BAB 2
LANDASAN TEORI

2.1. Optimasi

Pengertian optimasi, antara lain :

1. Anthony (2014 : 1) mengatakan bahwa “Teknik optimasi merupakan suatu cara yang dilakukan untuk memberikan hasil yang terbaik yang diinginkan.”
2. Sugioko (2013 : 113) mengatakan bahwa “Optimasi adalah suatu disiplin ilmu dalam matematika yang fokus untuk mendapatkan nilai minimum atau maksimum secara sistematis dari suatu fungsi, peluang maupun pencarian nilai lainnya dalam berbagai kasus.”

Banyak cara yang dapat dilakukan dalam menyelesaikan masalah untuk memberikan hasil terbaik. Cara untuk memberikan hasil terbaik ini disebut sistem optimasi atau teknik optimasi. Sistem optimasi ini umumnya mengacu kepada teknik program matematika yang biasanya membahas atau mengacu kepada jalannya program penelitian (*research programming*) tentang masalah yang sedang dihadapi. Teknik ini diharapkan dapat memberikan solusi yang terbaik dari hasil keputusan yang telah diambil dari permasalahan yang sedang dihadapi tersebut. Teknik optimasi digunakan untuk memberikan hasil terbaik dari hal yang terburuk atau hal yang terbaik, tergantung masalah yang dihadapi. Hasil optimasi mungkin hasil tertinggi (misalnya keuntungan) atau hasil terendah (misalnya kerugian). Optimasi memerlukan strategi yang bagus dalam mengambil keputusan agar diperoleh hasil yang optimum.

Penyelesaian suatu permasalahan optimasi akan lebih mudah bila masalah ini diubah dalam bentuk persamaan matematika dan kemudian diselesaikan dengan menggunakan teknik pemograman matematika. Sehingga untuk menyelesaikan masalah optimasi pendistribusian barang, penulis menggunakan teknik pemograman matematika.

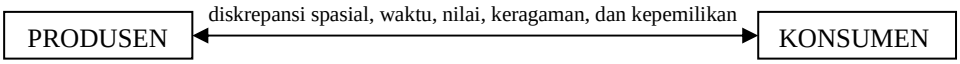
(Anthony, 2014: 1)

2.2. Pendistribusian

2.2.1. Proses Pendistribusian

Pendistribusian adalah kegiatan memasaran yang berusaha memperlancar serta mempermudah penyampaian produk dan jasa dari produsen kepada konsumen sehingga penggunaannya sesuai (jenis, jumlah, harga, tempat, dan saat) dengan yang diperlukan.

Distribusi yang paling efektif akan memperlancar arus atau akses barang oleh konsumen sehingga dapat diperoleh kemudahan memperolehnya. Disamping itu konsumen juga akan mendapat barang sesuai dengan yang diperlukan. Produsen dan konsumen mempunyai kesenjangan spasial, waktu, nilai, keragaman, dan kepemilikan produk karena perbedaan tujuan serta persepsi masing-masing. Dengan distribusi dapat diatasi kesenjangan antara produsen dan konsumen, dapat dilihat pada Gambar 2.1 dibawah ini:



Gambar 2.1. Kesenjangan Antara Produsen dan Konsumen

Secara fungsional, maka proses pendistribusian dapat dibedakan dalam tiga kegiatan, dapat dilihat pada Gambar 2.2 dibawah ini :

Gambar 2.2. Kegiatan Dalam Proses Pendistribusian

1. Kegiatan pemilihan

   Kegiatan pemilihan, meliputi :

   a. Fungsi akumulasi merupakan kegiatan pengumpulan dan penyimpanan persediaan dari beberapa pemasok barang untuk memenuhi kebutuhan permintaan pasar.

b. Fungsi klasifikasi adalah kegiatan mengelompokkan (*grading*) produk-produk kedalam beberapa tingkatan kualitas atau kriteria lain yang berbeda-beda.

c. Fungsi alokasi adalah kegiatan penguraian (*breaking-bulk*) besaran atau jumlah unit persediaan yang homogen menjadi besaran jumlah yang lebih kecil.

d. Fungsi gabungan adalah kegiatan pengumpulan (*product assortment*) beberapa jenis produk menjadi kelompok produk untuk penggunaan yang berkaitan.

2. Kegiatan pertemuan

Kegiatan pertemuan merupakan usaha mempertemukan produsen dengan konsumen. Kegiatannya meliputi usaha mencari informasi tentang permintaan produk dan informasi pasar yang lain serta mencari pelanggan melalui kegiatan promosi.

3. Kegiatan pertukaran

Kegiatan pertukaran merupakan kegiatan negosiasi dan transaksi yang meliputi pertukaran produk beserta kepemilikannya hingga kegiatan pembayaran dan pengiriman barang. Pertukaran meliputi keputusan-keputusan pembelian tentang jumlah, jenis, saat atau waktu, dan syarat-syarat pembayarannya dengan memperhatikan syarat atau kondisi pertukaran yang wajar.

(Budiarto, 2007 : 100-102)

2.2.2. Sistem Distribusi

Para produsen harus dapat menentukan jalur distribusi yang terbaik untuk produk-produknya guna memenuhi kebutuhan konsumen. Berikut ini pilihan saluran distribusi yang dapat digunakan produsen untuk mencapai konsumen:

1. Distribusi melalui pengecer

Produsen mendistribusikan produknya melalui saluran pengecer untuk sampai ke konsumen. *Levi's* dan *Goodyear* misalnya memiliki gerai eceran sendiri. Banyak pengecer menawarkan produk melalui internet.

2. Distribusi melalui grosir

   Produsen mendistribusikan produknya melalui saluran grosir untuk seterusnya didistribusikan ke pengecer sebelum sampai ke konsumen. Pola ini dapat menghemat ruang penyimpanan bagi pengecer.

3. Distribusi melalui agen atau *broken*

   Melalui agen penjualan (*sales agent)*, atau pedagang perantara (*broker)*, produsen mendistribusikan produknya untuk kemudian untuk dijual ke grosir, pengecer, atau keduanya. Industri perumahan biasanya memakai *broker* sebagai perantara pembeli dan penjual.

(Sudaryono, 2014 : 369)

2.2.3. Saluran Distribusi

Faktor-faktor yang mempengaruhi pilihan saluran distribusi, diantaranya adalah :

1. Saluran distribusi

   Saluran distribusi beberapa bersifat ringkas dan sederhana, yang lain bersifat panjang dan rumit. Banyak perusahaan membeli barang yang mereka gunakan dalam operasi mereka langsung dari produsen, sehingga saluran distribusinya menjadi pendek. Sebaliknya, saluran untuk konsumen biasanya lebih panjang dan lebih rumit.

2. Cakupan pasar

   Cakupan pasar yang tepat, jumlah grosir atau pengecer yang akan membawa produk, bergantung pada sejumlah faktor dalam strategi pemasaran. Barang sehari-hari murah atau persediaan organisatoris seperti kertas komputer dan pena-pena, akan terjual dengan baik jika tersedia di banyak gerai.

3. Biaya

   Biaya memainkan peranan utama dalam pemilihan saluran perusahaan dan merupakan salah satu alasan utama atas begitu banyak perusahaan-perusahaan kecil telah memanfaatkan internet. Produsen kecil atau baru sering tidak mampu melakukan banyak fungsi yang diperlukan untuk menjual ke pengecer atau konsumen, sehingga mereka memerlukan bantuan perantara. Tentu saja, perantara tidak gratis, sehingga produsen perlu memberikan kompensasi berupa

discount atau membayar komisi ketika perantara menjual produk kepada pelanggan akhir.

4. Kontrol

   Kontrol terhadap bagaimana, dimana, kapan, dan berapa banyak produk yang dijual. Saluran distribusi lebih panjang berarti kontrol semakin sedikit bagi produsen, dan menjadi semakin jauh dari penjual dan pembeli. Sebaliknya, perusahaan tidak ingin memusatkan terlalu banyak fungsi distribusi dalam kekuasaan terlalu sedikit perantara.

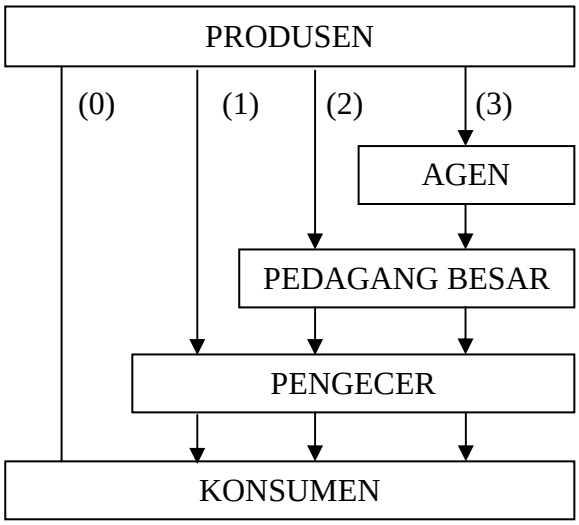
5. Konflik Saluran

   Karena keberhasilan setiap anggota saluran bergantung pada keberhasilan seluruh sistem saluran, idealnya semua anggota saluran kerja sama dengan lancar. Namun setiap anggota saluran juga menjalankan aktivitasnya sendiri, yang berarti mereka bisa tidak setuju pada peran masing-masing anggota. Perselisihan paham seperti itu menciptakan konflik saluran. Suatu sumber bersama dari konflik saluran adalah keputusan oleh produsen untuk menjual produk secara langsung ke target pasar yang sama sebagai mitra saluran yang ada.

(Sudaryono, 2014 : 368-369, 372-373)

2.2.4. Rantai Saluran Distribusi

Sistem distribusi tidak langsung mempunyai keragaman dalam penggunaan banyaknya tingkatan saluran yang diperlukan. Semakin panjang jenjang saluran distribusi yang diperlukan berarti semakin tidak langsung penyampaian barang dari produsen ke konsumen. Dengan kata lain, jenjang atau panjangnya rantai saluran distribusi menunjukkan tingkatan tidak langsungnya sistem saluran distribusi. Rantai saluran distribusi untuk produk konsumen dapat dilihat pada Gambar 2.3. dibawah ini :



Gambar 2.3. Saluran Distribusi Produk Konsumen
(Budiarto, 2007 : 104)

(0) Rantai saluran distribusi jenjang 0 : dipergunakan terutama untuk jasa, peralatan rumah tangga, kosmetika, minumam kesehatan

(1) Rantai saluran distribusi jenjang 1 : dipergunakan umumnya untuk produk-produk pakaian, mebel, peralatan rumah tangga

(2)(3) Rantaian saluran distribusi jenjang 2 atau 3 : dipergunakan untuk barang-barang kebutuhan sehari-hari, dan obat-obatan

(Budiarto, 2007 : 104)

2.2.5. Strategi Distribusi

Strategi *Retail Distribution* dibagi menjadi tiga jenis, yaitu :

1. Strategi Distribusi Intensif

   Distribusi intensif adalah strategi distribusi yang menempatkan produk dagangan di banyak retailer atau pengecer dan distributor di berbagai tempat. Teknik ini sangat cocok digunakan untuk produk atau barang kebutuhan pokok sehari-hari yang memiliki permintaan dan tingkat konsumsi yang tinggi, seperti sikat gigi, odol, sabun, ditergen, dan lain sebagainya.

2. Strategi Distribusi Selektif

Distribusi selektif adalah model distribusi yang menyalurkan produk barang atau jasa di daerah pemasaran tertentu dengan memilih beberapa distributor atau pengecer saja. Di antara distributor atau pengecer akan terdapat suatu persaingan untuk merebut konsumen dengan cara, teknik dan strategi masing-masing. Contoh saluran distribusi selektif adalah : produk elektronik, produk kendaraan bermotor, sepeda, pakaian, buku, dan lain sebagainya.

3. Strategi Distribusi Eksklusif

   Distribusi eksklusif memberikan hak distribusi suatu produk kepada satu atau dua distributor atau pengecer saja pada suatu area daerah. Barang atau jasa yang ditawarkan oleh jenis distribusi eksklusif adalah barang-barang dengan kualitas dan harga yang tinggi dengan jumlah konsumen yang terbatas, seperti : *showroom* mobil, *factory outlet*, restoran waralaba, *mini market*, *supermarket*, *hipermarket*, dan lain-lain.

(Buraruallo, 2014 : 50)

2.3. *Travelling Salesman Problem* (TSP)

Traveling Salesman Problem (TSP) dikenal sebagai salah satu masalah optimasi yang banyak menarik perhatian para ahli matematika dan khususnya ilmuwan komputer karena TSP mudah didefinisikan dan begitu sulit untuk diselesaikan. Masalah TSP dinyatakan dapat dinyatakan dimana seseorang ingin mengunjungi ke sejumlah kota, dimana rangkaian kota-kota yang dikunjungi harus membentuk suatu jalur sedemikian rupa sehingga kota-kota tersebut hanya boleh dilewati tepat satu kali dan kemudian kembali lagi ke kota awal. Tujuan dari masalah TSP ini adalah untuk mencari rute atau jarak terpendek.

Penyelesaian eksak untuk masalah TSP ini mengharuskan perhitungan terhadap semua kemungkinan rute yang dapat diperoleh, kemudian memilih salah satu rute yang terpendek. Untuk itu jika terdapat n kota yang harus di kunjungin, maka diperlukanan proses pencarian sebanyak $n!/2n$ rute. Dengan cara ini komputasi yang harus dilakukan akan meningkat seiring bertambahnya jumlah kota yang harus dilalui.

TSP dikenal sebagai permasalahan yang bersifat *Nondeterministic Polynomial-Hard* (*NP-Hard*). Hal inilah yang menyebabkan penyelesaian secara eksak sulit dilakukan. Permasalahan dari penulisan ini adalah bagaimana mendapatkan penyelesaian terbaik dalam hal ini jalur terpendek dari TSP.

