PENGUKURAN KUALITAS DAN PERBAIKAN STRUKTUR *CODE* PERANGKAT LUNAK BERBASIS *OBJECT ORIENTED PROGRAMMING* MENGGUNAKAN METRIK CHIDAMBER DAN KEMERER. (STUDI KASUS *SOFTWARE ACCOUNTING* XYZ.)

Aula Ayubi[1)], Feby Artwodini Muqtadiroh, S.Kom, M.T.[2)], Amna Shifia Nisafani, S.Kom, M.Sc.[3)]

Jurusan Sistem Informasi, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember (ITS)
Jl. Arief Rahman Hakim, Surabaya (60111) – Indonesia
e-mail: aulaayubi@gmail.com[1)], febyherbowo@gmail.com[2)], mafaniseventh@gmail.com[3)]

ABSTRAK

Peningkatan kualitas sebuah perangkat lunak mempunyai peran yang penting dalam praktek rekayasa perangkat lunak. Pendekatan engineering menginginkan bahwa kualitas perangkat lunak dapat diukur secara kuantitatif dalam bentuk angka yang dapat dipahami oleh orang lain. *Object Oriented Programming* (OOP) seringkali menjadi pilihan pengembang dalam membangun perangkat lunak.

Pengukuran kualitas software berdasarkan aspek *object oriented* dengan menggunakan Metrik Chidamber dan Kemerer. Metrik tersebut mempunyai enam metrik di dalamnya 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).

Karakteristik kualitas yang dapat diukur menggunakan metrik ini berdasarkan ISO/IEC 9126-1 adalah *efficiency, understandability, reusability, dan maintainability/testability*. Sehingga pengembang *software* dapat mengetahui sejauh mana kualitas konsep object oriented yang ada pada kode perangkat lunak tersebut.

Hasil akhir yang diharapkan dari tugas akhir ini adalah perbaikan kode program berupa *class diagram* yang lebih baik berdasarkan kualitas yang telah dipilih.

Kata kunci: Pengukuran Kualitas, *Object Oriented Programming* (OOP), Metrik Chidamber dan Kemerer, ISO/IEC 9126-1

1. PENDAHULUAN

Saat Salah satu faktor penting dari suatu perangkat lunak adalah kualitas [1]. Kualitas perangkat lunak akan mempengaruhi baik buruknya suatu kinerja dari perusahaan yang bersangkutan dalam menjalankan bisnisnya. Perangkat lunak yang berkulitas akan memudahkan perusahaan dalam menjalankan proses bisnisnya sehingga sumber daya yang dikeluarkan oleh perusahaan akan semakin efisien dan efektif.

Peningkatan kualitas sebuah perangkat lunak mempunyai peran yang penting dalam praktek rekayasa perangkat lunak [2]. Oleh karena itu pengukuran kualitas perangkat lunak sangat dibutuhkan agar Perangkat lunak dapat diperbaiki secara terus menerus. Pengukuran kualitas akan memberikan pandangan bagi organisasi atau perusahaan dalam mengalokasikan sumber dayanya sehingga biaya yang dikeluarkan sesuai dengan perencanaan.

Salah satu pengukuran kualitas perangkat lunak adalah pengukuran kualitas kode program pada perangkat lunak tersebut [5]. Pendekatan engineering menginginkan kualitas perangkat lunak dapat diukur secara kuantitatif dalam bentuk angka yang dapat dipahami oleh orang lain. Faktor efisiensi pada kode program akan mempengaruhi manajemen perusahaan terkait perangkat lunak tersebut. Banyak metode pengukuran kualitas kode program seperti metode LOC, Function Point, dan Cocomo [6]. Pengembang perangkat lunak saat ini lebih cenderung menggunakan konsep Object Oriented Programming (OOP) yang memudahkan untuk membangun perangkat lunak. Beragamnya pola implementasi berorientasi object menimbulkan perbedaan pendapat dalam melihat kualitas sebuah perangkat lunak [7].

Metrik Chidamber dan Kemerer telah dipilih dalam pengerjaan tugas akhir ini karena pada penulisan sebelumnya telah menghasilkan pemetaan antara Metrik Chidamber dan Kemerer dengan properti kualitas [11] [12] [13]. Metrik Chidamber dan Kemerer merupakan salah satu kumpulan metrik yang digunakan untuk mengukur kualitas perangkat lunak melalui kode program pada perangkat lunak tersebut. Metrik Chidamber dan Kemerer mengukur kualitas perangkat lunak berdasarkan enam metrik dengan melihat pada perspektif desain berorientasi object [14].

