Bab 2
Landasan Teori

2.1. Pengertian Sistem

Sistem berasal dari bahasa Latin (*systēma*) dan bahasa Yunani (*sustēma*) adalah suatu kesatuan yang terdiri komponen atau elemen yang dihubungkan bersama untuk memudahkan aliran informasi, materi atau energi untuk mencapai suatu tujuan. Istilah ini sering dipergunakan untuk menggambarkan suatu set entitas yang berinteraksi, di mana suatu model matematika seringkali bisa dibuat. Sistem pada dasarnya adalah sekelompok unsur yang erat hubungannya satu dengan yang lain, yang berfungsi bersama-sama untuk mencapai tujuan tertentu. Sistem juga merupakan kesatuan bagian-bagian yang saling berhubungan yang berada dalam suatu wilayah serta memiliki item-item penggerak, contoh umum misalnya seperti negara. Negara merupakan suatu kumpulan dari beberapa elemen kesatuan lain seperti provinsi yang saling berhubungan sehingga membentuk suatu negara dimana yang berperan sebagai penggeraknya yaitu rakyat yang berada dinegara tersebut.

Kata "sistem" banyak sekali digunakan dalam percakapan sehari-hari, dalam forum diskusi maupun dokumen ilmiah. Kata ini digunakan untuk banyak hal, dan pada banyak bidang pula, sehingga maknanya menjadi beragam. Dalam pengertian yang paling umum, sebuah sistem adalah sekumpulan benda yang memiliki hubungan di antara mereka.

2.1.1. Karakteristik Sistem

2.1.1.1. Komponen atau Elemen

Suatu sistem terdiri dari komponen yang saling berinteraksi, artinya saling bekerja sama membentuk satu kesatuan. Komponen-komponen dari suatu sistem dikenal dengan subsistem. Subsistem memiliki sifat-sifat dari sistem itu sendiri dalam menjalankan suatu fungsi tertentu dan mempengaruhi proses sistem secara

keseluruhan. Suatu sistem juga memiliki sistem yang lebih besar yang dikenal dengan suprasistem.

2.1.1.2. Batas Sistem

Batas sistem merupakan daerah yang membatasi antara sistem yang satu dengan sistem yang lainnya atau dengan lingkungan luarnya. Dengan adanya batas sistem maka sistem dapat membentuk suatu kesatuan, karena dengan batas sistem ini, fungsi dan tugas dari subsistem satu dengan yang lainnya berbeda tetapi tetap saling berinteraksi. Dengan kata lain batas sistem merupakan ruang lingkup atau *scope* dari sistem atau subsistem itu sendiri.

2.1.1.3. Lingkungan Luar Sistem

Lingkungan luar sistem adalah segala sesuatu diluar batas sistem yang mempengaruhi operasi suatu sistem. Lingkungan luar sistem dapat bersifat menguntungkan atau merugikan. Lingkungan luar sistem yang bersifat menguntungkan harus dipelihara dan dijaga agar tidak hilang pengaruhnya, sedangkan lingkungan yang bersifat merugikan harus dimusnahkan agar tidak mengganggu operasi sistem.

2.1.1.4. Penghubung Sistem

Penghubung sistem merupakan suatu media penghubung antara satu subsistem dengan subsistem lainnya yang membentuk satu kesatuan, sehingga sumber-sumber daya mengalir dari subsistem yang satu ke subsistem lainnya. Dengan kata lain melalui penghubung *output* dari subsistem akan menjadi *input* bagi subsistem lainnya.

2.1.1.5. *Input*, Proses dan *Output*

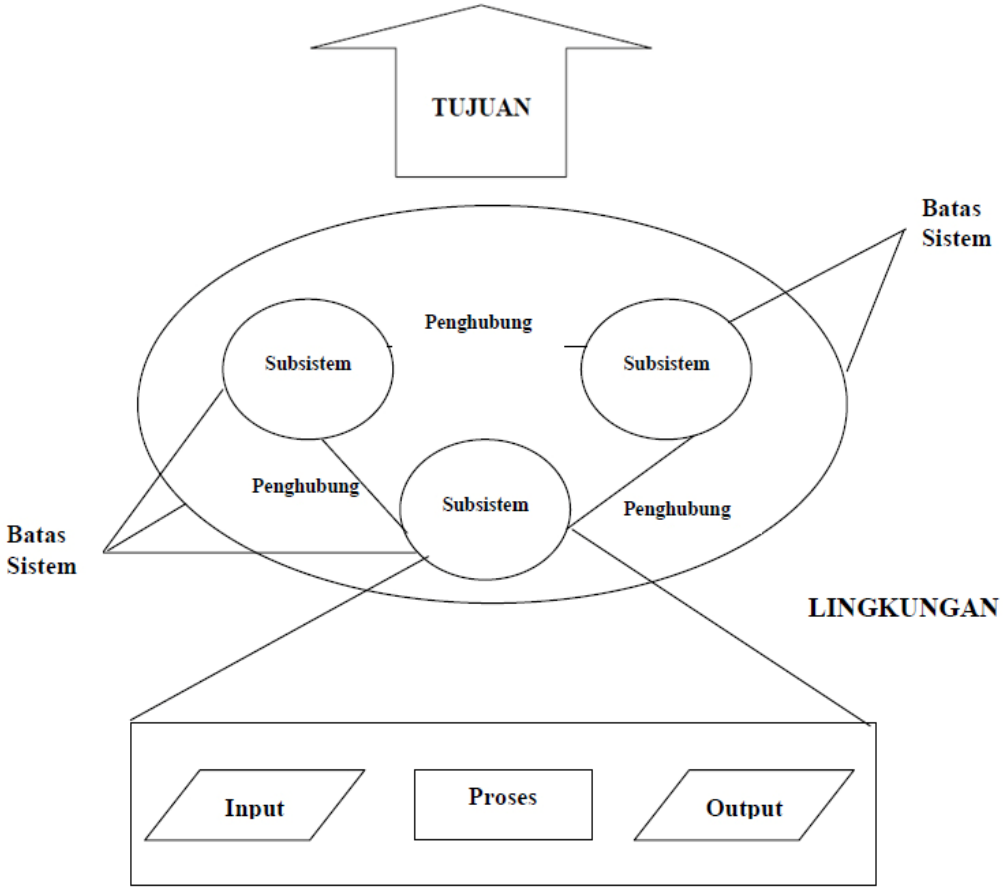
Input adalah energi atau sesuatu yang dimasukkan kedalam suatu sistem yang dapat berupa masukkan yaitu energi yang dimasukkan supaya sistem dapat beroperasi atau masukkan sinyal yang merupakan energi yang diproses untuk menghasilkan suatu keluaran. Sistem mempunyai bagian *pengolah* yang akan

mengubah *input* menjadi *output* dan sistem juga menghasilkan suatu keluaran atau *output* yang berasal dari energi yang diolah.