Adapun *objective function* untuk jarak terpendek pada penelitian ini yaitu:

$$T_c = \sum_{k=1}^{d} C_{ij}$$

Dimana :

T_c = *objective function*

C_{ij} = jarak dari kota-i ke kota-j

i = kota awal

j = kota tujuan

Dengan *Hard Constraint* sebagai berikut :

a. Kota-kota yang ada hanya dikunjungi sekali

$$X_{ij} + X_{ji} \leq 1 \quad for\ all\ i, j$$

b. Jarak dari kota i ke kota j ≠ jarak dari kota j ke kota i

$$d_{ij} \neq d_{ji}$$

c. *Non negative constraint*

$$X_{ij} \geq 0$$

d. Semua kota harus dikunjungi

$$\sum_{i}^{n} \sum_{j}^{n} X_{ij} = 1 \quad for\ i\ and\ j$$

Adapun *Soft Constrain* pada penelitian ini adalah jarak terpendek yang didapatkan.

(Amri, Nababan, & Syahputra, 2012 : 8-9)

2.4. Metaheuristik

Metaheuristik adalah generasi baru pengembangan dari algoritma heuristik. Metaheuristik dikembangkan karena tingginya stingkat kompleksitas masalah kombinatorial pada dunia nyata akibat makin luasnya dimensi kendala, sehingga pendekatan eksak sudah tidak mungkin digunakan.

Filosofi metaheuristik terbagi menjadi dua, yaitu sebagai berikut.

1. Perluasan dari algoritma *Local Search* (*Trajectory Method*)

   Tujuannya adalah untuk menghindari terjadinya *local minima*, sehingga dapat melakukan eksplorasi pada ruang pencarian sambil terus mencari *local minima* yang lebih baik. *Trajectory method* menggunakan satu atau lebih struktur *neoghbourhood*. Contoh: *Tabu Search, Iterated Local Search, Variable Neighboudhood Search, GRASP,* dan *Simulated Annealing.*

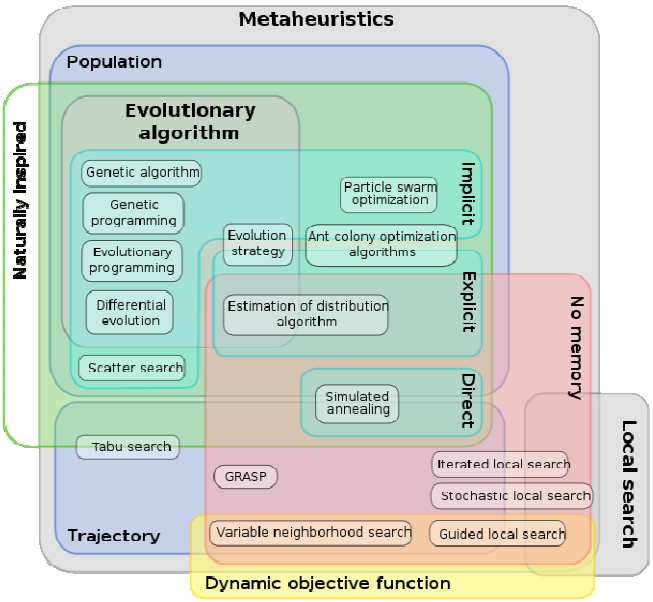
2. Menggunakan komponen pembelajaran (*Learning Population-Based Method*)

   Metode ini secara implisit maupun eksplisit mencoba mempejari korelasi atar variabel keputusan untuk mengidentifikasi area yang berkualitas pada ruang pencarian. Metode ini melakukan *sampling* maya pada ruang pencarian. Contoh: *Ant Colony Optimization, Particle Swarm Optimization,* dan *Evolutionary Computation* (*Genetic Algorithm*).

Klasifikasi pencarian pada metaheuristik adalah sebagai berikut :

1. Inspirasi oleh alam atau inspirasi oleh non alam.
2. Berbasis populasi atau berbasis satu titik.
3. *Objective function* yang dinamis atau statis.
4. Satu atau lebih struktur *neighbourhood*.
5. Penggunaan kapasitas memori yang besar atau sedikit.

Berikut adalah klasifikasi dari algoritma Metaheuristics dapat dilihat pada Gambar 2.4 di bawah ini :



Gambar 2.4 Klasifikasi *Metaheuristics Algorithm*
(Dréo & Candan, 2011)

Dari sekian banyak algoritma yang termasuk dari jenis ini, beberapa algoritma yang sering digunakan untuk menyelesaikan masalah *Travelling Salesman Problem* (TSP) adalah :

1. *Ant Colony Optimization* (ACO)

   Metaheuristik ini terinspirasi dari metafor alami, yaitu komunikasi dan kerja sama antarsemut untuk mencari jalur terpendek dari sarang semut hingga ke makanan. Media komunikasi ini berupa komponen kimia yang disebut feromon, yang disebar di darat. Ketika ada semut yang tertinggal, maka alat pendeteksi feromon pada semut dapat mendeteksi jalur feromon dan mengikutinya dengan beberapa kemungkinan dan diperkuat dengan feromon semut tersebut. Maka, probabilitas semut akan mengikuti jalur feromon yang telah terbentuk akan semakin bertambah dengan banyaknya semut yang kemudian mengikutinya.

   Hal ini kemudian mengarah pada pencarian rute terpendek karena akumulasi feromon yang cepat pada jalur yang terbentuk. Karena itu, dibentuk *metaphor* buatan, di mana sejumlah semut-semut buatan mencari solusi secara acak dalam suatu siklus. Setiap semut memilih elemen selanjutnya yang akan dihubungkan dengan elemen sementara yang telah terbentuk berdasarkan evaluasi heuristik dan jumlah feromon pada elemen tersebut. Feromon ini direpresentasikan dengan bobot.

Feromon menggambarkan memori dari sistem. Feromon ini juga berhubungan dengan solusi elemen baik yang sebelumnya telah terbentuk oleh semut-semut. *ACO* banyak digunakan dalam menyelesaikan *TSP*, di mana siklus *Hamilton* terpendek dapat ditemukan dalam graf lengkap.

2. *Greedy Randomized Adaptive Search Procedure* (GRASP)

   Ide dasar dari *GRASP* adalah menggunakan heuristik pengkonstruksian *randomized greedy* dalam prosedur banyak awalan untuk menghasilkan solusi yang bervariasi. Pada setiap langkah heuristik pengkonstruksian *randomized greedy*, elemen- elemen yang belum tergabung dalam solusi sementara dievaluasi dengan fungsi heuristik, dan elemen terbaik disimpan dalam daftar kandidat terbatas. Kemudian, satu elemen dipilih dari daftar tersebut dan digabungkan dengan solusi sementara yang telah terbentuk. Ketika proses pengkonstruksian telah selesai, solusi yang dihasilkan kemudian dikembangkan dengan pencarian lokal hingga diperoleh hasil terbaik.

3. *Simulated Annealing* (SA)

   Metaheuristik ini adalah metode pencarian lokal yang acak, di mana modifikasi untuk solusi sementara, yang mengarah pada hasil yang lebih baik, dapat diterima dengan beberapa kemungkinan. Mekanisme ini dapat mengantisipasi dari *local optima* yang buruk. *SA* berasal dari analogi proses penguatan fisik tanah dengan energi yang rendah.

   Pada proses pengentalan materi, penguatan atau *annealing* adalah sebuah proses mencairkan zat padat dengan cara menaikkan suhu. Kejadian ini kemudian diikuti dengan penurunan suhu secara bertingkat untuk mengembalikan kondisi zat padat dari energi rendah. Penguatan dilakukan perlahan-lahan berdasarkan tingkatan suhu, di mana suhu berada pada tingkat tertentu dalam waktu cukup lama supaya tercapai keseimbangan. Hal ini dilakukan supaya penguatan mengarah pada pembentukan struktur yang *solid* sehingga dapat berasosiasi dengan zat pada pada energi rendah.

   Pada TSP, himpunan rute berhubungan dengan keadaan, sedangkan

solusi biaya berhubungan dengan energi. Pada setiap iterasi, solusi sementara dimodifikasi dengan memilih modifikasi acak berdasarkan kelas modifikasi tertentu yang mendefinisikan struktur keterikatan lingkungan atau *neighbourhood*. Jika solusi terbaru lebih baik daripada yang sementara sudah terbentuk, maka solusi itu yang kemudian terpilih menjadi solusi sementara. Sebaliknya, solusi terbaru dapat diterima berdasarkan kriteria tertentu di mana modifikasi diperoblehkan jika parameter, yang disebut suhu, bersifat tinggi, tetapi peningkatan biaya rendah.

Selama proses berlangsung, parameter suhu diturunkan berdasarkan jadwal pendinginan, dan iterasi dilakukan pada setiap tingkatan suhu. Ketika suhu sudah cukup rendah, hanya modifikasi yang berkembang yang diperbolehkan dan metode berhenti pada *local optima*.

4. *Tabu Search* (TS)

   Seperti *SA*, *TS* adalah metaheuristik yang berdasarkan pencarian lokal, di mana pada setiap iterasi, apabila ditemukan solusi baru yang lebih baik dari yang ada sementara, maka solusi itu kemudian menjadi solusi sementara yang ada, meskipun solusi ini mengarah pada kenaikan biaya. Melalui mekanisme ini, maka metode ini akan terhindar dari *local optima*. Sebuah memori jangka pendek, yang dikenal sebagai daftar *tabu*, menyimpan solusi sementara terakhir untuk menghindari siklus jangka pendek. Pencarian berhenti hingga iterasi telah mencapai angka tertentu meskipun belum dihasilkan lagi solusi yang lebih baik dari solusi yang telah ada

5. *Variable Neighbourhood Search* (VNS)

   VNS adalah metaheuristik pencarian lokal lainnya yang mengambil berbagai kelas transformasi yang lain, atau *neighbourhood* untuk menghindari *local optima*. Ketika terjadi *local optima*, dengan memberitahukan pada *neighbourhood* yang sudah terbentuk, *neighbourhood* lain dipilih dan digunakan pada iterasi berikutnya.

   Misalkan diberikan himpunan *neighbourhood* (biasanya berangkai), dan sebuah solusi dihasilkan secara acak dari *neighbourhood* pertama

dari solusi sementara, yang berasal dari keturunan sementara yang telah terbentuk (dapat pula berasal dari *neighbourhood* yang memiliki struktur yang berbeda). Jika *local optima* yang terjadi tidak lebih baik dari solusi sementara, maka prosedur diulang pada *neighbourhood* selanjutnya dalam struktur berangkai. Pencarian ini diulang dari *neighbourhood* pertama ketika ada solusi yang lebih baik dari atau ketika semua *neighbourhood* telah dicoba.

(Felia, 2008 : 33-37)

6. *Artificial Bee Colony* (ABC)

Artificial Bee Colony (ABC) dikembangkan oleh Karaboga, terinspirasi dari kebiasaan *forgaging* dan *waggle dance* koloni lebah. Hasil eksperimen karya sastra menunjukkan bahwa Artificial Bee Colony (ABC) lebih baik dari algoritma lain seperti *Genetic Algorithm* (GA), *Particle Swarm Optimization* (PSO), dan *Differential Evolution* (DE).

(Kirain & Gündüz, 2012 : 6107-6121)

2.5. Algoritma *Artificial Bee Colony* (ABC)

2.5.1. Koloni Lebah

Lebah merupakan serangga yang terpadu. Kemampuan bertahan hidup seluruh koloni lebah tergantung dari tiap individu lebah. Lebah menggunakan tugas-tugas yang sistematik yang diantaranya bertujuan menjaga eksistensi koloninya. Mereka melakukan bermacam-macan pekerjaan seperti *forgaging*, reproduksi, membangun sarang, dan lainnya. Dari pekerjaan-pekerjaan ini, *forgaging* merupakan aktivitas yang penting untuk menjaga pasokan makanan pada sarang lebah.

Koloni dari lebah mampu menempuh jarak yang cukup jauh (lebih dari 10 km) dan mampu untuk bergerak ke segala arah secara simultan untuk memeriksa lebih dari satu sumber makanan. Koloni ini bekerja dengan mengirimkan lebah pengintainya ke ladang yang banyak sumber makanannya. Proses eksplorasi (*forgaging*) dimulai dari mengirimkan lebah pengintai untuk mencari bunga yang berpeluang memiliki madu yang banyak, lebah pengintai tersebut bergerak secara acak dari satu tangkai bunga ke tangkai bunga yang lainnya. Pada saat musim panen,

koloni tetap melakukan ekplorasinya, dan tetap mempertahankan populasi dari lebah pengintai. Ketika lebah pengintai kembali ke sarang dan menemukan bunga dengan kadar madu yang dianggap cukup tinggi daripada yang diharapkan, akan mengambil nektarnya sebagai sampel lalu melakukan tarian untuk memberikan lokasi bunga tersebut, yang disebut dengan ”*waggle dance*”. Tarian ini sangat penting bagi komunikasi dalam koloni, dan menyimpan akan tiga informasi mengenai bunga yang lebah tersebut temukan, yaitu: arah di mana dapat menemukan bunga tersebut, jarak yang harus ditempuh dari sarang lebah ke bunga, dan kualitas akan madunya. Informasi ini membantu koloni untuk mengirimkan lebah yang lainnya ke bunga tersebut tanpa menggunakan petunjuk atau peta.

Tiap lebah memiliki pengetahuan akan lingkungan sekitarnya dari ”*waggle dance*” serta terdapat *pheromone* yang juga membantu para lebah untuk mencari lokasi bunga tersebut. Dan tarian ini juga digunakan untuk mengevaluasi keuntungan dari bunga yang lainnya, yang berdasarkan energi yang diperlukan, dan jumlah hasil yang mereka bisa dapatkan. Setelah melakukan tarian, lebah pengintai tersebut akan kembali ke bunga yang telah ditemukan dan diikuti oleh beberapa lebah lainnya, hal ini diperuntukkan untuk mencari bunga berkualitas yang lainnya. Sehingga dengan ini koloni mampu untuk mengumpulkan makanan dengan cepat dan efisien.

