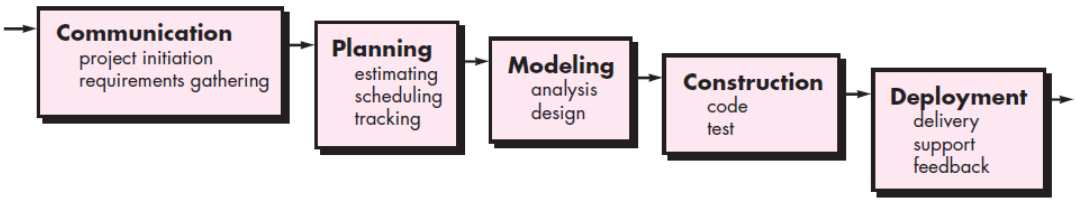
BAB 2

TINJAUAN PUSTAKA

2.1 *Waterfall Model*

Menurut Pressman (2010, p. 39), *waterfall model* juga disebut sebagai *classic life cycle* yang mengembangkan *software* dengan pendekatan sistematis dan berurutan. Pendekatan ini diawali dengan pengumpulan kebutuhan yang selanjutnya diikuti dengan perancangan, pemodelan, pembangunan dan implementasi dari sistem *software*. Tahapan-tahapan pada metode *waterfall* dapat dilihat pada gambar 2.1.



Gambar 2.1 *Waterfall Model*

1. *Communication*

Tahap ini merupakan tahap inisiasi proyek dengan melakukan analisis terhadap masalah yang ada dan tujuan yang ingin dicapai. Tahap ini juga melakukan analisis terhadap kebutuhan *software* dan pengguna yang didapatkan melalui wawancara dan analisis aplikasi terkait. Pengumpulan data-data tambahan juga dapat diperoleh di jurnal, artikel maupun dari sumber buku.

2. *Planning*

Tahap ini merupakan tahap lanjutan dari *communication*. Pada tahap ini dilakukan estimasi terhadap kebutuhan-kebutuhan dalam pembuatan sistem dan membuat perencanaan terhadap jadwal pengerjaan *software.*

3. *Modeling*

Pada tahap ini dilakukan pembuatan pemodelan terhadap *software* dengan melakukan analisis dan perancangan *software* berdasarkan hasil dari analisis kebutuhan pengguna yang sudah didapatkan sebelumnya. Perancangan *software*

tersebut berupa perancangan struktur data, arsitektur program, *user interface*, dan detail prosedur.

4. *Construction*

Setelah membuat pemodelan terhadap *software*, tahap selanjutnya adalah membuat kode (*coding*) berdasarkan kebutuhan pengguna dan *software* tersebut. Setelah pengerjaan *coding* selesai maka selanjutnya pengujian (*testing*) dilakukan untuk meminimalkan kesalahan-kesalahan yang mungkin terjadi dalam sistem.

5. *Deployment*

Setelah proses *construction* selesai, tahap terakhir adalah mengimplementasikan *software* tersebut kepada pengguna untuk mendapatkan umpan balik dari *software* tersebut. *Software* yang telah dibuat juga harus dilakukan pemeliharaan, perbaikan dan pengembangan secara berkala agar dapat berjalan dengan baik.

Waterfall model ini dapat diaplikasikan dengan mudah dan dapat mendefinisikan semua kebutuhan sistem secara utuh, eksplisit dan benar di awal proyek sehingga *software engineering* dapat berjalan dengan baik karena kebutuhan sistem dan masalah-masalah yang mungkin terjadi sudah dipikirkan secara matang. Namun, kekurangan yang utama dari model ini adalah perubahan yang sulit dilakukan karena sifatnya kaku dimana fase sebelumnya harus lengkap dan selesai terlebih dahulu sebelum mengerjakan fase berikutnya.

2.2 Interaksi Manusia dengan Komputer (IMK)

Rancangan antar muka merupakan jembatan komunikasi yang efektif antara manusia dengan komputer. Rancangan ini mengidentifikasi objek-objek dan aksi-aksi antarmuka dan kemudian membuat tampilan layar yang membentuk dasar dari *user interface prototypes*. (Pressman, 2010, p. 312).

Menurut Shneiderman (2010, p. 80-89), penggunaan delapan aturan emas berasal dari prinsip-prinsip yang didapatkan dari pengalaman-pengalaman dan direvisi selama lebih dari dua dekade. Delapan aturan emas tersebut adalah sebagai berikut :

1. **Berusaha Untuk Konsisten**

   Urutan aksi yang konsisten diperlukan dalam situasi yang sama. Konsistensi harus digunakan pada prompt, menu, layar bantu, warna tampilan, kapitalisasi, huruf dan sebagainya.

2. **Memenuhi Kegunaan yang *Universal***

   Mengenali kebutuhan pengguna yang beragam dan memudahkan dalam melakukan perubahan konten. Mencari perbedaan user pemula dan user ahli, rentang usia, keterbatasan kemampuan fisik, perbedaan teknologi merupakan panduan dalam merancang *interface*. Misalnya, untuk pengguna yang masih pemula diberikan keterangan dan pemberian fitur *shortcut* untuk pengguna yang sudah ahli.

3. **Memberikan Umpan Balik yang Informatif**

   Untuk setiap tindakan yang dilakukan pemakai, diharapkan adanya umpan balik dari sistem. Untuk tindakan yang sering terjadi dan tidak membutuhkan banyak aksi, respon yang ada dapat dibuat secara sederhana, sedangkan tindakan yang jarang dilakukan dan membutuhkan banyak aksi harus lebih ditonjolkan.

4. **Merancang Dialog yang Memberikan Penutupan (Keadaan Akhir)**

   Urutan aksi harus disusun ke dalam kelompok dengan permulaan, pertengahan dan penutupan. Suatu umpan balik yang informatif pada penutupan aksi-aksi memberikan pengguna indikator bahwa proses telah sukses dan dapat melanjutkan ke aksi berikutnya.

5. **Mencegah Kesalahan**

   Sistem yang dirancang harus dapat meminimalkan kesalahan serius yang dibuat oleh pengguna. Apabila kesalahan terjadi, sistem harus dapat mendeteksi kesalahan dan memberi instruksi perbaikan yang sederhana, membangun dan spesifik.

6. **Memungkinkan Pembalikan Aksi yang Mudah**

   Sistem ini berguna untuk mengembalikan suatu aksi yang telah dilakukan oleh pengguna. Sistem ini juga berfungsi untuk membantu pengguna ketika melakukan kesalahan sehingga dapat kembali ke keadaan. Sistem ini juga membuat user dapat mencoba fitur yang belum mereka ketahui tanpa harus takut melakukan kesalahan.

