**BAB II**

6

**LANDASAN TEORI**

1. **Perangkat Lunak**

Komputer merupakan mesin yang memproses fakta atau data menjadi informasi. Komputer di gunakan orang untuk meningkatkan hasil kerja dan memecahkan berbagai masalah. Yang menjadi pemroses data atau pemecah masalah itu adalah perangkat lunak. Perangkat lunak adalah perintah (program komputer) yang bila di eksekusi memberikan fungsi dan unjuk kerja seperti yang di inginkan, struktur data yang memungkinkan program memanipulasi informasi secara proporsional, dan dokumen yang menggambarkan operasi dan kegunaan program. Tidak ada lagi definisi yang lebih lengkap yang dapat ditawarkan, tetapi kita membutuhkan lebih dari sekedar definisi formal.

Jadi menurut Roger S (2002:10) mendefinisikan perangkat lunak yaitu “Perintah program komputer yang bila di eksekusi memberikan fungsi dan unjuk kerja seperti yang di inginkan”. Menurut Melwin (2007:22) mendefinisikan perangkat lunak yaitu “Berfungsi sebagai pengatur aktivitas kerja komputer dan semua intruksi yang mengarah pada sistem komputer. Perangkat lunak menjembatani interaksi user dengan komputer yang hanya memahami bahasa mesin”. Perangkat lunak dibangun berdasarkan permintaan atau kebutuhan penggunanya.

Perangkat lunak memiliki bermacam-macam spesifikasi tergantung kebutuhan dan tujuannya yang diinginkan oleh pembuatan perangkat lunak tersebut. Sifat dari perangkat lunak berbeda dengan perangkat keras, yaitu apabila perangkat keras merupakan komponen yang nyata yang dapat dilihat dan disentuh oleh manusia, sedangkan perangkat lunak tidak dapat disentuh dan dilihat secara fisik. *Software* memang tidak tampak secara fisik dan tidak berwujud benda namun dapat dioperasikan oleh manusia2.

Sifat dan karakteristik dari *software* adalah sebagai berikut3 :

1. *Software* merupakan elemen sistem logik dan bukan elemen sistem fisik seperti *hardware*
2. Elemen itu tidak aus, tetapi bisa rusak.
3. Elemen *software* itu direkayasa atau dikembangkan dan bukan dibuat di pabrik seperti *hardware*.
4. *Software* itu tidak bias dirakit atau disusun.

Perangkat lunak mempunyai karakteristik sebagai berikut2 :

1. *Maintanability* (dapat dirawat), perangkat lunak harus dapat memenuhi perubahan kebutuhan.
2. *Dependability*, perangkat lunak harus dapat dipercaya.
3. *Efisisensi*, perangkat lunak harus efisien dalam penggunaan *resource*.
4. *Usability*, perangkat lunak harus dapat digunakan sesuai dengan yang direncanakan.
5. Ian Sommerville, *Software Engineering Ninth Edition,* 2011

Perangkat lunak mempunyai mutu yang dipengaruhi oleh tiga pihak (minimal) yaitu3 :

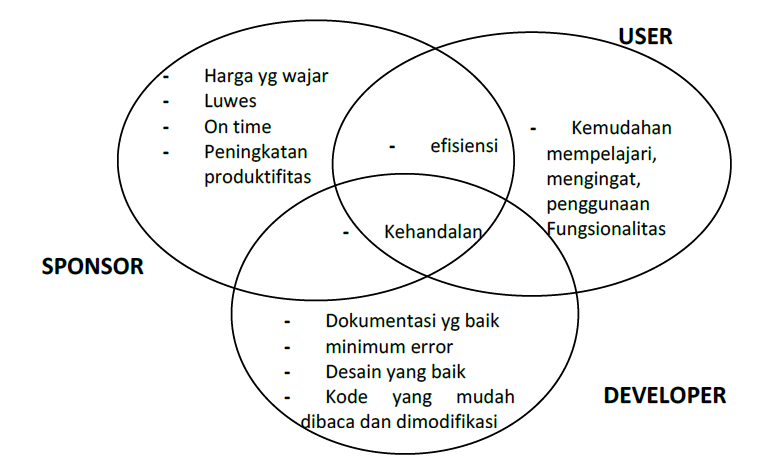
1. Sponsor

Seseorang atau oraganisasi yang membiayai atau membayar selama pengembangan atau perantaraan sistem *software* dan biasanya mempunyai respon terhadap pengembangan sistem *software* itu sendiri dengan melibatkan perhitungan biaya yang optimal.

