BAB II
TINJAUAN REFERENSI

2.1 *Web Application*

Menurut Pressman (2010, hal. 8), *Web Application* disebut "*WebApps*", kategori perangkat lunak jaringan-sentris (*network-centric*) ini mencakup beragam aplikasi. Dalam bentuknya yang paling sederhana, *WebApps* bisa menjadi sedikit lebih dari satu set *file hypertext* yang menyajikan informasi menggunakan teks dan grafis terbatas. Namun, saat *Web* 2.0 muncul, *WebApps* berkembang menjadi lingkungan komputasi yang canggih yang tidak hanya menyediakan fitur yang berdiri sendiri, fungsi komputasi, dan konten ke pengguna akhir, namun juga terintegrasi dengan *database* perusahaan dan aplikasi bisnis.

2.1.1 HTML

Menurut Brooks (2007, hal. 1), HTML adalah akronim untuk *HyperText Markup Language*. Dokumen-dokumen HTML, dasar semua konten yang muncul di *World Wide Web* (WWW), terdiri dari dua bagian penting: konten informasi dan satu set petunjuk yang memberitahu komputer bagaimana cara menampilkan konten itu. Instruksi-instruksi "*Markup*", dalam jargon editorial-terdiri dari bahasa HTML. Ini bukan bahasa pemrograman dalam pengertian tradisional, melainkan satu set petunjuk tentang cara menampilkan konten. Aplikasi komputer yang menerjemahkan deskripsi ini disebut *Web browser*. Idealnya, *online* konten harus selalu terlihat sama terlepas dari *browser* yang digunakan atau sistem operasi yang ditempatinya, namun tujuan *platform independence* hanya tercapai kira-kira dalam praktek.

2.1.2 CSS

Menurut Meloni (2012, hal. 46), teknologi di balik *style sheets* disebut CSS, yang merupakan singkatan dari *Cascading Style Sheets*. CSS adalah bahasa yang mendefinisikan konstruksi *style* seperti *font*, warna, dan posisi, yang digunakan untuk menggambarkan bagaimana informasi pada halaman *web* diformatkan dan ditampilkan. *Style* CSS bisa disimpan langsung di halaman *web* HTML atau di dalam *file style sheet* terpisah. Sebaiknya, *style*

sheet berisi *style rules* yang menerapkan *style* pada elemen dari *type* yang diberikan. Bila digunakan secara eksternal, *style sheet rules* akan ditempatkan dalam *external style sheet* dokumen dengan ekstensi *file* .css.

Style rules adalah instruksi pemformatan yang dapat diterapkan pada elemen halaman *web*, seperti paragraf teks atau *link*. *Style rules* terdiri dari satu atau lebih banyak sifat *style* dan nilai asosiasinya. *Internal Style Sheet* ditempatkan langsung di dalam halaman *web*, sedangkan *External Style Sheet* ada di dokumen terpisah dan hanya terhubung ke halaman *web* melalui *tag* khusus - lebih pada *tag* ini dalam sekejap.

2.1.3 PHP

Menurut Tatroe, MacIntyre & Lerdorf (2013, hal. 1), PHP adalah bahasa sederhana namun kuat yang dirancang untuk membuat konten HTML. PHP dapat digunakan dalam tiga cara utama:

1. *Server-side scripting*

   PHP pada awalnya dirancang untuk membuat konten *web* dinamis, dan PHP masih paling sesuai untuk tugas itu. Untuk menghasilkan HTML, memerlukan PHP *parser* dan *web server* untuk mengirimkan kode dokumen. PHP juga menjadi populer untuk menghasilkan dokumen XML, grafik, animasi Flash, *file* PDF, dan sebagainya.

2. *Command-line scripting*

   PHP dapat menjalankan *scripts* dari *command line*, seperti Perl, awk, atau the Unix shell. Anda bisa menggunakan *command-line scripts* untuk tugas administrasi sistem, seperti *backup* dan *log parsing*; Bahkan beberapa *script* jenis pekerjaan CRON bisa dilakukan dengan cara ini (*non-visual* PHP *tasks*).

3. *Client-side Graphical User Interface* (GUI) *applications*

   Dengan menggunakan PHP-GTK, Anda bisa menulis *full-blown*, aplikasi GUI *cross-platform* di PHP.

2.1.4 Javascript

Menurut Meloni (2012, hal. 66), Javascript dikembangkan oleh *Netscape Communications Corporation*, pembuat *web browser* Netscape. JavaScript adalah bahasa *web scripting* pertama yang akan didukung oleh *browser*, dan

masih jauh yang paling populer. Javascript hampir mudah dipelajari seperti HTML dan bisa disertakan langsung di dokumen HTML. Beberapa hal yang dapat dilakukan dengan Javascript:

a. Menampilkan pesan ke *user* sebagai bagian dari halaman *web*, melalui *browser's status line*, atau di *alert boxes*.
b. Validasi isi formulir dan membuat perhitungan (misalnya, formulir pemesanan secara otomatis dapat menampilkan jumlah total seperti memasukkan jumlah *item*)
c. Animasi gambar atau membuat gambar yang dapat berubah saat memindahkan *mouse* ke atas gambar.
d. Membuat *ad banners* yang dapat berinteraksi dengan *user*, daripada hanya menampilkan sebuah grafik.
e. Mendeteksi *browser* yang digunakan atau fiturnya dan melakukan fungsi lanjutan hanya pada *browser* yang mendukung Javascript.
f. Mendeteksi *plug-ins* yang terpasang dan memberitahu *user* jika *plug-in* diperlukan.
g. Mengubah semua atau sebagian halaman *web* tanpa mengharuskan *user* meng-*reload* (memuat kembali) halaman *web*.
h. Menampilkan atau berinteraksi dengan data yang diambil dari *remote server*.

2.1.5 CodeIgniter

Menurut CodeIgniter (2017), CodeIgniter adalah *framework* berbasis PHP yang dibangun bagi *developer* yang membutuhkan *toolkit* untuk menciptakan *web application* yang elegan dan sederhana. CodeIgniter dibuat oleh EllisLab berdasarkan konsep MVC.

Konsep MVC ini memisahkan tiga buah elemen, yaitu logika, tampilan, dan pengendali sebagai berikut:

a. *Model*

   Komponen pusat yang mengatur data, logika, dan aturan dari aplikasi.

b. *View*

   Komponen yang dapat dilihat dan berhubungan langsung dengan *user*.

c. *Controller*

Komponen penerima *input* dan mengubahnya menjadi perintah bagi *model* atau *view*.

2.1.6 jQuery

Menurut jQuery (2017), jQuery adalah *library* Javascript yang cepat, kecil, dan kaya fitur yang dapat dipakai untuk memanipulasi dokumen HTML, *event-handling*, animasi, dan AJAX menjadi lebih sederhana, mudah digunakan, dan dapat dipakai pada semua *web browser*. Dengan fleksibilitasnya, jQuery telah banyak mengubah cara orang dalam membuat Javascript.

2.1.7 Bootstrap

Bootstrap adalah *framework* CSS yang menyediakan banyak elemen *user interface*, *layout*, dan jQuery *plugins*. Bootstrap bersifat *open source* dan juga salah satu *framework* yang paling banyak digunakan pada *Github*. Bootstrap juga memberi hasil terbaik karena bersifat responsif yang memungkinkan kita untuk desain bagi berbagai macam ukuran layar (Niska, 2014, hal. 5).

2.1.8 JSON

Berdasarkan JSON (2017), JSON (JavaScript *Object Notation*) adalah format pertukaran data yang ringan atau *lightweight*. JSON mudah untuk dibaca dan ditulis oleh manusia dan mudah untuk dikonversi dan dibuat oleh komputer. JSON dibuat berdasarkan bahasa pemrograman JavaScript, *Standard* ECMA-262 3rd *Edition* – Desember 1999.

JSON adalah sebuah format teks yang tidak dipengaruh bahasa pemrograman apapun karena dapat digunakan dalam bahasa – bahasa yang umum digunakan oleh *programmer* seperti bahasa C, C++, C#, Java, JavaScript, Perl, Python, dan lainnya.

2.1.9 *Web Server*

Menurut Brooks (2007, hal. 168), *Web Server* adalah komputer yang terhubung ke internet yang menyimpan dokumen dan konten lainnya untuk

akses global (namun tidak harus publik) dengan cara alamat yang unik (*Uniform Resource Locator* atau URL).

Menurut Meloni (2012, hal. 5), *web server* bertugas menafsirkan atau menerjemahkan *request* (permintaan), menemukan *file* dan mengirimkan *content* dari *file* tersebut ke *web browser* yang melakukan *request* (permintaan) terhadapnya.

