# BAB II
# LANDASAN TEORI

## 3.1. Tinjauan Pustaka

Penelitian yang dilakukan oleh Maruli Tua Nahampun (2014) yang membahas bagaimana penerapan metode *Dempster-Shafer* dalam mendiagnosa penyakit kelapa sawit yang diimplementasikan dengan menggunakan Microsoft Visual Studio 2008 dan MySQL sebagai basis data. Tujuan dari penelitian tersebut yaitu merancang aplikasi sistem pakar untuk mendiagnosa penyakit tanaman kelapa sawit berbasis *desktop*.

Penelitian lain oleh Adhinta Nicho Pratama dan Sukadi (2012) merupakan penelitian yang bertujuan menghasilkan sistem pakar untuk mendiagnosa hama dan penyakit tanaman padi serta mengimplementasikan sistem pakar untuk memberikan terapi penanganan hama dan penyakit tanaman padi. Penelitian tersebut diimplementasikan ke dalam sebuah aplikasi berbasis *desktop* yang menggunakan Microsoft Access 2007 sebagai penyimpan data dan menggunakan Microsoft Visual Basic 6.0 untuk pengembangan aplikasi.

Pada penelitian lainnya membahas tentang penggunaan metode *Dempster-Shafer* dalam pemanfaatannya pada sistem pakar untuk mendiagnosa penyakit saluran pencernaan (Yasidah Nur Istiqomah dan Abdul Fadlil, 2013). Sistem pakar ini diimplementasikan dalam aplikasi *desktop* dengan bahasa Microsoft Visual Basic 6.0 dengan cara memasukkan gejala-gejala yang dirasakan pasien dan kemudian dari hasil proses sistem akan memberikan hasil diagnosa penyakit yang diderita pasien.

Sedangkan penelitian lain oleh Anton Setiawan Honggowibowo (2009) menggunakan metode *Forward* dan *Backward Chaining* merupakan sistem pakar yang berbasis *web*. Penelitian ini menggunakan basis aturan (*rule based reasoning*) dengan metode inferensi *forward chaining* dan *backward chaining*. Sistem pakar diagnosa penyakit tanaman padi berbasis *web* ini digunakan sebagai alat bantu diagnosa penyakit tanaman padi dan dapat diakses oleh petani di manapun juga

untuk mengatasi persoalan keterbatasan jumlah pakar pertanian dalam membantu petani mendiagnosa penyakit tanaman padi.

Dari empat sistem tersebut yang telah ada sebelumnya masih berbasis *desktop*, atau meskipun ada yang berbasis *web* maka diperlukan adanya koneksi internet maupun perangkat keras yang mendukung sehingga memiliki kelemahan sebagai berikut:

1) Dibutuhkan perangkat komputer *desktop* ataupun *notebook* yang memiliki spesifikasi tertentu untuk dapat memenuhi kebutuhan sistem serta konektivitas internet untuk sistem yang berbasis *web*.
2) Dibutuhkan pengetahuan terkait pengoperasian sistem mulai dari proses pemasangan, *input* data, hingga *troubleshooting*.
3) Kurangnya kompatibilitas sistem dan kurang mendukung tingkat mobilitas para pengguna dalam konteks ini yaitu petani padi.

Dari beberapa kelemahan yang ada tersebut solusi yang dapat dilakukan antara lain yaitu dengan membangun suatu sistem pakar yang tidak membutuhkan perangkat komputer klasik atau konvensional yang masih berukuran relatif besar, melainkan dengan memanfaatkan perkembangan teknologi khususnya *smartphone* dengan sistem operasi Android yang menggunakan SQLite sebagai basis data penyimpannya. Sehingga diharapkan dapat membantu para petani dalam mengidentifikasi dan mendiagnosa penyakit pada tanaman padi dengan mudah tanpa harus membawa perangkat komputer dan tanpa perlu adanya koneksi internet.

## 3.2. Kerangka Pemikiran

Penelitian yang dilakukan adalah dengan mengembangkan aplikasi deteksi penyakit pada tanaman padi. Pendeteksian penyakit dilakukan dengan cara memproses masukan yang bersifat *user friendly* sehingga memudahkan pengguna. Kemudian sistem pakar dengan sendirinya dapat melakukan proses pendeteksian dari setiap gejala tersebut. Berikut diagram untuk kerangka pemikiran dalam penulisan laporan tugas akhir dengan keterangan sebagai berikut :

1) Latar belakang masalah

   Tahapan paling awal, yakni menelusuri latar belakang kenapa masalah yang akan diangkat menjadi penting untuk dipilih.

