BAB II

LANDASAN TEORI

2.1. Rancang Bangun Perangkat Lunak

Rekayasa perangkat lunak merupakan suatu disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal yaitu analisa kebutuhan pengguna, menentukan spesifikasi dari kebutuhan pengguna, disain, pengkodean, pengujian sampai pemeliharaan sistem setelah digunakan (Mulyanto, 2008).

Dalam merancang dan membangun aplikasi *Expert System Manager*, *Flowchart* digunakan untuk menggambarkan alur dari proses-proses yang ada. *Flowchart* merupakan suatu diagram yang menunjukkan arus data dan urutan operasi dalam suatu sistem (Bodnar, 2004). *Flowchart* terbagi menjadi tiga jenis, yaitu (Romney, 2003).

- *Document Flowchart*, mengilustrasikan arus dokumen dan informasi di antara bidang tanggung jawab dalam organisasi.
- *System Flowchart*, menunjukkan hubungan antara *input*, proses dan output dari suatu sistem informasi.
- *Program Flowchart*, mengilustrasikan urutan proses logis yang dilaksanakan komputer dalam menjalankan suatu program

Setelah membuat *flowchart*, DFD dibuat untuk menggambarkan alur dari data-data yang dikirim atau diterima oleh sistem. DFD merupakan *modeling* data yang berbasis aliran data yang paling banyak

digunakan saat ini. DFD memperlihatkan *input-process-output* dari sebuah sistem. Itu merupakan bentuk dari obyek data yang masuk ke dalam perangkat lunak, dirubah oleh elemen proses dan data hasil transformasi yang keluar dari perangkat lunak. Objek data dilambangkan dengan panah berlabel dan proses dilambangkan dengan bentuk bulat. DFD direpresentasikan secara hirarkis. Model *data flow* pertama merepresentasikan sistem secara keseluruhan (biasa disebut level 0 DFD atau *context diagram*). Diagram selanjutnya yang memperjelas *context diagram,* memberikan detail yang semakin jelas di tiap tingkat berikutnya (Pressman, 2005).

Kemudian *Entity Relationship Diagram* (ERD) dibuat untuk menjelaskan hubungan antar tabel dalam basis data berdasarkan objek-objek dasar data yang memiliki hubungan antar relasi. ERD digunakan memodelkan struktur data dan hubungan antar data, untuk menggambarkannya digunakan beberapa notasi dan symbol. Elemen-elemen ERD yaitu (Mulyadi, 2011).

- Entitas, merupakan objek yang mewakili sesuatu yang nyata dan dapat dibedakan dari sesuatu yang lain. Simbol dari entitas ini biasanya digambarkan dengan persegi panjang.
- Atribut, dimiliki oleh setiap entitas yang berfungsi untuk mendeskripsikan karakteristik dari entitas tersebut. Simbol dari atribut ini biasanya digambarkan dengan elips.

- Hubungan relasi, adalah hubungan alamiah yang terjadi antara entitas. Pada umumnya relasi diberi nama dengan kata kerja dasar sehingga memudahkan untuk melakukan pembacaan relasinya.
- Kardinalitas relasi, menunjukkan jumlah maksimum tupelo yang dapat berelasi dengan entitas yang lainnya. Dari sejumlah kemungkinan banyaknya hubungan yang terjadi dari entitas, kardinalitas relasi merujuk kepada hubungan maksimum yang terjadi dari entitas yang satu ke entitas yang lainnya dan begitu juga sebaliknya.

Dengan menggunakan *Entity Relationship Diagram* yang sudah dibuat, maka dibentuklah database dengan menggunakan *My Structured Query Languange* (*MySQL)*. *MySQL* merupakan *software database open source* yang paling popular didunia. Dengan kehandalan, kecepatan dan kemudahan penggunaannya, *MySQL* menjadi pilihan utama bagi banyak pengembang *software* dan aplikasi baik di *platform* web maupun *desktop* (Solichin, 2010).

