BAB III

METODELOGI PENELITIAN

3.1 Metodologi Penelitian

Metodologi penelitian adalah langkah dan prosedur yang akan dilakukan dalam pengumpulan data atau informasi guna memecahkan permasalahan dan menguji hipotesis penelitian.

Metode yang digunakan dalam melaksanakan penelitian merupakan dasar penyusunan rancangan penelitian dan merupakan penjabaran dari metode ilmiah secara umum.

3.1.1 Desain Penelitian

Metode penelitian yang digunakan oleh penulis adalah metode penelitian deskriptif dan komparatif. Menurut Erlina (2008:20), penelitian deskriptif adalah penelitian terhadap fenomena atau populasi tertentu yang diperoleh oleh peneliti dari subjek berupa individu, organisasional, industri atau perspektif yang lain. Penelitian deskriptif dilakukan untuk menjawab pertanyaan tentang: apa, siapa, kapan, di mana, dan bagaimana yang berkaitan dengan karakteristik populasi atau fenomena tersebut. Penelitian juga dilakukan dengan menggunakan metode komparatif yang dilakukan dengan membandingkan teori yang ada dengan praktik yang ditemui dan menarik kesimpulan.

Untuk memudahkan dalam melakukan penelitian yang berhubungan dengan masalah yang diteliti, maka penulis melakukan pendekatan studi kasus. Dengan menggunakan pendekatan ini, data yang dikumpulkan dapat disesuaikan

dengan keadaan yang sebenarnya dan dibandingkan dengan teori yang menunjang. Dengan demikian, dapat memberikan gambaran yang cukup jelas serta dapat menarik kesimpulan dari objek yang diteliti.

3.1.2 Jenis dan Metode Pengumpulan Data

Jenis dan Metode pengumpulan data digunakan penulis untuk mendapatkan data sebagai bahan kajian dalam penulisan skripsi dengan tujuan membuat suatu perancangan Aplikasi Berat Badan Ideal berbasis Android . Dalam hal ini penulis menggunakan metode pengumpulan data berupa sumber data primer (observasi, wawancara dan pengamatan system) dan sumber data sekunder (dokumentasi).

3.1.2.1 Jenis Pengumpulan Data

1. **Sumber Data Primer**
   a. Teknik observasi yaitu dengan melakukan pengamatan secara langsung.
   b. Teknik wawancara yaitu dengan cara melakukan tanya jawab secara langsung kepada pihak-pihak yang berkompeten.
   c. Teknik kepustakaan yaitu dengan mengumpulkan data dari buku atau bahan tulisan yang ada relevansinya dengan skripsi ini.
2. **Data sekunder**

Merupakan data yang telah dikumpulkan lembaga pengumpul data dan dipublikasikan kepada masyarakat pengguna data (Erlina, 2008:24). Data sekunder dalam penelitian ini adalah data – data berat badan pengguna aplikasi.

3.1.2.2 Metode Pengumpulan Data

Pengumpulan data primer membutuhkan perancangan alat dan metode pengumpulan data

Metode pengumpulan data penelitian:

- Observasi
- Wawancara

Semua butir (item) yang ditanyakan dalam semua metode pengumpulan data haruslah sejalan dengan rumusan masalah dan/atau hipotesis penelitian Karenanya diperlukan proses Dekomposisi variabel penelitian menjadi sub-variabel, dimensi dan butir penelitian merupakan pekerjaan yang harus dilakukan dengan hati-hati. Proses dekomposisi ini juga memudahkan proses pengukuran dan pengumpulan data. Proses dekomposisi ini dikenal sebagai proses operasionalisasi variabel penelitian

3.1.3 Metode Pendekatan dan pengembangan sistem

Dalam pembangunan suatu aplikasi diperlukan suatu pendekatan dan pengembangan system yang akan menentukan proses penyelesaian rekayasa perangkat lunak, adapun pendekatan system yang digunakan adalah dengan menggunakan pendekatan berorientasi objek dan pengembangan system dengan menggunakan model *prototype* paradigma.

