BAB II
LANDASAN TEORI

2.1 Penjadwalan Kuliah

Jadwal menurut KBBI adalah pembagian waktu berdasarkan rencana pengaturan urutan kerja atau daftar atau tabel kegiatan atau rencana kegiatan dengan pembagian waktu pelaksanaan yang terperinci. Sedangkan penjadwalan adalah proses, cara, perbuatan menjadwalkan atau memasukkan dalam jadwal. Dalam penjadwalan perkuliahan terdapat beberapa batasan umum yang tidak boleh dilanggar seperti dosen tidak bisa mengajar dalam dua ruangan di waktu yang sama, ruangan yang sama tidak bisa dimasuki oleh dua kelas yang berbeda dalam satu waktu, bentrok waktu SKS ketika mata kuliah memiliki 3 SKS maka ruangan tersebut harus kosong selama 3 SKS. Jika ada satu saja batasan yang dilanggar maka jadwal tersebut tidak dapat dipakai. Menurut Rossi-Doria Olivia dan Paetcher Ben [6] dan dalam *University Course Timetabling Problem* (UCTP) batasan-batasan yang ada dalam penjadwalan perkuliahan dibagi menjadi dua yaitu *hard constraint* dan *soft constraint.*

2.2 Algoritma

Algoritma adalah urutan langkah-langkah untuk memecahkan suatu masalah[7]. Algoritma adalah deretan instruksi yang jelas untuk memecahkan masalah, yaitu untuk memperoleh keluaran yang diinginkan dari satu masukan dalam jumlah waktu yang terbatas[8]. Algoritma adalah prosedur komputasi yang redefinisi dengan baik yang menggunakan beberapa nilai masukan dan menghasilkan beberapa nilai yang disebut keluaran[9]. Jadi, algoritma adalah deretan langkah yang mentransformasikan masukan menjadi kelautan. Algoritma berfungsi untuk mempermudah kerja atau memudahkan kita untuk menyusun suatu program. Selain itu, algoritma juga dapat mengatasi masalah logika dan matematika termasuk dalam masalah penjadwalan. Pemilihan algoritma yang tepat dalam penyusunan jadwal berpengaruh dengan hasil dari jadwal itu sendiri[7].

2.3 Algoritma Genetika

Menurut David E. Goldberg [10] algoritma adalah algoritma komputasi yang di inspirasi teori evolusi Darwin yang menyatakan bahwa kelangsungan hidup suatu makhluk dipengaruhi aturan "yang kuat adalah yang menang". Darwin juga menyatakan bahwa kelangsungan hidup suatu makhluk dapat dipertahankan melalui proses reproduksi, *crossover*, dan mutasi. Konsep dalam teori evolusi Darwin tersebut kemudian diadopsi menjadi algoritma komputasi untuk mencari solusi satu permasalahan dengan cara yang lebih "alamiah". Sebuah solusi yang dibangkitkan dalam algoritma genetika disebut sebagai kromosom, sedangkan kumpulan kromosom-kromosom tersebut disebut sebagai populasi. Sebuah kromosom dibentuk dari komponen-komponen penyusun yang disebut sebagai gen dan nilainya dapat berupa bilangan numerik, biner, simbol ataupun karakter tergantung dari permasalahan yang ingin diselesaikan. Kromosom-kromosom tersebut akan berevolusi secara berkelanjutan yang disebut dengan generasi. Dalam tiap generasi kromosom-kromosom tersebut dievaluasi tingkat keberhasilan nilai solusinya terhadap masalah yang ingin diselesaikan (*fungsi_objektif*) menggunakan ukuran yang disebut dengan *fitness*.

Untuk memilih kromosom yang tetap dipertahankan untuk generasi selanjutnya dilakukan proses yang disebut dengan seleksi. Proses seleksi kromosom menggunakan konsep aturan evolusi Darwin yang telah disebutkan sebelumnya yaitu kromosom yang mempunyai nilai *fitness* tinggi akan memiliki peluang lebih besar untuk terpilih lagi pada generasi selanjutnya. Kromosom-kromosom baru yang disebut dengan *offspring*, dibentuk dengan cara melakukan perkawinan antar kromosom-kromosom dalam satu generasi yang disebut sebagai proses *crossover*. Jumlah kromosom dalam populasi yang mengalami *crossover* ditentukan oleh *paramater* yang disebut dengan *crossover_rate* (probabilitas *crossover*). Mekanisme perubahan susunan unsur penyusun makhluk hidup akibat adanya faktor alam yang disebut dengan mutasi direpresentasikan sebagai proses berubahnya satu atau lebih nilai gen dalam kromosom dengan suatu nilai acak. Jumlah gen dalam populasi yang mengalami mutasi ditentukan oleh parameter yang dinamakan *mutation_rate* (probabilitas mutasi). Setelah beberapa generasi akan

