BAB II
TINJAUAN PUSTAKA

2.1 PENELITIAN TERDAHULU

Hasil kerja Nurmaya yang didasarkan pada penelitian oleh [ABR96], melakukan modifikasi terhadap Metrics Calculator Christariny dengan menambahkan Graphical User Interface (GUI) dan melakukan pengurutan berdasarkan hasil perhitungan MOOD terhadap sekumpulan program.

MOOD terdari dari sekumpulan *metric* untuk mengukur desain program berorientasi objek. Terdapat 6 aspek yang diukur oleh MOOD antara lain, Method Hiding Factor (MHF), Attribut Hiding Factor (AHF), Method Inheritance Factor (MIF), Attribute Inheritance Factor (AIF), Polymorphism Factor (POF) dan Coupling Factor (COF). MHF dan AHF menghitung persentase operasi/atribut yang diekspos dari suatu kelas ke kelas lainnya di dalam sistem. MIF dan AIF menghitung persentase operasi/atribut yang diwariskan oleh suatu kelas ke *subclass*-nya. POF menghitung persentase penggunaan jumlah operasi dari suatu kelas yang di-*override* pada *subclass*-nya. Dan COF menghitung seberapa besar ketergantungan suatu kelas terhadap kelas lainnya.

MOOD dimaksudkan untuk mengukur kualitas desain dari suatu sistem yang dikembangkan secara objek. Hal ini terlihat dari kumpulan *metric* yang membentuk MOOD, seluruhnya menghitung persentase dari pemanfaatan operasi/atribut dari suatu kelas terhadap kelas lainnya yang terdapat pada sistem. Pada sub bab berikutnya akan dibahas ke-enam *metric* yang membentuk MOOD.

2.1.1 METRICS FOR OBJECT ORIENTED DESIGN (MOOD)

MOOD merupakan salah satu *metric* untuk mengukur kualitas desain program, khususnya untuk program yang dikembangkan secara objek. Aspek desain yang diukur pada MOOD adalah *encapsulation* (MHF & AHF), *inheritance* (MIF & AIF), *polymorphism* (POF) dan *coupling* antara kelas (COF). Aspek-aspek ini menggambarkan hubungan antara kelas-kelas yang membentuk sistem yang lengkap, MOOD digunakan untuk menilai kualitas sistem bukan kelas yang berdiri sendiri.

Pada *encapsulation*, MOOD mengukur seberapa banyak atribut dan operasi yang diekspos keluar kelas. Terdapat dua sub-*metric* untuk mengukur kualitas *encapsula-*

tion yaitu Method Hiding Factor (MHF) untuk mengukur seberapa banyak operasi dari suatu kelas yang diekspos ke kelas lainnya di dalam sistem dan Attribute Hiding Factor (AHF) untuk mengukur seberapa banyak atribut dari suatu kelas diekspos ke kelas lainnya.

Method Hiding Factor & Attribute Hiding Factor

Pada desain program yang berorientasi objek, data dan operasi disatukan ke dalam satu entitas yang disebut dengan kelas yang kemudian akan di-wujudkan dalam bentuk objek. Data (atribut) menyimpan *state* dan informasi di dalam objek dan akses terhadap objek dilakukan secara terkendali yaitu hanya melalui *method* (interface) yang di-publish oleh objek yang bersangkutan.

Sub-metric Method Hiding Factor (MHF) dan Attribute Hiding Factor (AHF) mengukur seberapa banyak method dan atribut dari suatu kelas yang dapat diakses dari kelas lainnya di dalam sistem. MHF dan AHF diukur dengan rumus berikut ini:

$$MHF = \frac{\sum_{i=1}^{TC} \sum_{m=i}^{M_d(C_i)} (1 - V(M_{mi}))}{\sum_{i=1}^{TC} M_d(C_i)}$$

dan

$$AHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{A_d(C_i)} (1 - V(A_{mi}))}{\sum_{i=1}^{TC} A_d(C_i)}$$

Fungsi $V(M_{mi})$ dan $V(A_{mi})$ merupakan fungsi untuk memeriksa rasio visibilitas operasi atau atribut kelas C_i dari kelas C_j, yang didefinisikan sebagai berikut:

$$V(C_i.M_\alpha) = \frac{\sum_{j=1}^{TC} is\_visible(C_i.M_\alpha, C_j)}{TC - 1}$$

dan

$$V(C_i.A_\delta) = \frac{\sum_{j=1}^{TC} is\_visible(C_i.A_\delta, C_j)}{TC - 1}$$

