BAB II
TINJAUAN PUSTAKA

2.1 Sistem Informasi

Pada teori mengenai sistem informasi, Elizabeth Hardcastle menjelesakan pentingnya membedakan antara data dan informasi dalam bukunya (Hardcastle, 2008). Data adalah suatu fakta dasar yang bisa dalam bentuk angka atau pernyataan. Data bisa diperoleh dari suatu proses pengukuran. Sedangkan informasi adalah data yang telah diproses sehingga menjadi sesuatu yang bermakna.

Sebuah sistem dapat didefinisikan sebagai kumpulan komponen yang bekerja sama menuju tujuan bersama. Tujuan dari sistem adalah untuk menerima masukan dan mengubahnya menjadi *output*. Hal ini dapat dilihat bahwa dalam sistem, data digunakan sebagai *input* untuk proses menciptakan informasi sebagai *output* (Hardcastle, 2008).

Sistem Informasi memiliki definisi suatu sistem terintegrasi yang mampu menyediakan informasi yang bermanfaat bagi penggunanya. Dalam sistem informasi terjadi suatu pengolahan data dan informasi yang diorganisir oleh suatu sistem (Hardcastle, 2008).

2.2 Daftar Hadir

Menurut Kamus Besar Bahasa Indonesia, daftar adalah catatan sejumlah nama atau hal yang disusun berderet dari atas ke bawah. Sedangkan hadir adalah ada, datang. Jadi dapat disimpulkan daftar hadir adalah catatan yang menyatakan kehadiran seseorang pada setiap hari belajar, bekerja dan sebagainya.

2.3 Penelitian Terdahulu

Dalam penelitian ini penulis memaparkan dua penelitian terdahulu yang relevan dengan permasalahan yang akan diteliti tentang sistem pencatatan kehadiran, atau yang lebih dikenal sebagai sistem absensi.

Penelitian pertama oleh Dodik Gunawan (2006) dalam thesisnya memaparkan bahwa tujuan dari penelitiannya adalah membuat sistem pencatatan kehadiran yang baru pada lingkungan STT Telkom, yaitu pencatatan kehadiran menggunakan sidik jari untuk mengurangi bahkan menghilangkan manipulasi data kehadiran mahasiswa tersebut. Metodologi yang digunakan dalam penelitan tersebut adalah studi literatur tentang metode otentikasi sidik jari, pengumpulan data dan survei untuk memperoleh data yang diperlukan, dan pengembangan sistem menggunakan metode pengembangan perangkat lunak dengan 4 tahapan yaitu Perencanaan, Analisis dan Desain, Implementasi, dan Pengujian.

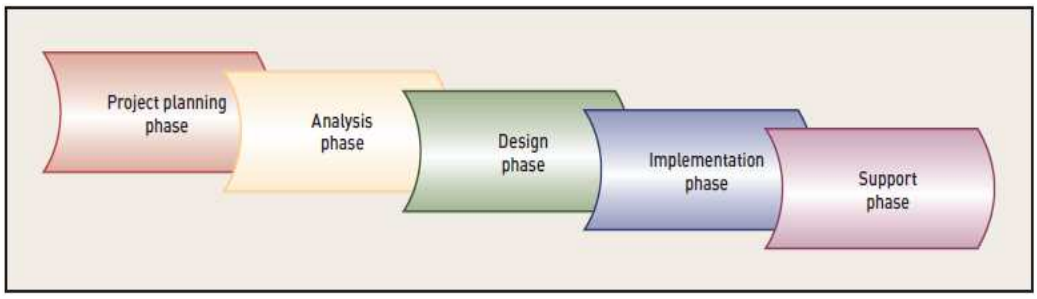
Penelitian kedua oleh Atin Triwahyuni (2012) dalam jurnalnya memaparkan tujuan penelitiannya adalah membuat aplikasi *client*/*server* untuk mengolah data absensi siswa yang dapat dijalankan dalam sebuah jaringan dan dapat diakses oleh seluruh penggunanya sesuai dengan *level* otoritas masing-masing. Metode yang digunakan meliputi wawancara, pengamatan, dan riset pustaka. Sistem yang dibangun menggunakan arsitektur *client*/*server* dengan menggunakan arsitektur 2 *tier* dan menggunakan aplikasi berbasis *desktop*. Aplikasi ini dijalankan menggunakan sebuah komputer server dengan nomor *IP* yang *disetup* dengan *IP Private* sehingga hanya dapat dipergunakan dalam jaringan lokal/*intranet* di lingkungan sekolah.

Cara penggunaan sistemnya ada 2, yaitu dengan model klasikal dan model absensi *barcode*. Model klasikal yaitu sistem menampilkan daftar kelas yang aktif pada periode pembelajaran yang berjalan, kemudian memilih salah satu kelas, lalu sistem akan menampilkan data siswa pada kelas tersebut. Guru sebagai *user* akan melakukan pencatatan kehadiran, sesuai dengan siswa yang hadir pada hari tersebut. Model absensi *barcode* adalah model *entry* absensi menggunakan *barcode scanner* dengan melakukan *scanning* pada kartu siswa, maka siswa yang bersangkutan akan di-*entry* dengan memasukkan Nomor Induk Siswa (NIS) yang tertera dalam kartu siswa.

2.4 Metodologi Pengembangan Sistem

Menurut Satzinger (Satzinger, et.al., 2007) *Sistem Development Life Cycle* (SDLC) atau Siklus Hidup Pengembangan Sistem adalah kerangka yang menggambarkan kegiatan yang dilakukan pada setiap tahap proyek pengembangan perangkat lunak.