Pengukuran perangkat lunak dengan menggunakan studi kasus Software Accounting XYZ akan menghasilkan nilai kuantitatif yang akan merepresentasikan tingkat efficiency, understandability, reusability, dan maintainability/testability software accounting XYZ. Sehingga tugas akhir ini akan dapat menghasilkan nilai kuantitatif yang mempresentasikan kualitas Software Accounting XYZ. Tugas akhir ini juga menghasilkan rekomendasi perbaikan struktur kode

program berupa class diagram berdasarkan design pattern. Hasil tugas akhir ini diharapkan dapat menjadi referensi bagi pengembang Software Accounting XYZ untuk memeperbaiki desain kode program perangkat lunak tersebut.

2. TINJAUAN PUSTAKA

2.1 *Object Oriented Programming*

Pemograman berorientasi object adalah teknik membuat suatu program berdasarkan object [15]. 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 field/atribut dan method. Field/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 [15].

Object mempunyai dua karakteristik yaitu variabel atau field 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. Cohesian yang tinggi mengindikasikan subdivisi class yang baik. Coupling adalah method dalam sebuah class memanggil method pada class lain. Inheritance adalah hirarki class, terdapat superclass dan subclass yang mewarisi atribut atau method dari superclass. Package merupakan sekelompok class dan interface yang saling terkait [16].

Pemograman object mempunyai empat ciri-ciri yaitu abstraksi (abstraction), pembungkusan (encapsulation), pewarisan (inheritance) dan polimorfisme (polymorphism) [17].

2.2 Metrik Chidamber dan Kemerer

Metrik merupakan suatu prosedur yang memasangkan karakteristik tertentu pada entitas yang diamati menjadi sebuah nilai numerik [1]. 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 adalah salah satu metrik yang digunakan untuk mengukur kualitas disain sebuah perangkat lunak berdasarkan enam metrik dengan melihat pada perspektif desain berorientasi object [22].

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).

Tabel 1. Pemetaan Metrik Chidamber dan Kemerer dengan Kode Program

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

Penelitian sebelumnya telah menghasilkan kategori dari Metrik Chidamber dan Kemerer [24] [25] yang telah ada pada Tabel 2, hijau mewakili kategori *good*, kuning mewakili kategori *medium*, merah mewakili kategori *bad*.

Tabel 2. Nilai Metrik Chidamber dan Kemerer

Metrik Chidamber dan Kemerer	Hijau	Kuning	Merah
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	$x < 100$	$100 \leq x \leq 1000$	$1000 < x$

2.3 ISO/IEC 9126-1

ISO 9126 merupakan *best practice* dalam melakukan evaluasi terhadap kualitas produk perangkat lunak. ISO/IEC 9126 adalah standar internasional yang diterbitkan oleh ISO/IEC, standar ini dibagi menjadi empat bagian yang masing-masing menjelaskan model kualitas, metrik eksternal, metrik internal, dan metrik kualitas [32]

ISO/IEC 9126 mempunyai enam model kualitas yang telah dicantumkan dalam ISO/IEC 9126-1. ISO/IEC 9126-1 (bagian pertama dari ISO 9126) membagi model kualitas perangkat lunak (software quality model) menjadi enam karakteristik yaitu fungsionalitas *(Functinability),* kehandalan *(Reliability),* kebergunaan *(Usability),* efisiensi *(Efficiency),* keterpeliharaan *(Maintainability)* dan portabilitas (*Portability)* [32].

Tabel 3. Pemetaan Quality Model ISO/IEC 9126-1 dengan Metrik Chidamber dan Kemerer

Properti Kualitas Software	Parameter Metric
Efficiency	LCOM, CBO, DIT, NOC
Understandbility	WMC, RFC, DIT
Reusability/ Replaceability	WMC, LCOM, CBO, DIT, NOC
Maintainability	WMC, RFC, DIT, NOC.

