EVALUASI KUALITAS PERANGKAT LUNAK DENGAN METRICS BERORIENTASI OBJEK

Kornelis Letelay[1] , Azhari SN[2]
1Jurusan Ilmu Komputer, Fakultas Sains dan Teknik, Universitas Nusa Cendana
Jl. Adi Sucipto, Penfui, P.O. Box 104, Kupang 85001, Nusa Tenggara Timur
Telp 0380-881580, 821674
2Jurusan Ilmu Komputer FakultasMIPA Universitas Gadjah Mada
Bulaksumur, Yogyakarta 55281, Telp (0274) 588688, Fax (0274) 565223
E-mail : kornelisletelay@ymail.com , arisn.softcom@yahoo.com

Abstrak
Pengembangan perangkat lunak berorientasi objek membutuhkan pendekatan yang berbeda dengan pengembangan perangkat lunak terstruktur. Perbedaan tersebut terdapat pada tahap desain, implementasi, dan evaluasi kualitas perangkat lunak tersebut. Dengan demikian, dibutuhkan metrics yang berbeda dengan traditional metrics untuk melakukan evaluasi kualitas perangkat lunak tersebut. Dalam penelitian ini akan diajukan enam buah object oriented matrics. Keenam metrics tersebut memiliki sudut pandang yang terkait dengan faktor-faktor kualitas perangkat lunak. Dengan demikian, metrics tersebut dapat digunakan untuk melakukan evaluasi kualitas perangkat lunak berorientasi objek dari beberapa faktor kualitas yang terkait., sehingga dapat menjadi rekomendasi bagi pengguna proyek peranhkat lunak dan pembaca dalam melakukan penelitian tentang evaluasi kualitas perangkat lunak berorientasi object maupun penggunaan matrics yang dikembangkan.

Kata Kunci: *Kualitas ,Object oriented matrics , Perangkat Lunak, Kualitas Prangkat Lunak*

1. PENDAHULUAN

Object oriented design adalah pendekatan yang dilakukan untuk membuat solusi dari hubungan antara objek, atribut, dan metode yang dimiliki oleh sistem dengan menggunakan bahasa pemrograman tertentu sehingga objek lain dapat mengenal dan mengaksesnya. Menurut Mathiassen et al. (2000, p4), *"a design object's identity expresses how other objects in the systems can recognize it, and thereby gain access to it"*. Sedangkan menurut Whitten et al. (2004, p686), *"Object oriented design is an approach used to specify to software solution in terms of collaborating objects, their attributes, and their methods"*.

Teknologi rancang bangun berorientasi-objek merupakan komplemen dan dalam banyak kasus menggantikan kedudukan rancang bangun terstruktur sebagai teknologi tinggi yang bagus ("*good*"). Seiring dengan perkembangan itu literatur dan alat bantu (*case tool*) untuk rancang bangun ber-orientasi objek juga telah banyak di rilis para pakar teknologi *objek-oriented* (Meyer, 1997).

Pengembangan berorientasi objek berbeda dari pengembangan konvenisonal yang memandang perangkat lunak sebagai fungsi dan data yang terisolasi (Hariyanto, Bambang . 2004)

Tradisional metrics seperti *Cylomatic Complexity* (CC), yang digunakan untuk mengevaluasi desain terstruktur, ternyata tidak dapat digunakan begitu saja dalam desain berorientasi objek. Bahkan beberapa *tradisional metrics* tidak dapat digunakan sama sekali dalam lingkungan berorientasi objek. Oleh karena itu dibutuhkan *metrics* yang dapat merefleksikan desain berorientasi objek (M. L. Brodie and D. Ridjanovic, 1984)

Banyak penelitian yang mengajukan *object oriented matrics*, salah satu penelitian dilakukan oleh Chidamber dan Kemerer (1994). Chidamber dan Kemerer, yang merupakan pelopor dalam penelitian *object oriented matrics*, mengajukan enam buah *object oriented matrics. Matrics* tersebut adalah :

1. *Weigted Method per Class* (WCM)
2. *Depth of Inheritance Tree* (DIT)
3. *Number of Children* (NOC)
4. *Couping Between Object* (CBO)
5. *Response for A Class* (RFC)
6. *Lack of Cohesion in Method* (LCOM)

Keenam *Matrics* tersebut dapat digunakan di dalam lingkungan berorientasi objek.