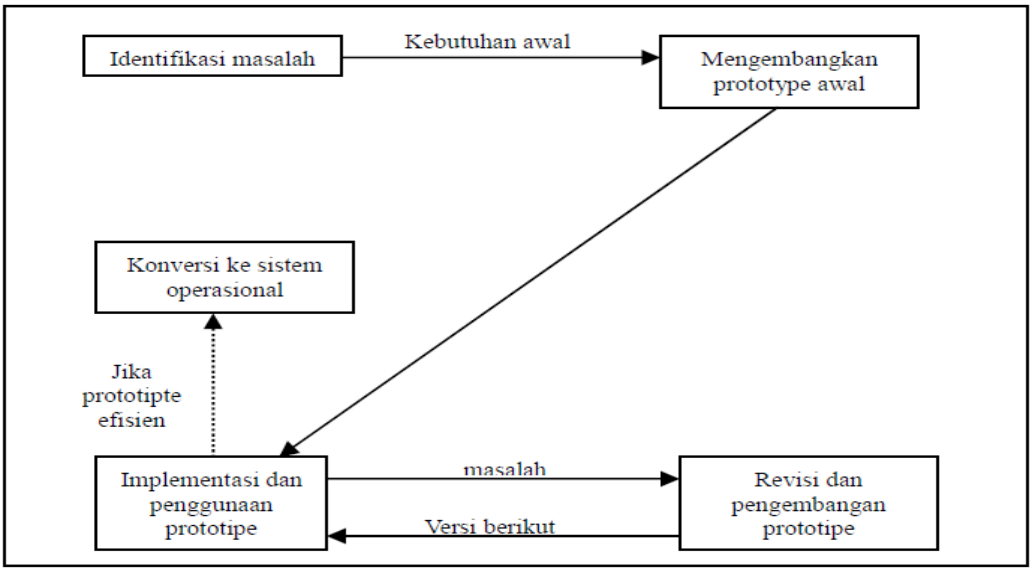
3.1.3.1 Metode Pendekatan Sistem

Metode pendekatan sistem yang di gunakan adalah pendekatan dengan *Object Oriented* yang menggunakan AOO(*Analisis Object Oriented*) dan DOO yang di visualisasikan dengan UML dan di antara nya adalah sebagai berikut : *Use Case Diagram, Sequence Diagram, Class Diagram, Collaboration Diagram, Component Diagram dan Deployment Diagram*.

3.1.3.2 Metode Pengembangan Sistem

Metode pengembangan sistem yang digunakan adalah dengan menggunakan metode pendekatan prototype. Prototyping merupakan salah satu metode pengembangan perangat lunak yang banyak digunakan. *Prototype* dibangun untuk mendefinisikan kebutuhan.

Prototype akan dihilangkan sebagian atau seluruhnya dan perangkat lunak aktual direkayasa dengan kualitas dan implementasi yang sudah ditentukan . Metode prototype di rancang agar dapat menerima perubahan-perubahan dalam rangka menyempurnakan *prototype* yang di buat dan perubahan-perubahan yang terjadi di anggap dapat merupakan sebagian dari proses pengembangan itu sendiri.



Gambar 3.2 Metode Prototype

(Sumber : Adi Nugroho ,ST.,MMSI , Konsep Pengembangan Sistem Basis Data , 2004, Informatika : Bandung)

Berikut adalah langkah-langkah dalam merancang sistem yang penulis gunakan dalam mekanisme pengembangan sistem dengan prototype , langkah-langkah tersebut adalah :

1. Mengidentifikasikan kebutuhan pemakai

   Analisis sistem mewarnai pemakai untuk mendapatkan gagasan dan apa yang diinginkan pemakai terhadap sistem.

