BAB 2
TINJAUAN PUSTAKA

2.1 Tinjauan Tempat Penelitian

2.1.1 Profil Instansi

Merupakan uraian mengenai tempat penelitian yang dilakukan dari studi kasus**.** Profil yang akan diuraikan terdiri dari sejarah, visi dan misi, dan struktur organisasi.

2.1.2 Sejarah CV. Fredavelop

CV. Fredavelop atau lebih dikenal dengan Fredavelop adalah perusahaan yang bergerak dalam bidang IT (Teknik Informatika), yang didirikan pada tahun 2011 oleh Azmi Anwarudin . Layanan Fredavelop secara umum meliputi :

1. Jasa konsultan *software* dan pembuatan *website*
2. Jasa perawatan *website*
3. *Facebook Pages*
4. Jasa *Marketing Online Aktif*

Mulai dari tahun 2012, Fredavelop telah berpengalaman menangani berbagai macam segmen klien dari seluruh Indonesia, mulai dari Instansi Pemerintah Daerah, Perusahaan BUMN dan Swasta sampai dengan Organisasi Nirlaba dan Personal.

Fredavelop berusaha memberikan solusi dan produk secara optimal, agar teknologi yang kami berikan dapat tepat guna dan bermanfaat bagi *user*. Sebagai Perusahaan *software* yang berpengalaman, Fredavelop bukan hanya berfokus pada tahap awal implementasi, layanan yang kami berikan juga mencakup pelatihan, pendampingan dan sosialisasi terkait dengan teknologi / *software* yang diterapkan. Tenaga-tenaga pelaksana kami terdiri dari tenaga-tenaga praktisi-praktisi profesional yang telah berpengalaman di bidangnya, serta didukung oleh tenaga teknis yang berasal dari mahasiswa-mahasiswa lulusan terbaik dari universitas terkemuka.

2.1.3 Visi dan Misi CV. Fredavelop

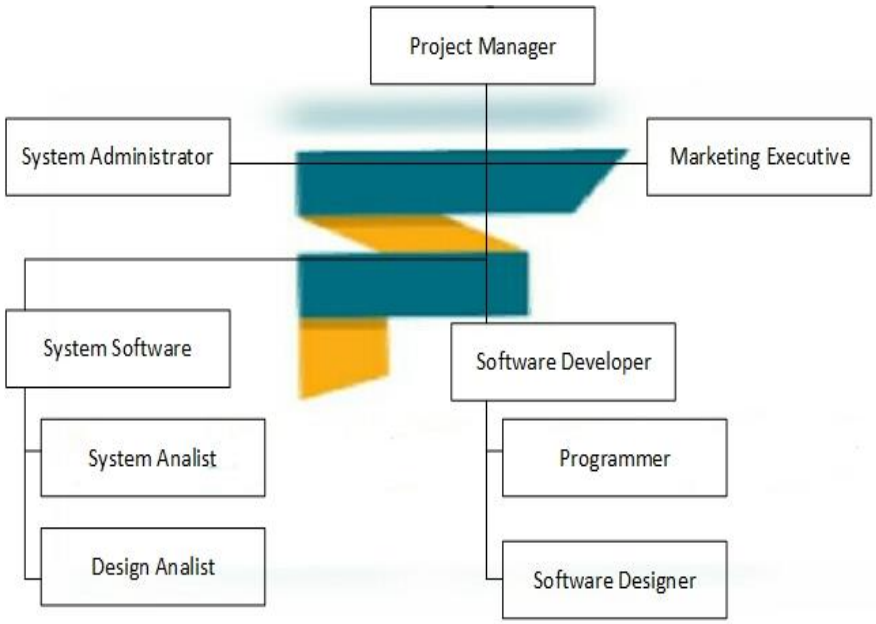
Visi dari CV. Fredavelop yaitu menjadi partner yang baik dalam mengembangkan bisnis bersama.

Misi dari CV. Fredavelop adalah:

1. Perusahaan *client* merasa beruntung memilih bekerja sama dengan kami.
2. *Client* merasa apa yang dia keluarkan serasa gratis dengan fasilitas yang kami berikan.

2.1.4 Struktur Organisasi

Berikut ini adalah struktur organisasi dari CV. Fredavelop seperti terlihat pada Gambar 2-1.



Gambar 2-1 Struktur Organisasi CV. Fredavelop

Berdasarkan gambar tugas pokok dan fungsi satuan organisasi pada CV. Fredavelop sebagai berikut :

1. *Project Manager*

mempunyai tanggung jawab dan tugas yang bermacam-macam, tidak hanya terfokus pada hal-hal yang teknis sifatnya. Bagaimana layaknya seorang *project manager* harus mempunyai kemampuan membuat tim tetap solid, mampu memonitor dan mengontrol *budget* serta mempunyai kemampuan analisis resiko yang baik.

Tanggung Jawab seorang *project Manager* terdiri dari:

1. Proyek

Proyek harus selesai sesuai dengan *budget*, sesuai dengan spesifikasi, dan waktu. Ketiga aspek itu harus dipenuhi oleh seorang *Project Manager*.

2. Organisasi

Seorang *Project Manager* juga mempunyai tanggung jawab terhadap organisasi. Proyek yang ditangani harus mempunyai *return* yang nyata terhadap organisasi. Taat kepada setiap kebijakan yang di keluarkan organisasi, harus mengambil keputusan dengan wewenang yang terbatas dari organisasi, dan juga kadang-kadang seorang *Project Manager* juga harus mengambil keputusan yang bukan yang terbaik bagi proyek tetapi terbaik buat Organisasi.

3. Tim Kerja

Seorang *Project Manager* harus memberikan *feedback* dari hasil pekerjaan proyek jika diperlukan, memberikan perhargaan terhadap anggota tim proyek yang mempunyai prestasi yang baik, dan tantangan yang paling sulit adalah menyeimbangkan antara kepentingan anggota tim, kepentingan tim, dan kepentingan proyek.

