BAB 2

LANDASAN TEORI

2.1 Teori Umum

2.1.1 Pengertian *Learning Management System*

Learning Management System adalah suatu perangkat lunak atau *software* untuk keperluan administrasi, dokumentasi, laporan sebuah kegiatan, kegiatan belajar mengajar dan kegiatan secara *online* (terhubung ke internet), *E-learning* dan materi-materi pelatihan. Dan semua itu dilakukan dengan *online*. Namun, tidak sesederhana itu. Sebuah *learning management system* yang kuat harus dapat melakukan hal-hal berikut (Ellis, 2009, p1):

- Memusatkan dan mengotomatisasi administrasi
- Menggunakan *self service* dan *self guided services*
- Membangun dan menyampaikan konten pembelajaran secara cepat
- Konsolidasi pelatihan inisiatif pada sebuah *scaleable web-based platform*
- Mendukung portabilitas dan standar
- Mendukung personalisasi konten dan memungkinkan penggunaan kembali

Learning Management System adalah sebuah sistem yang memungkinkan sebuah institusi untuk mengembangkan materi pembelajaran elektronik untuk siswanya. Semua *Learning Management System* mengatur *login* untuk pengguna yang teregistrasi, mengatur katalog pembelajaran,

menyimpan data siswa, dan menyediakan laporan ke manajemen (Paulsen, 2003, p134).

Learning Management System merupakan alat atau sistem yang digunakan untuk autentikasi, registrasi, dan akses untuk pembelajaran. Sebagian besar berisi katalog atau daftar materi yang tersedia dan metode bagi pembelajar untuk mendapatkan materi tersebut. Sistem harus dapat menelusuri keterlibatan peserta untuk setiap materi dan materi apa yang sudah diambil oleh pembelajar. Termasuk fitur-fitur untuk memungkinkan materi ditambah atau dihapus dari katalog. Beberapa sistem memungkinkan kustomisasi *learning path* atau *road map* bagi pembelajar berdasarkan fungsi pekerjaan mereka (Barritt et al, 2004, p233).

2.1.2 Pengertian Data

Data adalah fakta-fakta atau observasi yang mentah, biasanya mengenai kejadian atau transaksi bisnis (O'Brien, 2003, p13).

Data adalah kompenen terpenting dalam *Database Management System* (DBMS) yang berasal dari sudut pandang pengguna akhir. Data bertindak sebagai penghubung antara mesin dan pengguna (Connolly dan Begg, 2005, p20).

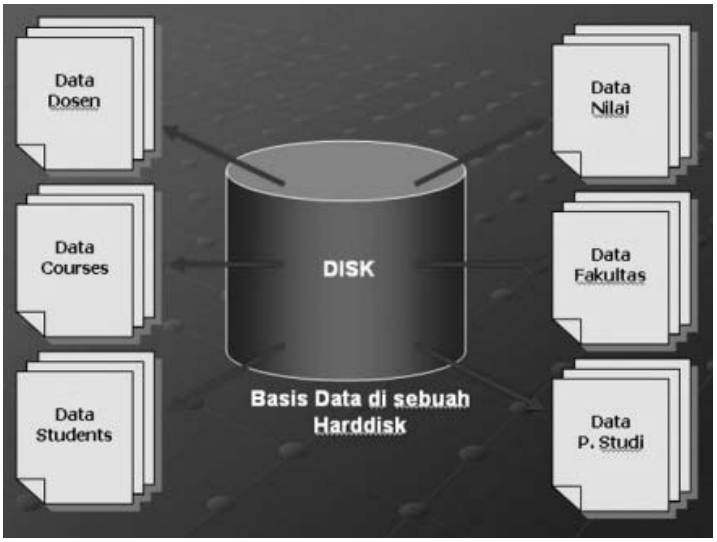
Data adalah fakta-fakta yang diberikan kepada peneliti dari lingkungan studinya (Cooper, 2003, p72).

Ciri-ciri data yaitu (Pearlson dan Saunders, 2004, p276):

1. *Easily captured* (mudah didapat)
2. *Easily structured* (mudah distruktur)
3. *Easily transferred* (mudah ditransfer)
4. *Compact, quantifiable* (ringkas, dapat diukur)

2.1.3 Pengertian *Database*

Database adalah kumpulan data logis yang saling terkait, dan sebuah deskripsi data, didesain untuk memenuhi kebutuhan dari sebuah organisasi. *Database* adalah sebuah data tunggal, besar, yang dapat digunakan secara bersamaan oleh banyak departemen dan pengguna (Begg dan Connolly, 2005, p15).



Gambar 2.1 Contoh Basis Data

Database adalah:

- Himpunan kelompok data (arsip) yang saling berhubungan yang diorganisasi sedemikian rupa agar kelak dapat dimanfaatkan kembali dengan cepat dan mudah.
- Kumpulan data yang saling berhubungan yang disimpan secara bersama sedemikian rupa dan tanpa pengulangan (redudansi) yang tidak perlu, untuk memenuhi berbagai kebutuhan.
- Kumpulan *file*/tabel/arsip yang saling berhubungan yang disimpan dalam media penyimpanan elektronis (Fathansyah, 1999, p2).

