BAB II
LANDASAN TEORI

2.1 Plasmodium Malaria

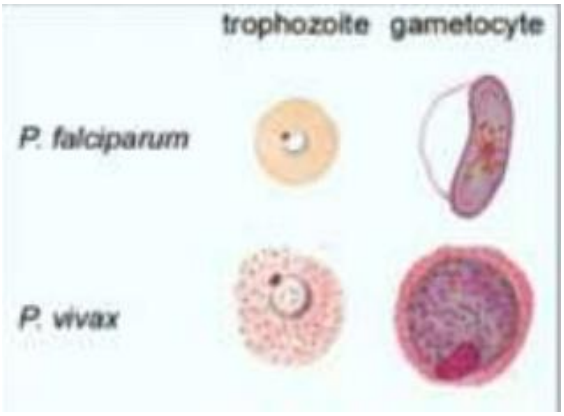
Protozoa bersporozoa dari genus plasmodium adalah parasit amuboid intrasel vertebrata yang menghasilkan pigmen dengan satu habitat dalam sel darah merah dan lainnya dalam sel jaringan lain, penularan pada manusia melalui gigitan dari berbagai spesies nyamuk betina dari genus anopheles yang telah menyelesaikan siklus perkembangan seksual dalam tubuhnya (siklus sporogoni yang menghasilkan sporozoit infektif)[7].

Penyakit malaria sudah dikenal sejak zaman Yunani. Klinik penyakit malaria khas, mudah dikenal, karena demam yang naik turun dan teratur disertai menggigil, maka pada waktu itu sudah dikenal demam tersiana (demam berulang tiga hari) dan kuartana (demam berulang empat hari). Disamping itu terdapat pula adanya kelainan pada limpa, berupa splenomegali (limpa membesar dan menjadi keras), sehingga tidak heran dulunya disebut sebagai demam kura. Namun awalnya penyakit ini dianggap sebagai penyakit hukuman para dewa karena terjadinya wabah disekitar kota Roma, yang kondisi daerahnya berawa-rawa yang mengeluarkan bau busuk disekitarnya, maka penyakit tersebut dinamakan malaria (mal = buruk, area = udara)[7].

2.1.1 Jenis Plasmodium

Jenis Plasmodium terdiri dari 2 spesies penting dalam parasitologi medik, yaitu :

1) Plasmodium falcifarum (malaria tertiana maligna) : penyebab malaria tropika yang sering menyebabkan penyakit malaria berat/malaria otak dengan kematian.
2) Plasmodium vivax (malaria tertiana benigna) : penyebab malaria tertiana yang ringan.



Gambar 2. 1 Jenis Plasmodium Malaria

2.1.2 Cara Penularan

Waktu antara nyamuk mengisap darah yang mengandung gametosit sampai mengandung sporozoit dalam kelenjar ludahnya, disebut masa inkubasi ekstrinsik sporozoit adalah bentuk infektif, penularan dapat terjadi dengan 2 cara yaitu:

1. Penularan secara alamiah (melalui vektor) ; bila sporozoit dimasukkan ke dalam badan manusia melalui gigitan nyamuk Anopheles.
2. Penularan non alamiah (induced) ; terjadi bila stadium asexual dalam eritrosit secara tidak sengaja masuk dalam badan manusia dengan cara:
    a) Malaria bawaan ; terjadi pada bayi yang baru dilahirkan karena ibunya menderita malaria, penularan terjadi melalui tali pusat atau plasenta.
    b) Secara mekanik ; terjadi melalui transfusi darah atau melalui jarum suntik yang tidak steril lagi ; cara penularan ini pernah dilaporkan terjadi di salah satu rumah sakit di Bandung pada tahun 1981, pada penderita yang dirawat dan mendapatkan suntikan intravena dengan menggunakan alat suntik yang dipergunakan untuk menyuntik beberapa pasien, dimana alat suntik itu seharusnya dibuang sekali pakai.
    c) Secara oral; cara penularan ini pernah dibuktikan pada burung, ayam (P. gallinasium), burung dara (P.relection) dan monyet (P. knowlesi).

Pada umumnya sumber infeksi malaria pada manusia lain yang sakit malaria baik dengan gejala maupun tanpa gejala klinis. Kecuali bagi simpanse di Afrika yang dapat terinfeksi oleh P.malariae belum diketahui ada hewan lain yang dapat

menjadi sumber bagi plasmodium yang biasanya menyerang manusia. Infeksi malaria pada waktu yang lalu sengaja dilakukan untuk mengobati penderita neurosifilis yaitu penderita sifilis yang sudah mengalami kelainan pada susunan sarafnya. Cara ini sekarang tidak pernah dilakukan lagi. Beberapa faktor yang mempengaruhi terjadinya penularan alamiah seperti adanya gametosit pada penderita, umur nyamuk, dan kontak antara manusia dengan nyamuk.

2.1.3 Diagnosa Menggunakan Mikroskop Cahaya

Sampai saat ini pemeriksaan malaria secara mikroskopis cahaya merupakan cara diagnosis laboratorium yang paling di andalkan dibanding dengan cara pemeriksaan lainnya, sehingga secara teknis, hal-hal yang perlu diperhatikan dalam diagnosis laboratorium secara mikroskopis ini perlu dikendalikan hal-hal sebagai berikut :

1. Dianjurkan untuk mengambil darah dari ujung jari apabila hanya sedikit darah yang dibutuhkan (tidak ada permintaan pemeriksaan lainnya) dan harus aseptis.
2. Darah yang keluar harus bebas ketika diambil untuk membuat sediaan dan tidak boleh tercampur dengan alkohol yang dipakai untuk membersihkan jari sebelumnya.
3. Darah dapat dikumpulkan dengan mempergunakan antikoagulan EDTA. Tetapi bila disimpan untuk waktu yang lama, titik-titik dalam eritrosit yang terinfeksi tidak akan terlihat (misalnya P.vivax). perlu diingat bahwa perbandingan yang akurat antara darah dan antikoagulant penting untuk mendapatkan morfologi parasit yang baik.
4. Dianjurkan untuk membuat sediaan darah tipis dan tebal, diutamakan dengan cara/metode sediaan darah tebal, dengan kriteria : (a) diameter sediaan darah atau lebar sisi 1 cm. (b) tebal darah 20 mikron, yang dapat dicek dengan meletakkan tulisan/koran ukuran sedang dibawah sediaan, tulisan tersebut masih dapat terbaca atau terlihat di bawah mikroskop.
5. Setiap lapangan pandang (LP) mikroskop terdapat +10 – 20 leukosit.
6. Sediaan darah diwarnai dengan giemsa 5% selama 45 – 60 menit atau 10% selama 15 – 20 menit. Larutan giemsa harus selalu dibuat baru saat akan

digunakan. Larutan dibuat sesuai kebutuhan dengan perhitungan 1 ml larutan untuk setiap slide. Dapat juga digunakan pulasan lainnya yaitu wright atau wright-giemsa.

