BAB 2
LANDASAN TEORI

2.1 Tinjauan Perusahaan

2.1.1 Profil Perusahaan

Luarsekolah adalah platform marketplace edukasi vokasi dan pengembangan diri berbasis online yang dapat diakses dimana saja untuk mendampingi generasi muda Indonesia.

Latar belakang didirikannya luarsekolah adalah fenomena ketika mahasiswa merasa ilmu yang didapat sewaktu kuliah kurang relevan dengan yang dibutuhkan dalam dunia kera. Mahasiswa yang baru lulus dari dunia perkuliahan cenderung kebingungan untuk mencari pekerjaan dan merasa harus belajar keahlian dari nol. Dengan demikian, luarsekolah hadir untuk membantu *softskill* dalam menunjang karirnya.

Luarsekolah juga melihat peluang dari permasalahan pendidikan di Indonesia, diantaranya kesenjangan kurikulum pendidikan dengan karir di bidang industri, biaya pendidikan yang mahal, kurangnya akses pada pendidikan yang berkualitas dan kesalahan dalam pemilihan jurusan saat kuliah. Luarsekolah hadir untuk memecahkan masalah tersebut dengan menyediakan kursus dengan bahan pembelajaran yang terbaru dan terpilih, menyediakan kursus dengan harga terjangkau, membantu pengguna dalam menemukan kursus yang mereka butuhkan dan juga memastikan kursus mudah diakses.

Pada perusahaan luarsekolah, terdapat tiga divisi yaitu operation, marketing dan IT dengan total karyawan sebanyak delapan orang dan satu orang CEO. Divisi operation terdiri dari tiga orang yaitu operational manager, product creative director dan content writer. Divisi marketing terdiri dari tiga orang, yaitu marketing manager & director, social media specialist dan marketing communication. Divisi IT terdiri dari dua orang, yaitu back-end developer dan Chief Technology Officer (CTO).

2.1.2 Logo Perusahaan

Berikut merupakan logo dari perusahaan tempat penelitian :

Gambar 2.1 Logo Luarsekolah

2.1.3 Visi dan Misi

2.1.3.1 Visi

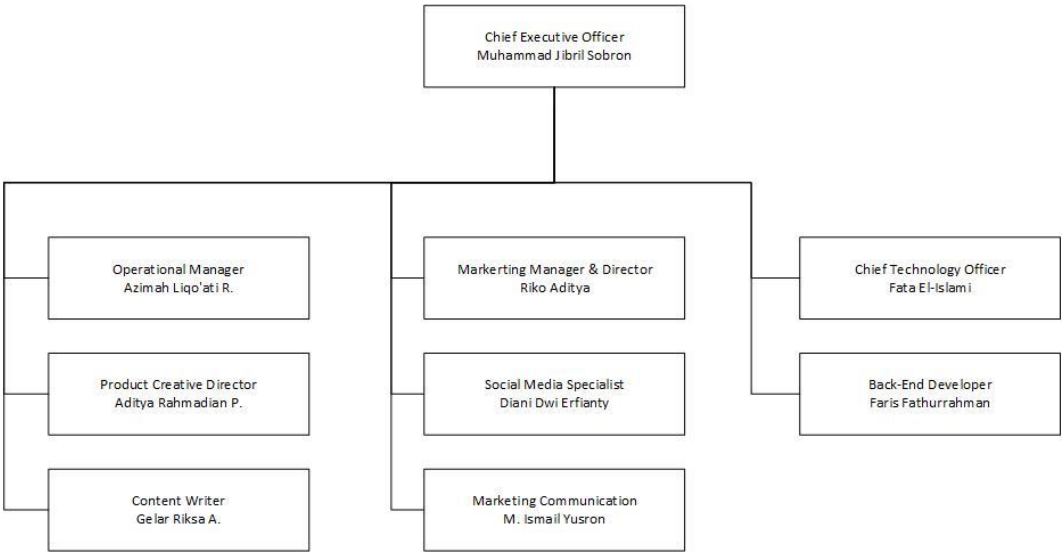
To create a place to learn, develop and collaborate together

2.1.3.2 Misi

1. Menghubungkan gap edukasi dengan industri zaman sekarang
2. Tidak perlu khawatirn masuk kuliah salah jurusan.
3. Menyediakan kelas dengan harga yang terjangkau
4. Memberikan akses edukasi yang mudah dan berkualitas

2.1.4 Struktur Organisasi

Berikut merupakan struktur organisasi yang terdapat dalam perusahaan :



Gambar 2.2 Struktur Organisasi

2.1.5 Deskripsi Tugas

Adapun deskripsi tugas struktur organisasi pada perusahaan luarsekolah adalah sebagai berikut

1. Chief Executive Officer (CEO)
   a. Merencanakan, mengelola, dan menganalisis segala aktivitas fungsional bisnis seperti operasional, sumber daya manusia, keuangan, dan pemasaran
   b. Merencanakan dan mengelola proses penganggaran, lalu mengamati dan menganalisis apabila ada kejanggalan dalam prakteknya
   c. Mengelola perusahaan sesuai dengan tujuan strategis perusahaan dengan keefektivan dan biaya seefisien mungkin
   d. Merencanakan dan mengelola kinerja pada sumber daya manusia agar sumber daya manusia yang berkompeten teridentifikasi dan dapat ditempatkan pada posisi yang sesuai sehingga dapat memaksimalkan kinerja perusahaan
   e. Merencanakan, mengelola, dan mengeksekusi perencanaan strategi bisnis atau korporat baik untuk jangka waktu menengah maupun panjang dengan mengacu pada visi dan misi perusahaan
   f. Mengidentifikasi dan meningkatkan performa operasional perusahaan dengan cara memotivasi berbagai divisi di perusahaan

g. Mengambil berbagai keputusan strategis yang berdampak baik bagi sustainabilitas perusahaan berdasarkan hasil analisis data dan fakta baik yang telah menjadi jejak rekam (record) perusahaan maupun analisis terhadap berbagai faktor lingkungan bisnis
h. Menjaga sustainabilitas keunggulan kompetitif perusahaan dan meningkatkan kompetensi utama perusahaan dan mengimplementasikannya
i. Menganalisis dan mengambil langkah paling prioritas bagi alokasi sumber daya dan penganggaran perusahaan
j. Membuat kebijakan, prosedur, dan standar pada organisasi perusahaan
k. Menganalisis segala masalah dalam perusahaan dan mengkoordinasikan manajemen puncak dalam menyelesaikan masalah tersebut secara efektif dan efisien
l. Membuat keputusan strategis dalam hal integrasi, divestasi, investasi, aliansi, dan joint venture