2.1.1.6. Tujuan

Setiap sistem mempunyai tujuan atau sasaran yang mempengaruhi *input* yang dibutuhkan dan *output* yang dihasilkan. Dengan kata lain, suatu sistem akan dikatakan berhasil apabila hasil operasi sistem tersebut mengenai sasaran atau tujuan yang telah ditetapkan.

atau



Gambar 2.1. Karakteristik Sebuah Sistem

Sistem terdiri dari struktur dan proses, dimana struktur sistem merupakan komponen-komponen yang membentuk sistem tersebut. Sedangkan proses sistem menjelaskan cara kerja setiap unsur sistem dalam mencapai tujuan sistem. Sistem dibuat untuk menangani sesuatu yang rutin atau berulang kali terjadi.

2.2. Sistem Informasi

Sistem Informasi (SI) adalah kombinasi dari teknologi informasi dan aktivitas orang yang menggunakan teknologi itu untuk mendukung operasi dan manajemen. Dalam arti yang sangat luas, istilah sistem informasi yang sering digunakan merujuk kepada interaksi antara orang, proses algoritmik, data, dan teknologi. Dalam pengertian ini, istilah ini digunakan untuk merujuk tidak hanya pada penggunaan organisasi teknologi informasi dan komunikasi (TIK), tetapi juga untuk cara di mana orang berinteraksi dengan teknologi ini dalam mendukung proses bisnis.

Ada yang membuat perbedaan yang jelas antara sistem informasi, dan komputer sistem TIK, dan proses bisnis. Sistem informasi yang berbeda dari teknologi informasi dalam sistem informasi biasanya terlihat seperti memiliki komponen TIK. Hal ini terutama berkaitan dengan tujuan pemanfaatan teknologi informasi. Sistem informasi juga berbeda dari proses bisnis. Sistem informasi membantu untuk mengontrol kinerja proses bisnis.

Alter berpendapat untuk sistem informasi sebagai tipe khusus dari sistem kerja. Sistem kerja adalah suatu sistem di mana manusia dan/atau mesin melakukan pekerjaan dengan menggunakan sumber daya untuk memproduksi produk tertentu dan/atau jasa bagi pelanggan. Sistem informasi adalah suatu sistem kerja yang kegiatannya ditujukan untuk pengolahan (menangkap, transmisi, menyimpan, mengambil, memanipulasi dan menampilkan) informasi.

Dengan demikian, sistem informasi antar-berhubungan dengan sistem data di satu sisi dan sistem aktivitas di sisi lain. Sistem informasi adalah suatu bentuk komunikasi sistem di mana data yang mewakili dan diproses sebagai bentuk dari memori sosial. Sistem informasi juga dapat dianggap sebagai bahasa semi formal yang mendukung manusia dalam pengambilan keputusan dan tindakan. Sistem informasi merupakan fokus utama dari studi untuk disiplin sistem informasi dan organisasi informatika.

Sistem informasi adalah gabungan yang terorganisasi dari manusia, perangkat lunak, perangkat keras, jaringan komunikasi dan sumber data dalam mengumpulkan, mengubah, dan menyebarkan informasi dalam organisasi.

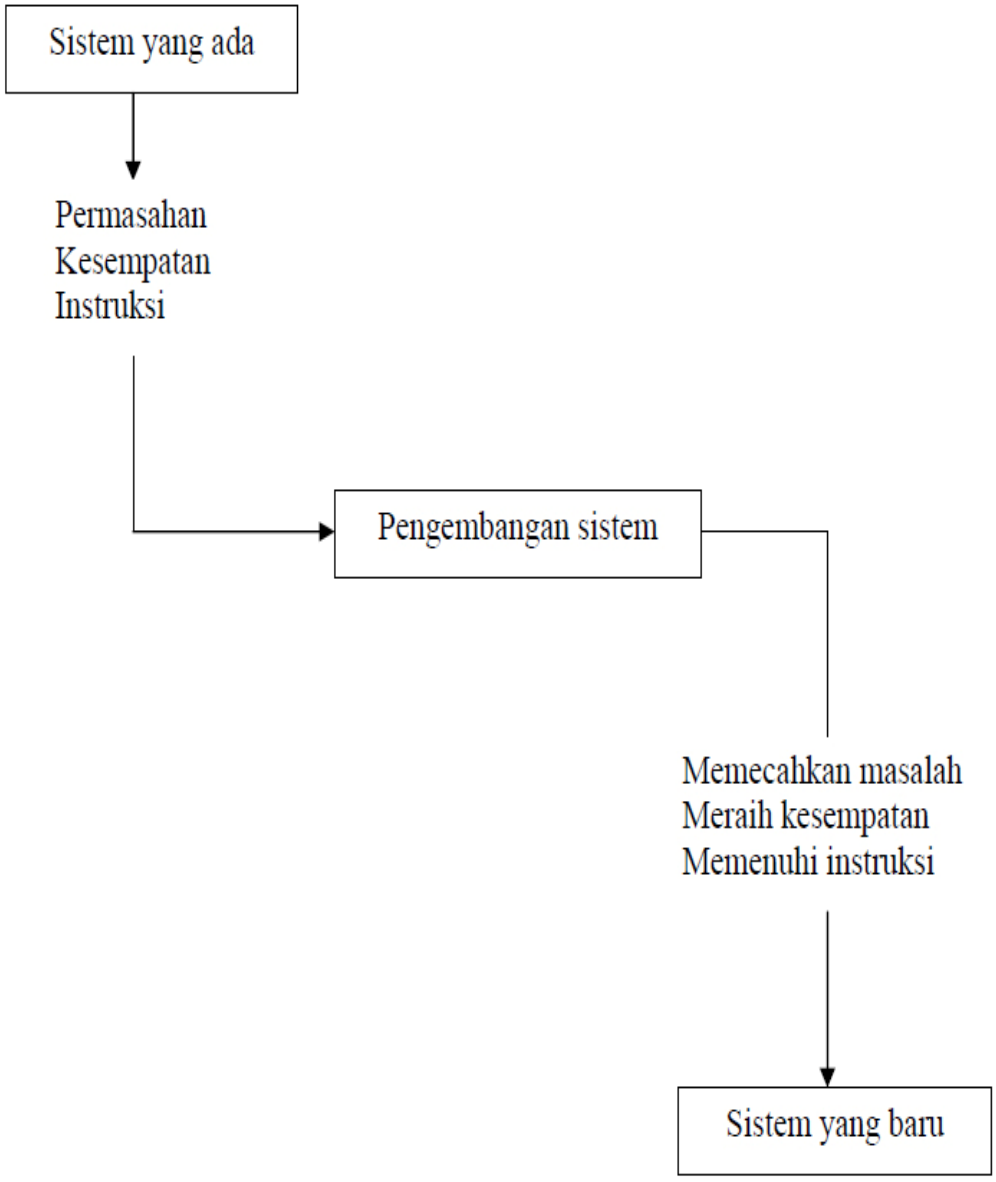
Sistem informasi adalah suatu sistem di dalam suatu organisasi yang mempertemukan kebutuhan pengolahan transaksi harian, mendukung operasi, bersifat manajerial dan kegiatan strategi dari suatu organisasi dan menyediakan pihak luar tertentu dengan laporan-laporan yang diperlukan.

