# BAB II
# DASAR TEORI

# BAB II
# LANDASAN TEORI

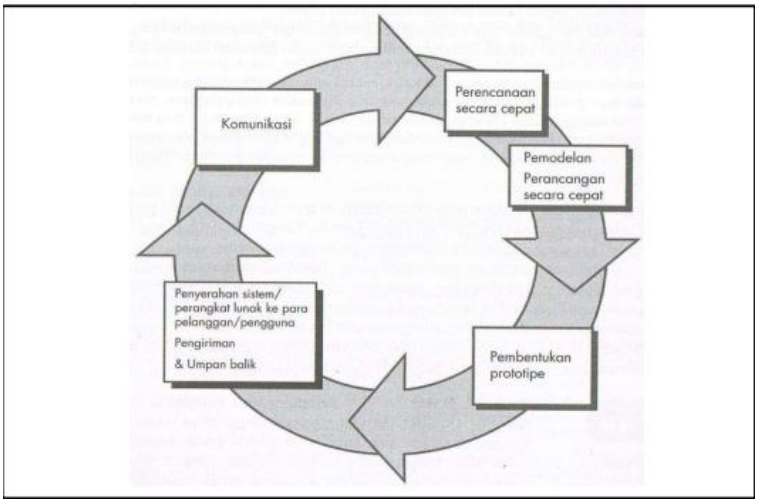
## 2.1 Landasan Teori

### 2.1.1 Rekayasa Perangkat Lunak

Rekayasa Perangkat Lunak (RPL) merupakan salah satu aspek terpenting dalam melakukan pengembangan perangkat lunak yang akan melibatkan proses sistematis dalam merancang, mengembangkan, menguji serta memelihara perangkat lunak. Rekayasa perangkat lunak bertujuan untuk menciptakan perangkat lunak yang berkualitas, efisien dan juga dapat digunakan dalam berbagai kasus atau permasalahan [3]. Rekayasa perangkat lunak adalah suatu disiplin ilmu yang membahas semua aspek produksi perangkat lunak mulai dari tahap awal yaitu *communication,* analisis kebutuhan pengguna, desain, pengkodean, pengujian sampai dengan pemeliharaan sistem setelah dikembangkan [4]. Berikut ini metode dan *tools* pembangunan rekayasa perangkat lunak yang digunakan :

***a) Prototype***

Metode *Prototype* merupakan suatu paradigma baru dalam metode pengembangan perangkat lunak dimana metode ini tidak hanya sekedar evolusi dalam dunia pengembangan perangkat lunak, tetapi juga merevolusi metode pengembangan perangkat lunak yang lama yaitu sistem sekuensial yang biasa dikenal dengan nama SDLC atau waterfall development model. Dalam model *prototype*, *prototype* dari perangkat lunak yang dihasilkan kemudian dipresentasikan kepada pelanggan, dan pelanggan tersebut diberikan kesempatan untuk memberikan masukan sehingga perangkat lunak yang dihasilkan nantinya betul-betul sesuai dengan keinginan dan kebutuhan pelanggan [5]. Gambaran dari metode pengembangan sistem dengan metode *prototype* ini dapat dilihat pada Gambar 2. 1 berikut:




**Gambar 2. 1** Tahapan *Prototype*

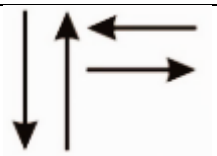
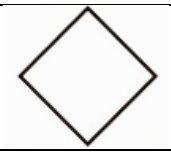
a. Pengumpulan data awal
   Tahapan awal dari model *prototype* guna mengidentifikasi permasalahan yang ada, serta informasi yang diperlukan untuk membangun sistem.
b. Perencanaan
   Tahapan ini dikerjakan dengan kegiatan penentuan sumberdaya, spesifikasi untuk pengembangan berdasarkan kebutuhan sistem, dan tujuan berdasarkan pada hasil komunikasi yang dilakukan agar pengembangan dapat sesuai dengan yang diharapkan.

