BAB II

LANDASAN TEORI

2.1 Pengertian Aplikasi

Aplikasi adalah seperangkat instruksi khusus dalam komputer yang dirancang agar kita dapat menyelesaikan tugas-tugas tertentu. Sebagai contoh. Aplikasi *Word Processing* adalah sebuah Aplikasi yang diperuntukan membuat dokumen tertulis. Aplikasi *Web Browser* adalah aplikasi yang diperuntukkan untuk mencari sesuatu dan menampilkan halaman web. (Shelly. Cashman dan Vermaat, 2009, p57).

2.2 Diagnosis

Diagnosis adalah upaya untuk menegakan atau mengetahui jenis penyakit yang diderita oleh seseorang atau masalah kesehatan yang dialami oleh masyarakat. Diagnosis sangat familiar pada dunia medis. Biasanya diagnosis dilakukan terhadap sebuah penyakit melalui gejala yang dialami oleh seseorang.

2.3 Penyakit

Penyakit adalah suatu keadaan abnormal dari tubuh atau pikiran yang menyebabkan ketidaknyamanan, disfungsi atau kesukaran terhadap orang yang dipengaruhinya. Untuk menyembuhkan penyakit, orang-orang biasa berkonsultasi dengan seorang dokter.

2.3.1 Konsep Dasar Penyakit

Penyakit adalah suatu keadaan abnormal dari tubuh atau pikiran yang dapat menyebabkan ketidaknyamanan, disfungsi atau kesukaran terhadap orang yang dipengaruhinya. Penyakit dari segi karakteristiknya dapat dibedakan menjadi dua, yaitu penyakit menular dan penyakit tidak menular. Penyakit menular merupakan penyakit yang disebabkan oleh kuman yang menjangkiti tubuh manusia. Kuman dapat berupa virus, bakteri atau jamur. Penyakit tidak menular merupakan penyakit yang tidak disebabkan oleh kuman, tetapi disebabkan karena adanya masalah fisiologis atau metabolisme pada jaringan tubuh manusia.

Untuk mendiagnosa suatu penyakit perlu diketahui dan dipelajari dahulu gejala-gejala yang ditimbulkan. Meskipun dari gejala-gejala yang terlihat langsung maupun yang dirasakan penderita, dapat diambil suatu kesimpulan penyakit yang diderita. Tetapi ada kalanya gejala-gejala yang ada tidak mengarah tepat kepenyakit tertentu, untuk itu perlu dilakukan pemeriksaan lebih mendalam, misalnya tes *urine*, tes darah, USG, atau pemeriksaan sinar X (*rontgen*).

2.3.2 Diagnosa Penyakit oleh Dokter

Dalam menganalisa kondisi pasien, seorang dokter pertama kali melakukan anamnesis atau *medical history talking* atau lazim disebut penelusuran riwayat kesehatan. Anamnesis merupakan cara untuk mendapatkan keterangan dan data-data klinis tentang keadaan penyakit pasien melalui tanya jawab lisan atas keluhan atau gejala-gejala yang dirasakan serta keterangan lain yang dialami oleh pasien tersebut. Sistem anamnesis biasanya terdiri dari 5 tahap, yaitu :

1. Keluhan Utama

Keluan utama yang menimbulkan perasaan dan pikiran pada penderita sehingga datang untuk meminta pertolongan medis.

2. Keluhan Tambahan

Keterangan yang menjelaskan keadaan klinis penderita baik yang ada hubungan dengan kelainan utama ataupun perihal lainnya yang menganggu kesehatan penderita saat ini.

3. Riwayat Kesehatan

Menyangkut riwayat penyakit terdahulu dan kebiasaan yang ada hubungan dengan penyakit.

4. Riwayat Keluarga

Keterangan atau data mengenai kehidupan bermasyarakat dan riwayat perkawinan untuk mencari apakah ada faktor *heredomafilial* yang mungkin merupakan faktor predisposisi.

5. Riwayat Sosial Ekonomi

Keterangan atau data mengenai kehidupan bermasyarakat dan riwayat pekerjaan serta status ekonomi penderita, termasuk juga dalam hal ini suasana dan lingkungan hidup dalam rumah.

2.4 Jenis Penyakit

Ada banyak jenis klasifikasi penyakit. Namun dalam jenis umum penyakit diklasifikasikan sebagai berikut:

1. Penyakit parasit dan Infeksi

Penyakit parasit dan menular yang disebabkan oleh masuknya bakteri, parasit, virus dan jamur ke dalam tubuh dengan cara apapun. Menurut sarana pintu masuk, penyakit dapat lebih diklasifikasikan sebagai penyakit transmisi lisan, seksual penyakit transmisi dll

2. Gizi dan Penyakit Imunitas

penyakit gizi disebabkan oleh kurangnya jumlah yang tepat dari tertentu atau sejumlah nutrisi dalam tubuh. Kebiasaan makanan yang salah dan asupan makanan kesehatan berbahaya menyebabkan penyakit gizi.Imunitas penyakit disebabkan oleh kurangnya daya kekebalan dalam tubuh, karena yang seorang individu lebih rentan untuk menerima penyakit dari lingkungan sekitar.

3. Penyakit genetik

Penyakit genetik ditularkan dari leluhur / orang tua kepada anak-anak melalui kode genetik yg mencelakakan.

4. Penyakit lainnya

Penyakit lain termasuk tumor, Gouts, batu, gondok, dll arthritis yang dihasilkan oleh fungsi yang tidak tepat dari kelenjar, organ, sel, sistem tubuh.

2.5 Alasan Umum Penyakit

Alasan umum penyakit dapat diklasifikasikan menurut kekebalan kita, kebiasaan makan, gaya hidup, dan kelainan genetik.

1. **Kurangnya Kekebalan**

   Kurangnya kekebalan atau resistensi yang rendah terhadap serangan penyakit merupakan alasan utama dari zaman modern. Alasan kurangnya kekebalan banyak, yaitu kurangnya olahraga, pemutusan dari alam, polusi.

2. **Kebiasaan buruk makanan**

   makan buruk dan kebiasaan makanan juga alasan utama untuk penyakit. Makanan yang salah seperti minuman beralkohol, hal goreng, minuman dingin, tembakau, dll merokok mengarah untuk menghasilkan banyak penyakit. Kebiasaan makan yang buruk, seperti, makan tepat sebelum tidur di malam hari, makan cepat, makan sambil berjalan atau menonton TV dll juga menyebabkan penyakit, karena ia menciptakan gangguan metabolisme dalam tubuh.

3. **Gaya hidup yang buruk**

   gaya hidup Buruk meliputi, tidur larut malam dan bangun terlambat di pagi hari, tidak ada latihan, merokok dan minum berlebihan, makan junk food, tinggal jauh dari alam, stres mental dll semacam ini gaya hidup yang buruk mengganggu fungsi sistem tubuh, yang hasil dalam berbagai penyakit.

4. **Penyakit genetik**

   penyakit genetik sangat berbeda dari semua jenis penyakit, karena penyebabnya adalah transmisi kode genetik yg mencelakakan dari nenek moyang kepada anak-anak mereka.

2.6 Kewaspadaan Terhadap Penyakit Umum

Pencegahan umum terhadap penyakit sangat mudah untuk diimplementasikan. Jika kita mengikuti beberapa aturan dasar alam, kita dapat menjauhkan diri dari semua penyakit. Beberapa Kewaspadaan terhadap penyakit adalah sebagai berikut:

1. Meningkatkan Kekebalan

Meningkatkan kekebalan dengan olahraga teratur dan dengan menghubungkan dengan alam.

2. Kebiasaan Makan yang baik

Mengadopsi kebiasaan makan yang baik dan makanan yang baik. Jangan gunakan minuman beralkohol, minuman dingin, minuman asam, tembakau, merokok dll Makanlah makanan setidaknya 2 jam sebelum kembali Anda pergi tidur. Jangan makan sambil berjalan, berdiri, berbicara dan menonton sesuatu. Makan Perlahan.

