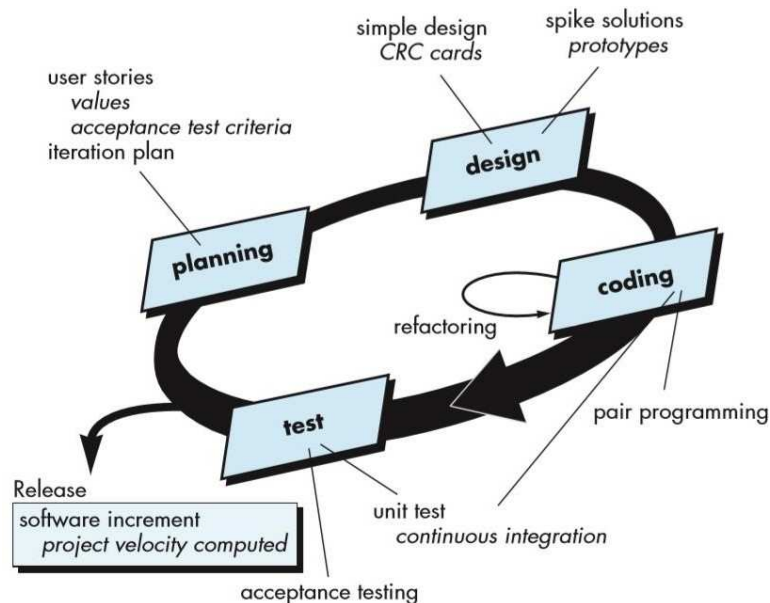


BAB 2

TINJAUAN PUSTAKA

2.1 *Extreme Programming (XP)*

Menurut Pressman & Maxim (2015, p. 72), *Extreme Programming* atau yang biasa disebut dengan XP merupakan salah satu metode yang paling banyak digunakan untuk pengembangan aplikasi dari pendekatan *Agile Software Development*. *Extreme Programming* menggunakan pendekatan berorientasi objek sebagai paradigma pengembangan pilihannya yang mencakup seperangkat aturan dan praktik yang terjadi dalam konteks empat aktivitas *framework*, yaitu *planning* (perencanaan), *design* (perancangan), *coding* (pengkodean), dan *testing* (pengujian). Proses pengembangan *software* dengan XP diilustrasikan pada Gambar 2.1.



Gambar 2.1 Proses *Extreme Programming*

(Pressman & Maxim, 2015, p. 72)

Planning. Kegiatan perencanaan ini dimulai dengan tahap *listening* sebagai kegiatan pengumpulan *requirement* (kebutuhan) yang memungkinkan anggota teknis dari tim XP memahami konteks bisnis untuk *software* dan untuk mendapatkan gambaran secara menyeluruh mengenai *output* serta fitur dan fungsionalitas utama yang dibutuhkan. Kegiatan *listening* yang dilakukan menghasilkan kumpulan “*stories*” (disebut juga *user stories*) yang mendeskripsikan *output*, fitur, dan fungsionalitas yang

diperlukan untuk pengembangan perangkat lunak. Selama proses pengembangan, *user* dapat menambahkan *stories*, mengubah *stories* yang sudah ada, memisahkan *stories*, atau menghapusnya. Setelah itu, tim XP akan mempertimbangkan dan menyesuaikan kembali sesuai rencana.

Design. Tahap desain pada *extreme programming* secara ketat mengikuti prinsip KIS (*keep it simple*) yang memiliki arti desain yang sederhana lebih diutamakan dari pada desain yang kompleks. Jika masalah desain yang sulit ditemui sebagai bagian dari desain *story*, XP merekomendasikan pembuatan prototipe secara langsung dari bagian desain tersebut. Selain itu, penggunaan desain dengan fungsi ekstra (dengan asumsi akan dibutuhkan pada masa mendatang) tidak disarankan karena desain yang diimplementasi mengacu pada *user stories* yang disepakati (tidak kurang, tidak lebih).

Coding. Setelah penulisan *stories* dan perancangan selesai, pengembang tidak langsung melakukan *code* tetapi membuat sekumpulan *unit tests* untuk menguji setiap *stories* yang ada. Setelah itu, pengembang berfokus kepada apa yang harus diimplementasikan untuk berhasil melalui setiap *unit tests* yang telah dibuat. Setelah selesai pembuatan *code*, dapat dilakukan pengujian secara langsung untuk mendapatkan umpan balik ke para pengembang. Konsep utama dari aktivitas pengkodean XP adalah *pair programming*. XP merekomendasikan dua orang untuk bekerjasama dalam pengembangan aplikasi. Hal ini dapat membantu mekanisme untuk pemecahan masalah secara langsung (*real-time problem solving*) karena dua kepala lebih baik dari satu serta memastikan kualitas secara langsung (*real-time quality assurance*) dengan *code* yang ditinjau setelah pembuatan.

Testing. Dalam pelaksanaannya, perlu dilakukan *individual unit tests* untuk validasi pengujian sistem. Hal ini dapat membantu pengembang mengindikasikan kesalahan yang ditemukan secara bertahap dan memberikan peringatan dini serta memperbaikinya. Setelah itu, *user acceptance testing* dilakukan untuk menguji fitur dan fungsionalitas sistem secara keseluruhan sesuai dengan skenario *testing* yang ada.

2.2 Unified Modelling Language

UML (*Unified Modelling Language*) merupakan sebuah standar konstruksi dan notasi model yang didefinisikan dengan *Object Management Group* (OMG), sebuah organisasi standar untuk pengembangan sistem. Dengan menggunakan UML, seorang *analyst* dan *end user* dapat memberikan gambaran dan memahami berbagai diagram

spesifik yang digunakan dalam proyek pengembangan sistem. Model grafis yang digunakan dalam pengembangan sistem digambar sesuai notasi yang ditentukan, seperti *use case diagram*, *class diagram*, *sequence diagram*, *communication diagram*, dan *state machine diagram*.

Dalam UML, tidak ada standar dalam penggunaannya, sehingga diagram untuk setiap perusahaan dapat bervariasi. (Satzinger, Jackson, & Burd, 2012, p. 46)

2.2.1 Use Case Diagram

Use Case Diagram merupakan model UML yang digunakan untuk menampilkan setiap *use cases* (aktifitas yang dilakukan oleh sistem sebagai tanggapan atas permintaan pengguna) secara grafis dan hubungannya dengan *user*. (Satzinger, Jackson, & Burd, 2012)

Adapun komponen dalam pembuatan *use case diagram* menurut Satzinger, Jackson, & Burd (2012) diantaranya:

1. Actor

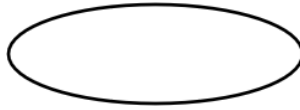
Actor adalah peranan yang melekat pada pengguna sistem. *Actor* selalu berada diluar batas sistem namun dapat menjadi bagian dari manual sistem. Dalam beberapa kasus, *actor* juga bisa berupa sistem lainnya maupun alat yang menerima *service* dari suatu sistem. (Satzinger, Jackson, & Burd, 2012, p. 72)



