BAB II
LANDASAN TEORI

2.1 Data, Informasi, dan Pengetahuan

Data adalah bahan utama dari pekerjaan manajemen sistem informasi. Tanpa data pekerjaan informasi tidak akan pernah ada. Data adalah fakta atau kenyataan dari suatu kejadian atau peristiwa, data merupakan suatu kebenaran, karena itu ciri atau karakteristik utama data adalah benar. Data yang merupakan hasil kegiatan pada sebuah organisasi disebut dengan data organisasi.[3]

Informasi adalah bahan yang dihasilkan dari pengolahan data. Informasi adalah data yang sudah diproses menjadi bentuk yang berguna bagi pemakai dan mempunyai nilai pikir yang nyata bagi pembuatan keputusan pada saat sedang berjalan atau untuk prospek masa depan. Definisi tersebut menekankan kenyataan bahwa data harus diproses dengan cara-cara tertentu untuk menjadi informasi dalam bentuk dan nilai yang berguna bagi pemakai.[3] Pengetahuan (*knowledge*) merupakan informasi yang digabung dengan kemampuan dan keterampilan, intuisi, pengalaman, gagasan, motivasi dari sumber yang kompeten.[4]

2.2 *Knowledge Management System*

Knowledge management (KM) adalah proses yang membantu sebuah organisasi untuk mengidentifikasi, memilih, mengatur, menyebarkan, mentransfer informasi dan keahlian penting yang merupakan bagian dari memori sebuah organisasi.[5]

Knowledge management system (KMS) adalah sistem yang dapat mempercepat pengelolaan manajemen pengetahuan baik secara intra ataupun antar perusahaan yang berpusat di sekitar pengetahuan *basic* perusahaan atau menjadi tempat penyimpananan *knowledge*. Tujuan *knowledge management system* adalah mengidentifikasi, menangkap, menyimpan, memelihara, menyampaikan pengetahuan yang bermanfaat dalam bentuk yang berarti bagi siapa saja yang membutuhkannya, dimanapun dan kapanpun didalam sebuah organisasi.[5]

Knowledge management system mendukung *transfer knowledge*, pengambilan keputusan, dan kolaborasi di tingkat organisasi terlepas dari lokasinya. *Knowledge management system* dimaksudkan untuk membantu organisasi dengan mengandalkan keahlian sumber daya manusia yang ada pada sebuah organisasi agar dapat diakses secara luas oleh seluruh pegawai. *Knowledge management system* juga membantu organisasi mempertahankan pengetahuan para pegawai yang sudah tidak ada pada organisasi tersebut.[5]

Fungsi dari *knowledge management system* mengikuti enam langkah berikut dalam sebuah siklus, yaitu [5]:

1. Menciptakan pengetahuan. Pengetahuan diciptakan saat manusia menentukan cara baru dalam melakukan sesuatu atau mengembangkan pengetahuan.
2. Menangkap pengetahuan. Pengetahuan baru harus diidentifikasi sebagai hal yang berharga.
3. Menyaring pengetahuan. Pengetahuan baru harus ditempatkan dalam konteks yang benar sehingga bisa ditindaklanjuti.
4. Menyimpan pengetahuan. Pengetahuan yang berguna kemudian harus disimpan dalam format yang masuk akal dalam repositori pengetahuan sehingga orang lain dalam organisasi dapat mengaksesnya.
5. Mengelola pengetahuan. Seperti perpustakaan, pengetahuan harus tetap terjaga.
6. Diseminasi pengetahuan. Pengetahuan harus tersedia dalam format yang berguna bagi siapa saja di organisasi yang membutuhkannya, di mana saja dan kapan saja.

Sistem KM dikembangkan dengan menggunakan rangkaian teknologi berikut: komunikasi dan kolaborasi, penyimpanan dan pengambilan. Teknologi komunikasi dan kolaborasi memungkinkan pengguna mengakses pengetahuan yang dibutuhkan dan berkomunikasi satu sama lain dengan para ahli. Komunikasi dan kolaborasi juga memungkinkan ajakan pengetahuan dari para ahli. Teknologi penyimpanan dan pengambilan awalnya dimaksudkan dengan menggunakan sistem manajemen basis data untuk menyimpan dan mengelola pengetahuan.[5]

Transfer knowledge merupakan berbagai interaksi baik antara individu maupun kelompok. *Transfer knowledge* adalah komunikasi pengetahuan yang terfokus dan searah antara individu, kelompok, atau organisasi dalam bentuk sedemikian rupa sehingga penerima pengetahuan dapat memiliki pemahaman kognitif, memiliki kemampuan untuk menerapkan pengetahuan, atau menggunakan pengetahuan. [6]

2.3 Rekayasa Perangkat Lunak

Rekayasa Perangkat Lunak adalah pembuatan dan penggunaan prinsip-prinsip keahlian teknik untuk mendapatkan perangkat lunak yang ekonomis, handal dan bekerja secara efisien pada mesin yang sesungguhnya. Rekayasa Perangkat Lunak mendirikan suatu pondasi untuk suatu proses perangkat lunak yang lengkap dengan mengidentifikasi sejumlah aktifitas kerangka kerja yang berlaku untuk semua proyek perangkat lunak, terlepas dari hal ukuran dan kompleksitas.[7]

