# BAB II
# LANDASAN TEORI

## II.1 Perangkat Lunak

*Software* adalah perintah (program komputer) yang dieksekusi memberikan fungsi dan petunjuk kerja seperti yang diinginkan. Struktur data yang memungkinkan program memanipulasi informasi secara proporsional dan dokumen yang menggambarkan operasi dan kegunaan program. *Software* memiliki dua peran, disatu sisi berfungsi sebagai sebuah produk dan disisi lain sebagai pengontrol pembuatan sebuah produk. Sebagai produk, *software* mengantarkan potensi perhitungan yang dibangun oleh *software* komputer [15] .

*Software* merupakan transformer informasi yang memproduksi mengatur, memperoleh, memodifikasi, menampilkan atau memancarkan informasi ini dapat sesederhana bit tunggal atau sekompleks sebuah simulasi multimedia. Sedangkan peran sebagai pengontrol yang dipakai untuk mengantarkan produk, *software* berlaku sebagai dasar untuk kontrol komputer (sistem operasi), komunikasi informasi (jaringan), dan penciptaan serta kontrol dari program-program lain (peranti dan lingkungan *software*) [15] .

## II.2 Domain Perangkat Lunak

Domain perangkat lunak merupakan perangkat lunak yang berada pada lingkup kerja tertentu. Area subjek, dimana pengguna mengimplementasikan ke program itulah yang disebut dengan domain [4]. Beberapa domain melibatkan dunia nyata, seperti domain dari program pemesanan penerbangan yang melibatkan orang-orang di dunia nyata untuk memesan sebuah pesawat terbang yang nyata. Domain dari program akutansi adalah uang dang keuangan. Domain perangkat lunak biasanya tidak ada hubungannya dengan computer, meskipun ada beberapa pengecualan. Domain Dari *source-code control system* adalah pembangunan dari *software* itu sendiri [4].

Untuk membuat perangkat lunak yang menyangkut tentang sebuah domain, pengembang/tim pengembang harus memiliki pengetahuan tentang aktivitas di dalam domain tersebut. Luasnya pengetahuan yang dibutuhkan bias menjadi sesuatu yang sangat membingungkan. Volume dan kompleksitas informasi bias sangat banyak. Oleh karena itu domain model itu ada untuk menampung kelebihan beban tersebut [4].

Domain model adalah model yang merupakan bentuk pengetahuan dalam domain yang disederhanakan dan secara sadar dan terstruktur. Model yang tepat akan memunculkan informasi dan memfokuskannya pada suatu masalah [4]. Sebuah domain model bukan hanya diagram saja, namun itu merupakan sebuah gagasan bahwa diagram merupakan sebuah alat yang ditujukan untuk menyampaikan apa yang dimaksud [4].

## II.1 Aplikasi Agregator Logistik

Aplikasi agregator logistik merupakan aplikasi dimana para penyedia logistik dan pelanggan diberikan sebuah tempat untuk melakukan transaksi logistik. Aplikasi ini menjembatani antara perusahaan, warehouse, transporter serta pemilik logistik untuk dapat berinteraksi langsung dengan pelanggan yang membutuhkan jasa logistik [16].

## II.2 Pemrograman Berorientasi Objek

Pemrograman berorientasi objek (O*bject-Oriented Programming* disingkat OOP) merupakan paradigma pemrograman yang berorientasikan kepada objek, dimana semua data dan fungsi dibungkus dalam kelas-kelas atau objek-objek. Setiap objek dapat menerima pesan, memproses data, mengirim, menyimpan dan memanipulasi data. Beberapa objek berinteraksi dengan saling memberikan informasi satu terhadap lainya. Masing-masing object harus berisikan informasi mengenai dirinya sendiri dan dapat dihubungkan dengan object yang lain [17]. Konsep dasar dari pemrograman berorientasi objek antara lain:

1. *Class*

Merupakan model yang berisi kumpulan attribute dan *method* dalam suatu *unit* untuk suatu tujuan tertentu. Sebagai contoh class manusia memiliki attribute berat, tinggi, usia kemudian memiliki *method* makan, minum, tidur. *Method* dalam sebuah kelas dapat merubah *attribute* yang dimiliki oleh class tersebut. Sebuah *class* merupakan dasar dari modularitas dan struktur dalam pemrograman berorientasi objek [17].