7. **Mendukung Pusat Kendali Internal**

   Membuat pengguna merasa memegang kendali atas *interface*. Pengguna yang sudah mahir dan memiliki pengalaman tentu menginginkan *interface* yang baik dan merasa bertanggung jawab atas segalanya termasuka *action* yang dilakukan, sehingga segala perubahan harus diminimalisir.

8. **Mengurangi Beban Ingatan Jangka Pendek**

   Karena keterbatasan manusia dalam memproses informasi pada ingatan jangka pendek, maka pembuatan *interface* sebaiknya dibuat menjadi lebih sederhana. Misalnya, beberapa tampilan yang sama dibuat menjadi satu halaman.

2.3 Pengertian *User Interface*

Menurut Satzinger (2012, p. 189), *User Interface* merupakan *input* dan *output* yang lebih melibatkan pengguna sistem secara langsung. Ada tiga aspek penting di dalam perancangan *User Interface,* yaitu sebagai berikut :

1. *Physical aspects,* terdiri dari perangkat yang disentuh oleh *user,* termasuk *desk, chair, light, keyboard, mouse, touch screen* atau *keypad*.
2. *Perceptual aspects,* terdiri dari semua hal yang dilihat, didengar atau disentuh oleh pengguna akhir (diluar *physical devices*), termasuk segala hal yang terdapat di layar monitor.
3. *Conceptual aspects*, terdiri dari semua hal yang diketahui oleh pengguna di dalam menggunakan sistem, seperti *customers, partners, orders, shipment* dan *feedback*.

2.4. Sistem Basis Data

Menurut Connolly dan Begg (2010, p. 54), Sistem Basis Data adalah kumpulan dari program aplikasi yang berinteraksi dengan *database* bersama DBMS dan *database* itu sendiri. Dapat disimpulkan bahwa Sistem Basis Data adalah sekumpulan data yang berhubungan yang digunakan oleh system dengan menggunakan DBMS (*Database Management System*).

Menurut Connolly dan Begg (2010, p. 66), *Database Management System* adalah sebuah sistem *software* yang memungkinkan *user* untuk mendefinisikan, membuat, menjaga dan memiliki akses ke basis data. DBMS memungkinkan *user* untuk mendefinisikan basis data dengan menggunakan DDL (*Data Definition Language*). DBMS juga memungkinkan *user* dalam melakukan *insert, update, delete*, dan mendapatkan data dari basis data melalui DML (*Data Manipulation Language*).

Suatu sistem aplikasi disebut DBMS jika memenuhi persyaratan sebagai berikut :

1. Menyediakan fasilitas untuk mengelola akses data
2. Mampu menangani integritas data
3. Mampu menangani akses data yang dilakukan secara bersamaan

4. Mampu menangani *backup* data

Karena pentingnya data bagi suatu organisasi/perusahaan, maka hampir sebagian besar memanfaatkan DBMS dalam mengelola data yang mereka miliki. Pengelolaan DBMS sendiri biasanya ditangani oleh DBA (*Database Administrator*).

2.5. *User Acceptance Test* (UAT)

Dalam membangun suatu sistem, penting bagi tim proyek untuk mengetahui apakah sistem yang dibuat atau dikembangkan telah sesuai dengan apa yang diharapkan oleh klien. Untuk mengetahui hal tersebut, tim proyek harus melakukan uji coba testing dengan mengacu pada harapan klien. Harapan klien untuk suatu sistem biasanya ditetapkan di awal pembuatan dari suatu sistem. Harapan-harapan yang telah ditetapkan ini merupakan tolak ukur bagi tim proyek untuk mengetahui apa saja hal yang masih kurang dari suatu sistem yang telah dibuat.

Menurut Pulusuri (2008, p. 62), *User Acceptance Test* dilakukan untuk memeriksa apakah produk perangkat lunak sudah siap untuk digunakan oleh *customer*. Pengujian dikatakan selesai apabila telah dilakukan oleh *user* yang bersangkutan. Sebelum melakukan pengujian ini, *developer* harus menyediakan sebuah *test case* yang menunjukkan kualitas dan penerimaan sebuah produk. *User Acceptance Test* ini dibuat sebagai sebuh scenario nyata melalui validasi aspek fungsionalitas maupun non-fungsionalitas dari produk perangkat lunak.

Menurut Naik dan Tripathy (2008), *User Acceptance* adalah kegiatan yang dilakukan oleh pihak klien untuk memastikan bahwa sistem memenuhi kriteria yang tertera pada kontrak atau perjanjian sebelum pihak klien melakukan *sign-off* yang menyatakan bahwa sistem sudah sesuai dengan kriteria yang diinginkan oleh pihak klien.

Menurut Roth, Dennis dan Wixom (2013, p. 453-454), *Acceptance Testing* dibagi menjadi 2 tahap yaitu *alpha testing* dan *beta testing*. Perbedaan dari kedua testing ini terletak pada data yang digunakan saat pengujian. *Alpha testing* menggunakan data buatan (*dummy*) sedangkan *beta testing* menggunakan data asli (*real*). Pada umumnya, *acceptance testing* yang digunakan adalah *alpha testing* dimana pengujian dilakukan oleh *user* sendiri untuk memastikan mereka menyetujui sistem tersebut.

1.5 *Acceptance Testing* sangat penting peranannya bagi keberhasilan sistem. Hal ini disebabkan karena sudut pandang pertama yang dibayangkan oleh *user* yaitu ketika mereka pertama kali mencoba sistem tersebut, yaitu

pada saat *acceptance testing* itu. Oleh karena itu, *acceptance testing* ini harus dilaksanakan dengan sebaik-baiknya dan harus melalui *system testing* yang sukses terlebih dahulu.

Menurut Lewis (2009, p. xiii), tahapan dalam *acceptance testing* dibagi menjadi 4, yaitu:

1. Melengkapi *acceptance test planning*.
2. Melengkapi *acceptance test case.*
3. Meninjau kembali atau menyetujui *acceptance test plan*.
4. Melaksanakan *acceptance test*.

Pada tahap pertama yaitu melengkapi *acceptance test planning*, tim proyek membuat terlebih dahulu rencana *acceptance test* yang akan dijalankan. Apa saja yang akan dilakukan pada saat *acceptance test*, semua dijabarkan secara lengkap.

Pada tahap kedua yaitu melengkapi *acceptance test case*, disini tim proyek menjelaskan mengenai urutan tahapan-tahapan yang akan di lakukan didalam *acceptance test*. Langkah apa yang harus dilakukan setelah tahapan sebelumnya selesai. Semua terhubung menjadi suatu *test case* yang dapat mempermudah tim proyek dalam melakukan *acceptance test*.