2) Rumusan masalah

   Penyimpulan latar belakang masalah menjadi suatu rumusan masalah yang akan diangkat untuk menjadi bahan penelitian.

3) Penguasaan Java, Android SDK dan SQLite

   Tahap untuk mempelajari aplikasi atau program-program yang akan digunakan untuk membangun sistem.

4) Pengumpulan data tertulis dan tidak tertulis

   Pengumpulan data dilakukan dengan mempelajari jurnal dan penelitian sejenis yang telah ada sebelumnya, observasi, maupun studi literatur di perpustakaan.

5) Observasi aplikasi Android dan SQLite

   Merupakan tahap pengamatan aplikasi yang telah ada, jurnal, buku, maupun karya ilmiah untuk kajian yang dapat dijadikan referensi untuk pembangunan sistem.

6) Analisis dan perancangan aplikasi

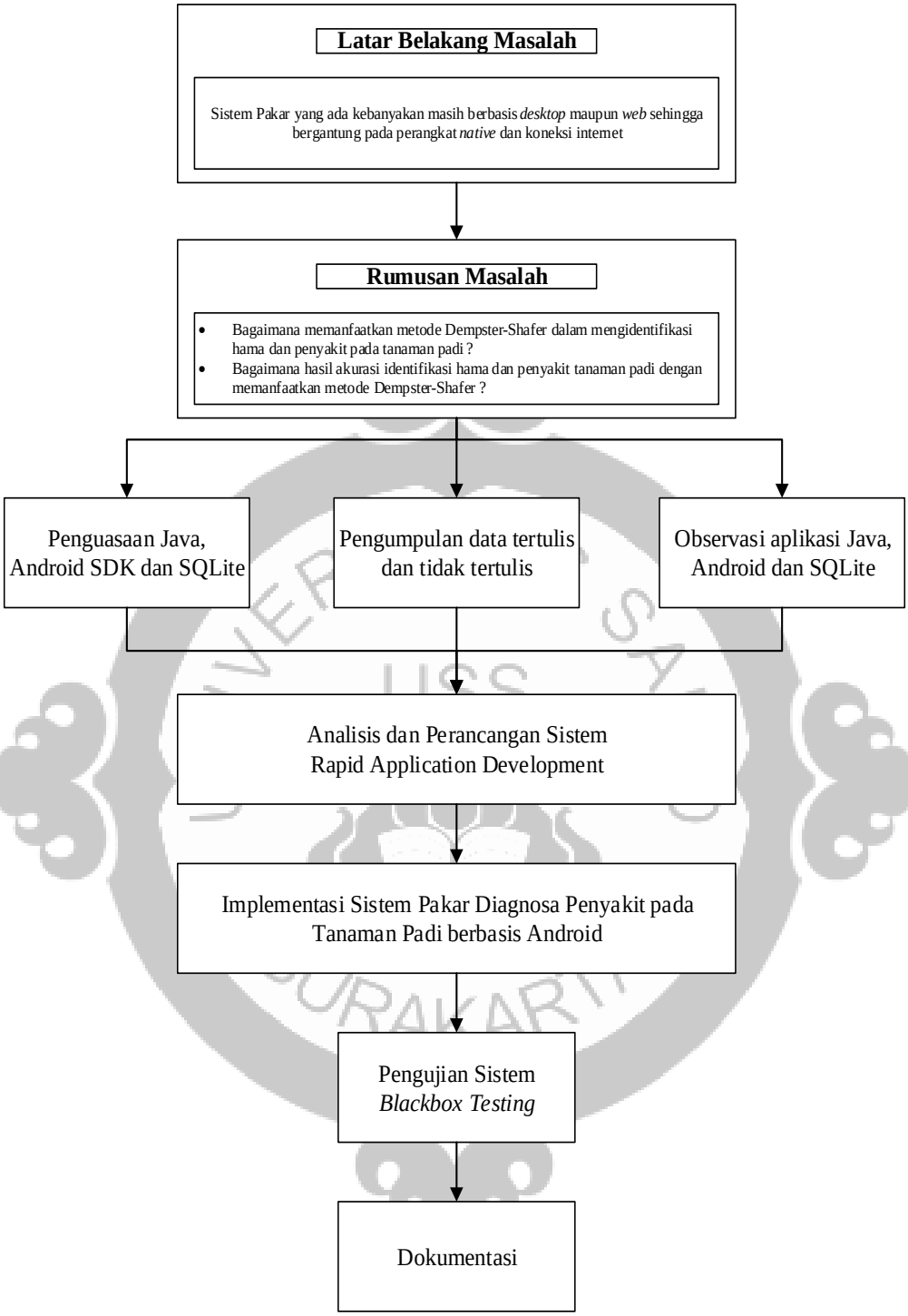
7) Implementasi Sistem Pakar

8) Pengujian Sistem

   Pengujian sistem menggunakan *black box testing* dilakukan dengan *emulator* dan beberapa perangkat *smartphone* untuk mengetahui jika ada kesalahan dan kekurangan pada sistem.