2.3. Pengembangan Sistem

Pengembangan sistem (sistem *development*) dapat berarti menyusun sistem yang baru untuk menggantikan sistem yang lama secara keseluruhan atau memperbaiki sistem yang sudah ada. Sistem yang lama perlu diperbaiki atau diganti disebabkan karena beberapa hal, yaitu sebagai berikut:

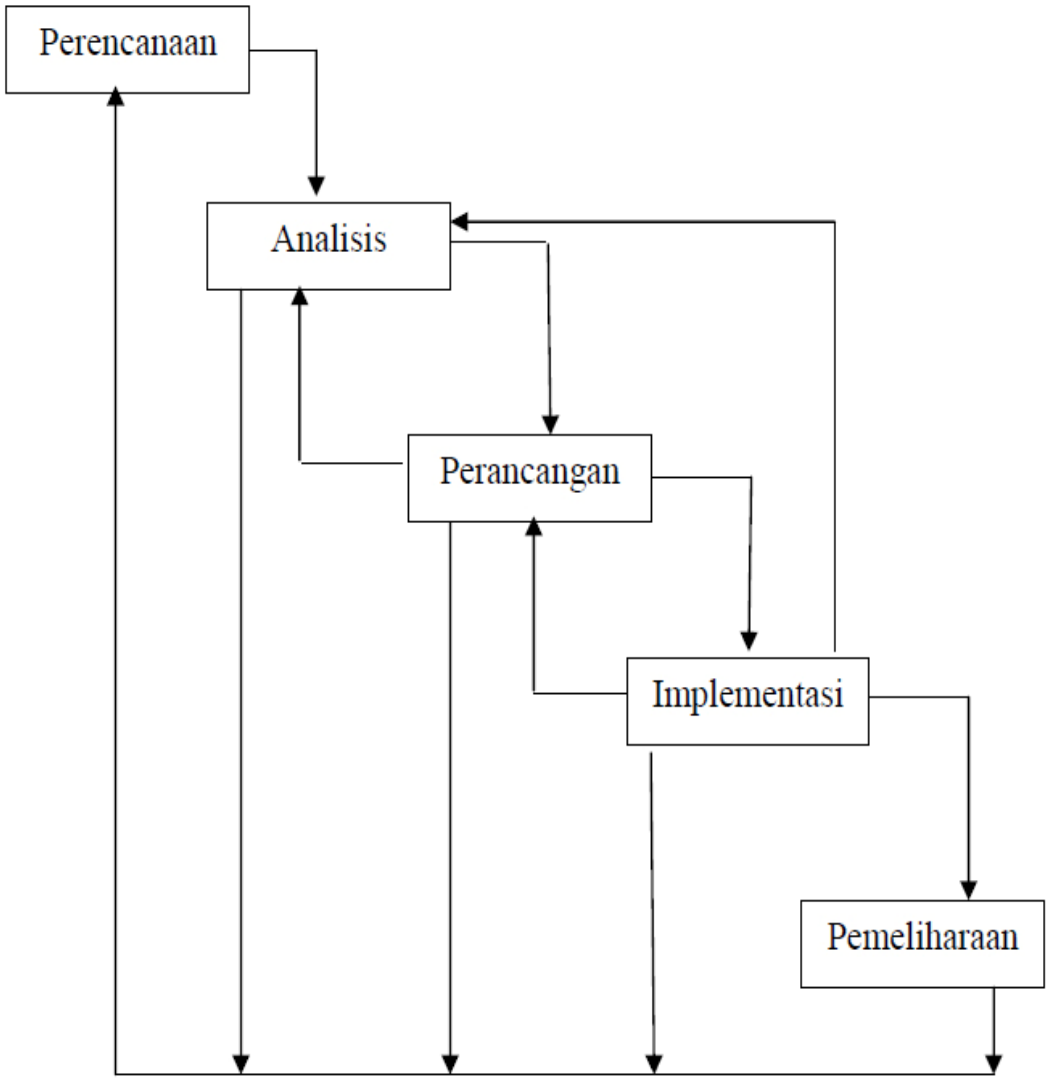
1. Adanya permasalahan-permasalahan *(problems)* yang timbul di sistem yang lama. Permasalahan yang timbul dapat berupa:
    - Ketidakberesan dalam sistem yang lama yang menyebabkan sistem yang lama tidak dapat beroperasi sesuai dengan yang diharapkan.
    - Pertumbuhan organisasi, yang menyebabkan harus disusunnya sistem yang baru. Pertumbuhan organisasi diantaranya adalah kebutuhan informasi yang semakin luas, volume pengolahan data yang semakin meningkat, perubahan prinsip akuntansi atau pengolahan data yang baru.
2. Untuk meraih kesempatan-kesempatan *(opportunities)*, teknologi informasi telah berkembang dengan cepatnya. Perangkat keras komputer, perangkat lunak dan teknologi komunikasi telah begitu cepat berkembang. Organisassi mulai merasakan bahwa teknologi informasi ini perlu digunakan untuk meningkatkan pelayanan informasi sehingga dapat mendukung dalam proses pengambilan keputusan yang akan dilakukan oleh pihak manajemen.
3. Adanya instruksi-instruksi *(directives)*, penyusunan sistem yang baru dapat juga terjadi karena adanya instruksi-instruksi dari atas pimpinan ataupun dari luar organisasi.