Data dalam basis data disimpan dalam tiga struktur, yaitu *file*, tabel atau objek. *File* terdiri dari *fields* dan *records*, tabel terdiri dari baris dan kolom. *Fields* adalah unit terkecil dari data yang disimpan dalam basis data (Whitten et. Al, 2003, p550). Sedangkan *records* adalah sebuah kumpulan *fields* yang diatur dalam format yang telah ditentukan sebelumnya (Whitten et al, 2003, p551).

Terdapat 3 macam field pada basis data (Whitten et al, 2003, p550-551), yaitu:

1. *Primary Key*

   Primary key adalah sebuah *field* yang mengidentifikasikan tanda unik dalam setiap *record.*

2. *Secondary Key*

   Secondary key adalah sebuah *field* yang mengidentifikasikan sebuah *record* tunggal.

3. *Foreign Key*

Foreign key adalah sebuah *field* pada *relational table* yang menunjuk ke sekumpulan *record* pada *table* lain di *database*.

2.1.4 Pengertian *Internet*

Internet adalah koleksi jaringan komputer yang ada di seluruh dunia yang menghubungkan jutaan komputer yang digunakan untuk bisnis, pemerintahan, lembaga pendidikan, organisasi, dan individu menggunakan modem, kabel telepon, kabel televisi, satelit, dan peralatan komunikasi lainnya (Shelly, Woods, dan Dorin, 2008, p2).

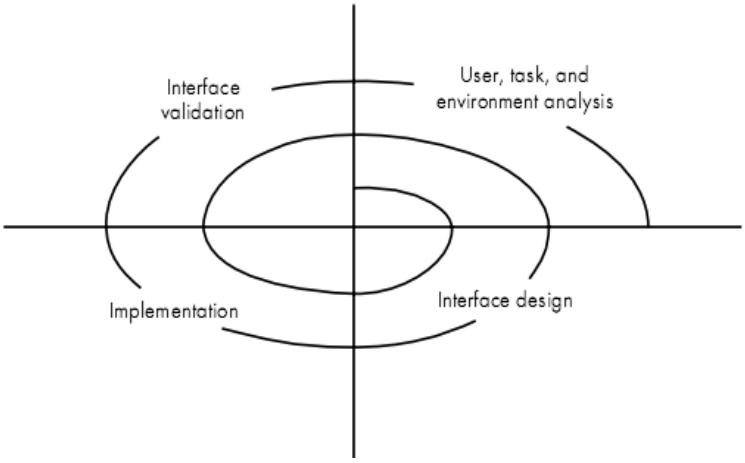
2.1.5 Teori Interaksi Manusia dan Komputer

2.1.5.1 Pengertian Interaksi Manusia dan Komputer

Interaksi manusia dan komputer merupakan sistem yang mampu menghubungkan antara *user* dengan komputer. Elemen-elemen yang terdapat di dalam *user interface* antara lain seperti *menu, window, keyboard, mouse* dan suara-suara komputer (Dastbaz**,** 2003, p108).

2.1.5.2 Pengertian *User Interface*

Desain *user interface* adalah menciptakan sebuah media komunikasi yang efektif di antara manusia dan komputer (Pressman, 2010, p312).



Gambar 2.2 Proses Desain Antarmuka

Sumber: Roger Pressman (2010, p319)

Berikut langkah-langkah dalam merancang antarmuka, yaitu sebagai berikut (Pressman, 2010, p312):

1. *User, task, and environmental analysis and modelling*

   Setelah *user tasks* teridentifikasi, skenario *user* dibuat dan dianalisa untuk mendefinisikan satu set objek dan aksi antarmuka.

2. *Interface design*

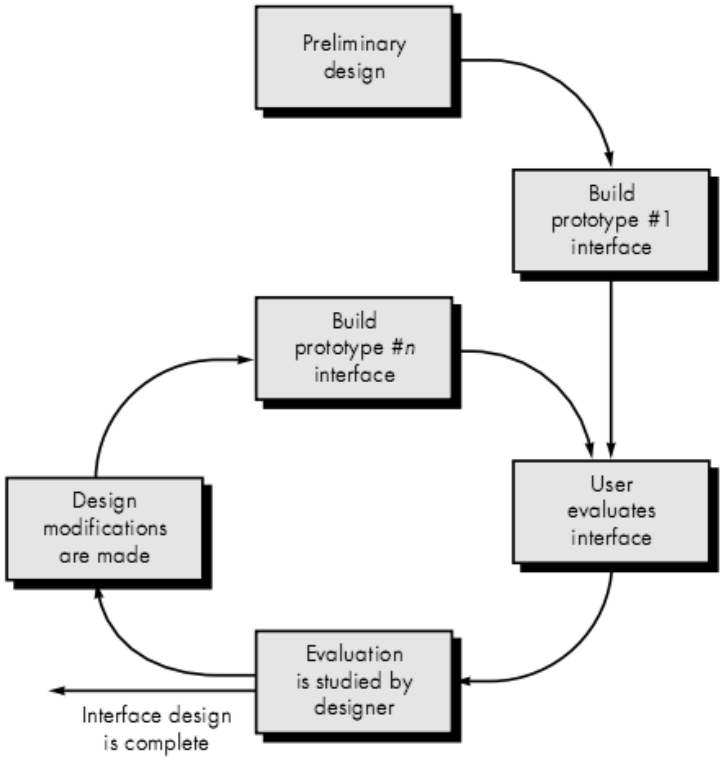
   Melakukan desain grafis antarmuka, meliputi desain tata letak *icons*, *menus*, *texts*, *images* dan lain-lain.