3. Gaya hidup yang baik

Mengadopsi gaya hidup yang baik, yaitu pergi lebih awal ke tempat tidur dan bangun pagi, melakukan latihan secara teratur, berhenti merokok, minum dll, tetap berhubungan alam teratur dan terus-menerus, dan mengurangi stres mental.

2.7 Sistem Pakar

Secara umum, sistem pakar merupakan sistem yang mengadopsi pengetahuan manusia ke dalam komputer sehingga komputer dapat digunakan

untuk menyelesaikan suatu masalah sebagaimana yang dilakukan oleh seorang pakar. Sistem pakar dibuat pada wilayah pengetahuan tertentu dan untuk suatu keahlian tertentu yang mendekati kemampuan manusia di salah satu bidang khusus. Sistem pakar mencoba mencari solusi yang memuaskan sebagaimana yang dilakukan seorang pakar dan dapat memberikan penjelasan terhadap langkah yang diambil serta memberikan alasan atas kesimpulan yang diambil.

2.7.1 Definis Sistem Pakar

Ada beberapa definisi tentang system pakar, antara lain sebagai berikut

1. Menurut Durkin : Sistem Pakar adalah suatu program komputer yang dirancang untuk memodelkan kemampuan penyelesaian masalah yang dilakukan seorang pakar.
2. Menurut Ignizio : Sistem pakar adalah suatu model dan prosedur yang berkaitan, dalam suatu dominan tertentu, yang mana tingkat keahlianya dapat dibandingkan dengan keahlian seorang pakar.
3. Menurut Giarratano dan Riley : Sistem pakar adalah suatu system komputer yang bisa menyamai atau meniru kemampuan seorang pakar

2.7.2 Apa Itu Pakar

Pakar adalah seseorang yang memiliki kemampuan khususnya terhadap suatu permasalahan, misalnya : dokter, petani, ahli pemesinan, dan lain-lain.

Ciri-ciri pakar ialah sebagai Berikut :

1. Dapat Mengenali dan merumuskan masalah.

2. Menyelesaikan masalah dengan cepat dan tepat.
3. Belajar dari pengalaman
4. Restrukturisasi pengetahuan
5. Menentukan relevansi

Jenis-jenis pengetahuan yang dimiliki dalam kepakaran adalah sebagai berikut :

1. Teori-teori dari permasalahan
2. Aturan dan prosedur yang mengacu pada area permasalahan
3. Aturan (heuristic) yang harus dikerjakan pada situasi yang terjadi
4. Strategi global untuk menyelesaikan berbagai jenis masalah.
5. Meta-knowladge (pengetahuan tentang pengetahuan).
6. Fakta-fakta

2.7.3 Apa itu Pengetahuan

Data → Procesing → Informasi → Pengetahuan

Alasan mendasar mengapa system pakar dikembangkan untuk menggantikan seorang pakar.

1. Akses informasi yang tiada batas untuk menyediakan informasi kepakaran
2. Secara otomatis mengerjakan tugas-tugas rutin yang membutuhkan seorang pakar.
3. Melanjutkan kepakaran dari seorang pakar.
4. Informasi mengenai kepakaran masih mahal, dan
5. Kapakaran dibutuhkan juga pada lingkungan yang kurang mendukung.

2.7.4 Perbandingan Sistem Konvensional dengan Sistem Pakar

Sistem konvensional berbeda dengan Sistem Pakar, berikut adalah perbandingan system konvensional dan sistem pakar.

1. Sistem Konvensional

Informasi dan pemrosesannya biasanya jadi satu dengan program. Biasanya tidak bisa menjelaskan mengapa suatu input data itu dibutuhkan aau bagimana output itu diperoleh. Pengubaan peogram cukup sulit dan membosankan. Sistem hanya akan beroperasi jika sistem tersebut sudah lengkap. Eksekusi dilakukan langkah demi langkah menggunakan data. Tujuan utamanya adalah efisiensi

2. Sistem Pakar

Basis pengetahuan merupakan bagian dari mekanisme inferensi. Penjelasan adalah bagian terpenting dari sistem pakar. Pengubahan aturan dapat dilaksanakan dengan mudah. Sistem dapat beroperasi hanya dengan beberapa aturan. Eksekusi dilakukan pada keseluruhan basis pengetahuan. Menggunakan pengetahuan, tujuan utamanya adalah efektivitas.

Tabel 2.1 Perbandingan Sistem Konvensional dan Sistem Pakar

Sistem Konvensional	Sistem Pakar
Informasi dan pengolahan biasanya digabungkan dalam satu program berurutan	Basis Pengetahuan secara nyata dipisahkan dari mekanisme pengolahan (Inferensi)
Program tidak melakukan kesalahan	Program dapat melakukan kesalahan

(programmer atau penggunaan yang melakukan kesalahan)	
Biasanya tidak menjelaskan mengapa data input diperlukan atau bagaimana kesimpulan dihasilkan.	Penjelasan adalah bagian dari sebagian besar ES.
Memerlukan semua data input berfungsi dengan tidak tepat jika ada data yagn hilang, kecuali jika telah dirancang demikian	Tidak memerlukan semua fakta awal. Biasanya dapat tiba pada kesimpulan yang masuk akal, sekalipun ada fakta yang hilang
Perubahan dalam program sangat membosankan (Kecuali dalam DOS)	Perubahan dalam aturan mudah dilakukan
Sistem beroperasi hanya jika telah lengkap	Sistem dapat beroperasi dengan hanya dengan sedikit aturan
Eksekusi dilakukan pada basis algoritma langkah demi langkah	Eksekusi dilakukan dengan menggunakan heuristik dan logika.
Manipulasi efektif pada database	Manipulasi efektif pada basis pengetahuan
Representasi dan penggunaan data	Representasi dan penggunaan pengetahuan
Efisiensi biasanya menjadi tujuan utama, efektivitas penting hanya untuk DSS	Efektivitas adalah tujua utama
Mudah menangani data kuantitatif	Mudah menangani kualitatif

Menggunakan representasi data numerik.	Menggunakan representasi pengetahan simbolik dan numerik.
Menyerap, memperbesar, dan mendistribusikan akses ke data atau informasi numerik.	Menyerap, memperbesar, dan mendistribusikan akses ke penilaian atau pengetahuan.

2.7.5 Bentuk Sistem Pakar

4 bentuk sistem pakar, yaitu sebagai berikut ;

1. Berdiri sendiri, sistem pakar ini merupakan software yang berdiri sendiri, tidak tergantung dengan software yang lainya.
2. Tergabung, sistem pakar jenis ini merupakan bagian program yang terkandung didalam suatu algoritma (konvensional) atau merupakan program dimana didalamnya memanggil algoritma subrutin lain (konvensional).
3. Menghubungkan ke software yang lain. Bentuk ini biasanya merupakan sistem pakar yang menghubungkan ke suatu paket program tertentu, misalnya DBMS.
4. Sistem mengabdi, sistem pakar merupakan bagian dari komputer khususnya yang dihubungkan dengan suatu fungsi tertentu, misalnya sistem pakar yang digunakan untuk membantu menganalisa data radar.

2.7.6 Struktur Sistem Pakar

Sistem pakar terdiri dari atas 2 bagian pokok, yaitu lingkungan pengembangan (development environment) dan lingkungan konsultasi (consultation environment).

1. Lingkungan pengembangan digunakan sebagai pembangunan sistem pakar, baik dari segi pembangunan komponen maupun basis pengetahuan.
2. Lingkungan konsultasi digunakan oleh seorang yang bukan ahli untuk berkonsultasi.

2.7.7 Basis Pengetahuan (Knowladge Base)

Basis pengetahuan berisi pengetahuan-pengetahuan dalam penyelesaian masalah, tentu saja di dalam domain tertentu. Ada 2 bentuk pendekatan basis pengetahuan yang sangat umum digunakan, yaitu sebagai berikut.

1. Penalaran Berbasis Aturan (Rule-Based Reasoning)

   Pada penalaran berbasis aturan, pengetahuan direpresentasikan dengan menggunakan aturan berbentuk. IF-THEN. Bentuk ini digunakan apalagi kita memiliki sejumlah pengetahuan pakar paa suatu permasalahan tertentu, dan si pakar dapat menyelesaikan masalah tersebut secara berurutan. Di samping itu, bentuk ini juga digunakan apabila dibutuhkan penjelasan tentang jejak (langkah-langkah) pencapaian solusi.