2. *Method*

*Method* adalah fungsi atau prosedur yang dibuat oleh seorang programmer di dalam suatu class. Dengan kata lain, *method* pada sebuah class hampir sama dengan fungsi atau prosedur pada pemrograman prosedural. Pada sebuah *method* di dalam sebuah class juga memiliki izin akses seperti halnya attribute [18].

3. *Object*

Membungkus data dan fungsi bersama menjadi suatu *unit* dalam sebuah program komputer. Objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek [18].

4. *Abstraction*

Adalah suatu cara untuk melihat suatu object dalam bentuk yang lebih sederhana. Dengan abstraksi, suatu sistem yang kompleks dapat dipandang sebagai kumpulan subsistem-subsistem yang lebih sederhana [17].

5. *Encapsulation*

Merupakan suatu mekanisme untuk menyembunyikan atau memproteksi suatu proses dari kemungkinan interferensi atau penyalahgunaan dari luar sistem dan sekaligus menyederhanakan penggunaan sistem tersebut. Memastikan pengguna sebuah *object* tidak dapat mengganti keadaan dalam dari sebuah *object* dengan cara yang tidak layak. Hanya metode dalam *object* tersebut yang diberi ijin untuk mengakses keadaannya. Setiap *object* mengakses *interface* yang menyebutkan bagaimana *object* lainnya dapat berinteraksi dengannya. *Object* lainnya tidak akan mengetahui dan tergantung kepada representasi dalam object tersebut [17].

1. *Polymorphism*

Dengan polymorphism, 2 objek yang berbeda dapat menerima pesan yang sama namun dengan aksi yang berbeda. Dalam pemrograman berorientasi objek kita dapat menerapkan jenis polymorphism yang bernama *overloading* yaitu penerapan metode atau aksi yang berbeda pada objek yang memiliki nama yang sama. Objek tersebut dapat membedakan dari objek yang memiliki nama yang sama tersebut mana objek yang harus dijalankan seperti membedakan berdasarkan parameter [19].

2. *Inheritance*

Sebuah pewarisan fungsi dan karakteristik dari objek yang umum pada objek yang lebih spesifik seperti objek orangtua ke objek anak sehingga objek anak tersebut mewarisi fungsi dan karakteristik yang ada pada objek orangtua dan menggunakannya. Contohnya yaitu objek bangun datar sebagai objek orangtua yang memiliki fungsi luas dan keliling, dan objek segitiga menjadi objek anak yang mempunyai karakteristik alas dan tinggi, sehingga dengan inheritance objek segitiga dapat memakai fungsi luas dan keliling [19].

## II.1 .NET CORE

.NET CORE sering dibaca *dot net core*, adalah *software framework* yang gratis dan *open-source* untuk Windows, macOS dan Linux. Terdiri dari Core CLR (Common Runtime Framework). Core CLR sendiri merupakan sebuah implementasi sempurna dari CLR dimana mesin virtual mengatur eksekusi dari program-program .NET. .NET Core sendiri berbagi fungsionalitas API dari .NET Framework namun mempunyai API tersendiri yang bukan merupakan bagian dari .NET Framework [20].

## II.2 *Class Library*

*Class library* secara literal dapat diartikan dengan perpustakaan kelas. Perpustakaan kelas yang dimaksud adalah kumpulan dari kelas-kelas atau template kode yang digunakan programmer dalam membuat sebuah program. *Class Library* sendiri hanya terdapat di dalam OOP (*Object Oriented Programming* atau Pemrograman berbasis Objek) [21].

