BAB II

LANDASAN TEORI

2.1 Konsep Perancangan Sistem

Hal yang paling dominan ketika perancangan suatu aplikasi dilakukan adalah memodelkan kebutuhan pemakai. Ada banyak cara untuk memodelkan aplikasi sebagaimana banyak cara yang digunakan oleh seorang arsitek untuk membangun sebuah rumah. Pada dasarnya pemodelan tersebut merupakan kombinasi antara perangkat lunak dan perangkat keras yang digunakan (Whitten *et al*, 2005). Perancangan suatu aplikasi termasuk dalam kegiatan rekayasa perangkat lunak. Proses rekayasa perangkat lunak dimulai jauh sebelum *coding* dilakukan dan berlanjut sampai tercapainya sebuah aplikasi yang diinginkan (Pohan, 1997). Pada dasarnya Rekayasa Perangkat Lunak dilakukan untuk merancang suatu aplikasi atau *software* dengan mengurutkan transformasi masalah menjadi solusi perangkat lunak yang dapat bekerja dengan baik.

2.1.1 Prinsip Dasar Perancangan Sistem

Proses perancangan perangkat lunak merupakan serangkaian kegiatan dan hasil yang berhubungan dengan perangkat lunak, yang bertujuan untuk dihasilkannya suatu produk perangkat lnak. Walaupun ada banyak proses dalam perancangan suatu perangkat lunak, ada kegiatan-kegiatan mendasar yang umum bagi semua proses perancangan perangkat lunak (Sommerville,2003), antara lain:

1. Penspesifikasian Perangkat Lunak

   Fungsionalitas Perangkat Lunak dan batasan operasinya harus didefenisikan.

2. Perancangan dan Implementasi Perangkat Lunak

   Perangkat Lunak yang memenuhi persyaratan harus dibuat.

3. Validasi Perangkat Lunak

   Perangkat lunak tersebut harus divalidasi untuk menjamin bahwa perangkat lunak bekerja sesuai dengan apa yang diinginkan.

4. Pengevolusian Perangkat Lunak

   Perangkat Lunak harus dapat berkembang untuk menghadapi kebutuhan yang dapat berubah sewaktu-waktu.

Dalam menciptakan sebuah aplikasi, terdapat beberapa hal yang perlu diperhatikan guna perolehan hasil yang maksimal (Whitten *et al*, 2005), antara lain sebagai berikut :

a. Produktivitas

   Saat ini hampir segala bidang memerlukan aplikasi yang dapat digunakan sesuai dengan keperluan dalam bidangnya. Hal ini menyebabkan permintaan terhadap pengadaan aplikasi lebih banyak. Dan tuntutan terhadap kualitas aplikasi yang lebih bagus dan handal. Tentunya hal ini membutuhkan lebih banyak *programmer* dan penganalisa sistem yang berkualitas, kondisi kerja ekstra, kemampuan pemakai untuk mengembangkan sendiri, bahasa pemrograman yang lebih baik, perawatan sistem yang lebih baik, disiplin teknis pemakaian perangkat lunak dan perangkat pengembangan sistem yang terotomasi.

b. Reliabilitas

Reliabilitas suatu perangkat lunak tidak seperti faktor kualitas lain yang dapat diukur, diarahkan dan diestimasi dengan menggunakan data pengembangan historis. Reliabilitas perangkat lunak didefenisikan dalam bentuk statistik sebagai kemungkinan operasi program komputer bebas kegagalan didalam suatu lingkungan dalam kurun waktu tertentu.

c. Maintabilitas

Maintabilitas mencakup perawatan aplikasi, seperti :

- Koreksi jika ditemukan kesalahan pada program.
- Pengadaptasian jika lingkungan berubah.
- Modifikasi jika pengguna membutukan perubahan kebutuhan.

d. Integritas

Integritas adalah mengukur kemampuan sistem suatu aplikasi untuk menahan serangan terhadap sekuritasnya. Dalam hal ini kekuatan sistem akan diuji terhadap serangan dari tipe tertentu yang dapat terjadi suatu waktu.

e. Usabilitas

Usabilitas merupakan ukuran terhadap kualitas interaksi yang terjadi antara aplikasi dengan pengguna. Ukuran usabilitas dapat diketahui melalui tampilan fisik suatu aplikasi (*user friendly)*, penggunaan waktu yang efisien dan lain sebagainya.

