BAB II

GAMBARAN UMUM PERUSAHAAN

BAB II

TINJAUAN PUSTAKA

II.1. Penerapan Kelayakan Generator

Generator sebagai pembangkit energi listrik mempunyai peranan vital pada industri-industri. Dimana dewasa ini sistem kontrol dan proteksi generator selalu dikembangkan dan sekarang ini demikian canggih. Jumlah beban yang diterima oleh generator yang bekerja secara paralel tergantung pada pengaturan daya yang masuk pada mesin penggerak mula. Perubahan penguatan hanya akan merubah kVA yang keluar dan dapat mengubah faktor daya beban yang dibangkitkan dengan tidak mengubah kW mesin tersebut. Pada arus I1 dan I2 masing-masing mempunyai faktor daya Cos Φ1 dan Cos Φ2 , arus total yang disuplai oleh beban merupakan perjumlahan vektor arus I1 dan I2 . Bila daya masuk untuk mesin penggerak mula generator ke 2 dinaikan, maka vektor gaya gerak magnet bergeser ke kanan. Kenaikan daya yang masuk pada mesin penggerak mula dapat menyebabkan pengambilan beban yang lebih besar pada faktor daya yang berbeda. Beban yang diambil oleh masing-masing generator tergantung pada pengaturan kopel. Penguatan hanya mengubah faktor daya. Jika daya yang masuk ke mesin penggerak di jaga konstan tetapi penguatannya diubah, komponen kVA yang keluar dari generator tersebut dapat berubah sedangkan komponen kWnya tetap. (*Juhari ; 2013 : 39*)

II.1.1. *Evaluasi*

a. Sebuah generator 1 fasa dengan tegangan nominal V = 1 p.u dan arus jangkar Ia = 1 p.u, arus jangkar tersebut diperoleh dari pengujian hubung singkat dengan mengalirkan arus penguatan If = 1 p.u dan tegangan beban nol sebesar 0,25 p.u, tahanan stator Ra = 0,04 p.u. Tentukan pengaturan tegangan pada faktor daya yaitu, Unity. 0,81 terdahulu. 0,71 tertinggal.
b. Sebuah generator 3 fasa 5000 kVA, 10 kV, 1500 rpm, 50 Hz. Generator bekerja secara paralel dengan mesin yang lain yang identik. Reaktansi sinkron Xs = 20 % Hitunglah daya sinkronisasi dan torsi sinkronisasi untuk pergeseran sudut mekanik 0,50.
c. Dua buah generator dioperasikan paralel dan memberikan daya sebesar 1500 kW pada tegangan 11 kV serta faktor daya Cos Φ = 0,867 terbelakang. Masing-masing generator memberikan setengah daya totalnya. Reaktansi masing-masing 50 ohm per fasa dan tahanan efektifnya 4 ohm per fasa. Bila penguatan generator pertama disesuaikan agar arus yang mengalir pada jangkar 50 ampere terbelakang. Tentukan arus jangkar pada generator ke dua dan tegangan yang dibangkitkan generator pertama.*(Juhari ; 2013 : 33-34/39)*

Penerapan kelayakan *generator* salah satu komponen terpenting dalam pembuatan sistem turbin angin. Generator ini dapat mengubah energi gerak menjadi energi listrik. Prinsip kerjanya dapat dipelajari dengan menggunakan teori medan *elektromagnetik*. Singkatnya, (mengacu pada salah satu cara kerja *generator*) poros pada *generator* dipasang dengan material ferromagnetik permanen. Setelah itu disekeliling poros terdapat stator yang bentuk fisisnya

adalah kumparan-kumparan kawat yang membentuk *loop*. Ketika poros *generator* mulai berputar maka akan terjadi perubahan fluks pada stator yang akhirnya karena terjadi perubahan fluks ini akan dihasilkan tegangan dan arus listrik tertentu. Tegangan dan arus listrik yang dihasilkan ini disalurkan melalui kabel jaringan listrik untuk akhirnya digunakan oleh masyarakat. Tegangan dan arus listrik yang dihasilkan oleh generator ini berupa AC (*alternating current*) yang memiliki bentuk gelombang kurang lebih sinusoidal. ***(****Muji Sukur ; 2014)*