Kegiatan yang dilakukan pada tahap ketiga yaitu meninjau kembali atau menyetujui *acceptance test plan*. Pada tahap ini, *project leader* akan meninjau terlebih dahulu tahapan-tahapan yang telah disusun oleh tim proyek agar diketahui apakah *acceptance test* tersebut telah sesuai dan cocok dengan sistem yang yang akan diuji. Bila belum sesuai, *project leader* akan mengembalikan rencana tersebut kepada tim proyek untuk diperbaiki atau diulang penyusunannya. Namun, bila rencana tersebut telah sesuai maka *project leader* akan menyetujui rencana tersebut sehingga tim proyek dapat melanjutkan ke tahap selanjutnya.

Pada tahap keempat atau tahap terakhir, tim proyek akan melaksanakan *acceptance test* yang telah disetujui oleh *project leader*. *Acceptance test* ini turut mengikutsertakan *user* sebagai pengguna sistem untuk menguji sistem tersebut. Hal itu disebabkan karena *acceptance test* ini bertujuan untuk mengetahui apakah sistem yang dibuat ini telah memenuhi ekspektasi yang diharapkan oleh *user* tersebut.

Mengenai data yang akan digunakan dalam *acceptance test* ini, akan bergantung pada jenis *acceptance test* apa yang digunakan. Bila sedang melakukan *alpha test*, maka data yang digunakan adalah data *dummy*. Sedangkan bila sedang melakukan *beta test*, maka data yang akan digunakan adalah data *real*.

2.6. Pengertian *Web Application*

Menurut Shelly (2012, p. 14), *Web Application* adalah sebuah *website* yang mengizinkan pengguna untuk mengakses dan berinteraksi menggunakan *software* dari komputer atau perangkat manapun yang terhubung ke dalam *internet*. Menurut Pressman (2010), *Web Application* atau disingkat *WebApps* merupakan suatu *software* yang terintegrasi dengan halaman *web* dan menjalankan fungsi-fungsi aplikasi yang dapat menjalankan fitur untuk menyampaikan konten kepada *user*, baik berupa grafik, teks ataupun konten lainnya yang terhubung dengan *database* dan aplikasi bisnis. *WebApps* memiliki beberapa atribut, yaitu :

1. ***Network Intensiveness***

   Sebuah ketentuan dimana *web application* harus bekerja berdampingan dengan *network* yang melayani sejumlah *user*.

2. ***Concurrency***

   Jumlah pengguna aktif sebuah *web application*.

3. ***Unpredictable load***

   Jumlah pengguna dalam sebuah *web application* dapat berubah dari hari ke hari.

4. ***Performance***

   Jika performa suatu *web application* rendah (contoh: *loading* lama), maka *user* akan memutuskan untuk ke tempat lain.

5. ***Availability***

   Sebuah tuntutan *user* agar *web application service* berjalan 24/7/365 (*non-stop*).

6. ***Data driven***

   Menggunakan *hypermedia* untuk menyampaikan konten kepada pengguna.

7. ***Content sensitive***

   Menjaga kualitas konten yang diberikan, karena hal tersebut menjadi tolak ukur sebuah *web application* yang baik.

8. ***Continuous Evolution***

   Web application akan terus berkembang sesuai dengan kebutuhan konsumen dan perkembangan teknologi.

9. ***Immediacy***

   Dapat menampilkan informasi yang dapat mempengaruhi pasar dalam beberapa hari/minggu ke depan.

10. ***Security***

    Karena *web application* terhubung dengan *network*, maka dibutuhkan pengamanan yang kuat untuk melindungu data-data *sensitive* yang ada di dalamnya.

11. ***Aesthetics***

    User experience saat menggunakan *web application* dari segi tampilan.

2.7. *Point ofsales* (POS)

Menurut Hendry (2010, p. 1), *Point of Sales* atau POS adalah sebuah sistem yang terdiri dari *hardware* dan *software* yang didesain sesuai dengan keperluan dan dapat diintegrasikan dengan beberapa alat pendukung agar dapat membantu mempercepat proses transaksi.

Sistem *Point of Sales* (POS) adalah sebuah sistem aplikasi yang diterapkan pada bisnis minimarket ataupun pertokoan untuk menangani pengolahan data dan transaksi pembelian, penjualan dan pelaporan transaksi yang secara penting dibutuhkan dalam pengambilan keputusan strategik oleh para pebisnis tersebut.

Menurut Hendry (2010, p. 2), sistem POS bertujuan untuk menunjang kegiatan usaha suatu perusahaan dagang. Hal ini tidak bisa diakomodasi oleh mesin kasir, yang biasa dikenal dengan nama *cash register*. Sistem POS mampu melakukan kontrol terhadap stok yang cepat dan akurat. Sebuah mesin POS juga dilengkapi fasilitas cetak nota atau faktur yang berbeda-beda, tidak selalu menggunakan ukuran kertas yang kecil. Sistem POS juga dapat diintegrasikan dengan alat lain seperti printer dan memiliki sistem operasi yang multiguna sehingga dapat dimodifikasi sesuai keperluan bisnis yang diinginkan. Perancangan sistem aplikasi *Point of Sales* dapat memberikan pelayanan yang lebih baik kepada konsumen, seperti dalam perhitungan harga dan jumlah barang yang dibeli dapat menjadi lebih cepat dan kuantitas barang tidak lagi bergantung kepada pencatatan manual.

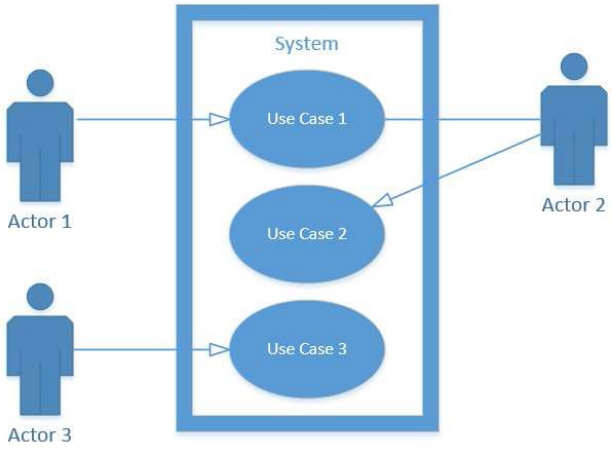
2.8. *Unifies Modeling Language* (UML)

Unified Modeling Language (UML) adalah kumpulan berbagai metode dalam pengembangan perangkat lunak yang menjelaskan sistem perangkat lunak dengan objek melalui metode grafis yang standar. UML diagram dapat dikatakan sebuah cetakan biru untuk membangun sebuah program. Cetakan biru itu pun terdiri dari berbagai macam dan memiliki fungsi berbeda yang akan dikumpulkan menjadi satu

set cetakan biru yang membantu tim pengembang dalam membuat sebuah projek. (Whitten and Bentley, 2007, p. 371). Ada pun UML yang digunakan dalam projek ini adalah sebagai berikut :

2.8.1.*Use Case* Diagram

Menurut Whitten and Bentley (2007, p. 246-250), *Use Case Diagram* adalah sebuah UML yang menggambarkan sistem dengan menggambarkan hubungan antara aktor *user* dengan *system use case*, yang harus mengikuti beberapa aturan yang telah ditentukan. Metode ini sangat cocok untuk menggambarkan interaksi secara jelas antara pengguna dan sistem.



