BAB 2

LANDASAN TEORI

2.1 Teori Umum

2.1.1 Internet dan Website

1. Pengertian *Internet*

Menurut Connolly dan Begg (2010, p1024) *internet* adalah sebuah jaringan global yang berinterkoneksi dengan jaringan-jaringan komputer.

Internet terdiri dari banyak jaringan yang terpisah namun saling berhubungan, jaringan tersebut dimiliki secara komersial oleh organisasi pendidikan dan pemerintahan, dan juga *Internet Service Provider* (ISP). Layanan yang ditawarkan oleh *Internet* meliputi surat elektronik (*email*), konferensi dan layanan mengrobrol (*chatting*), pengaksesan komputer dari jarak jauh, serta mengirim dan menerima berkas.

2. Sejarah *Internet*

Menurut Connolly dan Begg (2010, p1025) *Internet* dimulai pada akhir 1960-an dan awal 1970-an sebagai *Department of Defense* US poyek yang disebut ARPANET (*Advanced Research Projects Agency Network*). ARPANET

menyelidiki bagaimana cara membangun jaringan-jaringan yang dapat tetap bertahan terhadap pemadaman parsial (seperti serangan bom nuklir).

Pada tahun 1982, TCP / IP (*Transmission Control Protocol* dan *Internet Protocol*) diadopsi sebagai komunikasi standar protokol untuk ARPANET. TCP bertanggung jawab untuk memastikan kebenaran dalam pengiriman pesan yang berpindah dari satu komputer ke komputer lain. IP mengelola pengiriman dan penerimaan paket data melalui mesin, berdasarkan destinasi alamat empat-*byte* (nomor IP), yang ditugaskan kepada sebuah organisasi oleh otoritas *internet*. Istilah TCP / IP kadang-kadang mengacu ke seluruh jaringan *internet* protokol yang umumnya berjalan pada TCP / IP, seperti FTP (*File Transfer Protocol*), SMTP (*Simple Mail Transfer Protocol*), Telnet (*Telecommunication Network*), DNS (*Domain Name Service*), POP (*Post Office Protocol*), dan sebagainya.

Internet pertama kali mendapatkan sumbangan dana dari NSF AS, agar universitas di Amerika dapat berbagi *resource* yang berasal dari lima pusat super komputer nasional. Jumlah pengguna tumbuh dengan cepat dikarenakan harga aksesnya cukup murah bagi pengguna domestik yang ingin memiliki jaringan pada komputernya. Pada awal 1990-an, kekayaan informasi yang tersedia secara gratis pada jaringan telah meningkat, sehingga

sejumlah layanan pengindeksan dan pencarian bermunculan untukmenjawab permintaan pengguna seperti Archie, Gopher, Veronica, dan WAIS (*Wide Area Information Service*), yang menyediakan layanan melalui *menu-based interface*. Sebaliknya, *web* menggunakan *hypertext* untuk memungkinkan melakukan *browsing*, dan sejumlah *web-basedsearch engine* membuat Google, Yahoo, dan MSN.

Internet diperkirakan memiliki lebih dari 100 juta pengguna pada bulan Januari 1997. Satu tahun kemudian, perkiraan tersebut telah meningkat menjadi lebih dari 270 juta pengguna dalam 100 negara, dan pada, akhir tahun 2004 kenaikan lebih meningkat lagi menjadi 945 juta pengguna. Dan pada tahun 2010 menjadi lebih dari 2 miliar pengguna. Selain itu, sekarang terdapat 3,5 juta dokumen di dalam *internet*, dan jumlah tersebut terus meningkat sebanyak 7.5 juta per tahunnya. Jika kita menggabungkan *intranet* dan *extranet*, maka dokument tersebut akan meningkat secara signifikan menjadi 550 juta pertahunnya.

3. *Intranet* dan *Extranet*

a. *Intranet*

Menurut Connolly dan Begg (2010, p1026), *intranet* merupakan sebuah situs *web* atau sekelompok situs yang

dimiliki oleh suatu organisasi, yang pengaksesannya hanya dapat dilakukan oleh anggota organisasi.

Biasanya, *intranet* terhubung pada *internet* umum yang lebih luas melalui *firewall*, hal itu dilakukan dengan pembatasan terhadap jenis informasi yang masuk dan keluar dari *intranet*. Misalnya, *staff* mungkin diperbolehkan untuk menggunakan surat elektronik eksternal dan mengakses situs *web* eksternal, tetapi orang-orang dari luar organisasi memiliki kemampuan terbatas untuk mengirim surat elektronik ke dalam organisasi dan orang tersebut dilarang untuk melihat halaman *web* yang diterbitkan dalam intranet.

b. *Extranet*

Menurut Connolly dan Begg (2010, p1026) *extranet* merupakan sebuah *intranet* yang beberapa bagiannya dapat diakses oleh orang luar yang berwenang.

Extranet menyediakan berbagai tingkat aksesibilitas bagi orang luar. Biasanya, *extranet* dapat diakses hanya jika orang luar memiliki *username* dan *password* yang *valid*, dan identitas tersebutlah yang menentukan bagian mana dari *extranet* yang dapat dilihat. *Extranet* telah

menjadi sarana yang sangat populer untuk mitra bisnis dalam hal bertukar informasi.

4. *Website*

Menurut Connolly dan Begg (2010, p1028), *website* merupakan sebuah sistem berbasis *hypermedia* yang menyediakan sarana untuk melakukan pencarian informasi pada *internet* menggunakan *hyperlink* dengan cara tidak berurutan (*non-sequential*).

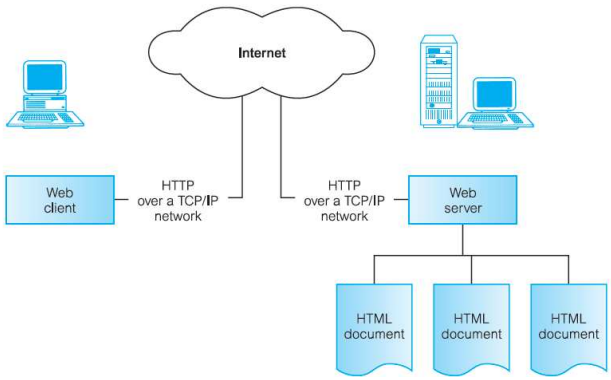
Informasi di *web* disajikan pada halaman *web*, yang muncul sebagai kumpulan teks, grafis, gambar, suara, dan video. Selain itu, halaman *web* berisi *hyperlink* ke halaman *web* yang lain, yang memungkinkan pengguna dapat melakukan menavigasi secara non-sekuensial melalui informasi.

Sebagian besar keberhasilan *web* adalah karena kesederhanaannya, yang memungkinkan pengguna dapat memberikan, menggunakan, dan melihat informasi yang terdistribusi secara geografis di seluruh dunia. Selain itu, pengguna memiliki kemampuan untuk menelusuri dokumen multimedia secara independen dari perangkat keras komputer yang digunakan. Hal ini juga kompatibel dengan data komunikasi protokol lainnya, seperti Gopher, FTP (*File Transfer Protocol*),

NNTP (*Network News Transfer Protocol*), dan Telnet (untuk sesi *remote login*).

Web terdiri dari jaringan komputer yang dapat bertindak dalam dua peran, yaitu sebagai *server* (memberikan informasi), dan sebagai klien / *browser* (meminta informasi).

Sebagian besar informasi di *web* disimpan dalam dokumen dengan menggunakan bahasa yang disebut HTML (*HyperText Markup Language*), dan browser harus memahami serta menafsirkan HTML untuk menampilkan dokumen-dokumen. Protokol yang mengatur pertukaran informasi antara *web server* dan *browser* disebut HTTP (*HyperText Transfer Protocol*). Dokumen dan lokasi dokumen tersebut diidentifikasi oleh alamat dalam suatu *Uniform Resource Locator* (URL).



Gambar 2.1 Komponen Dasar Dalam Lingkungan *Web*

(*Database Systems A Pratical Approach To Design, Implementation, And Managemet*, P1029)

5. Aplikasi *Web*

Menurut Pressman (2010, p8) aplikasi *web* yang sering disebut sebagai "*WebApps*" adalah kategori perangkat lunak yang berpusat pada jaringan *web*. Singkatnya, *WebApps* bukan hanya sekumpulan *hypertext file* yang saling terhubung yang dipresentasikan dalam tulisan dan gambar. Tetapi, sejalan dengan munculnya *web* 2.0, *WebApps* berkembang menjadi sebuah teknologi komputer canggih yang tidak hanya menyediakan fitur untuk sendiri, fungsi dasar komputer dan konten kepada pengguna, tetapi juga telah terhubung dengan *database* perusahaan dan aplikasi bisnis.

Sebuah aplikasi *web* dapat didefinisikan sebagai sebuah aplikasi yang dijalankan pada server *web* yang diakses melalui aplikasi *web browser* pada *desktop* atau pada perangkat melalui *internet* atau *intranet*. Sebuah aplikasi *web* pada dasarnya adalah sebuah aplikasi perangkat lunak yang di-kode kan untuk *browser* oleh *browser supported language* (misalnya, HTML, JavaScript, Java) dan bergantung pada *browser web* umum untuk membuat aplikasi dapat dieksekusi.