Tabel 4. Deskrip Kriteria dan Subkriteria ISO/IEC 9126-1

Kriteria/Sub Kriteria ISO/IEC 9126-1	Deskripsi Kriteria/Sub Kriteria ISO/IEC 9126-1
Understandability	Kemampuan produk software yang memungkinkan pengguna untuk memahami apakah software tersebut cocok, dan bagaimana hal itu dapat digunakan untuk tugas-tugas dan ketentuan penggunaan tertentu
Efficiency	Kemampuan produk software untuk dipahami, dipelajari, digunakan dan atraktif bagi pengguna, bila digunakan dalam kondisi tertentu
Maintainability	Kemampuan produk software untuk dimodifikasi. Modifikasi dapat mencakup koreksi, perbaikan atau adaptasi dari perangkat lunak untuk perubahan lingkungan, dan persyaratan dan spesifikasi fungsional
Replaceability	Kemampuan produk software yang akan digunakan di tempat produk software lain yang ditentukan untuk tujuan yang sama dalam lingkungan yang sama.

Tabel 5. Korelasi Metrik Chidamber dan Kemerer dengan ISO/IEC 9126-1

Metric	*Desirable Value*	*Efficiency*	*Understandability*	*Maintainability*	*Replaceability*
WMC	↓		↑	↑	↑
DIT	↓	↑	↑	↑	↑
NOC	↓	↑		↑	↑
CBO	↓	↑			↑
RFC	↓		↑	↑	
LCOM	↓	↓		↓	↓

2.4 *Design Pattren*

Dalam membuat desain sebuah perangkat lunak biasanya menggunakan asumsi atau pemahaman yang bersifat subjektif sehingga dibutuhkan gambaran secara formal dari masalah dan pemecahannya. *Design patterns* 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) [34].

Dalam pengembangan perangkat lunak sering terjadi permasalahan yang terjadi berulang-ulang, sehingga seorang arsitek mungkin akan banyak menghabiskan waktu dalam memberikan solusi dengan masalah yang serupa. Pada saat merancang perangkat lunak, *patterns* adalah suatu dokumen yang sangat penting untuk dimiliki dan dipahami. Dimana *design patterns* bukan sekedar menyediakan solusi terbaik dalam memecahkan suatu masalah, tetapi lebih dari pada itu *patterns* membuat desain perangkat lunak menjadi lebih efektif, fleksible dan waktu penyelesaikan desain perangkat lunak juga lebih efisien.

Ada banyak *Design Patterns* yang sudah diakui kemampuannya, diterima dan diaplikasikan oleh banyak praktisi. *Design Patterns* yang cukup populer adalah yang diperkenalkan *The Gang of Four* (GoF) - Erich Gamma, Richard Helm, Ralph Johnson dan John Vlissides. Dalam *The Gang of Four* (GoF) terdapat 23 *Pattern* yang dibagi menjadi 3 kelompok besar.

Pada *Gang of Four Patterns*, terdapat tiga katalog *design patterns* 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* [34].

2.5 *Class Diagram*

Pembuatan *class diagram* akan dilakukan setelah proses analisa kode program yang menghasilkan *code* dependency. Sebuah ketergantungan (*dependencies*) mengandung tiga unsur yaitu sumber (*source*), target dan tipe ketergantungan (*dependency kind*) [37].

Tabel 6. *Code dependency*

Source	*Dependency Kind*	Target	Deskripsi
class/interface	*extends*	*class/interface*	*class/interface extends* (perluasan) terhadap *class/interface/* yang lain.
class/interface	*contains*	*class/interface*	*class/interface* pada *source* berisi sebuah *class/interface* yang lain.
class/interface	*contains*	*field*	*class/interface/* pada *source* berisi *field* dari *class* yang lain.
method	*returns*	*class/interface*	*method* pada *source* mengembalikan sebuah nilai berdasarkan *class/interface* yang lain.
method	*has param*	*class/interface*	*method* pada *source* mendeklarasikan sebuah parameter berdasarkan *class/interface* yang lain.
method	*throws*	*class/interface*	*method* pada *source* mendeklarasikan *class/interface/* yang lain dalam mengirim/melempar clause-nya.
method	*calls*	*methods*	*method* pada *source* memanggil *method* pada *class* yang lain.

