BAB 2

LANDASAN TEORI

2.1 Teori Umum

2.1.1 Data, Informasi, dan *Database*

Mengacu pada pendapat Williams dan Sawyer (2007) data adalah figur-figur dan fakta-fakta dasar yang diproses menjadi suatu informasi. Sedangkan informasi adalah kumpulan data yang telah dimanipulasi atau dirangkum untuk digunakan dalam pengambilan keputusan.

Connolly dan Begg (2010) mengatakan bahwa *database* adalah suatu kumpulan data logikal yang saling berhubungan dan sebuah deskripsi data, dirancang untuk memenuhi kebutuhan informasi yang dibutuhkan oleh suatu organisasi. *Database* merupakan tempat penyimpanan sekumpulan *file* data yang telah terkomputerisasi. *Database* bersifat tunggal, yaitu suatu tempat penyimpanan besar yang dapat digunakan secara bersamaan oleh beberapa departemen dan *user*. *Database* digunakan untuk mencegah terjadinya pengulangan data, seluruh data terintegrasi dengan jumlah duplikasi data yang sedikit.

2.1.2 Sistem Informasi

Mengacu pada pendapat Whitten dan Bentley (2007) sistem informasi adalah sebuah pengaturan dari manusia, data, proses-proses, dan teknologi

informasi yang berinteraksi di dalamnya dengan tujuan untuk mengumpulkan, memproses, menyimpan, dan penyediaan sebagai *output* bagi sebuah perusahaan yang membutuhkan informasi tersebut.

2.1.3 *Database Management System* (DBMS)

Mengacu pada pendapat Connolly dan Begg (2010) DBMS adalah *software* yang memungkinkan *user* untuk mendefinisikan, menciptakan, memelihara, dan mengontrol akses ke *database.* DBMS berinteraksi dengan program aplikasi *user* dan *database.* Pada umumnya DBMS menyediakan beberapa fasilitas sebagai berikut:

a. *Data Definition Language* (DDL)

Suatu bahasa yang memungkinkan *Database Administrator* (DBA) atau *user* untuk menjelaskan dan menspesifikasikan entitas, atribut, dan hubungan yang dibutuhkan untuk aplikasi, serta beberapa integritas terhubung dan batasan.

b. *Data Manipulation Language* (DML)

Suatu bahasa yang digunakan oleh *user* untuk melakukan modifikasi dan pengambilan data pada suatu *database.* Modifikasi dapat berupa penambahan data ke dalam suatu *database* (*insert*), modifikasi data dalam *database* (*update*), mengambil data dari *database* (*retrieve*), dan menghapus data (*delete*) dari *database.*

c. Akses Kontrol *Database*

Fasilitas pengendalian akses kontrol ke dalam *database* berfungsi sebagai:

1. Sistem keamanan, mencegah penggunaan yang tidak memiliki hak akses ke dalam *database*.
2. Sistem integritas, mempertahankan konsistensi data yang disimpan.
3. Pengendalian *share* data, memberikan izin hak akses dalam *database*.
4. Sistem pengendalian *recovery*, mengembalikan data ke dalam kondisi konsisten sebelumnya pada *database* sebelum terjadinya kegagalan pada *hardware* atau *software*.
5. Katalog akses *user,* berisi deskripsi data dalam *database.*

Dengan adanya beberapa fungsi tersebut, DBMS menjadi perangkat yang sangat berguna dan bermanfaat. DBMS memberikan tampilan yang sulit bagi *user* karena *user* mendapatkan data yang lebih banyak dari yang dibutuhkan. Berdasarkan masalah yang timbul ini maka DBMS menyediakan fasilitas *view mechanism*, sehingga setiap *user* dapat memiliki tampilan *database* sendiri.

2.1.3.1 Komponen DBMS

Mengacu pada pendapat Connolly dan Begg (2010) DBMS *Environment* memiliki berbagai komponen, yaitu perangkat keras, perangkat lunak, data, prosedur, dan manusia.

Gambar 2.1 Komponen DBMS *Environment*

(sumber: Connolly & Begg, 2010:68)

a. Perangkat Keras (*Hardware*)

DBMS dan aplikasi membutuhkan perangkat keras untuk beroperasi. Perangkat keras meliputi *Personal Computer* (PC), *mainframe*, hingga jaringan komputer. DBMS hanya beroperasi pada beberapa *hardware* atau sistem operasi. *Back-end* adalah bagian dari DBMS yang mengatur dan mengontrol akses ke *database*. *Front-end* adalah bagian dari DBMS yang berinteraksi dengan *user*.

b. Perangkat Lunak (*Software*)

Software DBMS, program aplikasi, dan sistem operasi merupakan komponen perangkat lunak termasuk *software* jaringan jika DBMS sedang digunakan bersamaan dengan jaringan. Pada umumnya, aplikasi program ditulis dengan 3GL (*third generation*

programming language) seperti C, C++, Java, atau 4GL (*fourth generation language*) seperti SQL yang digabungkan pada 3GL.

c. Data

Komponen terpenting di dalam DBMS adalah data. *Database* mengandung data operasional dan metadata. *Schema* merupakan struktur dari d*atabase*. *Schema* terdiri dari beberapa tabel dan atribut.

d. Prosedur (*Procedure*)

Prosedur berkaitan dengan instruksi dan aturan yang mengatur desain dan penggunaan *database*. *User* sistem yang mengatur *database* membutuhkan prosedur dokumen untuk menjalankan sistem.

e. Manusia (*People)*

Selain data, manusia juga merupakan komponen yang memiliki peranan penting pada DBMS agar DBMS tersebut dapat beroperasi dengan baik dan optimal.

2.1.4 *Data Flow Diagram* (DFD)

Mengacu pada pendapat Whitten dan Bentley (2007) *data flow diagram* adalah sebuah proses model yang digunakan untuk menjelaskan

aliran atau proses data yang melewati sebuah sistem dan tugas apa saja yang dilakukan oleh sistem tersebut.

Data flow diagram memiliki notasi-notasi sebagai berikut:

a. *External Agent*

Gambar 2.2 Notasi *External Agent*

(Sumber: Whitten & Bentley, 2007:319)

*External agen*t dilambangkan dengan notasi berbentuk “persegi empat” yang merepresentasikan seseorang, unit dari sebuah organisasi ataupun sistem lain yang berinteraksi dengan sebuah sistem. *External agent* biasanya mengirim ataupun menerima *input* dari sistem.

b. *Data Store*

Gambar 2.3 Notasi *Data Store*

(Sumber : Whitten & Bentley, 2007:320)

Data store dilambangkan dengan notasi “persegi panjang” yang terbuka di satu sisinya. *Data store* berfungsi sebagai tempat penyimpanan data. *Data store* biasanya memberikan data kepada *data store* ataupun *external agent* lainnya melalui sebuah proses.

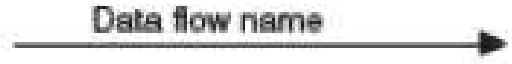
c. Proses

Gambar 2.4 Notasi Proses

(Sumber : Whitten & Bentley, 2007:321)

Proses dilambangkan dengan “persegi empat” dengan sudut melengkung di setiap sudutnya. Proses ini menjelaskan proses yang sedang terjadi pada sebuah *external agent t* ataupun pada sebuah *data store*.

d. *Data Flow*



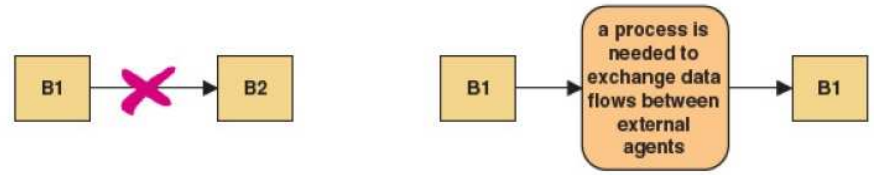
Gambar 2.5 Notasi *Data Flow*

(Sumber : Whitten & Bentley, 2007:325)

Data Flow dilambangkan dengan notasi berbetuk “garis panah hitam” yang melambangkan aliran data yang sedang berlangsung. Aliran data merupakan aliran yang berasal dari inputan *external agent* ataupun *data store* yang masuk ke dalam sistem/proses dan membawa hasil *output* proses ke *external agent* ke sistem, *external agent* ataupun data store lainnya.

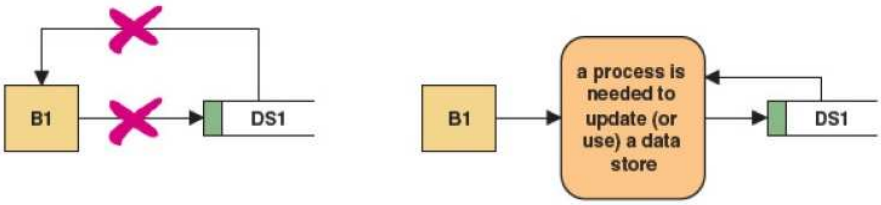
Data flow diagram memiliki 4 aturan *data flow* yang harus dipatuhi yaitu :

a. *Data flow* dari *external agent* ke *external agent* lainnya harus melalui proses.