Proses *coding* dilakukan dengan menggunakan bahasa pemrograman *C#.NET*. Bahasa pemrograman C# dikembangkan oleh Microsoft sebagai bahasa yang sederhana, modern, *general-purpose*, dan berorientasi objek. Pengembangan bahasa C# sangat dipengaruhi oleh bahasa pemrograman terdahulu, terutama C++, Delphi, dan Java. Kehadiran C# memberikan suntikan optimisme bagi para *programmer*

untuk dapat mengembangkan aplikasi yang berdaya guna dengan lebih cepat dan lebih mudah (Cybertron Solution, 2009).

2.2. Sistem Pakar

Sistem pakar adalah suatu program berbasis pengetahuan (*knowledge base*) yang menyediakan solusi-solusi dengan kualitas pakar untuk permasalahan-permasalahan dalam suatu bidang yang spesifik. Sistem pakar merupakan cabang dari ilmu kecerdasan buatan (*Artificial Intelligence*). Jenis program ini pertama kali dikembangkan oleh periset kecerdasan buatan pada dasawarsa 1960-an dan 1970-an dan diterapkan secara komersial selama 1980-an. Bentuk umum sistem pakar adalah suatu program yang dibuat berdasarkan suatu set aturan yang menganalisis informasi (biasanya diberikan oleh pengguna sistem) mengenai suatu kelas masalah spesifik serta analisis dari masalah tersebut. Tergantung dari desainnya, sistem pakar juga mampu merekomendasikan suatu rangkaian tindakan pengguna untuk dapat menerapkan koreksi. Sistem ini memanfaatkan kapabilitas penalaran untuk mencapai suatu kesimpulan (Setiawan, 1993). Tujuan dari sistem pakar sebenarnya bukan untuk menggantikan peran manusia, tetapi untuk mensubstitusikan pengetahuan dan pengalaman pakar ke dalam bentuk sistem sehingga dapat digunakan oleh orang banyak (Arisudana, 2010).

Berdasarkan dari pemecahan masalah (*problem solving tasks*), area permasalahan sistem pakar dapat dikategorikan menjadi beberapa tipe, yaitu (Anita dan Muhammad, 2006).

1. Kontrol (*Control*)

   Digunakan untuk mengontrol dan mengatur tingkah laku sebuah sistem tertentu. Misalnya pengontrol mesin-mesin manufaktur atau treatment untuk seorang pasien di rumah sakit.

2. Desain (*Design*)

   Digunakan untuk menentukan konfigurasi komponen-komponen sistem yang cocok dengan tujuan-tujuan kinerja tertentu yang memenuhi kendala-kendala tertentu. Misalnya mendesain sebuah komputer dengan batasan-batasan yang didefinisikan oleh *user*, seperti batasan harga, kecepatan, kemampuan multimedia dan lainnya.

3. Diagnosa (*Diagnosis*)

   Digunakan untuk mendeteksi kerusakan di dalam sebuah sistem, dengan cara mengobservasi semua informasi yang didapat dari sistem tersebut. Misalnya diagnosis penyakit seorang pasien berdasarkan atas gejala yang timbul.

4. Instruksi (*Instruction*)

   Digunakan untuk mendeteksi dan memperbaiki perilaku sistem dalam memahami bidang informasi tertentu. Misalnya petunjuk

belajar seorang siswa terhadap topik tertentu yang dipelajari, *debugging*.

5. Interpretasi (*Interpretation*)

   Digunakan untuk menghasilkan sebuah pemahaman terhadap suatu situasi tertentu berdasarkan atas informasi yang dihasilkan dari sebuah situasi yang spesifik. Biasanya informasi-informasi tersebut berasal dari sensor, *instrument*, hasil tes dan lainnya. Misalnya monitor sensor mesin, sistem pencitraan atau *speech analysis results.*

6. Pengamatan (*Monitoring*)

   Digunakan untuk memonitor dan membandingkan tingkah laku sebuah sistem dengan sistem yang diharapkan. Misalnya *Computer Aided Monitoring System.*

7. Perencanaan (*Planning*)

   Digunakan untuk merencanakan serangkaian tindakan yang dapat mencapai sejumlah tujuan dengan kondisi awal tertentu. Misalnya merencanakan beberapa tugas yang berbeda untuk sebuah robot dalam industri manufkatur.

