# BAB 2

# LANDASAN TEORI

## 2.1 *CodeIgniter*

Menurut Blanco & Upton (2009:7) CodeIgniter adalah *powerful open source* PHP *framework* yang mudah dikuasai, dibangun untuk PHP *programmers* yang membutuhkan *toolkit* sederhana dan baik untuk membuat *full-featured web applications*. CodeIgniter adalah MVC *framework* yang di *design* untuk mempermudah penggunanya.

## 2.2 *Hypertext Markup Language* (HTML)

Menurut Shelly, Woods, & Dorin (2008:8) HTML merupakan bahasa penulisan yang digunakan untuk membuat dokumen pada halaman *web*. HTML menggunakan sekumpulan instruksi khusus, yang dikenal dengan *tags* atau *markup* untuk mendefinisikan struktur dan susunan dari *web document* dan menetapkan apa yang akan ditampilkan pada *browser*.

## 2.3 *Javascript Object Notation* (JSON)

JSON adalah sebuah *format* data berbasis teks yang ringan dan dirancang agar mudah dibaca dan ditulis oleh manusia, ataupun oleh mesin. JSON merupakan *format* teks yang sepenuhnya *independent language* tetapi menggunakan konvensi yang akrab bagi *programmer* dari bahasa C, C++, C#, Java, JavaScript, Pearl, Phyton, dan lainnya yang membuat JSON menjadi bahasa pertukaran data yang ideal.

JSON dibangun diatas dua struktur:

1. *A collection of name / value pairs*.

   Dalam berbagai bahasa, hal ini direalisasikan sebagai sebuah *object, record, struct, dictionary, hash table, keyed list*, atau *assosiated array*.

2. *An ordered list of values*.

   Dalam kebanyakan bahasa, hal ini direalisasikan sebagai *array, vector, list*, atau *sequence.*

Struktur ini merupakan *universal data structure*. Hampir semua bahasa pemrograman terbaru mendukung JSON dalam satu *form* atau lainnya. *Format data* yang dipertukarkan dengan bahasa pemrograman juga didasarkan pada struktur ini. (Anonymous.n.d.*Introducing JSON.Retrieved:*11-10-2012 *from* http://www.json.org )

## 2.4 PHP

Menurut Valade (2004:9) PHP akronim dari *Hypertext Preprocessor* adalah *open source* yang banyak digunakan sebagai tujuan utama *scripting language*. Di desain untuk digunakan pada pengembangan *website*. PHP berawal dari *personal home page tools*, yang di kembangkan oleh Rasmus Lerdorf untuk membantu *user* dengan *web page tasks*. PHP dibuktikan sangat berguna dan popular serta secara bertahap berkembang untuk menjadi *full-featured language*.

## 2.5 *Software Engineering*

Menurut Lethbridge & Laganiere (2005:5) *Software Engineering* adalah proses memecahkan masalah konsumen dengan pengembangan sistematik dan evolusi, sistem perangkat lunak yang berkualitas tinggi membutuhkan biaya, waktu, meskipun terdapat

beberapa kendala namun membantu penerapan rekayasa sistem perangkat lunak apapun serta bidang studi tentang bagaimana cara efektif melakukan hal diatas.

*Software developer* berguna untuk mengembangkan produk perangkat lunak yang nantinya akan dijual ke pelanggan. Salah satu jenis perangkat lunak adalah *Generic Software*, yaitu:

- *Generic Software*

  *Generic Software* di desain untuk dijual bebas di pasaran untuk melakukan fungsi yang dibutuhkan banyak orang dengan tujuan yang umum. Persyaratannya ditentukan oleh riset pasar.Ada kecenderungan dalam dunia bisnis untuk menggunakan *generic software* karena harganya lebih murah dan dapat diandalkan. *Generic Software* kadang disebut juga dengan *shrink-wrapped software* karena sering kali dijual dengan kemasan plastik. Contohnya: *word processors, spreadsheets, compilers, web browsers, operating systems, computer game*, dan paket *accounting* bagi bisnis kecil.

*Software engineering* dilakukan dengan melalui serangkaian kegiatan yang disebut dengan *Software Process*, diantaranya:

1. Spesifikasi *Software*

   Fungsionalitas perangkat lunak dan batasan kemampuan operasinya harus di definisikan.

2. Pengembangan *Software*

   Perangkat lunak harus dibuat sesuai dengan kebutuhan pelanggan.

3. Validasi *Software*

   Perangkat lunak harus di validasi untuk menjamin bahwa perangkat lunak melakukan apa yang diinginkan pelanggan.

4. Evolusi *Software*

Perangkat lunak harus berkembang untuk memenuhi kebutuhan pelanggan yang berubah.

## 2.6 UML

Menurut Lethbridge & Laganiere (2004:169) *Unified Modelling Language* (UML) merupakan bahasa grafis standar untuk permodelan *software* berorientasi objek. UML dikembangkan di pertengahan 1990-an sebagai upaya kolaboratif antara James Rumbaugh, Grady Booch, dan Ivar Jacobson. Satu *model* berguna untuk menangkap set informasi mengenai sistem, sedangkan satu diagram hanya memberi satu sudut pandang suatu informasi. Jika kita menghapus sebuah elemen dari *diagram* maka tetap tersimpan dalam *model*, namun jika kita menghapus elemen dari *model* maka akan menghilang dari seluruh *diagram*. Sebuah *model* dapat menggiring *software engineers* untuk memiliki wawasan tentang sistem, mereka dapat menganalisis *model* (manual maupun menggunakan alat) untuk menemukan masalah yang lainnya. Diagram sederhana yang dihasilkan dari *model* juga dapat membantu komunikasi dengan klien dan pengguna.

### 2.6.1 *Use Case*

Menurut Ambler (2005:33) sebuah UML *use case diagram* menunjukkan hubungan antara aktor dan menggunakan kasus dalam sistem. *Use case* sering digunakan untuk:

- Memberikan gambaran dari semua atau bagian dari persyaratan penggunaan sistem atau organisasi, dalam bentuk *essential model* dan *business model.*

- Mengkomunikasikan lingkup pembangunan proyek.
- Model analisis bagi persyaratan penggunaan dalam bentuk sistem *use case mode.*

Sebuah *use case model* terdiri dari satu atau lebih *use case diagram* dan dokumentasi pendukung seperti spesifikasi *use case* dan definisi aktor. *Use case* harus dikembangkan berdasarkan sudut pandang *project stakeholders* dan bukan dari sudut pandang tehnik dari *developer*.