9) Dokumentasi

   Tahapan terakhir, yakni tahap pendokumentasian seluruh poses penyusunan tugas akhir ke dalam laporan.




*Gambar 2.1. Kerangka Pemikiran*

## 3.3. Implementasi

Definisi implementasi menurut Abdul Kadir (2003) adalah kegiatan yang dilakukan untuk menguji data dan menerapkan sistem yang diperoleh dari kegiatan seleksi. Implementasi merupakan salah satu unsur pertahapan dari keseluruhan pembangunan sistem komputerisasi, dan unsur yang harus dipertimbangkan dalam pembangunan sistem komputerisasi yaitu masalah perangkat lunak (*software*), karena perangkat lunak yang digunakan haruslah sesuai dengan masalah yang akan diselesaikan, disamping masalah perangkat keras (*hardware*) itu sendiri.

Buku lain berjudul "Konteks Implementasi Berbasis Kurikulum" dijelaskan bahwa: implementasi adalah bermuara pada aktivitas, aksi, tindakan, atau adanya mekanisme suatu sistem. Implementasi bukan sekedar aktivitas, tetapi suatu kegiatan yang terencana dan untuk mencapai tujuan kegiatan (Nurdin Usman, 2002:70).

Pengertian-pengertian tersebut menunjukkan bahwa kata implementasi bermuara pada aktivitas, adanya aksi, tindakan, atau mekanisme suatu sistem. Ungkapan mekanisme itu sendiri mengandung arti bahwa implementasi bukan hanya sekadar aktivitas, namun juga suatu kegiatan yang terencana dan dilakukan secara sungguh-sungguh dengan berdasarkan acuan norma tertentu untuk mencapai suatu tujuan kegiatan. Oleh karena itu, implementasi tidak berdiri sendiri tetapi dipengaruhi oleh obyek berikutnya yaitu kurikulum.

Menurut Nurdin Usman (2002) menjelaskan pendekatan-pendekatan yang telah dikemukakan di atas memberikan tekanan pada proses. Implementasi adalah suatu proses, suatu aktivitas yang digunakan untuk mentransfer gagasan, program atau harapan yang dituangkan dalam bentuk kurikulum desain (tertulis) agar dilaksanakan sesuai dengan desain tersebut. Masing-masing pendekatan itu mencerminkan tingkat pelaksanaan yang berbeda.

## 3.4. Metode Dempster-Shafer

Metode ini berdasarkan pada teori *Dempster-Shafer*, yaitu suatu teori matematika untuk pembuktian (Sri Kusumadewi, 2003). Pembuktian tersebut berdasarkan *belief functions* (fungsi kepercayaan) dan *plausible reasoning*

(pemikiran yang masuk akal) yang kemudian digunakan untuk mengkombinasikan potongan-potongan informasi yang terpisah sebagai bukti untuk menghitung sebuah kemungkinan dari suatu kejadian. Teori ini dikembangkan oleh Arthur P. Dempster dan Glenn Shafer. Teori ini secara umum dituliskan ke dalam suatu interval :

**[Belief, Plausability]**

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

*Plausability* (Pl) dinotasikan sebagai berikut:

**(Pl)s = 1 – Bel (¬ s)** .....(2.1)

*Plausibility* juga bernilai 0 sampai 1. Jika kita yakin akan ¬ s maka dapat dikatakan bahwa:

**Bel(¬ s) = 1 dan Pl(¬ s) = 0** .....(2.2)

*Plausibility* akan mengurangi tingkat kepercayaan dari *evidence*.

Pada teori *Dempster-Shafer* dikenal adanya *Frame of Discernment* (FOD) yang dinotasikan dengan Θ dan *mass function* yang dinotasikan dengan m. Sehingga fungsi dari kombinasi m1 dan m2 sebagai m3 dibentuk melalui persamaan :

$$\mathbf{m3(Z)} = \frac{\sum_{X \cap Y = Z} \boldsymbol{m1(X).m2(Y)}}{\mathbf{1} - \kappa} \quad \text{.....(2.3)}$$

dimana

$$\kappa = \sum_{X \cap Y = \emptyset} \boldsymbol{m1(X).m2(Y)} \quad \text{.....(2.4)}$$

dengan :

m1(X) adalah *mass function* dari *evidence* X

m2(Y) adalah *mass function* dari *evidence* Y

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

κ adalah jumlah *conflict evidence*

Beberapa kemungkinan range antara *Belief* dan *Plausibility* adalah:

Tabel 2.1. *Range Belief* dan *Plausibility*

| Kemungkinan | Keterangan |
|---|---|
| [1,1] | Semua Benar |
| [0,0] | Semua Salah |
| [0,1] | Ketidakpastian |
| [Bel,1] where 0 < Bel < 1 | Cenderung Mendukung |
| [0,Pls] where 0 < Pls < 1 | Cenderung Menolak |
| [Bel,Pls] where 0 < Bel ≤ Pls < 1 | Cenderung Mendukung dan Menolak |

*Frame of Discernment* Θ merupakan semesta pembicaraan dari sekumpulan hipotesis sehingga sering disebut dengan *environment*:

$$\Theta = \{\theta 1, \theta 2, .... \theta n\} \quad (2.5)$$

dimana:

Θ = FOD atau *environment*

$\theta 1 .. \theta n$ = elemen/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 biasa dinotasikan dengan P(Θ), setiap elemen dalam *power set* ini memiliki nilai interval antara 0 sampai 1.

$$m = P(\Theta) \rightarrow [0,1] \quad (2.6)$$

sebagai contoh:

P(*hostile*) = 0,7

P(*non-hostile*) = 1 – 07 = 0,3

Pada contoh di atas *belief* dari *hostile* adalah 0,7 sedangkan *disbelief hostile* adalah 0,3. Dalam teori *Dempster-Shafer*, *disbelief* dalam *environment* biasanya dinotasikan m($\theta$).

Sedangkan *mass function* (m) dalam teori *Dempster-Shafer* adalah tingkat kepercayaan dari suatu *evidence* (gejala), sering disebut dengan *evidence measure* sehingga dinotasikan dengan (m).

### 3.5. Tanaman Padi

Tanaman padi termasuk dalam genus *Oryza L* merupakan tanaman rumput-rumputan dengan nama latinnya *Oryza Sativa*. Tanaman ini termasuk tanaman yang umurnya pendek, kurang dari satu tahun dan berproduksi sekali. Akarnya pun dibedakan menjadi empat yaitu (Ina Hasanah, 2007) :

1) Akar tunggang, yaitu akar yang tumbuh ketika benih berkecambah.
2) Akar serabut, akar yang mulai tumbuh ketika tanaman padi berumur 5-6 hari setelah tanam.
3) Akar rumput, keluar dari akar tunggang dan serabut yang merupakan saluran pada kulit akar dan berfungsi untuk menghisap air serta zat-zat makanan.
4) Akar tanjuk, yaitu akar yang tumbuh dari ruas batang padi yang paling bawah.

Padi jenis unggul pada umumnya memiliki batang yang pendek, atau setidaknya lebih pendek dari tanaman padi jenis lokal. Daun pada tanaman padi mempunyai ciri yang khas dengan adanya daun telinga, bersisik, dan berukuran panjang dengan lebar daun yang tergantung dari varietasnya. Bagian pada daun padi di antaranya (Ina Hasanah, 2007) :

1) Helaian, terletak di batang padi dan bentuknya memanjang seperti pita. Ukuran panjang dan lebarnya berbeda-beda tergantung varietas padi.
2) Pelepah, merupakan bagian daun yang menyelubungi batang dan berfungsi untuk melindungi bagian ruas daun yang memiliki jaringan lunak.
3) Lidah daun, terletak pada perbatasan antara helai daun dan upih. Panjang dan warnanya berbeda-beda tergantung varietas padi.

### 3.6. Sistem Pakar

Sistem Pakar merupakan satu cabang dari Kecerdasan Buatan (*Artificial Intelligence*) yang membuat penggunaan secara pengetahuan yang khusus untuk menyelesaikan masalah. Sedangkan seorang pakar adalah orang yang memiliki suatu keahlian di bidang tertentu atau dengan kata lain, orang tersebut memiliki pengetahuan atau kemampuan tertentu yang tidak dimiliki oleh orang lain dan keahlian tersebut diperolehnya dari pengetahuan serta pengalamannya.