Dalam Rekayasa Perangkat Lunak terdapat beberapa model sebagai cara atau strategi untuk mewujudkan perangkat lunak yang diinginkan, salah satunya yaitu model *waterfall* dengan proses sebagai berikut :

1. Analisis Kebutuhan Perangkat Lunak

Analisis kebutuhan perangkat lunak merupakan langkah awal untuk menentukan gambaran perangkat yang akan dihasilkan ketika pengembang melaksanakan sebuah proyek pembuatan perangkat lunak.

2. Desain (*Design*)

Tahapan desain adalah tahap setelah analisis dari siklus pengembangan sistem yang merupakan pendefinisian dari kebutuhan-kebutuhan fungsional untuk menggambarkan bagaimana suatu sistem akan dibentuk.

3. Implementasi Kode (*Coding*)

Pada tahap ini setelah mendapatkan data dan desain yang jelas tentang seperti apa program yang akan di buat maka desain tadi harus diubah bentuknya menjadi bentuk yang dapat dimengerti oleh mesin, yaitu ke dalam bahasa pemrograman melalui proses *coding*.

4. Testing

Tahapan *testing* merupakan tahapan dimana semua fungsi-fungsi software harus diujicobakan, agar software bebas dari error, dan hasilnya harus benar-benar sesuai dengan kebutuhan yang sudah didefinisikan sebelumnya.

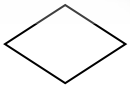
2.4 Basis Data

Basis data adalah kumpulan data yang terbagi dan terhubung secara logikal dan deskripsi dari data yang dirancang untuk memenuhi kebutuhan informasi suatu organisasi.[8] Basis data merupakan sekumpulan file yang saling berhubungan dan terorganisasi atau kumpulan *record-record* yang menyimpan data dan hubungan di antaranya.[9]

2.4.1 *Entity Relationship Diagram* (*ER Diagram*)

ER Diagram adalah diagram yang memperlihatkan *entitas-entitas* yang terlibat dalam suatu sistem serta hubungan-hubungan atau relasi antar *entitas* tersebut. Model *Entity-Relationship* yang berisi komponen-komponen himpunan *entitas* dan relasi yang masing-masing dilengkapi dengan atribut-atribut yang dapat digambarkan dengan lebih sistematis dengan menggunakan diagram *Entity-Relationship*.[10]

Tabel 2.1 Simbol-simbol *ER Diagram* (ERD)

NO	GAMBAR	NAMA	KETERANGAN
1		Entitas	Melambangkan himpunan entitas.
2		Relasi	Melambangkan himpunan relasi.
3		Penghubung	Penhubung antara himpunan relasi dengan himpunan entitas dan himpunan entitas dengan atributnya.

2.5 *Unified Modelling Languange* (UML)

UML yang merupakan singkatan dari *Unified Modelling Language* adalah sekumpulan pemodelan konvensi yang digunakan untuk menentukan atau menggambarkan sebuah sistem perangkat lunak dalam kaitannya dengan objek. [11]

UML merupakan suatu bahasa. Suatu bahasa terdiri dari kata-kata, dan memiliki aturan untuk menggabungkan kata-kata tersebut, sehingga tercipta komunikasi. Sebuah permodelan bahasa adalah suatu bahasa dimana kata-kata dan aturannya berfokus pada penggambaran sistem secara konseptual dan fisik. Sebuah permodelan bahasa seperti UML telah menjadi bahasa standar untuk merencanakan suatu aplikasi.[12]

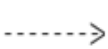
2.5.1 Komponen-komponen UML

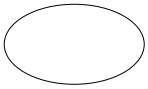
UML mendefinisikan diagram-diagram berikut ini :

1. *Use Case Diagram*

Suatu *use case diagram* menampilkan sekumpulan *use case* dan aktor (pelaku) dan hubungan diantara *use case* dan aktor tersebut. *Use case diagram* digunakan untuk penggambaran *use case* statik dari suatu sistem. *Use case diagram* penting dalam mengatur dan memodelkan kelakuan dari suatu sistem.

Tabel 2.2 Simbol-simbol *Use Case*

NO	GAMBAR	NAMA	KETERANGAN
1		*Actor*	Menspesifikasikan himpunan peran yang pengguna mainkan ketika berinteraksi dengan *use case*.
2		*Include*	Menspesifikasikan bahwa *use case* sumber secara *eksplisit*.
3		*Extend*	Menspesifikasikan bahwa *use case* target memperluas perilaku dari *use case* sumber pada suatu titik yang diberikan.

4		*Association*	Apa yang menghubungkan antara objek satu dengan objek lainnya.
5		*Use Case*	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu aktor.