Aplikasi *web* pada dasarnya merupakan satu set halaman *web* yang menghasilkan halaman statis dan juga halaman yang dinamis yang memungkinkan pengguna aplikasi untuk melakukan tugas spesifik. Berbeda dengan aplikasi *desktop* konvensional, aplikasi *web*

dapat diakses oleh pengguna aplikasi untuk mencari informas dimana saja, dan tidak terhalang oleh batas-batas geografis. Contoh aplikasi *web* yang paling sering digunakan adalah klien *email* (perangkat lunak untuk menerima, melakukan proses, dan mengirim surat elektronik), sistem manajemen kontak (perangkat lunak untuk menjaga rincian kontak bisnis), perbankan *online* dan berbagi sistem transaksi, aplikasi jejaring sosial, dan lain lain.

6. Arsitektur Aplikasi *Web*

Menurut Schneider (2011, p358), arsitektur *web* yang paling sederhana adalah model *two-tier*. Model arsitektur *web* ini hanya mempunyai 1 klien dan 1 *server*. Seluruh komunikasi berada diantara klien dan *server* tersebut.

Walaupun model arsitektur *two-tier* klien / *server* dapat bekerja dengan baik pada banyak halaman *web*, namun sebuah *website* yang menyampaikan konten dinamis dan mempunyai proses transaksi yang lebih daripada hanya menyampaikan konten statis, perlu lebih dari model arsitektur *two-tier*.

Arsitekur *three-tier* merupakan arsitektur pengembangan dari arsitektur *two-tier*. Arsitektur *three-tier* memungkinkan proses tambahan (seperti mengambil informasi dari *database* untuk menghasilkan *website* dinamis) terjadi sebelum *web server* merespon permintaan dari *web client*. Arsitektur *three-tier* biasanya

mengandung *database* dan aplikasi perangkat lunak yang dapat memberikan informasi ke *web server*. *Web server* kemudian dapat menggunakan output dari aplikasi perangkat lunak saat menanggapi permintaan klien, bukan hanya memberikan sebuah halaman *web*.

Arsitektur *web* dapat memiliki empat, lima, atau bahkan tingkatan lebih dibagi menjadi tingkatan-tingkatan yang memisahkan aplikasi perangkat lunak basis data dan program manajemen basis data (*Database Management Program*) yang bekerja dengan aplikasi perangkat lunak. Juga beberapa situs memiliki aplikasi perangkat lunak yang menghasilkan informasi (tingkat keempat) yang memberikan *input* ke aplikasi perangkat lunak lain atau *database* (di tingkat ketiga) yang kemudian akan menghasilkan informasi untuk *web server* untuk diubah menjadi halaman *web* (di lapis kedua), yang kemudian pergi ke klien yang meminta (di tingkat pertama). Arsitektur yang memiliki lebih dari tiga tingkatan sering disebut arsitektur *n-tier*.

7. HTTP

Menurut Connolly dan Begg (2010, p1029) *The HyperText Transfer Protocol* (HTTP) merupakan protokol yang digunakan untuk mentransfer halaman *web* melalui *Internet*. HTTP mendefinisikan bagaimana caranya klien dan *server* saling berkomunikasi dan mengirimkan informasi.

Transaksi HTTP terdiri dari empat tahapan yaitu:

a. *Connection* - Klien menetapkan koneksi dengan *web server*.

b. *Request* - Klien mengirimkan pesan permintaan kepada *web server*.

c. *Respond* - *web server* mengirimkan respon (misalnya, dokumen HTML) ke klien.

d. *Close* - Sambungan ditutup oleh *web server*.

Pemrograman Berorientasi Objek (*Object Oriented Programming*)

Menurut Zhu (2012, p237), *Object Oriented Programming* adalah sebuah paradigma *programming* yang menggunakan objek atau kumpulan struktur data yang mengandung data dan *method* secara bersamaan dalam interaksinya.

Menurut Kedar (2009, p6-3), Pemograman berorientasi objek merupakan suatu pendekatan yang menyediakan sebuah cara untuk memodulkan program dengan cara menciptakan area memori yang dipartisi untuk data dan suatu fungsi yang dapat digunakan sebagai *template* untuk membuat salinan modul tersebut sesuai permintaan.

Fitur yang terdapat dalam pemograman berorientasi objek:

- Penekanan terjadi pada data, daripada fungsi.

- Program dibagi menjadi beberapa obyek.
- Fungsi dan data terikat bersama-sama dalam struktur data.
- Data tersembunyi dan tidak dapat diakses oleh fungsi eksternal.
- Objek dapat berkomunikasi satu sama lain melalui fungsi.
- Data baru dan fungsi dapat dengan mudah ditambahkan bila diperlukan.

8. Konsep OOP

Menurut Sanders dan Cumaranatunge (2007, p11), OOP terdiri dari empat konsep, yaitu abstraksi, enkapsulasi, *inheritance,* polimorfisme.

a. Abstraksi

Abstraks*i* merupakan sebuah model, atau ideal. Abstraksi tidak memiliki semua *detail*, tetapi abstraksi memiliki parameter-parameter yang bersifat umum yang dapat diisi dengan *detail-detail* tertentu.

Abstraksi menunjukkan karakteristik penting dari sebuah objek yang membedakan dari objek lainnya. Abstraksi bersifat fleksibel dan fokus dalam pemecahan masalah terhadap bagian-bagian tertentu.

b. Enkapsulasi

Enkapsulasi dapat dikatakan sebagai suatu kumpulan oeprasi dan sifat dalam suatu objek. Enkapsulasi sering disebut juga sebagai komponen atau modul. Dalam konteks OOP, enkapsulasi sering disebut *black box*, yang berarti dapat melihat sebuah benda bekerja, tetapi tidak bisa melihat cara kerja di dalamnya. Hal yang baik dari konsep *black box* adalah bahwa kita tidak perlu khawatir dengan cara kerja yang terjadi didalamnya, kita hanya perlu tahu cara menghadapi benda tersebut bekerja dengan aman dan baik seperti yang kita pikirkan.

c. *Inheritance*

Inheritance mengarah pada bagaimana sebuah kelas dari objek dapat menurunkan *properties*, *method* dari kelas yang lain.

d. Polimorfisme

Polimorfisme merupakan sebuah proses metamorfosis, yang berarti berubah atau transformasi. Sebuah kelas bisa mewakili berbagai bentuk tipe data

9. Keuntungan OOP

Menurut Ravichandran (2011, P6) beberapa keuntungan dari menggunakan OOP adalah sebagai berikut

a. OOP menyediakan struktur sintaks yang lebih baik, masalah *modelling real world* menjadi mudah dan fleksibel.

b. Sistem perangkat lunak yang kompleks dapat di modular berdasarkan objek dan kelas.

c. Pembuatan dan perawatan kode OOP mudah dan menghemat waktu pengembangan perangkat lunak.

d. Karena teknik OOP mendukung komponen perangkat lunak perpustakaan yang *reusable*, maka rekayasa ulang perangkat lunak dapat disintesis, dilaksanakan dan direalisasikan dengan mudah.

e. Data enkapsulasi dan informasi yang tersembunyi meningkatkan keandalan perangkat lunak.

f. Polimorfisme dan *dynamic binding* meningkatkan fleksibilitas kode dengan memungkinkan penciptaan komponen *software* generik.

g. *Inheritance* memungkinkan kode *software* untuk diperluas dan dapat digunakan kembali..

Basis Data (*Database*)

Menurut Connolly dan Begg (2010, p65). Basis data merupakan sekumpulan data logikal yang selaing berkaitan dan deskripsi dari sebuah data yang dirancang untuk memenuhi kebutuhan informasi dari suatu organisasi.

Menurut Whitten dan Bentley (2007, p521), dalam basis data terdapat *field* dan *record*. *Field* adalah unit terkeceil dalam data yang disimpan dalam basis data. Sedangkan *record* adalah kumpulan *field* yang telah diatur dan ditentukan.

Terdapat beberapa *field* dalam basis data, antara lain:

- *Primary Key* merupakan sebuah *candidate key* atau *field* yang unik, dan dipilih untuk mengidentifikasi *tuples* secara unik dalam suatu relasi. *Primary key* yang menjadi identitas sebuah *record*.
- *Foreign Key* merupakan *field* atau atribut, dalam satu relasi yang sesuai dengan *candidate key* dari relasi yang sama, yang menunjuk ke tabel lain dalam basis data.
- *Secondary Key* adalah *field* yang menidentifikasi sebuah *record* atau sebagian *record* yang berhubungan.

- *Descritpive Field* adalah *field* yang tidak termasuk dalam *key field*.

10. Normalisasi

a. Pengertian Normalisasi

Menurut Connolly dan Begg (2010, p416), normalisasi merupakan sebuah teknik yang memproduksi satu set hubungan dengan sifat yang diinginkan, mengingat akan adanya kebutuhan data dari suatu perusahaan. Tujuan dari normalisasi adalah untuk mengidentifikasi satu set hubungan yang cocok yang mendukung kebutuhan persyaratan data suatu perusahaan.

Menurut Wang, Hu dan Lehmann (2010, p41) , tujuan dari normalisasi *database* adalah untuk memungkinkan penyimpanan data tanpa perulangan data yang tidak perlu dan mengurangi data yang tidak konsisten dengan demikian pengguna (*user*) dapat memelihara (*maintanance*) dan menerima data dari *database* tanpa kesulitan.

Database yang normal dapat mengeliminasi anomali dalam proses memasukan, menghapus, atau memperbaharui data. Hal ini dapat meningkatkan efektifitas dan efisiensi dari *database* itu sendiri.

b. Proses dalam Normalisasi

Menurut Connolly dan Begg (2010, p428), proses normalisasi merupakan sebuah metode formal yang menganalisis hubungan berdasarkan *primary key* atau *candidate key* dan ketergantungan fungsional antara atribut-atribut yang dimilikinya. Teknik normalisasi ini melibatkan serangkaian aturan yang dapat digunakan untuk menguji hubungan antar data sehingga *database* dapat normal. Bila persyaratan tidak terpenuhi, maka hubungan tersebut melanggar persyaratan, sehingga harus didekomposisi menjadi relasi yang secara individual memenuhi persyaratan normalisasi.