3. *Interface construction*

   Melakukan konstruksi dengan meletakkan *icons*, mendeskripsikan *screen text*, menspesifikasikan *menu-menu* menjadi satu kesatuan.

4. *Interface validation*

   Setelah sebuah *prototype* antarmuka pengguna diciptakan, kesemuanya harus dievaluasi untuk menentukan apakah kesemuanya itu memenuhi kebutuhan pengguna.



Gambar 2.3 Siklus Evaluasi Desain Antarmuka

Sumber: Roger Pressman (2010, p343)

Selain itu, pengguna dapat dikategorikan sebagai berikut (Pressman, 2010, p318):

1. *Novice User* adalah pengguna awam yang tidak memiliki pengetahuan yang cukup terhadap sistem dan hanya sebatas penggunaan komputer secara umum.

2. *Knowledgeable Intermittent Users* adalah pengguna yang memiliki pengetahuan *semantic* tentang aplikasi dan sistem. Namun, kurang mengeksplorasi lebih dalam fitur-fitur yang ada.
3. *Knowledgeable Frequent Users* adalah pengguna yang memiliki kemampuan se*mantic* dan *syntactic* serta menguasai sistem dan mampu mengembangkan cara-cara kreatif dalam penggunaan aplikasi.

Adapun delapan aturan emas perancangan *user interface,* yaitu sebagai berikut (Shneiderman, 2005, p74):

1. Berusaha untuk konsisten.

   Rangkaian aksi yang konsisten harus digunakan dalam keadaan seperti pada *prompts, menus,* dan layar *help* serta perintah yang konsisten.
2. Menyediakan *usability universal.*

   Usability universal mengacu pada desain informasi dan komunikasi produk dan layanan yang dapat digunakan oleh semua kalangan pengguna.
3. Memberikan umpan balik yang informatif.

   Untuk setiap aksi yang dilakukan, hendaknya selalu tersedia fasilitas umpan balik agar pengguna mengerti apa yang telah dilakukannya.
4. Merancang dialog yang memberikan penutupan.

   Urutan aksi hendaknya dibagi ke dalam kelompok dengan awal, tengah, dan akhir. Ketika telah mencapai bagian akhir, hendaknya pengguna

diberitahu melalui umpan balik. Tanpa adanya dialog untuk mencapai keadaan akhir maka pengguna akan menjadi bingung.

5. Menawarkan penanganan kesalahan sederhana.

   Sebisa mungkin, desainlah sistem sehingga pengguna tidak dapat melakukan kesalahan yang serius. Jika kesalahan dibuat, sistem harus mampu mendeteksi kesalahan dan membantu memberikan solusi untuk penanganan kesalahan.

6. Memungkinkan pembalikan aksi yang mudah.

   Fitur ini mengurangi kecemasan, karena pengguna tahu bahwa kesalahan dapat dibatalkan sehingga akan mendorong eksplorasi fungsi-fungsi lainnya.

7. Mendukung pusat kendali internal.

   Dengan pengaturan yang menyeluruh, pengguna dapat menggunakan sistem sesuai kebutuhan mereka dan menggunakan sistem lebih maksimal.

8. Mengurangi beban ingatan jangka pendek.

   Keterbatasan manusia dalam mengolah informasi dalam jangka waktu yang pendek harus diperhatikan dalam membuat tampilan sehingga tidak menyulitkan pengguna.

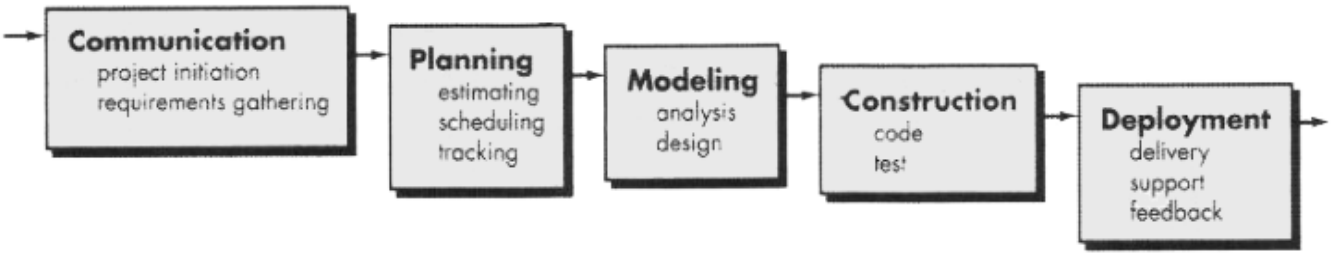
2.1.6 Teori *Development Software*

2.1.6.1 Metode Perancangan SDLC

System Development Life Cycle (SDLC) adalah proses keseluruhan meliputi pengembangan, implementasi, dan akhir penggunaan sistem informasi melalui proses bertingkat dari inisiasi, analisis, desain, implementasi, dan *maintenance* (Radack, 2009).

- **Model Proses *Waterfall***

Model proses *waterfall* adalah proses pengembangan *software* sekuensial, dimana kemajuan dipandang sebagai terus mengalir ke bawah (seperti air terjun) melalui tahapan konsepsi, inisiasi, analisis, konstruksi, pengujian dan pemeliharaan (Pressman, 2010, p39).



Gambar 2.4 *Waterfall Model*

Sumber: Roger Pressman (2010, p39)

Adapun tahap-tahap dalam model *waterfall* adalah sebagai berikut (Pressman, 2010, p39):

- ***Communication***

  Permodelan ini diawali dengan komunikasi dan kolaborasi dengan konsumen *(stackholders)* untuk mencari kebutuhan dari keseluruhan sistem yang akan diaplikasikan ke bentuk *software*. Hal ini sangat penting mengingat *software* harus dapat berinteraksi dengan elemen-elemen yang lain seperti *hardware*, *database*, dan sebagainya.

- ***Planning***

  Proses ini menetapkan rencana untuk pengerjaan *software* yang meliputi: tugas-tugas teknis yang akan dilakukan, resiko yang mungkin terjadi, sumber-sumber yang dibutuhkan, hasil yang akan dibuat, serta jadwal pengerjaannya.

- ***Modeling***

  Proses ini meliputi pembuatan model yang memungkinkan pengembang dan konsumen untuk lebih memahami kebutuhan perangkat lunak dan desain yang akan mencapai kebutuhan tersebut.

- ***Construction***

  Proses ini merupakan proses gabungan dari *coding* dan *testing*. Untuk dapat dimengerti oleh mesin komputer, maka desain tadi

harus diubah bentuknya menjadi bentuk yang dapat dipahami mesin, yaitu ke dalam bahasa pemrograman melalui proses *coding*. Tahap ini merupakan implementasi dari tahap desain yang secara teknis nantinya dikerjakan oleh *programmer*. Sedangkan *testing* adalah menguji coba sesuatu yang telah dibuat. Semua fungsi-fungsi *software* harus diujicobakan, agar *software* bebas dari *error*, dan hasilnya harus benar-benar sesuai dengan kebutuhan yang telah didefinisikan sebelumnya.

- ***Deployment***

  Pemeliharaan suatu *software* diperlukan, termasuk di dalamnya adalah pengembangan, karena *software* yang dibuat tidak selamanya hanya seperti itu. Ketika dijalankan mungkin saja masih ada *error* yang tidak ditemukan sebelumnya, atau mungkin terdapat fitur-fitur baru yang belum ada sebelumnya. Pengembangan dibutuhkan ketika adanya perubahan dari eksternal perusahaan seperti ketika ada pergantian sistem operasi, atau perangkat lainnya.

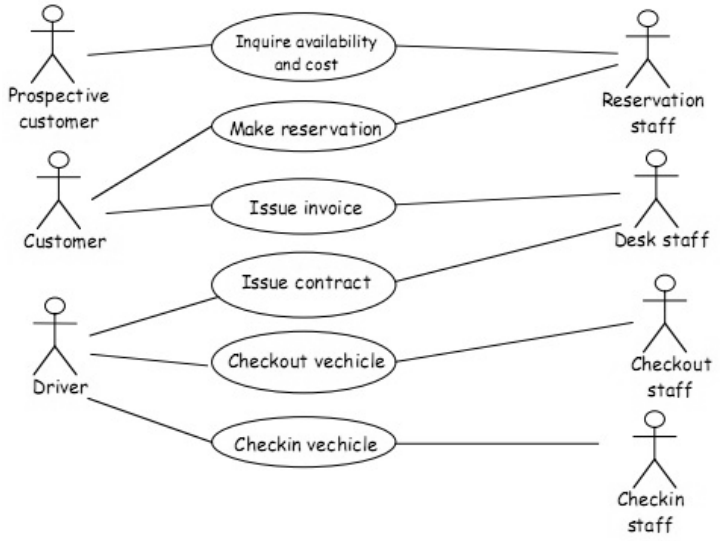
2.1.6.2 *Use Case Diagram*

Use case adalah spesifikasi dari urutan aksi yang sebuah sistem, sub sistem, atau *class* dapat lakukan dengan berinteraksi dengan aktor luar (Jacobson, 1992).

Beberapa komponen kunci analisis *use case* adalah sebagai berikut:

- *Actors*. Entitas yang menggunakan atau yang digunakan sistem. Umumnya orang, namun bisa saja sistem atau perangkat.
- *Connection*s. Penghubung antara aktor ke *use case*.
- *Relationships*. Hubungan antara aktor atau *use case*.