Gambar 2.2 Notasi Actor

2. Use Case

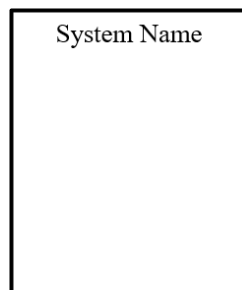
Use Case direpresentasikan dalam bentuk oval dan di dalamnya tertulis nama atau yang berupa kegiatan dari sistem (Satzinger, Jackson, & Burd, 2012, p. 69)



Gambar 2.3 Notasi *Use Case*

3. *Automation Boundary*

Mendefinisikan batasan antara bagian yang terkomputerisasi dari aplikasi dengan orang yang mengoperasikan aplikasi. *Automation Boundary* ditampilkan dengan bentuk persegi panjang yang berisi *use cases*. (Satzinger, Jackson, & Burd, 2012, p. 81)



Gambar 2.4 *Automation Boundary*

4. *Includes*

Mengidentifikasi bahwa suatu *use case* menggunakan *use case* lainnya untuk menjalankan fungsionalitasnya. Sering kali *includes* dikenal juga sebagai *user relationship*. (Satzinger, Jackson, & Burd, 2012, p. 82)



Gambar 2.5 Notasi <<*include*>>

5. *Association Relationship*

Suatu garis yang menghubungkan antara *actor* dengan *use case* yang menunjukkan *actor* yang berpartisipasi dalam *use case*.



Gambar 2.6 *Association Relationship*

2.2.2 Use Case Description

Use Case Description merupakan deskripsi detail proses dari setiap *use case*. Dengan menggunakan *use case description*, suatu interaksi dapat dijelaskan lebih detail untuk memperjelas beragam variasi proses bisnis dalam satu *use case*. Variasi proses bisnis yang membentuk sekumpulan aktivitas internal yang unik dalam suatu *use case* disebut skenario.

Berdasarkan penerapannya, *use case* dapat dideskripsikan dalam satu kalimat berisi gambaran singkat terhadap *use case* yang disebut *Brief Use Case Description*.

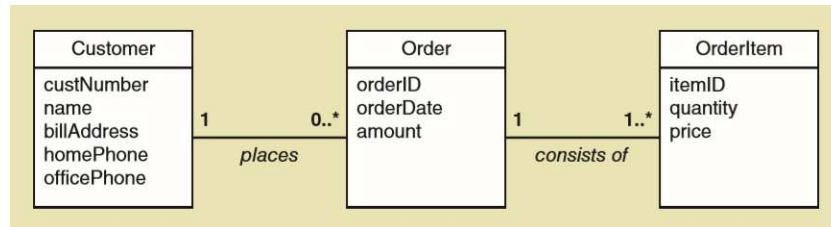
Tabel 2.1 Use Cases dan Brief Use Cases Description

<i>Use Case</i>	<i>Brief use case description</i>
<i>Create customer account</i>	<i>User/actor enters new customer account data, and the system assigns account number, creates a customer record, and creates an account record.</i>
<i>Look up customer</i>	<i>User/actor enters customer account number, and the system retrieves and display customer and account data.</i>
<i>Process account adjustment</i>	<i>User/actor enters order number, and the system retrieves customer and order data; actor enters adjustment amount, and the system creates a transaction record for the adjustment</i>

2.2.3 Class Diagram

Class Diagram adalah sebuah diagram yang terdiri dari *class* (yaitu, kumpulan objek) pada sistem dan hubungan yang ada di antar *class* satu dengan *class* lainnya. Pada *class diagram*, terdapat bentuk persegi panjang yang mewakili *class* dan garis – garis yang menghubungkan persegi panjang menunjukkan asosiasi antar *class* dengan beberapa bagian. Bagian atas berisi nama dari *class*, bagian keduanya berisi *attribute* dari *class* yang merupakan semua objek di dalam *class* yang memiliki nilai, dan bagian paling bawah berisi

daftar *method* dari *class*. *Class Diagram* digambarkan dengan menunjukkan *class* dan asosiasi hubungan antar *class*. Asosiasi hubungan antar *class* direpresentasikan dengan *multiplicity*. (Satzinger, Jackson, & Burd, 2012, p. 101)



Gambar 2.7 Class Diagram Sederhana

(Satzinger, Jackson, & Burd, 2012, p. 102)

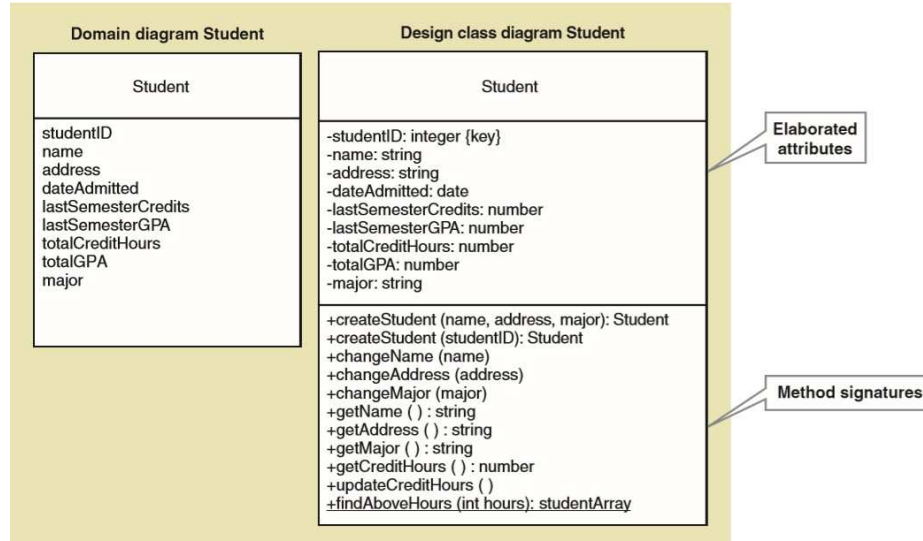
UML memiliki suatu notasi khusus yang disebut *stereotype* untuk merancang *class diagram* dengan tipe tertentu. *Stereotype* adalah salah satu cara untuk mengkategorikan suatu elemen model sebagai tipe tertentu. Selain itu, *stereotype* juga memperjelas definisi dasar dari model elemen dengan menunjukkan karakteristik spesial yang ingin diperlihatkan. Notasi untuk *stereotype* adalah nama dari tipe seperti berikut `<<controller>>`. (Satzinger, Jackson, & Burd, 2012, p. 309)

Notasi untuk komponen-komponen pada *Class Diagram* terdiri dari tiga bagian. Bagian pertama berisi nama dari *class* dan *stereotype class* dan *parent class*. Bagian kedua berisi detail dari *attribute* dan *method* dari *class*. (Satzinger, Jackson, & Burd, 2012, p. 310)

Format yang digunakan untuk mendefinisikan setiap *attribute* terdiri dari:

1. *Visibility* menunjukkan apakah *attribute* dapat diakses secara langsung oleh objek lain. *Visibility* ditulis sebelum nama *attribute*, seperti tanda *plus* yang menandakan *attribute* bersifat *public* dan tanda *minus* yang menandakan *attribute* bersifat *private*.
2. Nama *attribute*.
3. *Type-expression* dari *attribute* (seperti *character*, *string*, *integer*, dan *date*).
4. *Initial-value*, jika diperlukan.
5. *Property* (terletak di antara kurung kurawal), seperti {*key*}, jika diperlukan.