SDLC terdiri dari 5 fase dimana masing-masing fase terdiri dari aktivitas yang saling terkait/berhubungan. Fase perencanaan (*project planning phase*), analisa (*analysis phase*), desain (*design phase*), dan implementasi (*implementation phase*) disebut sebagai fase utama, fase ini adalah unsur-unsur yang menyediakan kerangka kerja untuk mengelola proyek. Fase pendukung, disebut sebagai fase tambahan, termasuk kegiatan yang diperlukan untuk meningkatkan dan memelihara sistem setelah sistem tersebut digunakan. Fase pendukung (*suppot phase*) merupakan bagian dari *framework* SDLC, tetapi biasanya tidak dianggap sebagai bagian dari proyek pengembangan awal. Gambar 2.1 mengilustrasikan lima fase dari *framework* SDLC.



Gambar 2.1. *Framework* SDLC
(Sumber : Satzinger, et.al., 2007)

2.4.1 Tahapan Pengembangan Sistem Dengan Metode *Framework* SDLC

Sub bab 2.5 sebagian besar diambil dari buku karangan Satzinger, et.al. (Satzinger, et.al., 2007).

2.4.1.1 Tahap Perencanaan (*Project Planning Phase*)

Tahap perencanaan merupakan tahap awal dari pengembangan sistem, hal-hal yang dilakukan pada tahap ini diantaranya adalah:

1. mendefinisikan masalah;
2. mengkonfirmasikan kelayakan proyek;
3. membuat jadwal proyek;
4. menentukan staff yang terlibat dalam proyek; dan
5. memulai proses pengembangan proyek.

2.4.1.2 Tahap Analisa (*Analysis Phase*)

Tahap analisa bertujuan untuk memahami dan mendokumentasikan secara rinci kebutuhan sistem dan persyaratan pengolahan sistem yang baru. Pada tahap analisa, hal-hal yang dilakukan diantaranya adalah:

1. mengumpulkan informasi;
2. mendefinisikan kebutuhan-kebutuhan sistem;
3. membangun *prototype* yang sesuai atau memenuhi kebutuhan sistem;
4. menentukan prioritas kebutuhan sistem;

5. membuat *prototype* atas prioritas dan melakukan evaluasi terhadap alternative yang dipilih; dan
6. me-*review* rekomendasi terhadap pihak manajemen.

2.4.1.3 Tahap Desain (*Design Phase*)

Tahap desain atau perancangan sistem dilakukan untuk merancang solusi sistem berdasarkan persyaratan yang ditetapkan dan keputusan yang dibuat selama analisis. Pada tahap desain, hal-hal yang dilakukan diantaranya sebagai berikut.

1. Desain Level Tinggi (Arsitektur Sistem), yaitu.
   - Desain dan integrasi jaringan.
   - Desain arsitektur aplikasi.
2. Desain Level Rendah, yaitu.
   - Desain *user interface.*
   - Desain sistem *interface.*
   - Desain dan integrasi database.
   - Prototype desain secara lengkap.
   - Desain dan integrasi pengawasan sistem.

2.4.1.4 Tahap Implementasi (*Implementation Phase*)

Tahap implementasi atau penerapan merupakan kegiatan untuk membangun, menguji, dan menginstal sistem informasi yang handal dengan pengguna yang terlatih serta siap untuk mendapatkan keuntungan seperti yang diharapkan dari penggunaan sistem. Pada

tahap implementasi, hal-hal yang dilakukan diantaranya sebagai berikut:

1. membangun komponen-komponen perangkat lunak;
2. melakukan verifikasi dan pengujian;
3. mengkonversi data;
4. melakukan training user dan mendokumentasikan sistem; dan
5. meng-*install* sistem.

2.4.1.5 Tahap Pendukung (*Support Phase*)

Tahap Pendukung bertujuan untuk menjaga sistem berjalan secara produktif, serta mendukung pengguna dalam menggunakan sistem tersebut dari awal hingga selama sistem tersebut masi digunakan.

2.5 *Unified Modeling Language* (UML)

UML (Unified Modeling Language) menurut Booch, *et al*., 1998 adalah sebuah bahasa yang berdasarkan grafik atau gambar untuk memvisualisasi, menspesifikasikan, membangun, dan pendokumentasian dari sebuah sistem pengembangan *software* berbasis OO (Object-Oriented). UML sendiri juga memberikan standar penulisan sebuah sistem *blue print*, yang meliputi konsep bisnis proses, penulisan kelas-kelas dalam bahasa program yang spesifik, skema database, dan komponen-komponen yang diperlukan dalam sistem software.

Pada penelitian dan pengembangan aplikasi sistem pencatatan kehadiran dengan pembatasan area *login* berbasis *web*, tipe UML yang digunakan adalah *use case* diagram, *activity* diagram, *sequence* diagram dan *class* diagram.

2.5.1 *Use case* Diagram

Menurut Whitten dan Bentley (2007), *Use case Diagram* dipakai untuk menggambarkan relasi antara sistem dan sistem eksternal dan *user,* dengan kasus yang disesuaikan dengan langkah-langkah yang telah ditentukan. *Use case Diagram* merupakan cara /metode yang cocok untuk digunakan untuk dapat menggambarkan interaksi yang jelas antara sistem dengan pengguna.

1. *Use cases*

Use case mendeskripsikan fungsi dari sebuah sistem dilihat dari sudut pandang pengguna.

Gambar 2.2. *Use cases*
(Sumber: Whitten dan Bentley, 2007)

2. *Actors*

Actors merupakan sesuatu yang berinteraksi dengan sistem untuk saling bertukar informasi. *Actors* tidak harus berupa manusia, tetapi dapat berupa suatu organisasi atau sistem informasi.

