BAB 2
LANDASAN TEORI

2.1 Sudoku

Sudoku merupakan permainan teka-teki silang angka, dimana setiap baris dan kolom harus diisi dengan angka-angka dengan aturan tertentu. Sudoku adalah teka-teki penempatan angka yang didasarkan pada logika matematika. Sudoku terdiri dari kotak-kotak kecil yang membentuk baris dan kolom seperti halnya teka-teki silang (TTS).

Sudoku terdiri dari kisi persegi yang disebut grid berukuran 9x9 (9 baris x 9 kolom) yang dibagi menjadi sembilan bagian yang disebut subgrid berukuran 3x3. Teka-teki Sudoku menyediakan beberapa angka pada posisi acak sebagai patokan untuk menyelesaikan keseluruhan teka-teki tersebut. Kotak yang kosong harus diisi dengan angka unik dari 1 sampai 9 yang masing-masing hanya muncul sekali dalam setiap baris, kolom, dan persegi 3x3.

2.2 *Artificial Intelligence* (AI)

2.2.1 Definisi AI

Sebagian kalangan menerjemahkan *Artificial Intelligence* sebagai kecerdasan buatan, kecerdasan artifisial, intelijensia artifisial, atau intelejensia buatan. Istilah *Artificial Intelligence* sengaja tidak diterjemahkan ke bahasa Indonesia karena istilah tersebut sudah sangat akrab bagi orang Indonesia. Begitu juga dengan singkatan istilah tersebut, yaitu AI, sudah sangat melekat di berbagai media ilmiah maupun non ilmiah.

Para ahli mendefinisikan AI secara berbeda-beda tergantung pada sudut pandang mereka masing-masing. Ada yang fokus pada logika berpikir manusia saja, tetapi ada juga yang mendefinisikan AI secara lebih luas pada tingkah laku manusia. Stuart Russel dan Peter Norvig mengelompokkan definisi AI, yang diperoleh dari beberapa *textbook* berbeda, ke dalam empat kategori, yaitu :

1. ***Thinking humanly** : the cognitive modeling approach*

   Pendekatan ini dilakukan dengan dua cara sebagai berikut :

   1.1 Melalui introspeksi : mencoba menangkap pemikiran-pemikiran kita sendiri pada saat kita berpikir. Tetapi, seorang psikolog barat mengatakan "*how do you know that you understand*?" Bagaimana anda tahu bahwa anda mengerti? Karena pada saat anda menyadari pemikiran anda. Sehingga definisi ini terkesan mengada-ngada dan tidak mungkin dilakukan.

   1.2 Melalui eksperimen-eksperimen psikologi.

2. ***Acting humanly** : the Turing test approach*

   Pada tahun 1950, Alan Turing merancang suatu ujian bagi komputer berintelijensia untuk menguji apakah komputer tersebut mampu mengelabui seorang manusia yang menginterogasinya melalui *teletype* (komunikasi berbasis teks jarak jauh). Jika *interrogator* tidak dapat membedakan yang diinterogasi adalah manusia atau komputer, maka komputer berintelijensia tersebut lolos dari Turning test. Komputer tersebut perlu memiliki kemampuan : *Natural Language Processing*, *Knowledge Representation*, *Automated Reasoning*, *Machine Learning*, *Computer Vision*, *Robotics*. *Turing test* sengaja menghindari interaksi fisik antara *interrogator* dan komputer karena simulasi fisik manusia tidak memerlukan intelijensia.

3. ***Thinking rationally** : the laws of thought approach*

   Terdapat dua masalah dalam pendekatan ini, yaitu :

   3.1 Tidak mudah untuk membuat pengetahuan informal dan menyatakan pengetahuan tersebut ke dalam *formal term* yang diperlukan oleh notasi logika, khususnya ketika pengetahuan tersebut memiliki kepastian kurang dari 100%.

   3.2 Terdapat perbedaan besar antara dapat memecahkan masalah "dalam prinsip" dan memecahkannya "dalam dunia nyata".

4. ***Acting rationally** : the rational agent approach*

Membuat inferensi yang logis merupakan bagian dari suatu *rational agent*. Hal ini disebabkan satu-satunya cara untuk melakukan aksi secara rasional adalah dengan menalar secara logis. Dengan menalar secara logis, maka bisa didapatkan kesimpulan bahwa aksi yang diberikan akan mencapai tujuan atau tidak. Jika mencapai tujuan, maka agent dapat melakukan aksi berdasarkan kesimpulan tersebut.

Thinking humanly dan *acting humanly* adalah dua definisi dalam arti yang sangat luas. Sampai saat ini, pemikiran manusia yang diluar rasio, yakni refleks dan intuitif (berhubungan dengan perasaan), belum dapat ditirukan sepenuhnya oleh komputer. Dengan demikian, kedua definisi ini dirasa kurang tepat untuk saat ini. Jika kita menggunakan definisi ini, maka banyak produk komputasi cerdas saat ini yang tidak layak disebut sebagai produk AI.

