BAB II
TINJAUAN PUSTAKA

II.1. Sistem Pakar

Sistem pakar adalah sistem yang mampu menirukan penalaran seorang pakar agar komputer dapat menyelesaikan masalah seperti yang biasa dilakukan oleh para ahli. Pengetahuan yang disimpan didalam sistem pakar umumnya diambil dari seorang manusia yang pakar dalam masalah tersebut (Nur Anjas Sari : 2013 : 101).

II.1.2. Konsep Dasar Sistem Pakar

Menurut (Kusrini, 2006) ada tiga orang yang terlibat dalam sistem pakar :

1. Pakar adalah orang yang memiliki pengetahuan, khusus, pendapat pengalaman dan metode, serta kemampuan untuk mengaplikasikan keahliannya tersebut guna menyelesaikan masalah.
2. *Knowledge engineer* (Perekayasa Sistem) adalah orang yang membantu pakar dalam menyususun area permasalahan dengan menginterprestasikan dan mengintegrasikan jawaban - jawaban pakar atas pertanyaan yang diajukan, menggambarkan analogi, mengajukan *counter example* dan menerangkan kesulitan - kesulitan konseptual.
3. Pemakai, sistem pakar memiliki beberapa pemakai, yaitu : pemakai bukan pakar, pelajar, pembangun sistem pakar yang ingin meningkatkan dan menambahkan basis pengetahuan, dan pakar (Laila Septiana : 2016 : 2).

II.1.3. Ciri-ciri Sistem Pakar

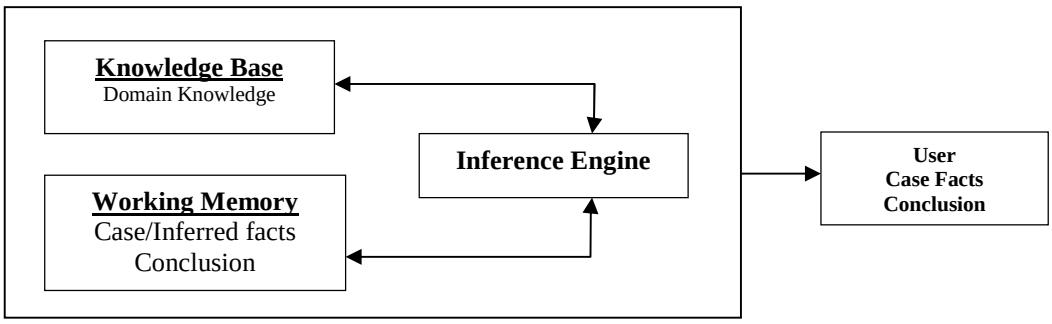
Beberapa ciri-ciri sistem pakar menurut (Kusumadewi, 2003) antara lain : (Laila septian : 2016 : 2)

1. Memiliki fasilitas informasi yang handal
2. Mudah dimodifikasi
3. Dapat digunakan dalam berbagai jenis komputer
4. Memiliki kemampuan untuk belajar beradaptasi

II.1.4. Struktur Sistem Pakar

Menurut Durkin ”Komponen utama pada struktur sistem pakar meliputi Basis Pengetahuan / *Knowledge Base*, Mesin Inferensi / *Inference Engin, Working Memory,* dan Antarmuka Pemakai / *User Interface”*. Struktur sistem pakar dapat ditunjukan pada gambar II.1. dibawah ini :

(Laila septian : 2016 : 2)



Gambar II.1. Struktur Sistem Pakar

II.2. Certainty Factor

Definisi *Certainty Factor* menurut David McAllister, *Certainty Factor* adalah suatu metode untuk membuktikan apakah suatu fakta itu pasti ataukah tidak pasti yang berbentuk metric yang biasanya digunakan dalam sistem pakar. Metode ini sangat cocok untuk sistem pakar yang mendiagnosa sesuatu yang belum pasti.

Certainty Factor didefinisikan sebagai persamaan berikut :

CF (H, E) = MB (H, E) – MD (H, E)

CF (H, E) : *Certainty Factor* dari hipotesis H yang dipengaruhi oleh gejala (*evidence*) E. Besarnya CF berkisar antara -1 sampai 1. Nilai -1 menunjukkan ketidak percayaan mutlak sedangkan nilai 1 menunjukkan kepercayaan mutlak.