Gambar 2.3 *Actors*

(Sumber: Whitten dan Bentley, 2007)

3. *Relationships*

Sebuah relasi antar sistem dan sistem atau *user* dan sistem digambarkan dengan sebuah garis di antara keduanya. Arti relasi yang digambarkan bisa beragam tergantung pada bagaimana garis itu digambarkan dan apa yang mereka hubungkan. Ada beberapa macam relasi, antara lain *associations*, *extends*, dan *uses.*

a) *Associations*

Associations adalah sebuah relasi antara seorang *actor* dengan sebuah *use case* di mana terjadi interaksi antar mereka. Asosiasi dengan panah tertutup (1) di ujung yang menyentuh *use case* mengindikasikan bahwa *actor* di ujung yang satu lagi melakukan *use case* tersebut. Sedangkan asosiasi tanpa panah(2) mengindikasikan sebuah interaksi dari *use case* ke *actor* yang menerima hasil dari *use case* tersebut.

Gambar 2.4 *Associations* dalam *Use case* Diagram
(Sumber: Whitten dan Bentley, 2007)

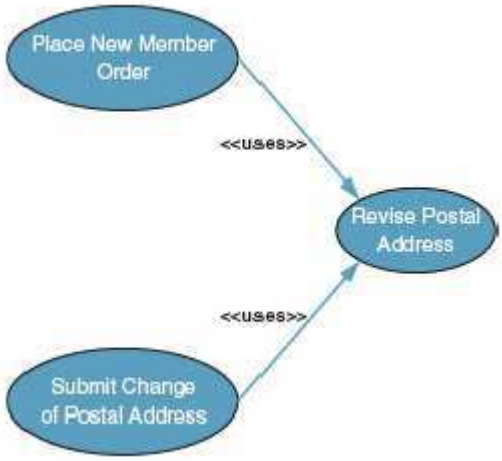
b) *Extends*

Extends perluasan dari *use case* lain jika kondisi atau syarat terpenuhi. Kurangi penggunaan *association Extend* ini, terlalu banyak pemakaian *association* ini membuat diagram sulit dipahami. Tanda panah terbuka harus terarah ke parent/base *use case*.

Gambar 2.5 *Extends* dalam *Use case* Diagram
(Sumber: Whitten dan Bentley, 2007)

c) *Uses (or Include)*

Uses (or Include) termasuk didalam *use case* lain (*required*) / (diharuskan). *Uses (or Include)* yaitu kelakuan yang harus terpenuhi agar sebuah *event* dapat terjadi, dimana pada kondisi ini sebuah *use case* adalah bagian dari *use case* lainnya.



Gambar 2.6 *Uses* dalam *Use case* Diagram
(Sumber: Whitten dan Bentley, 2007)

2.5.2 *Activity Diagram*

Menurut Whitten dan Bentley (2007). *Activity Diagram* merupakan gambaran dari alur yang berurutan dari aktivitas *use case* atau proses bisnis. *Activity Diagram* juga bisa dipakai untuk memodelkan berbagai aksi yang dilakukan saat sebuah operasi dieksekusi, dan memodelkan hasil dari aksi tersebut. Dari diagram ini, kita dapat melihat bagaimana aktivitas dalam suatu sistem, dari mulai hingga saat sistem berakhir.

Activity diagram dibentuk oleh beberapa notasi, antara lain *initial node, actions, flow, decision, merge, fork, join,* dan *activity final*, dan terkadang digunakan *swimlane* untuk mempartisi aksi yang terjadi berdasarkan pelaku.

1. *Initial node*

Initial node berupa lingkaran penuh yang menggambarkan titik mulai suatu proses.

Gambar 2.7 *Initial Node*
(Sumber: Whitten dan Bentley, 2007)

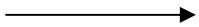
2. *Actions*

Actions adalah notasi segi empat bersudut tumpul yang menggambarkan langkah-langkah akivitas *sistem* yang terjadi.

Gambar 2.8 *Actions*
(Sumber: Whitten dan Bentley, 2007)

3. *Flow*

Flow (alur) merupakan panah dalam diagram yang mengindikasikan alur antar *actions.*

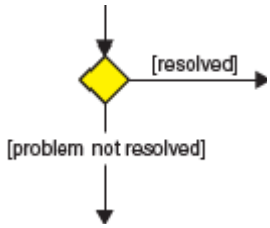


Gambar 2.9 *Flow*

(Sumber: Whitten dan Bentley, 2007)

4. *Decision*

Decision memiliki bentuk seperti wajik dengan satu alur masuk dan dua atau lebih alur keluar, alur keluar ditentukan dengan kondisi tertentu.

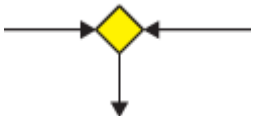


Gambar 2.10 *Decision*

(Sumber: Whitten dan Bentley, 2007)

5. *Merge*

Merge adalah wajik dengan dua atau lebih alur masuk dan satu alur keluar untuk menggabungkan alur yang sebelumnya terpisah oleh *decision.*



Gambar 2.11 *Merge*

(Sumber: Whitten dan Bentley, 2007)

6. *Fork*

Fork adalah bar hitam dengan satu alur masuk dan dua atau lebih alur keluar, aksi di bawah percabangan dapat terjadi dalam urutan apapun atau bahkan secara bersamaan.

Gambar 2.12 *Fork*

(Sumber: Whitten dan Bentley, 2007)

7. *Join*

Join adalah bar hitam dengan dua atau lebih alur masuk dan satu alur keluar untuk menyatukan lagi alur aksi yang dipisahkan oleh *fork.*