4. *System Administrator*

Bertugas melakukan administrasi terhadap sistem dan melakukan pemeliharaan sistem. "*System administrator* juga biasanya memiliki kewenangan mengatur hak akses terhadap sistem, serta hal-hal lain yang berhubungan dengan pengaturan operasional sebuah sistem,".

5. *Marketing Executive*

Mempromosikan produk atau jasa perusahaan kepada klien, *marketing executive* harus mampu mempromosikan produknya untuk memenangkan hati konsumen agar terjalin kerja sama bisnis untuk mencapai target

6. *System Software*

a. *System Analyst*

Beberapa hal penting yang dilakukan oleh seorang *system analyst* adalah sebagai berikut :

1. Bekerja dalam meneliti sebuah masalah
2. Merencanakan solusi terhadap masalah yang ada
3. Merekomendasikan *software* dan sistem yang dibutuhkan
4. Mengkoordinir pengembangan untuk memenuhi kebutuhan bisnis atau kebutuhan lainnya.

System analyst adalah penghubung antara *vendor* dan profesional teknologi informasi. Mereka juga bertanggung jawab untuk mengembangkan analisis biaya, pertimbangan desain, dan implementasi *timeline* yang telah ditetapkan.

Seorang *system analyst* memiliki beberapa kriteria yang harus dijalankan, yaitu :

1. Merencanakan aliran sistem dari bawah ke atas.
2. Berinteraksi dengan pelanggan untuk belajar dan mendokumentasikan kebutuhan yang nantinya akan digunakan untuk membuat *Bussiness Requirement Document*.
3. Menuliskan kebutuhan teknis dari fase kritis.
4. Berinteraksi dengan *designer* untuk memahami keterbatasan perangkat lunak.
5. Membantu *programmer* selama pengembangan sistem, seperti menyediakan *use case, flowchart,* atau bahkan *design database*.
6. Melakukan pengujian sistem.
7. Mendeploy sistem yang telah selesai dibangun.
8. Mendokumentasikan kebutuhan atau berkontribusi dalam pembuatan *user* manual.

9. Kapan pun proses pengembangan dilakukan, *system analyst* bertanggung jawab untuk merancang komponen dan memberikan informasi tersebut kepada *developer*.
10. *Software Develover*

b. *Programmer*

Programmer adalah seseorang yang bertugas untuk mengimplementasikan apa yang telah dirancang oleh *designer*. Seorang *programmer* akan membuat *code* yang menghasilkan aplikasi dengan *spesifikasi code* yang rapi, bersih, rapi, mudah dipahami, dan bebas dari *error*.

Tetapi seorang *programmer* tidak bertugas untuk memastikan produk yang mereka buat dapat digunakan dan diimplementasikan dengan dengan modul lainnya. *Programmer* adalah spesialis di bidang pembuatan, bukan hasil akhir ataupun perencanaan.

Seorang *programmer* memiliki kemampuan matematis dan kemampuan menulis bahasa pemrograman. Memiliki kemampuan komunikasi dengan anggota tim merupakan nilai lebih yang dimiliki oleh seorang *programmer*. Dia juga mengontrol kerja tim dan juga *life cycle* dari *software process*.

Berikut ini adalah beberapa kode etik yang kini digunakan oleh perkumpulan *programmer internasional* yang berlaku saat ini :

1. Seorang *programmer* tidak boleh membuat atau mendistribusikan *Malware*.
2. Seorang *programmer* tidak boleh menulis kode yang sulit diikuti dengan sengaja.
3. Seorang *programmer* tidak boleh menulis dokumentasi yang dengan sengaja untuk membingungkan atau tidak akurat.
4. Seorang *programmer* tidak boleh menggunakan ulang kode dengan hak cipta kecuali telah membeli atau telah meminta izin.
5. Tidak boleh mencari keuntungan tambahan dari proyek yang didanai oleh pihak kedua tanpa izin.
6. Etika profesi yang berlaku bagi *programmer* di Indonesia. Tidak boleh mencuri *software* khususnya *development tools*.

7. Tidak boleh menerima dana tambahan dari berbagai pihak eksternal dalam sebuah proyek secara bersamaan kecuali mendapatkan izin.
8. Tidak boleh menulis kode yang dengan sengaja menjatuhkan kode *programmer* lain untuk mengambil keuntungan dalam menaikkan status.
9. Tidak boleh membeberkan data-data penting karyawan dalam perusahaan.
10. Tidak boleh memberitahu masalah keuangan pada pekerja dalam pengembangan suatu proyek.
11. Tidak pernah mengambil keuntungan dari pekerjaan orang lain.
12. Tidak boleh mempermalukan profesinya.
13. Tidak boleh secara asal-asalan menyangkal adanya *bug* dalam aplikasi.
14. Tidak boleh mengenalkan *bug* yang ada di dalam *software* yang nantinya *programmer* akan mendapatkan keuntungan dalam membetulkan *bug*.
15. Terus mengikuti perkembangan ilmu komputer.

c. *Sofware Designer*

Software designer adalah seseorang yang bekerja untuk menciptakan sebuah perangkat lunak pada level yang cukup tinggi yang sesuai dengan kebutuhan pelanggan atau manajemen. Seorang *designer* tidak melakukan proses *coding* sendiri, namun bertugas untuk memastikan bahwa semua kebutuhan *software* telah ada dan diperhitungkan dengan baik sebelum proses *coding* yang sebenarnya dimulai. Mereka mengambil tujuan akhir dari pelanggan dan merencanakan berbagai tahap pengembangan dari konsep awal sampai dengan sistem selesai dibuat. Mereka menggunakan IT dalam setiap aspek pekerjaan mereka. *Pemrograman* dan perencanaan, keduanya dilakukan pada komputer, dan sinkronisasi data yang diperlukan untuk rencana dan komunikasi yang efektif membutuhkan penggunaan jaringan media yang modern.

