BAB II
LANDASAN TEORI

2.1 *Typhoid fever*

Typhoid fever merupakan infeksi yang disebabkan oleh bakteri *Salmonella typhi*. *Salmonella typhi* adalah bakteri yang menyebabkan seseorang terkena demam tifoid atau *Typhoid fever*. Bakteri ini masuk ke dalam tubuh atau tepatnya ke saluran pencernaan melalui makanan atau minuman yang terkontaminasi. Orang sehat bisa saja tak menunjukkan gejala sakit *Typhoid fever*, meski ia sebenarnya menjadi *carrier* atau pembawa bakteri *Typhoid fever* di dalam tubuhnya. Angka kesakitan dan kematian penyakit ini masih cukup tinggi di seluruh dunia. Demam tifoid dapat menyerang siapa saja tanpa melihat umur, jenis kelamin, dan ras serta dapat menyerang kapan saja sepanjang tahun. Di Indonesia saja rata - rata terdapat 900 ribu kasus per tahun dengan lebih dari 20 ribu kematian. Hal ini terjadi terutama pada daerah dengan sanitasi yang rendah. Sistem kekebalan tubuh yang masih dalam pertumbuhan juga menjadi penyebab terpapar nya seseorang dengan penyakit *Typhoid fever*, karena itulah kebanyakan penderita *Typhoid fever* adalah dewasa muda atau anak – anak (Dr. Indra K. Muhtadi, 2011).

2.1.1 Gejala Demam *Typhoid*

Gejala demam *typhoid* biasanya berkembang 1-3 minggu setelah paparan dan bisa berkembang menjadi *typhoid tever* ringan maupun berat. *typhoid fever* biasanya disertai demam tinggi, rasa tidak enak badan, sakit kepala, sembelit atau diare, bintik merah di dada, dan pembesaran limpa dan hati (hellosehat.com). Gejala – gejala berikut biasa terjadi pada seseorang

yang telah terpapar penyakit *typhoid fever*:

1. Demam tinggi yang berkepanjangan terutama pada sore dan malam hari.
2. Penderita akan merasakan sakit kepala dan sakit perut.
3. Penderita akan mengalami perubahan warna lidah serta munculnya bercak – bercak kemerahan.
4. Terjadinya gangguan pencernaan, pada orang dewasa biasanya akan sulit buang air besar sedangkan pada anak- anak biasanya akan terjadi diare.
5. Pada keadaan yang berat penderita bertambah sakit dan kesadaran mulai menurun.

2.1.2 Diagnosa Penyakit *Typhoid fever*

1. Reaksi Widal adalah sebuah *test* yang dilakukan pada penderita *Typhoid fever*, *test* ini merupakan sebuah *test* imunitas yang di timbulkan oleh *Salmonela typhi*. Dikatakan meningkat bila Titer nya lebih dari 1/400 atau naik dua kali lipat dari titer sebelumnya dalam waktu 1 minggu (Waoli Lase, 2011).
2. Melakukan *test* darah pada laboratorium, biasanya pada penderita *Typhoid fever* ditemukan *leukositopeni* dan *trombositopeni (*Dr. Indra K. Muhtadi, 2011).
3. Ditemukannya bakteri *Salmonela typhi* pada *feses* atau *urine*

2.1.3 Pengobatan *Typhoid fever*

Pengobatan penyakit *Typhoid fever* dapat dilakukan melakukan terapi *antibiotic*. Terapi antibiotik adalah cara paling efektif dalam menangani *Typhoid fever* dan harus diberikan sesegera mungkin. Sampel darah, tinja, dan urine Anda akan diperiksa di laboratorium untuk menentukan jenis antibiotik yang tepat untuk diberikan.

2.2 Campak

Penyakit Campak adalah suatu infeksi virus yang sangat menular, yang ditandai dengan demam, batuk, konjungtivitis (peradangan selaput ikat mata/konjungtiva) dan ruam kulit. Penyakit ini disebabkan karena infeksi virus campak golongan *Paramixovirus*. Program imunisasi campak di Indonesia dimulai tahun 1982. Menurut data Departemen Kesehatan tahun 2015, Indonesia memiliki cakupan imunisasi campak kategori sedang di Asia Tenggara, yakni 84%. Indonesia berkomitmen untuk mencapai angka cakupan imunisasi campak sebesar 95% pada akhir tahun 2020. Hal ini dikarenakan campak termasuk dalam 10 besar penyebab kematian terbanyak pada balita di Indonesia.

Data dari Departemen Kesehatan Republik Indonesia menunjukkan bahwa angka kejadian campak telah menurun signifikan dari total 18.488 kasus pada akhir 2007 menjadi 8.185 kasus pada tahun 2015. Meski mengalami penurunan yang berarti, cakupan imunisasi campak tetap perlu diperluas hingga ke seluruh daerah di Indonesia, guna mencapai target Indonesia Bebas Campak pada tahun 2020. (Sumber: alodokter.com)

Virus campak ada di dalam percikan cairan yang dikeluarkan saat mereka bersin dan batuk. Virus campak akan menularkan siapa pun yang menghirup percikan cairan ini. Virus campak bisa bertahan di permukaan selama beberapa jam dam bisa bertahan menempel pada benda-benda lain. Saat kita menyentuh benda yang sudah terkena percikan virus campak lalu menempelkan tangan ke hidung atau mulut, kita bisa ikut terinfeksi. Campak lebih sering menimpa balita. Tapi pada dasarnya semua orang bisa terinfeksi virus ini, terutama yang belum pernah terkena campak atau yang belum mendapat vaksinasi campak.