Fungsi *is_visible* didefinisikan sebagai berikut:

$$is\_visible(C_i.M_\alpha, C_j) = \begin{cases} 1 & iif \begin{cases} j \neq i \\ C_j\, may\, call\, M_\alpha \end{cases} \\ 0 & otherwise \end{cases}$$

dan

$$is\_visible(C_i.A_\delta, C_j) = \begin{cases} 1 & iif \begin{cases} j \neq i \\ C_j\, may\, reference\, A_\delta \end{cases} \\ 0 & otherwise \end{cases}$$

$M_d(C_i)$ dan $A_d(C_i)$ adalah operasi (*method*) dan atribut yang didefinisikan pada kelas C_i, dan TC adalah total kelas di dalam sistem yang diukur.

Nilai MHF & AHF berkisar antara 0% sampai dengan 100%. Kelas yang seluruh method dan atribut-nya adalah private memiliki nilai MHF 100% dan AHF 100%, dan apabila seluruh method dan atribut-nya adalah public, makanya nilai MHF dan AHF-nya adalah 0%. MHF yang rendah menunjukkan banyak *method* dari suatu kelas yang di-publikasikan ke kelas lainnya, hal ini menunjukkan tingginya fungsionalitas dari kelas tersebut. Sebaliknya MHF yang tinggi menunjukkan bahwa hanya sedikit *method* dari satu kelas yang dapat diakses dari kelas lainnya.

MHF yang terlalu rendah menunjukkan kurangnya abstraksi pada desain program, yang menyebabkan terlalu banyaknya *interface*. Hal ini meningkatkan waktu untuk mempelajari dan menggunakan kelas di dalam program oleh karena jumlah *interface*-nya yang relatif banyak, dan juga hal ini dapat meningkatkan kemungkinan kesalahan di dalam program. Sedangkan MHF yang terlalu rendah menunjukkan bahwa terlalu sedikit *interface*, hal ini menyebabkan berkurangnya fungsionalitas yang dapat ditawarkan oleh kelas yang bersangkutan.

Method Inheritance Factor & Attribute Inheritance Factor

Method Inheritance Factor (MHF) & Attribute Inheritance Factor (AHF) mengukur kualitas desain berorientasi objek dari aspek pewarisan operasi dan atribut dari *superclass* ke *subclass*-nya. Nilai MHF & AHF dipengaruhi oleh seberapa banyak operasi dan atribut yang diwariskan dari *superclass* ke *subclass*.

Sebuah kelas yang mewarisi banyak atribut atau operasi dari *superclass* memiliki nilai AHF/MHF yang tinggi. Sebaliknya, kelas yang mendefinisi ulang atribut/operasi

yang diwarisi dari *superclass* ataupun menambahkan atribut atau operasi baru memiliki nilai AHF/MHF yang lebih rendah. Nilai MHF/AHF berkisar antara 0% sampai dengan 100%. MHF dan AHF diukur dengan rumus:

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

dan

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

$M_i(C_i)$ dan $A_i(C_i)$ merupakan operasi dan atribut yang diwariskan ke kelas C_i. $M_a(C_i)$ dan $A_a(C_i)$ merupakan seluruh operasi dan atribut yang terdapat pada kelas C_i.

Polymorphism Factor

Polymorphism Factor (POF) mengukur aspek *polymorphism* dari suatu sistem. *Polymorphism* berhubungan erat dengan *inheritance*. *Polymorphism* didapatkan dengan mendefinisi ulang (*overriding*) operasi yang diwarisi dari *superclass* atau dengan mendefinisikan beberapa operasi dengan nama yang sama tetapi dengan *signature* yang berbeda (*overloading*). *Metric* POF pada MOOD tidak mengikutsertakan *overloading* sebagai bagian dari pengukuran. POF diukur dengan rumus:

$$POF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]}$$

$M_n(C_i)$ merupakan operasi baru yang didefinisikan pada kelas C_i. $M_o(C_i)$ merupakan operasi yang di-*override* (didefinisi ulang) pada kelas C_i. Dan $DC(C_i)$ merupakan banyak kelas turunan dari kelas C_i.

