BAB II

LANDASAN TEORI

II.1. Data Mining

”Data Mining adalah proses yang mempekerjakan satu atau lebih teknik pembelajaran komputer (*machine learning*) untuk menganalisis dan mengekstraksi pengetahuan (*knowledge*) secara otomatis” (Hermawati, 2009:3).

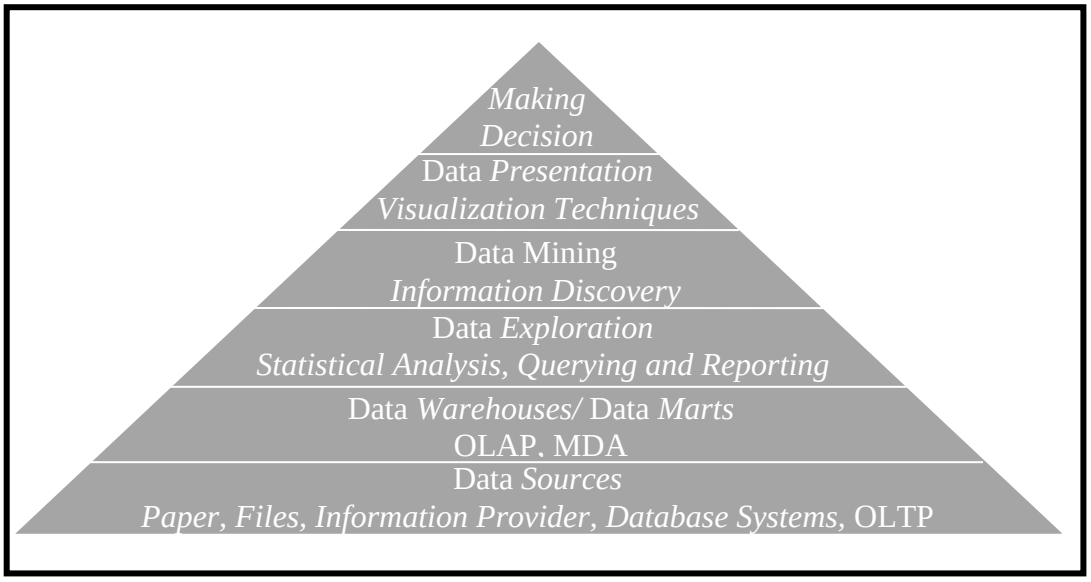
Definisi lain diantaranya adalah pembelajaran berbasis induksi (*induction-based learning*) adalah proses pembentukan definisi-definisi konsep umum yang dilakukan dengan cara mengobservasi contoh-contoh spesifik dari konsep-konsep yang akan dipelajari. *Knowledge Discovery in Databases* (KDD) adalah penerapan metode saintifik pada data mining. Dalam konteks ini, data mining merupakan satu langkah dari proses KDD.

Data mining merupakan proses iteratif dan interaktif untuk menemukan pola atau model baru yang sahih (sempurna), bermanfaat dan dapat dimengerti dalam suatu *database* yang sangat besar (*massive databases*).

- Sahih, yaitu dapat digeneralisasi untuk masa yang akan datang.
- Baru, yaitu apa yang sedang tidak diketahui.
- Bermanfaat, yaitu dapat digunakan untuk melakukan suatu tindakan.
- Iteratif, memerlukan sejumlah proses yang diulang.
- Interaktif, memerlukan interaksi manusia dalam prosesnya.

Data mining berisi pencarian *trend* atau pola yang diinginkan dalam *database* besar untuk membantu pengambilan keputusan di waktu yang akan

datang. Pola-pola ini dikenali oleh perangkat tertentu yang dapat memberikan suatu analisa data yang berguna dan berwawasan yang kemudian dapat dipelajari dengan lebih teliti, yang mungkin saja menggunakan perangkat pendukung keputusan yang lainnya.



