. Menggunakan JAXP untuk pemroses XML.
4. Menggunakan sistem operasi Windows 10.

4.6 Pemodelan kebutuhan

4.6.1 Use Case Diagram



Gambar 4.2 U*se case diagram* sistem

Kebutuhan fungsionalitas yang telah dispesifikasikan pada sub bab sebelumnya kemudian dimodelkan kedalam *use case diagram* dalam Gambar 4.2. Selanjutnya penjelasan secara rinci tentang perilaku pada setiap *use case* kemudian dijelaskan kedalam *use case scenario* yang diuraikan pada Tabel 4.4 sampai dengan Tabel 4.9. Pada *use case diagram* tersebut menunjukkan Pengguna adalah aktor tungal dalam sistem yang dapat menggunakan semua kebutuhan sistem. Kebutuhan sistem berjumlah 6 yaitu:

1. Browse berkas (F_REUS_001)
2. Ukur poin *coupling* (F_REUS_002)
3. Ukur poin *inheritance* (F_REUS_003)
4. Ukur poin *encapsulation* (F_REUS_004)
5. Ukur *reusability* (F_REUS_005)
6. Lihat hasil (F_REUS_006)

4.6.2 Use Case Scenario

Berikut ini *use case scenario* yang menjelaskan secara lebih rinci perilaku pengguna dengan sistem :

1. *Use Case Scenario Browse* Berkas (F_REUS_001)

   Perilaku dari interaksi Pengguna dengan *use case Browse* Berkas dijelaskan secara rinci pada Tabel 4.4.

Tabel 4.4 U*se case scenario browse* berkas

Browse Berkas	
Kode	F_REUS_001
Objectives	Meyediakan layanan untuk Pengguna memilih berkas *class diagram* rancangan perangkat lunak pada direktori lokal.
Actor	Pengguna
Pre Conditions	1. Halaman untuk b*rowse* berkas ditampilkan.
Main Flow	1. Pengguna menekan pada tombol *Browse.* 2. Sistem menampilkan dialog pilih file dan hanya menampilkan berkas XML. 3. Pengguna memilih sebuah berkas dan menekan tombol *Open*.
Alternative Flow	1. Pengguna menekan tombol *cancel* untuk membatalkan pemilihan berkas.
Post Conditions	Alamat direktori berkas yang dipilih ditampilkan.

2. *Use Case Scenario* Ukur Poin *Coupling* (F_REUS_002)

   Perilaku dari interaksi Pengguna dengan *use case* Ukur Poin *Coupling* dijelaskan secara rinci pada Tabel 4.5.

Tabel 4.5 *Use case scenario* ukur poin *coupling*

Ukur Poin *Coupling*	
Kode	F_REUS_002
Objectives	Menyediakan layanan untuk Pengguna mengukur poin *coupling* dari *class diagram* perangkat lunak yang telah dipilih.
Actor	Pengguna
Pre Conditions	1. Halaman untuk *browse* berkas ditampilkan .

Tabel 4.5 *Use case scenario* ukur poin *coupling* (lanjutan)

Pre Conditions	2. Berkas yang akan diukur telah dipilih.
Main Flow	1. Pengguna menekan pada tombol Proses. 2. Sistem memparsing berkas XML *class diagram,* mengukur *coupling,* dan mengukur nilai poin *coupling.*
Alternative Flow	1. Apabila berkas tidak berstruktur *simple* dari Visual Paradigm maka sistem akan menampilkan notifikasi bahwa berkas XML tidak sesuai. 2. Apabila tidak ada sebuah berkas yang dipilih maka sistem akan memberi notifikasi bahwa berkas belum dipilih.
Post Conditions	Nilai poin *coupling* dan nilai *reusability* telah diukur.

3. *Use Case Scenario* Ukur Poin *Inheritance* (F_REUS_003)

Perilaku dari interaksi Pengguna dengan *use case* Ukur Poin *Inheritance* dijelaskan secara rinci pada Tabel 4.6.

Tabel 4.6 U*se case scenario* ukur poin *inheritance*

Ukur Poin *Inheritance*	
Kode	F_REUS_003
Objectives	Menyediakan layanan untuk Pengguna mengukur poin *inheritance* dari *class diagram* perangkat lunak yang telah dipilih.
Actor	Pengguna
Pre Conditions	1. Halaman untuk *browse* berkas ditampilkan . 2. Berkas yang akan diukur telah dipilih.
Main Flow	1. Pengguna menekan pada tombol Proses. 2. Sistem memparsing berkas XML *class diagram,* mengukur *inheritnce,* dan mengukur nilai poin *inheritance.*
Alternative Flow	1. Apabila berkas tidak berstruktur *simple* dari Visual Paradigm maka sistem akan menampilkan notifikasi bahwa berkas XML tidak sesuai.

Tabel 4.6 U*se case scenario* ukur poin *inheritance* (lanjutan)

Alternative Flow	2. Apabila tidak ada sebuah berkas yang dipilih maka sistem akan memberi notifikasi bahwa berkas belum dipilih.
Post Conditions	Nilai poin *inheritance* dan nilai *reusability* telah diukur.

4. *Use Case Scenario* Ukur Poin *Encapsulation* (F_REUS_004)

   Perilaku dari interaksi Pengguna dengan *use case* Ukur Poin *Encapsulation* dijelaskan secara rinci pada Tabel 4.7.

Tabel 4.7 *Use case scenario* ukur poin *encapsulation*

Ukur Poin *Encapsulation*	
Kode	F_REUS_004
Objectives	Menyediakan layanan untuk Pengguna mengukur poin *encapsulation* dari *class diagram* perangkat lunak yang telah dipilih.
Actor	Pengguna
Pre Conditions	1. Halaman untuk *browse* berkas ditampilkan . 2. Berkas yang akan diukur telah dipilih.
Main Flow	1. Pengguna menekan pada tombol Proses. 2. Sistem memparsing berkas XML *class diagram,* mengukur *encapsulation,* dan mengukur nilai poin *encapsulation.*
Alternative Flow	1. Apabila berkas tidak berstruktur *simple* dari Visual Paradigm maka sistem akan menampilkan notifikasi bahwa berkas XML tidak sesuai. 2. Apabila tidak ada sebuah berkas yang dipilih maka sistem akan memberi notifikasi bahwa berkas belum dipilih.
Post Conditions	Nilai poin *coupling* dan nilai *reusability* telah diukur.

5. *Use Case Scenario* Ukur *Reusability* (F_REUS_005)

   Perilaku dari interaksi *use case* Ukur *Reusability* dengan Pengguna dijelaskan secara rinci pada Tabel 4.8.

Tabel 4.8 *Use case scenario* ukur *reusability*

Ukur *Reusability*	
Kode	F_REUS_005
Objectives	Menyediakan layanan untuk Pengguna mengukur *reusability* dari *class diagram* perangkat lunak yang telah dipilih.
Actor	Pengguna
Pre Conditions	1. Halaman untuk *browse* berkas ditampilkan . 2. Berkas yang akan diukur telah dipilih.
Main Flow	1. Pengguna menekan pada tombol Proses. 2. Sistem memparsing berkas XML *class diagram,* mengukur *reusability*
Alternative Flow	1. Apabila berkas tidak berstruktur *simple* dari Visual Paradigm maka sistem akan menampilkan notifikasi bahwa berkas XML tidak sesuai. 2. Apabila tidak ada sebuah berkas yang dipilih maka sistem akan memberi notifikasi bahwa berkas belum dipilih.
Post Conditions	Nilai *reusability* telah diukur.

6. *Use Case Scenario* Lihat Hasil (F_REUS_006)

   Perilaku dari interaksi *use case* Lihat Hasil dengan Pengguna dijelaskan secara rinci pada Tabel 4.9.

Tabel 4.9 *Use case scenario* lihat hasil

Lihat Hasil	
Kode	F_REUS_006
Objectives	Menyediakan layanan untuk Pengguna melihat hasil dari pengukuran *reusability,* poin *coupling,* poin *inheritance*, dan poin *encapsulation class diagram* rancangan perangkat lunak.
Actor	Pengguna
Pre Conditions	1. Halaman untuk *browse* berkas ditampilkan. 2. Nilai *reusability*, poin *coupling,* poin *inheritance*, dan poin *encapsulation* telah diukur.

Tabel 4.9 *Use case scenario* lihat hasil (lanjutan)

Main Flow	1. Pengguna menekan pada tombol navigasi yang menuju ke halaman lihat hasil.
Alternative Flow	-
Post Conditions	Hasil pengukuran *reusability*, poin *encapsulation,* poin *coupling*, poin *inheritance*, dan perincian rumus pengukuran ditampilkan.

4.7 Perhitungan Manual

Perhitungan manual ini dilakukan untuk menjelaskan dan memberikan pemahaman perhitungan *reusability* dengan rumus teknik regresi berganda menggunakan 3 variabel independen yaitu *coupling, inheritance,* dan *encapsulation.* Rumus yang digunakan adalah seperti persamaan 4.1:

Reusability = -37,111 + 3,973 * *Coupling* + 32,500 * *Inheritance*
+ 20,709 * *Encapsulation* (4.1)

Dimana,

Coupling : Dihitung menggunakan metrik DCC

Inheritance : Dihitung menggunakan metrik MFA

Encapsulation : Dihitung menggunakan metrik DAM

Gambar 4.3 contoh *class diagram* "sistem X" yang digunakan untuk perhitungan. Dari contoh *class diagram* sistem X didapatkan data seperti pada Tabel 4.10:

Tabel 4.10 Data set sistem X

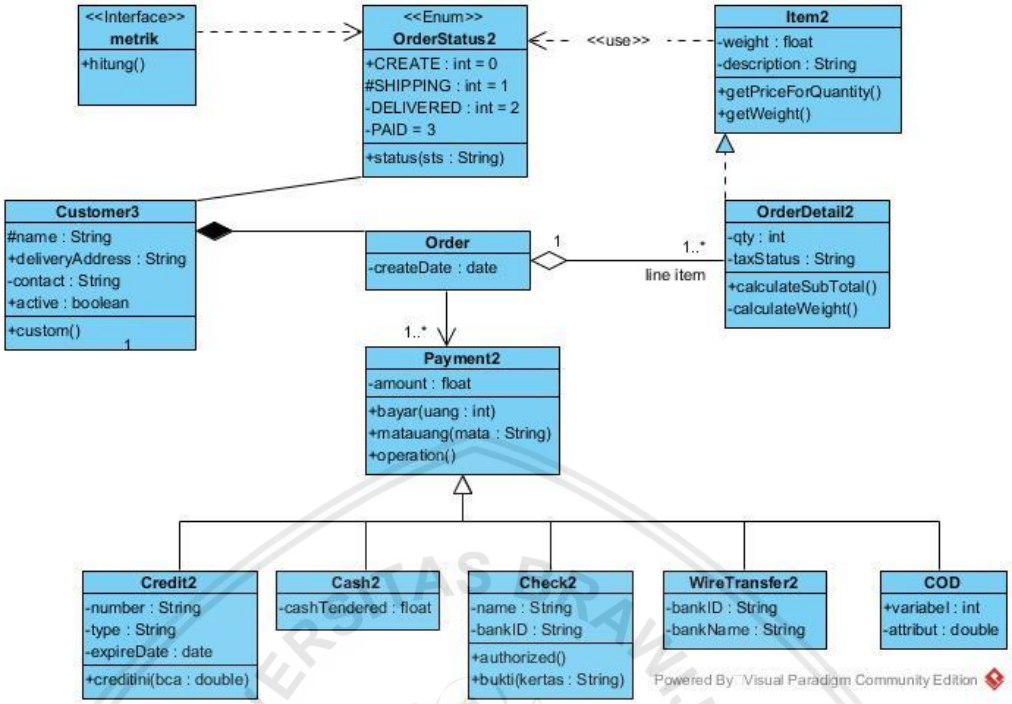
Sistem X		
Coupling	***Inheritance***	***Encapsulation***
26/12 = 2,17	4,35/8=0,54	9,75/11=0,89

Dengan menggunakan data set dari tabel 4.10 dan menggunakan rumus pengukuran *reusability* diperoleh nilai *reusability* sistem sepeda sebagai berikut:

Reusability = -37,111 + 3,973 * 2,17 + 32,500 * 0,54 + 20,709 * 0,89

Reusability = -37,111 + 8,608 + 17,672 + 18,356

Reusability = 7,525



Gambar 4.3 Contoh *class diagram* sistem X

BAB 5 PERANCANGAN DAN IMPLEMENTASI SISTEM

5.1 Perancangan Sistem

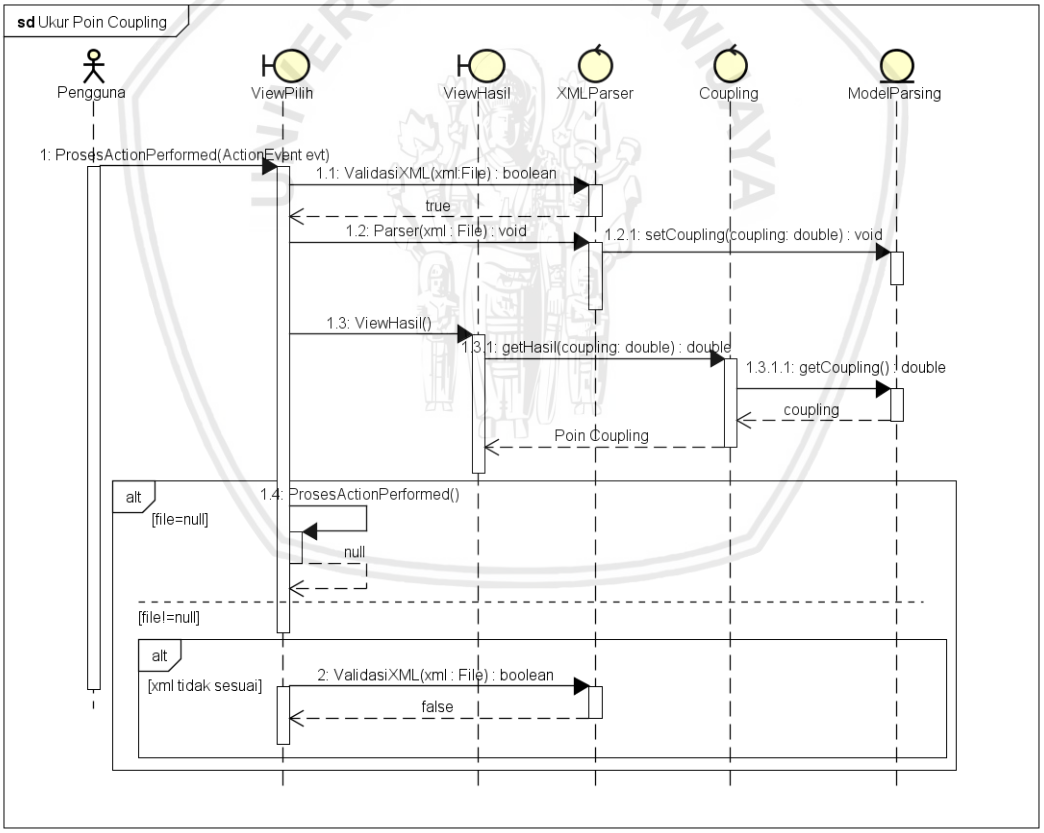
Perancangan sistem dibuat sesuai dengan hasil analisis kebutuhan. Pada bagian perancangan sistem ini terdapat perancangan arsitektur, algoritme, dan juga antarmuka. Perancangan sistem ini yang digunakan sebagai acuan tahap selanjutnya yaitu implementasi.