## II.1 Pedoman pembangunan *Class Library*

Dalam membangun sebuah kelas library menggunakan bahasa c# dengan teknologi .NET CORE dipastikan ada aturan-aturan yang membatasi programmer seperti aturan penulisan dan lain-lain. Tujuan dari pedoman ini adalah membatu desainer dari *class library* untuk memastikan bahwa kelas library yang dibangun konsisten dengan kelas yang telah ada dan tidak merugikan produktifitas pengembang. Oleh karena itu berikut ini adalah beberapa hal yang harus diperhatikan dalam membangun *class library*.

### II.7.1 Pedoman Penamaan

Penamaan dalam pembangunan *Class Library* merupakan hal yang penting karena hal ini diperlukan agar mudah dipahami oleh pengguna. Terdapat beberapa aturan penamaan yaitu *General Naming Conventions, Names of Namespaces, Name of Clases, Structs, and Interfaces, Name of Type Members, Names of Parameters, Names of Resources*. Pola penamaan yang konsisten akan mempengaruhi prediktibilitas dan dicoverability dalam pengelolaan class library [22].

1. Konvensi untuk kapitalisasi

Terdapat dua acara dalam penamaan yang biasa digunakan yaitu, Pascal Casing dan Camel Casing. Pada Pascal Casing huruf pertama pada identifier dan huruf pertama pada kata berikutnya harus kapital, contohnya “TimeTable”. Sedangkan untuk Camel Casing huruf pertama dari idetifier adalah huruf kecil dan huruf pertama pada kata berikutnya adalah huruf besar, contohnya “timeTable” [22].

2. Konvensi penamaan nama secara umum

Terdapat hal-hal yang berhubungan dengan penamaan yang harus diperhatikan seperti pemilihan nama *identifier* harus mudah dibaca, penamaan yang ringkas dan mewakili, jangan menggunakan garis bawah, tanda hubung, atau karakter nonalphanumeric lainnya, dan jangan menggunakan notasi Hungaria [22].

Untuk menghindari kebingungan dan menjamin operasi lintas bahasa, ikuti aturan penggunaan singkatan dibawah ini:

a. Jangan gunakan singkatan sebagai bagian dari nama identifier. Misalnya, menggunakan GetWindow bukan GetWin.

b. Jangan menggunakan singkatan yang tidak umum dalam komputasi.

c. Gunakan singkatan untuk menggantikan nama frase yang panjang. Misalnya menggunakan UI untuk User Interface.

d. Ketika menggunakan singkatan, gunakan *pascal case* dan *camel case* untuk singkatan yang karakternya lebih dari dua. Misalnya System.IO.

e. Jangan menggunakan singkatan dalam identifiers atau nama parameter. Jika Anda harus menggunakan singkatan, gunakan *camel case* untuk singkatan yang terdiri dari lebih dari dua karakter, bahkan jika ini bertentangan dengan singkatan standar dari kata.

1. Pedoman Penamaan Kelas

Dalam menamai sebuah kelas, pedoman yang perlu untuk deperhatikan adalah [22]:

a. Gunakan kata benda atau kata benda frase untuk nama kelas.

b. Gunakan *pascal case*.

c. Gunakan singkatan.

d. Jangan menggunakan tipe awalan, seperti C untuk kelas dalam penamaan kelas, misalnya gunakan nama FileStream daripada CFileStream.

e. Jangan gunakan karakter garis bawah/*underscore* (_).

f. Terkadang diperlukan penamaan kelas yang diawali dengan karakter I meskipun kelas tersebut bukan Interface. Hal ini masih dianggap sesuai selama I adalah huruf pertama dari seluruh kata yang merupakan bagian nama kelas. Misalnya nama kelas IdentityStore.

g. Apabila diperlukan gunakan kata majemuk untuk nama kelas turunan. Bagian kedua dari nama kelas turunan harus menjadi nama kelas dasar. Misalnya, ApplicationException adalah nama kelas turunan yang tepat dari kelas yang bernama Exception, karena ApplicationException adalah jenis Exception.

2. Pedoman Penamaan Parameter

Dalam penamaan attribute selalu harus ditambahkan akhiran attribute untuk kelas attribute khusus [22].

