BAB 2
LANDASAN TEORI

2.1 Software Quality Assurance

Software Quality Assurance(SQA) merupakan proses untuk memastikan bahwa perangkat lunak atau aplikasi yang direncanakan dari awal di evaluasi meliputi standar produk,proses dan prosedur dalam aplikasi.[4] *Software Quality Assurance* juga dapat memastikan bahwa standar proses dan prosedur yang dibuat sesuai *dengan Software Development Life Cycle*(SDLC) pada saat proses *planning* atau perencanaan.

Pada Software Quality Assurance orang yang melakukan pekerjaan ini dinamai Quality Assurance Tester(QA Tester) yang bertugas untuk membuat alur pengujian,membuat laporan pengujian,memberikan masukan aplikasi yang diuji dan banyak lainnya. Untuk pekerjaan ini dibutuhkan kemampuan kemampuan sebagai berikut :

1. Mindset Pengujian
2. Analisa dan pengujian fungsional
3. Perbaikan proses
4. Perbaikan performa
5. Otomatisasi
6. *User* Acceptance Test

Sementara dalam pengujian software ini terbagi menjadi 2 jenis yaitu:

1. Pengujian Manual

   Pengujian ini biasanya dilakukan untuk mengecek aliran aplikasi,pemeriksaan cacat(desain atau pemograman) , pengujian sistem operasi berbeda,serta uji migrasi aplikasi versi terdahulu
2. Pengujian Otomatis

   Berbeda dengan pengujian manual,pengujian otomatis terdiri dari pengujian regresi,pengujian otomatis yang dilakukan biasanya pada malam hari atau dini hari,pelaporan otomatis ini biasanya menggunakan aplikasi pihak ke 3 seperti slack,automated build,dan automated publish.

2.1.1 Maintainability

Maintenance merupakan sebuah fase dimana perangkat lunak atau apllikasi mengalami perbaikan dengan kebutuhan yang baru dan fungsionalitas yang baru. Kemampuan sistem perangkat lunak dalam menerima suatu perubahaan pada fase *maintenance* dinamakan *maintainability.* Menurut pendapat Patrick dalam bukunya practical reliability and engineering menjelaskan bahwa *maintainability* mempengaruhi tingkat *availability* secara langsung.Waktunya diambil untuk memperbaiki kerusakan dan menyelesaikan *preventive maintenance* secara rutin untuk mengambil sistem dari *available state* yang ada. Sehingga terdapat hubungan yang erat antara

reliability dengan *maintainability*,dimana yang satu mempengaruhi yang lainnya dan kedua-duanya mempengaruhi *availability* dan *cost* yang ada. Sistem dari *maintainability* itu cukup diatur dengan design dimana design tersebut menentukan features seperti aksesibilitas,kemudahan dalam tes,diagnosis kerusakan juga kebutuhan untuk kalibrasi,lubrikasi dan tindakan preventive maintenance lainnya.

Menurut Roger S. Pressman (2002: 612), maintainability adalah usaha yang diperlukan untuk mencari dan membetulkan kesalahan pada sebuah program. Sub karakteristik yang ada dalam maintainability adalah:

1. *Analyzability*

   Analyzability berhubungan dengan kemampuan menganalisis atau mendiagnosa kekurangan atau penyebab kegagalan.
2. *Changeability*

   Changeability berhubungan dengan kemampuan untuk dimodifikasi.
3. *Stability*

   Stability berhubungan dengan meminimalisir efek tak terduga dari modifikasi perangkat lunak.
4. *Testability*

   Testability berhubungan dengan kemampuan untuk dimodifikasi dan di validasi. Menurut Pressman (2002: 612), Testability adalah usaha yang diperlukan untuk menguji sebuah program untuk memastikan apakah program melakukan fungsi-fungsi yang telah ditentukan sebelumnya.
5. Compliance

   Compliance adalah kemampuan perangkat lunak dalam memenuhi standar dan kebutuhan sesuai peraturan yang berlaku. Dalam penelitian ini yang dimaksud Pengujian aspek maintainability