Secara umum, sistem pakar (*expert system*) adalah sistem yang berusaha mengadopsi pengetahuan manusia ke komputer, agar komputer dapat menyelesaikan masalah seperti yang biasa dilakukan oleh para ahli. Sistem pakar yang baik dirancang agar dapat menyelesaikan suatu permasalahan tertentu dengan meniru kerja dari para ahli. Struktur sistem pakar terdiri dari dua pokok (Sri Kusumadewi, 2003) yaitu:

1) Lingkungan pengembang (*development environment*)
   Lingkungan pengembang digunakan sebagai pembangunan sistem pakar baik dari segi pembangunan komponen maupun basis pengetahuan.
2) Lingkungan konsultasi (*consulatation environment*)
   Lingkungan konsultasi digunakan oleh seseorang bukan ahli untuk berkonsultasi.

### 3.7. Android

Android adalah *subset* perangkat lunak untuk perangkat *mobile* yang meliputi sistem operasi, *middleware*, dan aplikasi inti yang berbasis *Linux*. Android memakai basis kode komputer yang bisa didistribusikan secara terbuka (*open source*) sehingga pengguna bisa membuat aplikasi baru di dalamnya. Android SDK (*Sotftware Development Kit*) menyediakan *tools* dan API (*Application Programming Interface*) yang diperlukan untuk mengembangkan aplikasi pada *platform* Android dengan menggunakan bahasa pemrograman Java (Mulyadi, 2010).

Menurut Teguh Arifianto (2011) Android merupakan perangkat bergerak pada sistem operasi untuk telepon seluler yang berbasis Linux. Android merupakan *Mobile-OS* (*Operating System*) yang tumbuh ditengah OS lainnya yang berkembang dewasa ini. OS lainnya seperti Windows Mobile, i-Phone OS, Symbian, dan masih banyak lagi. Akan tetapi, OS yang ada ini berjalan dengan memprioritaskan aplikasi inti yang dibangun sendiri tanpa melihat potensi yang cukup besar dari aplikasi pihak ketiga.

Oleh karena itu, adanya keterbatasan dari aplikasi pihak ketiga untuk mendapatkan data asli ponsel, berkomunikasi antar proses serta keterbatasan

distribusi aplikasi pihak ketiga untuk *platform* mereka. Berdasarkan pendapat di atas, maka dapat ditarik kesimpulan bahwa Android adalah sistem operasi berbasis Linux yang sedang berkembang ditengah OS lainnya.

Sedangkan karakteristik dari Android itu sendiri ada empat (Nazruddin Safaat, 2012) yaitu:

1) Terbuka

   Android dibangun untuk benar-benar terbuka sehingga sebuah aplikasi dapat memanggil salah satu fungsi inti ponsel seperti membuat panggilan, mengirim pesan teks, menggunakan kamera dan lain-lain. Android merupakan sebuah *virtual machine* yang dirancang khusus untuk mengoptimalkan sumber daya memori dan perangkat keras yang terdapat di dalam perangkat. Android merupakan *open source*, dapat secara bebas diperluas untuk memasukkan teknologi baru yang lebih maju pada saat teknologi tersebut muncul.

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) Memecahkan hambatan pada aplikasi

   Android memecah hambatan untuk membangun aplikasi yang baru dan inovatif. Misalnya, pengembang dpat menggabungkan informasi yang diperoleh dari web dengan data pada ponsel seseorang seperti kontak pengguna, kalender atau lokasi geografis.

4) Pengembangan aplikasi yang cepat dan mudah

   Android menyediakan akses yang sangat luas kepada pengguna untuk menggunakan aplikasi yang semakin baik. Android memiliki sekumpulan *tools* yang dapat digunakan sehingga membantu para pengembang dalam meningkatkan produktivitas pada saat membangun aplikasi yang dibuat.

Perkembangan Android hingga laporan ini disusun dapat dilihat pada tabel berikut:

Tabel 2.2. Perkembangan Android OS

| Platform Version | API Level | Version Code |
|---|---|---|
| Android 6.0 | 23 | MARSHMALLOW |
| Android 5.1 | 22 | LOLLIPOP_MR1 |
| Android 5.0 | 21 | LOLLIPOP |
| Android 4.4W | 20 | KITKAT_WATCH |
| Android 4.4 | 19 | KITKAT |
| Android 4.3 | 18 | JELLY_BEAN_MR2 |
| Android 4.2<br>Android 4.2.2 | 17 | JELLY_BEAN_MR1 |
| Android 4.1<br>Android 4.1.1 | 16 | JELLY_BEAN |
| Android 4.0.3<br>Android 4.0.4 | 15 | ICE_CREAM_SANDWICH_MR1 |
| Android 4.0<br>Android 4.0.1<br>Android 4.0.2 | 14 | ICE_CREAM_SANDWICH |
| Android 3.2 | 13 | HONEYCOMB_MR2 |
| Android 3.1.x | 12 | HONEYCOMB_MR1 |
| Android 3.0.x | 11 | HONEYCOMB |
| Android 2.3.4<br>Android 2.3.3 | 10 | GINGERBREAD_MR1 |