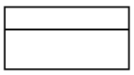
2. *Class Diagram*

Class diagram menunjukan sekumpulan kelas, antarmuka, dan kerjasama serta hubungannya. *Class diagram* digunakan untuk memodelkan perancangan statik dari gambaran sistem. Biasanya meliputi permodelan *vocabulary* dari sistem, permodelan kerjasama, atau permodelan skema.

Class diagram dapat digunakan untuk membangun sistem yang dapat dieksekusi melalui teknik *forward and reverse*, selain untuk penggambaran, penspesifikasian, dan pendokumentasian struktur model. *Class* Diagram terdiri dari:

a) Nama *Class*.
b) Atribut.
c) Operasi/Method.

Tabel 2.3 Simbol-simbol *Class Diagram*

NO	GAMBAR	NAMA	KETERANGAN
1		*Generalization*	Hubungan dimana objek anak *(descendent)* berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (*ancestor*).
2		*Class*	Himpunan dari objek-objek yang berbagi atribut serta operasi yang sama.
3		*Association*	Apa yang menghubungkan antara objek satu dengan objek lainnya.

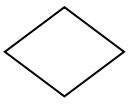
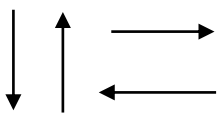
Atribut dan Operasi/*method* dapat memiliki tiga sifat berikut:

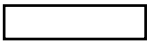
- *Public*, dapat dipanggil oleh *class* apa saja.
- *Protected*, hanya dapat dipanggil atau diakses oleh *class* yang bersangkutan dan *class* turunannya.
- *Private*, hanya dapat dipanggil oleh dirinya sendiri (tidak dapat diakses dari luar *class* yang bersangkutan).

3. *Activity Diagram*

Activity diagram adalah diagram yang menunjukan aliran dari aktivitas ke aktivitas lainnya dalam sebuah system. *Activity diagram* menangani sudut pandang sistem secara dinamis. Mereka biasanya penting dalam pemodelan fungsi dalam sistem dan menekankan pada kontrol aliran di antara objek.

Tabel 2.4 Simbol-simbol *Activity Diagram*

NO	GAMBAR	NAMA	KETERANGAN
1		*Action*	State dari sistem yang mencerminkan eksekusi dari suatu aksi.
2		*Initial Node*	Bagaimana objek dibentuk atau diawali.
3		*Final Node*	Bagaimana objek dibentuk dan diakhiri.
4		*Decision*	Untuk menggambarkan suatu keputusan/tindakan yang harus diambil pada kondisi tertentu.
5		*Line Connector*	Untuk menghubungkan satu simbol dengan simbol lainnya.

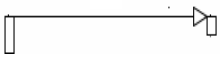
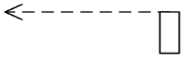
6		*Fork Node* dan *Join Node*	Untuk memecah behavior menjadi aktivitas yang paralel dan untuk menyatukan aktivitas yang paralel.

4. *Sequence Diagram*

Suatu *sequence diagram* adalah suatu diagram interaksi yang menekankan pada pengaturan waktu dari pesan-pesan. Diagram ini menampilkan sekumpulan peran dan pesan-pesan yang dikirim dan diterima oleh instansi yang memegang peranan tersebut. *Sequence diagram* menangkap objek dan *class* yang terlibat dalam skenario dan urut-urutan pesan yang ditukar antara objek diperlukan untuk melaksanakan fungsionalitas skenario.

Sequence diagram berasosiasi dengan *use case* selama proses pengembangan. Dalam *Unified Model Language* (UML), objek dalam *sequence diagram* digambar dengan segiempat yang berisi nama objek yang diberi garis bawah. Objek dapat diberi nama dengan tiga cara : (nama objek), (nama objek dan *class*) atau (hanya nama *class* (*anonymous* object)).

Tabel 2.5 Simbol-simbol *Sequence Diagram*

NO	GAMBAR	NAMA	KETERANGAN
1		*Life Line*	Objek *entity*, antarmuka yang saling berinteraksi.
2		*Message*	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktifitas yang terjadi.
3		*Return Message*	*Message* kembali dalam komunikasi antar objek tentang aktifitas yang terjadi.

2.6 *Objek Oriented Programming* (OOP)

Pemrograman berorientasi objek atau OOP adalah metode pemrograman di mana *developer* membuat dan mengelompokkan kode-kode yang berkaitan menjadi suatu objek. Sehingga setiap objek dapat memiliki data dan fungsi sendiri, data dan fungsi tersebut dapat digunakan untuk memanggil objek yang bersangkutan terlebih dahulu.[13]

Salah satu keunggulan OOP dibandingkan teknik pemrograman terstruktur adalah OOP memungkinkan *developer* untuk membuat modul yang tidak perlu berubah ketika suatu objek baru ditambahkan. Bahkan *developer* dapat membuat suatu objek baru yang mewarisi beberapa fitur dari objek yang sudah ada. Hal ini membuat aplikasi yang berorientasi objek lebih mudah dimodifikasi (diperbaiki) dan dikembangkan dibandingkan pemrograman terstruktur.[13]