BAB III

OBJEK DAN METODE PENELITIAN

3.1. Objek Penelitian

Objek penlitian dilakukan pada tempat-tempat kuliner yang ada di Kota Bandung. Penentuan tempat kuliner yang menjadi objek penelitian adalah tempat kuliner yang sangat banyak diminati oleh pengunjung kota Bandung, yang sesuai dengan kategori yang dibutuhkan.

3.2. Metode Penelitian

Metode penelitian adalah suatu teknik atau cara mencari, memperoleh, mengumpulkan atau mencatat data yang diperlukan untuk menyusun suatu karya ilmiah yang kemudian menganalisis faktor – faktor yang berhubungan dengan pokok permasalahan yang dihadapi sehingga didapat suatu kebenaran data yang di peroleh.

3.2.1. Desain Penelitian

Dalam melakukan penelitian, desain penelitian merupakan kerangka atau perincian prosedur kerja yang akan dilakukan pada waktu meneliti, sehingga diharapkan dapat memberikan gambaran dan arah mana yang akan dilakukan dalam melaksanakan penelitian tersebut, desain penelitian yang baik dapat memudahkan kita dalam melakukan penelitian dan sesuai dengan tujuan yang telah ditetapkan sebelumnya.

Sesuai dengan tujuan yang ingin dicapai dalam penelitian ini, penulis mengumpulkan data sebagai bahan laporan dengan menggunakan metode deskriptif, yaitu suatu metode yang pada tahap pertama penulis melakukan dengan cara mengumpulkan data dan bahan yang diperlukan terlebih dahulu dan pada tahap berikutnya, penulis mengolah dan membahas sampai pada suatu kesimpulan yang pada akhirnya dapat dibuat suatu laporan.

3.2.2 Jenis dan Metode Pengumpulan Data

Dalam pengambilan data, penulis menggunakan beberapa teknik dan metode. Adapun metode pengumpulan data yang digunakan adalah sebagai berikut :

3.2.2.1 Sumber Data Primer

Data primer merupakan pengumpulan data secara langsung dari objek yang sedang diteliti, cara yang digunakan untuk mengumpulkan data primer adalah sebagai berikut :

1. Observasi, yaitu mengumpulkan bahan laporan dengan melihat dan langsung mengamati setiap pembelian pulsa.
2. Wawancara, yaitu pengumpulan data dengan mengajukan pertanyaan-pertanyaan pada pihak yang terkait yaitu penjual pulsa.

3.2.2.2 Sumber Data Sekunder

Dalam penelitian ini data sekunder berupa informasi mengenai dokumen-dokumen yang berkaitan dengan permasalahan yang dibahas yaitu:

1) Studi Pustaka

Metode yang digunakan dengan cara mencari referensi buku-buku yang dapat diperlukan sebagai landasan teori maupun yang berhubungan dengan masalah pembuatan program aplikasi. Adapun buku referensi yang penulis gunakan untuk penyusunan landasan teori adalah buku-buku yang berasal dari perpustakaan, sebagai contoh adalah buku karangan Jogiyanto. Abdul Kadir.

2) Media internet

Metode ini digunakan untuk mendapatkan gambaran tentang bentuk dan penyajian program, serta mencari landasan teori serta mencari tambahan artikel-artikel yang diperlukan. Adapun alamat website yang penulis gunakan adalah Google dan Wikipedia.

3.2.3 Metode Pendekatan dan Pengembangan Sistem

Dalam pembangunan suatu sistem berbasis teknologi informasi diperlukan suatu pendekatan dan pengembangan sistem yang akan menentukan proses penyelesaian rekayasa perangkat lunak, adapun pendekatan sistem yang dilakukan penulis adalah dengan menggunakan pendekatan berorientasi objek (object oriented) dan pengembangan sistem dengan menggunakan model *prototype*.