Penelitian ini membahas penggunaan Chidamber dan Kemerer *object oriented metrics* untuk mengevaluasi perangkat lunak berorientasi objek. Oleh karena itu dibutuhkan *metrics* yang merefleksikan desain berorientasi objek

2. TINJAUAN PUSTAKA

2.1 Kualitas

Kualitas adalah kesesuaian dengan tujuan atau manfaatnya (Juran, 1962) . sedangkan menurut "Crosby (1979) "kualitas adalah kesesuaian dengan kebutuhan yang meliputi availability, delivery, realibility, maintainability, dan cost effectiveness. Selain itu berikut ada pengertian secara arti luas (Bina Produktivitas Tenaga Kerja, 1998:24-25) adalah:

(1) Derajat yang sempurna (degree of exelence): mengandung pengertian komperatif terhadap tingkat produk (grade) tertentu.
(2) Tingkat kualitas (quality level): mengandung pengertian kualitas untuk mengevaluasi teknikal.
(3) Kesesuaian untuk digunakan (fitness for purpose user satisfaction): kemampuan produk atau jasa dalam memberikan kepuasan kepada pelanggan.

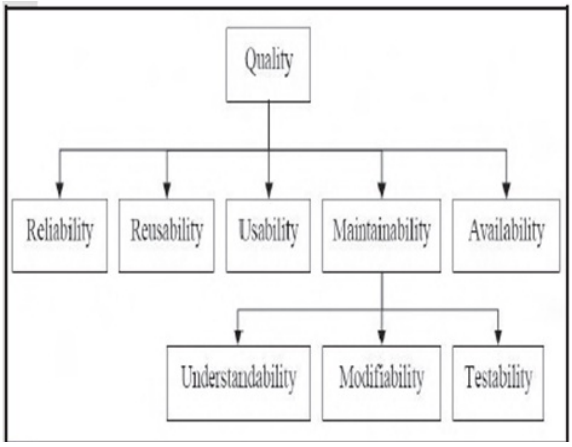
Sedangkan ada delapan dimensi kualitas yang dijelaskan menurut Philip Kotler (2000:329-333) adalah sebagai berikut :

(1) Kinerja (performance): karakteristik operasi suatu produk utama,
(2) Ciri-ciri atau keistimewaan tambahan (feature),
(3) Kehandalan (reliability): probabilitas suatu produk tidak berfungsi atau gagal,
(4) Kesesuaian dengan spesifikasi (conformance to specifications),
(5) Daya Tahan (durability),
(6) Kemampuan melayani (serviceability)
(7) Estetika (estethic): bagaimana suatu produk dipandang dirasakan dan didengarkan, dan
(8) Ketepatan kualitas yang dipersepsikan (perceived quality).

2.2 Kualitas Perangkat Lunak

Berbagai macam definisi kualitas perangkat lunak (software quality) tergantung dari mana pemakai (user) memandang dan melihat sesuai dengan kebutuhannya. Menurut Crosby (1979:34) mendefinisikan kualitas atau mutu sebagai "conformance to requirements". Selama seseorang dapat berdebat tentang perbedaan antara kebutuhan, keinginan dan kemauannya, definisi kualitas harus mempertimbangkan perspektif pemakai tersebut. Lain halnya jika dilihat dari sudut pandang pemgembangan. Pengembangan yang menggunakan pemikiran berorientasi objek memiliki tujuan pada terpenuhinya karakteristik tertentu. Terpenuhinya karakteristik-karakteristik tersebut merupakan kualitas dari sudut pandang pengembangan (El-Ahmadi, 2006).