***Gambar 2.2 Use Case* Diagram**

Komponen yang terdapat pada *use case* diagram :

a. *Use case*

Use case modeling mengidentifikasikan dan mendeskripsikan setiap fungsi dalam sistem yang digambarkan dengan bentuk *eclipse*.

Gambar 2.3 Use Case Symbol

b. Aktor

Aktor adalah segala sesuatu yang butuh atau perlu berinteraksi dengansistem untuk bertukar informasi, yang digambarkan dengan *stick figure* disertai dengan label perannya.

Gambar 2.4 Actor Symbol

Adapun beberapa tipe dari aktor, yaitu :

1) *Primary Business Actor*

Aktor yang mendapat manfaat langsung dari berjalannya sebuah proses *use case* berupa sesuatu yang dapat diukur atau dapat diobservasi.

2) *Primary System Actor*

Aktor yang secara langsung berhubungan dengan sistem untuk menginisiasi atau memulai bisnis maupun peristiwa pada sistem.

3) *External Server Actor*

Aktor yang merespon permintaan dari *use case*.

4) *External Receiver Actor*

Bukan aktor utama tetapi penerima hasil output dari sebuah *use case*.

c. *Relationships*

Relationships digambarkan dengan menggunakan sebuah garis antara dua simbol pada *use case* diagram. *Relationships* pada *use case* ada 4, yaitu *associations, extends, use*s, *depends on* dan *inheritance*.

1) *Associations*

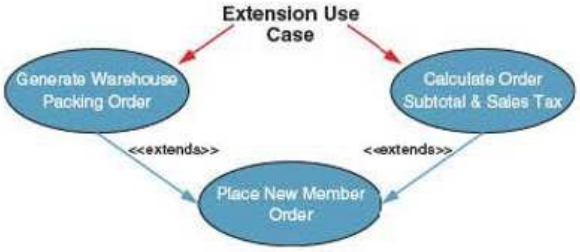
Associations adalah sebuah hubungan atau interaksi antara aktor dengan *use case* dimana *use case* mendeskripsikan interkasi tersebut. Tanda panah tertup menandakan bahwa aktor berinteraksi secara dua arah terhadap *use*

case tersebut, sedangkan yang tanpa tanda panah menandakan bahwa aktor hanya menerima hasil dari *use case* tersebut.

Gambar 2.5 Associations

2) *Extends*

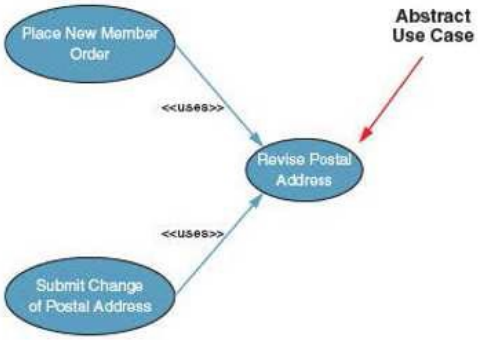
Extends adalah sebuah pemecahan dari sebuah *use case* yang kompleks menjadi *use case* baru yang lebih sederhana. Satu *use case* dapat memiliki banyak *extends,* akan tetapi setiap *extends* hanya berjalan dengan *use case* yang di-*extends*nya.



Gambar 2.6 Extends

3) *Uses (or include)*

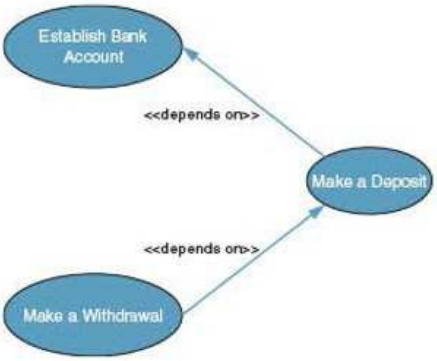
Uses adalah sebuah hubungan yang mempresentasikan *abstract use case* yaitu sebuah *use case* yang memiliki langkah-langkah yang sama dalam fungsinya tetapi tetap sebagai *use case* yang berbeda. Terkadang *uses* disebut juga sebagai *include*.



Gambar 2.7 Uses

4) *Depends on*

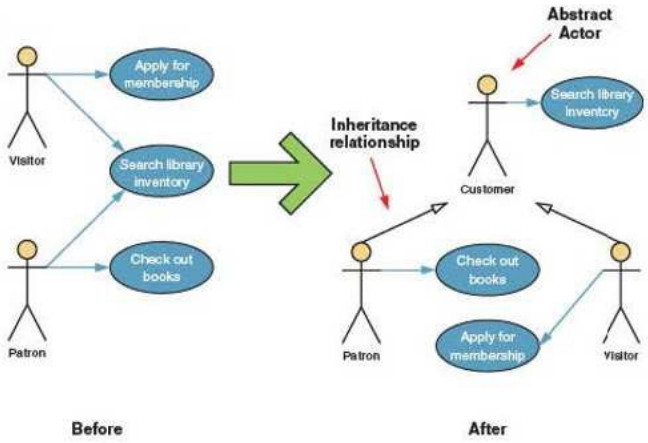
Depends on adalah sebuah hubungan dimana sebuah *use case* hanya berjalan setelah *use case* tertentu telah menjalankan fungsinya.



Gambar 2.8 Depends on

5) *Inheritance*

Inheritance digunakan ketika terdapat dua atau lebih aktor yang melakukan *use case* yang sama. Untuk mengurangi redundansi komunikasi pada sistem, digunakan aktor abstrak yang melakukan *use case* yang dilakukan oleh aktor-aktor tersebut.