Guntur, dkk, (2012) mengomparasikan generator HHO tipe wet cell dan dry cell yang menggunakan material elektroda SS304 dengan ukuran elektroda yang sama ukurannya yaitu 16 cm x 16 cm. Menyimpulkan bahwa generator gas jenis wet cell memiliki efisiensi lebih tinggi jika dibandingkan generator jenis dry cell. Dengan bantuan arus listrik dan dua elektroda. Dimana arus listrik tersebut dialirkan pada elektroda positif (anoda) dan elektroda negatif (katoda), apabila diterapkan pada air maka senyawa kimia H2O akan terpecah menjadi gas Hidrogen (H2) serta gas Oksigen (O2). Agar suatu proses elektrolisa bekerja dengan cepat maka diperlukan zat lain yang disebut dengan katalis.*(Jarot D. Rahadi ; 2014 : 46-47)*

II.2. Sistem

Sistem adalah sebuah tatanan atau keterpaduan yang terdiri dari sejumlah komponen fungsional (dengan satuan fungsi atau tugas khusus) yang saling berhubungan dan secara bersama – sama bertujuan untuk memenuhi suatu proses atau pekerjaan tertentu. Suatu sistem terdiri dari sejumlah komponen yang saling berinteraksi, artinya saling bekerja sama membentuk satu kesatuan. Suatu sistem

mempunyai karakteristik atau sifat–sifat tertentu, yaitu mempunyai komponen–komponen (components), batas sistem (boundary), lingkungan luar sistem (environment), penghubung (interface), masukan (input), keluaran (output), pengolah (process) dan sasaran (objectives) atau tujuan (goal). Komponen–komponen sistem atau elemen–elemen sistem dapat berupa suatu sub sistem atau bagian–bagian dari sistem (Jogiyanto, 1999).*(Hafsah ; 2011 : D-43)*

II.3. SPK (Sistem Penunjang Keputusan)

Sistem Pendukung Keputusan (*Decision Support System/DSS*) merupakan sistem informasi interaktif yang menyediakan informasi, pemodelan, dan pemanipulasian data. Sistem ini digunakan untuk membantu pengambilan keputusan dalam situasi yang semiterstruktur dan situasi yang tidak terstruktur, dimana tak seorang pun tahu pasti bagaimana keputusan seharusnya dibuat. Sistem pendukung keputusan adalah sistem berbasis komputer yang menggunakan pengetahuan, fakta, dan teknik penalaran dalam memecahkan masalah yang biasanya hanya dapat dipecahkan oleh seorang pakar dalam bidang tersebut (Martin dan Oxman, 1998). Pada dasarnya sistem pendukung keputusan diterapkan untuk mendukung aktivitas pemecahan masalah. Beberapa aktivitas pemecahan yang dimaksud antara lain: pembuatan keputusan (*decision making*), pemaduan pengetahuan (*knowledge fusing*), pembuatan desain (*designing*), perencanaan (*planning*), prakiraan (*forecasting*), pengaturan (*regulating*), pengendalian (*controlling*), diagnosis (*diagnosing*), perumusan (*prescribing*), penjelasan (*explaining*), pemberian nasihat (*advising*) dan pelatihan (*tutoring*).

Selain itu sistem penunjang keputusan juga dapat berfungsi sebagai asisten yang pandai dari seorang pakar.

Sistem Pendukung Keputusan biasanya dibangun untuk mendukung solusi atas suatu masalah atau untuk mengevaluasi suatu peluang. Biasanya Sistem Pendukung Keputusan lebih ditujukan untuk mendukung manajemen dalam melakukan pekerjaan yang bersifat analitis dalam *(Leni Natalia Zulita; 2013 : 95-96)*

Sistem Pendukung Keputusan merupakan sistem informasi interaktif yang menyediakan informasi, pemodelan, dan pemanipulasian data. Sistem ini digunakan untuk membantu pengambilan keputusan dalam situasi yang semiterstruktur dan situasi yang tidak terstruktur, dimana tidak seorang pun tahu secara pasti bagaimana keputusan seharusnya dibuat (Kusrini, 2007).

