BAB 2
TINJAUAN PUSTAKA

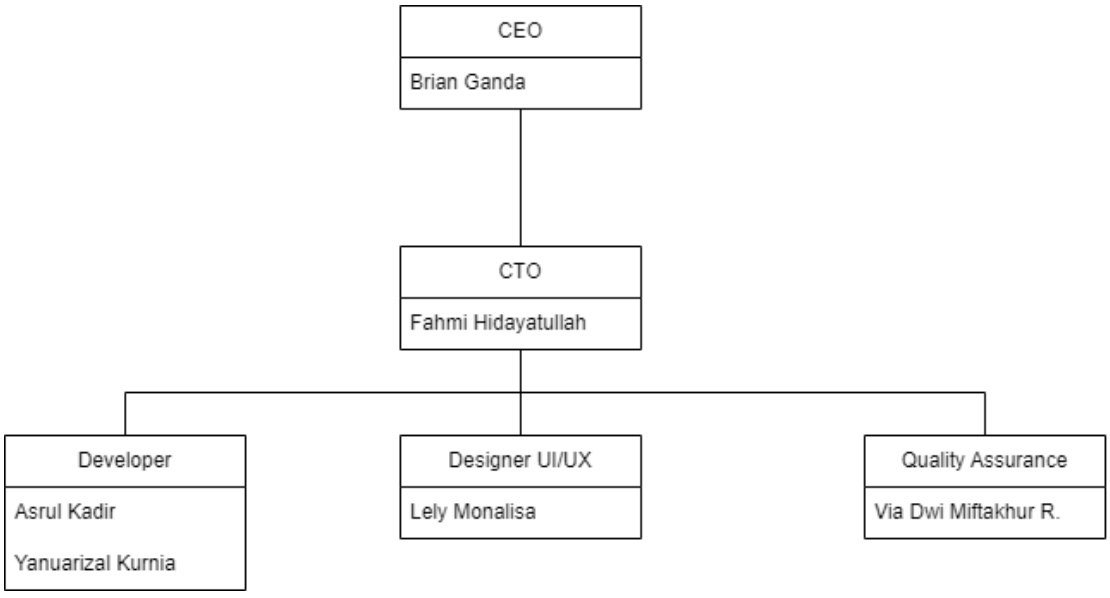
2.1 Profil Perusahaan

2.1.1 Identitas Perusahaan

Evomo didirikan pada tahun 2018 dengan nama Jaeger pada awalnya. Evomo adalah salah satu *startup* di program Digital Amoeba yang merupakan program inkubasi startup yang dibentuk oleh IDE. Telkom Ide. Evomo menciptakan solusi bagi industri manufaktur dan agribisnis untuk meningkatkan produktivitas menggunakan teknologi *Internet of Things*, *Big Data*, dan *Robotic Process Automation*.

2.1.2 Struktur Perusahaan

Struktur organisasi dari Evomo yang dapat dilihat pada gambar 1.2 berikut:



Gambar 2-1 Struktur Organisasi Evomo

2.1.3 Deskripsi Tugas/Jabatan

Tabel 2.1 Deskripsi Tugas/Jabatan

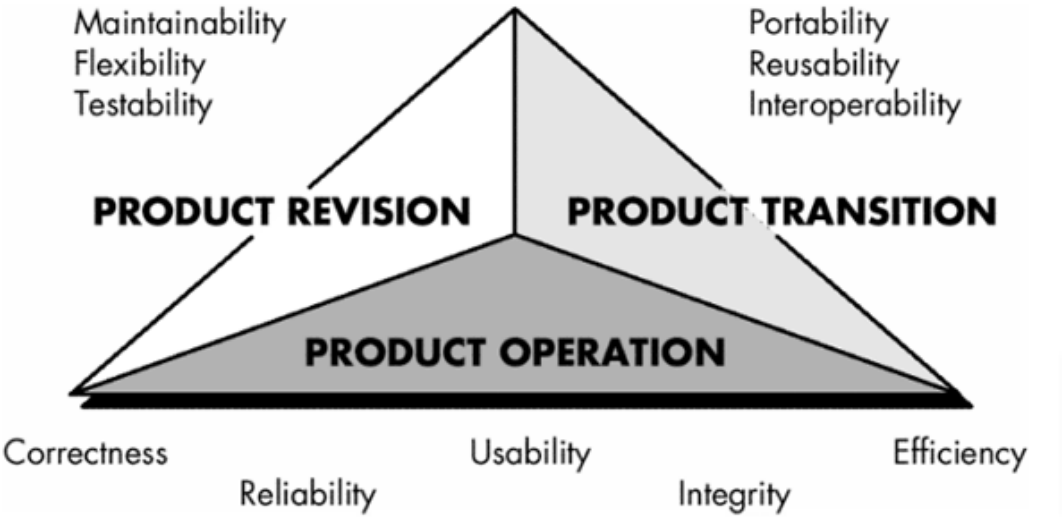
Jabatan	Deskripsi Tugas
Chief Executive Officer	Bertanggung jawab atas kebijakan dan strategi perusahaan, memimpin dan memotivasi tim, menjaga hubungan dengan stakeholder, dan memastikan pertumbuhan perusahaan.
Chief Technology Officer	Bertanggung jawab atas teknologi dan inovasi perusahaan, memastikan integritas dan keamanan sistem teknologi, memimpin tim pengembangan produk, dan menjaga kompetitif perusahaan dalam hal teknologi.
Developer	Bertanggung jawab atas pengembangan produk, membuat dan memelihara kode, berkolaborasi dengan tim, dan memastikan produk memenuhi standar kualitas.
UI/UX Designer	Bertanggung jawab atas desain antarmuka pengguna dan pengalaman pengguna, memastikan produk mudah digunakan dan memuaskan, berkolaborasi dengan tim pengembangan dan bisnis, dan menjaga konsistensi dalam desain.

Quality Assurance	Bertanggung jawab atas memastikan produk memenuhi standar kualitas dan melakukan tes untuk memastikan produk bekerja sebagaimana mestinya, membuat laporan *bug* dan membantu tim pengembangan dalam mengatasinya, dan memastikan produk memenuhi regulasi dan standar industri.

2.2 Landasan Teori

2.2.1 Software Quality Assurance

Software quality assurance menurut IEEE adalah tindakan yang dilakukan sesuai dengan pola terencana dan sistematis untuk memastikan barang atau produk tersebut telah sesuai dengan persyaratan teknis yang telah ditetapkan [3]. *Software quality assurance* memiliki beberapa model salah satunya adalah model McCall. Metode McCall merupakan metode yang mengukur atau mengevaluasi kualitas perangkat lunak yang terdiri dari 3 faktor utama dengan 11 sub faktor seperti pada gambar di bawah ini [3]



Gambar 2-2 Software Quality Assurance

Ide utama McCall adalah untuk menilai hubungan faktor-faktor kualitas dan kriteria kualitas produk atau meningkatkan kualitas perangkat lunak [6].

2.2.2 Clean Code

Clean code adalah kumpulan kode yang mudah dibaca, mudah dimengerti dan mudah di *maintenance* oleh pengembangnya sendiri maupun orang lain [4]. *Code* yang dapat dipahami, mudah dibaca, dan mudah dikembangkan merupakan tujuan utama dari *Clean Code*. *Clean Code* dipisah menjadi beberapa bagian kategori yang dijadikan pedoman dalam penulisan *Good Code* yaitu :

2.2.2.1 Meaningful Names

Dalam penulisan kode perlu dilakukannya penulisan nama yang dapat menjelaskan tujuan dan tugas dari variable, fungsi, atau objek [4]. Dalam *meaningful names* terdapat beberapa prinsip didalamnya yaitu.

1. *Use Intention-Revealing Names*

Nama variabel, fungsi, kelas, dan lainnya harus menunjukkan tujuan atau niat kode. Nama yang baik dapat menjelaskan apa yang dilakukan kode tanpa memberikan komentar tambahan [4].

```
1 var d int
2 d = 10
```

Gambar 2-3 Contoh Use Intention-Revealing Names yang Buruk

```
1 var daysInWeek int
2 daysInWeek = 10
```

Gambar 2-4 Contoh Use Intention-Revealing Names yang Baik

2. Avoiding Disinformation

   Dalam penulisa kode perlu menghindari penggunaan kata, singkatan, atau notasi yang dapat membuat bingung pembaca [4].

```
1 var rn int
2 rn = 5
```

Gambar 2-5 Contoh Avoiding Disinformation Names yang Buruk

```
1 var roomNumber int
2 roomNumber = 5
```

Gambar 2-6 Contoh Avoiding Disinformation Names yang Baik

3. Make Meaningful Distinctions

   Pada penulisan nama kode disarankan tidak menggunakan nama yang berbeda tanpa perbedaan tujuan dalam penggunaan kode tersebut [4].

```
1 func getUserInfo(){}
2 func getUserData(){}
```