Bagian ketiga memuat informasi untuk *method signature*. *Method signature* menunjukkan semua informasi yang dibutuhkan untuk menjalankan atau memanggil *method*. (Satzinger, Jackson, & Burd, 2012, p. 310)



Gambar 2.8 Class Diagram dengan Domain Class dan Design Class

(Satzinger, Jackson, & Burd, 2012, p. 305)

Format yang digunakan untuk mendeskripsikan *method* terdiri dari:

1. *Method visibility*
2. *Method name*
3. *Method parameter list* (argument yang masuk)
4. *Return type-expression* (tipe return parameter dari *method*)

2.2.4 Activity Diagram

Activity Diagram mendeskripsikan berbagai jenis aktivitas baik yang dilakukan oleh *user* maupun sistem dalam suatu *workflow*. *Workflow* sendiri merupakan urutan dari langkah – langkah pemrosesan yang bertanggungjawab sepenuhnya atas transaksi bisnis atau permintaan *customer*. (Satzinger, Jackson, & Burd, 2012, p. 57)

Berikut ini adalah beberapa notasi yang digunakan dalam *activity diagram* (Satzinger, Jackson, & Burd, 2012, p. 58) diantaranya:

1. *Start Activity*: Lingkaran hitam yang menandakan notasi awal dari suatu *flow*.



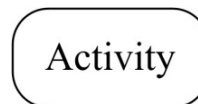
Gambar 2.9 Notasi *Start Activity*

2. *End Activity*: Lingkaran hitam dengan luaran putih sebagai notasi akhir dari suatu *flow*.



Gambar 2.10 Notasi *End Activity*

3. *Activity*: Persegi panjang dengan sudut tumpul sebagai notasi aktifitas yang dilakukan baik yang dilakukan aktor maupun sistem.



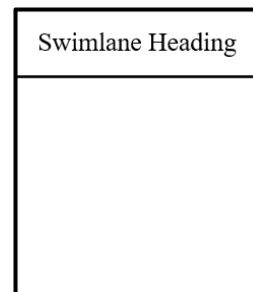
Gambar 2.11 Notasi *Activity*

4. *Transaction Arrow*: Tanda panah sebagai notasi perpindahan antara aktifitas satu dengan yang lainnya.



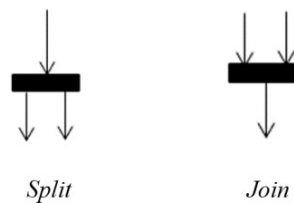
Gambar 2.12 Notasi *Transacion Arrow*

5. *Swimlane*: Persegi panjang dengan sudut lancip yang membatasi aktifitas antara aktor dengan sistem atau aktor lainnya. *Swimlane heading* merepresentasikan suatu *agent* yang melakukan aktifitas.



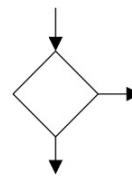
Gambar 2.13 Notasi *Swimlane*

6. *Synchronization Bar*: Persegi panjang yang menjadi pemisahan (*split*) alur ke beberapa alur yang berjalan bersamaan atau penggabungan (*join*) beberapa alur yang berjalan bersamaan menjadi satu alur.



Gambar 2.14 Notasi *Synchronization Bar*

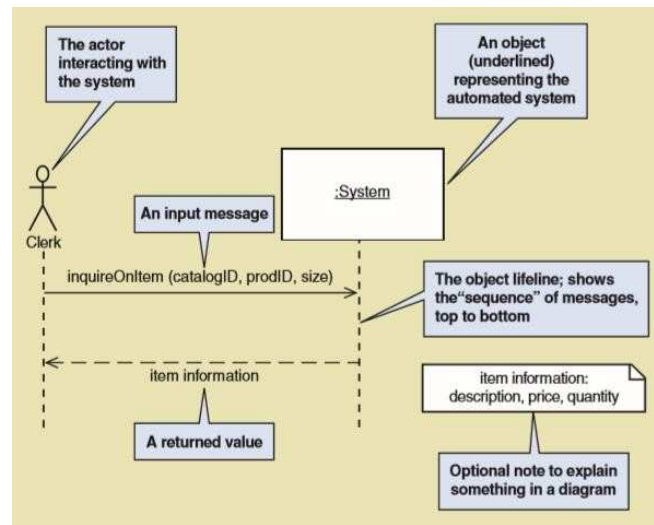
7. *Decision Activity*: Belah ketupat sebagai notasi percabangan aktifitas berdasarkan kondisi tertentu.



Gambar 2.15 Notasi *Synchronization Bar*

2.2.5 *Sequence Diagram*

Sequence Diagram digunakan untuk mendeskripsikan alur informasi masuk dan keluar dari sistem otomatis. Alur informasi mengidentifikasi interaksi baik antara *actor* dan sistem maupun antar internal objek *system*. (Satzinger, Jackson, & Burd, 2012, p. 126)

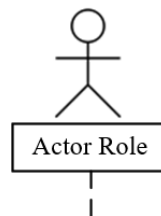


Gambar 2.16 Composition Relationship

(Satzinger, Jackson, & Burd, 2012, p. 126)

Berikut ini adalah notasi yang digunakan dalam *sequence diagram* (Satzinger, Jackson, & Burd, 2012)

1. *Actor*: *Stick figure* merepresentasikan pengguna yang berinteraksi dengan sistem.



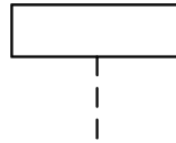
Gambar 2.17 Notasi Actor pada Sequence Diagram

2. *Activation Lifeline*: Batang sebagai notasi waktu suatu method dieksekusi.



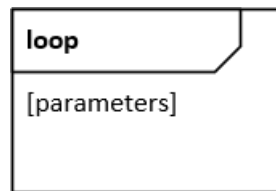
Gambar 2.18 Notasi Actor pada Sequence Diagram

3. *Object Lifeline*: Garis putus – putus sebagai notasi garis waktu suatu objek.



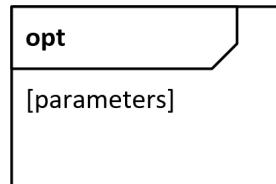
Gambar 2.19 Notasi *Object Lifeline*

4. *Loop Fragment*: Bagian dari diagram yang dijalankan secara berulang.



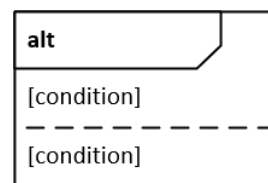
Gambar 2.20 Notasi *Loop Fragment*

5. *Optional Fragment*: Bagian dari diagram yang eksekusinya.



Gambar 2.21 Notasi *Optional Fragment*

6. *Alternative Fragment*: Bagian dari *sequence* diagram yang bekerja seperti conditional checking (if – then – else).



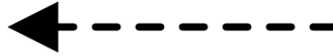
Gambar 2.22 Notasi *Alternative Fragment*

7. *Message*: Panah sebagai notasi pesan yang dikirimkan dari objek.