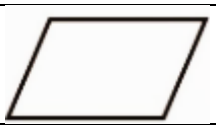
c. Pemodelan
Tahapan selanjutnya yaitu pembuatan desain awal secara sederhana tentang sistem yang akan dikembangkan. Dalam tahap ini, *prototype* yang dibangun dengan sistem rancangan sementara kemudian dievaluasi terhadap klien apakah sudah sesuai dengan yang diinginkan atau masih perlu untuk dievaluasi kembali.
d. Konstruksi (pembentukan *prototype*)
Tahapan ini digunakan untuk membangun *prototype* dan menguji coba sistem yang dibangun.
e. Penyerahan sistem
Tahapan ini dibutuhkan untuk mendapatkan *feedback* dari pengguna, sebagai hasil evaluasi dari tahapan sebelumnya dan implementasi dari sistem yang dikembangkan.

***b) Flowchart***

*Flowchart* adalah cara penulisan algoritma dengan menggunakan notasi grafis. *Flowchart* merupakan gambar atau bagan yang memperlihatkan urutan atau langkah-langkah dari suatu program dan hubungan antar proses beserta pernyataannya. Gambaran ini dinyatakan dengan simbol, dengan demikian setiap simbol menggambarkan proses tertentu. Sedangkan antara proses digambarkan dengan garis penghubung. Dengan menggunakan *flowchart* akan memudahkan programmer untuk melakukan pengecekan bagian-bagian yang terlupakan dalam analisis masalah [6]. Menurut Tominanto dan Subinarto *flowchart* didefinisikan juga sebagai bagan-bagan yang mempunyai arus yang menggambarkan langkah-langkah penyelesaian suatu masalah. *Flowchart* dapat juga merupakan penggambaran secara grafik dari langkah-langkah dan urut-urutan prosedur dari suatu program. Berikut ini adalah simbol-simbol pada flowchart dapat dilihat pada Tabel 2.1.

**Tabel 2. 1** Simbol-simbol *Flowchart*

| No | Nama | Gambar | Deskripsi |
|---|---|---|---|
| 1. | *Flow Direction Symbol* |  | Simbol yang digunakan untuk menghubungkan antara simbol yang satu dengan simbol yang lain. |
| 2. | Terminator |  | Simbol untuk permulaan (*Start*) atau akhir (*Stop*) dari suatu kegiatan. |
| 3. | *Processing Symbol* |  | Simbol yang menunjukkan pengolahan yang dilakukan oleh komputer. |
| 4. | Simbol Manual Operation |  | Simbol yang menunjukkan pengolahan yang tidak dilakukan oleh komputer. |
| 5. | *Decision Symbol* |  | Simbol pemilihan proses berdasarkan kondisi yang ada. |
| 6. | Simbol Dokumen |  | Simbol yang menyatakan input berasal dari dokumen dalam bentuk kertas atau output dicetak ke kertas. |

| No | Nama | Gambar | Deskripsi |
|---|---|---|---|
| 7. | Simbol Input-Output |  | Simbol yang menunjukkan proses input dan output tanpa tergantung dengan jenis peralatannya. |

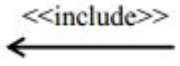
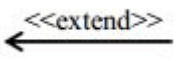
**c) UML (*Unified Modeling Language*)**

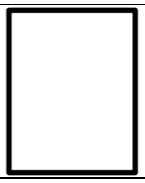
UML (*Unified Modeling Language*) adalah suatu metode dalam pemodelan secara visual yang digunakan sebagai sarana perancangan sistem berorientasi objek. UML juga didefinisikan sebagai suatu Bahasa standar visualisasi, perancangan, dan pendokumentasian sistem, atau dikenal juga sebagai Bahasa standar penulisan *blueprint* sebuah software. UML diharapkan mampu mempermudah pengembangan perangkat lunak serta memenuhi semua kebutuhan pengguna dengan efektif, lengkap dan tepat. Terdapat beberapa diagram dalam UML diantaranya:

**1. Use case Diagram**

Use case adalah sarana untuk menggambarkan persyaratan sebuah sistem yaitu sistem apa yang seharusnya digunakan. Komponen use case yaitu Aktor, use case, dan subjek (sistem) [7]. Menurut Tohari dalam Tabrani dan Aghniya menyimpulkan bahwa use case diagram adalah rangkaian atau uraian sekelompok yang saling terkait dan membentuk sistem secara teratur yang dilakukan atau diawasi oleh sebuah aktor. Use case adalah gambaran fungsionalitas dari suatu sistem, sehingga customer atau pengguna sistem paham dan mengerti mengenai kegunaan sistem yang akan dibangun. Berikut ini adalah simbol-simbol yang ada pada diagram use case dapat dilihat pada Tabel 2.2.