Tiga bentuk normal yang pertama kali diusulkan oleh E.F. Codd disebut bentuk normal pertama (1NF), bentuk normal kedua (2NF), dan bentuk normal ketiga (3NF).

Normalisasi sering dieksekusi sebagai serangkaian langkah-langkah. Setiap langkah sesuai dengan bentuk normal yang spesifik yang dikenal sebagai *properties*. Sebagai hasil normalisasi, hubungan menjadi semakin lebih kuat dan juga mengurangi kerentanan terhadap *update anomali*. Untuk model data relasional, penting untuk menyadari bahwa hanya bentuk normal pertama

(1NF) sangat penting dalam menciptakan hubungan, semua bentuk normal berikutnya adalah opsional.

Normalisasi menggunakan teknik *bottom-up* dalam penggalian informasi tentang atribut dari bentuk sampel yang pertama kali diubah menjadi format tabel, hal itu digambarkan dalam *unnormalized form* (UNF).

Satu set fungsional dependensi diberikan untuk setiap relasi dan setiap hubungan masing-masing ditunjuk ke *primary key*. Dengan kata lain, penting bahwa pengertian dari atribut dan hubungan dipahami dengan baik sebelum memulai proses normalisasi. Informasi ini adalah dasar normalisasi yang digunakan untuk menguji apakah relasi adalah merupakan bentuk normal.

- *Unnormalized form (UNF)*

  Tabel yang berisi satu atau lebih kelompok yang berulang.

- Bentuk normal pertama (1NF)

  Sebuah relasi di mana disetiap baris dan kolom berisi satu dan hanya satu nilai.

- Bentuk normal kedua (2NF)

  Semua relasi yang ada di bentuk normal pertama dan setiap atribut *non-primary-key* yang sepenuhnya bergantung secara fungsional pada setiap *candidate-key*. Dalam definisi ini, atribut *primary-key* merupakan bagian dari *candidate-key.*

  Full functional dependency menunjukkan bahwa jika A dan B adalah relasi atribut, maka B sepenuhnya bergantung secara fungsional pada A tetapi tidak pada setiap bagian yang *proper* dari A.

- Bentuk normal ketiga (3NF)

  Semua relasi yang ada di bentuk normal pertama dan bentuk normal kedua yang tidak memiliki atribut *non-primary-key* yang sepenuhnya bergantung transitif pada setiap *candidate-key*. Dalam definisi ini, atribut primary-key merupakan bagian dari *candidate-key*.

  Transitive dependency adalah suatu kondisi dimana A, B, dan C adalah atribut dari relasi tersebut, jika A → B dan B → C, maka C adalah

transitif tergantung pada A melalui B (A tidak secara fungsional tergantung pada B atau C).

Interaksi Manusia dan Komputer (IMK)

Menurut Shneiderman (2010, p113) interaksi manusia dan komputer merupakan ilmu yang berhubugan dengan perancangan, evaluasi, dan implemntasi yang mempelajari cara komunikasi antara komputer dan pengguna.

Interaksi manusia dan komputer merupakan desain ilmu pengetahuan interdisipliner yang menggabungkan metode pengumpulan data dan kerangka intelektual psikologi eksperimental dengan menggunakan alat yang kuat dan banyak digunakan manusia serta dikembangkan oleh ilmu komputer

11. Delapan Aturan Emas (*Eight Golden Rules*)

Menurut Shneiderman (2010, p5) ada 8 hal yang perlu diperhatikan untuk membuat suatu desain antar muka pengguna yang disebut dengan 8 aturan emas (*eight golden rules*)

a. Konsisten

Berusaha untuk konsisten dalam merancang desain antar muka pengguna. Konsistensi bisa dilakukan pada urutan tindakan dan perintah yang dilakukan. Konsistensi

juga perlu diterapkan untuk istilah yang digunakan pada *prompt*, menu, serta layar bantuan.

b. Menyediakan Kegunaan *Universal*

Untuk pengguna yang baru menggunakan sistem, sediakan fungsi-fungsi umum yang biasa ditemui pada sistem atau aplikasi lainnya sehingga dapat lebih mudah dan cepat dimengerti oleh pengguna. Untuk pengguna yang lebih berpengalaman (*expert user*), sediakan *shortcut* yang juga bersifat universal agar dapet membantu pengguna dalam menggunakan aplikasi.

c. Memberikan umpan balik yang informatif.

Untuk setiap tindakan operator, sebaiknya disertakan suatu umpan balik. Untuk tindakan yang sering dilakukan dan tidak terlalu penting, dapat diberikan umpan balik yang sederhana. Tetapi ketika tindakan merupakan hal yang penting, maka umpan balik sebaiknya lebih terperinci. Misalnya muncul suatu suara ketika salah menekan tombol pada waktu input data atau muncul pesan kesalahannya. Dengan adanya umpan balik yang informatif juga dapat membantu pengguna dalam memudahkan mereka menggunakan aplikasi.

d. Merancang dialog untuk menghasilkan suatu penutupan.

Urutan tindakan sebaiknya diorganisir dan diatur dalam suatu kelompok dengan bagian awal, tengah, dan akhir. Umpan balik yang informatif akan meberikan indikasi bahwa cara yang dilakukan sudah benar dan dapat mempersiapkan tindakan-tindakan berikutnya.

e. Memberikan penanagan kesalahan dengan mudah.

Sedapat mungkin sistem dirancang sehingga pengguna tidak dapat melakukan kesalahan fatal. Jika kesalahan terjadi, sistem dapat mendeteksi kesalahan dengan cepat dan memberikan mekanisme yang sedehana dan mudah dipahami untuk penanganan kesalahan.

f. Memungkinkan pengguna kembali ke tindakan sebelumnya

Hal ini dapat mengurangi kekuatiran pengguna karena pengguna mengetahui kesalahan yang dilakukan dapat dibatalkan sehingga pengguna tidak takut untuk mengekplorasi pilihan-pilihan lain yang belum biasa digunakan. Dengan kemingkinan pengguna lebih mengeksplorasi aplikasi, maka diharapkan pengguna dapat mengerti dan memahami aplikasi itu.

g. Mendukung tempat pengendali internal (*Internal Locus of Control*)

Pengguna ingin menjadi pengontrol sistem dan sistem akan merespon tindakan yang dilakukan pengguna daripada sistem yang mengontrol pengguna. Sebaiknya sistem dirancang sedemikan rupa sehingga pengguna menjadi inisiator daripada responden.

h. Mengurangi beban ingatan jangka pendek

Keterbatasan ingatan manusia membutuhkan tampilan yang sederhana atau tampilan halaman-halaman yang banyak yang sebaiknya disatukan jika memang dapat disatukan, serta diberikan cukup waktu pelatihan untuk kode, *mnemonic* dan urutan tindakan.

12. 5 Faktor Manusia Terukur

Menurut Shneiderman (2010, p38), terdapat lima faktor manusia terukur yang dapat dijadikan sebagai pusat evaluasi.

Kelima faktor tersebut adalah:

a. Waktu Belajar (*time to learn*), berapa lama waktu yang diperlukan pengguna untuk mempelajari cara yang relevan untuk melakukan suatu tugas.

b. Kecepatan Kinerja (*speed of performance*), berapa lama waktu yang diperlukan pengguna untuk melakukan suatu tugas.

c. Tingkat Kesalahan (*rate of error*), berapa banyak kesalahan dan kesalahan apa yang dapat terjadi ketika pennguna melakukan tugas.

d. Daya Ingat (*rentention over time*), bagaimana kemampuan pengguna mengingat pengetahuan yang didapat setelah jangka waktu tertentu. Daya ingat berkaitan dengan waktu belajar dan frekuensi penggunaan.

e. Kepuasan Subjektif (*subjective statisfaction*), seberapa suka pengguna meggunakan bermacam aspek antarmuka? Jawaban dapat diperoleh dengan melakukan survei atau wawancara.

2.1.2 Rekayasa Piranti Lunak

1. Pengertian Perangkat lunak

Menurut Pressman (2010, p5) perangkat lunak (*software*) adalah perintah instruksi (program komputer) yang jika dijalankan dapat menghasilkan fitur, fungsi, dan performa. Pengertian lain adalah struktur data yang membuat program dapat melakukan manipulasi informasi. Selain itu, perangkat lunak juga mempunyai pengertian lain yaitu menjelaskan

informasi baik secara fisik atau bentuk virtual yang menjelaskan kegunaan program.

Menurut Pressman (2010, p7) aplikasi perangkat lunak adalah sebuah program yang berdiri sendiri dan mampu memberikan solusi terhadap kebutuhan bisnis. Aplikasi dalam area ini dapat memfasilitasi operasi bisnis atau pengambilan keputusan oleh *management*.

2. Pengertian Rekayasa Piranti Lunak

Menurut Pressman (2010, p13) *software engineering* adalah aplikasi dari sebuah pendekatan yang sistematis, disiplin, dan kuantitiatif untuk membangaun, mengoperasikan dan memelihara *sofware*. *Sofware engineering* adalah teknologi berlapis dimana setiap pendekatan teknis harus bersesuaian dengan prinsip-prinsip yang telah disepakati.

