BAB II

LANDASAN TEORI

2.1. Konsep Dasar Program

2.1.1. Program

A. Pengertian Program

Menurut Sianipar (2013:8) "Program adalah instruksi-instruksi untuk komputer, yang mendeskripsikan apa yang harus dilakukan oleh komputer".

Sedangkan Menurut Kadir (2014:2) "Program adalah kumpulan instruksi yang ditujukan untuk komputer agar komputer dapat melakukan tujuan tertentu yang diharapkan oleh pemakai".

Program terkadang disebut aplikasi. Namun, penggunaan kedua istilah tersebut berbeda dalam hal perspektif. Istilah program digunakan oleh teknologi informasi (IT), sedangkan aplikasi dipakai oleh pemakai program. Secara teknis, program lebih cenderung pada kode yang ditulis oleh pemrogram, sedangkan aplikasi adalah bentuk akhir pembuatan program.

B. Pengertian Pemrograman

Menurut Sianipar (2013:8) "Pemrograman adalah pembuatan suatu program yang dapat dieksekusi oleh suatu komputer agar dapat melakukan tugas-tugas yang diperintahkan".

Pemrograman harus menggunakan bahasa yang dapat dimengerti oleh mesin. Bahasa mesin adalah sekumpulan instruksi primitif. Sedangkan orang yang melakukan pemrograman pada komputer disebut dengan *Programmer*.

Berikut adalah kegiatan-kegiatan penting dalam pemrograman menurut Kadir (2014:3), adalah:

1. Menganalisis masalah dan menuangkan pemecahan masalah dengan algoritma
2. Menuliskan kode
3. Mengompilasi dan membentuk kode yang dapat dieksekusi
4. Melakukan pengujian
5. Mencari penyebab kesalahan

2.1.2. Bahasa Pemrograman

Menurut Munir (2011:16) "Bahasa pemrograman adalah sekumpulan tulisan algoritma atau kode-kode yang berfungsi untuk menjalankan suatu perintah yang ada didalam komputer".

Untuk itu bahasa pemrograman dibagi menjadi 4 tingkatan yaitu:

1. Bahasa Mesin (*Machine Language*)

   Bahasa pemrograman yang hanya dapat dimengerti oleh mesin computer yang didalamnya terdapat CPU yang hanya mengenal dua keadaan yang berlawanan yaitu:

   a. Bila terjadi kontak atau ada arus bernilai 1
   b. Bila tidak ada kontak atau arus bernilai 0

2. Bahasa Tingkat Rendah (*Low Level Language*)

   Karena banyak keterbatasan yang dimiliki bahasa mesin maka dibuatlah symbol yang mudah diingat yang disebut dengan *Mnemonic* (pembantu untuk mengingat). Contoh: bahasa *Assembler* yang dapat menerjemahkan *Mnemonic.*

3. Bahasa Tingkat Menengah (*Middle Level Language*)

   Bahasa pemrograman yang menggunakan aturan-aturan gramatikal dalam penulisan pernyataannya, mudah untuk dipahami dan dimiliki instruksi-instruksi tertentu yang dapat langsung diakses oleh komputer.

   Contoh: Bahasa C

4. Bahasa Tingkat Tinggi (*High Level Language*)

   Bahasa pemrograman yang dalam penulisan pernyataannya mudah dipahami secara langsung.

   a. Bahasa Berorientasi pada Prosedur (*Procedure Oriented Language*)

      Contoh: Algoritma, Fotran, Pascal, Basic, Cobol, PL/1.

   b. Bahasa Berorientasi pada Masalah (*Problem Oriented Language*)

      Contoh: *Report Program Generator* (RPG).

2.1.3. Pemrograman Berorientasi Objek

Menurut Sibero (2010:38) “Pemrograman berorientasi obyek adalah metode pemrograman yang menyusun suatu metode proses dan elemen lainnya menjadi sebuah obyek yang dapat berinteraksi dengan obyek lainnya”.

Bahasa C++, Java, VB.NET dan sejenisnya adalah bahasa pemrograman berorientasi objek, sedangkan kelas adalah tempat berkumpulnya objek yang merupakan ciri khas pemrograman berorientasi objek.

A. Objek (*object*)

Menurut Dougdlas dalam Herlawati (2011:3) ”Objek adalah entitas yang memiliki atribut, karakter (*behavour*) dan kadangkala disertai kondisi (*state*)”.