2.2.1 Gejala Campak

Campak bisa sangat mengganggu dan mengarah pada komplikasi yang lebih serius. Gejala campak mulai muncul sekitar satu hingga dua minggu setelah virus masuk ke dalam tubuh. Gejala tersebut di antaranya adalah:

1. Panas badan / demam
2. Sakit tenggorokan
3. Gangguan pencernaan
4. Muncul bercak – bercak merah
5. Nyeri otot
6. Biasa terjadi pada anak - anak

2.2.2 Pengobatan Campak

Sistem kekebalan tubuh manusia secara alami akan melawan infeksi virus ini. Tapi jika komplikasi terjadi atau infeksi campak menjadi sangat

parah, perawatan di rumah sakit kemungkinan akan dibutuhkan. Untuk mempercepat proses pemulihan, terdapat beberapa hal yang bisa membantu:

1. Minum banyak air untuk mencegah dehidrasi.
2. Banyak istirahat dan hindari sinar matahari selama mata masih sensitif terhadap cahaya.
3. Minum obat penurun demam dan pereda sakit. Jangan berikan aspirin jika anak Anda berusia kurang dari 16 tahun.

2.3 *Dengue Fever*

Dengue fever atau demam berdarah *dengue* (disingkat DBD) adalah infeksi yang disebabkan oleh virus *dengue*. Nyamuk atau/ beberapa jenis nyamuk menularkan (atau menyebarkan) virus *dengue*. Demam *dengue* juga disebut sebagai "*break bone fever*" atau "*bone break fever*" (demam sendi), karena demam tersebut dapat menyebabkan penderita nya mengalami nyeri hebat seakan-akan tulang mereka patah. Terdapat empat jenis virus *dengue*. Apabila seseorang telah terinfeksi satu jenis virus, biasanya dia menjadi kebal terhadap jenis tersebut seumur hidupnya. Namun, dia hanya akan terlindung dari tiga jenis virus lainnya dalam waktu singkat. Jika kemudian dia terkena satu dari tiga jenis virus tersebut, dia mungkin akan mengalami masalah yang serius.

2.3.1 Gejala Demam Berdarah *Dengue*

Berikut ini beberapa gejala demam *dengue*, di antaranya:

1. Suhu badan tinggi yang bisa mencapai 41 derajat celcius
2. Tubuh menggigil

3. Kehilangan nafsu makan
4. Lemas
5. Sakit kepala & tenggorokan
6. Mual-mual & muntah
7. Munculnya bercak - bercak merah di kulit (terutama pada anak-anak)

Pada kasus yang jarang terjadi, demam *dengue* juga menyebabkan hidung dan gusi mengeluarkan darah yang jumlahnya sangat sedikit (berbeda dengan pendarahan yang terjadi pada *hemorrhagic dengue fever* yang mana volume darah yang dikeluarkan cukup banyak) (Sumber: alodokter.com).

2.3.2 Pengobatan Demam Berdarah *Dengue*

Jika kita bisa mengenali gejala penyakit demam *dengue* tanpa bantuan dari dokter, maka pengobatan demam *dengue* cukup mudah dilakukan di samping memang tidak adanya obat khusus yang diperuntukkan untuk penyakit ini. Namun yang menjadi masalah adalah kita bukan seorang dokter. Dokter sendiri pun masih membutuhkan pemeriksaan lebih lanjut, misalnya tes darah, untuk melihat keberadaan virus *dengue* di dalam darah atau memastikan apakah gejala yang ada memang disebabkan oleh demam *dengue* dan bukan kondisi lain. Apabila gejala yang kita alami sudah dipastikan akibat demam *dengue*, maka saran pengobatan yang umumnya diberikan oleh dokter adalah:

1. Banyak beristirahat.

2. Minum banyak cairan untuk mencegah dehidrasi (terutama untuk mengganti cairan tubuh yang terbuang akibat gejala demam tinggi dan muntah-muntah).
3. Mengonsumsi Parasetamol dan acetaminophen untuk meredakan demam dan nyeri.
4. Berhenti menjalani aktivitas untuk sementara waktu sampai tubuh benar-benar pulih.

2.4 *Android*

Android adalah sebuah sistem operasi berbasis Linux yang digunakan pada perangkat *mobile*. *Android* awalnya dikembangkan oleh *Android*, Inc., dengan dukungan finansial dari Googsle, yang kemudian membelinya pada tahun 2005. Sistem operasi ini dirilis secara resmi pada tahun 2007, bersamaan dengan didirikannya *Open Handset Alliance*, konsorsium dari perusahaan-perusahaan perangkat keras, perangkat lunak, dan telekomunikasi yang bertujuan untuk memajukan standar terbuka perangkat seluler. *Android* memiliki empat karakteristik seperti berikut:

1. ***Android* merupakan sistem operasi *open source***

   Android bersifat terbuka sehingga aplikasi dalam *Android* dapat mengambil alih suatu fitur dasar ponsel pada umumnya seperti *messaging*, *phone call*, menggunakan kamera dan bahkan *GPS*. *Android* menggunakan sebuah mesin *virtual* yang dirancang khusus untuk mengoptimalkan sumber daya memori dan perangkat keras yang terdapat di dalam perangkat.

2. **Semua aplikasi dibuat sama**

Android tidak memberikan perbedaan terhadap aplikasi utama dari telepon dan aplikasi pihak ketiga (*third-party application*). Semua aplikasi dapat dibangun untuk memiliki akses yang sama terhadap kemampuan sebuah telepon dalam menyediakan layanan dan aplikasi yang luas terhadap para pengguna.

3. **Pengembangan aplikasi yang cepat dan mudah**

   Android menyediakan akses yang sangat luas kepada pengguna untuk menggunakan *library* yang diperlukan dan *tools* yang dapat digunakan untuk membangun aplikasi yang semakin baik. *Android* memiliki *tools* yang dapat digunakan oleh developer untuk membantu proses dalam merancang sebuah aplikasi. (sumber: http://www.*Android*.com).