Gambar 2.2 *Software Engineering Layers*

(Pressman, 2010, P14)

Pondasi dari sebuah rekayasa piranti lunak adalah lapisan proses. Proses rekayasa piranti lunak adalah sebuah perekat yang

memegang lapisan teknologi dan menerapkan rasio dan waktu dari pembuatan *software*. Pada tahap ini dilakukan pengaturan kontrol *project* dan menetapkan metode teknikal apa yang akan diterapkan para proses pengembangan suatu *software*.

Metode pengembangan piranti lunak menentukan bagaimana cara membangun sebuah *software*. Metode pembuatan piranti lunak mencangkup beberapa kegiatan yaitu *communication, requirement analysis, design modeling, program construction, testing, support.*

Tools bagi rekayasa piranti lunak dibutuhkan untuk mendukung secara otomatis maupun semi-otomatis proses dan metode rekayasa piranti lunak yang dilakukan.

3. Proses Pembuatan Piranti Lunak

Sebuah kerangka proses akan menentukan pondasi dari seluruh proses rekayasa piranti lunak. Kerangka proses yang umumnya digunakan dalam rekayasa piranti lunak adalah

a. *Communication*

Sebelum memulai pekerjaan membangun sebuah piranti lunak, penting untuk melakukan komunikasidan berkolaborasi dengan pengguna dan orang-orang yang berkaitan dengan piranti lunak ini untuk menumpulkan

kebutuhan, pengambilan informasi yang diperlukan, menentukan tujuan dari *project* yang akan dilakukan yang semuanya itu akan berguna untuk menentukan fitur dan fungsi dari piranti lunak yang akan dibuat.

b. *Planning*

Sebuah project rekayasa piranti lunak sangat membutuhkan perencanaan yang akan sangat berguna untuk membantu tim yang mengerjakan *project* ini untuk mencapai target. Perencanaan dalam tahapan rekayasa piranti lunak mencangkup bagaimana menjelaskan tugas-tugas teknik yang perlu dilakukan, resiko yang mungkin dihadapi, sumber daya yang akan diperlukan, hal-hal yang akan dihasilkan, dan juga jadwal pekerjaan yang akan dilakukan

c. *Modelling*

Pada tahap ini, dilakukan proses pemodelan piranti lunak. Dengan melakukan pemodelan, seorang bisa mengerti bagaimana piranti lunak dilihat secara arsitektur, secara gambaran besar, bagaimana *detail* dari piranti lunak tersebut.

Pemodelan piranti lunak juga dilakukan untuk lebih memahami kebutuhan piranti lunak dan desain yang bertujuan untuk memenuhi kebutuhan tersebut. Dengan melakukan pemodelan, seorang bisa mengerti bagaimana piranti lunak dilihat secara arsitektur, secara gambaran besar, bagaimana *detail* dari piranti lunak tersebut.

d. *Construction*

Pada tahapan ini dilakukan penulisan kode program dan dilakukan pengujian terhadap kode program yang telah dibuat untuk menghindari terjadinya kesalahan (*error*) pada perangakat lunak yang dihasilkan.

e. *Deployment*

Setelah semua tahapan sebelumnya telah selesai, maka piranti lunak yang dihasilkan dari semua proses di atas telah selesai dan diberikan kepada pengguna aplikasi untuk kemudian dilakukan evaluasi dan juga memungkinkan untuk pengguna memberikan masukan-masukan atau saran-saran (*feedback*).

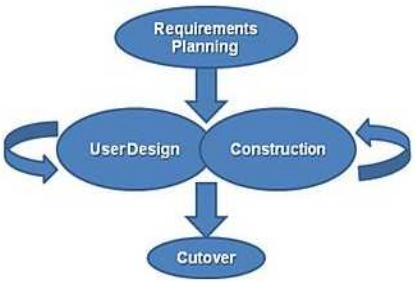
2.1.3 *Rapid Application Development* (RAD)

Menurut Shelly (2012, p145) *Rapid application development* (RAD) adalah sebuah teknik berdasarkan tim yang dapat menambah

kecepatan dalam membangun dan membangun sebuah fungsi sistem informasi. RAD menggunakan pendekatan kelompok dan berakhir dengan menghasilkan sebuah sistem informasi baru. RAD mengadopsi metodologi SDLC yang lengkap dengan 4 fase alur hidup. Metodologi RAD digunakan untuk mengurangi biaya dan waku pengerjaan aplikasi serta menambah peluang keberhasilan dari sebuah projek.

RAD sangat bergantung pada *prototyping* dan keterlibatan pengguna. RAD proses memungkinkan pengguna untuk memeriksa model kerja secepat mungkin, menentukan apakah sudah memenuhi kebutuhan pengguna dan menyarankan perubahan yang perlu. Berdasarkan dari masukan pengguna, *prototype* diubah dan proses interaksi berlanjut sampai seluruh sistem selesai dibangun dan pengguna sudah merasa puas.

RAD model mengandung 4 tahapan yaitu *requirements planning, user design, construction,* dan *cutover*



Gambar 2.3 Model RAD

1. *Requirements Planning*

Fase ini mengkombinasikan elemen elemen dari fase perencanaan sistem dan analisis sistem dari SDLC. *user, manager*, IT *staff member* membicarakan dan menyepakati kebutuhan bisnis, ruang lingkup project, batasan-batasannya, dan kebutuhan sistem. Fase ini berakhir pada saat seluruh team setuju dengan isu-isu kunci yang akan dilanjutkan ke tahapan selanjutnya.

2. *User Design*

Dalam fase ini, pengguna berinteraksi dengan system analis dan membuat model dan *prototype* yang merepresentasikan seluruh proses yang terjadi dalam sistem, *input*, dan *output*.

3. *Construction*

Fase konstruksi berfokus pada pembuatan program dan aplikasi. Pada metode RAD, pengguna terus dilibatkan dan terus dapat memberikan saran untuk perubahan atau perbankan.

4. *Cut Over*

Pada fase ini adalah tahapan akhir model RAD dimana dilakukan konversi data, pengujian program, perubahan sistem, dan pelatihan pengguna.

Tujuan utama dari pendekatan RAD adalah memangkas waktu pembuatan aplikasi dengan cara melibatkan pengguna lebih lagi di setiap fase pembuatan aplikasi. Karena berjalan secara terus menerus, RAD memungkinkan pengembang aplikasi untuk dapat membuat perubahan yang dirasa perlu secara cepat.

2.1.4 *Unified Modeling Language* (UML)

Menurut Whitten dan Bentley (2010, p371), *Unified Modeling Language* (UML) adalah sebuah kaidah pemodelan yang digunakan untuk menentukan dan menjelaskan sebuah sistem piranti lunak yang terkait dengan objek.

Menurut Connolly dan Begg (2010, p372), *Unified Modeling Language* (UML) adalah sebuah cara merepresetasikan *object oriented analysis* dan desain ke dalam sebuah model, UML secara grafis memvisualisasikan, menentukan, membangun, dan mendokumentasikan model sistem perangkat lunak. UML adalah sebuah standar untuk menulis cetak biru sistem, meliputi hal-hal konseptual, seperti proses bisnis dan fungsi sistem, serta hal-hal konkret, seperti kelas yang ditulis dalam bahasa pemrograman tertentu, skema *database*, dan komponen perangkat lunak yang dapat digunakan kembali.

Secara garis besar dapat dikatakan bahwa *Unified Modeling Language* (UML) adalah notasi yang lengkap untuk membuat visualisasi model dari suatu sistem. Sistem berisi informasi dan fungsi, tetapi secara

normal digunakan untuk memodelkan sistem. Beberapa contoh diagram yang dapat digunakan untuk memodelkan sistem dengan UML adalah:

1. *Use Case Diagram*

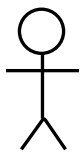
   Menurut Whitten dan Bentley (2010, p382), *use case diagram* adalah diagram yang menggambarkan interaksi antara sistem, eksternal sistem dan pengguna. Dengan kata lain *use case diagram* menggambarkan siapa yang akan menggunakan sistem dan bagaimana pengguna dapat berinteraksi dengan sistem.

   Use case narrative dapat digunakan sebagai tambahan untuk menjelaskan urutan langkah-langkah setiap interaksi yang terjadi antara pengguna dan sistem.

   Use case juga dapat menjelaskan fungsi sistem dari sudut pandang pengguna luar dengan cara yang mudah untuk dimengerti.

   Komponen-komponen yang digunakan dalam membuat *Use Case* Diagram :

   a. *Actor*

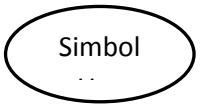


Gambar 2.4 Komponen *Actor*

Menurut Whitten dan Bentley (2010, p247), *Actor* menggambarkan semua hal yang perlu berinteraksi dengan sistem untuk bertukar informasi. *Actor* menggambarkan pengguna software aplikasi (*user*). *Actor* tidak selamanya harus manusia tetapi juga bisa berupa sistem informasi lain, organisasi, atau perangkat luar seperti sensor. Jika sebuah sistem melakukan proses terhadap sistem lain, maka sistem tersebut juga dapat disebut *actor*.

Sebuah aktor akan memulai aktifitas sistem yang bertujuan untuk melakukan sebuah proses bisnis yang akan menghasilkan *output*.

b. *Use Case*



Gambar 2.5 Komponen *Use Case*

Menurut Whitten dan Bentley (2010, p246), *use case* adalah sebuah perilaku yang berkaitan dengan urutan langkah-langkah atau skenario yang bertujuan untuk menyelesaikan suatu tugas bisnis.