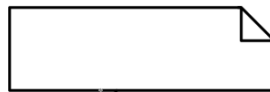
Gambar 2.23 Notasi *Message*

8. *Return Message*: Panah dengan garis putus – putus sebagai notasi pesan yang dikembalikan dari objek.



Gambar 2.24 Notasi *Return Message*

9. *Optional Notes*: Menjelaskan secara umum apa yang ada didalam *sequence diagram*.



Gambar 2.25 Notasi *Optional Notes*

2.3 *Framework*

Framework merupakan kumpulan suatu koleksi dari komponen *third-party* dengan beragam kerangka kerja seperti konfigurasi berkas, *service providers*, penentuan struktur direktori, dan *bootstrap* aplikasi. Keunggulan menggunakan *framework* secara umum adalah sudah ditentukannya bagaimana komponen – komponen yang ada bisa saling berhubungan. (Stauffer, 2019, p. 1)

2.4 *Model-View-Controller* (MVC)

Model-View-Controller (MVC) sudah menjadi pola arsitektur yang sangat penting dalam ilmu komputer selama bertahun – tahun. MVC memiliki cara khusus dalam memisahkan masalah dalam suatu aplikasi (sebagai contoh, memisahkan logika *data access* dari logika tampilan) dan menerapkannya dengan sangat baik pada aplikasi *web*. Hal ini membuat pola MVC *powerfull* dan *elegant*. (Galloway, Wilson, Allen, & Matson, 2014, p. 2)

MVC memisahkan *user interface* (UI) dari aplikasi menjadi 3 aspek utama:

1. ***The Model***: Kumpulan kelas yang mendeskripsikan data yang dikerjakan dan aturan bisnis mengenai bagaimana data dapat diubah dan dimanipulasi.
2. ***The View***: Menjelaskan bagaimana UI dari suatu aplikasi akan ditampilkan.
3. ***The Controller***: Kumpulan kelas yang mengelola komunikasi dari pengguna, alur aplikasi secara keseluruhan, dan logika aplikasi secara spesifik.

2.5 *Active Server Pages (ASP .NET) MVC*

ASP .NET adalah suatu *framework* untuk membangun aplikasi berbasis *web* yang menerapkan pola *Model-View-Controller* secara umum ke *framework* ASP .NET yang ada. (Satzinger, Jackson, & Burd, 2012, p. 1)

ASP .NET selalu mendukung dua lapisan abstrak (Galloway, Wilson, Allen, & Matson, 2014, p. 2), yaitu:

1. **System.Web.UI:** Lapisan *Web Form*, terdiri kontrol *server*, *ViewState*, dan lain – lain.
2. **System.Web: The Plumbing,** yang menyediakan *web stack* dasar, mencakup modul, penanganan (*handlers*), *HTTP stack*, dan lain – lain.

Pola MVC sering digunakan dalam pemrograman aplikasi *web*. Dengan ASP .NET MVC dapat diartikan sebagai berikut:

1. **Models:** Kelas yang merepresentasikan domain yang dituju. Objek dari domain tersebut seringkali mengenkapsulasi data yang disimpan dalam *database* seperti *code* yang memanupulasi data dan memberlakukan logika bisnis spesifik domain.
2. **View: Template** untuk menghasilkan HTML secara dinamis.
3. **Controller:** Kelas khusus yang menangani hubungan antara *View* dan *Model*. *Controller* merespons input pengguna, berkomunikasi dengan *Model*, dan memutuskan *View* mana yang akan ditampilkan.

2.6 *Application Programming Interface (API)*

Application Programming Interface atau yang biasa dikenal dengan sebutan API menyediakan sarana untuk kolega, mitra, atau pengembang pihak ketiga untuk mengakses data dan layanan untuk membangun suatu aplikasi. API pada dasarnya merupakan kontrak. Setelah kontrak diterapkan, pengembang tertarik untuk menggunakan API karena mereka tau bahwa mereka dapat mengandalkannya. Kontrak tersebut juga membuat hubungan antara *provider* dan *consumer* jauh lebih efisien karena *interface* didokumentasikan, konsisten, dan dapat diprediksi. Secara teknis, API merupakan suatu cara agar aplikasi berbeda dapat saling berkomunikasi melalui jaringan menggunakan bahasa yang umum dan dimengerti kedua belah pihak.

Contoh API yang terkenal dan sering digunakan yaitu API milik *Twitter* dan *Facebook*. (Jacobson, Brail, & Woods, 2012, p. 4)

Adapun beberapa jenis API di antaranya:

1. API yang terbuka untuk umum kepada semua pengembang.
2. API yang hanya dapat digunakan oleh mitra.
3. API yang digunakan secara internal untuk mendukung proses bisnis yang berjalan dan memfasilitasi kolaborasi antar tim.

2.7 C# (C *sharp*)

C# (dibaca “C *sharp*”) merupakan salah satu bahasa pemrograman yang digunakan untuk membuat program yang dapat dijalankan. C# menggabungkan bahasa pemrograman C++ (dibaca “*see plus plus*”) yang *powerful* tetapi rumit dengan *Visual Basic* yang lebih mudah digunakan namun bertele-tele. C# memiliki ekstensi file .cs. Inkarnasi .NET Visual Basic yang lebih baru hampir setara dengan C# dalam berbagai hal. Sebagai bahasa andalan .NET, C# cenderung memperkenalkan sebagian besar fitur baru. (Mueller, Sempf, & Sphar, 2018)

Berikut beberapa kelebihan C# di antaranya:

1. ***Flexible***: C# dapat mengeksekusi dari mesin yang digunakan saat itu, atau dapat ditransmisikan melalui web dan dieksekusi melalui komputer lain yang jauh.
2. ***Powerful***: C# pada dasarnya memiliki *command* penting yang sama seperti C++ akan tetapi lebih baik.
3. ***Easier to use***: C# memiliki kemampuan mengidentifikasi *error* daripada C++, sehingga waktu yang diperlukan dalam mendeteksi *error* menjadi lebih singkat.
4. ***Visually Oriented***: .NET (dibaca “*dot net*”) *code library* yang digunakan oleh C# telah menyediakan beberapa bantuan untuk membuat *display frame* yang rumit dengan *drop-down list*, *tabbed window*, *grouped button*, *scroll bar*, *background image*, dan lain – lain.
5. ***Internet-friendly***: C# berperan sangat penting pada .NET *Framework*, pendekatan *Microsoft* saat ini untuk pemrograman *Windows*, internet, dan sebagainya.

6. **Secure:** Setiap bahasa yang digunakan di internet harus memiliki keamanan yang serius untuk melindungi dari *malevolent hackers*.