4. **Bukanlah bahasa program**

   Android bukanlah bahasa program, melainkan sebuah *runtime environment* yang disebut DVM (*Davik Virtual Machine*). DVM inilah yang membuat sistem operasi ini dapat berjalan pada *device* dengan memori yang terbilang kecil.

2.4.1 Fitur Sistem Operasi *Android*

1. ***Application Framework***

   Application framework digunakan dalam menulis aplikasi dalam sistem operasi *Android*. Tidak seperti lingkungan sistem operasi *mobile* yang tertanam lainnya, aplikasi *Android* semua sama. Misalnya, sebuah aplikasi yang memiliki fungsi yang sama dengan telepon tidak berbeda dari fungsi telepon dari *developer* lainnya. Kerangka kerja ini didukung oleh berbagai

open source library seperti *OpenSSL*, *SQLite* dan *libc*. Hal ini juga didukung oleh *library* utama *Android*. Dari sudut keamanan, kerangka didasarkan pada hak akses *file* sistem UNIX yang menjamin aplikasi memiliki kemampuan sesuai dengan apa yang pemilik ponsel berikan pada waktu menginstal aplikasi tersebut.

2. *Dalvik Virtual Machine (DVM)*

Dalvik Virtual Machine adalah sebuah mesin virtual yang menggunakan sangat sedikit memori dan memang dirancang untuk sistem operasi *Android* untuk dijalankan pada *embedded system*. *Virtual machine* ini bekerja sangat baik dalam keadaan *low power* dan mengoptimalkan perangkat *mobile*. DVM juga mengatur atribut dari *Central Processing Unit* (CPU) serta membuat sebuah format *file* yang spesial (.DEX) yang dibuat selama *build time post processing*. DVM mengambil *file* yang dihasilkan oleh *class Java* dan menggabungkannya ke dalam satu atau lebih *Dalvik Executable* (.dex). DVM dapat menggunakan kembali salinan informasi dari beberapa *class file* dan secara efektif mengurangi kebutuhan penyimpanan oleh setengah dari Java *Archive* (.jar) *File* tradisional. Konversi antara kelas Java dan format (.dex) dilakukan dengan memasukkan "*dx tool*" (sumber: https://www.scribd.com/).

2.4.2 Arsitektur Sistem Operasi *Android*

Sistem Operasi *Android* memiliki komponen utama sebagai berikut:

1. Aplikasi

Android berisi sekumpulan aplikasi utama seperti: *email client, program Short Message Service* (SMS), kalender, peta, *browser,* daftar kontak, dan lain-lain. Semua aplikasi ditulis dengan menggunakan bahasa pemrograman Java.

2. Kerangka Aplikasi

Kerangka kerja aplikasi yang ditulis dengan menggunakan bahasa pemrograman Java merupakan peralatan yang digunakan oleh semua aplikasi, baik aplikasi bawaan dari ponsel seperti daftar kontak, dan kotak *SMS*, maupun aplikasi yang ditulis oleh Google ataupun pengembang *Android*. *Android* menawarkan para pengembang kemampuan untuk membangun aplikasi yang inovatif. Pengembang bebas untuk mengambil keuntungan dari perangkat keras, akses lokasi informasi, menjalankan *background services* mengatur alarm, menambahkan peringatan ke status bar, dan masih banyak lagi. Pengembang memiliki akses yang penuh ke dalam kerangka kerja *API* yang sama yang digunakan oleh aplikasi utama. Pada dasarnya, kerangka kerja aplikasi memiliki beberapa komponen sebagai berikut:

a) *Activity Manager*

Mengatur siklus dari aplikasi dan menyediakan navigasi *back stack* untuk aplikasi yang berjalan pada proses yang berbeda.

b) *Package Manager*

Untuk melacak aplikasi yang di-instal pada perangkat.

c) *Windows Manager*

Merupakan abstraksi dari bahasa pemrograman Java pada bagian atas dari *level services* (pada level yang lebih rendah) yang disediakan oleh *Surface Manager*.

d) *Telephony Manager*

Berisikan kumpulan *API* untuk memanggil aplikasi.

e) *Content Provider*

Digunakan untuk memungkinkan aplikasi mengakses data dari aplikasi lain atau untuk membagikan data mereka sendiri.

f) *Resource Manager*

Digunakan untuk mengakses sumber daya yang bersifat bukan *code* seperti *string* lokal, *bitmap*, deskripsi dari *layout file* dan bagian eksternal lain dari aplikasi.

g) *View System*

Digunakan untuk mengambil sekumpulan *button*, *list*, *grid*, dan *text box* yang digunakan di dalam antarmuka pengguna.

h) *Notification Manager*

Digunakan untuk mengatur tampilan peringatan dan fungsi-fungsi lain.

3. *Library*

Gambar 2.1. *Android Library* (Sumber: insinyoer.com)

Library membawa sekumpulan instruksi untuk mengarahkan perangkat *Android* dalam menangani berbagai tipe data. Contohnya, perekam dari berbagai macam format *video* dan *audio* ditangani oleh *Media Framework Library*. Berikut adalah beberapa kegunaan *library*:

a) *Surface Manager*: Berguna untuk mengolah tampilan windows pada layar.

b) SGL: SGL mendasari mesin grafis 2D dan bekerja bersama-sama dengan lapisan pada level yang lebih tinggi dari kerangka kerja (seperti *Windows Manager* dan *Surface Manager*) untuk mengimplementasikan keseluruhan *graphics pipeline* dari *Android*.

c) Open GL|ES : Berguna untuk menampilkan grafik 3 dimensi maupun 2 dimensi.

d) *Media Framework* : Menunjang perekaman dari berbagai macam format *audio, video,* dan gambar *Media Framework*. Menunjang perekaman dari berbagai macam format *audio, video,* dan gambar.

e) *Free Type*: Berfungsi sebagai penerjemah *font*.

f) *Web Kit*: Merupakan *web browser* modern yang menjadi kekuatan bagi *browser Android* dan sebuah *embeddable web view*.

g) *System C Libraries*: Merupakan implementasi turunan dari standar system *library* C (libc) yang diatur untuk peralatan berbasis *embedded* Linux.

h) SQLite: Merupakan *relational database* yang kuat dan ringan serta tersedia untuk semua aplikasi.

i) Open SSL: Sebagai pengamanan jaringan/*security*.

