PENGUKURAN KUALITAS *MAINTAINABILITY* APLIKASI MENGGUNAKAN METRIKS *CHIDAMBER* DAN *KEMERER* (STUDI KASUS SISTEM INFORMASI SKRIPSI INFORMATIKA UNIVERSITAS DIPONEGORO)

SKRIPSI

Disusun Sebagai Salah Satu Syarat
untuk Memperoleh Gelar Sarjana Komputer
pada Departemen Ilmu Komputer/Informatika

Disusun oleh :

Juan Timor Rahmadhika

24060117140074

DEPARTEMEN ILMU KOMPUTER/INFORMATIKA

FAKULTAS SAINS DAN MATEMATIKA

UNIVERSITAS DIPONEGORO

2020

KATA PENGANTAR

Pertama – tama penulis ucapkan segala puji dan syukur atas kehadirat Allah SWT yang telah melimpahkan rahmat dan hidayah-Nya, sehingga penulis bisa menyelesaikan skripsi dengan dengan judul "Pengukuran Kualitas *Maintainability* Aplikasi Menggunakan Metriks *Chidamber* dan *Kemerer* (Studi Kasus Sistem Informasi Skripsi Informatika Universitas Diponegoro) " sebagai syarat untuk memperolah gelar sarjana strata satu (S1) pada Departemen Ilmu Komputer/Informatika Fakultas Sains dan Matematika Universitas Diponegoro, Semarang.

Penulis menyadari bahwa penulisan skripsi ini tidak dapat terselesaikan tanpa dukungan dari berbagai pihak baik moril, maupun materiil. Oleh karena itu, pada kesempatan ini penulis menyampaikan ucapan terima kasih kepada:

1. Prof. Dr. Widowati, S.Si, M.Si selaku Dekan Fakultas Sains dan Matematika Universitas Diponegoro.
2. Dr. Retno Kusumaningrum, S.Si, M.Kom selaku Ketua Departemen Ilmu Komputer/ Informatika.
3. Edy Suharto, S.T, M.Kom selaku dosen pembimbing yang telah bersedia meluangkan waktu untuk memberikan arahan dan solusi pada setiap permasalahan hingga terselesaikannya laporan skripsi ini.
4. Kedua orang tua saya beserta adik yang selalu memberikan doa dan dukungan selama proses penyusunan skripsi.
5. Seluruh teman-teman mahasiswa Informatika Universitas Diponegoro yang senantiasa memberi semangat dan dukungan selama kuliah.

Akhir kata, penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan. Oleh karena itu, penulis mengharapkan segala bentuk saran dan kritik yang membangun dari berbagai pihak. Semoga skripsi ini dapat bermanfaat bagi kita semua.

Semarang, 07 Oktober 2021

Penulis

ABSTRAK

Kualitas perangkat lunak merupakan salah satu aspek penting bagi perangkat lunak. Suatu perangkat lunak yang memiliki banyak kecacatan misalnya respon yang lama maupun banyak kesalahan tampilan kemungkinan besar akan ditinggalkan oleh para penggunanya. Dengan adanya pengukuran kualitas dari suatu perangkat lunak akan diketahui kekurangan dari perangkat lunak tersebut sehingga dapat diperbaiki dan ditingkatkan kualitasnya guna memuaskan harapan dari penggunanya. Salah satu cara mengukur kualitas perangkat lunak adalah menggunakan metriks *chidamber kemerer* yang memiliki enam metriks yang masing-masing memiliki pemetaan tersendiri. Penggunaan metriks ini dikarenakan aplikasi yang dibangun berkonsep *object-oriented* sehinga metriks cocok diterapkan pada aplikasi. Pengukuran kualitas dilakukan pada Sistem Informasi Skripsi Informatika Universitas Diponegoro. Aplikasi yang diukur kualitasnya berdasarkan karakteristik dari standar ISO/IEC 25010 bagian *maintainability*. Setelah pengukuran nilai kualitas dilakukan perbaikan pada bagian yang nilainya masih kurang dengan menggunakan metode *refactoring* pada *code smells* dari kode aplikasi. Output yang dihasilkan yaitu kualitas *maintainability* aplikasi yang lebih baik dari versi sebelumnya karena sudah dilakukan *refactoring* sesuai dengan hasil metriks *chidamber* dan *kemerer*.

Kata Kunci: Kualitas Perangkat Lunak, Metriks *Chidamber* dan *Kemerer*, ISO/IEC 25010, *Refactoring*

ABSTRACT

Software quality is one of the important aspects for software. A software that has many defects such as long responses or many display errors is likely to be abandoned by its users. With the measurement of the quality of a software, the shortcomings of the software will be known so that it can be repaired and improved in quality to satisfy the expectations of its users. One way to measure the quality of software is to use the chidamber and kemerer metrics, which have six metrics, each of which has its own mapping. The use of this metric is because the application is build with an object-oriented concept so that the metric is suitable to be applied to the application Quality measurement was carried out at the Sistem Informasi Skripsi Informatika Universitas Diponegoro. Applications was measured for quality based on the characteristics of the ISO/IEC 25010 standard for maintainability. After measuring the quality value, improvements were made to the part whose value is still lacking by using the refactoring method on the code smells of the application code. The output is the application with better maintainability than the previous version because refactoring has been carried out according to the result of the chidamber and kemerer metrics.
Keywords: Software Quality, *Chidamber Kemerer Metrics*, ISO/IEC 25010, *Refactoring*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Kualitas perangkat lunak merupakan standar yang diharapkan dari setiap perangkat lunak yang dibangun. Suatu perangkat lunak harus memiliki standar kualitas yang baik agar memenuhi harapan atau ekspektasi dari penggunanya. Perangkat lunak yang memiliki tingkat kecacatan di luar ambang kualitas atau kualitasnya di bawah standar dari berbagai sisi baik kecepatan akses hingga tampilan, maka dapat dikatakan perangkat lunak tersebut tidak memiliki kelayakan standar yang kemungkinan besar akan ditinggalkan oleh pengguna. Oleh karena itu, kualitas perangkat lunak tidak dapat diabaikan dalam pembuatan perangkat lunak terlebih lagi pada aktivitas yang mengandalkan perangkat lunak guna melakukan pekerjaannya.

Perkembangan teknologi informasi telah membawa banyak perubahan contohnya penggunaan sistem komputerisasi untuk mempermudah pekerjaan manusia. Adanya akses internet yang tidak terbatas membuat perkembangan di kegiatan bisnis sehingga banyak dijumpai kegiatan yang dibuat digitalisasi seperti e-Commerce, e-Business, e-Government, e-Banking, maupun e-Learning. Salah satu layanan berbasis internet yang sedang populer adalah *website* (Andry et al., 2019). *Website* adalah suatu metode untuk menampilkan data berupa teks, gambar, video, angka, yang saling dapat terhubung satu dokumen dengan yang lain yang dapat diakses melalui *browser*. Setiap institusi atau perusahaan besar memiliki suatu *website* sebagai jalur informasi (Firdaus et al., 2019).

Kualitas *website* harus diperhatikan agar dapat memenuhi kepuasan dan kenyamanan pengguna. Suatu *website* yang baik kualitasnya harus memiliki kelayakan media yang bermanfaat. (Manik et al., 2017). Perlu adanya pengukuran kualitas dari perangkat lunak sehingga dari hasil analisis dapat diketahui kekurangan dari aplikasi sehingga dapat dilakukan suatu peningkatan demi menghasilkan kualitas aplikasi yang lebih baik. Pengukuran kualitas perangkat lunak dapat dilakukan menggunakan beberapa pendekatan atau metode. Salah satu pendekatan untuk melihat kualitas perangkat lunak adalah pendekatan dengan standar ISO/IEC 25010.

Standar kualitas perangkat lunak ISO/IEC 25010 adalah model terbaru dari standar sebelumnya yaitu ISO/IEC 9126. Terdapat beberapa perubahan dari versi ISO/IEC sebelumnya. Beberapa penambahan tersebut yaitu pada sisi *security* dan *compatibility*.

Metode pengukuran perangkat lunak ini sudah banyak digunakan oleh beberapa peneliti untuk mengevaluasi kualitas perangkat lunak. Salah satunya penelitian yang dilakukan oleh (Dewi, 2020) yaitu melakukan pengukuran dan evaluasi aspek *maintainability* pada aplikasi *mobile* "myITS" menggunakan standar ISO/IEC 25010.

Perangkat lunak yang dijadikan studi kasus pada tugas akhir ini adalah "Sistem Informasi Skripsi Informatika Universitas Diponegoro". *Website* ini adalah suatu sistem milik program studi Informatika Universitas Diponegoro yang terintegrasi dengan *database* yang digunakan untuk melakukan pendataan yang berhubungan dengan tugas akhir atau skripsi yang ada di program studi Informatika. Beberapa fitur yang ada pada website ini antara lain pendaftaran tugas akhir atau skripsi, catatan bimbingan skripsi mahasiswa dengan dosennya, dan pendaftaran sidang akhir. Pengguna dari sistem ini yaitu dosen dan mahasiswa, juga terdapat admin sistem yang bertugas untuk memverifikasi suatu data.

Pada perhitungan nilai kualitas menggunakan sebuah metriks yang biasa digunakan untuk mengukur nilai kualitas *software* yang berbasis *Object Oriented Programming* (OOP) yaitu suatu metriks yang bernama metriks*Chidamber* dan *Kemerer*. Untuk upaya peningkatan nilai kualitas aplikasi akan dilakukan rekayasa dengan melakukan *refactoring code* kemudian akan diukur kembali nilai kualitasnya untuk memastikan bahwa ada peningkatan. Perlu adanya evaluasi kualitas perangkat lunak pada "Sistem Informasi Skripsi Informatika Universitas Diponegoro" karena aplikasi ini masih baru sehingga lebih mudah untuk dilihat kekurangannya, dan lebih mudah ditingkatkan kualitasnya. Alasan untuk mengevaluasi kualitas perangkat lunak ini adalah juga karena belum pernah ada evaluasi kualitas serupa pada perangkat lunak ini.

Berdasarkan permasalahan di atas, perlu dilakukan suatu analisis dan peningkatan kualitas perangkat lunak *website* "Sistem Informasi Skripsi Informatika Universitas Diponegoro" pada aspek *maintainability*. Sistem informasi skripsi ini dikembangkan dengan menggunakan beberapa bahasa pemrograman *web* dan juga *framework* atau kerangka kerja. Bahasa pemrograman utama dan *framework* yang digunakan yaitu *JavaScript* dan *PHP* dengan menggunakan *framework Vue.Js* sebagai *front-end* dan *CodeIgniter* sebagai *back-end*. Sedangkan untuk sistem basis datanya menggunakan *database MySQL*.

Pengukuran dilakukan menggunakan metode standar ISO/IEC 25010 dengan metrik *chidamber* dan *kemerer* karena aplikasi menggunakan konsep *Object Oriented Programming* (OOP). Kemudian dilakukan langkah peningkatan kualitas dengan *refactoring code*. *Refactoring code* dilakukan untuk meningkatkan kualitas perangkat lunak dengan hanya mengubah struktur internal kode menjadi lebih rapi tanpa mengubah atau menambah fungsionalitas dari perangkat lunak.

1.2 Rumusan Masalah

Berdasarkan permasalahan di atas, maka rumusan masalah yang dapat diambil adalah bagaimana mengukur kualitas perangkat lunak berdasarkan metode ISO/IEC 25010 pada aspek *maintainability* dengan menggunakan metrik *chidamber* dan *kemerer*, dan kemudian meningkatkan hasil kualitasnya dengan melakukan *refactoring*.

1.3 Tujuan dan Manfaat

Tujuan dari penelitian skripsi ini adalah sebagai berikut:

Meningkatkan kualitas perangkat lunak "Sistem Informasi Skripsi Informatika Universitas Diponegoro" menggunakan standar ISO/IEC 25010 dan metrik *chidamber kemerer.*

Manfaat yang diharapkan dari penelitian ini adalah sebagai berikut :

1. Bagi Institusi

   Penelitian ini dapat memberikan aplikasi dengan kualitas yang lebih baik di sisi *maintainability* ketimbang sebelumnya dengan menggunakan standar ISO/IEC 25010 dan metrik *chidamber dan kemerer*.
2. Bagi Pengembang

   Penelitian ini dapat memberikan aspek penting dari sisi *maintainability* aplikasi sebagai referensi pengembang untuk pengembangan aplikasi tahap selanjutnya.
3. Bagi Peneliti Selanjutnya

   Penelitian ini dapat menjadi tambahan referensi untuk penelitian selanjutnya yang serupa.