2.8 *Hyper Text Markup Language (HTML)*

Hyper Text Markup Language atau biasa disingkat menjadi HTML adalah sebuah bahasa markah yang dapat ditafsirkan oleh *browser* untuk mendeskripsikan bagaimana teks, grafik, dan *file* yang memuat informasi lainnya terorganisir dan saling berhubungan. Kode HTML memberitahu *browser* bagaimana untuk menampilkan sebuah teks menjadi sebuah *paragraph*, *heading*, huruf berwarna, dan sebagainya. (Meloni, 2012, p. 1)

Pada setiap halaman *website* yang memuat HTML, setiap sintaksis yang dimulai dengan simbol ‘<’ dan berakhir dengan simbol ‘>’ merupakan sebuah *tag* HTML, disebut *tag* dikarenakan simbol-simbol tersebut menandai potongan-potongan kata dan menjelaskan kepada *browser* jenis kata apa yang ditandai tersebut sehingga *browser* mampu untuk merepresentasikannya dengan benar. (Meloni, 2012, p. 28)

2.9 *Cascading Style Sheets (CSS)*

Cascading Style Sheet atau dikenal juga sebagai CSS merupakan kumpulan instruksi yang mengontrol tampilan dari beberapa halaman HTML secara langsung, dimana tampilan yang dimaksud berupa jenis, ukuran, warna dan karakteristik – karakteristik lainnya yang dapat membedakan *website* yang satu dengan lainnya. Konsep CSS dapat dijabarkan sebagai berikut: membuat dokumen *style sheet* yang menspesifikasikan karakteristik dari suatu *website*, lalu menghubungkan setiap halaman HTML yang seharusnya memiliki tampilan yang sesuai dengan karakteristik yang sudah dibuat didalam *file style sheet* tersebut, sehingga tidak perlu untuk membuat ulang *style sheet* untuk masing-masing halaman HTML. Kemudian, apabila perlu untuk mengubah warna tema atau karakteristik global lainnya dari sebuah *website*, cukup untuk mengubah beberapa *style sheet* dan tidak perlu untuk merubah semua *style* disetiap *file*. (Meloni, 2012, p. 45)

2.10 *Javascript*

Javascript merupakan suatu bahasa pemrograman yang dikembangkan *Netscape Communications Corporations*, dimana *Javascript* merupakan *web scripting language* pertama yang dapat didukung oleh *browser* dan merupakan salah satu bahasa yang paling populer. (Meloni, 2012, pp. 66-67)

Beberapa hal yang dapat dilakukan menggunakan *Javascript*:

1. Menampilkan pesan kepada pengguna *browser*, di baris status *browser*, ataupun di kotak *alert*.
2. Memvalidasi konten dari sebuah *form* dan melakukan kalkulasi.
3. Menambahkan animasi pada gambar atau membuat gambar yang mampu berubah ketika kursor *mouse* diarahkan ke gambar tersebut.
4. Membuat suatu spanduk iklan yang dapat berinteraksi dengan pengguna.
5. Mengetahui jenis *browser* yang digunakan dan melakukan fungsi lanjutan hanya pada *browser* yang mendukungnya.
6. Mendeteksi *plug-in* yang terpasang dan memberitahu pengguna apabila diperlukan pemasangan *plug-in*.
7. Memodifikasi seluruh halaman *website* tanpa mewajibkan pengguna untuk memuat ulang halaman tersebut.
8. Menampilkan atau berinteraksi dengan data yang diperoleh dari *remote server*.

2.11 *JavaScript Object Notation (JSON)*

JavaScript Object Notation atau yang biasa disebut JSON adalah suatu format yang hampir sama dengan XML, untuk menyimpan dan bertukar data. JSON dapat dilihat melalui *editor* atau *browser*. Secara umum, JSON digunakan untuk mengoper data antar tingkatan (*tiers*). (Prettyman, 2016, p. 63)

2.12 *Asynchronous Javascript and XML (AJAX)*

Asynchronous JavaScript and XML atau yang lebih dikenal dengan AJAX merupakan teknik yang digunakan untuk membangun aplikasi berbasis *web* yang responsif dengan *user experience* yang baik. Menjadi responsif membutuhkan komunikasi *asynchronous*, akan tetapi tampilan yang responsif dapat juga berasal dari

animasi yang halus dan pergantian warna. (Galloway, Wilson, Allen, & Matson, 2014, pp. 213-214)

2.13 *JQuery*

JQuery merupakan salah satu *library* populer yang didesain khusus untuk *JavaScript* dan menjadi proyek *open source*. *JQuery* unggul dalam menemukan (*finding*), melintasi (*transversing*), memanipulasi (*manipulating*) HTML dalam dokumen HTML. Selain itu, *JQuery* juga mempermudah penggunaan *event handler* pada elemen, menghidupkan elemen, dan membangun interaksi *Ajax* disekitar elemen. (Galloway, Wilson, Allen, & Matson, 2014, p. 214)

2.14 *Database*

Database adalah kumpulan data yang saling terhubung secara logis, dirancang untuk memenuhi kebutuhan informasi dari suatu organisasi. *Database* merupakan suatu tempat penyimpanan data yang dapat digunakan secara bersamaan oleh banyak pengguna dari setiap departemen. *Database* bukan hanya berisi data operasional suatu organisasi, melainkan terdapat deskripsi dari data tersebut. Oleh karena itu, *database* didefinisikan juga sebagai *self-describing collection of integrated records*. Deskripsi dari data disebut sebagai *system catalog* atau *data dictionary* atau *metadata* (data mengenai data). (Connolly & Begg, 2015, p. 63)

2.15 *Database Management System (DBMS)*

Menurut Connolly & Begg (2015, p. 64), *Database Management System* (DBMS) adalah suatu sistem perangkat lunak yang memungkinkan *user* untuk *define* (menetapkan), *create* (membuat), *maintain* (mengelola), dan *control* (mengontrol) akses ke *database*. DBMS merupakan suatu *software* yang berinteraksi dengan program aplikasi pengguna dan *database*. Secara khusus, DBMS menyediakan fasilitas sebagai berikut:

1. Memungkinkan pengguna untuk mendefinisikan *database*, biasanya melalui *Data Definition Language* (DDL). DDL memungkinkan pengguna untuk menetapkan tipe data, struktur data, dan *constraints* (ketentuan) pada data untuk disimpan ke *database*.

2. Memungkinkan pengguna untuk *insert*, *create*, *update*, *delete*, dan *retrieve* (mengembalikan) data dari *database*, biasanya melalui *Data Manipulation Language* (DML).
3. Menyediakan akses terkontrol ke *database*, seperti sistem keamanan, sistem integritas, sistem kontrol konkurensi, sistem kontrol pemulihan, dan katalog yang dapat diakses oleh pengguna.