2.1.2 Karakteristik Sistem

Suatu sistem mempunyai karakteristik atau sifat-sifat yang tertentu, yaitu mempunyai komponen-komponen *(components)*, batas sistem *(boundary)*, lingkungan luar sistem *(environments)*, penghubung *(interface)*, masukan *(input)*, keluaran *(output)*, pengolah *(process)* dan sasaran *(objectives)* atau tujuan *(goal)*.

1. Komponen *(Components)*

   Suatu sistem terdiri dari sejumlah komponen yang saling berinteraksi dan bekerjasama membentuk satu kesatuan. Komponen-komponen sistem atau elemen-elemen sistem dapat berupa suatu subsistem atau bagian bagian dari sistem

2. Batas sistem *(Boundary)*

   Batas sistem merupakan daerah yang membatasi antara satu sistem dengan sistem lainnya atau dengan lingkungan luarnya. Batas sistem ini memungkinkan suatu sistem dipandang sebagai satu kesatuan. Batas suatu sistem menunjukan ruang lingkup *(scope)* dari sistem tersebut.

3. Lingkungan Luar Sistem *(Environments)*

   Lingkungan luar adalah segala sesuatu yang berada diluar batas dari sistem yang mempengaruhi operasi sistem baik itu yang bersifat merugikan ataupun menguntungkan. Lingkungan luar yang menguntungkan merupakan energi dari sistem dengan demikian harus tetap dijaga dan dipelihara. Sedangkan lingkungan luar yang merugikan harus ditahan dan dikendalikan, kalau tidak maka akan mengganggu kelangsungan hidup dari sistem.

4. Penghubung Sistem *(Interface)*

   Penghubung merupakan media penghubung antar subsistem yang memungkinkan sumber-sumber daya mengalir dari satu subsistem ke subsistem lain. Keluaran dari satu sistem akan menjadi masukan untuk subsistem yang lainnya dengan melalui penghubung. Dengan penghubung satu subsistem dapat berintegrasi dengan subsistem lainnya membentuk satu kesatuan.

5. Masukan *(Input)*

   Masukan adalah energi yang dimasukan kedalam sistem, yang dapat berupa masukan perawatan *(Maintenance Input)* dan masukan sinyal *(Signal Input)*. Masukan perawatan adalah energi yang dimasukkan supaya sistem tersebut dapat beroperasi. Masukan sinyal adalah energi yang diproses untuk mendapatkan keluaran.

6. Keluaran *(Output)*

   Adalah hasil dari energi yang diolah dan diklasifikasikan menjadi keluaran yang berguna dan sisa pembuangan. Keluaran dapat merupakan masukan untuk subsistem yang lain atau kepada supra sistem.

7. Pengolah *(Process)*

   Suatu sistem dapat mempunyai suatu bagian pengolah yang akan mengubah masukan menjadi keluaran. Suatu sistem produksi akan mengolah masukan berupa bahan baku dan bahan-bahan yang lain menjadi keluaran berupa bahan jadi.

8. Sasaran *(Objective)* atau Tujuan *(Goal)*

   Suatu sistem pasti mempunyai tujuan *(goal)*, jika suatu sistem tidak mempunyai tujuan yang jelas, maka semua operasi sistem tidak ada gunanya. Sasaran dari sistem sangat menentukan sekali masukan yang dibutuhkan sistem dan keluaran yang akan dihasilkan sistem.

2.1.3 Klasifikasi Sistem

Sistem dapat diklasifikasikan dari beberapa sudut pandangan, diantaranya adalah sebagai berikut :

1. Sistem diklasifikasikan sebagai sistem abstrak (*abstract system)* dan sistem fisik *(physical system).* Sistem abstrak adalah sistem yang berupa pemikiran atau ide-ide yang tidak tampak secara fisik. Sedangkan sistem fisik adalah sistem yang ada secara fisik.
2. Sistem diklasifikasikan sebagai sistem alamiah *(natural system)* dan sistem buatan manusia *(human made system).*

   Sistem alamiah adalah sistem yang terjadi melalui proses alam, tidak dibuat manusia. Sistem buatan manusia yang melibatkan interaksi antara manusia dengan mesin disebut dengan *man-machine system.* Sistem informasi merupakan contoh *man-machine system,* karena menyangkut penggunaan komputer yang berinteraksi dengan manusia.
3. Sistem diklasifikasikan sebagai sistem tertentu *(determinic system)* dan sistem tak tentu *(probabilistic system).*

Sistem tertentu beroperasi dengan tingkah laku yang sudah dapat diprediksi. Interaksi diantara bagian bagiannya dapat dideteksi dengan pasti, sehingga keluaran dari sistem dapat diramalkan. Sistem tak tentu adalah sistem yang kondisi masa depannya tidak dapat diprediksi karena dapat mengandung unsur probabilitas.

