

BAB 2

LANDASAN TEORI

2.1 Rekayasa Piranti Lunak (RPL)

2.1.1 Definisi RPL

Menurut Pressman (2010, p4), piranti lunak (*software*) adalah

- a. Instruksi (program-program komputer) yang ketika dieksekusi akan memberikan fitur, fungsi, dan performa sesuai yang diharapkan.
- b. Kumpulan struktur data yang memungkinkan program untuk memanipulasi informasi, dan
- c. Informasi yang bersifat deskriptif dalam bentuk *hard copy* ataupun bentuk-bentuk virtual yang mendeskripsikan aplikasi dan penggunaan program.

2.1.2 Karakteristik RPL

Secara garis besar, karakteristik perangkat lunak berbeda dengan perangkat keras. Menurut Pressman (2010, p4), karakteristik dari perangkat lunak adalah sebagai berikut:

1. Perangkat lunak dikembangkan atau direkayasa, tidak diproduksi dalam pengertian klasik.
2. Perangkat lunak tidak “usang”.
3. Meskipun industri bergerak menuju konstruksi berbasis komponen, sebagian besar perangkat lunak terus dibangun secara khusus.

2.1.3 Lapisan RPL

Pressman (2010, p13) mendefinisikan rekayasa perangkat lunak merupakan teknologi yang bertingkat atau berlapis. Lapisan/tingkatan tersebut terbagi atas empat lapisan, yaitu:

1. *Quality focus*

Quality focus merupakan batu landasan yang menopang *tools*, *methods*, dan *process* dalam RPL.

2. *Process Model*

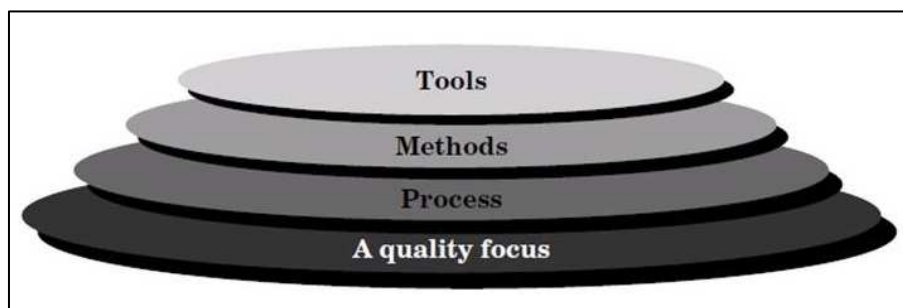
Process Model merupakan fondasi dari RPL yang mendefinisikan sebuah *framework* untuk sekumpulan *key process area* yang harus dibangun demi keefektifan penyampaian teknologi pengembangan RPL.

3. *Methods*

Methods menerangkan secara teknis tentang bagaimana membangun suatu perangkat lunak. Lapisan ini meliputi tugas-tugas yang mencakup analisis kebutuhan (*requirements analysis*), model desain (*design modeling*), pembuatan program (*program construction*), pengujian (*testing*), dan pendukung (*support*).

4. *Tools*

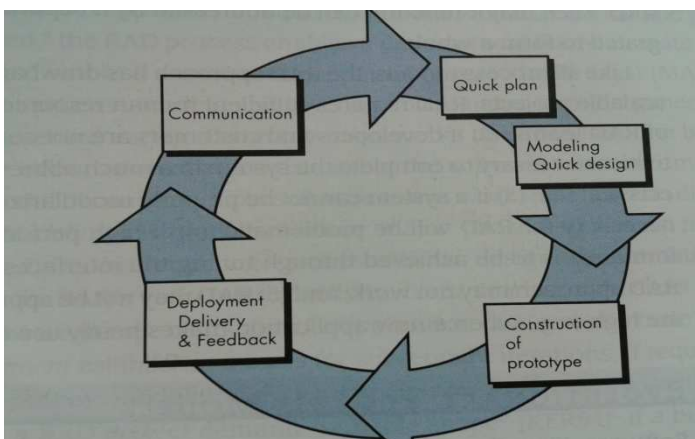
Tools menyediakan dukungan otomatis dan semi otomatis untuk *process model* dan *methods*.



Gambar 2.1 *Software Engineering Layers* (Sumber: Pressman, 2010, p14)

2.1.4 Prototyping Model

Menurut Pressman (2010, p43-44), seringkali pelanggan mendefinisikan satu set tujuan umum untuk perangkat lunak, tetapi tidak mengidentifikasi persyaratan rinci untuk fungsi dan fitur. Di lain kasus, pengembang mungkin tidak yakin dari efisiensi dari sebuah algoritma, adaptasi dari sistem operasi, atau bentuk yang interaksi manusia-mesin harus ambil. Dalam hal ini, dan situasi lain, paradigma prototipe mungkin menawarkan pendekatan yang terbaik.



Gambar 2.2 *Prototype Model* (Sumber: Pressman, 2010, p43)

Tahapan-tahapan dalam *Prototyping* adalah sebagai berikut:

1. Pengumpulan kebutuhan

Pelanggan dan *developer* bersama-sama mendefinisikan format seluruh perangkat lunak, mengidentifikasi semua kebutuhan, dan garis besar sistem yang akan dibuat.

2. Membangun *prototyping*

Membangun *prototyping* dengan membuat perancangan sementara yang berfokus pada penyajian kepada pelanggan (misalnya dengan membuat input dan format *output*).

3. Evaluasi *prototyping*

Evaluasi ini dilakukan oleh pelanggan apakah *prototyping* yang sudah dibangun sudah sesuai dengan keinginan pelanggan. Jika sudah sesuai maka langkah 4 akan diambil. Jika tidak *prototyping* direvisi dengan mengulangi langkah 1, 2, dan 3.

4. Mengkodekan sistem

Dalam tahap ini *prototyping* yang sudah disepakati diterjemahkan ke dalam bahasa pemrograman yang sesuai.

5. Menguji sistem

Setelah sistem sudah menjadi suatu perangkat lunak yang siap pakai, harus dites dahulu sebelum digunakan. Pengujian ini dilakukan dengan *White Box*, *Black Box*, *Basis Path*, pengujian arsitektur dan lain-lain.

6. Evaluasi sistem

Pelanggan mengevaluasi apakah sistem yang sudah jadi sudah sesuai dengan yang diharapkan. Jika ya, langkah 7 dilakukan, jika tidak, ulangi langkah 4 dan 5.

7. Menggunakan sistem

Perangkat lunak yang telah diuji dan diterima pelanggan siap untuk digunakan.

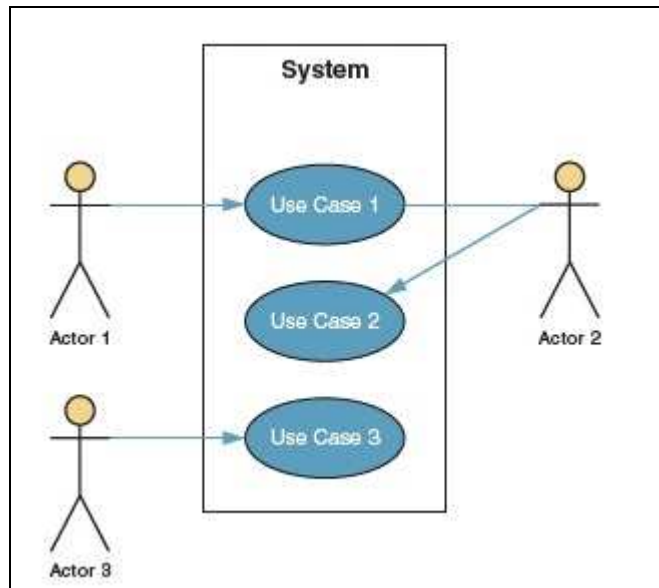
2.2 UML (*Unified Modeling Language*)

Menurut Whitten & Bentley (2007, p371), UML adalah suatu konvensi pemodelan yang digunakan untuk menspesifikasikan atau mendeskripsikan sebuah sistem piranti lunak yang terkait dengan objek. UML terdiri dari beberapa tipe diagram, yaitu:

a. *Use-Case* Diagram

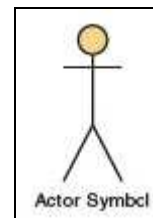
Menurut Whitten & Bentley (2007, p246-250), *use-case* diagram adalah diagram yang menggambarkan interaksi antara sistem dan pengguna. Dengan kata lain, diagram ini mendeskripsikan siapa yang akan menggunakan sistem itu dengan cara apa pengguna berinteraksi dengan sistem.

1. *Use case* adalah urutan langkah-langkah yang secara tindakan saling terkait, baik terotomatisasi maupun secara manual untuk melengkapi suatu tugas bisnis tunggal.



Gambar 2.3 Contoh Diagram Model *Use Case* (Sumber: Whitten & Bentley, 2007, p246)

2. Pelaku (*Actor*) adalah segala sesuatu yang perlu berinteraksi dengan sistem untuk pertukaran informasi.



Gambar 2.4 Simbol Aktor (Sumber: Whitten & Bentley, 2007, p247)

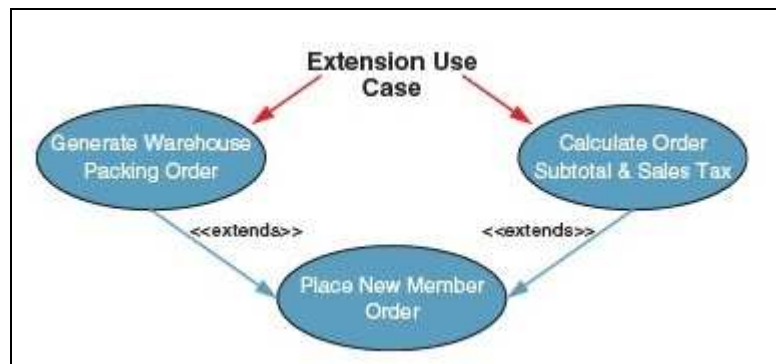
3. Hubungan (*Relationship*) pada diagram *use case* digambarkan sebagai sebuah garis antara dua simbol. Pemaknaan hubungan berbeda-beda tergantung bagaimana garis tersebut digambar dan tipe simbol apa yang digunakan untuk menghubungkan garis tersebut. Beberapa tipe hubungan yang terdapat pada diagram *use case*, yaitu sebagai berikut:

- 1) Gabungan (*Association*): hubungan antara pelaku dengan *use case*, dimana terjadi interaksi di antara mereka.