2. Penalaran Berbasis Kasus (Case-Based Reasoning)

   Pada penalaran berbasis kasus, basis pengetahuan akan berisi solusi-solusi yang telah dicapai sebelumnya, akan diturunkan suatu solusi untuk

keadaan yang terjadi sekarang (fakta yang ada). Bentuk ini digunakan apabila user menginginkan untuk tahu lebih banyak lagi pada kasus-kasus yagn hamper sama (mirip). Selain itu, bentuk ini juga digunakan apabila kita telah memiliki sejumlah situasi atau kasus tertentu dalam basis pengetahuan.

2.7.8 Keterbatasan Sistem Pakar

Kelemahan sistem pakar, adalah sebagai berikut.

1. Pengetahuan sulit selalu siap tersedia
2. Akan sulit mengekstrak keahlian dari manusia.
3. Pendekatan tiap pakar pada suatu penilaian situasi mungkin berbeda, tetapi benar
4. Sulit, bahkan bagi pakar berkemampuan tinggi untuk menghiktisarkan penilaian situasi yang baik pada saat berada dalam tekanan waktu.
5. Penggunaan sistem pakar memiliki batasan kognitif alami.
6. Sistem pakar bekerja dengan baik hanya dalam domain pengetahuan sempit.
7. Kebanyakan pakar tidak memiliki sarana mandiri untuk memeriksa apakah kesimpulannya masuk akal.
8. Kosa kata yang digunakan pakar untuk menyatakan fakta dan hubungan.

2.7.9 Fitur-Fitur Sistem Pakar

Sistem pakar harus memiliki fitur, yaitu sebagai berikut.

1. **Keahlian**

   Sistem pakar harus memiliki keahlian yang akan memungkinkan sistem membuat keputusan tingkat pakar. Sistem harus menampilkan performa pakar dan kekuatan yang cukup

2. **Pertimbangan Simbolik**

   Pengetahuan harus direpresentasikan secara simbolik dan mekanisme pertimbangan primer juga harus simbolik. Mekanisme pertimbangan simbolik biasanya menyertakan backward chaining dan forward chaining, yang akan dideskripsikan pada bagian selanjutnya.

3. **Deep Knowledge (kedalaman pengetahuan)**

   Basis pengetahuan harus berbasis pengetahuan yang kompleks yang tidak mudah diperoleh dari nonpakar.

4. **Self-Knowladge**

   Sistem pakar harus dapat menganalisa pertimbangannya sendiri dan menjelaskan mengapa dicapai suatu kesimpulan.

2.7.10 Manfaat dan Kemampuan Sistem Pakar

Manfaat dan kemampuan sistem pakar adalah sebagai berikut :

1. Meningkatkan output dan produktivitas
2. Menurunkan waktu pengambilan keputusan
3. Meningkatkan kualitas proses dan produk.

4. Mengurangi downtime.
5. Menyerap keahlian langka
6. Fleksibilitas.
7. Operasi peralatan yang lebih mudah.
8. Eliminasi kebutuhan peralatan yang mahal.
9. Operasi dilingkungan yang berbahaya.
10. Aksesiblilitas ke pengetahuan dan help desk.
11. Kemampuan untuk bekerja dengan informasi yang tidak lengkap/tidak pasti.
12. Kelengkapan pelatihan
13. Peningkatan pemecahan masalah dan pengembalian keputusan.
14. Meningkatkan proses pengambilan keputusan.
15. Mengingkatkan kualitas keputusan.
16. Kemampuan untuk memecahkan persoalan kompleks
17. Transfer pengetahuan ke lokasi terpencil.

2.8 Faktor Kepastian (*Certainty Factor*)

Certainty Factor (CF) menunjukan ukuran kepastian terhadap suatu fakta atau aturan.

2.8.1 Pengertian Faktor Kepastian (*Certainty Factor*)

Dalam menghadapi suatu masalah sering ditemukan jawaban yang tidak memiliki kepastian penuh. Ketidakpastian ini bisa berupa probabilitas atau

kebolehjadian yang tergantung dari hasil suatu kejadian. Hasil yang tidak pasti disebabkan oleh dua factor yaitu aturan yang tidak pasti dan jawaban pengguna yang tidak pasti atas suatu pertanyaan yang diajukan oleh sistem.

Ada tiga penyebab ketidakpastian aturan yaitu aturan tunggal, penyelesaian konflik dan ketidakcocokan (*incompatibility*) antar konskuen dalam aturan. Aturan tunggal yang dapat menyebabkan ketidakpastian dipengaruhi oleh tiga hal, yaitu kesalahan, probabilitas dan kombinasi gejala (*evidence*). Kesalahan dapat terjadi karena (Kusrini, 2006) adalah sebagai berikut :

1. Ambiguitas, sesuatu didefinisikan dengan lebih dari satu cara.
2. Ketidak lengkapan data
3. Kesalahan informasi
4. Ketidak percayaan terhadap suatu alat
5. Adanya bias

Probabilitas disebabkan ketidak mampuan seorang pakar merumuskan suatu aturan secara pasti. Misalnya jika seseorang mengalami sakit kepala, demam dan bersin-bersin ada kemungkinan orang tersebut terserang penyakit flu, tetapi bukan berarti apabila seseorang mengalai gejala tersebut pasti terserang penyakit flu.

Certainty Factor (CF) menujukkan ukuran kepastian terhadap suatu fakta atau aturan.Notasi Faktor Kepastian(Sri Kusumadewi, 2003) adalah sebagai berikut :

CF[h,e] = MB[h,e] – MD[h,e]

Dengan

CF[h,e] : Faktor Kepastian

MB[h,e] : ukuran kepercayaan terhadap hipotesis h , jika diberikan *evidence*e (antara 0 dan 1).

MD[h,e] : ukuran ketidakpercayaan terhadap evidence h,jika diberikan *evidence* e (antara 0 dan 1)

2.8.2 Kombinasi Aturan

Metode MYCIN untuk menggabungkan *evidence* pada *antecedent* sebuah aturan yang ditunjukka pada tabel berikut ini :

Tabel 2.2 Aturan kombinasi MYCIN

Evidence, E	**Antecedent Ketidak Pastian**
E_1 dan E_2	**Min [CF(H, E_1), CF(H, E_2)]**
E_1OR E_2	**Max[CF(H, E_1), CF(H, E_2)]**
TIDAK E	- **CF(H,E)**

Bentuk dasar rumus *certainty factor* sebuah aturan JIKA E MAKA H adalah sebagai berikut :

CF(H,e) = CF(E,e) * CF(H,E)

Di mana :

CF(E,e) : *Certainty Factor evidence* E yang dipengaruhi ileh *evidence* e

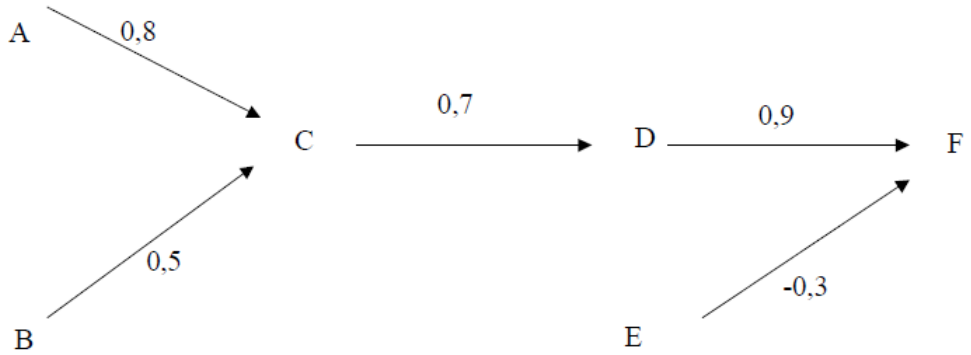
CF(H,E) : *Certainty Factor* hipotesis dengan asumsi *evidence* diketahui dengan pasti, yaitu ketika CF(E,e) = 1