Gambar 2.13 *Join*

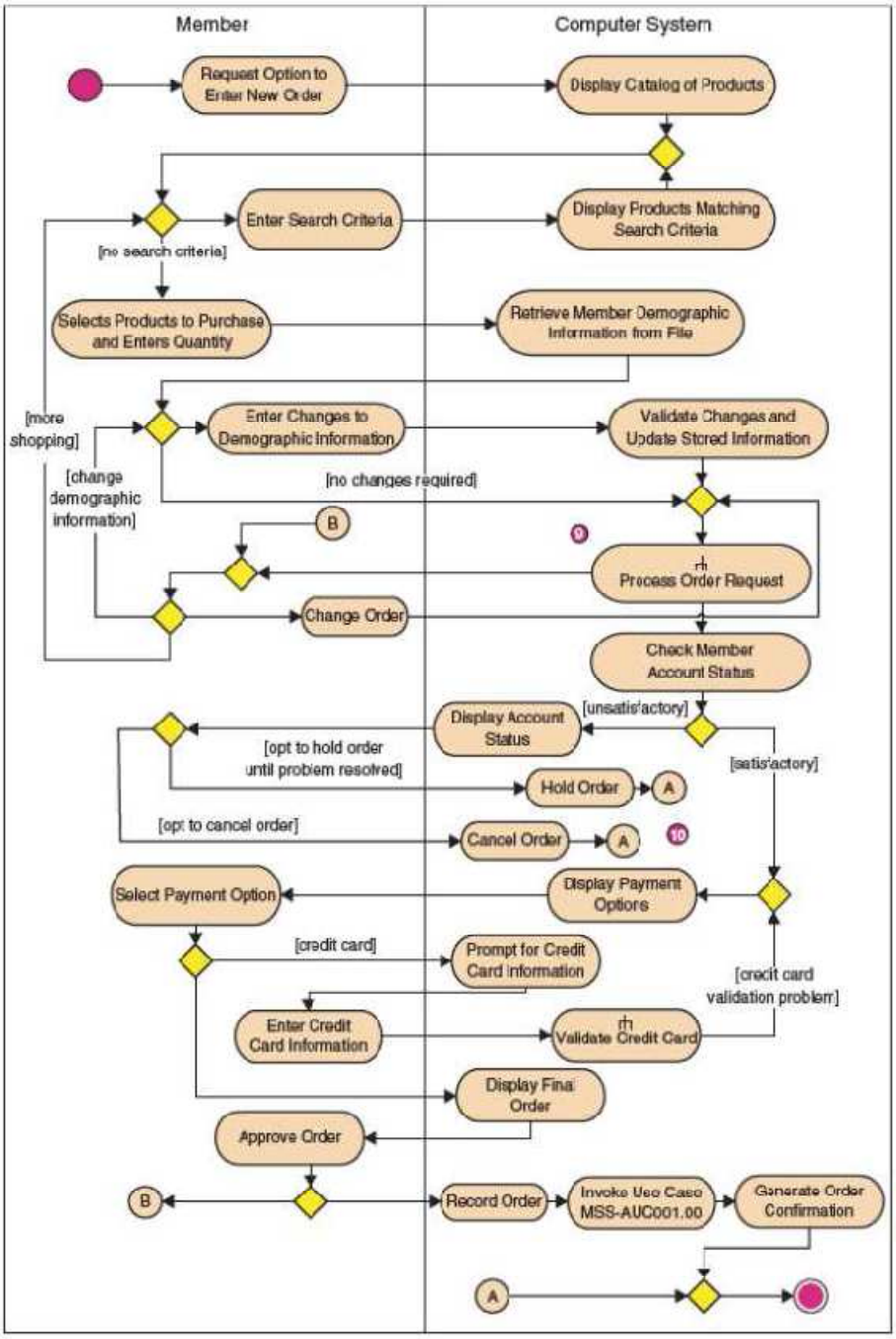
(Sumber: Whitten dan Bentley, 2007)

8. *Activity Final*

Activity final berbentuk lingkaran penuh dengan satu lingkaran di luarnya untuk menggambarkan titik akhir proses.

Gambar 2.14 *Activity Final*

(Sumber: Whitten dan Bentley, 2007)

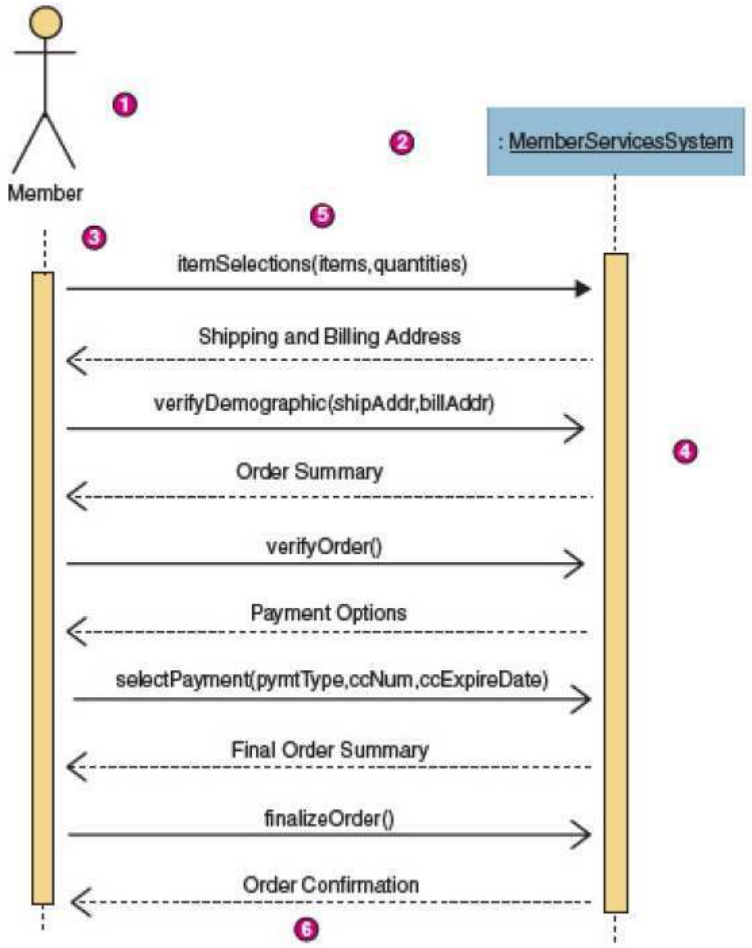


Gambar 2.15 *Activity Diagram*

(Sumber: Whitten dan Bentley, 2007)

2.5.3 *Sequence Diagram*

Menurut Whitten dan Bentley (2007), secara grafikal, *Sequence Diagram* merupakan diagram yang menggambarkan bagaimana objek berinteraksi satu sama lain melalui pesan dalam eksekusi *use case* atau operasi. Diagram ini mengilustrasikan bagaimana pesan dikirim dan diterima antara objek dan urutan yang seperti apa. Diagram ini lebih detail dalam penggambaran aliran data, termasuk data yang dikirim ataupun diterima.