2.1.2 *Maintainability Index*

Mainainablity index merupakan alat ukur untuk menentukan perangkat lunak mudah atau sulit untuk dilakukan *maintenance* atau untuk memodifikasi perangkat lunak pada saat dibutuhkan kelak. Menurut Laird & Brennan dalam bukunya mengatakan bahwa *Maintainability Index* bisa digunakan untuk memantau sistem apakah sistem itu mudah di rawat atau tidak, dan *Maintainability Index* bisa memberikan indikasi apakah suatu perangkat lunak perlu dilakukan perancangan ulang. Maintainability Index merupakan kalkulasi formula berdasaran *Line Of Codes* (LOC),*Cyclomatic Complexity* dan *Halsted Volume*(HV)[3]. Persamaan *Maitainability Index* dapat dilihat pada persamaan (1) dan 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 Volume

CC = *Cyclomatic Complexity*

Loc = *Lines Of Code*

perCM = *Percent Of Line Comment*

Tabel 2. 1 Klasifikasi *Maintainability Index*

Nilai maintainability index	Klasifikasi	Keterangan
MI > 85	*Highly Maintainable*	Mudah Dimaintenance
65 < MI ≤ 85	*Moderately maintainable*	Normal Untuk Dimaintenance
MI ≤ 65	*Difficult To Maintain*	Sulit Untuk Dimaintenance

1. ***Halstead metric***

   Halstead metric merupakan alat ukur untuk mengukur kerumitan suatu program langsung dari program sumber. Pengukuran dilakukan dengan menentukan ukuran kuantitatif kompleksitas dari operator dan *operand* dalam modul sistem (Laird dan Brennan 2006). Pada *halstead metric* terdapat enam komponen diantaranya adalah :

2. ***Length of program***

   Length of program adalah kalkulasi banyaknya dari keseluruhan operator dan operand yang ada di program. Persamaan *length of program* dirumuskan pada persamaan (2).

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

   Keterangan :

   N1 = Jumlah Operator

   N2 = Jumlah Operand

3. ***Vocabulary Of Program***

   Vocabulary of program adalah jumlah keseluruhan operator dan operand unik dalam suatu program. Untuk persamaaan vocabulary of program dapat diliat pada persamaan (3).

$$N = N1 + N2 \mid (3)$$

   Keterangan :

   N = *Vocabulary Of Program*

   N1 = Jumlah Operator Unik

   N2 = Jumlah Operand Unik

4. ***Volume Of The Program***

   Volume of the program adalah volume dalam suatu program pada *halstead metric*. Untuk persamaan volume of the pgoram dapat dilihat pada persamaan (4)

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

Keterangan :

V = *Volume Of The Program*

N = Nilai kalkulasi *Length of program*

n = nilai kalkulasi Vocabulary of the program

5. *Difficulty*

Dalam *halstead metric difficulty* digunakan untuk mengetahui kesulitan dan pengembangan suatu program. Untuk persamaan *difficulty* dapat dilihat pada persamaan (5)

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

Keterangan :

D = *difficulty*

N2 = total semua operan yang muncul

n1 = jumlah operator unik

n2 = jumlah operan unik

6. *Effort*

Dalam *halstead metric effort* digunakan untuk mengetahui *resource* atau sumber daya yang digunakan untuk pengembangan suatu program.Untuk persamaan *effort* dapat dilihat pada persamaan(6).

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

Keterangan :

E = *Effort*

D = nilai *difficulty*

V = *Volume of the program*

7. *Number Of Bugs Expected In Program*

Number of bugs expected in program digunakan untuk mengetahui deteksi *bugs* pada suatu aplikasi atau perangkat lunak. Untuk persamaan *Number of bugs expected in program* dapat dilihat pada persamaan(7).

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

Keterangan :

B = *Number of bugs expected in program*

V = *Volume Of The Program*

8. *Cyclomatic Complexity*

McCabe's Cyclomatic Complexity adalah salah satu metric yang cukup terkenal yang mana metric ini menghitung *control flow* dari suatu modul[3]. 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).

