BAB 2
TINJAUAN PUSTAKA

2.1 Profil Museum Mandhilaras

Museum mandilaras Pamekasan Jawa Timur merupakan sebuah museum yang dikelola oleh pemerintah setempat. Museum ini diresmikan pada tahun 2010, tepatnya di tanggal 18 maret. Museum ini dinamakan mandilaras karena berhubungan dengan sejarah kerajaan keratin mandilaras yang merupakan awal dari berdirinya kabupaten pamekasan pada abad ke 16. Hal ini terjadi karena penambahan ronggosukowati mulai memindahkan pusat pemerintahan dari Kraton Labangan Daja ke Kraton Mandilaras [8].

2.2 Landasan Teori

Pada penelitian ini dibutuhkan teori-teori yang dibutuhkan untuk mendukung permasalah dan ruang lingkup pembahasan dalam pembangunan penelitian ini.

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 [9].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 [10]. Dalam ISO-25010, *maintainability* memiliki beberapa sub faktor yang dapat mempengaruhi suatu sistem dapat dikatakan *maintainable* apabila memenuhi sub faktor tersebut [11].

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). Persamaan *maintainability index* ditujukan pada persamaan (1) dan klasifikasi *maintainability index* ditujukan pada Tabel 2.1.

$$MI = 171 - 5{,}2 \times \ln(HV) - 0.23 \times CC - 16.2 \times \ln(LOC) + \left(50 \times \sin\left(\sqrt{2.46} + perCM\right)\right) | \quad (1)$$

Keterangan :

HV = *Halstead Metrics Volumes*

CC = *Cyclomatic Complexity*

LOC = *Lines of Code*

perCM = Percent line of comment

Tabel 2.1 Klasifikasi *Maintainability Index*

Nilai *Maintainability Index*	Klasifikasi
MI > 85	*Highly maintainable*
65 < MI ≤ 85	*Moderately maintainable*
MI ≤ 65	*Difficult to maintain*

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 (2).

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

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 (3).

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

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 (4).

$$V = N \times \log_2 n \quad (4)$$

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 (5).

$$D = \frac{n1}{2} \times \frac{N2}{n2} \quad (5)$$

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 (6).

$$E = D \times V \quad (6)$$

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 (7).

$$B = \frac{V}{3000} \quad (7)$$

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 (8) dan dengan menghitung node percabangan atau *predicate node* dapat dilihat pada persamaan (9).

$$V(g) = e - n + 2 \quad (8)$$

$$V(g) = p + 1 \quad (9)$$

Keterangan :

e = jumlah edge

n = jumlah node

p = jumlah predicate node

2.2.3 Clean Code

Menurut Bjorne Stroustrup dalam buku Clean Code A Handbook of Agile menyebutkan bahwa *clean code* harus elegan dan efisien. Logikanya agar mudah untuk membuatnya sulit dalam menyembunyikan bug, minimal perawatannya mudah, penanganan kesalahannya selesai sesuai dengan strategi yang diartikulasikan dan kinerja mendekati optimal agar tidak tergoda oleh orang yang membuat *messy code* dengan prinsip optimasi. Sedangkan menurut Grady, *clean code* itu simpel, *clean code* dibuat dengan baik seperti prosa. *Clean code* tidak pernah merubah desain, tapi membuat *code* lebih mudah untuk di control. Sehingga dapat disimpulkan juga bahwa *Clean Code* merupakan sebuah metode untuk melakukan perbaikan terhadap suatu program, dimana program tersebut mempunyai struktur *code* yang sulit dipahami oleh developer lain [3]. Ada beberapa faktor yang mempengaruhi sebuah *code* tersebut dapat disebut sebagai *clean code*.

2.2.3.1 Meaningful Names

Meaningful Names merupakan aturan aturan untuk melakukan penamaan terhadap kelas,variabel dan fungsi agar mudah untuk dipahami oleh developer lain. Berikut adalah beberapa aturan dalam meaningfulnames.

1. *Use Intention-Revealing Names*

Use Intention-Revealing Names maksudnya adalah penamaan dalam suatu variable, fungsi, kelas, itu harus mengungkapkan suatu tujuan, apabila nama tersebut masih membutuhkan komentar maka nama tersebut tidak mempresentasikan makna atau tujuan dari nama tersebut.

2. *Avoid Disinformation*

   Avoid Disinformation maksudnya adalah hindari penggunakan nama sebuah platform atau varian unix. Hindari juga penamaan yang bervariaso dalam hal kecil dan juga penggunaan huruf L kecil dan O besar sebagai nama variabel, terutama dalam kombinasi.

3. *Use Pronounceable Names*

   Use Pronounceable Names maksudnya adalah menggunakan nama yang dapat diucapkan oleh manusia, sehingga manusia dapat mengerti maksud dari nama tersebut.