1.4 Ruang Lingkup

Ruang lingkup tugas akhir ini adalah sebagai berikut:

1. Perangkat lunak yang akan dianalisis adalah *website* "Sistem Informasi Skripsi Informatika Universitas Diponegoro".
2. Aplikasi dibangun dengan menggunakan beberapa bahasa pemrograman utama yaitu *JavaScript* dengan memanfaatkan *framework Vue.Js* sebagai *front-end* aplikasi, bahasa pemrograman PHP dengan *framework CodeIgniter* sebagai *back-end*, dan sistem basis data *MySQL*.
3. Analisis kualitas perangkat lunak menggunakan metode standar ISO/IEC 25010 dan metrik *chidamber kemerer* (CK).
4. Variabel yang dianalisis dan ditingkatkan dari perangkat lunak sesuai standar ISO/IEC 25010 adalah *maintainability* dengan sub-karakteristik *analyzability*, *modifiability, reuseability*, dan *testability*.
5. Peningkatan nilai kualitas dilakukan dengan melakukan proses *refactoring code* setelah pengukuran pertama.
6. *Tools* yang digunakan dalam membantu pengukuran adalah PHPMetrics.

1.5 Sistematika Penulisan

Sistematika penulisan memberikan gambaran laporan dari tugas akhir ini secara urut dan jelas. Berikut adalah sistematika penulisan tugas akhir ini:

BAB I PENDAHULUAN

Bab ini membahas latar belakang, rumusan masalah, manfaat dan tujuan, ruang lingkup dan sistematika penulisan laporan tugas akhir.

BAB II TINJAUAN PUSTAKA

Bab ini membahas teori-teori yang berhubungan dan mendukung topik yang dibahas pada tugas akhir ini seperti perangkat lunak, kualitas perangkat lunak, ISO/IEC 25010, metrik *chidamber* dan *kemerer*, hingga *refactoring*.

BAB III METODOLOGI PENELITIAN

Bab ini menjelaskan mengenai tahapan penyelesaian masalah dalam

tugas akhir. Tahapan tersebut meliputi pengaplikasian teori sebagai panduan dalam proses penelitian yang dimulai dari tahap studi literatur, observasi, analisis hasil observasi hingga peningkatan kualitas perangkat lunak.

BAB IV HASIL DAN PEMBAHASAN

Bab ini menjelaskan hasil dari tahapan penelitian dimulai dari fase analisis aplikasi, melakukan pengukuran kualitas aplikasi menggunakan standar ISO/IEC 25010 dan metrik *chidamber kemerer* (CK), tahap peningkatan kualitas aplikasi dengan melakukan *refactoring*, hingga proses evaluasi.

BAB V PENUTUP

Bab ini berisi tentang kesimpulan dari bab-bab sebelumnya dan berisi saran sebagai bahan masukan untuk penelitian serupa selanjutnya

BAB II
TINJAUAN PUSTAKA

Bab ini berisi tinjauan pustaka dari berbagai sumber yang digunakan dalam melakukan penelitian.

2.1 Sistem Informasi

Sistem informasi merupakan suatu sistem dam suatu organisasi yang merupakan kombinasi dari orang-orang, fasilitas, teknologi, media, prosedur, dan pengendalian untuk mendapatkan jalur komunikasi penting, memproses tipe transaksi rutin tertentu, memberi sinyal kepada manajemen dan yang lainnya terhadap kejadian-kejadian internal dan eksternal yang penting dan menyediakan suatu dasar informasi untuk pengambilan keputusan (Pamungkas, 2017).

Berdasarkan pendapat tersebut, dapat dikatakan bahwa sistem informasi adalah sekumpulan prosedur organisasi yang dilaksanakan untuk mencapai suatu tujuan yaitu memberikan informasi bagi pengambil keputusan dan untuk mengendalikan organisasi.

2.2 Perangkat Lunak

Perangkat lunak adalah instruksi yang ketika dijalankan memberikan fungsi dan kinerja yang diinginkan, struktur data yang memungkinkan program untuk memanipulasi informasi secara memadai, dan dokumen yang menjelaskan pengoperasian dan penggunaan program (Pressman, 2010).

2.3 Kualitas Perangkat Lunak

Kualitas perangkat lunak bisa didefinisikan sebagai kesesuaian yang diharapkan pada semua perangkat lunak yang dibangun berkaitan dengan fungsi perangkat lunak yang diutamakan dan untuk kerja perangkat lunak (Muhlis, 2017).

Menurut badan internasional ISO ada beberapa sudut pandang untuk menggambarkan kualitas perangkat lunak. Standar ISO menggambarkan tiga pandangan kualitas (Muhlis, 2017), yaitu: Pandangan pengguna (*user*)

1) Pandangan pengembang (*developer*)
2) Pandangan manager (*manager*)

Proses pembuatan perangkat lunak, selain dari sisi fungsionalitas sebuah program, juga harus mempertimbangkan sisi kualitas dan penjaminannya. Kualitas pada perangkat lunak merupakan tingkatan pada sebuah sistem komponen atau proses yang dihasilkan sesuai dengan harapan pengguna serta bebas dari *defect* (Kartiko, 2019) .

2.4 Dimensi Kualitas Perangkat Lunak Website

Menurut (Pressman, 2010), dimensi kualitas dari aplikasi *web* yaitu:

1) Konten : dievaluasi dari tingkat *syntactic* dan *semantic*. Pada tingkat *syntactic* dinilai dari *text-based document*. Sedangkan pada tingkat *semantic* dinilai dari sisi konsistensi, kebenaran, dan ambiguitas.
2) Fungsi : diuji untuk mengungkapkan kesalahan yang menunjukkan kurangnya kesesuaian terhadap kebutuhan pelanggan.
3) Struktur : dinilai untuk memastikan bahwa konten aplikasi *web* benar dan dapat berfungsi, dapat diperluas, dan dapat didukung sebagai konten baru.
4) Kegunaan : diuji untuk memastikan bahwa setiap kategori pengguna didukung oleh tampilan antarmuka, dapat dipelajari, dan menerapkan semua sintaks navigasi dan semantik yang diperlukan.
5) Portabilitas : diuji untuk memastikan bahwa semua navigasi sintaks dan semantik dilakukan untuk menemukan adanya kesalahan navigasi.

2.5 *Object Oriented Programming* (OOP)

Pemograman berorientasi *object* adalah teknik membuat suatu program berdasarkan *object*. Pendekatan berorientasi *object* mulai berkembang karena adanya kesulitan dalam pengembangan sistem pada skala besar untuk menghasilkan sistem yang berkualitas sesuai dengan biaya dan waktu yang ada. Pendekatan ini menggabungkan data dan proses secara bersamaan dalam bentuk *object*, hal tersebut menjadi kelebihan dari pendekatan ini karena kode program menjadi lebih mudah digunakan kembali. Pendekatan ini memperkenalkan istilah *class, object,* atribut dan *method*. Setiap *object* mempunyai atribut dan *method*. Atribut adalah segala sesuatu yang berhubungan dengan karakteristik *object*.

Method merupakan fungsi atau segala sesuatu yang dapat dilakukan oleh *object*. *Class* adalah tempat *object* tersebut berada (Ayubi et al., 2015).

Object mempunyai dua karakteristik yaitu variabel sebagai status dan *method* sebagai perilaku dari sebuah *object*. *Object* menyimpan statusnya pada variabel dan mendefinisikan perilakunya melalui sebuah *method*. *Method* akan mengakses nilai dari field *object* dan sebagai mekanisme utama komunikasi antar *object*, sehingga dunia luar tidak perlu mengetahui bagaimana *object* dapat saling berkomunikasi melalui sebuah *method*. *Method* adalah sebuah operasi pada sebuah *object* dan didefinisikan dalam deklarasi *class*. *Message* adalah permintaan dari *object* lain untuk melakukan sebuah operasi/fungsi. *Object* tersebut menjawab message dengan sebuah *method*. *Cohesion* adalah tingkat *method* dalam *class* yang direlasikan ke *class* lain. Desain berorientasi objek yang efektif memaksimumkan *cohesion* karena meningkatkan *encapsulation*. *Cohesion* yang tinggi mengindikasikan sub-divisi *class* yang baik. *Coupling* adalah *method* dalam sebuah *class* memanggil *method* pada *class* lain. *Inheritance* adalah hirarki *class*, terdapat super*class* dan sub*class* yang mewarisi atribut atau *method* dari super*class*. Package merupakan sekelompok *class* dan *interface* yang saling terkait (Ayubi et al., 2015).

Pada *Object Oriented Programming* (OOP), terdapat 4 prinsip utama yang menjadi dasar penggunaannya. Keempat prinsip tersebut yaitu :

1) *Encapsulation*

   Enkapsulasi atau penkapsulan adalah konsep pengikatan data berbeda yang disatukan menjadi sebuah unit data. Enkapsulasi bisa mempermudah pembacaan kode, karena data yang disiapkan tidak perlu dibaca secara rinci karena sudah menjadi satu kesatuan. Proses enkapsulasi ini mempermudah untuk menggunakan sebuah objek dari suatu kelas karena tidak perlu mengetahui segala hal dari objek secara rinci.

2) *Abstraction*

   Abstraksi yaitu menyembunyikan detail dan hanya mewakili informasi yang diperlukan bagi objek lain. Proses ini adalah penyederhanaan konsep dunia nyata menjadi komponen yang mutlak diperlukan.

3) *Inheritance*

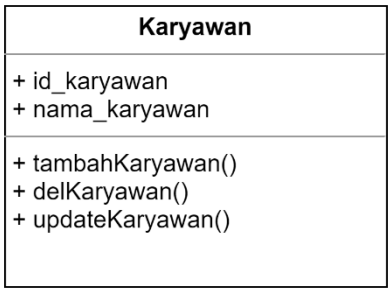
   Inheritance atau pewarisan adalah kemampuan membentuk kelas baru yang memiliki fungsi turunan mirip dengan fungsi sebelumnya. Konsep ini menggunakan hierarki yang berarti semakin jauh turunannya, semakin sedikit kemiripannya.

4) *Polymorphism*

Konsep ini adalah kemampuan suatu data untuk diproses lebih dari satu bentuk. Konsep ini merupakan ciri utama dari OOP di mana suatu objek yang berbeda dapat diakses melalui *interface* yang sama.

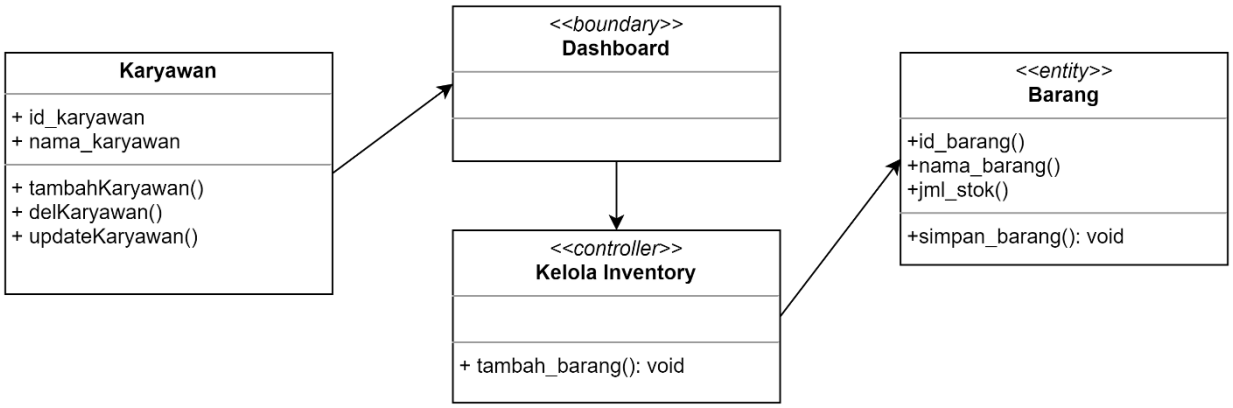
2.6 *Class Diagram*

Class diagram merupakan sebuah diagram yang menggambarkan sekumpulan *class*, antarmuka dan hubungan satu sama lain. *Class* memiliki tiga bagian yaitu nama *class*, atribut, dan *method*. Gambar 2.1 menunjukkan contoh *class*.



Gambar 2.1 Contoh *Class*

Berdasarkan Gambar 2.1, sebuah *class* karyawan memiliki beberapa atribut dan *method*. Atribut adalah ciri yang melekat di dalam *class* sedangkan *method* adalah behavior atau perilaku dari *class* tersebut 2 atribut di dalam *class* yaitu id_karyawan dan nama_lengkap, dan memiliki 3 *method* yaitu tambahKaryawan(), delKaryawan(), dan updateKaryawan(). Seperti yang dijelaskan sebelumnya bahwa *class diagram* merupakan kumpulan dari *class* beserta hubungannya. Berikut contoh *class diagram* ditunjukkan oleh Gambar 2.2.

