BAB II

TINJAUAN PUSTAKA

II.1. Sistem Pendukung Keputusan

Sistem pendukung keputusan (SPK) atau *Decision Support Systems (DSS)* adalah sistem informasi interaktif yang menyediakan informasi, pemodelan, dan pemanipulasian data yang digunakan untuk membantu pengambilan keputusan pada situasi yang semiterstruktur dan situasi yang tidak terstruktur di mana tek seorang pun tahu secara pasti bagaimana keputusan seharusnya dibuat. Konsep *DSS* dikemukakan pertama kali oleh Scott-Morton pada tahun 1971. Beliau mendefenisikan cikal bakal *DSS* tersebut sebagai ”Sistem berbasis komputer interaktif, yang membantu pengambil keputusan menggunakan data dan model untuk memecahkan persoalan-persoalan tidak terstruktur”.

DSS dibuat sebagai reaksi atas ketidakpuasan terhadap TPS dan MIS.Sebagaimana diketahui, TPS lebih memfokuskan diri dari pada perekaman dan pengendalian transaksi yang merupakan kegiatan yang bersifat berulang dan terdefenisi dengan baik, sedangkan MIS lebih berorientasi pada penyediaan laporan bagi manajemen yang sifatnya tidak fleksibel.*DSS* lebih ditujukan untuk mendukung manajemen dalam melakukan pekerjaan yang bersifat analitis, dalam situasi yang kurang terstruktur dan dengan kriteria yang kurang jelas.*DSS* tidak dimaksudkan untuk mengotomasikan pengambilan keputusan, tetapi memberikan perangkat interaktif yang memungkinkan pengambil keputusan dapat melakukan

berbagai analisis dengan menggunakan model-model yang tersedia. (Abdul Kadir, 2014 : 108).

Adapun beberapa karakteristik yang terdapat pada sistem pendukung keputusan adalah sebagai berikut :

- Menawarkan keluasan, kemudahan beradaptasi, dan tanggapan yang cepat.
- Memungkinkan pemakai memulai dan mengendalikan masukan dan keluaran.
- Dapat dioperasikan dengan sedikit atau tanpa bantuan pemrogram professional.
- Menyediakan dukungan untuk keputusan dan permasalahan yang solusinya tak dapat ditentukan di depan.
- Menggunakan analisis data dan perangkat pemodelan yang canggih. (Abdul Kadir ; 2014 : 108).

II.2. Metode*Analitic Hierarchy Process (AHP)*

Metode *Analitic Hierarchy Process (AHP)* merupakan suatu model pendukung keputusan yang dikembangkan oleh Thomas L. Saaty. Model pendukung keputusan ini akan menguraikan masalah multi faktor atau multi kriteria yang kompleks menjadi suatu hirarki. Hirarki didefenisikan sebagai suatu representasi dari sebuah permasalahan yang kompleks dalam suatu struktur multi level. (Sylvia Hartati Saragih;2013 : 83).

Contoh Kasus :

Sistem Penunjang Keputusan menggunakan metode AHP dan dimaksudkan untuk membantu dalam pengambilan keputusan untuk menentukan

kualitas gula tumbu.Dalam penentuannya ada tiga kriteria yaitu warnagula tumbu, rasa gula tumbu dan kekerasan gula tumbu.

a. Kriteria : kekerasan, warna, rasa
b. Alternatif : keras, sedang, lembek, merah, merah tua, hitam, manis, kurang manis, pahit
c. Sub alternatif : kualitas 1, kualitas 2, kualitas 3, kualitas 4.

1. Pembobotan Kriteria

Hasil dari analisis diperoleh perhitungan pembobotan untuk semua kriteria yaitu :

a. Kekerasan : Warna

1. Keras : Merah : 7, keras sangat penting daripada warna merah
2. Keras : Merah tua : 4, keras sedikit cukup penting daripada merah tua
3. Keras : Hitam : 3, keras agak lebih penting daripada hitam
4. Sedang : Merah : 5, sedang cukup penting daripada merah
5. Sedang : Merah tua : 3, sedang agak lebih penting daripada merah
6. Sedang : Hitam : 1, sedang sama penting dengan merah
7. Lembek : Merah : 3, lembek agak lebih penting daripada merah
8. Lembek : Merah tua : 2, lembek sedikit agak lebih penting daripada merah tua
9. Lembek : Hitam : 2, lembek sedikit agak lebih penting daripada hitam

b. Kekerasan : Rasa

1. Keras : Manis : 5, keras cukup penting daripada manis
2. Keras : Kurang manis : 5, keras cukup penting daripada kurang manis
3. Keras : Pahit : 4, keras sedikit cukup penting daripada pahit
4. Sedang : Manis : 4, sedang sedikit cukup penting daripada manis