5.1.1 Perancangan Arsitektur

Karena dalam pengembangan ini menggunakan pendekatan berorientasi objek, maka pada tahap perancangan arsitektur dimodelkan dengan *sequence diagram* dan *class diagram.*

5.1.1.1 Sequence Diagram

1. Ukur Poin *Coupling* (F_REUS_002)

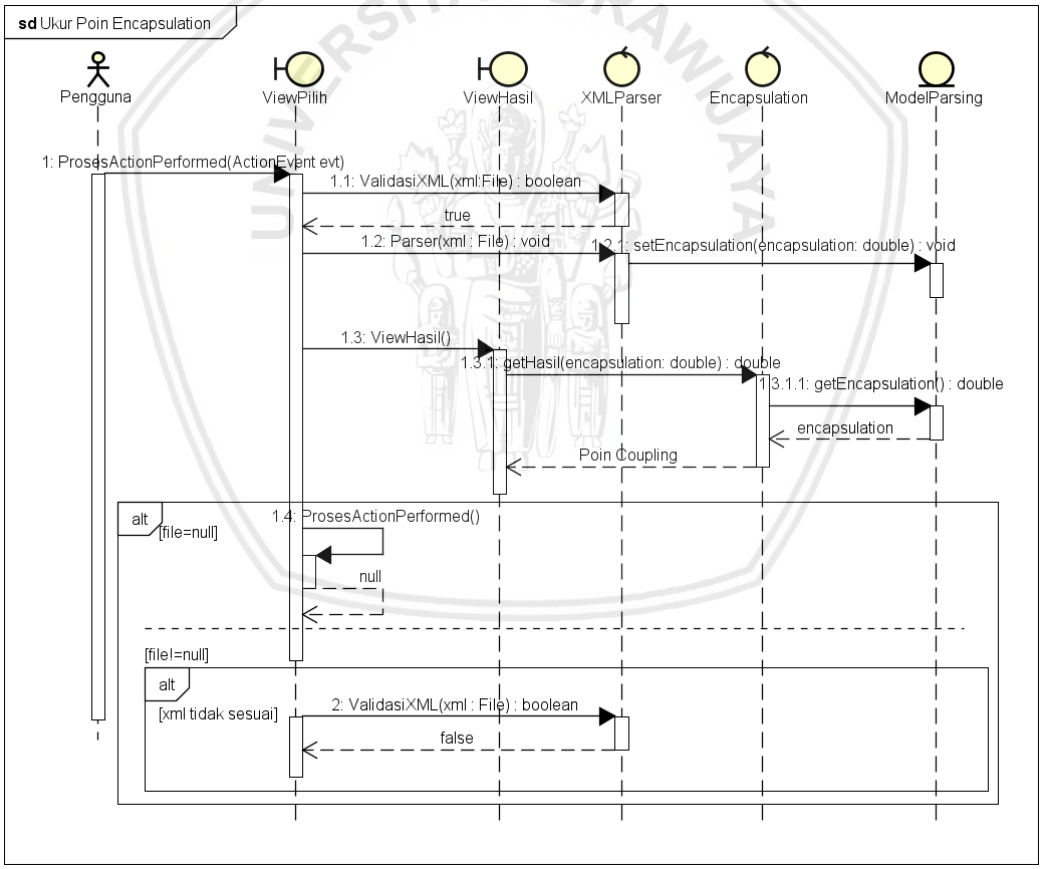


Gambar 5.1 *Sequence diagram* ukur poin *coupling*

Gambar 5.1 menjelaskan alur pengguna berinteraksi meminta sistem untuk melakukan pemrosesan berkas rancangan perangkat lunak dengan format *XML* untuk diukur nilai *reusability*nya yang terlebih dahulu sistem akan mengukur nilai poin *coupling* sesuai rumus persamaan. Terdapat empat objek yang terlibat dalam interaksi ini yaitu *ViewPilih* dan *ViewHasil* sebagai

boundary, Coupling sebagai *controller,* dan *XMLParser* sebagai *model*. Ketika *method prosesActionPerformed* dipanggil dari *ViewPilih* maka berkas XML yang dipilih diseleksi apakah struktur nya dari struktur *simple Visual Paradigm.* Jika struktur sesuai, selanjutnya berkas akan di*parsing* untuk diambil elemen yang dibutuhkan pada pengukuran nilai poin *coupling* yang dilakukan oleh *XMLParser.* Setelah elemen yang dibutuhkan didapat, kemudian ke kelas *ViewHasil* menjankan konstruktornya dan memanggil *getHasil dari kelas Coupling* untuk mengukur poin *coupling.* Poin *coupling* diukur dari *coupling* yang didapat dari hasil *parsing* di kelas *XMLParser.* Hasil dari poin *coupling* akan ditampilkan oleh *ViewHasil*. Jika struktur XML tidak sesuai maka proses pengukuran tidak akan dilakukan. Selain itu ketika tidak ada berkas maka tidak akan dilakukan proses pengukuran juga, hal ini dicek oleh *ViewPilih* saat melalui *method prosesActionPerformed*.

2. Ukur Poin *Encapsulation* (F_REUS_004)



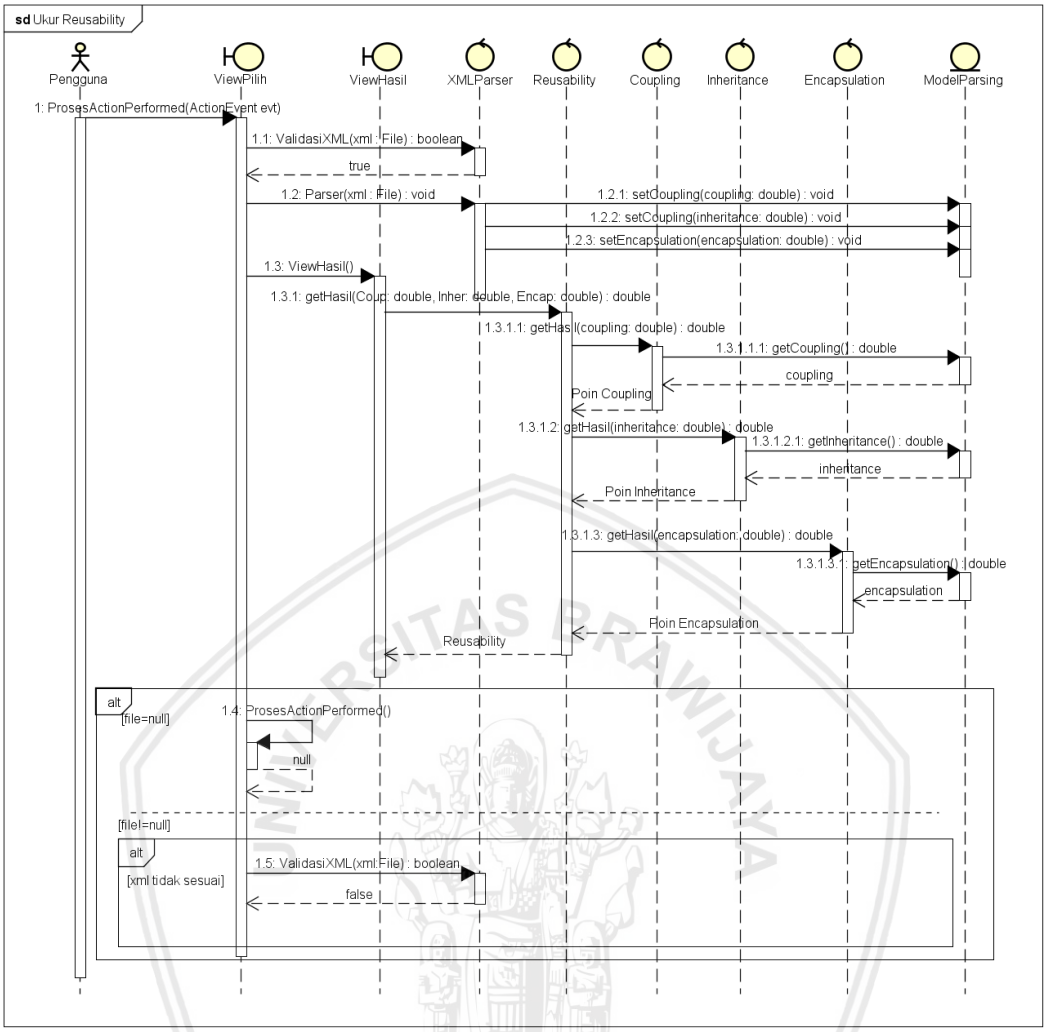
Gambar 5.2 *Sequence diagram* ukur poin *encapsulation*

Gambar 5.2 menjelaskan alur pengguna berinteraksi meminta sistem untuk melakukan pemrosesan berkas rancangan perangkat lunak dengan format *XML* untuk diukur nilai *reusability*nya yang terlebih dahulu sistem akan mengukur nilai poin *encapsulation* sesuai rumus persamaan. Terdapat empat objek yang terlibat dalam interaksi ini yaitu *ViewPilih* dan *ViewHasil* sebagai

boundary, Encapsulation sebagai *controller,* dan *XMLParser* sebagai *model*. Ketika *method prosesActionPerformed* dipanggil dari *ViewPilih* maka berkas XML yang dipilih diseleksi apakah struktur nya dari struktur *simple Visual Paradigm.* Jika struktur sesuai, selanjutnya berkas akan di*parsing* untuk diambil elemen yang dibutuhkan pada pengukuran nilai poin *encapsulation* yang dilakukan oleh *XMLParser.* Setelah elemen yang dibutuhkan didapat, kemudian ke kelas *ViewHasil* menjankan konstruktornya dan memanggil *getHasil dari kelas Encapsulation* untuk mengukur poin *encapsulation.* Poin *encapsulation* diukur dari *encapsulation* yang didapat dari hasil *parsing* di kelas *XMLParser.* Hasil dari poin *encapsulation* akan ditampilkan oleh *ViewHasil*. Jika struktur XML tidak sesuai maka proses pengukuran tidak akan dilakukan. Selain itu ketika tidak ada berkas maka tidak akan dilakukan proses pengukuran juga, hal ini dicek oleh *ViewPilih* saat melalui *method prosesActionPerformed*.

3. Ukur *Reusability* (F_REUS_005)

Gambar 5.4 menjelaskan alur pengguna berinteraksi meminta sistem untuk melakukan pemrosesan berkas *class diagram* rancangan perangkat lunak dengan format *XML* untuk mendapatkan nilai *reusability*nya setelah mendapatkan nilai poin *coupling,* nilai poin *inheritance,* dan nilai poin *encapsulation* sesuai rumus persamaan. Terdapat tujuh objek yang terlibat dalam interaksi ini yaitu *ViewPilih* dan *ViewHasil* sebagai *boundary, Coupling, Inheritance, Encapsulation,* dan *Reusability* sebagai *controller,* dan *XMLParser* sebagai *model*. Ketika *method prosesActionPerformed* dipanggil dari *ViewPilih* maka berkas XML yang dipilih diseleksi apakah struktur nya dari struktur *simple Visual Paradigm.* Jika struktur sesuai, selanjutnya berkas akan di*parsing* untuk diambil elemen yang dibutuhkan pada pengukuran *reusability* yang dilakukan oleh *XMLParser.* Setelah elemen yang dibutuhkan didapat, kemudian ke kelas *ViewHasil* menjankan konstruktornya dan memanggil *getHasil dari kelas Reusability* untuk mengukur *reusability. Reusability* didapat dari rumus yang memanggil *getHasil* dari kelas *Coupling, getHasil* dari kelas *Inheritance,* dan *getHasil* dari kelas *Encapsulation.* Selanjutnya hasil *reusability* ditampilkan pada *ViewHasil.* Jika struktur XML tidak sesuai maka proses pengukuran tidak akan dilakukan. Selain itu apabila tidak ada berkas yang dipilih maka tidak akan dilakukan proses pengukuran juga, hal ini dicek oleh *ViewPilih* saat melalui *method prosesActionPerformed*.