3.2.3.1 Metode Pendekatan Sistem

Metode pendekatan sistem yang penulis ambil adalah paradigma berorientasi objek (*Object Oriented*), model pemrograman yang berorientasi pada obyek. Maksudnya adalah, bahwa setiap bagian dari apa yang anda program - visual maupun non visual- adalah sebuah obyek. Obyek, dalam konteks ini adalah

"sesuatu", bisa dianalogikan dengan benda, yang padanya melekat attribut-attribut. Attribut tersebut bisa dikelompokkan ke dalam 2 jenis yaitu kelompok "property" dan kelompok "method" atau "event"(Y. Wibisono).

OOP adalah paradigma pemrograman yang cukup dominan saat ini, karena mampu memberikan solusi kaidah pemrograman modern. Meskipun demikian, bukan berarti bahwa pemrograman prosedural sudah tidak layak lagi (Efendi Hariyadi) .

OOP diciptakan karena dirasakan masih adanya keterbatasan pada bahasa pemrograman tradisional. Konsep dari OOP sendiri adalah, semua pemecahan masalah dibagi ke dalam objek. Dalam OOP data dan fungsi-fungsi yang akan mengoperasikannya digabungkan menjadi satu kesatuan yang dapat disebut sebagai objek. Proses perancangan atau desain dalam suatu pemrograman merupakan proses yang tidak terpisah dari proses yang mendahului, yaitu analisis dan proses yang mengikutinya. Pembahasan mengenai orientasi objek tidak akan terlepas dari konsep objek seperti inheritance atau penurunan, encapsulation atau pembungkusan. Empat konsep yang dimiliki oleh pemograman yang berorientasi object adalah :

1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism

3.2.3.1.1 Encapsulation

Encapsulation adalah merupakan penyembunyian informasi melalui private dan protected. Pada dasarnya adalah kita tidak perlu mengetahui apa yang

ada dalam objek itu (class), yang kita butuhkan adalah apa kegunaan, bagaimana cara memakainya dan apa yang akan terjadi.

3.2.3.1.2 Abstraction

Abstraction memiliki pengertian bahwa dalam perangkat lunak seorang programmer akan menggunakan atau menciptakan sejumlah objek dimana objek tersebut ada dan berprilaku pada system

3.2.3.1.3 Inheritance

Inheritance atau pewarisan adalah merupakan suatu objek yang berasal dari turunan pertama atau lebih objek lainnya. Intinya adalah mengembangkan objek yang sudah ada. Jadi kita tidak perlu membentuk objek dari awal, objek yang sudah ada bisa dimodifikasi dan/atau ditambah fungsinya sesuai dengan kebutuhan.

3.2.3.1.4 Polymorphism

Polymorphism adalah kemampuan objek berubah menjadi objek yang lain, beberapa objek bertipe sama bereaksi secara berbeda terhadap “pesan” yang sama.

3.2.3.2 Metode Pengembangan Sistem

Metode pengembangan sistem yang digunakan pada penelitian ini adalah *prototype*. *Prototyping* adalah proses yang digunakan untuk membantu pengembangan perangkat lunak dalam membentuk model dari perangkat lunak yang harus dibuat. *Prototyping* dimulai dengan mengumpulkan kebutuhan. Pengembang dan pelanggan bertemu dan mendefinisikan objektif keseluruhan

dari perangkat lunak, mengidentifikasi segala kebutuhan yang diketahui dan area garis besar dimana definisi lebih jauh merupakan keharusan, kemudian dilakukan "perancangan kilat".

Perancangan kilat berfokus pada penyajian dari aspek-aspek perangkat lunak tersebut yang akan nampak bagi pelanggan/pemakai (contohnya pendekatan *input* dan format *output*). Perancangan kilat membawa pada konstruksi sebuah *prototype*. *Prototype* tersebut dievaluasi oleh pelanggan/pemakai dan dipakai Uji Pelanggan.

Untuk menyaring kebutuhan pengembangan perangkat lunak. Ilustrasi terjadi pada saat *prototype* distel untuk memenuhi kebutuhan pelanggan, dan pada saat yang sama memungkinkan pengembang untuk secara lebih baik memahami apa yang harus dilakukannya. Dengan demikian secara ideal *prototype* berfungsi sebagai sebuah mekanisme untuk mengidentifikasi kebutuhan perangkat lunak.