4. Sistem diklasifikasikan sebagai sistem tertutup *(close system)* dan sistem terbuka *(open system).*

   Sistem tertutup merupakan sistem yang tidak berhubungan dan tidak terpengaruh dengan lingkungan luarnya. Sistem ini bekerja secara otomatis tanpa adanya turut campur tangan dari pihak diluarnya. Sistem terbuka adalah sistem yang berhubungan dan terpengaruh dengan lingkungan luarnya. Sistem ini menerima masukan dan menghasilkan keluaran untuk lingkungan luar atau subsistem yang lainnya.

2.2 Konsep Dasar Informasi

Informasi ibarat darah yang mengalir di dalam tubuh suatu organisasi, sehingga informasi ini sangat penting di dalam suatu organisasi. Suatu sistem yang kurang mendapatkan informasi akan menjadi luruh, kerdil dan akhirnya berakhir.

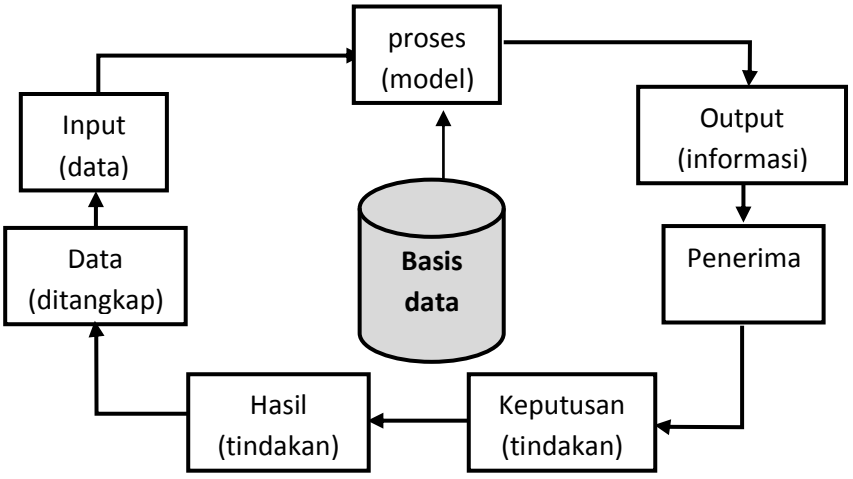
Sumber dari informasi adalah data, data merupakan bentuk jamak dari bentuk tunggal datum atau data-item. Data adalah kenyataan yang menggambarkan suatu kejadian-kejadian dan kesatuan nyata. Kejadian-kejadian

(event) adalah kejadian nyata yang sering terjadi adalah perubahan dari suatu nilai uang disebut dengan transaksi.

2.2.1 Siklus Informasi

Data merupakan bentuk yang masih mentah yang belum dapat berceritera banyak, sehingga perlu diolah lebih lanjut. Data diolah melalui suatu model untuk menghasilkan informasi.

Data yang diolah untuk menghasilkan informasi menggunakan suatu model proses yang tertentu. Data yang diolah melalui suatu model menjadi informasi, penerima kemudian menerima informasi tersebut membuat suatu keputusan dan melakukan tindakan, yang berarti menghasilkan suatu tindakan yang lain yang akan membuat sejumlah data kembali. Data tersebut akan ditangkap sebagai input, diproses kembali lewat suatu model dan seterusnya membentuk suatu siklus.



Gambar 2.1 Siklus Informasi

(Sumber : Jogiyanto : "Analisis dan Desain", Tahun 2005 hal. 9)

2.2.2 Kualitas Informasi

Kualitas informasi tergantung dari tiga hal yaitu:

1. Akurat

   Informasi harus terbebas dari kesalahan-kesalahan dan tidak bias atau menyesatkan. Akurat juga berarti informasi harus jelas mencerminkan maksudnya. Informasi harus akurat karena dari sumber informasi sampai ke penerima informasi kemungkinan banyak terjadi gangguan *(noise)* yang dapat merubah atau merusak informasi tersebut.