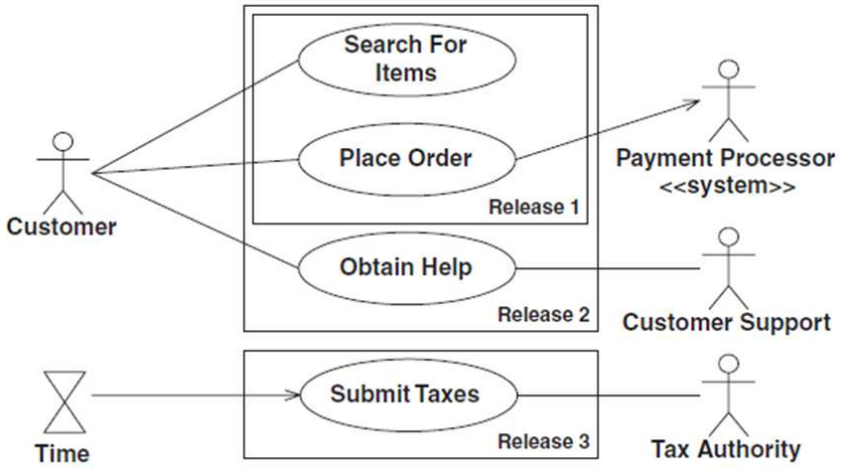
Ada beberapa jenis hubungan yang mungkin muncul pada use case diagram:

1. Asosiasi antara *actor* dan *use case*
2. Asosiasi  antara dua *use case*
3. Generalisasi antara dua aktor
4. Generalisasi antara dua *use case*

Asosiasi digambarkan sebagai garis yang menghubungkan dua unsur permodelan dengan *optional* kepala panah terbuka pada salah satu ujungnya yang menunjukkan arah ke permintaan hubungan. Generalisasi digambarkan sebagai kepala panah tertutup menuju ke unsur *model* yang umum.

Asosiasi *extend* yang digambarkan dengan <<*extend*>> adalah hubungan generalisasi dimana perluasan *use case* dilanjutkan dengan memasukkan urutan tindakan tambahan yang konseptual ke *use case* dasar, langkahnya bisa bekerja secara paralel ke langkah *use case* yang telah ada.

Asosiasi *include* yang digambarkan dengan <<*include*>> adalah hubungan permintaan *use case* dari *use case* lain seperti memanggil fungsi atau meminta operasi dalam *source code*. Hal ini sering digunakan untuk memperkenalkan *use case* yang merangkum logika umum yang dibutuhkan oleh beberapa *use case*, dan memiliki *use case* yang termasuk dalam *use case* yang membutuhkannya.




**Gambar 2.1** Contoh *Use Case* Pada *Online Shop*

Sumber: *The Elements of UML 2.0 (TM) 2.0 Style*, 2009

### 2.6.2 *Class Diagram*

Menurut Ambler (2005:47) UML *class diagram* menunjukkan kelas pada sistem, hubungannya, operasi serta atribut dari kelas, hal ini digunakan untuk:

- Mengeksplorasi konsep *domain* dalam bentuk *model domain.*
- Menganalisis persyaratan dalam bentuk analisis / *model* konseptual,
- Menggambarkan desain rinci berorientasi objek atau *object-based software*.

*Class model* terdiri dari satu atau lebih *class diagram* dan spesifikasi yang mendukung untuk mendeskripsikan *element model* termasuk *class*, hubungan antar *class* dan *interfaces*.

*Class* memiliki tiga area pokok, yaitu:

1. Nama (dan *stereotype*), merupakan nama sebuah *class*.
2. Atribut, merupakan properti dari sebuah *class* yang melambangkan batas nilai yang mungkin ada di objek dari *class*. Nama dan tipe atribut harus konsisten.
3. Metode, merupakan sesuatu yang bisa dilakukan oleh *class* atau dapat dilakukan *class* lain terhadap sebuah *class*.

Atribut dan metode dapat memiliki salah satu sifat sebagai berikut:

**Table 3.** Visibility Options on UML Class Diagrams

| Visibility | Symbol | Accessible to |
|---|---|---|
| Public | + | All objects within your system |
| Protected | # | Instances of the implementing class and its subclasses |
| Private | - | Instances of the implementing class |
| Package | ~ | Instances of classes within the same package |

**Gambar 2.2** *Visibility Options On* UML *Class Diagrams*

Sumber: *The Elements of UML 2.0 (TM) 2.0 Style*, 2009

a. *Public*, dapat dipanggil oleh semua objek di dalam sistem, ditampilkan dengan simbol (+).

b. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan *subclass*, ditampilkan dengan simbol (#).

c. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan, ditampilkan dengan simbol (-).

d. *Package*, hanya dapat dipanggil oleh *classes* dengan *package* yang sama, ditampilkan dengan simbol (~).

Berikut ini merupakan notasi dari *class diagram*:

1. *Class*

   Merupakan sebuah *template* dari objek mana yang dibuat. *Class* menentukan atribut yang bersangkutan dengan operasinya.

2. *Association*

   Merupakan hubungan antara dua *class*. Garis ini dapat melambangkan tipe hubungan dan juga hukum *multiplicity* seperti gambar dibawah ini.

**Table 6.** UML Multiplicity Indicators

| Indicator | Meaning |
|---|---|
| 0..1 | Zero or one |
| 1 | One only |
| 0..* | Zero or more |
| 1..* | One or more |
| *n* | Only *n* (where *n* > 1) |
| * | Many |
| 0..*n* | Zero to *n* (where n > 1) |
| 1..*n* | One to *n* (where *n* > 1) |
| *n*..*m* | Where *n* and *m* both > 1 |
| *n*..* | *n* or more, where *n* > 1 |

**Gambar 2.3** UML *Multiplicity Indicators*

Sumber: *The Elements of UML 2.0 (TM) 2.0 Style*, 2009

3. *Composition*

   *Composition* adalah bentuk yang lebih kuat dari agregasi dimana keseluruhan bagian memiliki *coincident lifetimes* dan sangat umum bagi keseluruhannya untuk mengelola *life cycle* dari bagian-bagiannya itu sendiri.

