BAB 2
LANDASAN TEORI

2.1 Definisi Perangkat Lunak

Menurut definisi IEEE *Institute of Electrical Enggineers*, Perangkat lunak adalah Program komputer, prosedur, dan dokumentasi yang menyertai serta data yang digunakan untuk mengoprasikan sistem komputer. Definisi IEEE tersebut hampir identik dengan definisi ISO [1]. Perangkat lunak terdiri dari 4 komponen, yaitu :

1. Program Komputer
2. Prosedur
3. Dokumentasi
4. Data yang diperlukan agar perangkat lunak beroperasi

2.2 Definisi Kualitas Perangkat Lunak

Menurut steve McConnell's, Code Complete membagi perangkat lunak kedalam dua hal yaitu *internal quality* dan *eksternal quality characteristics*. Karakteristik kualitas eksternal merupakan bagian dari suatu produk yang berhubungan dengan para pemakainya, sedangkan kualitas internal tidak secara langsung berhubungan dengan pemakai. *Software quality* didefinisikan sebagai kesesuaian yang diharapkan pada semua *software* yang dibangun dalam fungsi *software* yang diutamakan dan untuk kerja *software*, standar pembangunan yang terdokumentasi dan karakteristik yang ditunjukan *software* [1]. Definisi ini menekankan pada 3 hal yaitu :

1. Kebutuhan *software* adalah pondasi ukuran kebutuhan *software* adalah pondasi ukuran kualitas *software,* jika *software* tidak sesuai dengan kebutuhan yang ditentukan maka kualitasnya pun kurang.
2. Jika menggunakan suatu standar untuk pembangunan *software* maka jika *software* tidak memenuhi standar tersebut maka dianggap kurang berkualitas.

3. Seringkali ada kualitas yang secara langsung diutarakan (tersirat) seperti kemudahan penggunaan dan pemeliharaan yang baik. Kualitas *software* dipertanyakan jika tidak memenuhi kebutuhan ini.

Hal lain yang perlu diperhatikan dalam kualitas perangkat lunak adalah *quality assurarance* (QA) yang merupakan *acctivity of providing revidence*. Selain itu harus adanya *software quality management* (SQM). Tujuan dari SQM adalah untuk mengembangkan suatu pemahaman kuantitatif dari kualitas proyek produk perangkat lunak dan mencapai tujuan spesifik kualitasnya. Menurut IEEE, kualitas perangkat lunak didefinisikan sebagai derajat dari sebuah sistem, komponen atau proses yang memenuhi kebutuhan-kebutuhan yang dispesifikasikan dan kebutuhan pengguna atau harapan pengguna. Menurut Pressman, kualitas perangkat lunak adalah kesesuaian antara kebutuhan fungsional dan kebutuhan performansi yang terdefinisi, standar pengembangan yang terdokumentasi secara ekplisit dan karakteristik-karakteristik yang tersirat yang diharapkan dari perangkat lunak yang dikembangkan secara profesional [5].

2.3 Definisi *Website*

Definisi *website* adalah sekumpulan halaman yang terdiri dari beberapa halaman yang berisi informasi dalam bentuk digital baik itu teks, gambar, animasi yang disediakan melalui jalur internet sehingga dapat diakses dari seluruh dunia. Pada dasarnya *website* dibagi menjadi dua yaitu:

1. *Website* statis

   Merupakan *website* yang halamannya tidak berubah, biasanya untuk melakukan perubahan dilakukan secara manual dengan mengubah *code*. *Website* statis ini informasinya satu arah, yakni hanya berasal dari pemilik *software* nya saja dan hanya bisa di *update* oleh pemilik *software* nya saja.

2. Website dinamis

   Merupakan web halaman yang selalu di *update*, biasanya terdapat halaman *backend* (halaman administator) yang digunakan untuk mengubah konten. Web dinamis membutuhkan *database* untuk menyimpan data. Web dinamis mempunyai informasi dua arah, yakni berasal dari pengguna dan pemiliknya.

2.4 Pemrograman Berorientasi Objek

Objek adalah kesatuan entitas yang memiliki sifat dan tingkah laku. Dalam kehidupan sehari-hari, objek adalah benda, baik benda berwujud nyata seperti manusia, hewan, mobil, komputer, handphone, pena, ataupun benda yang tidak nyata atau konsep, seperti halnya tabungan bank, sistem antrian, sistem internet banking, dan sebagainya. Jadi pengertian OOP adalah konsep yang membagi program menjadi objek-objek yang saling berinteraksi satu sama lain. Objek adalah benda, baik benda yang berwujud nyata maupun benda yang tidak nyata (konsep). Jika menggunakan OOP maka akan ada enam keuntungan yang dapat diperoleh, yaitu [3]:

1. Alami (*Natural*).
2. Dapat diandalkan (*Reliable*).
3. Dapat digunakan kembali (*Reusable*).
4. Mudah untuk dalam perawatan (*Maintainable*).
5. Dapat diperluas (Extendable).
6. Efisiensi waktu.