Beberapa karakteristik yang membedakan Sistem Pendukung Keputusan dengan sistem informasi lainnya menurut *(Daihani,2001)* yaitu:

1. Sistem Pendukung Keputusan dirancang untuk membantu pengambilan keputusan dalam memecahkan masalah yang sifatnya semi terstruktur ataupun tidak terstruktur.
2. Dalam proses pengolahannya, sistem pendukung keputusan mengkombinasikan model-model analisis dengan teknik pemasukan dan konvensional secara fungsi-fungsi pencarian informasi.
3. Sistem Pendukung Keputusan dirancang sedemikian rupa sehingga dapat digunakan atau dioperasikan dengan mudah oleh orang-orang yang tidak memiliki dasar kemampuan pengoprasisan komputer yang tinggi. Oleh karena

itu pendekatan yang digunakan biasanya model interaktif.

4. Sistem Pendukung Keputusan dirancang dengan menekankan pada aspek fleksibilitas serta kemampuan adaptasi yang tinggi. Sehingga mudah disesuaikan dengan beragai perubahan lingkungan yang terjadi pada kebutuhan pemakai *(Hafsah ; 2011 : D-43/D-44)*

II.4. Metode *Certainty Factor*

Certainty Factor merupakan metode yang di gunakan untuk menyatakan kepercayaan dalam sebuah kejadian (fakta atau hipotesis) berdasakan bukti penilaian pakar. Secara konsep, *Certainty Factor* (CF) merupakan salah satu teknik yang digunakan untuk mengatasi ketidakpastian dalam mengambil keputusan. *Certainty Factor* (CF) dapat terjadi dengan berbagai kondisi yang terjadi adalah terdapat beberapa antasenden dalam *rule* yang berbeda dengan suatu konsekuen yang sama. kita harus mengagregasikan nilai CF keseluruhan dari setiap kondisi yang ada. Pada konsep Certainty Factor ini juga sering dikenal dengan adanya believe dan disbelieve. Believe merupakan keyakinan, sedangkan disbelive merupakan ketidakyakinan*(Siti Mujilahwati ; 2014 : 586-587)*

Ada dua cara dalam mendapatkan tingkat keyakinan (CF) *Certainty Factor* dari sebuah rule, yaitu :

1. Metode „Net Belief‟ yang diusulkan oleh

   E.H. shortliffe dan B.G. Buchaman

   CF(Rule) = MB(H,E)-MD(H,E) ..... (1)

$$MB(H,E) \begin{cases} P(H)=1 \\ \max[P(H|E),P(H)]-P(H) \end{cases}$$

Lainnya ..... (2)

$$\frac{\text{Max}|1,0|-P(H)}{1}$$

$$MD(H,E) \begin{cases} P(H)=0 \\ \min[P(H|E),P(H)]-P(H) \end{cases}$$

Lainnya ..... (3)

$$\text{min}|1,0|-P(H)$$

Di mana:

(CF)Rule = faktor kepastian

MB(H,E) = measure of belief (ukuran kepercayaan)terhadap hipotesis H, jika diberikan evidence (antara 0 dan 1)

MD(H,E) = measure of disbelief, (ukuran ketidakpercayaan) terhadap evidence H, jika diberikan evidence E (antara 0 dan 1)

P(H) = probabilitas kebenaran hipotesis H

P(H|E) = probabilitas bahwa H benar karena fakta E

2. Dengan cara mewawancarai seorang pakar Nilai CF (Rule) didapat dari interpretasi "term" dari pakar, yang diubah menjadi nilai CF tertentu sesuai tabel berikut :

   Aturan diantaranya :

   a. Ekspresi yang dihasilkan dari sebuah sistem lebih natural.
   b. Bagian pengendali terpisah dengan pengetahuan.
   c. Mudah dalam melakukan ekspansi sistem.
   d. Knowledge yang didapatkan lebih relevan.
   e. Dapat menggunakan pengetahuan yang bersifat heuristic.