Gambar 2-7 Contoh Make Meaningful Distinctions yang Buruk

```
1 func getUserProfile(){}
2 func getUserSettings(){}
```

Gambar 2-8 Contoh Make Meaningful Distinctions yang Buruk

4. Use Pronounceable Names

   Nama dari sebuah kode harus mudah diucapkan sehingga diskusi pada saat membahas kode menjadi mudah [4].

```
1 var genymdhms string
```

Gambar 2-9 Contoh Use Pronounceable Names yang Buruk

```
1 var generationTimeStamp string
```

Gambar 2-10 Contoh Use Pronounceable Names yang Baik

5. Use Searchable Names

   Dalam penulisan nama kode perlu unik sehingga mempermudah dalam pencarian kode oleh pembaca.

```
1 var e float64
2 e = 2.71828
```

Gambar 2-11 Contoh Use Searchable Names yang Buruk

```
1 var eulerNumber float64
2 eulerNumber = 2.71828
```

Gambar 2-12 Contoh Use Searchable Names yang Baik

6. Avoiding Encodings

   Dalam penulisan nama kode perlu dihindari penulisan tipe data pada nama kode karena tidak dapat membantu dalam pembacaan kode akan tetapi dapat membuat pembaca bingung.

```
1 var strName string
2 strName = "Alice"
```

Gambar 2-13 Contoh Avoiding Encodings yang Buruk

```
1 var name string
2 name = "Alice"
```

Gambar 2-14 Contoh Avoiding Encodings yang Baik

7. Avoiding Mental Mapping

Dalam penulisan nama kode perlu dihindari nama yang dapat membuat pembaca melakukan pemetaan mental dengan cara harus mengingat apa tugas dan tujuan kode tersebut.

```
1 var days []string
2 days = []string{"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
  "Sunday"}
3
4 fmt.Println(days[0])// Requires mental mapping, 0 -> Monday
```

Gambar 2-15 Contoh Avoiding Mental Mapping yang Buruk

```
1 var days []string
2 days = []string{1:"Monday", 2:"Tuesday", 3:"Wednesday", 4:"Thursday", 5:"Friday", 6:"Saturday",
3 7:"Sunday"}
4 fmt.Println(days[0])// Requires mental mapping, 0 -> Monday
```

Gambar 2-16 Contoh Avoiding Mental Mapping yang Baik

2.2.2.2 Clean Function

Clean function merupakan salah satu faktor dalam penulisan *clean code*. *Clean function* berfungsi sebagai pedoman dalam penulisan fungsi dan terdapat beberapa prinsip dalam *clean function* ini.

1. Small

   Sebuah fungsi seharunya kecil dan hanya memiliki satu tugas. Jumlah baris yang disarankan dalam penulisan fungsi adalah tidak boleh lebih dari 20 baris kode.

```
1 func printHello(){
2   fmt.Println("Hello, world!")
3 }
```

Gambar 2-17 Contoh Kode Small

2. Do One Thing

   Sebuah fungsi disarankan hanya boleh memiliki satu tugas dan tidak disarankan lebih dari satu.

```go
func calculateAndPrintSum(a, b int){
  sum := a + b
  fmt.Println(sum)
}
```

Gambar 2-18 Contoh Do One Thing yang Buruk

```go
func calculateSum(a, b int) int{
  return a + b
}
func printSum(sum int){
  fmt.Println(sum)
}
```

Gambar 2-19 Contoh Do One Thing yang Baik

3. Block and Indenting

   Dalam penulisan fungsi disarankan dala penggunaan blok if, for, switch tetap kecil untuk menghindari terjadinya *nested block* sehingga memiliki indentasi yang tepat dapat membuat mudah dibaca dan dipahami [4].

```go
if condition{
  //do something
} else {
  //do something
}else
```

Gambar 2-20 Contoh Block and Indenting

4. The Stepdown Rule

   Dalam penulisan kode perlu dibuat seperti narasi dari atas kebawah sehingga mudah dalam memahami dan membaca fungsi tersebut.

```
func main(){
  printWelcomeMessage()
  getInput()
  processData()
  printGoodbyeMessage()
}
```