2. Operational Manager

a. Mengelola dan mengarahkan tim operasi untuk mencapai target bisnis.
b. Membantu untuk mengembangkan atau memperbarui prosedur operasi standar untuk semua kegiatan operasional bisnis.
c. Membangun hubungan yang kuat dengan menangani masalah dan keluhan pelanggan secara tepat waktu.
d. Memberikan penilaian karyawan, promosi, kompensasi dan pemutusan hubungan kerja berdasarkan tinjauan kinerja.
e. Memberikan dukungan operasional dan bimbingan kepada staf.
f. Membantu mengembangkan anggaran operasional dan modal.
g. Memantau dan mengendalikan pengeluaran sesuai anggaran yang dialokasikan.
h. Membantu dalam mewawancarai, merekrut dan melatih kandidat.
i. Mengelola penugasan kerja dan alokasi untuk staf.
j. Meninjau kinerja dan memberikan umpan balik kinerja kepada staf.

k. Menyimpan dokumentasi yang akurat dan jelas untuk prosedur dan kegiatan operasional.
l. Bekerja sesuai dengan kebijakan dan prosedur perusahaan.
m. Memastikan tim mengikuti prosedur operasi standar untuk semua fungsi operasional.
n. Melakukan pertemuan rutin dengan tim untuk membahas tentang masalah, masalah, pembaruan, dll.

3. Product Creative Director
a. Berperan aktif dalam perekrutan dan pelatihan staf kreatif
b. Mengawasi penawaran dan proposal klien
c. Bertemu dengan klien atau manajemen untuk menjelaskan strategi kampanye dan solusi dari permasalahan yang dihadapi
d. Mengulas pekerjaan, memecahkan masalah, dan memberikan masukan kepada tim kreatif
e. Memimpin proses kreatif, dari konsep sampai selesai
f. Menerjemahkan tujuan pemasaran klien menjadi strategi kreatif yang jelas
Memastikan standar komunikasi visual di proyek yang dibuat buat terpenuhi

4. Content Writer
a. Mencari konten atau ide fresh untuk sosial media dan blog atau website
b. Menulis artikel yang berhubungan dengan perusahaan
c. Membantu pemenuhan kebutuhan divisi lain terkait artikel
d. Memanage serta monitoring sosial media
e. Melakukan wawancara apabila diperlukan
f. Memasukan artikel kedalam wordpress atau website
g. Meningkatkan branding dan SEO perusahaan
h. Melakukan inovasi terhadap artikel lama
i. Melakukan research sebelum menentukan topic

5. Marketing Manager & Director
   a. Mengkoordinasi dan meningkatkan penjualan melalui chanel online atau offline
   b. Mengkoordinasikan semua media, organizer acara dan rekan bisnis untuk keperluan promosi dan meningkatkan penjualan
   c. Menjaga efektifitas dari invetory level dengan penjualan
   d. Mengevaluasi pencapaian target sales
   e. Melakukan strategi pemasaran yang efektif serta berorientasi pada pencapaian dan peningkatan target sales
   f. Memberikan pengarahan serta problem solving terhadap masalah yang berkaitan dengan pencapaian sales
   g. Membangun serta menjaga hubungan dengan mitra bisnis, klien dan vendor
   h. Melakukan pembinaan dan penilaian terhadap perilaku dan prestasi bawahan
   i. Mengembangan produk atau jasa dari perusahaan

6. Social Media Specialist
   a. Mengupdate sosial media dengan konten yang sesuai dengan tren
   b. Melakukan monitoring terhadap sosial media kompetitor
   c. Rutin mengevaluasi kegiatan sosial media yang dikerjakan
   d. Membentuk personal connection dengan publik atau klien
   e. Melakukan tracking melalui sosial media
   f. Meresponi setiap keluhan atau masukan dari konsumen
   g. Membangun komunitas
   h. Meningkatkan advocacy brand melalui word of mouth
   i. Membuat strategi meningkatkan trust, kredibilitas serta sales melalui sosial media

7. Marketing Communication

a. Menganalisis perilaku konsumen, termaksud kebutuhan dan kesukaannya
b. Mengkomunikasikan atau mempromosikan produk perusahaan
c. Membentuk pangsa pasar yang loyal
d. Membentuk opini publik
e. Menunjang program atau rencana perusahaan
f. Mempertahankan tingkat permintaan
g. Membuat marketing plan
h. Membuat strategi untuk meningkatkan penjualan, mengangkat produk yang dipasarkan, serta *branding*
i. Memonitoring *digital traffic*
j. Membuat program atau acara yang kreatif

8. Chief Technology Officer (CTO)
   a. Mengembangkan aspek teknis dari strategi perusahaan untuk memastikan keselarasan dengan tujuan bisnisnya
   b. Temukan dan terapkan teknologi baru yang menghasilkan keunggulan kompetitif
   c. Bantu departemen menggunakan teknologi secara menguntungkan
   d. Mengawasi infrastruktur sistem untuk memastikan fungsionalitas dan efisiensi
   e. Membangun proses penjaminan kualitas dan perlindungan data
   f. Monitor KPI dan anggaran TI untuk menilai kinerja teknologi
   g. Gunakan umpan balik dari para pemangku kepentingan untuk menginformasikan perbaikan dan penyesuaian yang diperlukan untuk teknologi
   h. Mengkomunikasikan strategi teknologi kepada mitra dan investor