II.5. *Java*

Java merupakan bahasa pemrograman berorientasi objek daan bebas platform, dikembangkan oleh SUN Micro System dengan jumlah keunggulan yang memungkinkan java dijadikan sebagai bahasa pengembang entreprise. Java merupakan bahasa yang powerfull yang bisa digunakan dalam hampir semua bentuk pengembangan software. Anda dapat menggunakan java untuk membuat game, aplikasi desktop, aplikasi web, aplikasi enterprise, aplikasi jaringan, dan lain-lain. Yang menarik adalah bahwa java bias digunakan untuk membuat laporan yang dapat berjalan di atas HP, PDA, dan peralatan lain yang dilengkapi dengan Java Virtual Machine(JVM).*(Atik Rusmayanti ; 2014 : 2)*

Java memiliki cara kerja yang unik dibandingkan dengan bahasa perograman lainya yaitu bahasa perograman *java* bekerja mengunakan *interptreter* dan juga *compiler* dalam proses pembuatan program, *Interpreter java* dikenal sebagai perograman *bytecode* yaitu dengan cara kerja mengubah paket *class* pada *java* dengan *extensi*. *Java* menjadi *.class*, hal ini dikenal sebagai *class bytecode,* yaitunya *class* yang dihasilkan agar program dapat dijalankan pada semua jenis perangkat dan juga *platform*, sehingga program *java* cukup ditulis sekali namun mampu bekerja pada jenis lingkungan yang berbeda.*(Defni, Indri Rahmayun ; 2014 : 64)*

II.6. *Database MySQL*

Database merupakan kumpulan dari data yang saling berhubungan satu dengan yang lainnya, tersimpan diperangkat keras komputer dan digunakan diperangkat lunak untuk memanipulasinya (Jogiyanto HM : 1999:711). *Database*

merupakan salah satu komponen yang sangat penting dalam sistem informasi, karena merupakan basis sistem dalam menyediakan informasi bagi para pemakai.(*Indra Warman, M.Kom ; 2012 ; 45*)

MySQL Server 2000 adalah suatu Perangkat lunak *Relational Database Mangement system* (RDBMS) yang handal. Didesain untuk mendukung proses transaksi yang besar (seperti *order entri* yang online, *inventori*, akuntansi atau manufaktur). *MySQL Server* akan secara otomatis menginstal enam database utama, yaitu *master*, model, tempdb, pubs, *Northwind*, dan Msdb. *(Anis nurhanafi ; 2013 : 5)*

II.7. ERD(*Entity Relationship Diagram*)

ERD adalah model konseptual yang mendeskripsikan hubungan antara penyimpanan. ERD digunakan untuk memodelkan struktur data dan hubungan antar data. Dengan ERD, model dapat diuji dengan mengabaikan proses yang dilakukan. ERD pertama kali dideskripsikan oleh *Peter Chen* yang dibuat sebagai bagian dari perangkat lunak *CASE*.

Komponen – komponen yang termasuk dalam ERD antara lain, adalah:

1. Entitas (*Entity*)

   Sebuah barang atau obyek yang dapat dibedakan dari obyek lain.

2. Relasi (*Relationship*)

   Asosiasi 2 atau lebih entitas dan berupa kata kerja.

3. Atribut (*Attribute*)

   Properti yang dimiliki setiap entitas yang akan disimpan datanya.

4. Kardinalitas (Kardinality)

   Angka yang menunjukkan banyaknya kemunculan suatu obyek terkait dengan kemunculan obyek lain pada suatu relasi.

Kardinalitas relasi yang terjadi diantara dua himpunan *entitas* (misalnya A dan B) dapat berupa:

1. Modalitas (*Modality*) adalah Partisipasi sebuah entitas pada suatu relasi, 0 jika partisipasi bersifat "optional"/parsial, dan 1 jika partisipasi bersifat "wajib"/total.
2. Total *constraint* adalah constraint yang mana data dalam entitas yang memiliki constraint tersebut terhubung secara penuh ke dalam entitas dari relasinya*(Adelia ; 2011 : 116)*

II.8. Normalisasi

Normalisasi adalah proses pengelompokan elemen data menjadi tabel-tabel yang menunjukan *entity* dan relasinya. Pada proses ini selalu di uji pada beberapa kondisi. Apakah ada kesulitan pada saat menambah (*insert*) pada suatu *database*. Bila ada kesulitan pada pengujian tersebut maka relasi tersebut dapat dipecahkan pada beberapa tabel lagi atau dengan kata lain perancangan yang dilakukan belum mendapatkan suatu database yang optimal. Sebelum mengenal lebih jauh mengenai normalisasi ada beberapa konsep yang harus diketahui lebih dahulu seperti *field* atau *attribute* kunci dan ketergantungan kunci (*Functional Depencendy*).

1. Calon Kunci (*Candidate key*)

   Kunci kandidat atau calon kunci adalah suatu attribute atau satu set minimal attribute yang mengidentifikasi secara unik suatu kejadian yang spesifik dari suatu *entity*.

2. Kunci Primer (*Primary Key*)

   Kunci primer adalah suatu *attribute* atau satu set minimal *attribute* yang tidak hanya mengidentifikasi secara unik suatu kejadian yang spesifik, akan tetapi juga dapat mewakili setiap kejadian dari suatu *entity*. Setiap kunci kandidat punya peluang menjadi kunci primer, akan tetapi sebaiknya dipilih satu saja yang dapat mewakili secara menyeluruh terhadap *entity* yang ada.

3. Kunci Alternatif (*Alternate Key*)

   Kunci Alternatif adalah kunci kandidat yang tidak dipakai sebagai *primary key*. Dimana kerap kali kunci alternatif ini dipakai sebagai kunci pengurutan dalam pembuatan laporan.

4. Kunci Tamu (*Foreign Key*)

   Kunci tamu adalah satu attribute atau satu set attribute yang melengkapi satu relationship (hubungan) yang menunjukan ke induknya.

Teknik normalisasi ini juga merupakan satu teknik yang menstrukturkan data dalam cara tertentu untuk membantu mengurangi atau mencegah timbulnya masalah yang berhubungan dengan pengolahan data dalam database. Dalam pembuatan normalisasi terdapat beberapatahap pembentukan, setiap tahap mempunyai bentuk normalisasi yang berbeda. Bentuk-bentuk tersebut antara lain :

1. Bentuk Tidak Normal (*Unnormalized Form*)

   Bentuk ini merupakan data yang akan direkam, tidak ada keharusan mengikuti suatu format tertentu. Dapat saja data tidak lengkap atau terduplikasi. Data dikumpulkan apa adanya sesuai dengan saat penginputan atau saat kedatangannya.

2. Bentuk Normal Kesatu (1NF / *First Normal Form*)

   Bentuk normal kesatu mempunyai ciri yaitu setiap data dibentuk dalam *flat file* (*file* datar/rata), data dibentuk dalam satu *record* demi satu *record*, nilai dari *field-field* berupa "*atomic value*". Tidak ada set atribut yang berulang atau atribut bernilai ganda (*multivalue*). Tiap field hanya satu pengertian, bukan merupakan kumpulan kata yang mempunyai arti mendua, hanya satu arti saja dan juga bukanlah pecahan kata-kata sehingga artinya menjadi lain.

3. Bentuk Normal Kedua (2NF / *Second Normal Form*)

   Bentuk normal kedua mempunyai syarat yaitu bentuk data telah memenuhi kriteria dari bentuk normal kesatu. Atribut bukan kunci haruslah bergantung secara fungsi pada kunci utama / *primary key*. Sehingga untuk membentuk normal kedua harus sudah ditentukan kunci-kunci fieldnya. Kunci *field* harus unik dan dapat mewakili atribut lain yang menjadi anggotanya.

3. Bentuk Normal Ketiga (3NF / *Third Normal Form*)

   Untuk menjadi normal ketiga maka relasi harus dalam bentuk normal kedua dan semua atribut bukan primer tidak mempunyai hubungan yang transitif. Dengan kata lain, semua atribut bukan kunci haruslah bergantung hanya pada *primary key* secara menyeluruh.

II.9. UML *(Unified Modelling Language)*

UML adalah sebuah bahasa yang sudah menjadi standar dalam industri untuk merancang, menspesifikasi dan mendokumentasi sistem perangkat lunak . *UML* memberikan standar penulisan tersendiri pada sebuah sistem *blue print*, yang mencakup konsep proses bisnis, penulisan kelas-kelas pada bahasa program yang spesifik, skema database dan komponen-komponen yang dibutuhkan dalam sistem piranti lunak *(Arzan Muharom ; 2013 : 2)*

Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. 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 pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep 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 modeling

aplikasi prosedural dalam VB atau C. Seperti bahasa-bahasa lainnya, UML mendefinisikan 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 Design*), Jim Rumbaugh OMT (*Object Modeling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*). Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: *metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi wirfs-brock*, dsb. Masa itu terkenal dengan masa perang metodologi (*method war*) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan. Dimulai pada bulan Oktober 1994 *Booch, Rumbaugh dan Jacobson*, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease *draft* pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh *Object Management Group* (OMG – http://www.omg.org). Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 1.5 yang dirilis bulan Maret 2003. *Booch, Rumbaugh* dan *Jacobson* menyusun tiga

buku serial tentang UML pada tahun 1999. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek. *(Yuni Sugiarti ; 2013 : 33)*