Definisi *thinking rationally* terasa lebih sempit daripada *acting rationally*. Oleh karena itu, definisi AI yang paling tepat untuk saat ini adalah *acting rationally* dengan pendekatan *rational agent*. Hal ini berdasarkan pemikiran bahwa komputer bisa melakukan penalaran secara logis dan juga bisa melakukan aksi secara rasional berdasarkan hasil penalaran tersebut.

2.2.2 Teknik Pemecahan Masalah AI

1. *Searching*

   Terdapat beberapa langkah dalam teknik *searching* atau pencarian ini. Langkah pertama yang harus dilakukan adalah mendefinisikan ruang masalah untuk suatu masalah yang dihadapi. Ruang masalah ini dapat digambarkan sebagai himpunan keadaan (*state*) atau bisa juga sebagai himpunan rute dari keadaan awal (*initial state*) menuju keadaan tujuan (*goal state*). Langkah kedua adalah mendefinisikan aturan produksi yang digunakan untuk mengubah suatu *state* ke *state* lainnya. Langkah terakhir adalah memilih metode pencarian yang tepatsehingga dapat menemukan solusi terbaik dengan usaha yang minimal.

   Terdapat banyak metode pencarian yang telah diusulkan. Semua metode yang ada dapat dibedakan mendadi dua jenis, yaitu :

1.1 *Blind/Un-informed Search*

1.1.1 *Breadth-First Search* (BFS)

1.1.2 *Depth-First Search* (DFS)

1.1.3 *Depth-Limited Search* (DLS)

1.1.4 *Uniform Cost Search* (USC)

1.1.5 *Iterative-Deepening Search* (IDS)

1.1.6 *Bi-Directional Search* (BDS)

1.2 Metode Pencarian Heuristik

1.2.1 *Generate-and-Test* (Bangkitkan-dan-Uji)

1.2.2 *Hill Climbing* (Pendakian Bukit)

1.2.3 *Simulated Annealing* (SA)

1.2.4 *Best-First Search* (Pencarian Terbaik Terlebih Dahulu)

1.2.5 Greedy Best-First Search

1.2.6 A* (A *star* atau A bintang)

1.2.7 *Iterative Deepening* A* (IDA*)

1.2.8 *Simplified Memory-Bounded* A* (SMA*)

1.2.9 *Bi-directional* A* (BDA*)

1.2.10 *Modified Bi-directional* A* (MBDA*)

1.2.11 *Dinamic Weighting* A* (DWA*)

2. *Reasoning*

   Teknik *reasoning* atau penalaran merupakan teknik penyelesaian masalah dengan cara mempresentasikan masalah kedalam basis pengetahuan (*knowledge base*) menggunakan *logic* atau bahasa formal (bahasa yang dipahami komputer).

3. *Planning*

   Planning adalah suatu metode penyelesaian masalah dengan cara memecah masalah ke dalam sub-sub masalah yang lebih kecil, menyelesaikan sub-sub masalah satu demi satu kemudian menggabungkan solusi-solusi dari sub-sub masalah tersebut menjadi sebuah solusi lengkap dengan tetap mengingat dan menangani interaksi yang terdapat pada sub-sub masalah tersebut.

4. *Learning*

   Pada ketiga teknik sebelumnya, kita harus mengetahui aturan yang berlaku untuk sistem yang akan kita bangun. Tetapi pada masalah tertentu, terkadang kita tidak bisa mendefinisikan aturan secara benar dan lengkap karena data-data yang kita dapatkan tidak lengkap. Melalui suatu teknik yang disebut *learning*, kita bisa secara otomatis menemukan aturan yang diharapkan bisa berlaku umum untuk data-data yang belum kita ketahui.[2]

2.2.3 Algoritma *Ant Colony Optimization (ACO) System*

Ant Colony Optimization (ACO) diadopsi dari perilaku koloni semut yang dikenal sebagai sistem semut. Semut mampu mengindera lingkungannya yang kompleks untuk mencari makanan dan kemudian kembali ke sarangnya dengan meninggalkan zat *Pheromone* pada rute-rute yang mereka lalui.