CF(H,e) : *Certainty Factor* hipotesis yang dipengaruhi oleh *evidence* e

Jika semua *evidence* dan *antecedent* diketahui dengan pasti maka rumusnya menjadi

CF(H,e) = CF (H,E)

Dalam diagnosis suatu penyakit , hubungan antara gejala dengan hipotesis sering tidak pasti. Sangat dimungkinkan beberapa aturan menghasilkan satu hipotesis dan suatu hipotesis menjadi *evidence* bagi aturan lain. Kondisi tersebut dapat digambarkan sebagai berikut :



Gambar 2.1 Jaringan penalaran *certainty factor*

Dari gambar di atas ditunjukkan bahwa *certainty factor* dapat digunakan untuk menghitung perubahan derajat kepercayaan dari hipotesis F ketika A dan B bernilai benar. Hal ini dapat dilakukan dengan mengkombinasikan semua *certainty factor* pada A dan B menuju F menjadi sebuah alur hipotesis *certainty factor* seperti di bawah ini:

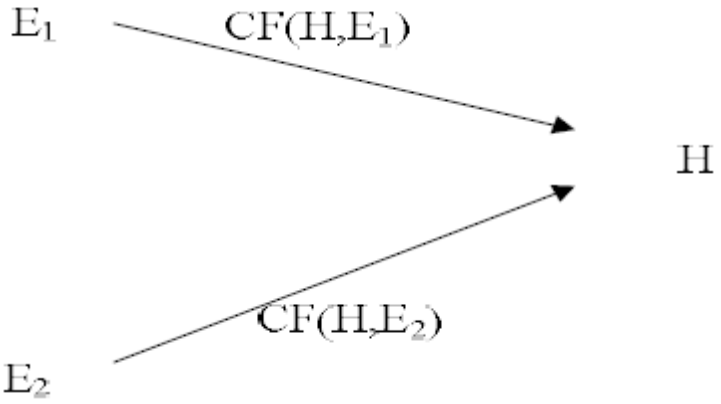
JIKA (A DAN B) MAKA F

Kondisi ini juga dapat digambarkan sebagai berikut:



Gambar 2.2 Kombinasi *Certainty Factor*

Kombinasi seperti ini disebut kombinasi paralel ,sebagaimana ditunjukkan oleh gambar di bawah ini :



Gambar 2.3 Kombinasi Paralel *Certainty Factor*

Pada kondisi ini *evidence* E1 dan E2 mempengaruhi hipotesis yang sama, yaitu H. Kedua *Certainty Factor* CF(H,E1) dan CF(H,E2) dikombinasikan menghasilkan *certainty factor* CF(H,E1,E2). Certainty kedua aturan dikombinasikan sehingga menghasilkan *certainty factor* CF(H,E'). Untuk menghitung kombinasi tersebut digunakan rumus berikut

CF(H,E') = CF(E,E') * CF (H,E)

2.8.3 Perhitungan *Certainty Factor*

Berikut ini adalah contoh ekspresi logika yang mengkombinasikan *evidence* :

E=(E1 DAN E2 DAN E3) ATAU (E4 DAN BUKAN E5)

Gejala E akan dihitung sebagai :

E = max[min(E1,E2,E3),min(E4,-E5)]

Untuk nilai E1 = 0,9 E2 = 0,8 E3 = 0,3 E4 = -0,5 E5 = -0,4

Hasilnya adalah :

E = max[min(E1,E2,E3),min(E4,-E5)]

= max(0,3, -0,5)

= 0,3

Bentuk dasar rumus *Certainty Factor* sebuah aturan JIKA E MAKA H ditunjukkan oleh rumus :

CF(H,e) = CF(E,e)*CF(H,E)

Dimana :

CF(E,e) : *Certainty Factor evidence* E yang dipengaruhi oleh *evidence*

CF(H,E) : *Certainty Factor* hipotesis dengan asumsi *evidence* diketahui dengan

pasti , yaitu ketika CF(E,e)=1

CF(H,e) : *Certainty factor* hipotesis yang dipengaruhi oleh *evidence e*

Jika semua *evidence* pada antecedent diketahui dengan pasti, maka rumusnya ditunjukkan sebagai berikut :

CF(H,e) = CF(H,E)

Karena CF(E,e) = 1.

Contoh kasus yang melibatkan kombinasi CF :

JIKA batuk

DAN demam

DAN sakit kepala

DAN bersin-bersin

MAKA influenza, CF : 0,7

dengan menganggap E1 : “batuk”, E2 :”demam”, E3 :”sakit kepala”, E4:”bersinbersin”, dan H:”influenza”, nilai *certainty factor* pada saat *evidence* pasti adalah :

CF(H,E) : CF(H,E1 ∩ E2 ∩ E3 ∩ E4)

: 0,7

Dalam kasus ini , kondisi pasien tidak dapat ditentukan dengan pasti . *Certainty factor evidence* E yang dipengaruhi oleh partial *evidence* e ditunjukkan dengan nilai sebagai berikut :

CF(E1,e) : 0,5 (pasien mengalami batuk 50%)

CF(E2,e) : 0,8 (pasien mengalami demam 80%)

CF(E3,e) : 0,3 (pasien mengalami sakit kepala 30%)

CF(E4,e) : 0,7 (pasien mengalami bersin-bersin 70%)

Sehingga

CF(E,e) = CF(H,E1 ∩ E2 ∩ E3 ∩ E4)

= min[CF(E1,e), CF(E2,e), CF(E3,e), CF(E4,e)]

= min[0,5, 0,8, 0,3, 0,7]

= 0,3

Maka nilai *certainty factor* hipotesis adalah :

CF(H,e) = CF(E,e)* CF(H,E)

= 0,3 * 0,7

= 0,21

2.8.4 Menentukan CF Gabungan

CF gabungan merupakan CF akhir dari sebuah calon konklusi. CF ini dipengaruhi oleh semua CF paralel dari aturan yang menentukan konklusi tersebut. CF Gabungan diperlukan jika suatu konklusi diperoleh dari beberapa aturan sekaligus. CF Akhir dari suatu aturan dengan aturan yang lain digabungkan untuk mendapatkan nilai CF Akhir bagi calon konklusi tersebut. Adapun rumus untuk melakukan perhitungan CF Gabungan adalah sebagai berikut:

CF (x) + (y) – (CF(x) * CF(y)), CF(x), CF(y) > 0

$$\frac{\mathrm{CF(x)} + \mathrm{CF(y)}}{\left(1 - \mathrm{Min}(|\mathrm{CF(x)}|, |\mathrm{CF(y)}|)\right)}, \text{Salah satu } \mathrm{CF}\ (\mathrm{x}), \mathrm{CF}(\mathrm{y} < 0)$$

2.9 Pengertian *Android*

Android adalah sebuah sistem operasi untuk perangkat *mobile* berbasis linux yang mencakup sistem operasi, *middleware* dan aplikasi.

Android merupakan generasi baru *platform mobile* yang memberikan kesempatan kepada pengembang untuk melakukan pengembangan sesuai dengan yang diharapkan. Sistem operasi yang mendasari *Android* merupakan lisensi di bawah naungan GNU, General *Public* License Versi 2(GPLv2), yang biasa

dikenal dengan istilah *Copyleft*. Istilah *copyleft* ini merupakan lisensi yang setiap perbaikan oleh pihak ketiga harus terus jatuh di bawah *terms*.

Distribusi *Android* berada di bawah lisensi Apache Software (ASL/Apache2), yang memungkin untuk distribusi kedua atau seterusnya. Pengembang aplikasi *Android* diperbolehkan untuk mendistribusikan aplikasi mereka di bawah skema lisensi apapun yang mereka inginkan.

Pengembang memiliki beberapa pilihan dalam membuat aplikasi yang berbasis *Android*. Namun kebanyakan pengembang menggunakan *Eclipse* sebagai IDE untuk merancang aplikasi mereka. Hal ini diikarenakan *Eclipse* mendapat dukungan langsung dari Google untuk menjadi IDE pengembangan aplikasi *Android*.