1. *User*

Setiap orang yang secara langsung berinteraksi terhadap eksekusi *software*, yang secara langsung member *input* ke komputer dan menggunakan atau menikmati *output* dari komputer.

1. *Developer*

Seseorang atau organisasi yang memberikan modifikasi dan memelihara terhadap *error* serta mengembangkan sistem *software* tersebut.

Gambar 2.1 sisi pandang dari komponen kategori terhadap mutu

perangkat lunak

1. Eddy Prasetyo Nugroho et al, Rekayasa Perangkat Lunak, 2009

Masing-masing komponen kategori mempunyai sudut pandang tersendiri terhadap mutu suatu perangkat lunak tetapi kriteria tersebut tidak saling independen.

1. **Pengukuran**

Pengukuran merupakan dasar dari setiap disiplin rekayasa yang berlaku juga dalam perekayasaan suatu perangkat lunak. Pengukuran dapat dikategorikan dalam dua cara, yaitu :

1. Pengukuran langsung

Pengukuran langsung dalam proses rekayasa perangkat lunak berhubungan dengan biaya dan sumber daya yang diperlukan. Pengukuran secara langsung lebih mudah dilakukan, Karena hasil dapat diperoleh secara langsung.

1. Pengukuran tidak langsung

Pengukuran tidak langsung dari suatu produk berhubungan dengan fungsionalitas, kualitas, kompleksitas, efisiensi, realibilitas, dan sebagainya. Pengukuran tidak langsung akan sulit dilakukan, karena harus melalui proses yang lebih kompleks.

Melalui pengukuran yang dilakukan, maka akan diperoleh tingkat pencapaian di dalam proyek perangkat lunak yang sedang diamati. Pengukuran berfungsi untuk mengevaluasi performa suatu sistem atau proses yang diperlukan suatu mekanisme dalam mengamati dan menentukan tingkat efisiensinya. Untuk setiap pengukuran yang akan dilakukan maka dibutuhkan suatu ukuran kuantitatif yang disebut metrik.

Ukuran merupakan faktor utama untuk menentukan biaya, penjadwalan, dan usaha. Kegagalan dari perkiraan ukuran yang tepat yang dilakukan mengakibatkan penggunaan biaya yang berlebih atau keterlambatan penyelesaian proyek. Pengukuran memiliki beberapa kriteria, antara lain :

1. Obyektif, yaitu pengukuran dilakukan lewat pendekatan yang obyektif tidak subyektif menggunakan semua tester yang mungkin dilakukan.
2. Reliabilitas yaitu pengukuran reliable (stabil dan presisi) jika dalam pengulangan yang dilakukan dalam kondisi yang sama, juga didapatkan hasil yang sama.
3. Validitas, yaitu pengukuran valid jika hasil pengukuran memenuhi karakteristik kualitas.
4. Normalisasi, yaitu diperlukan untuk dapat memiliki skala hasil pengukuran dapat dipresentasikan dengan mudah. Ini berhubungan dengan skala skalabilitas.
5. Mudah dibandingkan, yaitu pengukuran mudah dibandingkan ketika diatur suatu relasi ke pengukuran lainnya.
6. Ekonomis, yaitu pengukuran harus memiliki biaya rendah, tergantung pada derajat otomatisasi dan nilai pengukuran yang biasanya digunakan untuk pemilihan penggunaan perangkat bantu jenis tertentu.
7. Berguna, iatu mudah dibuktikan dengan validitas dan amat berguna dalam evaluasi kualitas4.
8. Analisis Kinerja Sistem, Teknik Pengukuran:14
9. **Kualitas Perangkat Lunak**

Kualitas merupakan suatu hal yang sulit didefinisikan, tetapi kualitas dapat didefiniskan sesuai dengan tingkat kemampuan suatu perangkat lunak. Kualitas perangkat lunak dapat didefinisikan sebagai suatu proses perangkat lunak yang efektif diterapkan pada yang menyediakan nilai yang dapat diukur untuk mereka yang menghasilkan dan untuk mereka yang menghasilkannya5.