Berikut ini beberapa bahasa pemrograman yang sudah menggunakan konsep OOP, adalah :

1. *C++*.
2. *C#*.
3. *Visual Basic*.
4. *Java*.
5. *PHP*

2.4.1 Analisis dan Desain Berorientasi Objek (Object-Oriented Analysis and Design Process)

Pemrograman berorientasi objek bekerja dengan baik ketika dibarengi dengan *object-oriented analysis and design process (OOAD).* Jika membuat program berorientasi objek tanpa OOAD, ibarat membangun rumah tanpa terlebih dahulu menganalisa apa saja yang dibutuhkan oleh rumah itu, tanpa perencanaan tanpa *blueprint*, tanpa menganalisis ruangan apa saja yang diperlukan, berapa besar rumah yang akan dibangun dan sebagainya.

2.4.2 Orientasi Objek (Object)

Orientasi objek merupakan teknik dalam menyelesaikan masalah yang kerap muncul dalam pengembangan perangkat lunak. Teknik ini merupakan titik kulminasi dalam menemukan cara yang efektif dalam membangun sistem dan menjadi metode yang paling banyak dipakai oleh para pengembang perangkat lunak saat ini. Orientasi objek merupakan teknik pemodelan sistem riil yang berbasis objek. Inti dari konsep ini adalah objek yang merupakan model dari sistem nyata.

Objek adalah entitas yang memiliki atribut, karakter dan kadangkala disertai kondisi. Objek merepresentasikan sesuatu sistem nyata seperti siswa, sistem kontrol permukaan sayap pesawat, sensor atau mesin. Objek juga merepresentasikan sesuatu dalam bentuk konsep seperti nasabah bank, merek dagang, pernikahan atau sekedar *listing*. Bahkan bisa juga mengatakan visualisasi seperti, bentuk huruf, histogram, poligon, garis atau lingkaran. Semuanya memiliki fitur atribut (untuk data), *behavior* (*operation* atau *method*), keadaan (memori), identitas dan tanggung jawab. Proses menjabarkan sistem nyata menjadi objek dinamakan abstraksi *(abstraction)*. Abstraksi mengeliminir aspek yang tidak perlu dalam suatu objek.

2.4.3 kelas *(Class)*

Kelas adalah penggambaran satu set objek yang memiliki atribut dan *behaviour* yang sama. Kelas mirip tipe data pada pemrograman non objek, tapi lebih komprehensif karena terdapat struktur sekaligus karakteristiknya. Programmer dapat membentuk kelas baru yang lebih spesifik dari kelas *general*-nya. Kelas dan objek merupakan jantung dari pemrograman berorientasi objek.Untuk menghasilkan program jenis ini sangat penting untuk selalu berfikir dalam bentuk objek [15].

2.4.4 Pembungkusan (Encapsulation)

Pembungkusan sebagai penggabungan potongan-potongan informasi dan perilaku-perilaku spesifik yang bekerja pada informasi tersebut, kemudian mengemasnya menjadi apa yang disebut sebagai objek. Dalam perbankan dikenal objek rekening yang memiliki perilaku-perilaku misalnya buka, tutup, penarikan, penyimpanan, ubah nama, ubah alamat, dan sebagainya. Akibatnya, perubahan-perubahan pada sistem perbankan yang berkaitan dengan rekening-rekening dapat secara sederhana diimplementasikan satu kali saja pada objek rekening. Keuntungan lainnya adalah membatasi efek-efek perubahan pada sistem. Misalnya, saat manajemen bank menentukan jika seseorang memiliki rekening pinjaman di bank yang bersangkutan, rekening pinjaman itu harus dapat juga digunakan sebagai sarana bagi penarikan rekening [3].

2.4.5 Pewarisan *(Inheritance)* dan Generalisasi/Spesialisasi

Konsep dimana metode dan atau atribut yang ditentukan di dalam sebuah objek kelas dapat diwariskan atau digunakan lagi atau digunakan lagi oleh objek kelas lainnya. Sedangkan generalisasi/spesialisasi merupakan teknik dimana atribut dan perilaku yang umum pada beberapa tipe kelas objek, dikelompokkan (atau diabstraksi) ke dalam kelasnya sendiri (dinamakan *supertype*). Atribut dan metode kelas objek *supertype* kemudian diwariskan oleh kelas objek tersebut (dinamakan *subtype*) [3].