Source	*Dependency Kind*	Target	Deskripsi
methods	*acceses*	*field*	*method pada source* mengakses *field* pada *class* yang lain.
field	*is of type*	*class/interface*	*field* pada *source* didasarkan pada *class/interface* yang lain.
any	*references*	*class/interface*	*remote* objek yang dipanggil oleh objek lain melalui *remote object refernces.*

Tabel 7. Pemetaan *Dependency Kind* dan *Relationship Class Diagram*

Relationship	*Deskripsi*	*Dependency Kind*
Associations	*Class* yang memiliki atribut berupa *class* lain atau *class* yang harus mengetahui ekstensi *class* lain	*Contains*
		Referencess
		Is Of Type
Dependencies	Operasi suatu *class* yang menggunakan *class* lain atau *field*/atribut *class* lain	*Retruns*
		Has Param
		Throws
		Calls
		Accesses
Nesting	Sebuah *class* yang didefinisikan didalam *class* (*outer class*) lain.	
Realization	Sebuah *class* yang mengimplementasikan *interface*	*Implements (Interface)*
Generalization	*Class* yang menunjukkan hubungan warisan (*inheritance*) dengan *class* lainnya.	*Extendss*

3. METODOLOGI PENELITIAN

Metode pengerjaan penelitian ini terdiri dari 3 tahap. Masing-masing tahap memiliki proses yang disesuaikan dengan tujuan tahap.

3.1 Tahap perancangan

Tahap perancangan meliputi dua tahapan yaitu analisa kode program dan pembuatan *class diagram*.

Analisa kode program *Software Accounting* XYZ dengan memetakan *code* yang ada dan menemukan *dependencies* masing-masing *code*. Sehingga *code Software Accounting* XYZ akan lebih mudah untuk diukur nilainya terhadap Metrik Chidamber dan Kemerer.

Class diagram dibuat berdasarkan hasil analisa dari tahapan sebelumnya yaitu analisa *code program* yang menghasilkan *code dependency*. *Code dependency* dapat memudahkan pembuatan *class diagram* dengan bantuan pemetaan antara relasi pada *class diagram* dengan *dependency kind.*

3.2 Tahap implementasi

Tahap implementasi meliputi perhitungan Metrik Chidamber dan Kemerer, yang terdiri dari Metrik WMC, DIT, NOC, CBO, RFC, dan LCOM.

Tahapan implementasi ini melakukan perhitungan Metrik Chidamber dan Kemerer. *Code Dependency* dan *Class diagram* dapat menjadi acuan untuk menghitung nilai Metrik Chidamber dan Kemerer. Setiap *class* akan diukur nilai Metrik Chidamber dan Kemerer-nya dan selanjutnya dihitung *average* (rata-rata) nya.

3.3 Tahap pembahasan hasil

Tahap pembahasan hasil meliputi empat tahapan yaitu Analisa perhitungan Metrik Chidamber dan Kemerer, membuat *class* diagram baru, menghitung nilai Metrik Chidamber dan Kemerer, dan membuat kesimpulan serta saran.

4. HASIL DAN PEMBAHASAN

Berikut merupakan hasil dan pembahasan dalam penelitian ini.

4.1 Analisa Hasil Perhitungan Metrik Chidamber dan Kemerer

Perhitungan pada bab sebelumnya telah menghasilkan nilai metrik metrik chidamber dan kemerer pada *Software Accounting* XYZ. telah didapatkan nilai *class* yang memerlukan perbaikan. Terdapat dua *class* dengan warna merah yang menunjukkan kategori *bad*, sehingga harus dirubah. Terdapat 22 *class* yang menunjukkan kategori *medium*, sehingga diperlukan perbaikan hingga mencapai status *green* atau *good*.

Tabel 8. *Class* yang Memerlukan Perbaikan

Nama *Package*	*Class*	Metrik	Nilai	Kategori
appLayer	accountsList	WMC	69	Medium
appLayer	configs	WMC	222	Bad
appLayer	appLayer.entry	WMC	132	Bad
dataLayer	dataLayer.DB	CBO	40	Medium
appLayer	appLayer.Messages	CBO	18	Medium
appLayer	account	CBO	15	Medium
appLayer	accountsList	CBO	19	Medium
appLayer	client	CBO	56	Medium
appLayer	configs	CBO	41	Medium
appLayer	entry	CBO	24	Medium
appLayer.taxRelated	taxList	CBO	16	Medium
appLayer.transactionRelated	transaction	CBO	26	Medium
appLayer.transactionRelated	transactions	CBO	18	Medium
GUILayer	Messages	CBO	40	Medium
GUILayer	balanceWindow	CBO	15	Medium
GUILayer	todoWindow	CBO	14	Medium
appLayer	entry	RFC	112	Medium

4.2 *Membuat* Class Diagram *Baru*

Pada bagian ini dijelaskan bagaimana membuat *class diagram* baru sehingga dapat memperoleh nilai Metrik Chidamber dan Kemerer yang lebih baik. Pembuatan *class diagram* berdasarkan pedoman *design pattern* bagian *refactoring.*

Design pattern yang dipilih dalam membuat *class diagram* baru adalah *strategy pattren*. Dalam melakukan *strategy* pattren digunakan *tools refactoring. Refactoring* dan *Design Pattern* memiliki hubungan yang erat. *Design Pattern* adalah pedoman, sedangkan *Refactoring* adalah cara untuk memperbaiki *code*.

Pada pembuatan *class diagram* baru ini terdapat beberapa *refactoring* yang digunakan.

Tabel 9. *Refactoring* yang Digunakan

No	Jenis *Bad Smell*	Diskripsi	Solusi
1	*Duplicated Code*	Terdapat *code* yang sama dan berulang di lokasi yang berbeda.	Menghapus atau diimplementasikan sebagai satu *method* atau *function* tunggal yang akan dipanggil saat diperlukan
2	*Long Method*	*Method* yang ditulis terlalu panjang	Diubah dan dikelompokkan menjadi *method* yang lebih pendek.
3	*Large Class*	*Class* yang besar (terllau banyak *method*) sehingga sulit di dibaca, di mengerti, dan di *maintenence*.	Dikelompokkan menjadi beberapa *class* atau dijadikan *partial class*
4	*Long Parameter List*	Parameter sebuah *method* terlalu banyak sehingga sulit di mengerti	menggunakan tipe objek sebagai parameter
5	*Feature Envy*	Sebuah *method* membutuhkan banyak informasi dari *class* lain daripada *class* nya sendiri	Mengelompokkan *method* yang sesuai
6	*Data Clumping*	Kelompok data yang sama (*field* dalam *class*, parameter dalam *method*) terulang kembali di beberapa tempat dalam sebuah program.	Mengganti dengan enkapsulasi semua data dalam satu objek
7	*Switch Statement*	Switch sebagai pengganti "*if*" biasanya bertebaran dimana-mana	Menggunakan *Polymorphism* untuk menggantikannya
8	*Lazy Class*	*Class* yang tidak mempunyai fungsi atau *class* yang jarang digunakan.	Menggabungkan *class-class* yang hanya memiliki sedikit tanggung jawab, atau menghilangkan *class* yang *method-method* nya tidak dipanggil sama sekali

4.3 Perhitungan Metrik Chidamber dan Kemerer pada *Class Diagram* Baru

Pada bagian ini dilakukan pengukuran Metrik Chidamber dan Kemerer yang baru. Nilai Metrik Chidamber dan Kemerer pada semua *class* telah berubah. Pada Tabel 10 terdapat nilai Metrik Chidamber dan Kemerer yang baru pada *class* yang perlu diperbaiki.

Tabel 10. Pengukuran Matrik Chidamber dan Kemerer terhadap *Class* yang Diperbaiki

Nama *Package*	*Class*	Metrik	Nilai	Kategori
appLayer	accountsList	WMC	53	Medium
appLayer	configs	WMC	75	Bad
appLayer	appLayer.entry	WMC	1	Bad
dataLayer	dataLayer.DB	CBO	37	Medium
appLayer	appLayer.Messages	CBO	14	Medium
appLayer	account	CBO	13	Medium
appLayer	accountsList	CBO	14	Medium
appLayer	client	CBO	49	Medium
appLayer	configs	CBO	35	Medium
appLayer	entry	CBO	0	Medium
appLayer.taxRelated	taxList	CBO	9	Medium
appLayer.transaction Related	transaction	CBO	24	Medium
appLayer.transaction Related	transactions	CBO	25	Medium
GUILayer	Messages	CBO	14	Medium
GUILayer	balanceWindow	CBO	8	Medium
GUILayer	todoWindow	CBO	12	Medium
appLayer	entry	RFC	1	Medium

Menghitung Metrik Chidamber dan Kemerer pada *class diagram* yang baru menggunakan cara yang sama dengan tahapan sebelumnya. Pada tahapan perhitungan *class* diagram baru ini di dapatkan hasil keseluruhan nilai Metrik Chidamber dan Kemerer.

Tabel 11. Nilai Metrik Chidamber dan Kemerer Baru

No.	Metrik	Nilai	Keterangan
1	WMC	8,83	Turun
2	DIT	1,08	Turun
3	NOC	0,09	Turun
4	CBO	6,27	Turun
5	RFC	10,46	Turun
6	LCOM	32,35	Naik

Pada Tabel 11 telah ada nilai Metrik Chidamber dan Kemerer yang baru. Metrik WMC, DIT, NOC, CBO, dan RFC menurun, hal ini menunjukkan kualitas perangkat lunak yang semakin baik karena nilai metrik tersebut berbanding terbalik dengan kriteria kualitas *software* pada ISO/IEC 9126-1. Nilai Matrik LCOM lebih tinggi dibandingkan nilai yang sebelumnya, hal ini menunjukkan kualitas perankat lunak yang semakin baik karena nilai metrik tersebut berbanding lurus dengan kriteria kualitas *software* pada ISO/IEC 9126-1.

5. KESIMPULAN

Kesimpulan yang dapat ditarik berdasarkan hasil dan pembahasan dalam penelitian ini adalah:

1. *Class* diagram baru yang diperbaiki berdasarkan *refactoring* tetap mempunyai 152 *class* namun, terdapat perubahan *method* sehingga tidak ada *class* yang *lazy* atau *class* yang kelebihan *method*.
2. Nilai Metrik Chidamber dan Kemerer lebih baik pada *class diagram* yang baru,
   - Berdasarkan perhitungan Metrik Chidamber dan Kemerer, diperoleh nilai metrik WMC yang semula 10,28 menjadi 8,83; hal ini menunjukkan kenaikan atau perbaikan *understandability*, *maintainability*, dan *replaceability* sebesar 14,11% berdasarkan jumlah *method* atau kompleksitas setiap *class*.
   - Berdasarkan perhitungan Metrik Chidamber dan Kemerer, diperoleh nilai metrik DIT yang semula 1,1 menjadi 1,08; hal ini menunjukkan kenaikan atau perbaikan *efficiency*, *understandability*, *maintainability*, dan *replaceability* sebesar 1,82% berdasarkan kedalaman sebuah *class*.
   - Berdasarkan perhitungan Metrik Chidamber dan Kemerer, diperoleh nilai metrik NOC yang semula 0,12 menjadi 0,09; hal ini menunjukkan kenaikan atau perbaikan *efficiency*, *maintainability*, dan *replaceability* sebesar 25% berdasarkan jumlah *subclass* (*child class*) setiap *class*.
   - Berdasarkan perhitungan Metrik Chidamber dan Kemerer, diperoleh nilai metrik CBO yang semula 7,52 menjadi 6,27; hal ini menunjukkan kenaikan atau perbaikan *efficiency* dan *replaceability* sebesar 16,62% berdasarkan jumlah *Coupling* setiap *class*.
   - Berdasarkan perhitungan Metrik Chidamber dan Kemerer, diperoleh nilai metrik RFC yang semula 11,58 menjadi 10,46; hal ini menunjukkan kenaikan atau perbaikan *understandability*, *maintainability*, dan *replaceability* AA sebesar 9,67% berdasarkan jumlah *Message* setiap *class*.

DAFTAR PUSTAKA

[1] Roni Yunis, Arwin Halim, "Studi Empiris Hubungan Metrik Kohesi Dengan Kecenderungan Kesalahan Pada Aplikasi Berorientasi Objek," *ISSN VOL 15, NO 1*, vol. 15, no. ISSN. 1412-0100, p. 2, 2014.

[2] N. Schneidewind, Body of Knowledge for Software Quality Measuremen, 2002.

[3] T. K. a. N. Y.Liu, Evolutionary Optimization of Software Quality Modeling with Multiple, 2010.

[4] K. K. a. Y. Guéhéneuc, On Issues with Software Quality Models, 2008.

[5] L. R. Ana Paula L¨udtke Ferreira, "A Graph-based Semantics For Object-oriented Programming Constructs," *Science Direct*, p. 1, 2005.