Perangkat lunak yang bagus yaitu perangkat lunak yang memiliki fitur-fitur dan fungsi-fungsi berkualitas tinggi sesuai dengan keinginan pengguna. Kualitas sebuah perangkat lunak merupakan hasil dari tindakan yang dilakukan manajemen proyek serta hasil dari praktek rekayasa perangkat lunak yang baik juga. Pengukuran memungkinkan pembuat perangkat lunak dapat meningkatkan proses perangkat lunak, membantu perencanaan kualitas, kontrol suatu proyek perangkat lunak, serta memperkirakan kualitas produk perangkat lunak yang diproduksi. Pengukuran yang dapat dilakukan pada perangkat lunak secara langsung dan tidak langsung. Pengukuran langsung berkaitan dengan deretan kode, kecepatan

eksekusi, ukuran memori, dan cacat yang dilaporkan pada periode tertentu. Sedangkan pengukuran tidak langsung yaitu tentang fungsionalitas, kualitas, kompleksitas efisien, realibilitas, kemampuan pemeliharaan dan sebagainya5.

Menurut Dunn pada tahun 1990, sebuah *software* dikatakan berkualitas apabila memenuhi tiga ketentuan pokok :

1. Memenuhi kebutuhan pemakai yang berarti bahwa apabila *software* tidak

dapat memenuhi kebutuhan pengguna *software* tersebut, maka *software*

1. Roger S.Pressman, Ph. D, *Software* Engineering Seventh Edition, 2010, Hal : 485

tersebut dapat dikatakan tidak atau kurang memiliki kualitas.

1. Memenuhi standar pengembangan *software* yang berarti bahwa apabila cara pengembangan *software* tidak mengikuti metodologi standar, maka hampir dapat dipastikan bahwa kualitas yang baik akan sulit atau tidak tercapai.
2. Memenuhi sejumlah kriteria implisit yang berarti bahwa jika salah satu kriteria implisit tersebut tidak dapat dipenuhi, maka *software* tersebut tidak dapat dikatakan memiliki kualitas yang baik.

Menurut McCall dan kawan-kawan pada tahun 1997 bahwa telah menitikberatkan faktor-faktor yang mempengaruhi kualitas *software* tersebut menjadi tiga aspek penting, antara lain :

1. Sifat-sifat operasional dari *software* (*Product Operation*).
2. Kemampuan *software* dalam menjalani perubahan (*Product Revision*).
3. Daya adaptasi atau penyesuaian *software* terhadap lingkungan baru (*Product Transition*).

Sebagai penjelasan dari ketiga aspek penting di atas adalah sebagai berikut :

1. *Product operations* merupakan sifat-sifat operasional suatu *software* yang berkaitan dengan hal-hal yang harus diperhatikan oleh perancang dan pengembang yang secara teknis akan melakukan penciptaan sebuah aplikasi. Hal-hal yang diukur adalah hal yang berhubungan dengan teknis analisa, perancangan, dan konstruksi sebuah *software*.
2. *Product revisions* merupakan tahapan dimana setelah sebuah *software* berhasil dikembangkan dan diimplementasikan, maka akan terdapat berbagai hal yang perlu diperbaiki berdasarkan hasil uji coba maupun evaluasi. Sebuah *software* yang dirancang dan dikembangkan dengan baik maka akan dengan mudah dapat direvisi apabila diperlukan.
3. *Product transition* merupakan sebuah tahapan dimana setelah integritas *software* secara teknis telah diukur dengan menggunakan faktor *product operational* dan secara implementasi telah disesuaikan dengan faktor *product revision*, faktor terakhir yang harus diperhatikan adalah faktor transisi yaitu bagaimana *software* tersebut dapat dijalankan pada beberapa *platform* atau kerangka sistem yang beragam.
4. **Konsep *Object Oriented***
5. **Kelas dan Objek**

Kelas adalah *blueprint*/ *prototype* yang mendefinisikan semua *field* dan operasi yang dimiliki semua objek dengan jenis yang sama6. Objek adalah sebuah entitas yang memiliki sebuah *state* dan sekumpulan *method* (operasi) yang dapat dilakukan terhadap *state* tersebut. Sedangkan operasi-operasi yang dimiliki sebuah objek mengubah *state* objek dengan jenis yang sama.