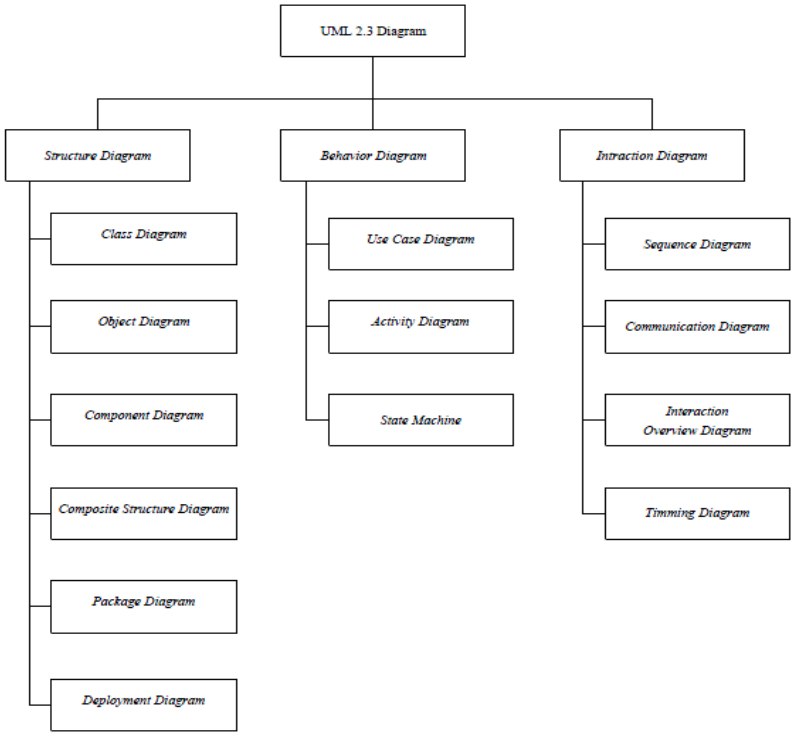
2.5 UML (Unified Modeling Language)

UML adalah keluarga notasi grafis yang didukung oleh meta-model tunggal, yang membantu pendeskripsian dan desain sistem perangkat lunak, khususnya sistem yang dibangun menggunakan pemrograman berorientasi objek. Definisi ini merupakan definisi sederhana. Pada kenyataannya, pendapat orang-orang tentang UML berbeda satu sama lain. Hal ini dikarenakan oleh sejarahnya sendiri dan oleh perbedaan persepsi tentang apa yang membuat sebuah proses rancang bangun perangkat lunak efektif.

UML merupakan standar yang relatif terbuka yang dikontrol oleh O*bject Management Group (OMG),* sebuah konsorsium terbuka yang terdiri dari banyak perusahaan. OMG dibentuk untuk membuat standar-standar yang mendukung interoperabilitas, khususnya interoperabilitas sistem berorientasi objek. OMG mungkin lebih dikenal dengan standar-standar *COBRA (Common Object Request Broker Architecture).*

UML lahir dari penggabungan banyak bahasa pemodelan grafis berorientasi objek yang berkembang pesat pada akhir 1980-an dan awal 1990-an. Sejak kehadirannya pada tahun 1997, *UML* menggantikan menara Babel yang telah menjadi sejarah. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM.

Bagian-bagian utama dari *UML* adalah *view*, *diagram*, *model element*, dan *general mechanism* [3]. Diagram berbentuk grafik yang menunjukkan simbol elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu dari sistem. Sebuah diagram merupakan bagian dari suatu view tertentu dan ketika digambarkan biasanya.



Gambar 2.1 Diagram UML [4]

Berikut ini penjelasan singkat dari pembagian kategori tersebut.

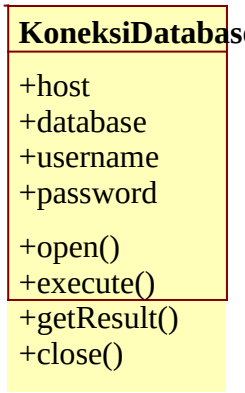
1. *Structure diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan.
2. *Behavior diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkain perubahan yang terjadi pada sebuah sistem.
3. *Interaction diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antarsubsistem pada suatu sistem.

2.5.1 Diagram Kelas *(Class Diagram)*

Diagram kelas menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi.

1. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas.
2. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

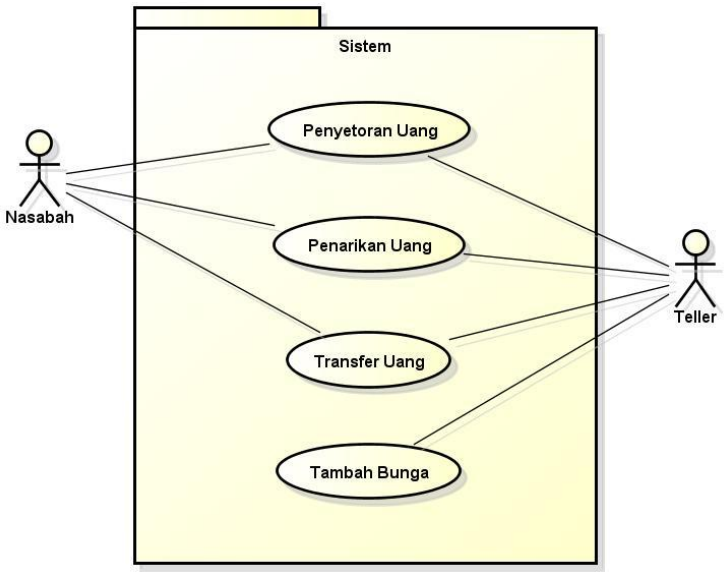
Diagram kelas dibuat agar pembuat program atau *programmer* membuat kelas-kelas sesuai rancangan di dalam diagram kelas agar antara dokumentasi perancangan dan perangkat lunak sinkron. Berikut adalah contoh dari diagram kelas.



Gambar 2.2 Contoh *Class Diagram* [4]

2.5.2 Diagram *Use Case*

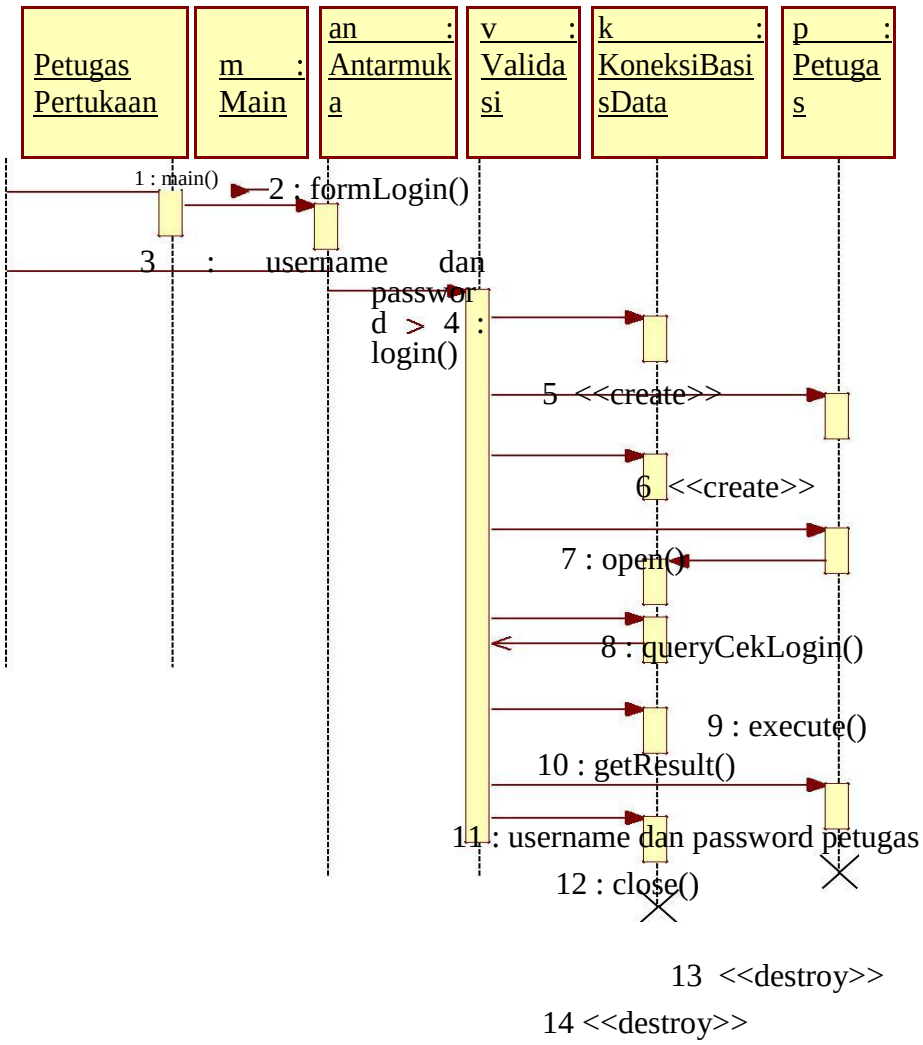
Diagram *use case* merupakan pemodelan untuk kelakuan *(behaviour)* sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu. Syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu pendefinisian apa yang disebut aktor dan *use case*.