Karena adanya permasalahan, kesempatan atau instruksi, maka sistem yang baru perlu dikembangkan untuk emecahkan permasalahan-permasalahan yang timbul, meraih kesempatan-kesempatan yang ada atau memenuhi instruksi yang diberikan.



Gambar 2.2. Pengembangan Sistem Informasi

Sistem yang baik adalah sistem yang selalu menyesuaikan dengan perubahan lingkungan yang terjadi disekitarnya atau sistem tersebut harus dinamis menuju pada keadaan yang lebih baik.



Gambar 2.3. Kerangka Kerja Pengembangan Sistem Informasi

Tahap awal, yaitu tahap perencanaan, menyangkut studi kebutuhan *user*, studi kelayakan baik secara teknis maupun teknologi serta penjadwalan pengembangan suatu proyek sistem informasi.

Tahap berikutnya adalah tahap analisis, yaitu tahap dimana kita berusaha mengenali segenap permasalahan yang muncul pada pengguna, mengenali komponen-komponen sistem, obyek-obyek, hubungan antar obyek dan sebagainya.

Kemudian tahap ketiga adalah tahap perancangan, yaitu tahap dimana kita mencoba mencari solusi permasalahan yang didapat dari tahap analisis.

Tahap keempat adalah tahap implementasi dimana kita mengimplementasikan perancangan sistem ke situasi yang nyata. Pada tahap ini. Kita mulai dengan pemilihan perangkat keras, penyusunan perangkat lunak aplikasi (pengkodean atau *coding*) apakah sistem yang dibuat sudah sesuai dengan kebutuhan *user* atau belum. Jika belum, proses selanjutnya adalah iteratif, yaitu kembali ke tahap-tahap sebelumnya. Disinilah keuntungan metodologi berorientasi obyek mulai terlihat, dimana mulai dari tahap analisis hingga tahap implementasi, kita bisa menggunakan *tools* yang sama sehingga proses iteratif itu dapat berjalan dengan lebih efektif serta efisien ditinjau dari segi uang dan waktu.

Kemudian tahap yang terakhir adalah tahap pemeliharaan atau perawatan, dimana kita bisa mulai melakukan pengoperasian sistem dan jika diperlukan dapat melakukan perbaikan-perbaikan kecil. Kemudian jika waktu penggunaan sistem habis, maka kita akan masuk lagi pada tahap perencanaan.

2.4. Teknologi *Object Oriented*

Teknologi *Object-oriented* merupakan paradigma baru dalam rekayasa *software* yang didasarkan pada obyek dan kelas. Diakui para ahli bahwa *Object-oriented* merupakan metodologi terbaik yang ada saat ini dalam rekayasa *software*. *Object-oriented* memandang *software* bagian per bagian dan menggambarkan satu bagian tersebut dalam satu obyek. Satu obyek dalam sebuah model merupakan suatu fokus selama dalam proses analisis, perancangan dan implementasi dengan menekankan pada *state*, perilaku (*behavior*) dan interaksi obyek dalam model tersebut.

Teknologi obyek menganalogikan sistem aplikasi seperti kehidupan nyata yang didominasi oleh obyek. Manusia adalah obyek, komputer adalah obyek. Obyek memiliki atribut seperti manusia memiliki nama, pekerjaan, rumah, dan lain-lain. Mobil memiliki warna, merek, sejumlah roda, dan lain-lain. Komputer memiliki

kecepatan, sistem operasi, dan lain-lain. Obyek dapat beraksi dan bereaksi. Manusia dapat berjalan, berbicara, makan, minum kemudian mobil dapat berjalan, mengerem dan komputer dapat mengolah data, menampilkan gambar, dan lain-lain.

Keunggulan teknologi obyek dengan demikian adalah bahwa model yang dibuat akan sangat mendekati dunia nyata yang masalahnya akan dipecahkan oleh sistem yang dibangun. Model obyek, atribut dan kelakuan bisa langsung diambil dari obyek yang ada di dunia nyata.

Sistem yang dibangun dengan teknologi obyek memiliki fleksibilitas yang tinggi terhadap perubahan karena menggunakan konsep komponen yang bisa digunakan kembali. Didalam dunia perangkat lunak, penggunaan berulang merupakan hal yang biasa. Contohnya ide dan formula yang hampir sama digunakan berulang oleh *programmer* yang berbeda untuk mengembangkan aplikasi keuangan yang khusus diadaptasikan sesuai kebutuhan dan permintaan masing-masing klien. Oleh karena itu penggunaan berulang suatu obyek merupakan hal yang seharusnya bisa dilakukan. Suatu obyek bisa diambil untuk dimodifikasi berupa penambahan atau pengulangan untuk memecahkan suatu masalah baru.

Ada empat prinsip dasar dari pemrograman berorientasi obyek, yaitu Abstraksi, enkapsulasi, modularitas dan hirarki. Abstraksi memfokuskan perhatian pada karakteristik obyek yang paling penting dan paling dominan yang bisa digunakan untuk membedakan obyek tersebut dari obyek lainnya. Dengan Abstraksi ini developer bisa menerapkan konsep KIS (*Keep It Simple*) pada obyek didunia nyata yang memiliki tingkat kerumitan yang tinggi.

Enkapsulasi menyembunyikan banyak hal yang terdapat dalam obyek yang tidak perlu diketahui oleh obyek lain. Dalam praktek pemrograman, enkapsulasi diwujudkan dengan membuat suatu kelas *interface* yang akan dipanggil oleh obyek lain, sementara didalam obyek yang dipanggil terdapat kelas lain yang mengimplementasikan apa yang terdapat dalam kelas *interface*. Obyek lain hanya

tahu dia perlu memanggil kelas *interface* tanpa perlu tahu proses apa saja yang dilakukan didalam kelas implementasinya dan untuk menuntaskan proses tersebut perlu berhubungan dengan obyek mana saja. Dengan demikian bila terjadi proses perubahan pada proses implementasi maka obyek pemanggil tidak akan terpengaruhi secara langsung.

Modularitas membagi sistem yang rumit menjadi bagian-bagian yang lebih kecil yang bisa mempermudah developer memahami dan mengelola obyek tersebut. Contohnya adalah sistem akademis yang bisa dibagi menjdi kemahasiswaan, daftar mata kuliah, pembayaran uang kuliah, dan lain-lain.