Coupling Factor

Coupling Factor (COF) mengukur seberapa besar ketergantungan suatu kelas terhadap kelas lainnya di dalam sistem. Suatu kelas A dikatakan bergantung kepada kelas B apabila pada kelas A terdapat pemanggilan operasi atau pengaksesan atribut dari kelas B. *Coupling* meningkatkan kompleksitas sistem, idealnya suatu kelas menjaga komunikasi terhadap kelas lainnya seminimum mungkin. COF diukur dengan mem-

bandingkan banyak kelas yang memiliki hubungan coupling dan total kelas yang ada. Rumus pengukuran COF adalah sebagai berikut:

$$COF = \frac{\sum_{i=1}^{TC}\left[\sum_{j=1}^{TC} is\_client(C_i, C_j)\right]}{TC^2 - TC}$$

Fungsi $is\_client(C_i, C_j)$ definisikan dengan rumus berikut:

$$is\_client(C_c, C_s) = \begin{cases} 1 & iff\, C_c \Rightarrow C_s \wedge C_c \neq C_s \\ 0 & otherwise \end{cases}$$

Fungsi $is\_client(C_c, C_s)$ merupakan fungsi yang digunakan untuk memeriksa apakah kelas C_c merupakan klien dari kelas C_s. Klien disini diartikan sebagai kelas yang mengakses atribut atau operasi dari kelas lain.

Nilai 0% untuk COF menunjukkan bahwa tidak ada hubungan sama sekali antara satu kelas dengan kelas lainnya. Hal ini merupakan nilai ideal yang diharapkan dari hasil pengukuran COF. Dengan tidak ada keterkaitan antara satu kelas dengan kelas lainnya, maka perubahan terhadap kelas tersebut tidak akan berdampak pada kelas lainnya. Tetapi hal ini tidaklah mungkin ditemukan, sebab bagaimanapun, sistem *software* yang lengkap pasti membutuhkan akses dari satu kelas ke kelas lainnya.

2.1.2 CONFIDENCE INTERVAL GOOD OBJECT ORIENTED DESIGN (CIGOOD)

Dalam menilai kualitas desain, Nurmaya memanfaatkan perhitungan statistik yang terdapat pada [ABR96]. Teknik yang digunakan dalam memberikan penilaian ini bergantung pada sampel program yang dimasukkan sebagai data perhitungan statistik. Perhitungan statistik yang digunakan adalah *confidence intervals* dan *standard scores.*

Confidence intervals digunakan untuk mencari batas bawah dan batas atas nilai rata-rata *metric* dari seluruh sampel yang dimasukkan sebagai data. *Confidence intervals* digunakan sebagai tolak ukur baik atau tidaknya kualitas desain, pada [NUR07] disebut *Confidence Intervals Good* OOD (CIGOOD). Sampel yang nilai *metric*-nya berada di dalam rentang nilai CIGOOD dianggap merupakan sampel dengan program yang memenuhi persyaratan desain objek yang baik. Dan sampel yang nilai *metric*-nya berada diluar rentang nilai CIGOOD ini dianggap tidak memenuhi persyaratan desain objek yang baik.

Lebih jauh, *standard scores* digunakan untuk memberikan peringkat terhadap kualitas desain dari sampel program yang dinilai. Sampel program yang memiliki *standard scores* positif menunjukkan bahwa nilai *metric* dari program sampel tersebut berada di atas rata-rata, dan sebaliknya *standard scores* negatif menunjukkan nilai *metric* yang berada di bawah rata-rata. Dengan demikian, apabila sebuah sampel program A memiliki *standard scores* yang lebih tinggi dibandingkan sampel program B, dapat disimpulkan bahwa sampel program A secara kualitas lebih baik dibandingkan sampel program B.

Kelemahan dari penentuan peringkat kualitas desain dengan menggunakan *confidence interval* dan *standard scores* adalah nilai *confidence interval* yang ditentukan oleh sampel yang digunakan. Apabila sampel yang digunakan dalam perhitungan sebagian besar memiliki kualitas yang kurang baik, dapat menyebabkan nilai *confidence interval* yang terlalu lebar sehingga menurunkan standar kualitas desain yang dianggap baik.