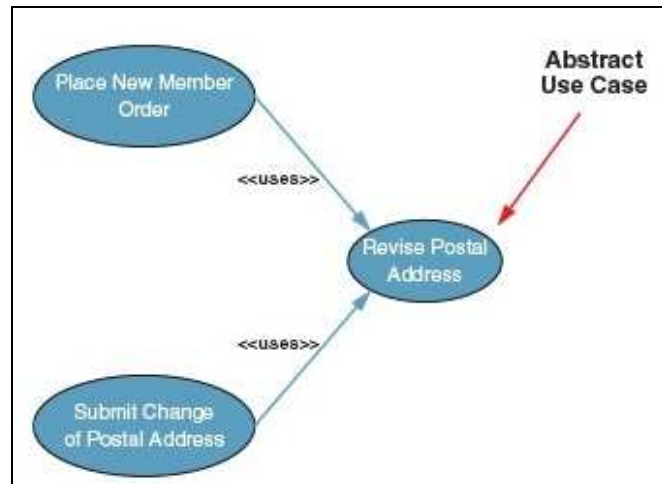
Gambar 2.5 Contoh Hubungan Asosiasi (Sumber: Whitten & Bentley, 2007, p248)

- 2) *Extend*: *use case* yang terdiri dari langkah yang diekstraksi dari *use case* yang lebih kompleks untuk menyederhanakan masalah orisinil dan karena itu memperluas fungsinya.



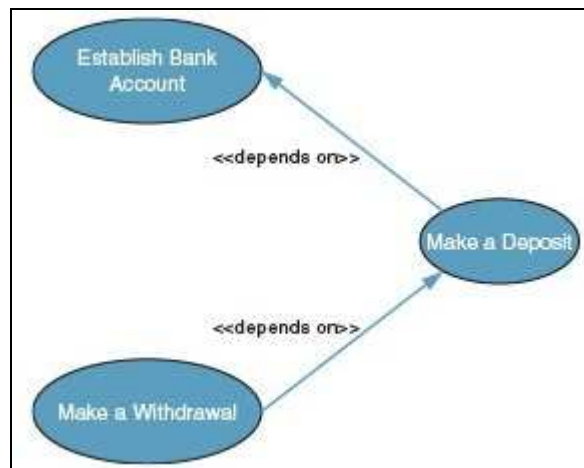
Gambar 2.6 Contoh Hubungan Extension (Sumber: Whitten & Bentley, 2007, p249)

- 3) *Uses*: satu atau lebih *use case* yang melakukan berbagai langkah fungsionalitas yang identik.



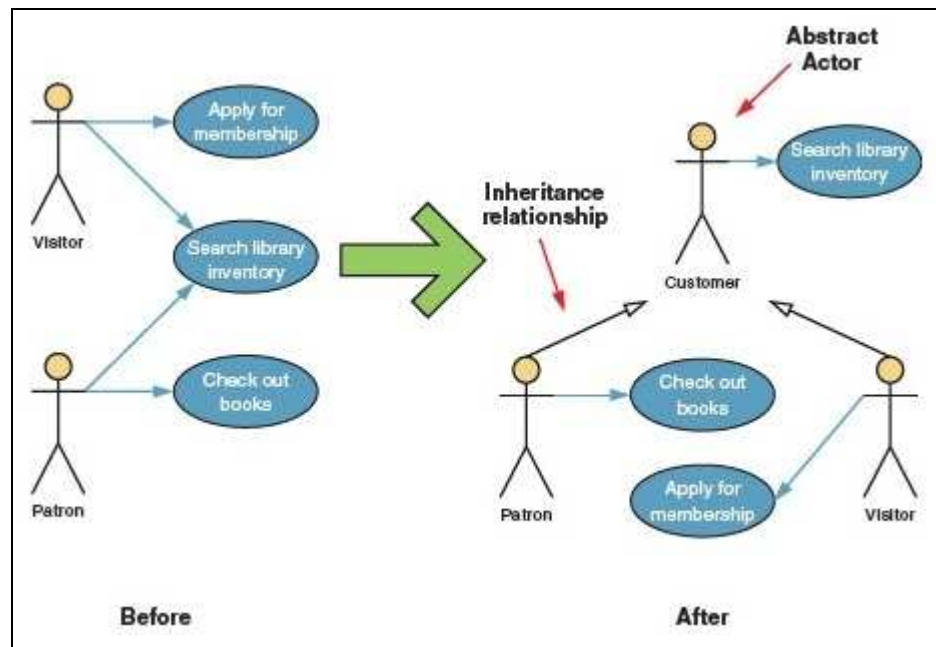
Gambar 2.7 Contoh Hubungan Uses (Sumber: Whitten & Bentley, 2007, p249)

- 4) *Depends On*: use case mana yang memiliki ketergantungan pada use case lain untuk menetapkan rangkaian use case yang perlu dikembangkan.



Gambar 2.8 Contoh Hubungan Depends On (Sumber: Whitten & Bentley, 2007, p250)

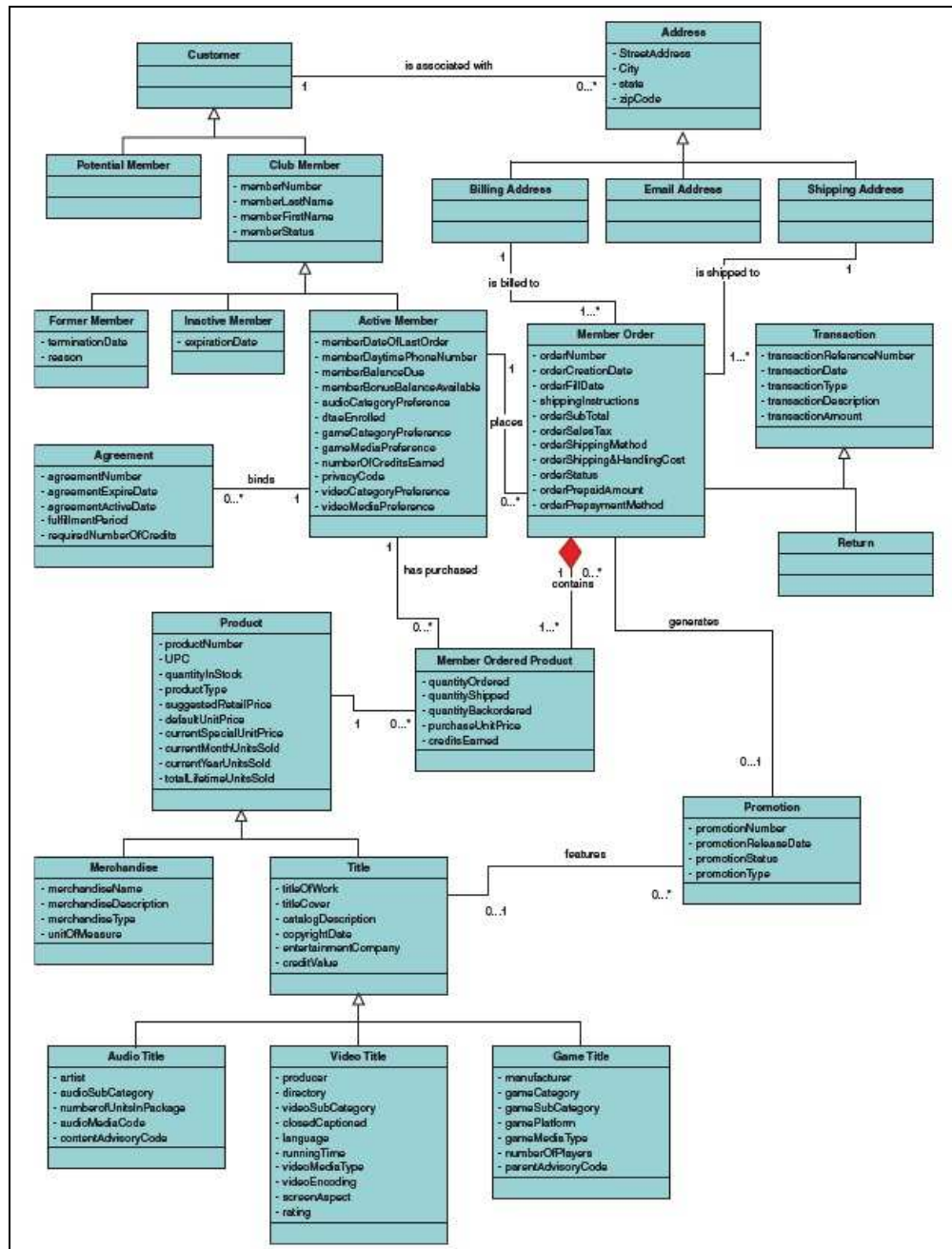
- 5) *Inheritance*: pada saat dua atau lebih pelaku berbagi kelakuan umum.



Gambar 2.9 Contoh Hubungan *Inheritance* (Sumber: Whitten & Bentley, 2007, p250)

b. *Class Diagram*

Menurut Whitten & Bentley (2007, p400), *class diagram* menggambarkan struktur objek yang terdapat pada sistem. Diagram ini menunjukkan objek yang terdapat pada suatu sistem serta relasi antara objek-objek tersebut.

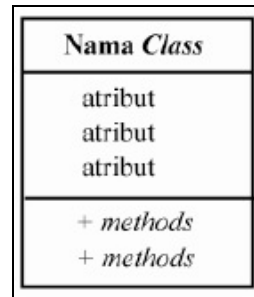


Gambar 2.10 Contoh *Class Diagram* (Sumber: Whitten & Bentley, 2007,

Beberapa elemen yang terdapat dalam *class diagram*, yaitu:

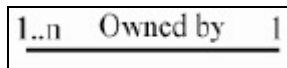
- 1) *Class*: digambarkan sebagai sebuah kotak yang terbagi atas tiga bagian.

Bagian atas adalah bagian nama dari *class*. Bagian tengah mendefinisikan atribut *class*. Bagian akhir mendefinisikan *method* dari sebuah *class*.



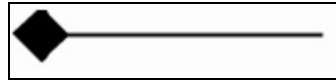
Gambar 2.11 Class

- 2) *Association*: merupakan sebuah hubungan yang paling umum antara dua *class* dan dilambangkan oleh sebuah garis yang menghubungkan antara dua *class*. Garis ini bisa melambangkan tipe-tipe hubungan. Contohnya *one-to-one*, *one-to-many*, *many-to-many*.



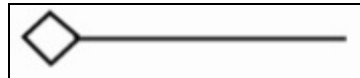
Gambar 2.12 Association

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



Gambar 2.13 Composition

- 4) *Aggregation*: jika sebuah *class* bisa berdiri sendiri walaupun merupakan bagian dari *class* yang lain.