4. *Dependency*

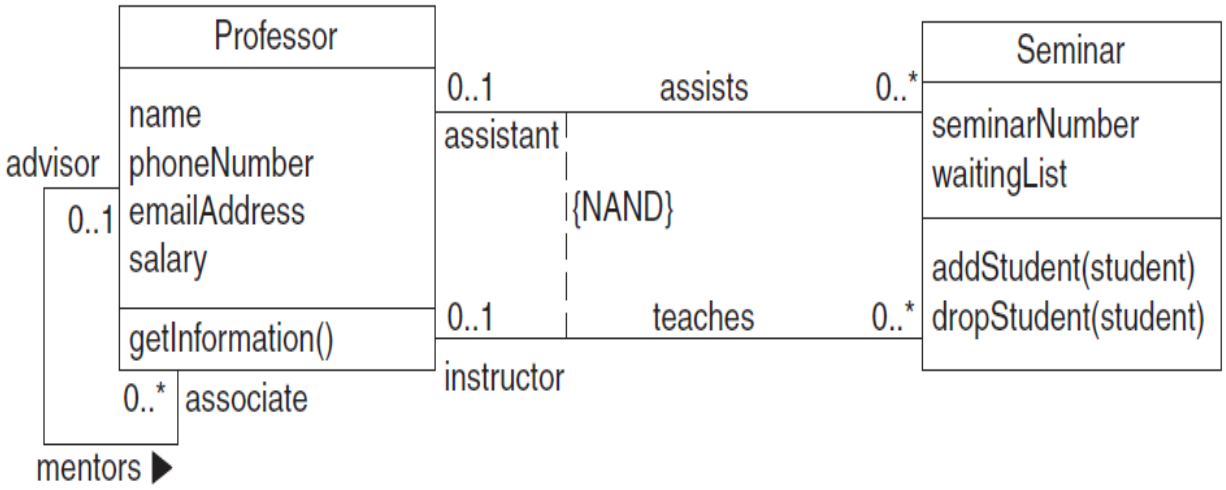
   Hubungan *Dependency* digunakan jika operasi sebuah *class* menggunakan *class* yang lain. *Dependency* digambarkan dengan sebuah panah bertitik-titik.

5. *Aggregation*

   Agregasi menyatakan “bagian dari” hubungan. Agregasi merupakan spesialisasi dari asosiasi, karena menspesialisasi seluruh bagian hubungan diantara dua objek.

6. *Inheritance*

   *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan *metode class* asalnya dan menambah fungsionalitas baru, sehingga disebut sebagai anak dari *class* yang diwarisinya. *Inheritance* digambarkan dengan garis dengan kepala panah tertutup menunjuk dari anak *class* ke *class* asalnya.




**Gambar 2.4** Contoh *Class Diagram*

Sumber: *The Elements of UML 2.0 (TM) 2.0 Style*, 2009

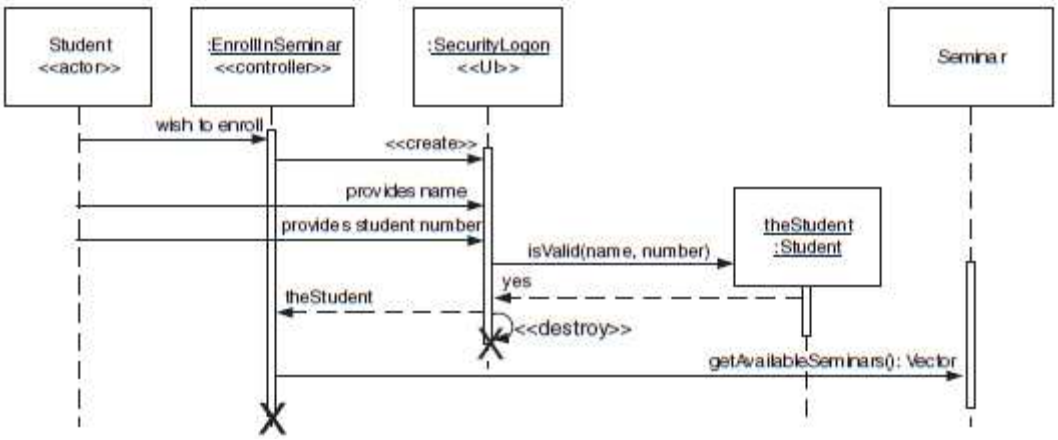
### 2.6.3 *Sequence Diagram*

Menurut menurut Ambler (2005:80) UML *sequence diagram* adalah teknik pemodelan yang dinamis. UML *sequence diagram* biasanya digunakan untuk :

- Validasi dan menyempurnakan logika dan kelengkapan penggunaan suatu skenario. Menunjukkan deskripsi cara bahwa sistem anda bisa digunakan.
- Menjelajahi desain karena menggambarkan langkah *visual* melalui permintaan dari operasi yang didefinisikan dari kelas.
- Memberikan perkiraan kelas mana dalam aplikasi yang akan menjadi kompleks.

- Mendeteksi hambatan dalam desain berorientasi objek, melihat pesan apa yang sedang dikirim ke objek, melihat kira-kira berapa lama waktu yang dibutuhkan untuk menjalankan metode, serta memahami tentang di mana perlunya mengubah desain untuk mempermudah jalannya sistem.

*Sequence diagram* menggambarkan interaksi antar objek di dalam dan disekitar sistem berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau langkah-langkah yang dilakukan sebagai respon dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang memicu aktifitas tersebut, proses dan perubahan apa saja yang terjadi secara *internal* dan *output* yang dihasilkan. *Lifeline* adalah pengklasifikasi atau contoh dari pengklasifikasi yang digambarkan di bagian atas *sequence diagram*. Aktor, kelas, komponen, objek, *use case*, dan sebagainya dianggap *lifeline*.