Lanjutan Tabel 2.2. Perkembangan Android OS

| Platform Version | API Level | Version Code |
|---|---|---|
| Android 2.3.2<br>Android 2.3.1<br>Android 2.3 | 9 | GINGERBREAD |
| Android 2.2.x | 8 | FROYO |
| Android 2.1.x | 7 | ECLAIR_MR1 |
| Android 2.0.1 | 6 | ECLAIR_0_1 |
| Android 2.0 | 5 | ECLAIR |
| Android 1.6 | 4 | DONUT |
| Android 1.5 | 3 | CUPCAKE |
| Android 1.1 | 2 | BASE_1_1 |
| Android 1.0 | 1 | BASE |

### 3.8. Java ™

Java merupakan bahasa pemrograman yang dikembangkan agar mampu berjalan di atas berbagai platform perangkat keras dan perangkat lunak yang berbeda. Hal ini merupakan terobosan yang cukup besar, sebab aplikasi sebelumnya dikembangkan hanya untuk sistem operasi tertentu. Bahasa pemrograman Java saat ini bisa digunakan untuk mengembangkan aplikasi desktop, yaitu menggunakan JSE (*Java Standard Edition*) dan aplikasi yang berjalan di internet, yaitu menggunakan JEE (*Java Enterprise Edition*).

### 3.9. Database

*Database* sering didefinisikan sebagai kumpulan data yang terkait. Secara teknis, yang berada dalam sebuah *database* adalah sekumpulan tabel atau objek lain (*index*, *view*, dan lain-lain). Tujuan utama pembuatan *database* adalah untuk memudahkan dalam mengakses data. Data dapat ditambahkan, diubah, dihapus, atau dibaca dengan relatif mudah dan cepat (Abdul Kadir 2009).

### 3.10. SQLite

SQLite adalah salah satu software yang *embedded*, kombinasi SQL *interface* dan penggunaan *memory* yang sangat sedikit dengan kecepatan yang sangat cepat merupakan salah satu keunggulan SQLite. SQLite di Android termasuk dalam *Android Runtime*, sehingga setiap versi dari Android dapat membuat *database* dengan SQLite (Nazruddin Safaat, 2012).

### 3.11. UML (*Unified Modelling Language*)

Banyak orang yang telah membuat bahasa pemodelan pembangunan perangkat lunak yang sesuai dengan teknologi pemograman yang berkembang pada saat itu, misalnya yang sempat berkembang dan digunakan oleh banyak pihak adalah *Data Flow Diagram* (DFD) untuk memodelkan perangkat lunak yang menggunakan pemograman prosedural atau struktural, kemudian juga ada *State Transition Diagram* (STD) yang digunakan untuk memodelkan sistem *real time* (Rosa AS dan M Shalahudin, 2013).

UML (*Unified Modelling Language*) merupakan bahasa yang telah menjadi standar dalam industri untuk visualisasi, merancang, dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem (Rosa AS dan M Shalahudin, 2013).

UML berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada metodelogi tertentu, meskipun pada kenyataannya UML paling banyak diguanakan pada metodologi berorientasi objek. Begitu juga dengan perkembangan UML bergantung pada level abstraksi penggunaannya. Jadi, belum tentu pandangan yang berbeda dalam penggunaan UML ada suatu yang salah, tapi perlu ditelaah dimanakah UML digunakan dan hal apa yang ingin divisualkan (Rosa AS dan M Shalahudin, 2013).

#### 3.11.1. *Use Case Diagram*

Menurut Rosa AS dan M Shalahudin (2013) *Use case diagram* menyajikan interaksi antara *use case* dan aktor. Dimana, aktor dapat berupa orang, peralatan, atau sistem lain yang berinteraksi dengan sistem yang sedang dibangun. *Use case*

menggambarkan fungsional sistem atau persyaratan-persyaratan yang harus dipenuhi sistem dari pandangan pemakai.

Tabel 2.3. Simbol *Use Case Diagram*

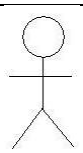
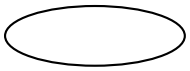
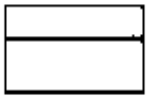
| Simbol | Keterangan |
|---|---|
|  | *Actor;* Sebuah peran yang dimainkan oleh seseorang, sistem, atau perangkat yang memiliki saham dalam keberhasilan operasi dari sistem. |
|  | *Use Case*; Untuk mengungkapkan tujuan bahwa sistem harus dicapai. |
|  | *Association*; Mengidentifikasi interaksi antara aktor dan *Use Case* |
|  | *Dependency;* Mengidentifikasi hubungan komunikasi antara dua *Use Case*. |

Diagram aktivitas atau *activity diagram* yang menggambarkan aliran fungsional sistem. Pada tahap pemodelan bisnis, diagram aktivitas dapat digunakan untuk menunjukan aliran kerja bisnis (*business work-flow*). *Use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut.