Objek dapat digunakan untuk memodelkan objek-objek dalam dunia nyata.Sebagai contoh objek yang memodelkan sepeda dalam dunia nyata akan memiliki *variable* kecepatan x mph dan gear ke-x. *Variable* tersebut meng indikasikan *state* sebuah sepeda pada saat tertentu dan dikenal dengan istilah *instance variable*. Selain *variable*, objek sepeda memiliki *method* untuk

1. Dr. Bambang Hariyanto, Esensi-esensi bahasa pemrograman JAVA 2011, Hal : 237

mengerem dan mengubah *gear.* *Method-method* ini dikenal dengan istilah *instance method* 7.

1. **Pewarisan (*Inheritance*)**

Pewarisan adalah salah satu konsep pemrograman berorientasi objek dan mempunyai dampak langsung pada cara merancang dan menulis kelas. Pewarisan adalah proses penciptaan kelas baru dengan mewarisi karakteristik kelas yang telah ada, ditambah karakteristik iunik kelas baru itu. Dalam terminology Java, kelas yang diwarisi disebut subkelas, sedangkan kelas yang mewarisi disebut superkelas. Dengan demikian, subkelas adalah versi spesial dari superkelas8.

Ketika *programmer* membuat subkelas, ia dapat menambah *instance variable* dan *method* yang dimiliki subkelas tertentu. Ketika medefinisikan *method*- *method* subkelas, ada tiga kemungkinan yang dapat dilakukan7:

1. Meng-*override method* superkelas, yaitu mendeklarasikan *method* yang memilik *signature* (nama dan parameter) yang sama dengan *method* superkelas.
2. Mewariskan *method* superkelas tanpa meng-*override method* superkelas.
3. Membuat *method* baru yang tidak didefinisikan pada superkelas.
4. Nurmaya, Metode Pengukuran 2007, Hal : 4
5. Dr. Bambang Hariyanto, Esensi-esensi bahasa pemrograman JAVA 2011, Hal : 345

Sedangkan ketika mendefinisikan *variable-variable* subkelas*,* ada dua kemungkinan9:

1. Mewariskan *variable* superkelas. Seluruh *instance variable* yang dimiliki superkelas akan di wariskan secara otomatis.
2. Membuat *variable* baru yang hanya dimiliki oleh subkelas. *Variable* baru yang memiliki nama yang sama dengan *variable* superkelas akan membayangi *variable* superkelas.

Maksudnya adalah, *variable* superkelas tetap ada, namun tidak dapat diakses oleh subkelasnya. Setiap kali *method* subkelas mengakses *variable* tersebut, *variable* yang diakses adalah *variable* yang baru dideklarasikan tersebut.

Konsep pewarisan mendukung code reusability dan extensibility karena untuk menambahkan objek bertipe baru, programmer tidak perlu menulis ulang kode program yang sudah ada.

1. **Polimorfisme (*Polymorphism*)**

Polimorfisme adalah kemampuan untuk menggunakan sebuah symbol atau ekspresi yang sama namun menunjuk pada operasi yang berbeda. Polimorfisme juga berarti kemampuan variable atau fungsi untuk mengambil bentuk yang berbeda-beda pada saat *run time*.

Pada bahasa pemrograman OO(*Object Oriented*), polimorfisme diimplementasikan dengan teknik pemrograman *dynamic binding*.

1. Nurmaya, Metode Pengukuran 2007, Hal : 5

Menurut Peter Muller ada tiga polimorfisme, yaitu10:

1. Kemampuan sebuah *variable* untuk mengambil tipe tertentu tergantung pada nilainya pada waktu tertentu.
2. Penggunaan nama method yang sama namun menunjuk pada operasi yangberbeda(*overloading*).
3. Pengisian objek superkelas oleh objek subkelas-nya.
4. **Enkapsulasi**

Menurut (Nurmaya, 2007:5) *method-method* yang dimiliki sebuah

objek menyembunyikan data-data (*variable*) yang dimilikinya dari objek lain. Pembungkusan variable objek yang hanya dapat diakses melalui *method-method*-nya disebut enkapsulasi.

Enkapsulasi memberikan dua keuntungan utama bagi pengembang perangkat lunak:

1. *Modularity*

Kode program yang mendefinisikan suatu objek dapat dibuat dan dirawat secara independen tanpa tergantung kode program objek-objek lain.

1. *Information Hidding*

Sebuah objek memiliki antarmuka public yang dapat dipergunakan oleh objek lain sebagai alat komunikasi dengan objek tersebut. Objek tersebut juga dapat me-rawat informasi dan *method* yang bersifat private yang dapat diubah kapan saja tanpa mempengaruhi objek lain yang.