2. Mengembangkan prototype

   Analisis sistem mungkin bekerja sama dengan spesialis informasi lainya, menggunakan satu atau lebih peralatan prototype untuk mengembangkan sebuah prototype

3. Memutuskan apakah prototype dapat diterima analisis mendidik pemakai untuk menggunakan prototype dan memberikan kesempatan kepada mereka untuk membiasakan diri dengan sistem, pemakai akan memberi saran dan masukan kepada analis apakah prototype memuaskan, jika ya langkah 4 diambil, jika tidak prototype direvisi dengan mengulangi langkah 1, 2, 3.

4. Menggunakan prototype

   Prototype ini menjadi sistem operasional pendekatan ini hanya mungkin jika peralatan prototyping memuat semua elemen penting dari sistem baru.

3.1.3.3 Alat Bantu Analisis Dan Perancangan.

a. Use Case Diagram.

Use Case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah usecase merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use Case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-create sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

Use Case diagram dapat sangat membantu bila kita sedang menyusun requirement sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang test case untuk semua feature yang ada pada sistem. Sebuah *Use Case* dapat meng-include fungsionalitas *Use Case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *Use Case* yang di-include akan dipanggil setiap kali *Use Case* yang meng-include dieksekusi secara normal. Sebuah *Use Case* dapat di-include oleh lebih dari satu *Use Case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang common. Sebuah *Use Case* juga dapat meng-extend *Use Case* lain dengan behaviour-nya sendiri. Sementara hubungan generalisasi antar *Use Case* menunjukkan bahwa *Use Case* yang satu merupakan spesialisasi dari yang lain.

b. Activ*ity Diagram*

*Activity diagrams m*erupakan diagram yang menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

c. Class Diagram

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. Class menggambarkan keadaan (atribut/properti) suatu sistem,

sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class diagram* menggambarkan struktur dan deskripsi class, package dan objek beserta belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

Use Case diagram dapat sangat membantu bila kita sedang menyusun requirement sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang test case untuk semua feature yang ada pada sistem. Sebuah *Use Case* dapat meng-include fungsionalitas *Use Case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *Use Case* yang di-include akan dipanggil setiap kali *Use Case* yang meng-include dieksekusi secara normal. Sebuah *Use Case* dapat di-include oleh lebih dari satu *Use Case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang common. Sebuah *Use Case* juga dapat meng-extend *Use Case* lain dengan behaviour-nya sendiri. Sementara hubungan generalisasi antar *Use Case* menunjukkan bahwa *Use Case* yang satu merupakan spesialisasi dari yang lain.

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. Class menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). Class diagram menggambarkan struktur dan deskripsi class, package dan objek beserta Hubungan Antar Class

- Asosiasi, yaitu hubungan statis antar class. Umumnya menggambarkan class yang memiliki atribut berupa class lain, atau class yang harus mengetahui eksistensi class lain. Panah navigability menunjukkan arah query antar class.
- Agregasi, yaitu hubungan yang menyatakan bagian.
- Pewarisan, yaitu hubungan hirarkis antar class. Class dapat diturunkan dari class lain dan mewarisi semua atribut dan metoda class asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari class yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
- Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-passing dari satu class kepada class lain. Hubungan dinamis dapat digambarkan dengan menggunakan *Sequence Diagram* yang akan dijelaskan kemudian.

d. *Sequence Diagram*

Sequence Diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, display, dan sebagainya) berupa message yang digambarkan terhadap waktu. *Sequence Diagram* terdiri atas dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence Diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah event untuk menghasilkan output tertentu. Diawali dari apa yang men- trigger aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan output apa yang dihasilkan. Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal.

Message digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, message akan dipetakan menjadi operasi/metoda dari class. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*.Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan icon khusus untuk *objek boundary*, *controller* dan *persistent entity*.

e. Collaboration Diagram

Collaboration diagram juga menggambarkan interaksi antar objek seperti *Sequence Diagram*, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian message.