4. *Android Runtime*

Merupakan lokasi dimana komponen utama dari DVM ditempatkan. DVM dirancang secara khusus untuk *Android* pada saat dijalankan pada lingkungan yang terbatas, dimana baterai yang terbatas, CPU, memori, dan penyimpanan data menjadi fokus utama. *Android* memiliki sebuah *tool* yang terintegrasi yaitu "dx" yang mengkonversi *generated byte code* dari (.JAR) ke dalam *file* (.DEX) sehingga *byte code* menjadi lebih efisien untuk dijalankan pada prosesor yang kecil. Hal ini memungkinkan untuk memiliki beberapa jenis dari DVM berjalan.

2.5 *Android studio*

Android studio adalah Lingkungan Pengembangan Terpadu - *Integrated Development Environment* (IDE) untuk pengembangan aplikasi *Android*, berdasarkan IntelliJ IDEA(Sumber : developer.*Android*.com). Selain merupakan editor kode IntelliJ dan alat pengembang yang berdaya guna, *Android studio*

menawarkan fitur lebih banyak untuk meningkatkan produktivitas saat membuat aplikasi *Android*, misalnya:

1) Sistem pembuatan berbasis *Gradle* yang fleksibel
2) *Emulator* yang cepat dan kaya fitur
3) Lingkungan yang menyatu untuk pengembangan bagi semua perangkat *Android*
4) *Instant Run* untuk mendorong perubahan ke aplikasi yang berjalan tanpa membuat APK baru
5) *Template* kode dan integrasi GitHub untuk membuat fitur aplikasi yang sama dan mengimpor kode contoh
6) Alat penguji dan kerangka kerja yang ekstensif
7) Alat Lint untuk meningkatkan kinerja, kegunaan, kompatibilitas versi, dan masalah-masalah lain
8) Dukungan C++ dan NDK
9) Dukungan bawaan untuk *Google Cloud Platform*, mempermudah pengintegrasian *Google Cloud Messaging* dan *App Engine*

Android studio menggunakan Gradle untuk manajemen proyek nya. Secara default Gradle adalah build *tools* yang digunakan pada *Android studio*, untuk men compile-menjalankan project aplikasi yang sedang kalian kembangkan, berdasarkan settingan atau konfigurasi yang berada di bagian Gradle *Script*. Jika di Eclipse *build tools* yang digunakan adalah Ant, Maka Gradle merupakan pengembangan yang di bangun dari Ant, Maven, dan repositori Ivy. *Android* Plugin untuk Gradle berjalan secara *independent* (bebas) di *Android studio*. Yang berarti

kalian bisa membangun Aplikasi *Android* di dalam *Android studio*, serta kalian juga dapat membangun lewat *Command Line* menggunakan Gradle. Serta mempunyai fitur yang spesifikasinya diperuntukkan untuk Aplikasi *Android*, menentukan *build* tipe, *flavors*, *Signing configurations*, menambahkan *library project*, dan masih banyak lagi.

2.6 Sistem Pakar

Sistem pakar merupakan suatu program komputer berbasis pengetahuan yang berusaha mengadopsi pengetahuan seorang pakar ke komputer, agar komputer dapat menyelesaikan masalah seperti yang dilakukan oleh pakar. Dengan adanya sistem pakar maka orang awam pun dapat menyelesaikan masalah atau untuk mencari tahu informasi yang akurat mengenai masalah tersebut. Sistem ini seperti halnya seorang pakar hanya terfokus pada suatu masalah yang spesifik.

Sistem pakar dibuat dan dikembangkan untuk mempermudah *user* atau pengguna komputer, agar mampu memahami berbagai macam hal yang ingin diketahui, namun *user* tidak memiliki akses langsung terhadap pakar atau ahli yang memahami tentang keingintahuan nya. Sistem pakar sengaja dibuat dan dikembangkan dengan cara mengadopsi pola pikir dan pengetahuan manusia (yang dalam hal ini adalah seorang *expert* atau pakar), yang ditujukan untuk mencari sebuah atau beberapa buah solusi yang memuaskan *user*-nya seperti ketika seorang pakar atau ahli memberikan penjelasan kepada murid atau penanyaannya.

2.6.1 Prinsip Kerja Sistem Pakar

Secara umum, sebuah sistem pakar itu sendiri memiliki sebuah prinsip kerja yang sangat sederhana. Para pengembang atau developer

mengumpulkan berbagai macam pandangan dan juga hasil penelitian dan juga presentasi yang sering dilakukan oleh para ahli dalam bidang tertentu. Setelah itu para developer kemudian membuat intisari dari seluruh hasil data yang diperoleh tersebut, yang kemudian dimasukkan ke dalam sebuah sistem yang dijadikan program komputer. Secara umum, sebuah sistem pakar itu sendiri memiliki sebuah prinsip kerja yang sangat sederhana. Para pengembang atau developer mengumpulkan berbagai macam pandangan dan juga hasil penelitian dan juga presentasi yang sering dilakukan oleh para ahli dalam bidang tertentu. Setelah itu para developer kemudian membuat intisari dari seluruh hasil data yang diperoleh tersebut, yang kemudian dimasukkan ke dalam sebuah sistem yang dijadikan program komputer.