2.1.10 AJAX

Menurut Ferguson & Heilmann (2013, hal. 248), AJAX merupakan kepanjangan dari *Asynchronous* Javascript and XML. Metode yang digunakan berbeda dari metode tradisional. Dimana *web* tradisional bekerja secara *synchronous* antara aplikasi dan *server*, setiap kali *submit form* maka *browser* mengirimkan data ke *server*, *server member* respons dan seluruh halaman akan *reload / refresh.*

Aplikasi *web* yang menggunakan AJAX bekerja secara *Asynchronouly*, yang berarti mengirim dan menerima data dari *user* ke *server* tanpa perlu meng-*load* semua halaman, hanya mengganti bagian tertentu yang ingin diubah. AJAX membuat aplikasi menjadi lebih baik, cepat, *user friendly*, dan lebih interaktif.

2.1.11 *Web Browser*

Menurut Jasmadi (2008), *Web browser* adalah perangkat lunak untuk membuka halaman *web* (*World Wide Web*). Dengan *web browser*, *user* bisa mengakses informasi dari berbagai *website* dengan cepat dan mudah. Ada beberapa macam jenis *web browser* yang digunakan saat ini seperti Google Chrome dan Mozilla Firefox.

2.1.12 Internet

Menurut Connolly & Begg (2015, hal. 1048), internet adalah kumpulan jaringan komputer yang terkoneksi di seluruh dunia. Internet terdiri dari banyak jaringan yang terpisah namun saling berhubungan ke organisasi komersial, pendidikan, dan pemerintahan serta *Internet Service Provider* (ISP). Layanan yang ditawarkan di Internet termasuk surat elektronik (*e-*

mail), konferensi dan layanan kolaborasi, serta kemampuan untuk mengakses *remote computers*, mengirim dan menerima *file*.

2.1.13 XML

Menurut Sunyoto (2007, hal. 160), XML (*Extensible Markup Language*) *Document Object Model* adalah sebuah standar cara mengakses dan memanipulasi dokumen XML. XML membentuk struktur pohon dengan elemen, atribut dan teks yang didefinisikan sebagai *node*.

XML memiliki isi yang dapat diubah, dihapus, dan ditambahkan elemen baru. Semua *node* dalam XML berbentuk *Tree*. *Tree* dimulai dari *node* dokumen dan berlanjut ke cabang sampai level *tree* paling bawah. Hubungan *parent* and *child* digunakan untuk menerangkan hubungan antar *node*.

2.2 *Database*

Menurut Connolly & Begg (2015, hal. 63), *database* adalah kumpulan data logis yang saling terkait dan deskripsinya dirancang untuk memenuhi kebutuhan informasi sebuah organisasi.

Menurut Satzinger, Jackson, dan Burd (2012, hal. 373), *database* (DB) adalah kumpulan data yang tersimpan, terintegrasi, serta dikelola dan dikendalikan secara terpusat. *Database* biasanya menyimpan informasi tentang puluhan atau ratusan kelas. *Database* dikelola dan dikendalikan oleh *Database Management System* (DBMS).

Database terdiri dari dua penyimpanan informasi yang saling terkait, yaitu *psychical data store* dan *schema*. *Psychical data store* berisi *raw bits* dan *bytes* dari data yang dibuat dan digunakan dalam sistem. *Schema* berisi informasi deskriptif tentang data yang tersimpan di dalam *psychical data store*, termasuk:

a. Organisasi individual dari data *item* yang disimpan kedalam kelompok yang tingkatnya lebih tinggi seperti tabel.
b. Asosiasi antara tabel atau *class* (Contoh: petunjuk dari *objects* pelanggan untuk *objects* penjualan terkait)
c. Rincian organisasi *psychical data store*, termasuk tipe, panjang, lokasi, dan *index* dari data *item.*

d. Akses dan kontrol dari konten termasuk *values* yang diperbolehkan untuk data *items* tertentu, dependensi *value* antara beberapa data *items* dan daftar pengguna yang diperbolehkan untuk membaca atau melakukan *update* data *items*.

2.2.1 *Database Management System* (DBMS)

Menurut Satzinger, Jackson, dan Burd (2012: hal.373), *Database Management System* (DBMS) adalah komponen sistem perangkat lunak yang umumnya dibeli dan dipasang terpisah dari komponen sistem perangkat lunak lainnya (misalnya, sistem operasi). Contoh *Modern Database Management Systems* meliputi Microsoft SQL Server, Oracle, dan MySQL.

Menurut Connolly & Begg (2015, hal. 64), DBMS adalah sistem perangkat lunak yang memungkinkan *user* untuk mendefinisikan, membuat, memelihara, dan mengkontrol akses ke *database*.

2.2.2 *Data Definition Language* (DDL)

Menurut Connolly & Begg (2015, hal. 90), *Data Definition Language* (DDL) adalah bahasa yang memungkinkan *Database Administrator* (DBA) atau *user* untuk menggambarkan dan memberi nama entitas, atribut, dan hubungan yang dibutuhkan untuk aplikasi, bersama dengan integritas yang terkait dan kendala keamanan. DDL digunakan untuk mendefinisikan skema atau memodifikasi yang sudah ada. DDL Tidak bisa digunakan untuk memanipulasi data.

2.2.3 *Data Manipulation Language* (DML)

Menurut Connolly & Begg (2015, hal. 90), *Data Manipulation Language* (DML) adalah bahasa yang menyediakan satu set operasi untuk mendukung operasi manipulasi data dasar pada data yang tersimpan dalam *database*. Operasi manipulasi data meliputi:

a. Penyisipan data baru ke dalam *database*.
b. Modifikasi data yang tersimpan dalam *database.*
c. Pengambilan data yang terkandung/berada dalam *database*.
d. Penghapusan data dari *database*.

2.2.4 *Stored Procedure*

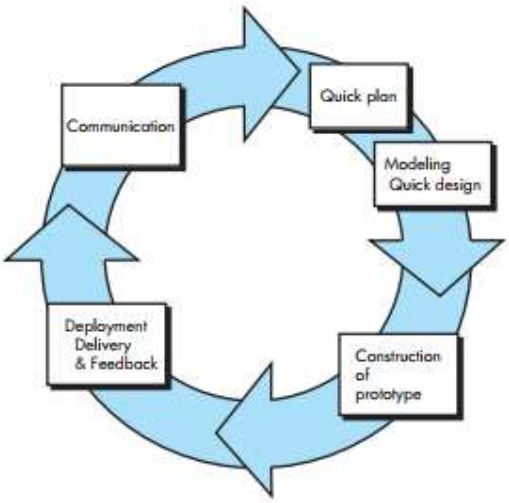
Menurut Whitten & Bentley (2007, hal. 528), *stored procedure* adalah program yang dimasukkan ke dalam tabel yang dapat dipanggil dari program aplikasi. Misalnya, algoritma validasi data mungkin dimasukkan ke dalam tabel untuk memastikan bahwa catatan baru dan yang diperbarui berisi data yang valid sebelum disimpan. *Stored procedure* ditulis dalam ekstensi *proprietary* dari SQL seperti Microsoft Transact SQL atau Oracle's PL/SQL.

2.3 *Prototyping Model*

Menurut Pressman & Maxim (2015, hal. 45-46), seringkali *customer* sudah memiliki gambaran besar *software* yang akan dibuat, tetapi tidak mendefinisikan fungsi dan fitur secara terperinci. Hal tersebut akan membuat tim *development* menjadi tidak yakin fungsi dan fitur apa saja yang harus dibuat. *Prototyping development* adalah metode terbaik dalam kasus seperti ini.

Walaupun *prototyping* dapat dipakai sebagai model proses independen, *prototyping* umumnya digunakan sebagai teknik yang dapat diimplementasikan dalam konteks mendukung para *stakeholders* untuk mengerti apa yang harus dibuat ketika *requirement* masih tidak jelas.

Paradigma *prototyping* dimulai dari komunikasi. *Developer* akan bertemu dengan *stakeholder* lain untuk mendefinisikan gambaran umum dari *software* yang akan dibuat, mengidentifikasi kebutuhan, dll. Sebuah *prototype* akan direncanakan dengan cepat dan didesain. Desain yang cepat berfokus kepada representasi dari aspek – aspek sebuah *software* yang akan terlihat oleh *end user*. *Prototype* akan diluncurkan dan dievaluasi oleh *stakeholder*, yang akan memberikan komentar yang akan digunakan untuk *requirement* yang lebih lanjut. Pengulangan akan terjadi ketika *prototype* disesuaikan dengan kebutuhan para *stakeholder*, ketika dalam waktu bersamaan memungkinkan para *developer* untuk mengerti apa yang harus dilakukan.