9. Back-End Developer
   a. Bertanggung jawab untuk menjaga dan mengembangkan semua bagian dari sistem pelayanan
   b. Merencanakan dan melaksanakan struktur model data untuk skalabilitas

c. Menulis kode dan merancang aplikasi aman
d. Berpartisipasi dalam semua fase siklus pengembangan, berfokus pada coding, pengujian, dan debugging
e. Meliharaan atau meningkatan struktur data yang ada
f. Melakukan riset, mengevaluasi dan menganalisa persyaratan teknis dan desain
g. Memecahkan masalah dan memperbaiki bug / kelemahan dalam website dan sistem lain untuk memastikan server berjalan secara optimal
h. Merumuskan konsep dan ide-ide untuk produk tambahan, alat dan layanan yang dapat diberikan dalam ruang digital

2.2 Landasan Teori

2.2.1 Software Quality Assurance

Software Quality Assurance (SQA) didefinisikan sebagai suatu pendekatan terencana dan sistematis untuk melakukan evaluasi kualitas dari sebuah perangkat lunak, standar produk perangkat lunak, proses dan prosedur dalam perangkat lunak [8]. Salah satu bagian dari software quality assurance adalah maintainability.

2.2.1.1 Maintainability

Menurut Rudolp Frederick Stapelberg definisi maintainability dari persepektif pemeliharaan adalah probabilitas terhadap item yang gagal akan dikembalikan ke kondisi efektif operasional dalam waktu tertentu. Sehingga dapat disimpulkan juga bahwa maintainability adalah kemampuan dalam melakukan perwatan terhadap suatu sistem dengan rentang waktu yang ditentukan ketika sistem tersebut memiliki suatu masalah [9]. Dalam ISO-25010, maintainability memiliki beberapa sub faktor yang dapat mempengaruhi suatu sistem dapat dikatakan maintainable apabila memenuhi sub faktor tersebut [3].

Sub Faktor Maintainability ada lima, yaitu.

1. Modularity

   Modularity menjelaskan tentang sejauh mana suatu sistem atau program yang terdiri dari kumpulan modul sedemikian rupa sehingga perubahan pada modul memiliki dampak minimal pada modul lainnya.

2. Reusability

   Reusability menjelaskan sejauh mana suatu asset dapat digunakan lebih dari satu sistem, atau dalam membangun asset lainnya.

3. Analysability

   Analysability menjelaskan tentang tingkat keefektifan dan efisiensi yang memungkinkan untuk menilai dampak dari suatu produk atau sistem dari perubahan yang dimaksudkan untuk satu modul atau lebih. Dapat juga dilakukan untuk mendiagnosis suatu penyebab kegagalan atau dilakukan untuk mengidentifikasi bagian yang akan dimodifikasi

4. Modifiability

   Modifiability menjelaskan tentang sejauh mana suatu produk atau sistem dapat secara efektif dan efisien ketika dilakukan modifikasi tanpa mendapatkan bug atau menurunkan kualitas produk yang ada.

5. Testability

   Testability menjelaskan tentang tingkat keefektifan dan efisiensi dengan kriteria pengujian yang ditetapkan untuk suatu sistem, produk atau komponen pengujian dapat dilakukan untuk menentukan apakah kriteria tersebut telah terpenuhi.

2.2.2 Maintainability Index

Menurut Terzimehic pada paper "Pembangunan Kakas Bantu Untuk Mengukur Maintainability Index Pada Perangkat Lunak Berdasarkan Nilai Halstead Metrics dan McCabe's Cyclomatic Complexity". Maintainability Index (MI) adalah software metric yang megukur sebuah perangkat lunak mudah atau sulit untuk dilakukan perawatan atau perubahan di masa yang akan datang. Maintainability Index dihitung berdasarkan nilai formula dari Halstead's Volume (HV), Cyclomatic Complexity (CC) dan Lines of Code (LOC) [5]. Semakin tinggi nilai dari kalkulasi maintainability index mengindikasikan bahwa kode program memiliki tingkat maintainability yang baik, dimana hal ini akan membuat source code lebih mudah dipahami dan dirawat sehingga akan lebih mudah dalam pencarian dan perbaikan kecacatan (bugs). Adapun formula *Maintainability Index*

dapat dilihat pada Gambar 2.3. dan klasifikasi maintainability index ditujukan pada Tabel 2.1

$$MI = 171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(LOC) + 50 * \sin(\sqrt{2.4 * COM})$$

Gambar 2.3 ***Maintainability Index Formula***

Keterangan :

HV = Halstead Metrics Volumes

CC = Cyclomatic Complexity

LOC = Lines of Code

COM = Percent line of comment

Tabel 2.1 Klasifikasi Penilaian ***Maintainability Index***

Nilai Maintainability Index	Klasifikasi
MI > 85	Dapat dipelihara dengan baik
65 < MI ≤ 85	Cukup untuk dapat dipelihara
MI ≤ 65	Sulit untuk dapat dipelihara

Untuk dapat menghitung maintainability index terdapat beberapa tahapan yang harus dilakukan untuk mendapatkan nilai masukan yang nantinya akan digunakan untuk menghitung nilai maintainability index.