2.2 LANDASAN TEORI

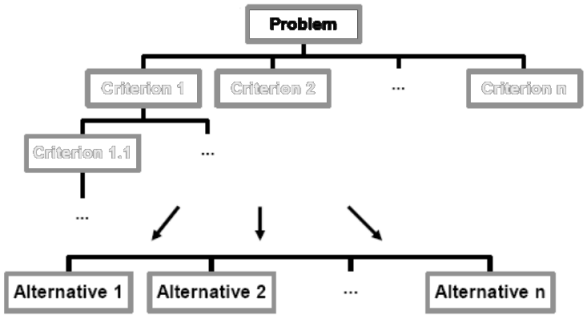
2.2.1 ANALYTIC HIERARCHY PROCESS

Analytic Hierarchy Process (AHP) merupakan suatu teknik untuk membantu pengambilan keputusan. Teknik ini dikembangkan oleh Dr. Thomas L. Saavy pada tahun 1970-an. Dengan memanfaatkan AHP, proses pengambilan keputusan yang didasarkan kepada beberapa kriteria yang harus dipenuhi dapat dilakukan secara transparan.

Langkah-langkah untuk pengambilan keputusan pada AHP adalah sebagai berikut [DRK98]:

1. Tentukan kriteria untuk menentukan keputusan yang diambil
2. Lakukan penilaian terhadap prioritas kepentingan kriteria ini secara berpasangan
3. Lakukan penilaian terhadap pilihan yang tersedia berdasarkan kriteria yang telah ditentukan. Penilaian pilihan ini juga dilakukan secara berpasangan.
4. Gabungankan hasil dari langkah 2 dan langkah 3 untuk mendapatkan nilai relatif diantara seluruh pilihan yang tersedia.

Secara visual, permasalahan yang akan diambil keputusannya akan dipecah ke dalam hirarki kriteria dan alternatif yang tersedia (Gambar 2.1).



Gambar 2.1: Membagi masalah ke dalam hirarki kriteria dan pilihan (sumber: [RAI])

Misalkan kita akan membeli mobil baru. Terdapat beberapa pilihan mobil yang yang akan dibeli, yaitu Civic, Saturn, Escort dan Clio. Pemilihan mobil yang akan dibeli ditentukan berdasarkan beberapa kriteria penilaian. Kita akan menentukan pilihan dengan menggunakan teknik AHP.

Langkah pertama dari AHP adalah menentukan kriteria penilaian yang akan digunakan untuk mengambil keputusan. Pada permasalahan pembelian mobil ini ada tiga kriteria yang dinilai, *reliability*, *style* dan *fuel economy*. Misalnya, *reliability* dianggap dua kali lebih penting dibandingkan dengan *style*, *style* tiga kali lebih penting dibandingkan *fuel economy* dan *reliability* empat kali lebih penting dibandingkan dengan *fuel economy*.

Langkah berikutnya adalah menentukan tingkat prioritas masing-masing kriteria dibandingkan dengan kriteria lainnya, dengan melakukan pairwise comparison. Kriteria yang sama pentingnya dengan kriteria pasangannya diberikan nilai 1, apabila suatu kriteria lebih penting dibandingkan kriteria pasangannya, maka kriteria tersebut akan diberikan nilai lebih besar 1 (sampai dengan 9) dan sebaliknya kriteria yang dipandang lebih tidak penting dibandingkan dengan kriteria pasangannya akan diberi nilai 1/n dari nilai prioritas kriteria pasangannya terhadap kriteria ini. Contoh perbandingan dari kriteria yang telah kita tentukan sebelumnya dapat dilihat pada Gambar 2.2.

Prioritas relatif antara masing-masing kriteria yang dikomparasikan dihitung dengan menggunakan eigenvector. Langkah-langkahnya adalah:

1. Hitung kuadrat dari matriks *pairwise comparison* kriteria

2. Hitung penjumlahan dari masing-masing baris dari matriks *pairwise comparison* kriteria dan kemudian lakukan normalisasi terhadap hasilnya

	STYLE	RELIABILITY	FUEL ECONOMY
STYLE	1/1	1/2	3/1
RELIABILITY	2/1	1/1	4/1
FUEL ECONOMY	1/3	1/4	1/1

Gambar 2.2: Matriks pairwise comparison kriteria (sumber: [RAI])

STYLE	0.3196	← THE SECOND MOST IMPORTANT CRITERION
RELIABILITY	0.5584	← THE MOST IMPORTANT CRITERION
FUEL ECONOMY	0.1220	← THE LEAST IMPORTANT CRITERION