Use-case menggambarkan perilaku *software* aplikasi, termasuk antara *actor* dengan *software* aplikasi tersebut.

c. Hubungan (*Relationship*)

Menurut Whitten dan Bentley (2010, p248), Hubungan dalam *use case diagram* digambarkan dengan sebuah garis diantara 2 simbol dalam *use case diagram*. Arti sebuah hubungan bisa berbeda tergantung dari bagaimana garis digambarkan dan simbol apa yang dihubungkan dengan garis tersebut.

Ada beberapa macam hubungan yang ada dalam pembuatan *use case diagram*. Hubungan tersebut yaitu:

d. *Associations*

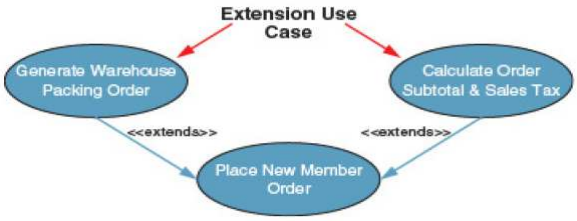
Hubungan *associatons* adalah hubungan antara aktor dan *use-case* yang mana terjadi interaksi diantara mereka.

Gambar 2.6 Hubungan *Association*

(Whitten dan Bentley, 2010, P248)

e. *Extends*

Extends adalah sebuah *use case* yang terdiri dari beberapa langkah yang diekstrak dari sebuah *use case* yang kompleks.

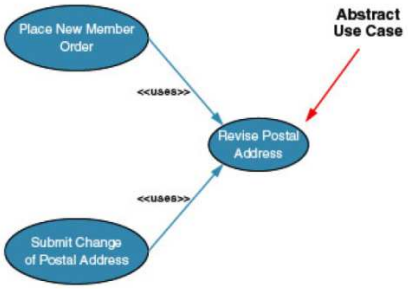


Gambar 2.7 Hubungan *Extends*

(Whitten dan Bentley, 2010, P249)

f. *Uses atau Include*

Uses atau Include digunakan untuk menghubungkan *use case* dengan *use case* abstrak.

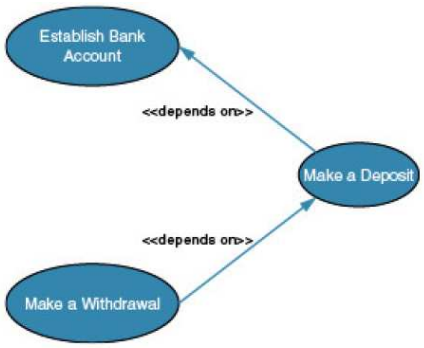


Gambar 2.8 Hubungan Uses

(Whitten dan Bentley, 2010, P249)

g. *Depends On*

Depends On adalah sebuah hubungan antara *use case* yang menyatakan bahwa sebuah *use case* tidak dapat dijalankan sebelum *use case* tertentu selesai dijalankan.

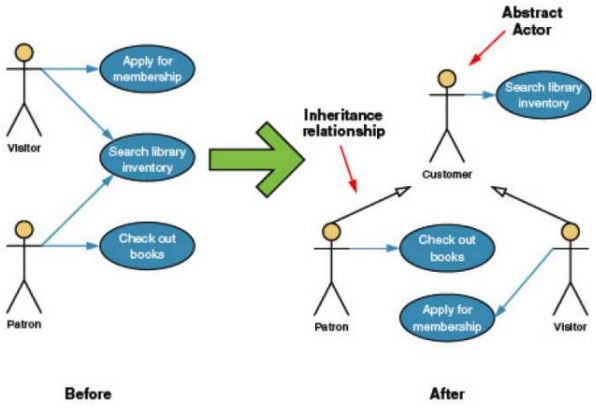


Gambar 2.9 Hubungan *Depends On*

(Whitten dan Bentley, 2010, P250)

h. *Inheritance*

Inheritance adalah hubungan yang terjadi antara aktor dan aktor abstrak. Hal ini dapat terjadi untuk mempersingkat penggambaran ketika sebuah aktor abstrak *inherit* peran dari beberapa aktor yang *real.*



Gambar 2.10 Hubungan *Inheritance*

(Whitten dan Bentley, 2010, P250)

2. *Use Case Narrative*

Menurut Whitten dan Bentley (2010, p246), *use case narrative* adalah sebuah deskripsi tekstual dari kegiatan bisnis dan bagaimana pengguna berinteraksi dengan sistem untuk menyelesaikan suatu tugas.

***USE CASE* NAME :**	*Log In*	
ACTOR(S) :	*Customer*	
DESCRIPTION :	*Use case* ini menjelaskan tahapan saat aktor yang sudah terdaftar sebagai anggota *customer*, ingin melakukan proses *log in*.	
TRIGGER :	Aktor membuka dan mengakses aplikasi, dan memutuskan untuk melakukan proses *log in* .	
PRE-CONDITION :	*Registration*	
TYPICAL COURSE OF EVENTS :	**Actor Action**	**System Response**
	Tahap 1: Aktor membuka aplikasi.	Tahap 2: Sistem menampilkan halaman *log in.*

	Tahap 3: Aktor mengisi alamat *e-mail* dan *password* dan kemudian menekan tombol *"Log In"*	Tahap 4: Sistem mem-validasi *e-mail* dan *password*
		Tahap 5: Sistem menampilkan halaman *home.*
ALTERNATIVE COURSE :	Alt – Tahap 3: - jika aktor belum memiliki *email* dan *password*, maka aktor harus melakukan registrasi terlebih dahulu - Jika aktor lupa *password*, aktor dapat menekan link *forget password*,sistem akan memberikan pertanyaan rahasia dan kemudian memberitahukan *password.* Alt – Tahap 4: jika aktor memasukan *e-mail* dan *password* yang salah, maka sistem akan menampilkan pesan	

	kesalahan.
POST-CONDITION :	Aktor berhasil melakukan *log in*
ASSUMPTION :	- Aktor sudah melakukan registrasi sebelumnya - Data yang dimasukan oleh aktor benar.

Tabel 2.1 Contoh Use Case Narrative

Use case narrative menggambarkan *event*, yang mencakup hal-hal sebagai berikut:

a. *Use case name*

Use case name harus mewakili tujuan yang ingin dicapai oleh *use case* tersebut. Nama *use case* harus diwakili oleh kata kerja.

b. *Actor(s)*

Actor merupakan *stakeholder* yang memiliki manfaat utama dari terlaksananya *use case*, dengan menerima sesuatu yang bernilai.

c. *Trigger*

Trigger merupakan *event* yang memprakarsai pelaksanaan *use case*. Biasanya berupa tindakan fisik, seperti aktor membuka dan mengakses aplikasi. Waktu juga bisa memacu kasus *trigger*.

d. *Pre-Condition*

Pre-condition merupakan kendala atau *event* yang terjadi pada keadaan sistem sebelum *use case* dieksekusi, biasanya ini mengacu pada *use case* lain yang harus dijalankan sebelumnya.

e. *Typical Course of Events*

Typical course of events merupakan urutan kegiatan yang harus dilakukan oleh aktor dan sistem dalam rangka untuk memenuhi tujuan dari *use case*.

f. *Alternative Course*

Alternative course mendokumentasikan perilaku dari *use case* ketika terdapat pengecualian yang memerlukan langkah-langkah tambahan diluar lingkup *typical course.*

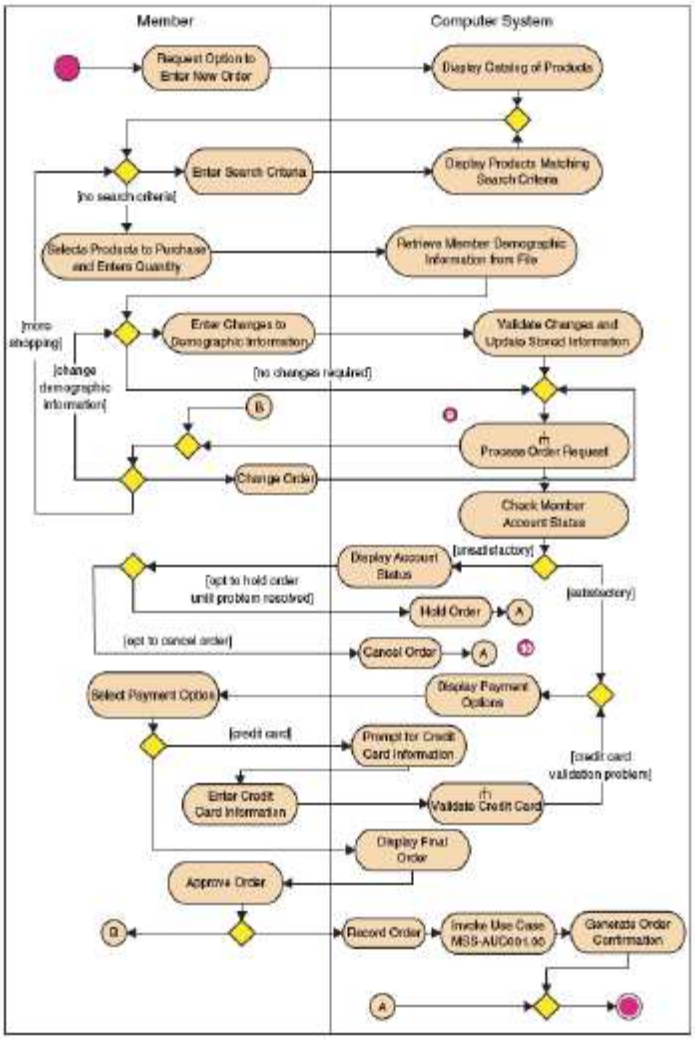
g. *Post-Condition*

Post-Condition adalah kendala atau hal *event* yang terjadi pada keadaan sistem setelah *use case* berhasil dieksekusi.

h. *Assumption*

Assumption merupakan setiap asumsi yang yang dibuat oleh penciptanya ketika mendokumentasikan *use case*.