Adapun tahapan merancang sistem yang penulis gunakan dalam mekanisme pengembangan sistem dengan Prototype adalah sebagai berikut :

1. Identifikasi Kebutuhan dan perbaikan

Penulis akan mengumpulkan segala kebutuhan untuk merancang sistem. Dengan cara observasi, wawancara pemakai menjelaskan kebutuhan sistem dan serta dengan teknik dokumentasi penulis mengumpulkan data-data yang dibutuhkan.

2. Membuat Prototipe

Pada tahap ini Penulis akan menerjemahkan data yang telah dirancang ke dalam bahasa pemrograman (Program contoh atau setengah jadi)

3. Menguji Prototipe

Pada tahap ini penulis akan melakukan pengujian prototipe terhadap pemakai dan memberikan kritikan dan saran.

4. Memperbaiki Prototipe

Pada tahap ini penulis akan melakukan modifikasi sesuai dengan masukan pemakai sesuai dengan. Kemudian dibuat program kembali dan di evaluasi oleh pemakai sampai semua kebutuhan user terpenuhi.

5. Mengembangkan Versi Produksi

Penulis menyelesaikan sistem dengan masukan terakhir dari pemakai.

Prototype yang merupakan mekanisme untuk mengidentifikasi kebutuhan perangkat lunak.

a. Keuntungan prototype

1. Evaluasi dan feedback pada rancangan interaktif.
2. Stakeholder (dalam hal ini user) dapat melihat, menyentuh, berinteraksi dengan prototype.
3. Anggota tim dapat berkomunikasi secara efektif.
4. Para perancang dapat mengeluarkan ide-idenya.
5. Memunculkan ide-ide secara visual dan mengembangkannya.
6. Dapat menjawab pertanyaan dan membantu pemilihan di antara alternatif-alternatif.

7. Mngurangi biaya pengembangan dan pembiayaan
8. Dapat memperbaikimasalah penggunaan sebelum dibuat programnya

b. Kelemahan prototype

1. Prototype hanya akan berhasil jika pemakain bersungguh-sungguh dalam menyediakan waktu dan pikiran untuk mengerjakan prototype.
2. Kemungkinan dokumentasi terabaikan karena pengembangan lebih berkonsentrasi pada pengujian dan pembuatan prototype.
3. Mengingat target waktu yang pendek, ada kemungkinan sistem yang dibuat tidak lengkap dan bahkan kurang teruji.
4. Jika terlalu banyak proses pengulangan dalam membuat prototype, ada kemungkinan pemakai merasa jenuh dan memberikan reaksi negatif
5. Apabila tidak terkelola dengan baik, prototype menjadi tidak pernah berakhir. Hal ini disebabkan permintaan perubahan terlalu mudah untuk dipenuhi

3.2.3.3 Alat Bantu Analisis dan Perancangan

Analisis sistem dapat didefinisikan sebagai penguraian dari suatu sistem informasi yang utuh ke dalam bagian-bagian komponennya dengan maksud untuk mengidentifikasikan dan mengevaluasi permasalahan-permasalahan, kesempatan-kesempatan dan hambatan-hambatan yang terjadi dalam kebutuhan yang diharapkan sehingga dapat diusulkan perbaikan-perbaikan.

Perancangan sistem memiliki tujuan men-design sistem baru yang dapat menyelesaikan masalah-masalah yang dihadapi perusahaan yang diperoleh dari pemilihan alternatif sistem yang terbaik, kegiatan yang dilakukan dalam tahap

perancangan meliputi perancangan *input*, *output* dan *file*. Alat permodelan perancangan sistem adalah sebagai berikut :

3.2.3.3.1 UML (Unified Modelling Language)

Menurut Bambang Hariyanto. Ir.,MT. (2004:17) UML adalah bahasa grafis untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. UML berorientasi objek menerapkan banyak level abstraksi, tidak bergantung proses pengembangan, tidak bergantung bahasa dan teknologi.