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

Keterangan :

e = jumlah *edge*

n = jumlah *node*

p = jumlah predicate *node*

2.2 *Clean Code*

Clean code adalah *code* di dalam software yang formatnya benar dan disusun dengan baik sehingga *programmer* lain dapat dengan mudah membaca,merubah atau bahkan memodifikasi *code* tersebut[1].*Clean code* diperlukan karena beberapa alasan, yang pertama adalah komunikasi, dengan membuat *code* yang rapi dan benar programmer dapat menyampaikan dengan tepat maksud dari *code* atau fungsi yang dibuat.[1] Clean *code* juga dapat menghemat waktu dalam maintenance(pemeliharaan) yang terdapat pada proses terakhir pada *System Development Life Cycle(*SDLC*).*

Terdapat dua jenis kode yaitu,bad *code* dan clean *code*. Bad *Code* atau Messy *Code* dan *Clean code*,*Bad code* menurut Bjarne Stroustrup selaku penemu dan penulis bahasa pemrograman C++ menjelaskan bahwa penulisan *bad code* dapat meningkatkan kekacauan, ketika programmer lain mencoba memodifikasi suatu *bad code* , mereka cenderung membuatnya menjadi lebih buruk. Sedangkan clean *code* menurut Robert C Martin merupakan beberapa *code* yang mudah dibaca[1].

Dapat dikatakan sebuah perangkat lunak memenuhi syarat *clean code* terdapat beberapa kriteria yang harus di penuhi diantaranya adalah :

1. *Coupling* Rendah

   Code yang dituliskan dengan coupling rendah akan meminimalisir bagian tertentu sehingga tidak perlu semua bagian diperbaiki.
2. Tinggi Kohesi

   Kohesi yang tinggi membuat di dalam *class* dan komponen membuat *code* menjadi lebih simple, struktur *code* jadi lebih mudah dipahami.
3. *Class-Class* Kecil

Sebuah aplikasi dapat dikatakan clean *code* diantaranya adalah jika isi dari sebuah *class* tidak lebih dari 20 baris. Semakin sedikit isi *class* semakin baik selain karna mudah untuk membaca *class* kecil dapat mengurangi resiko bugs.

4. Hindari Duplikasi

   Jangan mengulangi *class* atau fungsi yang sama di dalam sebuah *code*.

5. Nama Yang Sesuai

   Berikan penamaan *class* atau fungsi sesuai dengan tujuannya.

6. Konsisten Mengikuti aturan yang sudah ditetapkan.

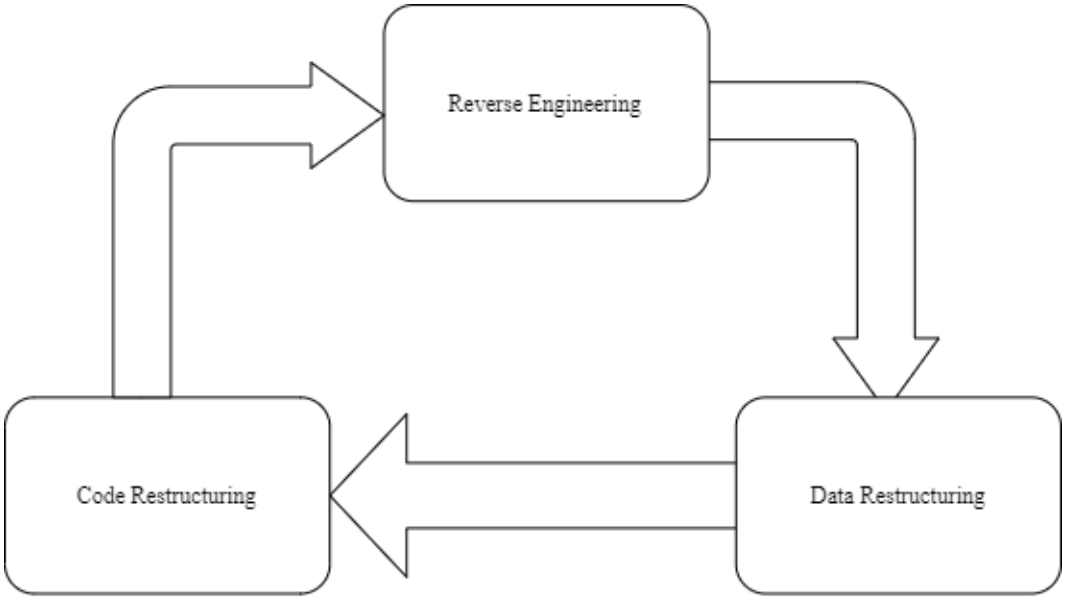
   Penamaan dalam sebuah *class* harus konsisten jika penamaan clas yang berjumlah dua kata di dipisahkan dengan (_) maka *class* yang lainnya pun harus mengikuti.