2.6.2 Kelebihan dan Kekurangan Sistem Pakar

Sistem pakar sendiri memilik banyak sekali kelebihan dan juga keunggulan, berikut ini adalah beberapa kelebihan dan juga keunggulan dari sistem pakar :

1. *User* tetap bisa memperoleh informasi yang *valid* dan juga *reliable*, meskipun tidak mengenal dan memiliki *channel* pakar – pakar yang dapat memecahkan masalahnya
2. *User* tidak perlu repot berkonsultasi secara tatap muka dengan pakar atau ahli
3. Sistem pakar mengintegrasikan pandangan dan juga pendapat dari satu atau lebih pakar, sehingga dapat diramu secara lebih objektif dan menghasilkan sebuah *output* yang lebih *detail* dan juga mendalam

4. Sistem pakar mampu memberikan *respond* dan juga jawaban yang lebih cepat, tidak seperti ketika kita bertanya langsung ke pakar, yang biasanya harus mengobrol terlebih dahulu
5. Sangat berguna untuk membantu menyelesaikan berbagai macam masalah yang dialami oleh *user*
6. Dapat menambah informasi bagi banyak *user*
7. Dapat digunakan untuk mengakses *database* secara cerdas
8. Meningkatkan minat masyarakat untuk lebih tahu banyak dan mau belajar banyak.

Meskipun dinilai memiliki banyak sekali kelebihan, terutama bagi *user*nya, namun demikian ternyata sistem pakar juga memiliki beberapa kekurangan. Berikut ini adalah beberapa kekurangan dari sistem pakar :

1. Biaya pengembangan yang cukup tinggi
2. Waktu pengembangan sistem pakar yang cenderung lama
3. Laju ilmu pengetahuan yang cepat, sehingga membutuhkan pembaruan atau *update* sistem pakar dalam jangka waktu tertentu
4. Meskipun dinilai *reliable* dan juga *valid*, namun sistem pakar tidak 100% dapat menjamin keberhasilannya, karena adanya faktor – faktor lainnya yang dimiliki oleh *user*
5. Untuk dapat mengintegrasikan pandangan dan juga pemikiran dari pakar yang ada, membutuhkan waktu yang cukup lama dan juga sulit, karena terkadang tiap – tiap pakar memiliki pandangan dan juga pendapat yang berbeda 180 derajat

6. *User* mungkin tidak akan tertarik untuk membeli aplikasi seperti ini, karena harganya yang mahal, dan hanya berlaku untuk kebutuhan spesifik saja.

2.6.3 Karakteristik Sistem Pakar

Turban (1995) mengatakan bahwa sistem pakar sendiri terdiri dari beberapa karakteristik yang melekat. Berikut ini adalah beberapa karakteristik dari sistem pakar:

1. *Domain* tertentu, merupakan kekhususan dari sebuah sistem pakar dalam membidangi suatu disiplin ilmu tertentu.
2. Memiliki kemampuan untuk mengolah data yang memiliki ketidakpastian, dan mampu memberikan semacam pertimbangan, saran dan juga anjuran sesuai dengan kondisi dari lingkungan.
3. Dirancang untuk dapat dikembangkan secara bertahap.
4. Memiliki nilai kepakaran, yang mampu membantu *user* dalam menyelesaikan tugas dan juga membantu memecahkan masalah dengan memberikan solusi.

2.7 Model *Prototype*

Prototyping perangkat lunak (*software prototyping*) atau siklus hidup menggunakan *prototyping* (*life cycle using prototyping*) adalah salah satu metode siklus hidup sistem yang didasarkan pada konsep model bekerja (*working* model). Tujuannya adalah mengembangkan model menjadi sistem *final*. Artinya sistem akan dikembangkan lebih cepat dari pada metode tradisional dan biayanya menjadi lebih rendah. Ada banyak cara untuk mem-*prototyping*, begitu pula dengan

penggunaannya. Ciri khas dari metodologi ini adalah pengembang sistem (*system* developer), klien, dan pengguna dapat melihat dan melakukan eksperimen dengan bagian dari sistem komputer dari sejak awal proses pengembangan.

Prototyping membantu dalam menemukan kebutuhan di tahap awal pengembangan, terutama jika klien tidak yakin dimana masalah berasal. Selain itu *prototyping* juga berguna sebagai alat untuk mendesain dan memperbaiki *user interface* – bagaimana sistem akan terlihat oleh orang-orang yang menggunakannya. (Sumber: academia.edu)

2.7.1 Tahapan dalam model *prototype*

Berikut adalah gambar tahapan atau fase *prototype* secara *detail*:

1. **Pengumpulan Kebutuhan**: *User* dan developer bersama-sama mendefinisikan format seluruh perangkat lunak, mengidentifikasikan semua kebutuhan yang di butuhkan untuk merancang sistem, dan garis besar sistem yang Akan dibuat.

2. **Membangun *Prototyping*:** Membangun *prototyping* dengan membuat perancangan sementara yang berfokus pada penyajian kepada *user* (misalnya dengan membuat *input* dan format *output*).

3. **Evaluasi *Prototyping***: Evaluasi ini dilakukan oleh *user* apakah *prototyping* yang sudah dibangun sudah sesuai dengan keinginan *user*. Jika sudah sesuai maka langkah selanjutnya akan diambil. Jika tidak *prototyping* direvisi dengan mengulang langkah 1, 2, dan 3.

4. **Mengkodekan Sistem**: Dalam tahap ini *prototyping* yang sudah di sepakati diterjemahkan ke dalam bahasa pemrograman yang sudah di setujui.

5. **Menguji Sistem**: Setelah sistem sudah menjadi suatu perangkat lunak yang siap pakai, testing harus dilakukan terlebih dahulu pada sistem yang telah selesai sebelum digunakan. Pengujian ini dilakukan dengan *White Box*, *Black Box*, *Basis Path*, pengujian arsitektur dan lain-lain.

6. **Evaluasi Sistem**: *User* akan mengevaluasi apakah sistem yang sudah jadi sudah sesuai dengan yang diharapkan. Jika ya, langkah selanjutnya akan dilakukan; jika tidak proses perancangan sistem harus mengulangi langkah 4 dan 5.