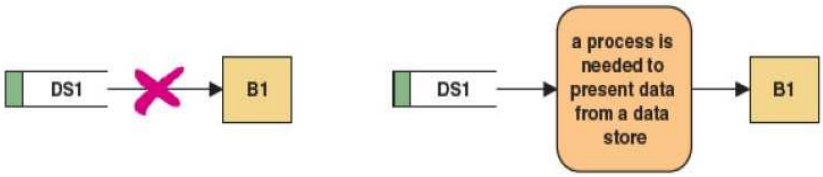
Gambar 2.6 Aturan *Data Flow* I

b. *Data flow* dari eksternal *agent* ke sebuah *data store* dan kembali ke eksternal *agent* harus melalui proses.



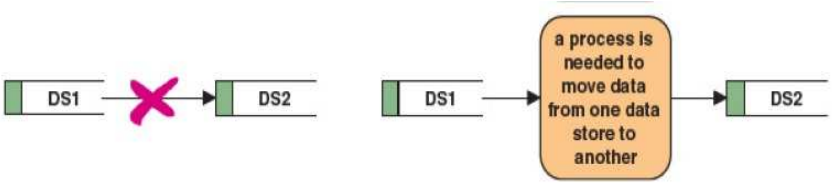
Gambar 2.7 Aturan *Data Flow* II

c. *Data flow* dari *data store* ke eksternal *agent* harus melalui proses.



Gambar 2.8 Aturan *Data Flow* III

d. *Data flow* dari *data store* ke *data store* lainnya harus melalui proses.



Gambar 2.9 Aturan *Data Flow* IV

(Sumber : Whitten & Bentley, 2007:330)

2.1.5 *Unified Modeling Language* (UML)

Mengacu pada pendapat Whitten dan Bentley (2007) *Unified Modeling Language* (UML) adalah suatu *set* dari konvensional model yang digunakan untuk menspesifikasikan atau menjelaskan *software system* dalam bentuk objek – objek.

2.1.5.1 *Use Case Diagram*

Mengacu pada pendapat Whitten dan Bentley (2007) *Use Case Diagram* merupakan diagram yang merepresentasikan interaksi antara sistem, sistem dengan eksternal sistem, dan sistem dengan *user*. Secara grafis, *use case diagram* menjelaskan siapa yang menggunakan sistem dan bagaimana *user* berinteraksi dengan sistem.

Tujuan *Use Case Diagram* adalah untuk menentukan permintaan sebuah sistem, menyampaikan komunikasi rancangan terhadap *client*, dan merancang *test case* untuk semua fitur yang ada pada sistem.

Diagram *Use Case* memiliki notasi-notasi sebagai berikut:

a. *Actor*

Gambar 2.10 Notasi *Actor* pada *Use Case*

(Sumber: Whitten & Bentley, 2007:247)

Disimbolkan dengan bentuk orang yang digunakan untuk menginisiasi sebuah sistem dengan tujuan mendapatkan informasi dari proses interaksi yang dilakukan dengan sistem. *Actor* memiliki 4 tipe yaitu: *primary business actor, primary system actor, external server actor,* dan *external receiver actor*.

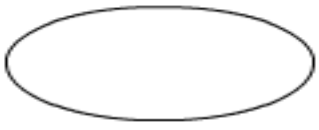
b. *Relationship*

Gambar 2.11 Notasi *Associations* pada *Use Case*

(Sumber: Whitten & Bentley, 2007:248)

Disimbolkan dengan bentuk garis yang menjelaskan hubungan antara *actor* dengan sistem pada diagram *use case*. Hubungan-hubungan tersebut meliputi *associations, extends, uses* (*include*), *depends on*, dan *inheritance*.

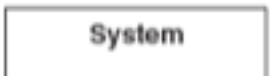
c. *Use Case*



Gambar 2.12 Notasi *Use Case Symbol*

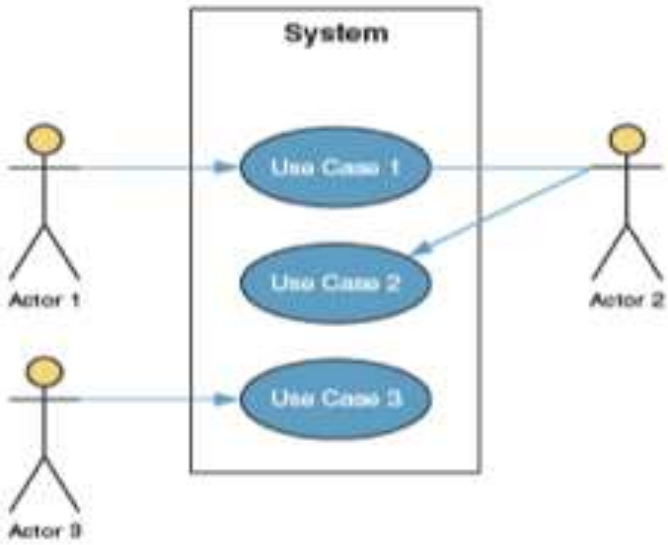
Disimbolkan dengan bentuk elips yang menjelaskan sistem-sistem yang ada pada diagram *use case*. Sistem ini nantinya berhubungan dengan *actor* yang ada pada diagram *use case* tersebut.

d. *Use Case System*



Gambar 2.13 Notasi Nama Sistem pada *Use Case*

Disimbolkan dengan bentuk persegi panjang yang di mendefinisikan nama dari *use case* tersebut dan sistem-sistem yang bekerja pada diagram *use case* tersebut.



Gambar 2.14 *Use Case Model Diagram*

(Sumber: Whitten & Bentley, 2007:246)

2.1.5.2 *Use Case Narrative*

Mengacu pada pendapat Whitten dan Bentley (2007) *Use Case Narrative* adalah sebuah deskripsi secara tekstual dalam bentuk teks yang mendeskripsikan proses dari sebuah kejadian dan bagaimana *user* berinteraksi dengan sistem untuk menyelesaikan suatu tugas.

Berikut di bawah ini adalah deskripsi bagian-bagian dari *use case narrative*:

a. *Author*

Nama individu yang berkontribusi dalam penulisan *use case narrative*.

b. *Date*

Tanggal dari *use case narrative* yang dibuat.

c. *Version*

Versi dari *use case narrative* yang digunakan.

d. *Use-case name*

Use case name merepresentasikan tujuan yang akan dicapai. Kata awal dari *use case name* diawali dengan kata kerja.

e. *Use-case type*

Bagian *use case* yang menyediakan pengertian secara umum terhadap *use case* yang dibuat.

f. *Use-case ID*

Identifier yang secara unik mengidentifikasi *use case*.

g. *Priority*

Prioritas dari komunikasi yang ada pada *use case* apakah *low, medium*, atau *high*.

h. *Source*

Sumber asal yang menjelaskan entitas pada *use case*.

i. *Primary business actor*

Merupakan *stakeholder* utama yang berinteraksi dengan sistem.

j. *Other participating actors*

Actor lain yang berinteraksi dengan sistem disamping *actor* utama.

k. *Interested stakeholder*(s)

Stakeholder lain yang ikut berpartisipasi dalam pengembangan *software system*.

l. *Description*

Menjelaskan secara garis besar isi dari *use case* yang dibuat.

Member Services System

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

Use-Case Name:	Place New Order	**Use-Case Type** **Business Requirements:** ☒
Use-Case ID:	MSS-BUC002.00	
Priority:	High	
Source:	Requirement — MSS-R1.00	
Primary Business Actor:	Club member	
Other Participating Actors:	• Warehouse (external receiver) • Accounts Receivable (external server)	
Other Interested Stakeholders:	• Marketing — Interested in sales activity in order to plan new promotions. • Procurement — Interested in sales activity in order to replenish inventory. • Management — Interested in order activity in order to evaluate company performance and customer (member) satisfaction.	
Description:	This use case describes the event of a club member submitting a new order for SoundStage products. The member's demographic information as well as his or her account standing is validated. Once the products are verified as being in stock, a packing order is sent to the warehouse for it to prepare the shipment. For any product not in stock, a back order is created. On completion, the member will be sent an order confirmation.	

Gambar 2.15 ***High Level Version of Place New Order Use-Case Narrative***

(Sumber: Whitten & Bentley, 2007:257)

Berikut di bawah ini adalah deskripsi *use case narrative* tingkat lanjutan:

a. *Precondition*

Status sistem pada keadaan awal sebelum sistem berjalan.

b. *Trigger*

Suatu hal ataupun kejadian yang memicu sistem berjalan.

c. *Typical course of events*

Events yang berjalan secara sekuensial yang dilakukan oleh *actor* dan sistem.

d. *Alternate courses*

Alternatif perilaku yang dijalankan oleh *actor* atau sistem di luar rute utama ketika terjadi sebuah kondisi khusus.

e. *Conclusion*

Kesimpulan yang dapat diambil ketika *use case* telah selesai.

f. *Postcondition*

Status keadaan akhir dari sistem ketika *use case* telah selesai.

g. *Business rules*

Aturan-aturan yang mendefinisikan kebijakan dan prosedur pada bisnis yang harus ditaati oleh sistem.

h. *Implementation constraints and specifications*

Mengimplementasikan batasan dan spesifikasi khusus pada sistem yang memiliki dampak pada saat *use case* direalisasikan.

i. *Assumptions*

Asumsi-asumsi yang dibuat oleh pembuat ketika mendokumentasikan *use case*.

j. *Open issues*

Pertanyaan atau isu yang perlu diselesaikan sebelumnya.