MB (H, E) : ukuran kenaikan kepercayaan (*measure of increased belief*) terhadap hipotesis H yang dipengaruhi oleh gejala E.

MD (H, E) : ukuran kenaikan ketidak percayaan (*measure of increased disbelief*) terhadap hipotesis H yang dipengaruhi oleh gejala E.

Bentuk dasar rumus certainty factor, adalah sebuah aturan JIKA E MAKA H seperti ditunjukkan pada persamaan II.1. berikut :

CF (H, e) = CF (E, e) * CF (H, E).....(II.1.)

Dimana :

CF (H, e) : *Certainty Factor* hipotesis yang dipengaruhi oleh *evidence* e.

CF (E, e) : *Certainty Factor evidence* E yang dipengaruhi oleh *evidence* e.

CF (H, E) : *Certainty Factor* hipotesis dengan asumsi *evidence* diketahui

dengan pasti, yaitu ketika CF (E, e) = 1.

Jika semua *evidence* pada *antecedent* diketahui dengan pasti maka persamaannya akan menjadi :

CF (E, e) = CF (H, E)

Dalam aplikasinya, CF (H,E) merupakan nilai kepastian yang diberikan oleh pakar terhadap suatu aturan, sedangkan CF (E,e) merupakan nilai kerpercayaan yang diberikan oleh pengguna terhadap gejala yang dialaminya (Laila septian : 2016 : 4)

II.3. Penyakit Addison

Menurut (Gardner DG *et all* 2007). Penyebab paling umum penyakit Addison adalah perusakan dan atau atrofi dari korteks adrenal. Pada sekitar 70% dari semua kasus, atrofi ini diduga terjadi karena adanya gangguan autoimun. Pada sekitar 20% dari semua kasus, perusakan korteks adrenal disebabkan oleh tuberkulosis. sisa kasus lainnya dapat disebabkan oleh infeksi jamur, seperti stoplasmosis, coccidiomycosis, dan kriptokokosis, yang mempengaruhi glandula adrenal (Said alfin, dkk : 2011 : 78).

Menurut (Liotta EA *et all* 2008). Pada sekitar 75% dari semua pasien, penyakit Addison cenderung sangat bertahap, perlahan - lahan berkembang penyakit. gejala signifikan tidak terjadi sampai sekitar 90% dari korteks adrenal telah dihancurkan. Gejala yang paling umum termasuk kelelahan dan hilangnya energi, penurunan nafsu makan, mual, muntah, diare, sakit perut, penurunan berat badan, lemah otot, pusing ketika berdiri, dehidrasi dan perubahan pigmen kulit (Said alfin, dkk : 2011 : 78).

II.4. Gejala Penyakit Addison

Gejala dari penyakit addison tidak spesifik. Gejala yang muncul biasanya berhubungan dengan kelelahan, kelemahan, anoreksia, nausea, nyeri abdomen, gastroenteritis, diare dan labilitas mood. Pada orang dewasa dengan penyakit Addison dapat dijumpai penurunan berat badan 1 – 15 kg. Kelemahan badan ini disebabkan karena gangguan keseimbangan air dan elektrolit serta gangguan metabolisme karbohidrat dan protein sehingga didapat kelemahan sampai paralisis otot bergaris. Di samping itu, akibat metabolisme protein, terutama pada sel-sel otot menyebabkan otot-otot bergaris atropi, bicaranya lemah. Gejala kelemahan otot ini berkurang setelah pemberian cairan, garam serta *kortikosteroid* (Said alfin, dkk : 2011 : 79).