Gambar II.1 Data Mining dan Teknologi *Database* Lainnya

(Sumber : Hermawati ; 2009)

Dari gambar di atas terlihat bahwa teknologi data *warehouse* digunakan untuk melakukan OLAP, sedangkan data mining digunakan untuk melakukan *information discovery* yang informasinya lebih ditujukan untuk seorang *Data Analyst* dan *Business Analyst* (dengan ditambah visualisasi tentunya). Dalam prakteknya, data mining juga mengambil data dari data *warehouse*. Hanya saja aplikasi dari data mining lebih khusus dan lebih spesifik dibandingkan OLAP mengingat *database* bukan satu-satunya bidang ilmu yang mempengaruhi data mining, banyak lagi bidang ilmu yang turut memperkaya data mining seperti : *information science* (ilmu informasi), *high performance computing*, visualisasi, *machine learning*, statistik, *neural network* (jaringan syaraf tiruan), pemodelan

matematika, *information retrieval* dan *information extraction* serta pengenalan pola. Bahkan pengolahan citra (*image processing*) juga digunakan dalam rangka melakukan data mining terhadap data *image/ spatial.*

Alasan mengapa melakukan data mining dari sudut pandang komersial karena :

1. Meledaknya volume data yang dihimpun dan disimpan dalam data *warehouse* seperti data *web*, *e-commerce*, penjualan di *departement store*, transaksi bank / *credit card*.
2. Proses komputasi yang dapat diupayakan.
3. Kuatnya tekanan kompetitif untuk dapat menyediakan yang lebih baik, layanan-layanan *custom*-isasi dan informasi sedang menjadi produk yang berarti.

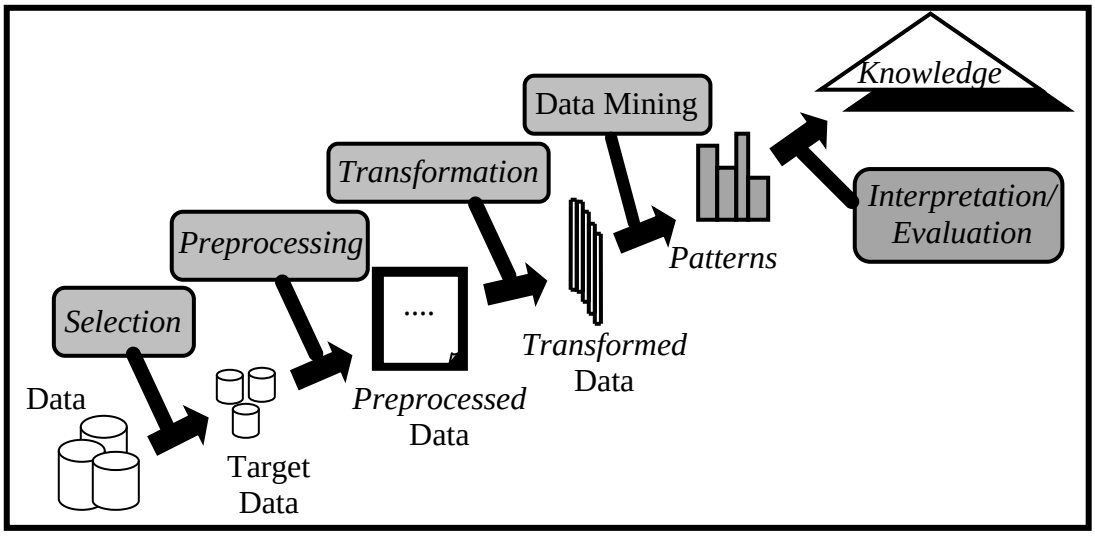
II.1.1. Operasi Data Mining

Operasi data mining menurut sifatnya dibedakan menjadi dua yaitu :

1. Prediksi (*prediction driven*), untuk menjawab pertanyaan apa dan sesuatu yang bersifat transparan. Operasi prediksi digunakan untuk validasi hipotesis, *querying* dan pelaporan, analisis multidimensi, OLAP (*Online Analytic Processing*) serta analisis statistik.
2. Penemuan (*discovery driven*) bersifat transparan dan untuk menjawab pertanyaan "mengapa?". Operasi penemuan digunakan untuk analisis data eksplorasi, pemodelan prediktif, segmentasi *database*, analisis keterkaitan (*link analysis*) dan deteksi deviasi.

Tahapan proses dalam penggunaan data mining yang merupakan proses *Knowledge Discovery in Databases* (KDD) seperti yang terlihat pada gambar II.2 dapat diuraikan sebagai berikut :

- Memahami *domain* aplikasi untuk mengetahui dan menggali pengetahuan awal serta apa sasaran pengguna.
- Membuat target data-set yang meliputi pemilihan data dan fokus pada sub-set data.
- Pembersihan dan transformasi data meliputi eliminasi derau, *outliers*, *missing value* serta pemilihan fitur dan reduksi dimensi.
- Penggunaan algoritma data mining yang terdiri dari asosiasi, sekuensial, klasifikasi, klasterisasi, dll.
- Interpretasi, evaluasi dan visualisasi pola untuk melihat apakah ada sesuatu yang baru dan menarik dan dilakukan iterasi jika diperlukan.