**Gambar 2.5** Contoh *Sequence Diagram Enrolling A Student In Seminar*

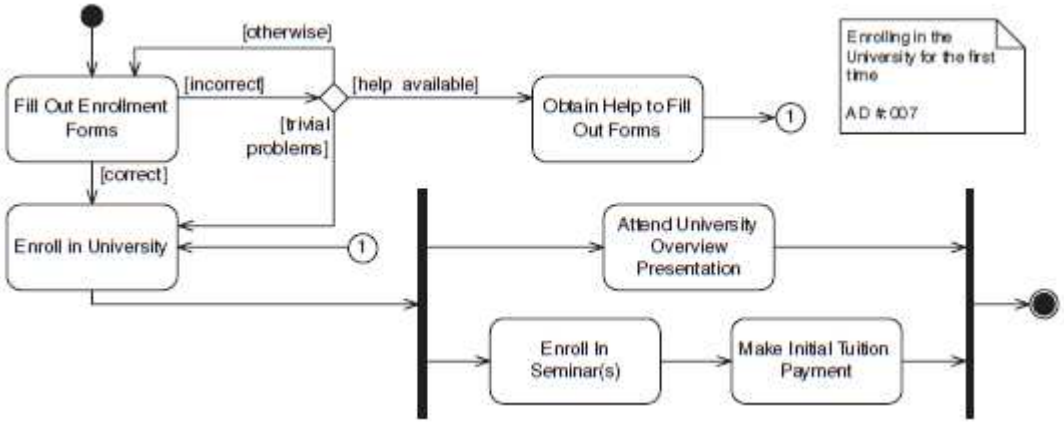
Sumber: *The Elements of UML 2.0 (TM) 2.0 Style*, 2009

### 2.6.4 *Activity Diagram*

Menurut Ambler (2005:113) *Activity diagram* pada UML setara dengan *flow charts* berorientasi objek dan *data flow diagram* dari *structured development*. *Activity Diagram* digunakan untuk mengeksplorasi logika dari:

- Operasi yang kompleks.
- Aturan bisnis yang kompleks.
- Satu *use case*.
- Beberapa *use case*.
- Proses bisnis.
- *Concurrent process*.
- Proses *software*.

*Activity Diagram* menggambarkan berbagai alur aktifitas dalam sistem yang dirancang, bagaimana alur berawal, keputusan yang terjadi, dan bagaimana alur berakhir.




**Gambar 2.6** Contoh *Activity Diagram Business Modelling Process*

Sumber: *The Elements of UML 2.0 (TM) 2.0 Style*, 2009

Beberapa bagian yang terdapat pada *activity diagram*, yaitu:

a. *Activity*

*Activity* berinteraksi dengan objek. Activity ini digambarkan berupa kotak dengan pinggiran yang tumpul.

b. *Action*

Merupakan sistem objek. Mencerminkan eksekusi suatu aksi. *Action* ini digambarkan dengan kotak dengan pinggiran yang lebih tajam.

c. *Starting Point*

Sebuah titik awal dimodelkan dengan lingkaran penuh, dengan menggunakan notasi yang sama dalam UML *state chart diagram*, setiap UML *activity diagram* harus memiliki titik awal, dan ditempatkan di pojok kiri atas.

d. *Ending Point*

Sebuah titik akhir dimodelkan dengan lingkaran penuh dengan perbatasan di sekitarnya. Sehingga ketika orang lain membaca *diagram*, maka mereka mengetahui bagaimana cara mengakhiri aktifitas.

e. *Fork*

Percabangan yang menunjukkan aliran. Dimana satu aliran masuk dan dua aliran meninggalkannya

f. *Joint*

Penggabungan yang menjadi arah aliran. Dimana dua aliran masuk dan satu aliran meninggalkannya.

## 2.7 *Waterfall*

Menurut Gomaa (2011:30) *Waterfall* adalah *process model* yang ideal dimana setiap tahap harus diselesaikan sebelum memulai tahap selanjutnya, dan *project* bergerak dari tahap satu ke tahap lainnya tanpa ada pengulangan atau penumpukan.

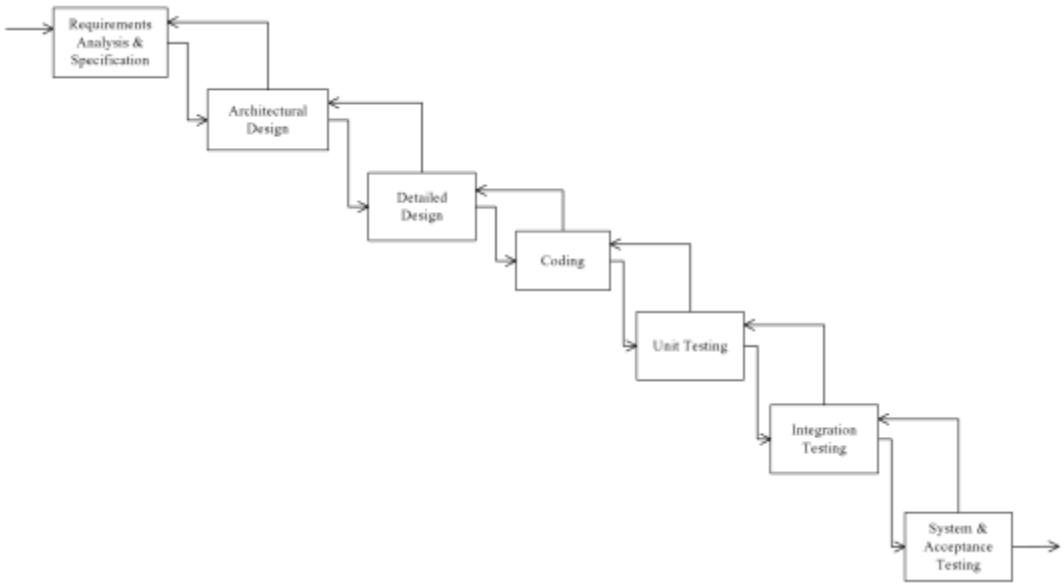
Namun, dalam prakteknya, penumpukan terkadang dibutuhkan untuk menyukseskan tahap dari *life cyle*, sama seperti tahap pengulangan ketika *error* terdeteksi.

Dalam beberapa *software development projects*, model *waterfall* memiliki masalah yang signifikan seperti:

- *Software requirements*, faktor kunci dari setiap proyek pengembangan software tidak di tes secara benar sebelum *working system* tersedia untuk di demonstrasikan ke *end-users*. Faktanya, beberapa pengamatan menunjukkan bahwa *error* di *requirement specification* biasanya terdeteksi paling akhir (seringkali sebelum *acceptance testing*) sehingga pengkoreksian adalah hal yang paling banyak membuang biaya.
- *Working system* menjadi tersedia di akhir *life cycle*. Desain utama atau masalah mungkin tidak terdeteksi sebelum sistem hampir di operasikan, dimana saat itu biasanya terlambat untuk dilakukan tindakan yang efektif.

Untuk *software development projects* dengan faktor resiko yang signifikan seperti *requirement* yang tidak dimengerti sepenuhnya atau diinginkan adanya perubahan, maka dibuatlah variasi dan alternatif bagi model *waterfall*.