Gambar 2.3 Contoh *Use Case Diagram* [4]

2.5.3 Diagram Sekuen (*Sequence* Diagram)

Diagram sekuen menggambarkan kelakuan pada objek *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambar diagram sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu. Membuat diagram sekuen juga dibutuhkan untuk melihat skenario yang ada pada *use case*. Banyaknya diagram sekuen yang harus digambar adalah minimal sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram sekuen sehingga semakin banyak *use case* yang didefinisikan maka diagram sekuen yang harus dibuat juga semakin banyak.



Gambar 2.4 Contoh *Sequence Diagram* [4]

2.6 Orientasi Objek Metriks

Menurut Chidamber dan Kemerer Metric untuk mengukur perangkat lunak berbasis objek oriented dapat menggunakan enam buah metriks [7], yaitu :

1. *Weight Methods per Class* (WMC)

   Pengkuran metriks WMC adalah menghitung jumlah method yang diimplementasikan dalam suatu *class*. Untuk mengetahuinya maka dihitung seluruh *method* lokal pada sebuah perangkat lunak [7].

   Rumus : m1+m2+m3+.....+mn

   m adalah sebuah method lokal yang dijumlahkan pada setiap *class*.

2. *Depth of Inheritance Tree* (DIT)

   DIT merupakan penghitungan jalur *inhereitance* maksimum dari sebuah class. Kedalaman sebuah class dalam hirarki pewarisan diukur oleh DIT.

   Rumus : Apabila ada pewarisan diberikan nilai 1 dan apabila tidak ada diberikan nilai 0 [4].

3. *Number of Children* (NOC)

   Pengukuran NOC adalah menghitung jumlah *children* atau anak *class* [7].

   Rumus : Apabila ada anak diberi nilai dan apabila tidak tidak diberi nilai.

4. *Coupling Between Objects* (CBO)

   Coupling adalah ketergantungan modul satu dengan modul lainnya, *coupling* yang baik adalah *coupling* yang rendah dimana modul satu dengan modul lainnya tidak saling ketergantungan. Suatu fungsi yang berpasangan dihitung sebagai *coupling*. Semakin kecil jumlah CBO maka semakin baik kualitas class tersebut.

   Rumus : Menghitung jumlah dari *method* lokal ditambah *method* eksternal [7].

5. *Respon for a Class* (RFC)

   RFC adalah *method* yangg diimplementasikan dalam *class* ditambahkan *class* yang diakses objek dari jenis *class* tersebut Semakin tinggi nilai RFC maka semakin besar kompleksitas *class* tersebut dan semakin membutuhkan waktu yang lama dalam pemeliharaannya. Pengukuran RFC yaitu menghitung jumlah *method* lokal yang diimplementasikan ditambah *method* yang dipanggil dalam objek [7].

   Rumus : jumlah method lokal + jumlah method eksternal.

6. *Lack OF Cohesion in Method* (LCOM)

   Cohesi yang baik adalah *cohesi* yang tinggi, semakin tinggi *cohesi* semakin baik perancangannya. *Cohesi* yang mempunyai nilai tumbukan antara 0-1 adalah nilai *cohesi* yang baik dan sedang. Sedangkan nilai *cohesi* yang rendah mempunyai nilai 2 yang menandakan *class* tersebut harus dipecah menjadi dua. *Cohesion* sebuah *class* adalah karakteristik seberapa dekat method-method lokal terhubung dengan variabel lokal dalam sebuah *class*. Pengukuran LCOM dilakukan dengan menghitung jumlah koneksi pada method. Rumus Perhitungannya adalah:

   LCOM*= (m – sum (mA) / a) / (m-1).

   Dimana:

   m adalah jumlah *method* dalam *class*.

   a adalah jumlah *variable* dalam *class*.

   mA adalah *method* yang mengakses *variable*.

   Sum (mA) adalah jumlah *variable* yang dipanggil.