1. Halstead Metric

   Halstead's Metric adalah pengukuran yang dikembangkan untuk mengukur kompleksitas dalam suatu program langsung dari source code. Pengukuran dilakukan dengan menentukan ukuran kuantitatif kompleksitas dari operator dan operand dalam suatu modul sistem. Pada Halstead's metric terdapat enam jenis komponen yaitu :

   a. Length of program

Length of program adalah kalkulasi jumlah total operator dan operand yang muncul. Persamaan Length of the program dirumuskan pada persamaan (1).

$$N = N1 + N2 \qquad (1)$$

Keterangan :

N1 = Jumlah Operator

N2 = Jumlah Operand

b. Vocabulary of the program Vocabulary of the program adalah kalkulasi jumlah operator unik dan operand unik yang muncul dalam program. Persamaan Vocabulary of the program dirumuskan pada persamaan (2).

$$n = n1 + n2 \qquad (2)$$

Keterangan :

n = Vocabulary of program

n1 = Jumlah operator unik

n2 = Jumlah operand unik

c. Volume of the program

Volume of the program adalah volume dalam halstead metric yang digunakan untuk mengetahui volume program. Persamaan volume of the program dirumuskan pada persamaan (3).

$$V = N \times \log 2\, n \qquad (3)$$

Keterangan :

V = Volume of the program

N = Nilai kalkulasi length of the program

n = nilai kalkulasi Vocabulary of the program

d. Difficulty

Difficulty dalam Hakstead Metric digunakan untuk mengetahui kesulitan dan pengembangan program. Persamaan difficulty dirumuskan pada persamaan (4).

$$D = \frac{n1}{2} \; x \; \frac{N2}{n2} \qquad (4)$$

Keterangan :

D = Difficulty

N2 = total semua operand yang muncul

n1 = jumlah operator unik

n2 = jumlah operand unik

e. Effort

Effort dalam halstead metric di gunakan untuk mengetahui sumber daya yang digunakan untuk pengembangan program. Persamaan effort dirumuskan pada persamaan (5).

$$E = D \times V \qquad (5)$$

Keterangan:

E = Effort

D = nilai difficulty

V = Volume of the program

f. Number of bugs expected in the program

Number of bugs expected in the program dalam halstead metric di gunakan untuk mengetahui prediksi bug pada program. Persamaan Number of bugs expected in the program dirumuskan pada persamaan (6).

$$B = \frac{V}{3000} \qquad (6)$$

Keterangan :

B = number of bugs expected in the program

V = nilai kalkulasi Volume of the program

2. Cyclomatic Complexity

McCabe’s Cyclomatic Complexity adalah salah satu metric yang cukup terkenal yang mana metric ini menghitung control flow dari suatu modul. Apabila kompleksitas semakin tinggi maka akan modul tersebut akan semakin sulit untuk diuji dan dirawat (Laird and Brennan 2006). Untuk menghitung Cyclometic Complexity dapat menggunakan dua cara yaitu dengan menghitung nodes dan edge dapat dilihat pada persamaan (7) dan

dengan menghitung node percabangan atau predicate node dapat dilihat pada persamaan (8).

$$V(g) = e - n + 2 \quad (7)$$

$$V(g) = p + 1 \quad (8)$$

Keterangan :

e = jumlah edge

n = jumlah node

p = jumlah predicate node

2.2.3 Phpmetrics

Phpmetrics adalah alat yang dapat digunakan untuk menganalisis statis untuk php. Phpmetrics menyediakan berbagai metric tentang proyek php dan phpmetric juga dapat dirancang agar dapat dipahami dan mudah digunakan. Tampilan yang diberikan oleh phpmetrics dapat membuat skor proyek yang dihasilkan menjadi grafik yang indah dan phpmetrics mempunyai mode buta warna. Phpmetrics dapat menghasilkan beberapa metrik seperti complexity, volume, object oriented, maintainability, dan lain-lain.[10].

2.2.4 Code Readability Metric

Code Readability Metric merupakan metrik yang digunakan untuk menilai suatu tingkat keterbacaan dari suatu kode sumber. Penilaian pada kode sumber dilakukan dengan menguji kode sumber tersebut ke partisipan atau dengan pengembang dari aplikasi yang diteliti [11]. Bobot penilaian disediakan oleh peneliti, sehingga partisipan cukup memberikan nilai yang sesuai berdasarkan rentang bobot nilai yang disediakan. Penggunaan *Code Readability Metric* sendiri dimaksudkan untuk mengumpulkan data latih yang digunakan oleh program yang dibangun untuk menilai *readability* atau tingkat keterbacaan dari suatu kode sumber berbasis Java. Penelitian tentang *Code Readability Metric* ini dikembangkan oleh Raymond P. L. Buse dan Westley R. Weimer. Sehingga pada penelitian ini, penilaian tingkat keterbacaan dari suatu kode sumber mengikuti metode yang sudah mereka lakukan.

Source Code Readability (SCREA) memiliki beberapa atribut untuk melakukan perhitungan nya, dapat dilihat pada Tabel 2.1.

Tabel 2.2 Tabel Atribut Code Readability

Akronim	Deskripsi	Akronim	Deskripsi
RMN	Readable Method Names	MN	Numbers of methods defined in a file
RBL	Readable Blank Lines	BL	Numbers of blank lines in a file
RCM	Readable Class Names	CM	Numbers of class names defined in a file
RVN	Readable Variable Names	VN	Numbers of variables defined in a file
RL	Readable Lines	LN	Numbers of lines defined in a file