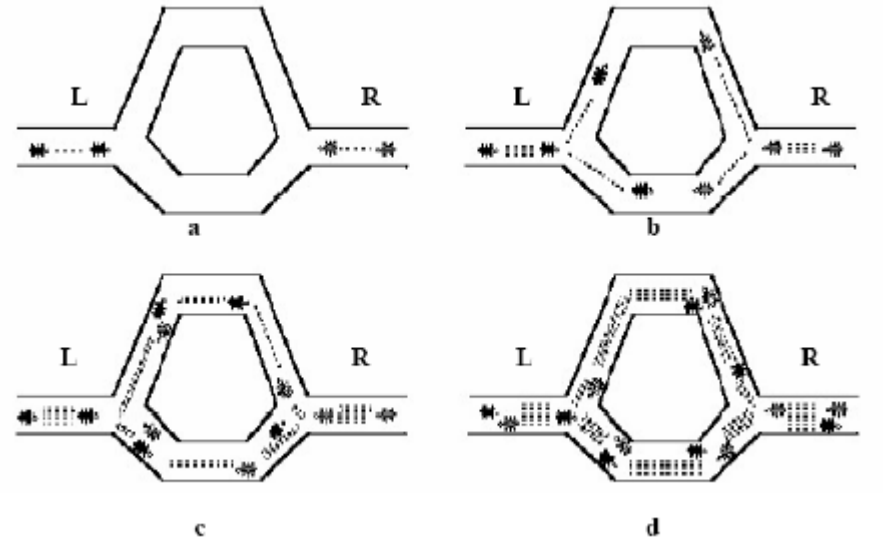
Pheromone adalah zat kimia yang berasal dari kelenjar endokrin dan digunakan oleh makhluk hidup untuk mengenali sesama jenis, individu lain, kelompok, dan untuk membantu proses reproduksi. Berbeda dengan hormon, *Pheromone* menyebar ke luar tubuh dan hanya dapat mempengaruhi dan dikenali oleh individu lain yang sejenis (satu spesies).

Proses peninggalan *Pheromone* ini dikenal sebagai *stigmery*, yaitu sebuah proses memodifikasi lingkungan yang tidak hanya bertujuan untuk mengingat jalan pulang ke sarang, tetapi juga memungkinkan para semut berkomunikasi dengan koloninya.

Seiring waktu, bagaimanapun juga jejak *Pheromone* akan menguap dan akan mengurangi kekuatan daya tariknya. Lebih cepat setiap semut pulang pergi melalui rute tersebut, maka *Pheromone* yang menguap lebih sedikit. Begitu pula sebaliknya jika semut lebih lama pulang pergi melalui rute tersebut, maka *Pheromone* yang menguap lebih banyak.

2.2.3.1 Cara kerja menemukan rute terpendek dalam ACO

Secara jelasnya cara kerja semut menemukan rute terpendek dalam ACO adalah sebagai berikut : Secara alamiah semut mampu menemukan rute terpendek dalam perjalanan dari sarang ke tempat-tempat sumber makanan. Koloni semut dapat menemukan rute terpendek antara sarang dan sumber makanan berdasarkan jejak kaki pada lintasan yang telah dilalui. Semakin banyak semut yang melalui suatu lintasan, maka akan semakin jelas bekas jejak kakinya. Hal ini akan menyebabkan lintasan yang dilalui semut dalam jumlah sedikit, semakin lama akan semakin berkurang kepadatan semut yang melewatinya, atau bahkan akan tidak dilewati sama sekali. Sebaliknya lintasan yang dilalui semut dalam jumlah banyak, semakin lama akan semakin bertambah kepadatan semut yang melewatinya, atau bahkan semua semut akan melalui lintasan tersebut.



Gambar 3.1 Perjalanan semut dari sarang ke sumber makanan

Gambar 3.1.a menunjukkan ada dua kelompok semut yang akan melakukan perjalanan. Satu kelompok bernama L yaitu kelompok yang berangkat dari arah kiri yang merupakan sarang semut dan kelompok lain yang bernama kelompok R yang berangkat dari kanan yang merupakan sumber makanan. Kedua kelompok semut dari titik awal keberangkatan sedang dalam posisi pengambilan keputusan jalan sebelah mana yang akan diambil. Kelompok semut L membagi

dua kelompok lagi. Sebagian melalui jalan atas dan sebagian melalui jalan bawah. Hal ini juga berlaku pada kelompok semut R. Gambar 3.1.b dan gambar 3.1.c menunjukkan bahwa kelompok semut berjalan pada kecepatan yang sama dengan meninggalkan *Pheromone* (jejak kaki semut) di jalan yang telah dilalui. *Pheromone* yang ditinggalkan oleh semut - semut yang melalui jalan atas telah mengalami banyak penguapan karena semut yang melalui jalan atas berjumlah lebih sedikit dari pada jalan yang di bawah. Hal ini dikarenakan jarak yang ditempuh lebih panjang daripada jalan bawah. Sedangkan *Pheromone* yang berada di jalan bawah, penguapannya cenderung lebih lama. Karena semut yang melalui jalan bawah lebih banyak daripada semut yang melalui jalan atas. Gambar 3.1.d menunjukkan bahwa semut-semut yang lain pada akhirnya memutuskan untuk melewati jalan bawah karena *Pheromone* yang ditinggalkan masih banyak. Sedangkan *Pheromone* pada jalan atas sudah banyak menguap sehingga semut-semut tidak memilih jalan atas tersebut. Semakin banyak semut yang melalui jalan bawah maka semakin banyak semut yang mengikutinya.