Gambar 2.1 *Prototyping development*

(Sumber: Pressman & Maxim, 2015, hal. 46)

Baik para *stakeholder* maupun *developer* menyukai paradigma *prototyping*. *User* dapat merasakan sistem secara aktual, dan *developer* dapat menciptakan sesuatu dengan segera. Walaupun begitu, *prototyping* dapat bermasalah seperti:

1. *Stakeholder* melihat versi apa yang sedang *developer* kerjakan, dan tidak menyadari apakah *prototype* digabungkan secara sembarangan, juga tidak menyadari saat terburu-buru para *developer* tidak terlalu memerhatikan kualitas dan pemeliharaan jangka panjang.
2. *Developer* sering mengimplementasikan kompromi untuk mengerjakan *prototype* dengan cepat. Sistem operasi atau bahasa pemrograman yang tidak sesuai dapat dipakai karena dikuasai dan dapat digunakan oleh *developer*. Algoritma yang tidak efisien dapat diimplementasikan karena dapat mendemonstrasikan kapabilitas. Setelah beberapa waktu, *developer* dapat menjadi nyaman dengan pilihan ini dan melupakan semua alasan mengapa hal – hal tersebut harus dihindari.

Walau masalah dapat terjadi, *prototyping* dapat menjadi paradigm efektif untuk pengembangan *software*. Kuncinya adalah mendefinisikan aturan main dari awal, yaitu para *stakeholder* harus setuju bahwa *prototype* dibuat untuk mendefinisikan *requirement*. Selanjutnya *prototype* tidak boleh dipakai lagi dan *software* sebenarnya harus dibuat dengan mementingkan kualitas.

Tahapan-tahapan dalam *Prototyping* adalah sebagai berikut:

1. *Communication*

   Pada tahap ini dilakukan pengumpulan informasi kebutuhan yang diharapkan oleh *user* dan ruang lingkup dari aplikasi yang akan dibuat.

2. *Quick Design*

   Pada tahap ini akan dilakukan diskusi untuk menentukan fitur-fitur apa saja yang perlu dibuat.

3. *Modelling Quick Design*

   Pada tahap ini dilakukan pembuatan sketsa *user interface* aplikasi berdasarkan diskusi yang dilakukan pada tahap sebelumnya.

4. *Construction of Prototype*

   Pada tahap ini, aplikasi akan mulai dikembangkan melalui koding berdasarkan hasil sketsa *user interface* yang telah dibuat pada langkah sebelumnya.

5. *Deployment, Delivery, Feedback*

   Pada tahap ini, *user* akan melakukan pengujian aplikasi *prototype* yang telah selesai dibuat dalam beberapa aspek seperti fungsionalitas. Setelah aplikasi selesai diuji coba pada tahap sebelumnya dan sudah dipastikan berjalan lancar, aplikasi akan diluncurkan ke halaman Binusmaya yang sebenarnya untuk mulai digunakan secara nyata. Jika *user* menginginkan perubahan atau penambahan fitur, proses akan kembali ke nomor satu.

Berikut dua macam *testing* yang umum digunakan:

2.3.1 *Black-Box Testing*

Menurut Pressman & Maxim (2015, hal. 499-500), *black-box testing* yang sering disebut sebagai *behavioral testing* atau *functional testing* adalah salah satu metode uji coba *software* untuk mengetahui fungsi spesifik yang ditampilkan dari sebuah produk dengan cara mendemonstrasikan setiap fungsi untuk beroperasi sepenuhnya ketika dalam waktu yang sama mencari *error* pada setiap fungsi.

Testing ini berfokus kepada *requirement* fungsional dari sebuah *software*. *Testing* ini memungkinkan kita untuk menghasilkan beberapa kondisi *input* yang akan menjalankan semua *requirement* fungsional dari sebuah program. *Black-box testing* bukanlah alternatif dari *white-box testing* melainkan salah

satu cara untuk mendeteksi *error* dengan cara yang berbeda dari *white-box testing*.

Black-box testing menyinggung kepada uji coba yang dilakukan pada *software interface*. *Testing* ini memeriksa beberapa aspek fundamental dari sistem dan juga struktur logika internal dari sebuah *software*.

Black-box testing mencari *error* dari kategori tersebut:

a. Fungsi yang tidak tepat atau tidak ada
b. *Interface error*
c. *Error* pada struktur data atau akses *external database*.
d. *Behavior error* atau *performance error*
e. *Initialization error* atau *termination error*

Menurut Pressman & Maxim (2015, hal. 509), tidak seperti *white-box testing*, yang dilakukan di awal proses uji coba, *black-box testing* dilakukan di akhir tahap *testing*. Dikarenakan *black-box testing* mengabaikan struktur kontrol, perhatiannya ditujukan kepada domain informasi. Uji coba didesain untuk menjawab pertanyaan-pertanyaan berikut:

a. Bagaimana validasi fungsional diuji?
b. Bagaimana *system behavior* dan *system performance* diuji?
c. *Input* apa saja yang akan menjadikan uji coba efektif?
d. Apakah sistem sensitif terhadap *input* tertentu?
e. Bagaimana ruang lingkup dari kelas data?
f. *Data rates* dan *data volume* apa saja yang bisa diterima sistem?
g. Pengaruh apa saja yang dapat dihasilkan oleh kombinasi spesifik dari sebuah sistem operasi?

2.3.2 *White-Box Testing*

Menurut Pressman & Maxim (2015, hal. 499-500), *white-box testing* yang sering disebut sebagai *glass-box testing* atau *structural testing* adalah salah satu metode uji coba *software* untuk mengetahui cara kerja internal dari sebuah produk dengan cara memastikan operasi internal dijalankan menurut spesifikasi dan semua komponen internal sudah dicoba. *Testing* ini menggunakan *control structure* yang dideskripsikan sebagai bagian dari *component-level design* untuk menghasilkan *test case*.

White-box testing memprediksi pemeriksaan prosedur secara lebih terperinci. Alur logika dari sebuah *software* dan kolaborasi antar komponen diuji dengan mencoba beberapa kondisi tertentu atau dilakukan hal yang sama berulang kali.

Dalam pandangan ini, mungkin terlihat *white-box testing* selalu dilakukan agar *software* menjadi 100% sempurna. Yang harus dilakukan adalah mendefinisikan alur logika, membuat *test case*, dan mengevaluasi hasil untuk menguji logika program tersebut secara mendalam. Sayangnya, *testing* yang mendalam akan membuat beberapa masalah logistik tertentu. Bahkan untuk program kecil, jumlah alur logika yang memungkinkan dapat sangat besar.

Dengan metode *white-box testing*, kita dapat membuat *test case* yang seperti berikut:

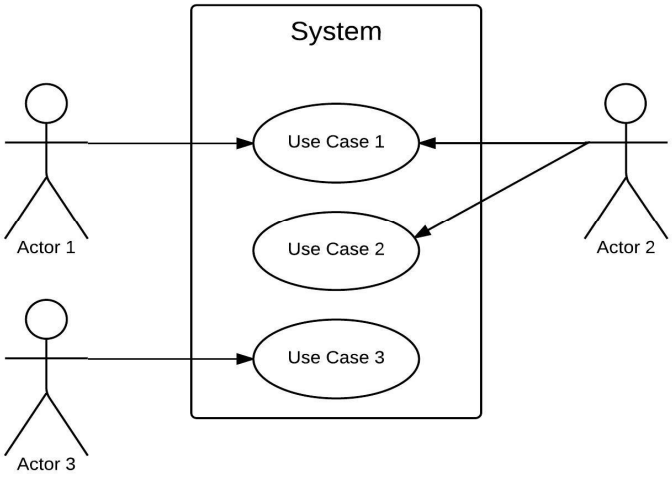
a. Menjamin semua alur independen dalam sebuah modul dijalankan minimal satu kali.
b. Menjalankan semua pilihan logika baik pilihan benar maupun pilihan salah.
c. Mengeksekusi semua perulangan dalam operasi
d. Menjalankan struktur internal data untuk memastikan validasi.

2.4 UML (*Unified Modeling Language*)

Menurut Whitten & Bentley (2007, hal. 381), *Unified Modeling Language* (UML) dapat digambarkan sebagai *blueprint* dalam membangun sebuah rumah. *Blueprint* menyediakan perspektif untuk pengaturan listrik, air, pemanas, dll bagi orang yang membangun rumah tersebut. UML pun berperan seperti *Blueprint*. UML diagram menyediakan berbagai perspektif sistem informasi bagi tim *development*.

2.4.1 *Use Case*

Use-case modelling mengidentifikasi dan menggambarkan fungsi sistem dengan menggunakan *use case*. *Use case* menggambarkan fungsi dari perspektif *user* eksternal dan terminologi yang dapat dimengerti. Dibutuhkan keterlibatan *user* dan ahli di bidang subjek yang sesuai agar dapat memenuhi permintaan *user* secara akurat dan berhasil.



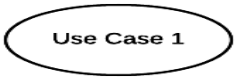
Gambar 2.2 *Use Case Diagram*

(Sumber: Whitten dan Bentley, 2007, hal. 246)

Sebuah *use case* merepresentasikan satu tujuan dari sistem dan menggambarkan interaksi *user* dalam mencapai tujuan. Penggunaan *use case* sudah terbukti sebagai teknik yang sangat baik dalam memberi pengertian dan *requirement* sistem yang diperlukan.