II.5. *Unified Modeling language* (UML)

Unified Modeling Language adalah sebuah bahasa yang diterima dan digunakan oleh *software developer* dan *software* analyst sebagai suatu bahasa yang cocok untuk merepresentasikan grafik dari suatu relasi antar entitas - entitas *software* (Gornik, 2003). Dengan menggunakan UML, tim pengembang *software* akan mempunyai banyak keuntungan, seperti memudahkan komunikasi dengan sesama anggota tim tentang *software* apa yang akan dibuat, memudahkan integrasi ke dalam area pengerjaan *software* karena bahasa ini berbasiskan meta-models dimana meta - models bisa mendefinisikan proses - proses untuk mengkonstruksikan konsep - konsep yang ada. UML juga menggunakan formatur *input* dan *output* yang sudah mempunyai bentuk standar yaitu XML *Meta data*

Interchange (XMI), menggunakan aplikasi dan pemodelan data yang *universal*, merepresentasikan dari tahap analisis ke implementasi lalu ke *deployment* yang terpadu, dan mendeskripsikan keutuhan tentang spesifikasi *software* (Edgar Winata : 2013 : 37).

II.5.1. *Use Case Model*

Use Case Model adalah teknik pemodelan untuk mendapatkan *functional requirement* dari sebuah sistem, menggambarkan interaksi antara pengguna dan sistem, menjelaskan secara naratif bagaimana sistem akan digunakan, menggunakan skenario untuk menjelaskan setiap aktivitas yang mungkin terjadi. Ada beberapa bagian didalam use case model (Edgar Winata : 2013 : 38).

Gambar II.2. Use Case pada Use Case Diagram

Gambar II.3. Actor pada Use Case Diagram

1. *Use Case*, untuk mengetahui action atau prosedur apa yang ada didalam sistem.
2. *Actor*, siapa saja yang terlibat dalam action tersebut.
3. *Relationship*, bagaimana actions saling berelasi satu sama lain didalam sistem.

II.5.2. *Class Diagram*

Class Diagram merupakan diagram paling umum yang dijumpai dalam pemodelan berbasis UML. Didalam Class diagram terdapat class dan interface beserta atribut-atribut dan operasinya, relasi yang terjadi antar objek, *constraint* terhadap objek-objek yang saling berhubungan dan *inheritance* untuk organisasi class yang lebih baik. *Class Diagram* juga terdapat *static view* dari elemen pembangun sistem. Pada intinya *Class Diagram* mampu membantu proses pembuatan sistem dengan memanfaatkan konsep *forward* ataupun *reverse engineering* (Edgar Winata : 2013 : 38).

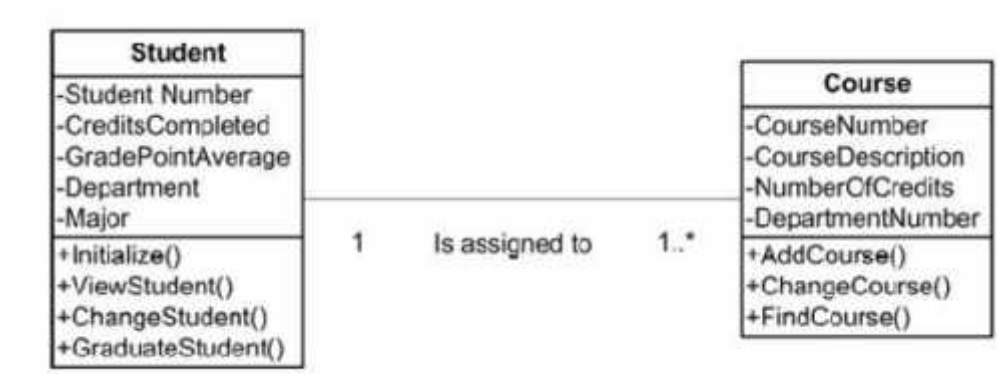
Class diagram mempunyai 2 komponen penting, yaitu :

1. *Structural*, yaitu ciri pembeda objek.
2. *Behavioral,* yaitu tingkah laku atau kegiatan yang mampu dilakukan oleh objek.

Berbagai simbol yang hadir didalam class diagram antara lain adalah :

1. Class, yang berfungsi untuk merepresentasikan tipe dari data yang dimilikinya. *Class Diagram* dapat ditampilkan dengan menunjukkan atribut dan operasi yang dimilikinya atau hanya menunjukkan nama *class* - nya saja. Dapat juga kita tuliskan nama *class* dengan atributnya saja atau nama *class* dengan operasinya.
2. *Attribute*, merupakan data yang terdapat didalam *class* dan *instance* - nya dengan operator.
3. *Operation*, berfungsi untuk merepresentasikan fungsi - fungsi yang ditampilkan oleh class dan instance-nya dengan operator.