Gambar 2-21 Contoh The Stepdown Rule

5. Function Argument

   Dalam penulisan fungsi perlu diperhatikan dalam penulisan argument. Fungsi yang baik adalah memiliki argument seminimal mungkin.

```
1 type Person struct{
2   Name      string
3   Age       int
4   Address   string
5   Job       string
6 }
7
8 func createPerson(p Person)
9 {
10 }
```

Gambar 2-22 Contoh Function Argument

6. Have No Side Effect

   Fungsi seharusnya tidak memiliki *side effect* yang dapat merubah keadaan sistem atau merubah nilai dari variable global.

```
1 func add(a, b int) int{
2   return a + b
3 }
```

Gambar 2-23 Contoh Have No Side Effect

2.2.2.3 Clean Comment

Clean comment merupakan salah satu faktor dalam penulisan *clean code*. *Clean comment* merupakan panduan dalam penulisan komentar pada kode dan memiliki beberapa prinsip.

1. Explain Yourself In Code

   Dalam prinsip ini cara terbaik untuk dipahami pembaca adalah dengan membuat nama kode yang dapat menjelaskan tugas dan tujuan kode tersebut, alih-alih menambahkan komentar untuk menjelaskan kode.

```
1 //check if the user is an admin
2 if u.Role == 1{
3   //do something
4 }
```

Gambar 2-24 Contoh Buruk Explain Yourself In Code

```
1 isAdmin := u.Role == 1
2 if isAdmin {
3   //do something
4 }
```

Gambar 2-25 Contoh Baik Explain Yourself In Code

2. Legal Comments

   Legal comments adalah komentar yang berisikan komentar hukum tentang hak cipta dan lisensi pada kode.

```
1 // Copyright 2023 My Company. All rights reserved.
2 // User of this source code is governed by a BSD-
3 // style license that can be found in the LICENSE file.
```

Gambar 2-26 Contoh Legal Comments

3. Informative Comments

Informative Comments merupakan komentar informatif yang memberikan informasi tentang kode. *Informative Comments* digunakan ketika kode tidak dapat dijelaskan oleh kode itu sendiri.

```
// Using bitwise OR to combine flags
flags := flag1 | flag2
```

Gambar 2-27 Contoh Informative Comments

4. Explanation Of Intent

   Explanation of Intent merupakan komentar yang menjelaskan niat dan tujuan dari dibuatnya kode tersebut agar memudahkan pembaca paham mengapa kode tersebut dibuat.

```
//we're using a map here for faster lookup times
var usersPermissions map[string]bool
```

Gambar 2-28 Contoh Explanation of Intent

5. Warning Of Consequences

   Warning of consequences digunakan untuk memperingatkan konsekuensi dari sebuah kode yang memiliki pontensi masalah.

```
//WARNING: This method is not thread safe
func unsafeMethod(){
  //do something
}
```

Gambar 2-29 Contoh Warning of Consequences

6. TODO Comments

   TODO Comments digunakan untuk menandai sebuah perkejaan yang akan dilakukan pada kode tersebut seperti menambahkan fitur pada kode tersebut.

```
1 //TODO: Implement caching
2 func getFromDatabase(){
3
4 }
```

Gambar 2-30 Contoh TODO Comments

2.2.2.4 Clean Formatting

Clean formatting merupakan salah satu faktor dari penulisan *clean code*. *Clean formatting* merupakan prinsip yang mengatur bagaimana penulisan kode sehingga dapat meningkatkan keterbacaan dan pemahaman kode tersebut.

1. Vertical Formatting

   Vertical formatting merupakan pedoman dalam penulisan jumlah baris kode. Baris kode dalam satu file disarankan sekitar 500 baris kode.

2. Vertical Opennes Between Concepts

   Prinsip adalah memberikan baris kosong pada setiap blok logika agar memudahkan dalam pemahaman kode.

```
1 var name = "John Doe"
2
3 var age = 30
4
5 var address = "123 Main St"
```

Gambar 2-31 Contoh Vertical Opennes Between Concepts

3. Vertical Distance

   Vertical Distance merupakan prinsip yang mengharuskan deklarasi varible dan penggunaan berdekatan.

```
1 name := "John Doe"
2 fmt.Println(name)
```

Gambar 2-32 Contoh Vertical Distance

4. Horizontal Formatting and Openness

   Prinsip ini membatas panjang baris kode horizontal agar tidak melebihi 80-120 karakter dan juga menggunakan spasi dan indentasi secara konsisten.

```
1 addResult:=addValues(firstValue,secondValue)
```

Gambar 2-33 Contoh Buruk Horizontal Formatting and Openness

```
1 addResult := addValues(firstValue, secondValue)
```

Gambar 2-34 Contoh Baik Horizontal Formatting and Openness

5. Indentation

   Prinsip ini digunakan agar penulisan kode menggunakan indetasi secara konsisten dan sesuai dengan standar sehingga struktur kode terlihat lebih jelas dan mudah dibaca.