Gambar II.2 Proses KDD

(Sumber : Hermawati ; 2009)

II.1.2. Tantangan Dalam Data Mining

Tantangan dalam data mining meliputi :

1. *Scalability*, yaitu besarnya ukuran basis data yang digunakan.
2. *Dimensionality*, yaitu banyaknya jumlah atribut dalam data yang akan diproses.
3. *Complex and Heterogeneous Data*, yaitu data yang kompleks dan mempunyai variasi yang beragam.
4. Data *Quality*, kualitas data yang akan diproses seperti data yang bersih dari *noise*, *missing value*, dsb.
5. *Data Ownership and Distribution*, yaitu siapa yang memiliki data dan bagaimana distribusinya.
6. *Privacy Preservation*, yaitu menjaga kerahasiaan data yang banyak diterapkan pada data nasabah perbankan.
7. *Streaming Data*, yaitu aliran data itu sendiri.

(Hermawati ; 2009:19)

II.1.3. Teknik Data Mining

Beberapa teknik dan sifat data mining adalah sebagai berikut :

1. *Classification*
2. *Clustering*
3. *Association Rule Discovery*
4. *Sequential Pattern Discovery*
5. *Regression*

II.1.4. *Association Rules*

Mendeteksi kumpulan atribut-atribut yang muncul bersamaan (*co-occur*) dalam frekuensi yang sering dan membentuk sejumlah kaidah dari kumpulan-kumpulan tersebut. Contoh : 90% orang yang berbelanja di suatu supermarket yang membeli roti juga membeli selai dan 60% dari semua orang yang berbelanja membeli keduanya (Hermawati, 2009:17).

Jika diberikan sekumpulan *record* yang masing-masing terdiri dari sejumlah *item* dari kumpulan yang diberikan, akan menghasilkan aturan ketergantungan (*dependency rules*) yang akan memprediksi kejadian dari satu item berdasarkan kejadian item lainnya.

Contoh aplikasi kaidah asosiasi adalah sebagai berikut :

1. *Marketing and Sales Promotion*

   Misalkan diketahui aturan ketergantungan dimana :

   {*Bagels, ...* } → {*Potato Chips*}

   Potato Chips sebagai *consequent*, dapat digunakan untuk menentuka apa yang dapat dilakukan untuk meningkatkan penjualan.

   Bagels in the antecedent, dapat digunakan untuk melihat produk mana yang akan terkena dampak jika toko tersebut tidak lagi menjual *bagels*.

   Bagels in antecedent and Potato Chips in consequent, dapat digunakan untuk melihat produk apa yang harus dijual dengan *bagels* untuk mempromosikan penjualan *potato chips*.

2. *Supermarket Shelf Management*

   Tujuan : Untuk mengenali item-item yang dibeli bersama-sama oleh cukup banyak pelanggan.

Aturan klasik : Jika seorang pelanggan membeli *diaper* dan susu maka dia juga akan membeli *beer*. Sehingga jangan kaget jika menemukan enam *pack beer* yang ditumpuk dekat *diapers*.

3. *Inventory Management*

Tujuan : Seorang pelanggan perusahaan perbaikan peralatan mengharapkan keaslian dari perbaikan produk konsumen dan menjaga pelayanan dengan menggunakan suku cadang yang baik untuk mengurangi jumlah kunjungan ke rumah pelanggan.

Berikut ini akan dijelaskan beberapa hal yang terkait pada *association rules*, antara lain pengertian, algoritma *apriori* dan proses mengubah *association rules*.