5. Sedang : Kurang manis : 3, sedang agak lebih penting daripada kurangmanis

6.Sedang : Pahit : 1, sedang sama penting dengan pahit

7. Lembek :Manis : 4, lembek sedikit cukup penting daripada manis

8. Lembek :Kurangmanis : 4, lembek sdikit cukup penting daripda kurang manis

9. Lembek : Pahit : 3, lembek agak lebih penting daripada pahit

c. Warna : Rasa

1. Merah : Manis : 1 , merah sama penting denganmanis

2. Merah : Kurang manis : 2, sedikit agak lebih pentng daripada kurang manis

3. Merah : Pahit : 3, merah agak lebih penting daripada pahit

4. Merah tua : Manis : 1, merah tua sama penting dengan merah

5. Merah tua : Pahit : 4, merah tua sedikit cukup penting daripada merah

6. Hitam : Manis : 2, hitam sedikit agak lebih penting daripada manis

7. Hitam : Kurang manis : 2, hitam sedikit agk lebih penting dari kurang manis

8. Hitam : Pahit : 3, hitam agak lebih penting daripada pahit

2. Matrik Perbandingan Berpasangan

Berikut ini adalah matrik perbandingan berpasangan dalam menentukan kualitas gula tumbu.Kriteria :

a. Kekerasan: keras

b. Warna : merah

c. Rasa : manis

Dibawah ini merupakan matrik perbandingan untuk criteria yang ditunjukkan oleh tabel II.1:

Tabel II.1. Matrik perbandingan untuk kriteria

	Kekerasan	Warna	Rasa
Kekerasan	1	7	5
Warna	1/7	1	1
Rasa	1/5	1/1	1

Selanjutnya matrik perbandingan untuk kriteria yang disederhanakan ditunjukkan oleh tabel II.2 :

Tabel II.2. Matrik perbandingan untuk kriteria yang disederhanakan

	Kekerasan	Warna	Rasa
Kekerasan	1	7	5
Warna	0,143	1	1
Rasa	0,2	1	1
Kolom	1,343	9	7

3. Menormalkan Data

Dengan unsur-unsur pada tiap kolom dibagi dengan jumlah total pada kolom yang bersangkutan, akandiperoleh bobot relatif yang dinormalkan. Nilai vektor eigen dihasilkan dari rata-rata nilai bobot relative untuk tiapbaris. Hasilnya dapat dilihat pada tabel II.3 berikut ini :

Tabel II.3. Matrik perbandingan untuk kriteria yang dinormalkan

	Kekerasan	Warna	Rasa	Baris	Eigen Vector
Kekerasan	0,745	0,778	0,714	2,237	0,746
Warna	0,106	0,111	0,143	0,360	0,120
Rasa	0,149	0,111	0,143	0,403	0,134

Berikut adalah perhitungan bobot relatif yang dinormalkan:

1 : 1,343 = 0,745 7 : 9 = 0,778 5 : 7 = 0,714

0,143 : 1,343 = 0,106 1 : 9 = 0,111 1 : 7 = 0,143

0,2 : 1,343 = 0,149 1 : 9 = 0,111 1 : 7 = 0,143

4. Menghitung Nilai Eigen Vector dan Menguji Konsistensinya

Menghitung nilai *eigen vector* dan menguji konsistensinya, jika tidak konsisten maka pengambilan data(prefensi) perlu diulangi. Nilai *eigen vector* yang dimaksud adalah nilai *eigen vector* maksimum yang diperoleh.

Berikut ini adalah perhitungan nilai *eigen vector*.

Eigen vector kekerasan = Σ Baris / kolom

= 2,237 / 3

= 0,746

Eigen vector rasa = Σ Baris / kolom

= 0,360 / 3

= 0,120

*Vector eigen*warna = Σ Baris / kolom

= 0,403 / 3

= 0,134

Selanjutnya nilai eigen maksimum (□maksimum) didapat dengan menjumlahkan hasil perkalian jumlahkolom dengan eigen vector. Nilai eigen maksimum yang dapat diperoleh adalah sebagai berikut :

□maksimum = (1,343 x 0,746) + (0,120 x 9) + (0,134 x 7)

= 1,014 + 0,937 + 1,119

= 3,069

Karena matrik berordo 3 (yakni terdiri dari 3 kolom), maka nilai indeks konsistensi (*CI*) yang diperoleh adalah :

CI = mak – n/ n-1

= 3,069 – 3/ 3-1

= 0,069/ 2

= 0,035

Untuk n = 3, RI = 0,580 (tabel skala Saaty), maka :

CR = C1/ R1 = 0,00,060< 0,100

Karena CR (Rasio Konsistensi) < 0,100 maka hasil konsisten.

Dari hasil perhitungan pada tabel diatas diperoleh hasil :

Kekerasan : 0,746 x 100% = 74,6%

Warna : 0,120 x 100% = 12%

Rasa : 0,134 x 100% = 13,4%

Kekerasan > 70% maka gula tumbu memiliki kualitas 1.