Gambar 2.14 Aggregation

- 5) Cara bagaimana atribut dan metode diakses oleh kelas lain didefinisikan dengan *visibility*. UML menyediakan tiga tingkat dari *visibility*, yaitu:

1. *Public* : ditandai dengan simbol “+”
2. *Protected* : ditandai dengan simbol “#”
3. *Private* : ditandai dengan simbol “-”

c. *Activity Diagram*




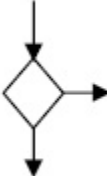
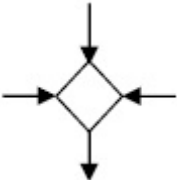
Menurut Whitten & Bentley (2007, p390), *Activity diagram* adalah sebuah diagram yang dapat digunakan untuk mendeskripsikan secara grafis aliran kerja dari sebuah proses bisnis, langkah-langkah dari sebuah *use case*, atau logika dari sebuah perilaku objek (*object behavior/method*).

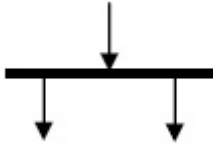
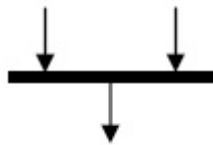

Activity diagram adalah representasi grafis dari aliran kerja (*workflow*) atas aktivitas-aktivitas yang bertahap dan aksi-aksi di dalam sebuah sistem.

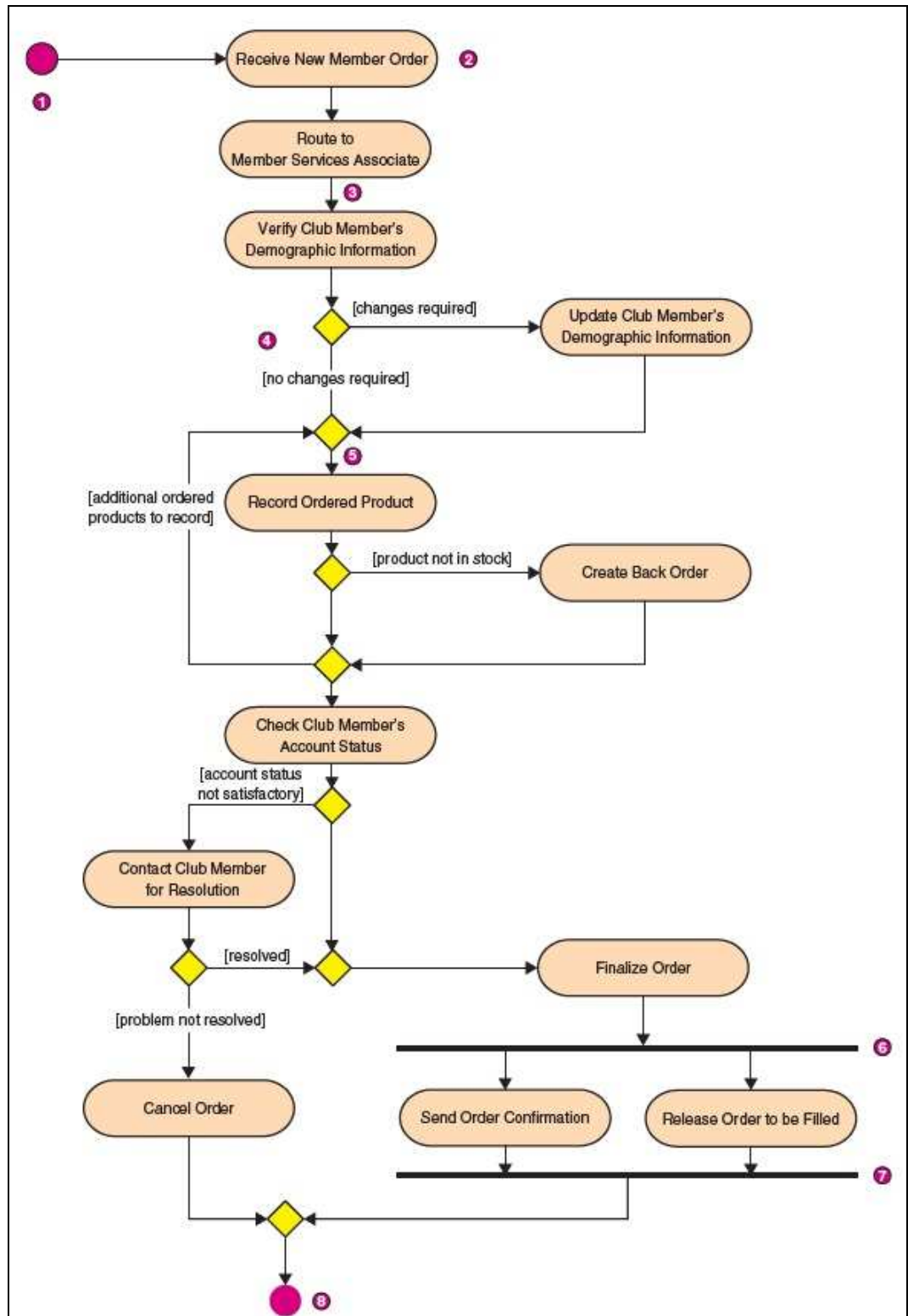
Diagram ini juga mendukung proses seleksi, pengulangan (iterasi), dan konkurensi.

Activity diagram sejenis dengan *state diagram/statechart diagram*. Diagram ini mendeskripsikan/menggambarkan kumpulan aktivitas dengan menunjukkan sekuensial dari aktivitas yang dilakukan. Diagram ini juga bisa menunjukkan aktivitas yang bersifat kondisional atau paralel. Berikut notasi yang digunakan pada *activity diagram*:

Tabel 2.1 Simbol pada *Activity Diagram*

No	Simbol	Gambar	Keterangan
1	<i>Initial node</i>		Menunjukkan awal dari suatu proses.
2	<i>Actions</i>		Menunjukkan langkah-langkah aktivitas sistem yang terjadi.
3	<i>Flow</i>		Untuk menunjuk ke aksi atau proses yang lain. Dilengkapi dengan kata-kata jika berasal dari <i>decision</i> .
4	<i>Decision</i>		Memiliki sebuah <i>flow</i> masuk dan dua atau lebih <i>flow</i> keluar. Menunjukkan aktivitas yang dapat dipilih.
5	<i>Merge</i>		Menyatukan <i>flow</i> yang sebelumnya terpisah oleh <i>decision</i> .

6	<i>Fork</i>		Menotasikan permulaan aksi atau proses paralel yang dapat terjadi dalam suatu urutan atau terjadi secara bersamaan.
7	<i>Join</i>		Semua aksi yang masuk ke <i>join</i> harus telah diselesaikan sebelum proses berlanjut.
8	<i>Activity Final</i>		Menunjukkan akhir dari proses.



Gambar 2.15 Contoh Activity Diagram (Sumber: Whitten & Bentley, 2007,

d. *Sequence Diagram*

Menurut Whitten & Bentley (2007, p394-395), *Sequence diagram* menggambarkan bagaimana objek-objek berinteraksi satu sama lain melalui pesan dalam eksekusi dari sebuah *use case* atau operasi. Sebuah *system sequence diagram* membantu untuk mengidentifikasi pesan tingkat tinggi yang masuk dan keluar dari sistem. Beberapa notasi yang terdapat di dalam *system sequence diagram*, yaitu:

1. *Actor*, berupa aktor yang memulai pada *use case* ditunjukkan dengan simbol *use case* aktor.
2. *System*, berupa kotak yang menunjukkan sistem yang sedang berjalan, ditandai dengan notasi titik dua (:) pada awal nama sistem.
3. *Lifelines*, berupa garis putus-putus menurun dari simbol aktor dan sistem.
4. *Activation bars*, berupa balok yang terletak di atas *lifelines*, menunjukkan periode waktu ketika terdapat interaksi secara aktif.
5. *Input messages*, berupa anak panah mendarat dari aktor ke sistem yang menunjukkan pesan masuk, dimana kata pertama diawali dengan huruf kecil, sedangkan kata berikutnya diawali dengan huruf besar dan tidak ada spasi, diikuti dengan tanda kurung yang berisi parameter yang dibutuhkan.
6. *Output messages*, berupa anak panah mendarat dengan garis putus-putus dari sistem ke aktor yang menunjukkan pesan keluar.
7. *Receiver Actor*, berupa aktor lain yang menerima pesan dari sistem.
8. *Frame*, berupa kotak yang menambahkan pesan terpisah untuk menunjukkan perulangan, alternative, atau pilihan.

2.3 Multimedia

Menurut Vaughan (2011, p1), multimedia adalah semua jenis kombinasi dari teks, gambar, suara, animasi, dan video yang saling berinteraksi satu sama lain dan membentuk satu kesatuan yang harmonis.

Menurut Rahman (2008, xxviii), kata multimedia berasal dari kata latin “multum” dan “media”, yang berarti kombinasi dari berbagai isi media. Secara umum, multimedia termasuk kombinasi teks, suara, gambar diam, animasi, video, dan konten interaktif. Integrasi teknologi multimedia dalam lingkungan komunikasi memiliki potensi untuk mengubah penonton dari penerima informasi pasif menjadi penerima informasi aktif dalam media yang kaya akan proses pembelajaran. Istilah “media yang kaya” adalah sinonim untuk multimedia interaktif.

Menurut Irianto (2009, p34), multimedia interaktif juga dapat memberikan pembelajaran kooperatif dan interaktif sesama pelajar (seorang siswa lebih suka belajar dari teman sebayanya). Dengan kehadiran multimedia ini seorang siswa berpeluang untuk memahami suatu masalah/topik, menyelesaikan masalah dan membuat keputusan, dan menghasilkan proyek multimedia bersama yang nantinya sepenuhnya akan digunakan dalam proses pembelajaran tersebut.

2.3.1 Elemen Multimedia

Menurut Vaughan (2011, p20-164), elemen-elemen multimedia dibagi menjadi lima, yaitu:

1. Teks

Kata tunggal dapat memuat banyak arti sehingga saat perancang multimedia mulai bekerja dengan teks sangat penting untuk menerapkan keakuratan dan kepadatan dalam kata tertentu yang dipilih. Dalam multimedia, kata akan muncul dalam judul, menu, navigasi, narasi, dan isi.

Teks memiliki beberapa atribut yang penting, diantaranya adalah sebagai berikut:

- a. *Typical font style: bold, italic, underlined.*
- b. *Size*: biasanya didefinisikan dalam *point* (pt). 1 point = 0.0138 *inch* atau sekitar 1/72 *inch*.
- c. *Leading*: spasi antar baris.
- d. *Kerning*: spasi antar karakter.

Dalam teks terdapat istilah *typeface* dan *font*. *Typeface* merupakan keluarga dari karakter grafis yang biasanya meliputi banyak jenis ukuran dan gaya. *Font* merupakan kumpulan dari karakter dari satu ukuran dan gaya yang menjadi bagian dari keluarga *typeface*.