7. Parasitemia rendah dengan bentuk cincin yang tidak jelas menyebabkan parasit tidak terdeteksi sehingga dianjurkan untuk memeriksa dengan minyak emersi atau anisol dengan pembesaran 500 – 600kali (5 atau 6 x 100).
8. Paling sedikit diperiksa 100– 150 LP (telah melihat 1000 – 1500 leukosit) sebelum melaporkan suatu hasil yang negative, karena satu set sediaan dengan hasil negatif belum menyingkirkan kemungkinan malaria, harus dilakukan pemeriksaan lagi 36 jam kemudian.
9. Jumlah LP yang diperiksa harus ditambah apabila pasien menelan obat profilaktit dalam 48 jam terakhir (jumlah sel yang terinfeksi dapat berkurang).
10. Setiap laboratorium yang mempunyai ahli untuk melakukan identifikasi malaria harus siap selama 24 jam/ 7 hari seminggu.
11. Penulisan Hasil Pemeriksaan;
    a) Kebutuhan epidemiologi ; F (P.falsifarum +)jika ditemukan ring, F+g (P.falsifarum +) jika ditemukan ring dan gametosit pisang (khas), Fg (P.falsifarum +) jika hanya ditemukan bentuk gametosit pisang (khas), V (P.vivax +) jika ditemukan bentuk tropozoit ameboid dan bentuk vivax lainnya, M (P.malariae +) jika ditemukan bentuk tropozoit dan lainnya dari P.malariae, Mix (jika ditemukan bentuk khas dari P.vivax dan atau P.falsifarum, atau Pmalariae.
    b) Tingkat Puskesmas atau rumah sakit ; cukup disebutkan negatif atau positif dengan keterangan jenisnya : Pf, Pv, mix atau gametosit Pf.

Evaluasi diagnostic dan terapi ; secara khusus hasil pemeriksaan ditujukan untuk mengetahui kepadatan parasit malaria, dan dapat dituliskan menurut cara WHO : Tidak ditemukan plasmodium pada 100 LP ; (-) negative, ditemukan 1–10 pada 100 LP ; (+), ditemukan 11–100 pada LP ; (++). ditemukan 1–10 pada setiap LP ; (+++). ditemukan 11–100 pada setiap LP ; (++++). (d) Pemeriksaan Cross –

Check ; Setiap sediaan yang positif dan 5% dari sediaan negatif yang dipilih secara acak dikirim ke laboratorium malaria dinkes kabupaten atau provinsi untuk diperiksa ulang dalam rangka pembinaan dan pemantauan kualitas mikroskopis yang melakukan pemeriksaan lebih awal.

2.2 Darah

Darah adalah cairan yang terdapat pada manusia dan hewan tingkat tinggi yang berfungsi sebagai alat transportasi zat seperti oksigen, bahan hasil metabolisme tubuh, pertahanan tubuh dari sengaran kuman, dan lain sebagainya. Beda halnya dengan tumbuhan, manusia dan hewan level tinggi punya sistem transportasi dengan darah[8].

Cairan ini berwarna merah yang terdapat di dalam pembulu darah. Warna merah tersebut tidak selalu tetap, tetapi berubah-ubah karena pengaruh zat kandungannya, terutama kadar oksigen dan CO_2. Bila kadar oksigen tinggi maka warnanya menjadi merah tua.

Darah pada tubuh manusia mengandung 55% plasma darah (cairan darah) dan 45% sel-sel darah (darah padat). Volume darah pada manusia atau hewan level tinggi (mamalia) adalah 8% berat badannya. Darah pada tubuh manusia sekitar sepertigabelas beratnya atau sekitar 4 atau 5 liter pada orang dewasa.

Darah merupakan cairan yang sangat penting bagi manusia karena berfungsi sebagai alat transportasi serta memiliki banyak kegunaan lainnya untuk menunjang kehidupan. Tanpa darah yang cukup seseorang dapat mengalami gangguan kesehatan dan bahkan dapat mengakitbatkan kematian.

2.3 Computer Vision

Computer Vision merupakan bidang yang berkembang pesat dikhususkan untuk menganalisa, memodifikasi, dan pemahaman gambar tingkat tinggi. Tujuannya adalah untuk menentukan apa yang terjadi di depan kamera dan menggunakan pemahaman tersebut untuk mengendalikan komputer, sistem robot, atau untuk memberikan citra baru yang lebih informatif dari pada gambar kamera asli. Aplikasi area untuk komputer visi teknologi termasuk video surveillance,

biometrik, otomotif, fotografi, produksi film, pencarian web, obat-obatan, game augmented reality, antarmuka pengguna yang baru, dan banyak lagi[9].

2.4 Teori Dasar Citra Digital

Dalam bidang pengolahan citra (*image processing*), citra yang diolah adalah citra digital, yaitu citra kontinu yang telah diubah ke dalam bentuk diskrit, baik koordinat ruangnya maupun intensitas (kecerahan) cahayanya melalui proses sampling dan kuantisasi[10]. Citra dapat diartikan sebagai suatu fungsi kontinu dari intensitas cahaya (x,y) dalam bidang dua dimensi, dengan (x,y) menyatakan suatu koordinat spasial dan nilai f pada setiap titik (x,y) menyatakan intensitas atau tingkat kecerahan atau derajat keabuan (brightness/gray level).

Piksel (0,0) terletak pada sudut kiri atas pada citra, indeks x bergerak ke kanan dan indeks y bergerak ke bawah. Secara matematis persamaan untuk fungsi intensitas f(x,y) adalah:

$$0 \leq f\,(x,y) < \infty \qquad (2.1)$$

Misalkan f merupakan sebuah citra digital 2 dimensi berukuran MxN. Maka representais f dalam sebuah matriks dapat dilihat pada gambar berikut.

$$f(x,y) = \begin{matrix} f(0,0) & f\ 0,1 & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \cdot & \cdot & \cdot & \cdot \\ f(M-0) & f(M-1) & \dots & f(M-1,N-1) \end{matrix}$$

Gambar 2. 2 Matriks Citra Digital

2.4.1 Citra RGB

RGB merupakan model warna yang terdiri dari merah (Red), hijau (Green), dan biru (Blue) yang digabungkan dan membentuk suatu susunan warna yang luas. Setiap warna dasar memiliki rentang nilai antara 0 smpai 255[11].