Aplikasi *Android* dapat dikembangkan pada berbagai sistem operasi, diantaranya adalah:

a. Windows XP/Vista/7

b. Mac OS X (Mac OS X 10.48 atau yang lebih baru)

c. Linux

2.9.1 Sejarah *Android*

Pada saat perilisan perdana *Android* pada tanggal 5 November 2007, *Android* bersama Open Handset Alliance mendukung pengembangan standar terbuka pada perangkat seluler tersebut. Di sisi lain, Google merilis kode-kode *Android* di bawah lisensi Apache. Sehingga terdapat dua jenis *distributor* sistem

operasi *Android* yaitu yang mendapat dukungan penuh dari Google dan yang mendapat dukungan penuh dari Open Handset Distribution (OHD).

Telepon selular pertama yang menggunakan sistem operasi *Android* adalah HTC Dream yang dirilis pada 22 Oktober 2008. Pada 9 Desember 2008, diumumkan anggota baru yang bergabung dalam program kerja *Android* ARM Holdings, Atheros Communication yang diproduksi oleh Asustek Komputer Inc, Garmin Ltd, Softbank, Sony Ericsson, Toshiba Corp dan Vodafone Group Plc.

Hingga saat ini terdapat beberapa versi dari sistem operasi *Android*, antara lain:

a. *Android* versi 1.1

Dirilis pada 9 Maret 2009. *Android* versi ini dilengkapi dengan adanya jam, alarm, *voice search*, pengiriman pesan dengan Gmail dan pemberitahuan *email*.

b. *Android* versi 1.5 (Cupcake)

Dirilis pada Mei 2009. Terdapat pembaruan dari versi 1.1 diantaranya adalah fitur *upload* video ke Youtube dan gambar ke Picasa langsung dari telepon, dukungan *bluetooth* A2DP, kemampuan terhubung secara otomatis ke headset *bluetooth*, animasi layar, dan *keyboard* pada layar yang dapat disesuaikan dengan sistem.

c. *Android* versi 1.6 (Donut)

Dirilis pada September 2009. Pembaruan yang terdapat pada versi ini diantaranya adalah proses pencarian yang lebih baik, penggunaan baterai indikator dan kontrol applet VPN. Fitur lainnya adalah memungkinkan

pengguna untuk memilih foto yang akan dihapus, kamera, camcorder dan galeri yang diintegrasikan, CDMA/EVDO, 802.1x, VPN, Gestures, *Text-to-speech engine*.

d. *Android* versi 2.1 (Éclair)

Dirilis pada 3 Desember 2009. Perubahan yang dilakukan adalah pengoptimalan *hardware*, peningkatan Google Maps 3.1.2, perubahan UI dengan *browser* baru dan dukungan HTML5, daftar kontak yang baru, dukungan *flash* untuk kamera 3.2 MP, digital zoom dan *bluetooth* 2.1.

e. *Android* versi 2.2 (Froyo)

Dirilis pada 20 Mei 2010. Versi *Android* inilah yang sekarang banyak digunakan sebagai standar sistem operasi mereka. Terdapat perubahan yang cukup signifikan dari versi sebelumnya diantaranya adalah kerangka aplikasi memungkinkan penggunaan dan penghapusan komponen yang tersedia, Dalvik Virtual Machine (DVM) yang dioptimalkan untuk perangkat *mobile*, grafik di 2D dan 3D berdasarkan *libraries* OpenGL, SQLite, mendukung berbagai format audio dan video, GSM, *bluetooth*, EDGE, 3G, Wifi, kamera, *Global Positioning System* (GPS), kompas dan accelerometer.

f. *Android* versi 2.3 (GingerBread)

Dirilis pada 6 Desember 2010. Beberapa perbaikan fitur dari versi sebelumnya adalah SIP-*based* VoIP, *Near Field Communications* (NFC), *gyroscope* dan sensor, *multiple cameras support*, *mixable audio effect* dan *download manager*.

g. *Android* versi 3.0 (Honeycomb)

Dirilis tahun 2011. *Android* versi ini dirancang khusus untuk tablet, sehingga terdapat perbedaan dari fitur UI (*User Interface*). Honeycomb sengaja dibuat untuk layar yang lebih besar dan juga dapat mendukung *multiprocessor*.

h. *Android* versi 4.0 (Ice Cream)

Versi ini masih dalam pengembangan. Dari berbagai informasi menyebutkan bahwa versi Ice Cream merupakan gabungan antara versi Gingerbread dengan Honeycomb. Sehingga bisa digunakan untuk ponsel maupun tablet. Dan kemungkinan dirilis pada quarter ke 4 tahun 2011

2.9.2 Arsitektur *Android*

Secara garis besar arsitektur *Android* dapat dijelaskan dan digambarkan sebagai berikut:

1. *Application* dan *Widget*

Application dan *Widget* ini adalah *layer* dimana kita berhubungan dengan aplikasi saja. Di *layer* terdapat aplikasi inti termasuk klien *email*, program SMS, kalender, peta, *browser*, kontak, dan lain-lain. Semua aplikasi ditulis dengan menggunakan bahasa pemrograman JAVA.

2. *Application Framework*

Application Framework adalah *layer* untuk melakukan pengembangan / pembuatan aplikasi yang akan dijalankan di sistem operasi *Android*, karena pada

layer inilah aplikasi dapat dirancang dan dibuat, seperti *content provider* yang berupa SMS dan panggilan telepon.

Komponen-komponen yang termasuk di dalam *Application Framework* adalah sebagai berikut:

a. *Views*
b. *Content Provider*
c. *Resource Manager*
d. *Notification Manager*
e. *Activity Manager*
f. *Libraries*

Libraries adalah *layer* tempat fitur-fitur *Android* berada, biasanya para pengembang aplikasi mengakses *libraries* untuk menjalankan aplikasinya

3. *Android Runtime*

Layer yang membuat aplikasi *Android* dapat dijalankan di mana dalam prosesnya menggunakan implementasi Linux. Dalvik *Virtual Machine* merupakan mesin yang membentuk dasar kerangka aplikasi *Android*.

Di dalam *Android Runtime* dibagi menjadi dua bagian yaitu:

a. Core *Libraries*

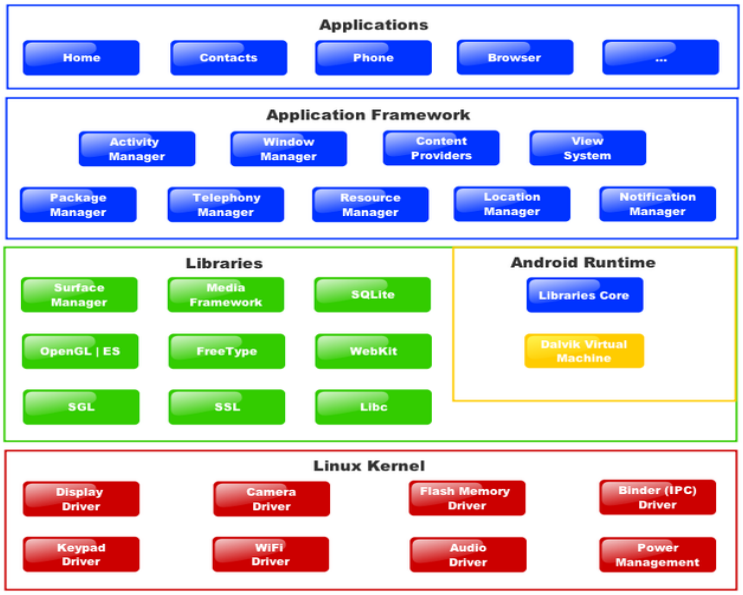
Aplikasi *Android* dibangun dalam bahasa Java, sementara DVM bukan merupakan *virtual machine* untuk Java. Sehingga diperlukan *libraries* yang berfungsi untuk menterjemahkan bahasa Java/C yang ditangani oleh *Core Libraries*

b. ***Dalvik Virtual Machine***

Virtual Machine berbasis *register* yang dioptimalkan untuk menjalankan fungsi-fungsi secara efisien, dimana merupakan pengembangan yang mampu membuat Linux kernel untuk melakukan *threading* dan manajemen tingkat rendah.

c. **Linux Kernel**

Linux Kernel adalah *layer* dimana inti sistem operasi dari *Android* itu berada. Berisi file system yang mengatur system processing *memory*, *resource*, *drivers*, dan sistem-sistem operasi *Android* lainnya. Linux Kernel yang digunakan *Android* adalah Linux Kernel *release* 2.4



Gambar 2.4 Arsitektur *Android*

(Sumber : NazruddinSafaat H.2012.*Pemrograman Aplikasi Mobile Smartphone dan Tablet PC berbasis Android***)**

2.9.3 Pengertian *Smartphone* (Telepone Pintar)

Smartphone adalah sebuah telepon yang kegunaan dasarnya sama dengan telepon biasa yang dapat dibawa kemana-mana dan tidak perlu disambungkan dengan kabel, namun memiliki kemampuan tingkat tinggi dengan fungsi yang menyerupai komputer.