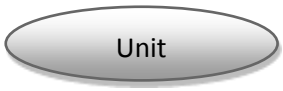
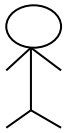
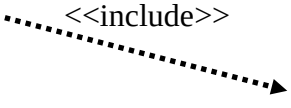
Dalam pembuatan skripsi ini penulis menggunakan diagram *Use Case* yang terdapat di dalam UML. Adapun maksud dari *Use Case* Diagram diterangkan dibawah ini.

1. *Use Case Diagram*

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. *Use case diagram* dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. Sebuah *use*

case juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain. *(Yuni Sugiarti ; 2013 : 41)*

Tabel II.1. *Use Case* Diagram

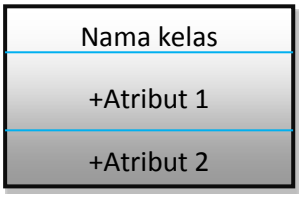
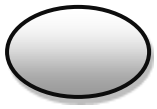
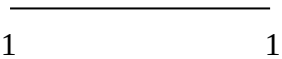
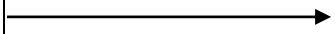
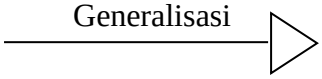
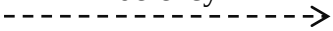
Simbol	Deskripsi
Use Case 	Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antara unit dan aktor, biasanya dinyatakan dengan menggunakan kata kerja diawali frase nama use case
Aktor  Nama aktor	Orang ,proses, atau sistem lain yang berinteraksi dengan sistem informasi 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 ; biasanya dinyatakan menggunakan kata benda diawali frase nama aktor
Asosiasi / association	Komunikasi antara aktor dan use case yang berpartisipasi pada use case memiliki interaksi dengan aktor
Extend  <<extend>>	Relasi use case tambahan kesebuah use case dimana use case ditambahan dapat berdiri sendiri walaupun tanpa use case tambahan itu ; mirip dengan prinsip inheritens pada pemograman berorientasi objek, biasanya use case tambahan memiliki nama depan yang sama dengan use case yang ditambahkan, arah panah menunjukan pada use case yang dituju
Include <<include>> 	Relasi use case tambahan kesebuah use case dimana use case yang ditambahkan memerlukan use case ini untuk menjalankan fungsinya atau sebagai syarat dijalankan use case ini. Ada dua sudut pandang yang cukup besar mengenai include di use case, include berarti use case tambahan yang dijalankan.

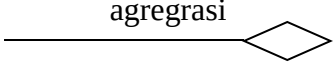
Sumber : (Yuni Sugiarti ; 2013 ; 42)