Dalam teks juga terdapat istilah *serif* dan *sanserif*. *Serif* dan *sanserif* adalah cara yang paling mudah untuk mendefinisikan *typeface*. *Serif* memiliki dekorasi kecil yang ada di ujung penulisan huruf. Contoh *font* dari *serif* adalah Times New Roman, Bookman, dan Palatino. Sedangkan contoh *font* dari *sanserif* adalah Helvetica, Verdana, Arial, Optima, dan Avant Garde.

2. Gambar

Gambar merupakan elemen visual yang membentuk multimedia. Elemen ini dapat diubah ke ukuran yang berbeda, diwarnai, diberi pola, dibuat terlihat atau transparan. Gambar dibagi menjadi dua jenis, yaitu:

a. Bitmap

Bit adalah elemen paling sederhana dalam dunia digital, benda elektronik yang dapat dihidupkan dan dimatikan, hitam atau putih, benar (1) atau salah (0). Bitmap merupakan matriks sederhana dari titik-titik kecil yang membentuk sebuah gambar dan ditampilkan di layar komputer.

b. Vektor

Vektor adalah sebuah garis yang dideskripsikan oleh 2 buah titik. Gambar vektor menggunakan kordinat Cartesius dimana sepasang angka menggambarkan titik dalam ruang dua dimensi sebagai persimpangan garis vertikal dan horisontal.

Gambar memiliki beberapa jenis format *file image*, yaitu:

- a. .PICT: merupakan format *file default* dari Apple Mac yang berjalan pada perangkat *platform* Macintosh.
- b. .BMP: merupakan format *file default* dari Windows yang mendukung RGB, *indexed color*, dan *grayscale*.
- c. .JPEG (.JPG): singkatan dari *Joint Photographic Experts Group* yang bertanggung jawab terhadap pengembangan format pemetaan pada gambar dan merupakan suatu standar yang digunakan di seluruh dunia.

- d. .GIF: *Graphic Interchange File* format merupakan format *file* yang telah terkompresi yang dikembangkan oleh CompuServe yang biasanya digunakan hampir diseluruh internet.
- e. .PSD: merupakan format yang digunakan oleh Photoshop untuk menyimpan *file* yang baru dibuat maupun *file* yang telah dimanipulasi.

3. Suara

Suara merupakan elemen paling sensasional dalam multimedia. Dengan adanya fasilitas suara, hasil visualisasi akan menjadi lebih sempurna dan nyata jika digunakan dengan benar.

Secara umum, ada dua jenis *file* audio yang dapat digabung dalam lingkungan multimedia, yaitu:

a. Suara digital

Suara digital dibuat dengan mengkonversikan sebuah gelombang suara ke dalam angka, prosesnya disebut *digitizing*. Suara digital dapat diciptakan dari sebuah mikrofon, sintesiser, CD Player, ataupun dari siaran TV.

b. MIDI (*Musical Instrument Digital Interface*)

MIDI merupakan standar komunikasi untuk instrumen musik elektronik dan komputer dengan menggunakan kabel, musik, dan *synthesizer* yang dapat berkomunikasi satu sama lain yang berisi detail tentang nada, rangkaian nada, maupun jenis alat musik yang memainkan nada-nada tersebut.

4. Animasi

Animasi membuat gambar statik menjadi lebih hidup. Animasi sebenarnya adalah sebuah objek yang bergerak menyeberangi, ke dalam, atau

ke luar layar, seperti: *globe* bumi yang berputar atau sebuah mobil yang berjalan menyusuri jalan. Animasi dibagi menjadi tiga jenis, yaitu:

a. Animasi 2D

Dalam animasi 2D, perubahan visual yang terjadi terdapat di bidang Cartesian x dan y di layar.

b. Animasi 2.5D

Dalam animasi 2.5D, ilusi kedalaman (sumbu z) akan ditambahkan ke gambar melalui bayangan dan sorotan lampu, namun gambar itu sendiri masih ada dalam aksis sumbu datar x dan y dalam dua dimensi. Pemberian efek *emboss*, *shadow*, *bevel*, dan *highlight* membangun ilusi kedalaman dengan menaikkan gambar atau memotongnya kedalam latar belakang.

c. Animasi 3D

Dalam animasi 3D ini, *software* membuat dunia *virtual* dalam tiga dimensi, dan perubahan dihitung sepanjang ketiga sumbu (x,y, dan z), sehingga memungkinkan gambar atau objek yang dibuat sendiri dengan sisi depan, belakang, atas, bawah, kiri, kanan untuk bergerak mendekati dan menjauhi pengamat, atau memungkinkan pengamat untuk mengelilingi dan melihat objek dari segala sudut.

Berikut ini adalah beberapa teknik animasi yang sering digunakan:

- a. *Cel Animation*: teknik animasi yang menggunakan serangkaian grafis progresif yang berbeda dalam tiap *frame* (yang dimainkan 24 *frame* per detik), yang berarti dalam 1 menit membutuhkan *frame* sebanyak 1440.

- b. *Computer Animation*: animasi komputer pada umumnya menggunakan logika yang sama dan konsep prosedural seperti animasi sel. Perbedaannya terletak pada berapa banyak yang perlu digambar oleh animator serta berapa banyak yang dihasilkan komputer.

5. Video

Video adalah suatu gambar bergerak yang dikirim melalui sinyal elektronik sehingga dapat muncul di layar. Video merupakan elemen yang paling menarik dari bidang multimedia dan merupakan alat yang ampuh untuk membawa pengguna komputer lebih dekat ke dunia nyata.

2.4 Interaksi Manusia dan Komputer

Menurut Shneiderman dan Plaisant (2010, p22), Interaksi Manusia dan Komputer (IMK) adalah disiplin ilmu yang berhubungan dengan perancangan, evaluasi, dan implementasi sistem komputer interaktif untuk digunakan oleh manusia. Titik berat IMK adalah perancangan dan evaluasi antarmuka pemakai (*user interface*). Antarmuka pemakai adalah bagian sistem komputer yang memungkinkan manusia berinteraksi dengan komputer.

2.4.1 Lima Faktor Manusia Terukur

Menurut Shneiderman dan Plaisant (2010, p32), ada lima faktor manusia terukur yang dapat dijadikan pusat evaluasi kebutuhan pengguna dalam perancangan suatu antarmuka, yaitu:

1. Waktu belajar

Merupakan suatu tolak ukur seberapa lama seorang *user* mampu mempelajari tindakan apa saja yang diperlukan untuk suatu tugas yang disediakan.

2. Kecepatan kinerja

Merupakan suatu tolak ukur seberapa lama suatu tugas dapat dilaksanakan oleh *user*.

3. Tingkat kesalahan *user*

Merupakan suatu tolak ukur seberapa banyak dan apa saja kesalahan yang dilakukan *user* dalam mengerjakan suatu tugas.

4. Daya ingat

Merupakan suatu tolak ukur seberapa baik seorang *user* dapat mengingat pengetahuan yang didapatkan dalam waktu tertentu.

5. Kepuasan subjektif

Merupakan suatu tolak ukur seberapa besar tingkat kesukaan *user* pada tampilan antarmuka yang ada pada aplikasi.

2.4.2 *Eight Golden Rules of Interface Design*

Menurut Shneiderman dan Plaisant (2010, p88-89), terdapat delapan aturan emas (*eight golden rules*) yang biasa digunakan dalam perancangan antarmuka pemakai, yaitu:

1. Berusaha untuk konsisten

Konsistensi harus dilakukan pada setiap tindakan, perintah, dan istilah yang digunakan pada *prompt*, menu, layar bantu, warna tampilan, kapitalisasi, huruf, dan sebagainya.

2. Menyediakan penggunaan yang universal

Menambahkan fitur seperti tombol cepat (*shortcuts*) untuk memudahkan pengguna. Pemakaian *shortcuts* digunakan untuk mengurangi jumlah interaksi dan meningkatkan kecepatan interaksi. Singkatan, tombol fungsi, dan fasilitas makro sangat membantu untuk pengguna yang sudah ahli.

3. Memberikan umpan balik yang informatif

Untuk setiap aksi yang dilakukan oleh *user* harus diberikan umpan balik agar tercipta suasana yang komunikatif. Pada aksi yang kecil dan sering digunakan, respon yang diberikan sederhana. Namun, pada aksi yang besar dan jarang digunakan, respon yang diberikan harus lebih banyak dan rinci.

4. Merancang dialog untuk memberikan penutupan

Urutan tindakan sebaiknya diorganisasi ke dalam kelompok awal, tengah, dan akhir. Tampilan harus memberikan dialog untuk menunjukkan penutupan ketika aplikasi selesai diproses atau dijalankan.

5. Memberikan pencegahan dan penanganan kesalahan yang sederhana

Sebisanya mungkin, sistem dirancang untuk dapat mencegah *user* dari kesalahan fatal yang dilakukan. Jika kesalahan dibuat, sistem harus mampu mendeteksi kesalahan, menjelaskan, dan menawarkan penanganan kesalahan untuk memperbaikinya.

6. Memungkinkan pembalikan aksi yang mudah

Diharapkan adanya fitur untuk kembali (*back*) ke aktivitas sebelumnya. Fitur ini bertujuan agar pengguna dapat kembali ke aktivitas sebelumnya jika ternyata terdapat suatu kesalahan yang disebabkan pengguna, sehingga kesalahan dapat diperbaiki.

7. Mendukung pusat kendali internal

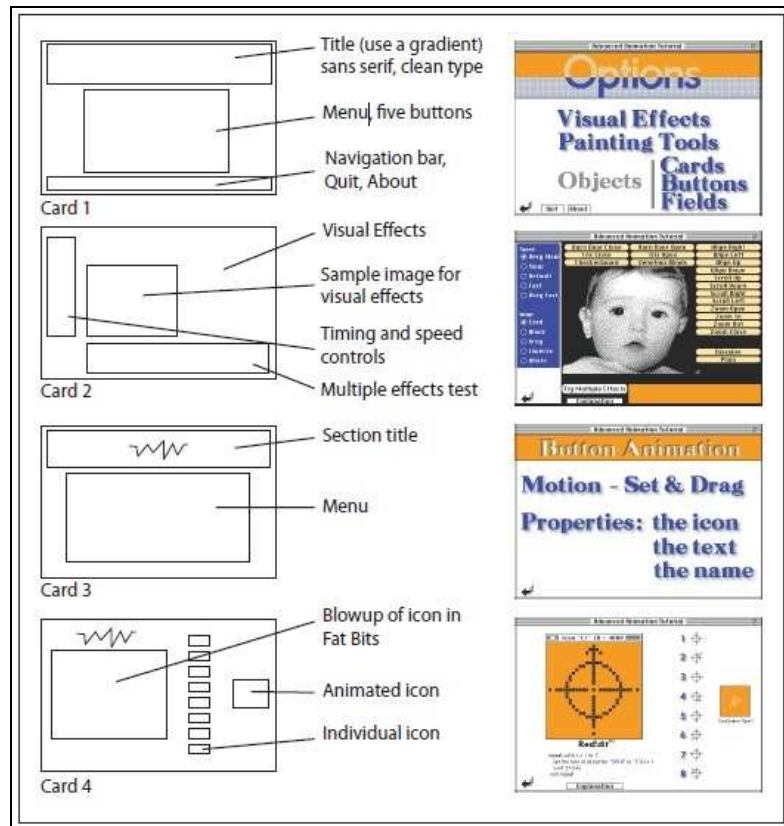
User memiliki kendali sebagai pemberi aksi pada sistem antarmuka. Kemudian antarmuka tersebut harus merespon kembali, sehingga dapat mengontrol program-program yang ada dalam sistem.