1. Peter Muller*, Introduction to Object-Oriented Programming Using C++. Globewide Network Academy,* 1997
2. ***Messages Passing***

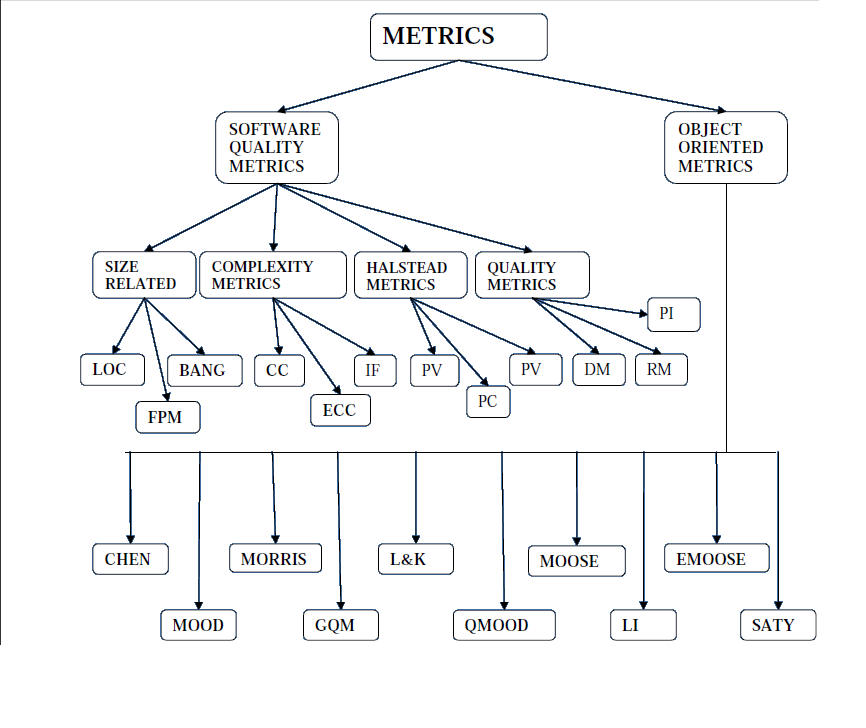
Objek yang bertindak sendirian jarang berguna. Sebagian besar objek memerlukan objek-objek lain untuk melakukan banyak hal. Objek-objek memerlukan mekanisme komunikasi untuk saling berinteraksi. Objek-objek perangkat lunak saling berinteraksi dan berkomunikasi lewat pesan. Saat berkomunikasi, objek mengirim pesan memberitahhu agar objek lain melakukan sesuatu.

Pengiriman pesan juga disertai informasi untuk memperjelas apa yang dikehendaki. Informasi yang dilewatkan dipesan ini adalah parameter pesan11.

1. ***Software Metrics***

“Untuk melakukan pengukuran terhadap perangkat lunak seorang rekayasawan perangkat lunak mengumpulkan ukuran-ukuran dan mengembangkan *metric*-*metric* yang menjadi indikator ukuran tersebut”. Menurut Roger S. Pressman (2012:723) Indikator adalah sebuah *metric* atau kombinasi beberapa *metric* yang member pemahaman terhadap *software process*, *software project*, atau *software product.* Sebuah indikator menyediakan gambaran yang memungkinkan sang manajer proyek atau rekayasawan perangkat lunak untuk menyesuaikan proses, proyek atau produk untuk mendapatkan hasil yang lebih baik.

1. Dr. Bambang Hariyanto, Esensi-esensi bahasa pemrograman JAVA 2011, Hal : 324



Gambar 2.5 Hirarki *Metrics* (IJCSMS 2012)

1. ***Object Oriented*(OO) *Metrics***

Pada periode ini, banyak perusahaan yang telah menggunakan teknologi OO pada lingkungan pengembangan perangkat lunak mereka. Metode analisa dan desain OO, bahasa pemrograman OO dan lingkungan pengembangan OO menjadi popular di organisasi perngkat lunak seluruh dunia baik yang kecil maupun besar. Penggunaan teknologi ini memberikan tantangan baru bagi perusahaan yang menggunakan *product metrics* sebagai alat untuk mengotrol, memonitor, memperbaiki cara pengembangan dan pemeliharaan perangkat lunak. Oleh karena itu, para peneliti berusaha untuk mengembangkan *metric set* yang dapat digunakan pada teknologi berparadigma OO. Salah satunya ialah *MOOD Metrics Suite*.