2. *Class Diagram*

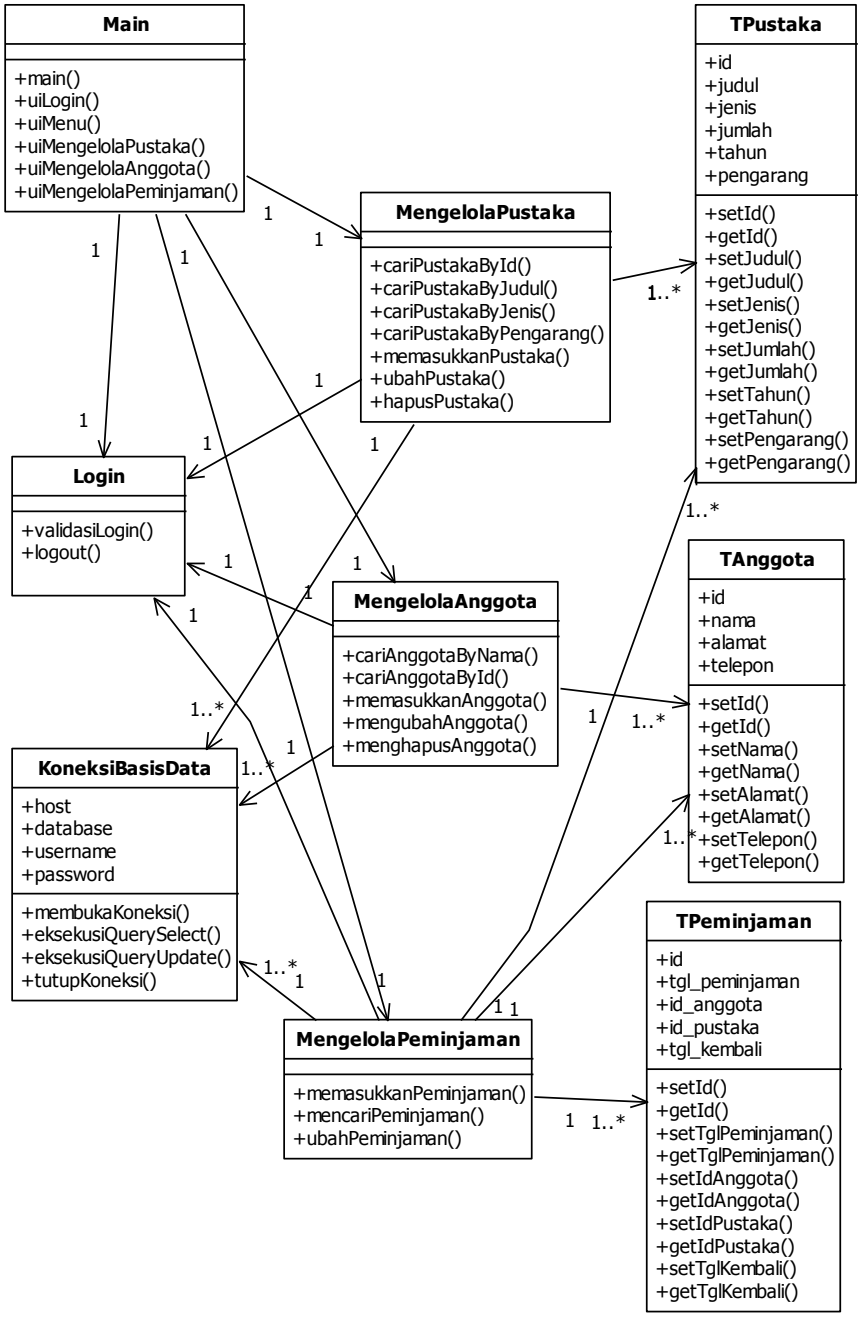
Diagram kelas atau *class* diagram menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi. Berikut adalah simbol-simbol pada diagram kelas :

Tabel II.2. *Use Case Diagram*

Simbol	Deskripsi
Package 	Package merupakan sebuah bungkusan dari satu atau lebih kelas
Operasi 	Kelas pada struktur sistem
Anataramuka / interface  Interface	Sama dengan konsep interface dalam pemrograman berorientasi objek
Asosiasi 	Relasi antara kelas dengan makna umum, asosiasi biasanya juga disertai dengan multiplicity
Asosiasi berarah/directed asosiasi 	Relasi antara kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan multiplictiy
Generalisasi 	Relasi antara dengan makna generalisasi sepesialisasi (umum-khusus)
Kebergantungan / defency 	Relasi antara kelas dengan makna kebergabungan antara kelas

agregrasi 	Relasi antara kelas dengan makna semua bagian (whole part)

Sumber : (Yuni Sugiarti ; 2013 : 59)