### **3.11.2.** *Class Diagram*

Menurut Rosa AS dan M Shalahudin (2013), *Class diagram* atau diagram kelas merupakan suatu diagram yang menggambarkan struktur sistem dari segi pendefinisian *class* yang akan dibuat untuk membangun sistem. *Class* memiliki apa yang disebut nama (dan *stereotype*), atribut dan mengandung metode atau operasi. Atribut merupakan variable-variabel yang dimiliki oleh suatu *class*. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu *class*. Dan untuk simbol *class diagram* kurang lebih adalah sebagai berikut :

Tabel 2.4. Simbol *Class Diagram*

| Simbol | Nama | Keterangan |
|---|---|---|
|  | *Generalization* | Hubungan di mana objek anak berbagi perilaku dan struktur data dari objek yang di atasnya yaitu objek induk (ancestor) |
|  | *Class* | Himpunan dari objek-objek yang berbagi atribut serta operasi yang sama |
|  | *Association* | Relasi antar kelas dengan makna umum, asosiasi biasanya juga disertai dengan *multiplicity*. |
|  | *Collaboration* | Interaksi aturan dan elemen lain yang bekerja sama untuk menyediakan perilaku yang lebih besar dari jumlah dan elemennya |
|  | *Realization* | Operasi yang benar-benar dilakukan oleh suatu objek |
|  | *Dependency* | Hubungan di mana perubahan yang terjadi pada suatu elemen mandiri akan mempengaruhi elemen yang bergantung padanya |

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

### 3.11.3. *Activity Diagram*

Menurut Rosa AS dan M Shalahudin (2013), diagram aktivitas atau *activity diagram* menggambarkan *workflow* atau aktivitas dari sebuah sistem atau proses bisnis. Yang perlu diperhatikan disini bahwa diagram aktivitas menggambarkan aktivitas sistem, bukan menggambarkan aktivitas apa yang dilakukan oleh aktor.

Tabel 2.5. Simbol *Activity Diagram*

| Simbol | Nama | Keterangan |
|---|---|---|
|  | *Initital state* | Status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah status awal. |
|  | *Activity* | Aktivitas yang dilakukan sistem biasanya diawali dengan kata kerja. |
|  | *Decision* | Asosiasi percabangan dimana jika ada pilihan aktivitas lebih atau satu. |
|  | *Fork Node / Join* | Asosiasi penggabungan dimana lebih dari satu aktivitas digabungkan menjadi satu. |
|  | *Final State* | Status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki sebuah status akhir. |

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

a. Rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
b. Urutan atau pengelompokkan tampilan dari sistem / *user interface* (UI) di mana setiap aktivitas dianggap memiliki sebuah rancangan UI.
c. Rancangan pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.

### **3.11.4.** *Sequence Diagram*

Menurut Rosa AS dan M Shalahudin (2013), *Sequence diagram* menjelaskan secara detail urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari *use case*: interaksi yang terjadi antar *class*, operasi apa saja yang terlibat, urutan antar operasi, dan informasi yang diperlukan oleh masing-masing operasi. Pembuatan *sequence diagram* merupakan aktivitas yang paling kritikal dari proses desain karena artifak inilah yang menjadi pedoman dalam proses pemrograman nantinya dan berisi *control flow* dari program. Diagram ini disebut juga diagram urutan karena untuk disusun berdasarkan urutan operasi pada sistem yang menekankan pada pengiriman pesan dalam waktu tertentu.

Tabel 2.6. Simbol *Sequence Diagram*

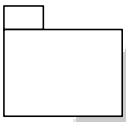
| Simbol | Nama | Keterangan |
|---|---|---|
|  Nama Aktor | *Actor* | Orang, proses atau sistem lain yang berinteraksi dengan sistem informasi lain diluar sistem informasi itu sendiri; biasanya dinyatakan menggunakan kata benda di awal frase nama actor |
|  | *Lifeline* | Menyatakan kehidupan suatu objek |
| Nama objek | *Objek* | Menyatakan objek yang berinteraksi |
|  | *Waktu aktif* | Menyatakan objek dalam keadaan aktif dan berinteraksi pesan |
| Message()  | *Message* | Spesifikasi dari komunikasi antar objek yang memuat informasi tentang aktifitas yang terjadi |
|  | *Boundary* | Digunakan untuk menggambarkan form |
|  | *Control Class* | Digunakan untuk menghubungkan boundary dengan tabel pada database |
|  | *Entity Class* | Digunakan untuk menggambarkan hubungan kegiatan yang akan dilakukan |
| X | *Pesan tipe destroy* | Menyatakan akhir hidup suatu objek |

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

### 3.11.5. *Deployment Diagram*

Diagram yang menunjukan konfigurasi komponen dalam proses ekskusi aplikasi. *Deployment* diagram menggambarkan *detail* bagaimana komponen di-*deploy* dalam infrastruktur sistem, dimana komponen akan terletak (pada mesin, *server* atau piranti keras), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server dan hal-hal lain yang bersifat fiskal. (Rosa AS dan M Shalahudin 2013).

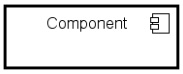
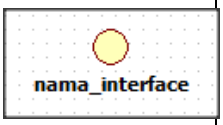
Tabel 2.7. Simbol *Deployment Diagram*

| NO | GAMBAR | NAMA | KETERANGAN |
|---|---|---|---|
| **1** |  | *Package* | *Package* merupakan sebuah bungkusan dari satu atau lebih *node* |
| **2** |  | *Node* | Biasanya mengacu pada perangkat keras (*hardware*), perangkat lunak yang tidak dibuat sendiri (*software*), jika didalam *node* disertakan komponen untuk mengkonsistensikan rancangan maka komponen yang diikutsertakan harus sesuai dengan komponen yang telah didefinisikan sebelumnya pada diagram komponen |
| **3** |  | *Depedency* | Ketergantungan antara *node*, arah panah mengarah pada *node* yang dipakai |
| **4** |  | *Link* | Relasi antar *node* |

### 3.11.6. *Component Diagram*

*Component diagram* adalah diagram UML yang menampilkan komponen dalam sistem dan hubungan antara mereka. Pada *component* View, akan difokuskan pada organisasi fisik sistem. Pertama, diputuskan bagaimana *class* akan diorganisasikan menjadi *library*. Kemudian akan dilihat bagaimana perbedaan antara berkas eksekusi, berkas *dynamic link library* (DDL), dan berkas *runtime* lainnya dalam system.

Tabel 2.8. Simbol *Component Diagram*

| Simbol | Nama | Keterangan |
|---|---|---|
|  | *Component* | Komponen sistem. |
|  | *Dependency* | Kebergantungan antar komponen, arah panah mengarah pada komponen yang dipakai. |
|  | *Interface* | Sebagai antarmuka komponen agar tidak mengakses langsung komponen. |
|  | *Link* | Relasi antar komponen. |

### 3.12. *Rapid Access Development* (RAD)

RAD adalah salah satu alternatif dari *System Development Life Cycle* (SDLC) yang ditujukan untuk menyediakan pengembangan yang jauh lebih cepat dan mendapatkan hasil dengan kualitas yang lebih baik dibandingkan dengan hasil yang dicapai melalui siklus tradisional. Metode ini adalah sebuah metode yang banyak digunakan dalam pengembangan aplikasi Android.

Menurut Rosa AS dan M Shalahudin (2013), model Rapid Application Development (RAD) adalah proses pengembangan prangkat lunak yang bersifat Inkremental terutama untuk waktu pengerjaan yang pendek. Model RAD adalah adaptasi dari *waterfall* versi kecepatan tinggi untuk pengembangan setiap komponen perangkat lunak, sedangkan menggunakan *Unified Modeling Language* (UML) sebagai alat pemodelannya. RAD adalah model proses pembangunan prangkat lunak yang tergolong dalam teknik *incremental* (bertingkat).

### 3.13. Pengujian *Black Box*

Pengujian *black box* "Yaitu menguji perangkat lunak dari segi spesifikasi fungsional tanpa menguji desain dan kode program". Pengujian dimaksudkan untuk

mengetahui apakah fungsi-fungsi, masukan dan keluaran dari perangkat lunak sesuai dengan *spesifikasi* yang di butuhkan. Pengujian kotak hitam dilakukan dengan membuat kasus uji yang bersifat mencoba semua fungsi dengan memakai perangkat lunak apakah sesuai dengan spesifikasi yang dibutuhkan. (Rosa AS dan M Shalahudin, 2013). Pengujian dimaksudkan untuk mengetahui apakah fungsi-fungsi, masukan, dan keluaran dari perangkat lunak sesuai dengan spesifikasi yang dibutuhkan.