2. Tepat pada waktunya

   Informasi yang datang pada penerima tidak boleh terlambat. Informasi yang sudah usang tidak akan mempunyai nilai lagi. Karena informasi merupakan landasan di dalam pengambilan keputusan. Bila pengambilan keputusan terlambat, maka dapat berakibat fatal bagi organisasi. Saat ini mahalnya nilai informasi disebabkan harus cepatnya inforamsi tersebut didapat, sehingga diperlukan teknologi-teknologi mutakhir untuk mendapatkan, mengolah dan mengirimkannya.

3. Relevan

   Informasi tersebut mempunyai manfaat untuk pemakainya. Relevansi informasi untuk tiap-tiap orang satu dengan yang lainnya berbeda.

2.2.3 Nilai Informasi

Nilai dari informasi *(value of information)* ditentukan dari dua hal, yaitu menfaat dan biaya mendapatkannya. Suatu informasi dikatakan bernilai bila manfaatnya lebih efektif dibandingakan dengan biaya mendapatkannya. Akan

tetapi perlu diperhatikan bahwa informasi yang digunakan dalam suatu sistem informasi umumnya digunakan untuk beberapa kegunaan. Sehingga tidak memungkinkan dan sulit untuk menghubungkan suatu bagian informasi pada suatu masalah tertentu dengan biaya untuk memperolehnya.

2.3 Pengertian Sistem Informasi

Setelah diketahui bahwa informasi merupakan sangat penting bagi suatu organisasi atau manajemen untuk mengambil keputusan, maka dapat didefinisikan sebagai berikut:

1. Suatu sistem yang dibuat oleh manusia yang terdiri dari komponen-komponen dalam berorganisasi untuk mencapai suatu tujuan yaitu menyajikan infomasi
2. Sekumpulan prosedur organisasi yang pada saat dilaksanakan akan memberikan informasi bagi pengambil keputusan dan atau untuk mengendalikan informasi
3. Suatu sistem didalam organisasi yang mempertemukan kebutuhan pengolahan transaksi, mendukung operasi, bersifat manajerial, dan kegiatan strategi dari suatu organisasi dan menyediakan pihak luar tertentu dengan laporan-laporan yang diperlukan.

2.4 Pelayanan Jasa

Pelayanan jasa memegang peranan amat penting didalam sektor industri belakangan ini. Definisi dari jasa menurut Kotler (1991:229) adalah :

” *Berbagai tindakan atau kinerja yang ditawarkan oleh suatu pihak kepada pihak*

lain yang pada dasarnya tidak dapat dilihat dan tidak menghasilkan hak milik terhadap sesuatu. Produksinya dapat berkenaan dengan sebuah produk fisik maupun tidak. ”

Jasa merupakan suatu merupakan suatu produk yang tidak berwujud. Membeli jasa sama dengan membeli sesuatu yang tidak berwujud. Bila dilihat lebih jauh antara produk dan jasa sebenarnya sulit dibedakan, karena biasanya pembelian dari suatu produk akan disertai juga dengan fasilitas jasa dan pembelian jasa juga mengikutsertakan barang. Jika suatu barang dihasilkan dari sebuah proses produksi, maka jasa dihasilkan melalui pemberian sarana dan prasarana yang harus ditunjang dengan penyampaian ketrampilan tertentu dari pihak pemberi jasa, karenanya kualitas pelayanan merupakan faktor utama dalam menyediakan layanan jasa tersebut

2.4.1 Pengertian Jasa Servis Sepeda Motor

Kata servis dalam bahasa inggris adalah memperbaiki, dan kesimpulannya membuat barang yang tadinya rusak atau ada salah satu bagian yang tidak berfungsi menjadi baik atau berfungi kembali atau serangkaian prosedur perawatan kendaraan bermotor dalam kurun waktu tertentu yang terekam dan tercatat dalam sebuah laporan. Laporan tersebut biasanya berupa catatan kerusakan yang pernah terjadi, suku cadang yang telah diganti dan sevis serta perbaikan apa saja yang pernah dilakukan.

Sebuah servis motor adalah serangkaian prosedur perawatan yang dilakukan pada interval waktu yang ditetapkan atau pada saat kendaraan telah

menempuh jarak tertentu. Interval layanan ditentukan oleh produsen kendaraan dalam jadwal layanan servis.