Software design tidak hanya terlihat pada perusahaan yang memproduksi program dan perangkat lunak, tetapi juga pada perusahaan yang membutuhkan solusi pada sebuah perangkat lunak namun tidak ada perangkat lunak yang mampu mengatasinya.

2.1.5 Logo CV. Fredavelop

Berikut ini adalah logo dari CV. Fredavelop seperti terlihat pada Gambar 2-2.



Gambar 2-2 Logo Fredavelop

2.2 Landasan Teori

Landasan teori adalah teori-teori yang *relevan* dan dapat digunakan untuk menjelaskan *variabel-variabel* penelitian. Landasan teori ini juga berfungsi sebagai dasar untuk memberi jawaban sementara terhadap rumusan masalah yang diajukan, serta membantu dalam penyusunan penelitian.

2.2.1 Kualitas Perangkat Lunak

Kualitas perangkat lunak adalah keberadaan karakteristik dari satu produk yang dijabarkan dalam kebutuhannya. Kualitas perangkat lunak bisa didefinisikan sebagai kesesuaian yang diharapkan pada semua perangkat lunak yang dibangun berkaitan dengan fungsi perangkat lunak yang diutamakan dan unjuk kerja perangkat lunak [13]. Menurut badan standar internasional (IEEE, ISO) ada beberapa sudut pandang untuk menggambarkan kualitas perangkat lunak. Standar ISO menggambarkan tiga pandangan kualitas [14], yaitu :

1. Pandangan pengguna (*user*)
2. Pandangan pengembang (*developer*), dan
3. Pandangan manajer (*manager*)

Pengguna akan tertarik pada kualitas di dalam penggunaan, terutama pada karakteristik eksternal, sedangkan pengembang dan manajer lebih tertarik pada kualitas internal yang mempengaruhi pemeliharaan, *portabilitas*, keefektifan

biaya dan lain-lain. Kualitas perangkat lunak adalah tujuan yang penting bagi pengembang perangkat lunak. Kualitas adalah kunci penentu keberhasilan berkaitan dengan upaya pengembangan perangkat lunak.

2.2.1.1 Standar Kualitas Perangkat Lunak

Standar yang digunakan untuk menilai kualitas perangkat lunak adalah Standar ISO 25010. ISO 25010 adalah salah satu metode standar internasional yang digunakan untuk menilai standar kualitas sebuah perangkat lunak yang dirilis oleh ISO/IEC. ISO 25010 sebenarnya merupakan dokumen revisi dari ISO 91261:2001. Standar ISO 25010 dapat digunakan untuk mengevaluasi kualitas perangkat lunak berdasarkan dua dimensi umum, yaitu *quality in use* dan *Products quality* [5] [6] [7]. Pada *quality in use,* terdapat beberapa karakteristik yang ditinjau dari perspektif *user*, antara lain *effectiveness, efficiency*, *satisfication, freedom for risk* dan *context coverage.* Sedangkan pada dimensi *product quality*, dimana prosesnya mengacu pada karakteristik dari sebuah produk perangkat lunak, memiliki beberapa elemen yang antara lain meliputi *functional suitability*, *reliability, operability, performance efficiency*, *security, compatibility, maintainability* dan *portability* [5] [6] [7].

A. *Product Quality Model*

Product quality model mengelompokan properti kualitas produk sistem perangkat lunak ke dalam delapan karakteristik. Tiap karakteristik mempunyai sekumpulan sub-karakteristik. Penerapan model ini hanya untuk produk perangkat lunak maupun sistem. Berikut adalah karakteristik dan sub-karakteristik dari Products quality [5] [6] [7], dapat dilihat pada Tabel 2-1.

Tabel 2-1 Faktor dan Sub-faktor *Product Quality* ISO-25010

Karakteristik	Sub-karakteristik
Functional Suitability	*Functional Completeness* *Functioanl Correctness* *Functioanl Appropiateness*
Performance Efficiency	*Time Behaviour* *Resouce Utilization* *Capacity*
Compatibility	*Co-existense* *Interoperability*
Usability	*Appopriateness Recognizability* *Learnability* *Operability* *Use Error Protection* *Use Interface Aesthetics* *Accessibility*
Reliability	*Maturity* *Availability* *Fault Tolerance* *Recoverability*
Security	*Confidentiality* *Integrity* *Non-repudation* *Accountability* *Authencity*
Maintability	*Modularity* *Reusability* *Analysability* *Modifiability* *Testability*
Portability	*Adaptability* *Instability* *Replaceability*

2.2.2 Definisi *Website*

Defini *website* adalah sekumpulan halaman yang terdiri dari beberapa halaman yang berisi informasi dalam bentuk digital baik itu teks, gambar, animasi yang disediakan melalui jalur internet sehingga dapat diakses dari seluruh dunia [15]. Pada dasarnya *website* dibagi menjadi dua yaitu:

1. *Website statis*

   Merupakan *website* yang halamannya tidak berubah, biasanya untuk melakukan perubahan dilakukan secara manual dengan mengubah kode. *Website statis* ini informasinya satu arah, yakni hanya berasal dari pemilik *software* nya saja, dan hanya bisa di *update* oleh pemilik *softwarenya* saja.

2. *Website dinamis*

   Merupakan *web* halaman yang selalu diupdate, biasanya terdapat halaman *backend* (halaman administrator) yang digunakan untuk mengubah *konten*. *Web* dinamis membutuhkan *database* untuk menyimpan data. *Web* dinamis mempunyai informasi dua arah , yakni berasal dari pengguna dan pemiliknya.