Tujuan UML

1. Menyediakan bahasa pemodelan visual yang ekspresif dan siap pakai untuk mengembangkan pertukaran model-model yang berarti.
2. Menyediakan mekanisme perluasan dan spesialisasi untuk memperluas konsep-konsep inti.
3. Mendukung spesifikasi independent bahasa pemprograman dan proses pengembangan tertentu.
4. Menyediakan basis formal untuk pemahaman bahasa pemodelan.

UML mendefinisikan diagram-diagram sebagai berikut:

1. use case diagram
2. class diagram
3. statechart diagram
4. activity diagram
5. sequence diagram
6. collaboration diagram
7. component diagram
8. deployment diagram

3.2.3.3.1.1 Use Case Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”. Sebuah use case 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 behavior- nya sendiri.
- Sementara hubungan generalisasi antar use case menunjukkan bahwa use case yang satu merupakan spesialisasi dari yang lain

3.2.3.3.1.2 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 hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. *Class* memiliki tiga area pokok :

1. Nama (dan stereotype)
2. Atribut
3. Metoda

Hubungan Antar *Class*

1. 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*.
2. Agregasi, yaitu hubungan yang menyatakan bagian ("terdiri atas..").
3. 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.
4. 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.

3.2.3.3.1.3 Statechhart Diagram

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu *state* ke *state* lainnya) suatu objek pada sistem sebagai akibat dari *stimuli* yang diterima. Pada umumnya *statechart diagram* menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu *statechart diagram*). Dalam UML, *state* digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya saat itu. Transisi antar *state* umumnya memiliki kondisi *guard* yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku. *Action* yang dilakukan sebagai akibat dari *event* tertentu dituliskan dengan diawali garis miring. Titik awal dan akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah.

3.2.3.3.1.4 Activity Diagram

Activity diagrams 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. *Activity diagram* merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*).

Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum. Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case*

menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

3.2.3.3.1.5 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 atar 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*.

3.2.3.3.1.6 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

3.2.3.3.1.7 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.

3.2.3.3.1.8 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.

3.2.4 Pengujian *Software*

Adalah suatu cara atau teknik untuk menguji perangkat lunak yang digunakan secara lengkap dan mempunyai kemungkinan tinggi untuk menemukan

kesalahan. Salah satu pengujian *software* yang penulis gunakan adalah pengujian *Black box*.

3.2.4.1 Blackbox

Digunakan untuk menguji fungsi-fungsi khusus dari perangkat lunak yang dirancang. Kebenaran perangkat lunak yang diuji hanya dilihat berdasarkan keluaran yang dihasilkan dari data atau kondisi masukan yang diberikan untuk fungsi yang ada tanpa melihat bagaimana proses untuk mendapatkan keluaran tersebut. Dari keluaran yang dihasilkan, kemampuan program dalam memenuhi kebutuhan pemakai dapat diukur sekaligus dapat diiketahui kesalahan-kesalahannya

Pengujian *black box* berfokus pada persyaratan fungsional perangkat lunak. Dengan demikian, pengujian *black box* memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi input yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program. Pengujian *black box* berusaha menemukan kesalahan dalam kategori sebagai berikut :

a. Fungsi – fungsi yang tidak benar atau hilang.
b. Kesalahan interface.
c. Kesalahan dalam struktur data atau akses database eksternal.
d. Kesalahan kinerja.
e. Inisialisasi dan kesalahan terminasi.

Berbeda dengan pengujian white box, pengujian black box cenderung diaplikasikan selama tahap akhir pengujian. Pengujian black box harus dapat menjawab pertanyaan sebagai berikut:

1) Bagaimana validitas fungsional diuji.
2) Kelas input apa yang akan membuat kasus pengujian menjadi lebih baik.
3) Apakah sistem akan sangat sensitive terhadap harga input tertentu.
4) Bagaimana batasan dari suatu data diisolasi.
5) Kecepatan data apa dan volume data apa yang akan ditoleransi oleh sistem.
6) Apa pengaruh kombinasi tertentu dari data terhadap sistem operasi