1. Pedoman Penamaan *Method*

a. Gunakan kata kerja atau frase kata kerja untuk nama *method*.

b. Gunakan *pascal case*.

Berikut ini adalah contoh dari penamaan *method* yang benar.

```
GetAll(params Expression<Func<T, object>>[] includes)

Delete(T model);

RemoveAll();
```

2. Pedoman Penamaan *Property*

Aturan berikut menguraikan pedoman penamaan property:

a. Gunakan kata benda atau frase kata benda untuk penamaan property.

b. Gunakan *pascal case*.

c. Jangan menggunakan notasi hungaria.

d. Pertimbangkan untuk membuat sebuah property dengan nama yang sama sebagai jenis yang mendasarinya. Berikut ini adalah contoh dari penamaan property yang benar.

```
public Guid Id { get; set; }
public string Name { get; set; }
public string Description { get; set; }
```

*3. Name of Assemblies and DLL's*

Sebuah assembly mengamdung semua atau bagian-bagian dari library yang dapat digunakan kembali dan terdapat dalam sebuah single dynamic-link library (DLL). Sebuah assembly dapat dibagi menjadi multiple DLL tetapi hal ini jarang sekali digunakan. Berikut adalah panduan penamaan untuk DLL [22].

a. Jangan pilih nama DLL dengan nama yang menunjukan potongan besar fungsionalitas seperti System.Data. Assembly dan nama DLL tidak harus sesuai dengan nama sama namespace akan tetapi akan

lebih baik mengikuti nama namespace saat penamaan assembly.

a. Pertimbangkan penamaan DLL sesuai dengan pola berikut.

<Company>.<Component>.dll

<Component> mengandung satu atau lebih klausa dot (titik) sebagai pemisah. Contoh penggunaannya adalah sebagai berikut.

Contoso.WebControls.dll

### II.7.1 Pedoman Perancangan Tipe

1. Tipe data dan Namespace

Berikut ini merupakan panduan yang dapat membatu dalam merancang jenis tipe data dan namespace:

a. Hindari namespace yang terlalu banyak.

b. Jangan gunakan namespace untuk mengatur types menjadi hirarki daerah fitur yang terkait.

c. Hindari hirarki namespace yang sangat dalam.

d. Hindari desain type untuk skenario yang rumit dalam namespace yang sama sebagai tipe yang digunakan untuk tugas pemrograman yang umum.

2. Rancangan Kelas Absrak

Karena abstract kelas sebaiknya tidak pernah dipakai, sangat penting untuk menentukan konstruktor yang benar untuk kelas abstrak tersebut. Berikut panduan agar kelas abstrak bekerja dengan benar pada saat diimplementasikan:

a. Jangan menentukan hak akses public atau protected internal pada konstraktor dalam tipe abstrak.

b. Jangan menentukan hak akses protected atau konstruktor internal dalam kelas abstrak.

a. Jangan menggunakan setidaknya satu tipe konkret yang mewarisi dari setiap kelas abstrak yang digunakan.

1. Rancangan Kelas *Static*

Kelas statik merupakan kelas yang tidak mengandung instance members lain dari yang diwarisi dari object, dan tidak memiliki konstuktor callable. Berikut panduan agar kelas static dirancang dengan benar:

a. Jangan gunakan kelas static yang sederhana. Kelas static digunakan hanya sebagai pendukung kelas-kelas untuk O*bject Oriented Framework.*

b. Jangan mendeklarasikan atau melakukan override instance member pada kelas static. Jika desain kelas menunjukan harus ada instance member, kelasa seharusnya tidak ditandai dengan static.

c. Jangan mendeklarasikan kelas statis sebagai sealed dan abstract, dan tambahkan sebuah private instance konstuktor jika bahasa pemrograman yang digunakan tidak memiliki built-in untuk mendukung kelas statis.

d. Jangan memperlakukan static class sebagai sebuah miscellaneous bucket. Environment class adalah contoh yang baik dari penggunaan yang tepat dari kelas statis. Kelas ini menyediakan akses ke informasi tentang lingkungan pengguna saat ini.