Faktor penilaian yang mempengaruhi ada pada tabel dibawah ini:

2.1 Tabel Faktor Penilaian Akhir Faktor *Internal*

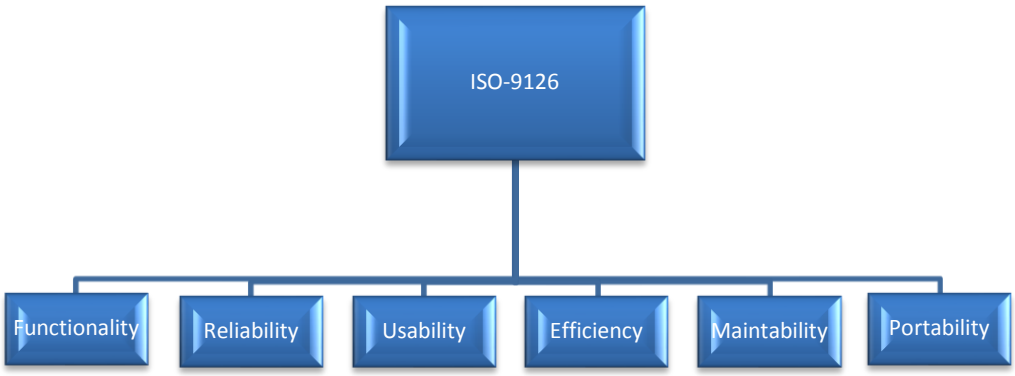
Faktor Penilaian	Metriks
Efficiency	$= \text{LCOM} + \text{CBO} + \text{DIT} + \text{NOC}$
Understandability	$= \text{WMC} + \text{RFC} + \text{DIT}$
Reusability	$= \text{WMC} + \text{LCOM} + \text{CBO} + \text{DIT} + \text{NOC}$
Maintanability/testability	$= \text{WMC} + \text{RFC} + \text{DIT} + \text{NOC}$

2.7 Model Metriks Kualitas Perangkat Lunak

Dalam ilmu rekayasa perangkat lunak terdapat beberapa model pennilaian kualitas perangkat lunak. Pada setiap model terdapat faktor penilaian yang menjadi poin utama dan subfaktor yang mendukung penilaian atas faktor tersebut. Model ISO-9126 dikenalkan pada tahun 1991 sebagai standarisasi kualitas produk perangkat lunak. Standarisasi ini dibuat karena banyaknya model kualitas yang ditawarkan sebagai penilaian kualitas perangkat lunak. Dalam model ISO 9126 terdiri dari beberapa bagian model kualitas untuk kualitas sebuah produk perangkat lunak, yaitu:

1. *Internal quality*
2. *External quality*
3. *Quality in use*

2.7.1 Standar ISO 9126



Gambar 2.5 Faktor ISO 9126 [8]

Standar ISO 9126 Adalah standar internasional yang diterbitkan ISO untuk evaluasi kualitas perangkat lunak. Ada 6 ukuran kualitas yang ditetapkan oleh ISO 9126, yaitu *Functionality* (Fungsionalitas), *Reliability* (Kehandalan), *Usability* (Kebergunaan), *Efficiency* (Efisiensi), *Maintanability* (Pemeliharaan), dan *Portability* (Portabilitas). Standar ISO 9126 pertama kali dikenalkan pada tahun 1991. Standar ISO 9126 mengidentifikasi 6 karakteristik kualitas perangkat lunak utama yaitu:

a. *Functionality*

   Kemampuan menutupi fungsi produk perangkat lunak yang menyediakan kepuasan kebutuhan user.

b. *Reliability*

   Kemampuan perangkat lunak untuk perawatan dengan level performansi.

c. *Usability*

   Kemampuan yang berhubungan dengan kebergunaan perangkat lunak.

d. *Efficiency*

   adalah faktor kemampuan yang berhubungan dengan pengaksesan sebuah peerangkat lunak yang sedang berjalan.

e. *Maintanability*

adalah kemampuan yang dibutuhkan untuk membuat perubahan serta pemeliharaan pada sebuah perangkat lunak.

f. *Portability*

Kemampuan yang berhubungan dengan kemampuan perangkat lunak yang dikirim kelingkungan yang berbeda.

2.7.2 Model *Quality In Use*



Gambar 2.6 Faktor Quality in Use ISO 9126 [8]

Model Quality In Use mempunyai 4 Faktor yaitu *Effectiveness, Productivity, Safety dan Satisfaction*. Masing masing faktor juga mempunyai beberapa metriks. Berikut penjelasannya :

a. *Effectiveness*

Effectiveness (Efektivitas) berhubungan dengan seberapa efektif suatu perangkat lunak terhadap penggunanya.