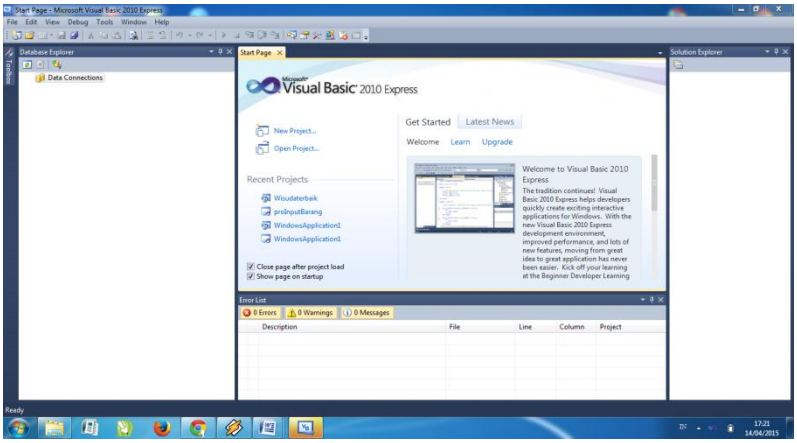
II.6. *Microsoft Visual Basic 2010*

Visual Basic 2010 merupakan salah satu bagian dari produk pemrograman terbaru yang dikeluarkan oleh *Microsoft*, yaitu *Microsoft Visual Studio 2010. Visual Studio* merupakan produk pemrograman andalan dari *mocrosoft corporation*, dimana di dalamnya berisi beberapa jenis *IDE* pemrograman seperti *Visual Basic, Visual C++, Visual Web Developer, Visual C#,* dan *Visual F#*.

Semua *IDE* pemrograman tersebut sudah mendukung penuh implementasi *.Net Framework* terbaru, yaitu *.Net Framework 4.0* yang merupakan pengembangan dari *.Net Framework 3.5*. Adapun *database* standar yang disertakan adalah *Microfot SQL Server 2008 express*.

Visual Basic 2010 merupakan versi perbaikan dan pengembangan dari versi pendahulunya yaitu *visual basic 2008*. Beberapa pengembangan yang terdapat di dalamnya antara lain dukungan terhadap *library* terbaru dari *Microsoft*, yaitu *.Net Framework 4.0,* dukungan terhadap pengembangan aplikasi menggunakan *Microsoft SilverLight*, dukungan terhadap aplikasi berbasis *cloudcomputing*, serta perluasan dukungan terhadap *database-database*, baik *standalone* maupun *database server.* (Wahana Komputer, 2011 :2).

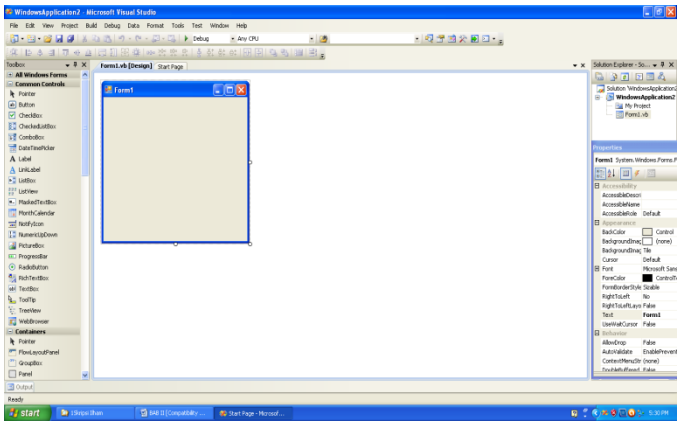
Untuk melihat tampilan *visual studio* 2010 dapat dilihat pada gambar II.1.sebagai berikut :



Gambar II.1. Tampilan Utama Visual Studio 2010

Sumber : (Wahana Komputer;2010:12)

Di dalam *Visual Studio.NET* menyediakan tampilan *Interface* yang sangat mudah untuk para pengguna merancang dan memodifikasi bentuk atau *Interface* dari program yang akan dibuat, dimana pada tampilan ini difasilitasi dengan *Tool* dan fasilitas pendukung lainnya untuk mempermudah pengerjaannya.