2. Rancangan *Interface*

Berikut panduan untuk memastikan bahwa interface diancang dengan benar:

a. Jangan mendefinisikan sebuah interface jika kita perlu beberapa fungsi umum yang akan didukung oleh seperangkat types yang mencakup beberapa nilai types.

b. Pertimbangkan pemakaian interface jika kita butuh untuk mendukung fungsionalitas pada types yang sudah mewarisi dari types lainnya.

c. Sediakan paling sedikit satu types yang merupakan implementasi dari sebuah interface.

d. Sediakan paling sedikit satu anggota yang menggunakan setiap interface, contohnya sebuah *method* yang mengambil interface sebagai parameter atau property.

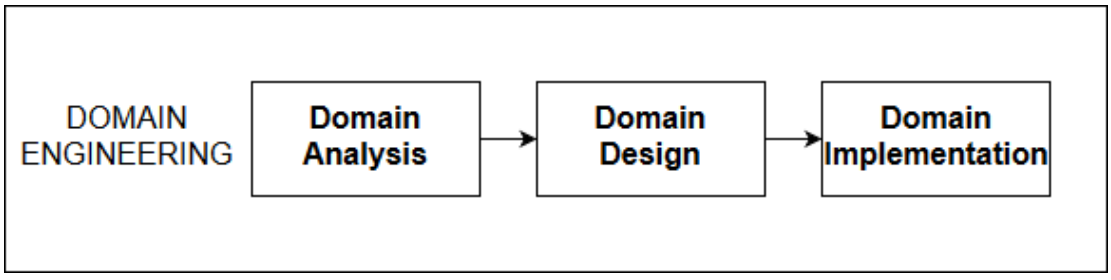
## II.1 Analisis Domain

Analisis domain adalah proses pengumpulan informasi dalam proses pembangunan perangkat lunak pada sebuah *scope* tertentu. Analisis domain mendefinisikan fitur-fitur umum yang terdapat pada *scope* tersebut [23]. Analisis domain secara umum mendukung ekstraksi, organisasi, analisis, abstraksi serta pemahaman dan pemodelan informasi yang dapat digunakan kembali dari proses perangkat lunak.

### II.8.1 Domain Engineering

*Domain Engineering* atau biasa disebut *product-line engineering* adalah keseluruhan proses dalam penggunaan ulang pengetahuan dalam domain tertentu [24]. Pada dasarnya, sebuah organisasi/perusahaan hanya bekerja pada beberapa domain tertentu saja, mereka hanya membuat sistem dengan domain yang sama namun dengan variasi yang berbeda untuk memenuhi kebutuhan user [24]. Daripada membuat setiap sistem dengan domain yang sama namun dengan variant yang berbeda dari awal, penggunaan ulang bagian dari sistem yang sudah dibuat sebelumnya dapat membantu untuk membuat sistem yang baru.

*Domain Engineering (*Rekayasa Domain*)* memiliki tiga tahapan yaitu analisis, desain dan implementasi. Ketiga tahapan tersebut dapat dilihat dari gambar dibawah ini.




**Gambar II.1 – Tahapan *Domain Engineering***

### *II.8.1 Feature-Oriented Domain Analysis*

*Feature-Oriented Domain Analysis* adalah metode sistematik yang bertujuan untuk menganalisis sebuah domain. Pada tahun 1990, Software Engineering Institute memperkenalkan FODA [25]. Tahapan pada FODA antaralain adalah:

1. *Context Analysis* merupakan tahapan untuk mendefinisikan batasan-batasan daripada domain yang dianalisis (*boundaries*) [25]. Hasil dari tahapan ini adalah mengetahui dan memahami tentang cakupan dari domain, relasi terhadap domain lain, input dan output ke dan dari domain lain serta kebutuhan penyimpanan data [26].
2. *Domain Modeling,* sebuah tahapan yang menjelaskan ruang lingkup masalah yang berada pada domain aplikasi [25]. Hasil dari tahapan ini adalah fitur dari *software* di dalam domain [26].