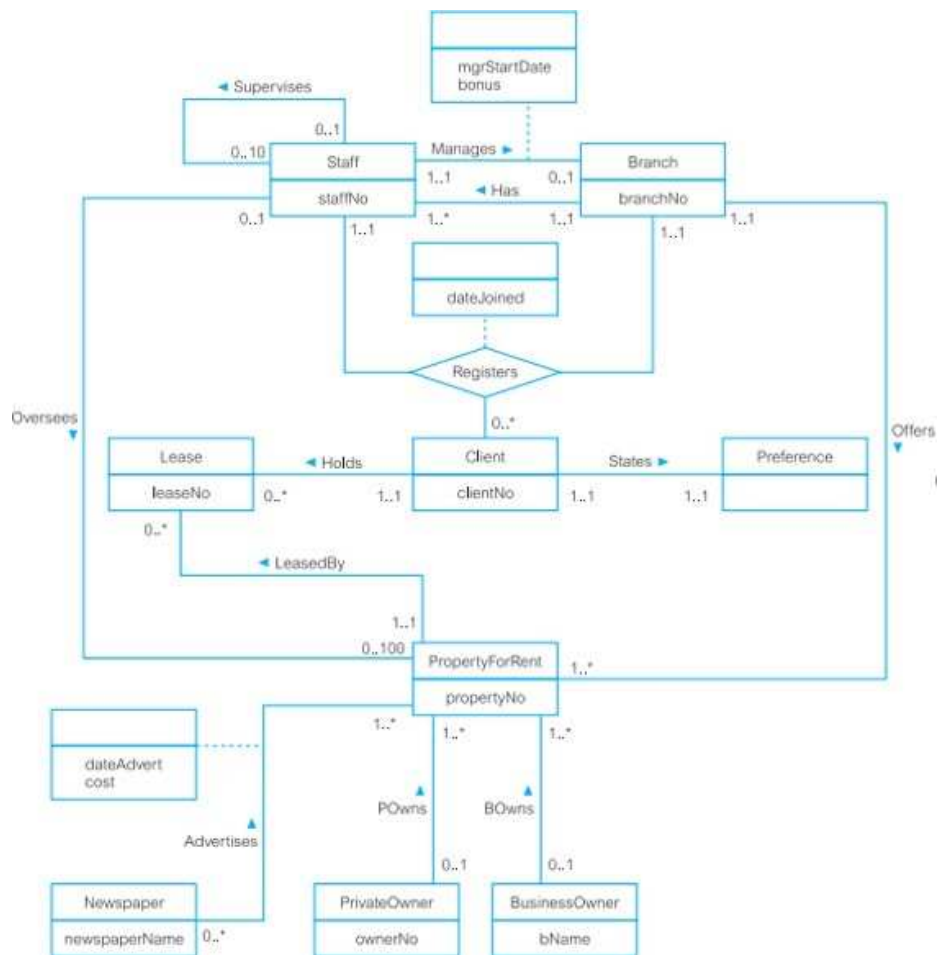
2.16 Entity Relationship Diagram (ERD)

ERD adalah pendekatan desain *database* yang dimulai dengan identifikasi data penting yang disebut *entities* dan *relationship* antar data yang harus direpresentasikan dalam model. Proses selanjutnya adalah menambahkan detail, seperti informasi mengenai *attributes* dan *constraints* (ketentuan) pada *entities*, *relationship*, dan *attributes*.

Arti dari *entity* itu sendiri adalah sekumpulan objek dengan *properties* yang sama (Connolly & Begg, 2015, p. 406). Pada sebuah *entity* terdapat banyak *attribute* yaitu *property* yang dimiliki oleh *entity*. Sebagai contoh *entity Staff*, memiliki *attribute* seperti *staffNo*, *name*, *position*, dan *salary attribute*. (Connolly & Begg, 2015, p. 413).

Beberapa jenis *attribute* dalam suatu *entity* menurut Connolly & Begg (2015) di antaranya:

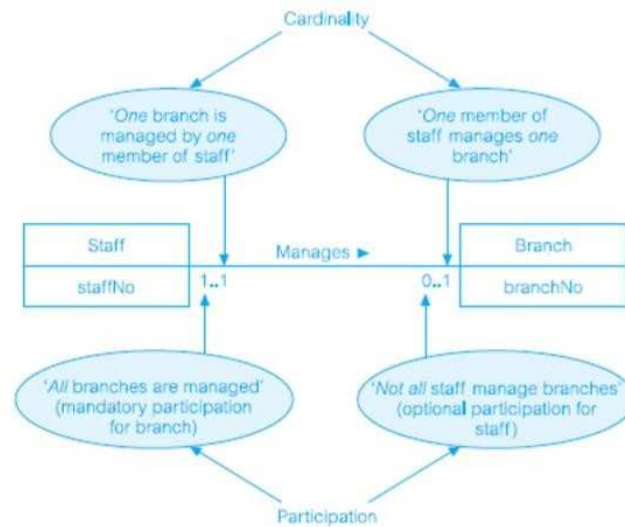
1. **Composite Attribute:** Atribut yang terdiri dari banyak komponen, masing – masing dapat berdiri sendiri jika dipisahkan. Sebagai contoh *address* (163 Main St, Glasgow, G11 9QX), atribut *address* tersebut dapat dipisah menjadi *street* (163 Main St), *city* (Glasgow), dan *postcode* (G11 9QX).
2. **Single-valued Attribute:** Atribut yang memiliki satu *value*, contohnya *entity branch* memiliki atribut *branchNo* dengan *value* B003.
3. **Multi-valued Attribute:** Atribut yang memiliki banyak *value*, contohnya nomor telepon bisa terdiri lebih dari satu.
4. **Derived Attribute:** Atribut yang merepresentasikan nilai turunan dari atribut lainnya. Sebagai contoh *entity Rent* memiliki atribut *duration* yang diturunkan dari *startDate* dan *endDate*.



Gambar 2.26 Entity Relationship Diagram

(Connolly & Begg, 2015, p. 407)

Relationship di dalam ERD merupakan bentuk hubungan antar *entity* yang terdapat *multiplicity* sebagai tipe *constraint* dari suatu *relationship*. *Multiplicity* menggambarkan jumlah kemungkinan kejadian dari suatu *entity* ke *entity* lainnya yang terdiri dari dua *constraint* terpisah, yaitu *cardinality* dan *participation*. *Cardinality* mendeskripsikan angka maksimum dari hubungan yang memungkinkan untuk muncul pada *entity* yang ada dalam tipe *relationship* yang diberikan. *Participation* menentukan semua atau hanya beberapa *entity* yang berpartisipasi didalam hubungan. (Connolly & Begg, 2015, pp. 419-424)



Gambar 2.27 Contoh Cardinality dan Participation

(Connolly & Begg, 2015, p. 425)

Terdapat 3 jenis *multiplicity* dalam *relationship* yaitu:

1. *One on One Relationship* (1:1)
2. *One to Many Relationship* (1:*)
3. *Many to Many Relationship* (*:*)

Selain itu, di dalam *relationship* terdapat *keys* yang merupakan suatu cara untuk mengidentifikasi jenis atribut yang ada dalam sebuah *entity* yang disebut *relational keys*. Berikut ini *relational keys* menurut Connolly & Begg (2015, pp. 158-159) diantaranya:

1. *Super Key*: Sebuah atribut atau sekumpulan atribut yang secara unik dapat membedakan tiap baris dalam suatu relasi.
2. *Candidate Key*: *Super Key* dalam suatu relasi.
3. *Primary Key*: *Candidate Key* yang dipilih untuk mengidentifikasi baris secara unik dalam suatu relasi.
4. *Foreign Key*: Suatu atribut atau sekumpulan atribut dalam relasi yang menghubungkan *candidate key* dari relasi lain atau relasi yang sama.

2.17 *Structured Query Language (SQL)*

Structured Query Language atau yang lebih sering disebut dengan SQL merupakan suatu bahasa yang tercipta dari pengembangan *relation model*. SQL adalah contoh dari *transform-oriented language*, atau bahasa yang didesain untuk mengubah *input* menjadi *output* yang dibutuhkan. (Connolly & Begg, 2015, pp. 191-192)

Sebagai suatu bahasa, SQL memiliki dua komponen utama di antaranya:

1. *Data Definition Language* (DDL) untuk mendefinisikan struktur *database* dan mengontrol akses ke data.
2. *Data Manipulation Language* (DML) untuk menghasilkan dan memperbarui data.