8. Proyeksi (*Projection*)

   Digunakan untuk meramalkan/memprediksikan aksi yang perlu dilakukan di masa depan berdasarkan atas informasi yang ada dan model-model yang telah diberikan. Misalnya prediksi ekonomi, prediksi lalu lintas.

9. *Debugging* dan *repair*

   Digunakan untuk menentukan dan mengimplementasikan cara-cara untuk mengatasi malfungsi. Misalnya *troubleshooting* pada berbagai macam sistem.

10. Seleksi (*Selection*)

    Digunakan untuk dapat mengidentifikasi pilihan terbaik dari suatu daftar kemungkinan atas dasar informasi yang didapat pada suatu kondisi tertentu.

11. Simulasi (*Simulation*)

    Digunakan untuk memodelkan sebuah proses atau interaksi sistem pada bermacam-macam kondisi. Misalnya pembelajaran bagi operator sebuah sistem sebelum diterjunkan langsung pada kenyataan.

Gambar 2.1 Kriteria Sistem Pakar
(Sumber: Gonzales, 1993)

Sistem pakar memiliki 3 kriteria, yaitu (Gonzales, 1993).

1. User Interface

User Interface merupakan interaksi antara seorang pakar dengan *user*, dimana *user* memasukkan *input* dan akan menerima *output*. *User Interface* berfungsi untuk membantu *user* dalam menyimpan informasi baru ke dalam *knowledge base* sistem pakar, menampilkan fasilitas

penjelasan sistem dan menuntun pemakai agar mengerti apa yang harus dilakukan terhadap sistem. Oleh karena itu, *user interface* haruslah memiliki tampilan yang interaktif dan komunikatif.

2. **Inference Engine**

   Inferensi adalah kesimpulan logis berdasarkan informasi yang tersedia. Dalam sistem pakar, proses inferensi dilakukan dalam suatu modul yang disebut *inference engine*. ”Otak” dari sistem pakar adalah *inference engine*, yang dikenal juga sebagai struktur control atau penerjemah aturan (dalam sistem pakar berbasis-aturan). *Inference engine* adalah bagian dari sistem pakar yang melakukan penalaran dengan menggunakan isi daftar rule berdasarkan urutan dan pola tertentu. Selama proses konsultasi antara *user* dengan sistem, *inference engine* menguji *rule* satu demi satu sampai kondisi *rule* itu benar.

3. **Knowledge Base**

   Knowledge base merupakan inti program sistem pakar dan representasi pengetahuan atau keahlian dari seorang pakar. *Knowledge base* dapat direpresentasikan dalam berbagai macam bentuk, salah satunya adalah dalam bentuk sistem berbasis aturan (*rule-based system*). Pada sistem berbasis aturan, setiap kasus akan diproses berdasarkan fakta yang ada, baru kemudian dapat diambil suatu kesimpulan. Apabila terjadi suatu kasus yang belum pernah ada, maka sistem pakar tidak mampu mengambil kesimpulan dari kasus tersebut. Untuk itu diperlukan

penambahan fakta baru untuk *knowledge base*. Berdasarkan dari *knowledge base* yang dimilikinya, sistem pakar dapat digolongkan menjadi 5 tingkatan, yaitu (Lydiawati, 2004).

a. Tingkat 1
   - Menggunakan *internal knowledge base.*
   - Dapat melakukan penambahan data dan membuat laporan data yang ada pada *knowledge base.*
   - Menggunakan metode *forward chaining* atau metode *backward chaining* untuk menyelesaikan suatu kasus.
   - Mempu menampilkan hasil kerja dari *inference engine* melalui *user interface.*

b. Tingkat 2
   - Menggunakan *external knowledge base*.
   - Mampu melakukan perhitungan matematis.
   - Mampu memilih metode *forward chaining* atau metode *backward chaining* yang akan digunakan untuk menyelesaikan suatu kasus.

c. Tingkat 3
   - Mampu melakukan ekspor atau impor suatu *knowledge base.*
   - Mampu melakukan perhitungan matematis secara dinamis.

d. Tingkat 4
   - Dapat dioperasikan pada berbagai sistem operasi.

e. Tingkat 5
   - Mampu merubah atau menambah isi *knowledge base* secara otomatis dan mempelajari suatu pola baru dari beberapa kasus yang pernah dialaminya.