Demikian juga dengan jalan atas, semakin sedikit semut yang melalui jalan atas, maka *Pheromone* yang ditinggalkan semakin berkurang bahkan hilang. Dari sinilah kemudian terpilihlah rute terpendek antara sarang dan sumber makanan.[4]

2.2.4 Algoritma Genetika

Algoritma genetika adalah algoritma pencarian (*search algorithm*) yang menggunakan prinsip seleksi alam dalam ilmu genetika untuk mengembangkan solusi terhadap permasalahan. Algoritma Genetika merupakan kelas algoritma pencarian stokastik berdasarkan evolusi biologi.

Kemunculan Algortima Genetika diinspirasikan dari teori-teori dalam ilmu biologi, sehingga banyak istilah dan konsep biologi yang digunakan dalam Algoritma Genetika. Sesuai dengan namanya, proses-proses yang terjadi dalam Algoritma Genetika sama dengan apa yang terjadi pada evolusi biologi.

Ide dasar algoritma genetika adalah mengelola suatu populasi individu yang merepresentasikan kandidat solusi sebuah permasalahan. Secara umum algoritma genetika memiliki lima komponen dasar yaitu:

1. Representasi genetik dari solusi-solusi masalah.
2. Cara membentuk populasi awal dari solusi-solusi.
3. Fungsi evaluasi yang me-rate (*rating*) solusi-solusi berdasarkan *fitness* mereka.
4. Operator-operator genetik yang merubah komposisi genetik dari *offspring* selama reproduksi.
5. Nilai-nilai untuk parameter algoritma genetika.

Algoritma me-*maintain* populasi individu-individu untuk setiap generasi. Masing-masing individu menyatakan solusi yang potensial untuk masalah yang dihadapi. Masing-masing individu di-evaluasi untuk memberi ukuran *fitness*-nya. Nilai *fitness* adalah nilai yang menunjukkan drajat ketangguhan kromosom dalam beradaptasi terhadap masalah.

Salah satu aplikasi algoritma genetika adalah pada permasalahan optimasi, yaitu mendapatkan suatu nilai solusi optimal terhadap suatu permasalahan yang mempunyai banyak kemungkinan solusi. Daya tarik algoritma genetika terletak pada kesederhanaan dan pada kemampuan untuk mencari solusi yang baik dan cepat untuk masalah yang kompleks.

2.2.4.1 Kelebihan Algoritma Genetika

Beberapa hal yang termasuk kelebihan dari Algoritma Genetika adalah sebagai berikut:

1. Mengoptimalkan dengan variabel kontinu atau diskrit.
2. Tidak memerlukan informasi derivatif.
3. Bersamaan pencarian dari sebuah *sampling* yang luas pada permukaan biaya.
4. Berkaitan dengan sejumlah besar *variable.*
5. Baik untuk komputer paralel.

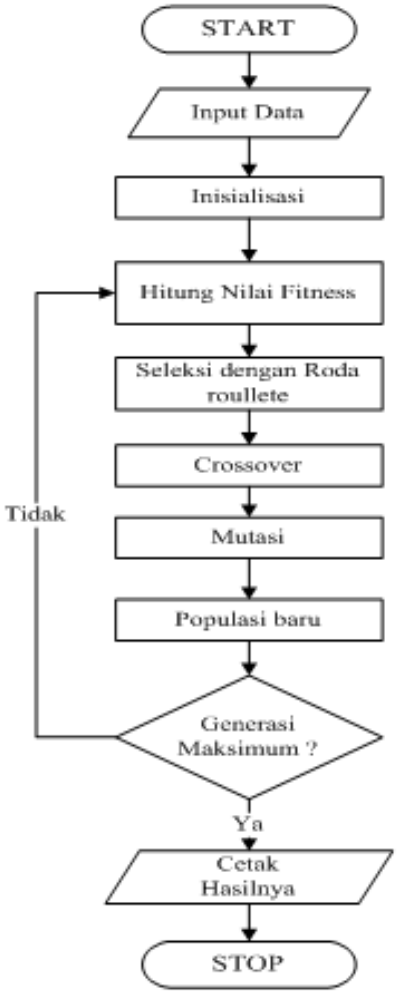
6. Mengoptimalkan permukaan *variable* dengan biaya yang sangat kompleks (GA bisa melompat dari minimum lokal).
7. Memberikan daftar *variable* yang optimal, bukan hanya solusi tunggal.
8. Dapat menyandikan *variable* sehingga optimasi dilakukan dengan mengkodekan *variable*.
9. Bekerja dengan data numerik yang dihasilkan, data eksperimen, atau analitis fungsi.