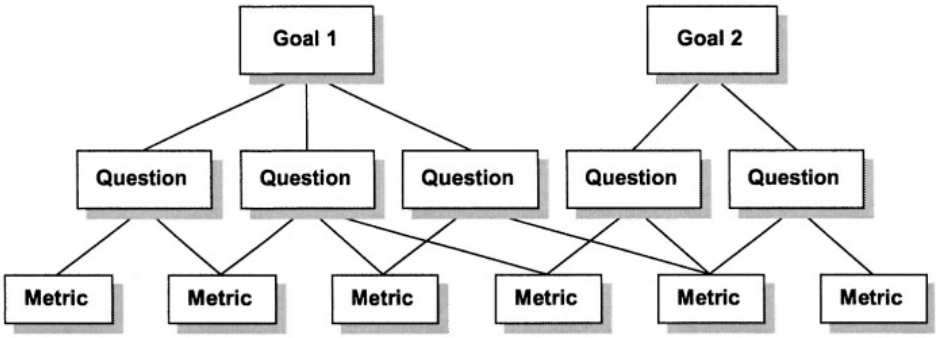
2.2.3 Metode *The Goal Question Metrics* (GQM)

The Goal Question Metric (GQM) adalah sebuah metode struktur hirarkis di mulai dengan tujuan (menetapkan tujuan pengukuran, objek yang akan diukur, masalah yang akan diukur dan ukuran dari sudut pandang mana yang akan diambil). Tujuan dari GQM adalah untuk membentuk beberapa pertanyaan, setiap pertanyaan kemudian dibentuk menjadi metrik. Beberapa metrik dapat digunakan untuk pertanyaan-pertanyaan yang berbeda dengan tujuan yang sama. Beberapa metode GQM juga dapat memiliki pertanyaan dan metrik yang sama [8].

Model pengukuran GQM terdapat 3 tingkatan, yaitu :

1. *Goal* : Tujuan yang ditetapkan untuk sebuah objek, untuk berbagai alasan, sehubungan dengan berbagai model kualitas.
2. *Question* : Seperangkat pertanyaan yang digunakan untuk menggambarkan cara penilaian atau pencapaian tujuan tertentu.
3. *Metric* : Satu set data terkait dengan setiap pertanyaan untuk menjawab dengan cara kuantitatif.

Berikut adalah struktur hirarki metode GQM seperti terlihat pada Gambar 2-3.



Gambar 2-3 Struktur Hirarki Metode GQM

Metode GQM dikembangkan dengan mengidentifikasi satu set kualitas di perusahaan, divisi atau tingkat proyek, misalnya kepuasan pelanggan, mengirimkan tepat waktu, meningkatkan kinerja. Dari tujuan-tujuan dan berdasarkan model untuk pengukuran, kita dapat memperoleh pertanyaan-pertanyaan yang menentukan tujuan-tujuan tersebut. Contohnya, untuk mengkarakterisasi sistem *software* dari masalah kualitas maka produk model kualitas harus dipilih sesuai dengan karakteristik dan langkah berikutnya menetapkan pertanyaan-pertanyaan untuk melacak tujuan kesesuaian produk dan proses.

2.2.4 Metode Pengolahan Data

Kuesioner merupakan sekumpulan pertanyaan yang diajukan kepada responden dengan tujuan dan maksud tertentu untuk memperoleh data, sehingga data tersebut dapat digunakan atau diolah kembali [16]. Untuk mengolah data dari kuesioner tersebut dapat menggunakan berbagai macam metode pengolahan data sesuai dengan rumus metrik yang digunakan, contoh metode pengolahan data, yaitu sebagai berikut :

1. Skala Likert

Skala likert berisi pertanyaan yang sistematis untuk menunjukan sikap seorang responden terhadap pertanyaan itu. Indeks ini mengasumsikan bahwa masing-masing kategori jawaban ini memiliki intensitas yang sama. Keunggulan indeks ini adalah kategorinya memiliki urutan yang jelas mulai dari "Sangat setuju", "Setuju", "Ragu-ragu","Tidak setuju", "Sangat tidak setuju" [17]. Prosedur dalam membuat skala likert adalah sebagai berikut :

a. Peneliti mengumpulkan *item-item* yang cukup banyak, relevan dengan masalah yang sedang diteliti, dan terdiri dari *item* yang cukup jelas disukai dan tidak disukai.
b. Kemudian *item-item* tersebut dicoba kepada sekelompok responden yang cukup *representatis* dari populasi yang ingin diteliti.
c. Responden di atas diminta untuk mengecek tiap *item*, apakah ia menyukai(+) atau tidak menyukainya (-). *Respon* tersebut dikumpulkan dan jawaban yang memberikan indikasi menyenangi diberikan skor tertinggi. Tidak ada masalah untuk memberikan angka 5 untuk yang tertinggi dan skor 1 untuk yang terendah atau sebaliknya. Yang penting adalah konsistensi dari arah sikap yang diperlihatkan. Demikian juga, apakah jawaban "setuju" atau"tidak setuju" disebut yang disenangi, tergantung dari isi pertanyaan dan isi dari *item-item* yang disusun.
d. Total skor dari masing-masing individu adalah penjumlahan dari skor masing-masing *item* dari individu tersebut.
e. *Respon* dianalisis untuk mengetahui *item-item* mana yang sangat nyata batasan antara skor tinggi dan skor rendah dalam skala total. Misalnya,*respon* responden pada *upper* 25% dan *lower* 25% dianalisis untuk melihat sampai berapa jauh tiap *item* dalam kelompok ini berbeda. *Item-item* yang tidak menunjukkan beda yang nyata, apakah masuk ke dalam skor tinggi atau rendah juga dibuang untuk mempertahankan konsistensi internal dari pertanyaan.

Berikut adalah bobot skala penilaian dengan skala likert dapat dilihat pada Tabel 2-2.