Selain 3 kriteria dari sistem pakar, terdapat juga *working memory* yang merupakan bagian dari sistem pakar yang berisi fakta-fakta dari suatu

masalah yang ditemukan selama proses konsultasi. *Working memory* berisi semua informasi mengenai suatu masalah baik yang itu disediakan oleh user maupun dari kesimpulan yang diperoleh sistem.

2.3. Metode Forward Chaining

Metode *forward chaining* adalah suatu metode dari mesin inferensi untuk memulai penalaran atau pelacakan suatu data dari fakta-fakta yang ada menuju suatu kesimpulan. Dalam *forward chaining*, *rules* diuji satu demi satu dalam urutan tertentu sesuai dengan *rule base* atau urutan yang ditentukan oleh *user*. Saat tiap *rule* diuji, sistem pakar akan memvalidasi apakah kondisinya benar atau salah. Jika kondisinya benar, maka *rule* tersebut akan disimpan dan *rule* berikutnya diuji. Sebaliknya apabila kondisinya salah, maka *rule* itu tidak akan disimpan dan *rule* berikutnya diuji. Proses ini berulang sampai seluruh *rule* teruji dengan berbagai kondisi dan didapatkan hasil akhirnya (Wijaya, 2006).

Pada aplikasi *forward chaining* sederhana, *inference engine* memilih *rule-rule* dimana bagian premisnya cocok dengan informasi yang ada pada *working memory*. Pertama-tama, sistem memperoleh informasi masalah dari *user* dan menyimpannya dalam *working memory*. Kemudian *inference engine* akan mencari *rules* pada beberapa urutan yang telah ditentukan oleh sistem, dimana premis-premisnya cocok dengan yang terdapat di dalam *working memory*. Jika *rule* ditemukan, maka kesimpulan dari *rule* akan dimasukkan ke dalam *working memory* lalu berlanjut dan

memeriksa *rules* lagi untuk mencari kecocokan yang baru. Proses ini akan berlanjut hingga tidak ditemukan lagi adanya kecocokan. Dalam hal ini, *working memory* berisi informasi yang didapat dari *user* dan kesimpulan yang didapat dari sistem (Lydiawati, 2004).

Kelebihan utama dari *forward chaining*, yaitu (Durkin, 1994).

1) Metode ini bekerja dengan baik ketika problem bermula dari mengumpulkan/menyatukan informasi lalu kemudian mencari kesimpulan apa yang dapat diambil dari informasi tersebut.
2) Merupakan pendekatan paling sempurna untuk beberapa tipe dari *problem solving tasks*, yaitu *planning, monitoring, control* dan *interpretation.*