7. **Penggunaan Sistem**: Perangkat lunak yang telah diuji dan diterima oleh *user* dapat langsung digunakan. (Sumber : ilearning.me)

2.8 Java

Java adalah bahasa pemrograman yang dapat dijalankan di berbagai komputer termasuk telepon genggam. Bahasa ini awalnya dibuat oleh James Gosling saat masih bergabung di Sun Microsystems saat ini merupakan bagian dari Oracle dan dirilis tahun 1995. Bahasa ini banyak mengadopsi sintaksis yang terdapat pada C dan C++ namun dengan sintaksis model objek yang lebih sederhana serta dukungan rutin-rutin atas bawah yang minimal. Aplikasi-aplikasi berbasis java umumnya dikompilasi ke dalam p-code (*bytecode*) dan dapat dijalankan pada

berbagai Mesin *Virtual* Java (JVM). Java merupakan bahasa pemrograman yang bersifat umum/non-spesifik (*general purpose*), dan secara khusus didisain untuk memanfaatkan dependensi implementasi seminimal mungkin. Karena fungsionalitasnya yang memungkinkan aplikasi java mampu berjalan di beberapa *platform* sistem operasi yang berbeda, java dikenal pula dengan slogannya, **"Tulis sekali, jalankan di mana pun".** Saat ini java merupakan bahasa pemrograman yang paling populer digunakan, dan secara luas dimanfaatkan dalam pengembangan berbagai jenis perangkat lunak aplikasi ataupun aplikasi (Sumber: Wikipedia.co.id).

2.8.1 Kelebihan dan Kekurangan Java

Java memiliki beberapa kelebihan jika dibandingkan dengan bahasa pemrograman lainnya, antara lain:

1) Multiplatform

Kelebihan utama dari Java ialah dapat dijalankan di beberapa *platform* / sistem operasi komputer, sesuai dengan prinsip tulis sekali, jalankan di mana saja. Dengan kelebihan ini pemrogram cukup menulis sebuah program Java dan dikompilasi (diubah, dari bahasa yang dimengerti manusia menjadi bahasa mesin / *bytecode*) sekali lalu hasilnya dapat dijalankan di atas beberapa platform tanpa perubahan. Kelebihan ini memungkinkan sebuah program berbasis java dikerjakan di atas *operating system* Linux tetapi dijalankan dengan baik di atas Microsoft Windows. Platform yang didukung sampai saat ini adalah Microsoft Windows, Linux, Mac OS dan Sun Solaris. Penyebabnya adalah setiap sistem operasi

menggunakan programnya sendiri-sendiri (yang dapat diunduh dari situs Java) untuk meninterpretasikan *bytecode* tersebut.

2) Pemrograman Berorientasi Objek

Pemrograman berorientasi objek (Inggris: *object-oriented programming* disingkat OOP) merupakan paradigma pemrograman yang berorientasi kan kepada objek. Semua data dan fungsi di dalam paradigma ini di bungkus dalam kelas-kelas atau objek-objek. Bandingkan dengan logika pemrograman terstruktur. Setiap objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya, model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam teknik peranti lunak skala besar. Lebih jauh lagi, pendukung OOP mengklaim bahwa OOP lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

3) *Complete Library*

Java terkenal dengan kelengkapan *library*/perpustakaan (kumpulan program program yang disertakan dalam pemrograman java) yang sangat memudahkan dalam penggunaan oleh para pemrogram untuk membangun aplikasinya. Kelengkapan perpustakaan ini ditambah dengan keberadaan komunitas Java yang besar yang terus menerus membuat perpustakaan-perpustakaan baru untuk melingkupi seluruh kebutuhan pembangunan aplikasi.

4) Pengumpulan Sampah Otomatis

Java memiliki fasilitas pengaturan penggunaan memori sehingga para pemrogram tidak perlu melakukan pengaturan memori secara langsung (seperti halnya dalam bahasa C++ yang dipakai secara luas).

5) Bahasa Program Mirip C++

Java memiliki sintaks seperti bahasa pemrograman C++ sehingga menarik banyak pemrogram C++ untuk pindah ke Java. Saat ini pengguna Java sangat banyak, sebagian besar adalah pemrogram C++ yang pindah ke Java. Universitas-universitas di Amerika Serikat juga mulai berpindah dengan mengajarkan Java kepada murid-murid yang baru karena lebih mudah dipahami oleh murid dan dapat berguna juga bagi mereka yang bukan mengambil jurusan computer (Sumber: Wikipedia.co.id).

```
// Outputs "Hello, world!" and then exits
public class HelloWorld {
   public static void main(String args[]) {
       System.out.println("Hello, world!");
   }
}
```

Gambar 2.2. Contoh Program Java (Sumber: Wikipedia.co.id)

Walaupun memiliki banyak kelebihan, java juga memiliki kekurangan, berikut adalah beberapa kekurangan dalam bahasa pemrograman java:

1) Tulis sekali, jalankan di mana saja

Masih ada beberapa hal yang tidak kompatibel antara *platform* satu dengan *platform* lain. Untuk J2SE, misalnya *SWT-AWT bridge* yang sampai sekarang tidak berfungsi pada Mac OS X.

2) Mudah didekompilasi

Dekompilasi adalah proses membalikkan dari kode jadi menjadi kode sumber. Ini dimungkinkan karena kode jadi Java merupakan *bytecode* yang menyimpan banyak atribut bahasa tingkat tinggi, seperti nama-nama kelas, metode, dan tipe data. Hal yang sama juga terjadi pada Microsoft .NET Platform. Dengan demikian, algoritma yang digunakan program akan lebih sulit disembunyikan dan mudah dibajak/di*reverse-engineer*.