2.2.2.5 Clean Error Handling

Clean error handling merupakan pedoman dalam penulisan *error handling* yang baik dan benar sehingga kode dapat dikatakan bersih. Ada beberapa aturan dalam penulisan *error handling* salah satunya adalah tidak boleh mengembalikan nilai null (nil).

```go
func FindUser(username string) *User {
  // implementation
  return nil // if user not found
}
```

Gambar 2-35 Contoh Buruk Error Handling

```go
func FindUser(username string) (*User, error) {
  // implementation
  return nil, errors.New("user not found") // if user not found
}
```

Gambar 2-36 Contoh Baik Error Handling

2.2.2.6 Clean Objects and Data Structures

Prinsip ini merupakan pedoman ddalam pegunaan *object* dan *data structures*. Dalam prinsip ini sebaiknya data dienkapsulasi dalam *object*.

```go
type User struct {
  name string
  email string
}

func (u *User) Email() string {
  return u.email
}
```

Gambar 2-37 Contoh Clean Objects and Data Structures

2.2.3 Software Maintainability

Software Maintainability merupakan salah satu atribut kualitas mendasar dari perangkat lunak [5]. Tujuan dari *Software Maintainability* adalah untuk menjaga *software* tetap beroperasi, untuk mencegah atau memperbaiki kesalahan dalam *software* dan meningkatkan fungsionalitas *software*. *Software Maintainability* sendiri memiliki beberapa sub seperti *simplicity, modularity,*

self-descriptiveness, coding and documentation guidelines, compliance, dan *document accessbility*

2.2.4 Cyclomatic Complexity

Cyclomatic Complexity adalah salah satu metrik yang cukup terkenal yang mana metrik ini menghitung *control flow* dari suatu modul. Apabila kompleksitas semakin tinggi maka akan modul tersebut akan semakin sulit untuk diuji dan dirawat [2]. *Cyclomatic Complexity* memiliki rumus sebagai berikut :

$$M = E - N + 2P \quad (1)$$

Dimana :

M = *Cyclomatic Complexity*

E = Jumlah *edge* pada graf kendali

N = Jumlah *node* (jalur) pada graf kendali

P = Jumlah komponen yang terhubung

2.2.5 Halstead Metrics

Halstead's metric adalah pengukuran yang dikembangkan untuk mengukur kompleksitas modul suatu program langsung dari kode sumber [6]. Ada beberapa *metric* dalam *Halstead metrics* yaitu:

- Program *length*

  Total munculnya operator serta *operand* dengan estimasi program *length* yaitu

  $$N = N1 + N2 \quad (2)$$

  Dimana:

  N1 = Jumlah total operator

  N2 = Jumlah total operand

- *Halstead Vocabulary*

  Total jumlah dari operator dan *operand* yang unik dengan

  $$n = n1 + n2 \quad (3)$$

  Dimana:

  n1 = Jumlah operator yang berbeda

  n2 = Jumlah operand yang berbeda

- Program Volume

  Merupakan referensi dari ukuran program (dalam ukuran bits) dengan

  $$V = Size * (\log2\ vocabulary) = N*\log2\ (n) \quad (4)$$

- Potensial Minimum Volume

  Merupakan kemungkinan nilai terkecil dari volume yang bisa dihasilkan oleh Code, dengan

  $$V^* = (2 + n2^*) * \log2(2 + n2^*) \quad (5)$$

- Program Level

  Tingkatan yang ditentukan berdasarkan bahasa pemrograman, semakin tinggi level bahasanya maka akan semakin sedikit usaha untuk mengembangkan program dengan bahasa tersebut,

  $$L = V^* / V \quad (6)$$

2.2.6 Maintainability Index

Maintainability index adalah salah satu metrik yang paling banyak digunakan dalam industri dan telah sepenuhnya sukses diterapkan di sistem perangkat lunak seperti Visual Studio [2]. *Maintainability Index* dihitung berdasarkan nilai dari *Halstead's Volume*, McCabe's *Cyclomatic Complexity*, dan jumlah baris kode sumber (*source code*) [2]. Terdapat 3 rumus yang sering digunakan *Maintainability Index*, yaitu :

the original formula:

$$MI = 171 - 5.2 \ln V - 0.23 G - 16.2 \ln L \quad (7)$$

Formula yang digunakan oleh SEI:

$$MI = 171 - 5.2 \log2\ V - 0.23\ G - 16.2 \log2\ L + 50 \sin(\sqrt{2.4C}) \quad (8)$$

