BAB II

TINJAUAN PUSTAKA

II.1. Sistem

Tujuan dari suatu sistem tergantung pada jenis sistem itu sendiri. Sebagai contoh , sistem peredaran darah manusia merupakan sistem biologi yang memiliki tujuan untuk mengedarkan darah yang mengandung oksigen dan sari makanan keseluruh tubuh. Sedangkan sistem buatan manusia seperti sistem yang terdapat di sekolah, organisasi bisnis, atau instansi pemerintah juga mempunyai tujuan yang berbeda. Organisasi bisnis biasanya memiliki tujuan yang lebih jelas, seperti yang telah disebutkan pada bagian sebelumnya, yaitu mendapatkan laba. Sistem merupakan serangkai bagian yang saling tergantung dan bekerja sama untuk mencapai tujuan tertentu.

Suatu sistem pasti tersusun dari sub-sub sistem yang lebih kecil dan juga saling tergantung dan bekerja sama untuk mencapai tujuan. Sebagai contoh, sistem administrasi universitas terdiri dari sistem dari sub-sub sistem administrasi fakultas dan sub-sub sistem administrasi jurusan fakultas. Sistem informasi yang kadang kala disebut sebagai sistem pemrosesan data, merupakan sistem buatan manusia yang biasanya terdiri dari sekumpulan komponen, baik manual ataupun berbasis komputer yang terintegrasi untuk mengumpulkan, menyimpan dan mengelola data serta menyediakan informasi kepada pihak yang berkepentingan sebagai pemakai informasi tersebut (Anastasia Diana; 2011 : 3).

II.2. Sistem Pakar

Pakar adalah seorang individu yang memiliki pengetahuan khusus, pemahaman pengalaman dan metode-metode yang digunakan untuk memecahkan persoalan dalam bidang tertentu. Seorang pakar memiliki kemapuan seorang kepakaran, yaitu :

1. Dapat mengenali dan merumuskan suatu masalah.
2. Menjelaskan solusi dari suatu masalah.
3. Restrukturisasi pengetahuan.
4. Belajar dari pengalaman.
5. Memahami batas kemampuan (Rika Rosnelly ; 2012: 10)

Sistem Pakar pada umumnya dirancang untuk memenuhi beberapa karakteristik umum berikut ini :

1. Kinerja sangat baik (*high performance*). Sistem harus mampu memberikan respon berupa saran (*advice*) dengan tingkat kualitas yang sama dengan seorang pakar atau melebihinya.
2. Waktu respon yang baik (*adequate respon time*). Sistem juga harus mampu bekerja dalam waktu yang sama baiknya (*reasonable*) atau lebih cepat dibandingkan dengan seorang pakar dalam menghasilkan keputusan.
3. Dapat diandalkan (*good reliability*). Sistem harus dapat diandalkan dan tidak mudah rusak/ crash.

4. Dapat dipahami (*understandable*). Sistem harus mampu menjelaskan lngkah-langkah penalaran yang dilakukannya seperti seorang pakar.
5. Fleksibel (*fleksibility*). Sistem harus menyediakan mekanisme untuk menambah, mengubah, dan menghapus pengetahuan (Rika Rosnelly; 2012 :20).

II.3. Certainty Factor

Menurut Rika Rosnelly (2012: 89) Certainty factor (CF) menunjukkan ukuran kepastian terhadap suatu fakta atau aturan. Notasi faktor kepastian :

CF[h,e]=MB[h,e]-MD[h,e]

Dengan :

CF[h,e] = ukuran kepastian

MB[h,e] = ukuran kepercayaan terhadap hipotesis h, jika diberikan evidence e (antara 0 dan 1)

MD[h,e] = ukuran ketidakpercayaan terhadap hipotesis h, jika diberikan evidence e (antara 0 dan 1)

Andaikan suatu observasi memberikan kepercayaan terhadap h dengan $MB[h,e_1]=0,3$ dan $MD[h,e_1]=0$. Sehingga $CF[h, e_1]=0,3-0=0,3$.