2.18 *Stored Procedure (SP)*

Stored Procedure atau yang lebih sering disebut dengan SP merupakan *subprogram* yang dapat mengambil parameter dan dipanggil. SP dapat mengambil kumpulan parameter yang diberikan dengan memanggil SP yang terdapat pada program dan menjalankan serangkaian *action*. SP dapat memodifikasi dan mengembalikan data sebagai parameter. Pada SP nilai yang dikembalikan bisa lebih dari satu. Di samping itu, SP juga memiliki kelebihan, seperti menyediakan modularitas (*modularity*) dan perpanjangan (*extensibility*), mendukung penggunaan berulang (*reusability*) dan pemeliharaan (*maintainability*), dan *aid abstraction*. (Connolly & Begg, 2015, p. 280)

2.19 *Eight Golden Rules of Interface Design*

Menurut Shneiderman, Plaisant, & Cohen (2016, p. 95) terdapat delapan aturan utama yang digunakan sebagai dasar dalam merancang *User Interface* (UI). Aturan ini dikenal sebagai *Eight Golden Rules of Interface Design*, diantaranya:

1. *Strive for Consistency*

Action harus berurutan dan diterapkan dalam kondisi yang serupa. Terminologi yang identik ini digunakan untuk konfirmasi, menu, *help screen*, warna, susunan, kapitalisasi, jenis huruf, dan lain sebagainya yang seharusnya digunakan secara menyeluruh.

2. *Cater to Universal Usability*

Memahami kebutuhan *user* yang beragam untuk dan mendesain untuk *plasticity* (memfasilitasi transformasi dari sebuah konten). Perbedaan karakteristik *user* seperti tingkat kemahiran, rentang usia, kebutuhan khusus, keberagaman universal, dan perbedaan teknologi yang memperkaya spektrum kebutuhan sebagai panduan dalam desain.

3. *Offer Informative Feedback*

Setiap *action* dari *user*, sebaiknya ada umpan balik dari sistemnya. *Action* yang minor yang sering terjadi, responnya dapat dibuat sederhana, sedangkan untuk *action* yang major dan tidak sering terjadi, responnya dapat dibuat lebih substansial.

4. *Design Dialog to Yield Closure*

Urutan *action* dikelompokkan menjadi awal, pertengahan, hingga akhir. Umpan balik yang informatif ketika menyelesaikan suatu *action* dapat memberikan kepuasan *user*, perasaan lega, dan menjadi sebuah indikator untuk bersiap melakukan *action* selanjutnya.

5. *Prevent Errors*

Sebisa mungkin untuk mendesain sistem yang dapat meminimalkan tingkat kesalahan serius dari *user*. Jika *user* membuat suatu kesalahan dalam menjalankan *action*, antarmuka harus mendeteksi kesalahan dan menawarkan instruksi yang sederhana, konstruktif, dan spesifik untuk pemulihan ke kondisi awal.

6. *Permit Easy Reversal of Actions*

Action yang diberikan sebisa mungkin dibuat reversibel. Fitur ini meringankan kecemasan karena *user* mengetahui bahwa kesalahan yang terjadi dapat diperbaiki dan mendorong *user* untuk melakukan eksplorasi terhadap pilihan lain.

7. *Support Internal Locus of Control*

User yang berpengalaman menggunakan sistem, menginginkan perasaan bahwa mereka dapat mengontrol antarmuka dan antarmuka merespons *action* yang diberikan. *User* tidak menginginkan perubahan yang terjadi secara tiba – tiba pada *behavior* yang biasa mereka lakukan dan dapat merasa terganggu dengan pengisian data yang membosankan, kesulitan

mendapatkan informasi yang dibutuhkan, dan ketidakmampuan untuk memperoleh hasil yang diinginkan.

8. *Reduce Short-term Memory Load*

Keterbatasan kemampuan manusia untuk memproses informasi dalam ingatan jangka pendek membutuhkan antarmuka terhindar dari desain yang membuat *user* harus mengingat informasi dari satu tampilan dan menggunakan informasi tersebut pada tampilan lain.

2.20 *Five Measurable Human Factor*

Ada beberapa kriteria yang harus diperhatikan dalam merancang suatu sistem agar *user friendly*. Menurut Shneiderman, Plaisant, & Cohen, (2016, pp. 33-34), dalam membuat antarmuka dibutuhkan evaluasi terhadap lima faktor manusia terukur untuk mengetahui tingkat efektivitas, efisiensi, serta kepuasan *user* terhadap sistem. Lima faktor tersebut adalah sebagai berikut:

1. *Time to learn*

Berapa lama waktu yang dibutuhkan untuk mempelajari cara menggunakan fitur – fitur dalam sistem?

2. *Speed of performance*

Berapa lama waktu yang dibutuhkan untuk menjalankan tugas tertentu?

3. *Rate of error by users*

Berapa banyak kesalahan dan jenis kesalahan apa yang dibuat oleh user?

4. *Retention over time*

Berapa lama kemampuan *user* untuk mempertahankan ingatan dan pengetahuannya dalam penggunaan sistem?

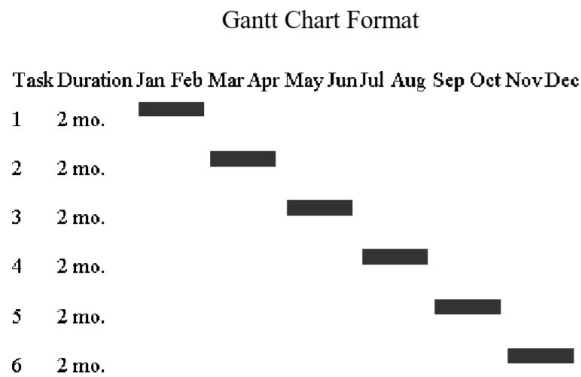
5. *Subjective satisfaction*

Bagaimana kepuasan *user* terhadap berbagai aspek sistem?

2.21 *Gantt Chart*

Selama era manajemen keilmuan, Henry Gantt mengembangkan suatu alat untuk menampilkan perkembangan proyek dalam bentuk diagram. Pada awalnya dikembangkan alat ini adalah untuk melakukan *tracking* dari proyek pengembangan kapal laut.