Komponen – komponen dalam *use case* adalah sebagai berikut:

a. *Use Case*

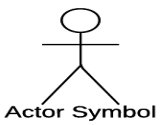


Gambar 2.3 Simbol *Use Case*

(Sumber: Whitten dan Bentley, 2007, hal. 246)

Use case digunakan untuk menggambarkan fungsi dari sebuah sistem dilihat dari sudut pandang *user* dengan menggunakan bahasa yang mudah dipahami.

b. *Actor*



Gambar 2.4 Simbol *Actor*

(Sumber: Whitten dan Bentley, 2007, hal. 247)

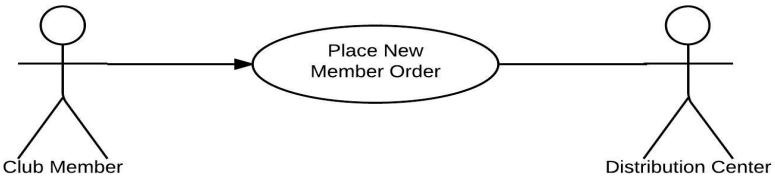
Actor menggambarkan *user* eksternal yang menginisiasi dan memicu *use case* untuk menyelesaikan sebuah tugas bisnis yang menghasilkan keuntungan. *Actor* tidak harus manusia, tetapi bisa juga untuk menggambarkan organisasi, sistem informasi lainnya, *external device* seperti *heat sensor*, bahkan konsep waktu.

c. *Relationship*

Relationship digambarkan sebagai garis antara dua simbol pada diagram *use case*. Arti dari sebuah *relationship* bisa berbeda tergantung bagaimana garis digambar dan simbol apa yang terhubung.

1. Associations

Sebuah hubungan antara *actor* dengan *use case* ketika *use case* mendeskripsikan interaksi di antaranya. *Relationship* ini disebut *association*. *Association* dapat berupa gambar anak panah yang ujungnya ada di *use case* yang mengindikasikan *use case* diimitasi oleh *actor* yang berhubungan atau dapat juga berupa sebuah garis yang mengindikasikan interaksi antara *use case* dan *external server* atau *actor* penerima.

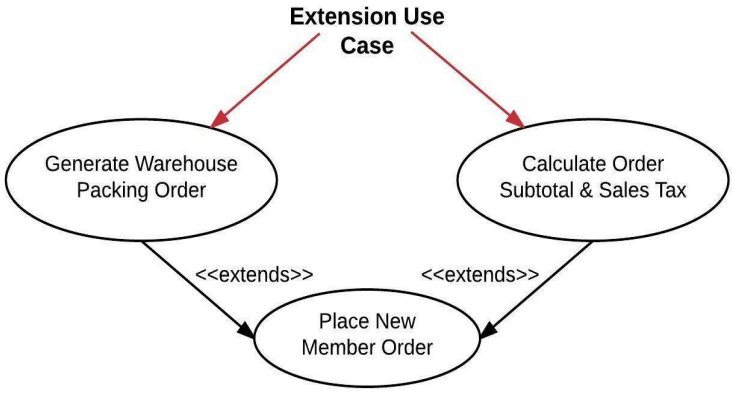


Gambar 2.5 *Association Relationship*

(Sumber: Whitten dan Bentley, 2007, hal. 248)

2. Extends

Use case dapat berisi fungsi kompleks yang terdiri dari beberapa langkah yang membuat logika *use case* sulit dimengerti. Maka dibuatlah *use case* yang disebut *extension* untuk menambah langkah kompleks dari *use case* sebelumnya. *Relationship* antara *use case* dengan *extension* disebut *extends relationship*.

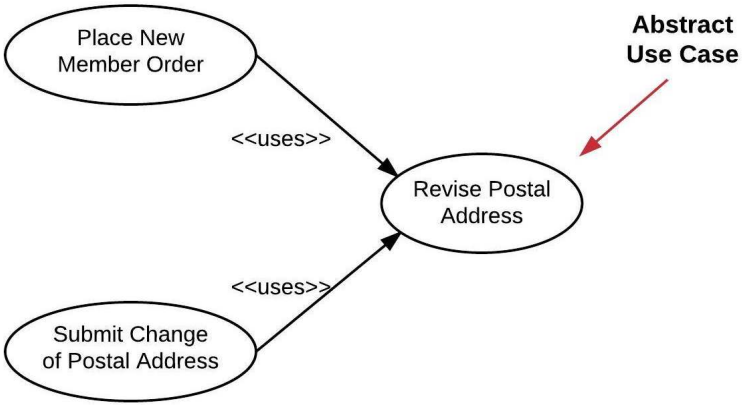


Gambar 2.6 *Extends Relationship*

(Sumber: Whitten dan Bentley, 2007, hal. 249)

3. *Uses* (*Includes*)

Tidak jarang ditemukan dua atau lebih *use case* melakukan langkah yang mirip secara fungsionalitas. Akan lebih baik untuk membagi kedua *use case* tersebut dengan yang disebut *abstract use case*. *Abstract use case* merepresentasikan bentuk daur ulang dan merupakan cara yang baik untuk mengurangi redundansi. *Relationship* antara *abstract use case* dan *use case* disebut *uses relationship*.

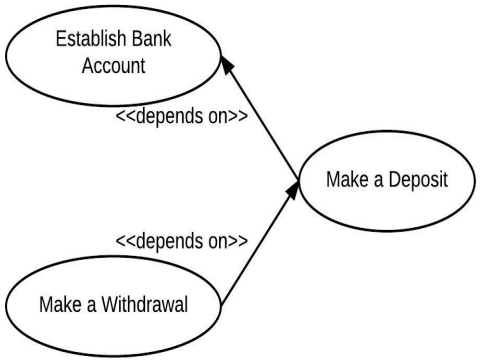


Gambar 2.7 *Uses Relationship*

(Sumber: Whitten dan Bentley, 2007, hal. 249)

4. *Depends On*

Sebagai *project manager* atau *lead developer*, sangat membantu untuk mengetahui *use case* mana yang bergantung dengan *use case* lain untuk menentukan urutan *use case* mana yang harus dibuat. Diagram *use case* mendesain ketergantungan *use case* sistem menggunakan *depends*. Ini akan menyediakan model yang baik untuk perencanaan dan penjadwalan.

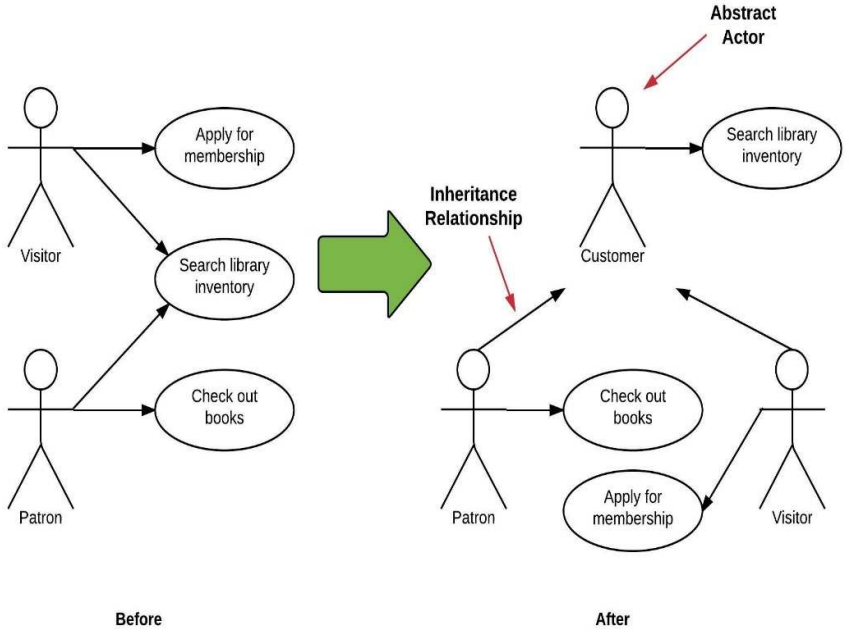


Gambar 2.8 *Depends On Relationship*

(Sumber: Whitten dan Bentley, 2007, hal. 250)

5. *Inheritance*

Ketika dua atau lebih *actor* mempunyai *behavior* yang sama, tidak mustahil untuk *actor – actor* tersebut menginisiasi *use case* yang sama. Ada baiknya untuk memprediksi kejadian seperti ini dan memakai *abstract actor* untuk mengurangi redundansi komunikasi dengan sistem.



Gambar 2.9 *Inheritance Relationship*

(Sumber: Whitten dan Bentley, 2007, hal. 250)

2.4.2 *Use Case Narrative*

Menurut Whitten & Bentley (2007, hal. 256-257), *use case narrative* merupakan dokumentasi naratif *requirement* dari setiap *use case* yang bersifat *high level* untuk mengerti *event* dan ukuran sistem secara cepat.

