BAB II

TINJAUAN PUSTAKA

2.1 E-Service

E-Service merupakan singkatan dari *Electronic Service* dalam bahasa inggris yang dalam Bahasa Indonesia artinya Layanan Elektronik atau E-Layanan. *E-Service* merupakan suatu aplikasi layanan yang memanfaatkan Teknologi dan Komunikasi dalam penerapannya. Layanan elektronik biasanya mengacu pada penyediaan layanan melalui *internet*, sehingga layanan elektronik bisa juga termasuk perdagangan *internet*, mungkin juga termasuk layanan non-komersial/*online*.

2.2 Service Kendaraan Bermotor

Service sering disebut dengan istilah perbaikan (jasa), Pengertian dari perbaikan itu sendiri adalah usaha untuk mengembalikan kondisi dan fungsi dari suatu benda atau alat yang rusak akibat pemakaian alat tersebut pada kondisi semula. Proses perbaikan tidak menuntut penyamaan sesuai kondisi awal, yang diutamakan adalah alat tersebut bisa berfungsi *normal* kembali.

Perbaikan memungkinkan untuk terjadinya pergantian bagian alat/*sparepart*. Terkadang dari beberapa produk yang ada dipasaran tidak menyediakan *sparepart* untuk penggantian saat dilakukan perbaikan, meskipun ada, harga *sparepart* tersebut hampir mendekati harga baru satu unit produk tersebut. Hal ini yang memaksa *user*/pelanggan untuk membeli produk yang baru. Tidak setiap perbaikan dapat diselesaikan dengan mudah, tergantung tingkat kesulitan dan kerumitan *assembling* / perakitan alat tersebut. Tingkat kesulitan tersebutlah yang menumbuhkan perbedaan jenis perbaikan, mulai jenis perbaikan ringan, perbaikan sedang, dan perbaikan yang sering dinamakan *service* berat. Dari jenis *service* di atas ditentukan biaya perbaikan sesuai dengan tingkat kesulitan.

2.3 E-Service Kendaraan Bermotor

Seiring dengan perkembangan dunia teknologi saat ini penggunaan alat elektronik dalam menunjang kebutuhan sehari-hari sudah menjadi tren. Misalnya *online shop*, resevasi kamar hotel *online*, pemesanan jasa ojek *online* dan masih banyak yang lainnya. Begitu juga pada layanan *service* kendaraan bermotor saat ini sudah banyak yang menggunakan sistem *booking service* secara *online*.

E-Service kendaraan bermotor merupakan layanan jasa perbaikan/*service* kendaraan bermotor menggunakan aplikasi layanan berbasis elektronik atau *online*. Dengan menggunakan *E-Service* kendaraan bermotor ini pengguna kendaraan dapat melakukan pemesanan layanan jasa *service* kendaraan tanpa perlu langsung membawa kendaraannya ke bengkel.

2.4 Sistem Informasi

Pengertian sistem informasi dibagi menjadi 3 yaitu pengertian sistem, pengertian data dan informasi untuk dapat memahami pengertian sistem informasi yang sebenarnya.

2.4.1 Sistem

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

Ada 3 elemen yang membentuk sebuah sistem yaitu:

- *Input*

  Segala sesuatu yang masuk ke dalam sistem dan selanjutnya menjadi bahan untuk di proses.

- *Proses*

  Bagian yang melakukan perubahan dari *input* menjadi *output* yang berguna, misalnya berupda informasi dan produk, tetapi juga bisa berupa hal-hal yang tidak berguna, misalnya sisa pembuangan atau limbah.

- *Output*

  Hasil dari pemrosesan, misalnya berupa suatu informasi, saran, cetakan laporan, dan lain-lain.

Berdasarkan pengertian diatas dapat disimpulkan sistem adalah cara yang kita lakukan untuk mencapai tujuan yang telah kita buat mulai dari menginput sesuatu memprosesnya kemudian menghasilkan *output*.

2.4.2 Data dan Informasi

Data adalah fakta atau apa pun yang dapat digunakan sebagai input dalam menghasilkan informasi. Informasi adalah hasil dari pengolahan data yang memiliki makna atau arti. (Deni Darmawan dan Kunkun Nur Fauzi, 2013)

Berdasarkan pengertian diatas, dapat disimpulkan bahwa data adalah fakta yang kita peroleh yang belum diolah dan disusun sedangkan informasi adalah kumpulan data yang telah disusun dan di proses yang hasilnya memiliki arti dan makna.

2.4.3 Pengertian Sistem Informasi

Sistem informasi adalah kumpulan dari subsistem yang saling berhubungan satu sama lain dan bekerja sama untuk mengelolah data menjadi informasi yang berguna. (Deni Darmawan dan Kunkun Nur Fauzi 2013)

Sistem informasi mencangkup sejumlah komponen (manusia, komputer, teknologi informasi dan prosedur kerja), ada sesuatu yang diproses (data menjadi informasi) dan dimaksudkan untuk mencapai suatu sasaran atau tujuan (Abdul Kadir, 2014).

2.5 Android

Android adalah istilah dalam bahasa inggris yang berarti "*Robot* yang menyerupai manusia". Logo *android* sendiri dicerminkan seperti sebuah *robot* berwarna hijau, yang mengacu kepada arti kata *Android*. *Android* adalah sebuah sistem operasi untuk *smartphone* dan *tablet*. Sistem operasi dapat diilustrasikan sebagai "jembatan" antara piranti (*device*) dan penggunaanya, sehingga pengguna bisa berinteraksi dengan *device* dan menjalankan aplikasi – aplikasi yang tersedia pada *device*. Di dunia personal komputer, sistem operasi yang banyak dipakai adalah *Windows, Mac,* dan *Linux*. (Nazruddin, Safaat H. 2012)

Android dikembangkan bersama oleh perusahaan-perusahaan yang tergabung dalam sebuah konsorsium bernama *Open Handset Alliance* (OHA). *OHA* dipimpin oleh *Google* dan didirikan bersama dengan 34 perusahaan lainnya, dengan tujuan untuk mengembangkan teknologi *mobile device*, semi konduktor, pembuatan aplikasi, komersialisasi, dan *mobile operator*.

Android adalah sistem operasi yang bersifat *open source* (sumber terbuka). Disebut *open source* karena *source code* (kode sumber) dari sistem operasi *android* dapat dilihat didownload, dan dimodifikasi secara bebas. Paradigma *open source* ini memudahkan pengembangan teknologi *android*, karena semua pihak yang tertarik dapat memberikan kontribusi, baik pada pengembangan sistem operasi maupun aplikasi (Nazruddin, Safaat H. 2012).

2.6 Peta

Peta merupakan gambaran wilayah geografis, bagian permukaan bumi yang disajikan dalam berbagai cara yang berbeda, mulai dari peta konvensional yang tercetak hingga peta *digital* yang tampil di layar komputer. Peta dapat digambarkan dengan berbagai gaya, masing-masing menunjukkan permukaan yang berbeda untuk subjek yang sama untuk menvisualisasikan dunia dengan mudah, informatif dan fungsional.

Peta berbasis komputer (*digital*) lebih serba guna dan dinamis karena bisa menunjukkan banyak *view* yang berbeda dengan subjek yang sama. Peta ini juga

memungkinkan perubahan skala, animasi gabungan, gambar, suara, dan bisa terhubung ke sumber informasi tambahan melalui *internet*. Peta digital dapat di *update* ke peta tematik baru dan bisa menambahkan detail informasi geografi lainnya (Denny Charter, Irma Agtrisari 2003).

2.7 Metode Pengembangan Sistem

Metode pengembangan sistem sangat dibutuhkan dalam perancangan sebuah sistem karena sebelum memulai pembuatan koding - koding hendaknya merancang terlebih dahulu metode pemodelan seperti apa yang harus digunakan dengan memprioritaskan ketepatan waktu selesai dan efektifitas dalam perancangan sebuah sistem.

2.7.1 UML (Unifield Modeling Language)

Perancangan dan pembangunan aplikasi perangkat lunak berbasis *Object Oriented Analysis* dan *Design* (OOAD) sedang marak digunakan saat ini. Dengan menganggap segala sesuatunya adalah objek serta sistem dipandang sebagai interaksi dari banyak objek menjadi ide utama pendekatan ini. Perancangan berbasis objek dimodelkan menggunakan *Unifield Modeling Language* (UML). *UML* merupakan kumpulan *Diagram-Diagram* yang sudah memiliki *standart* untuk pembangunan perangkat lunak berbasis objek. *UML* memiliki banyak *Diagram, Use Case, Class, Sequence,* dan *Activity.*

UML sendiri sebenarnya memiliki banyak sekali *Diagram* selain empat yang disebutkan diatas, akan tetapi sebagian besar memang jarang dguanakan oleh pengembang aplikasi. Keempat *Diagram* diatas merupakan *Diagram* inti dari *UML*. Bagaimanapun aplikasi perangkat lunak yang dibuat, bila dibangun dengan beriorentasi objek, keempat *Diagram* tersebut harus ada. Pengunaan lainnya dalam *UML* ditunjukan untuk mendapatkan gambaran arsitektur sistem informasi dengan sudut pandang yang berbeda, hal ini tepaut pada faktor kebutuhan.

Penting untuk dipahami, sewaktu spesialis informasi memiliki membuat permodelan arsitektur dengan pendekatan terstruktur (misalnya: membuat

Diagram Konteks dan *Diagram* Alir data), maka permodelan arsitektur berbasiskan objek (misalnya membuat *Diagram-Diagram UML* semisal *Use Case*, *Class*) tidak lagi digunakan, demikian pula sebaiknya (Feri Sulianta, 2017).

2.7.1.1 *Use Case Diagram*

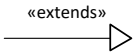
Use Case merupakan *Diagram* yang harus dibuat pertama kali saat pemodelan perangkat lunak berorientasi objek dilakukan. *Use Case Diagram* akan menggambarkan apa yang dikerjakan oleh aktor. Yang disebut aktor disini adalah pengguna aplikasi, sama seperti pembangunan perangkat lunak terstruktur saat membuat *DFD*, untuk menggambarkan *Use Case Diagram* mengacu pada proses sebelumnya, yaitu analisis kebutuhan pada *RPL*. (Feri Sulianta, 2017).

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, jadi, 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.

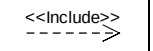
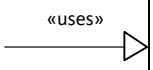
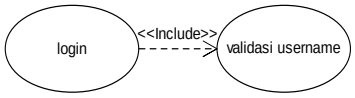
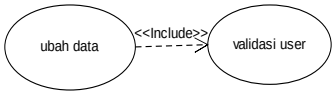
Tabel 2.1 Simbol *Use Case Diagram*

Keterangan	**Simbol**	**Deskripsi**
Use Case		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 *use case*

Tabel 2.1 Simbol *Use Case Diagram* (Lanjutan)

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.
Asosiasi		Komunikasi antara aktor dan *use case* yang berpartisipasi pada *use case* atau *use case* memiliki interaksi dengan aktor.
Ekstensi		Relasi use case tambahan ke sebuah *use case*, dimana *use case* yang ditambahkan dapat berdiri sendiri walau tanpa *use case* tambahan itu; mirip dengan prinsip *inheritance* pada pemograman berorientasi objek; biasanya *use case* tambahan memiliki nama depan yang sama dengan use case yang ditambahkan, misal  Arah panah mengarah pada *use case* yang ditambahkan.

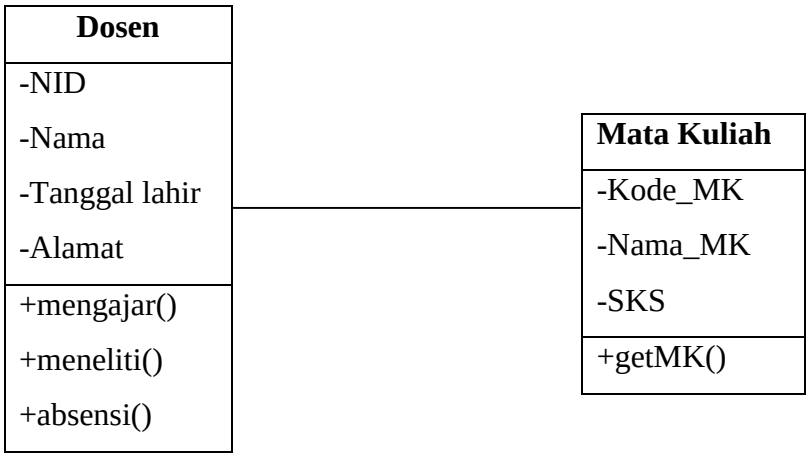
Tabel 2.1 Simbol *Use Case Diagram* (Lanjutan)

Generalisasi		Hubungan generalisasi dan spesialisasi (umum-khusus) antara dua buah *use case* dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya  Arah panah mengarah pada *use case* yang menjadi generalisasinya (umum).
Menggunakan /*include/uses*	 	Ada dua sudut pandang yang cukup besar mengenai *include* di *use case* : a. *Include* berarti *use case* yang ditambahkan akan selalu dipanggil saat *use case* tambahan dijalankan, misal pada kasus berikut :  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 :  Ke dua interpretasi di atas dapat dianut salah satu atau keduanya tergantung pada pertimbangan dan interpretasi,

2.7.1.2 *Class Diagram*

Diagram kelas dibuat setelah *Diagram Use Case* dibuat terlebih dahulu. Pada pembuatan *Diagram* ini harus menjelaskan hubungan apa saja yang terjadi antara suatu objek dengan objek lainnya sehingga terbentuklah suatu sistem aplikasi.

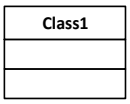
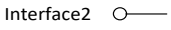
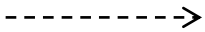
Pembuatan *Diagram* Kelas dibagi menjadi dua bagian, yaitu kelas itu sendiri dan relasi antar kelas. Kelas dibagi menjadi tiga bagian, yakni nama kelas, atribut kelas, serta operasi kelas (*methods*). Nama kelas adalah nama dari kelas itu sendiri, misalnya kelas mobil, kelas dosen, kelas mahasiswa, dan lain-lain. Penamaan kelas menggunakan kata benda. Atribut adalah data yang dimiliki oleh kelas tersebut. Misalnya kelas mahasiswa memiliki atribut NPM, nama, tanggal lahir, alamat, jemis kelamin dan lain sebagainya. Lalu perasi kelas adalah menunjukan apa yang kelas tersebut bisa lakukan, misalnya kelas dosen dapat melakukan operasi mengajar, absensi, peneltian, dan lain-lain (Feri Sulianta, 2017).



Gambar 2.1 Class *Diagram*

Relasi memilki *multiplicity*, misalnya satu dosen dapat mengajar banyak mata kuliah, dan satu mata kuliah dapat diajar oleh banyak dosen, berarti *multiplicity*-nya adalah banyak ke banyak.

Tabel 2.2 Simbol *Class Diagram*

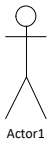
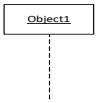
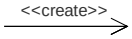
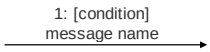
Keterangan	Simbol	Deskripsi
Kelas		Kelas pada struktur sistem.
Antarmuka / Interface		Sama dengan konsep *interface* dalam pemograman berorientasi objek.
Asosiasi		Relasi antar kelas dalam makna umum, asosiasi biasanya juga disertai dengan *multiplicity*.
Asosiasi berarah		Relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan *multiplicity*.
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 (*whole-part*).

2.7.1.3 *Sequence Diagram*

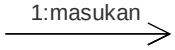
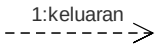
Sequence Diagram adalah *Diagram* yang dibuat untuk mengetahui alur dari interaksi antar objek. Isi dari *Sequence Diagram* harus sama dengan *use case* dan

Diagram Kelas. Satu *Use Case* tunggal akan digambarkan satu *Sequence Diagram* (Feri Sulianta, 2017).

Tabel 2.3 Simbol *Sequence Diagram*

Keterangan	**Simbol**	**Deskripsi**
Aktor		Orang, proses atau sistem lain yang bernteraksi 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.
Garis hidup		Menyatakan kehidupan suatu objek.
Objek		Menyatakan objek yang berinteraksi pesan.
Waktu aktif		Menyatakan objek dalam keadaan aktif dan berinteraksi pesan.
Pesan tipe create		Menyatakan suatu objek membuat objek yang lain, arah panah mengarah pada objek yang dibuat.
Pesan tipe call		Menyatakan suatu objek memanggil operasi/metode yang ada pada objek lain atau dirinya sendiri. Arah panah mengarah pada objek yang memiliki operasi atau metode karena ini

Tabel 2.3 Simbol *Sequence Diagram* (Lanjutan)

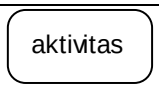
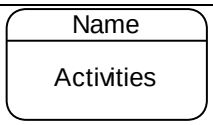
		memanggil operasi/metode maka operasi/metode yang dipanggil harus ada pada *Diagram* kelas sesuai dengan kelas objek yang berinteraksi.
Pesan tipe send		Menyatakan bahwa suatu objek mengirimkan data / masukan / informasi ke objek lainnya, arah panah mengarah pada objek yang dikirimi.
Pesan tipe return		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.

2.7.1.4 *Activity Diagram*

Diagram aktivitas atau *activity* menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Hal yang perlu diperhatikan disini adalah bahwa *Diagram* aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapta 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 2.4 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.
Swimlane		Memisahkan organisasi bisnis yang bertanggung jawab terhadap aktivitas yang terjadi.
Status akhir		Status akhir yang dilakukan sistem, sebuah *Diagram* aktivitas memiliki sebuah status akhir.

2.7.2 OOAD (*Object Oriented Analysis and Design*)

OOAD adalah metode pengembangan sistem yang lebih menekankan pada objek dibandingkan dengan data atau proses. Dalam tahapannya OOAD terbagi menjadi dua, yaitu OOA (*Object Oriented Analysis*) dan OOD (*Object Oriented Design*). (Feri Sulianta, 2017).

2.7.2.1 OOA (*Object Oriented Analysis*)

Object oriented analysis (OOA) merupakan metode analisis yang memeriksa *requirements* (syarat / keperluan yang harus dipenuhi oleh sistem) dari sudut

pandang kelas-kelas dan objek-objek yang di temui dalam ruang lingkup permasalahan. OOA mempelajari permasalahan dengan menspesifikasikannya atau mengobservasi permasalahan tersebut dengan menggunakan metode berorientasi objek. Biasanya analisis sistem dimulai dengan adanya dokumen permintaan yang diperoleh dari semua pihak yang berkepentingan. (Feri Sulianta, 2017). Adapun langkah dalam OOA adalah sebagai berikut :

1) Menganalisa masalah

   Mengumpulkan data-data yang diperlukan sebagai kebutuhan sistem. Setelah semua data yang diperlukan selanjutnya akan dilakukan analisa untuk merumuskan masalah yang terjadi.

2) Menjelaskan proses yang terjadi dalam sistem

   Semua data-data yang diperlukan untuk membangun sistem digambarkan dalam bentuk rancangan analisis berupa *Usecase Diagram, Class Diagram,* dan *Sequence Diagram*.

3) Identifikasi objek

   Objek merupakan benda yang secara fisik dan konseptual yang ada disekitar kita.

4) Menentukan atribut

   Atribut disebut juga dengan class yaitu definisi umum dari himpunan objek yang sejenis. Kelas menetapkan spesifikasi perilaku (*behaviour*) dan atribut-atribut dari objek tersebut. *Class* adalah abstraksi dari entitas dunia nyata.

5) Mendefinisakan operasi

   Menjelaskan operasi yang memungkinkan bisa untuk di implementasikan dan yang tidak bisa di implementasikan

2.7.2.2 OOD (Object Oriented Design)

Object oriented design (OOD) merupakan metode yang mengarah pada arsitektur *software* yang di dasarkan pada manipulasi objek-objek sistem atau *subsistem*. OOD memungkinkan *software engineer* untuk mengetahui objek-objek

yang dihasilkan oleh setiap kelas dan hubungan antar objek. Selain itu OOD menggambarkan bagaimana hubungan antar objek bisa dilakukan. Permodelan berorientasi objek biasanya dituangkan dalam dokumentasi perangkat lunak dengan menggunakan perangkat permodelan berorientasi objek, diantaranya adalah UML (*Unified Modeling Language*). (Feri Sulianta, 2017). Adapun tahapan dari *Object Oriented Design* (OOD), yaitu:

1) Desain *subsistem*

   Berisikan representasi masing-masing *subsistem* yang memungkinkan perangkat lunak mencapai persyaratan yang didefinisikan oleh pelanggannya dan untuk mengimplementasikan infrastruktur yang mendukung persyaratan pelanggan. Desain *subsistem* ini menggambarkan tabel-tabel yang digunakan dalam sistem.

2) Desain objek dan kelas

   Berisi hirarki kelas yang memungkinkan sistem diciptakan dengan menggunakan generalisasi dan spesialisasi yang ditarget secara perlahan. Lapisan ini juga berisi infrastruktur yang mendukung persyaratan pelanggan.

3) Desain pesan

   Berisi detail yang memungkinkan masing-masing objek berkomunikasi dengan kolaboratornya. Lapisan ini membangun interface internal dan eksternal bagi sistem tersebut.

2.8 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 oleh *DBMS*, program dapat ditulis sehingga tidak tergantung pada struktur data dalam basis data. Dengan pendekatan 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 ditunjukan untuk menjaga agar data selalu dalam keadaan yang *falied* 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 prokdutifitas

   Untuk menyediakan kemudahan bagi pemakan 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 seperangkat komputer dan periperal pendukungnya.
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 pemrogramman 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 yang perlu 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 mengmbalikan 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 ikatan antara satu *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 terdefinisikan 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.

2.9 MySQL

MySQL adalah sebuah *Database Manajemen System* (*DBMS*) popular yang memiliki fungsi sebagai *Relational Database Manajemen System* (*RDBMS*). Selain itu *MySQL software* merupakan suatu aplikasi yang sifatnya *open source* serta *server* basis data *MySQL* memiliki kinerja sangat cepat, *reliable*, dan mudah untuk digunakan serta bekerja dengan arsitektur *client server* atau *embedded system*. Dikarenakan faktor *open source* dan popular tersebut maka cocok untuk mendemontrasikan proses replikasi basis data. (Nugroho, dan Bunafit 2013)

2.10 Pengujian *Software*

Pengujian *software* sangat diperlukan untuk memastikan *software*/aplikasi yang sudah/sedang dibuat dapat berjalan sesuai dengan fungsionalitas yang diharapkan. Pengembang atau penguji *software* harus menyiapkan sesi khusus untuk menguji program yang sudah dibuat agar kesalahan ataupun kekurangan dapat dideteksi sejak awal dan dikoreksi secepatnya. Pengujian atau *testing* sendiri merupakan elemen kritis dari jaminan kualitas perangkat lunak dan merupakan bagian yang tidak terpisah dari siklus hidup pengembangan *software* seperti halnya analisis, desain, dan pengkodean. (Shi, 2010)

Ada beberapa jenis pengujian perangkat lunak, antara lain (Khan, 2011):

1. Pengujian *white box* adalah pengujian yang didasarkan pada pengecekan terhadap *detail* perancangan, menggunakan struktur kontrol dari desain program secara prosedural untuk membagi pengujian ke dalam beberapa kasus pengujian. Secara sekilas dapat diambil kesimpulan *white box testing* merupakan petunjuk untuk mendapatkan program yang benar secara 100%.
2. *Black-Box Testing* merupakan pengujian yang berfokus pada spesifikasi fungsional dari perangkat lunak, tester dapat mendefinisikan kumpulan kondisi input dan melakukan pengetesan pada spesifikasi fungsional program.

Teknik pengujian ini berfokus pada domain informasi dari perangkat lunak, yaitu melakukan kasus uji dengan mempartisi domain *input* dan *output* program. Metode *black-box* memungkinkan rekayasa perangkat lunak mendapatkan serangkaian kondisi *input* yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program. Pengujian ini berusaha menemukan kesalahan dalam kategori fungsi - fungsi yang tidak benar atau hilang, kesalahan *interface*, kesalahan dalam struktur data atau akses basis data *external*, kesalahan kinerja, inisialisasi dan kesalahan terminal (Pressman, 2010).

2.10.1 Black-Box Testing

Black Box Testing berfokus pada spesifikasi fungsional dari perangkat lunak. Tester dapat mendefinisikan kumpulan kondisi *input* dan melakukan pengetesan pada spesifikasi fungsional program. *Black Box Testing* bukanlah solusi alternatif dari *White Box Testing* tapi lebih merupakan pelengkap untuk menguji hal-hal yang tidak dicakup oleh *White Box Testing*.

Black Box Testing cenderung untuk menemukan hal-hal berikut :

1. Fungsi yang tidak benar atau tidak ada
2. Kesalahan antarmuka (*interface errors*)
3. Kesalahan pada struktur data dan akses basis data

4. Kesalahan performansi (*performance errors*)
5. Kesalahan inisialisasi dan terminasi

Pengujian didesain untuk menjawab pertanyaanpertanyaan berikut:

1. Bagaimana fungsi-fungsi diuji agar dapat dinyatakan *valid*?
2. *Input* seperti apa yang dapat menjadi bahan kasus uji yang baik?
3. Apakah sistem sensitif pada *input-input* tertentu?
4. Bagaimana sekumpulan data dapat diisolasi?
5. Berapa banyak rata-rata data dan jumlah data yang dapat ditangani sistem?
6. Efek apa yang dapat membuat kombinasi data ditangani spesifik pada operasi sistem?

Saat ini terdapat banyak metode atau teknik untuk melaksanakan *Black Box Testing*, antara lain:

1. *Equivalence Partitioning*
2. *Boundary Value Analysis/Limit Testing*
3. *Comparison Testing*
4. *Sample Testing*
5. *Robustness Testing*
6. *Behavior Testing*
7. *Requirement Testing*
8. *Performance Testing*
9. Uji Ketahanan (*Endurance Testing*)
10. Uji Sebab-Akibat (*Cause-Effect Relationship Testing*