Contoh *use case diagram*:

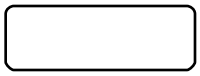


Gambar 2.5 *Use Case Diagram*

2.1.6.3 *Activity Diagram*

Activity diagram adalah diagram yang menggambarkan aksi-aksi dan keputusan-keputusan yang terjadi sesuai fungsi-fungsi yang telah dilakukan (Pressman, 2010, p853). *Activity diagram* terdiri dari beberapa komponen, yaitu:

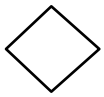
1. *Rounded rectangles*, menggambarkan sistem yang spesifik



2. *Arrows*, menggambarkan aliran aktivitas sistem

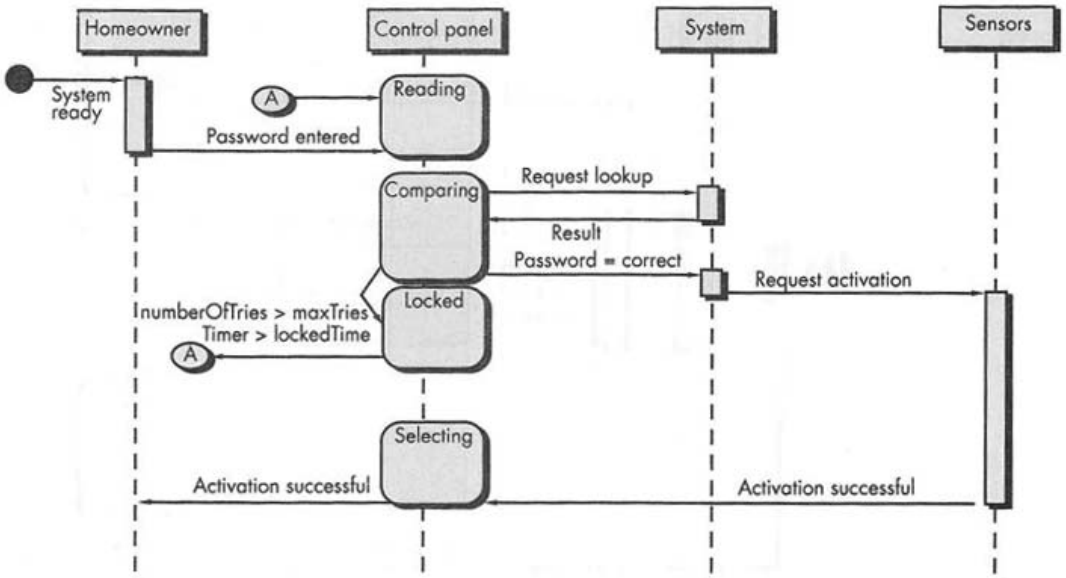
3. *Solid horizontal lines*, menggambarkan aktivitas paralel

4. *Decision diamond*, menggambarkan suatu keputusan



2.1.6.4 *Sequence Diagram*

Sequence diagram adalah diagram yang digunakan untuk menunjukkan komunikasi dinamis antara objek-objek saat suatu *task* dieksekusi, menunjukkan bagaimana peristiwa menyebabkan transisi dari objek ke objek (Pressman, 2010, p248).



Gambar 2.6 *Sequence Diagram*

Sumber: Roger Pressman (2010, p197)

2.1.7 Pengertian Java

Java adalah suatu bahasa pemrograman untuk keperluan umum, bersamaan, berbasiskan kelas, dan berorientasi objek. Java dirancang menjadi sederhana sehingga *programmer* dapat mencapai kefasihan dalam pengkodean. Bahasa pemrograman Java berhubungan dengan C dan C++ namun dirancang agak berbeda, dengan sejumlah aspek yang dimaksudkan untuk menjadi bahasa produk, bukan bahasa penelitian (Gosling et al, 2000, p1).

2.1.8 Pengertian Android

Android adalah *platform* pertama yang bersifat terbuka dan komprehensif untuk perangkat *mobile*. Sederhananya, Android adalah kombinasi dari tiga komponen, yaitu (Meier, 2010, p4):

1. Sistem operasi yang bersifat *open source* untuk perangkat *mobile*.
2. Sebuah *open source platform* untuk menciptakan aplikasi *mobile*.
3. Perangkat, telepon seluler khususnya, yang menjalankan sistem operasi Android dan aplikasi dibuat untuk itu.

Lebih spesifiknya, Android terdiri dari bagian-bagian yang saling dibutuhkan dan saling bergantung, seperti (Meier, 2010, p4):

1. Sebuah desain referensi *hardware* yang menggambarkan kemampuan yang dibutuhkan untuk sebuah perangkat *mobile* untuk mendukung *software stack*.

2. Sebuah *kernel* sistem operasi Linux yang menyediakan antarmuka tingkat rendah dengan *hardware*, manajemen memori, dan kontrol proses, semua dioptimalkan perangkat *mobile*.
3. *Open source libraries* untuk pengembangan aplikasi, termasuk SQLite, WebKit, OpenGL, dan *media manager*.
4. Sebuah *run time* yang digunakan untuk mengeksekusi dan *host* aplikasi Android, termasuk Dalvik *virtual machine* dan *core libraries* yang menyediakan fungsi spesifik dari Android. *Run time* ini dirancang kecil dan efisien untuk digunakan perangkat *mobile*.
5. Sebuah *framework* aplikasi yang menghadapkan layanan sistem ke lapisan aplikasi, termasuk *window manager* dan *location manager*, penyedia konten, *telephony*, dan sensor.
6. Sebuah *framework* antarmuka yang digunakan sebagai *host* dan menjalankan aplikasi.
7. Aplikasi yang sudah terpasang sebagai bagian dari *stack*.
8. Sebuah *software development kit* yang digunakan untuk membuat aplikasi, termasuk *tools, plug-ins*, dan dokumentasi.