(Sugioko, 2013 : 113-114)

2.5.2. Algoritma *Artificial Bee Colony* (ABC)

2.5.2.1. Pengertian Algoritma *Artificial Bee Colony* (ABC)

Pengertian algoritma *Artificial Bee Colony* (ABC), antara lain :

1. Sugioko (2013 : 113) mengatakan bahwa “Algoritma *Artificial Bee Colony* merupakan algoritma untuk optimalisasi yang terinspirasi dari kebiasaan eksplorasi lebah (*forgaging*) untuk mencari solusi yang optimal.”
2. Amri, Nababan, & Syahputra (2012 : 9) mengatakan bahwa “*Artificial Bee Colony Algorithm* adalah pendekatan *population-based metaheuristic* yang diusulkan oleh Karaboga

dan Basturk. Pendekatan ini terinsiprasi oleh perilaku cerdas kawanan lebah madu mencari makanan."

Swarm intelligent merupakan sebuah metode penyelesaian masalah yang memanfaatkan prilaku sekumpulan agen yang saling bekerja sama. Khususnya *swarn intelligent* pada algoritma lebah (*Algorithm Bee Colony)* terinspirasi dari perilaku sosial koloni lebah dimana seekor lebah bisa menjangkau sumber makanan sekaligus mengingat letak, arah dan jarak dari sumber makanan. Sekembalinya dari pencarian sumber makanan maka seekor lebah akan melakukan *waggle dance*. *Waggle dance* merupakan alat komunikasi yang dilakukan oleh koloni lebah.
(Andri, Suyandi & WinWin, 2013 : 62)

2.5.2.2. Kelompok Lebah pada Algoritma *Artificial Bee Colony* (ABC)

Pada model algoritma *Artificial Bee Colony* (ABC) memiliki tiga kelompok lebah, yaitu:

1. *Employed Bee* : berhubungan dengan sumber makanan tertentu.
2. *Onlookers Bee*
   a. Menyaksikan tarian lebah yang digunakan dalam sarang untuk memilih sumber makanan mencari sumber makanan secara acak
   b. Merupakan *unemployed bee*

3. *Scouts Bee*
   a. Mencari sumber makanan secara acak
   b. Merupakan *unemployed bee*

Awalnya *scout bee* menemukan posisi semua sumber makanan, setelah itu tugas dari *employed bee* dimulai. Sebuah *employed bee* buatan secara probabilitas memperoleh beberapa modifikasi pada posisi dalam memori untuk menargetkan sumber makanan baru dan menemukan jumlah nektar atau nilai *fitness* dari

sumber baru. Kemudian, *scout bee* mengevaluasi informasi yang diambil dari semua *employed bee* buatan dan memilih sumber makanan akhir dengan nilai probabilitas tertinggi terkait dengan jumlah nektar tersebut. Jika nilai *fitness* yang baru lebih tinggi dari yang sebelumnya, lebah itu akan melupakan yang lama dan menghafal posisi baru. Hal ini disebut sebagai *greedy selection*. Kemudian *employed bee* yang sumber makanan telah habis menjadi *scout bee* untuk mencari sumber makanan lebih lanjut sekali lagi.

Menurut Bhagade and Puranik dalam *Artificial Bee Colony*, solusi yang merupakan sumber makanan dan kuantitas madu dari sumber makanan sesuai dengan kebugaran dari solusi terkait. Jumlah *employed* dan *scout bee* adalah sama, dan jumlah ini sama dengan jumlah sumber makanan. *Employed bee* yang solusinya tidak dapat ditingkatkan melalui sejumlah percobaan, yang ditentukan oleh pemakai dari *Artificial Bee Colony* disebut *limit*, kemudian *employed bee* tersebut menjadi *scout bee* dan solusi yang dihasilkan diabaikan/ditinggalkan. Bhagade dan Puranik juga menjelaskan dalam algoritma ini, *employed bee* menghasilkan modifikasi dalam posisi (yaitu solusi) dalam memori dan memeriksa jumlah nektar (*fitness*) dari sumber (solusi). *Employed bee* kemudian mengevaluasi informasi nektar ini dan kemudian memilih sumber makanan dengan probabilitas yang berkaitan dengan nilai *fitness*-nya.

(Amri, Nababan & Syahputra, 2012 : 9)

2.5.2.3. Tahapan Algoritma *Artificial Bee Colony* (ABC)

Tahapan-tahapan *Artificial Bee Colony* (ABC) untuk *Travelling Salesman Problem* (TSP), sebagai berikut :

1. Pembentukan Rute

   Untuk pembentukan rute dalam masalah *Travelling Salesman Problem* digunakan metode *Nearest Neighboor* . Rute-rute yang terbentuk menjadi kumpulan sumber makanan dimana sumber-sumber makanan tersebut akan dijadikan acuan dalam

penyelesaian *Travelling Salesman Problem* menggunakan *Artificial Bee Colony* (ABC).

2. Inisialisasi

Dalam tahap inisialisasi ini semua rute yang sudah terbentuk dianggap sebagai sumber makanan dan akan diseleksi oleh para lebah untuk menentukan rute mana yang terbaik dan merupakan solusi dari permasalahan. Pada tahap ini diberikan nilai percobaan dari setiap kemungkinan solusi sama dengan 0. Proses inisialisasi dari kemungkinan solusi (sumber makanan) dilakukan secara *random* dengan persamaan sebagai berikut :

$$x_{ij} = x_{jmin} + rand\,(0,1).(x_{jmax} - x_{jmin})$$

Dimana :

x_{ij} = inisialisasi kemungkinan solusi ke-i dengan parameter ke-j

x_{jmin} = nilai kemungkinan solusi terkecil berdasarkan parameter j

x_{jmax} = nilai kemungkinan solusi terbesar berdasarkan parameter j

$rand\,(0,1)$ = nilai *random* antara 0 sampai 1

$i = 1 \ldots SN$, SN adalah jumlah kemungkinan solusi (sumber makanan)

$j = 1 \ldots D$, D adalah jumlah parameter yang digunakan

3. Tahap Lebah Pekerja (*Employed Bee Phase*)

Setelah penginisialisasian selesai para lebah pekerja bertugas untuk memperluas nilai dari setiap kemungkinan solusi yang ada dengan menggunakan persamaan sebagai berikut :

$$v_{ij} = x_{ij} + \phi_{ij}.(x_{ij} - x_{kj})$$

Dimana :

v_{ij} =nilai perluasan kemungkinan solusi ke-i dengan parameter j

x_{ij} = nilai kemungkinan solusi ke-i dengan parameter ke-j

$i = 1 \ldots SN$, SN adalah jumlah kemungkinan solusi (sumber makanan)

$j = 1 \ldots D$, D adalah jumlah parameter yang digunakan

$k = 1 \ldots SN$, SN adalah jumlah parameter yang digunakan

ϕ_{ij} = bilangan real *random* antara [-1,1]

Setelah setiap kemungkinan solusi diperluas, akan diaplikasikan *greedy selection* antara nilai kemungkinan solusi x_i dengan nilai baru hasil perluasan yaitu v_i . Jika nilai v_i lebih kecil dari nilai x_i maka nilai v_i tersebut akan dianggap sama dengan nilai x_i dan nilai percobaan tetap bernilai 0. Jika tidak, nilai x_i yang disimpan dan nilai percobaan ke-i ditambah dengan 1. Proses ini berulang sampai jumlah perluasan sama dengan jumlah sumber makanan.

4. Tahap evaluasi populasi

Setelah setiap kemungkinan solusi diperluas dan dibandingkan dengan nilai awal inisialisasi, tahap berikutnya adalah penghitungan kualitas dari setiap kemungkinan solusi menggunakan fungsi *fitness* sebagai berikut :

$$fitness\ (x_i)\begin{cases}\dfrac{1}{(1+f(x_i))}, f(x_i) \geq 0\\ 1+|f(x_i)|, f(x_i) < 0\end{cases}$$

Dimana :

$i = 1 \ldots SN$, SN adalah jumlah kemungkinan solusi (sumber makanan)

5. Tahap Lebah Penjaga (*Onlooker Bee Phase*)

Setelah semua perluasan untuk setiap kemungkinan solusi di tahap lebah pekerja terpenuhi dan telah dihitung masing – masing nilai *probability*, informasi yang diperoleh lebah pekerja akan ditunjukkan kepada lebah penjaga. Lebah penjaga yang menerima informasi tersebut bertugas untuk menghitung nilai *probability* dari setiap kemungkinan solusi yang ada dengan menggunakan rumus sebagai berikut :

$$p_i = \frac{fitness_i}{\sum_{i=1}^{SN} fitness_i}$$

Dimana :

$fitness_i$ = nilai *fitness* solusi ke-i

$\sum_{i=1}^{SN} fitness_i$ = jumlah dari nilai *fitness* ke-i sampai SN

Setelah nilai *probability* dari setiap kemungkinan solusi dihitung, lebah penjaga akan memilih kemungkinan solusi mana yang akan ditelusuri lebih lanjut oleh Lebah Pengintai dengan menggunakan metode *roulette-wheel.*

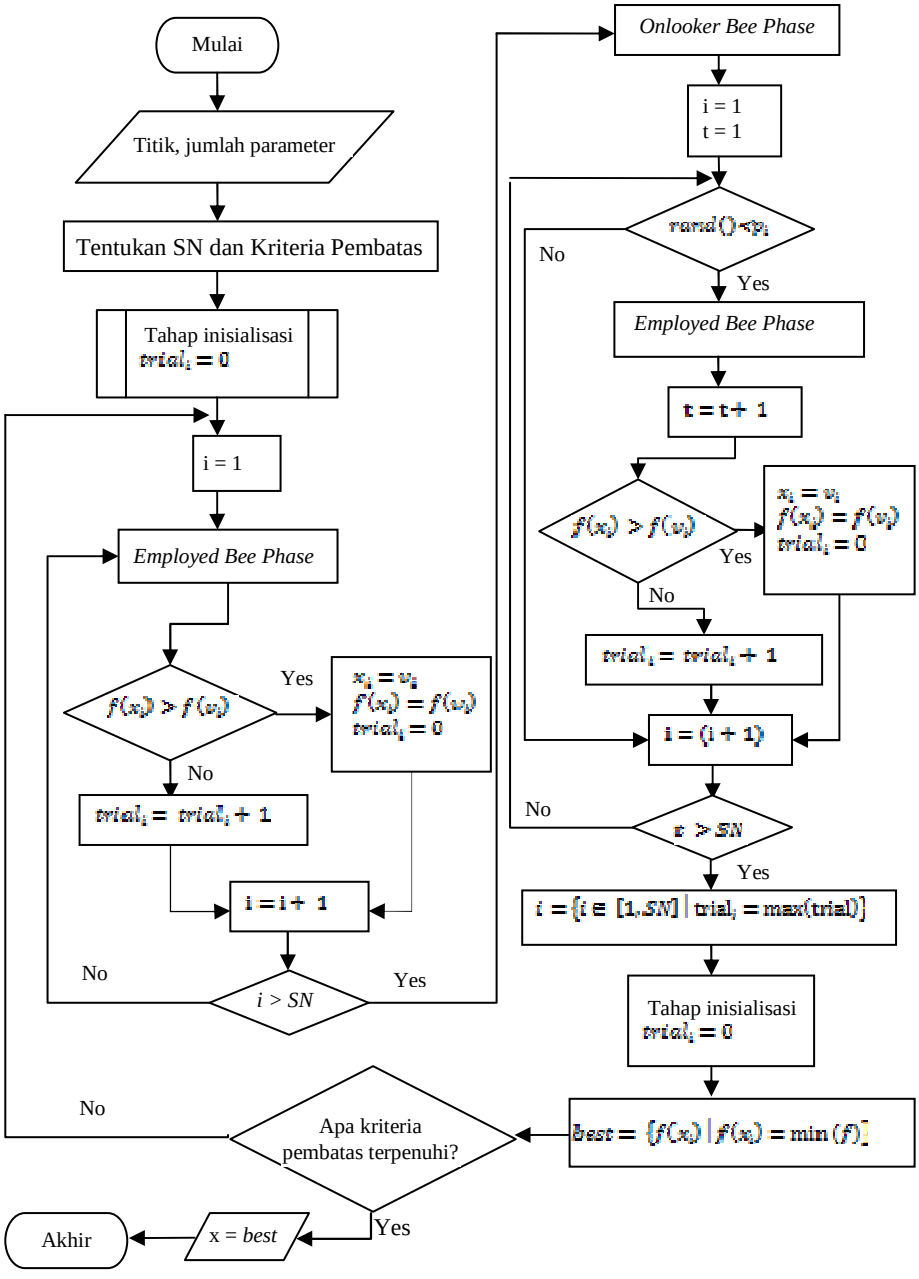
6. Tahap Lebah Pengintai (*Scout Bee Phase*)

   Untuk mengaplikasi metode *roullete-wheel* , dipilih bilangan real secara *random* antara [0,1] untuk setiap kemungkinan solusi. Jika nilai p_i lebih besar dari bilangan *random* yang ditentukan, maka lebah penjaga akan menugaskan lebah pengintai untuk memperluas kembali kemungkinan solusi yang terpilih tersebut sesuai dengan tahap lebah pekerja sebelumnya. Setelah kemungkinan solusi terpilih diperluas, akan diaplikasikan *greedy selection* antara nilai kemungkinan solusi x_i dengan nilai baru hasil perluasan yaitu v_i. Jika nilai *vi* lebih kecil dari nilai x_i maka nilai v_i tersebut akan dianggap sama dengan nilai x_i. Jika tidak, nilai v_i yang disimpan dan nilai percobaan ke-i ditambah dengan 1. Proses ini berulang sampai jumlah perluasan sesuai dengan kemungkinan solusi. Setelah semua kemungkinan solusi memiliki nilai percobaan, akan dipilih kemungkinan solusi dengan nilai percobaan maksimum dan kemungkinan solusi tersebut menjadi pilihan solusi terbaik. Proses kembali menuju tahap lebah pekerja (*Employed Bee Phase*) dan berulang sampai kriteria pembatas terpenuhi, dimana kriteria pembatas yang dimaksud adalah jumlah lebah dalam *colony*.