Tabel 2-2 Bobot Penilaian Dengan Skala Likert

Bobot Penilaian				
SS	S	BS	TS	STS
5	4	3	2	1

Contoh pengolahan kuesioner berdasarkan skala likert. Dalam hubungan teknik pengumpulan data angket, *instrument* tersebut disebarkan kepada 70 responden, kemudian direkapitulasi. Dari data 70 responden, misalnya :

Menjawab Sangat Setuju = 2 orang

Menjawab Setuju = 8 orang

Menjawab Biasa Saja = 15 orang

Menjawab Kurang Setuju = 25 orang

Menjawab Tidak Setuju = 20 orang

Menghitung skor dengan cara :

Jumlah skor untuk 2 orang menjawab SS : 2 x 5 = 10

Jumlah skor untuk 8 orang menjawab S : 8 x 4 = 32

Jumlah skor untuk 15 orang menjawab BS : 15 x 3 = 45

Jumlah skor untuk 25 orang menjawab KS : 25 x 2 = 50

Jumlah skor untuk 20 orang menjawab TS : 20 x 1 = 20

Jumlah = 157

Jumlah skor ideal untuk *item* No. 1 (skor tertinggi) = 5 x 70 = 350 (SS)

Jumlah skor rendah = 1 x 70 = 70 (TS)

2. Skala Guttman

Skala Guttman merupakan skala kumulatif. Jika seseorang menyisakan pertanyaan yang berbobot lebih berat, ia akan meng iyakan pertanyaan yang kurang berbobot lainnya. Skala Guttman disebut juga skala *scalogram* yang sangat baik untuk meyakinkan peneliti tentang kesatuan dimensi dari sikap atau sifat yang diteliti, yang sering disebut dengan *attribute universal*. Pada skala guttman terdapat beberapa pertanyaan hirarkis untuk melihat sikap tertentu seseorang. Jika seseorang menyatakan tidak terhadap pernyataan sikap tertentu dari sederetan pernyataan itu, ia akan menyatakan lebih dari tidak terhadap pernyataan berikutnya. Perbedaan skala likert dengan skala gutmann yaitu dari

Sangat Setuju (SS) sampai dengan Sangat Tidak Setuju (STS), sedangkan skala guttman hanya ada dua interval, yaitu Ya dan Tidak. Berikut adalah bobot skala penilaian dengan skala guttman seperti terlihat pada Tabel 2-3.

Tabel 2-3 Bobot Penilaian Dengan Skala Guttaman

Bobot Penelitian	
Ya	Tidak
5	1

Dalam perhitungan pengolahan data kuesioner ini setiap jawaban yang diperoleh dari responden kemudian dikalikan dengan masing-masing bobot yang telah ditentukan. Kemudian dari total seluruh bobot jawaban yang telah dihitung dibagi dengan total keseluruhan responden yang menjawab untuk mendapatkan nilai rata-rata. Berdasarkan nilai dari rata-rata tersebut maka diperoleh jawaban untuk ditarik sebuah kesimpulan. Berikut ini merupakan rumus dari perhitungan yang dapat digunakan untuk mengolah data kuesioner [18].

$$\bar{x} = \frac{\sum Total}{N}$$

$\bar{x}$ = rata – rata

$\sum Total$ = Jumlah total seluruh nilai setelah dikaitkan dengan bobot

n = total *responden*

2.2.5 Metode *Chidamber and Kemerer*

Metrik dari Chidamber dan Khemerer untuk mengukur *coding*/ program dari aplikasi akta notaris fidusia yang berkonsep OOP.

Menurut Chidamber dan Kemerer Metric untuk mengukur dan memberikan penilaian perangkat lunak berbasis *objek oriented* dapat menggunakan enam buah metriks [11] [19], yaitu :

1. *Weight Methods per Class* (WMC)

Pengukuran metrik WMC adalah menghitung jumlah *method* yang diimplementasikan dalam suatu *class*. Untuk mengetahuinya maka dihitung seluruh *method* lokal pada sebuah perangkat lunak

Rumus : m1+m2+m3+.....+Mh

2. *Depth of Inheritance Tree* (DIT)

DIT merupakan penghitungan jalur *inhereitance* maksimum dari sebuah *class*. Kedalaman sebuah *class* dalam hirarki pewarisan diukur oleh DIT.

Rumus : Apabila ada pewarisan diberikan nilai 1 dan apabila tidak ada diberikan nilai 0.

3. *Number of Children* (NOC)

Pengukuran NOC adalah menghitung jumlah *children* atau anak kelas

Rumus : Apabila ada anak diberi nilai dan apabila tidak tidak diberi nilai.

4. *Coupling Between Objects* (CBO)

Coupling adalah ketergantungan modul satu dengan modul lainnya, *coupling* yang baik adalah *coupling* yang rendah di mana modul satu dengan modul lainnya tidak saling ketergantungan. Suatu fungsi yang berpasangan dihitung sebagai *Coupling*. Semakin kecil jumlah CBO maka semakin baik kualitas *class* tersebut.

Rumus : Menghitung jumlah *method* lokal yang mengakses ke *method eksternal*

5. *Respon for a Class* (RFC)

RFC adalah *method* yang diimplementasikan dalam *class* ditambahkan *class* yang diakses objek dari jenis *class* tersebut. Semakin banyak *method* yang dapat dipanggil dalam sebuah *class*, semakin besar kompleksitasnya *class* tersebut. Pengukuran RFC yaitu menghitung jumlah *method* lokal yang diimplementasikan ditambah *method* yang dipanggil dalam objek.

Rumus : jumlah *method* lokal + jumlah *method eksternal*.

6. *Lack OF Cohesion in Method* (LCOM)

LCOM adalah metrik untuk menghitung sebuah *Cohesi*. *Cohesi* adalah fungsi yang terkait dalam sebuah modul. *Cohesi* yang baik adalah *cohesi* yang tinggi, semakin tinggi *cohesi* semakin baik perancangannya. *Cohesi* yang baik di

mana sebuah modul hanya bertanggung jawab terhadap satu pekerjaan saja. *Cohesion* sebuah *class* adalah karakteristik seberapa dekat *method-method* lokal terhubung dengan variabel lokal dalam sebuah *class*. Pengukuran LCOM dilakukan dengan menghitung jumlah koneksi pada *method* [19] [20] [21].