8. Mengurangi beban ingatan jangka pendek

Sebuah sistem diharapkan dapat ditampilkan sesederhana mungkin dan sebaiknya tampilan halaman yang banyak disatukan, serta diberikan cukup waktu dalam mempelajari kode, singkatan, dan urutan tindakan.

2.5 *Storyboard*

Menurut Rahman (2008, xxxviii), untuk materi video, termasuk wawancara, documenter, potongan narasi, dan sebagainya, buat sebuah storyboard sederhana yang mencantumkan urutan peristiwa, dialog, teks, lokasi, jepretan kamera, suara, lagu, dan sebagainya. Hal tersebut memberikan panduan kasar untuk proses *shooting* dan *editing*, dan perlu diperbarui selama proses produksi. Sebuah cetak biru dari aplikasi diputuskan dan dibuat, termasuk diagram navigasi untuk berpindah dari cerita satu ke cerita lainnya yang berbeda. Pilihan jenis media berperan penting dalam menentukan penerimaan dari proyek akhir.



Gambar 2.16 Contoh Storyboard (Sumber: Vaughan, 2011, p301)

2.6 Teknologi Informasi dan Komunikasi

Menurut Irianto (2009, p33), perkembangan Teknologi Informasi dan Komunikasi tidak terlepas dengan perangkat komputer yang dewasa ini menguasai teknologi pembelajaran di sekolah-sekolah. Komputer adalah alat atau seperangkat yang dipakai untuk mengolah informasi menurut prosedur yang telah dirumuskan. Kata komputer semula dipergunakan untuk menggambarkan orang yang pekerjaannya melakukan perhitungan aritmatika, dengan atau tanpa alat bantu, tetapi arti kata ini kemudian dipindahkan kepada mesin itu sendiri. Asal mulanya, pengolahan informasi hampir eksklusif berhubungan dengan masalah aritmatika. Namun dalam perkembangannya istilah yang lebih baik dan

yang cocok untuk arti luas seperti komputer adalah yang memproses informasi atau sistem pengolah informasi.

Dalam perkembangannya, akhirnya komputer merambah pada dunia pendidikan, yang mula-mula perangkat ini digunakan untuk membantu dalam pekerjaan administrasi pendidikan, namun akhirnya sekarang komputer bergeser penggunaanya dalam penggunaan pengajaran.

ICT telah membuka era baru bagi proses belajar mengajar. Bukan hanya bermanfaat bagi peningkatan mutu siswa, tetapi juga manajemen pendidikan, biaya pendidikan, dan sumber pendidikan.

2.7 *File Based*

Menurut Connolly & Begg (2010, p57-63), *file based system* adalah program-program aplikasi, dimana setiap program memiliki dan mengatur datanya masing-masing, serta digunakan untuk memenuhi kebutuhan *user*, misalnya dengan menghasilkan *report*.

Berikut ini adalah ciri-ciri *file based system* yang terdapat dalam suatu aplikasi:

1. Setiap program mempunyai datanya masing-masing dan tidak digunakan secara bersama-sama.
2. Definisi *file* terdapat pada program.
3. Memiliki format yang berbeda antar *file*, baik dalam nama *field*, tipe data, dan panjang data.

Dengan menggunakan *file based system*, biaya yang dibutuhkan menjadi lebih murah, karena dalam implementasinya tidak dibutuhkan *software* maupun

hardware yang banyak, sehingga ukurannya juga lebih kecil. *File based system* lebih sederhana, sehingga mudah digunakan oleh *user*, karena tidak adanya pengaturan fungsi-fungsi seperti pada DBMS. Selain itu, *file based system* memiliki dampak yang lebih rendah bila terjadi kegagalan. Setiap *file* memiliki datanya masing-masing dan terpisah antara *file* yang satu dengan yang lainnya, sehingga apabila ada kerusakan data pada *file*, tidak akan mengganggu *file* yang lainnya. Misalnya, *file* barang yang ada di dalam *file* persediaan mengalami kerusakan, maka *file* barang yang berada di dalam *file* pembelian atau penjualan tidak akan terpengaruh.

Dalam *file based system* dapat terjadi duplikasi data, karena dalam *file* yang berbeda terdapat data yang sama, namun tidak saling terhubung, sehingga data menjadi tidak konsisten. Selain itu untuk melakukan perubahan pada *query* dan laporan menjadi tidak fleksibel dan cukup sulit untuk melakukan *backup* dan *recovery*.

2.7.1 XML (eXtensible Markup Language)

Menurut Connolly & Begg (2010, p1113-1115), XML merupakan suatu metabahasa (bahasa untuk menggambarkan bahasa lain) yang memungkinkan desainer untuk membuat tag yang disesuaikan sendiri untuk menyediakan fungsionalitas yang tidak tersedia dengan HTML. Beberapa keuntungan menggunakan XML adalah:

1. *Simplicity*

XML didesain sebagai bahasa berbasis teks yang dapat dibaca manusia dan cukup jelas.

2. *Open standard and platform/vendor independent*

XML telah dibangun untuk mendukung teks dalam semua huruf di dunia, termasuk metode untuk menunjukkan bahasa dan pengkodean yang digunakan.

3. *Extensibility*

Tidak seperti HTML, XML dapat diperluas, yang memungkinkan pengguna untuk mendefinisikan tag mereka sendiri untuk memenuhi kebutuhan tertentu dari aplikasi mereka sendiri.

4. *Reuse*

Ekstensibilitas juga memungkinkan perpustakaan tag XML yang akan dibangun sekali dan digunakan kembali oleh banyak aplikasi.

5. *Separation of content and presentation*

XML memisahkan isi dari sebuah dokumen dari bagaimana dokumen akan disajikan.

6. *Improved load balancing*

Data dapat dikirim ke browser pada desktop untuk perhitungan lokal, pembongkaran perhitungan dari server, dan dengan demikian mencapai *load balancing* yang lebih baik.

2.8 OOP (*Object Oriented Programming*)

Menurut Dan Clark (2011, p1-3), OOP adalah sebuah pendekatan untuk pengembangan perangkat lunak dimana struktur perangkat lunak ini didasarkan pada benda-benda berinteraksi satu sama lain untuk menyelesaikan tugas. Interaksi ini membutuhkan bentuk pesan lewat bolak-balik antara objek. Dalam

menanggapi pesan, objek dapat melakukan suatu tindakan atau metode. Contohnya seperti ketika seseorang ingin pergi ke toko dan berinteraksi dengan sebuah mobil. Sebuah objek mobil terdiri dari benda-benda yang berinteraksi satu sama lain untuk menyelesaikan tugas untuk mengantar orang tersebut ke toko. Orang tersebut meletakkan kunci dan menyalakannya. Hal tersebut mengirimkan pesan (melalui sinyal listrik) ke objek pemula, yang berinteraksi dengan objek mesin untuk menyalakan mobil. Sebagai supir, hal tersebut membuatnya terisolasi dari logika bagaimana suatu objek pada suatu sistem bekerja sama untuk menyalakan mobil. Orang tersebut hanya memulai urutan kejadian dengan menjalankan metode mulai dari objek pengapian dengan kunci, kemudian menunggu respon keberhasilan atau kegagalan.

Banyak pengembang *software* bisnis yang beralih ke metode objek oriented dan bahasa pemrograman untuk menyelesaikan berbagai masalah. Keuntungannya adalah sebagai berikut:

- a. Sebuah transisi yang lebih intuitif dari analisis model bisnis perangkat lunak ke model penerapan perangkat lunak.
- b. Kemampuan untuk memelihara dan menerapkan perubahan dalam program yang lebih efisien dan cepat.
- c. Kemampuan untuk lebih efektif menciptakan sistem perangkat lunak menggunakan proses tim, memungkinkan para ahli (*specialists*) untuk bekerja pada bagian dari sistem.
- d. Kemampuan untuk menggunakan kembali komponen kode dalam program lain dan membeli komponen yang ditulis oleh pengembang pihak ketiga untuk meningkatkan fungsi dari program mereka dengan sedikit usaha.

- e. Integrasi yang lebih baik dengan sistem komputasi terdistribusi yang digabungkan secara longgar.
- f. Peningkatan integrasi dengan sistem operasi modern.
- g. Kemampuan untuk membuat tampilan antarmuka yang lebih intuitif untuk pengguna.

2.8.1 Karakteristik OOP

Menurut Dan Clark (2011, p3-5), beberapa karakteristik yang terdapat dalam OOP adalah:

a. *Objects*

Dalam OOP, objek adalah sebuah struktur data untuk menggabungkan data dan prosedur untuk bekerja dengan data. Misalnya, apabila ingin memiliki kemampuan pencetakan, maka harus bekerja dengan objek printer yang bertanggung jawab atas data dan metode yang digunakan dengan printer tersebut.

b. *Abstraction*

Tanpa kemampuan untuk melakukan abstraksi atau menyaring sifat benda asing, akan terjadi kesulitan untuk memproses sejumlah informasi dan konsentrasi pada tugas yang sedang dilakukan.

c. *Encapsulation*

Fitur penting lainnya dari OOP adalah enkapsulasi. Enkapsulasi adalah proses dimana tidak ada akses langsung yang diberikan untuk data, melainkan tersembunyi. Untuk mendapatkan akses ke data, diperlukan interaksi dengan objek yang bertanggung jawab untuk data.

d. Polymorphism

Polimorfisme adalah kemampuan dari dua objek yang berbeda untuk merespon pesan permintaan yang sama dalam cara yang unik.

e. Inheritance

Dalam OOP *inheritance* digunakan untuk mengklasifikasikan objek dalam program sesuai dengan karakteristik dan fungsi yang umum. Hal itu membuat bekerja dengan objek menjadi lebih mudah dan intuitif. Hal itu juga membuat program menjadi lebih mudah, karena memungkinkan pembuat untuk menggabungkan karakteristik umum kedalam objek orangtua/*parent* dan mewarisi karakteristik tersebut di dalam objek anak/*child*.

f. Aggregation

Agregasi adalah ketika sebuah objek terdiri dari gabungan dari benda-benda lain yang bekerja sama. Misalnya, mesin pemotong rumput adalah gabungan dari objek roda, mesin, pisai, dan sebagainya. Bahkan, objek mesin adalah gabungan dari berbagai objek. Kemampuan untuk menggunakan agregasi di dalam OOP merupakan fitur canggih yang memungkinkan implementasi proses bisnis dan model secara akurat di dalam suatu program.