1. ***MOOD Metrics Suite***

*Metrics* *set* ini pertama kali di usulkan oleh Fernando Brito e Abreu dalam12. *Metrics set* ini dioperasikan terhadap sistem OO yang terdiri atas beberapa kelas*.* Metrikyang digunakan antara lain:

1. *Method Hiding Factor* (MHF)
2. *Attribute Hiding Factor* (AHF)
3. *Method Inheritance Factor* (MIF)
4. *Attribute Inheritance Factor* (AIF)
5. *Polymorphism Factor* (PF)
6. *Coupling Factor* (CF)

*Metrics set* di atas merujuk pada konsep OO, yaitu enkapsulasi (MHF dan AHF), *inheritance* (MIF dan AIF), *polimorfisme* (PF) dan *message-passing* (CF).

* + - 1. *Method Hiding Factor* (MHF)

*Method Hiding Factor* (MHF) adalah nilai perbandingan antara jumlah *invisibility* setiap *method* yang didefinisikan oleh setiap kelasdan jumlah method yang didefinisikan oleh sistem. *Invisibility* sebuah *method* adalah persentase jumlah kelasyang tidak dapat memanggil *method* secara langsung5. Kemudian MHF didefinisikan sebagai berikut12:

*Md* (*Ci* ) *Mv* (*Ci* ) *Mh* (*Ci* ) (2.1)

1. Fernando Brito e Abreu et.al (1994). *Object-Oriented Software Engineering: Measuring and Controlling the Development Process,* *4th International. Conference. on Software Quality, McLean, VA, USA*

Keterangan:

1. *Mv* (*Ci* ) adalah banyak *method* yang terlihat dalam kelas *Ci*
2. *Mh* (*Ci* ) adalah banyak *method* yang tersembunyi dalam kelas *Ci*.
   * + 1. *Attribute Hiding Factor* (AHF)­­

*Attribute Hiding Factor* (AHF)­­ adalah rasio dari atribut tersembunyi kepada anggota data total didefinisikan. Atribut tersembunyi adalah jumlah dari invisibilities dari semua disebabkan didefinisikan di semua kelas. Terlihatnya atribut adalah persentase jumlah kelas dari mana atribut tidak terlihat16. Mengenai MHF dan kriteria validasi AHF secara langsung diukur bahwa peningkatan nilai bersembunyi akan berarti kompleksitas kurang, pemahaman yang lebih dan tingkat yang lebih tinggi rawatan, sehingga menghasilkan kualitas yang baik dari perangkat lunak. Memahami perangkat lunak dapat digunakan dan *software* kurang kompleks memiliki efisiensi tinggi9. Kemudian AHF didefinisikan sebagai berikut16:

*Ad* (*Ci* ) *Av* (*Ci* ) *Ah* (*Ci* ) (2.2)

Keterangan:

1. *Av* (*Ci* ) adalah banyak *attribute* yang terlihat dalam kelas *Ci*
2. *Ah* (*Ci* ) adalah banyak *attribute* yang tersembunyi dalam kelas *Ci*.
   * + 1. *Method Inheritance Factor* (MIF) *Extended*

*Method Inheritance Factor* (MIF) *Extended* adalah rasio dari dua pengukuran. Pembilang merupakan jumlah jumlah metode turunan dari semua kelas dalam sistem dan penyebut merupakan jumlah jumlah metode yang tersedia yang mungkin bersifat lokal atau diwariskan untuk semua kelas dalam sistem.