B. Kelas (*class*)

Menurut Herlawati (2011:3) ”Kelas adalah penggambaran satu set objek yang memiliki atribut dan *behavour* yang sama”.

C. Pembungkusan (*Encapsulation*)

Menurut Herlawati (2011:4) “Pembungkusan sebagai penggabungan potongan - potongan informasi dan perilaku-perilaku spesifik yang bekerja pada informasi tersebut, kemudian mengemasnya menjadi apa yang disebut sebagai objek”.

D. Pewarisan (*Inheritance*)

Menurut Whitten dalam Herlawati (2011:5) “Pewarisan adalah konsep dimana metode atau atribut yang ditentukan didalam sebuah objek kelas dapat diwariskan atau digunakan lagi oleh objek kelas lainnya”.

E. Polimorfisme

Menurut Herlawati (2011:5) “Polimorfisme berarti suatu fungsionalitas yang diimplementasikan dengan berbagai cara yang berbeda”.

2.1.4. Java

Menurut Sun dalam Shalahudin (2010:7) “Java adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer *standalone* ataupun pada lingkungan jaringan”.

Menurut Enterprise (2014:1) “Java bersifat *Write Once, Run Anywhere* (program yang ditulis satu kali dan dapat berjalan pada banyak *platform*)”.

Berikut ini fitur-fitur Java:

1. Berorientasi objek: dalam Java, semua adalah objek
2. Bersifat *platform independent*: Java di-*compile* dalam *bit code platform independent* dan bukan pada mesin *platform* spesifik seperti C dan C++.
3. Sederhana: Java didesain untuk dpat dengan mudah dipelajari.
4. Aman: dengan fitur keamanan Java, sistem bebas Virus dan *powerful*.
5. Bersifat *Architectural-neutral* : *compiler* Java membuat format file objek yang *architectural-neutral*, yang membuat kode yang *decompile* dapat dieksekusi pada berbagai prosesor yang memiliki sistem *runtime* Java.
6. *Portable*: Java bersifat *portable* Karena adanya fitur *platform independent* dan *architectural-neutral*.
7. Kuat dan *powerful*: Java mengeliminasi *error* dengan menjalankan pengecekan pada waktu *compile* dan *runtime.*

8. *Multithreaded*: dengan fitur *multithreaded* Java, program dapat mengerjakan banyak tugas sekaligus.
9. Terinterpretasi: kode *bit* Java ditranslasi secara langsung pada instruksi mesin dan tidak disimpan.
10. Performa tinggi: Java memiliki performa yang tinggi karena menggunakan *compiler* langsung.
11. Terdistribusi: Java didesain untuk lingkungan distribusi internet.
12. Dinamis: java lebih dinamis dari C dan C++ karena Java didesain untuk beradaptasi dengan lingkungan pengembangan.

Untuk dapat menggunakan Java untuk membuat sebuah program, maka diperlukan beberapa *tools software*, sebagai berikut:

1. Sistem Operasi (windows, Mac, Linux)
2. Java JDK
3. Notepad atau editor teks sejenis

A. Sintak Dasar Java

Berikut ini penjelasan singkat mengenai kelas, objek, method dan variable instance.

1. Objek

   Objek memiliki *state* (keadaan) dan *behavi*or (perilaku).

2. Kelas

Sebuah kelas dapat didefinisikan sebagai sebuah *template* atau *blue print* yang mendeskripsikan perilaku atau keadaan yang didukung oleh objek merupakan tipe kelas tersebut.

3. Method

   Method pada dasarnya adalah sebuah perilaku. Sebuah kelas dapat memiliki banyak method. Penulisan logika, manipulasi data dan eksekusi dari aksi lainnya semuanya dilakukan didalam method.

4. Variable instance

   Setiap objek memiliki variable uniknya sendiri.

B. Identifier Java

Nama-nama yang digunakan untuk kelas, variable, dan method disebut *identifier*.

C. *Modifier* Java

Modifier adalah kata kunci yang ditambahkan pada definisi variable untuk menentukan artinya.

D. Variabel Java

Setiap *variable* dalam Java memiliki tipe spesifik yang menentukan ukuran dan layout memori yaitu jangkauan nilai yang dapat disimpan dan operasi-operasi yang dapat dijalankan terhadap *variable*.

E. Tipe data dalam Java

1. Tipe Data *Primitive*

Tabel II.1.