Precondition:	The party (individual or company) submitting the order must be a member.	
Trigger:	This use case is initiated when a new order is submitted.	
Typical Course of Events:	**Actor Action**	**System Response**
	Step 1: The club member provides his or her demographic information as well as order and payment information.	**Step 2:** The system responds by verifying that all required information has been provided. **Step 3:** The system verifies the club member's demographic information against what has been previously recorded. **Step 4:** For each product ordered, the system validates the product identity. **Step 5:** For each product ordered, the system verifies the product availability. **Step 6:** For each available product, the system determines the price to be charged to the club member. **Step 7:** Once all ordered products are processed, the system determines the total cost of the order. **Step 8:** The system checks the status of the club member's account. **Step 9:** The system validates the club member's payment if provided. **Step 10:** The system records the order information and then releases the order to the appropriate distribution center (warehouse) to be filled. **Step 11:** Once the order is processed, the system generates an order confirmation and sends it to the club member.
Alternate Courses:	**Alt-Step 2:** The club member has not provided all the information necessary to process the order. The club member is notified of the discrepancy and prompted to resubmit. **Alt-Step 3:** If the club member information provided is different from what was previously recorded, verify what was recorded is current, then update the club member information accordingly. **Alt-Step 4:** If the product information the club member provided does not match any of SoundStage's products, notify the club member of the discrepancy and request clarification. **Alt-Step 5:** If the quantity ordered of the product is not available, a back order is created. **Alt-Step 8:** If the status of the club member's account is not in good standing, record the order information and place it in hold status. Notify the club member of the account status and the reason the order is being held. Terminate use case. **Alt-Step 9:** If the payment the club member provided (credit card) cannot be validated, notify the club member and request an alternative means of payment. If the club member cannot provide an alternate means, cancel the order and terminate the use case.	
Conclusion:	This use case concludes when the club member receives a confirmation of the order.	
Postcondition:	The order has been recorded and if the ordered products were available, they were released. For any product not available a back order has been created.	
Business Rules:	• The club member responding to a promotion or a member using credits may affect the price of each ordered item. • Cash or checks will not be accepted with the orders. If provided, they will be returned to the club member. • The club member is billed for products only when they are shipped.	
Implementation Constraints and Specifications:	• GUI to be provided for Member Services associate, and Web screen to be provided for club member.	
Assumptions:	Procurement will be notified of back orders by a daily report (separate use case).	
Open Issues:	1. Need to determine how distribution centers are assigned.	

Gambar 2.16 ***Expanded Version of Place New Order Use-Case Narrative***

(Sumber: Whitten & Bentley, 2007:259)

2.1.5.3 *Activity Diagram*

Mengacu pada pendapat Whitten dan Bentley (2007) *Activity Diagram* merupakan diagram yang digunakan untuk merepresentasikan alur dari proses bisnis secara grafis, langkah-langkah dari *use case* dan logika dari karakteristik objek. Tujuan *Activity Diagram* adalah untuk memodelkan aksi yang akan dilakukan saat suatu operasi dieksekusi dari aksi tersebut.

Notasi-notasi yang ada pada *activity diagram* adalah sebagai berikut:

a. *Initial node*

Gambar 2.17 Notasi *Initial Node* pada *Activity Diagram*

Disimbolkan dengan bentuk lingkaran solid yang merepresentasikan proses awal yang sedang berjalan.

b. *Actions*

Gambar 2.18 Notasi *Actions* pada *Activity Diagram*

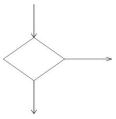
Disimbolkan dengan bentuk persegi panjang dengan sudut melengkung yang merepresentasikan langkah-langkah secara individu.

c. *Flow*

Gambar 2.19 Notasi *Flow* pada *Activity Diagram*

Disimbolkan dengan bentuk garis panah pada diagram, merepresentasikan proses yang berlangsung pada aksi yang sedang dijalankan.

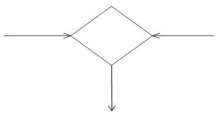
d. *Decision*



Gambar 2.20 Notasi *Decision* pada *Activity Diagram*

Disimbolkan dengan bentuk belah ketupat dengan satu arah garis panah masuk dan dua atau lebih arah garis panah keluar yang merepresentasikan kondisi arah pilihan.

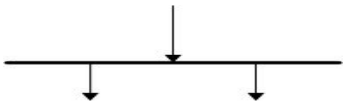
e. *Merge*



Gambar 2.21 Notasi *Merge* pada *Activity Diagram*

Disimbolkan dengan bentuk “belah ketupat” dengan dua atau lebih arah garis panah masuk dan satu arah garis panah keluar, merepresentasikan kondisi pilihan yang menyatu.

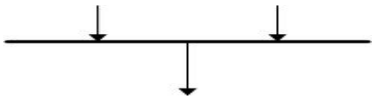
f. *Fork*



Gambar 2.22 Notasi *Fork* pada *Activity Diagram*

Disimbolkan dengan bentuk garis hitam dengan satu arah garis panah masuk dan dua atau lebih arah garis panah keluar, merepresentasikan kondisi yang berjalan secara bersamaan.

g. *Join*



Gambar 2.23 Notasi *Join* pada *Activity Diagram*

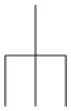
Disimbolkan dengan bentuk garis hitam dengan dua atau lebih garis panah masuk dan satu arah garis panah keluar, merepresentasikan kondisi yang menyatu.

h. *Activity Final*

Gambar 2.24 Notasi *Activity Final* pada *Activity Diagram*

Disimbolkan dengan bentuk lingkaran solid berada pada lingkaran kosong yang menandakan bagian akhir dari proses.

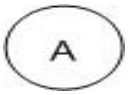
i. Subactivity Indicator



Gambar 2.25 Notasi *Subactivity Indicator* pada *Activity Diagram*

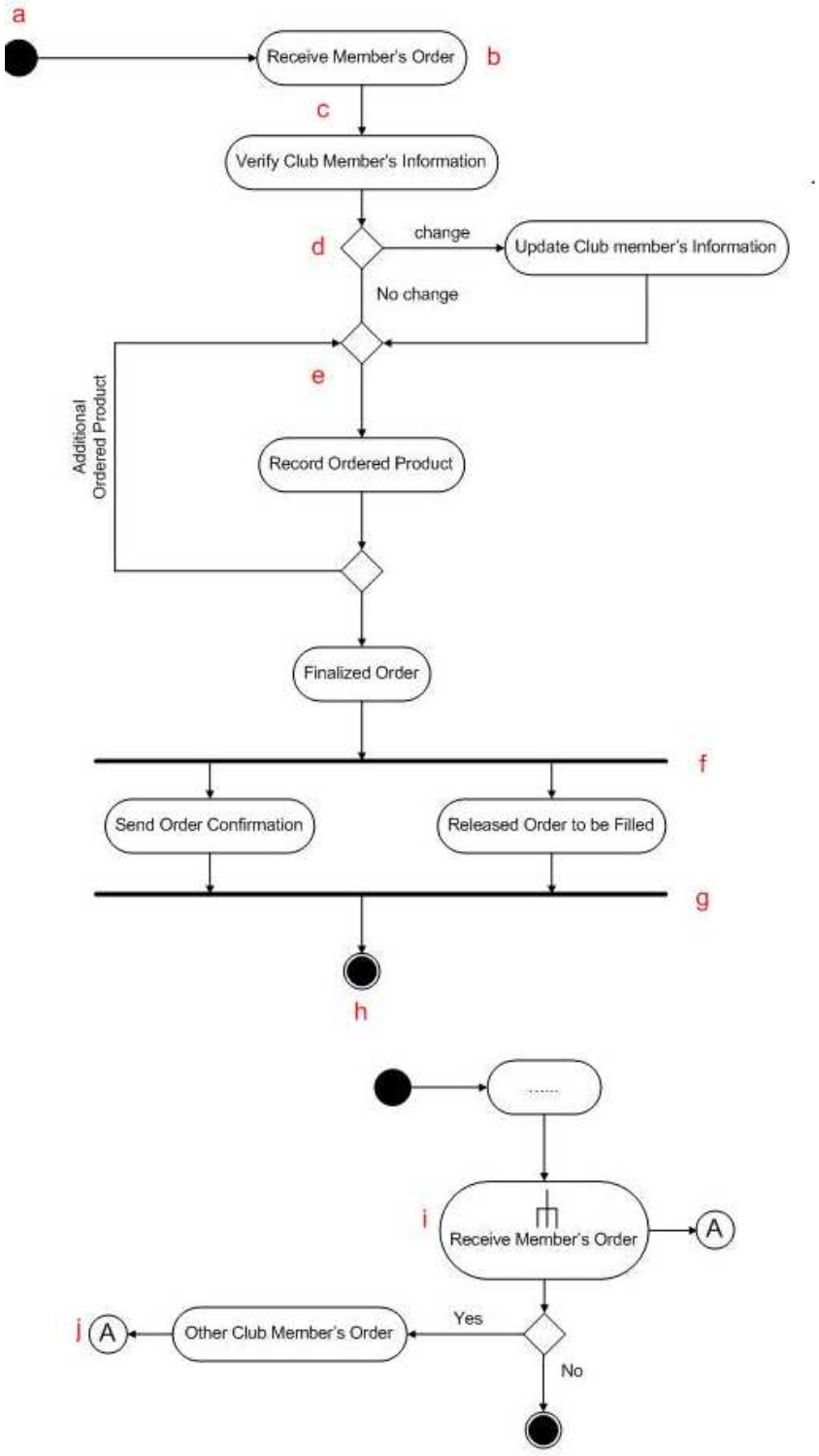
Disimbolkan dengan bentuk “garpu” yang merepresentasikan bagian tersebut dijelaskan lebih mendalam pada bagian *activity diagram* yang lain.

j. Connector



Gambar 2.26 Notasi *Connector* pada *Activity Diagram*

Disimbolkan dengan bentuk lingkaran berisi huruf yang digunakan sebagai penanda bagian.