Setiap *message* memiliki sequence number, di mana *message* dari level tertinggi memiliki nomor 1. *Messages* dari level yang sama memiliki prefiks yang sama.

f. Component Diagram

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency)* di antaranya. Komponen piranti lunak adalah modul berisi code, baik berisi *source code* maupun *binary code*, baik library maupun *executable*, baik yang muncul pada *compile time*, *link time*, maupun *run time*. Umumnya komponen terbentuk dari beberapa class dan/atau *package*, tapi dapat juga dari komponen-komponen yang

lebih kecil. Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.

g. Deployment Diagram

Deployment/physical diagram menggambarkan detail bagaimana komponen di-deploy dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, server atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisikal . Sebuah node adalah server, workstation, atau piranti keras lain yang digunakan untuk men-deploy komponen dalam lingkungan sebenarnya. Hubungan antar node (misalnya TCP/IP) dan requirement dapat juga didefinisikan dalam diagram ini.

3.1.4 Pengujian *software*

Pengujian dilakukan untuk mendapatkan suatu perangkat unak yang ‘layak‘ untuk digunakan. Suatu perangkat lunak yang telah selesai diujikan harus memiliki standard kualitas tertentu.

Black-box testing adalah metode pengujian perangkat lunak yang tes fungsionalitas dari aplikasi yang bertentangan dengan struktur internal atau kerja (lihat pengujian white-box). pengetahuan khusus dari kode aplikasi / struktur internal dan pengetahuan pemrograman pada umumnya tidak diperlukan. Uji kasus dibangun di sekitar spesifikasi dan persyaratan, yakni, aplikasi apa yang seharusnya dilakukan.Menggunakan deskripsi eksternal perangkat lunak, termasuk spesifikasi, persyaratan, dan desain untuk menurunkan uji kasus. Tes ini

dapat menjadi fungsional atau non-fungsional, meskipun biasanya fungsional. Perancang uji memilih input yang valid dan tidak valid dan menentukan output yang benar.Tidak ada pengetahuan tentang struktur internal benda uji itu.

Metode uji dapat diterapkan pada semua tingkat pengujian perangkat lunak: unit, integrasi, fungsional, sistem dan penerimaan. Ini biasanya terdiri dari kebanyakan jika tidak semua pengujian pada tingkat yang lebih tinggi, tetapi juga bisa mendominasi unit testing juga.

Adapun factor-faktor pengujian blackbox adalah :

1. File integrity

   Menekankan pada data yang dimasukkan melalui aplikasi akan tidak bias diubah.

   Prosedur yang akan memastikan bahwa file yang digunakan benar dan data dalam file tersebut akan disimpan sekuensial dan benar.

2. Service levels

   Menekankan bahwa hasil yang diinginkan did apat dalam waktu yang diinginkan oleh *user*. Untuk mencapai keinginan tersebut, harus dilakukan penyesuaian antara keinginan *user* dengan sumber daya yang ada.

3. Ease of use

   Menekankan perluasan usaha yang diminta untuk belajar, mengoprasikan dan menyiapakan inputan, dan menginterpretasikan output

dari sistem. Faktor ini tersangkut dengan *usability* system terhadap interaksi antara manusia dan system.

4. Authorization

Menjamin data diproses sesuai dengan ketentuan manajemen. *Authorization* menyangkut proses transaksi secara umum dan khusus. Fokus Pengujian *Black box testing* yaitu sebagai berikut :

a) Menguji fungsi-fungsi khusus dari aplikasi.

b) *Test input* dan *output* untuk fungsi yang ada tanpa memperhatikan prosesnya.

Beberapa jenis kesalahan yang dapat di identifikasi :

1) Fungsi tidak benar atau hilang,

2) Kesalahan antar muka,

3) Kesalahan pada struktur data (pengaksesan basis data),

4) Kesalahan inisialisasi dan akhir program, dan

5) Kesalahan