Tipe Data Primitif

Tipe data	Nilai Max-min	Nilai default	Penggunaan	Contoh
Byte	127 sampai -128	0	menghemat ruang array	byte a= 100, -50
Short	32,367 sampai -32,678	0	menghemat ruang array	short s=10000, -20000
Int	2,147,483,647 sampai -2,147,483,648	0	menampung nilai integral	int a= 10000L
Long	9,223,372,036,854,775,80 - 9,223,372,036,854,770,000	0L	menampung nilai yang lebih dari tipe data int	int a= 100000L int b= -200000L
Float	nilai decimal 32-bit presisi tunggal	0.0f	tipe data nilai decimal	float fl=234.5f
Double	nilai decimal 64-bit presisi ganda	0.0d	tipe data nilai decimal	double d1=123.4
Boolean	true(benar) atau False(salah)	False(salah)	penanda kondisi benar atau salah	Boolean satu= true
Char	ufff'- '\u000'	–	menampung karakter	char huruf A ='A'

Sumber: Enterprise (2014:19)

2. Tipe Data Referensi/Objek

Variabel referensi dibuat menggunakan konstruktor yang didefinisikan dalam kelas. *Variable* ini dideklarasikan untuk mengakses objek-objek.

F. Operator dalam Java

Operator dalam java yang biasa digunakan yaitu:

1. Operator Aritmatika

Tabel II.2

Operator Aritmatika

Operator	Deskripsi	Contoh
+	penjumlahan	A + B = 30
-	pengurangan	A - B = 10
*	perkalian	A * B = 200
/	pembagian	A / B = 2
%	modulus	A % B = 0

Sumber: Enterprise (2014:35)

2. Operator Relasional

Tabel II.3.

Operator Relasional

Operator	Deskripsi	contoh
==	Memeriksa apakah nilai kedua operan sama atau tidak. Jika sama maka kondisi bernilai benar	(A = = B) adalah tidak benar
!=	Memeriksa apakah nilai kedua operan sama atau tidak. Jika tidak sama maka kondisi bernilai benar	(A != B) adalah benar
>	Memeriksa apakah nilai operan disebelah kiri tidak lebih dari nilai operan disebelah kanan. Jika ya maka kondisi bernilai tidak benar	(A > B) adalah tidak benar

<	Memeriksa apakah nilai operan disebelah kiri kurang dari nilai operan disebelah kanan. Jika ya maka kondisi bernilai benar	(A < B) adalah benar
>=	Memeriksa apakah nilai operan disebelah kiri lebih dari atau sama dengan nilai operan disebelah kanan. Jika tidak maka kondisi tidak benar	(A >= B) adalah tidak benar
<=	Memeriksa apakah nilai operan disebelah kiri kurang dari atau sama dengan nilai operan disebelah kanan. Jika ya maka kondisi benar	(A >=B) adalah benar

Sumber: Enterprise (2014:37)

3. Operator Logika

Tabel II.4.

Operator Logika

Operator	Deskripsi	contoh
&&	Opeator logika AND. Jika kedua operan bukan nol, maka kondisi bernilai benar	(A && B) adalah tidak benar
||	Operator logika OR. Jika ada operan yang bukan nol, maka kondisi bernilai benar	(A || B) adalah benar
!	Operator logika NOT. Digunakan untuk membalik keadaan logika dari operan. Jika kondisi bernilai benar maka operator NOT akan membuatnya tidak benar	!(A && B) adalah benar

Sumber: Enterprise (2014:37)

G. Kontrol Perulangan dalam Java

Perulangan yang biasa digunakan dalam Java yaitu:

1. Perulangan *While*

   Perulangan *While* adalah struktur control yang memungkinkan untuk mengulangi suatu proses dengan jumlah perulangan tertentu.

   Berikut sintaks dari perulangan *while*:

```
While (ekspresi_boolean)
{
// Statemen
}
```

2. Perulangan *Do..While*

   Perulangan *do.. while* sama seperti perulangan *while*, tetapi perulangan *do.. while* pasti akan dieksekusi minimal satu kali.

   Berikut sintaksnya:

```
do
{
// Statemen
}
While(ekspresi_boolean);
```

3. Perulangan *For*

   Perulangan *for* adalah struktur *control repetitive* yang memungkinkan untuk menjalankan proses dengan jumlah perulngan tertentu (jumlah perulangan sudah diketahui sebelumnya).