II.1.5. Pengertian *Association Rules*

Association rules merupakan salah satu metode yang bertujuan mencari pola yang sering muncul diantara banyak transaksi, dimana setiap transaksi terdiri dari beberapa *item* sehingga metode ini akan mendukung sistem rekomendasi melalui penemuan pola antar *item* dalam transaksi-transaksi yang terjadi. Metodologi dasar analisis asosiasi terbagi menjadi tiga tahap, yaitu :

1. Analisa Pola Frekuensi Tinggi

Tahap ini mencari kombinasi *item* yang memenuhi syarat *minimum* dari nilai *support* dalam *database*. Nilai *support* sebuah *item* diperoleh dengan rumus berikut :

$$\textbf{\textit{Support}} = \frac{\text{Jumlah Transaksi mengandung A}}{\text{Total Transaksi}} \times 100\%$$

Sedangkan nilai *support* dari 2 *item* diperoleh dari 2 rumus berikut :

$$\textbf{\textit{Support}}\ A \cap B = \frac{\text{Jumlah Transaksi mengandung A dan B}}{\text{Total Transaksi}} \times 100\%$$

2. Pembentukan Aturan Assosiatif

Setelah semua pola frekuensi tinggi ditemukan, barulah dicari aturan assosiatif yang memenuhi syarat *minimum* untuk *confidence* dengan menghitung *confidence* aturan assosiatif A_B. Nilai *confidence* dari aturan A_B diperoleh dari rumus berikut :

$$\textbf{\textit{Confidence}} = P(A \mid B) \quad \frac{\text{Jumlah Transaksi Mengandung A dan B}}{\text{Jumlah Transaksi Mengandung A}} \times 100\%$$

Langkah pertama pada *association rule* adalah menghasilkan semua *Item set* yang memungkinkan dengan kemungkinan item *set* yang muncul dengan *m-item* adalah 2*m*. Karena besarnya komputasi untuk menghitung *frequent item set*, yang membandingkan setiap kandidat item *set* dengan setiap transaksi, maka ada beberapa pendekatan untuk mengurangi komputasi tersebut, salah satunya dengan algoritma *apriori*.

II.1.6. Algoritma *Apriori*

Algoritma *apriori* yaitu langkah yang membutuhkan pemrosesan lebih adalah *frequent Item set* berdasarkan sifat *frequent item set yaitu* setiap subset *frequent item set* harus menjadi *frequent item set* (Dana Sulistiyo Kusumo, et al. 2003).

Algoritma *apriori* digunakan untuk mencari *frequent item set* yang memenuhi *minimum support* kemudian mendapatkan *rule* yang memenuhi *minimum confidence* dari *frequent item set* tadi. Algoritma ini mengontrol berkembangnya kandidat item *set* dari hasil *frequent item set* dengan *support-*

based pruning untuk menghilangkan item *set* yang tidak menarik dengan menetapkan *minimum support*. Prinsip dari apriori ini adalah bila item *set* digolongkan sebagai *frequent item set*, yang memiliki *support* lebih dari yang ditetapkan sebelumnya, maka semua subsetnya juga termasuk golongan *frequent item set*, dan sebaliknya.

Cara algoritma ini bekerja adalah algoritma akan menghasilkan kandidat baru dari *k-Item set* dari *frequent item set* pada langkah sebelumnya dan menghitung nilai *support k*-i*tem set* tersebut. *Item set* yang memiliki nilai *support* di bawah dari *minsup* akan dihapus. Algortima berhenti ketika tidak ada lagi *frequent item set* baru yang dihasilkan.

Kedua, dari hasil *frequent item set* tersebut, langkah selanjutnya dihitung *minconf* mengikuti rumus sesuai yang telah ditentukan. Support tidak perlu dilihat lagi, karena *generate frequent item set* didapatkan dari melihat *minsup*-nya. Bila *rule* yang didapatkan memenuhi batasan yang ditentukan dan batasan itu tinggi, maka *rule* tersebut tergolong *strong rules*.

Menurut Santoso (2003) “Prinsip algoritma *apriori* terbagi dalam beberapa tahap” :

1. Kumpulkan jumlah *item* tunggal, dapatkan item besar .
2. Dapatkan *candidate pairs*, hitung => l*arge paris* dari *item-item.*
3. Dapatkan *candidate triplets*, hitung => large *triplets* dari *item-item set*erusnya.
4. Sebagai petunjuk : setiap subset dari sebuah *frequent Item sets* harus menjadi *frequent*.

Gambar dibawah ini menunjukkan bahwa algoritma akan menghasilkan kandidat baru dari 1-*item set* dari *frequent item set* dan menghitung nilai *support 1*-i*tem set* tersebut. Selanjutnya masuk ke 2-i*tem set* dan dihitung lagi *support* 2-i*tem set*. Kemudian i*tem set* yang memiliki nilai *support* di bawah dari *minsup* akan dihapus. Algortima berhenti jika tidak ada lagi *frequent item set* baru yang dihasilkan dan menetapkan *minimum support* nya = 3.