7. Tanpa Komentar

   buatlah *code* sejelas mungkin sehingga tanpa menggunakan comment fungsi dari *class* terebut sudah dapat diketahui. Tujuan lain adalah untuk menghindari kebingungan antara comment dan isi *code*. Sering terjadi saat *code* diubah namun comment yang menjelaskan *code* tersebut tidak ikut diubah sehingga menimbulkan kebingungan bagi yang membaca *code* tersebut.

2.3 *Reenginering*

Reengineering adalah proses membangun kembali sistem dimana produk yang *dihasilkan* diharapkan bertambah fungsionalitasnya, semakin baik performa & keandalannya, serta meningkatkan kemampuan maintainability-nya.Inti dari *reenginering* adalah meningkatkan kualitas dan desain *code* yang ditulis[9]. Dalam *reenginering* terdapat beberapa tahap untuk melihat tahap *reenginering* dapat dilihat pada gambar *Gambar* 2. 1 Tahap *Reenginering*



Gambar 2. 1 Tahap *Reenginering*

2.4.1 *Reverse Engineering*

Proses penemuan prinsip-prinsip teknologi dari suatu perangkat, objek, atau sistem melalui analisis strukturnya, fungsinya, dan cara kerjanya [11]. Dalam rekayasa ulang perangkat lunak proses yang dilakukan pada tahap *reverse engineering*, yakni melakukan abstraksi dari bentuk kode ke dalam bentuk konseptual yang biasanya berisi proses bisnis, variabel, dan fungsional dari aplikasi[10].

2.4.2 *Data Restructuring*

Tahap dilakukannya pembedahan data, data yang digunakan untuk dibedah dan dilakukannya restrukturisasi sehingga arsitektur, model, dan struktur data dapat jauh lebih baik dan mendukung untuk kelangsungan hidup sistem dalam jangka yang lebih panjang[10].

2.4.3 *Code Restructuring*

Tahap pembentukan ulang kode dari *legacy system* ke bentuk yang lebih baru dengan teknologi yang lebih baru, bahasa pemrograman yang lebih modern ataupun proses bisnis yang lebih baik, sehingga kode yang dibentuk mudah untuk dipahami, dilakukan pengujian, dan maintenance[10].

2.4 *Code*igniter

*Code*Igniter adalah sebuah framework yang digunakan untuk membangun sebuah situs web dengan menggunakan bahasa pemrograman php. Tujuan adanya *framework code*igniter 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. *Code*Igniter memungkinkan kita untuk focus terhadap kode dengan meminimalkan jumlah kode yang diperlukan untuk satu proses yang diberikan [1].

*Code*Igniter 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.

2.5 *Phpmetrics*

Phpmetric merupakan automated testing tool yang dapat dipergunakan untuk menghitung kualitas software yang dibuat dengan bahasa pemrograman PHP serta menampilkan hasil perhitungan tersebut dalam tabel, grafik, maupun ilustrasi.[8] Pengukuran phpmetrics memiliki pengukuran yang cukup lengkap beberapa diantaranya adalah sebagai berikut:

1. *Lack of Cohesion Method*

   menghitung jumlah dari method -method berbeda dalam suatu kelas yang menggunakan variabel dalam kelas tersebut.
2. *Cyclomatic Complexity*

   menghitung kompleksitas suatu program dengan mengukur banyaknya alur kontrol dalam suatu modul.
3. Perhitungan Operator Dan Operan

   Phpmetrics dapat menghitung keseluruhan operator dan operan pada tiap filenya sehingga dapat memudahkan untuk pengukuran halsted metric kedepannya.
4. Perhitungan *Volume of program*

   Phpmetrics dapat melakukan perhitungan volume of program pada *halstead metric*.