Berikut ini adalah gambar model *waterfall* setelah dilakukan perubahan dengan proses pengulangan:




**Gambar 2.7** Contoh *Waterfall Model* Setelah Dilakukan Perubahan Dengan Proses *Iteration* (Pengulangan)

Sumber: *Software Modeling and Design UML, Use Cases, Patterns, and Software Architectures,*2011.

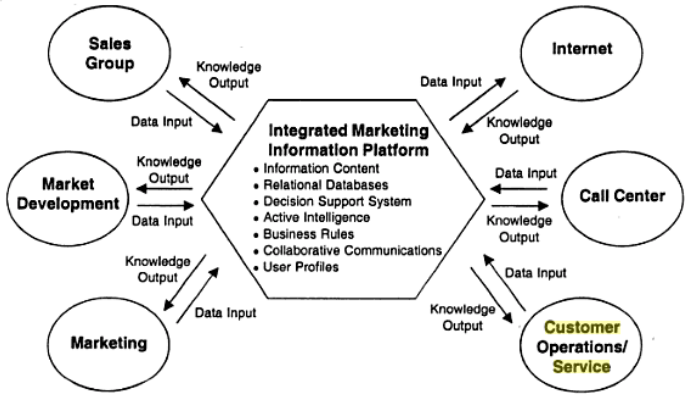
## 2.8 *Customer Relationship Management* (CRM)

Menurut Seth (2001:6) CRM atau *Customer Relationship Management* adalah strategi komprehensif, proses memperoleh, mempertahankan , dan *partnering* dengan selektif *customer* untuk membuat *superior value* bagi perusahaan dan *customer*. Proses formasi CRM menunjukan keputusan mengenai aktifitas relasional bagi perusahaan

untuk menghormati kelompok *customer* yang spesifik maupun *customer* perorangan demi *cooperative* dan *collaborative relationship*.

Kepuasan *customer* menjadi salah satu aspek penting untuk memajukan *marketing relationship.* Sehingga, pada lingkungan bisnis sekarang ini perusahaan melakukan hubungan dengan *customer* dengan cara termasuk *sales people, service personnel, call centers, internet websites, marketing departements, fulfillment houses, business development agents*, dan lainnya.

Oleh karena itu, perusahaan dengan antusias melakukan solusi CRM sebagai konsiderasi dasar untuk memuaskan *customer*. Hal ini membuat CRM juga harus terdiri dari komponen *Information Technology* (IT) yang signifikan, maka perusahaan menyerahkan tanggung jawab implementasi CRM ke bagian Departemen IT. CRM *tools* bertujuan untuk memaksimalkan strategi perusahaan untuk membuat hubungan *customer* yang lebih efektif. Strategi yang sesuai dan implementasi yang sempurna sangat dibutuhkan untuk mencapai hasil yang sukses.




**Gambar 2.8** *Information Platform Of CRM*

Sumber: *Customer Relationship Management: Emerging Concepts, Tools, and Applications*,2001.

## 2.9 *Customer Service*

Menurut Brink & Berndt (2009:56) *Customer Service* adalah penyongkong layanan dimana perusahaan membantu *customer* untuk tetap setia dan merasa aman terhadap perusahaan. *Customer service* tidak hanya dilihat dari hubungan perusahaan dengan *customer* tetapi juga hubungan terhadap *supplier*.

Layanan *customer service* sangat penting bagi *customer* dan *organization* sebagai dampak dari hubungan yang dibangun. Berikut ini adalah hal-hal mengenai tugas *customer service*:

- *Changing customer expectations*

  *Customer* pada saat ini lebih mengerti, meminta, dan modern daripada 30 tahun yang lalu. Sehingga memiliki ekspektasi dan lebih pemilih dari sebelumnya. Oleh karena itu *customer service* harus mendengarkan *customer* dengan lebih baik. *Customer service* juga harus mengantisipasi kebutuhan untuk menyelesaikan masalah sebelum masalah itu terjadi untuk mengesankan *customer* dan memberikan respon.

- *The increased importance of customer service*

  Dengan mengubah ekspektasi dari *customer,* kompetitor melihat *customer service* sebagai senjata yang kompetitif dengan membedakan produknya dari tawaran kompetitor lain.

- *The need for a relationship strategy*

  Memastikan strategi *customer service* terbuat, terimplementasi, dan terkontrol yang akan membuat ketertarikan

bagi *customer*, sehingga memiliki peran utama dan tidak hanya sebagai *sub-component* dari *marketing element*.

## 2.10 *Object Oriented Programming* (OOP)

Menurut Marrer (2009:215) OOP adalah *style programming* yang mengidentifikasi dan mengelompokkan variabel-variabel dan *modules* ke dalam *class objects. Class objects* mewakili hal seperti *customer*, pembelian, pembayaran dan mengekspresikannya di tiap hubungan data (*state*) dan *modules* (*behaviour*). *Class object* tersimpan di *file* yang bisa di akses oleh *logic program* dan yang lebih penting dapat digunakan kembali oleh program lainnya.

Keuntungan utama OOP adalah kemampuan membuat *snippets code* yang disebut *class objects* dan bisa digunakan kembali pada banyak aplikasi. *Generic code snippets* mengkombinasikan *key variables dan methods* ke dalam wadah yang disebut *class objects* sehingga dapat digunakan pada lebih dari satu program. OOP dikembangkan setelah *structure programming* dan meminjam banyak modularisasi dan abstraksi sehingga teknik *structure programming* akan membantu untuk memahami OOP. *Class* mewakili hal-hal, ide dan tempat memegang data yang disebut juga *properties* dan *atributte* serta *method* yang berhubungan dengan hal yang ada di *classes*. *Encapsulation, polimorphism, dan inheritance* menyediakan *code* yang bisa digunakan kembali pada *design time* dan *runtime.*

## 2.11 MySQL

Menurut King et.al (2009:4) MySQL adalah *open source database product* yang mendukung *key subsets* SQL di Linux dan Unix *systems*. MySQL digunakan secara gratis dan hanya menghabiskan sedikit biaya bagi periklanannya namun sangat populer. Tidak seperti *commercial database*, MySQL sangat terjangkau dan mudah digunakan.

Fitur yang terdapat di MySQL antara lain:

- *Openess*

  MySQL sangat terbuka dalam setiap hubungan. SQL *dialect* yang digunakan adalah ANSI SQL2 sebagai pembangunnya. *Database engine* menjalankan *platform* yang tidak terhitung, termasuk Windows 2000, Mac OS X, Linux, FreeBSD, dan Solaris. Jika *binary* tidak tersedia untuk *platform*, user dapat mengaksesnya dari *source* untuk di *compile* ke *platform* tersebut.