Pemeliharaan tugas yang biasa dilakukan dalam layanan jasa servis kendaraan sepeda motor meliputi :

1. Mengganti oli mesin
2. Mengganti busi bila diperlukan
3. Membersihkan karburator
4. Memeriksa tingkat dan cairan isi ulang rem
5. Memeriksa tingkat air Accu
6. Melumasi dan grease komponen

2.5 Pengertian Penjualan

Penjualan merupakan bagian dari kegiatan pemasaran, penjualan sangat penting untuk menetukan suatu pemasaran produknya yaitu dengan cara menjual produk tersebut dan apabila penjualan tidak dapat dilaksanakan maka fungsi-fungsi lain dari pemasaran tidak akan berjalan dengan lancar. Kegiatan penjualan terbagi dalam dua hal :

1. Penjualan kredit yaitu penjualan barang yang dilakukan dengan cara mengirim barang sesuai dengan pesanan dari pembeli dan tidak ada pembayaran langsung yang terjadi dalam jangka waktu tertentu dan *took* mempunyai tagihan pada pembeli tersebut.
2. Penjualan tunai yaitu apabila *took* tersebut menjual produknya secara langsung pada pembeli dan di bayar pada saat itu juga oleh pembelinya.

Sistem penjualan adalah pola hubungan antara bagian-bagian yang berkaitan untuk melakukan kegiatan yaitu memproses data penjualan sehingga menghasilkan informasi yang cepat, tepat, dan akurat sehingga memuaskan kedua belah pihak.

2.6 Pengertian Sparepart

Sparepart diterjemahkan dalam bahasa Indonesia yaitu suku cadang dapat disimpulkan dari kata tersebut yaitu komponen-komponen yang ada pada alat atau sebuah benda.

2.7 *Unified Modelling Language* (UML)

Unified Modelling Language (UML) adalah sebuah "bahasa" yang 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 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. Karena UML menggunakan class dan operation dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa-bahasa berorientasi objek seperti C++, Java, atau VB.NET. Walaupun demikian, UML tetap dapat digunakan dalam PHP.

Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax/semantic*. 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 tahun 1990, banyak puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: metodologi booch, metodologi coad , metodologi OOSE, metodologi OMT, dsb. Banyak sekali metodologi-metodologi yang digunakan dalam membuat desain perangkat lunak yang berorientasi objek. Masing-masing metodologi mempunyai notasi sendiri-sendiri, sehingga mengakibatkan timbul masalah baru yang dihadapi pada saat antara pengembang perangkat lunak yang satu dengan yang lainnya akan bekerjasama dengan group /perusahaan lain dengan menggunakan metodologi yang berlainan.

Dimulai tahun 1994 tepatnya bulan Oktober Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang 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, 2013, 11 Juli, 10 : 17).

UML (*Unified Modeling Language*) digunakan untuk melakukan pemodelan sistem / perangkat lunak lewat *tools* yang ada. Dengan pemodelan menggunakan UML ini, pengembang dapat melakukan :

1. Tinjauan umum bagaimana arsitektur sistem secara keseluruhan.
2. Penelaah bagaimana objek-objek dalam sistem saling mengirimkan pesan (message) dan saling bekerjasama satu sama lain.
3. Menguji apakah sistem/perangkat lunak sudah berfungsi seperti yang seharusnya.
4. Dokumentasi sistem/perangkat lunak untuk keperluan-keperluan tertentu di masa yang akan datang.
5. Notasi UML terbentuk dari kerjasama dan upaya Graddy Booch (yang sebelumnya dikenal sebagai notasi booch-nya), DR. James Rumbaugh yang sebelumnya terkenal dengan notasi OMT(*Object Modeling Technique*-nya), serta Ivar Jacobson (yang sebelumnya terkenal dengan OOSE (*Object Oriented Software Engineering*)-nya

Setiap sistem yang komplek seharusnya bisa dipandang dari sudut pandang yang berbedabeda sehingga bisa dilakukan pemahaman secara menyeluruh. Dalam upaya-nya tersebut, UML menyediakan 9 jenis diagram yang dapat dikelompokkan berdasarkan sifatnya yang statis ataupun dinamis. Ke-9 jenis diagram untuk UML adalah:

1. *Use case* diagram, bersifat statis, memperlihatkan himpunan use-case dan aktor-aktor. Diagram ini sangat penting terutama untuk memodelkan ataupun mengorganisasikan perilaku dari sistem yang dibutuhkan pengguna.