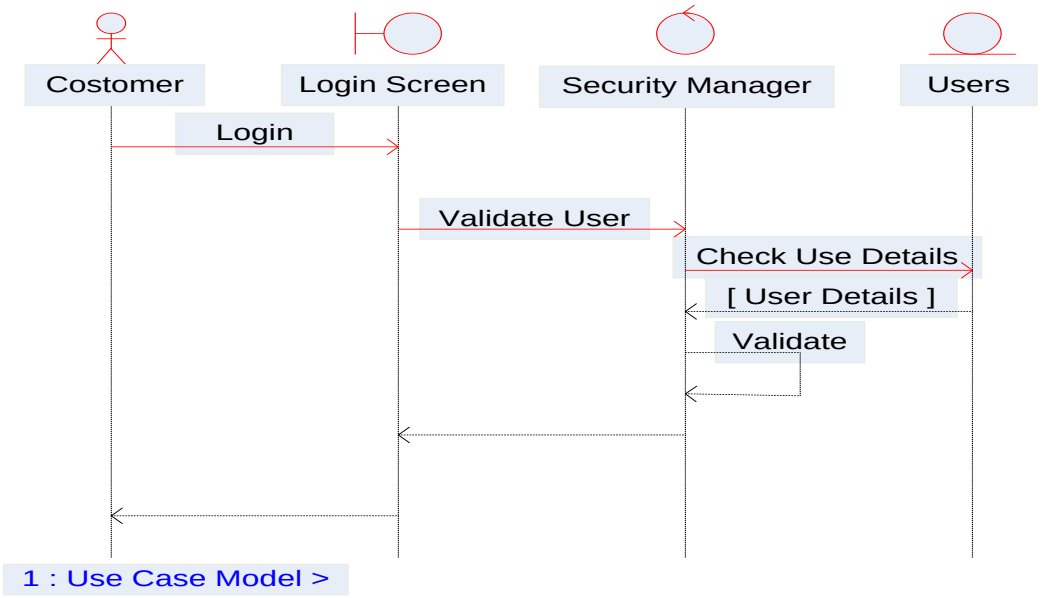
Gambar II.1. Contoh Class Diagram

Sumber : (Yuni Sugiarti ; 2013 : 63)

3. *Sequence Diagram*

Diagram *Sequence* menggambarkan kelakuan/prilaku objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram *sequence* maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

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



Gambar II.2. Contoh *Sequence* Diagram

Sumber : (Yuni Sugiarti ; 2013 : 63)

4. *Activity Diagram*

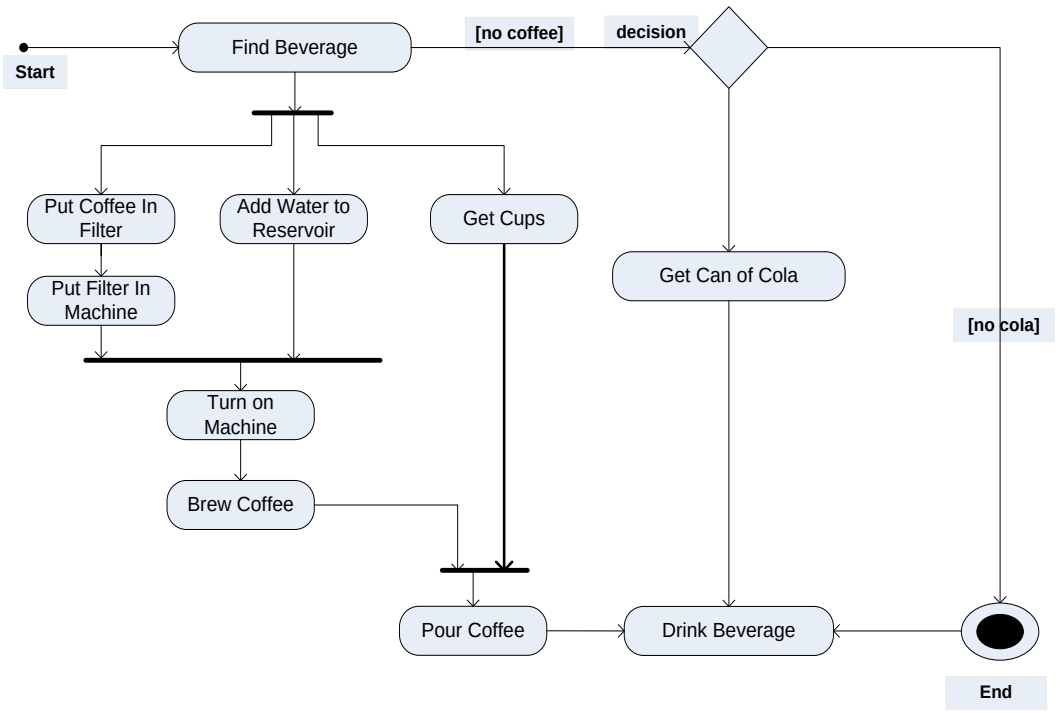
Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus, di mana 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 menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal.

Activity diagram dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.



Gambar II.3. Activity Diagram
Sumber : (Yuni Sugiarti ; 2013 : 76)