- *Application Support*

  MySQL memiliki API untuk semua bahasa pemrograman. *User* bisa menulis aplikasi *database* yang mengakses MySQL di C, C++, Eiffel, Java, Perl, PHP, Phyton, dan Tcl.

- *Cross-database joins*

  User bisa mengkonstruksi MySQL *queries* yang bisa menggabungkan *tabel* dari *database* yang berbeda

- *Outer join support*

  MySQL mendukung kiri dan kanan *outer joins* menggunakan ANSI dan sintaks ODBC.

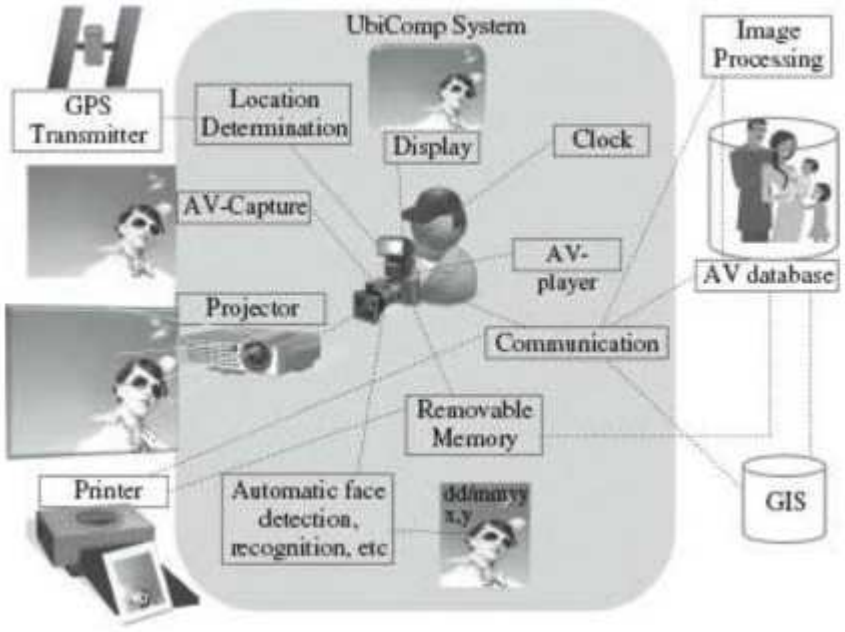
## 2.12 Ajax

Menurut Inc (2008:7) Ajax, *Asychronous* JavaScript and XM adalah tehnik baru yang mendeskripsikan bagaimana teknologi, JavaScript, DOM (*Document Object Model*) dan XML bisa digunakan bersamaan untuk membuat *web* aplikasi yang interaktif. Ajax menggunakan *asynchronus data transfer* antara *browsers* dan *web servers*, mengizinkan halaman *web* untuk meminta sedikit informasi dari *web server* dari seluruh halaman.

## 2.13 *Ubiquitous*

Menurut Poslad (2009:15) *Ubiquitous* memiliki arti muncul atau berada dimanapun, dikombinasikan dengan teknik *computing* dari *term Ubiquitous Computing* (UbiCom) yang digunakan untuk mendeskripsikan *Information and Communication Technology* (ICT) yakni sistem yang memungkinkan informasi dan *tasks* dibuat menjadi tersedia dimanapun untuk membantu kinerja manusia.

Aplikasi *Ubiquitous* menggunakan *network communication* untuk mengakses *relevant remote external information* dan *tasks* dimanapun dan kapanpun. Walaupun akses komunikasi bisa dimodelkan sebagai bagian dari sistem *internal,* inti dari jaringan infrastruktur komunikasi diperhitungkan sebagai *external* UbiCom *system* dan bagian dari *system virtual computing environment*.




**Gambar 2.9** Contoh *Ubiquitous Computing Application*

Sumber: *Ubiquitous Computing: Smart Devices, Environments and Interactions*,2011

*Ubiquitous* meliputi lingkup komputerisasi yang luas, tidak hanya alat-alat yang menggunakan teknik komputer secara *general*, namun juga alat ICT yang multifungsi seperti *phone, camera, game console, vehicle control system, mobile phones,* dan lainnya. Tiga tipe utama desain dari UbiCom sistem adalah:

- *Smart device*

  *Smart device* dapat termasuk *mobile smart devices, smart cards*, interaksi yang dilakukan kebanyakan dilakukan dalam dunia virtual dan mengurangi hubungan fisik. Hal ini langsung diakses pada *external devices* dan diaktivasi secara *manual* oleh *owner*.

- *Smart environment*

  *Smart environment* termasuk *device* seperti *sensor, controller,* dan *computer* yang tertanam atau dioperasikan dalam lingkungan, misalnya *robot*. Alat yang termasuk *smart environment* harus dapat memiliki kesadaran mengenai aktifitas *user* secara spesifik misalnya seperti pintu yang terbuka otomatis yang tidak memerlukan tuntunan dari *user*.

- *Smart interaction*

  *Smart interaction* terfokus pada *model* yang lebih kompleks yang di distribusikan dengan *software services, hardware resources, dynamic cooperation* untuk mencapai suatu tujuan.

## 2.14 Android

Menurut Meier (2009:1) Android adalah susunan *open source software* yang termasuk *operating system, middleware, dan key applications* bersama dengan satu set API *libraries* untuk menulis aplikasi *mobile* yang dapat membentuk, merasakan, dan memfungsikan *mobile handset*.

Android memiliki API yang sangat bermanfaat dan dokumentasi yang sempurna, *developer* yang berkembang, dan tidak membutuhkan biaya untuk pengembangan dan distribusi. Dalam Android, semua aplikasi memiliki kepentingan yang sama. Aplikasi milik *developer* dan aplikasi Android yang asli ditulis dengan API yang sama dan dieksekusi di waktu yang sama. Pengguna bisa menghapus dan mengganti aplikasi asli manapun dengan alternatif dari *developer*.

Secara garis besar arsitektur Android dapat dijelaskan sebagai berikut:

1. *Application dan widget*

   *Application* dan *widgets* adalah *layer* dimana dapat terhubung dengan aplikasi saja, dimana biasanya kita men-*download* aplikasi kemudian kita lakukan instalasi dan menjalankan aplikasi. Di *layer* terdapat aplikasi inti seperti klien *email*, program sms, kalender, peta, *browser*, kontak, dan lain-lain.