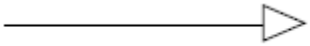
Gambar 2.9 Inheritance

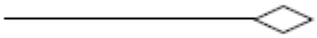
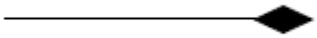
2.8.2.*Class* Diagram

Class diagram adalah sebuah gambar grafis dari struktur objek sistem *class* yang memiliki atribut dan operasi untuk melakukan relasi antara satu *class* dengan *class* yang lain. (Whitten *and* Bentley, 2007, p. 400)

Ada 4 jenis hubungan yang ada di dalam *class* diagram :

Tabel 2. 1 Hubungan dalam Class Diagram

1. Asosiasi	Asosiasi memungkinkan suatu *class* untuk menggunakan *attribute* atau *operation* yang dimiliki oleh *class* lain. Asosiasi digambarkan dengan garis tanpa tanda panah
2. Generalisasi dan Spesialisasi 	Generalisasi dan spesialisasi merupakan hubungan dimana suatu *class* dapat lebih *general* atau lebih spesifik dari *class* lainnya. *Class supertype* berisi atribut dan *behavior* umum , sedangkan *class subtype* berisi atribut dan *behavior* unik dari objek namun mewarisi atribut dan

	behavior dari *class supertype.*
3. Agregasi 	Agregasi merupakan hubungan dimana *class* yang lebih besar mengandung *class* yang lebih kecil. Hubungan agregasi dilambangkan sebagai garis dengan simbol berlian di ujungnya.
4. Komposisi 	Komposisi merupakan relasi yang lebih kuat dari asosiasi dan agregasi. Komposisi dilambangkan menggunakan garis dengan simbol berlian warna di ujungnya.

Ada 3 tingkatan *visibility*, yaitu :

Tabel 2. 2 Tingkatan Visibility

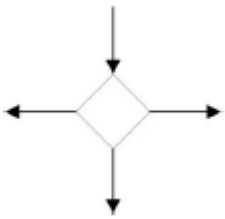
1. *Public*, dengan simbol "+"	*Public attribute* dapat diakses dan *public method* dapat dipanggil oleh *method* di *class* lain.
2. *Protected*, dengan simbol "#"	*Protected attribute* dapat diakses dan *protected method* dapat dipanggil oleh *method* di *class* yang atribut atau *method*-nya mendefinisikan atau merupakan *subclass* dari *class* itu sendiri.
3. *Private*, dengan simbol "-"	*Private attribute* dan *private method* hanya dapat dipanggil oleh *method* dari *class* itu sendiri.

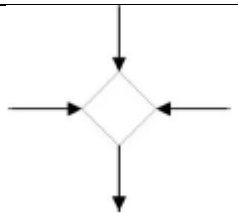
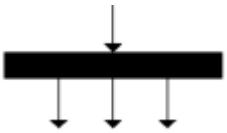
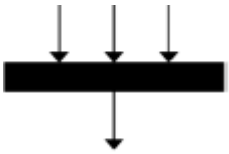
2.8.3.*Activity* Diagram

Activity diagram menggambarkan berbagai alur aktifitas dalam sistem yang sedang dirancang, bagaimana masing-masing alur dimulai, pilihan yang mungkin terjadi dan bagaimana mereka berakhir. *Activity* diagram juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. (Whitten *and* Bentley, 2007, p. 319)

Activity diagram merupakan *state* diagram khusus yang sebagian besar *action* dan transisi di-*trigger* oleh *state* sebelumnya (*internal processing)*. Oleh karena itu, *activity* diagram tidak menggambarkan perilaku internal sebuah sistem dan interaksi antar subsistem secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum. Ada 10 komponen dari *activity* diagram, yaitu :

Tabel 2. 3 Komponen Activity Diagram

	1. *Initial node*	Lingkaran pada mewakili awal dari proses.
	2. *Actions*	*Action* untuk menunjukkan suatu aksi atau aktifitas dari *user* dan sistem yang berbentuk bulat lonjong.
	3. *Flow*	*Flow* berbentuk anak panah untuk menunjukkan jalur dari satu aktifitas ke aktifitas lainnya.
	4. *Decision*	*Decision* berbentuk berlian dengan satu aliran masuk dan dua atau lebih arus keluar. Arus keluar ditandai untuk menunjukkan kondisi aktifitas berikutnya.
	5. *Merge*	*Merge* berbentuk berlian dengan dua atau lebih

		aliran masuk dan satu aliran keluar. Digunakan untuk menggabungkan *flow* dari aktifitas yang sebelumnya terpisah oleh *decision*.
	6. *Fork*	*Fork* berbentuk bar hitam dengan satu aliran masuk dan dua atau lebih arus keluar. Digunakan saat aktifitas berjalan bersamaan.
	7. *Join*	*Join* berbentuk bar hitam dengan dua atau lebih arus masuk dan satu aliran keluar, Digunakan untuk menutup aktifitas paralel yang sudah berakhir.
	8. *Activity Final*	*Activity final* berbentuk lingkaran pada di dalam lingkaran berlubang mewakili akhir proses.

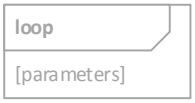
2.8.4.*Sequence* Diagram

Sequence diagram menggambarkan bagaimana objek berinteraksi antara satu sama lainnya melalui pesan dalam eksekusi *use case* atau operasi. (Whitten *and* Bentley, 2007, p. 394).

Diagram ini mengilustrasikan bagaimana pesan dikirim dan diterima antara objek dan bagaimana urutannya. Diagram ini lebih fokus pada detail aliran data, termasuk data yang dikirim ataupun data yang diterima. *Sequence* diagram terbentuk dari beberapa notasi, antara lain :

Tabel 2. 4 Notasi Sequence Diagram

	1. *Actor*	Aktor yang terdapat pada *use case* sebagai simbol aktor.
	2. *Controller class*	Kotak yang mengindikasikan sistem sebagai *"black box"*. Titik dua (:) standar notasi pada *sequence* diagram yang mengidentifikasikan contoh sebuah sistem berjalan.
	3. *Lifelines*	Garis horizontal yang memberikan simbol hubungan antara aktor dengan sistem.
	4. *Activation bars*	Diagram ini menunjukkan aktifitas-aktifitas dari *lifeline* yang berpartisipasi dalam sistem bergantung satu sama lain untuk berinteraksi.
	5. *Messages*	Baris horizontal dari aktor ke sistem dalam bentuk pesan masuk.
	6. *Return messages*	Baris horizontal dari sistem ke aktor sebagai garis berjalan / pesan keluar.
	7. *Self-message*	Sebuah object dapat

		memanggil *method*-nya sendiri dan mengembalikan pesan ke object itu sendiri.
	8. *Frame*	Kotak yang dapat menutup satu atau lebih pesan *divide off fragment* dari setiap langkah.

2.9. *Object Oriented Programming* (OOP)

Menurut Kendal (2009, p. 20), *Object Oriented Programming* adalah sebuah Bahasa pemrograman berbasis objek dan datanya lebih ke arah *logic*. *Object oriented programming* melihat bagaimana cara **memanipulasi objek** dibandingkan bagaimana **membuat *logic*** untuk memanipulasi objek. OOP terdiri dari objek dan *class*.