3. *Activity Diagram*



Gambar 2.11 *Activity Diagram*

(Whitten dan Bentley, 2010, P393)

Menurut Whitten dan Bentley (2010, p390) *Activity Diagram* adalah sebuah diagram yang dapat digunakan untuk dapat

menggambarkan secara grafis alur dari proses, langkah-langkah yang dilakukan pada *use-case diagram,* serta logika dari perilaku objek.

Activity diagram menggambarkan berbagai aliran aktifitas dalam sebuah sistem yang sedang dirancang, bagaimana alur-alur yang terjadi masing-masing berawal, hal-hal yang mungkin terjadi dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Notasi yang banyak digunakan dalam *Activity Diagram*

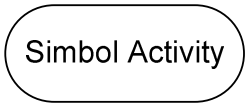
a. *Initial node* yang menggambarkan dimulainya suatu proses atau aktivitas.

Gambar 2.12 Notasi *Initial Node*

b. *Activity Final* menunjukkan tempat berakhirnya diagram. *Activity Diagram* dapat memiliki satu atau lebih akhir.

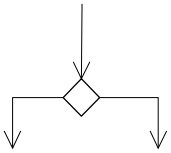
Gambar 2.13 Notasi *Activity Final*

c. *Action* yang menggambarkan setiap satuan aktivitas/proses yang dilakukan.



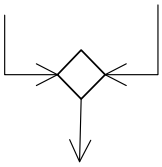
Gambar 2.14 Notasi *Action*

d. *Decission* yang menggambarkan kondisi if-else dengan lambang *diamond* yang menerima 1 input dan 2 atau lebih output.



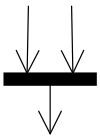
Gambar 2.15 Notasi *Decission*

e. *Merge* digambarkan dengan bentuk *diamond* dengan 2 atau lebih input yang menghasilkan 1 luaran.



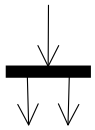
Gambar 2.16 Notasi *Merge*

f. *Join* digambarkan dengan *bar* hitam dengan dua atau lebih masukan dan dengan satu luaran.



Gambar 2.17 Notasi *Join*

g. *Fork* digambarkan dengan *bar* hitam dengan dua atau lebih luaran dan dengan satu masukan.



Gambar 2.18 Notasi *Fork*

4. *Class Diagram*

Menurut Whitten dan Bentley (2007, p382), *class diagram* menggambarkan struktur dari sistem. Serta menampilkan *class object* yang berada di dalam sistem serta hubungan antara objek tersebut dengan objek lainnya.

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan berorientasi objek. *Class* menggambarkan keadaan (*attribute* / *property*) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metode / fungsi).

Tiga Area Pokok yang dimiliki oleh suatu *class*:

a. Nama

b. Atribut

c. *Method*

Atribut dan *method* dapat memiliki salah satu sifat berikut :

a. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan.

b. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.

c. *Public*, dapat dipanggil oleh siapa saja.

Class diagram menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *association*,*compostion*,*dependency*, dan *agrregation*.

a. *Association*

Sebuah asosiasi merupakan sebuah struktur *relationship* antara 2 *class* dan dilambangkan oleh sebuah garis yang menghubungkan antara 2 *class*. Garis ini bisa melambangkan tipe-tipe relasi dan juga dapat menampilkan hukum-hukum multiplisitas pada sebuah *relationship* (contoh: *one-to-one*, *one-to-many*, *many-to-many*).

Gambar 2.19 *Association*

b. *Composition*

Jika sebuah class tidak bisa berdiri sendiri dan harus merupakan bagian dari *class* yang lain, maka *class* tersebut memiliki relasi *composition* terhadap *class* tempat dia bergantung tersebut. Sebuah *relationship composition* digambarkan sebagai garis dengan ujung berbentuk jajaran genjang berisi / *solid*.

Gambar 2.20 *Composition*

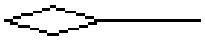
c. *Dependency*

Kadangkala sebuah *class* menggunakan *class* yang lain. Hal ini disebut *dependency*. Umumnya penggunaan *dependency* digunakan untuk menunjukkan operasi pada suatu *class* yang menggunakan *class* yang lain. Sebuah *dependency* dilambangkan sebagai sebuah panah bertitik-titik.

Gambar 2.21 *Dependency*

d. *Aggregation*

Aggregation mengindikasikan keseluruhan bagian *relationship* dan biasanya disebut sebagai relasi.



Gambar 2.22 *Aggregation*

2.2 Teori Khusus

2.2.1 Smartphone

Menurut Fling (2009, p8), *smartphone* adalah perangkat telepon yang sudah memiliki kemampuan seperti *future phone* seperti melakukan panggilan telepon, mengirim SMS, mengambil gambar atau video, dan mengakses mobile *web*. *Smartphone* secara khusus biasanya memiliki sistem operasi umum, QWERTY *keyboard* atau *stylus* sebagai perangkat masukan (*input*) dan *wi-fi* atau bermacam-macam konektifitas nirkabel.

2.2.2 Aplikasi *Mobile*

Menurut Fling (2009, p70) aplikasi *mobile* adalah sebuah tipe aplikasi dengan *mobile* teknologi yang dapat memberikan informasi kepada pengguna. Selain terdapat aplikasi *native*, juga terdapat mobile *website* yaitu aplikasi *website* yang di desain secara khusus untuk perangkat *mobile*. Karakteristik aplikasi *mobile* biasanya mempunyai arsitektur khusus dan presentasi yang simpel.

2.2.3 Android

1. Pengertian Android

Menurut. Darwin (2012, p. xii) Android adalah *platform* teknologi *mobile* yang mencangkup telepon selular, komputer *tablet* dan perangkat *mobile* lainnya dengan kekuatan dan portabilitas sistem operasi Linux serta keandalan dan portabilitas dari standar bahasa pemrograman tingkat tinggi dan API.

Android juga berkaitan erat dengan keluarga sistem operasi Openmoko, QT Embedded,MeeGo, Ophone, LiMo, dan *project* telpon seluar lain yang juga berbasis sistem operasi Linux. Secara bahasa pemrograman, Android berkaitan erat dengan Blackberry atau ponsel-ponsel Java ME, juga dengan Java dan aplikasi Java *Enterprise*

2. Sejarah Android

Menurut Gargenta (2011, p3) Sejarah android dimulai pada tahun 2005 ketika Google membeli Android, Inc. dimana pada saat itu banyak orang berpikir bahwa Google akan memasuki pasar *smartphone* dan pada saat itu juga rumor tentang perangkat dengan nama gPhone akan muncul. Beberapa kejadian penting dalam sejarah Android antara lain sebagai berikut:

a. Tahun 2005, Google membeli Android, Inc. banyak orang mulai memperbincangkan akan munculnya perangkat baru dengan nama gPhone. Namun setelah pembelian ini, isu ini justru semakin memudar dan keadaan mulai tenang.

b. Tahun 2007, *Open Headset Aliance* mengumumkan bahwa Android secara resmi bersifat *open source*.

c. Tahun 2008, Android SDK 1.0 keluar di pasar. Versi ini digunakan oleh ponsel G1 yang dikerjakan manufaktur HTC dan kemudian dijual oleh T-*Mobile*, USA.

d. Tahun 2009, setelah melihat berkembangnya perangkat berbasis Android, versi terbaru dari sistem operasi ini mulai bermunculan. Dimulai dari *Cupcake* (1.5), *Donut* (1.6) dan *Éclair* (2.0 dan 2.1). Setelah ini pengguna perangkat berbasis Android semakin bertambah banyak.

e. Tahun 2010, Android 2.2 (*Froyo*) diluncurkan ke pasar. Saat ini Android mulai disegani sebagi salah satu *platform* ponsel terbaik dan perangkat yang menggunakan android juga bertambah banyak.

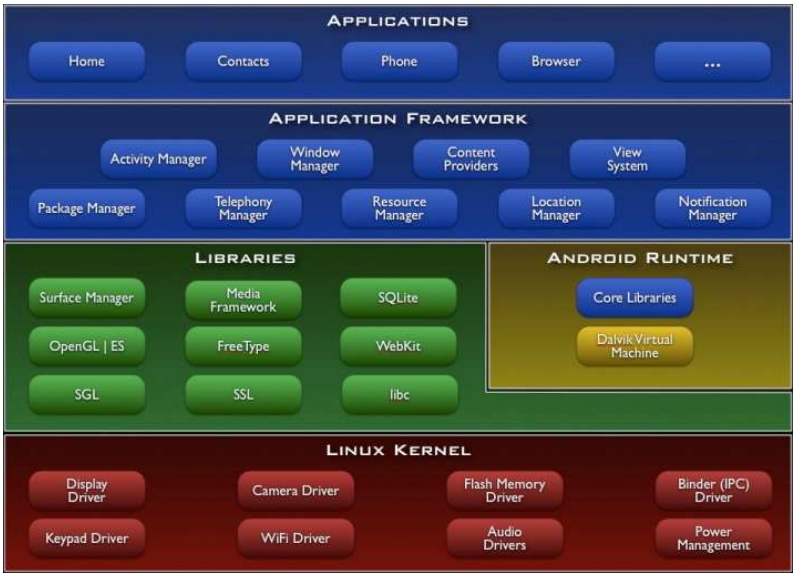
1. Pada tahun ini juga, Android versi 2.3 (*Gingerbread*) diluncurkan. Perubahan-perubahan umum yang didapat dari Android versi ini antara lain peningkatan kemampuan permainan (gaming), mendesain ulang tampilan layar antarmuka penguna, dukungan format video VP8 dan *Web*M, efek audio baru (*reverb*, *equalization*, *headphone virtualization*, dan *bass boost*), serta dukungan kemampuan *Near Field Communication* (NFC).

f. Tahun 2011, Android meluncurkan versi terbarunya yaitu Android versi 3.0 / 3.1 / 3.2 yang diberi nama *Honeycomb*. Menurut Michael J. Burton dan Felker (2012, p. 18) *Honeycomb* diperkenalkan kepada dunia sebagai sistem operasi untuk *tablet* Android. *Honeycomb* memiliki banyak perubahan untuk dapat menukung Tablet PC yang memiliki kelas perangkat yang berbeda dari perangkat *mobile phone* yang sebelumnya telah menggunakan Android. *Honeycomb* juga mendukung multi prosesor dan juga akselerasi perangkat keras (*hardware*) untuk grafis. Tablet pertama yang dibuat dengan menjalankan *Honeycomb* adalah Motorola Xoom.