dihasilkan kromosom-kromosom yang nilai gen-gennya konvergen ke suatu nilai tertentu yang merupakan solusi terbaik yang dihasilkan oleh algoritma genetika terhadap permasalahan yang ingin diselesaikan.

2.3.1 Pembangkitan

Pembentukan populasi awal yang bekerja pada populasi, yaitu kumpulan individu yang di dalamnya terdapat kromosom atau solusi yang akan diperbaharui pada setiap generasinya. Pembentukan populasi awal yang berisi kumpulan individu sebanyak ukuran populasi. Proses *inisialisasi* ini biasanya dilakukan secara acak.

2.3.2 Fungsi *Fitness*

Fungsi penalti Fungsi penalti memiliki nilai suatu kromosom yang menentukan kualitas atau kelayakan solusi masalah. Nilai penalti menunjukkan bobot pelanggaran terhadap suatu kendala atau batasan. Dalam masalah optimasi, nilai penalti biasanya diminimumkan sehingga diperoleh solusi dengan jumlah pelanggaran kendala yang paling sedikit. Semakin kecil nilai penalti maka akan semakin meningkatkan kualitas solusi, sedangkan kelayakan solusi ditentukan oleh kendala suatu nilai penalti. Rumus fungsi *fitness* yang digunakan adalah persamaan 2.1 :

$$Fitness = \frac{1}{(1 + \sum_{i=1}^{5} penalty - i)} \tag{2.1}$$

2.3.3 Seleksi

Seleksi orang tua Seleksi orang tua merupakan bagian evolusi yaitu proses mendekati kandidat solusi permasalahan yang diharapkan. Seleksi orang tua adalah operator algoritma *memetika* yang dilakukan pada setiap generasi untuk memilih kromosom-kromosom dari populasi yang akan menjadi orang tua bagi generasi berikutnya. Menurut Sri Kusumadewi[15] ,langkah pertama yang dilakukan dalam seleksi ini adalah pencarian nilai *fitness*. Masing-masing individu dalam suatu tempat seleksi akan menerima probabilitas reproduksi yang tergantung pada nilai

objektif individu tersebut terhadap semua individu dalam tempat seleksi tersebut. Metode *Rank Selection* merupakan salah satu metode seleksi yang dimiliki algoritma genetika. Metode *Rank Selection* bekerja dengan terlebih dahulu me-*ranking* atau mengurutkan kromosom di dalam populasi berdasarkan *fitness*-nya. Kemudian memberi nilai *fitness* baru berdasarkan urutannya. Kromosom dengan *fitness* terburuk akan memiliki *fitness* baru yang bernilai 1, terburuk kedua bernilai 2, dan seterusnya. Sehingga kromosom yang memiliki *fitness* terbaik akan memiliki nilai *fitness* yang sama dengan adalah jumlah kromosom dalam sebuah populasi. Setelah proses pengurutan dan pemberian nilai *fitness* baru, setiap kromosom akan memiliki kesempatan yang lebih kecil untuk terpilih, sehingga metode ini menyebabkan konvergensi menjadi lambat.