Gambar 2.3: Eigenvector untuk kriteria (sumber: [RAI])

3. Ulangi langkah 1 sampai selisih dari hasil langkah dua dari dua iterasi (iterasi sekarang dan iterasi sebelumnya) mendekati nilai yang telah ditentukan (dalam kasus ini kita berhenti apabila selisihnya mendekati nol)

Hasil normalisasi penjumlahan baris pada iterasi terakhir langkah di atas merupakan eigenvector yang dihasilkan. Eigenvector untuk matriks *pairwise comparison* kriteria pada contoh yang kita gunakan dapat dilihat pada Gambar 2.3.

Langkah berikutnya pada AHP adalah menentukan beban prioritas masing-masing alternatif pilihan berdasarkan kriteria. Proses penentuan prioritas untuk alternatif sama dengan proses penentuan prioritas untuk kriteria. Masing-masing alternatif akan dibandingkan secara *pairwise comparison* dengan alternatif lainnya. Gambar 2.4 dan Gambar 2.5 menampilkan matriks *pairwise comparison* dari alternatif pilihan yang tersedia.

Berikutnya kita akan menghitung eigenvector untuk menentukan rangking dari alternatif yang tersedia. Gambar 2.6 menampilkan eigenvector untuk masing-masing alternatif. Dari gambar tersebut, dapat dilihat bahwa Clio memiliki nilai eigenvector

STYLE	CIVIC	SATURN	ESCORT	CLIO
CIVIC	1/1	1/4	4/1	1/6
SATURN	4/1	1/1	4/1	1/4
ESCORT	1/4	1/4	1/1	1/5
CLIO	6/1	4/1	5/1	1/1

Gambar 2.4: Matriks pairwise comparison alternatif berdasarkan Style (sumber: [RAI])

RELIABILITY	CIVIC	SATURN	ESCORT	CLIO
CIVIC	1/1	2/1	5/1	1/1
SATURN	1/2	1/1	3/1	2/1
ESCORT	1/5	1/3	1/1	1/4
CLIO	1/1	1/2	4/1	1/1

Gambar 2.5: Matriks pairwise comparison alternatif berdasarkan Reliability (sumber: [RAI])

yang tertinggi untuk kriteria *Style* dan Civic untuk kriteria *Reliability*. Hal ini menunjukkan bahwa kedua alternatif tersebut menempati ranking teratas pada masing-masing kriteria tersebut.

Ranking alternatif pilihan untuk *fuel economy* dilakukan dengan cara yang agak berbeda. Oleh karena *fuel economy* bersifat kuantitatif (merupakan rasio antara jumlah bahan bakar dengan jarak tempuh), kita dapat menghitung rankingnya tanpa harus menggunakan eigenvector, cukup hanya dengan menghitung rasio jumlah bahan bakar dengan jarak tempuh. Ranking alternatif pilihan untuk kriteria fuel economy dapat dilihat pada Gambar 2.7.

Hasil yang kita dapatkan setelah melewati proses pemberian prioritas dan ranking untuk masing-masing kriteria dan alternatif pilihan dapat dilihat pada Gambar 2.8.

Langkah terakhir adalah menghitung ranking dari masing-masing alternatif berdasarkan prioritas kriterianya. Cara perhitungan ranking adalah dengan melakukan perkalian matriks antara matriks ranking alternatif dengan matriks prioritas kriteria. Hasil akhir dari perhitungan ini dapat dilihat pada Gambar 2.9.

RANKING		STYLE
3	CIVIC	.1160
2	SATURN	.2470
4	ESCORT	.0600
1	CLIO	.5770

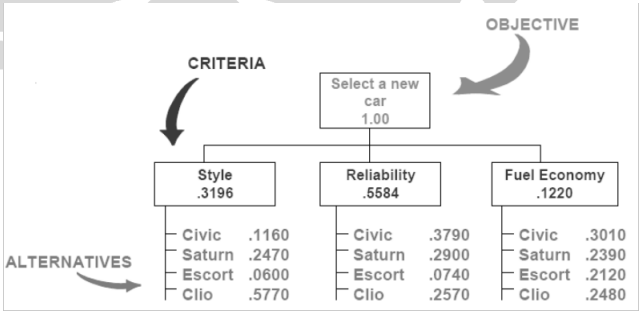
RANKING		RELIABILITY
1	CIVIC	.3790
2	SATURN	.2900
4	ESCORT	.0740
3	CLIO	.2570