Gambar 2.16 *Sequence Diagram*

(Sumber: Whitten dan Bentley, 2007)

Sebuah *sequence diagram* terbentuk dari beberapa ntasi, antara lain *actor, sistem, lifelines, activation bars, input message,* dan *output message.*

1. Actor

Actor, digambarkan dengan simbol *actor* pada *usecase.*

2. Sistem

Sistem, sebuah kotak digunakan untuk menggambarkan sistem yang bersangkutan.

3. Lifelines

Lifelines, garis vertikal putus-putus yang mengindikasikan masa hidup sistem/aktor.

4. Activation bars

Activation bars, balok panjang yang diletakkan di atas *lifelines* untuk menggambarkan masa waktu terjadinya interaksi aktif.

5. Input message

Input message, garis horizontal dengan panah ke kanan yang mengindikasikan pesan masuk.

6. Output message

Output message, garis horizontal dengan panah ke kiri yang mengindikasikan pesan balik.

2.5.4 Class Diagram

Menurut Booch (2005), *class diagram* menunjukan sekumpulan kelas, antarmuka, dan kerjasama serta hubungannya. *Class diagram* digunakan untuk memodelkan perancangan statik dari

gambaran sistem. Biasanya meliputi permodelan *vocabulary* dari sistem, permodelan kerjasama, atau permodelan skema.

Class diagram dapat digunakan untuk membangun sistem yang dapat dieksekusi melalui teknik *forward and reverse*, selain untuk penggambaran, penspesifikasian, dan pendokumentasian struktur model.

Class Diagram terdiri dari.

a) Nama *Class*.

b) Atribut.

c) Operasi/Method.

Tabel 2.1 *Class Diagram* (Wahono, R.S, 2003)

Nama *Class*.
Atribut;
Method;

Atribut dan Operasi/*method* dapat memiliki tiga sifat berikut.

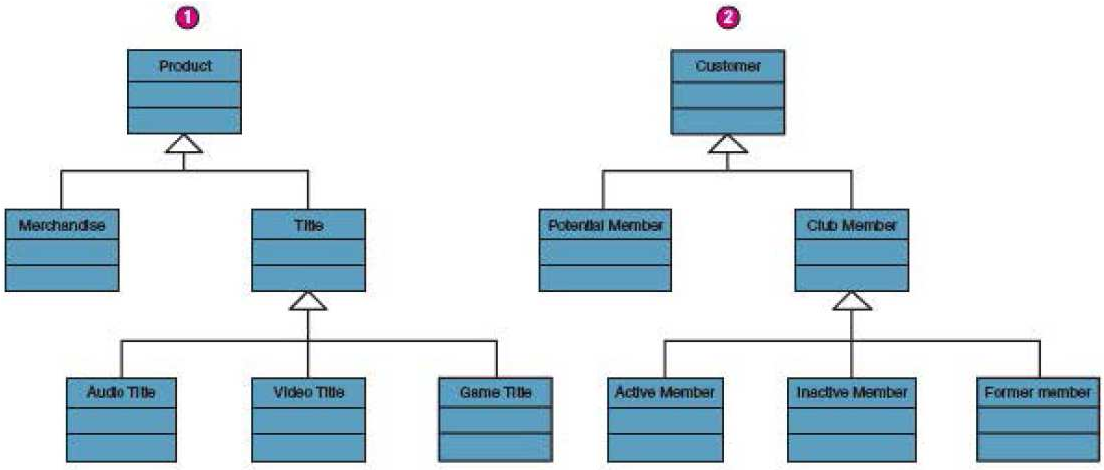
- *Public*, dapat dipanggil oleh *class* apa saja.
- *Protected*, hanya dapat dipanggil atau diakses oleh *class* yang bersangkutan dan *class* turunannya.
- *Private*, hanya dapat dipanggil oleh dirinya sendiri (tidak dapat diakses dari luar *class* yang bersangkutan).

Hubungan antar *class.*

1. Asosiasi, yaitu hubungan yang bersifat statis dalam *class.* Asosiasi menggambarkan *class* yang memiliki atribut berupa *class* lain atau *class* yang harus mengenal adanya *class* lain.
2. Agregasi, merupakan hubungan antara satu *object* dengan *object* lainnya dimana *object* satu dengan *object* lainnya sebenarnya terpisah namun disatukan, sehingga tidak terjadi kebergantungan (*Object* lain bisa ada walau *object* penampungnya tidak ada).
3. Pewarisan, yaitu hubungan hirarki antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metode *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
4. Hubungan dinamis, yaitu rangkaian pesan (*messaging*) uang di-*passing* dari satu *class* kepada *class* lain.

Notasi hubungan antara kelas terbagi atas.

1. *Exactly 1* (Tepat 1).
2. *Zero or* 1 (0..1).
3. *Zero or more* (0..* atau *).
4. *1 or more* (1..*).
5. *Specific range* (?..?).



Gambar 2.17 *Class Diagram*

(Sumber: Whitten dan Bentley, 2007)

2.6 *Black-Box Testing*

1. Menurut Myers (2004).
   - Proses menjalankan program dengan maksud menemukan kesalahan.

2. Menurut IEEE (1990) .
   - Pengujian yang mengabaikan mekanisme internal sistem atau komponen dan fokus semata-mata pada output yang dihasilkan yang merespon input yang dipilih dan kondisi eksekusi.
   - Pengujian yang dilakukan untuk mengevaluasi pemenuhan sistem atau komponen dengan kebutuhan fungsional tertentu.

Metode pengujian perangkat lunak Black Box digunakan untuk menguji fungsi-fungsi khusus dari perangkat lunak yang dirancang. Kebenaran pengujian dilihat dari keluaran yang dihasilkan dari data atau kondisi masukan yang diberikan untuk fungsi yang ada tanpa melihat bagaimana proses untuk mendapatkan keluaran tersebut.

Dari keluaran yang dihasilkan, kemampuan program dalam memenuhi kebutuhan pemakai dapat diukur sekaligus dapat diketahui kesalahannya. *Black-Box testing* berusaha untuk menemukan kesalahan dalam kategori berikut:

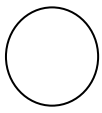
1. fungsi yang tidak benar atau hilang;
2. kesalahan *interface*;
3. error pada struktur data atau akses database *external;*
4. error pada kinerja;
5. error pada saat inisialisasi dan terminasi;
6. kesensitifan sistem terhadap nilai input tertentu; dan
7. batasan dari suatu data.

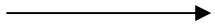
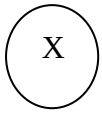
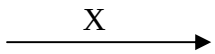
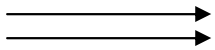
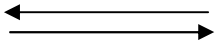
2.6.1 Metode/ Teknik Pengujian *Black Box*

2.6.1.1 Metode *Graph Based*

Pada teknik/metode ini langkah yang dilakukan adalah memahami objek (data dan program) yang dimodelkan didalam perangkat lunak. Langkah selanjutnya menentukan sederetan pengujian yang membuktikan bahwa semua objek memiliki hubungan antara satu dengan lainnya.

Tabel 2.2 Representasi simbolik dari grafik metode graph based:

Notasi	Arti
	Simpul atau node Menggambarkan suatu objek

	*Link* Menggambarkan hungungan antar objek
	*Node weight* Menggambarkan properti atau nilai dari data
	*Link weight* Menggambarkan karakteristik link
	*Link parallel* Menggambarkan hubungan yang berbeda yang dibangun antar simpul
	*Link simetris* Menggambarkan hubungan dua arah antara dua objek

Terdapat tiga pola *link weight*, yaitu.

1. *Transitivitas*, yaitu hubungan antara tiga objek atau lebih yang menentukan bagaimana pengaruh hubungan tersebut menyebar pada objek yang ditentukan.
2. *Simetris*, yaitu hubungan antara dua objek secara dua arah.
3. *Refleksif*, yaitu hubungan yang mengarah pada node itu sendiri atau *loop null*.

2.6.1.2 *Equivalence Partioning*

Merupakan *test case* yang ideal mengungkapkan kelas kesalahan, karena pada teknik ini berusaha mengungkapkan kelas-kelas kesalahan sehingga mengurangi jumlah total *test case* yang harus dikembangkan. Metode ini membagi domain input dari suatu program kedalam kelas - kelas data sehingga test case dapat diperoleh.

Kelas data yang terbentuk disajikan sebagai kondisi input dalam kasus uji.Kelas merupakan himpunan nilai-nilai yang valid dan tidak valid. Desain *test case* partisi ekivalensi didasarkan pada evaluasi terhadap kelas ekivalensi untuk suatu kondisi input. Kondisi input bisa merupakan suatu range harga, harga numerik (harga khusus/tertentu), serangkaian harga (himpunan), suatu kondisi boolean.

Kelas ekivalensi dapat ditentukan sesuai pedoman sebagai berikut:

1. bila kondisi input berupa *suatu range*, maka input kasus ujinya 1 valid dan 2 invalid;
2. bila kondisi input berupa *harga khusus*, maka input kasus ujinya 1 valid dan 2 invalid;
3. bila kondisi input berupa *anggota himpunan*, maka input kasus ujinya 1 valid dan 2 invalid; dan
4. bila kondisi input berupa *anggota boolean*, maka input kasus ujinya 1 valid dan 1 invalid.

Contoh ;

Sebuah aplikasi perbankan otomatis, dimana aplikasi ini digunakan oleh nasabah untuk bertransaksi dengan Bank menggunakan ATM. Untuk aksesnya menggunakan password/PIN dengan 4 digit dan diikuti dengan serangkaian perintah kata kunci yang memicu berbagai fungsi perbankan.

Sebagian input data dari aplikasi ini adalah.

- Password/PIN : 4 digit.
- Pilihan menu : “penarikan”, “pembayaran”, “informasi”, “transfer”, dll.

Pembahasan

Kondisi input yang sesuai dengan masing2 elemen data untuk aplikasi perbankan tersebut adalah.

- Password/PIN : kondisi input ***range*** (4 digit numeric).
- Pilihan menu : kondisi input ***himpunan*** (berisi beberapa pilihan/perintah) .

Data test case yang didesain adalah.

- Password/PIN (kondisi input : range).
  - Valid (0000, 1111, 1234, 9876, 9999).
  - Invalid (000, 789, 555, 999, 100).
  - Invalid (00, 11, 99, 12, 89).
- Pilihan menu (kondisi input : himpunan).
  - Valid (“penarikan”, “pembayaran”, “informasi”, “transfer”).
  - Invalid (1, 3, 5, 0).
  - Invalid (cancel, stop, enter, clear).

2.6.1.3 Teknik *State Transitin Table*

State Transition testing menggunakan model sistem, yang terdiri dari:

- status yang terdapat dalam program;
- transisi antar status–status;
- kejadian yang merupakan sebab dari transisi–transisi tersebut; dan
- aksi-aksi yang akan dihasilkan.

Model umumnya direpresentasikan dalam bentuk *state transition diagram.* Test case didesain untuk memeriksa validitas transisi antar status. Test case tambahan juga akan didesain untuk testing terhadap transisi-transisi yang tidak termasuk dan tidak dispesifikasikan.

Test case didesain untuk memeriksa transisi-transisi yang valid. Untuk setiap test case, terdapat spesifikasi sebagai berikut:

- status mulai;
- masukan;
- keluaran yang diharapkan; dan
- status akhir yang diharapkan.

2.6.1.4 *Boundary Value Analysis*

Boundary Value fokus pada suatu batasan nilai dimana kemungkinan terdapat cacat yang tersembunyi. *Boundary Value* mengarahkan pada pemilihan kasus uji yang melatih nilai-nilai batas. *Boundary Value* merupakan desain teknik kasus uji yang melengkapi *Equivalence class testing*. Dari pada memfokuskan hanya pada kondisi input, *Boundary Value Analysis* juga menghasilkan kasus uji dari domain output. Menguji untuk input di sekitar batas atas maupun bawah sebuah range nilai yang valid, menguji nilai maksimal dan minimal.
Menerapkan (1 & 2) untuk output, menguji batas struktur data yang dipakai misal ukuran array.

Langkah-langkah pengujian.

1. Identifikasi kelas-kelas yang ekuivalen (*equivalence class*).
2. Identifikasi batasan untuk tiap equivalence class.
3. Buat *test case* untuk tiap batasan suatu nilai dengan memilih titik pada batasan, satu titik pada nilai bawah batasan dan satu titik pada nilai atas batasan.

Contoh form pengujian.

1. Pengujian interface sistem.
2. Pengujian fungsi dasar sistem.
3. Pengujian form handle sistem.
4. Pengujian keamanan sistem.

2.7 IP Address

IP address digunakan sebagai alamat dalam hubungan antar *host* di *internet* sehingga merupakan sebuah sistem komunikasi yang *universal* karena merupakan metode pengalamatan yang telah diterima di seluruh dunia. Dengan menentukan *IP address* berarti kita telah memberikan identitas yang *universal* bagi setiap interface komputer. Jika suatu komputer memiliki lebih dari satu *interface* (misalkan menggunakan dua *ethernet*) maka kita harus memberi dua *IP address* untuk komputer tersebut masing-masing untuk setiap *interface*-nya.

2.7.1 Format Penulisan *IP Address*

IP address terdiri dari bilangan *biner* 32 *bit* yang dipisahkan oleh tanda titik setiap 8 *bit*-nya. Tiap 8 *bit* ini disebut sebagai oktet. Bentuk *IP address* dapat dituliskan sebagai berikut.

xxxxxxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx

Jadi *IP address* ini mempunyai range dari

00000000.00000000.00000000.00000000

sampai

11111111.11111111.11111111.11111111

Notasi *IP address* dengan bilangan *biner* seperti ini susah untuk digunakan, sehingga sering ditulis dalam 4 bilangan desimal yang masing-masing dipisahkan oleh 3 buah titik yang lebih dikenal dengan “notasi desimal bertitik”. Setiap bilangan desimal merupakan

nilai dari satu oktet *IP address*. Contoh hubungan suatu *IP address* dalam format *biner* dan desimal.

Desimal	**167**	**205**	**206**	**100**
Biner	10100111	11001101	11001110	01100100

2.7.2 Pembagian Kelas IP Address

Jumlah *IP address* yang tersedia secara teoritis adalah 255x255x255x255 atau sekitar 4 milyar lebih yang harus dibagikan ke seluruh pengguna jaringan internet di seluruh dunia. Pembagian kelas-kelas ini ditujukan untuk mempermudah alokasi *IP Address*, baik untuk *host*/jaringan tertentu atau untuk keperluan tertentu.

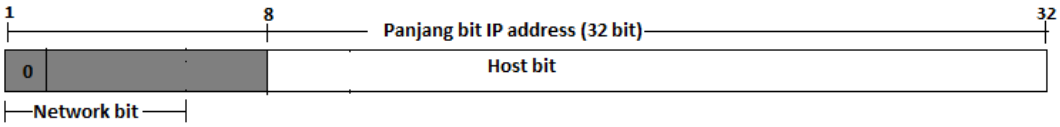
IP address dapat dipisahkan menjadi 2 bagian, yakni bagian *network* (*net ID*) dan bagian *host* (*host ID*). *Net ID* berperan dalam identifikasi suatu *network* dari *network* yang lain, sedangkan *host ID* berperan untuk identifikasi *host* dalam suatu *network*. Jadi, seluruh *host* yang tersambung dalam jaringan yang sama memiliki *net ID* yang sama. Sebagian dari *bit-bit* bagian awal dari *IP address* merupakan *network bit/network number*, sedangkan sisanya untuk *host*. Garis pemisah antara bagian *network* dan *host* tidak tetap, bergantung kepada kelas *network*. *IP address* dibagi ke dalam lima kelas, yaitu kelas A, kelas B, kelas C, kelas D dan kelas E. Perbedaan tiap kelas adalah pada ukuran dan jumlahnya. Contohnya *IP* kelas A dipakai oleh sedikit jaringan namun jumlah host yang dapat ditampung oleh tiap jaringan sangat besar. Kelas D dan E tidak

digunakan secara umum, kelas D digunakan bagi jaringan *multicast* dan kelas E untuk keprluan eksperimental. Perangkat lunak *Internet Protocol* menentukan pembagian jenis kelas ini dengan menguji beberapa bit pertama dari *IP address*. (Wajianto)**.**

2.7.1.1. Kelas *IP Address*

a) Kelas A

Gambar 2.18 merupakan struktur *IP address* kelas A.