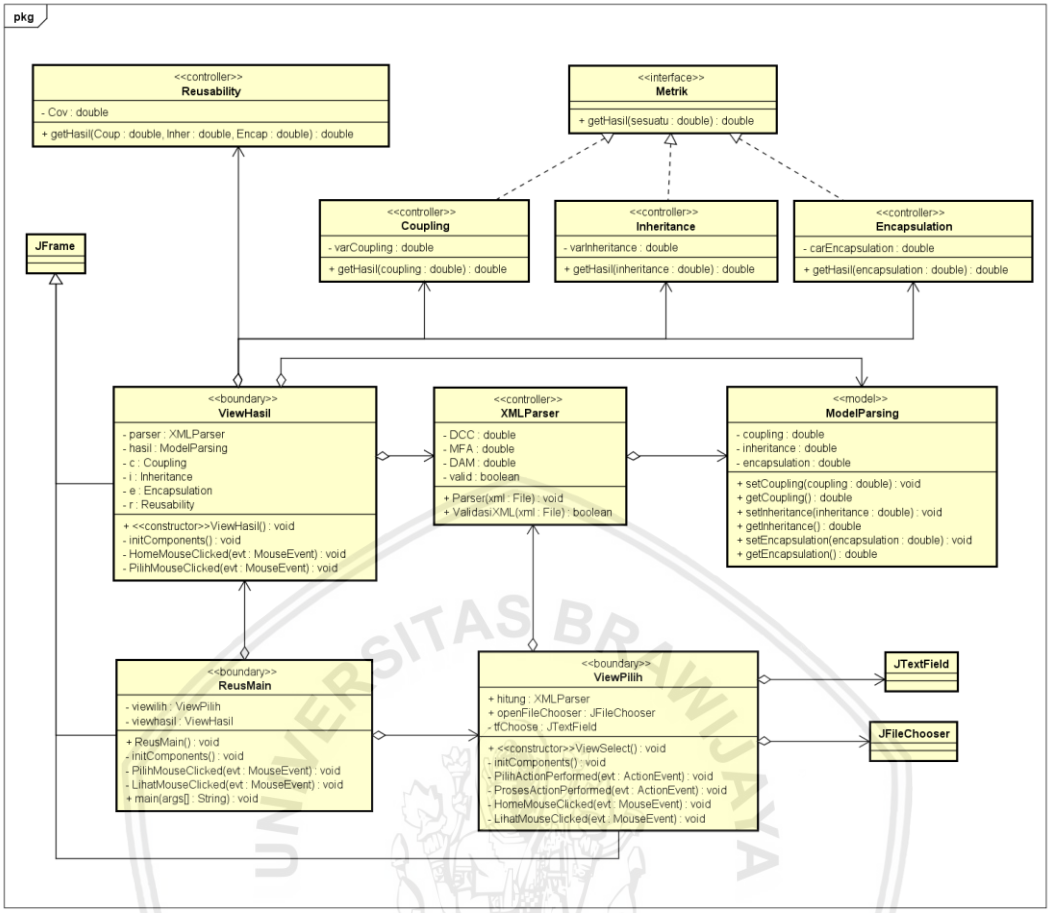


Gambar 5.3 *Sequence diagram* ukur *reusability*

5.1.1.2 Class Diagram

Class diagram dari sistem pengukuran nilai *reusability* ditunjukkan dalam Gambar 5.5. Didalam *class diagram* ini terdapat satu kelas *interface,* tiga kelas *boundary,* empat kelas *controller,* satu kelas *model*, dan tiga kelas dari penggunaan *API Swing.* Untuk lebih rinci penjelasan dari setiap kelasnya adalah sebagai berikut:

1. Kelas *interface* Metrik sebagai kelas abstraksi dari kelas *Inheritance,* kelas *Coupling,* dan *Encapsulation.* Dalam kelas ini terdapat satu operasi abstrak yaitu *getHasil*.



Gambar 5.4 *Class diagram*

2. Kelas *controller Coupling* bertugas untuk mengukur poin *coupling* dari masukan berkas rancangan perangkat lunak. Kelas ini adalah realisasi dari kelas *interface* Metrik. Dalam kelas ini terdapat satu atribut *final varCoupling* yang menyimpan nilai koefisian *coupling* untuk pengukuran *reusability.* Dalam kelas ini juga terdapat satu operasi yang merupakan *overriding* dari kelas *interface* Metrik yaitu *getHasil.* Kelas ini memiliki hubungan agregasi dengan kelas *boundary ViewHasil.*

3. Kelas *controller Inheritance* bertugas untuk mengukur poin *inheritance* dari masukan berkas rancangan perangkat lunak. Kelas ini adalah realisasi dari kelas *interface* Metrik. Dalam kelas ini terdapat satu atribut *final varInheritance* yang menyimpan nilai koefisian *inheritance* untuk pengukuran *reusability.* Dalam kelas ini juga terdapat satu operasi yang merupakan *overriding* dari kelas *interface* Metrik yaitu *getHasil.* Kelas ini memiliki hubungan agregasi dengan kelas *boundary ViewHasil.*

4. Kelas *controller Encapsulation* bertugas untuk mengukur poin *encapsulation* dari masukan berkas rancangan perangkat lunak. Kelas ini adalah realisasi dari kelas *interface* Metrik. Dalam kelas ini terdapat satu atribut *final varEncapsulation* yang menyimpan nilai koefisian *encapsulation* untuk pengukuran *reusability.* Dalam kelas ini juga terdapat satu operasi yang

merupakan *overriding* dari kelas *interface* Metrik yaitu *getHasil.* Kelas ini memiliki hubungan agregasi dengan kelas *boundary ViewHasil.*

5. Kelas *controller Reusability* bertugas untuk mengukur nilai *reusability* dari berkas masukan sesuai persamaan rumus mengukur *reusability*. Dalam kelas ini terdapat satu atribut yaitu *Cov* yang bertugas menyimpan nilai kovarian *reusability* dan memiliki satu operasi yaitu *getHasil*. Kelas ini memiliki hubungan agregasi dengan kelas *boundary ViewHasil.*
6. Kelas model *XMLParser* adalah kelas yang menyimpan hasil dari *parsing XML* yaitu elemen-elemen yang dibutuhkan untuk mengukur *reusability.* Dalam kelas ini terdapat empat atribut yaitu DCC, MFA, dan DAM yang menyimpan nilai dari elemen hasil *parsing*, dan juga atribut valid untuk menyimpan nilai validasi. Di dalam kelas ini terdapat lima operasi yaitu *getCoup, getInher, getEncap, Parser,* dan ValidasiXML. Kelas ini memiliki hubungan agregasi dengan kelas *boundary ViewPilih* dan *ViewHasil.*
7. Kelas Jframe adalah bawaan dari API Swing. Komponen *Frame* ini digunakan oleh kelas *boundary* yaitu *ReusMain, ViewPilih,* dan *ViewHasil.*
8. Kelas *boundary ViewPilih* bertugas menampilkan antarmuka pengguna untuk melakukan pemilihan berkas rancangan perangkat lunak yang akan diukur nilai *reusability*nya. Dalam kelas ini terdapat satu konstruktor dan empat operasi yaitu *PilihActionPerformed, ProsesActionPerformed, HomeMouseClicked,* dan *LihatMouseClicked.* Kelas ini memiliki hubungan agregasi dengan kelas *model XMLParser* dan kelas *boundary ReusMain.* Kelas ini menggunakan komponen *Frame, Filechooser, dan Textfield* dari API Swing.
9. Kelas JFileChoosen adalah bawaan dari API Swing. Komponen FileChooser digunakan oleh kelas boundary *ViewPilih.*
10. Kelas JTextField adalah bawaan dari API Swing. Komponen TextField digunakan oleh kelas boundary *ViewPilih.*
11. Kelas *boundary ViewHasil* bertugas menampilkan antarmuka pengguna untuk melihat hasil pengukuran *reusability* rancangan perangkat lunak. Dalam kelas ini terdapat satu konstruktor dan dua operasi yaitu *PilihMouseClicked* dan *HomeMouseClicked.* Kelas ini memiliki hubungan agregasi dengan kelas model *XMLParser,* kelas *Coupling, Inheritance, Encapsulation, Reusability* dan kelas *boundary ReusMain.* Kelas ini menggunakan komponen *Frame* dari API Swing.
12. Kelas *boundary ReusMain* bertugas sebagai antarmuka utama untuk navigasi ke panel-panel lain. Dalam kelas ini terdapat satu konstruktor dan empat operasi yaitu *initComponents*, *PilihMouseClicked*, *LihatMouseClicked*, dan *main.* Kelas ini memiliki hubungan agregasi dengan kelas *boundary ViewPilih* dan *ViewHasil.* Kelas ini menggunakan komponen *Frame* dari API Swing.

5.1.2 Perancangan Algoritme

Pada perancangan algoritme sistem pengukuran nilai *reusability* ini penulis mengambil empat buah sampel operasi utama yaitu operasi Pilih pada kelas *ViewPilih,* operasi *ValidasiXML* pada kelas *XMLParser,* operasi *getHasil* pada kelas *Reusability,* dan operasi *Parser* pada kelas *XMLParser.*

5.1.2.1 Kelas ViewPilih

Rancangan algoritme operasi *PilihActionPerformed* ditunjukkan dalam Kode Program 5.1.

No	ViewPilih.java
1	Void PilihActionPerformed (evt: ActionEvent)
2	String[] splitAlamat
3	String alamat
4	int returnValue = FileChooser CALL showOpenDialog(this: Object)
5	IF(returnValue = JfileChooser CALL APPROVE_OPTION) THEN
6	FILENAME = FileChooser CALL getSelectedFile
7	String separator = "\\"
8	splitAlamat = FILENAME replace with separator split by "\\\\"
9	alamat = splitAlamat[0] + "\\"
10	FOR(int i; i<splitAlamat.length; i++)
11	IF(i=splitAlamat.length-1){
12	alamat = alamat CONCAT splitAlamat[i]
13	ELSE alamat = alamat CONCAT splitAlamat[i] + "/"
14	END IF
15	FieldPilih CALL setText (alamat)
16	ELSE tfChoose CALL setText(null)
17	END IF

Kode Program 5.1 *Pseudocode* operasi *PilihActionPerformed*

5.1.2.2 Kelas XMLParser

Rancangan algoritme operasi *ValidasiXML* ditunjukkan dalam Kode Program 5.2.

No	XMLParser.java
1	Boolean ValidasiXML (xml: File)
2	DocumentBuilderFactory dbFactory = DocumentBuilderFactory CALL newInstance
3	DocumentBuilder dBuilder = dbFactory CALL newDocumentBuilder
4	Document doc = dBuilder CALL parse(xml: File)
5	doc CALL getDocumentElement, CALL normalize
6	doc CALL getDocumentElement
7	*Node*list strc = doc CALL getElementsByTagName("Project")
8	*Node* n*Node* = strc CALL item(0)
9	Element eElemenet = (Element) n*Node*
10	IF ("simple" != eElement CALL getAttribute(XMLstructure: String)
11	THEN valid = false
12	ELSE valid = true
13	END IF
14	Return valid

Kode Program 5.2 *Pseudocode* operasi *ValidasiXML*

5.1.2.3 Kelas Reusability

Rancangan algoritme operasi *getHasil* ditunjukkan dalam Kode Program 5.3.

No	Reusability.java
1	double Cov = -37.111
2	double getHasil (Coup: double, Inher: double, Encap: double)

```
3   double reusability = 0
4   reusability = Cov + Coup + Inher + Encap
5   return reusability;
```

Kode Program 5.3 *Pseudocode* operasi *getHasil*

5.1.2.4 Kelas XMLParser

Rancangan algoritme operasi *Parser* ditunjukkan dalam Kode Program 5.4.

No	XMLParser.java
1	//MENGHITUNG NILAI COUPLING
2	NodeList asList = CALL element with tag "AssociationEnd"
3	for (int temp = 0; temp < asList.getLength; temp++)
4	Node asNode = asList indeks ke-temp
5	Element asElement = asNode to Elemen
6	if (AttributeNode "Visibility" of asElement != null)
7	jmlAss++
8	NodeList gList = CALL element with tag "Generalization"
9	for (int temp = 0; temp < gList.getLength(); temp++)
10	Node gNode = gList indeks ke-temp
11	Element gElement = gNode to Elemen
12	if (AttributeNode "Visibility" of gElement != null)
13	jmlExAss++
14	NodeList rList = CALL element with tag "Realization"
15	for (int temp = 0; temp < rList.getLength(); temp++)
16	Node rNode = rList indeks ke-temp
17	Element rElement = rNode to Elemen
18	if (AttributeNode "Visibility" of rElement != null)
19	jmlExAss++
20	NodeList uList = CALL element with tag "Usage"
21	for (int temp = 0; temp < uList.getLength(); temp++)
22	Node uNode = uList indeks ke-temp
23	Element uElement = uNode to Elemen
24	if (AttributeNode "Visibility" of uElement != null)
25	jmlExAss++
26	NodeList dList = doc CALL element with tag "Dependency"
27	for (int temp = 0; temp < dList.getLength(); temp++)
28	Node dNode = dList indeks ke-temp
29	Element dElement = dNode to Elemen
30	if (AttributeNode "Visibility" of dElement != null)
31	jmlExAss++
32	NodeList cList = doc CALL element with tag "Class"
33	for (int i = 0; i < cList.getLength(); i++)
34	Node cNode = cList indeks ke-i
35	Element classElement = cNode to Elemen
36	if (AttributeNode "Visibility" of classElement != null)
37	Element cElement = cNode to Elemen
38	jmlClass++
39	jmlExAss = jmlExAss * 2
40	if (jmlClass > 0) {
41	DCC = (double) (jmlAss + jmlExAss) / jmlClass
42	else
43	DCC = 0
44	
45	//MENGHITUNG NILAI INHERITANCE
46	NodeList genList = CALL element with tag "Generalization"
47	for (int i = 0; i < genList.getLength(); i++)
48	Node parentGen = genList indeks ke-i
49	if (nodetype of parentGen = Node.ELEMENT_NODE)
50	Element eElement = parentGen to Elemen
51	if (eElement hasAttribute "To" && eElement hasAttribute "From" && eElement hasAttribute "Visibility")
52	String idfrom = Attribute of eElement"From"
53	String idto = Attribute of eElement "To"

```
54        listIdp.add(idfrom)
55        listIdc.add(idto)
56
57    NodeList classList = CALL element with tag "Class"
58    for (int i = 0; i < classList.getLength(); i++)
59      Node parentClassNode = classList indeks ke-i
60      if (nodetype of parentClassNode = Node.ELEMENT_NODE)
61        Element classElement = parentClassNode to Elemen
62        if (Atribut "Visibility" of classElement != null)
63          String id = Attribute"Id" of classElement
64          for (int h = 0; h < listIdp.size(); h++)
65            String idParent = (String) listIdp.get(h)
66            if (id.equalsIgnoreCase(idParent))
67              ArrayList mfal = new ArrayList()
68              Mfal ADD "default"
69              Element cElement = parentClassNode to Elemen
70              String namakelas1 = Attribute"Name" cElement
71              jmlOpParent = 0
72              NodeList parentcList = childNode of parentClassNode
73              for (int p = 0; p < parentcList.getLength(); p++)
74                if (parentcList.item(p) = "ModelChildren"))
75                  Node modelChildren = parentcList indeks ke-p
76                  NodeList opList = childNode of modelChildren
77                  for (int o = 0; o < opList.getLength(); o++)
78                    if (opList.item(o)="Operation"))
79                      Node operation = opList indeks ke-o
80                      Element opElement = operation to Elemen
81                      if (Atrribute"Name" of operation != namakelas1 &&
    !="Object")
82                        jmlOpParent++
83              fromMFA.add(jmlOpParent)
84          for (int h = 0; h < listIdc.size(); h++)
85            String toId = (String) listIdc.get(h)
86            String fromId = (String) listIdp.get(h)
87            if (id.equalsIgnoreCase(toId)) {
88              Element cElement = parentClassNode to Elemen
89              String namakelas1 = Attribute"Name" of cElement
90              jmlOpChild = 0
91              NodeList parentcList = childNode of parentClassNode
92              for (int p = 0; p < parentcList.getLength(); p++)
93                if (parentcList.item(p)="ModelChildren")
94                  Node modelChildren = parentcList indeks ke-p
95                  NodeList opList = childNode of modelChildren
96                  for (int o = 0; o < opList.getLength(); o++)
97                    if (opList.item(o)="Operation")
98                      jmlOpChild++;
99                      Node paranode = opList indeks ke-o
100                     NodeList paralist = childNode of paranode
101                     Element opElement = paranode to Elemen
102                     toMFA.add(jmlOpChild)
103     Element cElement = parentClassNode to Elemen
104     String namakelas1 = Attribute"Name" of cElement
105     NodeList parentcList = childNode of parentClassNode
106     String status = "-"
107     for (int p = 0; p < parentcList.getLength(); p++)
108       if (parentcList.item(p)="Stereotypes")
109         Node stereotypes = parentcList indeks ke-p
110         NodeList liststereo = childNode of stereotypes
111         for (int o = 0; o < liststereo.getLength(); o++)
112           if (liststereo.item(o)="Stereotype")
113             Node stereo = liststereo indeks ke-o
114             NodeList paralist = childNode of stereo
115             Element stereoElement = stereo to Elemen
116             String classinter = Attribute"Name" of stereoElement
117             if (classinter="Interface")
```

```
                status = "Interface"
      else if (parentcList.item(p)="ModelChildren")
        jmlMethod = 0
        Node modelChildren = parentcList indeks ke-p
        NodeList opList = child Node of modelChildren
        for (int o = 0; o < opList.getLength(); o++)
          if (opList.item(o)="Operation")
            Node paranode = opList indeks ke-o
            NodeList paralist = childNode of paranode
            jmlMethod++
            Element opElement = paranode to Elemen
            if (jmlMethod > 0)
              pembagiMFA++

    for (int i = 0; i < fromMFA.size(); i++)
        double induk = (double) fromMFA.get(i)
        double anak = (double) toMFA.get(i)
        if (induk == 0 && anak == 0)
            inherc = 0
        else
            inherc = induk / (induk + anak)

    tempMFA += inherc
    if (tempMFA > 0 && pembagiMFA > 0)
        MFA = tempMFA / pembagiMFA
    else
        MFA = 0
//END INHERITANCE

//MENGHITUNG NILAI ENCAPSULATION
for (int i = 0; i < classList.getLength(); i++)
  Node classNode = classList indeks ke-i
  Element classElement = classNode to Elemen
  if (AttributeNode"Visibility" of classElement != null)
    NodeList childList = childNode of classNode
    for (int a = 0; a < childList.getLength(); a++)
      if (childList.item(a)="ModelChildren")
        Node modelChildren = childList indeks ke-a
        jmlAtt = 0
        jmlAttPr = 0
        NodeList attList = childNode of modelChildren
        for (int x = 0; x < attList.getLength(); x++)
          if (attList.item(x)="Attribute")
            Node attribute = attList indeks ke-x
            Element attElement = attribute to Elemen
            jmlAtt++
            String visibi = Attribute"Visibility" of attElement
            if (visibi="private")
              jmlAttPr++
            else if (visibi="protected")
              jmlAttPr++
        Att.add(jmlAtt)
        AttPr.add(jmlAttPr)
        if (jmlAtt > 0)
          pembagiDAM++
for (int i = 0; i < Att.size(); i++)
  if ((int) Att.get(i) > 0)
    encapc = (double) (int) AttPr.get(i) / (int) Att.get(i)
  else
    encapc = 0
tempDAM += encapc
```

```
179  if (pembagiDAM > 0)
180      DAM = tempDAM / pembagiDAM
181  else
182      DAM = 0
```