Rumus Perhitungannya adalah:

LCOM*= (m – sum (Mu) / a) / (m-1).

Di mana:

m adalah jumlah *method* dalam *class*.

a adalah jumlah *variable* dalam *class*.

mu adalah jumlah *variable* yang dipanggil.

2.2.6 Metode Pembobotan

Rank Order Centroid (ROC) didasarkan pada tingkat kepentingan atau *prioritas* dari kriteria. Menurut Jeffreys dan Cockfield dalam Afiefah Rahma, teknik ROC memberikan bobot pada setiap kriteria sesuai dengan *ranking* yang dinilai berdasarkan tingkat prioritas. Biasanya dibentuk dengan pernyataan "Kriteria 1 lebih penting dari kriteria 2, yang lebih penting dari kriteria 3" dan seterusnya hingga kriteria ke n, ditulis . Untuk menentukan bobotnya, diberikan aturan yang sama yaitu di mana merupakan bobot untuk kriteria [9].

Atau dapat dijelaskan sebagai berikut:

Jika Cr1 ≥ Cr2 ≥ Cr3 ≥ ... ≥ Crn

Maka W1 ≥ W2 ≥ W3 ≥ ... ≥ Wn

Selanjutnya, jika k merupakan banyaknya kriteria, maka

$$W1 = \frac{1 + \frac{1}{2} + \frac{1}{3} + \cdots .... + 1/k}{k}$$

$$W2 = \frac{0 + \frac{1}{2} + \frac{1}{3} + \cdots .... + 1/k}{k}$$

$$W3 = \frac{0 + 0 + \frac{1}{3} + \cdots .... + 1/k}{k}$$

Secara umum pembobotan ROC dapat dirumuskan sebagai berikut :

$$Wk = 1/k \sum_{t=1}^{k} (1/\mathrm{i})$$

w = Nilai Pembobotan kriteria

k = Jumlah kriteria

i = Nilai alternatif

2.2.7 Metode Penilaian

Weigth Summation adalah sebuah metode kompensasi, yang berarti dapat menghitung skor kriteria skor yang "buruk", sehingga dapat dikopensasikan dengan kriteria skor yang "baik". Berikut adalah persamaan yang digunakan dalam metode *weighted Summation* [10].

Jika

$$score\ (Ci) = \sum_{i=1}^{n} Wi.Xi\ (Ci)$$

Maka

$$score\ (Ci) = \sum_{i=1}^{n} Wi.Xi\ (Ci)$$

Di mana :

Sore (Ci) = Nilai kriteria Ci ($0 \leq$ *score* Ci ≥ 1)

Wi = Bobot untuk Kriteria ke-i

Xi = Nilai Perhitungan metrik kriteria ke-i

2.2.8 *User Interface* (UI)

User interface (UI) merupakan bagian dari sistem komputer yang menghubungkan antara sistem dan pengguna. UI dikatakan baik apabila desain UI dapat mendorong interaksi yang mudah, alami dan menarik di antara *user* dan sistem. UI merupakan bagian dari komputer dan perangkat lunak di mana orang dapat melihat, mendengar, menyentuh, berkomunikasi dengannya. UI pada dasarnya memahami sesuatu sepeti menu, *windows,* papan ketik, tetikus dan suara yang berasal dari komputer dan pada umumnya, semua kanal informasi yang membantu *user* berinteraksi dengan komputer. UI sangat berperan penting

dalam dunia komputer karena dengan adanya *user interface* maka kemudahan pengguna dalam mengoperasikan perangkat komputer menjadi lebih gampang[30].

2.2.9 User Experience (UX)

UX (*User Experience*) adalah sebuah bidang baru yang berangkat dari hubungan manusia dengan teknologi. Awalnya, hubungan antara manusia dan komputer dipelajari melalui bidang HCI (*Human Computer Interaction*). Ahli HCI mempelajari kemampuan komputer (pada umumnya: produk teknologi) dalam berinteraksi dengan manusia, sedangkan unsur kemampuan manusia dipelajari oleh para ahli bidang *Psikologi Kognitif*, maupun para ahli bidang *Human Factors*, yang berangkat dari bidang Teknik Industri.

Menurut Jakob Nielsen, *user experience* mencakup seluruh aspek interaksi terhadap pengguna dengan perusahaan, layanan, dan produk-produknya. Berdasarkan kedua pernyataan di atas dapat disimpulkan bahwa *user experience* adalah hal-hal yang dirasakan seseorang terhadap penggunaan produk (situs *website*) yang menilai tentang tingkat kemudahan dan kenyamanan terhadap fungsionalitas yang disajikan oleh sebuah *website*[30].

2.2.10 OOP (*Object Oriented Programming*)

Secara spesifik, pengertian berorientasi objek berarti bahwa mengorganisasi perangkat lunak sebagai kumpulan dari objek tertentu yang memiliki struktur data dan perilakunya. Hal ini yang membedakan dengan pemograman *konvensional* di mana struktur data dan perilaku hanya berhubungan secara terpisah. Terdapat beberapa cara untuk menentukan karakteristik dalam pendekatan berorientasi objek, tetapi secara umum mencakup empat hal, yaitu identifikasi, klasifikasi, *polymorphism (polimorfisme)* dan *inheritance* (pewarisan) [22]

2.2.10.1 Desain Berorientasi Objek

Desain berorientasi objek atau *Object Oriented Design* (OOD) adalah tahapan perantara untuk spesifikasi atau kebutuhan sistem yang akan dibangun dengan konsep berorientasi objek ke desain pemodelan agar lebih mudah diimplementasikan dengan pemograman berorientasikan objek. Pemodelan berorientasi objek biasanya dituangkan dalam dokumentasi perangkat lunak dengan menggunakan perangkat lunak dengan menggunakan perangkat pemodelan berorientasi objek, di antaranya adalah UML (*Unified Modeling Language*). Kendala dan permasalahan pembangunan sistem berorientasi objek biasanya dapat dikenali dalam tahap ini [22] [23].