Hirarki berhubungan dengan Abstraksi dan modularitas, yaitu pembagian berdasarkan urutan dan pengelompokkan tertentu. Misalnya untuk menentukan obyek mana yang berada pada kelompok yang sama, obyek mana yang merupakan komponen dari obyek yang memiliki hirarki lebih tinggi. Semakin rendah hirarki obyek berarti semakin jauh Abstraksi dilakukan terhadap suatu obyek. Ke empat prinsip dasar ini merupakan hal yang mendasari teknologi obyek yang perlu ditanamkan dalam cara berpikir *developer* berorientasi obyek

.

2.4.1. *Object Oriented Analysis and Design*

Object-oriented mencakup bidang aplikasi yang sangat luas. Para pengguna sistem komputer dan sistem lain yang didasarkan atas teknologi komputer merasakan efek *Object-oriented* dalam bentuk meningkatnya aplikasi *software* yang mudah digunakan dan servis yang lebih fleksibel, yang muncul dalam berbagai bidang industri, seperti dalam perbankan, telekomunikasi, dan sebagainya. Sedangkan bagi *software engineer*, *Object-oriented* berpengaruh dalam bahasa pemrograman, metodologi rekayasa, manajemen proyek, hardware dan sebagainya.

Analisis dan perancangan berorientasi obyek amat sangat perlu dilakukan dalam pengembangan sistem berorientasi obyek. Hanya dengan kemampuan menggunakan bahasa pemrograman berorientasi obyek yang andal, kita dapat

membangun suatu sistem berorientasi obyek, namun sistem aplikasi yang dibangun akan menjadi lebih baik lagi bila langkah awalnya didahului dengan proses analisis dan perancangan berorientasi obyek, terutama untuk membangun sistem yang mudah dipelihara.

Analisis berorientasi obyek adalah metode analisis yang memeriksa *requirements* (syarat atau keperluan yang harus dipenuhi suatu sistem) dari sudut pandang kelas-kelas dan obyek-obyek yang ditemui dalam ruang lingkup permasalahan. Sedangkan perancangan berorientasi obyek adalah metode untuk mengarahkan arsitektur *software* yang didasarkan pada manipulasi obyek-obyek sistem atau subsistem.

2.4.2. Konsep Dasar dalam *Object Oriented Analysis and Design*

2.4.2.1. Obyek

Obyek adalah benda secara fisik atau konseptual, yang dapat kita temui disekeliling kita. *Hardware*, *software*, dokumen atau manusia dan bahkan konsep semuanya adalah contoh obyek. Untuk kepentingan memodelkan perusahaannya, seorang kepala eksekutif akan melihat gedung, karyawan, divisi, dokumen dan keuntungan sebagai obyek. Seorang teknisi mesin akan melihat ban, pintu, mesin, laju tertinggi dan banyaknya bahan bakar sebagai sebuah obyek. Dan seorang *software engineer* akan memandang tumpukan, antrian, jendela dan *check box* sebagai sebuah obyek.

Sebuah obyek memiliki keadaan *(state)* dan perilaku *(behavior). State* dari sebuah obyek adalah kondisi obyek tersebut atau himpunan dari keadaan yang menggambarkan obyek tersebut. Sebagai contoh, bola lampu adalah obyek, dan salah satu keadaan nyala atau tidak nyala adalah *state* dari obyek bola lampu tersebut.

State dinyatakan dengan nilai dari sebuah atribut obyeknya. Atribut adalah nilai internal suatu obyek yang mencerminkan antara lain karakteristik obyek, kondisi

obyek, kondisi sesaat, koneksi dengan obyek lain dan identitas. Perubahan *state* dicerminkan oleh perilaku (*behavior*) obyek tersebut.

Behavior suatu obyek mendefinisikan bagaimana sebuah obyek bertindak (beraksi) dan memberikan reaksi. *Behavior* ditentukan oleh himpunan semua atau beberapa operation yang dapat dilakukan dalam obyek itu sendiri. *Behavior* dari sebuah obyek dicerminkan oleh *interface, service dan method* dari obyek tersebut. *Interface* adalah pintu untuk mengakses service obyek. *Service* adalah fungsi yang bisa diemban obyek. *Method* adalah mekanisme internal obyek yang mencerminkan perilaku (*behavior*) obyek tersebut. Sebagai contoh, jika printer merupakan sebuah obyek maka perilaku (*behavior*) atau servisnya mencetak apapun yang dia terima.

2.4.2.2. Kelas (*Class*)

Kelas *(Class)* adalah definisi umum (pola, *template* atau cetak biru) untuk himpunan obyek sejenis. Kelas menetapkan spesifikasi perilaku (*behavior*s) dan atribut obyek-obyek tersebut. *Class* adalah Abstraksi dari entitas dalam dunia nyata. Obyek adalah contoh dari sebuah kelas. Sebagai contoh, atribut untuk kelas binatang adalah berkaki empat dan memiliki ekor. Perilakunya adalah makan dan tidur. Contoh yang mungkin dari kelas binatang ini adalah kucing, gajah dan kuda.

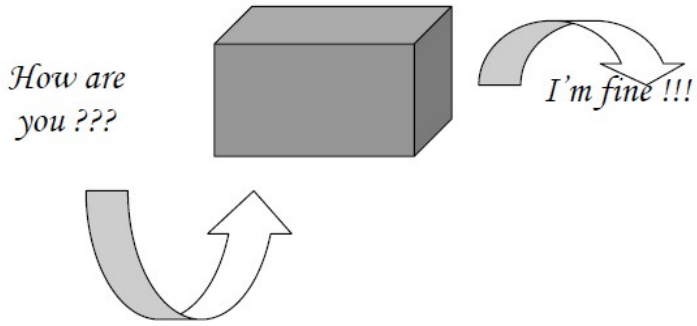
2.4.2.3. Kotak Hitam dan *Interface*



Gambar 2.4. Sebuah Obyek digambarkan sebagai Kotak Hitam

Encapsulation adalah proses menyembunyikan detil implementasi sebuah obyek. Satu-satunya jalan untuk mengakses data obyek tersebut adalah melalui *interface*. *Interface* melindungi internal *state* sebuah obyek dari campur tangan pihak luar. Oleh karena itu obyek sering digambarkan sebagai kotak hitam (*black box*) yang

menerima dan mengirim pesan-pesan (*message*). Dalam *Object-oriented* programming kotak hitam tersebut berisi kode (himpunan instruksi dengan bahasa yang dipahami komputer) dan data (informasi dimana instruksi tersebut beroperasi dengannya). Dalam *Object-oriented programming*, kode dan data disatukan dalam sebuah benda yang tersembunyi isinya yaitu obyek. Pengguna obyek tidak perlu mengetahui isi dalam kotak tersebut. Untuk dapat berkomunikasi dengan obyek, diperlukan pesan (*message*).