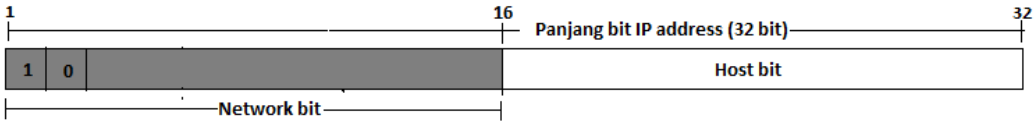
Gambar 2.18 Struktur *IP address* kelas A

(Sumber : Sofana 2013)

Jika *bit* pertama dari *IP address* adalah 0 maka *IP address* termasuk dalam *network* kelas A. *Bit* ini dan 7 *bit berikutnya* (8 *bit* pertama) merupakan *bit-bit network* (*network bit*) dan boleh bernilai berapa saja (kombinasi angka 1 dan 0), sedangkan 24 *bit* terakhir merupakan *bit host*. Ingatlah *IP address* harus dikoneversikan dari bentuk *biner* ke bentuk decimal. Dengan demikian, hanya ada 128 *network* kelas A, yakni dari nomor 0.xxx.xxx.xxx sampai 127.xxx.xxx.xxx Setiap *network* dapat menampung lebih dari 16 juta (2536) *host* (xxx adalah variable, nilainya dari 0 s.d 255).

b) Kelas B

Gambar 2.19 merupakan struktur *IP address* kelas B.



Gambar 2.19 Struktur IP *address* kelas B

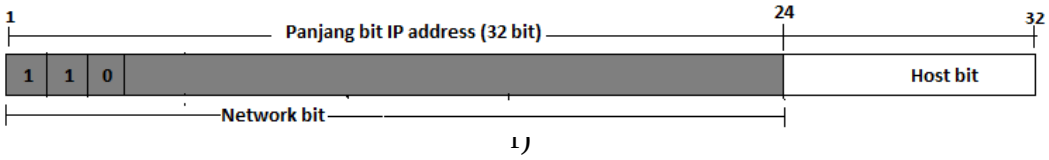
(Sumber : Sofana 2013)

Jika 2 *bit* pertama dari *IP address* adalah 1 0, maka *IP address* termasuk dalam *network* kelas B. Dua *bit* ini dan 14 *bit* berikutnya (16 *bit* pertama) merupakan *bit network* dan boleh bernilai berapa saja (kombinasi angka 1 dan 0), sedangkan 16 *bit* terakhir merupakan *bit host*.

Jika bentuk *biner* dikonversikan ke bentuk decimal maka akan terdapat lebih dari 16 ribu *network* kelas B, yakni dari *network* 128.0.xxx.xxx hingga 191.255.xxx.xxx. setiap *network* kelas B mampu menampung lebih dari 65 ribu *host* (2562).

c) Kelas C

Gambar 2.20 merupakan struktur *IP address* kelas C.



Gambar 2.20 Struktur *IP address* kelas C

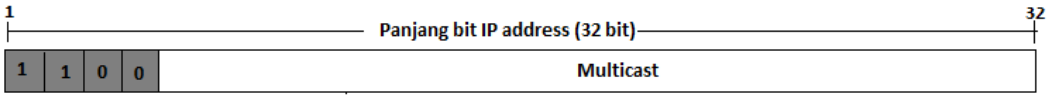
(Sumber : Sofana, Iwan. 2013)

Jika 3 *bit* pertama dari *IP address* adalah 110,maka *IP address* termasuk dalam *network* kelas C. Tiga *bit* ini dan 21 *bit* berikutnya (24 *bit* pertama) meruapakan *bit network* dan boleh bernilai berapa saja (kombinasi angka 1 dan 0), sedangkan 8 *bit* terakhir merupakan *bit host*.

Jika bentuk *biner* dikonversikan ke bentuk decimal maka akan terdapat lebih dari 2 juta *network* kelas C, yakni dari nomor 192.0.0.xxx hingga 223.255.255.xxx. setiap *network* kelas C hanya mampu menampung sekitar 256 *host*.

4) Kelas D

Selain tiga kelas diatas, ada 2 kelas lagi yang ditujukan untuk pemakaian khusus, yakni kelas D dan kelas E. Gambar 2.21 merupakan struktur *IP address* kelas D.



Gambar 2.21 Struktur *IP address* kelas D

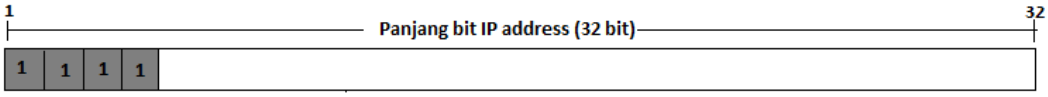
(Sumber : Sofana. 2013)

Jika 4 *bit* pertama adalah 1110, maka *IP address* termasuk dalam kelas D. *IP address* kelas D digunakan untuk *multicast address*, yakni sejumlah computer yang memakai bersama suatu aplikasi (bedakan dengan pengertian *network address* yang mengacu kepada sejumlah komputer yang memakai bersama suatu *network*).

Salah satu penggunaan *multicast address* yang sedang berkembang saat ini di *Internet* adalah untuk aplikasi *real-time video conference* yang melibatkan lebih dari dua *host* (*multipoint*), menggunakan *Multicast Backbone* (*Mbone*). Pada *IP address* kelas D tidak dikenal *bit-bit network* dan *host.*

5) Kelas E

Kelas terakhir adalah kelas E. *IP address* kelas E masih bersifat percobaan. Jika 4 *bit* pertama adalah 1111 (atau sisa dari seluruh kelas) maka *IP address* termasuk dalam kategori kelas E. Pemakai *IP address* kelas E dicadangkan untuk kegiatan eskperimental. Gambar 2.22 merupakan struktur *IP address* kelas E.



Gambar 2.22 Struktur IP *address* kelas E

(Sumber : Sofana 2013)

(Sofana, 2013)