2.2.10.2 Pemodelan

Pemodelan adalah gambaran dari realita yang sederhana dan dituangkan dalam bentuk pemetaan dengan aturan tertentu. Pemodelan dapat menggunakan bentuk yang sama dengan realitas. Pemodelan juga banyak digunakan untuk merencanakan suatu hal agar kegagalan dan resiko yang mungkin terjadi dapat meminimalisir. Pada dunia pembangunan perangkat lunak juga diperlukan pemodelan. Pemodelan perangkat lunak digunakan untuk mempermudah langkah berikutnya dari pengembangan sebuah sistem sehingga lebih terencana. Pemodelan pada pembangunan perangkat lunak untuk menvisualkan perangkat lunak yang akan dibuat [22] [23].

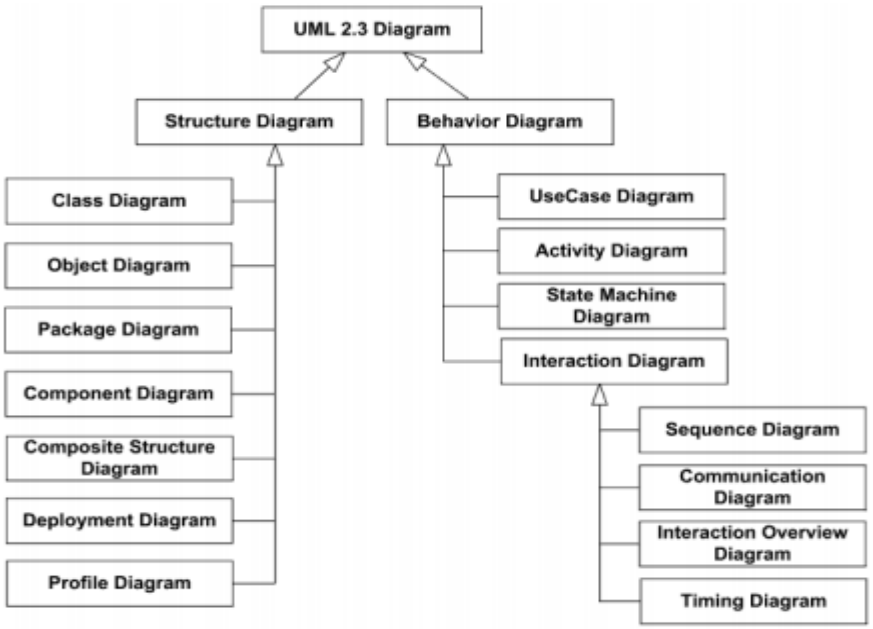
2.2.11 UML (*Unified Markup Language*)

UML *(Unified Modelling Language)* adalah sebuah “bahasa” yang telah menjadi standar dalam industri visualisasi, merancang, dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. UML dapat digunakan untuk membuat model untuk semua jenis perangkat lunak, di mana perangkat lunak tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa berorientasi objek. Seperti bahasa-bahasa lainnya, UML mendefinisikan

notasi dan *syntax*. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya yaitu : Grady Booch OOD (*Object Oriented Design*), Jim Rumbaugh OMT (*Object Modeling Technique*) dan Ivar Jacobson OOSE (*Object Oriented Software Engineering*) [24] [25].

2.2.11.1 Diagram UML

Pada UML 2.3 terdiri dari 13 macam diagram yang dikelompokan dalam 3 kategori. Pembagian kategori dan macam-macam diagram tersebut dapat dilihat pada Gambar 2-4.



Gambar 2-4 Diagram UML

Berikut ini penjelasan singkat dari pembagian kategori tersebut [24] [25].

1. *Structure diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan.
2. *Behavior diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada sebuah sistem.

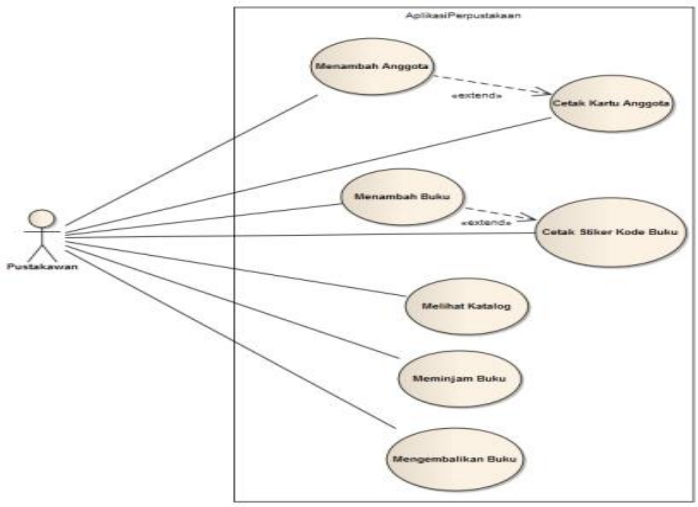
3. *Interaction diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antar sub sistem pada suatu sistem.

2.2.11.2 *Use Case* Diagram

Diagram use case merupakan pemodelan untuk kelakuan (*behaviour*) sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu. Syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu pendefinisian apa yang disebut aktor dan *use case* [24] [25].

1. Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat diluar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.
2. *Use case* merupakan *fungsionalitas* yang disediakan sistem sebagai *unit-unit* yang saling bertukar pesan antar *unit* atau aktor

Pada Gambar 2-5 berikut ini merupakan contoh *use case* diagram dalam pembuatan sistem informasi.