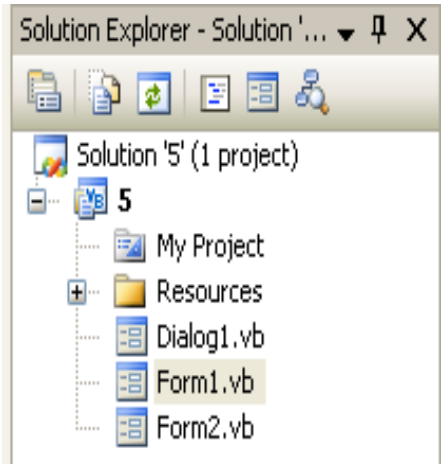
Gambar II.2. Tampilan *Area* Kerja *Visual Studio.NET*

Sumber: (Wahana Komputer; 2010: 10)

Dalam buku Belajar Pemrograman *Visual Basic* terbitan Wahana Komputer dmenjelaskan tampilan Microsoft *Visual Studio.NET.*Ada banyak hal penting yang harus diketahui oleh seorang programmer, diantaranya adalah:

1. *Solution Explorer*

Merupakan fitur yang terletak di sebelah kanan atas di bawah menu aplikasi yang digunakan untuk menampilkan daftar desain *form* dengan *struktur tree* dari project yang sedang dibuka. Dengan adanya fitur ini, memudahkan untuk berpindah antardesain *form* yang sudah anda buat secara cepat dan mudah.

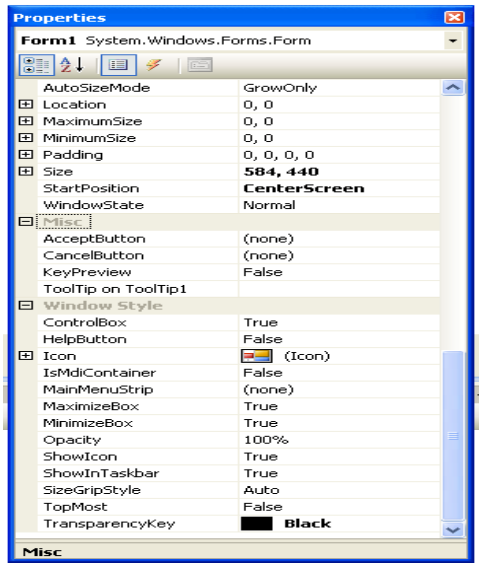


Gambar II.3. *Solution Explorer*

Sumber: (Wahana Komputer; 2010:14)

2. *Properties*

Merupakan fitur yang digunakan untuk melakukan pengaturan properti dari objek-objek yang digunakan dalam desain *form* yang akan dibuat, seperti pengaturan *text, visible, font, name* dan lain sebagainya.

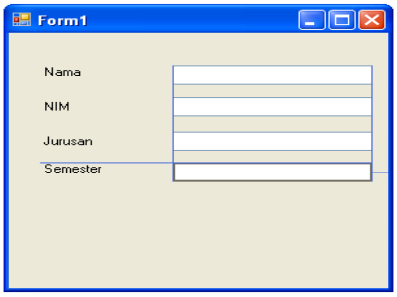


Gambar II.4. Jendela *Properties*

Sumber: (Wahana Komputer; 2010: 15)

3. *Form* Designer

Merupakan fitur yang digunakan untuk membuat desain antarmuka atau *interface* dari aplikasi yang akan dikembangkan. Dengan mengadopsi fitur *click and drop*, proses pemanbahan komponen pada *form* menjadi semakin dinamis dan mudah. Selain itu tersedia pula fitur *guidelines* yang memudahkan anda dalam menata komponen yang terdapat pada desain *form* yang sedang anda kerjakan.

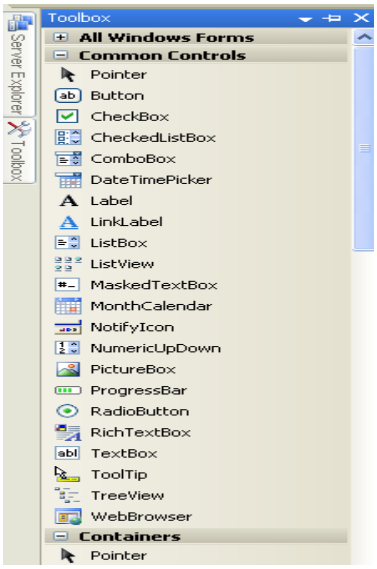


Gambar II.5.*Form Designer*

Sumber: (Wahana Komputer; 2010: 15)

4. *Component Toolbox*

Merupakan fitur *Visual Basic.NET* yang digunakan untuk menampilkan daftar dari komponen baik visual maupun non visual yang bias ditambahkan ke dalam desain *form* yang akan dibuat. Dalam komponen *toolbox*, terdapat berbagai macam kategori sesuai dengan kelompok komponen yang diakses sehingga akan memudahkan dalam mencari komponen yang akan ditambahkan kedalam desain *form*.