Jika observasi baru dengan $MB[h,e_2]=0,2$ dan $MD[h,e_2]=0$, maka :

$MB[h,e_1 \wedge e_2]=0,3+0,2*(1-0,3)=0,44$

$MD[h,e_1 \wedge e_2]=0$

$CF[h,e_1 \wedge e_2]=0,44-0=0,44$

II.4. Unified Modeling Language

Unified Modeling Language (UML) adalah suatu alat untuk memvisualisasikan dan mendokumentasikan hasil analisa dan desain yang berisi sintak dalam memodelkan sistem secara visual (Braun, *et. al*.2011). Juga merupakan satu kumpulan konvensi pemodelan yang digunakan untuk menentukan atau menggambarkan sebuah sistem *software* yang terkait dengan objek (Whitten, *et. al*. 2004)

UML merupakan salah satu alat bantu yang sangat handal dalam bidang pengembangan sistem berorientasi objek karena UML menyediakan bahasa pemodelan visual yang memungkinkan pengembang sistem membuat *blue print* atas visinya dalam bentuk yang baku. UML berfungsi sebagai jembatan dalam berkomunikasikan beberapa aspek dalam sistem melalui sejumlah elemen grafis yang bisa dikombinasikan menjadi diagram. UML mempunyai banyak diagram yang dapat mengakomodasi berbagai sudut pandang dari suatu perangkat lunak yang akan dibangun.(Yuni Sugiarti; 2013:36-37)

Dengan menggunakan UML, kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemograman apapun. Tetapi karena UML juga menggunakan dan *operation* dalam konssep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa-bahasa berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian UML tetap dapat digunakan untuk *modelling* aplikasi prosedural dalam VB atau C.

Seperti bahasa-bahasa lainnya, UML mengidentifikasikan notasi dan *syntax*/semantik. 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: *Grady Booch OOD* (*Object-Oriented Software Design*), Jim Rumbaugh OMT (*Object Modeling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*) (Yuni Sugiarti;2013:34).

Blok pembangunan utama UML adalah diagram. Beberapa diagram ada yang rinci (jenis *timming diagram*) dan lainnya ada yang bersifat umum (misalnya diagram kelas). Para pengembang sistem berorientasi objek menggunakan bahasa model untuk menggambarkan, membangundan mendokumentasikan sistem yang mereka rancang. UML memungkinkan para anggota team untuk bekerja sama dengan bahasa model yang sama dengan mengaplikasikan beragam sistem. Intinya UML merupakan alat komunikasi yang konsisten dalam mendukung para pengembang sistem saat ini.

Adapun beberapa bagian dari *Unified Modelling Languange* (UML) adalah sebagai berikut :

II.4.1. *Use Case* Diagram

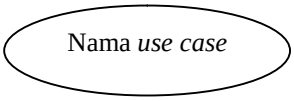
Dalam membuat sebuah sistem, langkah awal yang perlu dilakukan adalah menentukan kebutuhan. Terdapat dua jenis kebutuhan, yaitu kebutuhan fungsional dan kebutuhan nonfungsional. Kebutuhan fungsional adalah kebutuhan pengguna

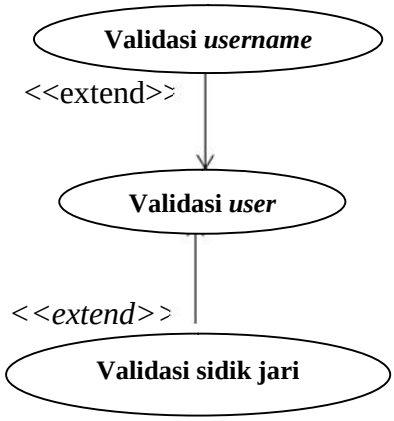
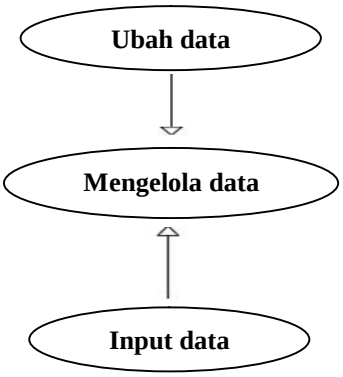
dan *stakeholder* sehari-hari yang akan dimiliki oleh sistem, dimana kebutuhan ini akan digunakan oleh pengguna *stakeholder*. Sedangkan kebutuhan nonfungsional adalah kebutuhan yang memperhatikan hal-hal berikut yaitu performansi, kemudahan dalam menggunakan sistem, kehandalan sistem, keamanan sistem, keuangan, legalitas, dan operasional.