Gambar 2.5. *User* Memakai Pesan dalam Menggunakan Obyek

Secara formal kita mendefinisikan *message* sebagai permintaan untuk obyek penerima (*receiver object*) untuk membawa metode yang ditunjukkan atau perilaku dan mengembalikan *result* dari aksi tersebut kepada obyek pengirim (*sender object*). Sebagai contoh, satu obyek manusia mengirim obyek bola lampu sebuah pesan untuk menyala (melalui saklar). Obyek bola lampu memiliki satu perilaku yang akan mengubah keadaan atau *state* nya dari padam menjadi menyala. Obyek bola lampu menyalakan dirinya dan menunjukkan kepada obyek orang tersebut bahwa *state* barunya adalah menyala.

2.4.2.4. *Association dan Aggregation*

Association (asosiasi) adalah hubungan antar obyek yang saling membutuhkan. Sedangkan *aggregation* (agregasi) adalah bentuk khusus dari asosiasi yang menggambarkan seluruh bagian suatu obyek merupakan bagian dari obyek lainnya. Sebagai contoh, obyek tanggal dapat disusun dari obyek hari, obyek bulan dan obyek tahun.

2.4.3. Keunggulan *Object Oriented* dalam Banyak Kasus

Sekarang ini terdapat beberapa paradigma yang digunakan dalam rekayasa *software*, diantaranya *procedure-oriented*, *Object-oriented*, data *structure-oriented*, data *flow-oriented* dan *constraint oriented*. Tiap-tiap paradigma tersebut cocok untuk beberapa kasus dan bagian dari seluruh kemungkinan ruang lingkup aplikasi, tetapi menurut Booch berdasarkan pengalamannya, *Object-oriented* dapat diaplikasikan dalam seluruh ruang lingkup.

Untuk memahami mengapa OOAD unggul dalam banyak kasus, harus memahami masalah yang dihadapi perusahaan rekayasa *software*. Pertama, *software* sulit untuk dimodifikasi bila memerlukan pengembangan. Kedua, proses pembuatan *software* memerlukan waktu yang cukup lama sehingga kadang kala melebihi anggaran dalam pembuatannya. Ketiga, para *programmer* selalu membuat *software* dari dasar karena tidak adanya kode yang bisa digunakan ulang (*reuse*).

Kebanyakan perusahaan tersebut sebelumnya telah menggunakan *pemrograman structural*. Metode *structural* menggunakan pendekatan dengan pendekatan *top-down*, yang mana pendekatan ini memecahkan masalah dengan membagi masalah kedalam komponen-komponen hingga didapatkan komponen yang tidak dapat dibagi-bagi lagi. Pendekatan dengan cara *top-down* ini telah membuat peningkatan dalam kualitas *software*, tetapi dalam sistem yang berskala besar, pendekatan ini sering menemui banyak masalah. Pemrograman *structural* seringkali tidak dapat mendesain apa yang akan terjadi dalam sistem yang telah selesai tanpa mengimplementasikan sistem terlebih dahulu. Jika ditemui kesalahan dalam sistem, maka desain harus disusun ulang dari awal sampai akhir. Hal ini akan terjadi pemborosan biaya dan waktu.

Perbedaan antara metode *structural* dan OOAD terletak pada bagaimana data dan fungsi disimpan. Dalam metode *structural*, data dan fungsi disimpan terpisah. Biasanya semua data ditempatkan sebelum fungsi ditulis. Seringkali tidak terpikirkan sebelumnya untuk mengetahui data mana yang digunakan dalam suatu

fungsi tertentu. Tetapi dalam OOAD, data dan fungsi yang berhubungan dalam suatu obyek disimpan bersama-sama dalam satu kesatuan.

Orang akan lebih mudah memahami sistem sebagai obyek daripada prosedur, karena biasanya orang berpikir dalam bentuk obyek. Sebagai contoh, orang melihat mobil sebagai sebuah sistem yang memiliki mesin, roda , stir, tank bahan bakar dan sebagainya. Kebanyakan orang tidak melihat mobil sebagai sebuah urutan prosedur yang membuat mobil tersebut dapat berjalan. Karena secara alamiah lebih mudah memikirkan suatu sistem dalam bentuk obyek-obyek, tidak heran kalau OOAD menjadi lebih terkenal.

Satu alasan mengapa *Object Oriented* menguntungkan bagi *programmer* adalah karena *programmer* dapat mendesain program dalam bentuk obyek-obyek dan hubungan antar obyek tersebut untuk kemudian dimodelkan dalam sistem nyata. Keuntungan yang lain adalah proses pembuatan *software* dapat dilakukan dengan lebih cepat karena *software* dibangun dari obyek-obyek standar, dapat menggunakan ulang model yang ada, dan dapat membuat model yang cepat melalui metodologi. Kualitas yang tinggi dari *software* dapat dicapai karena adanya tested *component*. Lebih mudah dalam *maintenance* karena perbaikan kode hanya diperlukan pada satu tempat (bukan diurut dari awal). Mudah dalam membangun sistem yang besar karena subsistem dapat dibuat dan diuji secara terpisah. Mengubah sistem yang sudah ada tidak memerlukan membangun ulang keseluruhan sistem.

2.4.4. Pemrograman Berorientasi Object

Pemrograman berorientasi obyek merupakan kelanjutan dari proses analisis dan desain berorientasi obyek. Dalam pemrograman berorientasi obyek ini, komponen yang didesain dalam proses desain kemudian diimplementasikan dengan menggunakan bahasa pemrograman berorientasi obyek. Syarat sebuah bahasa pemrograman berorientasi obyek adalah bila bahasa pemrograman tersebut memiliki fitur untuk mengimplementasikan ke 4 konsep berorientasi obyek, yaitu Abstraksi, *encapsulation*, *polymorphism* dan *inheritance*. Dikenal beberapa bahasa

pemrograman berorientasi obyek, seperti C++, Java, Visual Basic, dan sebagainya.