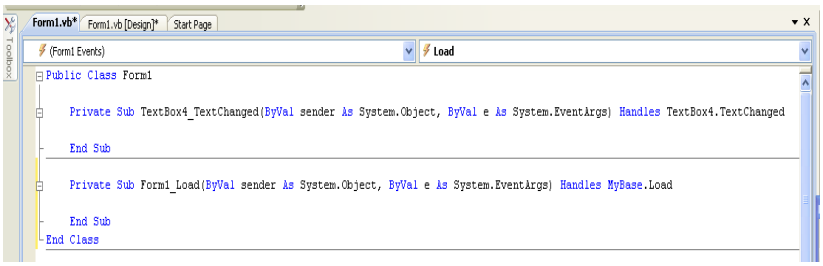


Gambar II.6.*Component Toolbox*

Sumber: (Wahana Komputer; 2010: 15)

5. *Code Editor*

Merupakan fitur yang digunakan untuk menambahkan kode program dari aplikasi atau *project* yang akan dikerjakan.



Gambar II.7.*Code Editor*

Sumber: (Wahana Komputer; 2010: 16)

II.3. Basis Data

Basis data dapat didefenisikan sebagai koleksi dari data-data yang terorganisasi sedemikian rupa sehingga data mudah disimpan dan dimanipulasi (diperbarui, dicari, diolah dengan perhitungan-perhitungan tertentu, serta dihapus).Secara teoritis, basis data tidak harus berurusan dengan komputer (misalnya, catatan belanja hari ini yang dibuat oleh seorang ibu rumah tangga juga merupakan basis data dalam bentuk yang sangat sederhana). (Adi Nugroho, 2011 : 4).

II.4. *SQL Server* 2008

SQL Server 2008 adalah sebuah*RDBMS (Relational Database ManagementSystem)* yang sangat powerful dan telahterbukti kekuatannya dalam mengolah data.Dalam versi terbarunya ini, *SQL Server* 2008memiliki banyak fitur yang bisa dihandalkanuntuk meningkatkan performa *database.SQL Server* 2008 memiliki suatu *GUI(Graphic User Interface)* yang kita gunakanuntuk melakukan aktivitas sehari hariberkaitan dengan database,seperti menulis*T-SQL*, melakukan *backup* dan *restoredatabase*, melakukan security databaseterhadap aplikasi, dan sebagainya. Pada *GUI*tersebut kita bisa melakukan settinganterhadap *SQL Server*

untuk berkerja lebihoptimal.*Setting*an juga bisa dilakukanmenggunakan *script* untuk memudahkan*developer* mengubah *Setting Options* pada*SQL Server* 2008. (Ruslan, 2013 : 39).

II.5. Normalisasi

Normalisasi dapat dipahami sebagai tahapan-tahapan yang masing-masing berhubungan dengan bentuk normal.Bentuk normal adalah keadaan relasi yang dihasilkan dengan menerapkan aturan sederhana berkaitan dengan konsep kebergantungan fungsional pada relasi yang bersangkutan. (Adi Nugroho, 2011 : 199). Kita akan menggambarkannya secara garis besar sebagai berikut :

1. Bentuk Normal Pertama (1NF/ *First Normal Form*)

   Bentuk normal pertama adalah suatu bentuk relasi dimana atribut bernilai banyak (*multivalues attribute*) telah dihilangkan sehingga kita akan menjumpai nilai tunggal (mungkin saja nilai *null*) pada perpotongan setiap baris dan kolom.

2. Bentuk Normal Kedua (2ND/ *Second Normal Form*)

   Semua kebergantungan fungsional yang bersifat sebagian (*particial functional dependency*) telah dihilangkan.

3. Bentuk Normal Ketiga (3RD/ *Thrid Normal Form*)

   Semua kebergantungan transitif (*transitive dependency*) telah dihilangkan.

4. Bentuk Normal *Boyce-Codd* (BCNF/ *Boyce-Codd Normal Form*)

   Semua anomaly yang tersisa dari hasil penyempurnaan kebergantungan fungsional sebelumnya telah dihilangkan.

5. Bentuk Normal Keempat (4NF/ *Fourth Normal Form*)

   Semua kebergantungan bernilai banyak telah dihilangkan.

6. Bentuk Normal Kelima (5NF/ *Fifth Normal Form*)

   Semua anomaly yang tertinggi telah dihilangkan.(Adi Nugroho, 2011 : 199).

II.6. *Unified Modeling Language* (*UML*)

Menurut Windu Gata (2013) Hasil pemodelan pada OOAD terdokumentasikan dalam bentuk *Unified Modeling Language* (*UML*). *UML* adalah bahasa spesifikasi standar yang dipergunakan untuk mendokumentasikan, menspesifikasikan dan membangun perangkat lunak.

UML merupakan metodologi dalam mengembangkan sistem berorientasi objek dan juga merupakan alat untuk mendukung pengembangan sistem. *UML* saat ini sangat banyak dipergunakan dalam dunia industri yang merupakan standar bahasa pemodelan umum dalam industri perangkat lunak dan pengembangan sistem. (Gellysa Urva dan Helmi Fauzi Siregar, 2015 : 93).