Kebutuhan fungsi akan digambarkan melalui sebuah diagram yang dinamakan diagram *use case.* *Use case* diagram atau diagram use case merupakan pemodelan untuk menggambarkan kelakuan (*behavior*) sistem yang akan dibuat. Diagram *use case* mendeskripsikan sebuah interaksi antar satu atau lebih *actor* dengan sistem yang akan dibuat. Dengan pengertian yang cepat, diagram *use case* digunakan untuk mengetahui fungsi apa saja yang ada didalam sebuah sistem dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut. Terdapat beberapa simbol dalam menggambarkan diagram *use case*, yaitu *use case, actor* dan relasi. Hal perlu diingat mengenai diagram *use case* adalah diagram *use case* bukan menggambarkan tampilan antarmuka (*user interface*), arsitektur dari sistem, kebutuhan nonfungsional, dan tujuan performansi. Sedangkan untuk penamaan *use case* adalah nama didefinisikan sesimpel mungkin, dapat dipahami dan menggunakan kata kerja (Yuni Sugiarti; 2013:41).

Berikut ini pada tabel II.1. adalah simbol-simbol yang ada pada diagram *use case*:

Tabel II.1. Simbol – Simbol *Use Case* Diagram

Simbol	**Deskripsi**
Use case 	Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor, biasanya dinyatakan dengan menggunakan kata kerja diawal diawal frase nama *use case*
Aktor / *actor*  **nama aktor**	orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari *actor* adalah gambar orang, tapi *aktor* belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal nama *aktor*.
Asosiasi/ *association* 	Komunikasi antara *actor* dan *use case* yang berpartisipasi pada *use case* atau *use case* memiliki interaksi dengan aktor
Ekstensi / *extend* <<*extend*>> 	Relasi *use case* tambahan kesebuah *use case* dimana *use case* yang ditambahkan dapat berdiri sendiri walau tanpa *use case* tambahan itu: mirip dengan prinsip *inheritance* pada pemograman berorientasi objek; biasanya *use*

	case tambahan memiliki nama depan yang sama dengan *use case* yang ditambahkan, misal  Arah panah mengarah pada *use case* yang ditambahkan; biasanya *use case* yang menjadi *extend*-nya merupakan jenis yang sama dengan *use case* yang menjadi induknya.
Generalisasi/*generalization* 	Hubungan generalisasi dan spesialisasi (umum – khusus) antara dua buah *use case* dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya, misalnya:  Arah panah mengarah pada *use case* yang menjadi generalisasinya (umum).

Menggunakan / *include* / *uses*	Relasi *use case* tambahan ke sebuah *use case* dimana *use case* yang ditambahkan memerlukan *use case* ini untuk menjalankan fungsinya atau sebagai syarat *use case* ini.
<<*include*>> 	Ada dua sudut pandang yang cukup besar mengenai *include* di *use case* : *include* berarti *use case* yang ditambahkan akan selalu dipanggil saat *use case* tambahan dijalankan.
<<*uses*>> 	*include* berarti *use case* yang tambahan akan selalu melakukan pengecekan apakah *use case* yang ditambahkan telah dijalankan sebelum *use case* tambahan dijalankan.

Sumber : (Rosa A.S, M.Shalahuddin; 2013; 155-158).

II.4.2. *Class Diagram* (Diagram Kelas)

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.

1. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas.
2. Atribut mendeskripsikan *property* dengan sebaris teks didalam kotak kelas tersebut.
3. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

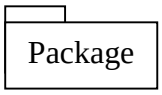
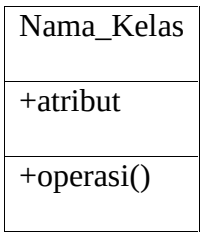
Diagram kelas mendeskripsikan jenis-jenis objek dalam sistem dan berbagai hubungan statis yang terdapat diantara mereka. Diagram kelas juga menunjukkan properti dan operasi sebuah kelas dan batasan-batasan yang terdapat dalam hubungan-hubungan objek. Diagram kelas menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, perwarisan, asosiasi, dan lain-lain (Yuni Sugiarti;2013:57).