Object adalah kumpulan perangkat lunak yang terdiri dari *variables* dan *methods*. *Variables* adalah suatu data yang diberikan nama oleh sebuah *identifier*. *Methods* adalah suatu fungsi yang berkaitan dengan sebuah objek. Sebuah objek juga dikenal dengan sebutan *instance*. Sebuah *instance* dapat mengacu pada sebuah objek tertentu. *Class* adalah sebuah *blueprint* yang mendefinisikan *variables* dan *methods* yang umum untuk semua objek dari jenis tertentu. *Class* mendeklarasikan *instance* dari *variables* yang dibutuhkan untuk menampung *state* dari seluruh objek. *Class* juga akan mendeklarasikan dan menyediakan implementasi dari *instance methods* yang dibutuhkan untuk beroperasi pada *state* dari objek. (Pilay, 2007, p. 16, p. 65, p.83).

Beberapa fungsi dalam *object oriented programming* adalah :

1. *Encapsulation*, sebuah metode dalam OOP yang dapat membuat sebuah kelas tergabung dalam kelas lainnya tetapi masing-masing kelas dianggap berbeda.
2. *Information Hiding*, memungkinkan sebuah objek dapat memiliki *public interface* yang objek lain dapat gunakan untuk berinteraksi dengan objek tersebut, sehingga objek dapat menjaga informasi yang bersifat *private* dan metode dapat diganti kapanpun tanpa mempengaruhi objek yang bergantung kepadanya.

3. *Modularity,* sebuah metode dimana *source code* untuk sebuah ojek dapat ditulis dan dijaga secara terpisah dari *source code* untuk objek yang lain. Objek juga dapat dengan mudah beredar melalui sistem.
4. *Inheritance*, metode data kelas yang memungkinkan sebuah *subclass* memiliki kesamaan dengan kelas utama, yang memudahkan analisis data, mengurangi waktu pengembangan dan memungkinkan pembuatan *code* lebih akurat.
5. *Polymorphism*, metode menambahkan fungsi baru ke dalam *class*. Fungsi yang ditambah bisa sedikit maupun banyak dan diproses secara general.
6. *Overloading*, kemampuan untuk mendefinisikan beberapa fungsi dengan nama yang sama dalam sebuah *class* selama fungsi tersebut memiliki parameter yang berbeda.
7. *Overriding*, terjadi bila sebuah *subclass* memiliki fungsi yang dengan *super class* namun implementasi fungsi *subclass* tersebut berbeda dengan *super class*-nya.

2.10. HTML

Menurut Hidayatullah (2014, p. 13), HTML (*Hypertext Markup Language*) adalah bahasa standar yang digunakan untuk menampilkan halaman *web* dan bahasa yang digunakan untuk mendesain kebanyakan halaman *web*. HTML adalah sistem untuk *marking-up, tagging* sehingga dokumen tersebut dapat dipublikasikan ke *web*. Hal-hal yang dapat dilakukan dengan HTML, yaitu:

1. Mengatur tampilan halaman *web* dan isinya
2. Membuat *table* dan halaman *web*
3. Mempublikasikan halaman *web* secara *online*
4. Membuat *form* yang dapat digunakan untuk menangani registrasi dan transaksi via *web*
5. Menambahkan objek-objek seperti gambar, audio, video dan animasi dalam halaman *web*
6. Menampilkan area gambar (canvas) di *browser*.

2.11. PHP

PHP *Hypertext Processor* atau disingkat PHP adalah suatu bahasa *scripting* khususnya digunakan untuk *web deployment*. Karena sifatnya yang *server side scripting*, maka untuk menjalankan PHP harus menggunakan *web server*. PHP juga dapat diintegrasikan dengan HTML, JavaScript, JQuery dan Ajax. Namun, pada

umumnya PHP lebih banyak digunakan bersamaan dengan *file* yang bertipe HTML. PHP sendiri dapat melakukan tugas-tugas yang dilakukan dengan mekanisme CGI seperti mengambil, mengumpulkan data dari *database*, meng-*generate* halaman dinamis atau bahkan menerima dan mengirim *cookie*. CGI (*Common Gateway Interface*) adalah suatu mekanisme yang berjalan di *web server*, bertugas untuk melayani komunikasi dua arah antara *web server* dan *web browser*. Keunggulan dari PHP sendiri adalah dapat digunakan dalam berbagai *operating system* diantaranya Linux, Unix, Windows, Mac OsX, RICS OS dan *operating system* lainnya. *Server side scripting* pada PHP dapat bekerja bila ada tiga komponen berikut : PHP Parser (CGI atau *server* modul), *web server* (contohnya Apache dalam XAMPP), *web browser*. *Output* PHP yang melewati *web server* dapat dilihat pada *web browser*. PHP mampu menghasilkan gambar sebagai *output, file* bertipe PDF bahkan Flash. (Hidayatullah dan Kawistara, 2014, p. 231).

2.12. JavaScript

JavaScript adalah suatu bahasa *scripting* yang digunakan sebagai fungsionalitas dalam membuat suatu *web*. *JavaScript* digunakan untuk pemrosesan di komputer pengguna (*client-side*). Kode *JavaScript* dapat dimasukkan ke dalam bagian *head* maupun *body* dari dokumen HTML. *JavaScript* mengabaikan spasi dan bersifat *case sensitive*. *JavaScript* dapat disisipkan pada bahasa lainnya seperti HTML tanpa harus berada dalam satu *file* HTML. (Hidayatullah dan Kawistara, 2014, p. 81).

2.12.1 *jQuery*

jQuery adalah *library* atau kumpulan kode *JavaScript* yang cepat, kecil dan kaya akan fitur. *jQuery* dapat membuat hal-hal seperti *traversal* dan manipulasi *file* HTML, *event handling*, animasi, dan Ajax yang lebih sederhana dengan API yang mudah dan dapat digunakan dibanyak *browser*.

Manfaat dari *jQuery* antara lain (Sigit, A., 2011) :

1. *jQuery* dapat digunakan dibanyak *browser*
2. *jQuery* mendukung semua versi CSS
3. Ukuran *jQuery* sangat kecil
4. Dokumentasi *jQuery* lengkap
5. Dukungan komunitas terhadap *jQuery*.

2.13. *Cascading Styles Sheets* (CSS)

CSS (*Cascading Styles Sheets*) merupakan sebuah bahasa pemrograman *web* yang dapat mengontrol format tampilan sebuah halaman *web* yang ditulis dengan menggunakan penanda. Umumnya CSS itu sendiri digunakan pada halaman HTML.

Menurut Hidayatullah dan Kawistara (2014, p. 54), CSS digunakan dalam memformat halaman *web* agar terlihat lebih bagus dimanapun *web* tersebut dibuka.