Alat bantu yang digunakan dalam perancangan berorientasi objek berbasiskan *UML* adalah sebagai berikut:

1. *Use case* Diagram

Use case adalah rangkaian atau uraian sekelompok yang saling terkait dan membentuk sistem secara teratur yang dilakukan atau diawasi oleh sebuah aktor. *Use case* digunakan untuk membentuk tingkah laku benda dalam sebuah model serta direalisasikan oleh sebuah kolaborasi. Diagram *use case* menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Hal yang ditekankan *pada* diagram ini adalah “apa” yang diiperbuat sistem, dan bukan “bagaimana”. Sebuah

use case merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* menyatakan sebuah aktivitas atas pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan lain sebagainya. Aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. (Hamim Tohari, 2014 : 47).

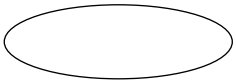
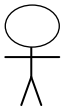
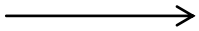
Diagram *use case* dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum, diasumsikan bahwa *use case* yang di-*include*akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal.

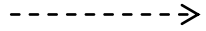
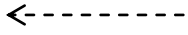
Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain. Oleh karena itu, duplikasi fungsionalitas dapat dihindari degan cara menarik keluar fungsionalitas yang *common*. Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisai dari yang lain. (Hamim Tohari, 2014 : 48).

Use case diagram merupakan pemodelan untuk kelakukan (*behavior*) sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Dapat dikatakan *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi

tersebut. Simbol-simbol yang digunakan dalam *use case* diagram dapat dilihat pada tabel II.4 dibawah ini:

Tabel II.4. Simbol *Use Case*

Gambar	Keterangan
	*Use case* menggambarkan fungsionalitas yang disediakan sistem sebagai unit-unit yang bertukar pesan antar unit dengan aktor, biasanya dinyatakan menggunakan kata kerja di awal nama *use case*.
	Aktor adalah *abstraction* dari orang atau sistem yang lain yang mengaktifkan fungsi dari target sistem. Untuk mengidentifikasikan aktor, harus ditentukan pembagian tenaga kerja dan tugas-tugas yang berkaitan dengan peran pada konteks target sistem. Orang atau sistem bisa muncul dalam beberapa peran. Perlu dicatat bahwa aktor berinteraksi dengan *use case*, tetapi tidak memiliki control terhadap *use case*.
	Asosiasi antara aktor dan *use case*, digambarkan dengan garis tanpa panah yang mengindikasikan siapa atau apa yang meminta interaksi secara langsung dan bukannya mengidikasikan aliran data.
	Asosiasi antara aktor dan *use case* yang menggunakan panah terbuka untuk mengidinkasikan

	bila aktor berinteraksi secara pasif dengan sistem.
	*Include,* merupakan di dalam *use case* lain (*required*) atau pemanggilan *use case* oleh *use case* lain, contohnya adalah pemanggilan sebuah fungsi program.
	*Extend,* merupakan perluasan dari *use case* lain jika kondisi atau syarat terpenuhi.

(Sumber:Gellysa Urva dan Helmi Fauzi Siregar; 2015 : 94)

II.6.1. Elemen-Elemen Diagram *Use Case*

Beberapa elemen yang digunakan pada diagram *Use Case* adalah sebagai berikut :

1. Sistem

   Sistem menyatakan batasan sistem dalam relasi dengan *actor-actor* yang menggunakannya (di luar sistem) dan fitur-fitur yang harus disediakan (dalam sistem). Sistem digambarkan dengan segi empat yang membatasi semua *use case* dalam sistem terhadap pihak mana sistem akan berinteraksi. Sistem disertai label yang menyebutkan nama dari sistem, tapi umumnya tidak digambarkan karena tidak terlalu memberi arti tambahan pada diagram.

2. *Actor*

   Actor atau aktor dapat berupa merupakan manusia, sistem, atau *device* yang memiliki peranan dalam keberhasilan operasi dari sistem. Digambarkan dengan *icon* yang mungkin bervariasi namun konsepnya sama :

a. Umumnya, untuk orang, digambarkan dengan sosok lengkap seperti dengan kepala, badan, tangan, dan kaki.
b. Umumnya, untuk sistem, digambarkan dengan segi empat disertai notasi "<<Actor>>" di atas label nama.

3. *Use Case*

   Use Case mengidentifikasi fitur kunci dari sistem. Tanpa fitur ini, sistem tidak akan memenuhi permintaan *user/actor*. Setiap *Use Case* mengekspresikan *goal* dari sistem yang harus dicapai. Diberi nama sesuai dengan *goal*-nya dan digambarkan dengan *elips* (dengan nama di dalamnya). Fokus tetap pada *goal*, bukan "bagaimana" mengimplementasikannya walaupun *use case* berimplikasi pada prosesnya nanti.