Komponen-komponen suatu *use case narrative* adalah sebagai berikut:

a. *Author*

Nama dari individu yang membuat *use case* dan menjadi alamat tujuan ketika ada orang yang ingin menanyakan informasi lebih lanjut tentang *use case* yang dibuat.

b. *Date*

Tanggal terakhir *use case* dimodifikasi.

c. *Use-case name*

Nama *use case* harus merepresentasikan tujuan yang ingin dicapai oleh *use case.*

d. *Use-case type*

Business requirement adalah tipe yang *business-oriented* dan menunjukkan sudut pandang dari yang diinginkan pada sistem. *Business*

requirement use case menyediakan pengertian umum dari masalah dan ruang lingkup tetapi tidak termasuk hal teknis.

e. *Use-case ID*

Identifikasi unik untuk membedakan *use case* yang satu dengan yang lainnya,

f. *Priority*

Menunjukkan tingkat kepentingan dari *use case*, bisa *high, medium,* atau *low*.

g. *Source*

Mendefinisikan entitas yang memicu dibuatnya use case. *Source* dapat berupa *requirement,* dokumen, atau *stakeholder*.

h. *Primary business actor*

Merupakan *stakeholder* yang mendapat keuntungan dari eksekusi *use case* dengan menerima suatu nilai.

i. *Other Participating actors*

Aktor lain yang ada dalam use case untuk mencapai tujuan meliputi *initiating actor, facilitating actor, server/receiver actor,* dan *secondary actor*.

j. *Interested stakeholder(s)*

Aktor yang mempunyai ketertarikan akan tujuan dari *use case*.

k. *Description*

Deskripsi singkat yang mendefinisikan tujuan dari use case dan aktivitas yang dilakukan.

l. *Trigger*

Event yang menginisiasi eksekusi dari *use case.*

m. *Typical Course of Events*

Urutan dari aktivitas yang dilakukan aktor dan sistem untuk mencapai tujuan *use case*.

n. *Alternate Courses*

Urutan aktivitas lain jika *typical course* gagal dilaksanakan.

Member Services System

Author (s) : _____ **Date Version : _____**

Use-Case Name :	Place New Order	**Use-Case Type** **Business Requirements** : √
Use-Case ID :	MSS-BUC002.00	
Priority :	High	
Source :	Requirement – MSS-R1.00	
Primary Business Actor :	Club member	
Other Participating Actors :	- Warehouse (external receiver) - Accounts Receivable (external server)	
Other Interested Stakeholders :	- Marketing - Interested in sales activity in order to plan new promotions . - Procurement - Interested in sales activity in order to replenish inventory .	
Description :	This use case describes the event of a club member submitting a new order for SoundStage products. The members demographic information as well as his or her account standing is validated. Once the products are verified as being in stock, a packing order is sent to the warehouse for it to prepare the shipment. For any product not in stock, a back order is created. On completion, the member will be sent an order confirmation.	
Trigger :	This use case is initiated when a new order is submitted	
Typical Course of Events :	**Actor Action**	**System Response**
	Step 1 : The club member provides his or her demographic information as well as order and payment information	Step 2: The system responds by verifying that all required information has been provided Step 3: The system verifies the club member's demographic information against what has been previously recorded Step 4: For each product ordered, the system validates the product identity. Step 5: For each product o rdered, the system verifies tte p10duct availability, Step 6: For each available product, the system deterrnires the price to be charged to the club member, Step 7: Once all ordered products are processed, the system determines the total cost of the order, Step 8: The system checks the status of the club member's account. Step 9: The system validates the club member's payment if provided Step 10: The System records the order information and then releases the order to the appropriate distribution center (warehouse) to be filled. Step 11: Once the order is processed, the system geneates a1l order confirmation and sends it to the club member.
Alternate Courses :	Alt· Step 2: The club member has not provided all the information necessary to process the order. The club member is notified of the discrepancy and prompted to resubmit.	

Gambar 2.10 *Use Case Narrative*

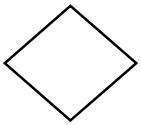
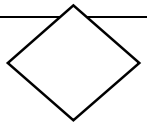
(Sumber: Whitten dan Bentley, 2007, hal. 259)

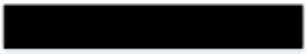
2.4.3 *Activity Diagram*

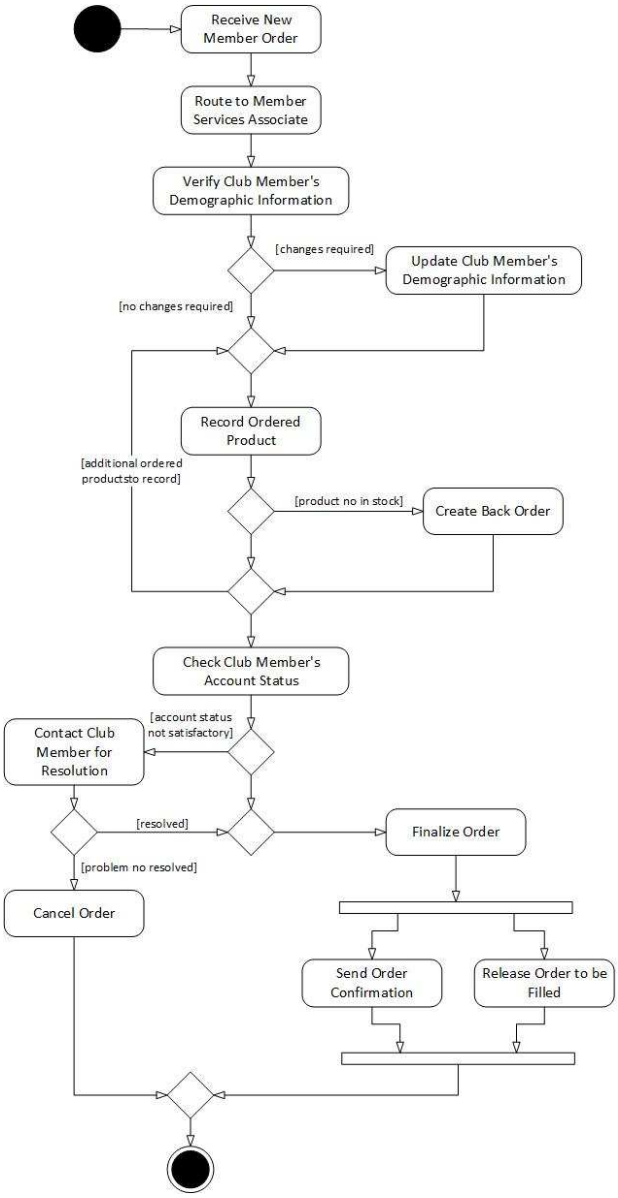
Menurut Whitten & Bentley (2007, hal. 390-391), *activity diagram* digunakan untuk menggambarkan langkah proses dan aktivitas pada sistem. *Activity diagram* mirip dengan *flowchart* karena sama-sama menggambarkan alur dan aktivitas. Perbedaannya adalah *activity diagram* menyediakan mekanisme yang menggambarkan aktivitas yang berlangsung secara paralel. *Activity diagram* dapat menggambarkan aksi yang dijalankan ketika operasi sedang berjalan dan juga hasil dari aksi tersebut.

Komponen-komponen dalam *activity diagram* adalah sebagai berikut:

Tabel 2.1 Komponen Activity Diagram

Nama	Simbol	Deskripsi
Initial node		Lingkaran hitam yang melambangkan awal proses
Actions		Persegi panjang yang melambangkan aksi atau langkah yang dilakukan
Flow		Panah yang melambangkan arah berjalannya sistem
Decision		Belah ketupat dimana ada satu *flow* yang masuk dan adanya dua atau lebih *flow* yang keluar
Merge		Belah ketupat dimana ada dua atau lebih *flow* yang masuk dan adanya satu *flow* yang keluar
Fork		Persegi panjang hitam dimana satu *flow* masuk dan dua atau lebih *flow* yang keluar. *Flow – flow* yang keluar dari *fork* dapat berjalan secara *concurrent*

Join		Persegi panjang dimana dua atau lebih *flow* masuk dan satu *flow* keluar. Digunakan untuk mengakhiri proses *concurrent*
Activity Final		Lingkaran hitam dalam lingkaran kosong yang melambangkan akhir dari proses
Subactivity Indicator		Simbol sapu dalam *action* yang melambangkan bahwa aksi tersebut terpecah ke dalam *activity diagram* lain yang terpisah
Connector		Sebuah huruf dalam sebuah lingkaran untuk mengatur kompleksitas dimana satu *flow* datang ke *connector* dan berpindah ke *connector* lain dengan huruf yang sama