Berikut sintaksnya:

```
for (inisialisasi; ekspresi_boolean; update)
{
// Statemen
}
```

2.1.5. Netbeans Environment

Menurut Komputer (2012:7) "Netbeans adalah salah satu IDE (*Integrated Development Environment*), yaitu sebuah lingkungan kerja yang digunakan untuk mengembangkan aplikasi dengan berbagai bahasa pemrograman khususnya Java".

Netbeans sendiri dikembangkan oleh pengembang Java, yaitu Sun Microsystem yang kini diakusisi oleh Oracle. Perkembangan Netbeans sangat cepat, kini telah keluar Netbeans terbaru yaitu Versi 8.0.2. Kebutuhan sistem yang dibutuhkan untuk melakukan instalasi dan menjalankan Netbeans 8.0.2 yaitu:

1. Sistem Operasi Microsoft Windows yang didukung seperti: XP professional SP3, Vista SP1, Windows 7 Professional
2. Prosesor: minimum menggunakan intel Pentium III 800MHZ
3. Memori: minimum 512 MB, rekomendasi 2GB
4. Hardisk Kosong Minimal 750 MB
5. Java Development Kit 7 atau lebih baru

2.1.6. Model Pengembangan Perangkat Lunak

Menurut Shalahuddin (2013:28) "Model SLDC air terjun (*water fall*) sering juga disebut model sekuensial linier (*sequential linear*) atau alur hidup klasik (*classic*

life). Model air terjun menyediakan pendekatan alur hidup perangkat lunak secara sekuensial atau terurut dimulai dari analisis, desain, pengodean, pengujian, dan tahap pendukung (support)".

Berikut adalah gambar model air terjun:

1. Analisis kebutuhan perangkat lunak

   Proses pengumpulan kebutuhan dilakukan secara intensif untuk menspesifikasikan kebutuhan perangkat lunak agar dapat dipahami perangkat lunak seperti apa yang dibutuhkan oleh *user*. Spesifikasi kebutuhan perangkat lunak pada tahap ini perlu untuk didokumentasikan.

2. Desain

   Desain perangkat lunak adalah proses multi langkah yang fokus pada desain pembuatan program perangkat lunak termasuk struktur data, arsitektur perangkat lunak, representasi antarmuka, dan prosedur pengodean. Tahap ini mentranslasi kebutuhan perangkat lunak dari tahap analisis kebutuhan ke representasi desain agar dapat diimplementasikan menjadi program pada tahap selanjutnya. Desain perangkat lunak yang dihasilkan pada tahap ini juga perlu didokumentasikan.

3. Pembuatan kode program

   Desain harus ditranslasikan ke dalam program perangkat lunak. Hasil dari tahap ini adalah program computer yang sesuai dengan desain yang telah dibuat pada tahap desain.

4. Pengujian

   Pengujian focus pada perangkat lunak secara dari segi lojik dan fungsional dan memastikan bahwa semua bagian sudah diuji. Hal ini dilakukan untuk

meminimalisir kesalahan (*error*) dan memastikan keluaran yang dihasilkan sesuai dengan yang diinginkan.

5. Pendukung (*support*) atau pemeliharaan (*maintaince*)

   Tidak menutup kemungkinan sebuah perangkat lunak mengalami perubahan ketika sudah dikirimkan ke *user*. Perubahan bisa terjadi karena adanya kesalahan yang muncul dan tidak terdeteksi saat pengujian atau perangkat lunak harus beradaptasi dengan lingkungan baru. Tahap pendukung atau pemeliharaan dapat mengulangi proses pengembangan mulai dari analisis spesifikasi untuk perubahan perangkat lunak yang sudah ada, tapi tidak untuk membuat perangkat lunak baru.

2.2 Teori Pendukung

2.2.1. Basis Data

Menurut Jogiyanto (2008:46) "Basis data (*data base*) adalah kumpulan dari data yang saling berhubungan datu dengan yang lainnnya, tersimpan di perangkat keras komputer dan digunakan perangkat lunak untuk memanipulasinya".

Dari definisi ini ada tiga hal yang berhubungan dengan basis data menurut Jogiyanto (2008:46), yaitu:

1. Data itu sendiri yang diorganisasikan dalam bentuk basis data (*database*).
2. Simpanan permanen (*storage*) untuk menyimpan basis data tersebut.
3. Perangkat lunak yang digunakan untuk memanipulasinya.