Gambar 2.27 Contoh *Activity Diagram*

2.1.5.4 *Sequence Diagram*

Mengacu pada pendapat Whitten dan Bentley (2007) *Sequence Diagram* merupakan diagram UML yang memodelkan penggunaan *use case* secara logika dengan menggambarkan interaksi dari pesan yang terjadi diantara objek dalam satuan waktu yang sekuensial.

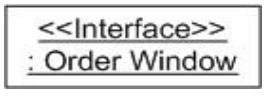
Notasi – notasi yang terdapat pada *sequence diagram* adalah sebagai berikut:

a. *Actor*

Gambar 2.28 Notasi *Actor* pada *Sequence Diagram*

Actor melakukan interaksi dengan *user interface* dengan menggunakan simbol *actor* yang terdapat pada *use case diagram*. *Actor* dapat juga disimbolkan dengan penulisan "<<Actor>>" sebagai bentuk simbol yang lain dari *actor*.

b. *Interface class*

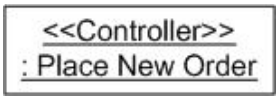


Gambar 2.29 Notasi *Interface Class* pada *Sequence Diagram*

Disimbolkan dengan bentuk kotak yang mengindikasikan kode kelas dari *user interface*. Untuk memperjelas makna dari simbol tersebut

biasanya diberikan “<<Interface>>” sebagai simbol tambahan. Terdapat juga simbol “:” yang berarti proses yang sedang berjalan dan simbol “---” yang menandakan “kehidupan” proses dari sekuensial tersebut.

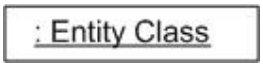
c. *Controller class*



Gambar 2.30 Notasi *Controller Class* pada *Sequence Diagram*

Untuk menjelaskan *controller class* yang ada pada diagram *sequence*. Biasanya ditambahkan dengan simbol “<<Controller>>” untuk memperjelas *controller class* tersebut.

d. *Entity classes*



Gambar 2.31 Notasi *Entity Classes* pada *Sequence Diagram*

Merupakan entitas dari suatu *class* yang disimbolkan dengan bentuk kotak. Notasi ini dibutuhkan agar dapat berkolaborasi dengan tahapan sekuensial. Simbol “:” mendefinisikan *object instance*.

e. *Messages*

Gambar 2.32 Notasi *Messages* pada *Sequence Diagram*

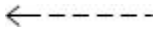
Disimbolkan dengan bentuk garis panah horizontal yang menggambarkan *input* pesan yang dikirimkan ke kelas-kelas.

f. *Activation bars*

Gambar 2.33 Notasi *Activation Bars* pada *Sequence Diagram*

Disimbolkan dengan bentuk persegi panjang *vertical* mengidentifikasikan alur hidup dari setiap *instance object* yang ada selama periode waktu tertentu.

g. *Return messages*



Gambar 2.34 Notasi *Return Messages* pada *Sequence Diagram*

Disimbolkan dengan bentuk garis panah titik-titik horizontal yang mengartikan pesan yang dikirim balik ke suatu *instance object*.

h. *Self-call*

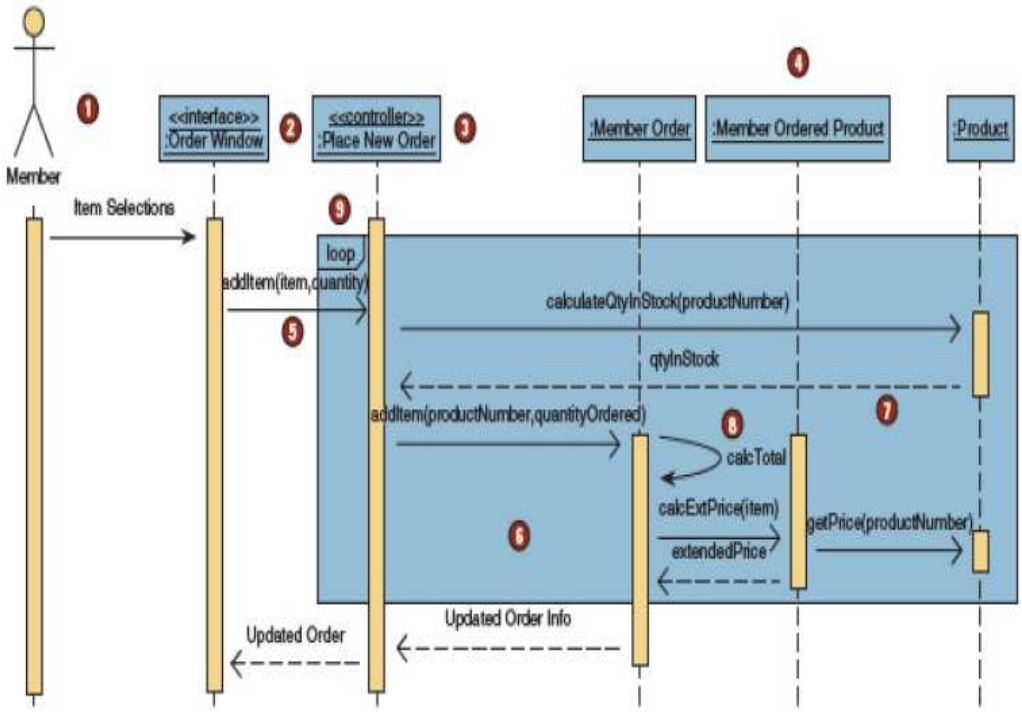
Gambar 2.35 Notasi *Self Call* pada *Sequence Diagram*

Disimbolkan dengan bentuk garis panah melengkung. Notasi ini berarti suatu objek yang memanggil fungsi dari dirinya sendiri.

i. *Frame*

Gambar 2.36 Notasi *Frame* pada *Sequence Diagram*

Disimbolkan dengan bentuk persegi panjang dengan judul di dalamnya yang mengindikasikan suatu *controller* yang dibutuhkan untuk melakukan pengulangan yang melewati semua *item*.



Gambar 2.37 Contoh *System Sequence Diagram*

(Sumber: Whitten & Bentley, 2007:659)

2.1.5.5 *Class Diagram*

Mengacu pada pendapat Whitten dan Bentley (2007) *Class Diagram* merupakan representasi grafis dan deskripsi dari *class* dan struktur objek statis sebuah sistem yang menjelaskan hubungan suatu objek kelas dengan objek kelas lainnya.

Untuk membuat *Class Diagram* ada beberapa langkah yang perlu diperhatikan diantaranya adalah:

a. Mengidentifikasi asosiasi dan keberagaman pada objek kelas untuk mencari hubungan yang terjadi baik diantara atribut-atribut yang ada pada objek kelas itu sendiri dan hubungan antara objek kelas yang satu dengan yang lain.

❖ *Association & Multiplicity*

Asosiasi merupakan bagian dari *class diagram* yang menjelaskan hubungan simetris terhadap objek-objek yang perlu diketahui antara suatu kelas dengan kelas lainnya.

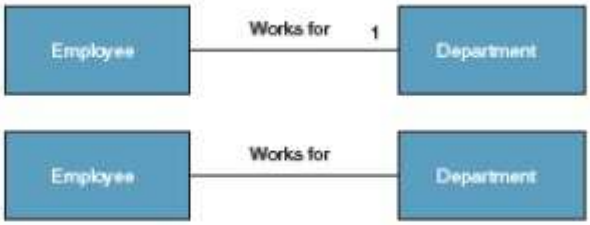
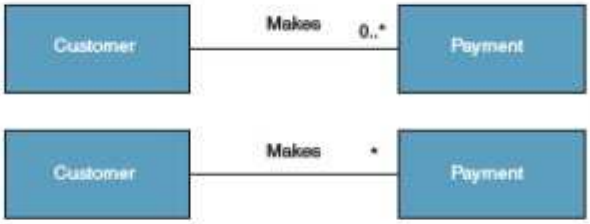
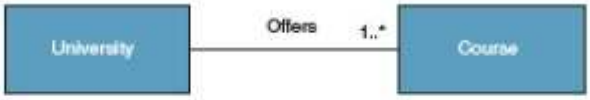
Gambar 2.38 Contoh *Association Class Diagram*

(Sumber: Whitten & Bentley, 2007:406)

Multiplicity adalah angka maksimum dan minimum dari kejadian pada suatu kelas objek untuk kejadian tunggal dari objek kelas yang berhubungan. Berikut ini adalah sebuah tabel *multiplicity* yang berisi mengenai notasi *multiplicity* dan pengertiannya.