**Tabel 2. 2** Simbol-simbol diagram use case

| No | Nama | Gambar | Deskripsi |
|---|---|---|---|
| 1. | Aktor / Actor |  | Aktor: Mewakili peran orang, sistem yang lain, atau alat ketika berkomunikasi dengan use case. |
| 2. | Use case |  | Use case: Abstraksi dari interaksi antara sistem dan actor. |
| 3. | *Association* |  | *Association:* Abstraksi dari penghubung antara aktor dengan use case. |
| 4. | Generalisasi |  | Generalisasi: Menunjukkan spesialisasi aktor untuk dapat berpartisipasi dengan use case. |
| *5.* | *Include* |  | *Include:* Menunjukkan bahwa suatu use case seluruhnya merupakan fungsionalitas dari use case lainnya. |
| 6. | *Extend* |  | *Extend:* Menunjukkan bahwa suatu use case merupakan tambahan fungsional dari use case |

| No | Nama | Gambar | Deskripsi |
|---|---|---|---|
| | | | lainnya jika suatu kondisi terpenuhi. |
| 7. | *System* |  | Menspesifikasikan paket yang menampilkan sistem secara terbatas. |

### 2.1.2 Pemrograman Berorientasi Objek (PBO)

Pemrograman berorientasi objek (PBO) merupakan sebuah konsep pemrograman yang menggambarkan suatu proses penyelesaian masalah pada program dianalogikan sebagai objek yang saling berinteraksi satu sama lain. Sebagaimana objek di dunia nyata PBO berusaha untuk memodelkan program ke dalam objek-objek saling berinteraksi. Data atribut suatu objek akan disimpan ke dalam bentuk *field* atau variabel sedangkan perilaku dari sesuatu akan disimpan ke dalam metode atau prosedur. Maka objek pada PBO berupa sekumpulan *field* dan metode terkait dengan objek tersebut. Pemrograman berorientasi objek atau *object oriented programming* (OOP) merupakan suatu pendekatan pemrograman yang menggunakan object dan class [8].

Berikut ini adalah beberapa konsep dasar yang harus dipahami tentang pemrograman berorientasi objek:

a. Class
   Class adalah kumpulan objek-objek dengan karakteristik yang sama. Sebuah class akan mempunyai sifat (atribut), kelakuan (operasi/metode), hubungan (relationship).
b. Object
   Objek adalah abstraksi dan sesuatu yang mewakili dunia nyata seperti benda, manusia, satuan organisasi, tempat, kejadian, struktur, status, atau hal-hal lain yang bersifat abstrak.
c. Method
   method pada sebuah kelas hampir sama dengan fungsi atau prosedur pada metodologi struktural. Sebuah kelas boleh memiliki lebih dari satu method. Method dapat berasal dari event, aktivitas atau aksi keadaan, fungsi, atau kelakuan dunia nyata. Contoh method misalnya read, write, move, copy, dan sebagainya.
d. Abstraksi
   Prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.
e. Enkapsulasi
   Enkapsulasi dapat dianggap sebagai sebuah bungkusan. Enkapsulasi inilah yang diimplementasikan dalam sebuah kelas bahwa di dalam sebuah kelas terdiri dari atribut dan metode yang dibungkus dalam suatu kelas. Enkapsulasi pada sebuah kelas bertujuan untuk melindungi atribut dan metode-metode yang ada di dalam kelas agar tidak sembarangan diakses oleh kelas lain.
f. Atribut
   Atribut dari sebuah kelas adalah variabel global yang dimiliki sebuah kelas. Atribut dapat berupa nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek. Atribut dipunyai secara individual oleh sebuah objek, misalnya berat, jenis, nama, dan sebagainya. Atribut sebaiknya bersifat privat untuk menjaga konsep enkapsulasi [9].

### 2.1.3 Metode FEFO (*First Expired First Out*)

Metode FEFO (*First Expired First Out*) adalah produk yang memiliki tanggal kadaluarsa terdekat yang harus keluar terlebih dahulu. Cara ini biasanya digunakan di Apotek atau toko retail

yang menjual makanan dan minuman (biasanya disimpan didalam kemasan) yang memiliki masa kadaluarsa. Jadi terlepas dari apakah barangnya baik yang pertama atau terakhir datang, produk dengan tanggal kadaluarsa terdekat adalah produk wajib yang dikeluarkan terlebih dahulu. Sedangkan produk dengan masa kadaluarsa yang lama biasanya disimpan lebih lama di dalam gudang penyimpanan [10].