Nilai MIF *Extended* diterima adalah 0 sampai 0.5. Hal ini sangat tergantung pada pola desain yang kita ikuti. Nilai-nilai tinggi dari MIF menunjukkan warisan berlebihan dan nilai-nilai yang rendah menunjukkan penggunaan berat menimpa atau kurangnya warisan13. Kemudian MIF *Extended* didefinisikan sebagai berikut13:

*Mex* (*Ci* ) *Md* (*Ci* ) *Mi* (*Ci* ) (2.3)

Keterangan:

1. *Mi* (*Ci* ) adalah hitungan *method* yang diwariskan.
2. *Md* (*Ci* ) adalah hitungan didefinisikan *method* yang bukan bersifat abstrak.
3. *Mex* (*Ci* ) adalah hitungan metode yang dapat dipanggil oleh kelas *Ci.*
4. TC adalah *total* kelas di sistem.
   * + 1. *Attribute Inheritance Factor* (AIF) *Extended*

*Attribute Inheritance Factor* (AIF) *Extended* adalah rasio dari dua pengukuran. Pembilang merupakan jumlah jumlah atribut yang diwariskan dari semua kelas dalam sistem dan penyebut merupakan jumlah atribut yang tersedia yang lokal atau diwariskan untuk semua kelas dalam sistem. Ini mengungkapkan tingkat penggunaan kembali dalam sistem.

Sebuah nilai awal dipertahankan untuk mengukur AIF yang kira-kira sekitar 50%. Nilai yang lebih tinggi dari AIF menunjukkan tingkat tinggi warisan sehingga menyebabkan kopling yang lebih besar dan mengurangi kemungkinan *reuse*13. Kemudian AIF *Extended* didefinisikan sebagai berikut17:

*Aex* (*Ci* ) *Ad* (*Ci* ) *Ai* (*Ci* ) (2.4)

Keterangan:

1. *Ai* (*Ci* ) adalah hitungan atribut yang diwariskan.
2. *Aex* (*Ci* ) adalah jumlah atribut yang dapat direferensikan oleh kelas *Ci.*
3. TC adalah *total* kelas di sistem.
   * + 1. *Polymorfime Factor* (PF)

Metrik ini mengukur konsep polimorfisme. Polimorfisme muncul karena adanya konsep pewarisan. Dengan adanya polimorfisme, kompleksitas sistem dapat dijaga agar tetap rendah, karena satu *message* yang sama dapat digunakan oleh berbagai kelas dalam hirarki yang sama. Namun, hal ini menyulitkan proses *debugging* dan *tracing*.

1. Shreya Gupta, Ratna Sanyal, “*Advanced Object Oriented Metrics for Process Measurement*”, *ICSEA The Sixth International Conference on Software Engineering Advances,* 2011*.*

Jadi nilai PF harus dijaga agar tidak terlalu rendah dan tidak terlalu tinggi9.

Kemudian PF didefinisikan sebagai berikut13:

(2.5)

Keterangan:

1. *Mo*(*Ci* ) adalah *override method* dalam kelas *Ci*
2. *Mn*(*Ci* ) adalah *method* baru dalam kelas *Ci*
3. *DC*(*Ci* ) adalah *Decendant counts* dalam kelas *Ci*
   * + 1. *Coupling Factor* (CF)

CF adalah ukuran penghubung antara kelas termasuk kopling yang timbul karena pewarisan. CF dihitung dengan mempertimbangkan semua kemungkinan set sepasang bijaksana kelas dan memvalidasi apakah kelas dalam pasangan yang terkait dengan pesan lewat. CF adalah ukuran langsung dari ukuran hubungan antara dua kelas. CF dievaluasi sebagai rasio kemungkinan jumlah kopling dalam perangkat lunak untuk jumlah sebenarnya kopling tidak imputable warisan14.

Sebagai tingkat tinggi hubungan interkelas akan memiliki nilai CF tinggi. Sebaliknya, CF tidak memberikan hasil yang valid dalam evaluasi kualitas.

Hal ini disebabkan oleh fakta bahwa tingginya nilai CF tidak menunjukkan kompleksitas tinggi atau rendah.

Karena itu layak untuk membangun sebuah sistem sederhana yang sangat digabungkan dan juga itu adalah mungkin untuk memiliki sistem yang kompleks dengan nilai diabaikan CF. Bahkan untuk nilai enkapsulasi mengukur CF tidak memberikan bantuan. Demikian pula untuk faktor kualitas pemahaman dan rawatan, kelas dengan nilai tinggi CF mungkin ada, tetapi tidak memiliki tingkat dampak pada saling pengertian dan rawatan. CF memainkan tidak berdampak pada usabilitas juga, tingkat rendah kopling mungkin memiliki tingkat tinggi warisan melalui *reuse*. Akibatnya, dapat disimpulkan bahwa CF bukan ukuran *valid* kualitas15.