2.9 Bahasa Pemrograman C#

Menurut Joseph Albahari & Ben Albahari (2012, p1-2), C# adalah bahasa pemrograman berorientasi objek yang berjenis aman (*type-safe*) dan untuk keperluan umum. Tujuan bahasa ini adalah produktivitas programmer. Untuk tujuan ini, bahasa ini mengimbangi kesederhanaan, ekspresif, dan kinerja.

C# pada dasarnya adalah bahasa berjenis aman (*type-safe*), yang berarti bahwa contoh dari tipe dapat berinteraksi hanya melalui protokol yang mereka definisikan, sehingga memastikan konsistensi internal masing-masing jenis ini.

C# adalah implementasi dari paradigma objek orientasi yang mencakup enkapsulasi (*encapsulation*), pewarisan (*inheritance*), dan polimorfisme (*polymorphism*). Enkapsulasi berarti menciptakan batas sekitar objek untuk memisahkan perilaku eksternal dari internal implementasi detail.

Beberapa fitur khas yang terdapat dalam C# dari sudut pandang orientasi objek, antara lain:

1. *Unified type system*

Hal yang mendasar dalam C# adalah unit enkapsulasi data dan fungsi yang disebut *type*. C# memiliki sistem tipe terpadu, dimana semua jenis pada akhirnya berbagi *type* yang umum. Ini berarti semua jenis, apakah mereka mewakili bisnis objek atau tipe primitif seperti nomor, berbagi set fungsi dasar yang sama.

2. *Classes and interfaces*

Dalam paradigma tradisional berorientasi objek, satu-satunya jenis tipe adalah kelas. Dalam C#, ada beberapa jenis lain dari tipe, salah satunya adalah antarmuka (*interface*). Sebuah antarmuka seperti kelas, kecuali bahwa itu hanya menggambarkan anggota.

3. *Properties, methods, and events*

Dalam paradigma berorientasi objek, semua fungsi adalah metode. Dalam C#, *methods* merupakan salah satu jenis anggota fungsi yang juga meliputi *properties* dan *event*. *Properties* adalah fungsi yang mengenkapsulasi bagian

dari suatu objek, seperti warna tombol atau tulisan dalam label. *Events* adalah fungsi yang mempermudah melakukan perubahan terhadap suatu objek.

2.10 *Game*

Menurut Schell (2008, p31) yang dikutip dari Elliot dan Brian, *game* adalah kegiatan sukarela yang dilakukan dan membentuk suatu sistem terkontrol, dimana ada sebuah kontes kekuatan, dibatasi oleh aturan-aturan untuk menghasilkan hasil yang tidak pasti menang atau kalahnya.

Menurut Schell (2008, p31) yang dikutip dari Costikyan, *game* adalah struktur interaktif yang memiliki nilai internal yang membutuhkan pemain untuk bertahan sampai ke tujuan.

Menurut Schell (2008, p34), terdapat 10 karakteristik yang harus dimiliki dalam sebuah *game*, yaitu:

1. *Game* dimainkan dengan kemauan sendiri.
2. *Game* mempunyai tujuan.
3. *Game* mempunyai konflik/masalah.
4. *Game* mempunyai aturan.
5. *Game* dapat menghasilkan kondisi menang atau kalah.
6. *Game* bersifat interaktif.
7. *Game* mempunyai tantangan.
8. *Game* dapat menciptakan nilai (*value*) internal di dalam *game* itu sendiri.
9. *Game* melibatkan *player*.
10. *Game* adalah sistem formal yang tertutup.

Dari 10 karakteristik di atas, Schell menyimpulkan definisi *game* adalah aktivitas pemecahan masalah (*problem solving*) dengan menggunakan pendekatan sikap yang menginginkan kesenangan.

Sedangkan menurut Rollings dan Adams (2003, p34), *game* merupakan bentuk hiburan partisipatif dan interaktif. Menonton televisi, membaca buku, dan menonton bioskop merupakan contoh hiburan pasif. Sebuah *game* adalah hal yang cukup rumit. Ketika seseorang memainkan *game*, mereka terhibur dengan berpartisipasi dengan aktif. Bentuk hiburan aktif ini perlahan-lahan membuat orang lebih cenderung memainkan *game* daripada menonton televisi, dikarenakan orang lebih menyukai interaksi dan keterlibatan yang diberikan oleh *game*. Sebuah *game* berlangsung pada dunia maya yang diatur oleh peraturan-peraturan. Peraturan-peraturan mendefinisikan aksi dan perpindahan yang diciptakan pemain di dalam *game* serta aksi-aksi yang tidak mereka ciptakan.

2.10.1 Elemen *Game*

Menurut Schell (2008, p41-43), terdapat empat elemen dasar dalam sebuah game, yaitu:

1. *Mechanics*

Mechanics adalah prosedur dan peraturan dalam sebuah *game*, dan juga menjelaskan tujuan dari sebuah *game*. Bagaimana pengguna dapat mencoba mencapai sesuatu atau melewatinya, serta yang terjadi bila pemain mencoba untuk mencapainya.

2. *Story*

Story menjelaskan apa yang terjadi di dalam *game* yang dibuat. Dapat berupa *linear* dan *pre-scripted*, atau bercabang. *Aesthetics* dapat membantu pembawa narasi dalam menceritakan kisahnya.

3. *Aesthetics*

Aesthetics menjelaskan bagaimana tampilan gambar dan suara. *Aesthetics* sangatlah penting di dalam *game*, karena elemen inilah yang langsung berinteraksi dengan pemain.

4. *Technology*

Technology di dalam *game* tidaklah harus selalu canggih, yang terpenting dari teknologi tersebut dapat memenuhi kebutuhan dari *game* yang dibuat.

2.10.2 *Genre Game*

Menurut Fullerton (2008, p415-421), *gameplay* dalam *game* dibedakan menurut *genre*-nya yang sekarang ini sudah bermacam-macam. *Genre* juga mempunyai tugas untuk membatasi para perancang *game* untuk dapat berkreasi dalam ide yang lebih spesifik. Tidak jarang juga di dalam suatu *game* dapat mengandung lebih dari satu *genre* yang disebut dengan *hybrid genre*. Berikut ini adalah jenis-jenis *genre* yang ada dalam *game*, yaitu:

1. *Action Games*

Action games melibatkan reaksi dari pengguna dalam memainkan sebuah *game*. *Game* berjenis *action* memberikan pengalaman secara *real-time* dengan menekankan pada keterbatasan waktu untuk melakukan tugas-tugas bersifat fisik.

2. *Strategy Games*

Strategy games biasanya melibatkan konsentrasi dan pengaturan akan sumber daya atau unit-unit. *Game* berjenis *strategy* biasa meliputi penaklukan, penjelajahan, dan pertukaran.

3. *Role-Playing Games*

Role-playing games meliputi menciptakan dan mengembangkan karakter dalam *game*. Sebelum memainkan *game*, pengguna membuat karakter dan sejalan dalam *game* maka karakter itu akan tumbuh dengan batasan yang sudah disediakan oleh *game* tersebut. *Massively multiplayer online role-playing games* (MMORPG) adalah pecahan dari *genre role-playing games*. Biasanya MMORPG dimainkan secara *online* dan juga dapat berinteraksi dengan pemain lain yang memainkan *game* tersebut.

4. *Sport Games*

Sport games adalah *game* simulasi bertemakan olahraga. Biasanya *sport games* mengadaptasi peraturan-peraturannya dari dunia asli. Tidak sedikit juga *sport games* dimodifikasi oleh perancang *game* menjadi *game* olahraga yang baru dan menarik.

5. *Racing/Driving Games*

Racing/driving games dipisah menjadi dua jenis, yaitu jenis *arcade* yang dapat dimainkan oleh semua umur dan *racing simulation* yang hanya dapat dimainkan oleh beberapa kalangan saja karena tampilan atau *gameplay* yang kompleks.

6. *Simulation/Building Games*

Simulation/building games biasanya berfokus pada pengaturan sumber daya yang ada, kemudian digabungkan dengan membangun sesuatu, baik yang dibangun itu perusahaan atau kota. Tidak seperti *strategy game* yang bersifat menakutkan, *simulation/building games* berisikan tentang pertumbuhan.

7. *Flight and Other Simulations*

Game berjenis ini biasanya menyediakan simulasi dengan berpatok pada dunia nyata, seperti menerbangkan pesawat, mengendarai tank, dan sebagainya. *Flight simulations* biasanya disukai oleh pengguna *game*, karena *flight simulations* membuat pengguna merasakan seperti menerbangkan pesawat di dunia nyata.

8. *Adventure Games*

Adventure games bersifat menjelajah, mengumpulkan barang, dan memecahkan puzzle. Biasanya juga pengguna disuruh untuk menyelesaikan misi di dalam *game* tersebut. Tidak seperti *role-playing game* dimana karakter di dalam *game* dapat bertumbuh, karakter dalam *adventure games* tidak bertumbuh dan tidak dapat dimodifikasi.

9. *Edutainment*

Game berjenis *edutainment* merupakan cara belajar yang menyenangkan. Dengan memasukkan cara pembelajaran di dalam *game*, pengguna dapat menemukan rasa senang dan juga mendapatkan pengetahuan ketika sedang memainkan *game* tersebut.

10. *Children's Games*

Children's games dibuat khusus untuk anak-anak berusia 2-12 tahun.

Children's games biasanya mengandung unsur pendidikan, tetapi yang dikhususkan adalah hiburan untuk anak-anak.

11. *Casual Games*

Casual games dibuat untuk dapat dinikmati oleh semua kalangan dan umur.

Casual games tidak mengandung unsur kekerasan di dalamnya, oleh karena itu *casual games* cocok dimainkan oleh semua kalangan dan umur.