4. *Association*, digunakan untuk menunjukkan bagaimana dua *class* berhubungan satu sama lainnya. *Association* ditunjukkan dengan sebuah garis yang terletak diantara dua *class*. Didalam setiap association terdapat *multiplicity*, yaitu simbol yang mengindikasikan berapa banyak instance dari *class* pada ujung *association* yang satu dengan *instance class* di ujung *association* lainnya.
5. *Generalizations*, berfungsi untuk mengelompokkan *class* ke dalam *hirarki inheritance.*
6. *Aggregation*, merupakan bentuk khusus dari association yang merepresentasikan hubungan "*part* - *whole*". Bagian "*whole*" dari hubungan ini sering disebut dengan *assembly* atau *aggregate. Class* yang satu dapat dikatakan merupakan bagian dari *class* yang lain yang ikut membentuk *class* tersebut.
7. *Composition*, merupakan jenis *aggregation* yang lebih kuat diantara dua *class* yang memiliki *association* dimana jika *whole* ditiadakan, maka *part* - nya juga ikut ditiadakan. Berbeda dengan *aggregation*, *part* akan tetap bisa berdiri sendiri meskipun bagian *whole* - nya ditiadakan.
8. Penggunaan *operator* (+) dalam *class* diagram diartikan dengan *public*, *operato*r (-) diartikan *private*, dan *operator* (#) diartikan *protected.*



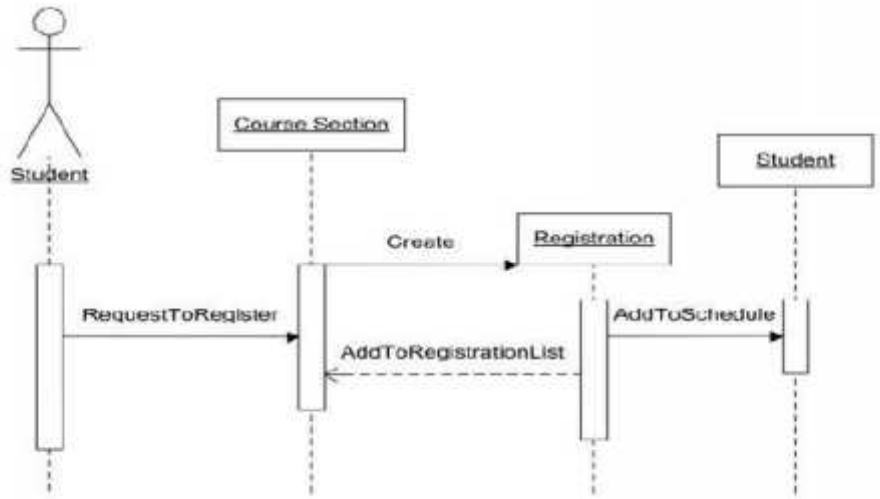
Gambar II.4. Contoh Class Diagram

II.5.3. *Sequence Diagram*

Menjelaskan interaksi obyek-obyek yang saling berkolaborasi (berhubungan), mirip dengan *activity diagram* yaitu menggambarkan alur kejadian sebuah aktivitas tetapi lebih detil dalam menggambarkan aliran data termasuk data atau behaviour yang dikirimkan atau diterima namun kurang mampu menjelaskan detil dari sebuah algoritma (Edgar Winata : 2013 : 39).

Dalam *sequence diagram* terdapat beberapa bagian.

1. *Participant*, yaitu objek yang terkait dengan sebuah urutan proses.
2. *Lifeline*, menggambarkan daur hidup sebuah objek.
3. *Activation*, suatu titik waktu dimana sebuah objek mulai berpartisipasi dalam sebuah *sequence*.
4. *Time*, elemen paling penting dalam *sequence diagram* yang konteksnya adalah urutan, bukan durasi.
5. *Return*, suatu hasil kembalian sebuah operasi. Operasi mengembalikan hasil tetapi boleh tidak ditulis jika tidak ada perbedaan dengan *Getter* - nya.