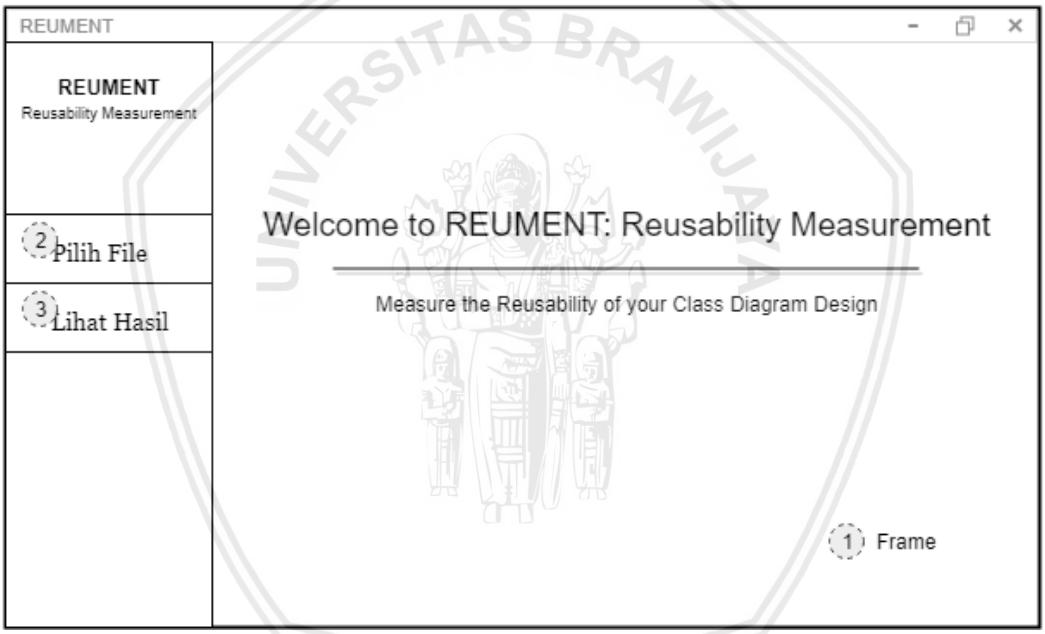
Kode Program 5.4 *Pseudocode* operasi *Parser*

5.1.3 Perancangan Antarmuka Pengguna

Pada pengembangan sistem pengukuran *reusability* ini terdapat tiga halaman yaitu halaman utama, halaman seleksi berkas, dan halaman lihat hasil. Berikut ini adalah perancangan antarmuka pengguna berupa *mock-up* yang telah dirancang dengan menggunakan *layout.*

5.1.3.1 Antarmuka Pengguna Halaman Utama

Rancangan antarmuka pengguna halaman utama berupa *mock-up* ditampilkan dalam Gambar 5.5



Gambar 5.5 *Mock-Up* halaman utama

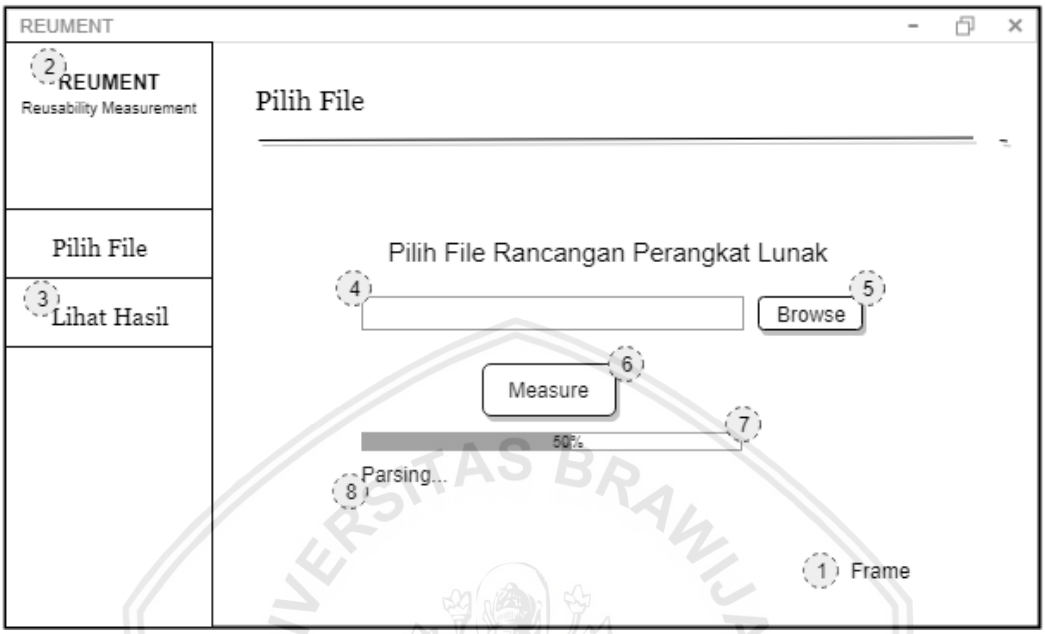
Pada Tabel 5.1 merupakan keterangan komponen-komponen yang harus ada dalam halaman utama.

Tabel 5.1 Keterangan *mock-up* halaman utama

No	Nama Komponen	Tipe Komponen	Keterangan
1	Frame Halaman Utama	Frame	Kontainer yang digunakan menampung komponen isi pada halaman utama
2	Menu 1	Label	Navigasi ke menu Pilih File
3	Menu 2	Label	Navigasi ke menu Lihat Hasil

5.1.3.2 Antarmuka Pengguna Halaman Pilih File

Rancangan antarmuka pengguna halaman pilih file berupa *mock-up* ditampilkan dalam Gambar 5.6



Gambar 5.6 *Mock-Up* halaman pilih file

Pada Tabel 5.2 merupakan keterangan komponen-komponen yang harus ada dalam halaman pilih file.

Tabel 5.2 Keterangan mock-up halaman pilih file

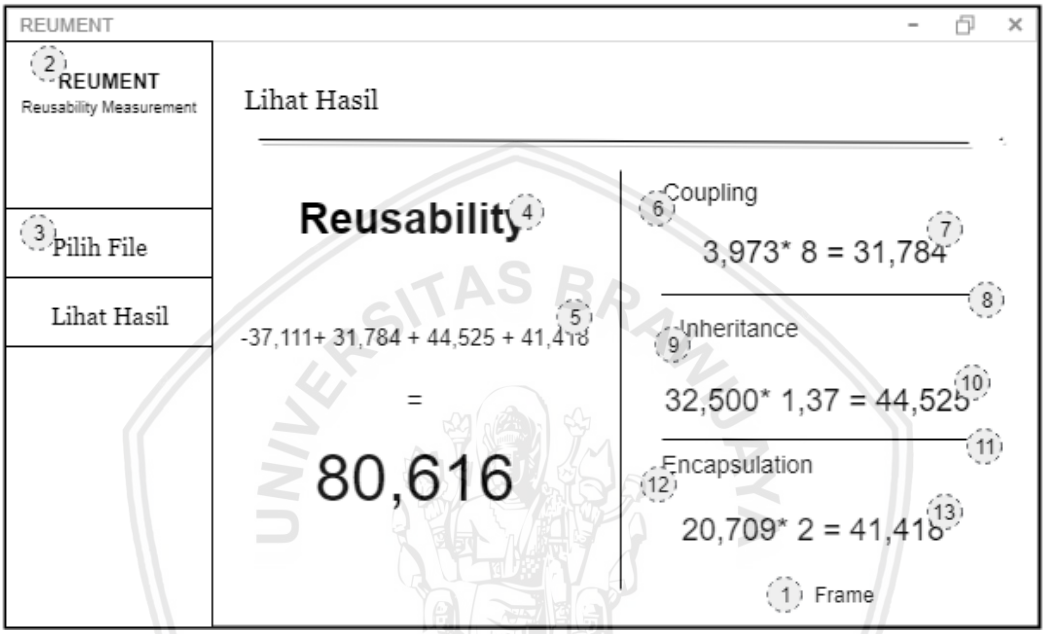
No	Nama Komponen	Tipe Komponen	Keterangan
1	Frame Halaman Utama	Frame	Kontainer yang digunakan untuk menampung komponen isi pada halaman utama
2	Judul	Label	Nama Sistem
3	Menu	Label	Navigasi ke menu Lihat Hasil
4	*Text Field* Alamat Direktori	Text Field	*Field* untuk mengisi alamat direktori berkas yang telah dipilih
5	Tombol *Browse*	Button	Tombol yang digunakan untuk melakukan pemilihan berkas
6	Tombol *Measure*	Button	Tombol yang digunakan untuk melakukan pemrosesan berkas masukan
7	Progres Indikator	Progress Bar	Indikator yang menunjukkan progress *loading*

Tabel 5.2 Keterangan mock-up halaman pilih file (lanjutan)

8	Keterangan progres	Label	Keterangan progres yang sedang berlangsung

5.1.3.3 Antarmuka Pengguna Halaman Lihat Hasil

Rancangan antarmuka pengguna halaman pilih file berupa *mock-up* ditampilkan dalam Gambar 5.7



Gambar 5.7 *Mock-Up* halaman lihat hasil

Pada Tabel 5.3 merupakan keterangan komponen-komponen yang harus ada dalam halaman pilih file.

Tabel 5.3 Keterangan *mock-up* halaman lihat hasil

No	Nama Komponen	Tipe Komponen	Keterangan
1	Frame Halaman Utama	Frame	Kontainer yang digunakan untuk menampung komponen isi pada halaman utama
2	Judul	Label	Nama Sistem
3	Menu	Label	Navigasi ke menu Pilih File
4	Judul hasil 1	Label	Judul hasil *reusability*
5	Hasil 1	Label	Hasil *reusability*
6	Judul hasil 2	Label	Judul hasil poin *coupling*
7	Hasil 2	Label	Hasil poin *coupling*

Tabel 5.3 Keterangan mock-up halaman lihat hasil (lanjutan)

8	Pembatas	Separator	Pembatas hasil *coupling* dan *inheritance*
9	Judul hasil 3	Label	Judul hasil poin *inheritance*
10	Hasil 3	Label	Hasil poin *inheritance*
11	Pembatas	Separator	Pembatas hasil *inheritance* dan *encapsulation*
12	Judul hasil 4	Label	Judul hasil poin *encapsulation*
13	Hasil 4	Label	Hasil poin *encapsulation*
14	Pembatas	Separator	Pembatas hasil *reusability* dan rincian poinnya

5.2 Implementasi Sistem

Pada implementasi sistem dijelaskan mengenai spesifikasi sistem, lingkungan pengembangan, batasan implementasi, kode program, dan hasil implementasi dalam bentuk antarmuka pengguna.

5.2.1 Spesifikasi Sistem

Aplikasi untuk mengukur nilai *reusability* dikembangkan sesuai dengan algoritme pada *reusability metric* persamaan (2.1). *Reusability* dipengaruhi oleh atribut rancangan perangkat lunak yaitu *coupling, inheritance,* dan *encapsulation. Metric* yang terlibat dalam persamaan (2.1) adalah DCC untuk *coupling,* MFA untuk *inheritance,* dan DAM untuk *encapsulation.* Spesifikasi sistem berdasarkan lingkungan perangkat keras, lingkungan perangkat lunak dan lingkungan sistem operasi dijelaskan berikut ini.

5.2.1.1 Lingkungan Perangkat Keras

Pada pengembangan aplikasi pengukuran *reusability* ini menggunakan perangkat keras dengan spesifikasi seperti pada Tabel 5.4.