(Pratama, Purwanto, & Yasin, 2012 : 2)

2.5.2.4. *Flowchart* Algoritma *Artificial Bee Colony* (ABC)

Untuk mempermudah pemahaman tentang *Artificial Bee Colony* (ABC), berikut *flowchart* dari tahapan – tahapan *Artificial Bee Colony* (ABC) yang dapat dilihat pada Gambar 2.5 :



Gambar 2.5. *Flowchart Artificial Bee Colony* (ABC) untuk TSP (Pratama, Purwanto, & Yasin, 2012 : 2)

2.6. Struktur Data *Graph*

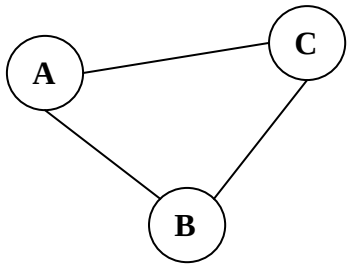
Dalam bidang ilmu komputer, sebuah *graph* dapat dinyatakan sebagai sebuah struktur data, atau secara spesifik dinamakan sebagai ADT (*Abstract Data*

Type) yang terdiri dari kumpulan simpul dan sisi yang membangun hubungan antar simpul. Konsep ADT *graph* ini merupakan turunan konsep *graph* dari bidang kajian matematika.

Secara umum terdapat dua macam representasi dari struktur data *graph* yang dapat diimplementasi, yaitu :

1. *Adjacency list*

   Dalam teori *graph*, *adjacency list* merupakan bentuk representasi dari seluruh sisi atau busur dalam suatu *graph* sebagai suatu senarai. Simpul-simpul yang dihubungkan sisi atau busur tersebut dinyatakan sebagai simpul yang saling terkait. Dalam implementasinya, *hash table* digunakan untuk menghubungkan sebuah simpul dengan senarai berisi simpul-simpul yang saling terkait tersebut. Dapat dilihat pada Gambar 2.6 di bawah ini :



Gambar 2.6. *Undirected Cyclic Graph*

Graph pada Gambar 2.6 dapat dideskripsikan sebagai senarai {a,b},{a,c},{b,c}. Representasi *adjacency list* dapat digambarkan melalui Tabel 2.1 di bawah ini :

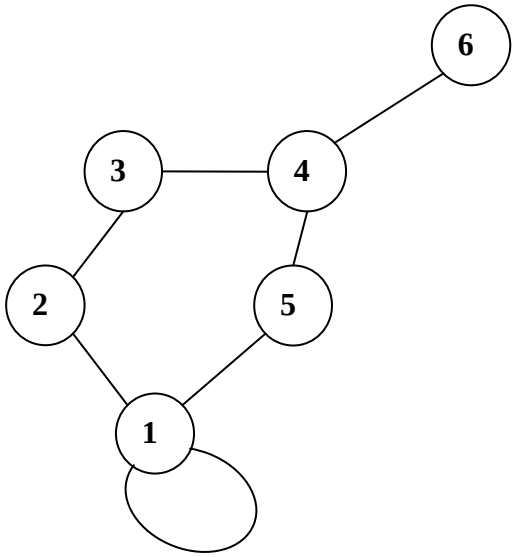
Tabel 2.1. Representasi *Adjacency List*

Vertex	Adjacency	Array of Adjacent Vertices
a	adjacent to	b,c
b	adjacent to	a,c
c	adjacent to	a,b

2. Adjacency Matrix

   Adjacency Matrix merupakan representasi matriks $n \times n$ yang menyatakan hubungan antar simpul dalam suatu *graph*. Kolom dan

baris dari matriks ini merepresentasikan simpul-simpul, dan nilai entri dalam matriks ini menyatakan hubungan antar simpul, apakah terdapat sisi yang menghubungkan kedua simpul tersebut. Pada sebuah matriks $n \times n$, entri non-diagonal a_{ij} merepresentasikan sisi dari simpul i dan simpul j. Sedangkan entri diagonal a_{ii} menyatakan sisi kalang (loop) pada simpul i. Dapat dilihat pada Gambar 2.7 dan Gambar 2.8 di bawah ini:



Gambar 2.7. *Graph* Tak Berarah Berlabel

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Gambar 2.8. *Adjacency Matrix*

(Wirdasari, 2011 : 27-28)

2.7. Interaksi Manusia dan Komputer

2.7.1. Pengertian Interaksi Manusia dan Komputer

Interaksi manusia dan komputer adalah disiplin ilmu yang diperkenalkan sebagai susunan dari sebagian psikologi dan sebagian *engineering*, untuk menawarkan solusi untuk suatu masalah. Tujuan dasar dari interaksi manusia dan komputer adalah peningkatan interaksi antara komputer dan pengguna.

Alasan menggunakan interaksi manusia dan komputer adalah :

a. Komputer digunakan dimana-mana
b. Fungsi penting dari komputer di beberapa daerah
c. Komputer memperkaya kehidupan manusia
d. Sering ditemukannya komponen yang baru
e. Meningkatkan daya saing
f. Meningkatkan kinerja alokasi sumber daya

(Meena & Sivakumar, 2015 : 2, 4)

2.7.2. Delapan Aturan Emas Desain *User Interface*

Delapan prinsip, yang sering disebut "*Golden Rules*", yang diterapkan di sistem interaktif saat ini. Delapan aturan emas desain *user interface* sebagai berikut :

1. Mengupayakan konsistensi (*strive for consistency*)

   Urutan konsekuensi tindakan harus diperlukan dalam situasi yang sama. Digunakan dalam *prompts*, menu-menu, dan layar bantuan dengan konsistensi warna, *layout*, kapitalisasi, ukuran, dan seterusnya harus diterapkan secara menyeluruh dan konsisten. Pengecualian, seperti konfirmasi yang diperlukan dari perintah *delete* atau kesalahan *password*, harus lengkap dan terbatas jumlahnya.

2. Memenuhi kegunaan secara umum (*Cater to universal usability*)

   Mengenali kebutuhan pengguna yang beraneka ragam dan merancang untuk plastisitas (kemampuan untuk menyesuaikan diri dengan lingkungan yang baru), memfasilitasi perubahan konten.

3. Menyediakan umpan balik yang informatif (*Offer informative feedback*)

   Untuk setiap tindakan pengguna, harus ada umpan balik dari sistem. Untuk tindakan yang sering dilakukan dan tidak terlalu penting, dapat diberikan umpan balik yang sederhana. Sedangkan untuk tindakan yang jarang dilakukan dan penting, umpan balik dari sistem tersebut harus lebih substansial.

4. Merancang dialog untuk menghasilkan suatu penutupan (*design dialog to yield closure*)

   Urutan tindakan harus diatur ke dalam suatu kelompok dengan bagian awal, tengah dan akhir. Umpan balik yang informatif pada penyelesaian sekelompok tindakan memberikan operator kepuasan prestasi, rasa nyaman, sebuah sinyal untuk menghentikan rencana kontingensi dari pikiran mereka, dan indikator untuk mempersiapkan sekelompok tindakan berikutnya. Sebagai contoh, situs *web e-commerce* memindahkan pengguna dari memilih produk ke kasir, berakhir dengan halaman konfirmasi yang jelas yang melengkapi transaksi.

5. Mencegah kesalahan-kesalahan (*prevent errors*)

   Sedapat mungkin merancang sistem sehingga pengguna tidak dapat melakukan kesalahan yang serius. Jika pengguna melakukan kesalahan, *interface* harus mendeteksi kesalahan dan memberikan instruksi sederhana, konstruktif, dan instruksi khusus untuk perbaikan.

6. Mudah kembali ke tindakan sebelumnya (*Permit easy reversal of actions*)

   Sedapat mungkin, tindakan harus dapat kembali ke tindakan sebelumnya. Dengan mudahnya kembali ke tindakan sebelumnya, dapat mengurangi kecemasan pengguna, karena pengguna tahu bahwa kesalahan dapat dibatalkan dan mendorong pengguna untuk mengeksplorasi pilihan yang asing bagi pengguna.

7. Mendukung tempat pengendalian internal (*Support internal locus of control*)

   Pengguna ingin menjadi pengontrol sistem dan sistem akan merespon tindakan mereka. Mereka terganggu oleh urutan entri data yang membosankan, kesulitan memperoleh informasi yang diperlukan, dan ketidakmampuan untuk menghasilkan hasil yang diinginkan.

8. Mengurangi beban ingatan jangka pendek (*Reduce short-term memory load*)

   Keterbatasan kemampuan manusia untuk memproses informasi dalam ingatan jangka pendek mengharuskan *designers* menghindari

interface dimana pengguna harus mengingat informasi dari satu layar dan kemudian menggunakan informasi itu di layar lain.

(Shneiderman & Plaisant, 2010 : 88-89)

2.7.3. Evaluasi *Interface*

ISO 9241 standar *Ergonomics of Human-System Interaction* (ISO, 2008) berfokus pada tujuan efektivitas, efisiensi, dan kepuasan.

Berikut lima kategori yang dijadikan ukuran evaluasi antarmuka ini, antara lain :

1. Waktu untuk belajar (*Time to learn*)

   Berapa lama waktu yang diperlukan pengguna untuk belajar bagaimana cara menggunakan perintah-perintah yang digunakan untuk mengerjakan tugas.
2. Kecepatan kinerja (*speed of performance*)

   Berapa lama waktu yang diperlukan untuk menyelesaikan tugas.
3. Tingkat kesalahan pengguna (*Rate of errors by users*)

   Berapa banyak dan apa jenis kesalahan yang dilakukan pengguna dalam melaksanakan tugas? Meskipun saat membuat dan memperbaiki kesalahan dapat dimasukkan ke dalam kecepatan kinerja, penanganan kesalahan adalah suatu komponen penting dari penggunaan *interface.*
4. Daya ingat jangka panjang (*Retention over time*)

   Seberapa baik pengguna mempertahankan pengetahuan mereka setelah satu jam, satu hari, dan satu minggu. Daya ingat mungkin berhubungan erat dengan waktu belajar, dan frekuensi penggunaan memainkan peran penting.
5. Kepuasan subjektif (*Subjective satisfaction*)

   Berapa banyak pengguna menyukai berbagai aspek *interface*. Jawabannya dapat dipastikan dengan wawancara atau *survey* tertulis yang mencangkup skala kepuasan.

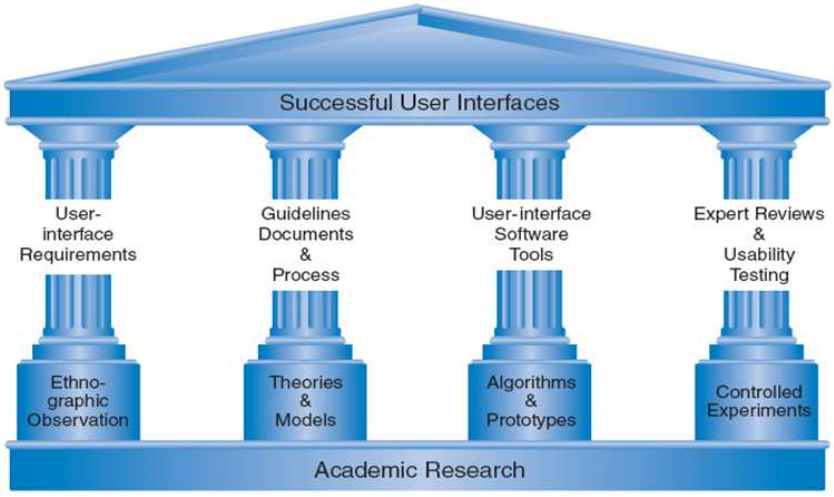
(Shneiderman & Plaisant, 2010 : 32)

2.7.4. Empat Pilar Desain *User Interface*

Keempat pilar desain *user interface* dapat membantu *user interface* mengubah ide-ide yang baik dalam kesuksesan sistem. Keempat pilar

tersebut tidak dijamin untuk bekerja secara sempurna, tetapi pengalaman menunjukkan bahwa masing-masing pilar dapat memfasilitasi pembuatan sistem yang sangat baik.

Keempat pilar desain *user interface* dapat dilihat pada Gambar 2.9 di bawah ini :



Gambar 2.9. Empat Pilar Desain *User Interface*
(Shneiderman & Plaisant, 2010 : 121)

Keempat pilar desain *user interface*, antara lain :

1. Persyaratan atau kebutuhan *user interface* (*User Interface Requirements*)

   Menentukan kebutuhan pengguna yang jelas merupakan hal utama dalam keberhasilan setiap pengembangan sistem. Tujuan dari dibutuhkannya persyaratan *user interface* ini diperuntukkan untuk komunitas pengguna dan juga mengenai tugas-tugas atau *tasks* yang pengguna bisa lakukan. Meletakkan *user interface requirement* adalah bagian dari keseluruhan persyaratan dari proses pengembangan dan manajemen.