Contoh dari penerapan algoritma *apriori* adalah diilustrasikan digambar berikut :

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Item (1-*Item Set*)

Item set	Count
Bread, Milk	3
Bread, Beer	2
Bread, Diaper	3
Milk, Beer	2
Milk, Diaper	3
Beer, Diaper	3

Pairs (2-*Item Set)*

***Minimum Support* = 3**

Triplets (3-*Item Set*)

Item set	Count
Bread, Milk, Diaper	3
Milk, Diaper, Beer	2

Gambar II.3 Ilustrasi Algoritma *Apriori*

II.1.7. *Rule Generation*

Setelah mendapatkan *frequent item set* menggunakan algoritma apriori, langkah selanjutnya adalah mendapatkan *rule* yang memenuhi *confidence*. Karena *rule* yang dihasilkan berasal dari *frequent item set*, dengan kata lain, dalam

menghitung *rule* menggunakan *confidence*, tidak perlu lagi menghitung *support*-nya karena semua calon *rules* yang dihasilkan telah memenuhi *minimum support* sesuai yang ditentukan. Penghitungan ini juga tidak perlu melakukan perulangan *scanning* pada *database* untuk menghitung *confidence*, cukup dengan mengambil *Item set* dari hasil *support.*

II.2. Pemodelan Sistem

II.2.1. Unified Modelling Language (UML)

Pada perkembangan teknologi perangkat lunak, diperlukan adanya bahasa yang digunakan untuk memodelkan perangkat lunak yang akan dibuat dan perlu adanya standarisasi agar orang diberbagai negara dapat mengerti pemodelan perangkat lunak. Seperti yang kita ketahui bahwa menyatukan banyak kepala untuk menceritakan sebuah ide dengan tujuan untuk memahami hal yang sama tidaklah mudah, oleh karena itu diperlukan sebuah bahasa pemodelan perangkat lunak yang dapat dimengerti oleh banyak orang.

Banyak orang yang telah membuat bahasa pemodelan pembangunan perangkat lunak sesuai dengan teknologi pemrograman yang berkembang pada saat itu, misalnya yang sempat berkembang dan digunakan oleh banyak pihak adalah *Data Flow Diagram* (DFD) untuk memodelkan perangkat lunak yang menggunakan pemrograman prosedural atau struktural, kemudian juga ada *State Transition Diagram* (STD) yang digunakan untuk memodelkan sistem *real time* (waktu nyata).

Pada perkembangan teknik pemrograman berorientasi objek, muncullah sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang

dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modelling Language* (UML). UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasi, menggambarkan, membangun dan dokumentasi dari sistem perangkat lunak. "UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung" (Shalahuddin, M. dan Rosa A.S, 2014:137).

UML hanya berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada metodologi tertentu, meskipun pada kenyataannya UML paling banyak digunakan pada metodologi berorientasi objek. Seperti yang kita ketahui bahwa banyak hal di dunia sistem informasi yang tidak dapat dibakukan, semua tergantung kebutuhan, lingkungan dan konteksnya. Begitu juga dengan perkembangan penggunaan UML bergantung pada *level* abstraksi penggunaannya. Jadi belum tentu pandangan yang berbeda dalam penggunaan UML adalah suatu yang salah, tapi perlu ditelaah dimanakah UML digunakan dan hal apa yang ingin divisualkan.

II.2.2. *Use Case Diagram*

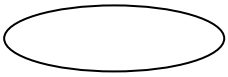
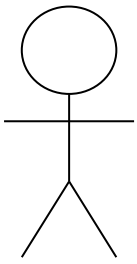
Use case atau diagram use case merupakan pemodelan untuk melakukan (*behavior*) sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu.

Syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu pendefinisian apa yang disebut aktor dan *use case.*

1. Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat diluar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.
2. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor.