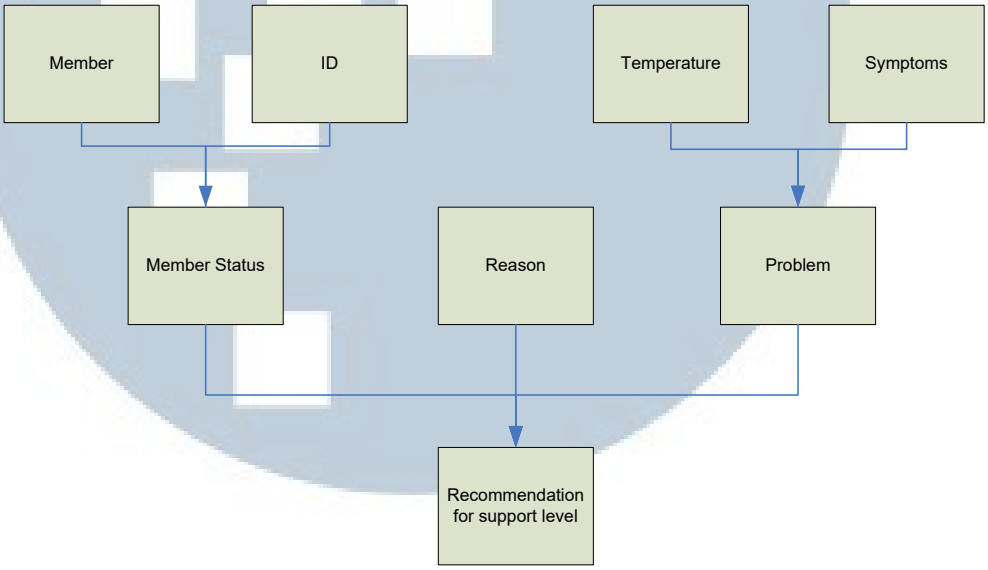
Sedangkan kelemahan dari *forward chaining*, yaitu.

1) Kemungkinan tidak adanya cara untuk mengenali dimana beberapa fakta lebih penting dari fakta lainnya.
2) Sistem bisa saja menanyakan pertanyaan yang tidak berhubungan. Walaupun jawaban dari pertanyaan tersebut penting, namun hal ini akan membingungkan *user* umum untuk menjawab pertanyaan pada subjek yang tidak berhubungan.

2.4. Rule-Based System

Rule-based system adalah sebuah program yang menggunakan aturan IF-THEN, menggunakan modus ponens sebagai dasar untuk memanipulasi aturan. Langkah awal dalam menerjemahkan suatu bidang

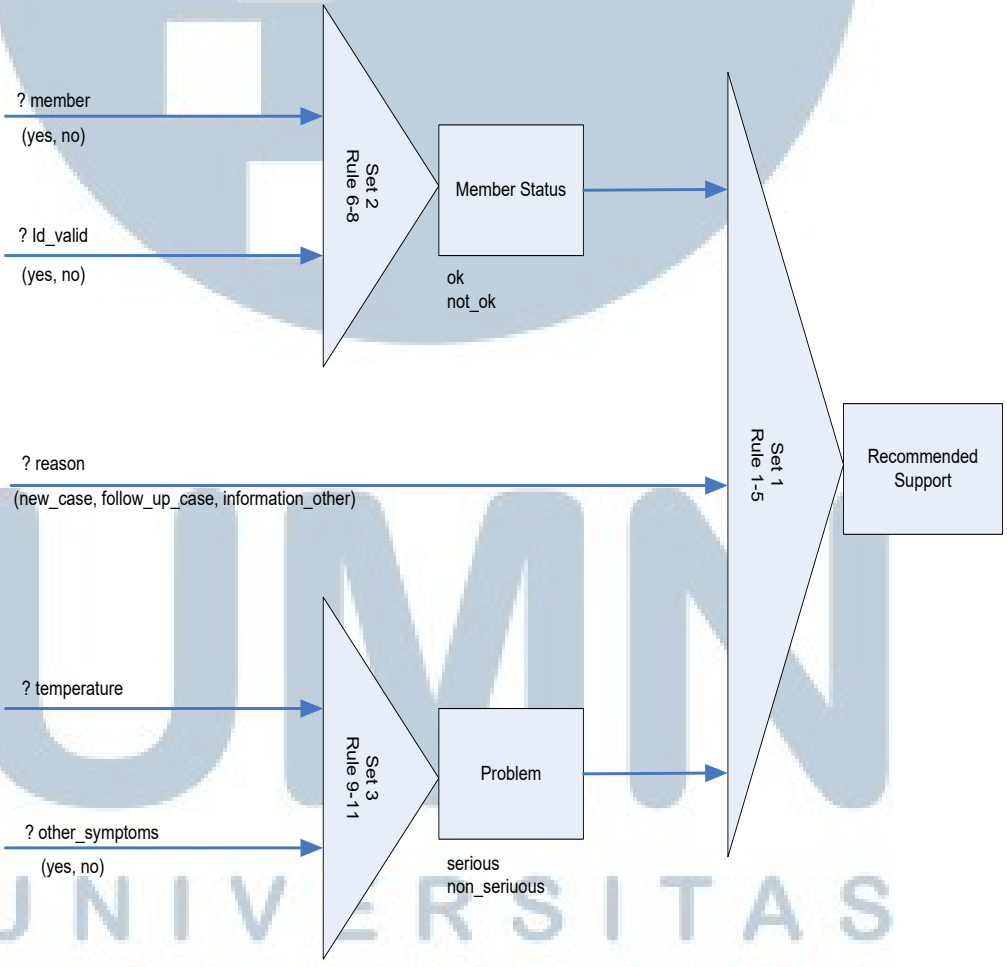
ilmu ke dalam sistem berbasis aturan adalah membuat *block diagram*. *Block diagram* merupakan susunan dari aturan-aturan yang terdapat di dalam sebuah bidang ilmu. Dengan membuat *block diagram* di dalam sistem pakar, maka dapat diketahui urutan kerja sistem dalam mencari keputusan (Dologite, 1993).