2. Pedoman dokumen dan proses (*Guidelines Documents & Process*)

   Pada awal proses desain, *designer user interface* harus menghasilkan sekumpulan pedoman kerja. Setiap *project* memiliki kebutuhan yang berbeda-beda, namun beberapa pedoman yang harus dipertimbangkan, antara lain :

   a. *Word*, *icon*, dan *graphics*

- Peristilahan (objek dan tindakan), singkatan, dan kapitalisasi
- Sekumpulan karakter, *fonts*, ukuran *font*, dan gaya penulisan (*bold, italic, underline*)
- *Icons*, tombol, grafik, dan ketebalan garis.
- Penggunaan warna, *backgrounds*, penyorotan/*highlighting* dan kedipan/*blinking*

b. Masalah tata letak layar (*Screen layout issues*)

- *Menu selection, form fill-in*, format pada *dialog-box*
- *Wording of prompts, feedback*, dan pesan *error*.
- Justifikasi, *white space*, dan margin
- Entri data dan menampilkan format untuk item dan daftar.
- Penggunaan dan isi dari *header* dan *footer*
- Strategi untuk beradaptasi dengan tampilan yang besar dan kecil

c. Perangkat *input* dan *output* (*Input and output devices*)

- *Keyboard, display, cursor control, dan pointing device*
- *Audible sound, voice feedback, speech I/O, touch input*, dan *device spesial* lainnya.
- Waktu respon pada tugas-tugas yang berbeda
- Alternatif bagi pengguna penyandang cacat

d. *Action Sequences*

- *Direct manipulation clicking, dragging, dropping* dan *gestures*
- *Command syntax, semantics*, dan *sequences*
- *Shortcuts* dan *programmed function keys*
- Navigasi layar sentuh untuk *device*
- Penanganan kesalahan dan prosedur perbaikan

e. *Training*

- *Online help, tutorials* dan *support group*
- *Training* dan referensi materi terkait

3. Peralatan perangkat lunak *user interface* (*User Interface Software Tools*)

   Persoalan yang utama dalam merancang suatu sistem yang interaktif adalah *customer* dan *user* tidak mempunyai ide yang jelas bagaimana

sistem tersebut akan terlihat pada akhir perancangan. Karena sistem interaktif adalah baru di berbagai situasi, maka *user* tidak menyadari implikasi dari berbagai keputusan desain. Sayangnya, itu menjadi hal yang sulit, memakan biaya yang besar, dan memerlukan waktu yang lama untuk melakukan perubahan yang besar ke sistem tersebut ketika sistem itu telah diimplementasikan. Sehingga, *User Interface Software Tools* dapat menangani masalah tersebut.

4. Review dari ahli dan pengujian kegunaan (*Expert Reviews & Usability Testing*)

   Beberapa metode dalam *Expert Review* misalnya dengan melalukan tes ke calon *user* yang akan menggunakan sistem tersebut, survey dan *automated analysis tools*. Prosedur yang digunakan dalam metode tersebut berbeda-beda tergantung dengan apa yang akan dicapai dari testing tersebut, berapa *user* yang diharapkan, dan seberapa tingkat bahaya dalam implementasi sistem baru itu, serta level investasi untuk sistem itu.

(Shneiderman & Plaisant, 2010 : 120-126)

2.8. Rekayasa Perangkat Lunak

Perangkat lunak (*software*) adalah :

- Petunjuk (program komputer) yang ketika dijalankan menyediakan fitur, fungsi dan kinerja yang diinginkan.
- Struktur data yang memungkinkan program untuk memanipulasi informasi secara memadai.
- Informasi yang deskriptif di *hard copy* dan bentuk virtual yang menggambarkan operasi dan penggunaan program.

Rekayasa perangkat lunak adalah :

Penerapan sistematis, disiplin, pendekatan kuantitatif untuk pengembangan, operasi, dan pemeliharaan perangkat lunak, yaitu penerapan rekayasa perangkat lunak.

Perangkat lunak komputer meliputi program-program yang dijalankan dalam sebuah komputer dari berbagai ukuran dan arsitekturnya, konten yang disajikan sebagai program komputer yang dijalankan, dan informasi deskriptif baik *hard copy* dan bentuk virtual yang mencakup hampir semua media elektronik. Rekayasa perangkat lunak mencangkup sebuah proses, sebuah koleksi metode (*practice*) dan berbagai *tools* yang memungkinkan para professional membangun perangkat lunak komputer berkualitas tinggi. (Pressman, 2010 : 1, 4, 13)

2.8.1. Lapisan Layer dalam Rekayasa Perangkat Lunak

Rekayasa perangkat lunak dapat dikelompokkan ke dalam beberapa lapisan layer, dapat di lihat dari Gambar 2.10 di bawah ini :

Gambar 2.10. *Software Engineering Layers*
(Pressman, 2010 : 14)

Software Engineering Layer terbagi menjadi empat lapisan, yaitu :

1. Fokus Kualitas (*A Quality Focus*)

   Pada saat ingin membangun sebuah perangkat lunak, fokus pertama yang penting untuk dipikirkan adalah kualitas seperti apa yang ingin dibangun, siapa yang menjadi sasarannya, siapa penggunanya, dan lain-lain. Sistem yang akan dibuat harus komitmen dengan kualitas. Hal itu sangat mempengaruhi peningkatan keefektifan dari suatu perangkat lunak. Landasaran dasar yang mendukung rekayasa perangkat lunak adalah *quality focus* ini.

2. Proses (*Process*)

   Setelah mengetahui *quality focus* dari perangkat lunak yang akan dibangun, maka harus mengetahui bagaimana proses yang harus dijalani sehubungan dengan *quality focus* dari perangkat lunak yang

diharapkan. Proses-proses ini harus tepat dan terurut supaya tidak terjadi kesalahan ketika perangkat lunak sedang bekerja.

3. Metode (*Methods*)

   Methods menyediakan teknik bagaimana untuk membuat sebuah perangkat lunak. Memilih metode-metode yang akan digunakan agar proses model yang dibuat dapat diimplementasikan ke dalam sistem. Metode mencakup tugas-tugas yang meliputi komunikasi, analisis kebutuhan, pemodelan desain, membangun program, pengujian, dan *support*. Metode perangkat lunak bergantung pada seperangkat prinsip-prinsip dasar yang mengatur setiap bidang teknologi dan termasuk kegiatan pemodelan dan teknik deskriptif lainnya.

4. *Tools*

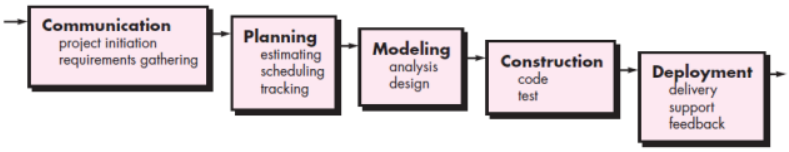
   Tools merupakan alat bantu yang digunakan dalam membuat sistem.

(Pressman, 2010 : 13-14)

2.8.2. *Waterfall Model*

Waterfall model, kadang-kadang disebut *classic life cycle*, adalah metode klasik yang bersifat sistematis, berurutan dalam pengembangan perangkat lunak.

Waterfall Model dapat dilihat pada Gambar 2.11 di bawah ini :



Gambar 2.11. *Waterfall Model*

(Pressman, 2010 : 39)

Langkah-langkah pada *waterfall model*, antara lain :

1. Komunikasi (*Communication*)

   Proses ini dimulai dengan komunikasi mengenai *platform* yang digunakan pada sistem dan melakukan analisis terhadap kebutuhan perangkat tunak. Pada tahapan ini dilakukan pengumpulan data

dengan melakukan pertemuan dengan *customer*, maupun mengumpulkan data-data tambahan.

2. Perencanaan (*Planning*)

   Setelah melakukan proses komunikasi, maka proses selanjutnya adalah proses perencanaan mengenai pembangunan perangkat lunak. Tahapan ini akan mengetahui keinginan *user* dalam pembuatan *software*, termasuk rencana yang akan dilakukan.

3. Pemodelan (*Modeling*)

   Proses ini akan merepresentasikan kebutuhan perangkat lunak ke sebuah perancangan perangkat lunak yang dapat diperkirakan sebelum membuat *coding*. Proses ini berfokus kepada perancangan struktur data, representasi *interface*, arsitektur perangkat lunak, detail (algoritma) *procedural.*

4. Penulisan kode program (*Construction*)

   Construction merupakan proses membuat kode, *coding* atau pengkodean. Pada tahap ini *programmer* akan membuat *software* sesuai dengan yang diminta oleh *user*, sehingga penggunaan komputer akan dimaksimalkan pada tahapan ini.

5. *Deployment*

   Tahap ini adalah tahapan terakhir dalam pembuatan sebuah *software*. Setelah melakukan analisis, desain, pengkodean, maka *software* yang sudah jadi akan digunakan oleh *user*, kemudian *software* yang telah dibuat harus dilakukan pemeliharaan secara berkala.

(Pressman, 2010 : 39)

2.9. *Flowchart* (Diagram Alir)

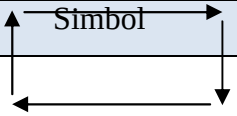
Flowchart adalah diagram alir yang baik untuk menggambarkan proses kerja karena dapat membantu mendefinisikan dokumen dan menganalisis proses. *Flowchart* dapat memberikan garis besar ringkasan dan deskripsi keseluruhan proses transaksi dalam suatu sistem. Tujuan dari *flowchart* adalah untuk menyajikan yang jelas, ringkas dan deskripsi dari sistem atau operasi, baik manual atau otomatis.

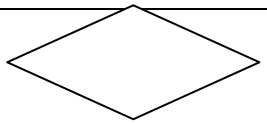
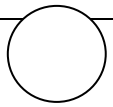
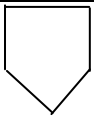
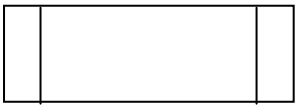
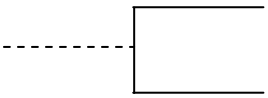
(Vallabhaneni, 2015 : 189-190)

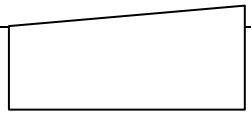
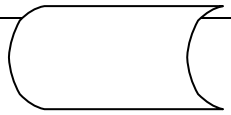
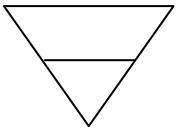
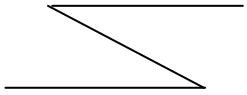
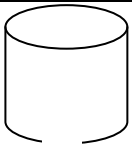
Simbol-simbol *flowchart* yang biasanya dipakai adalah simbol-simbol *flowchart* standar yang dikeluarkan oleh ANSI dan ISO. Simbol-simbol ini dapat dilihat pada Tabel 2.2 di bawah ini :

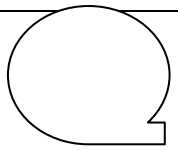
Tabel 2.2. Simbol *Flowchat*

(Pearson, 2011 : 232-234)

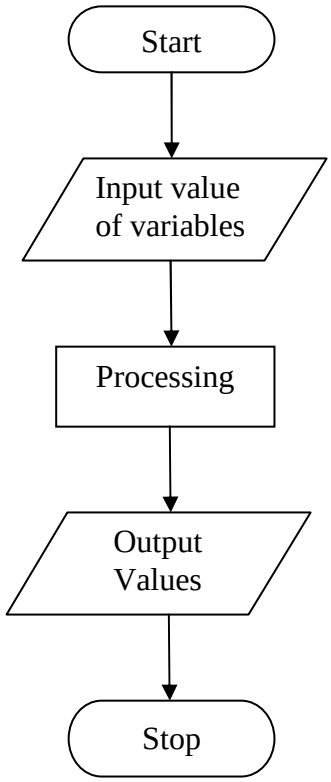
Nama Simbol	Simbol	Fungsi Simbol
Flow Lines		Digunakan untuk menghubungkan simbol. Garis-garis ini menunjukkan langkah-langkah dan arah alirnya.
Terminal		Digunakan untuk merepresentasikan start, *end* atau *pause* di dalam logika program.
Input/Output		Merepresentasikan informasi yang masuk atau keluar dari sistem seperti pesanan pelanggan (*input*) dan pembayaran (*output*).
Processing		Digunakan untuk mewakili *arithmetic* dan data *movement instructions*. Hal ini dapat mewakili satu langkah atau seluruh *subprocess* dalam proses yang lebih besar.

Decision		Menunjukkan sebuah keputusan (atau cabang) yang harus dibuat. Program harus dilanjutkan bersama salah satu dari dua rute (*IF/ELSE*). Simbol ini mempunyai satu entri dan dua jalur keluar. Jalur yang dipilih tergantung dari jawaban untuk pertanyaan ya atau tidak.
Connector		Digunakan untuk *join* dengan *flow line* yang berbeda.
Off-page connector		Digunakan untuk menunjukkan bahwa *flowchart* berlanjut pada halaman selanjutnya.
Predefined process		Merepresentasikan operasi atau proses yang sebelumnya telah ditentukan di tempat lain.
Annotation		Digunakan untuk memberikan tambahan informasi tentang simbol *flowchart* yang lain. Isinya mungkin dalam bentuk komentar deskriptif, pernyataan, atau catatan penjelasan.
Document		Digunakan untuk merepresentasikan sebuah

		paper document yang diproduksi selama proses *flowchart*
Multipage document		Digunakan untuk merepresentasikan sebuah dokumen dengan berbagai halaman
Manual input		Merepresentasikan masukan yang akan diberikan oleh *programmer*
Manual operation		Menunjukkan proses yang harus dilakukan oleh *programmer*
Online storage		Merepresentasikan penyimpanan data *online* seperti *hard disk, magnetic drum,* atau penyimpanan lainnya.
Offline storage		Merepresentasikan penyimpanan data *offline* seperti penjualan pada OCR dan data di *punched card.*
Communicati on link		Merepresentasikan data yang diterima atau ditransmisikan dari sistem eksternal
Magnetic disk		Merepresentasikan data input atau output dari dan ke *magnetic disk.*

Magnetic Tape		Merepresentasikan data *input* atau *output* dari dan ke *magnetic tape*.