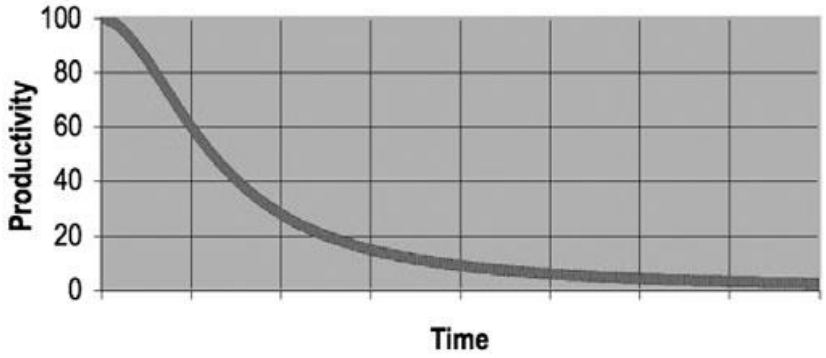
Setelah mendapatkan atribut tersebut dapat dilakukan perhitungan metric nya dengan formula dibawah ini

$$SCREA_{php} = \frac{\sum_{c=1}^{d} RMN}{\sum_{a=1}^{b} MN} \times w1 + \frac{\sum_{g=1}^{h} RBL}{\sum_{e=1}^{f} BL} \times w2 + \frac{\sum_{k=1}^{l} RCM}{\sum_{i=1}^{j} CM} \times w3 + \frac{\sum_{o=1}^{p} RVN}{\sum_{m=1}^{n} VN} \times w4 + \frac{\sum_{s=1}^{t} RL}{\sum_{q=1}^{r} LN} \times w5$$

2.2.5 Clean Code

Clean code merupakan suatu konsep atau petunjuk penulisan struktur kode yang bersih *(Clean)*, di mana kode yang ditulis dapat dibaca oleh pengembang lain dan dapat diubah dengan mudah [3]. *Clean code* bertujuan untuk mengatasi penurunan tingkat produktivitas pengembangan perangkat lunak akibat dari

struktur kode yang berantakan seperti yang dapat dilihat pada Gambar 2.4 di mana waktu dapat mempengaruhi tingkat produktivitas [3].



Gambar 2.4 *Productivity vs. Time*

Adapun hal inti yang dibahas dalam *Clean code* adalah sebagai berikut:

1. *Meaningful Names*
2. *Clean Functions*
3. *Clean Comments*
4. *Clean Code Formatting*
5. *Clean Error Handling*
6. *Clean Object and Data Structures*
7. *Clean Classes*

2.2.5.1 Meaningful Name

Pada konsep ini terdapat petunjuk dalam melakukan pemberian nama terhadap variabel, *method*/fungsi, dan *modul* agar lebih mudah untuk dipahami. Berikut merupakan petunjuk yang dapat digunakan dalam melakukan pemberian nama:

1) Nama yang dapat dieja dan memiliki arti

Penggunaan nama yang dapat dieja dan secara eksplisit menerangkan kegunaan dari kode yang ditulis dapat memudahkan pengembang lain dalam memahaminya. Sehingga pengembang tidak perlu mengeluarkan usaha yang banyak hanya untuk memahami kegunaan kode tersebut.

2) Nama yang mudah untuk dicari

Petunjuk ini digunakan untuk memudahkan pencarian kode. Dengan menggunakan penamaan yang mudah dicari, pengembang tidak perlu memahami program secara keseluruhan. Hal ini biasa diterapkan untuk menamai suatu konstanta.

3) Menghindari *Mental Mapping*

Penggunaan singkatan seperti i dan n untuk penamaan variabel iterasi dan jumlah, atau penggunaan singkatan lain yang tidak diketahui oleh orang pada umumnya dapat mempersulit proses pemahaman dari kode yang ditulis. Sehingga pengembang disarankan untuk menghindari penggunaan singkatan tersebut.

4) Kata benda untuk *modul*

Dalam melakukan pemberian nama terhadap *modul*, disarankan untuk menggunakan kata benda dan menghindari penggunaan kata kerja. Hal tersebut didasari oleh penggunaan *modul* yang biasanya digunakan mewakili suatu entitas. Adapun contohnya seperti penggunaan kata *User*, *AuthController*, *MessageParser*, dan *Controller*.

5) Kata kerja untuk *method*/fungsi

Tidak seperti penamaan *modul*, penamaan pada *method*/fungsi disarankan untuk menggunakan kata kerja. Hal tersebut dilakukan untuk mendeskripsikan tujuan dari pekerjaan yang dilakukan oleh *method*/fungsi yang ditulis secara eksplisit. *Method*/fungsi *accessors*, *mutators*, dan *predicates* biasanya memiliki prefiks *set*, *get*, dan *is*.

6) Menghindari penggunaan konteks yang tidak diperlukan

Hal ini biasa terjadi pada suatu *modul* atau pada objek, di mana atributnya mengandung nama dari *modul* atau objek tempat atribut tersebut berada. Hal tersebut tidak disarankan, karena bersifat repetitif.

2.2.5.2 Clean Function

Pada konsep ini terdapat petunjuk dalam menulis *method*/fungsi yang bersih agar lebih mudah untuk dipahami. Berikut merupakan petunjuk yang dapat digunakan dalam menulis *method*/fungsi:

1) Jumlah parameter

Dalam menulis *method*/fungsi yang memiliki parameter, jumlah parameter ideal dari suatu fungsi maksimal sebanyak 2 parameter. Hal tersebut dimaksudkan untuk menghindari banyaknya varian pengujian terhadap *method*/fungsi yang ditulis. Apabila terdapat kebutuhan di mana parameter yang digunakan lebih dari jumlah ideal, cukup menggunakan objek sebagai parameter dari *method*/fungsi tersebut.

2) Menghindari penggunaan *flag* sebagai parameter

Penggunaan *flag* sebagai parameter dapat menandakan *method*/fungsi yang dibangun memiliki pekerjaan lebih dari satu.

3) Jumlah pekerjaan

Suatu *method*/fungsi disarankan untuk melakukan hanya satu pekerjaan saja. Suatu *method*/fungsi yang melakukan lebih dari satu pekerjaan akan sulit untuk dibuat, diuji, dan dipahami. *Method*/fungsi yang lebih kecil akan lebih mudah untuk

dimodifikasi. Apabila *method*/fungsi yang ditemukan melakukan lebih dari satu pekerjaan, pecah pekerjaan-pekerjaan tersebut ke dalam *method*/fungsi yang berbeda.