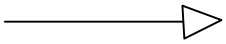
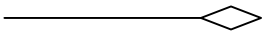
Kelas memiliki tiga area pokok :

1. Nama
2. Atribut
3. Operasi

Berikut adalah tabel II.2 simbol-simbol yang ada pada diagram kelas :

Tabel II.2. Simbol-Simbol Diagram Kelas

Simbol	**Deskripsi**
Package 	Package merupakan bungkusan dari satu kelas atau lebih
Operasi 	Kelas pada struktur system
Antarmuka / *interface*  Nama *interface*	Sama dengan konsep *interface* dalam pemrograman berorientasi objek

Asosiasi / *association* 1 1..*	Relasi antar kelas dengan makna umum, asosiasi biasanya juga disertai dengan *multiplicity*
Asosiasi berarah / *Directed association* 	Relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan *multiplicity*
Generalisasi 	Relasi antar kelas dengan makna generalisasi-spesialisasi (umum khusus)
Kebergantungan 	Relasi antar kelas dengan makna Kebergantungan antar kelas
Agregasi / *aggregation* 	Relasi antar kelas dengan makna Semua bagian (*whole part*)

Sumber : (Yuni Sugiarti;2013:59).

II.4.3. *Activity Diagram* (Aktivitas)

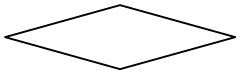
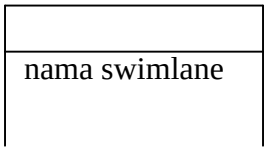
Diagram aktivitas atau *activity diagram* menggambarkan *workflow*(aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. *Activity diagram* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir.

Activity Diagram merupakan *state* diagram khusus, dimana 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 menggambarka proses- proses dan jalur-jalur aktivitas dari level atas secara umum (Yuni Sugiarti;2013:75).

Berikut adalah tabel II.3. yang menggambarkan simbol-simbol yang digunakan pada diagram aktivitas :

Tabel II.3. Simbol *Activity Diagram*

Simbol	**Deskripsi**
Status awal 	Status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah status awal.
Aktivitas 	Aktivitas yang dilakukan sistem, aktivitas biasanya diawali dengan kata kerja.
Percabangan / *decision* 	Asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu.
Penggabungan / *join*	Asosiasi penggabungan dimana lebih dari satu aktivitas digabungkan menjadi satu.
Status akhir 	Status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki sebuah status akhir.
Swimlane 	Memisahkan organisasi bisnis yang bertanggungjawab terhadap aktivitas yang terjadi.

Sumber : (Rosa A.S, M.Shalahuddin; 2013; 161-163).

II.4.4. *Sequence* Diagram

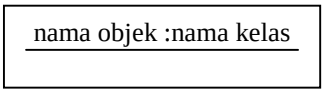
Diagram sequence menggambarkan kelakuan/perilaku objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek.

Banyaknya diagram *sequence* yang digambarkan adalah sebanyak pendefenisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefenisikan interaksi jalannya pesan sudah dicakup pada diagram *sequence* sehingga semakin banyak *use case* didefenisikan maka *diagram sequnce* yang harus dibuat juga semakin banyak (Yuni Sugiarti;2013:69).

Berikut adalah tabel II.4. yang menerangkan simbol-simbol yang ada pada diagram *sequence* :

Tabel II.4. Diagram *Sequence*

Simbol	**Deskripsi**
Aktor  **nama aktor**	orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari *actor* adalah gambar orang, tapi *actor* belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal nama *actor*.

Garis hidup / *lifeline* 	Menyatakan kehidupan suatu objek.
Objek 	Menyatakan objek yang berinteraksi pesan.
Waktu aktif 	Menyatakan objek dalam keadaan aktif dan berinteraksi, semua yang terhubung dengan waktu aktif ini adalah tahapan yang dilakukan didalamnya.
Pesan tipe create <<*create*>> 	Menyatakan suatu objek membuat objek yang lain, arah panah mengarah pada objek yang dibuat
Pesan tipe call 1 : nama_metode{} 	Menyatakan suatu objek memanggil operasi / metode yang ada pada objek lain atau dirinya sendiri.
Pesan tipe send 1 : masukan 	Menyatakan bahwa suatu objek mengirimkan data / masukan / informasi ke objek lainnya, arah panah mengarah pada objek yang dikirimi.
Pesan tipe return	Menyatakan bahwa suatu objek yang telah menjalankan suatu operasi atau metode

1 : keluaran	menghasilkan suatu kembalian ke objek tertentu, arah panah mengarah pada objek yang menerima kembalian.
Pesan tipe destroy <<*destroy*>> 	Menyatakan suatu objek mengakhiri hidup objek yang lain, arah panah mengarah pada objek yang diakhiri, sebaiknya jika ada create maka ada *destroy*.

Sumber : (Rosa A.S, M.Shalahuddin; 2013; 165-167).