2.2.2. *Entity Relationship Diagram*

Menurut Yakub (2008:25)"*Entity Relationship Diagram* merupakan suatu model jaringan yang menggunakan susunan data yang tersimpan pada sistem secara abstrak". ERD juga menggambarkan hubungan antara satu entitas yang memiliki sejumlah atribut dengan entitas yang lain dalam suatu sistem yang terintegrasi. ERD digunakan oleh perancang sistem untuk memodelkan data yang nantinya akan dikembangkan menjadi basis data (*database*).

ERD terbagi atas tiga komponen, yaitu:

1. Entitas (*Entity*)

   Entitas menunjukan obyek-obyek dasar yang terkait didalam sistem. Obyek dasar itu bisa berupa orang, benda atau hal lain yang keterangannya perlu disimpan dalam basis data. Untuk menggambarkan entitas dilakukan dengan mengikuti aturan-aturan sebagai berikut:

   a. Entitas dinyatakan dengan *symbol* persegi panjang
   b. Nama entitas berupa kata benda tunggal
   c. Nama entitas sedapat mungkin menggunakan nama yang mudah dipahami dan menyatakan maknanya dengan jelas.

2. Atribut (*Attribute*)

   Atribut sering juga disebut dengan *property*, merupakan keterangan-keterangan yang terkait pada sebuah entitas yang perlu disimpan sebagai basis data. Atribut mengikuti aturan sebagai berikut:

   a. Atribut dinyatakan dengan simbol *elipps*.
   b. Nama atribut dituliskan dalam simbol *elipps*.

c. Nama atribut berupa kata benda tunggal.

d. Nama atribut sedapat mungkin menggunakan nama yang mudah dipahami dan dapat menyatakan maknanya dengan jelas.

e. Atribut dihubungkan dengan entitas yang bersesuaian dengan menggunakan garis.

3. Hubungan (*Relation*)

Hubungan atau relasi adalah kejadian atau transaksi yang terjadi diantar dua entitas yang keterangannya perlu disimpan dalam basis data. Aturan penggambaran relasi antar entitas adalah:

a. relasi dinyatakan dengan simbol belah ketupat

b. nama relasi dituliskan didalam simbol belah ketupat

c. relasi menghubungkan dua entitas

d. Nama relasi menggunakan kata kerja aktif (diawali awalan me tunggal)

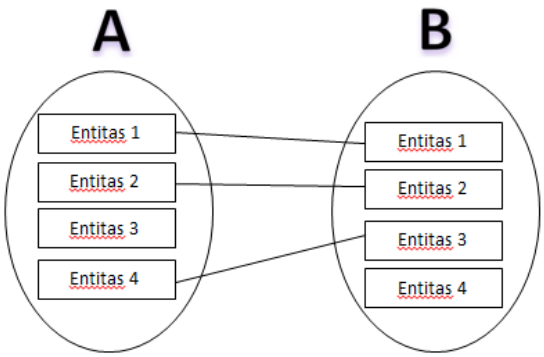
e. Nama relasi sedapat mungkin menggunakan nama yang mudah dipahami dan dapat menyatakan maknanya dengan jelas.

4. Derajat relasi (Kardinalitas)

Kardinalitas relasi menunjukan maksimum entitas yang dapat berelasi dengan entitas pada himpunan entitas yang lain. kardinalitas relasi yang terjadi bisa berupa satu ke satu (*one to one*), satu ke banyak (*one to many*), dan banyak ke banyak (*many to many*).

a. Satu ke satu (*one to one*)

Berarti setiap entitas paling banyak berhubungan dengan satu entitas yang lain.



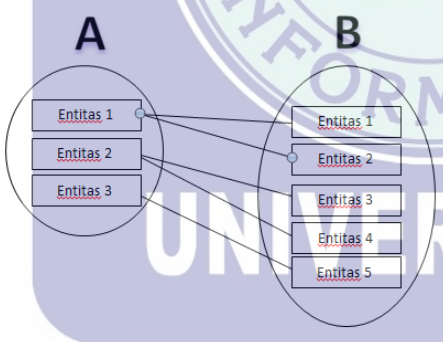
Sumber: Fathansyah (2012:79)

Gambar II.1.

Kardinalitas *One to One*

b. Satu ke banyak (*one to many*)

Berarti setiap entitas dapat berhubungan dengan banyak entitas yang lain, tetapi tidak sebaliknya.