2.4.2 Citra Grayscale

Grayscale adalah warna-warna piksel yang berada dalam rentang gradasi warna hitam dan putih. Intensitas grayscale biasanya disimpan sebagai data citra 8 bit atau 256 intensitas warna gray dari nilai 0 (hitam) dan 255 (putih) [11]. Citra abu-abu dapat dihasilkan dari citra warna RGB dengan mengalikan ketiga

komponen warna pokok merah, hijau, dan biru dengan suatu koefisien yang jumlahnya satu (a+b+c = 1).

$$Y=a.R+b.G+c.B$$

$$Gray[x,y]=0.3.R+0.59.G+0.11.B=\cdots \quad (2.2)$$

2.4.3 Citra Biner

Citra biner diperoleh melalui proses pemisahan piksel-piksel berdasarkan derajat keabuan yang dimilikinya. Piksel yang memiliki derajat keabuan lebih kecil dari nilai batas (threshold) yang ditentukan akan diberikan nilai nol (hitam). Sedangkan piksel yang memiliki derajat keabuan lebih besar dari nilai threshold akan diubah menjadi satu (putih)[11].

2.5 Pengolahan Citra Digital

Pengolahan citra digital bertujuan untuk memanipulasi dan menganalisis citra dengan bantuan komputer. Pengolahan citra digital dapat dikelompokkan dalam dua jenis kegiatan:

1) Memperbaiki kualitas suatu gambar, sehingga dapat lebih mudah diinterpretasi oleh mata manusia.
2) Mengolah informasi yang terdapat pada suatu gambar untuk keperluan pengenalan objek secara otomatis.

Secara umum tahapan pengolahan citra digital meliputi akusisi citra, peningkatan kualitas citra, segmentasi citra, representasi dan uraian, pengenalan dan interpretasi[11].

2.5.1 Akuisisi Citra

Pengambilan data dapat dilakukan dengan menggunakan berbagai media seperti kamera analog, kamera digital, handycam, scanner, optical reader dan sebagainya.

2.5.2 Peningkatan Kualitas Citra

Pada tahap ini dikenal dengan pre-processing dimana dengan meningkatkan kualitas citra dapat meningkatkan kemungkinan dalam keberhasilan pada tahap pengolahan citra digital berikutnya.

2.5.3 Pemotongan Citra

Pemotongan citra (cropping) adalah suatu proses pembuangan pixel citra yang tidak diperlukan. *Cropping* merupakan salah satu proses yang sangat sering digunakan dalam memanipulasi gambar. Kegunaan dari proses *cropping* ini adalah untuk menghilangkan subjek dari gambar yang tidak diperlukan, untuk mengubah aspek ratio suatu gambar, atau untuk meningkatkan komposisi gambar secara keseluruhan.

2.5.4 *Filter*

Filter merupakan suatu proses penyaringan yang mempengaruhi hasil dari suatu gambar. Filter ini digunakan dalam hal penghilangan derau (*noise*) terhadap gambar, dan pengaturan pencahayaan terhadap gambar. *Filter* ini biasa digunakan untuk menghasilkan gambar yang bagus dan jelas tanpa ada gangguan noise ataupun pencahayaan yang berlebihan.

2.6 *Pattern Recognition*

Pola adalah suatu *entitas* yang terdefinisi (mungkin secara samar) dan dapat diidentifikasi serta diberi nama. Pola bisa merupakan kumpulan hasil pengukuran atau pemantauan dan bisa dinyatakan dalam notasi vektor. Contoh: sidik jari, raut wajah, gelombang suara, tulisan tangan, tekstur kulit dan lain sebagainya. Dalam pengenalan pola data yang akan dikenali biasanya dalam bentuk citra atau gambar, akan tetapi ada pula yang berupa suara[11]. Secara umum pengenalan pola (*pattern recognition*) adalah suatu ilmu untuk mengklasifikasikan atau menggambarkan sesuatu berdasarkan pengukuran kuantitatif fitur (ciri) atau sifat utama dari suatu obyek.

Pengenalan pola (pattern recognition) merupakan teknik yang bertujuan untuk mengklasifikasikan citra yang telah diolah sebelumnya berdasarkan kesamaan atau kemiripan ciri yang dimilikinya. Bagian terpenting dari teknik pengenalan pola adalah bagaimana memperoleh informasi atau ciri penting yang terdapat dalam sinyal[11].

2.7 Metode Principal Component Analysis (PCA)

Principal Component Analysis adalah teknik statistik yang sudah digunakan secara luas baik dalam hal pengenalan wajah maupun pengenalan pola dari sebuah gambar. Metode *Principal Component Analysis* (PCA) dibuat pertama kali oleh para ahli statistik dan ditemukan oleh Karl Pearson pada tahun l90l yang memakainya pada bidang biologi. Kemudian tidak ada perkembangan baru pada teknik ini dan perkembangannya baru mulai pesat pada akhir tahun l930 dan awal 1940. Analisis komponen utama merupakan suatu teknik statistik untuk mengubah dari sebagian besar variabel asli yang digunakan dan saling berkorelasi satu dengan yang lainnya menjadi satu set variabel baru yang lebih kecil dan tidak berkorelasi [4]. Setiap 4 pengukuran multivariat (atau observasi), komponen utama merupakan kombinasi linier dari variabel p awal. Tujuan utama analisis komponen utama ialah untuk mengurangi dimensi peubah-peubah yang saling berhubungan dan cukup banyak variabelnya sehingga lebih mudah untuk menginterpretasikan data-data tersebut [4]. Metode yang digunakan yaitu menentukan komponen utama dengan melakukan alih ragam orthogonal atau membentuk kombinasi linier Y = A' X[6]. Dari sini akan dipilih beberapa komponen utama yang dapat memberikan sebagian besar keragaman total data semula. Langkah - langkah untuk melakukan PCA secara garis besar adalah sebagai berikut:

a) Standarisasikan data – data yg ada.