3) Penggunaan memori yang banyak

Penggunaan memori untuk program berbasis Java jauh lebih besar daripada bahasa tingkat tinggi generasi sebelumnya seperti C/C++ dan Pascal (lebih spesifik lagi, Delphi dan *Object* Pascal). Biasanya ini bukan merupakan masalah bagi pihak yang menggunakan teknologi terbaru (karena trend memori terpasang makin murah), tetapi menjadi masalah bagi mereka yang masih harus berkutat dengan mesin komputer berumur lebih dari 4 tahun (sumber : Wikipedia.co.id).

2.9 *Dempster Shafer*

Teori Dempster-Shafer adalah suatu teori matematika untuk pembuktian berdasarkan *belief functions* and *plausible reasoning* (fungsi kepercayaan dan pemikiran yang masuk akal), yang digunakan untuk mengkombinasikan potongan informasi yang terpisah (bukti) untuk mengkalkulasi kemungkinan dari suatu peristiwa. Teori ini dikembangkan oleh Arthur P. Dempster dan Glenn Shafer. Secara umum Teori Dempster-Shafer ditulis dalam suatu interval:

[*Belief, Plausibility*]

Belief (Bel) adalah ukuran kekuatan *evidence* dalam mendukung suatu himpunan proposisi. Jika bernilai 0 mengindikasikan bahwa tidak ada *evidence*, dan *Plausibility* (Pl) jika bernilai 1 menunjukkan adanya kepastian. *Plausibility* dinotasikan sebagai :

Pl(s) = 1 – Bel(Øs)

Menurut Giarratano dan Riley fungsi *Belief* dapat diformulasikan dan ditunjukkan pada persamaan (1):

$$Bel\ (X) = \sum_{Y \subseteq X} m(Y)$$

Equation 2.1. Persamaan pertama *Dempster Shafer*

Dan *Plausibility* dinotasikan pada persamaan (2):

$$Pls(X) = 1 - Bel(X) = 1 - \sum_{Y \subseteq X} m(X)$$

Equation 2.2. Persamaan kedua metode *Demspter Shafer*

Dimana :

1. Bel (X) = *Belief* (X)
2. Pls (X) = *Plausibility* (X)
3. m (X) = *mass function* dari (X)
4. m (Y) = *mass function* dari (Y)

Teori Dempster-Shafer menyatakan adanya *frame of discrement* yang dinotasikan dengan simbol (Θ). *frame of discrement* merupakan

semesta pembicaraan dari sekumpulan hipotesis sehingga sering disebut dengan *environment* yang ditunjukkan pada persamaan (3) :

$$\Theta = \{ \theta 1, \theta 2, \ldots \theta N\}$$

Dimana :

1. Θ = *frame of discrement* atau *environment*
2. θ1,…,θN = *element*/ unsur bagian dalam *environment*

Environment mengandung elemen-elemen yang menggambarkan kemungkinan sebagai jawaban, dan hanya ada satu yang akan sesuai dengan jawaban yang dibutuhkan. Kemungkinan ini dalam teori Dempster-Shafer disebut dengan *power set* dan dinotasikan dengan P (Θ), setiap elemen dalam *power set* ini memiliki nilai interval antara 0 sampai 1.

$$\mathbf{m : P(\Theta) \rightarrow [0,1]}$$

Sehingga dapat dirumuskan pada persamaan (4) :

$$\sum_{X \in P(\theta)} m(X) = 1$$

Equation 2.3. Persamaan keempat metode *Demspter Shafer*

Dengan :

P(Θ) = *power set*

m (X) = *mass function* (X)

Mass function (m) dalam teori Dempster-shafer adalah tingkat kepercayaan dari suatu *evidence* (gejala), sering disebut dengan *evidence measure* sehingga dinotasikan dengan (m). Tujuannya adalah mengaitkan

ukuran kepercayaan elemen-elemen θ. Tidak semua *evidence* secara langsung mendukung tiap-tiap elemen. Untuk itu perlu adanya probabilitas fungsi densitas (m). Nilai m tidak hanya mendefinisikan elemen-elemen θ saja, namun juga semua subsetnya.

Sehingga jika θ berisi n elemen, maka subset θ adalah 2*n*. Jumlah semua m dalam subset θ sama dengan 1. Apabila tidak ada informasi apapun untuk memilih hipotesis, maka nilai :

m{θ} = 1,0

Apabila diketahui X adalah subset dari θ, dengan m1 sebagai fungsi densitasnya, dan Y juga merupakan subset dari θ dengan m2 sebagai fungsi densitasnya, maka dapat dibentuk fungsi kombinasi m1 dan m2 sebagai m3, yaitu ditunjukkan pada persamaan (5) :

$$m3(Z) = \frac{\sum_{x \cap y = z} m1(X).m2(Y)}{1 - \sum_{x \cap y = \emptyset} m1(X).m2(Y)}$$

Equation 2.4. Persamaan kelima metode *Demspter Shafer*

Dimana :

m3(Z) = *mass function* dari *evidence* (Z)

m1 (X) = *mass function* dari *evidence* (X), yang diperoleh dari nilai keyakinan suatu *evidence* dikalikan dengan nilai *disbelief* dari *evidence* tersebut.

m2 (Y) = *mass function* dari *evidence* (Y), yang diperoleh dari nilai keyakinan suatu *evidence* dikalikan dengan nilai *disbelief* dari *evidence* tersebut.

$$\sum_{x \cap y = z} m1\,(X).m2(Y)$$

Equation 2.5. Nilai kekuatan *evidence*

Persamaan 2.4 Merupakan nilai kekuatan dari *evidence* Z yang diperoleh dari kombinasi nilai keyakinan sekumpulan *evidence*.

2.10 *Class* Diagram

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).