Gambar 2.11 Contoh *Activity Diagram*

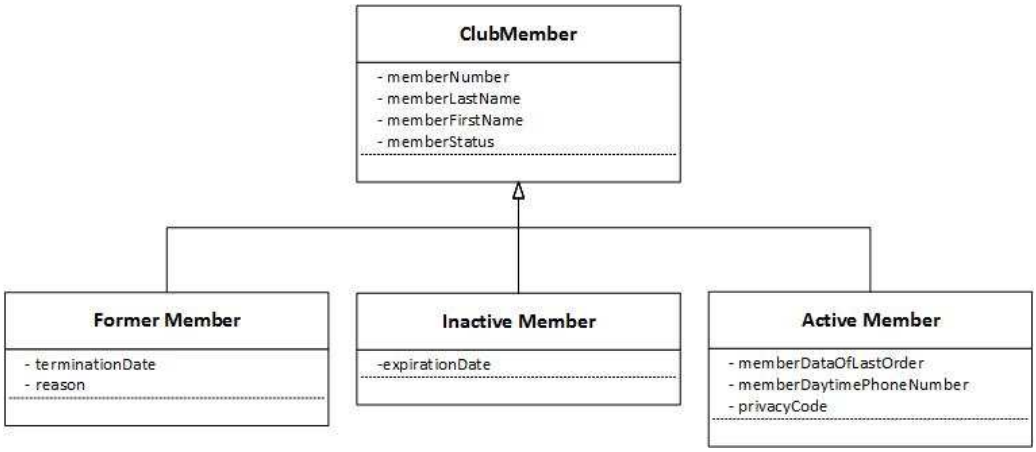
(Sumber: Whitten dan Bentley, 2007, hal. 392)

2.4.4 *Class Diagram*

Menurut Whitten & Bentley (2007, hal. 400) *Class Diagram* merupakan gambaran grafis mengenai struktur objek statis dari suatu sistem, dan menunjukkan kelas-kelas objek yang menyusun sebuah sistem dan juga hubungan antar kelas objek tersebut.

Sebuah *class* diwakili oleh bentuk kotak di dalam *class diagram.* Dalam bentuk paling sederhana, kotak tersebut dapat berisi nama kelasnya saja,

namun juga dapat berisi *attribute* dan *method*. *Attribute* merupakan apa yang kelas tahu sebagai ciri khas dari sebuah objek, dan *method* atau *operations* adalah apa yang *class* tahu mengenai bagaimana melakukan sebuah tindakan.



Gambar 2.12 *Class Diagram*

(Sumber: Whitten dan Bentley, 2007, hal. 406)

3 jenis relasi pada *class diagram* antara lain :

1. *Association and Multiplicity*

   Association adalah sebuah relasi antara 2 objek *class* yang merepresentasikan apa yang suatu objek butuh untuk mengetahui objek lainnya. *Multiplicity* adalah angka minimum dan maximum dari suatu objek *class* untuk sebuah aksi dari objek *class* yang berhubungan (Whitten & Bentley, 2007 : hal. 400,378).

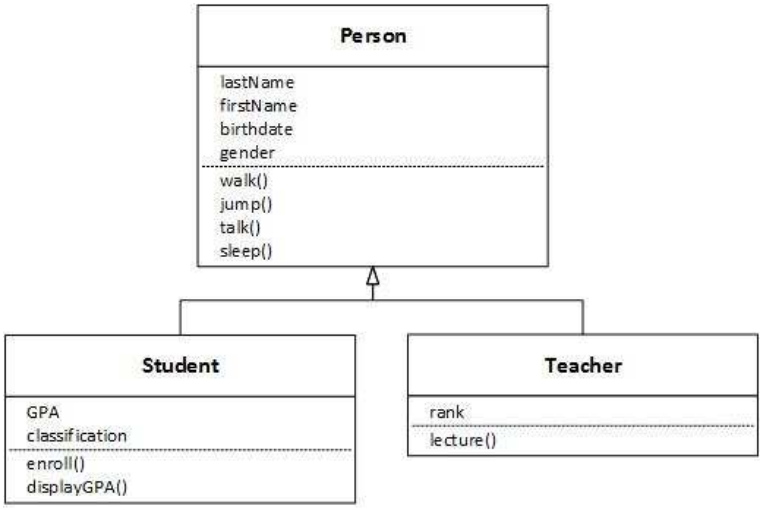
Gambar 2.13 *Association and Multiplicity*

(Sumber: Whitten dan Bentley, 2007, hal. 377)

2. *Generalization / Specialization*

   Generalization / *specialization* adalah sebuah relasi yang juga dikenal sebagai *classification hierarchies* atau sebuah relasi. Hal tersebut terdiri dari *supertype* (*abstract* atau *parent*) *class* dan *subtype* (*concrete* atau *child*) *class*. *Class supertype* adalah *class general* yang mengandung atribut umum dan metode dari *hierarchy*. Di samping itu, *class subtype* adalah kelas spesialis yang mengandung atribut unik dan metode dari

objek, tetapi tetap mewarisi atribut *supertype* dan metode nya (Whitten & Bentley, 2007 : hal.400).

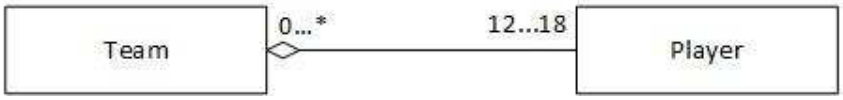


Gambar 2.14 *Generalization / Specialization*

(Sumber: Whitten dan Bentley, 2007, hal. 376)

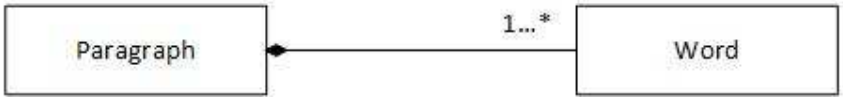
3. *Aggregation and Composition*

Aggregation adalah sebuah *unique type* dari relasi dimana sebuah objek adalah bagian dari objek lain. Relasi tersebut tidak simetris, dimana objek B adalah bagian dari objek A, tapi objek A bukan bagian dari objek B. *Composition* adalah bentuk yang lebih kuat dari *aggregation*. Ini adalah sebuah relasi *aggregation* dimana seluruh bagian bergantungan untuk pembuatan dan penghancuran bagian (Whitten & Bentley, 2007 : hal. 405, 378).



Gambar 2.15 *Aggregation*

(Sumber: Whitten dan Bentley, 2007, hal. 379)



Gambar 2.16 *Composition*

(Sumber: Whitten dan Bentley, 2007, hal. 379)

Dalam sebuah *class diagram*, tidak harus memasukkan sebuah atribut *primary key* kecuali apabila hal tersebut sebuah atribut bisnis yang nyata.

Dengan kata lain, sebuah *primary key* buatan yang mana biasanya direpresentasikan melalui sebuah *value* yang *auto increment* tidak akan masuk dalam *class diagram*. *Foreign key* juga akan ditolak dalam *class diagram*.

Menurut Whitten and Bentley (2007, hal. 650), *class diagram* memiliki tiga level *visibility*, yaitu:

a. *Public*

   Dinotasikan dengan simbol "+", atribut *public* dan *public method* dapat digunakan oleh semua *class* yang berhubungan .

b. *Protected*

   Dinotasikan dengan simbol "#", atribut *protected* dan *protected method* dapat digunakan oleh *class* itu sendiri dan *class* turunannya.

c. *Private*

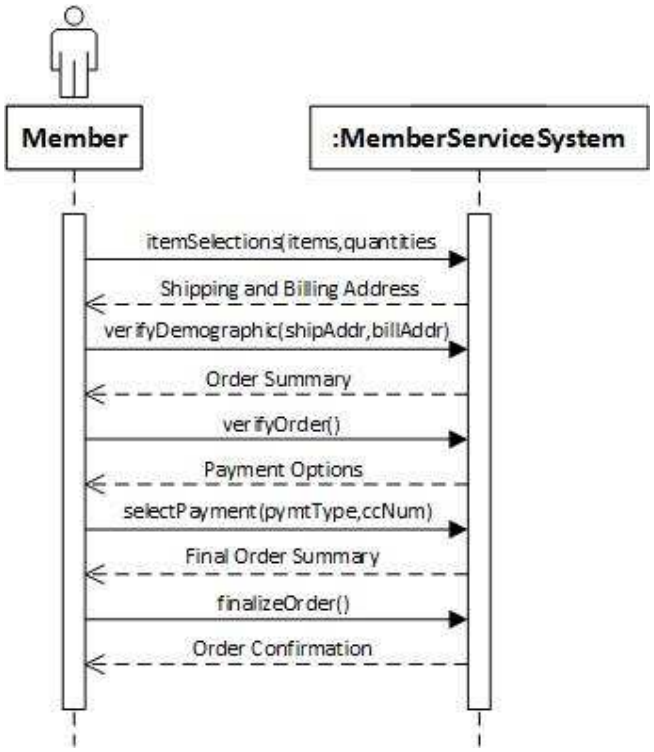
   Dinotasikan dengan simbol "-", atribut *private* dan *private method* hanya dapat digunakan oleh *class* itu sendiri.