2.4.5. Pengetahuan Dasar UML

2.4.5.1. Pengertian UML

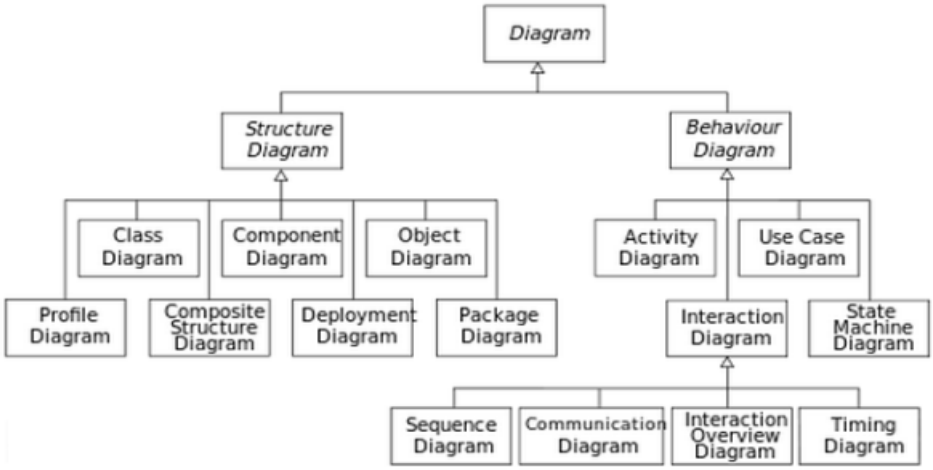
Pemodelan adalah gambaran dari realita yang simpel dan dituangkan dalam bentuk pemetaan dengan aturan tertentu. Pemodelan digunakan untuk menggambarkan desain sistem.

Pada perkembangan teknik pemrograman berorientasi objek, muncullah sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu Unified Modeling Language (UML). UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram.

UML terdiri dari bermacam-macam diagram yg digunakan untuk permodelan pada saat pengembangan sistem mulai dari tahap analisi sampai implementasi. Pada saat melakukan desain sistem, tidak harus semua diagram pada UML diimplementasikan akan tetapi UML merupakan diagram yang saling terkait oleh karena itu perlu adanya kekonsistenan rancangan diagram yang satu dengan lainnya.

Diagram dalam UML dikelompokan menjadi 2 :

1. Diagram Struktur /statis diagram .
2. Diagram prilaku system/behaviour diagram.



Gambar 2.6. Diagram UML.

Penjelesan singkat diagram tersebut antara lain:

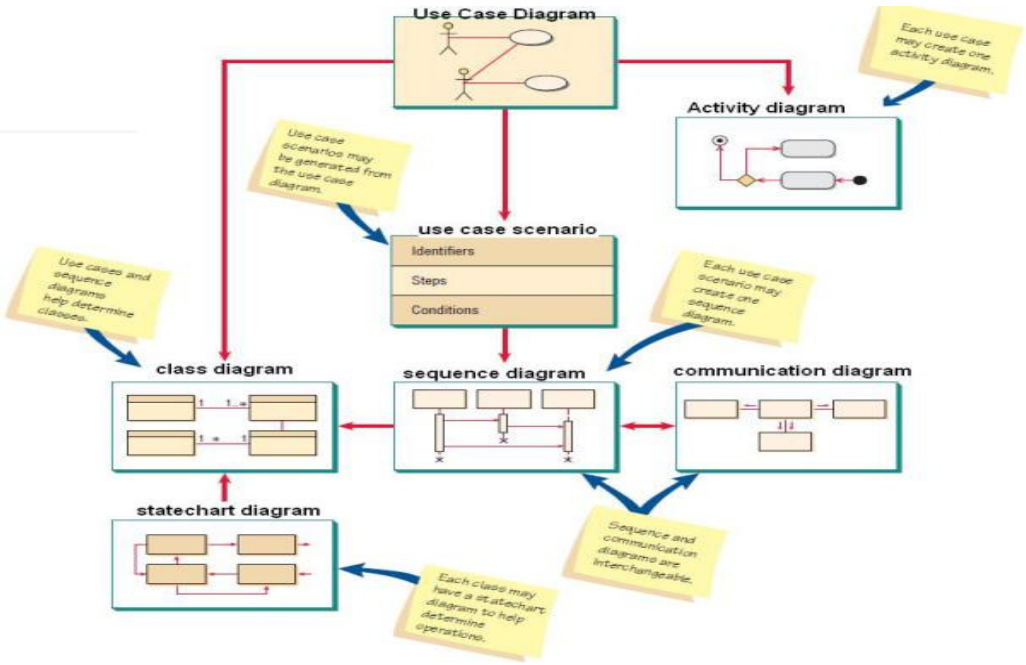
Tabel 2.1. Penjelasan Singkat Diagram Dalam UML.

Nama Diagram	**Digunakan untuk**	**Digunakan pada tahapan:**
Diagram *Class*	Menggambarkan hubungan antara model *class* dalam system.	*Design*
Diagram status	Diagram status menjelaskan aliran kontrol dari satu status ke status lain. Status didefenisikan sebagai suatu kondisi dari suatu obyek yang ada dan perubahan yang terjadi sekiranya ada event yang terpicu..	*Analysis, Design*
Diagram Aktivitas	Menggambarkan hubungan aliran kerja business terlepas dari classes, aliran activitas dlm sebuah use case, atau detail design dari method.	*Analysis, Design*
Diagram *Use Case*	Mendapatkan persyaratan/kebutuhan system dan menggambarkan hubungan antara system dgn lingkungan.	*Analysis, Design*

Nama Diagram	Digunakan untuk	Digunakan pada tahapan:
Diagram *Sequence*	Memodelkan prilaku *objects* dlm sebuah *use case*. Focus pd urutan berdasar waktu dari sebuah *activity*.	*Design*

Diagram yang akan dibahas pada saat ini hanya 4 diagram UML yang efektif biasa dipakai antara lain diagram use case , diagram sekuen, diagram class dan diagram aktifitas.

Gambar berikut dibawah ini menggambarkan diagram diagram tsb dan memperlihatkan bagaimana diagram yang satu membantu membentuk diagram yang lainnya.