   Ada kalanya data – data yg kita dapatkan memiliki perbedaan skala yg cukup mencolok. Satu data berkisar antara 10^{-1}, sementara lainnya bernilai 10^{2}. Perbedaan skala ini akan menyebabkan ketidaksimetrisan persebaran (variance) data. Oleh karena itu, data-data tersebut mesti distandardisasikan degan cara seperti berikut:

$$x_{baru} = \frac{x_{lama} - nilai\ rata - ratanya}{standar\ deviationnya} = \frac{x_i - \bar{x}}{\sigma} \qquad (2.1)$$

b) Buat covariance matriks dari variabel – variabel yang ada

Covariance adalah variance antara dua variable, misalnya X dan Y. Dengan merujuk ke formula variance, maka covariance dihitung sebagai berikut:

$$variance\ (x,x) = \sum_{i-1}^{N} \frac{(x_i - \bar{x})(x_i - \bar{x})}{N-1} \quad (2.2)$$

$$variance\ (x,y) = \sum_{i-1}^{N} \frac{(x_i - \bar{x})(y_i - \bar{y})}{N-1} \quad (2.3)$$

Semakin banyak variable yang terlibat, maka semakin besar matriks covariance - variance nya.

c) Cari eigen value dan eigen vector dari covariance matriks tersebut.

Nilai eigen value menentukan principal component yg terlibat. Nilai ini dihitung dengan menyelesaikan persamaan berikut:

$$\det[A - \lambda I] = 0 \quad (2.4)$$

Dimana:

A = Covariance matriks

λ = eigen value

I = Identity matriks

Dari persamaan di atas, jumlah eigen value akan sebanyak jumlah variabel yg terlibat. Kemudian, eigen vector dihitung dengan menyelesaikan persamaan di bawah:

$$[A - \lambda I\ \mathrm{I}X] = 0 \quad (2.5)$$

X = eigen vector untuk eigen value yang digunakan.

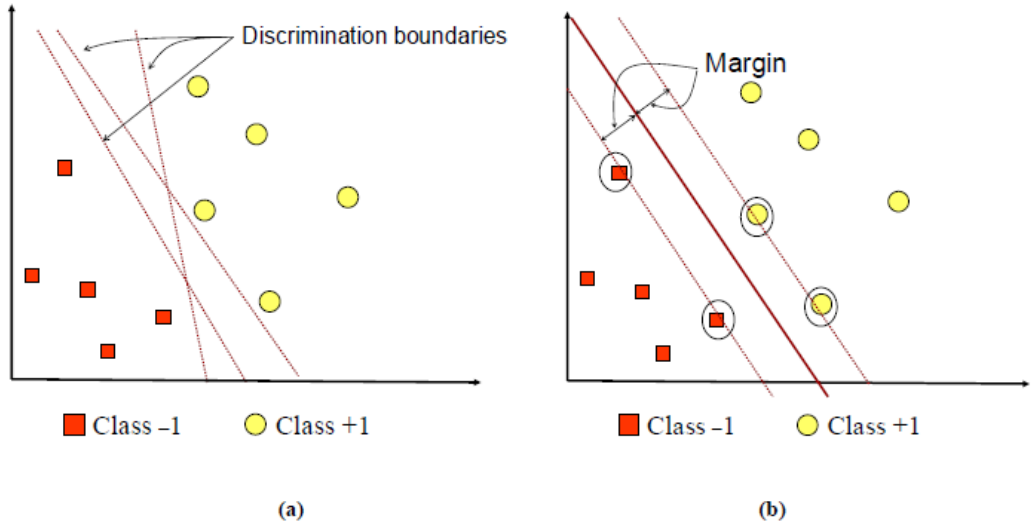
Persamaan di atas diselesaikan untuk setiap eigenvalue yg didapat. Eigenvector dgn eigenvalue terbesar disebut principal component pertama, dst. Biasanya, dua principal component sudah cukup utk merangkum sekitar 80-90% variations di data.

2.8 Metode Support Vector Machines

Support Vector Machines (SVM) merupakan metode klasifikasi jenis terpadu (*supervised*) karena ketika proses pelatihan, diperlukan target pembelajaran tertentu. SVM juga merupakan algoritma yang bekerja menggunakan pemetaan

nonlinear untuk mengubah data pelatihan asli ke dimensi yang lebih tinggi. Dalam hal ini dimensi baru, akan mencari *hyperplane* untuk memisahkan secara linear dan dengan pemetaan nonlinear yang tepat ke dimensi yang cukup tinggi, data dari dua kelas selalu dapat dipisahkan dengan *hyperplane* tersebut. SVM menemukan *hyperplane* ini menggunakan *support vector* dan *margin*[12].

SVM muncul pertama kali pada tahun 1992 oleh Vladimir Vapnik bersama rekannya Bernhard Boser dan Isabelle Guyon. Dasar untuk SVM sudah ada sejak tahun 1960-an (termasuk karya awal oleh vapnik dan Alexei Chervonenkis pada teori belajar statistik).



Gambar 2. 3 SVM berusaha menemukan hyperplane terbaik yang memisahkan kedua class –1 dan +1

Hyperplane pemisah terbaik antara kedua class dapat ditemukan dengan mengukur *margin hyperplane* tersebut dan mencari titik maksimalnya. *Margin* adalah jarak antara *hyperplane* tersebut dengan *pattern* terdekat dari masing-masing class. Pattern yang paling dekat ini disebut sebagai *support vector*. Garis *solid* pada gambar 2.7-b menunjukkan *hyperplane* yang terbaik, yaitu yang terletak tepat pada tengah-tengah kedua class, sedangkan titik merah dan kuning yang berada dalam lingkaran hitam adalah *support vector*. Usaha untuk mencari lokasi *hyperplane* ini merupakan inti dari proses pembelajaran pada SVM[12].

Konsep SVM dapat dijelaskan secara sederhana sebagai usaha mencari *hyperplane* terbaik yang berfungsi sebagai pemisah dua buah class pada input space. Gambar 2.3-a memperlihatkan beberapa *pattern* yang merupakan anggota dari dua buah class : +1 dan –1. *Pattern* yang tergabung pada class –1 disimbolkan dengan warna merah (kotak), sedangkan *pattern* pada class +1, disimbolkan dengan warna kuning (lingkaran). Problem klasifikasi dapat diterjemahkan dengan usaha menemukan garis (*hyperplane*) yang memisahkan antara kedua kelompok tersebut.

Berbagai alternatif garis pemisah (*discrimination boundaries*) ditunjukkan pada gambar 2.3 (a). Hyperplane pemisah terbaik antara kedua *class* dapat ditemukan dengan mengukur *margin* hyperplane tersebut dan mencari titik maksimalnya. *Margin* adalah jarak antara hyperplane tersebut dengan *pattern* terdekat dari masing-masing class. *Pattern* yang paling dekat ini disebut sebagai *support vector*. Garis solid pada gambar 2.3 (b) menunjukkan hyperplane yang terbaik, yaitu yang terletak tepat pada tengah-tengah kedua *class*, sedangkan titik merah dan kuning yang berada dalam lingkaran hitam adalah *support vector*. Usaha untuk mencari lokasi hyperplane ini merupakan inti dari proses pembelajaran pada SVM. Hyperplane dapat didefinisikan dengan persamaan berikut.

$$g:\ w^t + b = 0 \qquad (2.6)$$

Data yang tersedia dinotasikan sebagai $\overrightarrow{x_l} \in \Re^d$ sedangkan label masing-masing dinotasikan $y_i \in \{-1, +1\}$ untuk $i = 1,2,3,\ldots,l$ yang mana l adalah banyaknya data. Diasumsikan kedua class –1 dan +1 dapat terpisah secara sempurna oleh hyperplane berdimensi d , yang didefinisikan

$$\vec{w}.\vec{x} + b = 0 \qquad (2.7)$$

Pattern $\overrightarrow{x_l}$ yang termasuk class –1 (sampel negatif) dapat dirumuskan sebagai pattern yang memenuhi pertidaksamaan,

$$\vec{w}.\overrightarrow{x_l} + b \leq -1 \qquad (2.8)$$

sedangkan pattern $\overrightarrow{x_l}$yang termasuk class +1 (sampel positif).

$$\vec{w}.\overrightarrow{x_l} + b \leq +1 \qquad (2.9)$$

Margin terbesar dapat ditemukan dengan memaksimalkan nilai jarak antara hyperplane dan titik terdekatnya, yaitu $1/\|\vec{w}\|$. Hal ini dapat dirumuskan sebagai

Quadratic Programing (QP) problem, yaitu mencari titik minimal persamaan 2.10 dengan memperhatikan constraint persamaan 2.11.

$$\frac{min}{w}\ \tau\ (w) = \frac{1}{2}\ ||w||^2 \qquad (2.10)$$

$$y_i\ (\overrightarrow{x_l}.\vec{w}+b)-1 \geq 0\ ,\ \forall_i \qquad (2.11)$$

Problem ini dapat dipecahkan dengan berbagai teknik komputasi, di antaranya *Lagrange Multiplier*.

$$L(\vec{w}, b, a) = \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{l} a_i\ (yi\big((xi.w + b) - 1\big))$$

$$i=1,2,3,\ldots,l \qquad (2.12)$$

α_i adalah lagrange multiplier, yang bernilai nol atau positif ($\alpha_i \geq 0$). Nilai optimal dari persamaan 2.12 dapat dihitung dengan meminimalkan *L* terhadap *w* dan *b*, dan memaksimalkan *L* terhadap α_i. Dengan memperhatikan sifat bahwa pada titik optimal gradient *L*= 0, persamaan 2.12 dapat dimodifikasi sebagai maksimalisasi problem yang hanya mengandung α_i saja, sebagaimana persamaan 2.13 dibawah.

Maximize :

$$\sum_{i=1}^{l} ai - \frac{1}{2}\sum_{i,j=1}^{l} a_i a_j y_i y_j x_i x_j \qquad (2.13)$$

Subject to :

$$a_i \geq 0(i = 1,2,3, \ldots, l)\ \sum_{i=1}^{l} a_i y_i = 0 \qquad (2.14)$$

Dari hasil perhitungan ini diperoleh αi yang kebanyakan nilai positif. Data yang berkorelasi dengan αi yang positif inilah yang disebut support vector.

2.9 *Object – Oriented Programing*

Metodologi berorientasi objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya[13]. Metodologi berorientasi objek merupakan suatu cara bagaimana sistem perangkat lunak dibangun melalui pendekatan objek secara sistematis. Metode berorientas objek didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas. Metode berorientasi objek meliputi rangkaian aktivitas analisis berorientasi objek,

perancangan berorientasi objek, pemrograman berorientasi objek, dan pengujian berorientasi objek.

Pada saat ini, metode berorientasi objek banyak dipilih karena metodologi lama banyak menimbulkan masalah seperti adanya kesulitan pada saat mentransformasi hasil dari satu tahap pengembangan ke tahap berikutnya, misalnya pada metode pendekatan terstruktur, jenis aplikasi yang dikembangkan saat ini berbeda dengan masa lalu. Aplikasi yang dikembangkan pada saat ini beragam dengan platform yang berbeda-beda, sehingga menimbulkan tuntutan kebutuhan metodologi pengembangan yang dapat mengakomodasi ke semua jenis aplikasi tersebut.

Keuntungan menggunakan metodologi berorientasi objek adalah sebagai berikut :

a) Meningkatkan produktivitas

   Karena kelas dan objek yang ditemukan dalam suatu masalah masih dapat dipakai ulang untuk masalah lainnya yang melibatkan objek tersebut (reusable).

b) Kecepatan pengembangan

   Karena sistem yang dibangun dengan baik dan benar pada saat analisis dan perancangan akan menyebabkan berkurangnya kesalahan pada saat pengkodean.

c) Kemudahan pemeliharaan

   Karena dengan model objek, pola-pola yang cenderung tetap dan stabil dapat dipisahkan dan pola-pola yang mungkin sering berubah-ubah.

d) Adanya konsistensi

   Karena sifat pewarisan dan penggunaan notasi yang sama pada saat analisis, perancangan maupun pengkodean.

e) Meningkatkan kualitas perangkat lunak

   Karena pendekatan pengembangan lebih dekat dengan dunia nyata dan adanya konsistensi pada saat pengembangannya, perangkat lunak yang dihasilkan akan mampu memenuhi kebutuhan pemakai serta mempunyai sedikit kesalahan.

2.9.1 Konsep Dasar Berorientasi Objek

Pendekatan berorientasi objek merupakan suatu teknik atau cara pendekatan dalam melihat permasalahan dan sistem (sistem perangkat lunak, sistem informasi, atau sistem lainnya)[13]. Pendekatan berorientasi objek akan memandang sistem yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata.

Ada banyak cara untuk mengabstraksikan dan memodelkan objek-objek tersebut, mulai dan abstraksi objek, kelas, hubungan antar kelas sampai abstraksi sistem. Saat mengabstraksikan dan memodelkan objek, data dan proses-proses yang dipunyai oleh objek akan dienkapsulasi (dibungkus) menjadi suatu kesatuan.

Dalam rekayasa perangkat lunak, konsep pendekatan berorientasi objek dapat diterapkan pada tahap analisis, perancangan, pemrograman, dan pengujian perangkat lunak. Ada berbagai teknik yang dapat digunakan pada masing-masing tahap tersebut, dengan aturan dan alat bantu pemodelan tertentu.

Sistem berorientasi objek merupakan sebuah sistem yang dibangun dengan berdasarkan metode berorientasi objek adalah sebuah sistem yang komponennya dibungkus (dienkapsulasi) menjadi kelompok data dan fungsi. Setiap komponen dalam sistem tersebut dapat mewarisi atribut dan sifat dan komponen lainnya, dan dapat berinteraksi satu sama lain.

Berikut ini adalah beberapa konsep dasar yang harus dipahami tentang metodologi berorientasi objek:

1. Kelas (*Class*)

   Kelas adalah sekumpulan objek-objek dengan karakteristik yang sama. Kelas merupakan definisi statis dan himpunan objek yang sama yang mungkin lahir atau diciptakan dan kelas tersebut. Sebuah kelas akan mempunyai sifat (atribut), kelakuan (metode/operasi), hubungan (*relationship*) dan arti. Suatu kelas dapat diturunkan dan kelas yang lain, dimana atribut dan kelas semula dapat diwariskan ke kelas yang baru. Secara teknis kelas adalah sebuah struktur dalam pembuatan perangkat lunak. Kelas merupakan bentuk struktur pada kode program yang menggunakan metodologi berorientasi objek.

2. Objek (*object*)

   Objek adalah abstraksi dari sesuatu yang mewakili dunia nyata benda, manusia, satu organisasi, tempat, kejadian, struktur, status, atau hal-hal lain yang bersifat abstrak. Objek merupakan suatu *entitas* yang mampu menyimpan informasi (status) dan mempunyai operasi (kelakuan) yang dapat diterapkan atau dapat berpengaruh pada status objeknya. Objek mempunyai siklus hidup yaitu diciptakan, dimanipulasi, dan dihancurkan. Secara teknis, sebuah kelas saat program dieksekusi maka akan dibuat sebuah objek. Objek dilihat dari segi teknis adalah elemen pada saat *runtime* yang akan diciptakan, dimanipulasi, dan dihancurkan saat eksekusi sehinga sebuah objek hanya ada saat sebuah program dieksekusi. Jika masih dalam bentuk kode, disebut sebagai kelas jadi pada saat *runtime* (saat sebuah program dieksekusi), yang kita punya adalah objek, di dalam teks program yang kita lihat hanyalah kelas.

3. Metode (*method*)

   Operasi atau metode pada sebuah kelas hampir sama dengan fungsi atau prosedur pada metodologi struktural. Sebuah kelas boleh memiliki lebih dari satu metode atau operasi. Metode atau operasi yang berfungsi untuk memanipulasi objek itu sendiri. Operasi atau metode merupakan fungsi atau transformasi yang dapat dilakukan terhadap objek atau dilakukan oleh objek.

   Metode atau operasi dapat berasal dari *event*, aktifitas atau aksi keadaan, fungsi, atau kelakuan dunia nyata. Contoh metode atau operasi misalnya *Read, Write, Move, Copy,* dan sebagainya.

4. Atribut (*attribute*)

   Atribut dari sebuah kelas adalah variabel global yang dimiliki sebuah kelas. Atribut dapat berupa nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek. Atribut dipunyai secara individual oleh sebuah objek, misalnya berat, jenis, nama, dan sebagainya.

5. Abstraksi (*abstraction*)

   Prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.

6. Enkapsulasi (*encapsulation*)

   Pembungkusan atribut data dan layanan (operasi-operasi) yang dipunyai objek untuk menyembunyikan implementasi dan objek sehingga objek lain tidak mengetahui cara kerja-nya.

7. Pewarisan (*inheritance*)

   Mekanisme yang memungkinkan satu objek mewarisi sebagian atau seluruh definisi dan objek lain sebagai bagian dan dirinya.

8. Antarmuka (*interface*)

   Antarmuka sangat mirip dengan kelas, tapi tanpa atribut kelas dan memiliki metode yang dideklarasikan tanpa isi. Deklarasi metode pada sebuah interface dapat diimplementasikan oleh kelas lain.

9. *Reusability*

   Pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan pada permasalahan lainnya yang melibatkan objek tersebut.

10. Generalisasi dan Spesialisasi

    Menunjukkan hubungan antara kelas dan objek yang umum dengan kelas dan objek yang khusus. Misalnya kelas yang lebih umum (generalisasi) adalah kendaraan darat dan kelas khususnya (spesialisasi) adalah mobil, motor, dan kereta.

11. Komunikasi Antarobjek

    Komunikasi antarobjek dilakukan lewat pesan (*message*) yang dikirim dari satu objek ke objek lainnya.

12. Polimorfisme (*polymorphism*)

    Kemampuan suatu objek digunakan di banyak tujuan yang berbeda dengan nama yang sehingga menghemat baris program.

13. *Package*

    Package adalah sebuah kontainer atau kemasan yang dapat digunakan untuk mengelompokkan kelas-kelas sehingga memungkinkan beberapa kelas yang bernama sama disimpan dalam *package* yang berbeda.

2.9.2 Pengenalan UML

Pada perkembangan teknologi perangkat lunak, diperlukan adanya bahasa yang digunakan untuk memodelkan perangkat lunak yang akan dibuat dan perlu adanya standarisasi agar orang di berbagai negara dapat mengerti pemodelan perangkat lunak. Seperti yang kita ketahui bahwa menyatukan banyak kepala untuk menceritakan sebuah ide dengan tujuan untuk memahami, merupakan hal yang tidak mudah, oleh karena itu diperlukan sebuah bahasa pemodelan perangkat lunak yang dapat dimengerti oleh banyak orang.