Scheduling tools Gantt ini berbentuk grafik batang horizontal dan saat ini dikenal sebagai Gantt *chart* sesuai dengan nama penemunya. Garis horizontal dari Gantt *chart* adalah skala waktu yang diekspresikan dalam bentuk waktu baik absolut atau relatif yang merujuk ke awal proyek. Satuan waktu yang biasa digunakan yaitu secara mingguan atau bulanan. Baris dalam batang dalam *chart* menampilkan tugas – tugas individual yang dikerjakan dalam proyek. (Pathak, 2015, p. 233)



Gambar 2.28 Format Gantt Chart

(Pathak, 2015)

Bar yang ada memungkinkan untuk tumpang tindih jika tugas dapat dimulai sebelum menyelesaikan tugas lain dan terdapat tugas yang dilakukan secara paralel. Gantt *chart* efektif digunakan untuk mengkomunikasikan pengaturan waktu di berbagai tugas. Dalam proyek yang lebih besar, tugas dapat dipecah menjadi beberapa subtask yang memiliki Gantt *chart* tersendiri untuk menjaga visibilitasnya. (Pathak, 2015, p. 233)

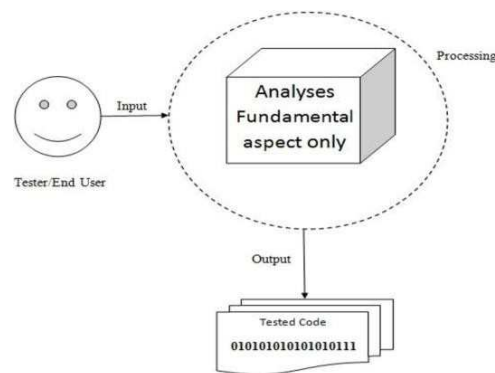
2.22 Testing

Testing (pengujian) menurut standar yang ditetapkan oleh ANSI/IEEE-1059 adalah prosedur analisa modul dalam *software* untuk mengetahui perbedaan antara kondisi saat ini (seperti *defects/errors/bugs*) dan sebagai penilaian atas fitur dalam *software*. Tujuan utama dari *testing* adalah verifikasi, konfirmasi (validasi) dan penemuan kesalahan untuk dapat segera diperbaiki. *Software testing* adalah salah satu teknik utama untuk memperoleh *software* dengan kualitas yang baik.

Software testing merupakan suatu metode untuk verifikasi dan validasi apakah aplikasi yang dikembangkan sesuai dengan *requirement* baik secara bisnis maupun *scientific* dan berfungsi sesuai dengan perancangan awal dan pengembangan. (Jat & Sharma, 2017, p. 77)

2.2.6 Black-Box Testing

Black-Box Testing adalah pengujian tanpa informasi mengenai mekanisme internal yang terjadi dibalik *software* yang sedang diuji dan hanya memeriksa aspek penting dari sistem. Ketika melakukan *black box testing*, penguji harus mengetahui desain sistem dan tidak akan memiliki akses ke *source code*. Testing jenis ini seringkali dikenal sebagai *functional testing* atau *input-output driven testing*. (Jat & Sharma, 2017, p. 78)



Gambar 2.29 Ilustrasi Blackbox Testing

(Jat & Sharma, 2017, p. 78)

2.23 Gamification

Gartner mendefinisikan *gamification* sebagai penggunaan mekanisme *game* dan desain pengalaman untuk melibatkan dan memotivasi orang secara digital untuk mencapai tujuan mereka. Mekanisme *game* di sini mendeskripsikan *key element* yang umum pada banyak *game*, seperti poin, lencana, dan papan peringkat. Tujuan dari *gamification* adalah untuk memotivasi orang untuk mengubah perilaku, mengembangkan keterampilan, atau mendorong motivasi. *Gamification* berfokus pada memungkinkan pemain mencapai tujuan mereka dan manfaat lainnya suatu organisasi dapat mencapai tujuannya. (Burke, 2014)

2.24 *Complaint Handling*

Menurut Slater & Higginson (2016, pp. 64-65), penanganan keluhan (*complaint handling*) dari perspektif konsumen yang disukai, melibatkan pengelolaan dengan organisasi yang proaktif dan berfokus pada konsumen, memperlakukan mereka dengan rasa hormat dan empati serta mempertimbangkan pendapat mereka. Dengan demikian, selama berjalannya penelitian secara kualitatif dihasilkan proses ideal untuk menangani pengaduan.

Proses itu sendiri mencakup:

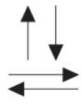
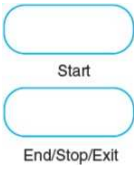





1. Informasi jelas mengenai proses pengaduan yang diberikan kepada konsumen sejak awal.
2. Berbagai metode untuk menyampaikan keluhan dan berkomunikasi dengan organisasi.
3. Media yang dapat digunakan untuk mengajukan keluhan.
4. Saluran telepon khusus yang memastikan bahwa orang tidak ditahan.
5. Titik pertemuan yang memiliki nama.
6. Timeline yang memberikan estimasi berapa lama proses akan berlangsung dan kapan konsumen dapat mengharapkan hasil akhirnya.
7. *Staff* yang diberikan wewenang untuk menawarkan solusi dan tidak perlu bergantung pada orang lain.
8. Solusi yang diberikan secara langsung.



Selama proses menangani pengaduan, konsumen memerlukan informasi yang lengkap dan jelas tentang langkah selanjutnya yang diinginkan serta harus terus mendapat informasi. Frekuensi pembaruan akan sesuai dengan *timeline* yang disediakan diawal. (Slater & Higginson, 2016, p. 61)

2.25 *Flowchart*

Flowchart merupakan representasi *visual* dari algoritma dan *pseudocode*. Berikut ini terlampir simbol-simbol pada *flowchart* (Sprankle & Hubbard, 2011)

Tabel 2.2 Simbol Simbol pada Flowchart

<i>Flowchart Symbol</i>	<i>Explanation</i>
 Flowlines	<p>Flowlines are Indicated by straight lines with optional arrows to show the direction of data flow. The arrowhead Is necessary when the flow direction might be in doubt. Flowlines are used to connect blocks by exiting from one and entering another.</p>
 Start End/Stop/Exit	<p>Flattened ellipses indicate the start and the end of a module. An ellipse uses the name of the module at the start. The end Is indicated by the word end or stop for the top or Control module and the word exit for all other modules. A start has no flowlines entering it and only one exiting It; an end or exit has one flowline entering It but none exiting it.</p>
 Processing	<p>The rectangle indicates a processing block, for such things as calculations, opening and closing files, and so forth. A processing block has one entrance and one exit.</p>
 I/O	<p>The parallelogram indicates input to and output from the computer memory. An input/output (I/O) block has one entrance and only one exit.</p>
 Decision	<p>The diamond indicates a decision. It has one entrance and two and only two exits from the block. One exit is the action when the resultant is True and the other exit is the action when the resultant is False.</p>
 Process Module	<p>Rectangles with lines down each side indicate the process of modules. They have one entrance and only one exit.</p>
 Automatic-Counter Loop	<p>The polygon indicates a loop with a counter. The counter starts with A (the beginning value) and is incremented by S (the incrementor value) until the counter is greater than B (the ending value). Counter is a variable. A, B, and S may be constants, variables, or expressions.</p>

<i>Flowchart Symbol</i>	<i>Explanation</i>
 On-Page Connectors*  Off-Page Connectors*	<p><i>Flowchart sections can be connected with two different symbols. The circle connects sections on the same page, and the home base plate connects flowcharts from page to page. Inside these two symbols the programmer writes letters or numbers. The on-page connector uses letters inside the circle to indicate where the adjoining connector is located. An A connects to an A, a B to a B, etc. The off-page connectors use the page number where the next part or the previous part of the flowchart is located. This allows the reader to easily follow the flow-chart. On- and off-page connectors will have either an entrance or an exit.</i></p>