Dengan kata lain, jika diinginkan kualitas perangkat lunak yang tinggi, haruslah ditentukan karakteristik apa saja yang diinginkan. Karakteristik tersebut berupa faktor-faktor kualitas. Faktor-faktor tersebut dapat dilihat pada gambar 1.



Gambar 1. Faktor-faktor kualitas perangkat lunak (El-Ahmadi 2006).

Definisi faktor-faktor tersebut sebagai berikut:

1. *Reliability*
   Menurut Jawadekar (2004), *reliability* adalah tingkat ketepatan fungsi-fungsi perangkat lunak tanpa adanya kesalahan.
2. *Reusability*
   Reusability adalah tingkat kemampuan perangkat lunak atau bagian perangkat lunak untuk dapat digunakan ulang di lain tempat sebagai komponen yang *reusable* (Jawadekar 2004).
3. *Usability*
   Usability adalah tingkat usaha yang dibutuhkan untuk mempelajari, mengoperasikan, dan

menggunakannya untuk tujuan yang diinginkan (Jawadekar, 2004).

4. *Maintainability*
   Maintainability adalah tingkat usaha untuk mengetahui letak kesalahandan memperbaikinya. Faktor ini dapat dipecah menjadi tiga subfaktor, masing-masing *understandability, modifiability,* dan *testability* (El-Ahmadi 2006).
5. *Understandability*
   Understandability terkait dengan peningkatan kompleksitas psikologis (SATC 1995).
6. *Modifiability*
   Modifiability dari segi *flexibility* adalah tingkat kemudahan sebuah sistem atau komponen agar dapat dimodifikasi untuk penggunaan dalam suatu aplikasi atau lingkungan (IEEE Std. 610.12 diacu dalam Barbacci 2004).
7. *Testability*
   Menurut Jawadekar (2004), *testability* adalah tingkat usaha yang dibutuhkan untuk menguji perangkat lunak dalam proses *quality assurance.*
8. *Availability*
   Availabillity adalah tingkat kemudahan suatu sistem atau komponen agar dapat dioperasikan dan diakses ketika ingin digunakan (IEEE Std. 610.12, diacu dalam Barbacci 2004).
9. *Complexity*
   El-Ahmadi (2006) menyatakan masih terdapat satu faktor lagi yang tidak secara langsung terkait dalam hierarki faktor pada gambar 1. Faktor tersebut adalah *complexity. Complexity* dibagi menjadi dua jenis, yaitu *complexity of problem* dan *complexity of solution. Complexity of problem* adalah jumlah sumber daya yang dibutuhkan untuk sebuah solusi yang optimal terhadap sebuah permasalahan. *Complexity of solution* terdiri dari *time complexity* dan *space complexity*. *Time complexity* terkait dengan sumber daya waktu sedangkan *space complexity* terkait dengan sumber daya memori (Fenton & Pfleeger 1997). *Complexity* berpengaruh pada *understandability*, *modifiability*, dan *testability* (El-Ahmadi 2006).

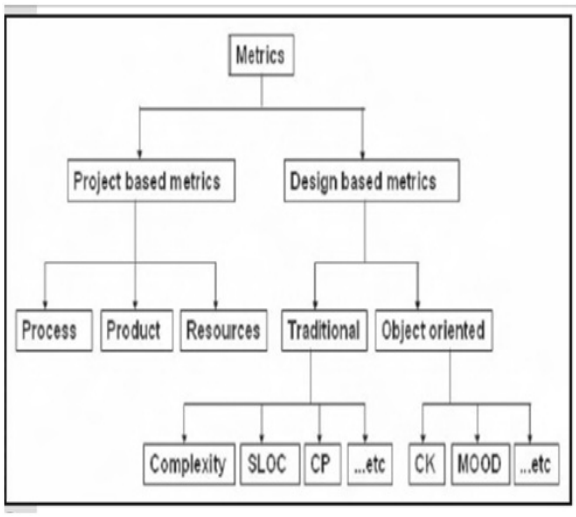
Semua faktor kualitas tersebut dapat diukur secara kuantitatif menggunakan *metrics*.