Banyak orang yang telah membuat bahasa pemodelan pembangunan perangkat lunak sesuai dengan teknologi pemrograman yang berkembang pada saat itu, misalnya yang sempat berkembang dan digunakan banyak pihak adalah *Data Flow Diagram* (DFD) untuk memodelkan perangkat lunak yang menggunakan pemrograman prosedural atau struktural, kemudian juga ada *State Transition Diagram* (STD) yang digunakan untuk memodelkan sistem *real time* (waktu nyata).

Pada perkembangan teknik pemrograman berorientasi objek, munculah sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modeling Language* (UML). UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung.

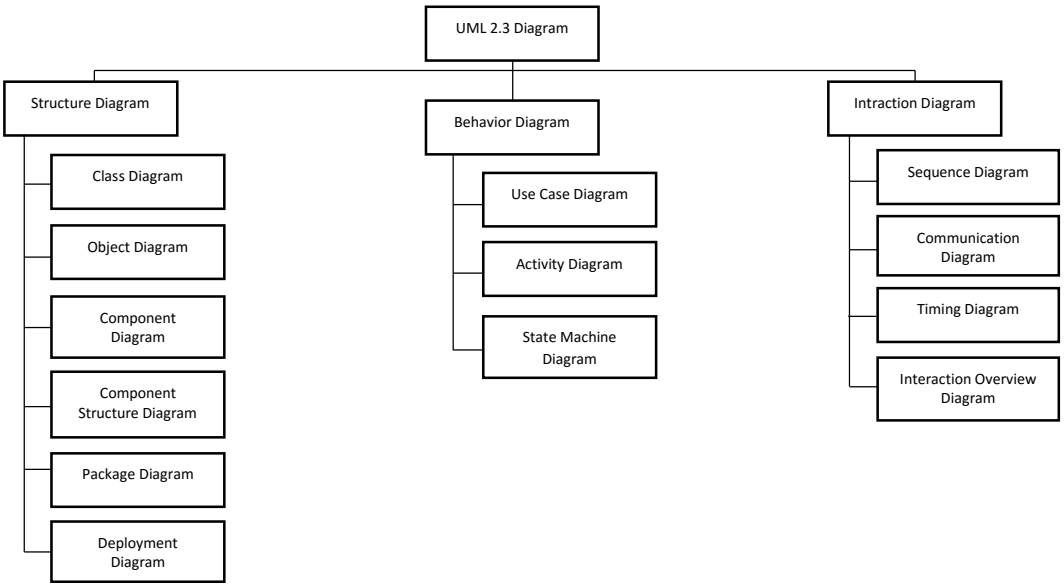
UML hanya berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada metodologi tertentu, meskipun pada kenyataannya UML paling banyak digunakan pada metodologi berorientasi objek.

Seperti yang kita ketahui di dunia sistem informasi yang tidak dapat dibakukan, semua tergantung kebutuhan, lingkungan dan konteksnya. Begitu juga

dengan perkembangan penggunaan UML bergantung pada level abstraksi penggunaannya. Jadi, belum tentu pandangan yang berbeda dalam penggunaan UML adalah suatu yang salah, tapi perlu ditelaah dimanakah UML digunakan dan hal apa yang ingin digambarkan. Secara analogi jika dengan bahasa yang digunakan sehari-hari, belum tentu penyampaian bahasa dengan puisi adalah hal yang salah. Sistem informasi bukanlah ilmu pasti, maka jika ada banyak perbedaan dan interpretasi di dalam bidang sistem informasi merupakan hal yang sangat wajar.

2.9.3 Digram UML

Pada UML 2.3 terdiri dari 13 macam diagram yang dikelompokkan dalam 3 kategori. Pembagian kategori dan macam-macam diagram tersebut dapat dilihat pada Gambar 2.4



Gambar 2. 4 Diagram UML 2.3

Berikut ini penjelasan singkat dari pembagian kategori tersebut :

a) *Structure diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan.
b) *Behavior diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan perilaku sistem atau rangkaian perubahan yang terjadi pada sebuah sistem.

c) *Interaction diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antarsubsistem pada suatu sistem.

2.9.4 Use Case Diagram

Use Case atau diagram *Use Case* merupakan pemodelan untuk kelakuan (*behavior*) 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 didalam 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*.

2.9.4.1 Aktor

Aktor adalah sesuatu (*entitas*) yang berhubungan dengan sistem dan berpartisipasi dalam *use case*. Aktor menggambarkan orang, sistem atau *entitas eksternal* yang secara khusus membangkitkan sistem dengan *input* atau masukan kejadian-kejadian, atau menerima sesuatu dari sistem. Aktor dilukiskan dengan peran yang mereka mainkan dalam *use case*, seperti Staff, Kurir dan lain-lain.

Dalam *use case* diagram terdapat satu aktor pemulai atau *initiator actor* yang membangkitkan rangsangan awal terhadap sistem, dan mungkin sejumlah aktor lain yang berpartisipasi atau *participating actor*. Akan sangat berguna untuk mengetahui siapa aktor pemulai tersebut.

2.9.4.2 *Use Case*

Use case yang dibuat berdasarkan keperluan aktor merupakan gambaran dari “apa” yang dikerjakan oleh sistem, bukan “bagaimana” sistem mengerjakannya. *Use case* diberi nama yang menyatakan apa hal yang dicapai dari interaksinya dengan aktor.

2.9.4.3 *Relationship*

Relasi (*relationship*) digambarkan sebagai bentuk garis antara dua simbol dalam *use case* diagram. Relasi antara *actor* dan *use case* disebut juga dengan

asosiasi (*association*). Asosiasi ini digunakan untuk menggambarkan bagaimana hubungan antara keduanya.

Relasi-relasi yang terjadi pada *use case* diagram bisa antara *actor* dengan *use case* atau *use case* dengan *use case*.

Relasi antara *use case* dengan *use case* :

a) Include, pemanggilan *use case* oleh *use case* lain atau untuk menggambarkan suatu *use case* termasuk di dalam *use case* lain (diharuskan). Contohnya adalah pemanggilan sebuah fungsi program. Digambarkan dengan garis lurus berpanah dengan tulisan <<include>>.

b) Extend, digunakan ketika hendak menggambarkan variasi pada kondisi perilaku normal dan menggunakan lebih banyak kontrol *form* dan mendeklarasikan ekstension pada *use case* utama. Atau dengan kata lain adalah perluasan dari use case lain jika syarat atau kondisi terpenuhi. Digambarkan dengan garis berpanah dengan tulisan <<extend>>.

c) Generalization/Inheritance, dibuat ketika ada sebuah kejadian yang lain sendiri atau perlakuan khusus dan merupakan pola berhubungan *base-parent use case*. Digambarkan dengan garis berpanah tertutup dari *base use case* ke *parent use case*.