Tabel 5.4 Lingkungan perangkat keras

System Modell	Asus A456U
Processor	Intel Core i5-7th Generation 3.18GHz
Graphic	NVIDIA-GEFORCE 940MX
Memory	4GB
HDD	1TB

5.2.1.2 Lingkungan Perangkat Lunak

Pada pengembangan aplikasi pengukuran *reusability* ini menggunakan perangkat lunak dengan spesifikasi seperti pada Tabel 5.5.

Tabel 5.5 Lingkungan perangkat lunak

IDE	Netbeans 8.2
Programming Language	Java
Library/API/Framework	JAXP, IO, AWT, Swing
XML maker	Visual Paradigm 15.2 Community Edition
Modeling Tool	Draw.io, Astah UML

5.2.1.3 Lingkungan Sistem Operasi

Pengembangan aplikasi pengukuran *reusability* ini menggunakan Sistem Operasi Windows 10.

5.2.2 Batasan Implementasi

Berikut ini adalah batasan dalam implementasi sistem pengukuran *reusability*:

1. Rancangan perangkat lunak hanya dalam bentuk *class diagram.*
2. Masukan yang diterima adalah dokumen XML dengan struktur *simple* dari Visual Paradigm.
3. Sistem tidak menentukan batasan nilai *reusability* yang baik atau buruk, hanya mengukur nilai saja.

5.2.3 Kode Sumber

Berikut ini adalah beberapa contoh kode sumber dari hasil implementasi aplikasi pengukuran *reusability.* Contoh yang diuraikan disini meliputi kode sumber dari kelas *ViewPilih,* kelas *XMLParser,* dan *Reusability.*

Pada Kode Program 5.4 adalah contoh kode sumber dari kelas *ViewPilih:*

No	ViewPilih.java
1	private void PilihActionPerformed(java.awt.event.ActionEvent evt) {
2	String[] splitAlamat;
3	String alamat;
4	int returnValue = openFileChooser.showOpenDialog(this);
5	if (returnValue == JFileChooser.APPROVE_OPTION) {
6	FILENAME = openFileChooser.getSelectedFile().toString();
7	String separator = "\\";
8	splitAlamat = FILENAME.replaceAll(Pattern.quote(separator), "\\\\").split("\\\\");
9	alamat = splitAlamat[0] + "\\";
10	for (int i = 1; i < splitAlamat.length; i++) {
11	if (i == splitAlamat.length - 1) {
12	alamat = alamat.concat(splitAlamat[i]);

```
13                      } else {
14                          alamat = alamat.concat(splitAlamat[i] +
"/");
15                      }
16                  }
17                  FieldPilih.setText(alamat);
18              } else {
19                  tfChoose.setText(null);
20              }
21      }
```

Kode Program 5.5 Kode sumber kelas *ViewPilih* operasi *PilihActionPerformed*

Pada Kode Program 5.5 adalah contoh kode sumber dari kelas *XMLParser:*

No	XMLParser.java
1	public boolean ValidasiXML(File xml) throws ParserConfigurationException, SAXException, IOException {
2	DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
3	DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
4	Document doc = dBuilder.parse(xml);
5	doc.getDocumentElement().normalize();
6	doc.getDocumentElement();
7	NodeList strc = doc.getElementsByTagName("Project");
8	Node nNode = strc.item(0);
9	Element eElement = (Element) nNode;
10	if (!"simple".equalsIgnoreCase(eElement.getAttribute("Xml_structure"))) {
11	valid = false;
12	} else {
13	valid = true;
14	}
15	return valid;
16	}

Kode Program 5.6 Kode sumber kelas *XMLParser* operasi *ValidasiXML*

Pada Kode Program 5.6 adalah contoh kode sumber dari kelas *Reusability:*

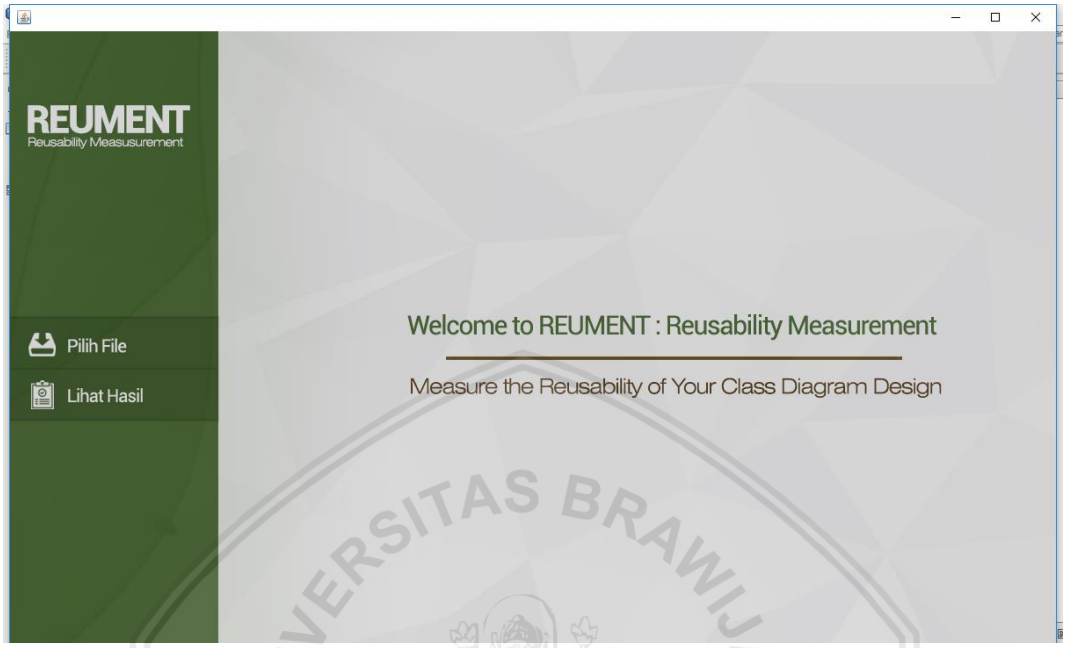
No	Reusability.java
1	double Cov=-37.111;
2	public double getHasil(double Coup, double Inher, double Encap){
3	double reusability = 0;
4	reusability = Cov + Coup + Inher + Encap;
5	return reusability;
6	}

Kode Program 5.7 Kode sumber kelas *Reusability* operasi *getHasil*

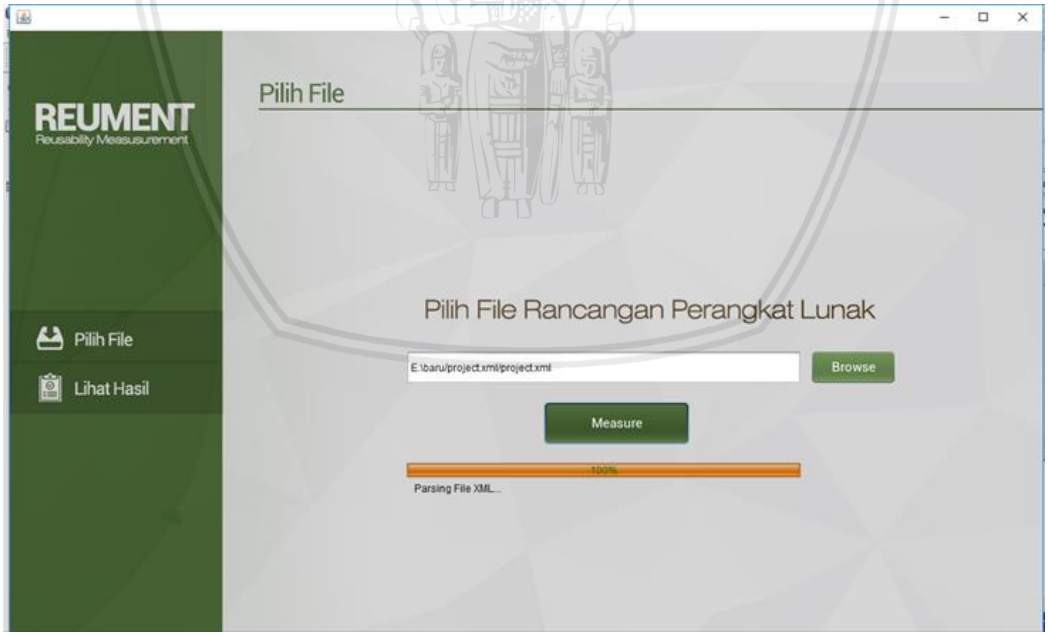
5.2.4 Hasil Implementasi

Hasil dari implementasi aplikasi pengukuran *reusability* dan bagaimana kebutuhan fungsional seperti *browse* berkas, ukur poin *coupling,* ukur poin *inheritance,* ukur poin *encapsulation,* ukur *reusability,* dan lihat hasil ditampilkan pada bagian ini. Gambar 5.8 merupakan halaman awal ketika aplikasi dibuka. Gambar 5.9 merupakan halaman seleksi berkas dan mulai pengukuran yang merupakan hasil dari implementasi kebutuhan *browse* berkas, ukur poin *coupling,* ukur poin *inheritance,* ukur poin *encapsulation,* dan ukur *reusability*. Gambar 5.10 adalah halaman lihat hasil yang merupakan hasil dari implementasi kebutuhan

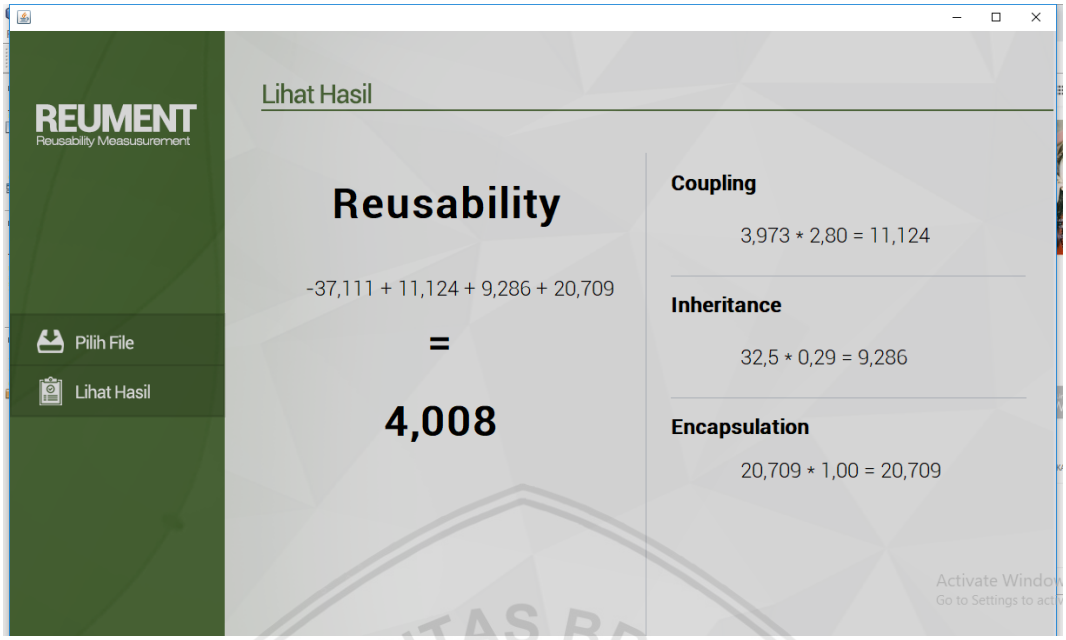
lihat hasil. Pada halaman tersebut menampilkan hasil dari pengukuran *reusability* beserta dengan rincian pengukurannya yaitu poin *coupling*, poin *inheritance*, dan poin *encapsulation*.



Gambar 5.8 Tampilan halaman awal aplikasi



Gambar 5.9 Tampilan halaman pilih file



Gambar 5.10 Tampilan halaman lihat hasil

BAB 6 PENGUJIAN SISTEM DAN PEMBAHASAN HASIL

6.1 Pengujian Unit

Pengujian unit pada bagian ini menggunakan metode *white box testing.* Pengujian unit diambil tiga buah operasi antara lain operasi *PilihActionPerformed* dari kelas *ViewPilih, ValidasiXML* dari kelas *XMLParser,* dan *getHasil* dari kelas *Reusability.*

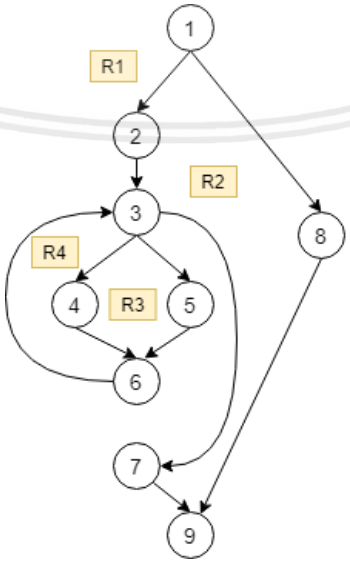
6.1.1 Pengujian Unit Operasi *PilihActionPeformed*

Identifikasi jalur kasus uji untuk operasi *PilihActionPerformed* dari kelas *ViewPilih* seperti dalam Gambar 6.1

```
1  Void PilihActionPerformed (evt: ActionEvent)
   String[] splitAlamat
   String alamat
   int returnValue = FileChooser CALL showOpenDialog(this: Object)
   IF(returnValue = JfileChooser CALL APPROVE_OPTION) THEN
2        FILENAME = FileChooser CALL getSelectedFile
         String separator = "\\"
         splitAlamat = FILENAME replace with separator split by "\\\\"
         alamat = splitAlamat[0] + "\\"
3        FOR(int i; i<splitAlamat.length; i++)
4           IF(i=splitAlamat.length-1){
5              alamat = alamat CONCAT splitAlamat[i]
6           ELSE alamat = alamat CONCAT splitAlamat[i] + "/"
7           END IF
         FieldPilih CALL setText (alamat)
8  ELSE tfChoose CALL setText(null)
9  END IF
```

Gambar 6.1 Algoritme operasi *PilihActionPerformed*

Berdasarkan gambar 6.1 didapatkan *flow graph* untuk operasi *PilihActionPerformed* seperti dalam Gambar 6.2



Gambar 6.2 *Flow graph* operasi *PilihActionPerformed*

Berdasarkan Gambar 6.2 didapatkan hasil *cyclomatic complexity* seperti berikut:

a. V(G) = R = 4
b. V(G) = E – N + 2 = 11-9+2 = 4
c. V(G) = 3 + 1 = 3+1 = 4

Sehingga didapatkan *independent path* untuk kasus uji sebagai berikut:

a. Jalur 1 = 1-8-9
b. Jalur 2 = 1-2-3-4-6-7-9
c. Jalur 3 = 1-2-3-5-6-7-9
d. Jalur 4 = 1-2-3-7-9