4) Hindari efek samping

Maksud dari menghindari efek samping ini, yaitu menghindari mutasi terhadap variabel global atau parameter yang bersifat referensi oleh *method*/fungsi yang ditulis karena hal tersebut bersifat fatal. Pada bahasa pemrograman yang digunakan pada penelitian ini, yaitu *Javascript*, tipe data primitif dimasukkan sebagai suatu nilai saat dijadikan sebagai argumen, sedangkan larik dan objek dimasukkan sebagai suatu referensi.

2.2.5.3 Clean Comment

Pada konsep ini terdapat petunjuk dalam melakukan pemberian komentar yang baik dan efisien, agar komentar yang ditulis dapat memberikan informasi yang tidak tersampaikan oleh kode sumber yang sudah ditulis. Akan tetapi, pemanfaatan *clean comment* berhubungan dengan pemanfaatan *meaningful names*, di mana semakin jelas penamaan dari suatu variabel, *method*/fungsi, maupun *modul* semakin sedikit pula komentar yang harus diberikan. Berikut merupakan petunjuk dalam melakukan penulisan komentar:

1) Komentar hanya untuk hal-hal yang memiliki kompleksitas logika

Penulisan kode sumber yang baik dapat menghasilkan kode yang mudah dipahami sehingga penggunaan komentar dapat dikurangi. Akan tetapi, tidak semua kode yang baik dapat mengurangi kompleksitas logika yang rumit, sehingga pengembang membutuhkan petunjuk dalam memahami kode sumber tersebut.

2) Komentar yang informatif dan memberikan klarifikasi

Komentar yang baik dapat menjelaskan informasi mendasar dari kode sumber yang ditulis, seperti menjelaskan keluaran dari *method*/fungsi yang dibuat atau menjelaskan format yang tidak bisa dibaca oleh manusia (*Regular Expression*).

Upaya tersebut dilakukan untuk memberikan klarifikasi terhadap pengembang yang sedang mencoba untuk memahami kode sumber yang ada.

3) Menghindari kode yang dikomentari pada basis kode

Suatu kode yang terpakai dan dikomentari tidak disarankan untuk dibiarkan begitu saja. Karena akan merumitkan kode sumber dan membuat pengembang menerka-nerka kegunaan dari kode tersebut. Simpan kode tersebut ke dalam riwayat dengan memanfaatkan *version control* seperti *Git*, sehingga kode tersebut akan tetap aman dan basis kode menjadi lebih bersih.

2.2.5.4 Clean Code Formatting

Pada konsep ini terdapat petunjuk dalam melakukan penulisan kode sumber agar mudah untuk dibaca dengan memanfaatkan format penulisan, hal ini meliputi penggunaan *indentation*, spasi, tab dan penempatan kode berdasarkan penggunaannya. Penentuan format penulisan dapat mengikuti *code convention/code styling* dari bahasa pemrograman yang digunakan atau mengikuti kesepakatan dari tim pengembang. Berikut merupakan petunjuk penulisan kode sumber dengan format penulisan yang baik:

1) Konsisten

Konsistensi terhadap format penulisan perlu diperhatikan agar kode yang ditulis tidak berantakan dan dapat mempermudah pengembang lain dalam memahami kode tersebut. Meskipun kode yang ditulis tidak sesuai dengan *code convention/code styling* dari bahasa pemrograman yang digunakan tidak menjadi kendala selama format penulisan yang digunakan konsisten dan mudah untuk dibaca.

2) Indentation

Penggunaan *indentation* pada penulisan kode dapat mempermudah pengembang dalam membedakan setiap blok logika yang terdapat pada kode tersebut.

3) Penempatan *method*/fungsi yang saling berkaitan

Penempatan *method*/fungsi yang dipanggil dan *method*/fungsi yang memanggil disarankan untuk saling berdekatan agar penelusuran logika dan kesalahan dapat dilakukan dengan mudah.

2.2.5.5 Clean Error Handling

Pada konsep ini terdapat petunjuk dalam menggunakan penanganan kesalahan *(Error Handling)* agar dapat digunakan secara efisien dan pesan kesalahan dapat ditampilkan dengan mudah. Petunjuk sederhana dalam menerapkan penanganan kesalahan yang bersih yaitu dengan tidak mengabaikan kesalahan yang tertangkap.

Kesalahan yang terjadi biasanya hanya ditanggulangi dengan cara menampilkannya ke *logger* tanpa tindakan lebih lanjut sehingga pengguna tidak mengetahui kesalahan apa yang terjadi saat menggunakan aplikasi. Diperlukan tindakan lebih lanjut agar pesan kesalahan tidak hanya ditampilkan ke *logger*, tetapi aplikasi dapat memberikan pesan berupa notifikasi, baik terhadap pengguna maupun pengembang, tentang kesalahan yang terjadi.

2.2.5.6 Clean Object and Data Structure

Pada konsep ini terdapat petunjuk dalam membuat struktur data yang bersih di mana struktur data memiliki abstraksi sehingga tidak dapat diakses langsung secara publik. Berikut merupakan petunjuk dalam membuat struktur data yang bersih:

1) Membuat objek memiliki *private member*

Agar struktur data yang digunakan memiliki abstraksi, *modul* yang digunakan harus memiliki *private member* di dalamnya. Pada bahasa pemrograman yang digunakan pada penelitian ini, *private member* pada suatu *modul* tidak secara eksplisit dapat

dilakukan, akan tetapi hal tersebut dapat dilakukan dengan memanfaatkan teknik *Closure.*

2) Getter dan *Setter*

Getter dan *Setter* digunakan untuk mengakses struktur data yang bersifat *private*. *Getter* dan S*etter* bertujuan untuk menghindari mutasinya nilai pada struktur data yang dilakukan sengaja maupun tidak.