Algoritma genetika berangkat dari himpunan solusi yang dihasilkan secara acak yang disebut populasi. Sedangkan setiap individu dalam populasi disebut kromosom yang merupakan representasi dari solusi dan masing-masing dievaluasi tingkat ketanggguhannya (*fitness*) oleh fungsi yang telah ditentukan. Melalui proses seleksi alam atas operator genetik, gen-gen dari dua kromosom (disebut *parent*) diharapkan akan menghasilkan kromosom baru dengan tingkat *fitness* yang lebih tinggi sebagai generasi baru atau keturunan (*offspring*) berikutnya. Kromosom-kromosom tersebut akan mengalami iterasi yang disebut generasi (*generation*). Pada setiap generasi, kromosom dievaluasi berdasarkan nilai fungsi *fitness*. Setelah beberapa generasi maka algoritma genetika akan konvergen dapat kromosom terbaik, yang merupakan solusi optima.

2.2.4.2 Struktur Umum Algoritma Genetika

Algoritma genetika memberikan suatu pilihan bagi penentuan nilai parameter dengan meniru cara reproduksi genetik, pembentukan kromosom baru serta seleksi alami seperti terjadi pada mahluk hidup.

Struktur umum algoritma genetika dapat diilustrasikan dalam diagram alir berikut ini:



Gambar 2.2 Struktur Umum Algoritma Genetika

Inisialisasi populasi awal dilakukan untuk menghasilkan solusi awal dari suatu permasalahan algoritma genetika. Inisialisasi ini dilakukan secara acak sebanyak jumlah kromosom/populasi yang diinginkan. Selanjutnya dihitung nilai *fitness* dan seterusnya dilakukan seleksi dengan menggunakan metode roda *roullete*, *tournament* atau *ranking*. Kemudian dilakukan perkawinan silang (*crossover*) dan mutasi. Setelah melalui beberapa generasi maka algoritma ini akan berhenti sebanyak generasi yang diinginkan.

2.2.4.3 Karakteristik Algoritma Genetika

Goldberg (1989) mengemukakan bahwa algoritma genetika mempunyai karakteristik-karakteristik yang perlu diketahui sehingga dapat terbedakan dari prosedur pencarian atau optimasi yang lain yaitu:

1. AG bekerja dengan pengkodean dari himpunan solusi permasalahan berdasarkan parameter yang telah ditetapkan dan bukan parameter itu sendiri. Sebagai contoh untuk mendapatkan minimum dari fungsi $f(x)=y=x^4+2x^3+5$, AG tidak secara langsung mencari nilai x atau y, tetapi terlebih dahulu merepresentasikan x dalam bentuk *string* biner.
2. AG melakukan pencarian pada sebuah populasi dari sejumlah individu-individu yang merupakan solusi permasalahan bukan hanya dari sebuah individu.
3. AG merupakan informasi fungsi objektif (*fitness*), sebagai cara untuk mengevaluasi individu yang mempunyai solusi terbaik, bukan turunan dari suatu fungsi.
4. AG menggunakan aturan-aturan transisi peluang, bukan aturan-aturan deterministik.

Parameter yang digunakan pada algoritma genetika adalah:

1. Fungsi *fitness* (fungsi tujuan) yang dimiliki oleh masing-masing individu untuk menentukan tingkat kesesuaian individu tersebut dengan kriteria yang ingin dicapai.
2. Populasi jumlah individu yang dilibatkan pada setiap generasi.
3. Probabilitas terjadinya persilangan (*crossover*) pada suatu generasi.
4. Probabilitas terjadinya mutasi pada setiap individu.
5. Jumlah generasi yang akan dibentuk yang menentukan lama dari penerapan AG.

Sebagaimana halnya proses evolusi di alam, suatu algoritma genetika yang sederhana umumnya terdiri dari tiga operator yaitu : operator reproduksi, operator *crossover* (persilangan) dan operator mutasi.

2.2.4.4 Langkah-langkah Algoritma Genetika

Struktur umum dari suatu algoritma genetika dapat didefinisikan dengan langkah-langkah sebagai berikut:

1. Pengkodean

   Pengkodean disini meliputi pengkodean gen dan kromosom.

2. Inisisalisasi populasi awal

   Membangkitkan sejumlah kromosom (sesuai dengan ukuran populasi) untuk dijadikan anggota populasi awal. Populasi itu sendiri terdiri dari sejumlah kromosom yang merepresentasikan solusi yang diinginkan.

3. Evaluasi nilai *fitness*

   Setiap kromosom pada populasi dihitung nilai *fitness*-nya berdasarkan fungsi *fitness*. Nilai *fitness* suatu kromosom menggambarkan kualitas kromosom dalam populasi tersebut. Proses ini akan mengevaluasi setiap populasi dengan menghitung nilai *fitness* setiap kromosom dan mengevaluasinya sampai terpenuhi kriteria berhenti.