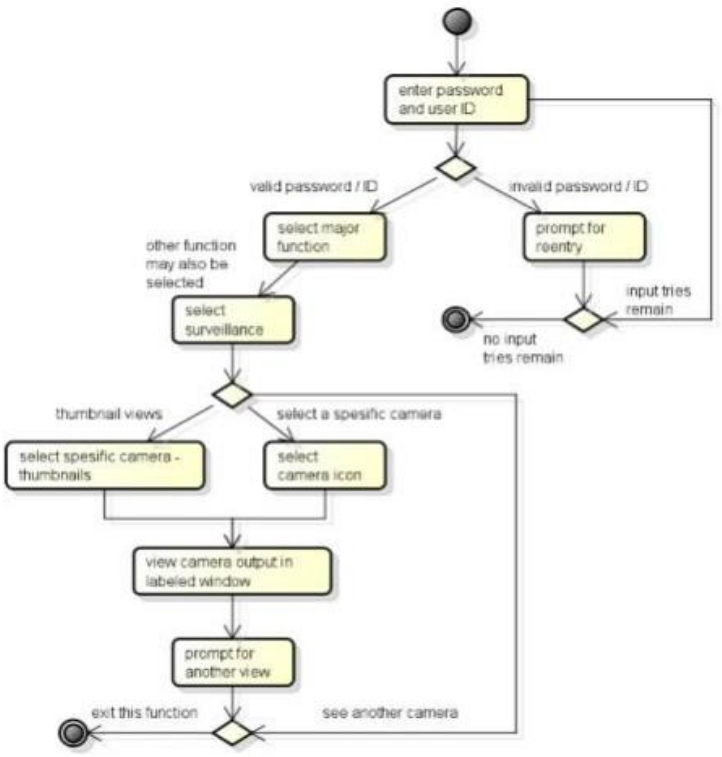
Gambar 2-5 *Use Case* Diagram

2.2.11.3 *Activity* Diagram

Diagram aktivitas menggambarkan *workflow* (aliran kerja) atau *aktivitas* dari sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak. Yang perlu diperhatikan di sini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem [24] [25]. *Diagram aktivitas* juga banyak digunakan untuk mendefinisikan hal-hal berikut:

1. Rancangan proses bisnis di mana setiap urutan *aktivitas* yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
2. Urutan atau pengelompokkan tampilan dari sistem/*user interface* di mana setiap *aktivitas* dianggap memiliki sebuah rancangan antarmuka tampilan.
3. Rancangan pengujian di mana setiap *aktivitas* dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.

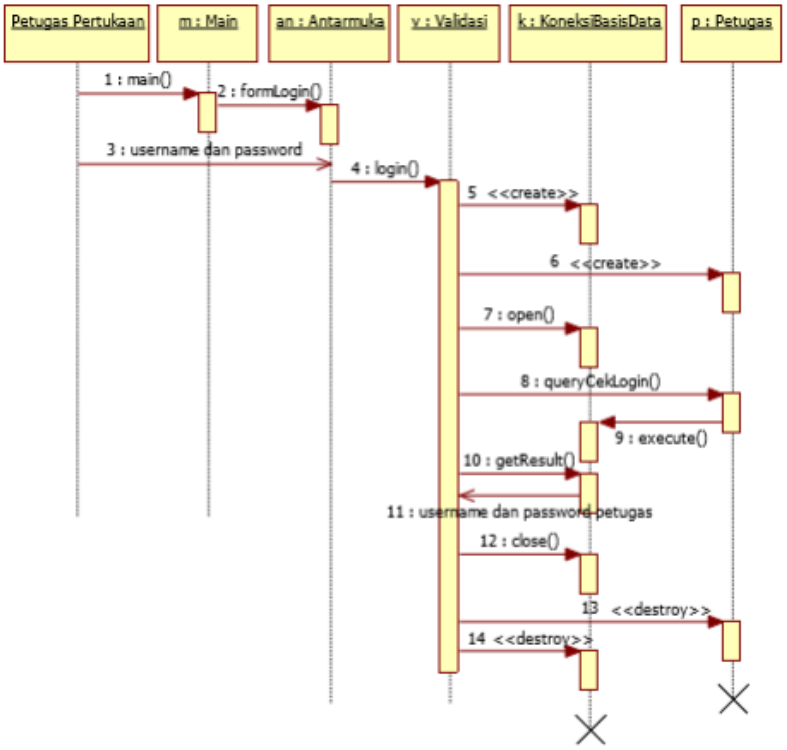
Rancangan menu yang ditampilkan pada perangkat lunak. Contoh *Activity Diagram* dapat dilihat pada Gambar 2-6.



Gambar 2-6 Contoh *Activity* Diagram

2.2.11.4 *Sequence* Diagram

Diagram sequence menggambarkan kelakuan pada objek *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek [24] [25]. Oleh karena itu untuk menggambar *diagram sequence* maka harus diketahui objek-objek yang terlibat dalam sebuah use case beserta metode-metode yang dimiliki kelas yang *diinstansiasi* menjadi objek itu. Membuat *diagram sequence* juga dibutuhkan untuk melihat *skenario* yang ada pada *use case*. Banyaknya *diagram sequence* yang harus digambar adalah minimal sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram *sequence* sehingga semakin banyak *use case* yang didefinisikan maka diagram *sequence* yang harus dibuat juga semakin banyak. Contoh *Sequence* Diagram dapat dilihat pada Gambar 2-7.



Gambar 2-7 *Sequence* Diagram

2.2.11.5 *Class* Diagram

Diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi. Atribut merupakan *variabel-variabel* yang dimiliki oleh suatu kelas. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas. Contoh *Class Diagram* dapat dilihat pada Gambar 2-8.



Gambar 2-8 *Class* Diagram

Syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu pendefinisian apa yang disebut aktor dan *use case* [24] [25].

1. Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem yang akan dibuat di luar sistem yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.
2. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor.