

## **BAB II LANDASAN TEORI**

Teori-teori dasar yang melandasi dalam pembuatan Sistem Informasi Elektronik Mading (E-Mading) IBI Darmajaya Berbasis Android dijelaskan pada sub-sub sistem di bawah ini.

### **2.1 Sistem**

Pada dasarnya, sistem adalah sekumpulan elemen yang saling terkait atau terpadu yang dimaksudkan untuk mencapai suatu tujuan. Sebagai gambaran, jika sebuah sistem terdapat elemen yang tidak memberikan manfaat dalam mencapai tujuan yang sama, maka elemen tersebut dapat dipastikan bukanlah bagian dari sistem (Kadir, 2014).

### **2.2 Informasi**

McFadden, mendefinisikan informasi sebagai data yang telah diproses sedemikian rupa sehingga meningkatkan pengetahuan seseorang yang menggunakan data tersebut. Shannon dan Weaver, dua orang insinyur listrik melakukan pendekatan secara matematis untuk mendefinisikan informasi (Kroenke). Menurut Shannon dan Weaver, informasi adalah jumlah ketidak pastian yang dikurangi ketika sebuah pesan diterima. Artinya, dengan adanya informasi, tingkat kepastian menjadi meningkat. Menurut Davis, informasi adalah data yang telah diolah menjadi sebuah bentuk yang berarti bagi penerimanya dan bermanfaat dalam pengambilan keputusan saat ini atau saat mendatang (Kadir, 2014).

### **2.3 Sistem Informasi**

Sesungguhnya yang dimaksud sistem informasi tidak harus melibatkan komputer. Sistem informasi yang menggunakan komputer biasa disebut sistem informasi berbasis komputer (*Computer Based Information System* atau CBIS). Dalam praktik, istilah sistem informasi lebih sering dipakai tanpa embel-embel berbasis

komputer, walaupun dalam kenyataannya komputer merupakan bagian yang penting. Di buku ini, yang dimaksudkan dengan sistem informasi adalah sistem informasi berbasis komputer. Ada beragam definisi sistem informasi, yaitu :

- a. Alter, sistem informasi adalah kombinasi antar prosedur kerja, informasi, orang dan teknologi informasi yang diorganisasikan untuk mencapai tujuan dalam sebuah organisasi.
- b. Bodnar dan Hopwoo, sistem informasi adalah kumpulan perangkat keras dan perangkat lunak yang dirancang untuk mentransformasikan data ke dalam bentuk informasi yang berguna.
- c. Gelinas, Oram dan Wiggins, sistem informasi adalah suatu sistem buatan manusia yang secara umum terdiri atas sekumpulan komponen berbasis komputer dan manual yang dibuat untuk menghimpun, menyimpan dan mengelola data serta menyediakan informasi keluaran kepada para pemakai.
- d. Hall, Sistem informasi adalah sebuah rangkaian prosedur formal, dimana data dikelompokkan, diproses menjadi informasi dan didistribusikan kepada para pemakai.
- e. Turban, McLean dan Wetherbe, Sebuah sistem informasi mengumpulkan, memproses, menyimpan, menganalisis dan menyebarkan informasi untuk tujuan yang spesifik.
- f. Wilkinson, Sistem informasi adalah kerangka kerja yang mengkoordinasikan sumber daya (manusia dan komputer) untuk mengubah masukan (*input*) menjadi keluaran (informasi) guna mencapai sasaran-sasaran perusahaan.

Berdasarkan berbagai definisi tersebut, dapat disimpulkan bahwa sistem informasi mencakup sejumlah komponen (manusia, komputer, teknologi informasi dan prosedur kerja), ada sesuatu yang diproses (data menjadi informasi) dan dimaksudkan untuk mencapai suatu sasaran atau tujuan (Kadir, 2014).

## **2.4 Pemograman Berorientasi Objek**

Metodologi berorientasi objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai komponen objek yang berisi

data dan operasi yang diberlakukan terhadapnya (Rosa, 2011). Metodologi berorientasi objek merupakan suatu cara bagaimana sistem perangkat lunak dibangun melalui pendekatan objek secara sistematis. Metodologi berorientasi objek didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas, yang meliputi rangkaian aktivitas analisis berorientasi objek, perancangan berorientasi objek, pemrograman berorientasi objek dan pengujian berorientasi objek. Keuntungan menggunakan metodologi pemrograman berorientasi objek adalah sebagai berikut :

- a. Meningkatkan produktivitas, karena kelas dan objek yang ditemukan dalam suatu masalah masih dapat dipakai ulang untuk masalah lainnya yang melibatkan objek tersebut (*reusable*).
- b. Kecepatan pengembangan, karena sistem yang dibangun baik dan benar pada saat analisis dan perancangan akan menyebabkan berkurangnya kesalahan pada saat pengodean.
- c. Kemudahan pemeliharaan, pola-pola yang cenderung tetap dan stabil dapat dipisahkan dan pola-pola yang mungkin sering berubah-ubah.
- d. Adanya konsistensi, karena sifat pewarisan dan pengurangan notasi yang sama pada saat analisis, perancangan maupun pengodean.
- e. Meningkatkan kualitas perangkat lunak, karena pendekatan pengembangan lebih dekat dengan dunia nyata dan adanya konsistensi pada saat pengembangannya, perangkat lunak yang dihasilkan akan mampu memenuhi kebutuhan pemakai serta mempunyai sedikit kesalahan.

## **2.5 Konsep Dasar Berorientasi Objek**

Dalam rekayasa perangkat lunak, konsep pendekatan berorientasi objek dapat diterapkan pada tahap analisis, perancangan, pemrograman dan pengujian perangkat lunak (Rosa, 2011). Beberapa konsep dasar yang harus dipahami tentang metodologi berorientasi objek adalah sebagai berikut :

- a. Kelas (*class*)

Kelas adalah kumpulan objek-objek dengan karakteristik yang sama dan memiliki sifat (atribut). Secara teknis, kelas adalah sebuah struktur tertentu

dalam pembuatan perangkat lunak. Kelas merupakan bentuk struktur pada kode program yang menggunakan metodologi berorientasi objek.

b. Objek (*object*)

Objek adalah abstraksi dan sesuatu yang mewakili dunia nyata seperti benda, manusia, satuan organisasi, tempat, kejadian, struktur dan hal-hal lainnya yang bersifat abstrak. Objek merupakan suatu entitas yang mampu menyimpan informasi (status) dan mempunyai operasi (kelakuan) yang dapat berpengaruh pada status objeknya.

c. Metode (*method*)

Operasi atau metode sebuah kelas hampir sama dengan fungsi atau prosedur pada metodologi struktural. Sebuah kelas boleh memiliki lebih dari satu metode atau operasi.

d. Atribut (*attribute*)

Atribut dapat berupa nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas.

e. Abstraksi (*abstraction*)

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

f. Enkapulasi (*encapsulation*)

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

g. Pewarisan (*inheritance*)

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

h. Antarmuka (*interface*)

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

i. *Reusability*

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

j. Generalisasi dan Spealisasi

Menunjukkan hubungan antara kelas dan objek yang umum dengan kelas dan objek yang khusus.

k. Komunikasi Antar Objek

Komunikasi antar objek dilakukan lewat pesan yang dikirim dari satu objek ke objek lainnya.

l. Poliformisme (*polymorphism*)

Kemampuan suatu objek untuk digunakan dibanyak tujuan yang berbeda dengan nama yang sama sehingga menghemat baris program.

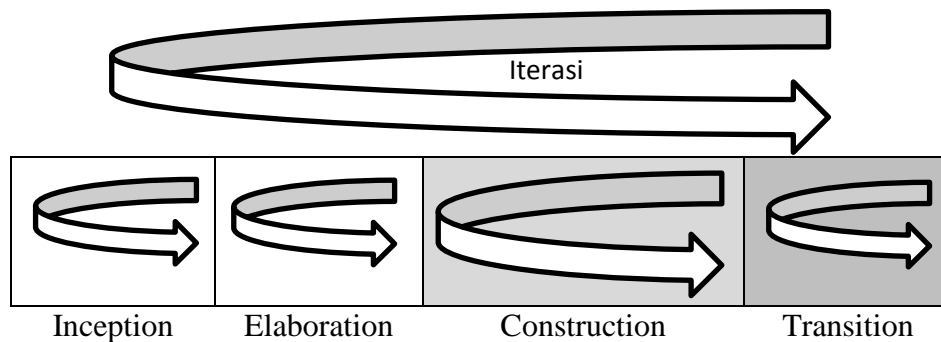
m. *Package*

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

## 2.6 Metode Pengembangan Sistem

*Unified Process* atau dikenal juga dengan proses iteratif dan inkremental merupakan sebuah proses pengembangan perangkat lunak yang dilakukan secara iteratif (berulang) dan inkremental (bertahap dengan proses menaik). Iteratif bisa dilakukan di dalam setiap tahap atau iteratif tahap pada proses pengembangan perangkat lunak untuk menghasilkan perbaikan fungsi yang inkremental, dimana setiap iterasi akan memperbaiki iterasi berikutnya (Rosa, 2011). Salah satu *Unified Process* yang terkenal adalah RUP (*Rational Unified Process*).

RUP adalah pendekatan pengembangan perangkat lunak yang dilakukan berulang-ulang, fokus pada arsitektur, lebih diarahkan berdasarkan penggunaan kasus (*use case driven*). RUP merupakan proses rekayasa perangkat lunak dengan pendefinisian yang baik dan penstrukturan yang baik. RUP memiliki empat buah tahap fase, yaitu seperti pada Gambar 2.1.



Gambar 2.1 Alur Hidup RUP

a. *Inception* (permulaan)

Tahap ini lebih pada memodelkan bisnis yang dibutuhkan dan mendefinisikan kebutuhan akan sistem yang akan dibuat. Tahap yang dibutuhkan pada permulaan ini adalah :

1. Memahami ruang lingkup dari proyek (termasuk biaya, waktu, kebutuhan, resiko dan lainnya).
2. Membangun kasus bisnis yang dibutuhkan.

Hasil yang diharapkan pada tahap ini adalah memenuhi *lifecycle objective milestone* (batas/tonggak objektif dari siklus) dengan kriteria berikut :

1. Umpan balik dari pendefinisian ruang lingkup, perkiraan biaya dan perkiraan jadwal.
2. Kebutuhan dimengerti dengan pasti dan sejalan dengan kasus primer yang dibutuhkan.
3. Kredibilitas dari perkiraan biaya, perkiraan jadwal, penentuan skala prioritas, risiko dan proses pengembangan.
4. Ruang lingkup purwarupa (*prototype*) yang akan dikembangkan.
5. Membangun garis dasar dengan membandingkan perencanaan aktual dengan perencanaan yang direncanakan.

Jika pada akhir tahap ini target yang diinginkan tidak dicapai maka dapat dibatalkan atau diulang kembali setelah dirancang ulang agar kriteria yang diinginkan dapat dicapai.

b. *Elaboration* (perluasan atau perencanaan)

Tahap ini lebih difokuskan pada perencanaan arsitektur sistem. Tahap ini lebih pada analisis dan desain sistem serta implementasi sistem yang fokus pada purwarupa sistem (*prototype*). Hasil yang diharapkan pada tahap ini adalah memenuhi *lifecycle objective milestone* (batas/tonggak objektif dari siklus) dengan kriteria berikut :

1. Model kasus yang digunakan (*use case*) dimana kasus dan aktor yang terlihat telah didefinisikan dan sebagian besar kasus harus dikembangkan.
2. Deskripsi dari arsitektur perangkat lunak telah dibuat.
3. Rancangan arsitektur yang dapat diimplementasikan dan mengimplementasikan *use case*.
4. Kasus bisnis atau proses bisnis dan daftar resiko yang sudah mengalami perbaikan.
5. Rencana pengembangan untuk seluruh proyek telah dibuat.
6. Purwarupa (*prototype*) yang dapat didemonstrasikan untuk mengurangi setiap resiko teknis yang diidentifikasi.

Jika pada akhir tahap ini target yang diinginkan tidak dicapai, maka dapat dibatalkan atau diulang kembali.

c. *Construction* (konstruksi)

Tahap ini fokus pada pengembangan komponen dan fitur-fitur sistem. tahap ini lebih pada implementasi dan pengujian sistem yang fokus pada implementasi perangkat lunak atau kode program. Tahap ini menghasilkan produk perangkat lunak dimana menjadi syarat dari *Initial Operational Capability Milestone* atau batas/tonggak kemampuan operasional awal.

d. *Transition* (transisi)

Tahap ini lebih pada *deployment* atau inialisasi sistem agar dapat dimengerti oleh *user*. Tahap ini menghasilkan produk perangkat lunak dimana menjadi syarat dari *Initial Operational Capability Milestone* atau batas/tonggak kemampuan operasional awal. Aktivitas pada tahap ini termasuk pada pelatihan *user*, pemeliharaan dan pengujian sistem.

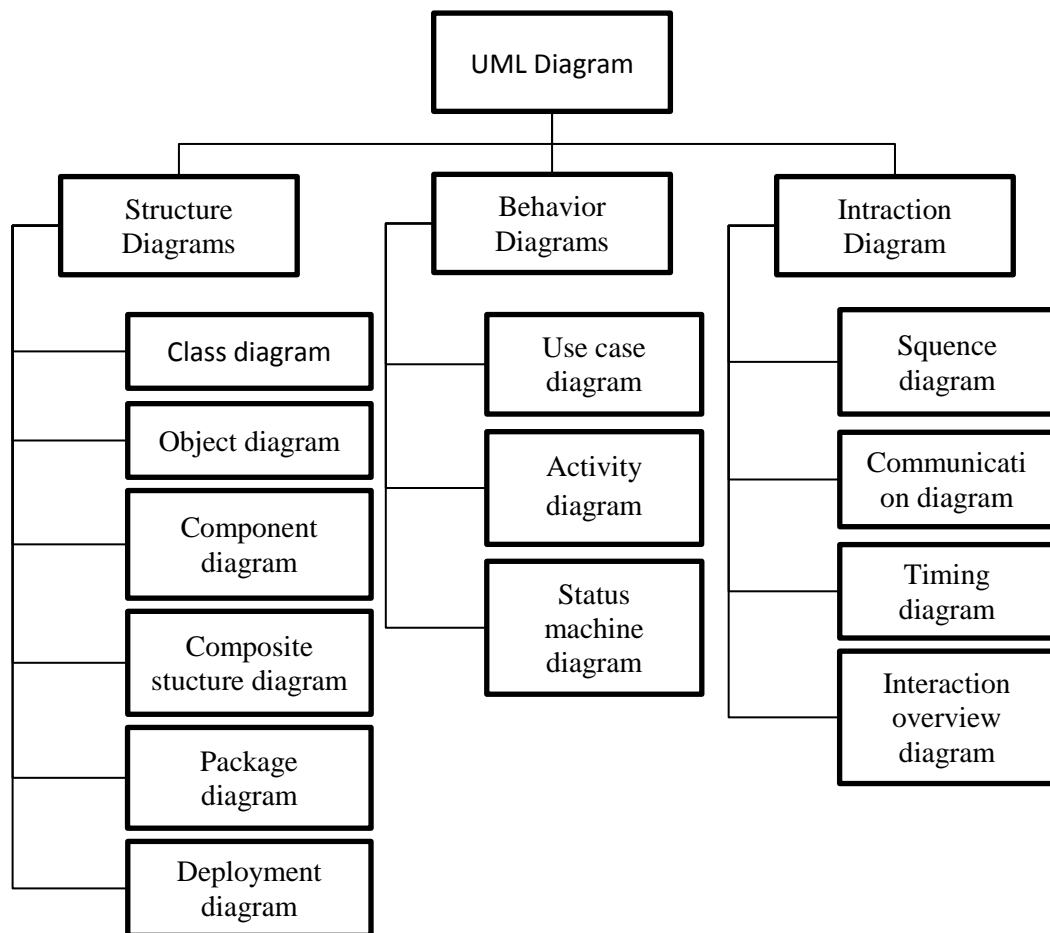
## 2.7 Alat Bantu Perancangan Sistem

### 2.7.1 UML

Banyak orang yang telah membuat bahasa pemodelan pembangunan perangkat lunak yang sesuai dengan teknologi pemrograman 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 pemrograman prosedural atau struktural, kemudian juga ada *State Transition Diagram* (STD) yang digunakan untuk memodelkan sistem *real time* (waktu nyata).

Pada perkembangan teknik pemrograman berorientasi objek, munculah sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modeling Language* (UML). UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun dan mendokumentasi dari sistem perangkat lunak. UML terdiri dari 13 macam diagram yang dikelompokkan dalam tiga kategori, yaitu seperti pada Gambar 2.2 (Rosa, 2011).





Gambar 2.2 Diagram UML

Penjelasan dari pembagian kategori tersebut adalah :



- a. *Structure diagram*, yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan.
- b. *Behavior diagram*, yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada sebuah sistem.
- c. *Interaction diagram*, yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antar sub sistem pada suatu sistem.

### 2.7.1.1 Use Case


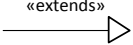



*Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu. Syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami (Rosa, 2011). Ada dua hal utama pada *use case* yaitu pendefinisian apa yang dibuat aktor dan *use case*.

- a. Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.
- b. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor.

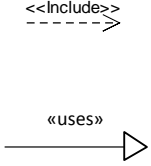
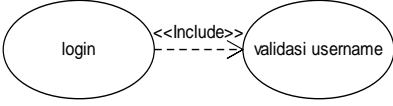
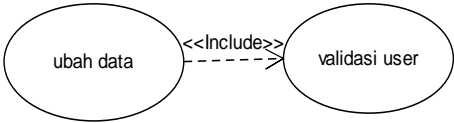
Tabel 2.1 Simbol *Use Case* Diagram

Keterangan	Simbol	Deskripsi
<i>Use Case</i>		Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor; biasanya dinyatakan dengan menggunakan kata kerja diawal-awal frase nama <i>use case</i>
Aktor		Orang, proses atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar itu sendiri. Jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda diawal frase nama aktor.

Tabel 2.1 Simbol *Use Case Diagram* (Lanjutan)

Asosiasi		Komunikasi antara aktor dan <i>use case</i> yang berpartisipasi pada <i>use case</i> atau <i>use case</i> memiliki interaksi dengan aktor.
Ekstensi		<p>Relasi <i>use case</i> tambahan ke sebuah <i>use case</i>, dimana <i>use case</i> yang ditambahkan dapat berdiri sendiri walau tanpa <i>use case</i> tambahan itu; mirip dengan prinsip <i>inheritance</i> pada pemrograman berorientasi objek; biasanya <i>use case</i> tambahan memiliki nama depan yang sama dengan <i>use case</i> yang ditambahkan, misal</p>  <p>Arah panah mengarah pada <i>use case</i> yang ditambahkan.</p>
Generalisasi		<p>Hubungan generalisasi dan spesialisasi (umum-khusus) antara dua buah <i>use case</i> dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya, misalnya :</p>  <p>Arah panah mengarah pada <i>use case</i> yang menjadi generalisasinya (umum).</p>

Tabel 2.1 Simbol *Use Case Diagram* (Lanjutan)





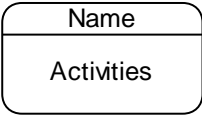

<p>Menggunakan <i>/include/uses</i></p>		<p>Ada dua sudut pandang yang cukup besar mengenai <i>include</i> di <i>use case</i> :</p> <p>a. Include berarti use case yang ditambahkan akan selalu dipanggil saat use case tambahan dijalankan, misal pada kasus berikut :</p>  <p>b. Include berarti use case yang tambahan akan selalu melakukan pengecekan apakah use case yang ditambahkan telah dijalankan sebelum use case tambahan dijalankan, misal pada kasus berikut :</p>  <p>Ke dua interpretasi di atas dapat dianut salah satu atau keduanya tergantung pada pertimbangan dan interpretasi yang dibutuhkan.</p>
---	---	---

### 2.7.1.2 Activity Diagram

Diagram aktivitas atau *activity diagram* menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Yang perlu diperhatikan disini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem. Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut :

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

Tabel 2.2 Simbol Diagram Aktivitas

Keterangan	Simbol	Deskripsi
Status awal		Status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah status awal.
Aktivitas		Aktivitas yang dilakukan sistem, aktivitas biasanya diawali dengan kata kerja.
Percabangan		Asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu.
Penggabungan		Asosiasi penggabungan dimana lebih dari satu aktivitas digabungkan menjadi satu.
<i>Swimlane</i>		Memisahkan organisasi bisnis yang bertanggung jawab terhadap aktivitas yang terjadi.
Status akhir		Status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki sebuah status akhir.

### 2.7.1.3 Class Diagram

Diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. kelas memiliki apa yang disebut atribut dan metode atau operasi. Atribut merupakan variabel-variabel yang dimiliki suatu kelas, sedangkan operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas (Rosa, 2011).

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

a. Kelas main

Kelas yang memiliki fungsi awal dieksekusi ketika sistem dijalankan.

b. Kelas yang menangani tampilan sistem

Kelas yang mendefinisikan dan mengatur tampilan ke pemakai.

c. Kelas yang diambil dari pendefinisian *use case*

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

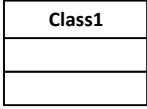
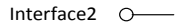


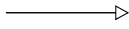
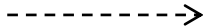

d. Kelas yang diambil dari pendefinisian data

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

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

Dalam mengidentifikasi metode yang ada di dalam kelas perlu memperhatikan apa yang disebut dengan *cohesion* dan *coupling*. *Cohesion* adalah ukuran seberapa dekat keterkaitan instruksi di dalam sebuah metode terkait satu sama lain, sedangkan *coupling* adalah ukuran seberapa dekat keterkaitan instruksi antara metode yang satu dengan metode yang lain dalam sebuah kelas. Sebagai aturan secara umum, maka sebuah metode yang dibuat harus memiliki kadar *cohesion* yang kuat dan kadar *coupling* yang lemah. Simbol-simbol yang ada pada diagram kelas adalah seperti pada Tabel 2.3.

Tabel 2.3 Simbol *Class Diagram*



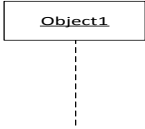

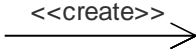
Simbol	Deskripsi
Kelas 	Kelas pada struktur sistem.
Natarmuka/ <i>interface</i> 	Sama dengan konsep <i>interface</i> dalam pemograman berorientasi objek.
Asosiasi 	Relasi antar kelas dalam makna umum, asosiasi biasanya juga disertai dengan <i>multiplicity</i> .
Asosiasi berarah 	Relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan <i>multiplicity</i> .
Generalisasi 	Relasi antar kelas dengan makna generalisasi-spesialisasi (umum-khusus).
Kebergantungan 	Relasi antar kelas dengan makna kebergantungan antar kelas.
Agregasi 	Relasi antar kelas dengan makna semua bagian ( <i>whole-part</i> ).

#### 2.7.1.4 Sequence Diagram

Diagram sekuen menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Banyaknya diagram sekuen yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interkasi jalannya pesan sudah dicakup pada diagram sekuen sehingga semakin banyak *use case* yang

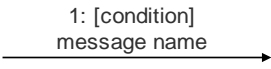
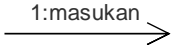
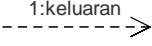
didefinisikan maka diagram sekuen yang harus dibuat juga semakin banyak (Rosa, 2011).

Tabel 2.4 Simbol *Sequence Diagram*

Simbol	Deskripsi
<p data-bbox="336 555 416 584">Aktor</p> 	<p data-bbox="719 555 1362 913">Orang, proses atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat diluar sistem informasi yang dibuat itu sendiri. Jadi, walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang biasanya dinyatakan menggunakan kata benda di awal frase nama aktor.</p>
<p data-bbox="336 945 491 974">Garis hidup</p> 	<p data-bbox="719 1003 1187 1032">Menyatakan kehidupan suatu objek.</p>
<p data-bbox="336 1131 416 1160">Objek</p> 	<p data-bbox="719 1234 1283 1263">Menyatakan objek yang berinteraksi pesan.</p>
<p data-bbox="336 1400 496 1429">Waktu aktif</p> 	<p data-bbox="719 1458 1283 1541">Menyatakan objek dalam keadaan aktif dan berinteraksi pesan.</p>
<p data-bbox="336 1635 560 1664">Pesan tipe <i>create</i></p> 	<p data-bbox="719 1666 1315 1809">Menyatakan suatu objek membuat objek yang lain, arah panah mengarah pada objek yang dibuat.</p>



Tabel 2.4 Simbol *Sequence Diagram* (Lanjutan)

<p>Pesan tipe <i>call</i></p> 	<p>Menyatakan suatu objek memanggil operasi/metode yang ada pada objek lain atau dirinya sendiri.</p> <p>Arah panah mengarah pada objek yang memiliki operasi atau metode karena ini memanggil operasi/metode maka operasi/metode yang dipanggil harus ada pada diagram kelas sesuai dengan kelas objek yang berinteraksi.</p>
<p>Pesan tipe <i>send</i></p> 	<p>Menyatakan bahwa suatu objek mengirimkan data/masukan/informasi ke objek lainnya, arah panah mengarah pada objek yang dikirim.</p>
<p>Pesan tipe <i>return</i></p> 	<p>Menyatakan bahwa suatu objek yang telah menjalankan suatu operasi atau metode yang menghasilkan suatu kembalian ke objek tertentu, arah panah mengarah pada objek yang menerima kembalian.</p>

### 2.7.2 Basis Data

Basis data (*database*) adalah suatu pengorganisasian sekumpulan data yang saling terkait sehingga memudahkan aktivitas untuk memperoleh informasi. Basis data di maksudkan untuk mengatasi problem pada sistem yang memakai pendekatan berbasis berkas.

Untuk mengelola basis data diperlukan perangkat lunak yang disebut *Database Management System* (DBMS). DBMS adalah perangkat lunak sistem yang memungkinkan para pemakai membuat, memelihara, mengontrol, dan mengakses basis data dengan cara yang praktis dan efisien. DBMS dapat digunakan untuk mengakomodasikan berbagai macam pemakai yang memiliki kebutuhan akses yang berbeda-beda.

Umumnya DBMS menyediakan fitur-fitur sebagai berikut :

a. Independensi data program

Karena basis data ditangani oleh DBMS, program dapat ditulis sehingga tidak tergantung pada struktur data dalam basis data. Dengan perkataan lain, program tidak akan terpengaruh sekiranya bentuk fisik data diubah.

b. Keamanan

Keamanan dimaksudkan untuk mencegah pengaksesan data oleh orang yang tidak berwenang.

c. Integritas

Hal ini ditujukan untuk menjaga agar data selalu dalam keadaan yang *valid* dan konsisten.

d. Konkurensi

Konkurensi memungkinkan data dapat diakses oleh banyak pemakai tanpa menimbulkan masalah.

e. Pemulihan (*recovery*)

DBMS menyediakan mekanisme untuk mengembalikan basis data ke keadaan semula yang konsisten sekiranya terjadi gangguan perangkat keras atau kegagalan perangkat lunak.

f. Katalog sistem

Katalog sistem adalah deskripsi tentang data yang terkandung dalam basis data yang dapat diakses oleh pemakai.

g. Perangkat produktivitas

Untuk menyediakan kemudahan bagi pemakai dan meningkatkan produktivitas, DBMS menyediakan sejumlah perangkat produktivitas seperti pembangkit *query* dan pembangkit laporan.

Komponen-komponen yang menyusun lingkungan DBMS terdiri atas:

a. Perangkat keras

Perangkat keras digunakan untuk menjalankan DBMS beserta aplikasi-aplikasinya. Perangkat keras berupa komputer dan periferal pendukungnya. Komputer dapat berupa PC, minikomputer, mainframe, dan lain-lain.

b. Perangkat lunak

Komponen perangkat lunak mencakup DBMS itu sendiri, program aplikasi, serta perangkat lunak pendukung untuk komputer dan jaringan. Program aplikasi dapat dibangun dengan menggunakan bahasa pemrograman seperti *C++*, *Pascal*, *Delphi*, atau *Visual BASIC*.

c. Data

Bagi sisi pemakai, komponen terpenting dalam DBMS adalah data karena dari data inilah pemakai dapat memperoleh informasi yang sesuai dengan kebutuhan masing-masing.

d. Prosedur

Prosedur adalah petunjuk tertulis yang berisi cara merancang hingga menggunakan basis data. Beberapa hal yang dimasukkan dalam prosedur:

1. Cara masuk ke DBMS (*login*).
2. Cara memakai fasilitas-fasilitas tertentu dalam DBMS maupun cara menggunakan aplikasi.
3. Cara mengaktifkan dan menghentikan DBMS.
4. Cara membuat cadangan basis data dan cara mengembalikan cadangan ke DBMS.

e. Orang

Komponen orang dapat dibagi menjadi tiga kelompok, yaitu :

1. Pemakai akhir (*end-user*).
2. Pemogram aplikasi.
3. Administrator basis data.

Terdapat beberapa elemen basis data, yaitu :

a. *Database*

*Database* atau basis data adalah kumpulan tabel yang mempunyai kaitan antara suatu tabel dengan tabel lainnya sehingga membentuk suatu bangunan data.

b. Tabel

Tabel adalah kumpulan *record-record* yang mempunyai panjang elemen yang sama dan atribut yang sama namun berbeda data *valuenya*.

c. Entitas

Entitas adalah sekumpulan objek yang terdefiniskan yang mempunyai karakteristik sama dan bisa dibedakan satu dengan lainnya. Objek dapat berupa barang, orang, tempat atau suatu kejadian.

d. Atribut

Atribut adalah deskripsi data yang bisa mengidentifikasi entitas yang membedakan entitas tersebut dengan entitas yang lain. Seluruh atribut harus cukup untuk menyatakan identitas objek atau dengan kata lain, kumpulan atribut dari setiap entitas dapat mengidentifikasi keunikan suatu individu.

e. *Data Value* (Nilai Data)

*Data value* adalah data aktual atau informasi yang disimpan pada tiap data, elemen atau atribut. Atribut nama pegawai menunjukkan tempat dimana informasi nama karyawan disimpan, nilai datanya misalnya adalah Anjang, Arif, Suryo dan lain-lain yang merupakan isi data nama pegawai tersebut.

f. *File*

*File* adalah kumpulan *record* sejenis yang mempunyai panjang elemen yang sama, atribut yang sama namun berbeda nilai datanya.

g. *Record/Tuple*

Kumpulan elemen-elemen yang saling berkaitan menginformasikan tentang suatu entitas secara lengkap. Satu *record* mewakili satu data atau informasi.

## 2.8 Kebutuhan Perangkat Lunak

### 2.8.1 *Android*

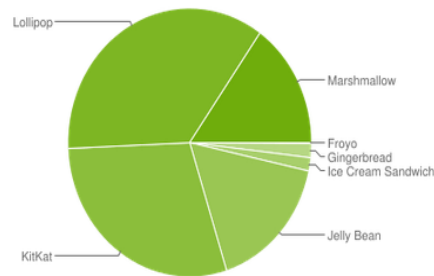
Awalnya, *Android* dikembangkan oleh perusahaan kecil di Silicon Valley yang bernama *Android Inc.* Selanjutnya, *Google* mengambil alih sistem operasi tersebut pada tahun 2005 dan mencanangkannya sebagai sistem operasi yang bersifat “*Open Source*”. Sebagai konsekuensinya, siapapun boleh memanfaatkannya dengan gratis, termasuk dalam hal kode sumber yang digunakan untuk menyusun sistem operasi tersebut.

Hal yang menarik, Android tidak hanya ditujukan untuk ponsel, tetapi juga perangkat elektronik bergerak lainnya. Pada tahun 2012, Android telah digunakan pada peranti-peranti berikut :

- a. *Smartphone*.
- b. Tablet.
- c. Peranti pembaca buku elektronik.
- d. *Netbook*.
- e. *MP4 player*.
- f. TV internet.

Android terus berkembang dan hal itu ditandai dengan versinya. Versi pertama hingga sekarang, dapat dilihat pada Gambar 2.3.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	1.7%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.6%
4.1.x	Jelly Bean	16	6.0%
4.2.x		17	8.3%
4.3		18	2.4%
4.4	KitKat	19	29.2%
5.0	Lollipop	21	14.1%
5.1		22	21.4%
6.0	Marshmallow	23	15.2%



Gambar 2.3 Versi Android (<https://developer.android.com>)

Level API menyatakan suatu bilangan unik yang digunakan untuk mengidentifikasi API (Application Programming Interface) yang digunakan pada suatu versi Android. Dengan kata lain, setiap versi Android ditandai dengan sebuah level API (Abdul Kadir, 2013).

### 2.8.2 Java

*Java* adalah bahasa pemrograman berorientasi objek yang dikembangkan oleh Sun Microsystems sejak tahun 1991. Bahasa ini dikembangkan dengan model yang mirip dengan bahasa C++ dan *smalltalk*, namun dirancang agar lebih mudah dipakai dan *platform independent*, yaitu dapat dijalankan di berbagai jenis sistem operasi dan arsitektur komputer. Bahasa ini juga dirancang untuk pemrograman di internet sehingga dirancang agar aman dan portabel.

*Platform independent* berarti program yang ditulis dalam bahasa *Java* dapat dengan mudah dipindahkan antar berbagai jenis sistem operasi dan berbagai jenis arsitektur komputer. Aspek ini sangat penting untuk dapat mencapai tujuan *Java* sebagai bahasa pemrograman internet di mana sebuah program akan dijalankan oleh berbagai jenis komputer dengan berbagai jenis sistem operasi. Sifat ini berlaku untuk level *source code* dan *binary code* dari program *Java*. Berbeda dengan bahasa C dan C++, semua tipe data dalam bahasa *Java* mempunyai ukuran yang konsisten disemua jenis *platform*. *Source code* program *Java* sendiri tidak perlu dirubah sama sekali jika Anda ingin mengkompile ulang di *platform* lain. Hasil dari mengkompile *source code Java* bukanlah kode mesin atau instruksi prosesor yang spesifik terhadap mesin tertentu, melainkan berupa *bytecode* yang berupa *file* berekstensi *class*. *Bytecode* tersebut dapat langsung dieksekusi di tiap *platform* yang dengan menggunakan *Java Virtual Machine (JVM)* sebagai interpreter terhadap *bytecode* tersebut (Joyce, 2007).

### 2.8.3 Android Studio

Menurut Eric (2016), dalam jurnalnya tertulis bahwa *Android Studio* merupakan sebuah *Integrated Development Environment (IDE)* untuk *platform Android*. *Android Studio* ini diumumkan pada tanggal 16 Mei 2013 pada Konferensi Google I/O oleh Produk Manajer Google, Ellie Powers. *Android Studio* bersifat *free* dibawah *Apache License 2.0*. *Android studio* awalnya dimulai dengan versi 0.1 pada bulan mei 2013, Kemudian dibuat versi *beta* 0.8 yang dirilis pada bulan juni 2014. Yang paling stabil dirilis

pada bulan Desember 2014, dimulai dari versi 1.0. Berbasiskan *JetBrains' IntelliJ IDEA*, Studio didesain khusus untuk Android Development yang kini sudah bisa di *download* untuk *Windows*, *Mac OS X*, dan *Linux*.

#### 2.8.4 MySQL

Menurut Achmad Solichin (2016), MySQL adalah sebuah perangkat lunak sistem manajemen basis data SQL (*Database Management System*) atau DBMS yang *multithread*, *multi-user*, dengan sekitar 6 juta instalasi di seluruh dunia. MySQL AB membuat MySQL tersedia sebagai perangkat lunak gratis di bawah lisensi GNU *General Public License* (GPL), tetapi mereka juga menjual dibawah lisensi komersial untuk kasus-kasus dimana penggunaannya tidak cocok dengan penggunaan GPL. Tidak seperti *Apache* yang merupakan *software* yang dikembangkan oleh komunitas umum, dan hak cipta untuk kode sumber dimiliki oleh penulisnya masing-masing, MySQL dimiliki dan disponsori oleh sebuah perusahaan komersial Swedia yaitu MySQL AB. MySQL AB memegang penuh hak cipta hampir atas semua kode sumbernya. Kedua orang Swedia dan satu orang Finlandia yang mendirikan MySQL AB adalah: David Axmark, Allan Larsson, dan Michael "*Monty*" Widenius. Beberapa kelebihan MySQL antara lain : *free* (bebas di *download*), stabil dan tangguh, fleksibel dengan berbagai pemrograman, *security* yang baik, dukungan dari banyak komunitas, kemudahan management *database*, mendukung transaksi dan perkembangan *software* yang cukup cepat.