Class adalah kumpulan objek-objek yang mempunyai struktur umum, *behavior* umum, relasi umum, dan semantic/kata yang umum. *Class* ini sebenarnya merupakan *blueprint* dari sebuah objek. Dengan mendeklarasikan suatu class, maka kita telah mendeklarasikan suatu tipe data baru (tipe data referensi).

Class diagram menggambarkan struktur dan deskripsi class, *package* dan objek beserta hubungan satu sama lain seperti containment, pewarisan, asosiasi, dan lain-lain.

Class memiliki tiga area pokok yaitu nama (*stereotype*) , atribut dan metoda. Atribut dan metoda dapat memiliki salah satu sifat berikut:

1) *Private* : tidak dapat dipanggil dari luar *class* yang bersangkutan.

2) *Protected* : hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisi nya.

3) *Public* : dapat dipanggil oleh siapa saja.

Class dapat merupakan implementasi dari sebuah interface, yaitu class abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class*. Dengan demikian *interface* mendukung resolusi metoda pada saat run-*time* (sumber: ilearning.me).

2.10.1 Hubungan Antar *Class*

1) Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan class yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah query antar *class*.

2) Agregasi, yaitu hubungan yang menyatakan bagian.

3) Pewarisan, yaitu hubungan hierarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan Fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisi nya. Kebalikan dari pewarisan adalah generalisasi.

4) Hubungan dinamis, yaitu rangkaian pesan yang di passing dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence* diagram.

2.11 UML

Unified Modeling Language (UML) adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat *tool* untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar *OOP* sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi *OOP*. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM.

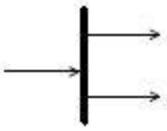
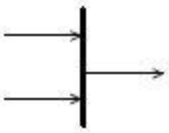
UML adalah suatu bahasa yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi. UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson. Namun demikian UML dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi. Penggunaan UML dalam industri terus meningkat. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri peranti lunak dan pengembangan sistem. Secara khusus, Unified Modeling Language (UML) menspesifikasikan langkah-langkah penting dalam pengambilan keputusan analisis, perancangan serta implementasi dalam sistem yang sangat bernuansa perangkat lunak (software intensive system).

Dalam hal ini, Unified Modeling Language (UML) bukanlah merupakan bahasa pemprograman tetapi model-model yang tercipta berhubungan langsung dengan berbagai macam bahasa pemprograman, sehingga adalah mungkin melakukan pemetaan (mapping) langsung dari model-model yang dibuat dengan Unified Modeling Language (UML) dengan bahasa-bahasa pemprograman berorientasi obyek, seperti Java, Borland Delphi, Visual Basic, C++, dan lain-lain.

Pemetaan (mapping) Unified Modeling Language (UML) bersifat dua arah yang pertama yaitu generasi kode bahasa pemprograman tertentu dari Unified Modeling Language (UML) forward engineering dan yang kedua generasi kode belum sesuai dengan kebutuhan dan harapan pengguna, pengembang dapat melakukan langkah balik bersifat iterative dari implementasi ke Unified Modeling Language (UML) hingga didapat sistem/peranti lunak yang sesuai dengan harapan pengguna dan pengembang. (Sumber: http://informatika.web.id)

2.12 *Activity Diagram*

Activity Diagram adalah sebuah diagram yang menggambarkan tentang aktifitas yang terjadi pada sistem. Dari pertama sampai akhir, diagram ini menunjukkan langkah – langkah dalam proses kerja sistem yang kita buat. Berikut adalah *element – element* pada *Activity Diagram*:

Simbol	Keterangan
	Start Point
	End Point
	Activities
	Fork (Percabangan)
	Join (Penggabungan)
	Decision
Swimlane	Sebuah cara untuk mengelompokkan activity berdasarkan Actor (mengelompokkan activity dalam sebuah urutan yang sama)

Gambar 2.3. *Element Activity Diagram* (Sumber: codepolitan.com)

2.13 *Class* Diagram

Class diagram adalah diagram yang menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. kelas memiliki 3 bagian utama yaitu attribute, operation, dan name. kelas-kelas yang ada pada struktur sistem harus dapat melakukan fungsi-fungsi sesuai dengan kebutuhan sistem. Sususan struktur kelas ynag baik pada diagram kelas sebaiknya memiliki jenis-jenis kelas berikut :

- Kelas Main. Yaitu kelas yang memiliki fungsi awal dieksekusi ketika sistem dijalankan.

- Kelas Interface. Kelas yang mendefinisikan dan mengatur tampilan ke pemakai. Biasanya juga disebut kelas *boundaries*.
- Kelas yang diambil dari pendefinisian usecase. Merupakan kelas yang menangani fungsi-fungsi yang harus ada dan diambil dari pendefinisian usecase.
- Kelas Entitas. Merupakan kelas yang digunakan untuk memegang atau membungkus data menjadi sebuah kesatuan yang diambil maupun akan disimpan ke basis data.

Object adalah gambaran dari entity, baik dunia nyata atau konsep dengan batasan-batasan yang tepat. *Object* bisa mewakili sesuatu yang nyata dalam domain problem kita seperti komputer, barang, konsumen, dapat berupa konsep seperti proses penarikan uang, pembayaran, pengembalian buku dan lain-lain. Dari *object - object* ini kita bisa mengabstraksikan *candidate class* yang mungkin terlibat.

Class adalah deskripsi sekelompok *object* dari *property* (atribut), sifat (operasi), relasi antar *object* dan semantik yang umum. Setiap *object* merupakan contoh dari beberapa *class* dan *object* tidak dapat menjadi contoh lebih dari satu *class*.

Atribut merepresentasikan beberapa *property* dari sesuatu yang kita model kan, yang dibagi dengan semua *object* dari semua class yang ada.

Methods / Operasi adalah abstraksi dari segala sesuatu yang dapat kita lakukan pada sebuah *object* dan ia berlaku untuk semua *object* yang terdapat dalam *class* tersebut (Sumber: ibm.com).