Tabel 2.1 Tabel *Multiplicity* pada *Class Diagram*

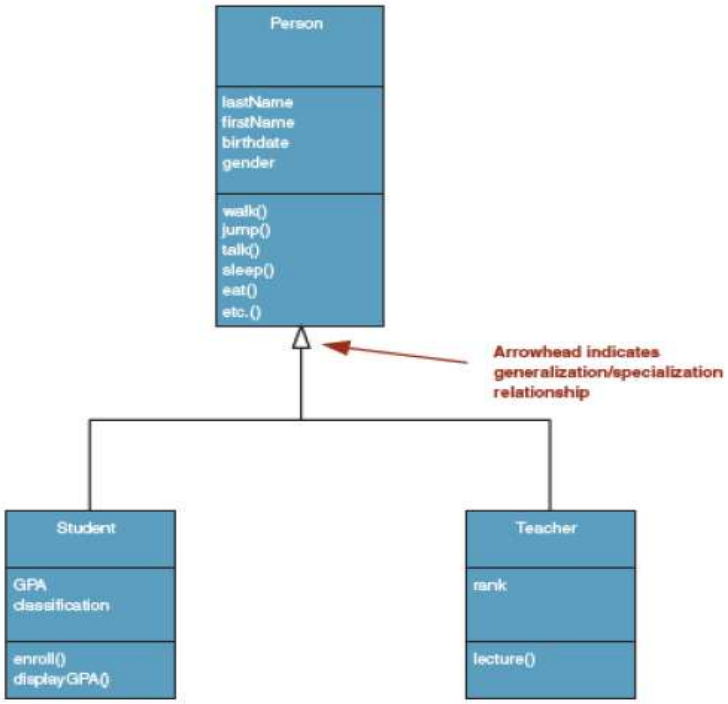
(Sumber: Whitten & Bentley, 2007:377)

Multiplicity	UML Multiplicity Notation	Association with Multiplicity	Association Meaning
Exactly 1	1 — or — *leave blank*		An employee works for one and only one department.
Zero or 1	0..1		An employee has either one or no spouse.
Zero or more	0..* — or — *		A customer can make no payment up to many payments.
1 or more	1..*		A university offers at least 1 course up to many courses.
Specific range	7..9		A team has either 7, 8, or 9 games scheduled.

b. Mengidentifikasi hubungan yang bersifat generalisasi ataupun spesialisasi yang biasa disebut dengan *classification hierarchies* untuk menentukan *supertype class* (*abstract* atau *parent*) dan *subtype class* (*concrete* atau *child*).

❖ *Generalization*

Generalisasi merupakan bagian dari *class diagram* yang menjelaskan hubungan yang terjadi pada suatu kelas dengan kelas yang lain. Generalisasi adalah hubungan yang mendeskripsikan kelas *parent* dan kelas *child* pada suatu kelas hierarki.



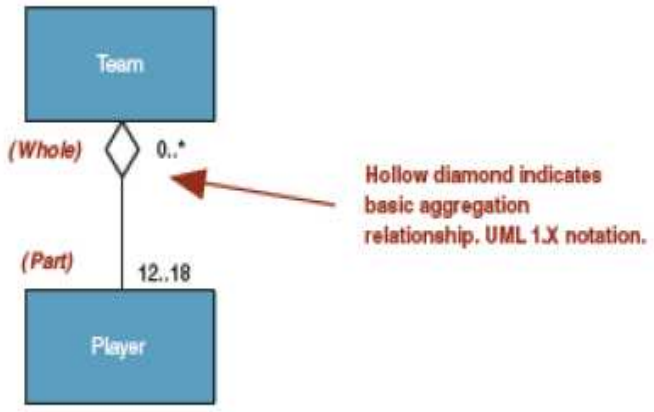
Gambar 2.39 Contoh *Generalization Class Diagram*

(Sumber: Whitten & Bentley, 2007: 376)

c. Mengidentifikasi hubungan yang bersifat agregasi atau komposisi untuk menentukan apakah atribut yang ada pada suatu objek kelas merupakan komposisi ataupun agregrasi dari objek kelas yang lain.

❖ *Aggregation*

Agregasi merupakan bagian dari *class diagram* yang menjelaskan hubungan suatu objek pada suatu kelas yang "merupakan bagian" dengan kelas yang lain. Hubungan yang terjadi di sini adalah hubungan asimetris dimana dapat dikatakan suatu objek yang ada pada kelas B dapat ditemukan pada kelas A namun objek yang ada pada kelas A tidak dapat ditemukan pada kelas B. Hubungan inilah yang disebut dengan hubungan agregasi.

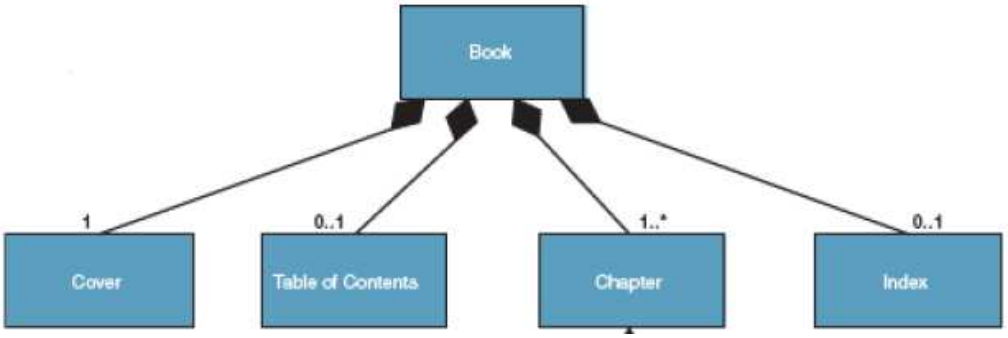


Gambar 2.40 Contoh *Aggregation Class Diagram*

(Sumber: Whitten & Bentley, 2007:379)

❖ *Composition*

Komposisi merupakan bagian dari *class diagram* yang menjelaskan hubungan suatu objek pada suatu kelas dengan kelas yang lain. Hubungan antar objek yang terjadi di sini adalah hubungan simetris dimana dapat dikatakan suatu objek yang ada pada kelas B dapat ditemukan pada kelas A dan begitu juga sebaliknya. Hubungan inilah yang disebut dengan hubungan komposisi.



Gambar 2.41 Contoh *Composition Class Diagram*

(Sumber: Whitten & Bentley, 2007:379)

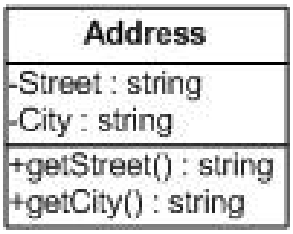
d. Mempersiapkan *class diagram* dengan mengetahui aturan bahwa dalam *class diagram* tidak dibutuhkan *foreign keys* dan atribut dari *primary key*.

Mengacu pada pendapat Whitten dan Bentley (2007) UML memiliki 3 level dari *visibility* yaitu:

a. *Public* – dinotasikan dengan simbol “+”. Atribut *public* dapat diakses dan *public method* dapat dipanggil oleh *method* lain oleh *class* lainnya.

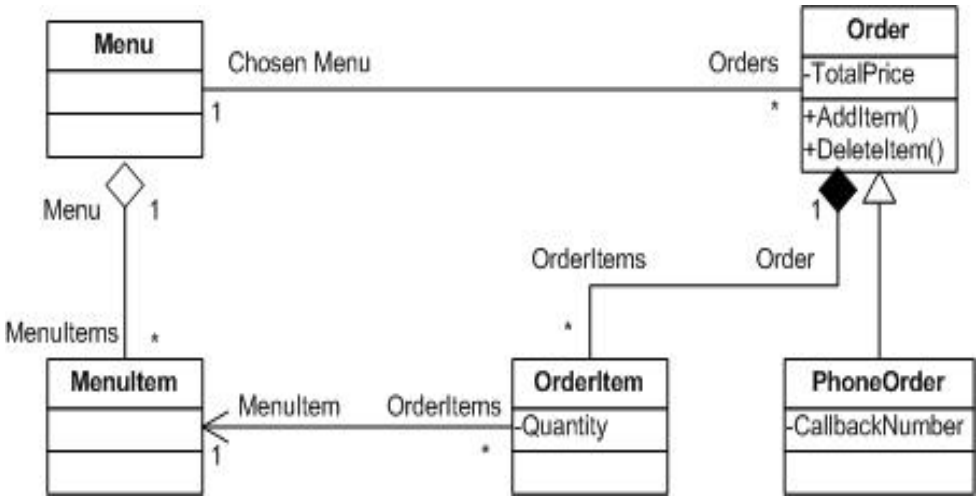
b. *Protected* – dinotasikan dengan simbol “#”. Atribut *protected* dapat diakses dan *protected method* dapat dipanggil oleh *method* lain di *class* dimana atribut dan *method* didefinisikan dan di *subclass* dari *class* tersebut.

c. *Private* – dinotasikan dengan simbol “-“. Atribut *private* dapat diakses dan *private method* dapat dipanggil oleh *method* lain yang didefinisikan pada *class* itu sendiri .