### 2.1.4 *Database*

*Database* atau basis data adalah sekumpulan fakta berupa representasi tabel yang saling berhubungan dan disimpan dalam media penyimpanan secara digital, dalam suatu basis data terdiri dari sekumpulan tabel yang saling berelasi ataupun tidak berelasi. Semua tabel tersebut merupakan representasi tempat untuk menyimpan data, yang mendukung fungsi dari basis data tersebut untuk suatu sistem. Database merupakan sekumpulan data yang terstruktur yang disimpan secara terpadu dalam suatu sistem yang dapat diakses dan dikelola oleh pengguna atau aplikasi. Biasanya, data dalam database diatur dalam tabel atau relasi, dan dapat diakses dan diubah menggunakan bahasa query SQL [11].

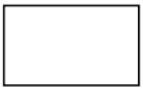
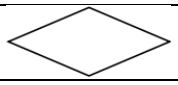
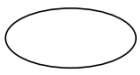
#### a) ERD (*Entity Relationship Diagram*)

Menurut Brady dan Loonam, *Entity Relationship Diagram* (ERD) merupakan teknik yang digunakan untuk memodelkan kebutuhan data dari suatu organisasi, biasanya oleh System Analyst dalam tahap analisis persyaratan proyek pengembangan sistem. ERD menyediakan cara untuk mendeskripsikan perancangan basis data pada peringkat logika. ERD merupakan suatu model untuk menjelaskan hubungan antardata dalam basis data berdasarkan objek-objek dasar data yang mempunyai hubungan antar relasi. Dalam pembentukan ERD terdapat 3 komponen yang akan dibentuk, yaitu entitas, relasi, dan atribut.

1.) Entitas adalah objek yang menarik di bidang organisasi yang dimodelkan.
2.) Relasi adalah hubungan antara dua jenis entitas dan direpresentasikan sebagai garis lurus yang menghubungkan dua entitas.
3.) Atribut memberikan informasi lebih rinci tentang jenis entitas.

Berikut ini adalah simbol-simbol yang ada pada ERD dapat dilihat pada Tabel 2.3.

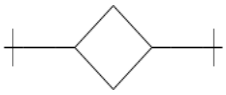
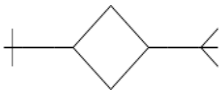
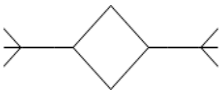
**Tabel 2. 3** Simbol-simbol ERD

| No | Nama | Gambar | Deskripsi |
|---|---|---|---|
| 1. | *Entity* |  | Objek yang dapat dibedakan dalam dunia nyata. |
| 2. | *Relationship* |  | Hubungan yang terjadi antara satu atau lebih *entity.* |
| 3. | *Attribut* |  | Atribut yang bernilai Tunggal atau atribut atomic yang tidak dapat dipilah-pilah lagi. |
| 4. | *Key Attribut* |  | Satu atau gabungan dari beberapa atribut yang membedakan semua baris data (*row*) dalam *table* secara unik. |
| 5. | *Link* |  | Simbol berupa garis ini digunakan sebagai penghubung antara himpunan relasi dengan |

| No | Nama | Gambar | Deskripsi |
|---|---|---|---|
| | | | himpunan entitas dan himpunan entitas dengan atributnya. |

Selain simbol-simbol yang ada di tabel, dalam ERD juga terdapat simbol kardinalitas. Kardinalitas dalam sebuah aturan relationship untuk menentukan jumlah entitas yang dapat direlasikan dengan entitas lainnya melalui relationship set. Simbol-simbol kardinalitas dapat dilihat pada Tabel 2.4.

**Tabel 2. 4** Simbol Kardinalitas

| No | Nama | Gambar | Deskripsi |
|---|---|---|---|
| 1. | *One to One* |  | Setiap anggota entitas A hanya boleh berhubungan dengan satu anggota entitas B |
| 2. | *One to Many* |  | Setiap anggota entitas A dapat berhubungan dengan lebih dari satu anggota entitas B tetapi tidak sebaliknya |
| 3. | *Many to Many* |  | Setiap entitas A dapat berhubungan dengan banyak entitas himpunan entitas B dan demikian pula sebaliknya. |