4. *Use Searchable Name*

   Use Searchable Name maksudnya adalah untuk menggunakan nama yang mudah dicari. Nama huruf tunggal dan konstanta numerik memiliki masalah tertentu karena tidak mudah untuk ditemukan di seluruh badan teks, penggunaan huruf e untuk variable juga pilihan yang buruk, karena e adalah kata yang paling umum dalam bahasa inggris. Pastikan bahwa penamaan bukan nama yang sangat berguna, tapi setidaknya dapat dicari.

5. *Avoid Mental Mapping*

   Avoid Mental Mapping maksudnya adalah masalah dalam penamaan variable satu huruf. Dalam *looping* biasanya menggunakan huruf i, j atau k. apabila skalanya kecil hal tersebut tidak jadi masalah. Dalam hal lain penamaan satu huruf adalah pilihan yang sangat buruk.

6. *Class Names*

   Class Names maksudnya adalah nama kelas dan objek harus memiliki nama frasa kata benda atau kata benda seperti *Customer, WikiPage* dan *Account*. Hindari kata-kata seperti *Manager, Processor, Data* atau *Info*

untuk penamaan sebuah kelas. Penamaan kelas tidak boleh menggunakan kata kerja dan huruf pertama kelas harus huruf capital.

7. *Method Names*

   Method Names maksudnya adalah nama metode harus memiliki nama dari kata kerja atau frasa kata kerja seperti *postpayment, deletePage.*

8. *Don't be Cute*

   Don't be Cute maksudnya adalah jangan menggunakan kata-kata yang menurut kita lucu, tetapi gunakan kata apa yang kita maksud, apa yang kita maksud yaitu aoa yang kita katakan.

9. *Don't Pun*

   Don't Pun maksudnya adalah kita sebagai developer harus menggunakan kata yang mempunyai tujuan yang sama tetapi memiliki sintaks yang berbeda. Contohnya adalah penggunaan kata add yang sudah terlalu banyak dalam sebuah *class*, maka kita dapat menggunakan sintaks lain yang memiliki arti yang sama seperti *insert* atau *append*, yang dimana penulisannya sangat berbeda tetapi secara arti mempunyai tujuan yang sama dengan *add*.

2.2.3.2 Functions

Function merupakan sebuah method yang ada dalam sebuah program yang bertujuan untuk menampung sebuah proses yang ada dalam program tersebut.

1. *Small*

   Small maksudnya adalah penamaan fungsi, pernyataan dalam fungsi dan baris yang dimana sebisa mungkin di dalam suatu fungsi itu harus singkat dan jelas. contohnya itu baris pernyataan sebisa mungkin tidak lebih 20 sampai 30 dan akan lebih baik apabila hanya 3 sampai dengan 4 baris saja.

2. *Do One Thing*

   Do One Thing maksudnya adalah fungsi hanya melakukan satu hal, tidak melakukan lebih dari satu hal, kecuali fungsi tersebut tidak dapat di pecah menjadi beberapa bagian karena saling berhubungan.

3. *One Level of Abstraction per Function*

   One Level of Abstraction per Function maksudnya adalah untuk memastikan fungsi kita melakukan "satu hal", kita perlu memastikan bahwa pernyataan dalam fungsi kita berada pada tingkat abstraksi yang sama. Pembacaan *code* fungsi dibaca dari atas kebawah.

4. *Use Descriptive Names*

   Use Dexcriptive Names maksudnya adalah memilih nama yang baik untuk fungsi kecil yang melakukan satu hal. Semakin kecil dan focus suatu fungsi, semakin mudah untuk memilih nama yang deskriptif. Nama yang panjang lebih baik daripada nama yang pendek tetapi membingungkan.

5. *Function Arguments*

   Function Arguments maksudnya adalah argument yang ideal untuk suatu fungsi adalah nol, apabila tidak bisa maka minimal satu argument dan maksimal dua argument. Apabila sampai harus tiga argument, maka harus dihindari jika memungkinkan. Lebih dari tiga argument lebih baik untuktidak digunakan. Ketika membaca sebuah *code* yang diceritakan dalam suatu modul, "includeSetupPage ()" lebih mudah dipahami daripada "includeSetupPageInfo(newPageContent)".

6. *Command Query Separation*

   Command Query Separation maksudnya adalah fungsi harus melakukan sesuatu atau menjawab sesuatu, tidak bisa keduanya. Melakukan dua pekerjaan sering menyebabkan kebingungan dalam pembacaan code.

7. *Don't Repeat Yourself*

   Don't Repeat Yourself maksudnya adalah jangan melakukan duplikasi fungsi, hal tersebut dapat menyebabkan dalam melakukan kelalaian

dalam melakukan perawatan, karena developer membaca fungsi yang mirip sehingga sulit mana yang harus diperbaiki.