[6] R. S. Wahono, "Teknik Pengukuran Kualitas Perangkat Lunak," *SF*, p. 1, 2006.

[7] M. E. Khan, "Different Forms of Software Testing Techniques for Finding Error," 2010.

[8] J. P. S. K. R. Yeresime Suresh, "Effectiveness of software metrics for object-oriented system," *ScienceDirect*, p. 1, 2012.

[9] S. C. &. C. Kemerer, MOOSE Metrics for Object Oriented Software Engineering, 1993.

[10] F. B. e. Abreu, Design Metric for Object Oriented Software Systems, 2009.

[11] L. H. Rosenberg & L. E. Hyatt, Software Quality Metrics for Object Oriented Enviroments, Unisys Technology Conference, 2003.

[12] J. L. &. M. Hagglund, Maintainability Metrics for Object Oriented Systems, 2004.

[13] M. B. &. A. Deursen, Predicting Class Testability using Object Oriented Metrics, 2004.

[14] S. C. &. C. Kemerer, A Metrics Suite for Object Oriented Design, MIT Sloan Scholl of Managemnet, 2009.

[15] A. R. M, "Konsep Pemograman Berorientasi Objek (OOP)," 2008.

[16] Rosenberg, Linda H., "Metrics for Object Oriented Environments," *EFAITP/AIE Third Annual Software Metrics Conference*, 1999.

[17] azkohaq, "Object Oriented Programming," 2013.

[18] Julio, "Pemograman Berorientasi Objek," 2011.

[19] Jain, Er. V.K., The Complete Guide to java, First Edition, 2001.

[20] Patrick Naughton & Herbert Schildt, "java : The complete reference," McGraw-Hill Professional, UK, 2008.

[21] Hariyanto,Bambang, Ir., MT,, Esensi-esensi Bahasa

Pemrograman, Bandung: Informatika, 2005.

[22] W. R. D. Septian, "Analisis Perbandingan Framework PHP Berdasarkan MOOSE CK dan Properti Kualitas Disain Menggunakan Metode AHP," 2010.

[23] S. R. Chidamber & C. F. Kemerer, "Towards a Metrics Suite for Object Oriented Design," *OOPSLA Conference,* pp. 197-221, 1991.

[24] Houari A. Sahraoui, Robert Godin, Thierry Miceli, "Can Metrics Help Bridging the Gap Between the Improvement of OO Design Quality and Its Automation," [Online]. Available: http://www.iro.umontreal.ca/~sahraouh/papers/ICSM00.pdf.

[25] Victor Basili, Lionel Briand and Walcelio Melo., "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering,* vol. 22, 1996.

[26] Shyam R. Chidamber, Chris F. Kemerer. A, "Metrics suite for Object Oriented design," *M.I.T. Sloan School of Management,* pp. 53-315, 1993.

[27] P. S. Stoecklin, "One hour presentation to inform you of new techniques and practices in software development," Florida State University – Computer Science, Panama, 2003.

[28] Daniela Glasberg, Khaled El Emam, Walcelio Melo, Nazim Madhavji, "Validating Object-Oriented Design Metrics on a CommercialJava Application," *iit-iti.nrc,* 2000.

[29] Khaled El Emam, Walcelio Melo, Javam, C. Machado., "The prediction of faulty classes using object-oriented design metrics," *The Journal of Systems and Software,* vol. 56, p. 2001, 2001.

[30] Chidamber, S., Darcy, D., Kemerer, C., "Managerial use of Metrics for Object Oriented Software," *IEEE Transaction on Software Engineering,* Vols. 24, no. 8,, no. an Exploratory Analysis, pp. 629-639, 1998.

[31] Arti Chhikara and R.S.Chhillar, "Analyzing the Complexity of Java Programs using Object Oriented Software Metrics," IJCSI International Journal of Computer Science Issues, vol. 9, no. 1, No 3, p. 1, 2012.

[32] Ho-Won Jung and Seung-Gweon Kim, Chang-Shin Chung, "Measuring Software Product Quality: A Survey of ISO/IEC 9126," IEEE SOFTWARE , 2004.

[33] ISO/IEC, "Information technology — Software," ISO/IEC 2000, Geneva, 2000.

[34] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides,, "Design Patterns: Elements of Reusable Object-Oriented Software," 1995.