Berikut adalah simbol-simbol yang ada pada diagram *use case* :

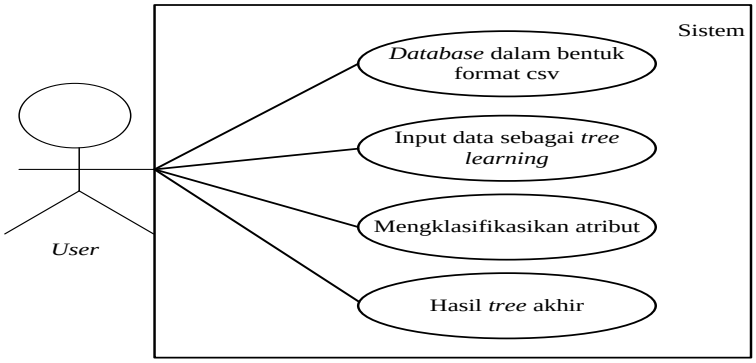
Tabel II.1 Simbol-Simbol *Use Case Diagram*

Simbol	Nama	Keterangan
	*Use Case*	Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor; biasanya dinyatakan dengan menggunakan kata kerja di awal frase nama *use case*.
	*Actor*	Orang, proses, atau sistem lain yang berorientasi dengan sistem informasi yang akan dibuat diluar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal frase nama aktor.
	Association	Komunikasi antar aktor dan *use case* yang berpartisipasi pada *use case* atau *use case* memiliki interaksi dengan aktor.

<<extend>>	*Extend*	Relasi *use case* tambahan ke sebuah *use case* dimana *use case* yang ditambahkan dapat berdiri sendiri walau tanpa *use case* tambahan itu; mirip dengan prinsip *inheritance* pada pemrograman berorientasi objek; biasanya *use case* tambahan memiliki nama depan yang sama dengan *use case* yang ditambahkan.
	*Generalization*	Hubungan generalisasi dan spesialisasi (umum-khusus) antara dua buah *use case* dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya.
<<include>>	*Include*	Relasi *use case* tambahan ke sebuah *use case* dimana *use case* yang ditambahkan memerlukan *use case* ini untuk menjalankan fungsinya atau sebagai syarat dijalankan *use case* ini.

(Sumber : Shalahuddin, M. dan Rosa A.S, 2014:156, Rekayasa Perangkat Lunak)

Berikut ini adalah contoh penggunaan *use case system* kerja pembiayaan PT. Prudential Life Assurance Medan.



Gambar II.4 Contoh Penggunaan *Use Case* Diagram Sistem Kerja

II.2.3. *Activity Diagram*

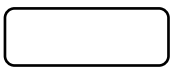
Diagram aktivitas atau *activity diagram* menggambarkan *work flow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak. Yang perlu diperhatikan disini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan *actor*, jadi aktivitas yang dapat dilakukan oleh sistem.

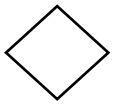
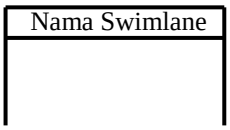
Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut :

1. Rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
2. Urutan atau pengelompokan tampilan dari sistem/ *user interface* dimana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan.
3. Rancangan pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.
4. Rancangan menu yang ditampilkan pada perangkat lunak.

Berikut adalah simbol-simbol yang ada pada diagram aktivitas :

Tabel II.2 Simbol *Activity Diagram*

Simbol	Nama	Keterangan
	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 bertanggung jawab terhadap aktivitas yang terjadi.

(Sumber : Shalahuddin, M. dan Rosa A.S, 2014:162, Rekayasa Perangkat Lunak)

II.2.4. *Class Diagram*

Diagram kelas atau *class diagram* menggambarkan struktur system dari segi pendefenisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi.

a. Atribut merupakan variable-variabel yang dimiliki oleh suatu kelas

b. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

Diagram kelas dibuat agar pembuat program atau *programmer* membuat kelas-kelas sesuai rancangan di dalam diagram kelas agar Antara dokumentasi perancangan dan perangkat lunak sinkron. Banyak berbagai kasu, perancangan kelas yang dibuat tidak sesuaidengan kelas-kelas yang dibuat pada perangkat lunak, sehingga tidaklah ada gunanya lagi sebuah perancangan karena apa yang dirancang dan hasil jadinya tidak sesuai.