2.9.4 *Java Software Development Kit* (SDK Java)

SDK Java merupakan kebutuhan utama bagi programmer untuk membuat dan menjalankan java. Komponen JDK antara lain compiler(javac), interpreter(java) disebut juga java virtual machine atau java runtime environment, applet viewer(appletviewer), debugger(jdb), java class library(jcl), header dan stub generator(javah), dan yang paling penting yaitu java documentation(javadoc).

2.9.5 *Eclipse Software Development Kit* (SDK *Eclipse*)

SDK *Eclipse* adalah sebuah IDE (*Integrated Development Environment*) untuk mengembangkan perangkat lunak dan dapat dijalankan di semua platform (*platform-independent*).

2.9.6 *Android Software Development Kit* (SDK *Android*)

SDK *Android* adalah tools API (*Application Programming Interface*) yang diperlukan untuk mulai mengembangkan aplikasi pada platform *Android* menggunakan bahasa pemrograman Java

2.9.7 *Android Eclipse Plugin* (ADT)

Definisi *Android Eclipse Plugin* adalah suatu perangkat tambahan koneksi *Eclipse* dengan *Android* SDK, sehingga bisa lebih memudahkan membuat program untuk *Android* di *Eclipse*.

2.9.8 *Android Package* (APK)

APK adalah paket aplikasi *Android* (***Android* PacKage**). APK umumnya digunakan untuk menyimpan sebuah aplikasi atau program yang akan dijalankan pada perangkat *Android*. APK pada dasarnya seperti zip file, karena berisi dari kumpulan file, dapat diperoleh melalui berbagai metode, seperti menginstal sebuah aplikasi melalui Market, download dari sebuah situs web, atau membuat sendiri dengan bahasa Java.

2.9.9 *Android Virtual Devices* (AVD)

AVD adalah konfigurasi dari emulator sehingga kita dapat menjalankan perangkat *Android* sesuai model yang dipilih, misalkan *Android* 1.5 atau 2.2. Untuk dapat menjalankan emulator.

2.9.10 Emulator

Emulator adalah Aplikasi yang memungkinkan sebuah software dari sebuah platform berjalan di atas platflorm lain. Emulator menjalankan kode-kode software pada virtual machine. Sehingga software tersebut mengira bahwa ia sedang berjalan di atas platform aslinya. Dalam hal ini emulator *Android* dapat

membuat *virtual phone* di komputer seperti telepon genggam berbasis sistem operasi *Android* yang sebenarnya.

2.10 Unified Modeling Language (UML)

Unified Modeling Language (UML) adalah bahasa pemodelan secara grafis untuk menspesifikasikan, menvisualisasikan, membangun, dan mendokumentasikan seluruh rancangan sistem perangkat lunak. Penggunaan model ini bertujuan untuk mengidentifikasikan bagian-bagian yang termasuk dalam lingkup sistem yang dibahas dan bagaimana hubungan antara sistem dengan subsistem maupun sistem lain di luarnya.

Selain itu UML adalah bahasa pemodelan yang menggunakan konsep orientasi object. UML dibuat oleh Grady Booch, James Rumbaugh, dan Ivar Jacobson di bawah bendera Rational Software Corp [HAN98]. UML menyediakan notasi-notasi yang membantu memodelkan sistem dari berbagai perspektif. UML tidak hanya digunakan dalam pemodelan perangkat lunak, namun hampir dalam semua bidang yang membutuhkan pemodelan.

Berikut ini definisi *Unified Modeling Language* (UML) menurut para ahli:

1. Menurut (Hend, 2006) "*Unified Modeling Language* (UML) adalah bahasa yang telah menjadi standard untuk visualisasi, menetapkan, membangun dan mendokumentasikan artifak suatu sistem perangkat lunak".
2. Menurut (Adi Nugroho : 2005). "*Unified Modeling Language* (UML) adalah alat bantu analisis serta perancangan perangkat lunak berbasis objek".

3. Menurut (Joomla dari http://soetrasoft.com : 2007). *"Unified Modeling Language* (UML) merupakan standard modeling language yang terdiri dari kumpulan-kumpulan diagram, dikembangkan untuk membantu para pengembang sistem dan *software* agar bisa menyelesaikan tugas-tugas seperti: Spesifikasi, Visualisasi, Desain Arsitektur, Konstruksi, Simulasi dan testing serta Dokumentasi".

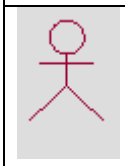
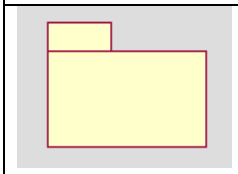
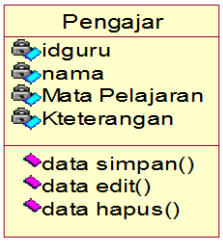
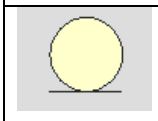
Berdasarkan beberapa pendapat yang dikemukakan diatas dapat ditarik kesimpulan bahwa "*Unified Modeling Language* (UML) adalah sebuah bahasa yang berdasarkan grafik atau gambar untuk menvisualisasikan, menspesifikasikan, membangun dan pendokumentasian dari sebuah sistem pengembangan perangkat lunak berbasis OO (*Object Oriented*)".

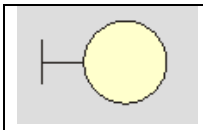
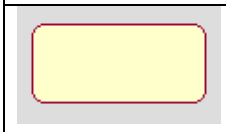
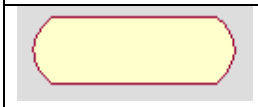
Unified Modelling Language (UML) adalah sebuah bahasa untuk menentukan, visualisasi, konstruksi, dan mendokumentasikan *artifacts* dari sistem *software*, untuk memodelkan bisnis, dan sistem *nonsoftware* lainnya. *Artifacts* adalah sepotong informasi yang digunakan atau dihasilkan dalam suatu proses rekayasa *software*. *Artifacts* dapat berupa model, deskripsi, atau software. Untuk membuat suatu model, UML memiliki diagram grafis yang diberi nama berdasarkan sudut pandang yang berbeda-beda terhadap sistem dalam proses analisa atau rekayasa. Diagram grafis tersebut antara lain

Daftar simbol-simbol dalam UML (Unified Modelling Language)

Tabel 2.3 Daftar Simbol-Simbol UML

Sumber : http://freetechebooks.com/ebook-2011/daftar-simbol-uml.html

Gambar simbol	Nama simbol	Keterangan
	*Usecase*	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu aktor
	*Actor*	Menspesifikasikan himpuan peran yang pengguna mainkan ketika berinteraksi dengan *use case*.
	*Package*	suatu simbol yang memberikan batasan dan komentar yang dikaitkan pada suatu elemen atau kumpulan elemen
	*Class*	Himpunan dari objek-objek yang berbagi atribut serta operasi yang sama.
	*Control*	Mengkordinasikan aktifitas dalam sistem
	*Entity*	Kelas yang berhubungan data dan informasi yang dibutuhkan oleh sistem

	*Boundery*	Kelas yang memodelkan interaksi antar satu atau lebih aktor dengan sistem
	*Activity*	Memperlihatkan bagaimana masing-masing kelas antarmuka saling berinteraksi satu sama lain
	*State*	Nilai atribut dan nilai link pada suatu waktu tertentu, yang dimiliki oleh suatu objek.