Dari *independent path* yang telah didapat, kasus uji untuk operasi *PilihActionPerformed* adalah seperti pada Tabel 6.1

Tabel 6.1 Hasil pengujian unit operasi *PilihActionPerformed*

Jalur	Prosedur Uji	Ekspektasi Hasil	Hasil Aktual	Status
1	Kelas driver memanggil method PilihActionPerformed() dengan variabel returnValue=0	tfChoose = null	tfChoose = null	Valid
2	Kelas driver memanggil method PilihActionPerformed() dengan variabel returnValue=1, FILENAME= "E:\\file\\project.xml"	Alamat direktori berkas dipecah kemudian digabung kembali dan pada kata terakhir tidak ditambah “/”	Alamat direktori berkas dipecah kemudian digabung kembali dan pada kata terakhir tidak ditambah “/”	Valid
3	Kelas driver memanggil method PilihActionPerformed() dengan variabel returnValue=1, FILENAME= "E:\\file\\project.xml"	Alamat direktori berkas dipecah kemudian digabung kembali dengan karakter penghubung “/”	Alamat direktori berkas dipecah kemudian digabung kembali dengan karakter penghubung “/”	Valid
4	Kelas driver memanggil method PilihActionPerformed() dengan variabel returnValue=1,	FieldPilih = “E:\file/project.xml”	FieldPilih = “E:\file/project.xml”	Valid

Tabel 6.1 Hasil pengujian unit operasi *PilihActionPerformed* (lanjutan)

	FILENAME= "E:\\file\\project.xml"			

6.1.2 Pengujian Unit Operasi *ValidasiXML*

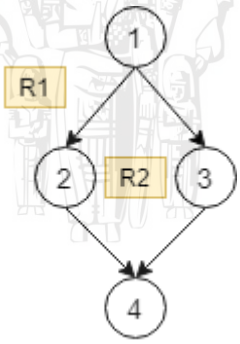
Identifikasi jalur kasus uji untuk operasi *ValidasiXML* dari kelas *XMLParser* seperti dalam Gambar 6.3

```
Boolean ValidasiXML (xml: File)
DocumentBuilderFactory dbFactory = DocumentBuilderFactory CALL newInstance
DocumentBuilder dBuilder = dbFactory CALL newDocumentBuilder
Document doc = dBuilder CALL parse(xml: File)
doc CALL getDocumentElement, CALL normalize
doc CALL getDocumentElement
Nodelist strc = doc CALL getElementsByTagName("Project")
Node nNode = strc CALL item(0)
Element eElemenet  = (Element) nNode
IF ("simple"  != eElement CALL getAttribute(XMLstructure: String)
     THEN valid = false
ELSE valid = true
ENDIF
Return valid
```

1 2 3 4

Gambar 6.3 Algoritme operasi *ValidasiXML*

Berdasarkan gambar 6.3 didapatkan *flow graph* untuk operasi *ValidasiXML* seperti dalam Gambar 6.4



Gambar 6.4 *Flow graph* operasi *ValidasiXML*

Berdasarkan Gambar 6.4 didapatkan hasil *cyclomatic complexity* seperti berikut:

a. V(G) = R = 2
b. V(G) = E – N + 2 = 4-4+2 = 2
c. V(G) = P + 1 = 1+1 = 2

Sehingga didapatkan *independent path* untuk kasus uji sebagai berikut:

a. Jalur 1 = 1-2-4
b. Jalur 2 = 1-3-4

Dari *independ path* yang telah didapat, kasus uji untuk operasi *ValidasiXML* adalah seperti pada Tabel 6.2

Tabel 6.2 Hasil pengujian unit operasi *ValidasiXML*

Jalur	Prosedur Uji	Ekspektasi Hasil	Hasil Aktual	Status
1	Kelas driver memanggil method ValidasiXML() dengan variabel doc = "E:\\classdiagram uji\\traditional\\project.xml"	valid = false	valid = false	Valid
2	Kelas driver memanggil method ValidasiXML() dengan variabel doc = "E:\\classdiagram uji\\baru\\project.xml"	valid = true	valid = true	Valid

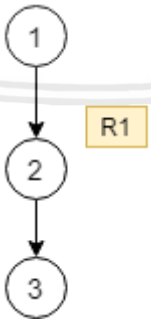
6.1.3 Pengujian Unit Operasi *getHasil*

Identifikasi jalur kasus uji untuk operasi *getHasil* dari kelas *Reusability* seperti dalam Gambar 6.5

```
double getHasil (Coup: double, Inher: double, Encap: double)
1 double reusability = 0
2 reusability = Cov + Coup + Inher + Encap
3 return reusability;
```

Gambar 6.5 Algoritme operasi *getHasil*

Berdasarkan gambar 6.5 didapatkan *flow graph* untuk operasi *getHasil* seperti dalam Gambar 6.6



Gambar 6.6 *Flow Graph* Operasi *getHasil*

Berdasarkan Gambar 6.6 didapatkan hasil *cyclomatic complexity* seperti berikut:

a. V(G) = R = 1

b. V(G) = E – N + 2 = 1–2+2 =1

c. V(G) = P + 1 = 0+1 = 1

Sehingga didapatkan *independent path* untuk kasus uji sebagai berikut:

e. Jalur 1 = 1-2

Dari *indepent path* yang telah didapat, kasus uji untuk operasi *getHasil* adalah seperti pada Tabel 6.3

Tabel 6.3 Hasil pengujian unit operasi *getHasil*

Jalur	Prosedur Uji	Ekspektasi Hasil	Hasil Aktual	Status
1	Kelas driver memanggil method getHasil() dengan argumen Coup = 8, Inher = 18, Encap = 19.	Reusability = 7, 889	Reusability = 7, 889	Valid

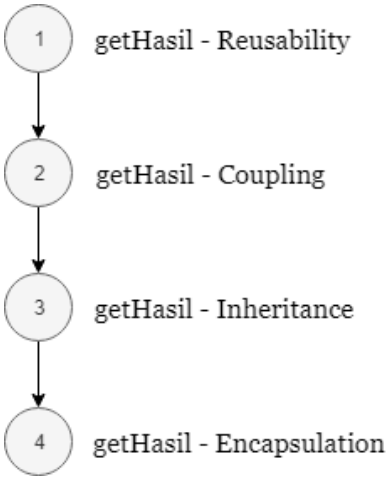
6.2 Pengujian Integrasi

Pengujian integrasi adalah pengujian dengan menggabungkan unit-unit untuk membentuk satu fungsional. Pengujian integrasi pada penulisan ini dilakukan dengan cara menguji hubungan antar kelas menggunakan pendekatan *top-down* yang meliputi kelas *Reusability,* kelas *Inheritance,* kelas *Coupling* dan kelas *Encapsulation* untuk menjalankan fungsi ukur *reusability.* Pada Tabel 6.4 menunjukkan identifikasi kelas/modul yang diuji.

Tabel 6.4 Identifikasi kelas pengujian integrasi

Operasi	Kelas	Tujuan
getHasil(double Coup, double Inher, double Encap)	*Reusability*	Mengukur nilai *reusability* berdasarkan masukkan dari pengguna
getHasil(double coupling)	*Coupling*	
getHasil(double inheritance)	*Inheritance*	
getHasil(double encapsulation)	*Encapsulation*	

Berdasarkan pada Tabel 6.4 maka didapatkan model pengujian integrasi dalam Gambar 6.7. Berdasarkan model *top-down* pada Gambar 6.7 menunjukan interaksi antara *node*1 ke *node*2 pada operasi *getHasil* dari *Coupling* akan memberikan hasil poin *coupling* yang menjadi masukan parameter *Coup* dari operasi *getHasil* pada kelas *Reusability*. Kemudian interaksi antara *node*1 dan node2 ke *node*3 pada operasi *getHasil* pada *Inheritance* akan memberikan hasil poin *inheritance* yang menjadi masukan parameter *Inher* dari operasi *getHasil* pada kelas *Reusability*. Dan interaksi *node*1, node2, dan node3 ke *node* 4 pada operasi *getHasil* pada *Encapsulation* akan memberikan hasil poin *encapsulation* yang menjadi masukan parameter *Encap* dari operasi *getHasil* pada kelas *Reusability*. Maka didapatkan langkah pengujian integrasi seperti pada Tabel 6.5.



Gambar 6.7 Model *Top-Down Testing*

Tabel 6.5 Langkah pengujian integrasi

No	Langkah Pengujian	Keterangan
1	Node1 + Node2	*Operasi getHasil*(*double coupling*) pada kelas *Coupling* dipanggil operasi *getHasil*(*double Coup, double Inher, double Encap*) pada kelas *Reusability*
2	(Node1 + Node2) + Node3	*Operasi getHasil*(*double inheritance*) pada kelas *Inheritance* dipanggil operasi *getHasil*(*double Coup, double Inher, double Encap*) pada kelas *Reusability* yang telah terdapat nilai *getHasil*(*double coupling*) pada kelas *Coupling*
3	(Node1 + Node2 + Node3) + Node4	*Operasi getHasil*(*double encapsulation*) pada kelas *Encapsulation* dipanggil operasi *getHasil*(*double Coup, double Inher, double Encap*) pada kelas *Reusability* yang telah terdapat nilai *getHasil*(*double coupling*) pada kelas *Coupling* dan nilai *getHasil(double inherit)* pada kelas *Inheritance.*

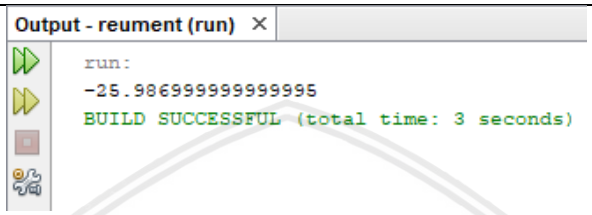
Dari langkah pada Tabel 6.5 selanjutnya dibuat stub pengganti dalam operasi *getHasil* dari kelas *Reusability.* Pada Tabel 6.6 dilakukan penggunaan *stub* untuk menjalankan langkah 1.

Tabel 6.6 Integrasi langkah 1

```
public class Reusability {
    double Cov=-37.111;
    public double getHasil(double Coup, double Inher, double Encap){
```

Tabel 6.6 Integrasi langkah 1 (lanjutan)

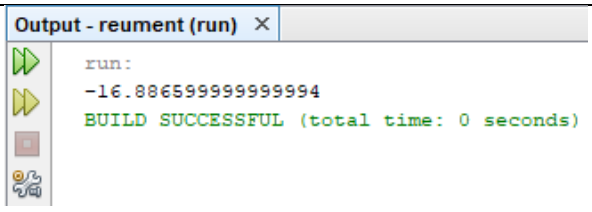
```
        double reusability=0;
        reusability = Cov+Coup+Inher+Encap;
        return reusability;
    }
}
public class Integrasi_testing {
    public static void main(String[] args) {
        Reusability r = new Reusability();
        double stubCoup = 11.1244; //stub untuk mencoba langkah 1
        System.out.println(r.getHasil(stubCoup, 0, 0));
    }
}
```



Hasil integrasi langkah 1 ditunjukkan pada Tabel 6.6 ketika menggunakan *stubCoup* mendapatkan hasil -25.98695 atau tidak 0 pada *getHasil Reusability*. Kemudian pada Tabel 6.7 dilakukan penggunaan *stub* untuk menjalankan langkah 2.

Tabel 6.7 Integrasi langkah 2

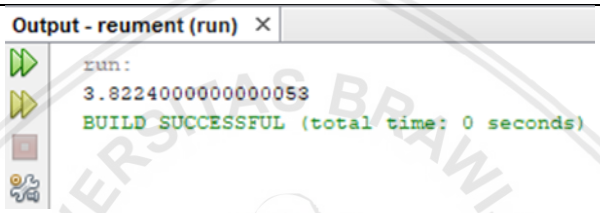
```
public class Reusability {
    double Cov=-37.111;
    public double getHasil(double Coup, double Inher, double Encap){
        double reusability=0;
        reusability = Cov+Coup+Inher+Encap;
        return reusability;
    }
}
public class Integrasi_testing {
    public static void main(String[] args) {
        Reusability r = new Reusability();
        double stubCoup = 11.124;
        double stubInher = 9.1; //stub untuk mencoba langkah 2
        System.out.println(r.getHasil(stubCoup,stubInher, 0));
    }
}
```



Hasil integrasi langkah 2 ditunjukkan pada Tabel 6.7 ketika menggunakan *stubInher* dan *stubCoup* mendapatkan hasil -16.709 pada *getHasil Reusability*. Kemudian pada Tabel 6.8 dilakukan penggunaan *stub* untuk menjalankan langkah3.

Tabel 6.8 Integrasi langkah 3

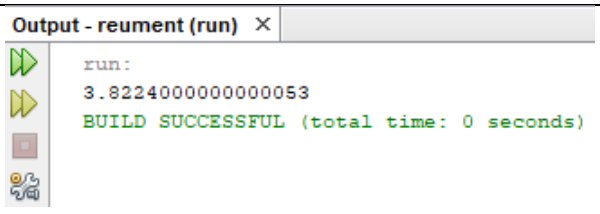
```
public class Reusability {
    double Cov=-37.111;
    public double getHasil(double Coup, double Inher, double Encap){
        double reusability=0;
        reusability = Cov+Coup+Inher+Encap;
        return reusability;
    }
}
public class Integrasi_testing {
    public static void main(String[] args) {
        Reusability r = new Reusability();
        double stubCoup = 11.124;
        double stubInher = 9.286;
        double stubEncap = 20.709; //stub untuk mencoba langkah 3
        System.out.println(r.getHasil(stubCoup, stubInher, stubEncap));
    }
}
```



Hasil integrasi langkah 3 ditunjukkan pada Tabel 6.7 ketika menggunakan *stubEncap, stubInher,* dan *stubCoup* mendapatkan hasil 4.008 pada *getHasil Reusability*. Kemudian *stub* diganti dengan komponen aslinya yaitu memanggil *method* yang sebenarnya seperti pada Tabel 6.9.