2.2.3.3 Comment

Comment merupakan sebuah pemberi informasi terhadap developer lain untuk dapat memahami *code* yang kita buat. Namun tidak sedikit juga yang memberikan comment yang tidak perlu.

1. *Good Comment*

   Dalam good comment, komentar diperlukan atau mempunya manfaat, tetapi satu-satunya komentar yang benar-benar baik adalah komentar yang kita tidak harus tulis. Apabila kita membutuhkan komentar, ada beberapa komentar yang dapat digunakan yaitu :

   a. Legal Comment yaitu komentar yang terpaksa harus ditulis karena alasan hokum seperti komentar *copyright*.
   b. Informative Comments yaitu komentar yang berguna untuk menyediakan informasi.
   c. Explanation of Intent yaitu komentar yang melampaui informasi bermanfaat tentang implementasi *code*. Ketika kita membandingkan dua objek, penulis memutuskan bahwa dia ingin mengurutkan objek kelasnya lebih tinggi dari objek yang lain.
   d. Clarification yaitu komentar yang digunakan untuk membantu menerjemahkan arti dari beberapa argument yang tidak jelas.
   e. Warning of Consequences yaitu komentar yang berguna untuk memperingati developer lain tentang konsekuensi apabila kode tersebut berubah.

2. *Bad Comment*

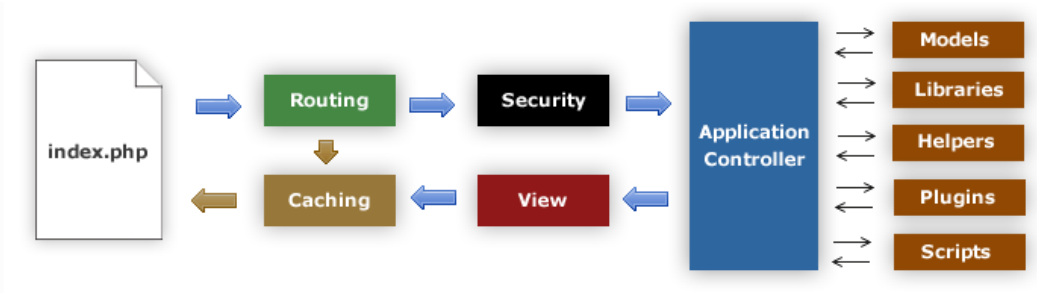
   Sebagian besar dalam suatu aplikasi komentar termasuk dalam *bad comment*. Komentar yang diberikan terkadang seharusnya tidak usah diberikan, tetapi developer memasukan komentar terhadap suatu *code* yang berarti *code* tersebut tidak ekspresif. Beberapa komentar yang selalu diberikan oleh developer antara lain :

a. Numbling yaitu komentar yang hanya karena kita harus melakukannya atau karena proses mengharuskannya. Kita harus memastikan bahwa komentar yang kita berikan adalah komentar yang baik atau tidak.
b. Redudant Comments yaitu komentar yang diberikan secara berulang. Hal ini dapat membuat program saat dieksekusi akan terasa sangat lambat.
c. Journal Comments yaitu komentar yang ditambahkan oleh developer ke awal modul setiap kali developer tersebut melakukan perubahan kode. Sehingga komentar-komentar ini terakumulasi sebagai semacam jurnal atau log dari setiap perubahan yang pernah dilakukan.
d. Noise Comments yaitu komentar yang menyatakan kembali kode yang sudah jelas tanpa memberikan informasi yang baru.

2.2.4 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 [2].

Dalam CodeIgniter terdapat beberapa tahapan flow chart yang harus dilalui seperti pada Gambar 2.1.



Gambar 2.1 Flow Chart CodeIgniter[2]

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.

2.2.5 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.[12].

2.2.6 Refactoring

Menurut Ralp Johnson Group refactoring merupakan perubahan yang dilakukan pada struktur internal perangkat lunak untuk membuatnya lebih mudah dipahami dan lebih mudah untuk memodifikasi tanpa mengubah perilaku yang dapat diamati. Sedangkan menurut assorted associates refactoring adalah meresuktrurisasi perangkat lunak dengan menerapkan serangkaian perbaikan tanpa mengubah perilaku yang dapat diamati. Sehingga dapat disimpulkan bahwa refactoring adalah sebuah perubahan terhadap sebuah code internal menjadi lebih baik tanpa mengubah fitur atau fungsi utama yang sudah ada [7].

Terdapat beberapa tahapan yang harus dilakukan saat melakukan refactoring, diantaranya adalah sebagai berikut:

1. *Composing Method*
2. *Remove Temps*
3. *Moving Features Object*
4. *Making Method Calls Simpler*