UML menyediakan 10 macam diagram yang merupakan salah satu alat bantu yang sangat handal dalam mengembangkan system berorientasi objek. Ada 9 jenis diagram yang ditangani oleh UML, yakni:

2.9.1 Use Case Diagram

Use Case Diagram adalah suatu kumpulan urutan interaksi di antara user dengan system untuk mencapai suatu tujuan di mana use case ini menggambarkan kebutuhan fungsional suatu system tanpa menampilkan struktur internal system.

2.9.2 Sequence Diagram

Sequence diagram adalah suatu diagram yang menggambarkan interaksi antar obyek dan mengindikasikan komunikasi diantara obyek-obyek tersebut. Diagram ini juga menunjukkan serangkaian pesan yang dipertukarkan oleh obyek-obyek yang melakukan suatu tugas atau aksi tertentu. Obyek-obyek tersebut

kemudian diurutkan dari kiri ke kanan, aktor yang menginisiasi interaksi biasanya ditaruh di paling kiri dari diagram.

2.9.3 Collaboration Diagram

Collaboration diagram adalah perluasan dari objek diagram. Objek diagram menunjukkan objek-objek dan hubungannya dengan yang lain. Collaboration diagram menunjukkan pesan-pesan objek yang dikirim satu sama lain.

2.9.4 Activity Diagram

Activity diagram adalah representasi secara grafis dari proses dari proses dan control flow dan berfungsi untuk memperlihatkan alur dari satu aktivitas ke aktivitas yang lain serta menggambarkan perilaku yang kompleks.

2.9.5 Deplyoment Diagram

Deployement adalah penggambaran tugas-tugas kongkrit dari setiap node/software yang terlibat dalam jaringan system, menampilkan keseluruhan node dalam jaringan serta hubungan dari node-node tersebut termasuk proses-proses yang terlibat di dalamnya.

2.9.6 Component Diagram

Component diagram adalah diagram yang menunjukkan organisasi dan kebergantungan di antara sekumpulan komponen. Diagram ini memodelkan pandangan implementasi fisik dari sistem.

2.9.7 Langkah-langkah Penggunaan *Unified Modeling Language* (UML)

Menurut (Afif Amrullah:2002). "Langkah-langkah penggunaan *Unified Modeling Language* (UML) sebagai berikut:

1. Buatlah daftar *business process* dari *level* tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul.
2. Petakan *use case* untuk setiap *business process* untuk mendefinisikan dengan tepat fungsional yang harus disediakan oleh sistem, kemudian perhalus *use case diagram* dan lengkapi dengan *requirement, constraints* dan catatan-catatan lain.
3. Buatlah *deployment diagram* secara kasar untuk mendefinisikan arsitektur fisik sistem.
4. Definisikan *requirement* lain *non fungsional, security* dan sebagainya yang juga harus disediakan oleh sistem.
5. Berdasarkan *use case diagram,* mulailah membuat *activity diagram.*
6. Definisikan obyek-obyek level atas *package* atau *domain* dan buatlah *sequence* dan/atau *collaboration* utuk tiap alir pekerjaan, jika sebuah *use case* memiliki kemungkinan alir normal dan *error,* buat lagi satu diagram untuk masing-masing alir.
7. Buatlah rancangan *user interface model* yang menyediakan antamuka bagi pengguna untuk menjalankan *skenario use case*.
8. Berdasarkan model-model yang sudah ada, buatlah *class diagram*. Setiap *package* atau *domian* dipecah menjadi *hirarki class* lengkap dengan

atribut dan metodenya. Akan lebih baik jika untuk setiap *class* dibuat *unit test* untuk menguji *fungsionalitas class* dan interaksi dengan *class* lain.

9. Setelah *class diagram* dibuat, kita dapat melihat kemungkinan pengelompokkan *class* menjadi komponen-komponen karena itu buatlah *component diagram* pada tahap ini. Juga, definisikan *test integrasi* untuk setiap komponen meyakinkan ia bereaksi dengan baik.
10. Perhalus *deployment diagram* yang sudah dibuat. Detilkan kemampuan dan *requirement* piranti lunak, sistem operasi, jaringan dan sebagainya. Petakan komponen ke dalam node.
11. Mulailah membangun sistem. Ada dua pendekatan yang tepat digunakan:
    a. Pendekatan *use case* dengan meng*assign* setiap *use case* kepada tim pengembang tertentu untuk mengembangkan unit kode yang lengkap dengan *test.*
    b. Pendekatan *komponen* yaitu *mengassign* setiap komponen kepada tim pengembang tertentu.
12. Lakukan uji modul dan uji integrasi serta perbaiki model beserta *code*nya. Model harus selalu sesuai dengan *code* yang aktual.Perangkat lunak siap dirilis".

2.11 Object Oriented Programming (OOP)

Object Oriented Progamming (OOP) atau Pemrograman Berorientasi Objek adalah konsep pemrograman yang difokuskan pada penciptaan kelas yang merupakan abstraksi/ *blueprint*/ *prototype* dari suatu objek. Kelas ini harus

mengandung sifat (data) dan tingkah laku (method) umum yang dimiliki oleh objek-objek yang kelak akan dibuat (diinstansiasi). Data dan method merupakan anggota (*member*) dari suatu kelas.

Pemrograman prosedural murni yang tidak menerapkan konsep *object oriented* (karena ada bahasa pemrograman prosedural yang juga sudah berorientasi objek, meskipun belum sepenuhnya) banyak menitikberatkan ke arah pembentukan fungsi-fungsi, sehingga di dalam program akan terdapat banyak sekali fungsi dan variabel yang menyulitkan pemrogram untuk mengelola dan mengembangkannya. Oleh karena itu, dengan memperhatikan kekurangan-kekurangan tersebut, maka dibentuklah bahasa yang menerapkan pendekatan *object oriented* untuk menyederhanakan fungsi-fungsi dan variabel-variabel ke dalam bentuk objek.

Dalam OOP dibutukan *memory* lebih besar dibandingkan dengan program prosedural (tradisional). Dua objek yang identik akan memerlukan dua area *memory* berbeda walaupun dari sisi data dan proses keduanya memiliki jumlah dan jenis yang sama. Hal ini disebabkan karena data dan proses pada kedua objek tersebut dipisahkan oleh komputer.

Secara garis besar yang menjadi ciri dari OOP adalah adanya proses abstraksi (*abstraction*), pengkapsulan (*encapsulation*), penurunan sifat (*inheritance*), dan polimorfisme (*polymorphism*) pada objek-objek yang dibentuk.

2.12 Bahasa Pemograman

Java adalah sebuah teknologi yang diperkenalkan oleh Sun Microsystems pada pertengahan tahun 1990. Menurut definisi dari Sun, Java adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer *standalone* ataupun pada lingkungan jaringan.

Java2 adalah generasi kedua dari Java platform (generasi awalnya adalah Java Development Kit). Java berdiri di atas sebuah mesin interpreter yang diberi nama JVM (*Java Virtual Machine*). JVM inilah yang akan membaca bytecode dalam file .class dari suatu program sebagai representasi langsung program yang berisi bahasa mesin. Oleh Karena itu bahasa Java disebut sebagai bahasa pemrograman yang *portable* karena dapat dijalankan berbagai system operasi, asalkan pada system operasi tersebut terdapat JVM.

Platform Java terdiri dari kumpulan library, JVM, kelas-kelas yang dipaket dalam sebuah lingkungan rutin Java, dan sebuah *compiler*, *debugger*, dan perangkat lain yang dipaket dalam Java Development Kit (JDK). Java2 adalah generasi yang sekarang sedang berkembang dari *platform* Java. Agar sebuah program Java dapat dijalankan, maka file dengan ekstensi ".java" harus dikompilasi menjadi file *bytecode*. Untuk menjalankan *bytecode* tersebut dibutuhkan JRE (*Java Runtime Environment*) yang memungkinkan pemakai untuk menjalankan program Java, hanya menjalankan, tidak untuk membuat kode baru lagi. JRE berisi JVM dan library Java yang digunakan.