### *II.8.2 Feature-Oriented Reuse Method*

FORM (*Feature-Oriented Reuse Method*) adalah metode sistematik yang digunakan untuk mencari serta menangkap persamaan dan perbadaan dari aplikasi-aplikasi di dalam sebuah domain dengan batasannya adalah fitur-fitur dari aplikasi tersebut. Hasil dari analisis tersebut digunakan untuk membangun arsitektur dan komponen domain [14]. Hasil dari analisis tersebut adalah model yang dinamakan *Feature Model. Feature Model* atau model fitur digunakan untuk mendukung proses rekayasa artefak domain yang dapat digunakan kembali, serta untuk membangun aplikasi menggunakan artefak domain tersebut. Setelah domain dideskripsikan dan dijelaskan dalam bentuk *unit* komputasi yang umum dan berbeda, maka hal tersebut digunakan untuk membangun sebuah kemungkinan yang berbeda dalam konfigurasi arsitektur yang dapat digunakan kembali.

## II.1 UML

UML (Unified Modeling Language) adalaha sebuah bahasa yang digunakan secara standard untuk menuliskan cetak biru (*blueprint*) sebuah perangkat lunak [27].

UML dikhususkan untuk pembuatan dan pengembangan perangkat lunak berorientasi objek. UML pertamakali dikembangkan oleh Grandy Booch, Jim Rumbaugh, dan Ivars Jacobson pada pertengahan tahun 1990 [28].

Pada pengimplementasiannya UML menyediakan 4 diagram yang digunakan untuk memodelkan perangkat lunak berorientasi objek yaitu:

1. *Use Case* Diagram

Sebuah diagram pemodelan yang memberikan gambaran apa yang harus dilakukan oleh sistem berdasarkan fungsionalitas yang harus ada pada sistem yang dibangun. Komponen yang terdapat pada *use case* diagram antara lain adalah:

*a.* *Actor*

Sebuah aktor bukan bagian dari dalam sistem, melainkan adalah yang berinteraksi dengan sistem yaitu dengan fungsionalitas pada sistem. Sebuah aktor tidak selalu merupakan user atau pengguna.

*b.* *Use Case*

*Use Case* merupakan sebuah pekerjaan pada sistem yang akan berhubungan dengan aktor, digambarkan dengan elips dan diberikan keterangan pada setiap pekerjaan tersebut.

*c.* *Use Case Relationship*

Menjelaskan bagaimana perilaku sistem dalam memenuhi kebuthan. Perilaku tersebut dapat menunjukan penggunaan ulang, opsional, bahkan spesialisasi pada *use case*.

*2.* *Activity Diagram*

*Activity diagram* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, decision yang mungkin terjadi dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. *Activity diagram* merupakan state diagram khusus, dimana sebagian besar state adalah action dan sebagian besar transisi di-*trigger* oleh selesainya state sebelumnya (*internal processing*

). Oleh karena itu activity diagram tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

*1.* *Sequence Diagram*

*Sequence diagram* untuk *use case* adalah diagram interaksi yang mengekspresikan interaksi antara aktor dan sistem dengan penekanan waktu. Setiap skenario pada *use case* harus dibuat sequence diagramnya. Yang dideskripsikan adalah keinginan aktor, bukan mekanisme implementasi. Sequence diagram menunjukkan hubungan dinamis antara objek. Setiap sequence diagram halnya mendeskripsikan satu skenario.

*2.* *Class Diagram*

*Class diagram* memberikan gambaran struktur dan deskripsi kelas, *interaface* serta hubungannya dengan kelas lain, sebuah kelas memiliki tiga area utama yaitu nama kelas, atribut kelas yang di dalamnya terdapat hak akses untuk atribut tersebut, dan motode kelas dimana metode tersebut yang didalamnya memiliki hak akses untuk setiap metode pada kelas. Sebuah kelas juga dapat mempunyai hubungan diantaranya seperti asosiasi, agregasi, pewarisan, dan komposisi.