2. *Applications framework*

   Android adalah "*open development platform*", yaitu Android menawarkan kepada *developer* atau memberi kemampuan kepada *developer* untuk membangun aplikasi yang baik dan inovatif. *Developer* dapat bebas untuk mengakses perangkat keras, informasi *resource*, menjalankan *service background*, mengatur *alarm*, menambahkan *notification* dan sebagainya. *Developer* memiliki akses penuh menuju *API Framework* seperti yang dilakukan oleh aplikasi yang kategori inti. Arsitektur aplikasi dirancang supaya kita dengan mudah dapat menggunakan kembali komponen yang sudah digunakan.

   Komponen yang termasuk didalam *Application framework*:

   a. *Views.*
   b. *Content provider.*
   c. *Resource manager.*
   d. *Notification manager.*
   e. *Activity manager.*

3. *Libraries*

   *Libraries* adalah *layer* dimana fitur-fitur Android berada, biasanya pembuat aplikasi mengakses *libraries* untuk menjalankan aplikasinya. Berjalan di atas *kernel* layer ini meliputi berbagai *library* seperti libc dan SSL, misalnya:

   - *Libraries media* untuk pemutaran media *audio* dan *video.*
   - *Libraries* untuk *management* tampilan.
   - *Libraries Graphics* mencakup SGL dan OpenGL untuk grafis 2D dan 3D.
   - *Libraries* Sqlite untuk mendukung *database.*
   - *Libraries* SSL dan *webkit* terintegrasi dengan *web browser* dan *security.*
   - *Libraries* liveWebcore mencakup *modern web browser* dengan *engine embeded web view.*
   - *Libraries* 3D yang mencakup implementasi OpenGL.

4. Android *Run Time*

   *Layer* yang membuat aplikasi Android dapat dijalankan, dimana dalam prosesnya menggunakan implementasi Linux. *Dalvik Virtual Machine* (DVM) merupakan mesin untuk membentuk dasar kerangka aplikasi *android*. Di dalam Android *run time* dibagi menjadi tiga bagian yaitu:

   - *Core Libraries*

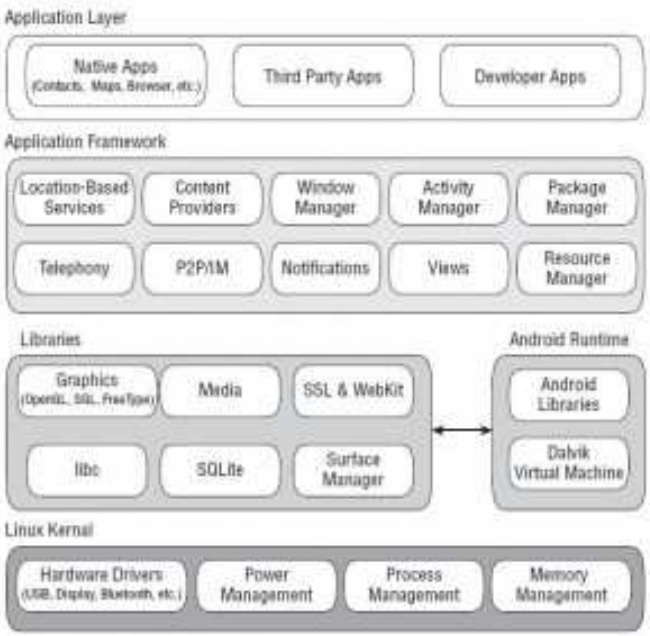
     Android dibangun dalam bahasa Java sementara Dalvik sebagai *Virtual Machine* bukan *Virtual Machine* Java, sehingga diperlukan sebuah *libraries* yang berfungsi menerjemahkan bahasa Java yang di tangani oleh *Core Libraries.*

-*Dalvik Virtual Machine* (DVM)

*Virtual machine* berbasis *register* yang di optimalkan untuk menjalakan fungsi-fungsi secara efisien. Dimana merupakan pengembangan yang mampu membuat *linux kernel* untuk melakukan *threading* dan manajemen tingkat rendah.

-*Linux Kernel*

*Linux Kernel* adalah inti servis (termasuk *hardware drivers*, proses, dan *management memory*, *security*, *network*, dan *power management*) ditangani oleh *Linux 2.6 kernel*. *Kernel* juga menyediakan lapisan abstraksi antara *hardware* dan sisa tumpukan dari *stack*.




**Gambar 2.10** *Application Layer, Application Framework, Libraries,* Dan *Linux Kernel*

Sumber: *Professional Android Application Development*,2009

## 2.15 *The Dalvik Virtual Machine* (DVM)

Menurut Meier (2009:15) daripada menggunakan traditional Java *Virtual Machine* (JVM) seperti Java ME *(*Java *Mobile Edition),* Android menggunakan rancangan *virtual machine* sendiri di desain untuk memastikan beberapa instansi berjalan dengan efisien dalam *single device*. *Dalvik* VM menggunakan alat pokok *Linux kernel* untuk menangani fungsional *level* rendah seperti *security*, *threading*, *process* dan *memory management*.

Semua Android *hardware* dan akses sistem servis diatur menggunakan Dalvik sebagai tingkat menengah. Dengan menggunakan VM sebagai *host application execution, developer* memiliki lapisan abstraksi yang memastikan mereka tidak perlu merasa cemas terhadap implementasi *hardware*. *Dalvik* VM me-*execute Dalvik executeable files*, sebuah format dioptimalkan untuk memastikan jejak memori minimal. *Dex executables* dibuat dengan mengubah bahasa Java di *compile classes* menggunakan *tools* yang disediakan menggunakan SDK.

## 2.16 Android *Software Development Kit* (SDK)

Menurut Meier (2009:9) Android SDK termasuk segala hal yang dibutuhkan dalam *developing, testing*, dan *debuging*. Hal yang termasuk dalam SDK *download* adalah:

- API (*Application Programming Interface*) yang *libraries*-nya menyediakan *developer* untuk mengakses Android *stack*.