2.10.3 *Game Balancing*

Mengacu pada Schell (2008, p172-200), *game balancing* dibagi menjadi 12 tipe. 12 tipe *game balancing* yang biasanya digunakan dalam pembuatan suatu *game*, yaitu:

1. *Fairness*

Tipe ini merupakan tipe *game balancing* yang paling sering digunakan dalam permainan papan seperti catur atau otello. Tipe ini memungkinkan setiap pemain untuk menang. Tidak ada pihak yang mendapat keuntungan atau kerugian yang signifikan. Untuk mengatasinya, digunakan symetrical, yaitu setiap pemain mendapatkan sumber daya dan kekuatan yang sama. Yang membedakan hanya bagaimana cara pemain untuk menggunakan dan mengelola sumber daya tersebut. Tetapi, ada juga *game* yang menggunakan *Asymetrical*. Berikut ini adalah beberapa alasan mengapa menggunakan *asymetrical*:

a. Untuk membuat lebih real

Ada beberapa *game* yang membuat dengan tema perang dunia kedua, dimana dua kubu yang berperang adalah Axis dan Allied. Dalam sejarah, pihak yang Axis mengalami kekurangan sumber daya manusia, dan Allied memiliki keuntungan sumber daya manusia dan teknologi. Hal ini membuat *game symmetrical* tidak digunakan dalam *game* tersebut.

b. Agar pemain dapat mengeksplorasi *game*

Mengeksplorasi *gameplay* merupakan hal yang sangat disukai pemain, sehingga banyak *game* yang menggunakan *asymmetrical*. Contohnya saja, dalam sebuah *game fighting* ada 10 pilihan petarung. Pemain bebas mengkombinasikan dua petarung untuk menaklukan misi atau *stage* khusus yang membutuhkan petarung-petarung tertentu.

c. Personalisasi

Setiap orang mempunyai kemampuan yang berbeda dalam memainkan setiap *game*. Karena itu, jika pemain diberikan pilihan karakter, kekuatan dan sumber daya yang berbeda, pemain akan bisa memainkan *game* tersebut dengan baik.

d. Membuat setiap *level* lebih menantang

Ketika bermain dengan komputer, pemain akan mudah memenangkan *game* jika sudah hafal dengan gerak dan respon komputer. Untuk itu, setiap memenangkan *level*, kemampuan komputer akan ditambah, agar *game* lebih menantang. Selain itu, pemain yang memiliki kemampuan lebih, bisa memilih lawan yang seimbang, begitu pula dengan pemain yang masih pemula.

e. Membuat situasi yang menarik

Dalam sebuah *game*, akan lebih menarik jika ada perbedaan kemampuan dan perbedaan objektif yang harus dipenuhi agar menang. Setiap pemain akan berpikir bagaimana cara dan strategi yang tepat agar bisa memenangkan *game* tersebut. Selain itu terdapat kelebihan dan keuntungan dalam setiap pilihan pemain. Inilah yang membuat permainan menjadi semakin menarik.

2. *Challenge vs Success*

Membuat pemain tetap memainkan *game* merupakan hal penting. *Game* dengan kesulitan tinggi dapat membuat pemain frustrasi dan stress, tetapi *game* yang mudah dimenangkan akan membuat pemain cepat bosan karena terlalu mudah. Setiap pemain mungkin memiliki persepsi yang berbeda mengenai setiap *game*, apakah terlalu mudah atau terlalu sulit. Berikut ini adalah teknik-teknik yang digunakan untuk membuat *game* tetap *balance*:

a. Menambah kesulitan setiap kemenangan

Akan membosankan jika setiap *level* permainan memiliki kesulitan yang sama. Hal tersebut akan membuat pemain cepat bosan. Karena itu, setiap *level* dibuat memiliki tantangan yang berbeda agar pemain semakin tertarik untuk memainkan *game* tersebut.

b. Melewati tahap mudah dengan cepat

Pemain biasanya cepat menguasai teknik permainan pada bagian yang mudah. Karena itu sebaiknya tahapan yang mudah diberikan waktu yang singkat agar cepat sampai ke tahapan yang sulit dan menantang pemain.

c. Membuat lapisan tantangan

Dalam sebuah *game* biasanya diberikan suatu *grade* jika telah menyelesaikan suatu misi. Jika mendapat *grade* kecil, misi tersebut harus diulang. Ada batasan *grade* yang harus dipenuhi agar bisa melewati misi tersebut. Bahkan jika mencapai *grade* tertentu, akan muncul misi khusus. Hal ini membuat setiap pemain ingin bermain terus menerus agar mendapatkan *grade* yang diinginkan.

d. Bisa memilih tingkat kesulitan

Biasanya setiap *game* sekarang ini telah memiliki pilihan tingkat kesulitan. Hal ini membuat pemain bisa menyesuaikan kesulitan dengan kemampuan yang dimilikinya.

e. *Testing* dengan berbagai macam pemain

Biasanya, *testing* sebuah *game* hanya ditujukan kepada orang-orang yang sering bermain *game*, atau dengan orang-orang yang sama sekali belum pernah bermain *game*. Hal ini membuat *game* tersebut tidak cocok untuk dimainkan semua pemain. Untuk itu, perlu dilakukan *testing* dengan banyak orang, baik yang sering memainkan *game*, atau yang belum pernah bermain *game* sama sekali.

3. *Meaningful Choice*

Sebuah *game* yang bagus memiliki pilihan yang sangat berarti. Pilihan tersebut harus memiliki hubungan dengan *game* yang kita mainkan. Sebagai contoh dalam permainan *racing*, ada 10 mobil dan setiap mobil memiliki kelebihan dan kekurangannya, sehingga ketika digunakan akan memberikan perbedaan dari mobil lainnya.

4. *Skill vs Chance*

Kemampuan dan kesempatan adalah hal yang bertolak belakang dalam *game*. Terlalu banyak menggunakan kemampuan akan membuat pemain yang memiliki kemampuan bagus terus memenangkan permainan. Sebaliknya, *game* yang terlalu banyak menggunakan kesempatan akan membuat pemain yang memiliki keberuntungan tinggi terus menang. Untuk itu diperlukan *balancing* dalam hal ini. Sebaiknya sebuah *game* menggunakan dua hal tersebut.

5. *Head vs Hands*

Tipe *game balancing* ini mengacu pada seberapa besar *game* tersebut menggunakan aktivitas otak dan tangan. Sekarang ini banyak *game* yang menggunakan dua hal tersebut dalam *gameplay*-nya.

6. *Competition vs Cooperation*

Kompetisi dan kerja sama merupakan hal yang dasar dalam sebuah *game*. Untuk berada dalam tingkatan khusus, diperlukan sebuah kompetisi untuk menentukan siapa yang terbaik. Sedangkan kerja sama diperlukan untuk bertahan hidup dan bersosialisasi.

7. *Short vs Long*

Salah satu hal penting dalam *game balancing* adalah jangka waktu permainan. Jika waktu permainan terlalu sebentar, pemain tidak bisa mengembangkan permainan secara bebas. Sedangkan jika permainan terlalu lama, pemain akan bosan atau mungkin tidak akan memainkan *game* tersebut karena membutuhkan waktu lama untuk menyelesaikannya.

8. *Rewards*

Tidak mungkin seorang pemain mau bermain sebuah *game* dalam waktu yang cukup lama tanpa sebuah penghargaan atau hadiah. Berikut ini adalah beberapa hadiah atau penghargaan yang diterima pemain secara umum:

a. Pujian

Pujian merupakan bentuk hadiah yang paling mendasar. Setiap orang akan senang jika diberikan pujian ketika menyelesaikan sesuatu dengan baik. Begitu pula dalam sebuah *game*.

b. Poin

Hampir dalam semua *game*, terdapat poin yang didapatkan pemain dengan berbagai cara. Poin ini bisa berperan dalam *game* sebagai mata uang, atau sebagai tolak ukur keberhasilan seorang pemain.

c. Penambahan waktu bermain

Dalam beberapa *game*, banyak yang menyediakan waktu bermain lebih dengan berbagai cara. Dalam beberapa *game*, jika menyentuh poin tertentu, pemain akan mendapatkan nyawa tambahan.

d. Sebuah jalan baru

Seorang pemain akan senang jika bisa mengeksplorasi sebuah *game*. Karena itu, banyak *game* yang menyediakan hadiah seperti ini. Ketika mereka menyelesaikan misi tertentu, mereka akan mendapatkan sebuah misi tertentu atau sebuah akses khusus ke *level* tertentu.

e. Sebuah tontonan

Dalam beberapa *game*, kita bisa mendapatkan beberapa video atau animasi yang hanya bisa didapatkan dengan menyelesaikan misi tertentu. Tidak sedikit pemain *game* yang senang dengan hal ini.

f. Berekspresi

Beberapa pemain lebih suka berekspresi dalam sebuah *game* dengan pakaian dan dekorasi tertentu. Contohnya saja dalam *game* “The Sims”, jika pemain mempunyai uang yang banyak, pemain dapat membeli dekorasi yang bagus.

g. Kekuatan

Menjadi kuat adalah impian setiap orang di dunia, tetapi di dunia *game*, hal tersebut bukanlah mustahil. Beberapa *game* memberikan kekuatan sebagai hadiah sebuah permainan. Contohnya Mario dalam Mario Bros yang menjadi kuat setelah memakan jamur.

h. Sumber daya

Beberapa *game* memberikan mata uang virtual yang bisa digunakan untuk membeli barang-barang di dunia *game* tersebut. Hadiah uang virtual ini sangat populer, terlebih di dunia MMORPG.

i. Penyelesaian

Penyelesaian semua misi dan *level* dalam sebuah *game* adalah hadiah yang paling umum dalam dunia *game*. Setelah menyelesaikan *game* atau semua *level* dalam *game* tersebut, kita akan merasa puas.

9. *Punishment*

Sebuah *game* memang seharusnya memberikan kesenangan, bukan hukuman. Tetapi memberikan hukuman sebagai *game balancing* bukan tidak berdasarkan alasan. Berikut adalah alasan mengapa menggunakan hukuman sebagai *game balancing*:

- a. Memberikan nilai endogen
- b. Mengambil resiko menyenangkan
- c. Memberikan hukuman membuat *game* semakin menantang

Berikut ini adalah hukuman yang sering diberikan dalam sebuah *game*:

- a. Malu
- b. Poin berkurang
- c. Waktu bermain berkurang
- d. Tidak bisa bermain
- e. Kembali ke suatu titik
- f. Hilangnya kekuatan
- g. Sumber daya berkurang

10. *Freedom vs Controlled Experience*

Dalam sebuah *game*, kebebasan dan control terhadap permainan merupakan hal yang penting. Tetapi kebebasan dan control harus dibatasi sesuai dengan kebutuhan. Sebuah *game* dengan kebebasan penuh akan membuat pemain menjadi bosan.

11. *Simple vs Complex*

Dalam setiap *game*, kesederhanaan dan kompleksitas sebuah *game* sangat bertolak belakang. Sebuah *game* yang sederhana seringkali dikritik sebagai

sebuah *game* yang membosankan, tetapi seringkali *game* yang sederhana menjadi sebuah *game* yang elegan, begitu pula dengan *game* kompleks. Untuk itu, ada baiknya desainer *game* memperhatikan seberapa kompleks dan sederhana sebuah *game*.