Gambar 2.42 *Visibility Attribute* dan *Method* pada *Class Diagram*

(Sumber : Whitten & Bentley, 2007:651)



Gambar 2.43 Contoh *Class Diagram*

2.1.6 Rekayasa Perangkat Lunak

2.1.6.1 Pengertian Rekayasa Perangkat Lunak

Mengacu pada pendapat Pressman (2010) rekayasa perangkat lunak merupakan aplikasi pendekatan yang sistematis, disiplin, dapat dihitung untuk pengembangan, operasi, dan pemeliharaan perangkat lunak. Komitmen terhadap kualitas menjadi salah satu pendekatan teknik (rekayasa perangkat lunak).

Tiga lapisan yang menjadi dasar *quality focus*, adalah:

a. Proses

Proses merupakan fondasi yang menghubungkan lapisan teknologi yang ada dan mengembangkan perangkat lunak yang

rasional. Proses akan mendefinisikan *framework* yang harus dibuat demi keefektifan penyampaian rekayasa perangkat lunak. Proses membentuk dasar dari kontrol manajemen dan mendirikan konteks yang metode tekniknya diaplikasikan untuk pembentukan produk pekerjaan (model, dokumen, data, laporan, *form,* dan lainnya), pembentukan *milestone*, penjaminan kualitas dan pengaturan perubahan secara tepat.

b. Metode

Metode merupakan langkah atau cara secara teknik untuk membangun perangkat lunak. Metode tersebut terdiri dari komunikasi, analisa kebutuhan, model desain, konstruksi program, pengujian, dan *support.*

c. Tools

Tools merupakan suatu *support* semi-otomatis atau otomatis untuk proses dan metode. CASE (*Computer-Aided Software Engineering*) adalah sistem yang berfungsi untuk mendukung perkembangan perangkat lunak dan dibuat saat *tools* diintegrasikan dengan tujuan agar informasi yang dibuat oleh satu *tool* dapat digunakan dengan yang lainnya.

2.1.6.2 Metode *Waterfall*

Mengacu pada pendapat Pressman (2010) *Waterfall model* merupakan metode yang terdiri dari 5 tahap pengembangan yang meliputi:

a. *Communication*

Berisi mengenai *project initiation* dan *requirements gathering* terhadap *software* yang akan dibuat.

b. *Planning*

Merupakan estimasi, penjadwalan, dan *tracking* terhadap *software* yang akan dibuat.

c. *Modeling*

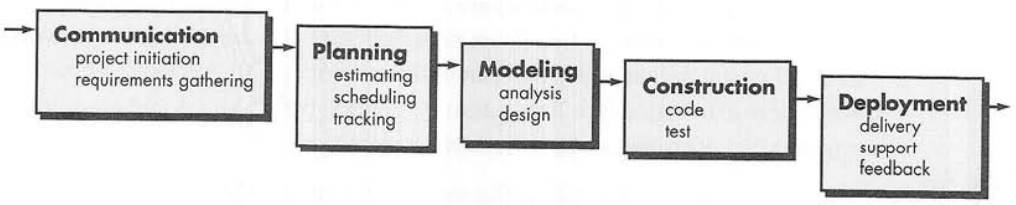
Meliputi pemodelan analisis dan desain dari *software* yang akan dibuat.

d. *Construction*

Merupakan pembuatan *code* dan *test* terhadap *code* yang dibuat pada *software*.

e. *Deployment*

Meliputi proses *delivery, support,* dan *feedback* terhadap *software* yang sudah diselesaikan.

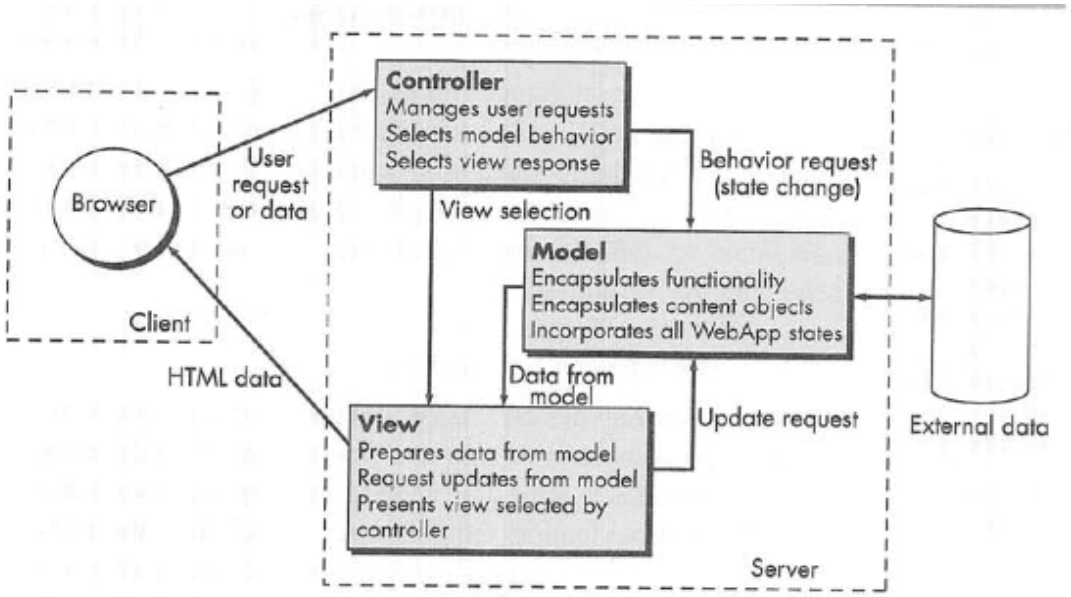


Gambar 2.44 *Waterfall Model*

(Sumber: Pressman, 2010:39)

2.1.6.3 Arsitektur *Model View Controller* (MVC)

Mengacu pada pendapat Pressman (2010) arsitektur *Model View Controller* (MVC) merupakan salah satu dari model infrastruktur *Web App* yang disarankan. Model sebagai objek model yang mengandung semua aplikasi – konten yang spesifik dan proses logikal, termasuk semua konten objek, akses ke data eksternal/sumber informasi, dan semua fungsi proses yang aplikasi spesifik. *View* mengandung semua *interface* – fungsi yang spesifik dan memperbolehkan presentasi dari konten dan proses logikal, termasuk semua konten objek, akses ke data eksternal/sumber informasi, dan semua fungsi proses yang dibutuhkan oleh *end user*. *Controller* mengatur akses ke model dan *view* serta melakukan koordinasi alur data. *View* diperbaharui oleh *controller* dengan data dari model. *User request* atau data diatur oleh *controller*.



Gambar 2.45 Arsitektur *Model View Controller* (MVC)

(Sumber: Pressman, 2010:387)

2.1.7 Interaksi Manusia Komputer

2.1.7.1 Pengertian Interaksi Manusia dan Komputer

Mengacu pada pendapat Shneiderman dan Plaisant (2010) interaksi manusia dan komputer merupakan ilmu yang mempelajari implementasi dan perancangan interaksi manusia sebagai *user* dengan sistem komputer. Perancangan yang dilakukan dengan memanfaatkan kelebihan teknologi yang dimiliki oleh komputer sehingga dapat menjangkau kapasitas dan kebutuhan manusia sebagai *user*.

2.1.7.2 Lima Faktor Manusia Terukur

Mengacu pada pendapat Shneiderman dan Plaisant (2010) ada lima pengukuran yang dijadikan pedoman dasar pengukuran, yaitu:

a. Waktu belajar

Pengukuran berdasarkan berapa lama waktu yang diperlukan oleh *user* untuk mempelajari langkah-langkah untuk melakukan tugas.

b. Kecepatan kinerja

Pengukuran berdasarkan berapa lama waktu yang diperlukan untuk menyelesaikan tugas.

c. Tingkat kesalahan *user*

Pengukuran berdasarkan berapa banyak kesalahan dan kesalahan apa saja yang dilakukan oleh *user* dalam melakukan suatu tugas.

d. Daya ingat *user* setelah jangka waktu tertentu

Pengukuran berdasarkan berapa lama *user* dapat mengelola dan mempertahankan pengetahuannya dalam jangka waktu tertentu. Daya ingat dihubungkan dengan waktu belajar dan frekuensi *user* yang merupakan peranan penting.

e. Kepuasan Subjektif

Pengukuran berdasarkan tingkat kepuasan *user* terhadap tampilan antar muka / *interface.*

2.1.7.3 Delapan Aturan Emas Desain Antar Muka

Mengacu pada pendapat Shneiderman dan Plaisant (2010) untuk merancang *User Interface* atau antarmuka *user*, ada beberapa hal yang harus diperhatikan dalam mengembangkan antarmuka. Delapan aturan emas merupakan beberapa hal yang berkaitan dengan antarmuka, yaitu:

a. Konsistensi

Hal ini berarti bahwa konsistensi harus dapat diterapkan pada berbagai macam aksi seperti pada *prompts, menu* dan *help screen*. Konsistensi pada warna *layout*, kapitalisasi, huruf dan sejenisnya juga harus mampu diterapkan.

b. Penggunaan bahasa yang universal

Hal ini berarti memaksimalkan efektivitas dan efisiensi dari rancangan antar muka yang dibuat. Misalnya pada pemula diberikan bantuan petunjuk cara penggunaan aplikasi dan pada pengguna tingkat atas diberikan fitur *shortcut* untuk mempermudah pemakaian aplikasi.