4. Pembentukan kromosom baru

   a. Seleksi

      Memilih sejumlah kromosom yang akan menjadi kromosom calon *parent*.

   b. *Crossover*

      Mengkombinasikan dua kromosom *parent* (induk) berdasar nilai probabilitas *crossover*-nya untuk menghasilkan *offspring*.

   c. Mutasi

      Mengubah sejumlah gen berdasar nilai probabilitas mutasinya untuk menghasilkan kromosom baru.

   d. *Update* Generasi

      Membaharui kromosom yang terdapat dalam populasi.

   e. Pengecekan faktor pemberhenti

      Jika memenuhi dari salah satu dari kondisi untuk berhenti, maka siklus algoritma genetika berhenti. Proses evolusi bisa dihentikan berdasarkan beberapa kondisi, misalnya ketika evolusi telah mencapai

jumlah generasi maksimum yang diizinkan, terdapat suatu individu yang telah memiliki fitness tertentu yang diharapkan, keberagaman populasi telah mencapai tingkat minimum yang diizinkan, dalam beberapa generasi tertentu, tidak ada peningkatan nilai *fitness* yang diharapkan.

Sebelum algoritma genetika dilakukan, ada dua hal penting yang harus dilakukan yaitu pendefinisian kromosom yang merupakan suatu solusi yang masih berbentuk simbol dan fungsi *fitness* atau fungsi obyektif. Dua hal ini berperan penting dalam algoritma genetika untuk menyelesaikan suatu masalah.[5]

2.3 *Tools* Analisis

2.3.1 Pemrograman Berorientasi Objek atau *Object Oriented Programming* (OOP)

Metodologi berorientasi objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya. Metodologi berorientasi objek merupakan suatu cara bagaimana sistem perangkat lunak dibangun melalui pendekatan objek secara sistematis. Metode berorientasi objek didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas. Metode berorientasi objek meliputi rangkaian aktivitas analisis berorientasi objek, perancangan berorientasi objek, pemrograman berorientasi objek, dan pengujian berorientasi objek.

Pada saat ini, metode berorientasi objek banyak dipilih karena metodologi lama banyak menimbulkan masalah seperti adanya kesulitan pada saat mentransformasi hasil dari satu tahap pengembangan ke tahap berikutnya, misalnya pada metode pendekatan terstruktur jenis aplikasi yang dikembangkan saat ini berbeda dengan masa lalu. Aplikasi yang dikembangkan pada saat ini sangat beragam (aplikasi bisnis, *real-time*, *utility*, dan sebagainya) dengan *platform* yang berbeda-beda, sehingga membutuhkan tuntutan kebutuhan metodologi pengembangan yang dapat mengakomodasi ke semua jenis aplikasi tersebut.

Keuntungan menggunakan metodologi berorientasi objek adalah sebagai berikut :

1. Meningkatkan Produktivitas

   Karena kelas dan objek yang ditemukan dalam suatu masalah dapat dipakai ulang untuk masalah lainnya yang melibatkan objek tersebut (*reusable*).

2. Kecepatan Pengembangan

   Karena sistem yang dibangun dengan baik dan benar pada saat analisis dan perancangan akan menyebabkan berkurangnya kesalahan pada saat pengkodean.

3. Kemudahan Pemeliharaan

   Karena dengan model objek, pola-pola yang cenderung tetap dan stabil dapat dipisahkan dengan pola-pola yang mungkin sering berubah-ubah.

4. Adanya Konsistensi

   Karena sifat pewarisan dan penggunaan notasi yang sama pada saat analisis, perancangan maupun pengkodean.

5. Meningkatkan Kualitas Perangkat Lunak

   Karena pendekatan pengembangan lebih dekat dengan dunia nyatadan adanya konsistensi pada saat pengembangannya, perangkat lunak yang dihasilkan akan mampu memenuhi kebutuhan pemakai serta mempunyai sedikit kesalahan.

Saat ini sudah banyak bahasa pemrograman berorientasi objek. Banyak orang berfikir bahwa pemrograman berorientasi objek identik dengan bahasa java. Memang bahasa java merupakan bahasa yang paling konsisten dalam mengimplementasikan paradigma pemrograman berorientasi objek. Namun sebenarnya bahasa pemrograman yang mendukung pemrograman berorientasi objek tidak hanya bahasa java.