Metrics

Sarker (2005) membagi *metrics* ke dalam dua kategori, yaitu *project based metrics* dan *design based metrics. Project based metrics* terdiri dari *process*, *product*, dan *resources* sedangkan *design based metrics* terdiri dari *traditional metrics* dan *object oriented metrics.* Gambar 2 memperlihatkan hierarki *metrics* tersebut.

Process metrics, disebut juga *management metrics*, berhubungan dengan proses dalam mengembangkan sebuah sistem (Systa & Yu 1999). *Process metrics* biasanya digunakan untuk :

- Membantu memprediksi ukuran akhir sebuah sistem,
- Memprediksi tingkat usaha yang dibutuhkan dalam sebuah proyek,
- Dan menentukan apakah sebuah proyek sesuai dengan jadwal.



Gambar 2. Hirarki Metrics (Saker, 2005)

Produk metrics, disebut juga quality metric, digunakan untuk mengontrol kualitas dari produk perangkat lunak. Metrics ini digunakan untuk perangkat lunak yang belum selesai dibuat agar dapat diprediksi properti dan kompleksitas hasil akhir perangkat lunak tersebut (Sarker, 2005).

Resources adalah entitas yang dibutuhkan dalam aktifitas proses pengembangan. Resources yang diukur adalah semua input dalam menghasilkan perangkat lunak (Sarker, 2005).

Pada sistem berorientasi objek, tardisional metrics umumnya digunakan dalam ruang linhkup methods yang terdiri atas operasi-operasi sebuah class. Beberapa contoh tradisional metrics antara lain cyclomatic complexity (CC), source line of code (SLOC), dan comment percentage (CP). (Chidamber & Kemerer 1994)

Object oriented metrics digunakan untuk merefleksikan perangkat lunak berorientasi objek (Chidamber & Keremer, 1994).

Object Oriented Metrics

Systa & Yu (1999), membagi object oriented metrics ke dalam tiga kategori, yaitu :

1. Inheritance metrics, mengukur hieraki pewarisan.
2. Complexity metrics, memperkirakan komleksitas perangkat lunak.
3. Communication metrics, mengukur pasangan dan keterpaduan antar class.

Object oriented metrics yang diajukan oleh Chidamber & Kemerer (1994) adalah:

1. Complexity metrics terdiri dari weighted *methods per class (WMC).*
2. Inheritance metrics terdiri dari depth of *in heritance tree (DIT) dan number of children (NOC).*
3. Communication metrics terdiri dari *coupling between object (CBO), response fo r a class (RFC), dan lack of cohesion in method (LCOM).*

Weighted Methods per Class (WMC)

Misalnya sebuah class C1 memiliki methods M1, M2,..Mn dan c1, c2, ...cn, adalah kompleksitas tiap methods tersebut, maka :

$$WMC : \sum_{i=1}^{n} C1$$

Kompleksitas methods dapat didefinisikan oleh masing-masing pengembang (Chidamber & Kemerer 1994). Penelitian ini menggunakan cyclo matic complexity untuk perhitungan kompleksitas tiap method.

Semakin banyak methods sebuah class, semakin besar pula pengaruh pada class yang mewarisinya. Akibatnya, semakin besar upaya dan waktu yang dibutuhkan untuk maintenance dan testing (childamber & kemerer, 1994 ; Lindroos, 2004). Selain itu, class yang dimiliki banyak method yang kompleks berarti class tersebut semakin spesifik. Hal ini dapat membatasi reuse (childamber & kemerer, 1994; Systa & Yu, 1999; Lindroos, 2004).

Depth of Inheritance Tree (DIT)

Kedalaman sebuah class dalam hierarki pewarisan diukur oleh DIT. Apabila terdapat multiple inheritance, DIT didapat dari panjang maksimum dari node ke root pada tree tersebut (Chidamber & Kemerer ,1994).