Contoh *flowchart* dapat dilihat pada Gambar 2.12 di bawah ini :



Gambar 2.12. Contoh *Flowchart*

(Garrido, 2012 : 87)

2.10. *Unified Modelling Language* (UML)

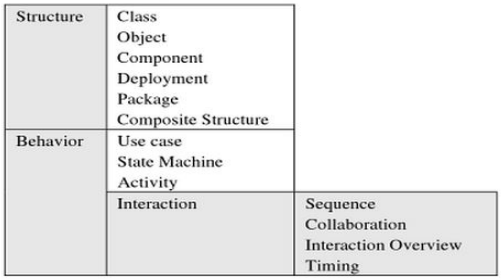
Pengertian UML, antara lain :

1. Whitten & Bentley (2007 : 371) mengatakan bahwa “UML adalah sekumpulan model konvensi yang digunakan untuk menentukan atau menggambarkan sistem perangkat lunak dalam istilah dari objek.”
2. Swain (2010 : 24) mengatakan bahwa “UML adalah bahasa untuk memvisualisasikan, menspesifikasikan, membangun, mendokumentasikan sebuah artifak dari sistem perangkat lunak yang intensif.”

Terdapat 13 tipe diagram UML yang dapat dipakai untuk pengembangan *software*, antara lain :

1. *Use case diagram*
2. *Activity diagram*
3. *Object diagram*
4. *Class diagram*
5. *State machine diagram*
6. *Composite structure diagram*
7. *Sequence diagram*
8. *Communication diagram*
9. *Interaction overview diagram*
10. *Component diagram*
11. *Deployment diagram*
12. *Package diagram*
13. *Timing diagram*

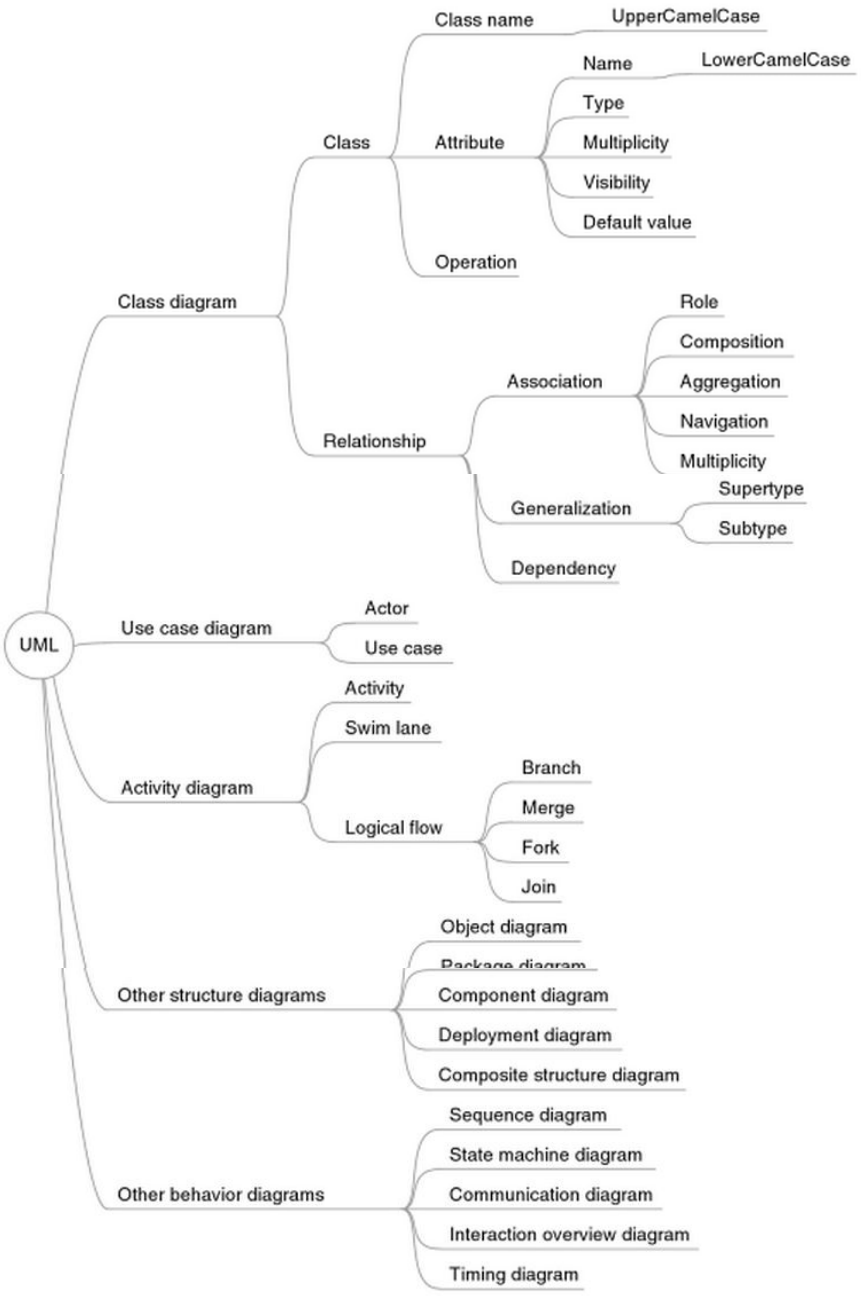
Bisa dilihat pada Gambar 2.13 di bawah ini :



Gambar 2.13. 13 Tipe Diagram UML

(Halpin & Morgan, 2008 : 347)

Dapat ditunjukkan melalui Gambar 2.14 di bawah ini :



Gambar 2.14. Diagram UML dan Komponennya
(Abies, Hermitage, Thatcham, & Berkshire, 2012 : 54)

Terdapat 13 diagram UML yang masing-masing memiliki fungsi dan tujuan yang sama dalam perangcangan suatu pangkat lunak. Namun, tidak selalu ke-13 diagram tersebut digunakan. UML yang akan digunakan dipilih sesuai

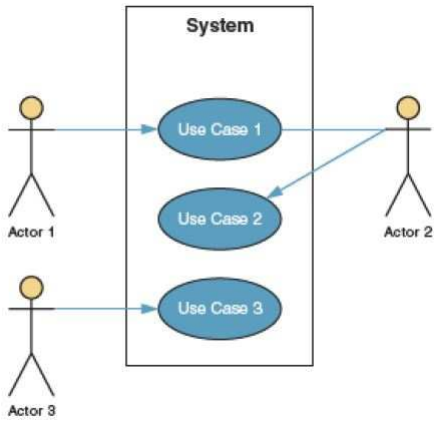
dengan kebutuhan, dengan syarat sudah dapat menggambarkan proses pengembangan sistem yang jelas.

Penulis akan menggunakan 4 diagram UML yang dibutuhkan, yaitu :

1. *Use case diagram*

   Use case diagram adalah diagram yang menggambarkan interaksi antara sistem, eksternal sistem dan pengguna. Dengan kata lain, secara grafis menggambarkan siapa yang akan menggunakan sistem dan dengan cara apa pengguna mengharapkan untuk berinteraksi dengan sistem.
   (Whitten & Bentley, 2007 : 246)

   Contoh *Use case diagram* dapat dilihat melalui Gambar 2.15 di bawah ini:

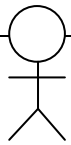
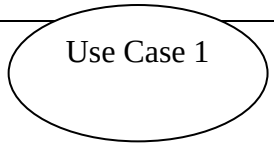
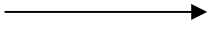


Gambar 2.15. *Use Case Diagram*
(Whitten & Bentley, 2007 : 246)

Dalam *use case diagram* terdapat tiga komponen utama yaitu *actor*, *use case*, dan *relationship*, seperti dijelaskan dalam tabel berikut Tabel 2.3 di bawah ini :

Tabel 2.3. *Syntax for Use Case Diagram*
(Whitten & Bentley, 2007:384)

Unsur-unsur	Notasi	Deskripsi

Actor	 Actor	- *Actor* merupakan pengguna yang akan berinteraksi dengan sistem untuk saling bertukar informasi. - *Actor* digambarkan berupa *stick figure*, dengan label peran *actor/user* dalam sistem. - *Actor* dapat berupa manusia, organisasi, sistem informasi yang lain, dan perangkat eksternal
Use Case	 Use Case 1	- *Use case* adalah aktivitas yang akan dilakukan dalam sistem berupa interaksi antara sistem dan *user* untuk mencapai tujuan tertentu. - *Use case* digambarkan dalam bentuk horizontal elips dengan nama *use case* di bagian dalam, atas atau bawah horizontal elips tersebut.
Relationship		Menggambarkan hubungan antara dua simbol pada *use case diagram*

Dalam *relationship* terdapat tipe hubungan dalam *use case diagram*, yaitu :

a. *Associations*

Hubungan antara *use case* dan *actor* terjadi ketika *use case* menjelaskan interaksi antara kedua simbol tersebut. *Associations* digambarkan berupa garis solid yang menghubungkan *use case* dengan *actor*. Garis yang memiliki anak panah berarti *actor* berperan sebagai pelaku dari

use case, sedangkan garis tanpa anak panah berarti *actor* berperan sebagai *external database* atau penerima *use case* tersebut.

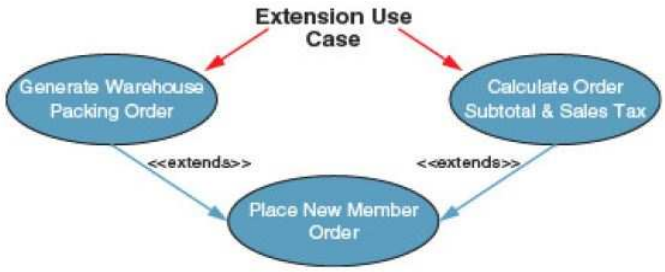
Contoh *Associations Relationship* bisa dilihat pada Gambar 2.16 di bawah ini :

Gambar 2.16. *Association Relationship*
(Whitten & Bentley, 2007 : 248)

b. *Extends*

Use case dapat berisikan sebuah fungsi yang kompleks yang dapat membuatnya susah dimengerti. Oleh karena itu, dengan tujuan mempermudah *use case* tersebut, dibuatlah *extension use case* yaitu hubungan *extends* antara *use case* awal yang rumit dengan *use case* baru yang mewakili fungsi tertentu *use case* awal.

Contoh *Extends Relationship* bisa dilihat pada Gambar 2.17 di bawah ini :



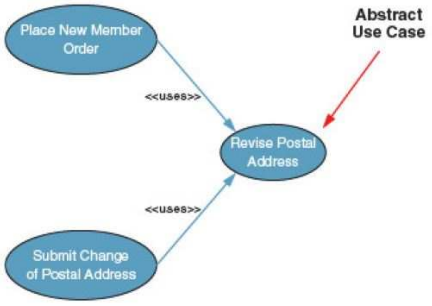
Gambar 2.17. *Extends Relationship*
(Whitten & Bentley, 2007 : 249)

c. *Uses/Includes*

Uses (includes) adalah *relationship* antara *use case* dengan *abstract use case* yang digunakan. *Abstract use case* adalah sebuah *use case* yang

digunakan untuk mengurangi *redundancy* dari dua *use case* atau lebih yang memiliki langkah yang sama dengan menggabungkan langkah-langkah yang terdapat dalam *use case* tersebut. *Abstract use case* ditunjukan dengan arah panah yang mengarah dari *use case* ke arah *abstract use case* tersebut.

Contoh *Uses/Include Relationship* bisa dilihat pada Gambar 2.18 di bawah ini :



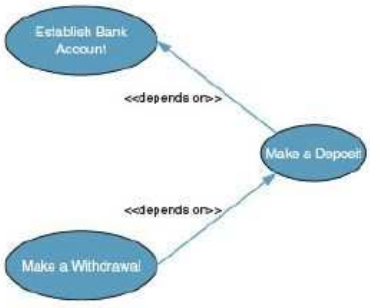
Gambar 2.18. *Uses/Include Relationship*

(Whitten & Bentley, 2007 : 249)

d. *Depends On*

Depends on menunjukan hubungan keterkaitan antara *use case* dimana sebuah *use case* tidak dapat dijalankan jika *use case* yang lain belum dijalankan.