2. *Class diagram*, bersifat statis tetapi sering pula memuat kelas-kelas aktif dan memperlihatkan himpunan kelas-kelas, antarmuka-antarmuka, kolaborasi-kolaborasi, serta relasi-relasi.
3. *Statechart diagram*, bersifat dinamis yang memperlihatkan state – state dari sistem, memuat state, transisi, event, serta aktivitas. Penting untuk memperlihatkan sifat dinamis dari antarmuka (interface), kelas, kolaborasi, terutama penting pada pemodelan system-sistem yang reaktif.
4. *Activity diagram*, bersifat dinamis. Merupakan tipe khusus dari diagram state yang memperlihatkan aliran dari suatu aktivitas ke aktivitas lainnya dalam suatu sistem.
5. *Sequence diagram*, bersifat dinamis yang menekankan pada pengiriman pesan (*message*) dalam suatu waktu tertentu.
6. *Collaboration diagram*, bersifat dinamis yang menekankan organisasi struktural dari objek-objek yang menerima serta mengirim pesan (*message*).
7. *Component diagram*, bersifat statis, diagram ini berhubungan dengan diagram kelas dimana komponen secara tipikal dipetakan ke dalam satu atau lebih kelas-kelas, antarmuka (*interface*) ataupun kolaborasi.
8. Digram objek, bersifat statis, memperlihatkan objek-objek serta serta relasi-relasi antar objek. Selain itu juga memperlihatkan instansiasi statis dari segala sesuatu yang dijumpai pada diagram kelas.
9. Depeloyment diagram, bersifat statis, digram memperlihatkan konfigurasi saat aplikasi dijalankan (*run-time*). Digram inisangat berguna saat aplikasi kita

berlaku sebagai aplikasi yang dijankan pada banyak mesin (*distributed computing*).

2.7.1 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 untuk 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.

2.7.2 *Class Diagram*

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. Class menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).

Class diagram menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. *Class* memiliki tiga area pokok :

1. Nama (dan *stereotype*)
2. Atribut
3. Metoda

Atribut dan metoda dapat memiliki salah satu sifat berikut :

1. *Private*, tidak dapat dipanggil dari luar class yang bersangkutan.
2. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya
3. *Public*, dapat dipanggil oleh siapa saja

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak instan dapat dipergunakan, tetapi harus diimplementasikan dahulu menjadi sebuah class. Dengan demikian *interface* mendukung resolusi metoda padasaat *run-time*. Sesuai dengan perkembangan *class model*, class dapat dikelompokkan menjadi *package*.

Hubungan Antar *Class*

1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa class lain, atau class yang harus mengetahui eksistensi class lain. Panah *navigabilit*y menunjukkan arah *query* antar *class*.
2. Agregasi, yaitu hubungan yang menyatakan bagian ("terdiri atas.").
3. Pewarisan, yaitu hubungan hirarkis antar *class*. Class dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda class asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
4. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang dipassing dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram*.

2.7.3 *Activity Diagram*

Activity diagrams 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. 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.

2.7.4 *Squence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, display, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence* diagram terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan output tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan output apa yang dihasilkan.

Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali

dengan diterimanya sebuah *message*. Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan icon khusus untuk objek boundary, *controller* dan *persistent entity*.

2.7.5 *Collaboration Diagram*

Collaboration diagram juga menggambarkan interaksi antar objek seperti *sequence diagram*, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian message. Setiap message memiliki *sequence number*, di mana *message dari level* tertinggi memiliki nomor 1. *Messages* dari *level* yang sama memiliki prefiks yang sama.

2.7.6 *Component Diagram*

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) diantaranya. Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*, baik yang muncul pada *compile time*, *link time*, maupun *run time*. Umumnya komponen terbentuk dari beberapa *class* dan/atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.

2.7.7 *Deployment Diagram*

Deployment/physical diagram menggambarkan detail bagaimana komponen di-*deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, server atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisikal. Sebuah

node adalah server, workstation, atau piranti keras lain yang digunakan untuk men-*deploy* komponen dalam lingkungan sebenarnya. Hubungan antar node (misalnya TCP/IP) dan *requirement* dapat juga didefinisikan dalam diagram ini.