Gambar 2.2 Contoh *Block Diagram* Sistem Berbasis Aturan
(Sumber: Dologite, 1993)

Setelah itu, ubah *block diagram* ke dalam *dependency diagram*. *Dependency diagram* adalah suatu relasi yang menunjukkan hubungan atau ketergantungan antara *input* jawaban, aturan-aturan dan nilai-nilai yang direkomendasikan ke dalam sistem berbasis pengetahuan. Dari tiap bagian yang memberikan sebuah value kepada proses pengambilan keputusan, dibuat sebuah diagram berbentuk segitiga dan sebuah diagram berbentuk persegi panjang di sebelahnya. Dalam kasus ini, bagian-bagian tersebut adalah *Member Status*, *Reason* dan *Problem*. Karena bagian

Reason berhubungan langsung dengan proses pengambilan keputusan, maka diagramnya dapat dihilangkan. Pada setiap segitiga, dapat digambarkan garis panah menuju segitiga tersebut dan di atas garis tersebut dituliskan variabel-variabel yang mempengaruhi hasil beserta kemungkinan *values* yang dimiliki variabel tersebut yang ditulis di bagian bawah garis. Pada setiap persegi panjang juga dituliskan kemungkinan *values* yang dimiliki bagian tersebut (Dologite, 1993).



Gambar 2.3 Contoh *Dependency Diagram* Sistem Berbasis Aturan
(Sumber: Dologite, 1993)

Decision table dalam sistem berbasis aturan diperlukan sebagai tabel yang menyajikan nilai-nilai pada hasil rekomendasi sistem berbasis pengetahuan. Proses pembuatan *decision table* berawal dari perencanaan jumlah baris yang diperlukan dalam tabel dengan cara mengalikan jumlah *value* yang ada pada setiap kondisi, yang pada *dependency diagram* dinotasikan dalam bentuk notasi segitiga. Langkah selanjutnya adalah membuat tabel, dimulai dengan membuat tabel kosong berisi nama kolom dan nomor baris. Nomor baris menjadi nomor *rule* pada *knowledge base* dan nama kolom berisi nama kondisi. Kemudian tambahkan satu kolom pada tabel yang berisi kemungkinan-kemungkinan yang bisa terjadi dari kombinasi antar *value*. Setelah itu setiap *value* diisi ke dalam tabel. Pengisian *value* tersebut diatur sedemikian rupa sehingga *value* yang sama berkumpul pada bagian tertentu. Pada kolom sebelahnya, *value* yang sama berkumpul dan berulang pada bagian dari kolom sebelumnya dan seterusnya sehingga setiap baris tidak ada kombinasi *value* yang sama (Dologite, 1993).