c. Memberikan umpan balik yang informatif

Untuk setiap aksi yang dilakukan oleh *user*, sistem sebaiknya dapat memberikan umpan balik. Pada aksi yang sering dilakukan oleh *user*, maka umpan balik dari sistem dapat berupa suatu pernyataan persetujuan mengenai aksi yang dilakukan oleh *user*.

d. Merancang dialog yang memberikan solusi

Setiap urutan aksi yang dilakukan oleh *user* diorganisasikan ke sebuah pengelompokkan yang menandakan bagian awal, tengah dan akhir. Misalnya pada situs *e-commerce* yang menyatakan transaksi sudah selesai dilakukan ketika *user* telah melakukan *checkout*.

e. Mencegah terjadinya kesalahan

Sedapat mungkin sistem yang dibuat dapat membantu mencegah terjadi sebuah kesalahan yang serius ketika *user* melakukan sebuah aksi. Misalnya kotak pengisian angka akan berubah menjadi warna abu-abu ketika diisi dengan huruf alfabet.

f. Memberikan pengembalian aksi yang mudah

Sedapat mungkin aksi yang telah dilakukan oleh *user* dapat dibalikkan ke proses yang sebelumnya. Misalnya pada saat *user* ingin kembali dari menu registrasi ke *home*, maka sistem seharusnya dapat mewujudkan aksi tersebut.

g. Mendukung pusat kendali internal

Hal ini berarti *user* memegang kendali atas aksi yang dilakukan. *User* dapat merasakan bahwa setiap aksi yang dilakukan merupakan pilihan dari *user* sendiri dan bukan dikontrol oleh sistem.

h. Mengurangi beban ingatan jangka pendek

Sistem mampu memberikan kemudahan kepada *user* terhadap berbagai aksi yang akan dilakukan. Misalnya sistem memberi tahu *user* mengenai *link* yang telah dibuka sebelumnya dengan memberi tanda warna pada *link* tersebut.

2.2 Teori Khusus

2.2.1 PHP dan MySQL

2.2.1.1 PHP

Mengacu pada pendapat Welling dan Thompson (2009) PHP adalah *server-side scripting language* yang dirancang secara khusus untuk *user web*. Dalam satu halaman HTML, *user* bisa menyertakan kode PHP yang kemudian akan dijalankan pada *server* setiap kali halaman tersebut dikunjungi. Kode PHP yang ada didalamnya akan diterjemahkan pada *web server* dan menghasilkan *output* HTML yang akan dilihat oleh pengunjung. PHP merupakan singkatan dari *PHP Hypertext Preprocessor*.

Beberapa kelebihan dari PHP adalah sebagai berikut:

a. *Performance*

Performa pada PHP sangat cepat. Dengan menggunakan *server* yang tidak mahal, PHP dapat melayani jutaan pengunjung tiap harinya.

b. *Scalability*

PHP memiliki arsitektur "*shared-nothing*" yang berarti *user* dapat dengan efektif dan murah dalam mengimplementasikan skala horizontal dengan angka yang besar pada komoditas *server*.

c. *Database Integration*

PHP mempunyai "*native connections*" yang tersedia untuk berbagai macam sistem *database*. Sebagai contoh pada MySQL, *user* dapat langsung menghubungkan PHP dengan Postgre SQL, Oracle, dbm, FilePro dan sejenisnya.

d. *Built-in Libraries*

Karena PHP dirancang untuk penggunaan pada web, PHP dibangun dalam berbagai macam "*built-in function"* untuk melakukan berbagai macam tugas yang berhubungan dengan web.

e. *Cost*

Dari segi harga, PHP dapat di-*download* secara gratis.

f. *Ease of Learning*

Syntax yang ada pada PHP didasarkan pada bahasa pemrograman lainnya seperti C, C++ atau Java.

g. *Object-Oriented Support*

PHP memiliki fitur objek orientasi yang dirancang dengan baik seperti *inheritance, private and protected attributes, abstract classes,* dan sejenisnya.

h. *Portability*

PHP tersedia dan dapat digunakan untuk berbagai macam sistem operasi seperti *Linux, Unix,, OS X* ataupun *Microsoft Windows*.

i. *Flexibility of Development Approach*

PHP memungkinkan *user* untuk dapat mengimplementasikan tugas sederhana dengan mudah, dan dapat dengan mudah beradaptasi dengan aplikasi besar yang menggunakan *framework.*

j. *Source Code*

PHP memiliki *free source code* yang terbuka bagi setiap orang yang ingin melakukan modifikasi atau menambahkan perintah ke dalam PHP tersebut.

k. *Availability of Support and Documentation*

PHP memudahkan *use*r untuk memanfaatkan fitur dokumentasi dan bantuan yang disediakan oleh perusahaan yang membuat bahasa pemrograman PHP tersebut.

2.2.1.2 MySQL

Mengacu pada pendapat Welling dan Thompson (2009) MySQL adalah *relational database management system* (RDBMS) yang sangat cepat dan handal. MySQL memungkinkan *user* untuk dapat menyimpan, mencari, mensortir, dan mengambil data dengan efisien. MySQL adalah sebuah *database multithread* yang dapat digunakan oleh *multiuser*. Bahasa yang digunakan dalam MySQL adalah *Structured Query Language* (SQL) yang merupakan *standard database query language*.

Beberapa keunggulan yang terdapat pada MySQL adalah sebagai berikut :

a. *Performance*

MySQL tidak diragukan lagi dalam hal kecepatan. MySQL dikatakan memiliki kecepatan memproses yang sama dengan RDBMS lainnya seperti Oracle.

b. *Low Cost*

MySQL tersedia dengan gratis pada lisensi *open source* atau harga yang relatif murah pada lisensi komersial. Lisensi komersial dibutuhkan jika *user* ingin mendistribusikan ulang MySQL sebagai bagian dari aplikasi dan tidak ingin lisensi berupa *open source*.

c. *Ease of Use*

Penggunaan yang mudah karena *database* pada umumnya menggunakan SQL sebagai *standard query language*.

d. *Portability*

MySQL dapat digunakan dalam berbagai macam sistem operasi seperti Unix dan Microsoft Windows.

e. *Source Code*

MySQL termasuk bahasa yang mempunyai sistem *open source* yang dapat dimodifikasi atau ditambahkan perintah ke dalamnya.

f. Availability of Support

MySQL memiliki fitur bantuan seperti *training, consulting,* dan *certification* yang disediakan oleh perusahaan.

2.2.2 *Radio Frequency Identification* (RFID)

2.2.2.1 Pengertian RFID

Mengacu pada pendapat Hunt dan Puglia (2007) pengertian sebuah RFID (*Radio Frequency Identification*) adalah teknologi komunikasi *wireless* yang digunakan untuk mengidentifikasi "*tagged objects*" atau manusia.

2.2.2.2 Komponen RFID

Mengacu pada pendapat Hunt dan Puglia (2007) RFID terdiri dari 3 komponen, yaitu:

a. *Tag*, terdiri dari *chip* semi konduktor, antena, dan terkadang terdapat baterai.

b. *Interrogator* atau biasa disebut dengan *Tag Reader*, terdiri dari antena, modul elektronik radio frekuensi, dan pengatur modul elektronik.

c. Pengontrol atau biasa disebut juga dengan *Host*, biasanya menggunakan komputer yang menjalankan perangkat lunak untuk pengaturan dan untuk *database*.

2.2.2.3 Jenis - Jenis RFID

Mengacu pada pendapat Hunt dan Puglia (2007) jenis RFID dibedakan berdasarkan 2 faktor. Faktor pertama adalah jenis RFID berdasarkan komponen dimana RFID dibagi menjadi 2 jenis, yaitu aktif dan pasif. RFID aktif mengandung sumber tenaga seperti baterai di dalam komponennya, sedangkan RFID pasif tidak. RFID aktif menggunakan sumber tenaga yang tersedia untuk memancarkan informasi sehingga jarak frekuensi sinyalnya lebih jauh dibandingkan dengan yang pasif. RFID pasif hanya mengandalkan sinyal gelombang yang dipancarkan oleh *reader* yang digunakan sebagai pemicu energi sehingga baru mengirimkan informasi yang terdapat di kartu RFID tersebut. Sedangkan faktor kedua adalah jenis RFID berdasarkan tipe memori dimana RFID dibagi menjadi 2 jenis, yaitu *read only* dan *read/write*. RFID *read only* hanya bisa untuk dibaca saja. RFID

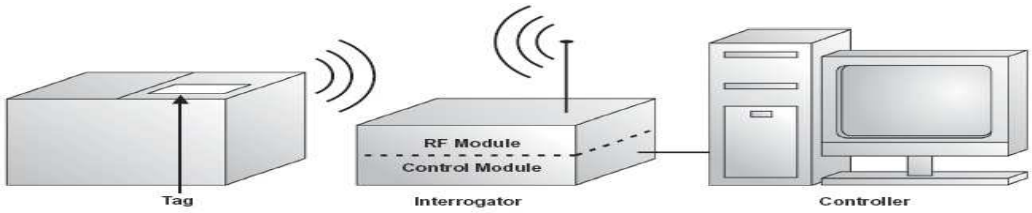
read/write lebih sering dipanggil *smart card*. Kartu ini lebih menyediakan fleksibilitas dibandingkan RFID *read only*. Kartu ini dapat menyimpan data yang banyak dan mempunyai alamat memory yang dapat diubah dengan mudah.