Contoh *Depends On Relationship* bisa dilihat pada Gambar 2.19 di bawah ini :



Gambar 2.19. *Depends On Relationship*

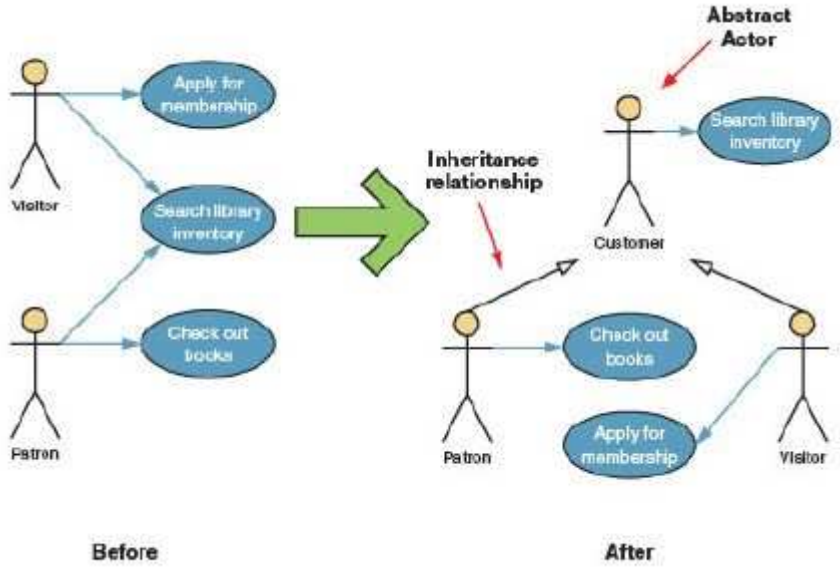
(Whitten & Bentley, 2007 : 250)

e. *Inheritance*

Apabila dua atau lebih *actor* menggunakan *use case* yang sama, maka diperlukan *abstract actor* untuk mengurangi redudansi komunikasi

dengan sistem. Hubungan antara *abstract actor* dengan aktor-aktor sebenarnya dalam *use case diagram* inilah yang disebut *inheritance.*

Contoh *Inheritance Relationship* bisa dilihat pada Gambar 2.20 di bawah ini :



Gambar 2.20. *Inheritance Relationship*
(Whitten & Bentley, 2007 : 250)

Use case narrative berisikan rincian setiap *event* dan bagaimana pengguna berinteraksi dengan sistem untuk menyelesaikan tugas.
(Whitten & Bentley, 2007 : 246)

Contoh *Use case narrative* bisa dilihat pada Gambar 2.21 di bawah ini:

Use-Case Name:	Place New Order	**Use-Case Type** **Business Requirements:** ☒
Use-Case ID:	MSS-BUC002.00	
Priority:	High	
Source:	Requirement — MSS-R1.00	
Primary Business Actor:	Club member	
Other Participating Actors:	• Warehouse (external receiver) • Accounts Receivable (external server)	
Other Interested Stakeholders:	• Marketing — interested in sales activity in order to plan new promotions. • Procurement — interested in sales activity in order to replenish inventory. • Management — interested in order activity in order to evaluate company performance and customer (member) satisfaction.	
Description:	This use case describes the event of a club member submitting a new order for SoundStage products. The member's demographic information as well as his or her account standing is validated. Once the products are verified as being in stock, a packing order is sent to the warehouse for it to prepare the shipment. For any product not in stock, a back order is created. On completion, the member will be sent an order confirmation.	
Precondition: 1	The party (individual or company) submitting the order must be a member.	
Trigger: 2	This use case is initiated when a new order is submitted.	
Typical Course of Events: 3	**Actor Action**	**System Response**
	Step 1: The club member provides his or her demographic information as well as order and payment information.	**Step 2:** The system responds by verifying that all required information has been provided. **Step 3:** The system verifies the club member's demographic information against what has been previously recorded. **Step 4:** For each product ordered, the system validates the product identity. **Step 5:** For each product ordered, the system verifies the product availability. **Step 6:** For each available product, the system determines the price to be charged to the club member. **Step 7:** Once all ordered products are processed, the system determines the total cost of the order. **Step 8:** The system checks the status of the club member's account. **Step 9:** The system validates the club member's payment if provided. **Step 10:** The system records the order information and then releases the order to the appropriate distribution center (warehouse) to be filled. **Step 11:** Once the order is processed, the system generates an order confirmation and sends it to the club member.

Gambar 2.21. *Use Case Narrative*

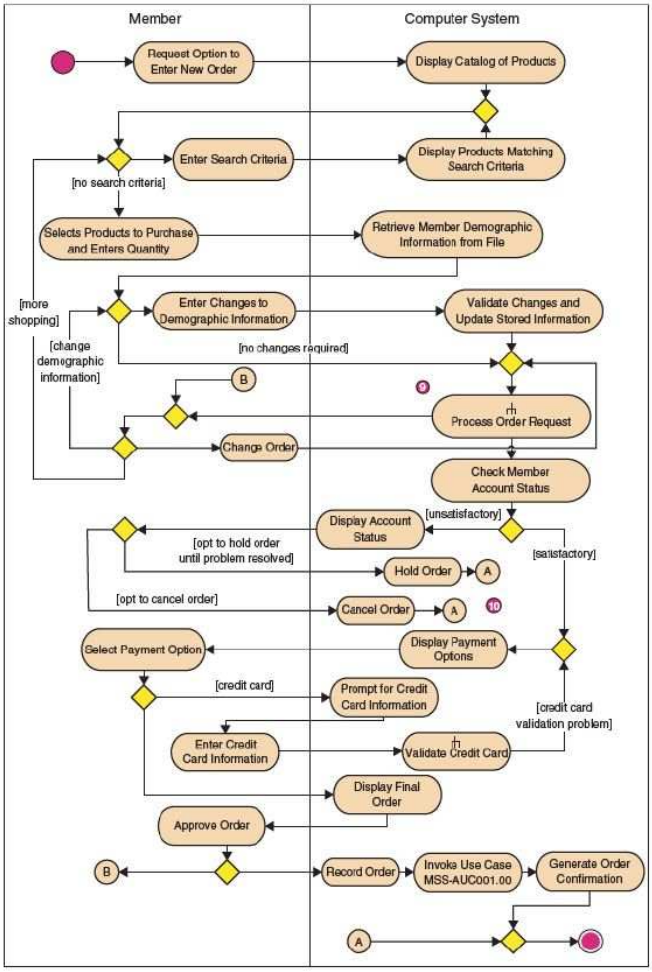
(Whitten & Bentley, 2007 : 260)

2. A*ctivity diagram*

   A*ctivity diagram* adalah diagram yang dapat digunakan untuk proses bisnis, langkah-langkah *use case*, atau logika perilaku objek.

   (Whitten & Bentley, 2007 : 390)

Contoh *Activity Diagram* bisa dilihat pada Gambar 2.22 di bawah ini :



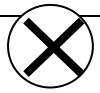
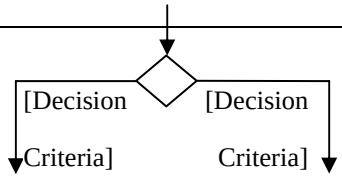
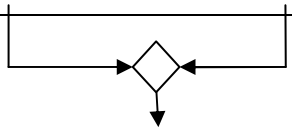
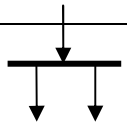
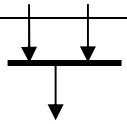
Gambar 2.22. *Activity Diagram*

(Whitten & Bentley, 2007 : 393)

Unsur-unsur dalam *activity diagram* dapat dilihat pada tabel 2.4 di bawah ini :

Tabel 2.4. *Syntax for an Activity Diagram*

Unsur-unsur	Notasi	Deskripsi
Actions	Activity *Actions*	Notasi ini menggambarkan sebuah aktivitas yang jika disusun secara keseluruhan akan menggambarkan keseluruhan aktivitas dalam *activity diagram.*

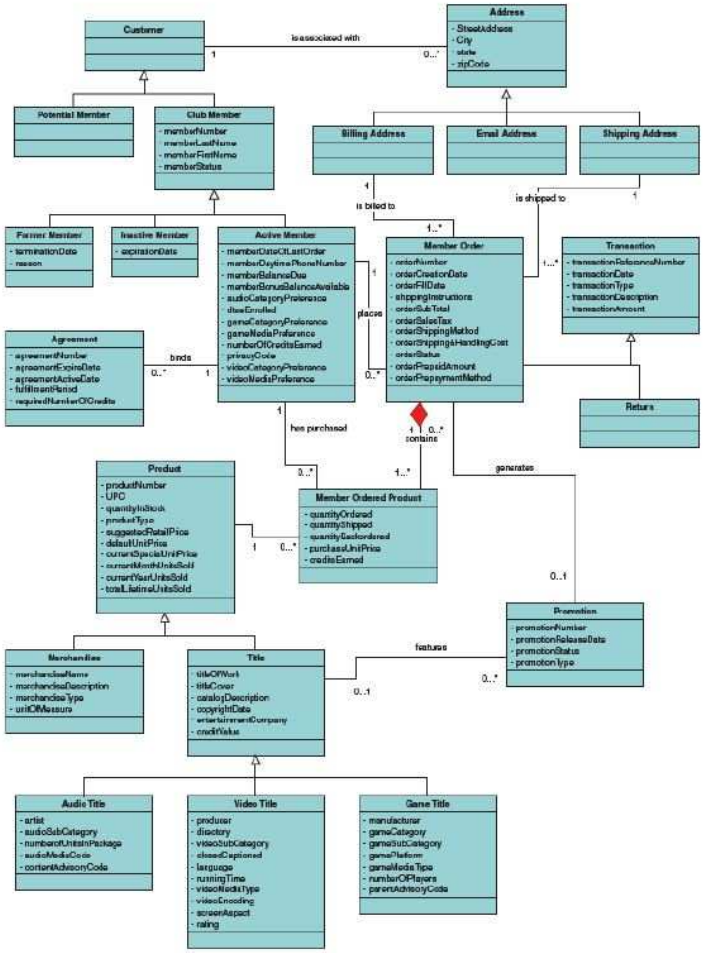
Flow		Menggambarkan jalannya suatu aktivitas
Initial Node		Menggambarkan awal dari sekelompok *action* atau *activity*
Subactivity Indicator/ Rake		Menunjukkan adanya dekomposisi
Activity Final		Menggambarkan akhir proses dari *activity diagram*
Decision Node		Menggambarkan suatu kondisi dimana suatu keputusan harus diambil.
Merge Node		Menggabungkan kembali proses yang sebelumnya dipisahkan oleh *decision*
Fork Node		Menunjukkan kegiatan yang dilakukan secara bersamaan
Join Node		Menggabungkan dua kegiatan atau lebih yang dilakukan bersamaan menjadi satu

3. C*lass diagram*

   Class diagram adalah gambaran grafis dari struktur objek pada sistem statis dan menunjukkan *class-class* objek yang terdapat pada sistem serta hubungan antara *class* tersebut.

   (Whitten & Bentley, 2007 : 400)

Contoh *class diagram* bisa dilihat pada Gambar 2.23 di bawah ini :



Gambar 2.23. *Class Diagram*

(Whitten & Bentley, 2007 : 406)

Notasi-notasi dalam *class diagram*, yaitu:

1. *Class*

   Merupakan elemen utama dari *class diagram*. *Class* ini akan membentuk suatu objek yang akan memiliki semua elemen *class* tersebut. *Class* digambarkan sebagai sebuah kotak yang terdiri dari 3 bagian yaitu :

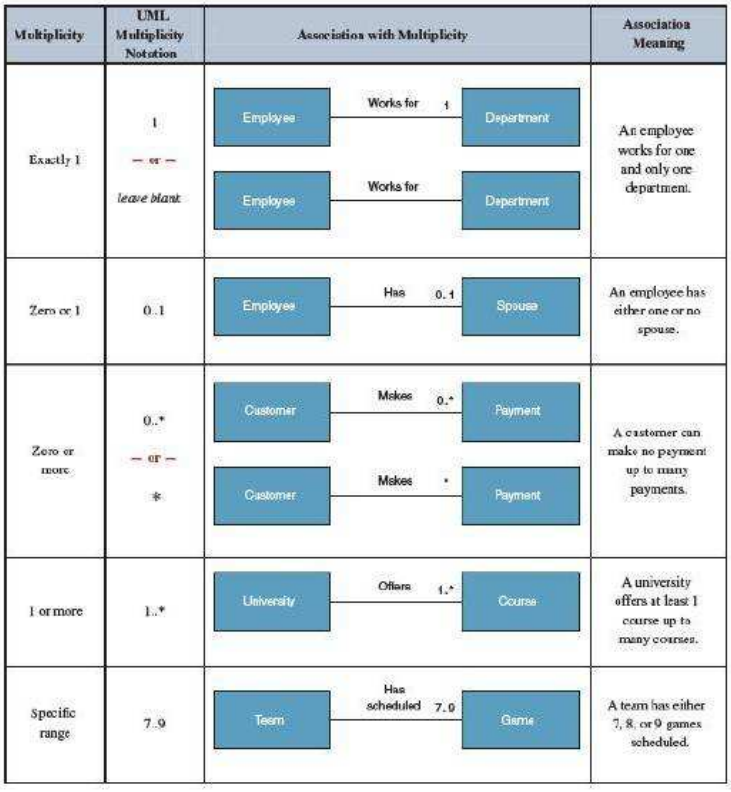
   a. Bagian atas : *class name*
   b. Bagian tengah : *attribute*
   c. Bagian bawah : *operational*

2. *Relationship*

   a. *Association*

   Association adalah hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *cla*ss lain. Hubungan ini diperlukan agar sebuah *class* dapat menyampaikan pesan kepada *class* lainnya. *Association* digambarkan dengan sebuah garis tanpa tanda panah.

   Contoh *association* bisa dilihat pada Gambar 2.24 di bawah ini :



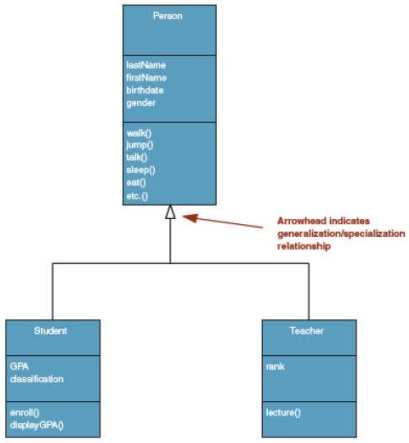
Gambar 2.24. Notasi *Association* dan *Multiplicity*
(Whitten & Bentley, 2007 : 377)

   b. *Generalization*

   Generalization memungkinkan suatu *class* mewarisi *attribute* dan *operation* yang dimiliki oleh *base class*. *Attribute* dan *operation* yang dapat diwarisi adalah yang memiliki *access modifier public*, *protected*, dan *default*. Hubungan ini digambarkan dengan garis

yang memiliki tanda panah tertutup kosong pada salah satu ujungnya yang mengarah ke *base class*.