Gambar 2.6: Eigenvector untuk masing-masing alternatif berdasarkan Style & Reliability (sumber: [RAI])

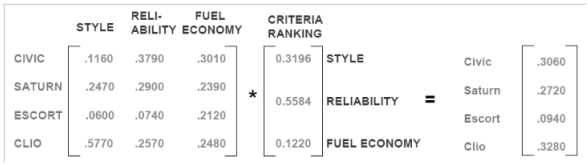
	FUEL ECONOMY (MILES/GALLON)		
CIVIC	34	34 / 113 =	.3010
SATURN	27	27 / 113 =	.2390
ESCORT	24	24 / 113 =	.2120
CLIO	28	28 / 113 =	.2480
	113		1.0000

Gambar 2.7: Ranking alternatif pilihan untuk Fuel Economy (sumber: [RAI])

Dari hasil perhitungan tersebut, didapatkan Clio sebagai pilihan yang paling sesuai berdasarkan kriteria yang telah ditetapkan.



Gambar 2.8: Prioritas kriteria dan ranking untuk masing-masing alternatif (sumber: [RAI])



Gambar 2.9: Ranking alternatif (sumber: [RAI])

Tabel 2.1: Random Consistency Index (RI); n – banyak komparasi (sumber: [TLS06])

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
RI	0	0	0.58	0.9	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.54	1.56	1.57	1.58

2.2.2 CONSISTENCY RATIO

Consistency Ratio (CR) berperan penting dalam penentuan nilai matriks *pairwise comparison* AHP. CR digunakan untuk menentukan apakah nilai yang dimasukkan pada matriks *pairwise comparison* konsisten atau tidak.

Prof. Saaty, pengembang AHP membuktikan bahwa matriks *pairwise comparison* yang konsisten nilai Pincipal Eigen Value-nya sama dengan banyak komparasi. Kemudian, Prof. Saaty menghasilkan sebuah rumus untuk menghitung derajat konsistensi yang disebut dengan *Consistency Index* (CI). Kemudian dia menghitung consistency index dari 500 matriks acak dan kemudian mencari rata-ratanya untuk menghasilkan *Random Consistency Index* (RI) (2.1). Rumus perihitungan CI adalah sebagai berikut:

$$CI = \frac{\lambda_{max} - n}{n - 1}$$

dimana CI adalah *Consistency Index*, λ_{max} adalah *Principal Eigen Value* dan n adalah jumlah komprasi.

Consistency Ratio dihitung dengan membandingkan (membagi) *consistency index* dengan *random consistency index*. Jika hasil dari *consistency ratio* lebih kecil atau sama dengan 10%, maka inkonsistensi dalam penetapan nilai pada matriks *pairwise comparison* masih dapat ditoleransi. Apabila *consistency ratio* lebih besar dari 10%, maka nilai pada matriks *comparison* tidak konsisten dan harus diulangi lagi proses penetapan nilainya.

STYLE	RELIABILITY	FUEL ECONOMY
10/3	7/4	8/1

Gambar 2.10: Jumlah Kolom dari matriks pairwise comparison kriteria

Berikut ini, akan diperiksa apakah kriteria pada contoh sebelumnya konsisten atau tidak. Langkah pertama adalah menghitung Principal Eigen Value dari matriks *pairwise comparison* kriteria pada contoh sebelumnya (Gambar 2.2). Principel Eigen Value dihitung dengan menjumlahkan hasil perkalian antara Eigenvector kriteria (Gambar 2.3) dengan jumlah dari masing-masing kolom kriteria (Gambar 2.10).

Perhitungan Principel Eigen Value dari contoh di atas adalah 10/3 * (0.3196) + 7/4 * (0.5584) + 8/1 * (0.1220) = 3.0185. Selanjutnya, hitung Consistency Index, (3.0185 – 3)/(3 – 1) = 0.0092. Dan terakhir hitung CR, 0.0092/3.0185 = 0.30%. Nilai CI sebesar 0.30% lebih kecil dari 10%, hal ini menunjukkan bahwa nilai matriks pairwise comparison kriteria (Gambar 2.2) pada contoh di atas konsisten.