1. Pada tahun yang sama, *Ice Cream Sandwich* (Android 4.0) juga diluncurkan dengan konsep untuk membawa teknologi yang ada pada *honeycomb* ke *mobile phone* dan dengan menambahkan fitur lainnya seperti *face unlock* untuk membuka kunci telepon dengan teknologi pengenal wajah.

3. Arsitektur Android

Menurut Duffy (2012, p73) Arsitektur Android terdiri dari 5 hal yaitu:



Gambar 2.23 Arsitektur Android

a. *Linux Kernel*

Komponen ini adalah sistem operasi. Sebagai mana sistem operasi pada umumnya, komponen ini akan bertindak sebagai sistem operasi yang akan mengelola *memory*, *input* / *output* (*camera, keypad, display audio driver*) *management file*, jaringan serta proses dan sumber daya. Komponen ini bertindak sebagai lapisan komunikasi antara perangkat lunak dan perangkat keras.

b. *Libraries*

Lapisan ke 2 setelah Linux Kernel adalah Andorid *Native Libraries*. Komponen ini mengandung semua *libraries* Android yang telah dibuat sebelumnya dengan bahasa pemrograman C atau C++ dan telah dikompilasi

untuk beberapa arsitektur perangkat keras yang digunakan pada *mobile phone*. Dengan *libraries* inilah Android dapat berdiri sendiri. Menurut Burnette (2010, p16) beberapa *library* yang penting dan cukup sering digunakan antara lain:

- *Surface Manager*
- 2D *and* 3d *Graphics*
- *Media Codecs*
- SQL *database*
- *Browser Engine*

c. *Android Runtime*

Pada komponen ini, terdapat Dalvik *Virtual Machine* dan *Core Libraries,* Menurut Burnette (2010, p17) Dalvik *Virtual Machine* adalah mesin virtual khusus yang dirancang dan dibuat oleh Dan Bornstein di Google. Dalvik akan menjalankan kode aplikasi android yang dibuat dalam bahasa pemrograman Java yang akan dikompilasi menjadi perintah mesin yang disebut *bytecodes*. Dalvik khusus digunakan untuk perangkat yang memiliki memori kecil (perangkat *moblie*). Dalvik memungkinkan lebih dari satu mesin virtual jalan

bersamaan dan mengambil keuntungan dari sistem operasi, dan karena alasan tersebut maka Dalvik mampu untuk meningkatkan performa aplikasi saat *runtime*.

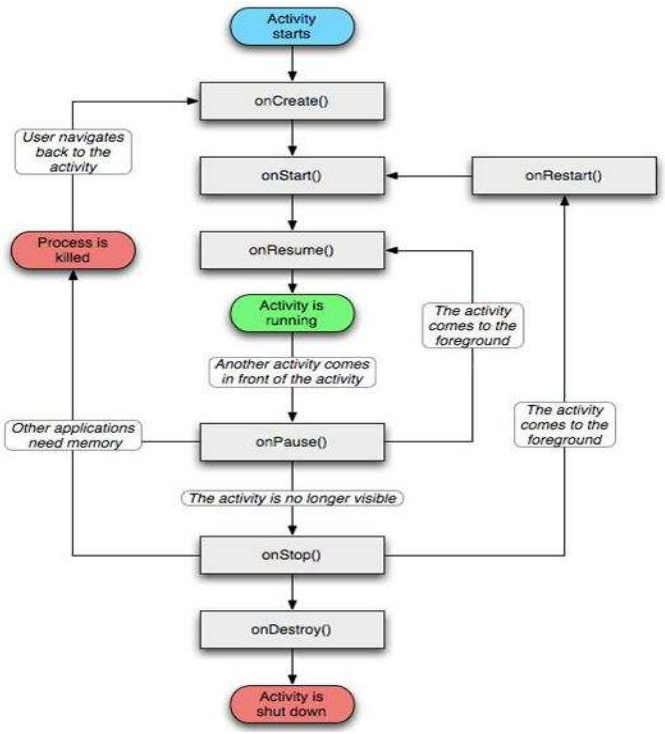
d. *Application Framework*

Komponen selanjutnya setelah *Android Runtime* dan *Libraries* adalah *Application Framework*. Komponen ini digunakan untuk menuliskan kode dan membuat aplikasi dengan menggunakan *library* yang ada dan *library* yang anda buat sendriri.

Menurut Burnette (2010, p16) beberapa bagian yang paling penting penting dari *framework* ini antara lain:

- *Activity Manager*

  Bagian ini akan melakukan kontrol alur hidup dari aplikasi dan memelihara "*back stack*" dari navigasi yang dilakukan pengguna.



Gambar 2.24 *Activity Manager*

Menurut Priyatna (2011, p46) alur hidup sebuah aplikasi Andoid terdiri dari beberapa *method* yaitu:

- *onCreate*()

  Method onCreate() akan dijalankan ketika sebuah *activity* pertama kali dipanggil.

- *onStart*()

  Method onStart() akan dijalankan ketika tampilan antar muka pengguna sudah tampil berbeda dengan *method onCreate* yang

dijalankan pada saat sebuah *activity* dipanggil.

- *onRestart*()

  Method onRestart() adalah sebuah *method* yang dijalankan jika sebuah *activity* kembali dijalankan setelah sebelumnya menjalankan *method onPause*()

- *onResume*()

  Method ini dipanggil ketika *activity* dapat mulai kembali berinteraksi dengan user.

- *onPause*()

  Method *onPause*() akan dijalankan jika aplikasi berjalan di belakang layar atau aplikasi membuka aplikasi lain yang akan menjalankan *activity* lain.

- *onStop*()

  Method onStop() akan dijalankan jika activity sudah tidak dijalankan atau sudah tidak perlu dijalankan.

- *onDestroy*()

  Method onDestroy() akan dieksekusi pada saat *activity* dihancurkan atau pada saat perangkat pengguna aplikasi kehabisan RAM untuk menjalankan aplikasi yang akan menghancurkan *activity* secara otomatis.

o *Content Provider*

Objek ini akan menyatukan data yang perlu dan dapat digunakan antar aplikasi.

o *Resource Manager*

Bagian ini akan mengatur sumber daya (*resource*) yang dibutuhkan dan dijalankan bersama program kita.

o *Location Manager*

Ponsel android selalu dapat mengetahui lokasi dimana saat ini ponsel itu berada dengan menggukan GPS, atau dengan mendeteksi menara pemancar sinyal telepon terdekat, dan jika terkoneksi dengan *wifi hotspot* itu juga dapat digunakan.

- *Notification Manager*

  Bagian ini mengatur pemberitahuan yang akan diinformasikan kepada pengguna. Contohnya pada saat terjadi sebuat *event* seperti pesan masuk.

e. *Appilication*

Layer paling atas dalam arsitektur android adalah *Application Layer*. Komponen inilah yang langsung berhubungan dengan pengguna. Pada komponen ini terdapat aplikasi umum yang langsung dipasang oleh android seperti *email, contacs, web browser, Android market,* dan juga tentunya aplikasi buatan sendiri atau buatan orang lain yang dipasang oleh pengguna.

4. Fitur Android

Menurut DiMarzio (2008, p6), beberapa fitur-fitur yang terdapat pada Android antara lain yaitu terdapat *accelerated* 3-D *graphics engine* (bergantung pada dukungan *hardware* yang mengadopsi Android), dukungan *database* yang menggunakan SQLite, dan *web browser* yang terintegrasi.

Salah satu fitur yang menarik dari Android adalah bahwa aplikasi pihak ketiga, sekalipun sederhana dijalankan dengan prioritas yang sama dengan aplikasi yang telah terpasang pada sistem inti Android.

Hal ini merupakan sebuah kemajuan bila dibandingkan dengan sistem lain pada umumnya. Selain itu setiap aplikasi juga dijalankan pada *thread* sendiri dengan menggunakan mesin virtual yang ringan

Di atas semua fitur yang tersedia dari sisi Android, fitur yang tak kalah menarik yang ditawarkan oleh Google adalah pengembang aplikasi Android akan mampu menyertakan aplikasi Google yang sudah teruji seperti Google *Maps* atau Google *Search*.