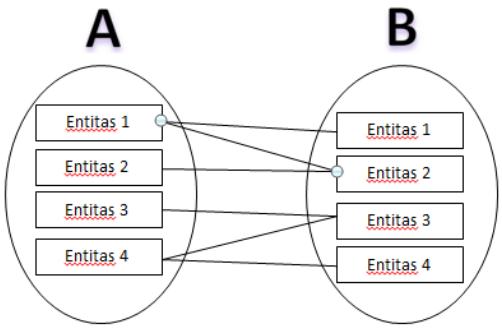
Sumber: Fathansyah (2012:80)

Gambar II.2.

Kardinalitas *One to Many*

c. Banyak ke banyak (*many to many*)

Berarti setiap entitas dapat berhubungan dengan banyak entitas yang lainnya, demikian sebaliknya.



Sumber: Fathansyah (2012:81)

Gambar II.3.

Kardinalitas *Many to Many*

2.2.3. Unitied Modeling Language

Menurut Herlawati (2011:6) "UML singkatan dari *Unitied Modeling Language* yang berarti bahasa pemodelan standar".

UML diaplikasikan untuk maksud tertentu menurut Herlawati (2011:6) antara lain untuk:

1. Merancang perangkat lunak
2. Sarana komunikasi antara perangkat lunak dengan proses bisnis.
3. Menjabarkan sistem secara rinci untuk analisa dan mencari apa yang diperlukan sistem.
4. Mendokumentasikan sistem yang ada, proses-proses dan organisasinya.

Blok pembangun utama UML adalah diagram. Menurut Herlawati (2011:7) ada sembilan diagram, antara lain:

A. Diagram Kelas

Diagram ini memperlihatkan himpunan kelas-kelas, antarmuka-antarmuka, kolaborasi-kolaborasi, serta relasi-relasi.

Menurut Herlawati (2011:39) “Diagram kelas merupakan kumpulan dari kelas-kelas objek”.

Fitur-fitur yang ada pada diagram kelas menurut Herlawati (2011:38) adalah sebagai berikut:

1. Atribut (*Attribute*)

   Atribut adalah rincian suatu kelas misalnya warna mobil, jumlah sisi suatu bentuk, dan sebagainya. Digambarkan dalam dua notasi yang berbeda. *Inline* atau hubungan antar kelas.

2. Operasi (Operation)

   Operasi sebagai fitur dari kelas yang menspesifikasikan bagaimana perilaku dijalankan. Notasi operasi diletakan dalam kompartemen terpisah dengan sintaks-nya:

   Visibility name (parameters) : return-type {properties}

3. Batasan Operasi

   Batasan adalah kontrak yang harus dipatuhi oleh operasi. Kondisi awal dinyatakan pada tiap parameter. Kondisi akhir bermaksud memberi jaminan terhadap kondisi sistem setelah operasi dilaksanakan.

4. Metode (*methods*)

Metode adalah implementasi proses. Jika suatu kelas tidak memiliki implementasi operasi, maka harus menurunkan (*inheritance*) dari suatu kelasnya, dan operasinya dinyatakan sebagai *abstract*.

5. Kelas Abstrak

   Kelas abstrak adalah kelas yang menyediakan operasi merinci implementasinya. Kelas abstrak bermanfaat untuk mengidentifikasikan fungsi antar objek.

6. Hubungan (*Relationships*)

   Hubungan antar kelas memiliki tipe yang berbeda dan sesuai kebutuhan.

   a. Ketergantungan (*dependency*)
   b. Asosiasi
   c. Agregasi
   d. Komposisi
   e. Generalisasi

B. Diagram Paket (*Package Diagram*)

Diagram ini memperlihatkan kumpulan kelas-kelas, merupakan bagian dari diagram komponen.

C. Diagram *Use-Case*

Diagram ini memperlihatkan *Use-case* dan aktor-aktor. Digunakan untuk memodelkan perilaku suatu sistem yang dibutuhkan serta diharapkan pengguna.

Menurut Pooley dalam Herlawati (2011:16) "*Use Case* menggambarkan *external view* dari sistem yang akan kita buat modelnya. Bahwa model *use case*

dapat dijabarkan dalam diagram *use case*, tetapi yang harus diingat, diagram tidak identik dengan model karena model lebih luas dari diagram". Komponen pembentuk diagram *use case* adalah:

1. Aktor (*actor*), menggambarkan pihak-pihak yang berperan dalam sistem. Simbol aktor adalah gambar orang. Ada empat macam tipe aktor:
2. *Use Case*, aktifitas atau sarana yang disiapkan oleh bisnis atau sistem.