1. ***Unified Modelling Language*(UML)**

*Unified Modelling Language* yang selanjutnya akan ditulis UML adalah tujuan standar bahasa pemodelan umum di bidang ilmu komputer dan rekayasa perangkat lunak. Standar ini dikelola dan diciptakan oleh kelompok manajemen objek16. Diagram UML merupakan dua pandangan yang berbeda dari sistem  
Model:

1. Statis (struktural) melihat: Menekankan struktur statis dari sistem dengan menggunakan objek, atribut, operasi dan hubungan. Pandangan struktural termasuk Diagram Kelasdan *Composite Structure* Diagram.

1. Dinamis (perilaku) melihat: Menekankan perilaku dinamis sistem dengan

objek. Pandangan ini termasuk *Sequence* Diagram, *Activity* Diagram dan

*State Machine* Diagram.

1. Rachel Harrison, Steve J. Counsell, and Reuben V. Nithi, *“An Evaluation of the MOOD Set of Object-Oriented Software Metrics”, IEEE Transactions on Software Engineering,* Vol. 24*,* No*.* 6*,* 1998*.*

Setiap sistem yang kompleks seharusnya bisa dipandang dari sudut yang berbeda-beda sehingga kita bias mendapatkan pemahaman secara menyeluruh. Untuk upaya tersebut UML menyediakan 9 jenis diagram yang dapat dikelompokkan berdasarkan sifatnya statis atau dinamis. Jenis diagram UML tersebut adalah17:

Diagram sering digunakan pada pemodelan sistem berorientasi objek.

1. Diagram Kelas

Diagram ini bersifat statis yang mampu memperlihatkan himpunan kelas-kelas, antarmuka, kolaborasi serta relasi.

1. Diagram Objek

Diagram ini bersifat statis yang mampu memperlihatkan objek-objek serta realasi antar objek. Diagram ini juga mampu memperlihatkan instansiasi statis dari segala sesuatu yang dijumpai pada kelasDiagram.

1. *Use-Case* Diagram

Bersifat statis yang mampu memperlihatkan himpunan *use case* dan aktor (suatu jenis khusus dari kelas).

Diagram ini terutama sangat penting untuk mengorganisasi dan memodelkan perilaku dari suatu sistem yang dibutuhkan serta diharapkan pengguna.

1. *Sequence* Diagram

Diagram interaksi yang menekankan pada pengiriman pesan dalam waktu tertentu dan bersifat dinamis.

1. Aman Kumar.S et.al (2012). *Metrics Identification for Measuring Object Oriented System Quality,* Vol 2*. International Journal of Soft Computing and Engineering (IJSCE).*
2. Siddique.Q et al (2010). *Unified Modeling to Object Oriented Software Development*, Vol 1 No.3 *International Journal of Innovation, Management and Technology*.
3. *Collaboration* Diagram

Diagram interaksi yang menekankan organisasi struktural dari objek yang menerima serta mengirim pesan dan bersifat dinamis.

1. *Statechart* Diagram

Diagram ini memperlihatkan *state*-*state* pada sistem, memuat *state*, transisi, *event* serta aktifitas. Diagram ini sangat penting untuk memperlihatkan sifat dinamis dari antarmuka, kelas, kolaborasi dan pemodelan sistem yang reaktif.

1. *Activity* Diagram

Diagram yang bertipe khusus dari diagram *state* yang memperlihatkan aliran dari suatu aktifitas ke aktifitas yang lain dalam suatu sistem. Diagram ini terutama penting dalam pemodelan fungsi-fungsi dalam suatu sistem dan member tekanan pada aliran kendali antar objek.

1. *Component* Diagram

Diagram komponen memperlihatkan organisasi serta kebergantungan sistem/ perangkat lunak pada komponen yang telah ada sebelumnya. Diagram ini berhubungan dengan diagram kelas dimana komponen secara tipikal dipetakan ke dalam satu atau lebih kelas-kelas, antarmuka serta kolaborasi.

1. *Deployment* Diagram

Bersifat statis. Diagram ini memperlihatkan konfigurasi saat aplikasi dijalankan. Diagram ini memuat simpul-simpul (node) beserta komponen yang ada didalamnya. *Deployment* diagram berhubungan erat dengan diagram komponen dimana *deployment* diagram memuat satu atau lebih komponen-komponen.

1. Nugroho.A (2005). Rational Roseuntuk Pemodelan Berorientasi Objek.