Sistem operasi Android mirip sebuah kue yang terdiri dari berbagai lapisan. Setiap lapisan memiliki karakteristik dan fungsinya masing-masing. Setiap *layer* tidak seutuhnya terpisahkan tetapi saling bergantungan (Gargenta, 2011, p7).

Android *software stack* terdiri dari unsur-unsur yang ditunjukkan pada gambar 2.8. Sederhananya, sebuah *kernel* Linux dan koleksi *libraries* C/C++

mengarah pada *framework* aplikasi yang menyediakan layanan, pengelolaan, *run time*, dan aplikasi (Meier, 2010, p13).

Kernel Linux menjadi layanan inti termasuk penyedia *hardware drivers*, manajemen proses dan memori, keamanan, jaringan, dan manajemen tenaga. Kesemuanya itu ditangani oleh *kernel* 2.6 Linux. *Kernel* tersebut juga menyediakan sebuah *layer* abstaksi di antara *hardware* dan sisa dari *stack* (Meier, 2010, p13).

Libraries berjalan di *kernel* teratas, terdiri dari berbagai macam C/C++ *core libraries* seperti libc dan SSL, serta (Gargenta, 2011, p3):

- Sebuah *media library* untuk pemutaran media suara dan video.
- Sebuah *surface manager* yang menyediakan manajemen tampilan.
- *Graphics libraries* yang mencakup SGL dan OpenGL untuk grafis 2D dan 3D.
- SQLite untuk dukungan *database native*.
- SSL dan WebKit untuk integrasi *web browser* dan keamanan internet.

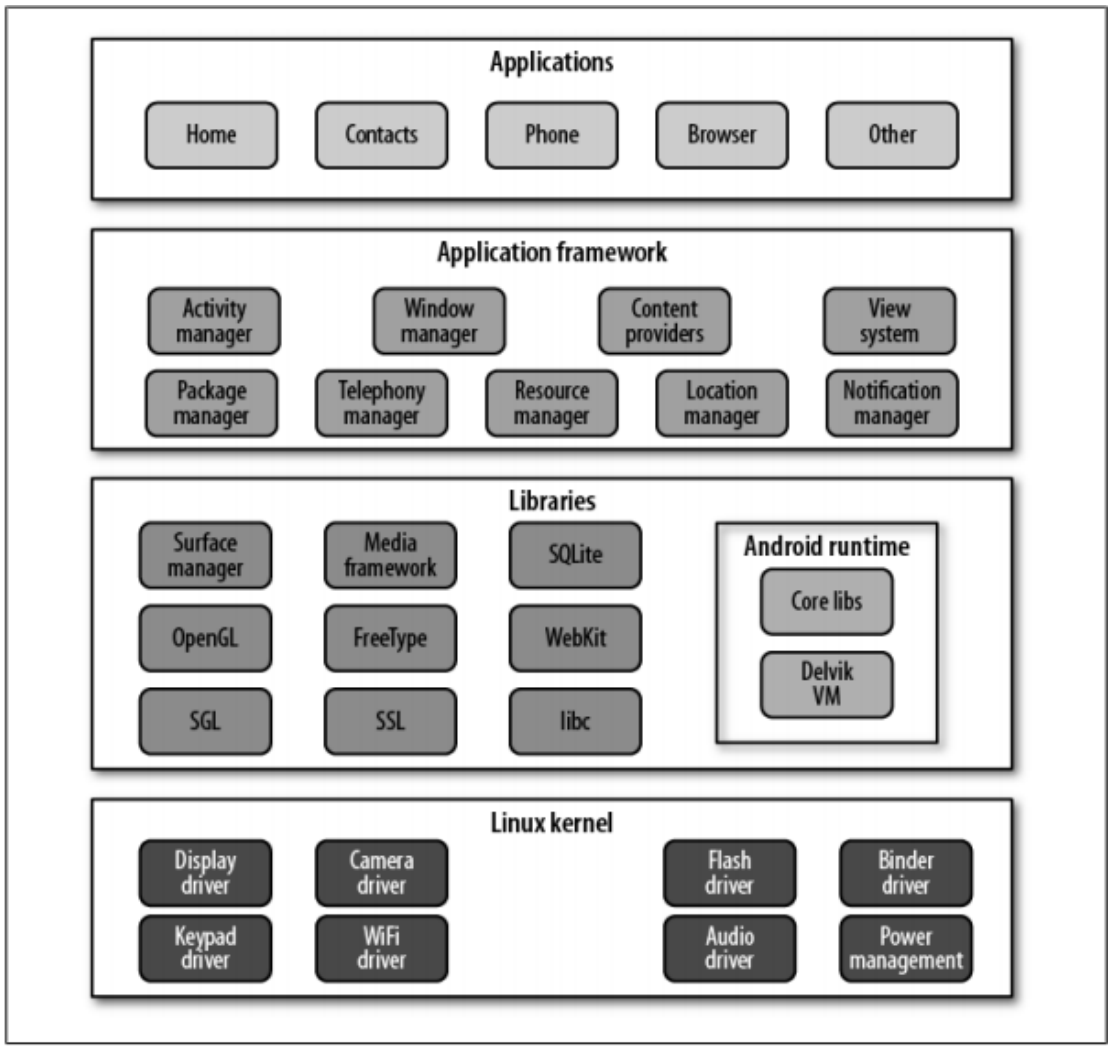
Android *run time* membuat perangkat Android lebih dari sebuah perangkat *mobile* dengan implementasi Linux. Terdapat *core libraries* dan Dalvik *virtual machine* di dalamnya. *Run time* pada Android adalah mesin yang memberikan kekuatan pada aplikasi, bersama dengan *libraries*¸ membentuk dasar *framework* aplikasi. *Core libraries* menyediakan sebaguan besar fungsi yang tersedia di dalamnya. Sedangkan Dalvik *virtual machine* adalah *virtual machine* yang telah dioptimalkan untuk memastikan bahwa

sebuah perangkat dapat menjalankan beberapa hal secara efisien (Meier, 2010, p13).