Menurut Pilone dalam Herlawati (2011:21) "*Use case* menggambarkan fungsi tertentu dalam suatu sistem berupa komponen, kejadian atau kelas".

Cara membuat *use case* yang baik menurut Chonoles dalam Herlawati (2011:22), yaitu:

a. Pilihlah nama yang baik
b. Ilustrasikan perilaku yang lengkap
c. Identifikasi perilaku dengan lengkap
d. Menyediakan *use case* lawan (*inverse*)
e. Batasi *use case* hingga satu perilaku saja
f. Nyatakan *use case* dari sudut pandang aktor

3. Hubungan (*link*), aktor mana saja yang terlibat dalam *use case* ini.

Hubungan dalam *use case diagram* biasa disebut relasi. Relasi digambarkan sebagai sebuah garis antar dua simbol. Relasi yang digunakan UML 2.0 adalah sebagai berikut:

a. Generalisasi
b. Ekstensi
c. Inklusi

Aturan penulisan UML untuk diagram *use case* yang disarankan oleh Universitas Cambridge dalam Herlawati (2011:32) yaitu:

1. Panduan pembuatan aktor
   a. Tempatkan aktor utama di pojok kiri atas
   b. Gambarkan aktor terpisah dari *use case*
   c. Berilah nama aktor dengan kata benda tunggal
   d. Aktor minimal harus terhubung dengan satu *use case*
   e. Berilah nama aktor sesuai dengan perannya terhadap model bukan jabatannya.
   f. Tambahkan <<*system*>> pada aktor yang berjenis sistem
   g. Jangan menghubungkan langsung antar aktor (kecuali generalisasi), aktor harus terhubung lewat *use case*.
   h. Tambahkan aktor "waktu (*time*)" untuk sistem yang terjadwal otomatis.
2. Panduan pembuatan *use case*
   a. Buatlah nama *use case* sejelas mungkin
   b. Susunlah *use case* berdasarkan urutannya dari atas ke bawah.
3. Panduan pembuatan relasi
   a. Hindari penggunaan anak panah antara aktor dan *use case* kecuali salah satunya bersifat pasif.
   b. Gunakan <<*include*>> jika yakin sebuah *use case* harus menggunakan *use case* lain.
   c. Gunakan <<*extend*>> jika suatu *use case* mungkin melibatkan *use case* lain.
   d. Gunakan <<*extend*>> seperlunya
   e. Gunakan kata *include* dan *extend* bukan *includes* atau *extends*

f. Tempatkan *include use case* di sebelah kanan *use case* dasar.

g. Tempatkan *extend use case* dibawah *use case* dasar.

h. Tempatkan generalisasi *use case* dibawah use case induk.

i. Tempatkan generalisasi aktor dibawah actor induk.

j. Hindari pembuatan *use case* lebih dari dua tingkatan.

D. Diagram interaksi dan *Sequence* (urutan)

Diagram interaksi yang menekankan pada pengiriman pesan dalam suatu waktu tertentu.

Menurut Pilone dalam Herlawati (2011:174) ”Diagram interaksi dimaksudkan untuk mngembangkan komunikasi antar objek, bukan manipulasi data saat berkomunikasi”.

Dalam penggambaran interaksi atau sequence diagram ada beberapa komponen penyusunnya, yaitu:

1. Anggota interaksi
2. Pesan (*messages*)
3. Pelaksanaan eksekusi (*Excecution Occurrences*)
4. *State Invariants*
5. Pelaksanaan peristiwa (*Event Occurences*)
6. Kombinasi *fragment* (*Combined Fragments*)
7. Dekomposisi (*decomposition*)
8. Lanjutan (*Continuation*)
9. Kolaborasi (*collaboration*)

10. Diagram komunikasi (*communication diagram*)
11. Diagram global interaksi (*interaction overview diagram*)
12. Diagram pewaktuan (*timing diagram*)

F. Diagram Komunikasi (*Communication Diagram*)

Diagram yang menekankan organisasi truktural dari objek-objek yang menerima serta mengirim pesan.

G. Diagram *Statechart* (*Statechart Diagram*)

Diagram status yng memperlihatkan keadaan-keadaan pada sistem, memuat status (*state*), transisi, kejadian serta aktifitas.

H. Diagram Aktivitas (*Activity Diagram*)

Diagram tipe khusus dari diagram status yang memperlihatkan aliran dari suatu aktivitas ke aktivitas lainnya dalam suatu sistem.