4. *Dependency*

   Hubungan *depencdency* digunakan untuk memodelkan asosiasi antara dua kelas pada dua objek, pertama untuk mengindikasikan ketika suatu perubahan terjadi di sebuah kelas yang mempengaruhi kelas lain, kedua untuk mengindikasikan asosiasi antara kelas yang persisten atau sementara. Kelas *Interface* biasanya sementara dan dimodelkan dengan metode ini (Whitten & Bentley, 2007 : hal. 650).

2.4.5 *Sequence Diagram*

Menurut Menurut Whitten & Bentley (2007, hal. 395), *Sequence Diagram* adalah sebuah diagram yang menggambarkan interaksi antar objek yang diukur berdasarkan urutan kejadian dari kiri ke kanan.



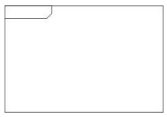
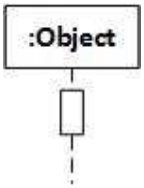
Gambar 2.17 *Sequence Diagram*

(Sumber: Whitten dan Bentley, 2007, hal. 395)

Berikut notasi *sequence diagram*:

Tabel 2.2 Notasi *Sequence Diagram*

Nama	Simbol	Deskripsi
Actor		Aktor merupakan simbol untuk mewakili pengguna dalam berinteraksi dengan *object class interface*
Object	:SavingAccount	*Object* merupakan simbol yang digunakan untuk mewakili kelas-kelas pada *class diagram.*
Interface Class	*<<interface>>*	Suatu notasi yang berfungsi untuk memastikan *user interface class code,*

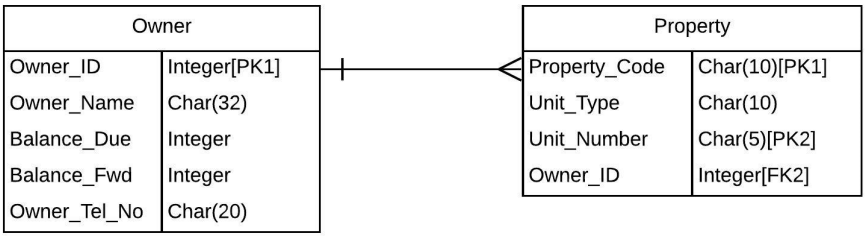
		untuk memastikan jenis *class*.
Controller Class	<<*controller*>>	Setiap *use case* memiliki satu atau lebih *controller*, digambarkan sama seperti notasi *interface class*.
Entity Class	:	Menambahkan kotak pada setiap entitas yang diperlukan untuk kolaborasi dalam urutan langkah-langkah.
Messages		Menggambarkan komunikasi antara sebuah *object* dengan *object* lain
Frame		Simbol yang digunakan menandakan area pada diagram yang melakukan perulangan, seleksi , dan suatu pilihan khusus.
Activation Bar		Berfungsi untuk menunjukkan berapa lamanya waktu *object* tersebut ada (*exists*).
Return Message		*Return message* yang dikembalikan ke *object* atau *actor* setelah melakukan suatu operasi atau sebagai tanggapan *message* yang dikirim sebelumnya

Self-Call		*Object* yang memanggil *method*-nya sendiri.

2.4.6 *Entity Relationship Diagram* (ERD)

Menurut Whitten & Bentley (2007, hal 271-275), *Entity Relationship Diagram* (ERD) adalah model data yang mengunakan beberapa notasi untuk menggambarkan data serta entitas dan hubungan yang dijelaskan oleh data tersebut. Berikut beberapa komponen yang ada pada ERD:

1. *Entity*: mempresentasikan sebuah objek (orang, tempat, objek, konsep atau kegiatan) tentang apa yang perlu diambil dan disimpan datanya. Contohnya adalah '*student*'
2. Atribut: bagian penting dalam *entity* yang merupakan deskripsi karakteristik dari *entity*. Contohnya adalah '*idStudent*', '*LastName*', '*FirstName*'
3. *Key* : sebuah variabel identitas unik yang ada pada *entity*. Tiap *entity* memiliki nilai *key* yang unik dan berbeda satu sama lain. Contohnya setiap mahasiswa memiliki NIM yang berbeda-beda.
4. *Relationship*: hubungan antara satu atau lebih *entity*. Contohnya adalah '*Student*' dengan '*Curriculum*'.



Gambar 2.18 *Entity Relationship Diagram*

(Sumber: Whitten dan Bentley, 2007, hal. 449)

2.5 Interaksi Manusia dan Komputer

Menurut Insap Santoso (2009, hal. 5), interaksi manusia dan komputer adalah sebuah disiplin ilmu yang mempelajari yang mempelajari perancangan, implementasi dan evaluasi sistem komputer interaktif untuk digunakan oleh manusia.

Menurut Shneiderman & Plaisant (2010, hal. 88-89), terdapat 8 aturan emas dalam merancang *interface* yang digunakan yaitu:

1. Konsistensi

   Tampilan yang ada dalam aplikasi harus konsisten artinya tidak berubah-ubah pada setiap halamannya baik dari segi warna, tata letak, ukuran, dan pemilihan jenis huruf.

2. Memenuhi kebutuhan universal

   Pemilihan *interface* yang dibuat harus mampu mewakili kebutuhan setiap pengguna. Terdapat perbedaan usia, pengalaman, dan pemahaman tentang teknologi.

3. Memberikan umpan balik yang informatif

   Untuk setiap aksi yang dilakukan oleh pengguna sebaiknya diberikan umpan balik yang informatif dan sederhana dari sistem yang mudah dimengerti oleh pengguna.

4. Design kotak dialog untuk menghasilkan keadaan akhir

   Urutan kotak dialog yang diberikan harus terorganisir secara teratur agar pengguna mengerti langkah yang harus diambil berikutnya, Contohnya adalah "*data has been saved*". Maka pengguna mengetahui bahwa proses telah selesai.

5. Menawarkan pencegahan masalah dan penanganan kesalahan sederhana

   Sistem dirancang agar mampu mendeteksi masalah dengan cepat agar mampu memberikan informasi yang jelas dan spesifik agar pengguna dapat memperbaiki kesalahannya.

6. Memberikan pengembalian aksi yang mudah

   Aplikasi dapat memberikan fitur sederhana untuk dapat kembali kepada keadaan sebelumnya. Contohnya penggunaan '*undo*' dan '*cancel*'.

7. Mendukung pusat kendali internal

   Pengguna memiliki kemampuan penuh dalam mengatur aplikasi dengan memberikan aksi yang dilakukan. Sistem dirancang agar pengguna merasa nyaman daripada merasa dikendalikan oleh sistem aplikasi.

8. Mengurangi beban ingatan jangka pendek

   Aplikasi sebaiknya dibuat sesederhana mungkin agar mudah diingat oleh pengguna. Sistem sebaiknya menyediakan data atau memberikan opsi untuk pengguna agar pengguna tidak perlu mengingat banyak data.

2.6 Kurikulum

2.6.1 Definisi Kurikulum

Menurut Hernawan & Susilana (2008, hal. 10), pengertian kurikulum diorganisir menjadi dua, kurikulum adalah sejumlah rencana isi yang merupakan sejumlah tahapan belajar yang didesain untuk siswa dengan pentunjuk institusi pendidikan yang isinya berupa proses yang statis ataupun dinamis dan kompetensi yang harus dimiliki. Selanjutnya, kurikulum adalah seluruh pengalaman dibawah bimbingan dan arahan dari institusi pendidikan yang membawa ke dalam kondisi belajar.

Konsep kurikulum meliputi sebagai substansi yang dipandang sebagai rencana pembelajaran bagi siswa atau seperangkat tujuan yang ingin dicapai, sebagai sistem merupakan bagian dari sistem persekolahan, pendidikan, dan bahkan masyarakat, dan sebagai bidang studi merupakan kajian para ahli kurikulum yang bertujuan untuk mengembangkan ilmu tentang kurikulum dan sistem kurikulum.

Istilah kurikulum menunjukan beberapa dimensi pengertian dimana setiap dimensi tersebut memiliki hubungan satu dengan yang lainnya. Keempat dimensi tersebut adalah:

a. Kurikulum sebagai suatu ide

   Pengertian kurikulum sebagai dimensi yang berkaitan dengan ide pada dasarnya mengandung makna bahwa kurikulum itu adalah sekumpulan ide yang akan dijadikan pedoman dalam pengembangan kurikulum selanjutnya. (Hernawan & Susilana, 2008: hal. 8).

b. Kurikulum sebagai suatu rencana

   Kurikulum ini sebenarnya merupakan perwujudan dari kurikulum sebagai suatu ide. Makna dari dimensi kurikulum ini adalah sebagai seperangkat rencana dan cara mengadministrasikan tujuan, isi, dan bahan pelajaran serta cara yang digunakan sebagai pedoman penyelenggaraan kegiatan pembelajaran untuk mencapai tujuan Pendidikan tertentu (Hernawan & Susilana, 2008: hal. 8).

c. Kurikulum sebagai aktivitas

   Kurikulum ini sering disebut dengan istilah kurikulum sebagai suatu realita atau implementasi kurikulum. Secara teoritis, dimensi kurikulum ini

adalah pelaksanaan dari kurikulum sebagai suatu rencana tertulis (Hernawan & Susilana, 2008: hal. 10).

d. Kurikulum sebagai hasil

Kurikulum ini merupakan konsekuensi dari kurikulum sebagai suatu aktivitas. Definisi kurikulum sebagai dimensi hasil memandang kurikulum itu sangat memperhatikan hasil yang akan dicapai agar sesuai dengan apa yang telah direncanakan dan yang menjadi tujuan dari kurikulum tersebut (Hernawan & Susilana, 2008: hal. 9).