4. *Assosiation*

   Mengidentifikasikan interaksi antara setiap *actor* tertentu dengan setiap *use case* tertentu. Digambarkan dengan garis antara *actor* terhadap *use case* yang bersangkutan. Asosiasi bisa berarah (garis dengan anak panah) jika komunikasi satu arah, namun umumnya terjadi kedua arah (tanpa anak panah) karena selalu diperlukan demikian.

5. *Stereotape*

   Memungkinkan perluasan *UML* tanpa memodifikasinya.Berperan sebagai kualifier pada satu elemen model.Menyediakan informasilebih banyak mengenai peranan dari elemen tanpa menyebutkan implementasinya. Terutama untuk menggambarkan :

a. *Use Case Dependency*

b. *Class-class*

c. *Package-package*

d. *Classifier*

Notasi dalam diagram dengan *Guil emet* "<<…>>"

6. *Depedency*

Depedensi <<include>>

a. Mengidentifikasi hubungan antar dua *use case* dimana yang satu memanggil yang lain.

b. Jika pada beberapa *use case* terdapat bagian yang memiliki aktivitas yang sama maka bagian aktivitas tersebut biasanya dijadikan *use case*tersendiri dengan relasi depedensi setiap *use case* semula ke *use case* yang baru ini sehingga memudahkan pemeliharaan.

c. Digambarkan dengan garis putus-putus bermata panah dengan notasi <<include>> pada garis.

d. Arah mata panah sesuai dengan arah pemanggilan.

e. Depedensi <<extend>>

1. Jika pemanggilan memerlukan adanya kondisi tertentu maka berlaku depedensi <<extend>>.
2. Note: konsep "extend" ini berbeda dengan "extend" dalam *java*!
3. Digambarkan serupa dengan depedensi <<include>> kecuali arah panah berlawanan.

7. *Generalization*

   a. Mendefinisikan relasi antar dua *actor* atau dua *use case*. Salah satunya meng-*inherit* dan menambahkan atau *override* sifat dari yang lainnya.

   b. Penggambaran menggunakan garis bermata panah kosong dari yang meng-*inherit* mengarah ke yang di-*inherit*.

   Sebagai acuan dalam membangun diagram *use case*, dapat menerapkan langkah-langkah :

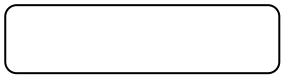
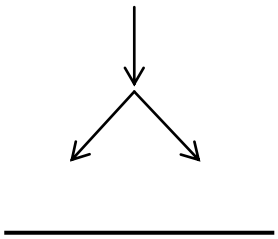
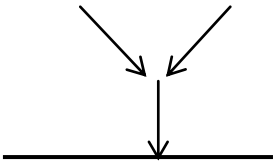
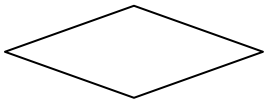
   1. Set konteks dari target sistem.
   2. Identifikasi semua *actor*.
   3. Identifikasi semua *use case*.
   4. Definisikan asosiasi antara setiap *actor* dan setiap *use case*.
   5. Evaluasi setiap *actor* dan setiap *use case* untuk mendapatkan kemungkinan *refinement*.
   6. Evaluasi setiap *use case* untuk depedensi <<include>>.
   7. Evaluasi setiap *use case* untuk depedensi <<extend>>.

Evaluasi setiap *actor* dan setiap *use case* untuk generalisasi. (Hamim Tohari, 2014 : 54).

2. Diagram Aktivitas (*Activity Diagram*)

Activity Diagram menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Simbol-simbol yang digunakan dalam *activity diagram* dapat dilihat pada tabel II.5 dibawah ini :

Tabel II.5. Simbol *Activity Diagram*

Gambar	Keterangan
	*Start point*, diletakkan pada pojok kiri atas dan merupakan awal aktifitas.
	*End point*, akhir aktifitas.
	*Activites*, menggambarkan suatu proses/kegiatan bisnis.
	*Fork* (Percabangan), digunakan untuk menunjukkan kegiatan yang dilakukan secara parallel atau untuk menggabungkan dua kegiatan pararel menjadi satu.
	*Join* (penggabungan) atau rake, digunakan untuk menunjukkan adanya dekomposisi.
	*Decision Points*, menggambarkan pilihan untuk pengambilan keputusan, *true*, *false.*
	*Swimlane*, pembagian *activity* diagram untuk menunjukkan siapa melakukan apa.