Gambar II.5. Contoh Sequence Diagram

II.5.4. *Activity Diagram*

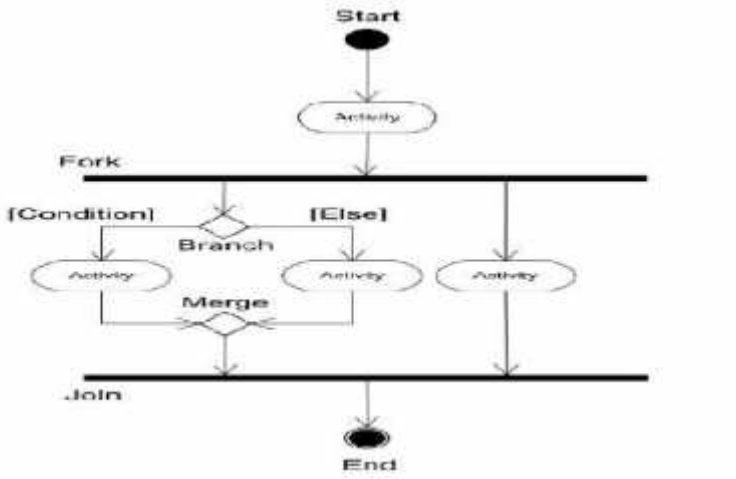
Teknik untuk menjelaskan *business process procedural logic*, dan *work flow*. Bisa dipakai untuk menjelaskan teks *Use Case* dalam notasi grafis dengan menggunakan notasi yang mirip *flow chart*, meskipun terdapat sedikit perbedaan notasi (Edgar Winata : 2013 : 39).

Dalam *Activity Diagram* terdapat beberapa bagian.

1. *Nodes*, menandakan initial dan final node, final node boleh lebih dari 1.
2. *Activity*, aktivitas sistem dapat berupa aktivitas fisik juga bagi user.
3. *Flow / edge*, arah sebuah proses.
4. *Fork*, awal sebuah proses paralel.
5. *Join* akhir proses paralel.
6. *Condition*, kondisi yang dituliskan dalam bentuk teks
7. *Decision*, implementasi if dan then.

8. *Merge*, penyatuan beberapa flow.

9. *Partition*, siapa atau apa yang menjalankan aktivitas.



Gambar II.6. Contoh *Activity Diagram*

II.6. *Microsoft Visual Basic 2010*

Microsoft Visual Basic merupakan sebuah bahasa pemrograman yang menawarkan IDE (*Integrated Development Environment*) visual untuk membuat program perangkat lunak berbasis sistem operasi *Microsoft Windows* dengan menggunakan model pemrograman. *Visual Basic* merupakan turunan bahasa pemrograman *BASIC* yang menawarkan pengembangan perangkat lunak berbasis grafik dengan cepat, beberapa bahasa *script* seperti VBA (*Visual Basic for Applications)* dan VBScript (*Visual Basic Scripting Edition*), mirip seperti halnya *Visual Basic*, tetapi cara kerjanya yang berbeda. Para *programmer* dapat membangun aplikasi dengan menggunakan komponen yang disediakan oleh *microsoft visual basic*. Program - program yang ditulis dengan *visual basic* juga

dapat menggunakan *windows API*, tapi membutuhkan deklarasi fungsi luar tambahan (Anggoro S Pramudyo1, dkk : 2013 : 39).

II.8. SQL *Server 2008*

SQL (*Structured Query Language*) adalah sebuah bahasa yang dipergunakan untuk mengakses data dalam basis data relasional. Bahasa ini secara *de facto* merupakan bahasa *standart* yang digunakan dalam manajemen basis data relasional. Saat ini hampir semua *server* basis data yang ada mendukung bahasa ini untuk melakukan manajemen datanya. SQL terdiri dari dua bahasa, yaitu *Data Definition Language* (DDL). Dan *Data Manipulation Language* (DML). Implementasi DDL dan DML berbeda untuk tiap sistem manajemen basis data (SMBD). Namun secara umum implementasi setiap bahasa ini memiliki bentuk standar yang ditetapkan oleh ANSI (Adelia,dkk : 2011 : 115).