2.2.5.7 Clean Class

Pada konsep ini terdapat petunjuk dalam membuat *modul* yang lebih bersih dan terorganisir. Berikut merupakan petunjuk yang dapat dalam membuat *modul*:

1) Class Organization

Pembuatan *modul* yang baik dapat dilakukan dengan mengikuti *code convetion/code styling* dari bahasa pemrograman yang digunakan

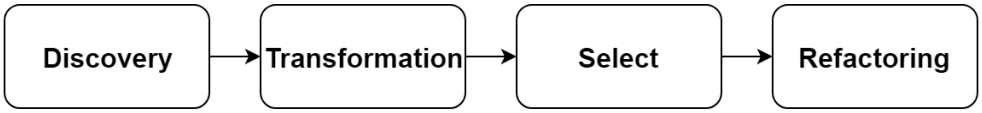
2) Single Responsibility Principle

Suatu *modul* disarankan untuk berukuran tidak terlalu besar, selain banyak jumlah baris yang ditulis, pengembang akan kerepotan dalam memahami *modul* tersebut. Dengan menjaga ukuran *modul* untuk tidak terlalu besar, dapat mempermudah proses modifikasi. Ukuran suatu *modul* diukur berdasarkan jumlah *responsibility* yang ditanggung oleh *modul* tersebut. *Modul* yang bersih merupakan *modul* yang hanya memiliki satu *responsibility* saja. Hampir sama dengan *clean function*, apabila *modul* tersebut memiliki lebih dari satu *responsibility*, pisahkan *responsibility* ke dalam *modul* yang berbeda.

2.2.6 Refactoring

Refactoring merupakan suatu tindakan perubahan terhadap struktur internal dari perangkat lunak agar lebih mudah untuk dipahami dan melakukan modifikasi tanpa mempengaruhi perilakunya (UML Distilled, 2003) [6]. Proses *refactoring* meliputi penyederhanaan logika yang kompleks, penghapusan duplikasi, dan

melakukan klarifikasi terhadap kode yang tidak jelas [6]. Adapun tahapan proses yang dilakukan dalam *refactoring* secara umum dapat dilihat pada Gambar 2.5.



Gambar 2.5 Tahapan *Refactoring* Secara Umum

1. *Discovery*

Discovery merupakan tahap dalam melakukan pencarian kecacatan pada kode sumber yang diperiksa. Kecacatan tersebut dapat berupa duplikasi, struktur kode yang berantakan, atau pun pustaka yang sudah usang *(deprecated)*.

2. *Transformation*

Pada tahap ini, dari hasil pencarian kecacatan yang ditemukan, ditentukan bentuk-bentuk perubahan yang dapat diterapkan untuk memperbaiki kecacatan yang ditemukan.

3. *Select*

Pada tahap ini dilakukan pemilihan dari bentuk solusi yang terbentuk agar solusi yang diterapkan merupakan solusi terbaik yang dapat diberikan.

4. *Refactoring*

Pada tahap ini dilakukan *refactoring* atau penerapan solusi yang dipilih.

2.2.7 Analisis dan Desain Berorientasi Objek

Analisis dan Desain Berorientasi Objek (*Object Oriented Analysis and Design*) adalah cara baru dalam memikirkan suatu masalah dengan menggunakan model yang dibuat menurut konsep. Dasar pembuatannya sendiri adalah objek yang merupakan kombinasi antara struktur data dan perilaku dalam satu entitas. Alasan mengapa harus memakai metode berorientasi objek yaitu karena perangkat lunak

itu sendiri yang bersifat dinamis, di mana hal ini disebabkan karena kebutuhan pengguna berubah dengan cepat [12].

Selain itu bertujuan untuk menghilangkan kompleksitas transisi antar tahap pada pengembangan perangkat lunak, karena pada pendekatan berorientasi objek, notasi yang digunakan pada tahap analisis perancangan dan implementasi relatif sama tidak seperti pendekatan konvensional yang dikarenakan notasi yang digunakan pada tahap analisanya berbeda-beda. Hal itu menyebabkan transisi antar tahap pengembangan menjadi kompleks [12].

Di samping itu dengan pendekatan berorientasi objek membawa pengguna kepada abstraksi atau istilah yang lebih dekat dengan dunia nyata, karena di dunia nyata itu sendiri yang sering pengguna lihat adalah objeknya bukan fungsinya. Beda ceritanya dengan pendekatan terstruktur yang hanya mendukung abstraksi pada level fungsional. Adapun dalam pemrograman berorientasi objek menekankan berbagai konsep seperti: *Class, Object, Abstract, Encapsulation, Polymorphism, Inheritance* dan tentunya UML (*Unified Modeling Language*) [12].

UML (*Unified Modeling Language*) sendiri merupakan salah satu alat bantu yang dapat digunakan dalam Bahasa pemrograman berorientasi objek. Selain itu UML merupakan *standard modeling language* yang terdiri dari kumpulan-kumpulan diagram, dikembangkan untuk membantu para pengembang sistem dan perangkat lunak agar bisa menyelesaikan tugas-tugas seperti: Spesifikasi, Visualisasi, Desain Arsitektur, Konstruksi, Simulasi dan Testing. Dapat disimpulkan bahwa UML (*Unified Modeling Language*) adalah sebuah Bahasa yang berdasarkan grafik atau gambar untuk memvisualisasikan, melakukan spesifikasi, membangun dan pendokumentasian dari sebuah sistem pengembangan perangkat lunak berbasis objek (*Object Oriented Programming*) [12].

Dokumentasi UML menyediakan 10 macam diagram untuk membuat model aplikasi berorientasi objek yang 4 di antaranya sebagai berikut:

1. *Use Case Diagram*

   Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Di dalam *use case diagram* ini sendiri lebih ditekankan kepada apa yang diperbuat sistem dan bagaimana sebuah sistem itu bekerja. Sebuah *use*

case merepresentasikan sebuah interaksi antara *actor* dengan sistem. *Use case* merupakan bentuk dari sebuah pekerjaan tertentu, misalnya *login* ke dalam sistem, *cetak document* dan sebagainya, sedangkan seorang *actor* adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu [13].