Kelas-kelas yang ada pada struktur system harus dapat melakukan fungsi-fungsi-fungsi sesuai dengan kebutuhan system sehingga pembuat perangkat lunak dapat membuat kelas-kelas didalam program perangkat lunak sesuai dengan

perancangan diagram kelas. Susunan struktur kelas yang baik pada diagram kelas sebaiknya memiliki jenis-jenis kelas berikut :

a. Kelas main : kelas yang memiliki fungsi awal dieksekusi ketika system dijalankan.
b. Kelas yang menangani tampilan system (*view*) : kelas yang mendefenisikan dan mengatur tampilan ke pemakai.
c. Kelas yang diambil dari pendefenisian *use case* (*controller*) : kelas yang menangani fungsi-fungsi yang harus ada diambil dari pendefenisian *use case*, ,kelas ini biasanya disebut dengan kelas proses yang menangani proses bisnis pada perangkat lunak.
d. Kelas yang diambil dari pendefenisian data (*model)* : kelas yang digunakan untuk memegang atau membungkus data menjadi sebuah kesatuan yang diambil maupun akan disimpan ke basis data. Semua table yang dibuat di basis data dapat dijadikan kelas, namun untuk table dari hasil ralasi atau atribut mutivalue pada ERD dapat dijadikan kelas tersendiri dapat juga tidak asalkan pengaksesannya dapat dipertanggungjawabkan atau tetap ada didalam perancangan kelas.

II.2.5. *Sequence Diagram*

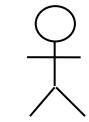
Diagram sekuen menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirmkan dan diterima antar objek. Oleh karean itu untuk menggambar diagram sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode

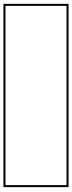
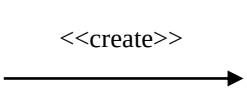
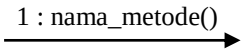
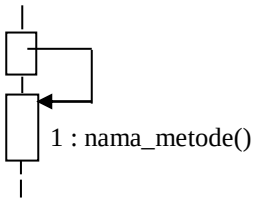
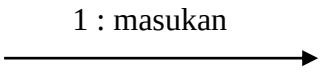
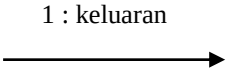
yang dimilik kelas yang diinstansiasi menjadi objek itu. Membuat diagram sekuen juga dibutuhkan untuk melihat scenario yang ada pada *use case*.

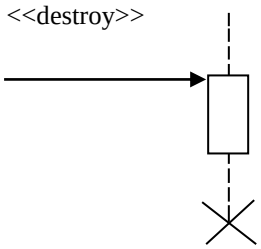
Banyaknya diagram sekuen yang harus digambar adalah minimal sebanyak pendefenisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefenisikan interaksi jalannya pesan sudah dicakup pada diagram sekuen sehingga semakin banyak *use case* yang didefenisikan maka diagram sekuen yang harus dibuat juga semakin banyak.

Berikut adalah simbol-simbol yang ada pada diagram sekuen:

Tabel II.3. Simbol *Sequence Diagram*

Simbol	**Deskripsi**
 Nama aktor Atau Nama aktor Tanpa waktu aktif	Orang, proses, atau sistem lain yanng berinteraksi dengan sistem informasi yang akan dibuat diluar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal frase nama aktor
Garis hidup / *lifeline* 	Menyatakan kehidupan suatu objek
Objek Nama objek ; nama kelas	Menyatakan objek yang berinteraksi pesan

Waktu aktif 	Menyatakan objek dalam keadaan aktif dan berinteraksi, semua yang terhubung dengan waktu aktif ini adalah sebuah tahapan yang dilakukan didalamnya.
Pesan tipe create 	Menyatakan suatu objek membuat objek yang lain, arah panah mengarah pada objek yang dibuat.
Pesan tipe call 	Menyatakan suatu objek memanggil operasi/metode yang ada pada objek lain atau dirinya sendiri,  Arah panah mengarah pada objek yang memiliki operasi/metode, karena ini memanggil operasi/metode maka operasi/metode yanng dipanggil harus ada pada diagram kelas sesuai dengan kelas objek yang berinteraksi.
Pesan tipe send 	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 menghasilkan suatu kembalian ke objek tertentu, arah panah mengarah pada objek yang menerima kembalian

Pesan tipe destroy  	Menyatakan suatu objek mengakhiri hidup objek yang lain, arah panah mengarah pada objek yang diakhri, sebaiknya jika ada create maka ada destroy.

(Sumber : Shalahuddin, M. dan Rosa A.S, 2014:165-167, Rekayasa Perangkat Lunak)