Tabel 6.9 Menjalankan integrasi mengganti dengan komponen asli

```
public class Reusability {
    double Cov=-37.111;
    public double getHasil(double Coup, double Inher, double Encap){
        double reusability=0;
        reusability = Cov+Coup+Inher+Encap;
        return reusability;
    }
}
public class Integrasi_testing {
    public static void main(String[] args) {
        Reusability r = new Reusability();
        Coupling c = new Coupling();
        Inheritance i = new Inheritance();
        Encapsulation e = new Encapsulation();
        System.out.println(r.getHasil(c.getHasil(2.8), i.getHasil(0.29)
,e.getHasil(1)));
    }
}
```

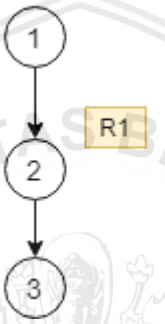


Hasil dari langkah yang ditunjukkan pada Tabel 6.9 ketika menggunakan komponen asli sama dengan hasil langkah 3 pada Tabel 6.10 menggunakan *stub* yang artinya integrasinya berhasil. Dari hasil yang didapatkan secara keseluruhan membentuk algoritme sesuai dalam Gambar 6.8

```
double getHasil (Coup: double, Inher: double, Encap: double)
1 double reusability = 0
2 reusability = Cov + Coup + Inher + Encap
3 return reusability;
```

Gambar 6.8 Algoritme integrasi *getHasil* kelas *Reusability*

Berdasarkan algoritme dalam Gambar 6.8 maka didapatkan *flow graph* operasi *getHasil* seperti dalam Gambar 6.9



Gambar 6.9 *Flow graph* pengujian integrasi *gethasil* kelas *Reusability*

Berdasarkan Gambar 6.9 didapatkan hasil *cyclomatic complexity* seperti berikut:

a. V(G) = R = 1
b. V(G) = E – N + 2 = 2–3+2 =1
c. V(G) = P + 1 = 0+1 = 1

Sehingga didapatkan *independent path* untuk kasus uji sebagai berikut:

f. Jalur 1: 1-2-3

Dari *indepent path* yang telah didapat, maka kasus uji untuk operasi *getHasil* adalah seperti pada Tabel 6.10

Tabel 6.10 Kasus uji integrasi operasi *getHasil*

Jalur	Data Input	Ekspektasi Hasil	Hasil Aktual	Status
1	Coup = 11.1244, Inher = 9.1, Encap = 20.709.	Reusability = 3.8224	Reusability = 3.8224	Valid

6.3 Pengujian Validasi

Pengujian validasi pada bagian ini menggunakan *black box testing* untuk menguji apakah sistem yang dibangun telah memenuhi kebutuhan fungsional dan non-fungsional. Pengujian validasi untuk kebutuhan fungsional terdapat enam kebutuhan antara lain *browse* berkas, ukur poin *coupling,* ukur poin *inheritance,* ukur poin *encapsulation,* ukur *reusability,* dan lihat hasil. Sedangkan pengujian validasi untuk kebutuhan non-fungsional terdapat satu kebutuhan yaitu efisiensi.

6.3.1 Pengujian Validasi *Browse* Berkas

Pengujian validasi kebutuhan *browse* berkas ditunjukkan pada Tabel 6.11 da Tabel 6.12. Pada Tabel 6.11 adalah hasil pengujian validasi dari *main flow kebutuhan Browse* Berkas dan pada Tabel 6.12 adalah hasil pengujian validasi dari *alternative flow kebutuhan Browse* Berkas.

Tabel 6.11 Pengujian validasi *main flow browse* berkas

Kode kebutuhan	F_REUS_001
Kasus uji	Melakukan *Browse* Berkas
Tujuan	Memastikan sistem dapat memenuhi *main flow* kebutuhan fungsional *Browse* Berkas
Data masukan	Berkas XML
Prosedur	1. Membuka halaman pilih file 2. Menekan pada tombol *Browse* 3. Memilih sebuah berkas XML 4. Menekan tombol *Open*
Hasil yang diharapkan	Sistem mengisi *text field* dengan alamat direktori berkas yang dipilih
Hasil	Sistem mengisi *text field* dengan alamat direktori berkas yang dipilih
Status	Valid

Tabel 6.12 Pengujian validasi *alternative flow browse* berkas

Kode kebutuhan	F_REUS_001
Kasus uji	Melakukan *Browse* Berkas
Tujuan	Memastikan sistem dapat memenuhi *alternative flow* kebutuhan fungsional *Browse* Berkas
Data masukan	-
Prosedur	1. Membuka halaman pilih file

Tabel 6.12 Pengujian validasi *alternative flow browse* berkas (lanjutan)

Prosedur	2. Menekan pada tombol *Browse* 3. Memilih sebuah berkas XML 4. Menekan tombol *Cancel*
Hasil yang diharapkan	*Text field* bernilai kosong
Hasil	*Text field* bernilai kosong
Status	Valid

6.3.2 Pengujian Validasi Ukur Poin *Coupling*

Pengujian validasi kebutuhan Ukur Poin *Coupling* ditunjukan pada Tabel 6.13 dan Tabel 6.14. Pada Tabel 6.13 adalah hasil pengujian validasi dari *main flow* kebutuhan Ukur Poin *Coupling* dan pada Tabel 6.14 adalah hasil pengujian validasi dari *alternative flow* kebutuhan Ukur Poin *Coupling*.

Tabel 6.13 Pengujian validasi *main flow* ukur poin *coupling*

Kode kebutuhan	F_REUS_002
Kasus uji	Mengukur Poin *Coupling*
Tujuan	Memastikan sistem dapat memenuhi *main flow* kebutuhan fungsional Ukur Poin *Coupling*
Data masukan	Berkas XML *class diagram* dengan struktur *simple*
Prosedur	1. Membuka halaman pilih file 2. Menekan tombol *Browse* 3. Memilih sebuah berkas XML *class diagram* dengan struktur *simple* Visual Paradigm 4. Menekan tombol *Open* 5. Menekan tombol *Measure*
Hasil yang diharapkan	Sistem berhasil mengukur poin *coupling* berkas XML *class diagram* perangkat lunak
Hasil	Sistem berhasil mengukur poin *coupling* berkas XML *class diagram* perangkat lunak
Status	Valid

Tabel 6.14 Pengujian validasi *alternative flow* ukur poin *coupling*

Kode kebutuhan	F_REUS_002
Kasus uji	Mengukur Poin *Coupling*

Tabel 6.14 Pengujian validasi *alternative flow* ukur poin *coupling* (lanjutan)

Tujuan	Memastikan sistem dapat memenuhi *alternative flow* kebutuhan fungsional Ukur Poin *Coupling*
Data masukan	Berkas XML *class diagram* dengan struktur bukan *simple*
Prosedur	1. Membuka halaman pilih file 2. Menekan tombol *Browse* 3. Memilih sebuah berkas XML *class diagram* dengan struktur *traditional* Visual Paradigm 4. Menekan tombol *Open* 5. Menekan tombol *Measure*
Hasil yang diharapkan	Sistem menampilkan pemberitahuan Struktur XML tidak sesuai
Hasil	Sistem menampilkan pemberitahuan Struktur XML tidak sesuai
Status	Valid

6.3.3 Pengujian Validasi Ukur Poin *Inheritance*

Pengujian validasi kebutuhan Ukur Poin *Inheritance* ditunjukan pada Tabel 6.15 dan Tabel 6.16. Pada Tabel 6.15 adalah hasil pengujian validasi dari *main flow* kebutuhan Ukur Poin *Inheritance* dan pada Tabel 6.16 adalah hasil pengujian validasi dari *alternative flow* kebutuhan Ukur Poin *Inheritance*.

Tabel 6.15 Pengujian validasi *main flow* ukur poin *inheritance*

Kode kebutuhan	F_REUS_003
Kasus uji	mengukur Poin *Inheritance*
Tujuan	Memastikan sistem dapat memenuhi *main flow* kebutuhan fungsional Ukur Poin *Inheritance*
Data masukan	Berkas XML *class diagram* dengan struktur *simple*
Prosedur	1. Membuka halaman pilih file 2. Menekan tombol *Browse* 3. Memilih sebuah berkas XML *class diagram* dengan struktur *simple* Visual Paradigm 4. Menekan tombol *Open* 5. Menekan tombol *Measure*
Hasil yang diharapkan	Sistem berhasil mengukur poin *inheritance* berkas XML *class diagram* perangkat lunak

Tabel 6.15 Pengujian validasi *main flow* ukur poin *inheritance* (lanjutan)

Hasil	Sistem berhasil mengukur poin *inheritance* berkas XML *class diagram* perangkat lunak
Status	Valid

Tabel 6.16 Pengujian validasi *alternative flow* ukur poin *inheritance*

Kode kebutuhan	F_REUS_003
Kasus uji	Mengukur Poin *Inheritance*
Tujuan	Memastikan sistem dapat memenuhi *alternative flow* kebutuhan fungsional Ukur Poin *Inheritance*
Data masukan	Berkas XML *class diagram* dengan struktur bukan *simple*
Prosedur	1. Membuka halaman pilih file 2. Menekan tombol *Browse* 3. Memilih sebuah berkas XML *class diagram* dengan struktur *traditional* Visual Paradigm 4. Menekan tombol *Open* 5. Menekan tombol *Measure*
Hasil yang diharapkan	Sistem menampilkan pemberitahuan Struktur XML tidak sesuai
Hasil	Sistem menampilkan pemberitahuan Struktur XML tidak sesuai
Status	Valid

6.3.4 Pengujian Validasi Ukur Poin *Encapsulation*

Pengujian validasi kebutuhan Ukur Poin *Encapsulation* ditunjukan pada Tabel 6.17 dan Tabel 6.18. Pada Tabel 6.17 adalah hasil pengujian validasi dari *main flow* kebutuhan Ukur Poin *Encapsulation* dan pada Tabel 6.18 adalah hasil pengujian validasi dari *alternative flow* kebutuhan Ukur Poin *Encapsulation*.

Tabel 6.17 Pengujian validasi *main flow* ukur poin *encapsulation*

Kode kebutuhan	F_REUS_004
Kasus uji	Mengukur Poin *Encapsulation*
Tujuan	Memastikan sistem dapat memenuhi *main flow* kebutuhan fungsional Ukur Poin *Encapsulation*
Data masukan	Berkas XML *class diagram* dengan struktur *simple*

Tabel 6.17 Pengujian validasi *main flow* ukur poin *encapsulation* (lanjutan)

Prosedur	1. Membuka halaman pilih file 2. Menekan tombol *Browse* 3. Memilih sebuah berkas XML *class diagram* dengan struktur *simple* Visual Paradigm 4. Menekan tombol *Open* 5. Menekan tombol *Measure*
Hasil yang diharapkan	Sistem berhasil mengukur poin *encapsulation* berkas XML *class diagram* perangkat lunak
Hasil	Sistem berhasil mengukur poin *encapsulation* berkas XML *class diagram* perangkatlunak
Status	Valid

Tabel 6.18 Pengujian validasi *alternative flow* ukur poin *encapsulation*

Kode kebutuhan	F_REUS_004
Kasus uji	Mengukur Poin *Encapsulation*
Tujuan	Memastikan sistem dapat memenuhi *alternative flow* kebutuhan fungsional Ukur Poin *Encapsulation*
Data masukan	Berkas XML *class diagram* dengan struktur bukan *simple*
Prosedur	1. Membuka halaman pilih file 2. Menekan tombol *Browse* 3. Memilih sebuah berkas XML *class diagram* dengan struktur *traditional* Visual Paradigm 4. Menekan tombol *Open* 5. Menekan tombol *Measure*
Hasil yang diharapkan	Sistem menampilkan pemberitahuan Struktur XML tidak sesuai
Hasil	Sistem menampilkan pemberitahuan Struktur XML tidak sesuai
Status	Valid

6.3.5 Pengujian Validasi Ukur *Reusability*

Pengujian validasi kebutuhan Ukur *Reusability* ditunjukan pada Tabel 6.19 dan Tabel 6.20. Pada Tabel 6.19 adalah hasil pengujian validasi dari *main flow* kebutuhan Ukur *Reusability* dan pada Tabel 6.20 adalah hasil pengujian validasi dari *alternative flow* kebutuhan Ukur *Reusability*.

Tabel 6.19 Pengujian validasi *main flow* ukur *reusability*

Kode kebutuhan	F_REUS_005
Kasus uji	Mengukur *Reusability*
Tujuan	Memastikan sistem dapat memenuhi *main flow* kebutuhan fungsional Ukur *Reusability*
Data masukan	Berkas XML *class diagram* dengan struktur *simple*
Prosedur	1. Membuka halaman pilih file 2. Menekan tombol *Browse* 3. Memilih sebuah berkas XML *class diagram* dengan struktur *simple* Visual Paradigm 4. Menekan tombol *Open* 5. Menekan tombol *Measure*
Hasil yang diharapkan	Sistem berhasil mengukur *reusability* berkas XML *class diagram* perangkat lunak
Hasil	Sistem berhasil mengukur *reusability* berkas XML *class diagram* perangkat lunak
Status	Valid

Tabel 6.20 Pengujian validasi *alternative flow* ukur *reusability*

Kode kebutuhan	F_REUS_005
Kasus uji	Mengukur *Reusability*
Tujuan	Memastikan sistem dapat memenuhi *alternative flow* kebutuhan fungsional Ukur *Reusability*
Data masukan	Berkas XML *class diagram* dengan struktur bukan *simple*
Prosedur	1. Membuka halaman pilih file 2. Menekan tombol *Browse* 3. Memilih sebuah berkas XML *class diagram* dengan struktur *traditional* Visual Paradigm 4. Menekan tombol *Open* 5. Menekan tombol *Measure*
Hasil yang diharapkan	Sistem menampilkan pemberitahuan Struktur XML tidak sesuai
Hasil	Sistem menampilkan pemberitahuan Struktur XML tidak sesuai
Status	Valid

6.3.6 Pengujian Validasi Lihat Hasil

Berikut ini merupakan pengujian validasi untuk kebutuhan Lihat Hasil. Pada Tabel 6.21 merupakan hasil pengujian validasi dari *main flow* kebutuhan Lihat Hasil.

Tabel 6.21 Pengujian validasi *main flow* lihat hasil

Kode kebutuhan	F_REUS_006
Kasus uji	Melihat Hasil Pengukuran
Tujuan	Memastikan sistem dapat memenuhi *main flow* kebutuhan fungsional Lihat Hasil
Data masukan	-
Prosedur	1. Menekan tombol navigasi yang menuju ke halaman Lihat Hasil
Hasil yang diharapkan	Sistem menampilkan halaman Lihat Hasil yang berisi hasil dari pengukuran *reusability* beserta rinciannya poin *Coupling,* poin *Inheritance,* dan poin *Encapsulation.*
Hasil	Sistem menampilkan halaman Lihat Hasil yang berisi hasil dari pengukuran *reusability* beserta rinciannya poin *Coupling,* poin *Inheritance,* dan poin *Encapsulation.*
Status	Valid

6.3.7 Pengujian Validasi Efisiensi Sistem

Pengujian efisiensi sistem dilakukan untuk mengetahui perbandingan waktu perhitungan *reusability* rancangan perangkat lunak dengan menggunakan sistem dan perhitungan manual. Pada pengujian ini dilakukan oleh seorang pengguna menggunakan empat berkas uji dengan jumlah kelas yang berbeda-beda. Berkas uji yang digunakan sebagai masukan adalah dari proyek perangkat lunak *open source.* Pada Tabel 6.22 adalah hasil dari pengujian efisiensi sistem.