(Sumber : Gellysa Urva dan Helmi Fauzi Siregar; 2015 : 94)

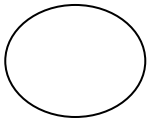
3. Diagram Urutan (*Sequence Diagram*)

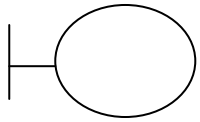
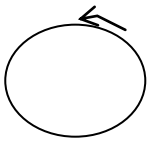
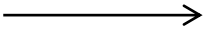
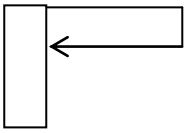
Sequence Diagram menggambarkan interaksi antara sejumlah objek dalam urutan waktu.Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara objek juga interaksi antar objek yang terjadi pada titik tertentu dalam eksekusi sistem. Dalam *UML*, objek, pada *diagram sequence* digambarkan dengan segi empat, yang berisi nama dari objek yang digaris bawahi. Terdapat tiga cara untuk menamai objek yaitu, nama objek, nama objek, dan *class* serta nama *class*.(Hamim Tohari, 2014 : 101).

Pada *sequence diagram*, setiap objek hanya memiliki garis yang digambarkan garis putus-putus ke bawah. Pesan antar objek digambarkan dengan anak panah dari objek yang mengirimkan pesan ke objek yang menerima pesan. (Hamim Tohari, 2014 : 101).

Sequence diagram menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan pesan yang dikirimkan dan diterima antar objek. Simbol-simbol yang digunakan dalam *sequence diagram* dapat dilihat pada tabel II.6 dibawah ini :

Tabel II.6. Simbol *Sequence Diagram*

Gambar	Keterangan
	*EntityClass*, merupakan bagian dari sistem yang berisi kumpulan kelas berupa entitas-entitas yang membentuk gambaran awal sistem dan menjadi landasan untuk menyusun basis data.

	*Boundary Class*, berisi kumpulan kelas yang menjadi *interface* atau interaksi antara satu atau lebih aktor dengan sistem, seperti tampilan formentry dan *form* cetak.
	*Control class*, suatu objek yang berisi logika aplikasi yang tidak memiliki tanggung jawab kepada entitas, contohnya adalah kalkulasi dan aturan bisnis yang melibatkan berbagai objek.
	*Message*, simbol mengirim pesan antar *class*.
	*Recursive*, menggambarkan pengiriman pesan yang dikirim untuk dirinya sendiri.
	*Activation*, *activation* mewakili sebuah eksekusi operasi dari objek, panjang kotak ini berbanding lurus dengan durasi aktivitas sebuah operasi.
	Lifeline, garis titik-titik yang terhubung dengan objek, sepanjang *lifeline* terdapat *activation*.

(Sumber : Gellysa Urva dan Helmi Fauzi Siregar; 2015 : 95)

4. *Class Diagram* (Diagram Kelas)

Kelas (*Class*) adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan perancangan berorientasi objek. Kelas menggambarkan keadaan (atribut/ properti)

suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metode/ fungsi).(Hamim Tohari, 2014 : 84).

Dalam pemodelan statis dari sebuah sistem, diagram kelas biasanya digunakan untuk memodelkan salah satu dari tiga hal berikut :

1. Perbendaharaan dari sistem.
2. Kolaborasi.
3. Skema basis data *logical*.

Kelas memiliki tiga area pokok :

1. Nama (dan *stereotype*).
2. Atribut.
3. Metode atau operasi.

Atribut dan metode dapat memilki salah satu sifat berikut :

1. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan.
2. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.
3. *Public*, dapat dipanggil oleh siapa saja.

Kelas dapat berupa implementasi dari sebuah *interface*, yaitu iclass iabstrak yang hanya memiliki metode. *Interface* tidak dapat langsung diinstansikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class*. Dengan demikian *interface* mendukung resolusi metode pada saat *run-time*.(Hamim Tohari, 2014 : 84).

Class Diagram Merupakan hubungan antar kelas dan penjelasan detail tiap-tiap kelas di dalam model desain dari suatu sistem, juga memperlihatkan aturan-aturan dan tanggng jawab entitas yang menentukan perilaku sistem.

Class diagram juga menunjukkan atribut-atribut dan operasi-operasi dari sebuah kelas dan *constraint* yang berhubungan dengan objek yang dikoneksikan. *Class diagram* secara khas meliputi: Kelas (*Class*), Relasi, *Associations*, *Generalization* dan *Aggregation*, Atribut (*Attributes*), Operasi (*Operations/Method*), *Visibility*, tingkat akses objek eksternal kepada suatu operasi atau atribut.Hubungan antar kelas mempunyai keterangan yang disebut dengan *multiplicity* yang dapat dilihat pada tabel II.7 dibawah ini:

Tabel II.7. *Multiplicity Class Diagram*

Multiplicity	Penjelasan
1	Satu dan hanya satu
0..*	Boleh tidak ada atau 1 atau lebih
1..*	1 atau lebih
0..1	Boleh tidak ada, maksimal 1
n..n	Batasan antara.

(Sumber : Gellysa Urva dan Helmi Fauzi Siregar; 2015 : 95)