Menurut Hernawan & Susilana (2008, hal. 12), pada dasarnya kurikulum itu berfungsi sebagai pedoman atau acuan. Bagi pengajar, kurikulum itu berfungsi sebagai pedoman dalam melaksanakan proses pembelajaran. Bagi kepala sekolah dan pengawas, kurikulum itu berfungsi sebagai pedoman dalam melaksanakan supervisi atau pengawasan. Bagi orang tua, kurikulum itu berfungsi sebagai pedoman dalam membimbing anaknya belajar dirumah. Bagi masyarakat, kurikulum itu berfungsi sebagai pedoman untuk memberikan bantuan bagi terselenggaranya proses pendidikan di sekolah. Bagi siswa, kurikulum berfungsi sebagai suatu pedoman belajar.

Kurikulum memiliki enam fungsi, yaitu:

a. Fungsi penyesuaian (*the adjustive or adaptive function*)

Fungsi penyesuaian mengandung makna bahwa kurikulum sebagai alat pendidikan harus mampu mengarahkan siswa agar memiliki sifat *well-adjusted*, yaitu mampu menyesuaikan dirinya dengan lingkungan, baik lingkungan fisik maupun lingkungan sosial. Lingkungan itu sendiri senantiasa mengalami perubahan dan bersifat dinamis. Oleh karena itu, siswa pun harus memiliki kemampuan untuk menyesuaikan diri dengan perubahan yang terjadi di lingkungannya.

b. Fungsi Integrasi (*the integrating function*)

Fungsi integrasi mengandung makna bahwa kurikulum sebagai alat pendidikan harus mampu menghasilkan pribadi-pribadi yang utuh. Siswa pada dasarnya merupakan anggota dan bagian integral dari masyarakat. Oleh karena itu, siswa harus memiliki kepribadian yang dibutuhkan untuk dapat hidup dan berintegrasi dengan masyarakatnya.

c. Fungsi Diferensiasi (*the differentiating function*)

Fungsi diferensiasi mengandung makna bahwa kurikulum sebagai alat pendidikan harus mampu memberikan pelayanan terhadap perbedaan individu siswa. Setiap siswa memiliki perbedaan, baik dari aspek fisik maupun psikis yang harus dihargai dan dilayani dengan baik.

d. Fungsi Persiapan (*the propaedeutic function*)

Fungsi persiapan mengandung makna bahwa kurikulum sebagai alat pendidikan harus mampu mempersiapkan siswa untuk melanjutkan studi ke jenjang pendidikan berikutnya. Selain itu , kurikulum juga diharapkan dapat mempersiapkan siswa untuk dapat hidup dalam masyarakat seandainya karena sesuatu hal, tidak dapat melanjutkan pendidikannya.

e. Fungsi Pemilihan (*the selective function*)

Fungsi pemilihan mengandung makna bahwa kurikulum sebagai alat pendidikan harus mampu memberikan kesempatan kepada siswa untuk memilih program-program belajar yang sesuai dengan kemampuan dan minatnya. Fungsi pemilihan ini sangat erat hubungannya dengan fungsi diferensiasi, karena pengakuan atas adanya perbedaan individual siswa berarti pula diberinya kesempatan bagi siswa tersebut untuk memilih apa yang sesuai dengan minat dan kemampuannya. Untuk mewujudkan kedua fungsi tersebut, kurikulum perlu disusun secara lebih luas dan bersifat fleksibel.

f. Fungsi Diagnostik (*the diagnostic function*)

Fungsi diagnostik mengandung makna bahwa kurikulum sebagai alat pendidikan harus mampu membantu dan mengarahkan siswa untuk dapat memahami dan menerima kekuatan (potensi) dan kelemahan yang dimilikinya. Apabila siswa sudah mampu memahami kekuatan-kekuatan dan kelemahan-kelemahan yang ada pada dirinya, maka diharapkan siswa dapat mengembangkan sendiri potensi kekuatan yang dimilikinya atau memperbaiki kelemahan-kelemahannya.

Kurikulum memiliki tiga peranan yang dinilai sangat penting, yaitu:

a. Peranan Konservatif

Peranan konservatif menekankan bahwa kurikulum itu dapat dijadikan sebagai sarana untuk mentransmisikan nilai-nilai warisan budaya masa lalu yang dianggap masih relavan dengan masa kini kepada generasi muda,

dalam hal ini para siswa. Peranan konservatif ini pada hakikatnya menempatkan kurikulum yang berorientasi ke masa lampau. Peranan ini sifatnya menjadi sangat mendasar, disesuaikan dengan kenyataan bahwa pendidikan pada hakikatnya merupakan proses sosial. Salah satu tugas pendidikan yaitu mempengaruhi dan membina perilaku siswa sesuai dengan nilai-nilai sosial yang hidup di lingkungan masyarakatnya (Hernawan & Susilana, 2008: hal. 13).

b. Peranan Kreatif

Perkembangan ilmu pengetahuan dan aspek-aspek lainnya senantiasa terjadi setiap saat. Peranan kreatif menekankan bahwa kurikulum harus mampu mengembangkan sesuatu yang baru sesuai dengan perkembangan yang terjadi dan kebutuhan-kebutuhan masyarakat pada masa sekarang dan masa mendatang. Kurikulum harus mengandung hal-hal yang dapat membantu setiap siswa mengembangkan semua potensi yang ada pada dirinya untuk memperoleh pengetahuan-pengetahuan baru, kemampuan-kemampuan baru, serta cara berpikir baru yang dibutuhkan dalam kehidupannya (Hernawan & Susilana, 2008: hal. 13).

c. Peranan Kritis dan Evaluatif

Peranan ini dilatarbelakangi oleh adanya kenyataan bahwa nilai-nilai dan budaya yang hidup dalam masyarakat senantiasa mengalami perubahan, sehingga pewarisan nilai-nilai dan budaya masa lalu kepada siswa perlu disesuaikan dengan kondisi yang terjadi pada masa sekarang. Selain itu, perkembangan yang terjadi pada masa sekarang dan masa mendatang belum tentu sesuai dengan apa yang dibutuhkan. Oleh karena itu, peranan kurikulum tidak hanya mewariskan nilai dan budaya yang ada atau menerapkan hasil perkembangan baru yang terjadi, melainkan juga memiliki peranan untuk menilai dan memilih nilai dan budaya serta pengetahuan baru yang akan diwariskan tersebut. Dalam hal ini, kurikulum harus turut aktif berpartisipasi dalam kontrol atau filter sosial. Nilai-nilai sosial yang tidal sesuai lagi dengan keadaan dan tuntutan masa kini dihilangkan dan diadakan modifikasi atau penyempurnaan-penyempurnaan (Hernawan & Susilana, 2008: hal. 14).

2.6.2 Pemetaan Kurikulum

Menurut Drake & Burns (2004, hal. 57-58), pemetaan kurikulum adalah sebuah proses untuk merekam konten dan keterampilan yang benar-benar diajarkan di kelas selama periode waktu tertentu. Pemetaan kurikulum sering digunakan sebagai alat untuk mencapai keselarasan eksternal dan internal. Manfaat dari pemetaan kurikulum bagi pengajar adalah pengajar mendapat pemahaman yang lebih dalam tentang konten kurikulum yang akan digunakan.

Pemetaan kurikulum mencakup pencatatan kurikulum yang dilaksanakan, membandingkannya dengan *curricula* (dokumen standar kurikulum) yang tertulis dan teruji, dan memperbaikinya jika diperlukan. Beberapa negara bagian dan provinsi menyediakan ruang lingkup dan urutan konsep yang disarankan untuk digunakan oleh pendidik. Dengan menggunakan ruang lingkup dan urutan kurikulum, pengajar akan semakin mudah untuk mencapai keselarasan kurikulum eksternal dan membuat peta kurikulum.

Pemetaan kurikulum membantu pengajar untuk mempersiapkan integrasi kurikulum. Pengajar dapat membandingkan peta kurikulum mereka dengan peta kurikulum orang lain yang mengajar bidang yang sama atau bidang lainnya di tingkat yang sama.