Tabel 6.22 Hasil pengujian efisiensi sistem

Tugas	Jumlah kelas	*Computation Time System* (dalam detik)	Perhitungan Manual (dalam detik)
Mengukur *reusability* berkas uji I	7	1	101
Mengukur *reusability* berkas uji II	10	2	189
Mengukur *reusability* berkas uji III	22	3	247

Tabel 6.22 Hasil pengujian efisiensi sistem (lanjutan)

Mengukur *reusability* berkas uji IV	39	2	337
Time based efficiency		0,583 tugas/detik	0,0055 tugas/detik
Overall relative efficiency		100%	100%

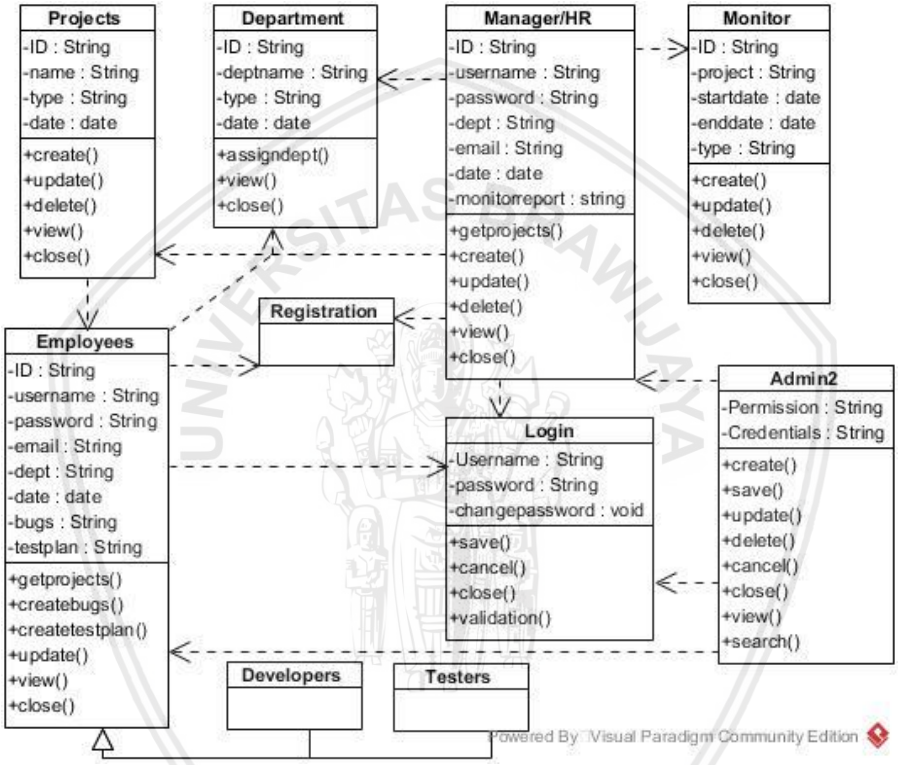
Hasil yang telah ditunjukkan pada Tabel 6.21 untuk *computation time system* pada berkas uji sederhana dan kompleks menghasilkan *time based efficiency* 0,583 tugas/detik sementara perhitungan secara manual menghasilkan 0,0055 tugas/detik. Untuk hasil *overall relative efficiency* perhitungan sistem dan perhitungan manual sama-sama menghasilkan 100% karena pengguna berhasil menyelesaikan semua tugas. Berdasarkan nilai tersebut maka efisiensi perhitungan menggunakan sistem lebih baik daripada perhitungan manual. Dari hasil yang didapatkan maka dapat disimpulkan efisiensi perhitungan *reusability* menggunakan sistem telah memenuhi kebutuhan non-fungsional yaitu sistem memiliki efisiensi lebih dari 80% yaitu 100% dapat melakukan perhitungan dan secara waktu lebih cepat dari perhitungan manual baik untuk berkas uji sederhana maupun kompleks.

6.4 Pembahasan Hasil

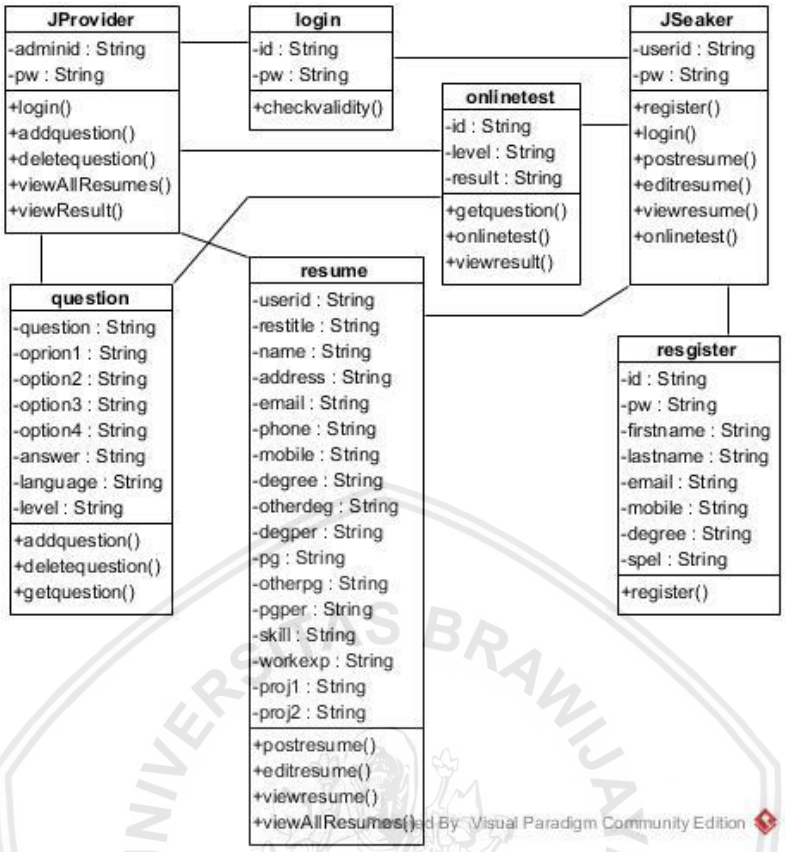
Berdasarkan pengujian yang telah dilakukan pada sistem perhitungan *reusability* menghasilkan status valid untuk keseluruhan kasus uji, sehingga sistem yang dibangun dapat digunakan untuk kakas bantu perhitungan nilai *reusability.* Pengujian unit menguji tiga *method* dari sistem yaitu *method PilihActionPerformed* dari kelas *ViewPilih, method* ValidasiXML dari kelas *XMLParser,* dan *method getHasil* dari kelas *Reusability,* semua kasus uji yang dihasilkan mendapatkan status valid. Pengujian integrasi menguji integrasi antara *method getHasil* dari kelas *Reusability,* kelas *Coupling,* kelas *Inheritance,* dan kelas *Encapsulation* menghasilkan status valid. Untuk pengujian validasi telah diuji semua kebutuhan fungsional termasuk kondisi alternatifnya dan mendapatkan hasil valid untuk semua kasus uji. Selain kebutuhan fungsional, pengujian validasi juga menguji kebutuhan non-fungsional yang mendapatkan hasil valid.

Pada bagian ini akan dijelaskan sistem yang diuji rancangan *class diagram* nya yaitu sistem *Bug Tracking* dan juga *Career Information Management* yang merupakan proyek *open source. Class diagram* dari sistem *Bug Tracking* adalah seperti dalam Gambar 6.10. Dalam sistem *Bug Tracking* terdapat 10 kelas dan dihasilkan *coupling* sebesar 2,8, *inheritance* sebesar 0,29, dan *encapsulation* sebesar 1 sehingga *reusability*-nya sebesar 4,008. Sedangkan *class diagram* dari sistem *Career Information Management* adalah seperti dalam Gambar 6.11. Dalam sistem *Career Information Management* terdapat 7 kelas dan dihasilkan *coupling* sebesar 2,57, *inheritance* sebesar 0, dan *encapsulation* sebesar 1 sehingga *reusability*-nya sebesar -6,186.

Hasil minus dari sistem *Career Information Management* disebabkan nilai *inheritance*-nya 0 karena nilai koefisien poin *inheritance* yang paling tinggi yaitu 32,5 sehingga *inheritance* memiliki peran yang besar dalam nilai *reusability,* jadi ketika nilainya 0 dan nilai dari komponen lain kecil juga maka mengakibatkan hasil nilai *reusability* minus. Tidak ada batasan nilai *reusability* seberapa yang termasuk golongan baik atau tidak baik karena belum ada penelitian lebih lanjut mengenai batasan tersebut. Namun sesuai dengan penelitian "*Quantifying Reusability of Object Oriented Design"* oleh Huda dkk (2015) bahwa semakin tinggi nilai *reusability* maka semakin tinggi juga ratingnya.



Gambar 6.10 ***Class diagram bug tracking system***



Gambar 6.11 ***Class diagram career information management system***

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan hasil penelitian dari pengembangan aplikasi kakas bantu perhitungan nilai *reusability* berdasarkan rancangan perangkat lunak diperoleh kesimpulan sebagai berikut:

1. Pengembangan aplikasi kakas bantu perhitungan nilai *reusability* dengan menerapkan *waterfall model* karena pada penelitian ini keseluruhan kebutuhan telah didapatkan diawal dan tidak terjadi perubahan yang signifikan selama proses pengembangan aplikasi. Pada tahap analisis kebutuhan dilakukan proses pengkajian kepustakaan yang kemudian didapatkan kebutuhan fungsional berjumlah 6 dan kebutuhan non-fungsional berjumlah 1. Pada tahap perancangan dihasilkan perancangan algoritme, perancangan arsitektur, dan perancangan antarmuka pengguna. Dari rancangan arsitektur dihasilkan *sequence diagram* berjumlah 6 yang pada dokumen ini dituliskan sampel 3 *sequence diagram* dan *class diagram* yang memuat satu kelas *interface,* tiga kelas *boundary,* empat kelas *controller,* satu kelas *model*, dan tiga kelas dari *API Swing*. Kemudian pada tahap implementasi berhasil mengimplementasikan rancangan arsitektur, rancangan algoritme, dan rancangan antarmuka pengguna. Pada tahap pengujian dilakukan pengujian unit dengan teknik *basis path* yang berhasil menguji semua jalur uji, pengujian integrasi menggunakan pendekatan *top-down* berhasil menunjukkan modul telah terintegrasi, dan pengujian validasi berhasil memvalidasi seluruh kebutuhan fungsional yang berjumlah 6.
2. Pada pengujian efisiensi mendapatkan hasil sistem dapat melakukan perhitungan *reusability* 100% berhasil dan lebih cepat dari perhitungan manual sehingga sistem telah memenuhi kebutuhan non-fungsionalnya.

7.2 Saran

Saran untuk pengembangan lanjut aplikasi kakas bantu perhitungan nilai *reusability* berdasarkan rancangan perangkat lunak adalah sebagai berikut:

1. Sistem dapat dikembangkan supaya *compatible* dengan standar xml yang lain karena untuk saat ini masih tidak fleksibel untuk menerima berkas xml yang di parsing kerena hanya mampu memparsing xml dari Visual Paradigm dengan format *simple*.
2. Metrik-metrik yang digunakan dalam pengukuran *reusability* menggunakan metrik *Quality Model for Object-Oriented Design* (QMOOD) yang masih bisa ditingkatkan menggunakan metrik-metrik yang lebih mutakhir.

DAFTAR REFERENSI

Ashdown, L., Greenberg, J., & Melnick, J. (2014). *Oracle XML Developer's Kit Programmer's Guide 11.1 Release.* Oracle.

Bansiya, J. dan Davis, C.G., 2002. A Hierarchical Model for Object-Oriented Design Quality Assessment. 28(1), hal.4–17.

Booch, G., Jacobson, I. dan Rumbaugh, J., 1999. *The Unified Modeling Language Reference Manual*. [daring] Tersedia pada: <http://www.citeulike.org/group/1374/article/3944245>.

Cole, B., dkk. (2002). *Java Swing, 2nd Edition.* Sebastopol: O'Reilly.

Fujita, H. & Pisanelli, D. M., 2007. *New Trends in Software Methodologies, Tools and Techniques.* 161 penyunt. Netherlands: IOS Press.

Goyal, N. dan Gupta, E.D., 2014. Reusability Calculation of Object Oriented Software Model by Analyzing CK Metric. 3(7), hal.2466–2470.

Huda, M., Sharma Arya, Y.D. dan Hasan Khan, M., 2015. Quantifying Reusability of Object Oriented Design: A Testability Perspective. *Journal of Software Engineering and Applications*, [daring] 08(04), hal.175–183. Tersedia pada: <http://www.scirp.org/journal/PaperDownload.aspx?DOI=10.4236/jsea.2015.84018>.

Nair, T.R.G. dan Selvarani, R., 2010. Estimation of software reusability. *ACM SIGSOFT Software Engineering Notes*, 35(1), hal.1.

Padhy, N., Singh, R.P. dan Satapathy, S.C., 2018. Software reusability metrics estimation: Algorithms, models and optimization techniques. *Computers and Electrical Engineering*, 69, hal.653–668.

Pressman, R.S., 2001. *Software Engineering: A Practitioner's Approach*.

Sandhu, P.S., Kaur, H. dan Singh, A., 2018. Modeling of Reusability of Object Oriented Software System. hal.1–5.

Sergeev, A., 2010. *Efficiency.* [Online] Available at: http://ui-designer.net/usability/efficiency.htm [Diakses 12 2 2019].

Sommerville, I. (2011). *Software Engineering 9th Edition.* Boston: Addison-Wesley.

Tomas, P., Escalona, M.J. dan Mejias, M., 2013. Open source tools for measuring the Internal Quality of Java software products. A survey. *Computer Standards and Interfaces*, [daring] 36(1), hal.244–255. Tersedia pada: <http://dx.doi.org/10.1016/j.csi.2013.08.006>.

Ubaidillah, M., Pradana, F. & Priyambadha, B., 2017. Kakas Bantu Perhitungan Nilai Kopling Menggunakan Metrik Cognitive Weighted Coupling Between Object. *KINETIK,* II(1), pp. 63-62.

Washizaki, H., Koike, T., Namiki, R. dan Tanabe, H., 2012. Reusability metrics for program source code written in C language and their evaluation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7343 LNCS, hal.89–103.