2.3.4 *Crossover*

Pindah silang *(crossover)* Pindah silang *(crossover)* merupakan komponen paling penting dalam algoritma genetika dan *memetika* karena proses inilah yang dapat mendekati kromosom-kromosom sebagai kandidat solusi yang diharapkan. Kromosom yang mengarah pada nilai penalti yang lebih bagus diperoleh dari proses memindah silangkan dua buah kromosom yang diperoleh dari seleksi orang tua. Pindah silang dapat menghasilkan satu atau beberapa kromosom yang biasa disebut kromosom anak. Pindah silang dapat berakibat buruk jika jumlah populasinya sangat kecil karena suatu kromosom dengan gen-gen yang mengarah ke solusi (kromosom yang diharapkan) akan sangat cepat menyebar ke kromosom-kromosom lainnya. Untuk mengatasi masalah tersebut pindah silang hanya dapat dilakukan untuk probabilitas tertentu. Pindah silang dilakukan jika suatu bilangan random (0,1) yang dibangkitkan kurang dari yang ditentukan. Pada umumnya, ditentukan mendekati satu, misal 0,8. Menurut Suhartono E[17], nilai probabilitas *crossover* berkisar antara 85% hingga 90%. Menurut Suyanto [16], pindah silang yang paling sederhana adalah *one point crossover*. Dalam *one point crossover* dua titik dipilih secara random untuk membagi orang tua yang pertama, kemudian gen pertama hingga titik yang terpilih diturunkan ke kromosom anak sesuai posisinya,

sedangkan orang tua yang lainnya diperiksa, jika terdapat nilai gen yang belum ada pada anak, maka ditambahkan ke kromosom anak dengan cara *left-to-right*.

2.3.5 Mutasi

Mutasi merupakan proses pertukaran atau perubahan informasi yang dibawa oleh masing-masing gen dalam kromosom anak hasil pindah silang. Mutasi dilakukan pada beberapa gen dalam kromosom anak dengan probabilitas mutasi yang relatif kecil. Peluang mutasi didefinisikan sebagai persentase dari jumlah total gen pada populasi yang mengalami mutasi. Peluang mutasi mengendalikan banyaknya gen baru yang akan dimunculkan untuk dievaluasi. Jika peluang mutasi terlalu kecil, banyak gen yang mungkin berguna tidak pernah dievaluasi. Apabila peluang mutasi terlalu besar maka akan terlalu banyak gangguan acak, sehingga anak akan kehilangan kemiripan dari induknya. Biasanya sebesar dengan *n* adalah jumlah gen dalam kromosom. Proses mutasi dikendalikan oleh nilai probabilitas mutasi. Menurut Suhartono E[17], untuk nilai probabilitas mutasi disarankan kecil antara 5% hingga 10%. Mutasi berperan untuk menggantikan gen yang hilang dari populasi akibat proses seleksi yang memungkinkan munculnya kembali gen yang tidak muncul pada *inisialisasi* populasi.

2.4 Pencarian Lokal

Pencarian lokal merupakan pertukaran atau penempatan kembali informasi yang di bawa oleh gen-gen dalam satu kromosom anak dengan harapan dapat meningkatkan kualitas kromosom tersebut. Pencarian lokal dapat dilakukan dengan menukar dua gen atau permutasi beberapa gen tanpa mengurangi kualitas dari kromosom sebelumnya. Dalam kasus tertentu, pencarian lokal hanya dilakukan kepada gen yang terkena penalti, artinya hanya informasi yang kurang tepat yang dipertukarkan antar gen dalam satu kromosom yang sama[1].

Menurut Rossi-Doria Olivia dan Paetcher Ben [6], dalam penjadwalan pencarian lokal yang efektif terdiri atas dua tahap yaitu tahap memperbaiki dan tahap meningkatkan kualitas jadwal. Penjelasan tahap tersebut adalah sebagai berikut.

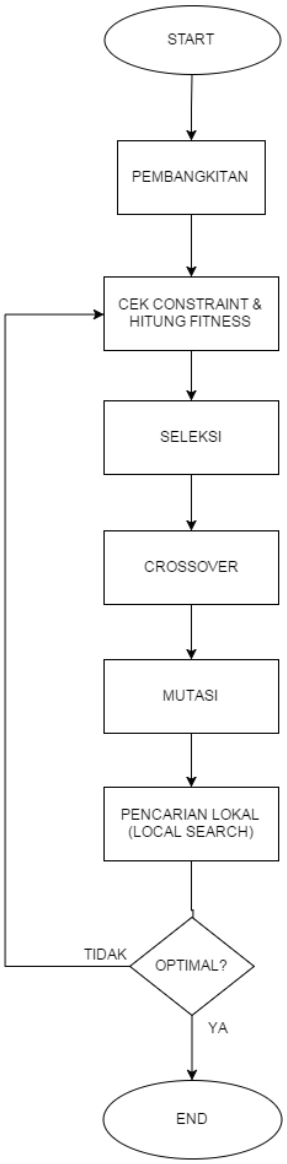
a. Tahap memperbaiki jadwal dari yang tidak layak menjadi layak adalah dengan mengurangi *slot* waktu yang digunakan.
b. Tahap meningkatkan kualitas jadwal layak adalah dengan mengurangi pelanggaran pada kendala *soft*.

Menurut Sadaf N. Jat, Shengxiang Yang [2], proses pencarian lokal dibagi menjadi 3 tahapan:

a. Mengganti *slot* waktu di dalam gen secara berurutan.
b. Menukar *slot* waktu dari 2 kromosom di dalam 1 individu.
c. Melakukan permutasi.

2.5 Algoritma *Memetika*

Algoritma *Memetika* (AM) adalah perluasan dari Algoritma Genetika. Kelebihan dari AG adalah pada cara kerjanya yang paralel. AG bekerja dalam ruang pencarian yang menggunakan banyak individu sekaligus, sehingga kemungkinan AG untuk terjebak pada ekstrem lokal lebih kecil dibandingkan metode lain. Kekurangan dari AG adalah dalam hal waktu komputasi karena harus melakukan evaluasi *fitness* pada semua solusi di setiap iterasi, sehingga AG bisa lebih lambat dibandingkan dengan metode lain. [5] Namun telah diketahui bahwa AG tidak cukup baik untuk mencari solusi yang sangat dekat dengan solusi optimal. Kekurangan ini dapat di akomodasi dengan cara menambahkan metode *Local Search* pada AG. Hasil penggabungan ini dikenal dengan sebutan Algoritma *Memetika* (AM). AM adalah suatu metode pencarian *heuristic* yang memiliki karakteristik yang sama dengan AG dikombinasikan dengan metode *Local Search* yang secara bersama-sama dapat meningkatkan kualitas pencarian solusi (Moscato, 1989). Pada AM, *Local Search* bertujuan untuk melakukan perbaikan lokal yang dapat diterapkan sebelum dan atau sesudah proses seleksi, *crossover*, dan mutasi. *Local Search* juga dapat berguna untuk mengontrol besarnya ruang pencarian solusi. AM dapat memberikan hasil yang lebih baik daripada AG, namun memerlukan waktu komputasi yang lebih lama[1]. Untuk alur dari algoritma *memetika* ini bisa dilihat di gambar 2.1 sebagai berikut:



Gambar 2. 1 Alur Algoritma *Memetika*

2.6 *Unified Modelling Language* (UML)

UML adalah suatu bahasa yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi [10]. UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson. Namun demikian UML dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi. Penggunaan UML dalam industri terus meningkat. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam

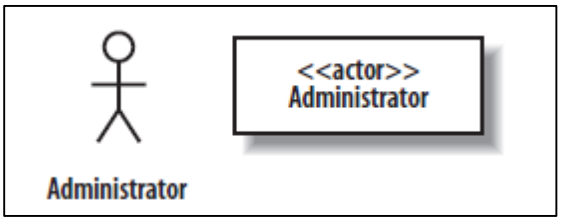
industri peranti lunak dan pengembangan sistem. Proses pemodelan data menggambarkan keseluruhan proses bisnis yang akan dilakukan oleh sebuah sistem informasi. Proses model juga menjelaskan data-data yang terlibat dalam proses tersebut. Salah satu pemodelan data adalah dengan UML *(Unified Modelling Language).*

2.6.1 *Usecase* Diagram

Usecase adalah kasus atau situasi di mana sistem digunakan untuk memenuhi salah satu atau lebih dari kebutuhan pengguna. *Usecase* menangkap sepotong fungsionalitas yang sistem sediakan. *Usecase* merupakan jantung dari sebuah model, karena *usecase* mempengaruhi dan membimbing semua elemen lain dalam sistem [11].

a. Aktor

Aktor digambarkan dalam UML, menggunakan notasi *stickman* atau *stereotyped box* dan berlabel dengan nama yang tepat seperti gambar 2.2 berikut [11].



Gambar 2. 2 Aktor.

b. Usecase

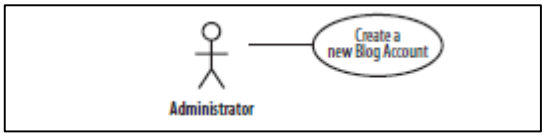
Sebuah *usecase* atau pekerjaan, dapat sesimpel memperbolehkan pengguna untuk *log-in* atau dapat serumit mengeksekusi transaksi *database* global yang terdistribusi. Gambar di bawah ini menunjukkan bagaimana interaksi ditangkap menggunakan *usecase* [11]. Contoh bisa dilihat di gambar 2.3 sebagai berikut :



Gambar 2. 3 *Usecase*.

c. Communication line

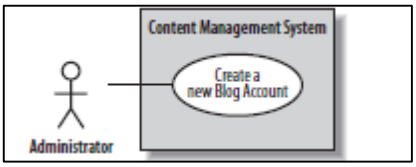
Sebuah *communication line* menghubungkan aktor dan *usecase* untuk menunjukkan aktor yang berpartisipasi dalam *usecase*. Sebagai contoh, aktor Administrator terlibat dalam *usecase Create a new Blog Account* [11]. Contoh bisa dilihat di gambar 2.4 sebagai berikut :



Gambar 2. 4 *Communication Line.*

d. System boundaries

Untuk menampilkan *system's boundary* pada *usecase* diagram, gambar sebuah kotak di sekitar semua *usecase* tapi aktor tetap berada di luar kotak [11]. Contoh bisa di lihat di gambar 2.5 sebagai berikut :



Gambar 2. 5 System *Boundaries*.

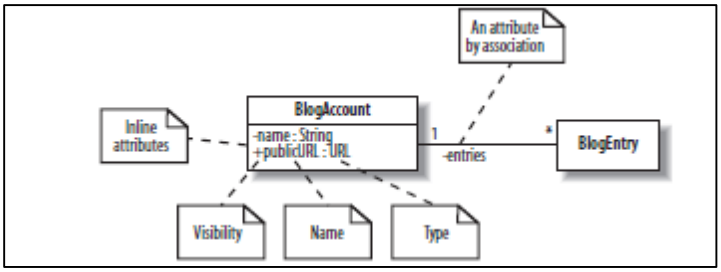
2.6.2 *Activity* Diagram

Usecase menggambarkan perilaku sistem. kelas menggambarkan berbagai jenis benda-benda yang dibutuhkan dalam sistem. Diagram aktivitas menunjukkan proses bisnis dari sistem yang sedang berjalan. Diagram aktivitas berupa urutan proses dari proses bisnis yang ada.

2.6.3 *Class* Diagram

a. Atribut

Sebuah *class's attributes* adalah potongan informasi yang mewakili keadaan suatu objek. Atribut ini dapat direpresentasikan pada diagram kelas baik dengan menempatkan mereka di dalam bagian dari kotak atau oleh asosiasi dengan kelas lain, seperti yang di tunjukan pada gambar 2.6 berikut [11].



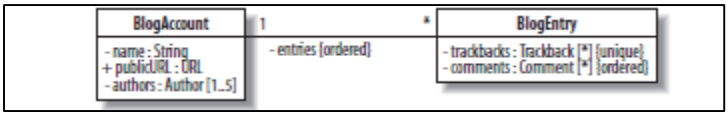
Gambar 2. 6 Atribut.

b. Nama dan Tipe

Dalam atribut nama dapat menjadi bagian dari karakter, tapi tidak boleh ada dua atribut dalam kelas yang sama dan mempunyai nama yang sama. Jenis atribut dapat bervariasi tergantung bagaimana kelas akan diimplementasikan dalam sistem [11].

c. Multiplicity

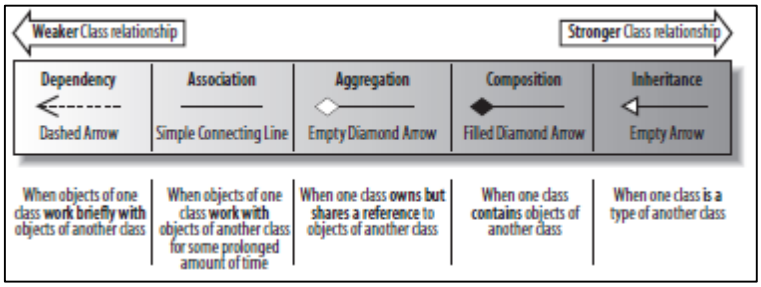
Terkadang atribut akan mewakili lebih dari satu objek. Bahkan atribut bisa mewakili sejumlah objek, seperti mendeklarasikan bahwa atribut adalah *array*. *Multiplicity* memungkinkan untuk menentukan bahwa atribut sebenarnya merupakan koleksi benda-benda dan dapat diterapkan baik *inline* dan atribut oleh asosiasi, seperti ditunjukkan pada gambar 2.7 berikut [11].



Gambar 2. 7 *Multiplicity*.

d. *Class Relationship*

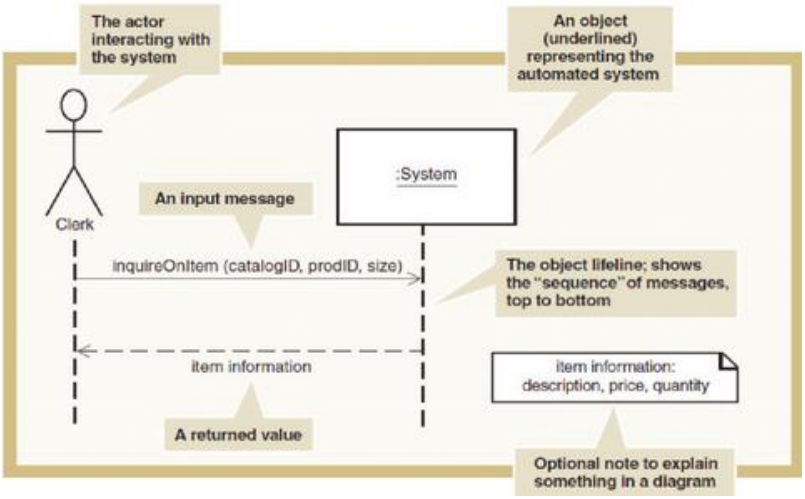
Kelas tidak hidup di ruang hampa, mereka bekerja bersama-sama menggunakan berbagai jenis hubungan. Hubungan antar kelas datang dalam kekuatan yang berbeda, seperti ditunjukkan pada gambar 2.8 di bawah ini [11].



Gambar 2. 8 *Class Relationship.*

2.6.4 *Sequence* Diagram

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan lain-lain) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internaldan *output* apa yang dihasilkan. Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metode dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*. Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk objek *boundary, controller* dan *persistent entity*[13]. Contoh *sequence diagram* bisa dilihat di gambar 2.9 sebagai berikut :



Gambar 2. 9 *Sequence Diagram*.

2.7 *Database*

Database atau basis data adalah kumpulan data yang disimpan secara sistematis di dalam komputer dan dapat diolah atau dimanipulasi menggunakan perangkat lunak (program aplikasi) untuk menghasilkan informasi. Pendefinisian basis data meliputi spesifikasi berupa tipe data, struktur, dan juga batasan-batasan data yang akan disimpan. Basis data merupakan aspek yang sangat penting dalam sistem informasi di mana basis data merupakan gudang penyimpanan data yang akan diolah lebih lanjut. Basis data menjadi penting karena dapat menghindari duplikasi data, hubungan antar data yang tidak jelas, organisasi data, dan juga *update* yang rumit. Dalam aplikasi yang penulis buat, maka penulis menggunakan *database* yang disebut MySQL[12].

2.8 OOP (*Obejct Oriented Programming*)

Pemrograman berorientasi objek (OOP) merupakan paradigma pemrograman yang menganggap semua benda adalah objek, yaitu menganggap segala sesuatu dalam bahasa pemrograman adalah objek, bahkan *variable* pun dianggap sebagai suatu objek. Konsep dasar dari pemrograman berorientasi objek menekankan kepada konsep: Objek (*Object*) "Objek merupakan kumpulan variabel dan fungsi yang saling berkaitan"[14].

1. Kelas (*Class*) "*Class* merupakan suatu rancangan dari objek itu sendiri. Jadi satu *class* dapat berisi banyak objek."
2. Pembungkusan (*Encapsulation*) "*Encapsulation* merupakan cara menyembunyikan detail informasi dan fungsionalitas yang ada pada suatu kelas."
3. Pewarisan (*Inheritance*) "*Inheritance* merupakan kemampuan untuk meng-*extends class* yang akan mewarisi semua karakter dari *partner class*-nya."
4. Polimorfisme (*Pholymorphism*) "*Pholymorphism* merupakan kemampuan objek-objek yang berbeda kelas, tetapi terkait dalam perwarisan untuk merespons secara berbeda terhadap suatu pesan yang sama.

2.9 *Tools* Pembangunan Sistem

2.9.1 *C Sharp* (C#)

C# (dibaca: 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 # dalam seni musik (U+266F), dan tanda pagar # (U+0023) tersebut digunakan karena karakter # dalam seni musik tidak terdapat di dalam *keyboard* standar[12].

2.9.2 *Microsoft Visual Studio*

Untuk membuat sebuah aplikasi penjadwalan yang menggunakan bahasa *C#* maka diperlukan sebuah perangkat lunak yang bernama *Microsoft Visual Studio. Microsoft Visual Studio* merupakan sebuah perangkat lunak lengkap (*suite*) yang dapat digunakan untuk melakukan pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya, dalam bentuk aplikasi *console*, aplikasi *Windows*, ataupun aplikasi *Web*. *Visual Studio* mencakup *kompiler*, SDK,

Integrated Development Environment (IDE), dan dokumentasi (umumnya berupa *MSDN Library*). *Kompiler* yang dimasukkan ke dalam paket *Visual Studio* antara lain *Visual C++, Visual C#, Visual Basic, Visual Basic .NET, Visual InterDev, Visual J++, Visual J#, Visual FoxPro,* dan *Visual SourceSafe. Microsoft Visual Studio* dapat digunakan untuk mengembangkan aplikasi dalam *native code* (dalam bentuk bahasa mesin yang berjalan di atas *Windows*) ataupun *managed code* (dalam bentuk *Microsoft Intermediate Language* di atas *.NET Framework*). Selain itu, *Visual Studio* juga dapat digunakan untuk mengembangkan aplikasi *Silverlight*, aplikasi *Windows Mobile* (yang berjalan di atas *.NET Compact Framework*).

Visual Studio kini telah menginjak versi *Visual Studio 9.0.21022.08*, atau dikenal dengan sebutan *Microsoft Visual Studio 2008* yang diluncurkan pada 19 November 2007, yang ditujukan untuk platform *Microsoft .NET Framework 3.5*. Versi sebelumnya, *Visual Studio 2005* ditujukan untuk platform *.NET Framework 2.0* dan *3.0. Visual Studio 2003* ditujukan untuk *.NET Framework 1.1*, dan *Visual Studio 2002* ditujukan untuk *.NET Framework 1.0*. Versi-versi tersebut di atas kini dikenal dengan sebutan *Visual Studio .NET*, karena memang membutuhkan *Microsoft .NET Framework.* Sementara itu, sebelum muncul *Visual Studio .NET*, terdapat *Microsoft Visual Studio 6.0 (VS1998)*. Dalam pembangunan sistem informasi perpustakaan ini digunakan *Microsoft Visual Studi 2010* dan menggunakan versi *.NET Framework 4.0*[12].

2.9.3 *MySQL*

MySQL adalah sebuah sistem manajemen *database* relasi (*relation database management system*) yang bersifat *open source. MySql* merupakan buah pikiran dari Michael "Monty" Widenius, Daveid axmark dan Allan Larson yang dimulai tahun 1995. Mereka bertiga kemudian mendirikan perusahaan bernama *MySQL AB* di Swedia. *MySQL* merupakan aplikasi *database server*. Perkembangannya disebut SQL yang merupakan kepanjangan dari *Structured Query Language*. *SQL* merupakan bahasa terstruktur yang digunakan untuk mengolah *database*. *MySQL* dapat digunakan untuk membuat dan mengelola

database beserta isinya. Kita dapat memanfaatkan *MySQL* untuk menambahkan, mengubah, dan menghapus data yang berada dalam *database*.

MySQL versi 1.0 rilis pada Mei 1996 dan penggunaannya hanya terbatas di kalangan perusahaan saja. Barulah pada bulan Oktober 1996, *MySQL* versi 3.11.0 di rilis ke masyarakat luas. *MySQL* menggunakan bahasa standar *SQL*(*Structure Query Languange*) sebagai bahasa interaktif dalam mengelola data. *MySQL* memiliki kinerja, kecepatan proses dan ketangguhan yang tidak kalah dibanding *database-database* besar lainnya yang komersial seperti *ORACLE*, *Sysbase*, *Unify* dan lain-lain. *MySQL* dapat berjalan di atas banyak sistem operasi seperti *Linux,Windows*, *Solaris*, *FreeBSD*, *Mac OS X*, dan lain-lain[12].