Application framework menyediakan kelas-kelas yang digunakan untuk membuat aplikasi Android. Selain itu juga menyediakan abstraksi umum untuk akses *hardware* dan mengatur *user interface* dan *application resources* (Meier, 2010, p14).

Application layer berjalan dalam *run time* Android, menggunakan kelas-kelas dan layanan yang disediakan dari *application framework* (Meier, 2010, p14).

Arsitektur pada Android mendorong konsep penggunaan kembali komponen, memungkinkan untuk mempublikasikan dan berbagi *activities*, layanan, dan data dengan aplikasi lainnya.



Gambar 2.7 *Stack* Pada Android

Sumber: Marko Gargenta (2011, p8)

Berikut layanan-layanan aplikasi yang menjadi pilar arsitektur dari semua aplikasi Android (Meier, 2010, p15):

- *Activity Manager*, digunakan untuk mengontrol daur hidup dari aktivitas, termasuk manajemen aktivitas *stack*.
- *Views*, digunakan untuk membangun *user interfaces* untuk aktivitas.

- *Notification Manager* menyediakan mekanisme yang konsisten dan tidak mengganggu untuk memberitahu *user*.
- *Content Providers* membiarkan aplikasi berbagi data.
- *Resource Manager* mendukung *non-code resources* seperti *strings* dan grafis.

Android mendukung aplikasi dan layanan yang didesain untuk berjalan secara tidak terlihat di *background*. Ponsel modern saat ini merupakan perangkat yang multifungsi. Namun, dengan ukuran layar yang terbatas berarti hanya ada satu aplikasi interaktif yang dapat terlihat dalam satu waktu. *Platform* yang tidak mendukung layanan *background* membatasi kelangsungan hidup suatu aplikasi. Layanan *background* memungkinkan untuk membuat komponen aplikasi yang tidak terlihat yang secara otomatis menjalankan proses tanpa tindakan langsung dari pengguna.

Android menawarkan kesempatan bagi pengembang untuk menciptakan aplikasi perangkat lunak yang inovatif untuk perangkat *mobile* tanpa batasan umum terkait dengan pemilik *mobile development frameworks* yang ada.

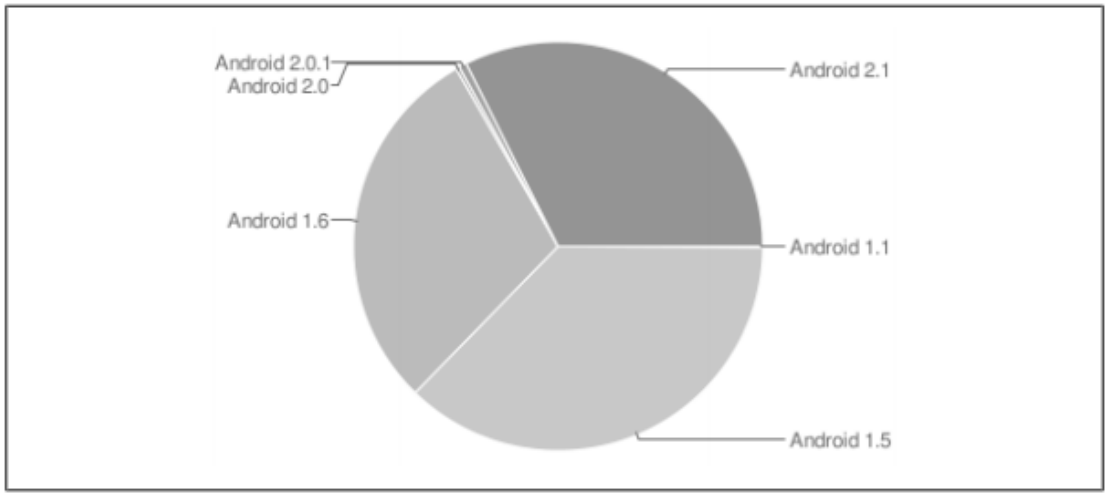
Seperti halnya *software* lainnya, Android berkembang dari waktu ke waktu, yang mana terlihat dari versi yang dikeluarkan Android.

Berikut daftar versi Android:

Tabel 2.1 Tabel Versi Android

Android Version	API Level	Nickname
Android 1.0	1	
Android 1.1	2	
Android 1.5	3	Cupcake
Android 1.6	4	Donut
Android 2.0	5	Eclair
Android 2.01	6	Eclair
Android 2.1	7	Eclair
Android 2.2	8	Froyo
Android 2.3	9	Gingerbread
Android 2.3.3	10	Gingerbread
Android 3.0	11	Honeycomb

Sumber: Marko Gargenta (2011, p4)



Gambar 2.8 Distribusi Pengguna Versi Android Sampai Dengan Januari 2011

Sumber: Marko Gargenta (2011, p5)

2.2 Teori Khusus

2.2.1 Pengertian *Web Service*

Web service adalah aplikasi yang berdiri sendiri yang dapat diakses melalui internet, modular, dan dideskripsikan sendiri. *Web service* dideskripsikan menggunakan *Web Services Description Language* (WSDL). *Web Services Description Language* (WSDL) menspesifikasikan struktur komponen pesan menggunakan skema XML (Cardoso, 2007, p17).

2.2.2 Pengertian SQLite

SQLite adalah sebuah *embedded database* yang sangat terkenal karena menggabungkan antarmuka SQL dengan memori yang sangat kecil dan kecepatan yang baik (Murphy, 2010, p225).

SQLite adalah sebuah *open source database* yang telah ada cukup lama, cukup stabil, dan sangat terkenal pada perangkat kecil, termasuk Android (Gargenta, 2011, p119).

Android menyediakan *database* relasional yang ringan untuk setiap aplikasi menggunakan SQLite. Aplikasi dapat mengambil keuntungan dari itu untuk mengatur *relational database engine* untuk menyimpan data secara aman dan efisien (Meier, 2010, p7).

Untuk Android, SQLite dijadikan satu di dalam Android *runtime*, sehingga setiap aplikasi Android dapat membuat basis data SQLite. Karena SQLite menggunakan antarmuka SQL, cukup mudah untuk digunakan orang-

orang dengan pengalaman lain yang berbasis *databases* SQL (Murphy, 2010, p225).

Terdapat beberapa alasan mengapa SQLite sangat cocok untuk pengembangan aplikasi Android, yaitu:

- *Database* dengan konfigurasi nol. Artinya tidak ada konfigurasi *database* untuk para *developer*. Ini membuatnya relatif mudah digunakan.
- Tidak memiliki *server*. Tidak ada proses *database* SQLite yang berjalan. Pada dasarnya satu set *libraries* menyediakan fungsionalitas *database*.
- *Single-file database*. Ini membuat keamanan *database* secara langsung.
- *Open source*. Hal ini membuat *developer* mudah dalam pengembangan aplikasi.

2.2.3 Pengertian XML

XML adalah bahasa *markup* yang menggunakan *tags* untuk memberi label, mengkategorikan, dan mengorganisir informasi dengan cara tertentu. *Markup* mendeskripsikan dokumen atau struktur data serta organisasi. Konen, seperti teks, gambar, dan data adalah bagian dari kode yang mengandung *tag* markup. XML tidak dibatasi untuk satu set *markup* tertentu. Anda dapat membuat *markup* sendiri sesuai dengan data dan kebutuhan dokumen Anda (Dykes dan Tittel, 2005, p11).

XML adalah bahasa *markup* yang dapat diperluas yang menjadi rekomendasi *World Wide Web Consortium*. XML telah hampir secara

universal diterima sebagai solusi interoperabilitas untuk sistem komputer yang berbeda-beda. Meskipun tidak tanpa kekurangan, XML mungkin adalah solusi terbaik untuk menangani masalah interoperabilitas perangkat lunak, terutama dikarenakan oleh penerimaan yang luas dan kehadirannya. Spesifikasi XML mendefinisikan sintaks untuk membuat *markup*. *Markup* terdiri dari elemen-elemen, atribut, dan struktur lain yang memungkinkan Anda untuk memberi label dokumen-dokumen dan data (Fitzgerald, 2004, pxiii).

2.2.4 *Multi-Channel Learning* (MCL)

Sistem pembelajaran di Universitas Bina Nusantara, dimana sistem ini mengkombinasikan beberapa saluran pembelajaran, pembelajaran di kelas, *e-learning* dan belajar mandiri, serta mengkombinasikan sesi F2F (*Face to Face*) dan sesi GSLC (*Guided Self Learning Class*) (BINUS University).

2.2.5 Binusmaya

Binusmaya merupakan bagian dari *Multi Channel Learning*, materi pembelajaran disimpan secara digital di Binusmaya. Binusmaya merupakan tempat bagi mahasiswa untuk mengakses materi kuliah serta berdiskusi dengan sesama mahasiswa dan dosen (BINUS University).

2.2.6 *Face to Face* (F2F)

Sesi dimana mahasiswa dan dosen melakukan kegiatan belajar mengajar melalui tatap muka secara langsung di kelas. Adapun kegiatan

belajar mengajar seperti membaca materi pembelajaran secara lengkap, memberikan atau mengumpulkan tugas, memberikan ide/konsep/aplikasi dan aktif berpartisipasi pada forum diskusi dan menggunakan waktu untuk proses pembelajaran secara maksimal (BINUS University).

2.2.7 *Guided Self Learning Class* (GSLC)

Sesi dimana mashasiwa dan dosen melakukan kegiatan belajar mengajar melalui sarana *online* seperti Binusmaya. Umumnya kegiatan belajar mengajar yang dilakukan seperti berdiskusi melalui forum dan menjawab/menanggapi topik diskusi yang diberikan dosen (BINUS University).