2.13.1. Bootstrap

Bootstrap merupakan sebuah *framework* CSS yang menyediakan kumpulan komponen-komponen antarmuka dasar dalam membangun *website*. Selain komponen antarmuka, *Bootstrap* juga menyediakan sarana untuk membangun *layout* halaman dengan menarik dan *responsive*. *Bootstrap* merupakan *CSS-Driven* yang dapat memasukkan sejumlah *plugin JavaScript* dan *icons* yang berjalan seiringan dengan *forms* dan *buttons*. Hal tersebut dapat membantu *frontend-webdevelopment* untuk terus maju dan berkembang dalam membangun landasan yang stabil mengenai pandangan ke depan tentang desain dan *development*. (Spurlock, J., 2013)

2.14. *MySQL*

Menurut DuBois (2008, p. 1), *MySQL* adalah suatu *relational database management system* (RDBMS) *client/sercer* yang bersifat *open-source* berasal dari Skandinavia. *MySQL* menggunakan bahasa *query* standar yang dimiliki SQL (*Structured Query Language*). *MySQL* dikenal sebagai RDBMS yang cepat dan sederhana. Pada awalnya, *MySQL* dinilai rendah karena kekurangan fitur pada dukungan akan *foreign key*. *MySQL* pun dikembangkan dengan lebih lanjut sehingga banyak fitur ditambahkan seperti, *replication, subquery, store procedure, view* dan *trigger*. *MySQL* dapat berjalan pada berbagai sistem operasi, seperti Mac OsX, HP-UX, Windows dan pada berbagai *hardware*.

2.14.1 *phpMyAdmin*

phpMyAdmin adalah salah satu aplikasi GUI (*Graphical User Interface*) yang digunakan untuk mengelola *database MySQL*. (Arief, 2011, p. 429).

Menurut Hidayatullah dan Kawistara (2014, p. 184), *phpMyAdmin* adalah *tool open source* yang ditulis dalam bahasa PHP untuk menangani administrasi *MySQL* berbasis *World Wide Web*. Kegunaan *phpMyAdmin* adalah yaitu, membuat *database* dan *table*, mengisi *table*, mengedit data *table*,

menghapus data *table*, menghapus (*drop*) *table*, mengubah struktur *table* serta ekspor dan impor data.

2.15. *Laravel*

Laravel adalah sebuah *framework* PHP MVC yang dikembangkan oleh Taylor Otwell pada tahun 2011 yang lisensinya dipegang oleh MIT yang *framework*-nya menggunakan bahasa pemrograman PHP dan berbasis *Model-View-Controller*. (David, 2014).

Menurut McCool (2012, p. 3), *Laravel* adalah *framework* pengembangan *web MVC* yang ditulis dalam PHP. *Laravel* adalah salah satu dari beberapa kerangka bahasa pemrograman PHP yang menawarkan *code modular*. Hal ini dicapai melalui kombinasi *driver* dan sistem *bundle*-nya. Untuk mengubah dan memperluas *caching, session, database* dan fungsi otentikasi dapat dengan mudah dilakukan dengan *driver*.

Framework ini menekankan kesederhanaan dan fleksibilitas pada desainnya sehingga pengguna *framework* ini meningkat dari tahun ke tahun.

Menurut Aminudin (2015, p. 5), beberapa fitur yang dimiliki oleh *framework Laravel* adalah sebagai berikut.

1. *Bundles*

   Sebuah fitur dengan sistem pengemasan modular dan berbagai *bundle* telah tersedia untuk digunakan dalam aplikasi.

2. *Eloquent ORM*

   Penerapan PHP lanjutan dari pola "*active record"* menyediakan metode internal untuk mengatasi kendala hubungan antara objek *database*. Pembangunan *query Laravel Fluent* didukung *Eloquent*.

3. *Application Logic*

   Bagian dari aplikasi yang dikembangkan, baik menggunakan *controllers* maupun sebagai bagian dari deklarasi *route*. Sintaks yang digunakan untuk mendefinisikannya mirip dengan yang digunakan oleh *framework* Sinatra.

4. *Reverse Routing*

   Mendefinisikan hubungan antara *Link* dan *Route*, sehingga bila terjadi perubahan pada *route* maka secara otomatis akan tersambung dengan *link* yang relevan. Ketika *link* yang dibuat dengan menggunakan nama-nama dari *route* yang ada, secara otomatis *Laravel* akan membuat URL yang sesuai.

5. *Restful Controllers*

   Memberikan sebuah pilihan untuk memisahkan logika dalam melayani *HTTP GET* dan permintaan *POST*.

6. *Class Auto Loading*

   Menyediakan *loading* otomatis untuk *class-class* PHP tanpa membutuhkan pemeriksaan manual terhadap jalur masuknya. Fitur ini mencegah *loading* yang tidak diperlukan.

7. *View Composers*

   Kode unit *logical* yang dapat dijalankan ketika sebuah *view* di-*load*.

8. *IoC Container*

   Memungkinkan objek baru yang dihasilkan mengikuti prinsip *control* pembalik dengan pilihan contoh dan referensi dari objek baru sebagai *singletons*.

9. *Migrations*

   Menyediakan versi sistem *control* untuk skema *database* sehingga dapat menghubungkan perubahan basis kode aplikasi dan keperluan yang dibutuhkan dalam merubah tata letak *database*, mempermudah penempatan dan memperbarui aplikasi.

10. *Unit Testing*

    Mempunyai peran penting dalam *framework Laravel*, dimana *unit testing* ini mempunyai banyak tes untuk mendeteksi dan mencegah regresi. *Unit testing* dapat dijalankan melalui fitur ”artisan *command-line*”.

11. *Automatic Pagination*

    Menyederhanakan tugas dari penerapan halaman, menggantikan penerapan manual dengan metode otomatis yang terintegrasi ke *Laravel*.

Menurut Aminudin (2015, p. 4), keuntungan *framework Laravel* adalah sebagai berikut.

1. Ekspresif

   Laravel adalah *framework* PHP yang ekspresif, maksudnya ketika melihat suatu sintaks *Laravel*, seorang programmer dapat langsung mengetahui kegunaan dari sintaks tersebut meskipun belum pernah menggunakannya. Dua buah kode berikut memiliki tujuan yang sama tetapi ditulis dengan gaya yang berbeda.

```
//kode 1, framework xxx
        $uri = Uri::create('some/uri'.array(),array(),true);
//kode 2, framework Laravel
        $url = URL::to_secure('some/uri');
```

Laravel melakukan pendekatan yang berbeda dengan membuang parameter yang bersifat *flagging* dan memilih untuk membuat dua fungsi yang berbeda.

2. Simple

   Salah satu fitur yang membuat *Laravel* begitu simple adalah adanya *Eloquent ORM*. Misalnya, ketika semua data yang ada di dalam *table users* ingin diambil, maka hanya perlu membuat sebuah *class* model bernama *user:* dan memasukkan semua data dari *table users* dengan cara sebagai berikut.