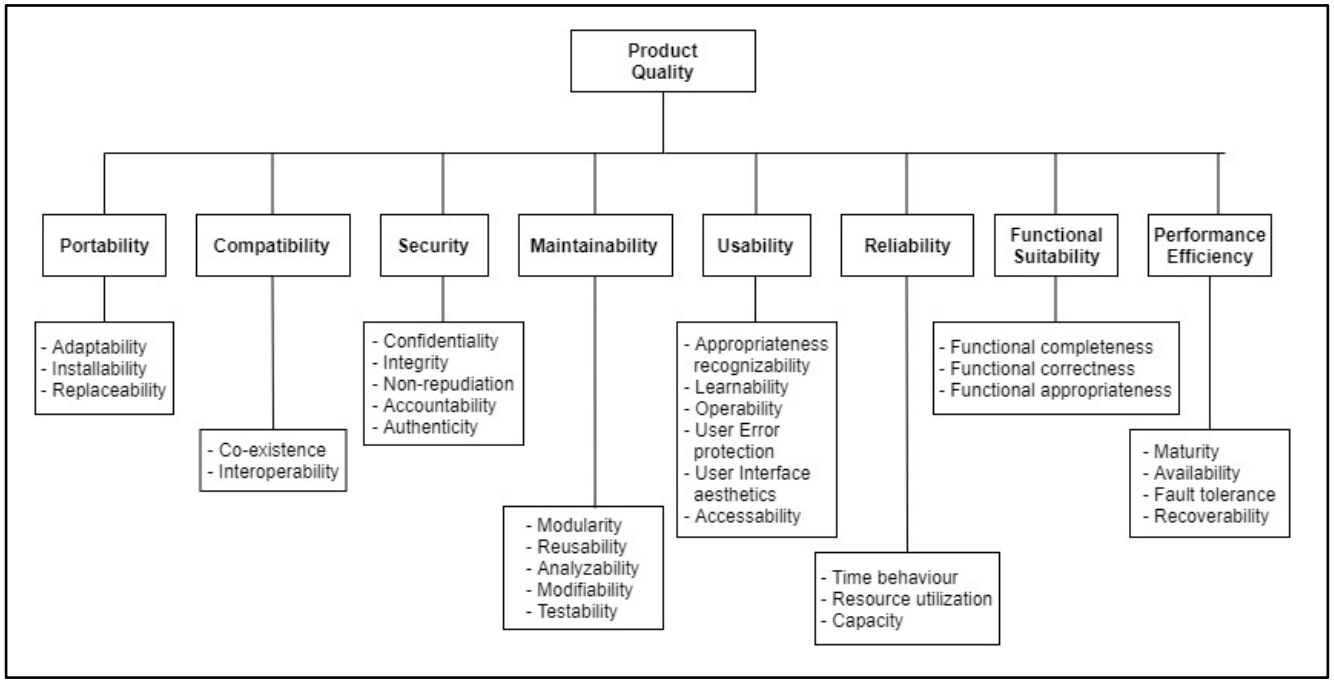


Gambar 2.2 Contoh *Class Diagram*

Berdasarkan Gambar 2.2, *class diagram* tersebut memiliki 4 *class* yaitu *class* karyawan, *class* dashboard, *class* kelola inventory, dan *class* barang. *Class* admin memiliki beberapa atribut seperti: id_admin, nama lengkap, dan tgl_lahir dan memiliki *method* tambahAdmin(), delAdmin, dan updateAdmin(). *Class dashboard* sebagai *class boundary*. *Class* kelola barang sebagai *class controller*. *Class* barang sebagai *class entity* yang memiliki atribut id_barang, nama_barang, dan jml_stock dan memiliki satu *method* yaitu simpan_barang().

2.7 ISO/IEC 25010

ISO (*the International Organization for Standardization*) dan IEC (*the International Electroechnical Commision*) merupakan suatu organisasi dan komisi yang membentuk suatu sistem standardisasi yang berlaku untuk digunakan seluruh dunia. ISO/IEC 25010 merupakan bagian dari standarisasi SQuaRE (*System and Software Quality Requirements and Evaluation*). ISO/IEC 25010 merupakan standarisasi baru yang dikeluarkan menggantikan versi sebelumnya yaitu ISO/IEC 9126, sehingga penelitian yang membahas tentang metode ISO/IEC 25010 masih relatif sedikit jika dibandingkan metode sebelumnya (D. Nurul Huda, 2005). Karakteristik dari ISO/IEC 25010 ditunjukkan pada Gambar 2.3.



Gambar 2.3 Standar Kualitas ISO/IEC 25010.

Pada ISO/IEC 25010 terdapat beberapa penambahan struktur dan bagian dari standar model kualitas ISO/IEC 9126. Selain itu perbedaan lainnya adalah terdapat beberapa sub-karakteristik dari ISO/IEC 9126 yang diubah menjadi karakteristik dari ISO/IEC 25010.

Secara keseluruhan ISO/IEC 25010 memiliki 8 karakteristik untuk mengukur kualitas perangkat lunak secara menyeluruh (Wattiheluw et al., 2019).

Penjelasan karakteristik dari ISO 25010 menurut (Gunawan & Triantoro, 2017) :

1) *Functional suitability* : sejauh mana perangkat lunak mampu menyediakan fungsi yang memenuhi kebutuhan yang dapat digunakan dalam kondisi tertentu. Karakteristik ini dibagi menjadi beberapa sub-karakteristik, yaitu :
   a. *Functional completeness*: sejauh mana fungsi yang disediakan mencakup semua tugas dan tujuan pengguna secara spesifik.
   b. *Functional correctness*: sejauh mana produk atau sistem menyediakan hasil yang benar sesuai kebutuhan.
   c. *Functional appropriateness*: sejauh mana fungsi yang disediakan mampu memfasilitasi penyelesaian tugas dan tujuan tertentu.
2) *Compatibility* : sejauh mana sebuah produk, sistem atau komponen dapat bertukar informasi dengan produk, sistem atau komponen dan/atau menjalankan fungsi lain yang diperlukan secara bersamaan ketika berbagi perangkat keras dan *environment* perangkat lunak yang sama. Karakteristik ini dibagi menjadi 2, yaitu :
   a. *Co-existence*: sejauh mana produk atau sistem dapat menjalankan fungsi yang dibutuhkan secara efisien sementara berbagi sumber daya dengan produk atau sistem yang lain tanpa merugikan produk atau sistem tersebut.
   b. *Interoperability*: sejauh mana dua atau lebih produk, sistem atau komponen dapat bertukar informasi dan menggunakan informasi tersebut.
3) *Usability* : sejauh mana sebuah produk atau sistem dapat digunakan oleh user tertentu untuk mencapai tujuan dengan efektif, efisiensi, dan kepuasan tertentu dalam konteks penggunaan. Karakteristik ini dibagi menjadi beberapa karakteristik, yaitu:
   a. *Appropriateness recognizability*: sejauh mana pengguna dapat mengetahui apakah sistem atau produk sesuai kebutuhan mereka.
   b. *Learnability*: sejauh mana produk atau sistem dapat digunakan oleh pengguna untuk mencapai tujuan tertentu yang belajar menggunakan sistem atau produk dengan efisien, efektif, kebebasan dari risiko dan kepuasan dalam konteks tertentu.
   c. *Operability*: sejauh mana produk atau sistem mudah dioperasikan dan dikontrol.
   d. *User error protection*: sejauh mana produk atau sistem melindungi pengguna

terhadap membuat kesalahan.

e. *User interface aesthetics*: sejauh mana antarmuka pengguna dari produk atau sistem memungkinkan interaksi yang menyenangkan dan memuaskan pengguna.
f. *Accessibility*: sejauh mana produk atau sistem dapat digunakan oleh semua kalangan untuk mencapai tujuan tertentu sesuai konteks penggunaan.

4) *Reliability*: sejauh mana sebuah sistem, produk atau komponen dapat menjalankan fungsi tertentu dalam kondisi tertentu selama jangka waktu yang ditentukan. Karakteristik ini terbagi menjadi beberapa subkarakteristik, yaitu :
   a. *Maturity*: sejauh mana produk atau sistem mampu memenuhi kebutuhan secara handal di bawah keadaan normal.
   b. *Available*: sejauh mana produk atau sistem siap beroperasi dan dapat diakses saat perlu digunakan.
   c. *Fault tolerance*: sejauh mana produk atau sistem tetap berjalan sebagaimana yang dimaksud meskipun terjadi kesalahan pada perangkat keras atau perangkat lunak.
   d. *Recoverability*: sejauh mana produk atau sistem mampu dapat memulihkan data yang terkena dampak secara langsung dan menata ulang kondisi sistem seperti yang diinginkan ketika terjadi gangguan.
5) *Security*: sejauh mana sebuah produk atau sistem melindungi informasi dan data sehingga seseorang atau sistem lain dapat mengakses data sesuai dengan jenis dan level otorisasi yang dimiliki. Karakteristik ini terbagi menjadi beberapa karakteristik, yaitu:
   a. *Confidentiality*: sejauh mana produk atau perangkat lunak memastikan data hanya bisa diakses oleh mereka yang berwenang untuk memiliki akses.
   b. *Integrity*: sejauh mana produk atau perangkat lunak mampu mencegah akses yang tidak sah untuk memodifikasi data.
   c. *Non-repudiation*: sejauh mana peristiwa atau tindakan dapat dibuktikan telah terjadi, sehingga tidak ada penolakan terhadap peristiwa atau tindakan tersebut.
   d. *Accountability*: sejauh mana tindakan dari suatu entitas dapat ditelusuri secara unik untuk entitas.
   e. *Authenticity*: sejauh mana identitas subjek atau sumber daya dapat terbukti menjadi salah satu yang diklaim.

6) *Portability*: sejauh mana keefektifan dan efisiensi sebuah sistem, produk atau komponen dapat dipindahkan dari satu perangkat keras, perangkat lunak atau digunakan pada lingkungan yang berbeda. Karakteristik ini dibagi menjadi beberapa karakteristik, yaitu:
    a. *Adaptability*: sejauh mana produk atau sistem dapat secara efektif dan efisien disesuaikan pada perangkat lunak, perangkat keras dan lingkungan yang berbeda.
    b. *Installability*: sejauh mana produk atau sistem dapat berhasil dipasang atau dihapus dalam lingkungan tertentu.
    c. *Replaceability*: sejauh mana produk atau sistem dapat menggantikan produk atau sistem lain yang ditentukan untuk tujuan yang sama pada lingkungan yang sama.
7) *Performance Efficiency*: kinerja relatif terhadap sumber daya yang digunakan dalam kondisi tertentu. Karakteristik ini terbagi menjadi beberapa subkarakteristik yaitu :
    a. *Time behaviour*: sejauh mana respon dan pengolahan waktu produk atau sistem dapat memenuhi persyaratan ketika menjalankan fungsi.
    b. *Resource utilization*: sejauh mana jumlah dan jenis sumber daya yang digunakan oleh produk atau sistem dapat memenuhi persyaratan ketika menjalankan fungsi.
    c. *Capacity*: sejauh mana batas maksimum parameter produk atau sistem dapat memenuhi persyaratan.
8) *Maintainability*: sejauh mana keefektifan dan efisiensi dari sebuah produk atau sistem dapat dirawat. Karakteristik ini terbagi menjadi beberapa subkarakteristik, yaitu:
    a. *Modularity*: sejauh mana sistem terdiri dari komponen terpisah sehingga perubahan atau modifikasi pada salah satu komponen tersebut memiliki dampak yang kecil terhadap komponen yang lain.
    b. *Reuseability*: sejauh mana aset dapat digunakan lebih oleh satu sistem atau digunakan untuk membangun aset lain.
    c. *Analyzability*: tingkat efektivitas dan efisiensi untuk mengkaji dampak perubahan pada satu atau lebih bagian-bagian produk atau sistem, untuk mendiagnosis kekurangan atau penyebab kegagalan produk, untuk mengidentifikasi bagian yang akan diubah.
    d. *Modifiability*: sejauh mana produk atau sistem dapat dimodifikasi secara efektif dan efisien tanpa menurunkan kualitas produk yang ada.

e. *Testability*: tingkat efektivitas dan efisiensi untuk membentuk kriteria uji dari produk, sistem atau komponen dan uji dapat dilakukan untuk menentukan apakah kriteria tersebut telah terpenuhi.

2.8 Metriks *Chidamber* dan *Kemerer*

Metrik *Chidamber* dan *Kemerer* adalah salah satu metrik yang digunakan untuk mengukur kualitas disain sebuah perangkat lunak berdasarkan enam metrik dengan melihat pada perspektif desain berorientasi *object* (Ayubi et al., 2015).

Metrik merupakan suatu prosedur yang memasangkan karakteristik tertentu pada entitas yang diamati menjadi sebuah nilai numerik. Nilai numerik pada metrik akan memberikan pengetahuan pengamat mengenai nilai yang terlalu tinggi atau terlalu rendah, terlalu banyak atau terlalu sedikit. Manfaat metrik sangat tergantung pada apa yang akan dicapai dari hasil pengukuran yang telah dilakukan.

Metrik *Chidamber* dan *Kemerer* mempunyai enam metrik yaitu *Weighted Method per Class* (WMC), *Depth of Inheritance Tree* (DIT), *Number of Children* (NOC), *Response For a Class* (RFC), *Coupling Between Object Classes* (CBO), dan *Lack of Cohesion Method* (LCOM). Pemetaan Metrik *Chidamber* dan *Kemerer* dengan kode program digambarkan pada Tabel 2.1.

Tabel 2.1 Pemetaan Metriks *Chidamber* dan *Kemerer* dengan Kode Program

Metrik	***Object Oriented Construct***
Weighted Method per Class (WMC)	*Class/Method*
Depth of Inheritance Tree (DIT)	*Inheritance*
Number Of Children (NOC)	*Inheritance*
Response For a Class (RFC)	*Class/Message*
Coupling Between Object Classes (CBO)	*Coupling*
Lack of Cohesion Method (LCOM)	*Class/Cohesion*

Menurut (Ayubi et al., 2015), Metrik *Chidamber* dan *Kemerer* dapat dideskripsikan pada Tabel 2.2.

Tabel 2.2 Batasan Metrik Chidamber dan Kemerer

Metrik *Chidamber* dan *Kemerer*	**Hasil (semakin tinggi)**
Weighted Method per Class (WMC)	Membatasi potensi *reuse*
	Semakin kompleks
	Membutuhkan waktu dan usaha yang tinggi pada tahapan *developing and maintenance*
	Kemungkinan banyaknya *bug*
Depth of Inheritance Tree (DIT)	Kemungkinan *reuse* semakin tinggi
	Semakin kompleks
Number Of Children (NOC)	Kemungkinan *reuse* semakin tinggi
	Membutuhkan usaha *testing* yang lebih tinggi
	Semakin besar kemungkinan ketidakcocokan s*ubclass*
Coupling Between Object Classes (CBO)	Kemungkinan banyaknya *bug* dalam perangkat lunak tersebut
	Membatasi potensi *reuse*
Response For a Class (RFC)	*Testing* dan *debugging* akan semakin kompleks
	Kemungkinan banyaknya *bug* dalam perangkat lunak
Lack of Cohesion Method (LCOM)	Membatasi potensi *reuse*
	Mengindikasikan *class* tersebut sederhana (tidak

2.8.1 Weighted Method per Class (WMC)

Metrik *Weighted Method per Class* adalah pengukuran jumlah *method* dalam setiap *class*nya. Kompleksitas suatu *class* dapat dinilai langsung dengan WMC. *Method* merupakan properti dari *object*, Kompleksitas sebuah *object* ditentukan dengan menghitung propertinya.

Metrik WMC memprekdisikan waktu dan usaha yang diperlukan untuk membangun dan *maintenance* suatu *class*. Nilai WMC yang tinggi dapat mengindikasikan kesalahan yang lebih banyak. *Class* dengan WMC rendah seringkali mengindikasikan *polimorfisme* yang lebih besar. *Class* dengan *method* yang banyak akan cenderung menjadi aplikasi atau perangkat lunak yang spesifik sehingga akan membatasi kemungkinan penggunaan kembali. Semakin besar nilai WMC maka akan membatasi potensi penggunaan kembali, oleh karena itu WMC harus dijaga serendah mungkin.

Contoh perhitungan metriks WMC dapat dilihat pada gambar 2.4. Pada gambar 2.4 *class* Kontak memiliki nilai WMC berjumlah tiga karena memiliki 7 method dan tanpa ada *decision points* pada tiap *method*,

```
<?php
class Kontak{
   var $nama;

   function setNama($nama){
        $this->nama = $nama;
   }
   function getNama(){
        return $this->nama;
   }
   function printNama(){
        echo $this->nama;
   }
?>
```

Gambar 2.4 Source Code Contoh Metriks WMC.

Apabila terdapat *class* C1 dengan *method* c1, …., cn maka WMC dirumuskan pada persamaan 2.1 (Cheikhi et al., 2014).

$$\text{WMC} = \sum_{i=1}^{n} Ci \tag{2.1}$$

Keterangan :

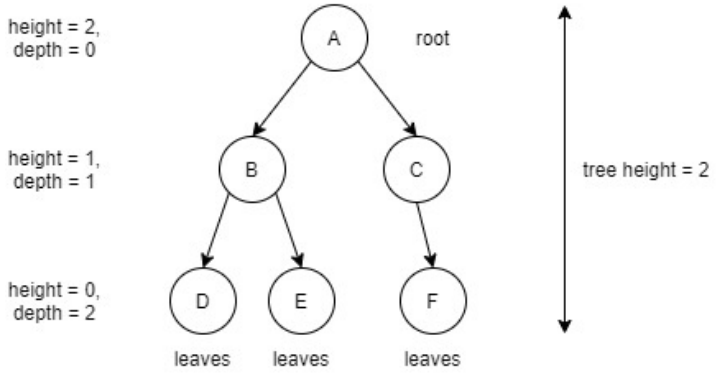
c1, c2, … , c*n* : *method* dalam *class* C.

i : *method* ke 1 sampai *n*.

2.8.2 Depth of Inheritance Tree (DIT)

Metrik *Depth of Inheritance Tree* adalah panjang maksimum dari *root* ke *leaves* dalam sebuah pohon hirarki pada sebuah *class*. Kedalaman suatu *class* disebut hirarki, DIT akan mengukur kedalaman sebuah *class*, dimulai dari *class leaves* dan menurun pada *class root*. Hirarki yang dalam akan menyebabkan tingginya kompleksitas dari disain karena semakin banyak *method* dan *class* yang terlibat, sehingga akan sulit untuk membangun dan *maintenancenya*.

Hirarki yang dalam meningkatkan kemungkinan penggunaan. Kembali.Nilai DIT yang direkomendasikan adalah maksimal 5. Ilustrasi penggambaran *inheritance* dan DIT digambarkan pada gambar 2.4.



Gambar 2.5 Ilustrasi DIT

Pada gambar 2.4, terdapat gambaran *tree* dengan node A sebagai *root* atau akar, dan node D, E, F sebagai *leaves* atau node terbawah. Perhitungan DIT dilakukan dengan mengukur kedalaman dari sebuah *class* yakni *class* yang diukur diposisikan sebagai *leaves* menuju *root.*

2.8.3 Number OF Children (NOC)

Metrik *Number Of Children* menghitung jumlah *subclass* dari *superclass* dalam sebuah hirarki *class*. Metrik NOC dapat menjadi indikator besarnya pengaruh sebuah *class* terhadap disain sistem secara keseluruhan. Semakin besar nilai NOC,semakin besar kemungkinan ketidakcocokan *subclass* dengan abstraksi yang ada pada *superclass*. Hal ini akan berpengaruh pada kesalahan penggunaan *subclass*. Sebuah *class* dengan nilai NOC dan WMC yang tinggi menunjukkan kompleksitas pada *superclass*. Hal ini menunjukkan perlunya desain ulang pada sistem tersebut.

Metrik NOC menghitung jumlah anak (*subclass*) dari keturunan langsung dari *superclass* pada sebuah *class*. Nilai NOC akan menunjukkan tingkat *reusability* dan usaha *testing* yang diperlukan. Semakin banyak jumlah *subclass* maka akan semakin besar tingkat penggunaan kembali karena *inheritance* merupakan bentuk dari *reuse*, semakin banyak *testing* yang harus dilakukan karena apabila terjadi perubahan di *superclass* dapat mempengaruhi *subclass* dari *class* tersebut.

Jika pada gambar 2.4 diilustrasikan sebagai DIT, maka pada pengukuran NOC akan dilakukan sebaliknya. Tiap *class* yang diukur diposisikan sebagai *root*, jadi untuk mengukur NOC adalah *height* dari *root* menuju *leaves*.

2.8.4 Coupling Between Object Classes (CBO)

Metrik *Coupling Between Object Classes* digunakan untuk menghitung *class* yang berhubungan/tergantung dengan *class* lainnya. Semakin banyak *class* yang tidak tergantung satu sama lain (semakin rendah nilai CBO) akan meningkatkan kemungkinan modularity dan penggunaan kembali (*reuse*). Hal ini berhubungan dengan tingkat independency antar modul. Nilai CBO yang tinggi akan mengindikasikan *coupling* yang berlebihan sehingga merugikan desain modular dan membatasi *reuse*, menyulitkan *testing* dan modifikasi. Semakin independen suatu *class* maka akan semakin mudah untuk menggunakan kembali pada aplikasi lain. *Coupling* yang berlebihan juga akan mengidentifikasikan kelemahan dari *class* dalam melakukan enkapsulasi.

2.8.5 Response For a Class (RFC)

Metrik *Response For a Class* menghitung jumlah semua *method* yang dipanggil sebagai respon terhadap *object* luar dari sebuah *class*. RFC mencangkup semua *method* yang diakses dalam hirarki *class* tersebut. RFC digunakan untuk mengukur banyaknya komunikasi antar *object* dalam suatu *class*. Sehingga semakin tinggi nilai RFC, maka akan semakin banyak *method* yang digunakan untuk merespon *object* dari luar dan semakin kompleks sehingga akan meningkatkan waktu untuk *testing*. Nilai RFC yang tinggi akan meningkatkan kemungkinan banyak kesalahan, karena *class* dengan RFC tinggi akan lebih kompleks dan sulit dimengerti.

Persamaan RFC ditunjukkan oleh persamaan 2.2.

$$\text{WMC} = \sum_{i=1}^{n} Mci$$

Keterangan :

M*ci* : *method* yang dipanggil untuk merespons pesan yang dipanggil *method* M*i.*

i : *method* ke 1 sampai *n.*

Berikut merupakan ilustrasi perhitungan dari RFC (Vinet & Zhedanov, 2011) :

- RFC = |RS| dimana RS adalah himpunan respons *class.*
- |RS| = {M*i*} ∪ *i* {M*ci*} dimana {M*ci*} adalah himpunan *method* yang dipanggil oleh *method i* dan {M*i*} adalah himpunan semua *method* dalam *class*.
- A::f1() memanggil B::f2()
- A::f2() memanggil C::f1()
- A::f3() memanggil A::f4()
- A::f4() tidak memanggil method lain
- RS = {A::f1, A::f2, A::f3, A::f4} U {B::f2} U {C::f1} U {A::f4}
  = {A::f1, A::f2, A::f3, A::f4, B::f2, C::f1}, RFC = 6

2.8.6 Lack of Cohesion Method (LCOM)

Metrik *Lack of Cohesion Method* mengukur kemiripan *method* dalam sebuah *class* dari *instance* variabel atau atribut. LCOM mengukur *method* yang tidak terhubung dengan *class* lain. Tingginya *cohesion* mengidentifikasikan potensi yang baik pada *class* tersebut, mengindikasikan *class* tersebut sederhana dan memiliki sifat *reusability* yang tinggi. Sedangkan semakin rendah *cohesion* maka semakin kompleks *class* tersebut.

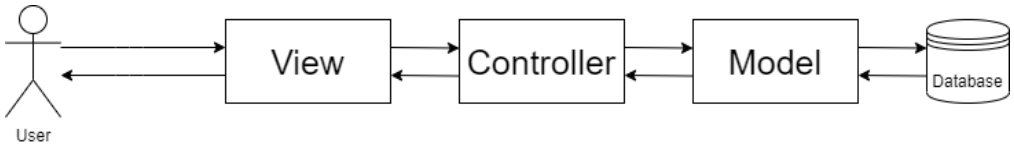
Penelitian yang dilakukan oleh (Basili et al., 1996) menghasilkan kategori dari metriks *chidamber* dan *kemerer* yang akan menjadi tolak ukur dalam perhitungan nilai metriks yang digambarkan pada tabel 2.3.

Tabel 2.3 Nilai Standar Metrik *Chidamber* dan *Kemerer*

Metrik *Chidamber* dan *Kemerer*	***Good***	***Medium***	***Bad***
WMC	$1 \leq x < 50$	$50 \leq x \leq 150$	$150 < x$
DIT	$x < 5$	$5 \leq x < 8$	$8 \leq x$
NOC	$x < 4$	$4 \leq x < 8$	$8 \leq x$
CBO	$x < 14$	$14 \leq x \leq 150$	$150 < x$
RFC	$x < 100$	$100 \leq x \leq 1000$	$1000 < x$

2.9 *CodeIgniter*

CodeIgniter adalah framework pengembangan *back-end* dari Web dengan basis bahasa pemrograman *PHP* yang berkonsep MVC (*Model*, *View*, *Controller*). Merupakan *framework* yang ringan dan cepat juga memiliki dokumentasi yang lengkap sehingga *framework* ini banyak digunakan.



Gambar 2.6 Konsep MVC *Pattern*

Adapun penjelasan dari masing-masing komponen MVC adalah:

1. *Model*

   Komponen *Model* berfungsi untuk mengatur data yang keluar masuk dari *database* sesuai dengan aturan yang telah ditetapkan. Di dalam *framework* CodeIgniter, lokasi komponen *Model* berada di folder /app/Models.

2. *View*

   Komponen *View* adalah komponen yang berinteraksi langsung dengan pengguna. Komponen ini juga yang bertugas untuk memanggil fungsi-fungsi yang ada di komponen *Controller*. Di dalam *framework* CodeIgniter, lokasi komponen *View* berada di folder /app/Views.

3. *Controller*

   Komponen *Controller* adalah komponen yang bertugas untuk menerima input dari komponen *View* lalu memprosesnya. Komponen ini juga berhubungan dengan komponen *Model* secara langsung untuk memberikan data-data yang diterima dari komponen *View* atau mengambil data-data yang diberikan dari komponen *Model*. Tugas lainnya dari komponen ini adalah untuk memproses segala *HTTP Request* yang diterima, *encoding*, *authentication*, *authorization*, dan lainnya. Di dalam *framework* CodeIgniter, lokasi komponen *Controller* berada di folder /app/Controllers.

2.10 *Design Pattern*

Design pattern adalah unsur-unsur rancangan yang seringkali muncul pada berbagai sistem yang berbeda. Setiap pemakaian *patterns* akan menguji *pattern* tersebut di berbagai situasi. Sebuah *design pattern* harus mendokumentasikan permasalahan, pemecahan, serta akibat-akibat penggunaannya. *Class diagram* adalah salah satu bentuk dari interprestasi dari suatu *pattern* dengan memanfaatkan kemampuan UML yang sudah berorentasi pada perancangan yang berbasiskan objek (OOP).

Dalam buku *Design Pattern* yang ditulis oleh Erich Gamma, Richard Helm, Ralph Johnson, dan John Vlissides tahun 1995 terdapat tiga katalog *design pattern* utama yaitu *creational, structural, dan behavioral*. *Creational patterns* berhubungan dengan penciptaan objek. Pola-pola ini berkisar seputar objek mana yang diciptakan, siapa yang menciptakannya, serta berapa banyak objek diciptakan. *Structural patterns* berhubungan dengan struktur statis objek dan kelas. Pola-pola dalam *structural patterns* dapat dilihat pada saat program di-compile melalui struktur *inheritance*, *properties*, serta *agregasi* objek-objek. *Behavioral patterns* terkait perilaku *run-time program*. Pola-pola ini berkaitan dengan algoritma serta interaksi antar objek saat program berjalan. Penekanan *behavioral patterns* lebih pada komposisi objek ketimbang *inheritance*.

Creational patterns adalah *pattern* yang berhubungan dengan menginisialisasi dan mengkonfigurasi kelas serta objek. *Structural patterns* adalah *pattern* yang digunakan untuk memisahkan antarmuka (*interface*) dan implementasi kelas serta objek, *pattern* ini juga berhubungan dengan susunan jumlah kelas atau objek. *Behavioral patterns* adalah *pattern* yang berhubungan dengan interaksi dinamis antara kumpulan semua kelas dan objek, *pattern* ini juga terkait bagaimana *pattern* mendistribusikan tugasnya masing-masing.

A. Creational Patterns (cara *class*/*object* di-inisiasi)

1) *Abstract Factory* (Menciptakan sebuah *instance* dari beberapa keluarga kelas, digunakan untuk membangun objek terkait)
2) *Builder* (Memisahkan konstruksi objek dari perwakilannya, digunakan untuk membangun objek yang kompleks secara bertahap)
3) *Factory Method* (Menciptakan sebuah *instance* dari beberapa kelas yang diturunkan, *method* dalam suatu kelas turunan untuk menciptakan asosiasi)
4) *Prototype* (sebuah *instance* yang sepenuhnya diinisiasikan untuk disalain atau dikloning, digunakan untuk mengkloning instances dari prototipe)
5) *Singleton* (sebuah kelas yang mana hanya sebuah instance yang dapat *exist*, *digunakan untuk instance yang tunggal*).

B. Structural Patterns (struktur/relasi antar *object*/*class*)

1) *Adapter* (Mencocokkan *interface* dari kelas yang berbeda, *translator* yang beradaptasi terhadap suatu *server interface* untuk suatu *client*)
2) *Bridge* (Memisahkan antarmuka objek dari implementasinya, *abstraksi* yang digunakan untuk mengikat menjadi satu dari banyak implementasi yang dibuat)
3) *Composite* (Struktur *tree* dari objek sederhana dan komposit, struktur untuk membangun *agregasi rekursif*)
4) *Decorator* (Menambahkan responsbilities untuk objek dinamis, dekorator memperluas suatu objek secara terbuka)
5) *Facade* (Kelas tunggal yang merepresentasikan seluruh subsistem, menyederhanakan *interface* untuk subsistem)
6) *Flyweight* (*Instance* yang digunakan untuk berbagi efisiensi, banyak objek kecil/spesifik yang di-*share* secara efisien)
7) *Proxy* (Sebuah objek yang mewakili objek lain, satu objek yang mendekati objek lain).

C. Behavioral Patterns (tingkah laku atau fungsi dari *class*/*object*.)

1) *Chain of Responbility* (sebuah cara untuk melewati permintaan antara rangkaian objek, Melakukan *request* yang didelegasikan kepada penyedia layanan yang menjadi tugas dan tanggung jawabnya)
2) *Command* (Enkapsulasasi permintaan *command request* sebagai objek, melakukan *request* objek *first-class*)
3) *Interpreter* (Sebuah cara untuk memasukkan elemen bahasa dalam program, menggabungkan elemen-elemen yang diakses secara skuensial)
4) *Iterator* (Secara berurutan mengakses elemen lokasi, *Language interpreter for a small grammar*, kompiler skala kecil)
5) *Mediator* (Mendefinisikan komunikasi sederhana antara kelas satu dengan lainnya, mengkoordinasi interaksi antar asosiasi yang ada)
6) *Mementto* (Menangkap dan memulihkan state internal objek, gambaran melakukan *captures* dan *restores* keadaan suatu objek secara *private*)
7) *Observer* (Sebuah cara untuk memberitahukan perubahan sejumlah kelas, ketergantungan untuk *update* secara otomatis ketika seorang *programmer* melakukan perubahan pada kodenya)
8) *State* (Mengubah *behavior* objek ketika keadannya berubah, objek yang perilakunya tergantung pada keadaannya)

9) *Strategy* (Melakukan enkapsulasi algoritma dalam kelas, *abstraksi* untuk memilih salah satu dari sekian banyak algoritma)
10) *Template Method* (Memperlambat langkah-langkah yang tepat dari suatu algoritma untuk sub*class*, algoritma dengan beberapa langkah yang telah disediakan oleh suatu kelas turunan)
11) *Visitor* (Mendefinisikan operasi baru untu kelas tanpa perubahan, operasi yang diterapkan pada elemen-elemen dari struktur objek yang heterogen/ sangat beragam).

2.11 *Refactoring*

Pada konsep perangkat lunak, *refactoring* adalah teknik atau proses mengubah sistem dari suatu perangkat lunak yang bertujuan untuk meningkatkan kualitasnya tanpa mengubah perilaku eksternal atau fungsionalitas perangkat lunak tersebut (Langsari et al., 2018).

Refactoring berarti meningkatkan perangkat lunak tanpa mengubah sesuatu yang terlihat, pengembang tidak menambahkan fitur baru selama proses dari *refactoring* dan juga tidak melakukan perbaikan *bug* pada perangkat lunak yang akan terlihat oleh pengguna. Sebaliknya, hanya struktur internal dari perangkat lunak yang diubah.

Tujuan utama dari *refactoring* adalah meningkatkan kualitas struktur internal dari perangkat lunak. Proses pada *refactoring* yaitu membersihkan atau merapikan kode program sehingga meminimalkan peluang terjadinya *bug* pada perangkat lunak yang dibuat. Secara umum *refactoring* meningkatkan desain perangkat lunak.

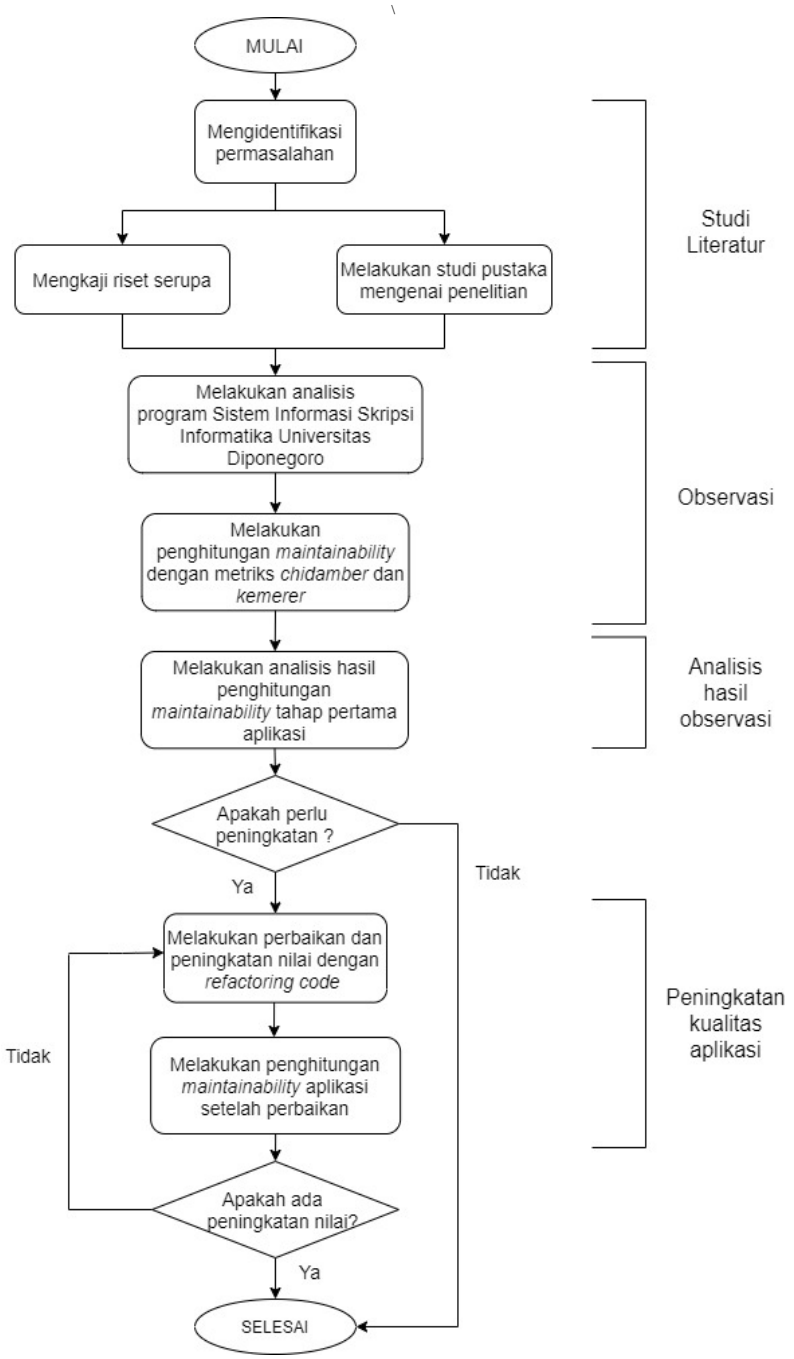
2.12 PHPMetrics

PHPMetrics adalah sebuah *tools* yang menyediakan *software metrics* yang mendukung *project* dan *class* bahasa pemrograman PHP. *Tools* ini dibuat untuk mengatasi perhitungan metrik secara manual karena membutuhkan waktu yang tidak sedikit, juga menyediakan *report* yang mudah dibaca dan diakses mengenai kualitas dan kompleksitas sebuah kode bahkan untuk yang bukan seorang professional.

BAB III
METODOLOGI PENELITIAN

3.1 Alur Metodologi Penelitian

Alur metodologi penelitian menggambarkan langkah-langkah secara urut dan terstruktur dalam memecahkan permasalahan dalam sebuah penelitian. Berikut alur metodologi tugas akhir ditunjukkan pada Gambar 3.1:



Gambar 3.1 Alur Metodologi Penelitian

Berdasarkan Gambar 3.1, secara garis besar metodologi penelitian ini dibagi menjadi beberapa bagian yaitu studi literatur, observasi, analisis hasil observasi, dan juga tahap peningkatan kualitas aplikasi. Berikut penjelasan alur metodologi penelitian dijelaskan pada subbab 3.2 sampai dengan subbab 3.6:

3.2 Studi Literatur

Studi literatur dilakukan dengan memahami dasar teori yang mendukung penyelesaian masalah sehingga tujuan penelitian sesuai yang diharapkan. Langkah ini dilakukan dari mengidentifikasi permasalahan. Permasalahan di sini yaitu perangkat lunak yang baru dibuat belum pernah dilakukan pengukuran kualitasnya, sehingga dengan adanya pengukuran kualitas dapat dilakukan langkah perbaikan yang dapat meningkatkan kualitas perangkat lunak. Selanjutnya melakukan studi pustaka yang berkaitan dengan peningkatan kualitas perangkat lunak khususnya bagian *maintainability*. Selain itu dilakukan juga pengkajian riset serupa yang pernah dilakukan. Sumber studi literatur yang digunakan dapat berupa buku, artikel ilmiah, maupun internet.

3.3 Observasi

Observasi dilakukan dengan tahapan pertama yaitu menganalisis program dari "Sistem Informasi Skripsi Informatika Universitas Diponegoro" dengan melihat struktur *package* dan *class* pada aplikasi juga tiap kode pada tiap *class*. Pada proses ini dilakukan pengubahan hasil analisis menjadi desain sistem dan diagram yang berisi *method-method* dari aplikasi agar mempermudah pembacaan struktur *class* dari aplikasi. Selanjutnya melakukan pengukuran nilai kualitas aplikasi bagian *maintainability* dengan menggunakan metrik *chidamber* dan *kemerer*.

Menurut (Ayubi et al., 2015) keempat kriteria dari *maintainability* tersebut dapat diukur dengan menggunakan parameter bagian dari metrik *chidamber* dan *kemerer* yaitu WMC, DIT, NOC, dan RFC. Sehingga dengan mengukur nilai metrik *chidamber* dan *kemerer* sudah mewakili pengukuran nilai *maintainability* aplikasi dengan sub-karakteristik dari ISO/IEC 25010 bagian *maintainability*.

3.3.1 *Weighted Method per Class* (WMC)

Metrik ini menghitung jumlah dari semua *method* yang ada di dalam suatu *class* dan jumlah *decision* yang ada di *class* tersebut (*if / else / or / for*).

3.3.2 *Depth of Inheritance Tree* (DIT)

Metrik ini menghitung jumlah panjang turunan suatu *class* yang berupa *leaves* dari *inheritance tree* menuju ke *root* atau *superclass* yang bernilai 0.

3.3.3 *Number of Children* (NOC)

Metrik ini menghitung jumlah *subclass* dari suatu *class* atau jumlah *class* yang merupakan inheritance dari *class* tersebut.

3.3.4 *Response For a Class* (ROC)

Metrik ini menghitung jumlah *method* pada *class* ditambah jumlah *method* pada *class* lain dari *code base*, yang dipanggil langsung dari salah satu *method* di *class* tersebut (setiap *method* dihitung sekali, walaupun telah dipanggil beberapa kali).

Dalam melakukan pengukuran nilai metrik, struktur yang sudah ada pada *framework* yang digunakan yaitu *CodeIgniter* contohnya *class* bawaan CI_Controller akan diabaikan atau tidak diukur karena sudah menjadi satu sistem yang baku.

Hubungan antara ISO/IEC 25010 dan metrik *chidamber* dan *kemerer* yaitu sub-karakteristik yang ada pada ISO/IEC 25010 yaitu *maintainability* memiliki beberapa kriteria yaitu *modularity*, *reusability*, *analyzability*, dan *testability* yang dapat dihitung menggunakan metrik *chidamber* dan *kemerer*. Metrik *chidamber* dan *kemerer* memiliki nilai yang berbanding terbalik dengan kualitas, sehingga semakin rendah nilai metrik semakin baik kualitas kode.

Menurut (Ayubi et al., 2015) keempat kriteria dari *maintainability* tersebut dapat diukur dengan menggunakan parameter bagian dari metrik *chidamber* dan *kemerer* yaitu WMC, DIT, NOC, dan RFC. Sehingga dengan mengukur nilai metrik *chidamber* dan *kemerer* sudah mewakili pengukuran nilai *maintainability* aplikasi dengan sub-karakteristik dari ISO/IEC 25010 bagian *maintainability*.

3.4 Analisis Hasil Observasi

Hasil observasi dianalisis dengan tujuan untuk melihat hasil pengamatan observasi. Di bagian ini dapat dilihat kekurangan dari aplikasi sehingga bisa dilakukan langkah yang dapat memperbaiki kekurangan tersebut. Hasil analisis observasi dapat menjadi dasar dalam peningkatan kualitas perangkat lunak pada tahap selanjutnya.

Menurut penelitian yang dilakukan oleh (Basili et al., 1996) menghasilkan kategori Metrik *Chidamber* dan *Kemerer* yang digambarkan pada Tabel 2.3. Dengan mengacu kepada tabel 2.3, nilai metrik yang masuk dalam kategori *good* berarti sudah bagus dan tidak

diperlukan adanya perubahan. *Class* yang masuk dalam tingkat *medium* akan ditingkatkan menjadi *good* dan *class* yang masuk ke dalam kategori *bad* akan dirubah.

3.5 Peningkatan Kualitas Aplikasi

Setelah analisis hasil observasi dilakukan, maka dapat dilakukan langkah yang dapat memperbaiki kekurangan dari perangkat lunak atau meningkatkan nilai kualitasnya. Pada tahapan ini yaitu peningkatan kualitas aplikasi, dilakukan langkah peningkatan nilai kualitas perangkat lunak bagian *maintainability* dengan melakukan *refactoring code* atau merapihkan struktur internal kode program tiap *class* yang memiliki nilai kondisi tidak bagus sesuai pada tabel 2.3.

Selanjutnya dilakukan kembali penghitungan nilai kualitas untuk kedua kali guna menguji apakah sudah ada peningkatan nilai kualitas dari pengujian pertama sebelum dilakukan perbaikan ataupun belum. Jika belum terdapat peningkatan dari nilai kualitas, maka langkah yang dilakukan yaitu mengkaji ulang langkah-langkah yang telah dilakukan dan memperbaiki sekali lagi hingga terjadi peningkatan dari nilai sebelumnya pada pengukuran selanjutnya. Namun, jika hasil nilai kualitas telah meningkat dari hasil pengukuran pertama maka peningkatan nilai kualitas *maintainability* aplikasi dinyatakan berhasil.

BAB IV
HASIL DAN PEMBAHASAN

Bab ini berisi tentang implementasi yang dilakukan pada tugas akhir sesuai dengan alur metodologi penelitian. Tahapan ini berisi pembuatan *class* diagram, perhitungan metrik *chidamber* dan *kemerer,* hingga tahap *refactoring* guna meningkatkan nilai *maintainability*.

4.1 Analisis Program

Program yang dianalisis dibangun dengan menggunakan bantuan *framework* CodeIgniter (CI) yang berkonsep pola desain *Model View Controller* (MVC). *Model* bertugas mengatur dan mengorganisasikan data dari basis data. *Controlleri* bertugas mengatur perintah bagi *model* dan *view*, sedangkan *view* hanya bertugas menyampaikan data kepada pengguna sesuai instruksi yang diberikan *controller*.

Bagian dari aplikasi yang akan diukur nilai *maintainability*-nya adalah selain *view* karena hanya berisi sintaks untuk menampilkan data ke pengguna dan tidak berisi sintaks *logic* bisnis dari aplikasi. Bagian yang akan diukur kualitasnya yaitu *model* dan *controller*.

Isi dari *package model* yaitu sebagai berikut :

1. BerkasModel
2. BimbinganModel
3. DokumenModel
4. JadwalModel
5. PengumumanModel
6. SidangModel
7. TimelineModel
8. UserModel
9. UserTimelineModel

Isi dari *package controller* yaitu sebagai berikut :

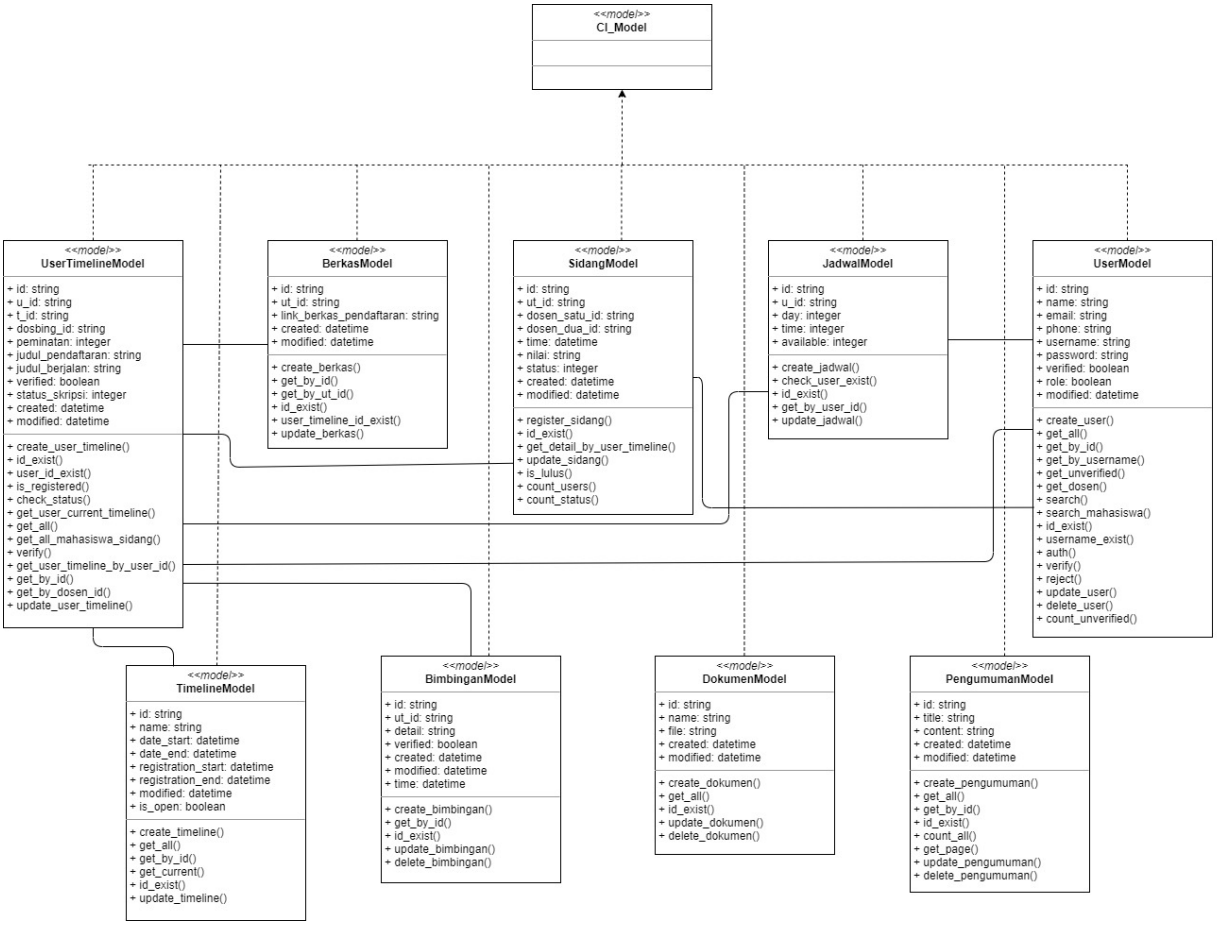
1. Berkas
2. Bimbingan
3. Dokumen
4. Jadwal
5. Pengumuman
6. Sidang
7. Timeline

8. User
9. UserTimeline

Pada tahapan analisis program ini yang pertama yaitu pembuatan *class diagram* aplikasi guna mempermudah proses analisis. *Class diagram* dibuat tiap *package* yaitu *model* dan *controller*. Tahapan berikutnya perhitungan metrik *chidamber* dan *kemerer*. Terdapat empat metrik yang akan digunakan untuk mengukur nilai *maintainability* dari aplikasi yaitu *Weighted Method per Class* (WMC), *Depth of Inheritance Tree* (DIT), *Number Of Children* (NOC), dan *Response For a Class* (RFC).

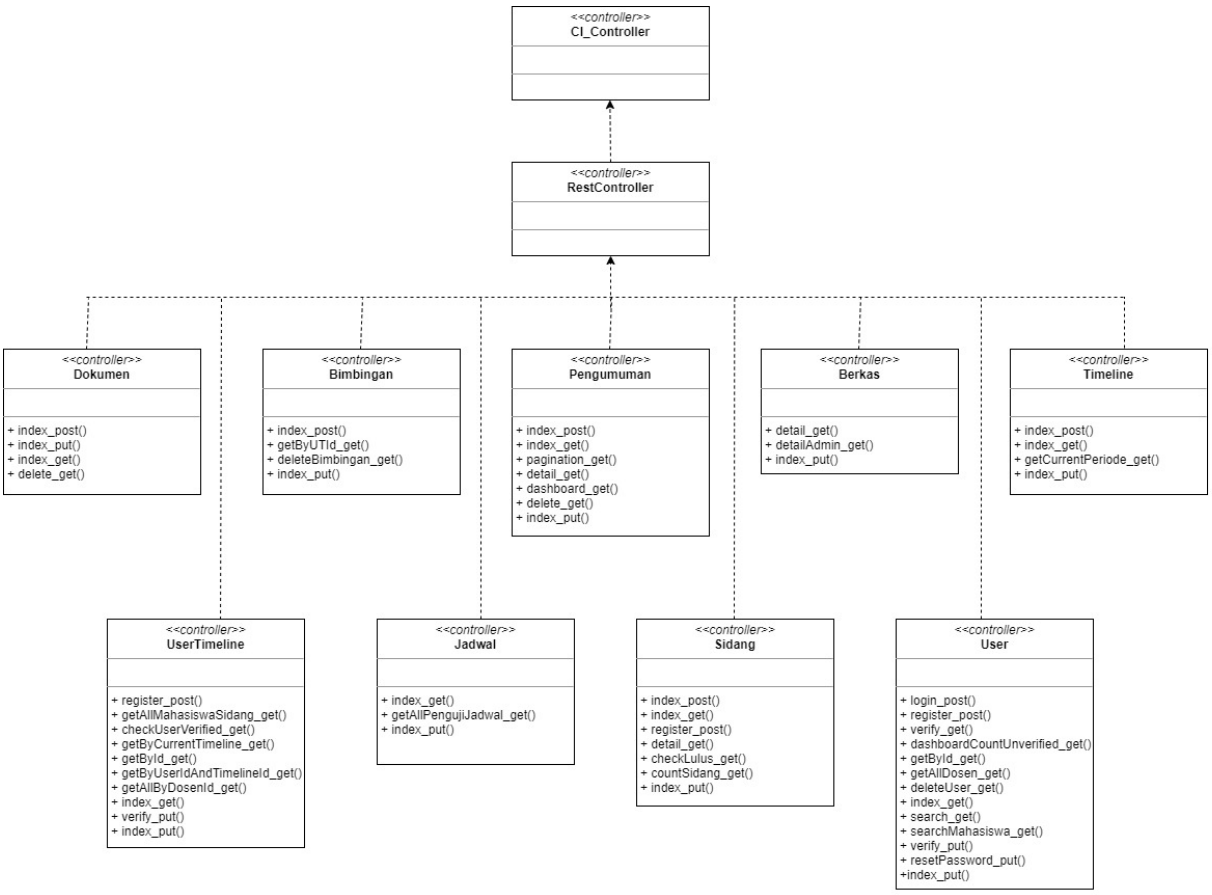
4.1.1 Class Diagram

Pembuatan *class diagram* dari *source code* dilakukan guna mempermudah dalam menganalisis sistem aplikasi sehingga lebih mudah diliat *class* dan berbagai *method* yang dimiliki *class*. Terdapat 2 *class diagram* yaitu *class diagram model* dan *class diagram* dari *controller* yang ditunjukkan oleh gambar 4.1 dan gambar 4.2.



Gambar 4.1 *Class Diagram Model*

Pada gambar 4.1 digambarkan bentuk *class diagram* dari *package model* sistem aplikasi yang dianalisis. Terdapat 9 class yang ada di *package model* sesuai dengan yang ditulis pada subbab 4.1. Sistem aplikasi berbasis *object-oriented* yang menggunakan *design pattern* MVC dengan *framework CodeIgniter* pada tiap kelasnya dibuat *inherit* pada *class* bawaan *framework* yaitu CI_Model. Sehingga pada 9 *class* di *package model* dibuat *inherit* atau mewariskan *class* CI_Model.



Gambar 4.2 *Class Diagram Controller*

Pada gambar 4.2 digambarkan bentuk *class diagram* dari *package controller* sistem aplikasi yang dianalisis. Terdapat 9 class yang ada di *package controller* sesuai dengan yang ditulis pada subbab 4.1. Sistem aplikasi berbasis *object-oriented* yang menggunakan *design pattern* MVC dengan *framework CodeIgniter* dan terdapat *class* tambahan yaitu *class* RestController karena aplikasi yang dibuat menggunakan sistem API untuk mempermudah dalam komunikasi data. Sehingga pada 9 *class controller* yang dibuat *inherit* atau mewariskan *class* RestController terlebih dahulu dan *class* RestController *extends* ke *class* bawaan *framework* yaitu CI_Controller.

4.1.2 Weighted Method per Class (WMC)

Tabel 4.1 merupakan hasil pengukuran nilai metrik WMC pada *package Controller*.

Tabel 4.1 Nilai Metrik WMC *Package Controller*

No	*Class Name*	WMC
1	Berkas	33
2	Bimbingan	23
3	Dokumen	23
4	Jadwal	17
5	Pengumuman	22
6	Sidang	56
7	Timeline	28
8	User	64
9	UserTimeline	59

Nilai WMC pada tabel 4.1 terdapat 3 *class* yaitu Sidang, User, dan UserTimeline yang berada pada range kuning atau *medium* sesuai dengan tabel 2.3 dan sisanya berada pada *range* bagus.

Tabel 4.2 merupakan hasil pengukuran nilai metrik WMC pada *package Model*.

Tabel 4.2 Nilai Metrik WMC *Package Model*

No	*Class Name*	WMC
1	BerkasModel	7
2	BimbinganModel	6
3	DokumenModel	6
4	JadwalModel	6
5	PengumumanModel	9
6	SidangModel	8
7	TimelineModel	7
8	UserModel	22
9	UserTimelineModel	15

Nilai WMC pada tabel 4.2 keseluruhan *class* berada pada *range good* sesuai dengan tabel 2.3 yaitu dengan nilai WMC di bawah 50.

Nilai rata-rata metrik WMC didapatkan dengan membagi jumlah semua nilai WMC dengan jumlah semua *class*. Perhitungan rata-rata nilai WMC terdapat pada tabel 4.3.

Tabel 4.3 Nilai Rata-Rata Metrik WMC

Package	Jumlah Kelas	Nilai WMC
Controller	9	325
Model	9	86
Total	18	411
Rata - rata	$\frac{\sum WMC}{\sum Class}$ = **22,83**	

Hasil perhitungan nilai rata-rata WMC keseluruhan pada tabel 4.3 adalah 22,83 yang menunjukkan bahwa nilai metrik WMC aplikasi terletak pada *range good* sesuai referensi pada tabel 2.3.

4.1.3 Depth of Inheritance Tree (DIT)

Tabel 4.4 merupakan hasil pengukuran nilai metrik DIT pada *package Controller*.

Tabel 4.4 Nilai Metrik DIT *Package Controller*

No	*Class Name*	DIT
1	Berkas	2
2	Bimbingan	2
3	Dokumen	2
4	Jadwal	2
5	Pengumuman	2
6	Sidang	2
7	Timeline	2
8	User	2
9	UserTimeline	2

Nilai DIT pada tabel 4.4 keseluruhan *class* berada pada *range* bagus sesuai dengan tabel 2.3 yaitu nilai DIT bernilai di bawah 5.

Tabel 4.5 merupakan hasil pengukuran nilai metrik DIT pada *package Model*.

Tabel 4.5 Nilai Metrik DIT *Package Model*

No	*Class Name*	DIT
1	BerkasModel	1
2	BimbinganModel	1
3	DokumenModel	1
4	JadwalModel	1
5	PengumumanModel	1
6	SidangModel	1
7	TimelineModel	1
8	UserModel	1
9	UserTimelineModel	1

Nilai WMC pada tabel 4.5 keseluruhan *class* berada pada range *good* sesuai dengan tabel 2.3 yaitu dengan nilai DIT di bawah 5.

Nilai rata-rata metrik DIT didapatkan dengan membagi jumlah semua nilai DITdengan jumlah semua *class*. Perhitungan rata-rata nilai DIT terdapat pada tabel 4.6.

Tabel 4.6 Nilai Rata-Rata Metrik DIT

Package	Jumlah Kelas	Nilai DIT
Controller	9	18
Model	9	9
Total	18	27
Rata - rata	$\frac{\sum DIT}{\sum Class}$ = **1,5**	

Hasil perhitungan nilai rata-rata DIT keseluruhan pada tabel 4.6 adalah 1,5 yang menunjukkan bahwa keseluruhan nilai metrik DIT aplikasi terletak pada *range good* sesuai referensi pada tabel 2.3.

4.1.4 Number of Children (NOC)

Tabel 4.7 merupakan hasil pengukuran nilai metrik NOC pada *package Controller*.

Tabel 4.7 Nilai Metrik NOC *Package Controller*

No	*Class Name*	NOC
1	Berkas	0
2	Bimbingan	0
3	Dokumen	0
4	Jadwal	0
5	Pengumuman	0
6	Sidang	0
7	Timeline	0
8	User	0
9	UserTimeline	0

Nilai NOC pada tabel 4.7 keseluruhan *class* berada pada *range* bagus sesuai dengan tabel 2.3 yaitu nilai NOC bernilai di bawah 4.

Tabel 4.8 merupakan hasil pengukuran nilai metrik NOC pada *package Controller.*

Tabel 4.8 Nilai Metrik NOC *Package Model*

No	*Class Name*	NOC
1	BerkasModel	0
2	BimbinganModel	0
3	DokumenModel	0
4	JadwalModel	0
5	PengumumanModel	0
6	SidangModel	0
7	TimelineModel	0
8	UserModel	0
9	UserTimelineModel	0

Nilai NOC pada tabel 4.8 keseluruhan *class* berada pada *range* bagus sesuai dengan tabel 2.3 yaitu nilai NOC bernilai di bawah 4.

Nilai rata-rata metrik NOC didapatkan dengan membagi jumlah semua nilai NOC dengan jumlah semua *class*. Perhitungan rata-rata nilai NOC terdapat pada tabel 4.9.

Tabel 4.9 Nilai Rata-Rata Metrik NOC

Package	Jumlah Kelas	Nilai NOC
Controller	9	0
Model	9	0
Total	18	0
Rata - rata	$\frac{\sum NOC}{\sum Class} = \mathbf{0}$	

Hasil perhitungan nilai rata-rata NOC keseluruhan pada tabel 4.9 adalah 0 karena setiap *class* yang diukur semua tidak memiliki *class* turunan sehingga nilainya menjadi 0 yang menunjukkan bahwa keseluruhan nilai metrik NOC aplikasi terletak pada *range good* sesuai referensi pada tabel 2.3.

4.1.5 Response For a Class (RFC)

Tabel 4.10 merupakan hasil pengukuran nilai metrik RFC pada *package Controller*.

Tabel 4.10 Nilai Metrik RFC *Package Controller*

No	*Class Name*	RFC
1	Berkas	19
2	Bimbingan	10
3	Dokumen	9
4	Jadwal	17
5	Pengumuman	17
6	Sidang	25
7	Timeline	11
8	User	34
9	UserTimeline	47

Nilai RFC pada tabel 4.10 keseluruhan *class* berada pada *range* bagus sesuai dengan tabel 2.3 yaitu nilai RFC bernilai di bawah 100.

Tabel 4.11 merupakan hasil pengukuran nilai metrik RFC pada *package Controller*.

Tabel 4.11 Nilai Metrik RFC *Package Model*

No	*Class Name*	RFC
1	BerkasModel	6
2	BimbinganModel	5
3	DokumenModel	5
4	JadwalModel	5
5	PengumumanModel	8
6	SidangModel	7
7	TimelineModel	6
8	UserModel	16
9	UserTimelineModel	13

Nilai RFC pada tabel 4.11 keseluruhan *class* berada pada *range* bagus sesuai dengan tabel 2.3 yaitu nilai RFC bernilai di bawah 100.

Nilai rata-rata metrik RFC didapatkan dengan membagi jumlah semua nilai RFC dengan jumlah semua *class*. Perhitungan rata-rata nilai RFC terdapat pada tabel 4.12.

Tabel 4.12 Nilai Rata-Rata Metrik RFC

Package	Jumlah Kelas	Nilai RFC
Controller	9	189
Model	9	71
Total	18	260
Rata - rata	$\frac{\sum RFC}{\sum Class}$ = **14,4**	

4.2 Analisis Hasil Perhitungan

Pada subbab sebelumnya yaitu pada subbab 4.1.2 hingga 4.1.5 telah dilakukan pengukuran nilai *maintainability* aplikasi dengan menggunakan metriks *chidamber* dan *kemerer* bagian *Weighted Method per Class* (WMC), *Depth of Inheritance Tree* (DIT), *Number of Children* (NOC), dan *Response For a Class* (RFC). Metrik *chidamber* dan *kemerer* digunakan untuk mengukur nilai kualitas *maintainability* aplikasi ”Sistem Informasi Skripsi Informatika Universitas Diponegoro” yang sudah mencakup karakteristik *maintainability* sesuai dengan ISO/IEC 25010.

Hasil perhitungan rata-rata WMC aplikasi menghasilkan nilai 22,83 yang menunjukkan bahwa nilai metrik WMC aplikasi terletak pada *range good* atau keadaan bagus sesuai referensi pada tabel 2.3.

Hasil perhitungan rata-rata DIT aplikasi menghasilkan nilai 1,5 yang menunjukkan bahwa nilai metrik DIT aplikasi terletak pada *range good* atau keadaan bagus sesuai referensi pada tabel 2.3.

Hasil perhitungan rata-rata NOC aplikasi menghasilkan nilai 0 yang menunjukkan bahwa nilai metrik NOC aplikasi terletak pada *range good* atau keadaan bagus sesuai referensi pada tabel 2.3.

Hasil perhitungan rata-rata RFC aplikasi menghasilkan nilai 14,4 yang menunjukkan bahwa nilai metrik RFC aplikasi terletak pada *range good* atau keadaan bagus sesuai referensi pada tabel 2.3.

Secara keseluruhan dapat dikatakan bahwa aplikasi memiliki kualitas *maintainability* yang cukup bagus jika diukur dari hasil perhitungan dengan menggunakan metrik *chidamber* dan *kemerer*. Tetapi terdapat beberapa *class* yang berada pada kondisi *medium* dan memerlukan perbaikan menjadi nilai yang lebih baik dan statusnya menjadi *good*. *Class* yang memerlukan perbaikan status atau nilai ditunjukkan pada tabel 4.13.

Tabel 4.13 *Class* yang Memerlukan Peningkatan Nilai

Package	*Class*	Metrik	Nilai	Kategori
Controller	Sidang	WMC	56	*Medium*
Controller	User	WMC	64	*Medium*
Controller	UserTimeline	WMC	69	*Medium*

4.3 Peningkatan Kualitas

Peningkatan nilai kualitas dilakukan dengan menggunakan metode *refactoring*. Digunakannya metode *refactoring* agar kode lebih mudah dipahami dari segi struktur seperti penghilangan duplikasi kode, dan juga dapat mempermudah menemukan adanya suatu *bugs*.

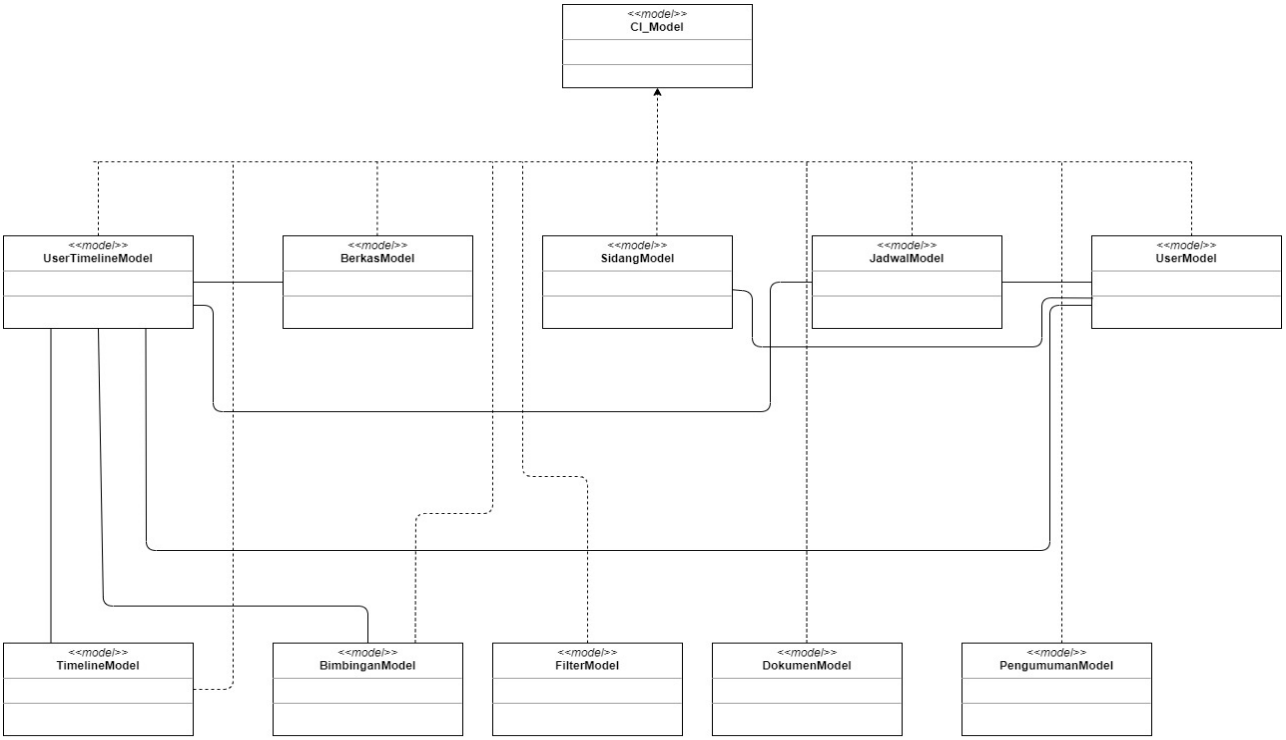
Berdasarkan tabel 4.13 peningkatan akan berfokus pada nilai *Weighted Method per Class* (WMC) 3 *class* pada *package controller*, sehingga *refactoring* akan difokuskan kepada kompleksitas per *class*. *Refactoring* dilakukan dengan mendeteksi adanya *bad smells* pada kode.

Beberapa *bad smells* dan *refactoring* yang diterapkan ditunjukkan oleh tabel 4.14.

Tabel 4.14 *Refactoring* yang diterapkan

No	*Bad Smells*	Penyebab	Solusi
1	*Long Method*	Method yang terlalu panjang dan melakukan banyak tugas	Diubah atau dikelompokkan menjadi method yang lebih pendek
2	Large Class	*Class* yang memiliki banyak komponen	Memisahkan masing-masing ke sebuah *class*
3	Duplicated Codes	Terdapat banyak kode yang sama atau mirip yang berjalan di berbagai tempat	Menjadikan suatu method yang akan dipanggil jika dibutuhkan
4	Long Parameter	Sebuah method yang memiliki banyak parameter	Menjadikan parameternya sebagai objek

Refactoring dilakukan dengan menjalankan solusi untuk mengatasi penyebab dari *bad smells* pada *source code*. *Refactoring* yang dilakukan menghasilkan satu *class* baru yaitu FilterModel yang berada pada *package Model*. Kegunaan dari *class* ini yaitu berisi *method* atau tugas yang mirip pada tiap *class* dikarenakan kompleksnya *method* yang ada pada beberapa tempat, sekaligus juga membantu jalannya. Perubahan struktur *class diagram* hanya terjadi pada *package model* dengan tambahan satu *class* baru yaitu FilterModel. *Class diagram model* baru ditunjukkan oleh gambar 4.3



Gambar 4.3 *Class Diagram Model* Baru

Pengukuran kembali metriks *chidamber* dan *kemerer* dilakukan guna membuktikan adanya peningkatan pada *class* yang membutuhkan peningkatan nilai. Perhitungan metriks dilakukan dengan cara yang sama seperti sebelumnya. Nilai metriks pada *class* yang membutuhkan peningkatan sesuai dengan tabel 4.13 ditunjukkan oleh tabel 4.15.

Tabel 4.15 Pengukuran Terhadap *Class* yang Membutuhkan Peningkatan Nilai

Package	*Class*	Metrik	Nilai	Kategori
Controller	Sidang	WMC	37	*Good*
Controller	User	WMC	43	*Good*
Controller	UserTimeline	WMC	48	*Good*

Pengukuran nilai total keseluruhan metriks *chidamber* dan *kemerer* baru ditunjukkan oleh tabel 4.16.

Tabel 4.16 Nilai Metriks *Chidamber* dan *Kemerer* Baru

No	Metriks	Nilai
1	*Weighted Method per Class* (WMC)	20,14
2	*Depth of Inheritance Tree* (DIT)	1,52
3	*Number Of Children* (NOC)	0
4	*Response For a Class* (RFC).	13,2

Pada tabel 4.16 ditunjukkan nilai total baru dari tiap metriks. Terjadi beberapa perubahan pada nilai total dari sebelum dilakukan *refactoring*. Nilai WMC menurun yang artinya nilai kompleksitasnya menurun yang berarti aplikasi semakin baik karena kompleksitas *class* semakin rendah. Pada nilai DIT terjadi peningkatan sedikit dibanding sebelumnya dikarenakan adanya satu *class* tambahan yaitu FilterModel. Nilai NOC tetap 0 karena tidak adanya perubahan ataupun tambahan struktur *inheritance* dari tiap *class*, dan yang terakhir nilai RFC menurun yang artinya kompleksitas menurun sehingga nilai kualitas aplikasi semakin baik.

BAB V
PENUTUP

5.1 Kesimpulan

Berdasarkan uraian dari pembahasan bab – bab sebelumnya, maka pada penelitian ini dapat disimpulkan :

1. Telah dilakukan pengukuran nilai kualitas *maintainability* aplikasi "Sistem Informasi Skripsi Informatika Universitas Diponegoro" dengan menggunakan metriks *chidamber* dan *kemerer* berdasarkan standar ISO/IEC 25010.
2. Pada *Class diagram* baru dapat dilihat bahwa terdapat tambahan satu *class* baru yaitu *class* FilterModel yang dibuat untuk mengurangi kompleksitas.
3. Setelah dilakukan pengukuran dengan metriks *chidamber* dan *kemerer*, terdapat beberapa perubahan nilai hasil metriks meliputi :
    a. Penurunan kompleksitas *class* yang ditunjukkan oleh penurunan nilai metriks WMC dari 22,83 menjadi 20,14.
    b. Peningkatan kedalaman sebuah *class* yang ditunjukkan oleh kenaikan nilai metriks DIT dari 1,5 menjadi 1,52.
    c. Tidak adanya perubahan struktur *sub-class* yang ditunjukkan oleh ketetapan nilai metrik NOC dari sebelumnya 0 dan menjadi tetap 0.
    d. Penurunan jumlah *message class* yang ditunjukkan oleh penurunan nilai metriks RFC dari 14,4 menjadi 13,2.

5.2 Saran

Berdasarkan hasil penelitian skripsi, beberapa saran yang bisa diberikan untuk pengembangan penelitian lebih lanjut adalah sebagai berikut:

Mengukur aspek lain aplikasi selain *maintainability* atau adanya tambahan metode lain untuk mengukur semua karakteristik standar ISO/IEC 25010.

DAFTAR PUSTAKA

Andry, J. F., Christianto, K., & Wilujeng, F. R. (2019). Using Webqual 4.0 and Importance Performance Analysis to Evaluate E-Commerce Website. *Journal of Information Systems Engineering and Business Intelligence*, *5*(1), 23. https://doi.org/10.20473/jisebi.5.1.23-31

Ayubi, A., Muqtadiroh, F. A., & Nisafani, A. S. (2015). Pengukuran Kualitas dan Perbaikan Struktur Kode Perangkat Lunak Berbasis Object Oriented Programming Menggunakan Metrik Chidamber dan Kemerer (Studi Kasus Software Accounting XYZ). 1–7. https://repository.its.ac.id/id/eprint/72217

Basili, V. R., Briand, L. C., Melo, W. L., & Society, I. C. (1996). A Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Transactions on Software Engineering*, *22*(10).

Cheikhi, L., Al-Qutaish, R. E., Idri, A., & Sellami, A. (2014). Chidamber and kemerer object-oriented measures: Analysis of their design from the metrology perspective. *International Journal of Software Engineering and Its Applications*, *8*(2), 359–374. https://doi.org/10.14257/ijseia.2014.8.2.35

D. Nurul Huda, D. I. S. (2005). Peningkatan Kualitas Sistem Informasi Akademik dan Keuangan Berdasarkan Webqual 4.0 dan ISO/IEC 25010: Studi Kasus Sekolah Tinggi Teknologi Indonesia TanjungPinang.

Dewi, M. R. (2020). Maintainability Measurement and Evaluation of myITS Mobile Application Using ISO 25010 Quality Standard. *International Seminar on Application for Technology of Information and Communication (iSemantic)*, 530–536.

Firdaus, M. B., Puspitasari, N., Budiman, E., Widians, J. A., & Bayti, N. (2019). Analysis of the effect of quality mulawarman university language center websites on user satisfaction using the webqual 4.0 method. *Proceedings of ICAITI 2019 - 2nd International Conference on Applied Information Technology and Innovation: Exploring the Future Technology of Applied Information Technology and Innovation*, 126–132. https://doi.org/10.1109/ICAITI48442.2019.8982143

Gunawan, H., & Triantoro, A. (2017). Sistem Informasi Pengolahan Rapor Kurikulum 2013. *1*, 51–60. https://doi.org/10.21460/jutei.2017.11.6

Kartiko, C. (2019). Evaluasi Kualitas Aplikasi Web Pemantau Menggunakan Model Pengujian Perangkat Lunak ISO / IEC 9126. *8*(1), 16–23.

Langsari, K., Rochimah, S., & Akbar, R. J. (2018). Measuring Performance Efficiency of Application applying Design Patterns and Refactoring Method. *IPTEK Journal of Proceedings Series*, *4*(1), 149. https://doi.org/10.12962/j23546026.y2018i1.3527

Manik, A., Salamah, I., & Susanti, E. (2017). Pengaruh Metode WebQual 4.0 Terhadap Kepuasan Pengguna Website Politeknik Negeri Sriwijaya. *Jurnal Elektro Dan Telekomunikasi Terapan*, 477–484.

Muhlis, A. F. (2017). Penilaian Kualitas Perangkat Lunak Pada Aplikasi Akta Notaris Fidusia di CV. FREDAVELOP.

Pamungkas, C. A., (2017). Pengantar Dan Implementasi Basis Data. Yogyakarta: Deepublish.

Pressman, R.S. (2010). Software Engineering : a practitioner's approach. McGraw-Hill, New York, 68.

Vinet, L., & Zhedanov, A. (2011). A "missing" family of classical orthogonal polynomials. In *Journal of Physics A: Mathematical and Theoretical* (Vol. 44, Issue 8, pp. 476–493). https://doi.org/10.1088/1751-8113/44/8/085201

Wattiheluw, F. H., Rochimah, S., & Fatichah, C. (2019). Klasifikasi Kualitas Perangkat Lunak Berdasarkan Iso/Iec 25010 Menggunakan Ahp Dan Fuzzy Mamdani Untuk Situs Web E-Commerce. *JUTI: Jurnal Ilmiah Teknologi Informasi*, *17*(1), 73. https://doi.org/10.12962/j24068535.v17i1.a820

LAMPIRAN