12. *Detail vs Imagination*

Tampilan sebuah *game* menyita banyak perhatian pemain. Tampilan dan latar yang bagus dan menarik akan membuat pemain menyukai *game* tersebut.

2.10.4 *Game Design*

Menurut Rollings & Adams (2003, p4), *game design* adalah proses dari:

- a. Membayangkan permainan
- b. Menentukan cara permainan tersebut bekerja
- c. Menjelaskan unsur-unsur yang membentuk permainan tersebut
- d. Mengirimkan informasi kepada kelompok yang akan membuat *game* tersebut

Menurut Rollings & Adams (2003, p8-13), *game design* dipecah menjadi tiga bagian inti, yaitu:

- a. *Core Mechanics*

Aturan yang mendefinisikan aturan dari permainan yang membentuk mekanisme inti permainan, atau aturan dasar *gameplay*. Mekanisme inti permainan merupakan terjemahan dari visi perancang ke dalam sekumpulan peraturan yang dapat diinterpretasikan oleh komputer.

- b. Cerita dan narasi

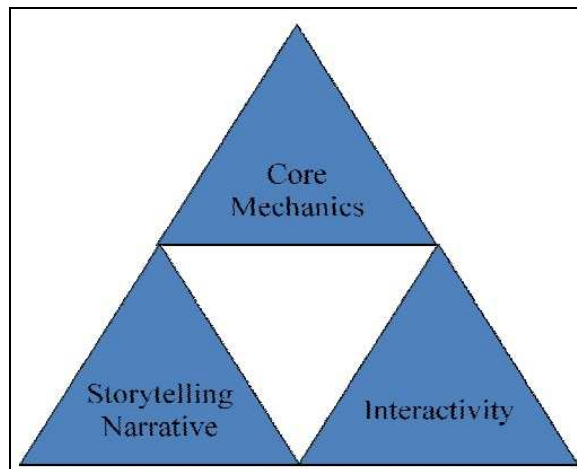
Seluruh *game* mempunyai cerita. Namun yang membedakan cerita antara *game* tersebut adalah kedalaman dan kompleksitas cerita. Narasi berarti

cerita yang dibawakan oleh penulis atau designer kepada pemain permainan anda. Narasi tidak bersifat interaktif namun bagian dari presentasi cerita.

c. Interaktivitas

Interaktivitas adalah sebuah cara pemain untuk melihat, mendengar dan beraksi dalam dunia permainan. Topik-topik yang mencakup interaktivitas antara lain: grafis, suara, tampilan antarmuka, serta segala sesuatu yang mempresentasikan pengalaman ketika bermain. Interaktivitas telah menjadi kata kunci yang menghubungkan *game* dengan komputer.

Aturan dasar yang diterapkan pada *game design* ini dapat digambarkan sebagai berikut:



Gambar 2.17 Aturan Dasar *Game Design* (Sumber: Rollings and Adams, 2003, p9)

2.11 *Role Playing Games (RPG)*

Menurut Ernest Adams (2010, p455-469), RPG adalah suatu jenis permainan, dimana pemain mengontrol satu atau lebih karakter, biasanya dirancang oleh pemain, dan membimbing mereka melalui serangkaian *quests*

dikelola oleh komputer. Kemenangan terdiri dari menyelesaikan *quests*. Pertumbuhan dan pengembangan karakter dalam kekuatan dan kemampuan adalah fitur kunci dari *genre* ini. Tantangan umum termasuk tempur taktis, logistik, pertumbuhan ekonomi, eksplorasi, dan memecahkan teka-teki.

a. *Progression/story*

RPG pasti memiliki sebuah cerita utama. Cerita dalam *game* RPG biasanya dibagi menjadi beberapa *chapter/bab/episode*. Dalam setiap *chapter*, pemain biasanya dibawa ke lokasi baru atau kembali ke lokasi yang sudah pernah didatangi untuk melakukan sesuatu yang baru kita sebut saja itu *quest*. Dalam setiap *chapter*, pemain dapat memiliki *side quest* yang biasanya berpengaruh terhadap *story line game* tersebut, biasanya tidak ada dampak apa-apa bila kita tidak menjalankan *side quest* tersebut, kecuali dalam beberapa *game* kita harus menyelesaikan semua *quest* termasuk *side quest* untuk mendapatkan *good ending*, atau kita bisa sebut *perfect ending*. *Story line* biasanya berbentuk linear dengan beberapa cabang *ending*, tergantung kepada pemain dalam menjalankan *story* tersebut. Dan pada akhir setiap *chapter* hampir setiap *game* pasti akan menemukan musuh yang kuat sebagai *boss* akhir *chapter*. Dalam setiap *chapter*, pemain harus memikirkan tentang kekuatan karakter atau grup dalam menjalankan *quest-quest* di *chapter* selanjutnya.

b. *Gameplay*

Gameplay dalam RPG sangat banyak, beberapa contohnya adalah seperti *tactical combat*, *stealth operation*, *conversation*, *buying and selling*, dan *inventory management*. Tetapi sebagian besar *game* RPG memiliki *gameplay* eksplorasi dan pertarungan, pertukaran, dan memiliki *inventory*. Eksplorasi

dan pertarungan adalah dimana karakter dapat menjelajahi sebuah tempat baru atau tempat yang sudah pernah didatangi untuk melakukan sesuatu yang baru. Pertarungan adalah dimana karakter biasanya melawan monster atau *race-race* tertentu untuk mendapatkan *reward*. Pertukaran atau *trade* adalah sistem beli atau jual di dalam *game*. Kebanyakan game RPG menyediakan NPC untuk melakukan aksi *trade* ini, contohnya *blacksmith*, *healer*, dan lain-lain. *Inventory* adalah sistem tas atau tempat untuk menaruh *item*. Biasanya berupa *backpack* atau dapat bisa berupa *array of box*.

c. *Magic*

Magic atau yang sejenisnya adalah hal yang tidak dapat dipisahkan dari *game* RPG. *Magic* juga menjadi hal yang khusus di *game* RPG. Kehadiran *magic* di *game* RPG membuat *game* RPG menjadi lebih menarik. *Magic* adalah hal fiksi yang tidak ada di dunia nyata. Biasanya *magic* juga dibatasi dengan *class-class* tertentu dan kemampuan-kemampuan tertentu.

d. *Character Development*

Status *attribute* adalah keadaan sekarang sebuah karakter. Biasanya status *attribute* sangat sering berubah-ubah sesuai dengan unsur-unsur yang mempengaruhinya. Status *attribute* akan menentukan apakah karakter tersebut dapat atau tidak melakukan tindakan-tindakan, dan jika dapat, status *attribute* akan menentukan seberapa baik, seberapa kuat, seberapa cepat dan probabilitas keberhasilannya. *Design* karakter yang pertama adalah *race*, *race* adalah suku atau ras-ras/jenis-jenis makhluk hidup yang ada di sebuah *game*. *Game* RPG memiliki beberapa *fantasy race* yang biasanya ada di dalam sebuah *game*, sebagai contoh *dwarf*, *elves*, raksasa, manusia, dan lain-

lain. Pada setiap *race* biasanya tetap memiliki jenis kelamin pria dan wanita. *Design* karakter selanjutnya adalah karakter *class* atau beberapa *game* menyebutnya *job*. *Class* di setiap karakter berpengaruh terhadap beberapa aksi yang dapat dilakukan karakter tersebut dan juga menentukan *skill-skill* dan spesialisasi karakter. Karakter *class* juga menentukan *grow rate* dari karakter dan kemampuan fisik dari karakter tersebut. Karakter di RPG juga memiliki sifat-sifat unik yang ditentukan oleh RPG yang sesuai dengan cerita yang ada.

2.12 *Game Edukasi*

Menurut Mz & Sy (2008), *Game Based Learning* (GBL) atau *game* edukasi adalah sebuah paradigma yang menggunakan *game* sebagai media untuk menyampaikan isi materi. *Game* edukasi adalah semua tentang memanfaatkan kekuatan permainan komputer untuk memikat dan melibatkan pengguna untuk tujuan spesifik, seperti mengembangkan pengetahuan baru dan keterampilan. *Game* edukasi juga didefinisikan sebagai aplikasi yang menggunakan karakteristik video dan *game* komputer untuk membuat pengalaman menarik dan mendalam dari belajar untuk memberikan tujuan belajar yang spesifik, hasil, dan pengalaman.

2.13 *Game Engine*

Menurut Goldstone (2009, p1), *game engine* adalah kerangka dari *game* yang akan kita buat. *Game engine* yang membuat keputusan dalam menentukan *frame* sampai menentukan *artwork* yang ada di dalam *scene*. *Modern 3D game*

engine sekarang banyak memiliki *script-script* cermat sehingga dapat memenuhi kebutuhan yang ada.

2.14 Unity3D

Menurut Goldstone (2009, p14), Unity3D membuat produksi *game* menjadi lebih mudah dengan memberikan beberapa logika untuk membangun skenario *game* yang sudah dibayangkan. Unity3D adalah salah satu *game engine* yang mudah digunakan, hanya membuat objek dan diberikan fungsi untuk menjalankan objek tersebut. Dalam setiap objek mempunyai variabel, variabel inilah yang harus dimengerti supaya dapat membuat *game* yang berkualitas.

Berikut ini adalah bagian bagian yang terdapat dalam Unity3D:

a) *Assets*

Assets adalah tempat penyimpanan dalam Unity3D. Suara, gambar, video, tekstur, dan semua objek yang akan dipakai dalam Unity3D disimpan dalam *Assets*.

b) *Scenes*

Scenes adalah sebuah area yang berisi konten-konten dalam *game*, seperti membuat tampilan *level*, membuat menu, tampilan *loading*, dan lain-lain.

c) *Game objects*

Ketika sebuah objek dipindahkan dari *assets* menuju *scenes* maka objek tersebut menjadi *game object*, dimana objek tersebut dapat digerakkan, dirotasi, dan dibentuk sesuai dengan keinginan.

d) *Components*

Components adalah bagian dari *game objects*, dimana *components* membuat suatu reaksi baru yang dapat dilakukan terhadap *game objects*.

e) *Script*

Script adalah sebuah cara untuk menulis bahasa pemrograman dan membuat aksi dalam *game* yang dirancang. Pada Unity3D terdapat tiga bahasa pemrograman, yaitu: C#, Javascript, dan BOO. Penggunaan *script* pada Unity menggunakan program khusus yang dinamakan Mono yang sudah terintegrasi dengan Unity3D.

f) *Prefabs*

Prefabs adalah tempat penyimpanan satu jenis *game objects*, sehingga dapat dengan mudah diperbanyak. Tujuan utama *prefabs* adalah agar tidak perlu membuat objek yang sudah pernah dibuat.