   ```
   $all_user = User::all();
   ```

   Dengan begitu, semua data dari *table users* dapat dengan mudah diakses dengan melakukan *looping* terhadap *variable $all_user*.

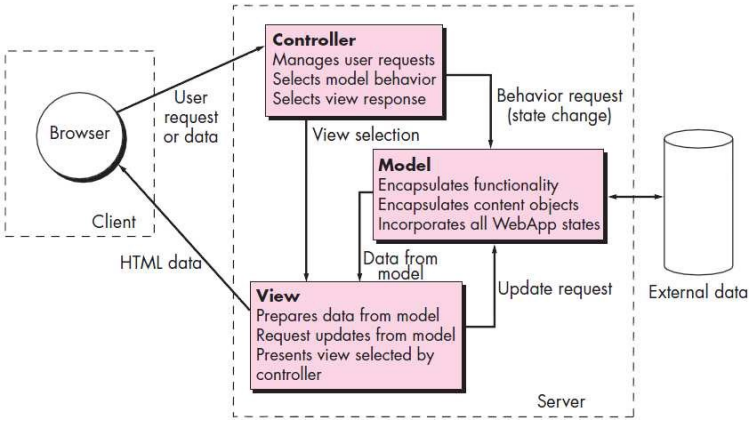
3. *Accesible*

   Laravel dibuat dengan dokumentasi yang selengkap mungkin. *Code Developer* dari *Laravel* sendiri berkomitmen untuk selalu menyertakan dokumentasi yang lengkap setiap kali versi terbarunya dirilis.

2.16. *Model-View-Controller* (MVC)

Menurut Pressman (2010, p. 386-387), MVC adalah sebuah metode dalam pembuatan atau pengembangan aplikasi yang membuat aplikasi tersebut menjadi 3 komponen utama, yaitu *model, view,* dan *controller*.

1. *Model*, adalah bagian paling dalam dari MVC yang berfungsi sebagai letak dimana *database* dan tempat *code* berada. *Model* hanya menerima perintah dari *controller*, sehingga tidak berinteraksi secara langsung kepada *user*.
2. *Controller,* adalah bagian dalam MVC yang menjadi penghubung antara *model* dan *view*. *Controller* berisi perintah-perintah yang dapat dijalankan oleh *user* dan mengirim perintah tersebut ke *model* yang kemudian akan memberikan *output* melalui *view*.
3. *View*, adalah *interface* dari *model* program MVC yang berfungsi untuk menampilkan *output-output* dari *model* melalui *controller* kepada *user*.



Gambar 2.10 Arsitektur MVC

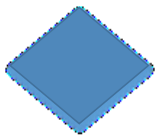
2.17. *Flow*chart

Flowchart adalah jenis diagram (grafis atau simbolik) yang mewakili suatu algoritma atau proses. Setiap langkah dalam prosesnya diwakili oleh simbol yang berbeda dan berisi deskripsi singkat mengenai langkah proses yang dilakukan. Simbol-simbol pada *flowchart* saling terhubung dengan anak panah yang menunjukkan arah aliran proses. *Flowchart* biasanya menunjukkan aliran data dalam suatu proses dan menjelaskan detail suatu langkah atau operasi dalam format gambar yang lebih mudah dimengerti daripada format teks.

Flowchart biasanya digambarkan dengan simbol-simbol standar, namun terdapat simbol khusus yang dapat digunakan bila diperlukan. Beberapa simbol standar yang sering digunakan dalam pembuatan *flowchart,* yaitu:

Tabel 2. 5 Flowchart Symbols

Symbols	*Symbols Name*	*Description*
	terminator	Menandakan dimulai atau diakhirinya suatu proses, biasanya diberikan keterangan "*start*" atau "*end*"
	process	Menunjukkan sebuah langkah proses atau operasi. Umumnya, menggunakan kata kerja dalam deskripsi yang singkat dan jelas

	*decision*	Menunjukkan sebuah langkah pengambilan keputusan. Umumnya, menggunakan bentuk pertanyaan dan jawabannya terdiri dari 'yes' atau 'no' yang menentukan bagaimana alur dalam *flowchart* berjalan selanjutnya berdasarkan kriteria atau pertanyaan tersebut
	On-page reference	Menunjukkan hubungan simbol dalam *flowchart* sebagai pengganti garis untuk menyederhanakan bentuk saat simbol yang akan dihubungkan jaraknya berjauhan dan rumit jika dihubungkan dengan garis
	Input / output	Menunjukkan data yang menjadi *input* atau *output* proses
	document	Menunjukkan proses atau keberadaan dokumen
	subprocess	Menunjukkan bahwa dalam langkah yang dimaksud terdapat *flowchart* lain yang menggambarkan langkah tersebut dengan lebih rinci
	connector	Menunjukkan arah aliran dari satu proses ke proses yang lain
	Off-page reference	Fungsinya sama dengan *connector*, akan tetapi digunakan untuk menghubungkan simbol-simbol

		yang berada pada halaman yang berbeda

2.18. Penelitian Terkait

Kajian penelitian 1 : Analisis dan Perancangan Aplikasi *Point of Sales* yang Terhubung dengan *Electronic Data Capture* (Petrus Hartono, 2009), dengan tujuan untuk melakukan pengembangan aplikasi *Point of Sales* (POS) yang terhubung dengan *Electronic Data Capture* untuk mengurangi kesalahan yang terjadi dalam penggunaan POS dan EDC dalam pembayaran dengan kartu.

Kajian penelitian 2 : Perancangan Aplikasi *Point of Sales* Berbasiskan *Customer Relationship Management* pada Toko Buku Notre-Dame (Gintoro, 2010), dengan tujuan untuk memberikan beberapa solusi bagi masalah yang dihadapi Toko Buku Notre-Dame, seperti pencatatan transaksi yang lebih mudah, pengelolaan data pelanggan yang lebih rapi dan tidak mungkin hilang, serta pelaporan yang lebih praktis dan terperinci.

Kajian penelitian 3 : Pengaruh Penerapan Aplikasi *Point of Sales* (POS) Berbasis Komputer Terhadap Kecepatan Proses Transaksi Penjualan dan Pembelian (Haris Klana Sanif, 2010), dengan tujuan untuk memberikan bantuan dalam pengambilan keputusan dan menentukan langkah-langkah yang akan ditempuh oleh perusahaan dalam melaksanakan aktivitas proses penjualan dan pembelian dengan baik dan benar.

Kajian penelitian 4 : Perancangan Sistem Informasi *Point of Sales* (POS) pada PD. Tokyo (Novita, 2010), dengan tujuan untuk memberikan bantuan dalam mempermudah, mempercepat dan mengurangi tingkat kesalahan dalam pengelolaan baik data penjualan, pembelian, stok, *cash flow* dan laporan laba rugi.