Tabel 2.1 Contoh *Decision Table* Sistem Berbasis Aturan
(Sumber: Dologite, 1993)

Rule	**Member Status**	**Reason**	**Problem**	**Concluding Recommendation for support level**
A1	ok	new-case	serious	level-1
A2	ok	new-case	non- serious	level-2
A3	ok	follow-up-case	serious	level-1

Tabel 2.1 Contoh *Decision Table* Sistem Berbasis Aturan (Lanjutan)
(Sumber: Dologite, 1993)

Rule	Member Status	Reason	Problem	Concluding Recommendation for support level
A4	ok	follow-up-case	non- serious	level-3
A5	ok	information-other	serious	information-other
A6	ok	information-other	non- serious	information-other
A7	not-ok	new-case	serious	non-member
A8	not-ok	new-case	non- serious	non-member
A9	not-ok	follow-up-case	serious	non-member
A10	not-ok	follow-up-case	non- serious	non-member
A11	not-ok	information-other	serious	non-member
A12	not-ok	information-other	non- serious	non-member

Verifikasi merupakan sekumpulan aktifitas yang memastikan suatu sistem apakah telah berlaku dalam kondisi yang ditetapkan. Verifikasi itu sendiri terdiri dari dua proses, pertama memeriksa pelaksanaan sistem, kedua memeriksa konsistensi dan kelengkapan dari basis pengetahuan. Verifikasi dijalankan ketika ada penambahan atau perubahan pada *rule*, karena *rule* tersebut sudah ada pada sistem. Tujuan verifikasi adalah untuk memastikan adanya kecocokan antara sistem dengan apa yang dikerjakan sistem dan juga untuk memastikan sistem itu bebas dari kesalahan (Arisudana, 2010).

Berikut ini adalah beberapa metode pemeriksaan *rule* dalam sistem berbasis pengetahuan (Gonzales, 1993).

1. *Redundant rules*

   Redundant rules terjadi jika dua *rule* atau lebih mempunyai *premise* dan *conclusion* yang sama.

   Contoh:

   Rule 1: *if the humidity is high and the temperature is hot*
   Then there will be thunderstorms

   Rule 2: *if the temperature is hot and humidity is high*
   Then there will be thunderstorms

2. *Conflicting rules*

   Conflicting rules terjadi jika dia buah *rule* atau lebih mempunyai *premise* yang sama, tetapi memiliki *conclusion* yang berlawanan.

   Contoh:

   Rule 1: *if the temperature is hot and humidity is high*
   Then there will be sunshine

   Rule 2: *if the temperature is hot and humidity is high*
   Then there will not be sunshine

3. *Subsumed rules*

   Subsumed rules terjadi jika *rule* tersebut mempunyai *constraints* yang lebih atau kurang tetapi mempunyai *conclusion* yang sama.

   Contoh:

   Rule 1: *if the temperature is hot and humidity is high and*

Then there will be thunderstorms

Rule 2: *if the temperature is hot*

Then there will be thunderstorms

4. *Circular rules*

   Circular rules terjadi jika suatu *premise* dari salah satu *rule* merupakan *conclusion* dari *rule* yang lain, atau kebalikannya.

   Contoh:

   Rule 1: *if X and Y are brothers*

   Then X and Y have the same parents

   Rule 2: *if X and Y have the same parents*

   Then X and Y are brothers

5. *Unnecessary if condition*

   Unnecessary if coindition terjadi jika dua *rule* atau lebih memiliki *conclusion* yang sama, tetapi salah satu dari *rule* tersebut mempunyai *premise* yang tidak perlu dikondisikan dalam *rule* karena tidak mempunyai pengaruh apapun.

   Contoh:

   Rule 1: *if the patient has pink spots and the patient has a fever*

   Then the patient has measles

   Rule 2: *if the patient has pink spots and the patient does not have fever*

   Then the patient has measles

6. *Dead end rules*

   Dead end rules adalah suatu *rule* yang *conclusion*-nya tidak diperlukan oleh *rules* lainnya.

   Contoh:

   Rule 1: *if the gauge read empty*

   Then the gas tank is empty