5. Application *Building Blocks*

Untuk membangun sebuah aplikasi Android tentunya diperlukan objek-objek untuk membangunnya menjadi sebuah aplikasi siap pakai. Menurut Burnette (2010, p.23) objek yang paling penting untuk diketahui pada saat akan membangun sebuah aplikasi Android adalah:

a. *Activities*

Sebuah *Activities* adalah sebuah tampilan layar antarmuka pengguna. Setiap aplikasi bisa memiliki satu atau lebih *Activities* untuk menangani beberapa bagian program.

b. *Intents*

Sebuah *Intents* dapat diartikan sebagai sebuah cara untuk melakukan suatu tindakan. Sebagai contoh: "*Send Email*" atau "*Call Home*." Jika sebuah aplikasi ingin mengirim surat

elektronik maka aplikasi akan meminta intents “Send Email” untuk dijalankan.

c. *Services*

Services adalah sebuah tugas yang berjalan pada *background* sistem Android tanpa interaksi langsung dengan pengguna aplikasi. Sebagai contoh perubahan layar dari *vertikal* menjadi *horizontal*. *Services* yang berjalan tidak memerlukan pengguna untuk mengubah orientasi tampilan layar pada saat posisi layar pengguna berubah dari *horizontal* ke *vertical* tetapi *services* yang berjalan langsung dapat mendeteksi perubahan tersebut dan langsung dapat mengubah orientasi tampilan layar pengguna.

d. *Content Providers*

Komponen yang memungkinkan kejadian saling membagi data antar proses atau aplikasi. Ini adalah cara untuk aplikasi dapat saling berinteraksi dengan aplikasi lainnya

2.2.4 PHP

Menurut Welling dan Thomson (2009, p3), PHP adalah sebuah *server side scripting language* yang secara khusus didesain untuk pemrograman *web*. Di dalam halaman HTML dapat disisipkan kode PHP yang akan dieksekusi setiap kali halaman tersebut diakses. Kode PHP

akan diinterpretasikan dalam *web server* dan akan menghasilkan kode HTML atau luaran lain yang akan dilihat pengguna pada halaman *web*.

PHP adalah sebuah *open source project*, yang artinya pengguna PHP dapat memiliki akses ke *source code* dan dapat menggunakannnya, mengubah dan mendistribusikannya tanpa biaya.

Menurut Welling (2009, p4), beberapa keunggulan dari PHP yaitu:

a. Performa yang cepat

b. PHP dapat terkoneksi dengan berbagai macam *database* antara lain MySQL, PostgreSQL, Oracle, dbm, FilePro, DB2, Hyperwave, Informix.

c. PHP menyediakan banyak *built-in libraries* yang berguna untuk melakukan hal-hal yang berkaitan dengan *web*. Sebagai contoh mengirim *e-mail*, bekerja dengan *cookies*, melakukan koneksi dengan *web service*.

d. PHP tidak bayar atau gratis.

e. *Syntax* PHP berbasis pada bahasa pemrograman C dan *perl* atau C-*like language* seperti C++ dan Java.

f. PHP tersedia pada banyak sistem operasi. Kode PHP umumnya dapat bekerja tanpa modifikasi di setiap sistem berbeda yang menjalankan PHP

g. PHP terutama versi 5 sudah didesain dengan fitur *object oriented*. Fitur ini beberapa sudah tersedia pada versi 3 dan 4 namun *support* untuk *object oriented* pada versi 5 lebih lengkap.

h. PHP memungkinkan untuk mengimplementasi tugas secara sederhana ataupun dengan *framework* seperti *Model-View-Controller* (*MVC*)

i. PHP merupakan *open source* sehingga pengembang aplikasi yang menggunakan PHP dapat mengubah atau menambahakan sessuatu kedalam bahasa PHP itu sendiri tanpa harus menunggu *release patch* dari manufaktur.

j. Dokumentasi PHP dan komunitas PHP sudah banyak tersebar dan kaya akan informasi. Begitu pula dengan *Zend Technology* yang menyediakan *support* pada PHP.

2.2.5 MySQL

Menurut Welling (2009, p3), MySQL adalah sebuah *Relational Database Management System (RDBMS)* yang cepat. Server MySQL mengendalikan akses ke data-data dalam *database* untuk meyakinkan beberapa pengguna dapat bekerja dengan data-data tersebut secara bersamaan, untuk menyediakan akses yang cepat ke data-data yang ada, dan meyakinkan hanya pengguna terotorisasi yang mendapatkan akses.

MySQL bersifat *multiuser* dan *multihread server*. MySQL menggunakan *Structure Query Language* (SQL) sebagai standar bahasa *query database.*

2.2.6 *Global Positioning System* (GPS)

Menurut McNamara (2008, p51), *Global Positioning System* (GPS) adalah sebuah radio khusus yang dapat mengukur jarak dari posisi pengguna ke satelit yang mengorbit bumi dan mengirimkan signal radio. GPS dapat secara tepat mendeteksi posisi pengguna di dunia.

GPS dapat bekerja karena terdiri dari komponen-komponen seperti satelit, *ground station* dan perangkat penerima (GPS *receiver*).

Menurut Deitel (2012, p4), GPS menggunakan jaringan satelit untuk mendapatkan informasi berbasis lokasi. Beberapa satelit mengirim signal ke perangkat GPS, yang menghitung jarak ke setiap satelit berdasarkan waktu pada saat signal dikirim oleh satelit dan pada saat signal tersebut diterima. Lokasi dari satelit dan jarak ke satelit digunakan untuk menentukan posisi dari perangkat GPS. Bedasarkan lokasi itu, perangkat GPS dapat menunjukan arah, membantu menemukan tempat terdekat, atau membantu mencari lokasi tertentu.

2.2.7 SQLite

Menurut Allen dan Owen (2010, p1), SQLite adalah sebuah *embeded relational database* yang bersifat *open source*. SQLite didesain untuk memberikan cara yang mudah untuk sebuah aplikasi dapat

mengatur data tanpa menimbulkan *overhead* seperti yang biasanya terjadi pada *dedicated relational database management system.* SQLite memiliki reputasi yang baik dalam portabilitas tinggi, mudah digunakan, *compact* dan handal.

SQLite adalah sebuah *embedded database*. Jika dibandingkan dengan *database* yang berjalan sendiri, *embedded database* berjalan secara berdampingan dalam aplikasi sehingga terlihat seolah-olah program tidak memiliki RDBMS yang berjalan. Salah satu keuntungan dari memiliki *database server* dalam aplikasi adalah tidak lagi diperlukan pengaturan jaringan atau *admini*strasi karena baik *server* dan *client* berjalan bersamaan dalam proses yang sama. Hal ini dapat mempermudah pengguna dalam proses *deploy* aplikasi.

2.2.8 Google Map API

Menurut Svennerberg (2010, p2), Google Map yang diperkenalkan pada Februari 2005 telah merevolusi bagaimana peta digital dalam *web* bekerja dengan memungkinkan pengguna untuk melakukan navigasi terhadap peta dengan cara menarik-narik peta itu seolah-olah bukan seperti peta *digital*.

Google Map API adalah sebuah API (*Aplication Programming Interface*) yang membantu dalam membangun piranti lunak yang berkaitan dengan Google Map. Google Map API merupakan API yang terpopuler digunakan di *internet* dengan sebanyak 43% *Mashup* (Aplikasi

web yang menggabungkan data atau fungsionalitas dari 2 sumber atau lebih) menggunakan Google Map API.

2.2.9 *E-Advertising*

Menurut Hamidizadeh, Yazdani, Tabriz dan Latifi (2012, p 131) *E-advertising* merupakan bentuk promosi yang menggunakan *internet* untuk menyampaikan pesan *marketing* untuk menarik konsumen. Beberapa sistem elektronik telah mengubah dunia komunikasi dan memberikan peluang lebih besar untuk masyarakat berkomunikasi.

Sebuah perusahaan membutuhkan iklan untuk melaksanakan penjualannya dan mencapai tujuan keuntungan perusahaan itu. Perusahaan dapat menggunakan *e-advertising* sebagai sebuah sarana memasarkan bisnisnya, menyebarluaskan informasi, memantapkan *brand* perusahaan yang merupakan tujuan penting dari proses *advertising* itu sendiri.

2.2.10 *Location Based Service* (LBS)

Menurut Brimicombe dan Li (2009, p2), *Location-Based Services* (LBS) adalah suatu cara menyampaikan layanan data dan informasi dimana isi dari layanan tersebut disesuaikan pada lokasi terkini dari pengguna *mobile phone*. Ini adalah sebuah teknologi baru yang berkembang cepat dengan menggabungkan Sistem Informasi Geografi

(GIS), teknologi nirkabel, sistem penentuan lokasi dan interaksi manusia dan komputer pada perangkat *mobile*.

Menurut Zhou (2011, p212) dengan bantuan *Location Based Service* (LBS), penyedia layanan *mobile* dapat menyediakan informasi optimal dan layanan kepada pengguna berdasarkan lokasi mereka. Pada umumnya LBS menyediakan layanan seperti navigasi *mobile*, *location-based advertisements*, evakuasi darurat, dan layanan *check-in* seperti pada beberapa jejaring sosial *mobile*. LBS dapat memberikan nilai tersendiri pada pengguna dan mempengaruhi loyalitas pengguna.

Menurut Rashid, Coulton dan Edward (2008, p3) beberapa contoh aplikasi yang menerapkan layanan berbasis lokasi seperti pemetaan aplikasi dan alat untuk menemukan rute terpendek dan restoran terdekat. Aplikasi lain dapat menggunakan informasi lokasi untuk memberikan informasi atau iklan.