2.2.2.4 Cara Kerja RFID

Mengacu pada pendapat Hunt dan Puglia (2007) *Tag card* dan *tag reader* saling berkomunikasi melalui gelombang radio. Jika benda yang didekatkan pada *tag card* masuk ke zona baca dari sebuah *tag reader*, maka *tag reader* akan mengirimkan sinyal ke *tag card* untuk mengambil data yang tersimpan di dalam *tag card* tersebut. Dalam sistem yang paling umum yaitu sistem yang menggunakan *tag* pasif, *tag reader* memancarkan energi gelombang radio untuk menyediakan energi bagi *tag* untuk bisa beroperasi. Sedangkan jika *tag* aktif, baterai dalam label *tag* digunakan untuk memperoleh jangkauan operasi *tag* RFID yang efektif. *Tag* dapat menyimpan berbagai jenis informasi tentang objek yang mereka melekat, salah satunya adalah nomor seri. Setelah *tag reader* menerima data *tag* tersebut, informasi disampaikan ke pengatur melalui antarmuka jaringan standar, seperti LAN *ethernet* bahkan *internet*. Kemudian pengatur dapat mengelola data tersebut sesuai dengan keperluannya.

Dalam sistem RFID dapat terdiri dari banyak *tag reader* yang tersebar didalam gedung. Namun, semua *tag reader* tersebut dalam

dihubungkan ke pengatur tunggal. Demikian juga sebuah *tag reader* dapat berkomunikasi dengan lebih dari satu *tag* secara bersamaan.



Gambar 2.46 Simulasi Penggunaan Komponen RFID

(Sumber: Hunt & Puglia, 2007:6)

2.2.3 *Internet*

Schneider (2011) mengatakan bahwa *internet* adalah sebuah grup dari jaringan komputer yang terkoneksi antara satu dengan yang lainnya. Internet dapat juga dikatakan sebagai suatu jaringan komputer yang memiliki serangkaian set aturan yang spesifik yang terkoneksi dengan jaringan komputer lainnya di berbagai belahan dunia.

2.2.3.1 Fungsi *Internet*

Mengacu pada pendapat Schneider (2011) pada awal pembentukannya *internet* masih digunakan untuk mengontrol sistem persenjataan dan mengirimkan data penelitian. Perkembangan penggunaan *internet* mulai berkembang pada awal tahun 1970-an dimana pada tahun 1972, seorang peneliti yang bernama Ray Tomlinson menulis sebuah program yang dapat mengirim dan menerima data melalui *internet*. Akibat dari metode baru dalam pengiriman informasi tersebut,

penggunaan *internet* semakin berkembang luas. Perkembangan terpesat terjadi pada tahun 1979 – 1989 dimana para *user* sudah banyak yang menggunakan *internet*. Banyaknya penggunaan ini dikarenakan sistem komunikasi yang dapat terkoneksi antar satu dengan yang lainnya dan pengunaan *internet* tersebut banyak digunakan oleh universitas-universitas pada zamannya.

2.2.3.2 *Intranet* dan *Extranet*

Mengacu pada pendapat Schneider (2011) *Intranet* merupakan *interconnected network* yang menggunakan serangkaian protokol TCP/IP yang digunakan dalam lingkup ruang yang kecil misalnya pada sebuah organisasi. *Extranet* adalah *intranet* yang sudah diperluas cakupan lingkungannya yang mencakup entitas-entitas tertentu di luar batas organisasi seperti mitra bisnis, *customer* ataupun *supplier*.

2.2.3.3 *Uniform Resource Locators* (URL)

Mengacu pada pendapat Schneider (2011) *Uniform Resource Locator* (URL) merupakan nama atau singkatan yang merepresentasikan *IP address* dari halaman *web* tertentu. URL tersebut memiliki *protocol* yang digunakan untuk mengakses halaman *web* dan lokasi dari halaman tersebut. Sehingga, URL dapat dikatakan sebagai kombinasi dari nama *protocol* dan nama *domain* untuk dapat mengakses halaman *web* yang dituju.

2.2.3.4 *Web* dan *Mobile Web*

2.2.3.4.1 *World Wide Web*

Mengacu pada pendapat Schneider (2011) *World Wide Web*, atau lebih sederhana, *web*, adalah subset dari komputer di internet yang terhubung antara satu dengan yang lainnya dengan cara tertentu yang membuat mereka dan isinya mudah untuk diakses satu dengan yang lain. Hal yang paling penting mengenai *web* adalah bahwa *web* termasuk dalam standar antarmuka yang mudah digunakan.

2.2.3.4.2 *Mobile Web*

Mengacu pada pendapat Fling (2009) *mobile web application* merupakan aplikasi-aplikasi *mobile* yang tidak perlu di-*install* atau di-*compile* pada sebuah *device*. *Mobile web application* ini menggunakan XHTML, CSS, dan JavaScript. *Mobile web application* dapat memberikan sebuah pengalaman seperti memakai aplikasi kepada *user* ketika menjalankannya di *mobile web browser*. *Web application* memungkinkan *user* untuk dapat berinteraksi dengan isi ataupun konten dalam kondisi yang sebenarnya (*real-time condition*).

Kelebihan pada *mobile web* adalah sebagai berikut:

a. Mudah dibuat dengan HTML dasar, CSS dan JavaScript.
b. Mudah disesuaikan dengan beberapa telepon seluler.
c. Menawarkan pengalaman yang lebih baik dan desain yang lebih kaya kepada *user*.

Kekurangan pada *mobile web* adalah sebagai berikut:

a. Banyak rintangan untuk mendukung beberapa telepon seluler.
b. Tidak semua *mobile web application* mendukung fitur *mode offline*, *location lookup*, *filesystem access*, kamera, dan sebagainya.

2.2.3.5 *Ten Principles of Mobile Interface Design*

Menurut Stark (2012), ada 10 prinsip yang dibutuhkan untuk merancang desain antarmuka untuk ponsel, yaitu :

a. *Mobile Mindset*

Karena perbedaan penggunaan antara *mobile* dan *desktop*, maka sangat penting jika kita mengarahkan pikiran ke dalam *mobile mindset* sebelum melakukan pengembangan dengan cara berusaha menjadi lebih fokus, unik, menawan, dan peka terhadap aplikasi yang akan kita buat.

b. *Mobile Contexts*

Memperhatikan jenis konteks yang akan ditampilkan pada *mobile* bergantung dari jenis pemakainya apakah pemakai tersebut termasuk dalam golongan yang memiliki aktivitas yang padat atau tidak atau pemakai yang “tidak” memiliki aktivitas.

c. *Global Guidelines*

Adanya 7 hal mendasar yang dijadikan sebagai panduan global dalam mengembangkan berbagai jenis aplikasi pada *mobile*, yaitu pada setiap aplikasi *mobile* yang dibuat harus memperhatikan tingkat kecepatan *responsive* aplikasi, pengembangan aplikasi yang berkelanjutan, desain aplikasi yang menggunakan ibu jari sebagai prioritas utama, peletakan menu-menu yang baik sebagai target utama, konten yang dapat berinteraksi dengan pengguna, *control* yang diletakkan pada bagian bawah layar, dan mengurangi *scrolling* pada aplikasi yang dibuat.

d. *Navigation Models*

Memperhatikan penggunaan model navigasi dari salah satu pilihan berikut apakah *none*, *tab bar* atau *drill down* yang akan diimplementasikan pada aplikasi *mobile* yang dibuat.

e. *User Input*

Meningkatkan kemudahan *user* dalam memasukkan huruf, simbol atau angka misalnya dengan penggunaan *keyboard virtual* dengan desain yang baik dan dilengkapi dengan fitur *auto-correct*.

f. *Gestures*

Mendukung penggunaan *gestures-based* yang akan diimplementasikan pada aplikasi *mobile web* untuk meningkatkan interaksi antara *user* dengan aplikasi.

g. *Orientation*

Memperhatikan orientasi dari aplikasi yang dibuat agar tidak menyulitkan *user* dalam menggunakan aplikasi *mobile* misalnya dengan menggunakan layar tipe *portrait* untuk diimplementasikan pada aplikasi *mobile* dan penggunaan *landscape orientation* jika aplikasi yang dibuat menghendaki banyaknya pengetikan yang harus dilakukan oleh *user*.

h. *Communications*

Memperhatikan penggunaan komunikasi dalam pengembangan aplikasi *mobile* misalnya dengan menyediakan *feedback* untuk setiap interaksi yang dilakukan oleh *user*, memberikan peringatan jika terjadi suatu kesalahan, dan konfirmasi terhadap setiap aksi yang dilakukan oleh *user*.

i. *Launching*

Pada saat *user* kembali memasuki aplikasi setelah *user* pernah menggunakan aplikasi tersebut sebelumnya, maka sebaiknya *user* diberikan tampilan terakhir dimana *user* meninggalkan tampilan tersebut terakhir kali.

j. *First Impressions*

Kesan pertama sangat penting dalam suatu aplikasi *mobile* yang dibuat. Kesan pertama yang perlu diperhatikan agar *user*

dapat tertarik menggunakan aplikasi tersebut adalah desain dari *icon* yang digunakan dan tampilan pertama pada saat aplikasi dijalankan.