Contoh *generalization* bisa dilihat pada Gambar 2.25 di bawah ini :



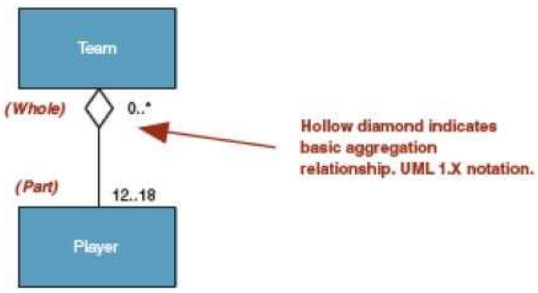
Gambar 2.25. *Generalization*

(Whitten & Bentley, 2007 : 377)

c. *Aggregation*

Hubungan yang menyatakan bagian dari objek lainnya, dapat didefinisikan seperti objek a mengandung objek b dan objek b bagian dari objek a. Hubungan asimetris, artinya tidak berlaku sebaliknya. Hubungan agregasi merupakan suatu bentuk relasi yang jauh lebih kuat dari asosiasi. Agregasi dapat diartikan bahwa suatu *class* merupakan bagian dari *class* yang lain namun bersifat tidak wajib. *Agregasi* digambarkan dengan sebuah garis yang memiliki bentuk *diamond* kosong di salah satu ujung garisnya.

Contoh *aggregation* bisa dilihat pada Gambar 2.26 di bawah ini :



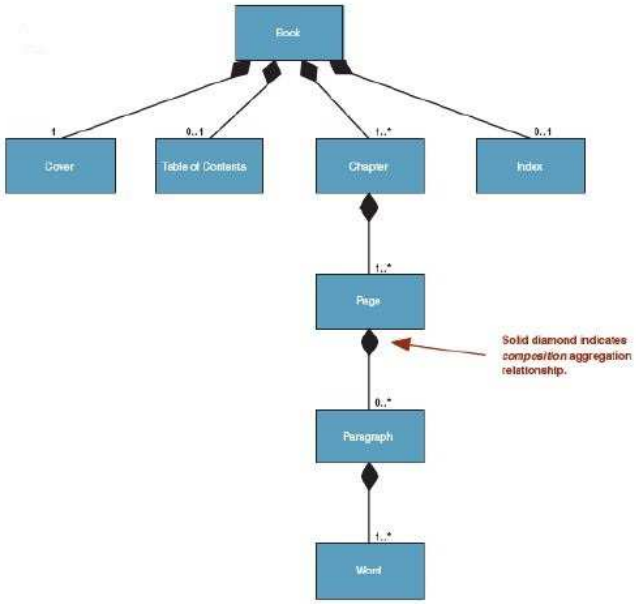
Gambar 2.26. *Aggregation*

(Whitten & Bentley, 2007 : 379)

d. *Composition*

Hubungan ini merupakan yang paling kuat dibandingkan *association* atau *aggregation*. Hubungan ini berarti suatu *class* merupakan bagian wajib dari *class* lain. Hubungan ini digambarkan dengan sebuah garis yang memiliki bentuk *diamond* utuh di salah satu ujung garisnya.

Contoh *composition* bisa dilihat pada Gambar 2.27 di bawah ini :



Gambar 2.27. *Composition*

(Whitten & Bentley, 2007 : 379)

e. *Dependency*

Melambangkan suatu koneksi antar *class* yang disimbolkan dengan garis putus-putus. Sebuah *class* bergantung pada *class* lain apabila perubahan pada sebuah *class a*kan membuat *class* lain ikut berubah.

3. *Multipilicity*

Pada akhir sebuah *relationship* biasanya terdapat angka pada salah satu ujung garis *relationship* yang melambangkan jumlah objek dari *class* tersebut yang berasosiasi dengan *class* lain. *Multiplicity* dilambangkan dengan angka sebagai berikut :

a. Angka “0..1” yang berarti ada 0 atau 1 objek pada akhir *association*.

b. Angka “1..*” yang berarti ada 1 atau lebih objek.

c. Angka “0..*” atau biasa dituliskan “*” berarti ada 0 atau lebih objek.

4. Sifat *Attribute* dan *Method*

Attribute dan *method* dalam *class* memiliki salah satu sifat berikut :

a. *Private* (-) : tidak dapat dipanggil di luar *class* yang bersangkutan
b. *Protected* (#) : hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya
c. *Public* (+) : dapat dipanggil oleh semua *class* lain

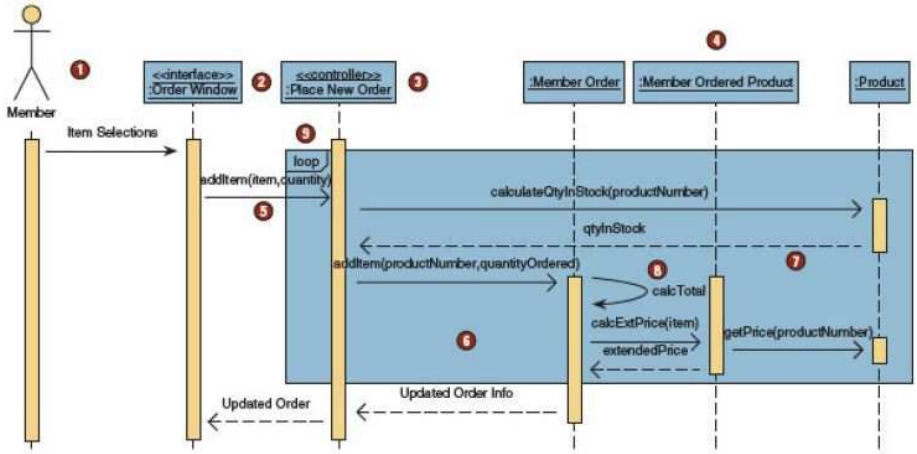
(Whitten & Bentley, 2007:373-38)

4. *Sequence Diagram*

Sequence diagram membangun logika dari *use case* yang tergambarkan dari interaksi antar objek dalam suatu urutan waktu. Logika *use case* akan terbentuk dengan menggambarkan interaksi pesan antar objek pada suatu waktu

(Whitten & Bentley, 2007:659).

Contoh *sequence diagram* bisa dilihat pada Gambar 2.28 di bawah ini :

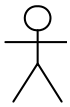
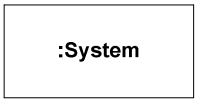
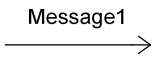
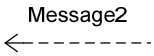
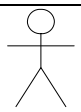
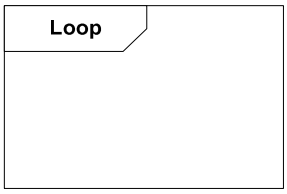


Gambar 2.28 Contoh *Sequence Diagram*

(Whitten & Bentley, 2007:659)

Terdapat beberapa notasi yang digunakan dalam pembuatan *sequence diagram* yang ditunjukan pada tabel di bawah ini.

Tabel 2.5 Notasi *Sequence Diagram*

Notasi	Keterangan	Simbol
Actor	Menggambarkan *user* yang berinteraksi dengan sistem	
System	Menggambarkan *instance* dari sebuah *class* pada *class diagram*	
Lifelines	Menggambarkan keberadaan sebuah objek dalam suatu waktu atau waktu dari *sequence*	
Activation Bars	Menggambarkan waktu dimana *user* sedang aktif berinteraksi dengan sistem	
Input Messages	Menggambarkan pesan masuk yang dikirimkan berupa *behavior*	
Output Messages	Menggambarkan balasan dari pesan masuk yang berupa *attribute*	
Receiver Actor	Aktor lainnya atau sistem external yang menerima pesan dari sistem	
Frame	Menggambarkan area pada sistem yang mengalami perulangan (*loop*), seleksi (*alternate fragments*), atau kondisi opsional (*optional*)	

2.11. *Database* SQL

Pengertian SQL, antara lain :

1. Scott (2010 : 21) mengatakan bahwa “SQL adalah singkatan dari *Structured Query Language*.”
2. Fehily (2008 : xii) mengatakan bahwa “SQL adalah bahasa resmi dimana kita dapat menulis program untuk membuat, memodifikasi, dan *query database*.”

SQL *statement* terbagi menjadi tiga kategori, yaitu :

1. *Data Manipulation Language* (DML)
   Berisi pernyataan *retrieve, reckon, insert, edit, dan delete* data yang tersimpan di *database.*
   Contoh dari DML *statements* adalah (Sharan, 2011 : 393) :
   SELECT, INSERT, UPDATE, DELETE, dan lain-lain.
2. *Data Definition Language* (DDL)
   Berisi pernyataan *create, modify*, dan *destroy* objek *database* seperti *tables, indexs*, dan *views*.
   Contoh dari DML *statements* adalah (Sharan, 2011 : 393) :
   CREATE TABLE, ALTER TABLE, dan lain-lain.
3. *Data Control Language* (DCL)
   Berisi pernyataan *view, change* atau *delete* data dan *database* object.
   Contoh dari DCL *statements* adalah (Sharan, 2011 : 393) :
   GRANT dan *REVOKE*

(Fehily, 2008 : xv)

2.12. *Framework* .NET

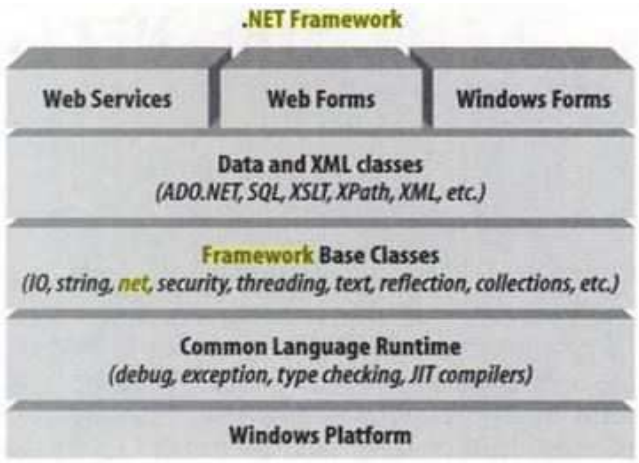
.NET adalah *development framework* yang menyediakan sebuah aplikasi *programming interface* untuk layanan dan APIs dari sistem operasi *Windows* klasik dan menyatukan beberapa teknologi yang berbeda yang muncul dari *Microsoft* selama akhir 1990-an. Teknologi-teknologi baru mencakup layanan komponen COM +, komitmen untuk XML dan *object-oriented design*.

Microsoft .NET mendukung *Common Type Specification* (CTS) yang dapat membuat pengguna memilih *syntax*.

Framework .NET berisi tentang :

a. Memperluas daftar dari *official language*, misalnya C#, VB.NET, dan Jscript.NET.
b. *Common Language Runtime* (CLR), sebuah *platform* berorientasi objek untuk *Windows* dan *web development*.
c. Sejumlah *class libraries* yang saling terkait, biasa dikenal sebagai *Framework Class Library* (FCL).

Arsitektur *Framework* .NET dapat dilihat pada Gambar 2.29 di bawah ini :



Gambar 2.29. Arsitektur *Framework* .NET
(Liberty & Hurwitz, 2004 : 2)

Penjelasan Arsitektur *Framework* .NET, yaitu :

a. *Web services*
   Memungkinkan pengembangan aplikasi yang menyediakan pemanggilan metode melalui internet
b. *Web Forms*
   Memungkinkan pengembangan yang kuat, *web pages* dan *web sites* yang terukur.
c. *Windows Forms*
   Memungkinkan pengembangan windows *desktop application* dengan kekayaan dan *user interface* yang *flexible*. Aplikasi *desktop* ini dapat berinteraksi dengan komputer lain di jaringan lokal atau melalui internet dengan penggunaan *web service*.

(Liberty & Hurwitz, 2004 : 1-3)

2.13. Visual Basic.Net (VB.Net)

Visual Basic 2010 adalah versi terbaru yang diluncurkan oleh Microsoft pada tahun 2010. Visual Basic telah melalui berbagai tahapan perkembangan sejak zaman BASIC yang dibangun untuk DOS. BASIC adalah singkatan dari *Beginners' All-purpose Symbolic Instruction Code*. Kode program di Visual Basic menyerupai bahasa Inggris. Perusahaan perangkat lunak yang berbeda telah menghasilkan berbagai versi BASIC untuk DOS, seperti Microsoft QBASIC, QUICKBASIC, GWBASIC, IMB BASICA, dan masih banyak lagi. Ketika Microsoft meluncurkan BASIC grafis pertama, Visual Basic versi 1 di 1991. Visual Basic versi 1 berbasis GUI dan terutama dikembangkan untuk MS Window. Sejak itu Microsoft perlahan bertahap menghapus DOS versi BASIC dan benar-benar menggantikan mereka dengan Visual Basic.

Visual Basic awalnya adalah bahasa pemrograman fungsional atau procedural sampai Visual Basic 6 terkenal. Kemudian Microsoft mentransformasikan Visual Basic menjadi bahasa pemrograman berorientasi objek yang lebih kuat dengan memunculkan VB.Net, Visual Basic 2005, Visual Basic 2008 dan terakhir mengeluarkan Visual Basic 2010. Visual Basic 2010 adalah bahasa *Object-Oriented Programming* (OOP) yang lengkap dan melengkapi dengan bahasa OOP seperti C++, Java, C# dan lainnya.

(Kiong, 2011 : 1)

Kelebihan dari VB.Net adalah :

a. VB.Net adalah sebuah bahasa yang dimana pengguna dapat memberitahu komputer untuk melakukan sesuatu.
b. VB.Net memberikan bahasa yang sederhana untuk menjelaskan beberapa hal yang sangat kompleks. Meskipun tidak ada salahnya untuk memiliki pemahaman tentang apa yang terjadi di tingkat terendah, VB.Net membebaskan *programmer* dari keharusan untuk berurusan dengan kompleksitas dari menulis program *Windows*. Sehingga *programmer* dapat berkonsentrasi pada pemecahan masalah.
c. VB.Net membantu pengguna membuat solusi yang berjalan pada sistem operasi Microsoft Windows.

d. VB.Net dapat digunakan untuk membuat aplikasi untuk digunakan melalui internet.
e. VB.Net merupakan pilihan yang baik untuk semua level *programmer*.

(Reynolds, Blair, Crossland & Willis, 2004 : 9-11)