Tabel 2.2 *Effectiveness*

Karakteristik	Metriks	Rumus
Effectiveness	*Task completion*	X = A/B A = jumlah tugas diselesaikan B = jumlah tugas yang dicoba
	Error frequency	X = A/T A = jumlah kesalahan yang dibuat oleh pengguna T = jumlah tugas dicoba

b. Productivity

Productivity (Produktivitas) berhubungan dengan seberapa produktif kah perangkat lunak terhadap penggunanya

Tabel 2.3 *Productivity*

Karakteristik	Metriks	Rumus
Productivity	*Task time*	X = Ta Ta = Waktu mencoba
	Task efficiency	X = M1 / T M1 = efektivitas tugas T = Waktu
	Economic productivity	X = M1 / C M1 = efektivitas tugas C = total biaya tugas
	Productive proportion	X = Ta / Tb Ta = waktu produktif = waktu tugas – waktu bantu - waktu error - waktu pencarian Tb = waktu tugas

Karakteristik	Metriks	Rumus
	Relative user efficiency	*Relative user efficiency* X = A / B A = efisiensi tugas pengguna biasa ini B = efisiensi tugas ahli pengguna

c. *Safety*

Safety (Keamanan) berhubungan dengan seberapa aman perangkat lunak yang dianalisis terhadap penggunanya.

Tabel 2.4 *Safety*

Karakteristik	Metriks	Rumus
Safety	*Safety of people affected by use of the system*	X = 1-A / B A = jumlah orang menaruh di hazard B = Jumlah orang yang berpotensi terkena dampak oleh sistem
	Economic damage	X = 1-A / B A = jumlah kejadian kerusakan ekonomi B = Jumlah situasi penggunaan
	Software damage	X = 1-A / B A = jumlah kejadian perangkat lunak corrupt B = Jumlah situasi penggunaan

d. Satisfaction

Satisfaction (Kepuasan) berhubungan dengan kepuasan pelanggan terhadap perangkat lunak yang digunakan.

Tabel 2.5 *Satisfaction*

Karakteristik	Metriks	Rumus
Satisfaction	*Discretionary usage*	X = A / B A = berapa kali fungsi perangkat lunak tertentu / aplikasi / sistem yang digunakan B = jumlah pengguna menggunakan
	Satisfaction questionnaire	X = A / n Ai) menanggapi pertanyaan n = jumlah tanggapan

e. Kriteria nilai akhir kualitas perangkat lunak *quality in use* standar ISO 9126.

Tabel 2.6 Kriteria Nilai Metriks Quality in use

Weight	***Range* Nilai**
Kurang	0 – 0.33
Cukup Baik	0.34 – 0.67
Baik	0.68 – 1

Apabila nilai akhir dari sisi *Quality in Use* berada dalam kategori kurang maka perangkat lunak tersebut mendapatkan respon yang kurang baik dari masyarakat, apabila penilaian akhir dalam kategori cukup maka perangkat lunak tersebut mendapatkan respon yang cukp baik dari masyarakat dan apabila penilaian akhir mendapatkan respon baik maka perangkat lunak mendapatkan respon yang baik dari masyarakat.

2.8 Metode Pengukuran Kualitas Perangkat Lunak

Pengukuran kualitas perangkat lunak merupakan hal yang terpenting dari praktek rekayasa perangkat lunak yang baik. Pengukuran ini membantu untuk membuat karakteristik khusus dari proses dan produk informasi. Pengukuran meliputi evaluasi kuantitatif yang biasanya menggunakan metriks serta ukuran dapat menggunakan secara langsung menentukan pencapaian tujuan kualitas numerik. Pengukuran selalu menjadi fundamental bagi kemajuan untuk setiap disiplin rekayasa dan pengujian perangkat lunak. pengukuran perangkat lunak dan metrik menjadi peran mendasar dalam siklus hidup pengembangan perangkat lunak, dan Metriks perangkat lunak telah digunakan dalam membuat keputusan kuantitatif / kualitatif maupun dalam penilaian risiko dan pengurangan dalam proyek perangkat lunak. Proses pengukuran merupakan cara untuk melakukan pengukuran faktor kualitas dari sebuah perangkat lunak.

Pengukuran kualitas perangkat lunak *website* Polda Jabar akan menggunakan pehitungan OO Metriks untuk menghitung bagian internal program, karena OO metriks sesuai dengan website Polda Jabar yang berorientasi objek. Sedangkan untuk perhitungan ekstenal menggunakan metriks ISO 9126, Metriks 9216 ini melakukan perhitungan *External* dan *Quality in use* berdasarkan karakteristik dari 6 faktor yang ada pada model ISO 9126 [7][9]. Penghapusan kesalahan adalah Sebuah ukuran seberapa banyak cacat yang ditemukan selama *review* dihapus (dikoreksi) selama desain dan implementasi tahap. Metrik dapat dihitung dengan rumus X = A / B. dimana A adalah jumlah *bug* tetap selama desain dan *coding* dan B adalah Jumlah yang ditemukan selama *review*. Semakin dekat nilai adalah 1, yang lebih baik. A Nilai penghapusan kesalahan orang akan berarti bahwa setiap cacat terdeteksi telah dihapus. Metriks Eksternal ISO 9126 mempunyai beberapa metrik eksternal. Metriks eksternal didefinisikan dalam ISO 9126-2 dan diukur selama pengujian dinamis. Kategori metrik ini sesuai dengan yang internal *yaitu maturity, fault tolerance, recoverability, and compliance*. Metriks Kematangan mengukur kebebasan sistem perangkat lunak dari kegagalan disebabkan oleh *bug* di sistem itu sendiri.