2.9.5 *Activity Diagram*

Diagram aktivitas atau *activity diagram* menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Yang perlu diperhatikan disini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem.

Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut ini:

1) Rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan
2) Urutan atau pengelompokan tampilan dari sistem/*user interface* dimana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan
3) Rancangan pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.

2.9.6 *Sequence Diagram*

Diagram *sequence* menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram *sequence* maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya diagram *sequence* yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram *sequence* sehingga semakin banyak *use case* yang didefinisikan maka diagram *sequence* yang harus dibuat juga semakin banyak.

Penomoran pesan berdasarkan urutan interaksi pesan. Penggambaran letak pesan harus berurutan, pesan yang lebih atas dari lainnya adalah pesan yang berjalan terlebih dahulu.

2.9.7 *Class Diagram*

Diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

Kelas-kelas yang ada pada struktur sistem harus dapat melakukan fungsi-fungsi sesuai dengan kebutuhan sistem. Susunan struktur kelas yang baik pada diagram kelas sebaiknya memiliki jenis-jenis kelas berikut :

1) Kelas Main

   Kelas yang memiliki fungsi awal dieksekusi ketika sistem dijalankan.

2) Kelas yang menangani tampilan sistem

   Kelas yang mendefinisikan dan mengatur tampilan ke pemakai.

3) Kelas yang diambil dari pendefinisian *usecase*

   Kelas yang menangani fungsi-fungsi yang harus ada diambil dari pendefinisian *use case*.

4) Kelas yang diambil dari pendefinisian data

   Kelas yang digunakan untuk memegang atau membungkus data menjadi sebuah kesatuan yang diambil maupun akan disimpan ke basis data.

Jenis-jenis kelas diatas juga dapat digabungkan satu sama lain sesuai dengan pertimbangan yang dianggap baik asalkan fungsi-fungsi yang sebaiknya ada pada struktur kelas tetap ada. Susunan kelas juga dapat ditambahkan kelas utilitas seperti koneksi ke basis data, membaca *file* teks, dan lain sebagainya sesuai kebutuhan.

Dalam mendefinisikan metode yang ada di dalam kelas perlu memperhatikan apa yang disebut dengan *cohesion* dan *coupling*. *Cohesion* adalah ukuran seberapa dekat keterkaitan instruksi di dalam sebuah metode terkait satu sama lain sedangkan *coupling* adalah ukuran seberapa dekat keterkaitan instruksi antara metode yang satu dengan metode yang lain dalam sebuah kelas. Sebagai aturan secara umum maka sebuah metode yang dibuat harus memiliki kadar *choesion* yang kuat dan kadar *coupling* yang lemah.

Dalam *class diagram* terdapat beberapa relasi (hubungan antar class) yaitu:

1. *Generalization and Inheritence*

   Diperlukan untuk memperlihatkan hubungan pewarisan (*inheritance*) antar unsur dalam diagram kelas. Pewarisan memungkinkan suatu kelas mewarisi semua atribut, operasi, relasi, dari kelas yang berada dalam hirarki pewarisannya.

2. *Associations*

   Hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui ekstensi *class* lain. Dalam notasi UML kita mengenal asosiasi 2 arah (*bidirectional*) dan 1 arah (*undirectional*).

3. Aggregation

   Hubungan antar-*class* dimana *class* yang satu (*part class*) adalah bagian dari *class* lainnya (*whole class*).

4. *Composition*

   Aggregation dengan ikatan yang lebih kuat. Di dalam *composite aggregation*, siklus hidup *part class* sangat bergantung pada *whole class*

sehingga bila objek *instance* dari *whole class* dihapus maka objek *instance* dari *part calss* juga akan terhapus.

5. *Depedency*

   Hubungan antar-*class* dimana sebuah *class* memiliki ketergantungan pada *class* lainnya tetapi tidak sebaliknya.

6. *Realization*

   Hubungan antar-*class* dimana sebuah *class* memiliki keharusan untuk mengikuti aturan yang ditetapkan *class* lainnya. Biasanya *realization* digunakan untk menspesifikasikan hubungan antara sebuah *interface* dengan *class* yang mengimplementasikan *interface* tersebut.

2.10 Perangkat Lunak Pendukung

Adapun perangkat lunak pendukung yang digunakan dalam implementasi metode di aplikasi pendeteksi plasmodium malaria pada citra sampel darah sebagai berikut:

2.10.1 MATLAB

Matlab dikembangkan sebagai bahasa pemograman sekaligus alat visualisasi, yang menawarkan banyak kemampuan untuk menyelesaikan berbagai kasus yang berhubungan langsung dengan disiplin ilmu matematika. Matlab dibangun dari bahasa induknya yaitu bahasa C, namun tidak dapat dikatakan sebagai varian dari bahasa C, karena dalam sintak maupun cara kerjanya sama sekali berbeda dari bahasa C. Namun dengan hubungan langsungnya dengan bahasa C, Matlab memiliki kelebihan-kelebihan dari bahasa C bahkan mampu berjalan pada semua *platform* Sistem Operasi tanpa mengalami perubahan sintak sama sekali. Bahasa ini mengintegrasikan kemampuan komputasi, visualisasi dan pemograman dalam sebuah lingkungan yang tunggal dan mudah digunakan. Matlab memberikan sistem interaktif yang menggunakan konsep array/matrik sebagai standar variabel elemennya tanpa membutuhkan pendeklarasian array seperti pada bahasa lainnya.

Matlab dikembangkan oleh MathWorks, yang pada awalnya dibuat untuk memberikan kemudahan mengakses data matrik pada proyek LINPACK dan

EISPACK. Selanjutnya menjadi sebuah aplikasi untuk komputasi matrik. Dari sejak awal dipergunakan, matlab memperoleh masukan ribuan pemakai. Dalam lingkungan pendidikan ilmiah menjadi alat pemrograman standar bidang matematika, rekayasa dan keilmuan terkait. Dan dalam lingkungan industri dapat menjadi pilihan paling produktif untuk riset, pengembangan dan analisa[14].