Semakin dalam sebuah class dalam sebuah heararki pewarisan, semakin besar pula potensi penggunaan ulang method yang diwarisinya (Chidamber & Kemerer, 1994; Systa & Yu, 1999; Lindroos, 2004). Selain itu, semakin dalam sebuah class dalam hierarki pewarisan maka semakin banyak method yang dimilikinya, baik method pada class itu sendiri maupun method yang diwarisisnya. Hal ini mengakibatkan class tersebut semakin kompleks (Systa & Yu, 1999 dalam Lidroos, 2004).

Number of Children (NOC)

NOC adalah jumlah subclasses langsung yang mewarisi sebuah class pada hierarki *class (Chidamber & Kemerer 1994). Semakin* besar NOC maka semakin tinggi reusability. Hal ini dikarenakan pewarisan merupakan bentuk dari reuse. Selain itu, semakin besar NOC maka semakin besar upaya dan waktu untuk testing. (Chidamber & Kemerer, 1994; Lindroos 2004). Selain itu, semakin banyak SATC 1995; Lindroos 2004; Sarker 2005).

Coupling between object (CBO)

CBO didefinisikan untuk sebuah class, yaitu banyaknya class yang terpasang oleh class yang diukur. Sebuah class A dikatakan terpasang dengan class B jika class A menggunakan method atau intance variable pada class B (Chidamber & Kemerer, 1994).

CBO yang berlebihan merupakan perusak desain modular dan mengurangi reuse. Semakin tidak tergantung sebuah class dengan class lain maka semakin mudah class tersebut digunakan ulang untuk aplikasi lain (Lindroos, 2004). Selain itu, class dengan CBO yang besar berarti semakin sensitif class tersebut terhadap

perubahan pasangan class-nya. Hal ini mengakibatkan upaya untuk maintenance class tersebut semakin besar (Chidamber & Kemerer, 1994; Systa & Yu, 1999; Lindroos, 2004).

Response for A Class (RFC)

RFC adalah |**RS**|. RS didefinisikan sebagai berikut :

RS = {*M*}*UaI1 i*{*Ri*}

Dimana {Ri} adalah himpunan method yang dipanggil oleh method i dan {M} adalah himpunan semua method yang ada di class tersebut (Chidamber & Kemerer, 1994).

Semakin banyak method yang dapat dipanggil sebagai respon sebuah pesan, semakin besar juga upaya dan waktu untuk testing. Hal ini dikarenakan tester membutuhkan pemahaman yang tinggi terhadap class tersebut sebelum melakukan testing (Chidamber & Kemerer, 1994; Lidroos, 2004). Selain itu, semakin banyak method yang dapat dipanggil pada sebuah class, semakin besar kompleksitas class tersebut. Ini berarti semakin besar upaya pemeliharaan class tersebut (El-Ahmadi, 2006).

Lack of Cohesion in Methode (LCOM)

Misalkan sebuah class C1 dengan n buah method M1, M2,..., Mn. Lah Lalu {1j} adalah himpunan intance variabel yang digunakan oleh method Mi. Dengan demikian terdapat n buah himpunan I1, I2, ..., In.

Kemudian P = {I1, Ij |Ii n Ij = Q} dan Q =
{I1, Ij |Ii n Ij Ø}. Jika semua n himpunan {I1}, {I2},..., {In} adalah Ø maka P = Ø.

$$LCOM = \begin{cases} |P| - |Q|, & |P| > |Q| \\ 0, & \text{Selainnya.} \end{cases}$$

LCOM diharapkan rendah pada sebuah class (cohesiveness tinggi) karena kenaikan encapsulation. LCOM yang tinggi (cohesiveness rendah) menandakan sebuah class yang semestinya dipecah menjadi 2 atau lebih class. Selain itu, LCOM yang tinggi menandakan kompleksitas yang tinggi (Chidamber & Kemerer, 1994; Ledroos, 2004).

Sampai saat ini, belum dapat ditemukan berapa besar nilai batas atas metric ini. Akan tetapi, dari definisi Chidamber dan Kemerer (1994), nilai LCOM bergantung pada jumlah methods pada sebuah class. Dengan demikian, terdapat nilai maksimum LCOM pada class tersebut, yaitu |Q| = 0.