2.3.2 *Unified Modeling Language* (UML)

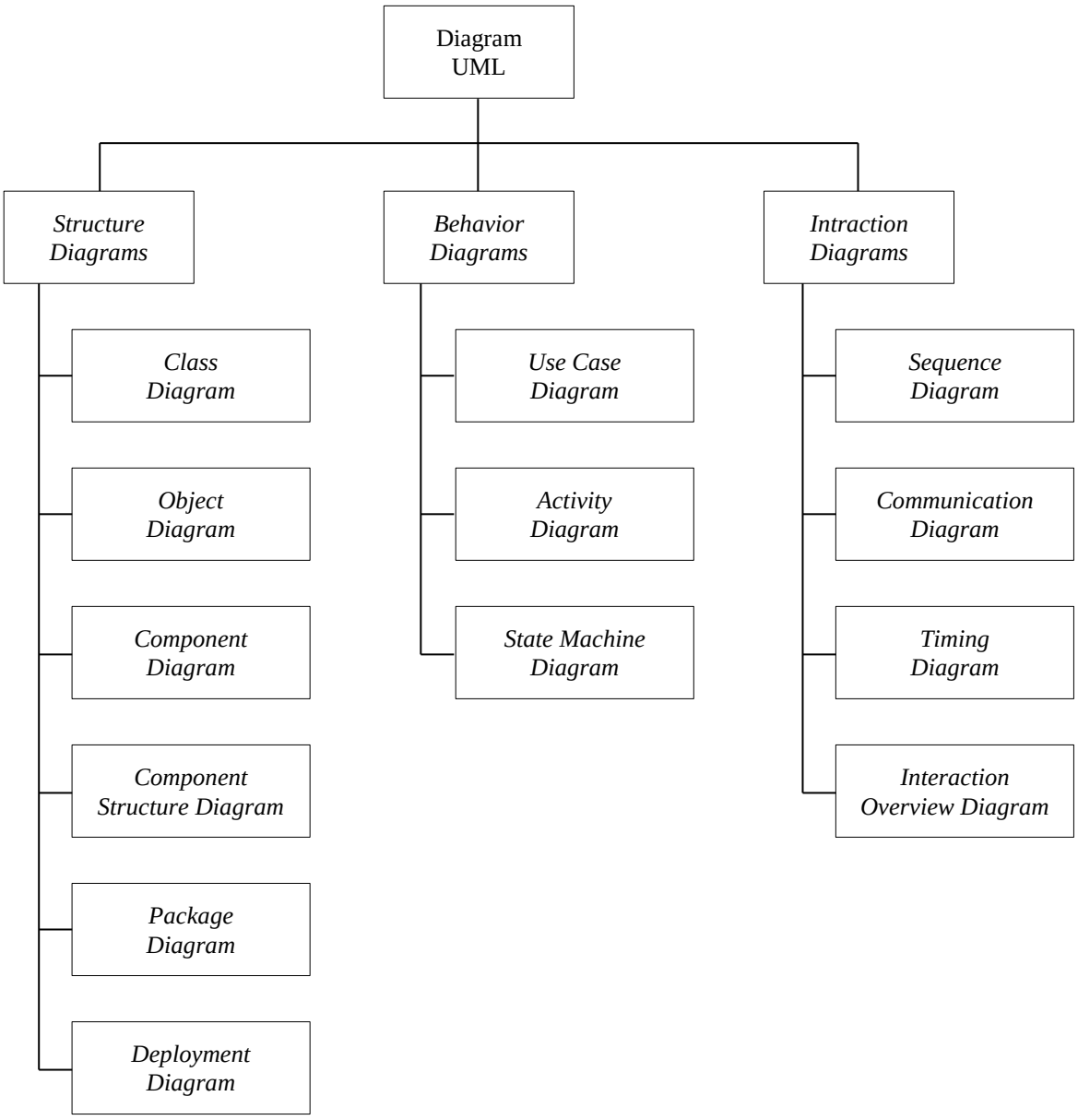
Pada perkembangan teknik pemrograman berorientasi objek, muncul sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang

dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modeling Language* (UML). UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung.

UML hanya berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada metodologi tertentu, meskipun pada kenyataanya UML paling banyak digunakan pada metodologi berorientasi objek.

Seperti banyak kita ketahui bahwa banyak hal di dunia informasi yang tidak dapat dibakukan, semua tergantung kebutuhan, lingkungan, dan konteksnya. Begitu juga dengan perkembangan penggunaan UML bergantung pada level abstraksi penggunanya. Jadi, belum tentu pandangan yang berbeda dalam penggunaan UML adalah suatu yang salah, tapi perlu ditelaah dimanakah UML digunakan dan hal apa yang ingin divisualkan. Secara analogi jika dengan bahasa yang kita gunakan sehari-hari, belum tentu penyampaian bahasa dengan puisi adalah hal yang salah. Sistem informasi bukanlah ilmu pasti, maka jika ada banyak perbedaan dan interpretasi di dalam bidang sistem informasi merupakan hal yang sangat wajar.

UML terdiri dari 13 macam diagram yang dikelompokkan dalam 3 kategori. Pembagian kategori dan macam-macam diagram tersebut dapat dilihat pada gambar di bawah.



Gambar 2.3 Diagram UML

Berikut ini penjelasan singkat dari pembagian kategori tersebut.

1. *Structure diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan.
2. *Behavior diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada sebuah sistem.

3. *Interaction diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antar subsistem pada suatu sistem.[6]

2.3.3 C# (*C Sharp*)

C# (*C sharp*) merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET *Framework*. Bahasa pemrograman ini dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya (seperti Java, Delphi, Visual Basic, dan lain-lain) dengan beberapa penyederhanaan. Menurut standar ECMA-334 C# *Language Specification*, nama C# terdiri atas sebuah huruf Latin C (U+0043) yang diikuti oleh tanda pagar yang menandakan angka # (U+0023). Tanda pagar # yang digunakan memang bukan tanda kres dalam seni musik (U+266F), dan tanda pagar # (U+0023) tersebut digunakan karena karakter kres dalam seni musik tidak terdapat di dalam *keyboard* standar.

1. Kesederhanaannya

   Pointer telah ‘dihilangkan’ dari C#. (diijinkan mengakses tetapi diberi tanda “*unsafe*”) Operasi-operasi yang ‘tidak aman’ seperti misalnya akses memori langsung sudah tidak diizinkan. Tidak ada lagi penggunaan operator “::” atau “->” seperti pada pendahulunya. Karena berada dalam lingkungan .NET, maka C# memiliki turunan manajemen memori otomatis dan *Garbage Collection*. Dukungan tipe data primitif yang lebih banyak seperti misalnya *Integer*, *Floats*, dan sebagainya. Nilai *integer* 0 dan 1 tidak diterima lagi sebagai alternatif nilai *Boolean*. Nilai *Boolean* adalah murni *true* atau *false*.

2. Modern

   C# didasarkan pada trend yang berkembang saat ini dan sangat ampuh serta mudah digunakan untuk membangun aplikasi yang bersifat interopable, skalabilitas, dan cepat. C# menyertakan dukungan bawaan yang dapat membuat setiap komponennya diintegrasikan kedalam suatu

web service yang dapat dipanggil dari aplikasi apa saja dan *platform* apa saja melalui *internet*.

3. *Object Oriented*

   C# mendukung enkapsulasi, *inheritance*, *polymorphism*, dan *interfaces* secara penuh. (*int*, *float*, *double*) bukanlah merupakan *object* dalam bahasa Java, tetapi C# memperkenalkan struktur (*struct*) yang memungkinkan tipe-tipe primitif dapat menjadi *object*.

4. *Type Safe*

   Pada C#, kita tidak dapat melakukan hal-hal yang 'tidak aman' seperti misalnya mengkonversi nilai *variable* bertipe *double* ke *boolean*. Kompiler secara otomatis akan mengkonversi setiap *variable* bertipe *Value* dengan angka Nol, dan setiap *variable* bertipe *Reference* dengan nilai *null*. Indeks *array* selalu dari Nol dan batasannya selalu diperhatikan. Tipe-tipe yang menghasilkan nilai *overflow* dapat dievaluasi.

5. *Interoperability*

   C# menyertakan dukungan bawaan pada COM dan aplikasi berbasis *windows*. Mengizinkan penggunaan *pointer* dengan batasan-batasan. *User* sekarang tidak lagi secara eksplisit mengimplementasikan *inteface* COM yang tidak diketahui, fitur-fitur seperti itu sudah ada pada C#. C# mengizinkan *user* untuk menggunakan *pointer* sebagai blok kode yang bersifat *unsafe*. Komponen yang dibuat oleh VB.NET atau bahasa pemrograman lain yang mendukung .NET dapat langsung digunakan oleh C#.

6. *Scalable and Updateable*

   .NET memperkenalkan *assembly* yang merupakan piranti yang menggambarkan dirinya sendiri melalui suatu *manifest*. *Manifest* menunjukkan identitas *assembly*, versi, bahasa, tandatangan digital, dan banyak lagi. *Assembly* tidak perlu lagi diregistrasikan ke dalam *registry windows*. Untuk proses *update* aplikasi, kita tidak perlu lagi melakukan registrasi menggunakan regsvr32.exe. Cukup hapus file yang lama, dan

copy-kan *file* yang baru. Proses update komponen software hampir dapat dilakukan tanpa kesulitan sama sekali.[7]

2.3.4 Microsoft Visual Studio

Visual Studio adalah program aplikasi keluaran dari Microsoft dimana dalam program *editing* tersebut di install maka bahasa pemrograman seperti VB.NET, C++.NET, C#.NET sudah tertera dengan pemrograman tersebut kita bisa men-*design* program dan megkodekan program teresebut sesuai keinginan kita. Sedangkan C#.NET bahasa pemrograman yang sudah menyatu dalam paket aplikasi Visual studio dimana bahasa prmrograman tersebut berbasis Dot NET.[8]



Gambar 2.4 Logo Visual Studio