2.9 Pengukuran

Pengukuran perangkat lunak adalah faktor utama untuk mengetahui seberapa baik perangkat lunak yang akan dinilai berdasarkan metode yang digunakan. Pengukuran perangkat lunak dilakukan dengan cara mencari nilai kriteria perangkat lunak tersebut dan juga mencari bobot kepentingan berdasarkan metode yang akan digunakan. Penilaian keseluruhan dapat diperoleh dari hasil nilai kriteria dan juga nilai bobot. Perhitungan bobot dan nilai kriteria bisa bervariasi tergantung dengan metode apa yang digunakan [10].

2.10 Metode Pembobotan

Rank Order Centroid (ROC) didasarkan pada tingkat kepentingan atau prioritas dari kriteria. Menurut Jeffreys dan Cockfield dalam Afiefah Rahma, teknik ROC memberikan bobot pada setiap kriteria sesuai dengan ranking yang dinilai berdasarkan tingkat prioritas. Biasanya dibentuk dengan pernyataan "Kriteria 1 lebih penting dari kriteria 2, yang lebih penting dari kriteria 3" dan seterusnya hingga kriteria ke n, ditulis . Untuk menentukan bobotnya, diberikan aturan yang sama yaitu dimana merupakan bobot untuk kriteria [10][13].

Atau dapat dijelaskan sebagai berikut:

Jika
$Cr1 \geq Cr2 \geq Cr3 \geq \ldots \geq Crn$

Maka
$W1 \geq W2 \geq W3 \geq \ldots \geq Wn$

Selanjutnya, jika k merupakan banyaknya kriteria, maka

$$W1 = \frac{1 + \frac{1}{2} + \frac{1}{3} + \cdots \ldots . + 1/k}{k}$$

$$W2 = \frac{0 + \frac{1}{2} + \frac{1}{3} + \cdots \ldots . + 1/k}{k}$$

$$W3 = \frac{0 + 0 + \frac{1}{3} + \cdots \ldots . + 1/k}{k}$$

$$Wk = \frac{0 + 0 + 0 + \cdots \ldots . + 1/k}{k}$$

Secara umum pembobotan ROC dapat dirumuskan sebagai berikut :

$$Wk = 1/k \sum_{t=1}^{k} (1/\mathrm{i})$$

2.11 Metode Pengolahan Kuesioner

Kuesioner merupakan sekumpulan pertanyaan yang diajukan kepada responden dengan tujuan dan maksud tertentu untuk memperoleh data, sehingga data tersebut dapat digunakan atau diolah kembali. Untuk mengolah data dari kuesioner tersebut dapat menggunakan berbagai macam metode pengolahan data sesuai dengan rumus metriks yang digunakan pada ISO 9126. Kuisioner ini diajukan untuk memenuhi nilai A, B maupun T yang dibutuhkan untuk menghitung nilai metriks dai setiap kriteria yang sudah ada. Dari hasil jawaban yang diberikan oleh responden akan dicari nilai rata rata, hasil jawaban responden akan dijumlahkan dan totalnya akan dibagi dengan jumlah responden. Maka akan didapat nilai A, B ataupun T.

2.12 Metode Penilaian

Perhitungan nilai sub faktor , faktor maupun nilai keseluruhan menjadi faktor penting untuk mengetahui hasil akhir sebuah nilai. menentukan nilai kriteria per faktor nya dan nilai akhir dengan rumus perhitungan *Weight Summation*. Nilai w adalah nilai metriks yang diperoleh dari hasil pengolahan kuisioner dan m adalah bobot dari sub faktor atau faktor yang sudah diperoleh dari hasil perhitungan dengan menggunakan metode *Rank Order Centroid* (ROC) [12].

1 Nilai kriteria

$$\text{Kriteria (a)} = \sum_{j=1}^{M} w_{j.}\, m_{ij} \ldots\ldots\ldots$$

$$= w_1 m_1 + w_2 m_2 + \ldots\ldots \text{ wnmn}$$