Semakin kecil nilai aktual LCOM dibanding nilai maksimumnya, semakin baik cohesiveness clas tersebut (Rosenberg, 1998).

Cyclomatic Complexity (CC)

CC, disebut juga McCabe cyclomatic complexity, digunakan untuk evaluasi kompleksitas sebuah method (Rosenberg at all, 1997 diacu dalam Sarker, 2005). CC adalah jumlah test class yang dibutuhkan untuk menguji methods secara komprehensif (Rosenberg, 1998). Perhitungan dapat dilakukan dengan menggambarkan urutan program sebuah method menjadi graph dengan semua path yang mungkin terjadi.

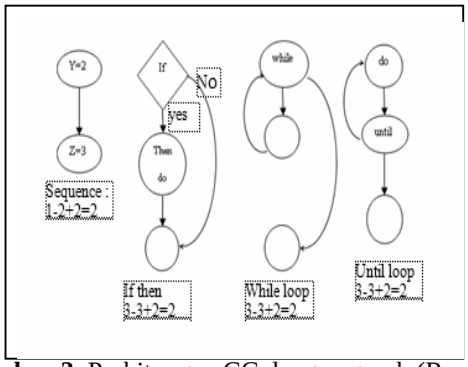
Kompleksitasnya dihitung dengan rumus :

v(G) = e – n + 2

dimana,

- **v(G)** adalah clymatic complexity untuk graf G.
- **e** adalah jumlah edges pada graf G, dan
- **n** adalah jumlah node pada graf G.

Gambar 3, menjelaskan bagaimana perhitungan CC pada sebuah graf.



Gambar 3. Perhitungan CC dengan graph (Rosenberg, 1998)

Ada cara lain menghitung v(G). Anderson dan Vestergren, 2004). merumuskannya sebagai berikut :

$$v(G) = P + 1$$

dimana,

- P adalah jumlah predicate nodes yang ada dalam graph G.
- Predicate node adalah sebuah node yang mempresentasikan pernyataan Boolen di dalam code sehingga terdapat dua buah *outgoing edges.*

Method dengan CC yang rendah umumnya lebih baik (El-Ahmadi, 2006). CC yang rendah berarti semakin baik dalam faktor testability dan understandability. Semakin kompleks sebuah method (CC tinggi) maka semakin tinggi testability dan understanbility. Akan tetapi, method yang memiliki CC yang rendah bukan berarti method tersebut tidak kompleks. Hal ini mungkin berarti bahwa method tersebut message passing dalam implementasinya (SATC, 1995).

3. PEMBAHASAN

3.1 Analisis Metrics Berorientasi Objek

Pada bagian ini, sesuai dengan kajian diatas dilakukan analisis terhadap 6 metrics berorientasi objek dengan masing-masing fungsi, sebagai berikut :

1. ***Weighted Methods per Class (WMC)***
   WCM mempengaruhi terhadap faktor-faktor kualitas, yaitu :
   a. Maintainability (testability)
   b. Usability
   c. Reusability
2. ***Depth of Inheritance Tree (DIT)***
   DIT berpengaruh pada :
   a. Reusability
   b. Understandability
   c. Modifiability
   d. dan Testability.
3. ***Number of Children (NOC)***
   NOC berpengaruh pada :
   a. Reusability
   b. dan Testability.
   RefactorIT menyarankan nilai batas atas NOC sebesar 10 untuk sebuah class.
4. ***Coupling between object (CBO)***
   CBO berpengaruh dengan :
   a. Reusability
   b. *dan Maintainability*.
5. ***Response for A Class (RFC)***
   RFC terkait dengan :
   a. Maintainability
   b. Understandability
   c. Modifiability
   d. dan Testabilit.
6. ***Lack of Cohesion in Methode (LCOM)***
   LCOM berpengaruh pada :
   a. Understandability
   b. Modifiability
   c. Testability