II.5. Entity Relationship Diagram

Entity Relationship Diagram/ER_M merupakan suatu model data yang dikembangkan berdasarkan objek. ER_M digunakan untuk menjelaskan hubungan antara data dalam basis data kepada pengguna secara logik. ER_M didasarkan pada suatu persepsi bahwa *real world* terdiri atas objek-objek dasar yang mempunyai hubungan/kerelasian antar objek-objek dasar tersebut. ER_M digambarkan dalam bentuk diagram yang disebut dengan ER *(ER_Diagram/ER_D).* Untuk menggambarkan ER_D digunakan simbol-simbol grafis tertentu (Edhy Sutanta ; 2011 : 91)

II.6. Visual Basic

Visual Basic merupakan salah satu bahasa pemrograman yang andal dan banyak digunakan oleh pengembang untuk membangun berbagai macam aplikasi *Windows*. *Visual Basic* 2008 atau *Visual Basic* 9 adalah versi terbaru yang telah diluncurkan bersama C#, Visual C++, dan Visual *Web Developer* dalam satu paket *Visual Studio* 2008.

Visual Basic 2008 merupakan aplikasi pemrograman yang menggunakan teknologi .NET *Framework*. Teknologi .NET *Framework* merupakan komponen *windows* yang terintegrasi serta mendukung pembuatan, penggunaan aplikasim dan halaman web. Teknologi .NET *Framework* mempunyai 2 komponen utama, yaitu CLR (*Common Language Runtime*) dan *Class Library*. CLR digunakan untuk menjalankan aplikasi yang berbasis .NET sedangkan *Library* adalah kelas pustaka atau perintah yang digunakan untuk mengembangkan aplikasi. (Wahana Komputer, 2010 : 2).

II.7. SQL Server

Dalam suatu rancangan *database*, data *dictionary* digunakan untuk menjelaskan atau mendeskripsikan kolom-kolom pada masing-masing tabel yang akan dibuat ke dalam *database*. Deskripsi kolom yang dimaksud di sini meliputi tipe data, lebar karakter atau digit, serta keterangan tentang kunci relasi (Budi Raharjo ; 2011 : 59)

Bahasa *query* merupakan bahasa khusus yang digunakan untuk melakukan manipulasi dan menanyakan pertanyaan *(query)* yang berhubungan dengan bahasa

pemrograman, dimana bahasa *query* tidak memiliki kemampuan untuk menyelesaikan banyak masalah seperti bahasa pemrograman pada umumnya. Dalam pemrograman basis data, salah satu bahasa yang harus di kuasai adalah SQL. SQL merupakan bahasa komputer standar yang digunakan untuk berkomunikasi dengan sistem manajemen basis data relasional (RDBMS) (Ema Utami dan Anggi Dwi Hartanto ; 2012 : 63)

II.8. Kamus Data

Sebelum memperoleh defenisi formal basis data, kita akan mencoba memahaminya secara sederhana terlebih dahulu. Istilah basis data tersusun atas dua suku kata, yaitu basis dan data (basis data = basis + data). Dalam sistem bilangan *biner*, kita dapat menuliskan beberapa contoh bilangan sebagai berikut (Edhy Sutanta ; 2011 : 25) :

0 → sama dengan 0 dalam sistem bilangan desimal

1 → sama dengan 1 dalam sistem bilangan desimal

10 → sama dengan 2 dalam sistem bilangan desimal

11 → sama dengan 3 dalam sistem bilangan desimal

100 → sama dengan 4 dalam sistem bilangan desimal

II.9. Normalisasi

Normalisasi diartikan sebagai suatu teknik yang menstrukturkan atau mendekomposisikan data dalam cara-cara tertentu untuk mencegah timbulnya pemasalahan dalam pengolahan data dalam basis data. Permasalahan yang

dimaksud adalah berkaitan dengan penyimpangan-penyimpangan *(anomalies)* yang terjadi akibat adanya kerangkapan data dalam relasi dan in-efisiensi pengolahan. Proses normalisasi menghasilkan relasi yang optimal yaitu :

1. Memiliki struktur record yang konsisten secara logik
2. Memiliki struktur record yang mudah untuk dimengerti
3. Memiliki struktur *record* yang sederhana dalam pemeliharaan
4. Memiliki struktur *record* yang mudah ditampilkan kembali untuk memenuhi kebutuhan pengguna
5. Minimalisasi kerangkapan data guna meningkatkan kinerja sistem (Edhy Sutanta ; 2011 : 174)