- *Development tools* untuk mengubah *source code* Android menjadi *executable* Android *applications,* SDK termasuk beberapa *development tools* yang mengizinkan untuk me-*compile* dan *debug* aplikasi.
- Android *Emulator* adalah Android *device* yang sepenuhnya interaktif dengan beberapa alternatif *skins*. Dengan menggunakan *emulator*, kita bisa melihat bagaimana aplikasi terlihat dan berjalan dalam Android *device*.
- *Full documentation* SDK termasuk *extensive code-level* referensi informasi secara detail persis dengan yang termasuk dalam setiap *package* dan *class* serta bagaimana menggunakannya.

## 2.17 *Android Development Tools* (ADT)

Menurut Lee (2011:7) *Android development Tools* (ADT) adalah *plugin* yang didesain untuk IDE(*Integrated Development Environtment*) Eclipse yang memberikan kemudahan dalam mengembangkan aplikasi Android. Dengan menggunakan ADT untuk Eclipse akan memudahkan *developer* dalam membuat aplikasi *project* Android, membuat GUI aplikasi, dan menambahkan komponen-komponen lainnya, serta kita dapat melakukan *running* aplikasi menggunakan Android SDK melalui Eclipse. Dengan ADT, kita dapat melakukan pembuatan *package Android* (.apk) yang digunakan untuk distribusi aplikasi *Android* yang dirancang.

## 2.18 Eclipse

Menurut Holzner (2004:1) Eclipse adalah sebuah IDE (*Integrated Development Environtment*) yang mengembangkan perangkat lunak dan dapat dijalankan di semua

*platform*. Eclipse pada saat ini merupakan salah satu IDE yang disukai dikarenakan gratis dan *open source*, yang berarti setiap orang boleh melihat kode pemrograman perangkat lunak ini. Selain itu kelebihan dari Eclipse yang membuatnya terkenal adalah kemampuannya untuk dapat dikembangkan oleh pengguna dengan komponen *plug-in*.

## 2.19 SQLite

SQLite adalah *library* perangkat lunak yang *self contained, serverless, zero-configuration,* dan *transactional* SQL *database engine*. SQLite adalah *database* yang paling banyak digunakan mesin *database* di dunia. Kotak sumber SQLite berada pada *public domain*.

(Anonymous.n.d.*About Sqlite*.*Retrieved*:11-10-2012 *from* http://www.sqlite.org)

## 2.20 *Javascript*

Menurut Flanagan (2011:1) *Javascript* adalah bahasa pemrograman untuk *web*. Javascript merupakan *high-level, dynamic, untyped-intepreted programming language* yang cocok dengan *object-oriented* dan *functional programming style*. *Syntax* pada *Javascript* berasal dari Java namun sangat berbeda dengan *Java programming language*. Seiring dengan berjalannya waktu, *scripting language Javascript* menjadi lebih kuat dan efisien.

## 2.21 *Cascading Style Sheets* (CSS)

Menurut Collison, Budd, & Moll (2009:245) Dibandingkan dengan *programming language* lainnya, CSS merupakan bahasa yang mudah dimengerti karena kumpulan kode-kode yang berurutan dan saling berhubungan untuk mengatur tampilan

suatu HTML memiliki logika yang tidak sulit. Namun kesulitan banyak ditemui ketika melakukan *test* pada *browser* yang berbeda. *Browser bug* dan penamaan yang tidak konsisten sering kali menjadi masalah utama bagi kebanyakan CSS *developers*.

## 2.22 *Model View Controller* (MVC)

Menurut Mackey (2010:295) MVC atau *Model View Controller* memiliki arti sebagai berikut:

- *Model* merupakan *database*.
- *View* merupakan *pages* dan *controls*.
- *Controller* akan mengatur interaksi antara *pages*/ mengkontrol *view* dan *model*.

Keuntungan MVC adalah sebagai berikut:

- *Division/testability*

  *Cotrollers* mengatur interaksi antara *User Interface* dan *data (model).*

- *Flexibility*

  Setiap *individual layer* mudah dibuang tanpa mempengaruhi *layer* lainnya karena dapat di *costumize* sehingga *user interface* dapat diganti atau menggunakan *database* yang berbeda.

- *Maintability*

  Walaupun dapat di *costumize*, tapi *projects* harus tetap memiliki *code* yang teratur. Oleh karena itu, *project structure* biasanya kaku dan *new developers* harus dapat memahami arsitekturnya dengan cepat.

## 2.23 *Eight Golden Rules*

Menurut Kumar (2005:72) Sudut pandang *user* menentukan tampilan baik atau buruknya tampilan. Sebuah *interface* yang disukai oleh seorang *user* bisa jadi tidak disukai oleh *user* lainnya. Dengan mengikuti prinsip yang dikenal *dengan eight golden rules*, maka *interface* akan terlihat lebih mudah dimengerti. *Eight golden rules* ini pertama kali diutarakan oleh Schneiderman yang terdiri dari:

- *Strive for consistency in action sequences, layout, terminology, command use, etc.*

  Tampilan harus konsisten dalam urutannya, rancangan, penggunaan perintah, dan lainnya.

- *Enable frequent users to use shortcuts, such as abbreviations, special key sequence, regular actions, etc.*

  *User* yang sering menggunakan aplikasi membutuhkan adanya *shortcut* seperti tombol khusus untuk mempercepat akses pada aksi yang sering dilakukan.

- *Offer informative feedback for every user action.*

  Memberikan umpan balik kepada *user* sesuai dengan setiap aksi yang dilakukan, hal ini dapat berupa konfirmasi atau informasi atas suatu aksi.

- *Design dialogs to yield closure so that the user is aware of when they have completed a task.*

  Adanya peringatan untuk memberitahu bahwa *user* telah menyelesaikan suatu perintah.

- *Offer error prevention and simple error handling.*

  Sistem dibuat untuk mencegah *user* untuk melakukan kesalahan, apabila terjadi kesalahan, sistem memberi instruksi sederhana untuk memperbaikinya.

- *Allow easy reversal of actions in order to relieve an essential part and encourage exploration.*

  Saat *user* melakukan aksi yang tidak diinginkan, maka sistem harus memungkinkan *user* melakukan pembatalan agar user tidak khawatir untuk menggunakan aplikasi tersebut.

- *Support internal locus of control so that the user is in control of the system.*

  Sistem yang tidak terduga dan sulitnya melakukan aksi akan menyulitkan *user*. Oleh karena itu sistem harus didukung kontrol yang kuat agar *user* merasa menguasai sistem tersebut.

- *Reduce short term memory load by keeping displays simple and consolidated multiple page display and providing time for learning the action sequences.*

  Mengurangi memori jangka pendek dengan membuat tampilan lebih sederhana