2.13 Definisi Perangkat Yang Digunakan

Eclipse adalah sebuah IDE (*Integrated Development Environment*) untuk mengembangkan perangkat lunak dan dapat dijalankan di semua *platform* (*platform-independent*). *Eclipse* pada saat ini merupakan salah satu IDE favorit dikarenakan gratis dan *open source*, yang berarti setiap orang boleh melihat kode pemrograman perangkat lunak ini. Selain itu kelebihan dari *Eclipse* yang membuatnya popular adalah kemampuannya untuk dapat dikembangkan oleh pengguna dengan komponen yang dinamakan *plug-in*.

Karakteristik *Eclipse* Indigo:

1. Components

Eclipse terdiri dari model komponen-komponen (*plug-ins*) yang dapat digunakan lebih dari satu aplikasi.

2. Middleware and Infrastructure

Komponen utama dalam *Eclipse* dibuat berdasarkan *framework* dan fasilitas, sehingga mempermudah penulisan kode oleh pengguna. Fasilitas yang diperoleh, seperti: paradigma UI yang fleksibel, aplikasi yang dapat diekstensi, *help support*, *scalable* UI, *context-sensitive help*, *network updates*, *error handling*, dan lain-lainnya.

3. Native User Experience

Eclipse Standard Widget Toolkit menyediakan *toolkit* GUI untuk java yang efisien dan akses yang portable ke fasilitas bawaan (*native*) UI di OS.

4. Portability

Eclipse memiliki aplikasi yang fleksibel di berbagai OS dan *client environments* (syaratnya: *hardware* itu dapat diinstal *Java Runtime Environment*).

5. Intelligent Install and Update

Aplikasi di *Eclipse* memiliki fitur *update plugins* melalui HTTP, *Java Web Start*, *Update Site*, *copy file*, atau system manajemen perusahaan yang canggih.

6. Disconnected Operation

Tidak membutuhkan koneksi internet, karena aplikasi *Eclipse* berjalan di local komputer.

7. Development Tooling Support

Eclipse menyediakan class pertama di IDE java yang dapat diintegrasikan untuk *develop*, *test*, dan *package* aplikasi *rich clients*.

8. Component Libraries

Komponen di *framework* tidak akan lengkap tanpa set plugins yang komprehensif. Sehingga *Eclipse* akan memproduksi plugins yang dibutuhkan dalam membuat aplikasi secara utuh.

2.14 *Location Based Service* (LBS)

Location Based Service (LBS) atau layanan berbasis lokasi adalah istilah umum yang digunakan untuk menggambarkan teknologi yang digunakan untuk menemukan lokasi perangkat yang kita gunakan. Dua unsur utama LBS adalah:

1. *Location Manager* (API Maps)

Menyediakan *tools/source* untuk LBS, *Application Programming Interface* (API) Maps menyediakan fasilitas untuk menampilkan, memanipulasi *maps*/peta beserta *feature-feature* lainnya seperti tampilan satelit, *street* (jalan), maupun gabungannya. Paket ini berada pada com.google.*Android*.maps

2. *Location Providers* (API Location)

Menyediakan teknologi pencarian lokasi yang digunakan oleh device/perangkat. API Location berhubungan dengan data GPS (*Global Positioning System*) dan data lokasi real-time. API *Location* berapa pada paket *Android* yaitu dalam paket *Android.location.* Dengan Location Manager, kita dapat menentukan lokasi kita saat ini, *Track* gerakan/perpindahan, serta kedekatan dengan lokasi tertentu dengan mendeteksi perpindahan.

2.15 Arsitektur Andorid

Secara garis besar arsitektur *Android* dapat dijelaskan dan digambarkan sebagai berikut:

1.* *Applications dan Widgets

Applications dan *Widgets* ini adalah layer dimana kita berhubungan dengan aplikasi saja, dimana biasanya kita download aplikasi kemudian kita lakukan instalasi dan jalankan aplikasi tersebut. Di layer terdapat aplikasi inti termasuk klien email, program SMS, kalender, peta, browser,

kontak, dan lain-lain. Semua aplikasi ditulis menggunakan bahasa pemrograman Java.

2. Applications Frameworks

Android adalah "*Open Development Platform*" yaitu *Android* menawarkan kepada pengembang atau memberi kemampuan kepada pengembang untuk membangun aplikasi yang bagus dan inovatif. Pengembang bebas untuk mengakses perangkat keras, akses informasi *resources*, menjalankan *service background*, mengatur alarm, dan menambahkan status *notifications*, dan sebagainya.

3. Libraries

Libraries ini adalah layer dimana fitur-fitur *Android* berada, biasanya para pembuat aplikasi mengakses libraries untuk menjalankan aplikasinya. Berjalan di atas kernel, layer ini meliputi berbagai library C/C++ inti seperti Libc dan SSL, serta libraries lainnya.

4. Android Run Time

Layer yang membuat aplikasi *Android* dapat dijalankan dimana dalam prosesnya menggunakan Implementasi Linux. *Dalvik Virtual Machine* (DVM) merupakan mesin yang membentuk dasar kerangka aplikasi *Android*.

5. Linux Kernel

Linux Kernel adalah layer dimana inti dari operating system dari *Android* itu berada. Berisi file-file system yang mengatur *system processing*, *memory*, *resource*, *drivers,* dan system-sistem operasi *Android* lainnya.

2.16 *Android Lifecycle*

Setiap aktifitas yang ada di dalam aplikasi pasti akan melalui *lifecycle*. Misalnya jika menjalankan aplikasi fungsi *onCreate* akan digunakan untuk membuat aplikasi tampil di layar *Android*, jika keluar dari aplikasi fungsi *OnDestroy* akan digunakan

2.17 Database *SQLite*

Android juga memiliki fasilitas untuk membuat *database* yang dikenal dengan *SQLite* yaitu salah satu *software* yang *embedded* yang sangat popular, kombinasi *SQL interface* dan penggunaan memory yang sangat sedikit dengan kecepatan yang sangat cepat. SQLite di *Android* termasuk dalam *Android runtime*, sehingga setiap versi dari *Android* dapat membuat *database* dengan *SQLite*.

Dalam system *Android* terdapat beberapa teknik untuk melakukan penyimpanan data. Teknik yang umum digunakan adalah sebagai berikut:

1. *Shared prefences* yaitu menyimpan data beberapa nilai (*value*) dalam bentuk *groups key* yang dikenal dengan *prefences*.
2. *Files* yaitu menyimpan data dalam file, dapat berupa menulis ke file atau membaca dari file.
3. *SQLite Database*, yaitu menyimpan data dalam bentuk *database.*
4. *Content Providers*, yaitu menyimpan data dalam bentuk *content providers service*.

2.18 *Eclipse*

Eclipse adalah sebuah IDE (*Integrated Development Environment*) untuk mengembangkan perangkat lunak dan dijalankan di semua *platform*. *Eclipse* sendiri juga merupakan sebuah komunitas *open source*, yang memiliki proyek yang berfokus pada membangun sebuah *platform* pengembangan terbuka dari extensible *Framework*, *tools* dan runtime untuk membangun, menyebarkan dan mengelola perangkat lunak di seluruh siklus hidup perangkat lunak tersebut.

Umumnya *Eclipse* digunakan untuk membuat sebuah program yang menggunakan bahasa pemrograman Java. Namun, *Eclipse* juga bisa digunakan untuk penggunaan bahasa pemrograman lainnya seperti C, C++, COBOL, Perl, PHP, Python, dan sebagainya.

2.18.1 Versi *Eclipse*

Berikut ini adalah versi *Eclipse* yang telah dirilis:

Tabel 2.4 Versi *Eclipse*

Nama	**Tanggal rilis**	**Versi**
Eclipse 3.0	21 Juni 2004	3.0
Eclipse 3.1	28 Juni 2008	3.1
Callisto	30 Juni 2006	3.2
Europa	29 Juni 2007	3.3
Ganymade	25 Juni 2008	3.4
Galileo	24 Juni 2009	3.5

Helios	23 Juni 2010	3.6
Indigo	22 Juni 2011	3.7

Sumber: *Eclipse Foundation* (2011)