3.2 Kualitas Perangkat Lunak

Dari hasil analisis keenam metrics, dapat dianalisis lebih lanjut seberapa baik kualitas perangkat lunak yang dipakai berdasarkan faktor-faktor kualitas yang terkait dengan keenam metrics tersebut. Tinggi rendahnya nilai metrics akan mempengaruhi pada tinggi rendahnya faktor-faktor kualitas yang terkait pada masing-masing metrics. Sebagai contoh, nilai WMC yang tinggi menyebabkan menurunnya tingkat *Maintainability (testability), Usability, Reusability.*

Pada pembahasan sebelumnya telah dipaparkan bahwa penentuan nilai metrics pada perangkat lunak yang dipakai itu rendah atau tinggi. Dengan demikian, dapat diketahui seberapa tinggi reusability, usability, understandability, modifiability, dan testability perangkat lunak dari nilai-nilai yang dipakai untuk pengukuran keenam metrics dan untuk mengetahui nilai hasil dari pengukuran tersebut.

4. KESIMPULAN DAN SARAN

4.1 Kesimpulan

Enam metrics yang diajukan oleh Chidamber dan Kemerer (1994) dapat digunakan untuk mengetahui kualitas perangkat luanak berorientasi objek dari beberapa faktor kualitas. selain itu, metrics tersebut juga dapat digunakan untuk mengetahui class mana saja yang perlu diperiksa ulang/atau direvisi. Akan tetapi LCOM tidak dapat digunakan langsung untuk mengetahui class mana saja yang perlu diperiksa ulang/atau direvisi.

Berdasarkan pembahasan diatas telah dijelaskan bahwa penentuan kualitas perangkat lunak dilakukan dengan penentuan nilai yang dipakai untuk mengukur rendah atau tingginya kualitas perangkat lunak dengan menggunakan enam metrics beroreintasi objek.

4.2 Saran

Pada penelitian selanjutnya disarankan menggunakan object oriented metrics lainnya sehingga kualitas perangkat lunak dapat diketahui dari faktor-faktor kualitas lainnya. Selain itu, selanjutnya dapat pula mengimplementasikan Chidamber dan Kemerer metrics untuk penelitian-penelitian yang berhubungan dengan kualitas perangkat lunak.

PUSTAKA

Barbacci, Mario R. 2004. *Software Quality*
Atributes: Modifiability And Usability.
Pittsburgh: Carnegie Mellon University.

Chidamber, S dan Chris F. Kemerer. 1994. *A Metrics Suite for Ob ject Oriented Design.*
IEEE Transaction on Software Engineering, vol. 20 no. 6.

El-Ahmadi, Ab dellatif. 200 6. *Software Quality Metrics for Object Oriented Systems. Kongens Lyngby : Technical* University of Denmark.

SATC. 1995. *Software Quality Metrics for Object Oriented System Environments.* Grenbelt aryland : NASA Goddard Space Flight Center.

Fenton, Norman E. & Shari Lawrence Pfleeger. 1997. Software Metrics: *A Rigourous and Practical Approach.*
Boston: PWS Publishing Company

Hogan, Jer. 1997. *An Analysis of OO Software Metrics. Coventry: Department of Computer Science*, University of Warwick.

Jawadekar, Waman S. 2004. *Software Engineering: Principles and Practice.* New Delhi: Tata McGraw-Hill.

Lindroos, Jaana. 2004. Code and Design *Metrics for Object-Oriented Systems.* Helsinski: Department of Computer Science, University of Helsinsky.

Rosenberg, Linda H. 1998. Applying and *Interpreting Object Oriented Metrics.*
Utah: SATC NASA.

Sarker, Muktamyee. 200 5. *An Overview of Object Oriented Design Metrics [Master* Thesis]. Umea: Department of Computer Science, Umea University

SATC. 1995. Software Quality Metrics for *Object Oriented System Environments.* Grenbelt Maryland : NASA Goddard Space Flight Center

Systa, Tarja & Ping Yu. 1999. Using OO *Metrics and Rigi to Evaluate Java Software. Tampere: Department of* Computer Science, University of Tampere.