2. *Use Case Scenario*

   Sebuah diagram yang menunjukkan *use case* dan aktor mungkin menjadi titik awal yang bagus, tetapi tidak memberikan detail yang cukup untuk desainer sistem untuk benar-benar memahami persis bagaimana sistem dapat terpenuhi. Cara terbaik untuk mengungkapkan informasi penting ini adalah dalam bentuk penggunaan *use case scenario* berbasis teks per *use case*-nya [13].

3. *Activity Diagram*

   Activity Diagram adalah sebuah tahapan yang lebih fokus kepada menggambarkan proses bisnis dan urutan aktivitas dalam sebuah proses. Di mana biasanya dipakai pada bisnis modeling untuk memperlihatkan urutan aktivitas proses bisnis. *Activity diagram* ini sendiri memiliki struktur yang mirip dengan *flowchart* atau data *flow* diagram pada perancangan terstruktur. *Activity diagram* dibuat berdasarkan sebuah atau beberapa *use case* pada *use case diagram* [13].

4. *Sequence Diagram*

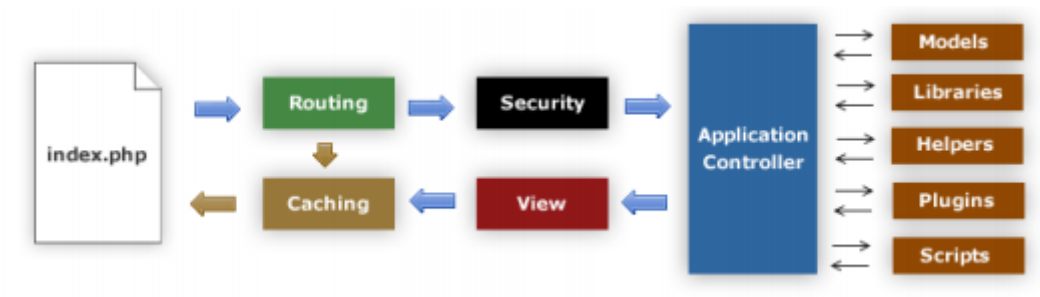
   Sequence diagram digunakan untuk menggambarkan perilaku pada sebuah *scenario*. Diagram jenis ini memberikan kejelasan sejumlah objek dan pesan-pesan yang diletakkan di antaranya di dalam sebuah *use case*. Komponen utamanya adalah objek yang digambarkan dengan kotak segi empat atau bulat, *message* yang digambarkan dengan gari putus dan waktu yang ditunjukkan dengan *progress vertical*. Manfaat dari *sequence diagram* adalah memberikan gambaran detail dari setiap interaksi yang terjadi pada *use case diagram* yang dibuat sebelumnya [13].

5. *Class Diagram*

   Class diagram adalah sebuah *class* yang menggambarkan struktur dan penjelasan *class,* paket dan objek serta hubungan satu sama lain. *Class diagram* juga menjelaskan hubungan antar *class* secara keseluruhan di dalam sebuah sistem yang sedang dibuat dan bagaimana caranya agar mereka saling berkolaborasi untuk mencapai sebuah tujuan [13].

2.2.8 CodeIgniter

CodeIgniter adalah sebuah framework yang digunakan untuk membangun sebuah situs web dengan menggunakan bahasa pemrograman php. Tujuan adanya framework codeigniter adalah untuk memungkinkan kita mengembangkan proyek lebih cepat daripada menulis kode dari awal. CodeIgniter juga menyediakan library yang biasanya digunakan, agar pengerjaan juga semakin cepat. CodeIgniter memungkinkan kita untuk focus terhadap kode dengan meminimalkan jumlah kode yang diperlukan untuk satu proses yang diberikan [1]. Dalam CodeIgniter terdapat beberapa tahapan flow chart yang harus dilalui seperti pada Gambar 2.6.



Gambar 2.6 Flow Chart CodeIgniter

Berikut adalah penjelasan dari flow chart yang ada pada CodeIgniter :

1. Index.php berfungsi sebagai pengontrol depan, menginisialisasi sumber daya dasar yang diperlukan untuk menjalankan CodeIgniter.
2. Router berfungsi untuk memeriksa permintaan HTTP untuk menentukan apa yang harus dilakukan dengan permintaan tersebut.
3. Jika ada file cache, maka akan dikirim langsung ke browser, melalui eksekusi sistem secara normal.
4. Security berfungsi untuk mengontrol aplikasi yang dimuat, permintaan HTTP dan data yang dikirimkan pengguna difilter untuk keamanan.
5. Application controller berfungsi sebagai pengendali untuk memuat model, library, dan sumber daya lainnya yang diperlukan untuk memproses permintaan spesifik.
6. View berfungsi untuk menampilkan tampilan yang telah selesai direnderkemudian dikirim ke browser web untuk dilihat. Jika caching diaktifkan, tampilan di-cache terlebih dahulu sehingga pada permintaan selanjutnya dapat dilayani.

CodeIgniter didasarkan pada pola pengembangan Model-View-Controller. MVC adalah pendekatan perangkat lunak yang memisahkan logika aplikasi dari presentasi.

1. Model mewakili struktur data developer. Biasanya kelas model akan berisi fungsi yang membantu anda mengambil, menyisipkan dan memperbarui informasi dalam database.

2. View adalah informasi yang disajikan kepada pengguna. View biasanya akan menjadi halaman web, tetapi dalam CodeIgniter, tampilan juga bisa berupa fragmen halaman seperti header atau footer.
3. Controller berfungsi sebagai perantara antara model, view dan sumber daya lainnya yang diperlukan untuk proses permintaan HTTP dan menghasilkan halaman web.