Formula yang digunakan oleh Visual Studio:

$$MI = \max [0,100 * (171 - 5.2 \ln V - 0.23\ G - 16.2 \ln L) / 171] \quad (9)$$

Dimana :

- V adalah *Halstead Volume*
- G adalah total *Cyclomatic Complexity*
- L adalah nomor dari *Source Lines of Code* (SLOC)
- C adalah nilai persen dari *comment lines*

Kategori nilai dari *Maintainability Index* terdiri dari:

- 0-9 = Red
- 10-19 = Yellow
- 20-100 = Green

2.2.7 Go Language

Go language atau yang biasa disebut Golang merupakan Bahasa pemrograman yang digunakan untuk membangun website terutama Back-end dan aplikasi lainnya. Go adalah bahasa pemrograman serba guna dengan fitur-fitur canggih dan sintaks yang bersih. Karena itu Ketersediaan yang luas di berbagai platform, bahasa ini kuat dalam *library* yang terdokumentasi dengan baik [7]. Golang diciptakan di Google oleh Robert Griesemer, Rob Pike, dan Ken Thompson dan rilis pada bulan Maret 2012. Golang tidak menyediakan kelas tetapi menyediakan struktur [8]. Metode dapat ditambahkan pada struktur. Hal ini memberikan perilaku penggabungan data dan metode yang beroperasi pada data secara bersama-sama dan struktur dalam Golang terdapat *Struct, Method,* dan, *Interface*.

2.2.8 Maintidx

Maintidx merupakan *open source tools* yang berfungsi untuk menghitung *maintainability index, cyclomatic complexity*, dan *halstead volume* pada Bahasa pemrograman Go [9].

2.2.9 Refactoring

Refactoring adalah teknik yang digunakan untuk meningkatkan kualitas dan desain kode sumber dengan memperbaiki *code smells*, yang merupakan indikator kode sumber buruk yang biasanya menyebabkan masalah teknis dan kinerja dalam suatu sistem [10]. *Refactoring* memiliki dampak signifikan pada kualitas sistem, efisiensi dan efektivitas. Namun, saat melakuan implementasi *refactoring*, ada juga tantangan yang harus diatasi, seperti kompleksitas algoritma, keterbatasan dalam mendeteksi *code smells* yang kompleks, dan masalah dalam menjaga konsistensi dan konsistensi kode sumber setelah *refactoring* [11].

2.2.10 Visual Studio Code

Visual Studio Code adalah editor kode sumber yang ringan namun kuat yang dapat dijalankan di desktop Anda dan tersedia untuk Windows, macOS, dan Linux. Muncul dengan dukungan bawaan untuk JavaScript, TypeScript dan Node.js dan memiliki ekosistem ekstensi yang kaya untuk bahasa dan runtime lain (seperti C++, C#, Java, Python, PHP, Go, .NET) [12].

2.2.11 Prinsip SOLID

Prinsip desain SOLID adalah prinsip desain berorientasi objek yang memungkinkan pengelolaan sebagian besar masalah terkait kualitas desain perangkat lunak [13]. *Single Responsibility Principle* (SRP) adalah Prinsipterhadap kelas yang seharusnya memiliki satu tanggung jawab saja, yang tidak berarti bahwa suatu kelas hanya dapat melakukan satu hal [14].

Open/Closed Principle (OCP) menyatakan bahwa kelas harus terbuka untuk ditambahkan namun tertutup untuk modifikasi [14]. *Liskov Substitution Principle* (LSP) yaitu metode untuk mewariskan yang mana kelas turunan harus sepenuhnya mendukung substitusi kelas induk dan setiap kelas turunan harus dapat diganti dengan kelas induknya [14]. *Interface Segretion Principle* (ISP) yaitu ketika fungsionalitas akan digunakan oleh beberapa kelas, interface untuk kelas harus dibuat [14]. *Dependency Inversion Principle* (DIP) adalah Kode yang menerapkan kebijakan tingkat tinggi tidak boleh bergantung pada kode yang mengimplementasikan detail tingkat rendah [14].