I. Diagram Komponen (*Component Diagram*)

Diagram yang memperlihatkan organisasi atau kebergantungan sistem atau perangkat lunak pada komponen-komponen yang telah ada sebelumnya. Diagram ini berhubungan dengan diagram kelas dimana komponen secara tipikal dipetakan ke dalam satu atau lebih kelas-kelas, antarmuka-antarmuka serta kolaborasi-kolaborasi.

J. Diagram *Deployment (Deployment Diagram*)

Diagram ini memperlihatkan konfigurasi pada saat aplikasi dijalankan (*run time*). Diagram ini sangat erat hubungannya dengan diagram komponen dimana diagram ini memuat satu atau lebih komponen-komponen.

2.2.4. Black Box Testing

Klasifikasi *black box testing* mencangkup beberapa pengujian menurut Simarmata (2010:316), yaitu:

1. Pengujian fungsional (*fungsional testing*)

   Perangkat lunak diuji dengan persyaratan fungsional dalam bentuk tertulis untuk memeriksa apakah aplikasi sudah berjalan sesuai dengan yang diharapkan.

2. Pengujian tegangan (*stress testing*)

   Pengujian kualitas aplikasi dengan lingkungan yang lebih menuntut aplikasi, tidak seperti saat aplikasi dijalankan pada beban kerja normal.

3. Pengujian beban (*load testing*)

   Pengujian dengan memasukan beban berat pada sistem untuk mengetahui apakah aplikasi gagal atau kinerjanya menurun atau tetap berfungsi dengan baik.

4. Pengujian khusus (*ad-hoc testing*)

   Pengujian yang dilakukan tanpa rencana pengujian. Pengujian ini untuk mempelajari aplikasi sebelum memulai pengujian dengan pengujian yang lainnya.

5. Pengujian penyelidikan (*exploratory testing*)

   Pengujian ini mirip dengan pengujian khusus yaitu untuk mempelajari aplikasi.

6. Pengujian usabilitas (*usability testing*)

   Pengujian bagaimana proses kerja antamuka aplikasi dengan pengguna secara langsung maupun tidak langsung untuk menilai bagaimana mereka berinteraksi dengan aplikasi. Tujuannya untuk membatasi atau menghilangkan kesulitan bagi penggguna.

7. Pengujian asap (*smoke testing*)

   Pengujian kenormalan dilakukan untuk memeriksa apakah aplikasi sudah siap untuk pengujian yang lebih besar dan bekerja dengan baik tanpa cela sampai tingkat yang paling diharapkan.

8. Pengujian pemulihan (*recover testing*)

   Pengujian yang pada dasarnya dilakukan untuk memeriksa seberapa baik dan cepatnya aplikasi bisa pulih terhadap semua *crash* atau kegagalan *hardware*, masalah bencana dan lain-lain.

9. Pengujian volume (*volume testing*)

   Pengujian dengan memproses data dalam jumlah yang besar ke dalam aplikasi (yang sedang diuji) untuk memeriksa keterbatasan ekstrem dari sistem.

10. Pengujian domain (*domain testing*)

    Pengujian dengan mengambil variabel individu dengan membaginya lagi kedalam subset (dalam berbagai cara) yang sama. Kemudian menguji perwakilan dari masing-masing subset.

11. Pengujian skenario (*scenario testing*)

    Pengujian yang *realistis*, *kredibel* dan memotivasi *stakeholder*, tantangan untuk program dan mempermudah penguji untuk melakukan evaluasi.

12. Pengujian regresi (*regression testing*)

    Pengujian yang berfokus pada pengujian ulang setelah ada perubahan pada daerah yang sama dengan pengujian yang berbeda.

13. Penerimaan pengguna (*user testing*)

    Perangkat lunak akan digunakan oleh pengguna untuk mengetahui apakah perangkat tersebut memenuhi harapan pengguna dan bekerja seperti yang diharapkan pengembang.

14. Pengujian alfa (*alpha testing*)

    Pengguna akan menggunakan aplikasi dan pengembang mencatat setiap masukan atau tindakan yang dilakukan oleh pengguna. Semua perilaku yang tidak normal sistem dicatat dan dikoreksi oleh para pengembang.

15. Pengujian beta (*beta testing*)

    Pengujian beta dilakukan setelah pengujian alfa. Perangkat lunak dilepaskan ke kelompok masyarakat agar dapat memastikan bahwa perangkat tersebut memiliki kesalahan atau *bug*.