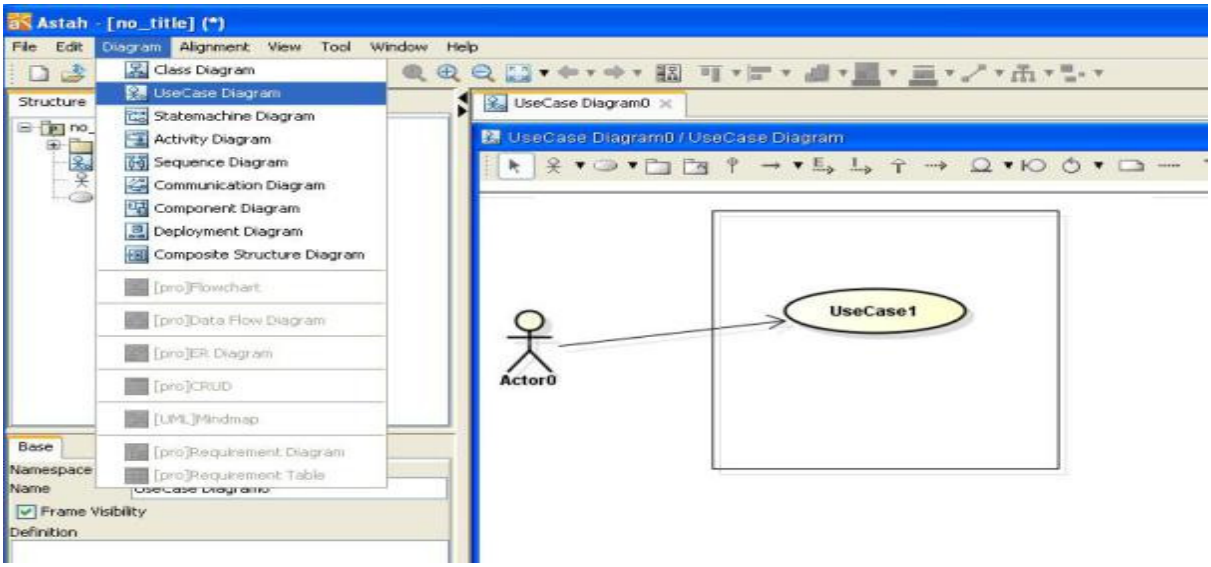


Gambar 2.7. Diagram UML Yang Sering Digunakan.

2.4.5.2. CASE (Computer Aided Software Engineering) Tools

Case tool adalah sejenis software untuk membuat secara otomatis/wizard sebagian atau keseluruan proses depelopment sistem. Untuk membuat berbagai diagram UML baik pada tahap analisis maupun design digunakan Case Tools

diataranya adalah ArgoUML, StartUML dan Astah Comunity. Selain itu anda juga bisa menggunakan Ms Visio . Berikut ini gambar tampilan salah satu case tool yaitu Astah Comunity :



Gambar 2.8. Contoh CASE Tools Pada Astah Comunity.

2.4.5.3. Diagram Dalam UML

2.4.5.3.1. *Use Case Diagram*

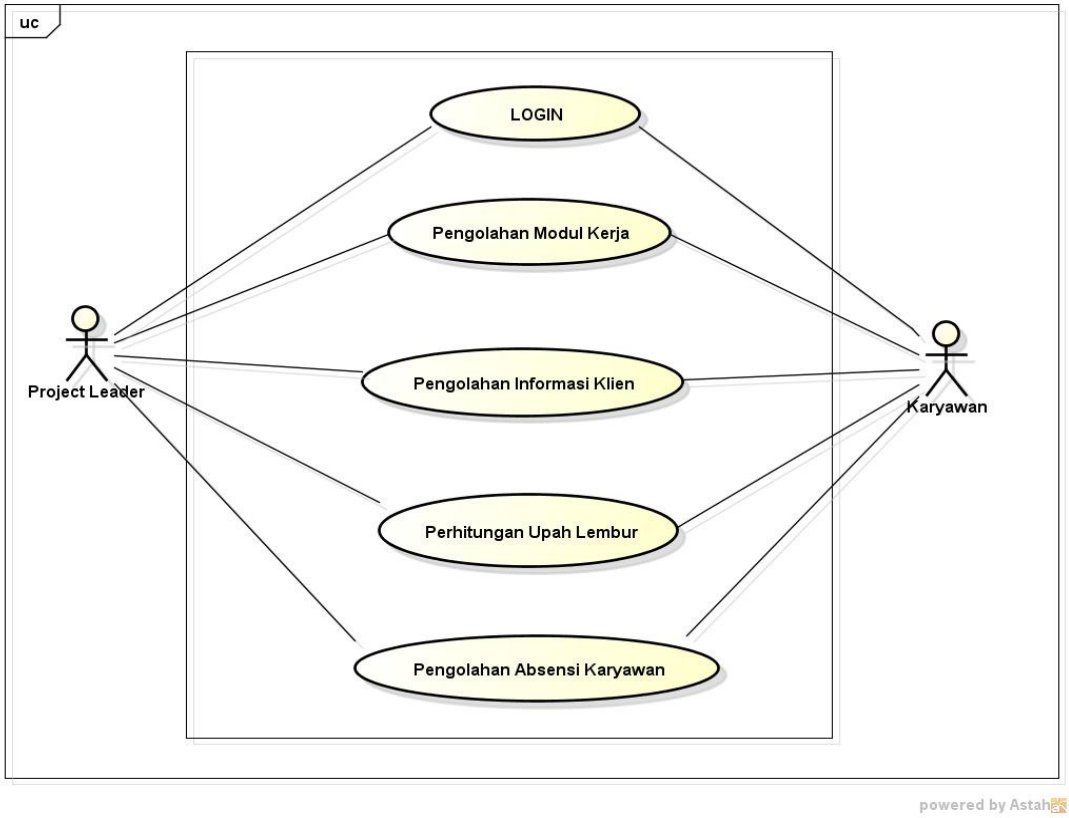
Diagram use case merupakan pemodelan untuk menggambarkan kelakuan (behavior) sistem secara keseluran yang akan dibuat. Diagram use case mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem yang akan dibuat. Dengan pengertian yang cepat, diagram use case digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut.

Use-case modeling/permodelan *use-case* adalah sebuah pendekatan yang memfasilitasi pengembangan berpusatkan kegunaan. Permodelan *use-case* adalah proses permodelan fungsi-fungsi sistem dalam konteks peristiwa-peristiwa bisnis, siapa yang mengawalinya, dan bagaimana sistem itu merespons hal tersebut. Permodelan ini berakar pada permodelan berorentasi-objek, tetapi permodelan *use-case* terkenal di ligkungan pengembangan non-objek.

Use case diagram digunakan untuk memodelkan bisnis proses berdasarkan perspektif pengguna sistem. Use case diagram terdiri atas diagram untuk use case dan *actor*. *Actor* merepresentasikan orang yang akan mengoperasikan atau orang yang behhhrinteraksi dengan sistem aplikasi.

Use case merepresentasikan operasi-operasi yang dilakukan oleh *actor*. Use case digambarkan berbentuk elips dengan nama operasi dituliskan di dalamnya. *Actor* yang melakukan operasi dihubungkan dengan garis lurus ke use case.

Contoh:

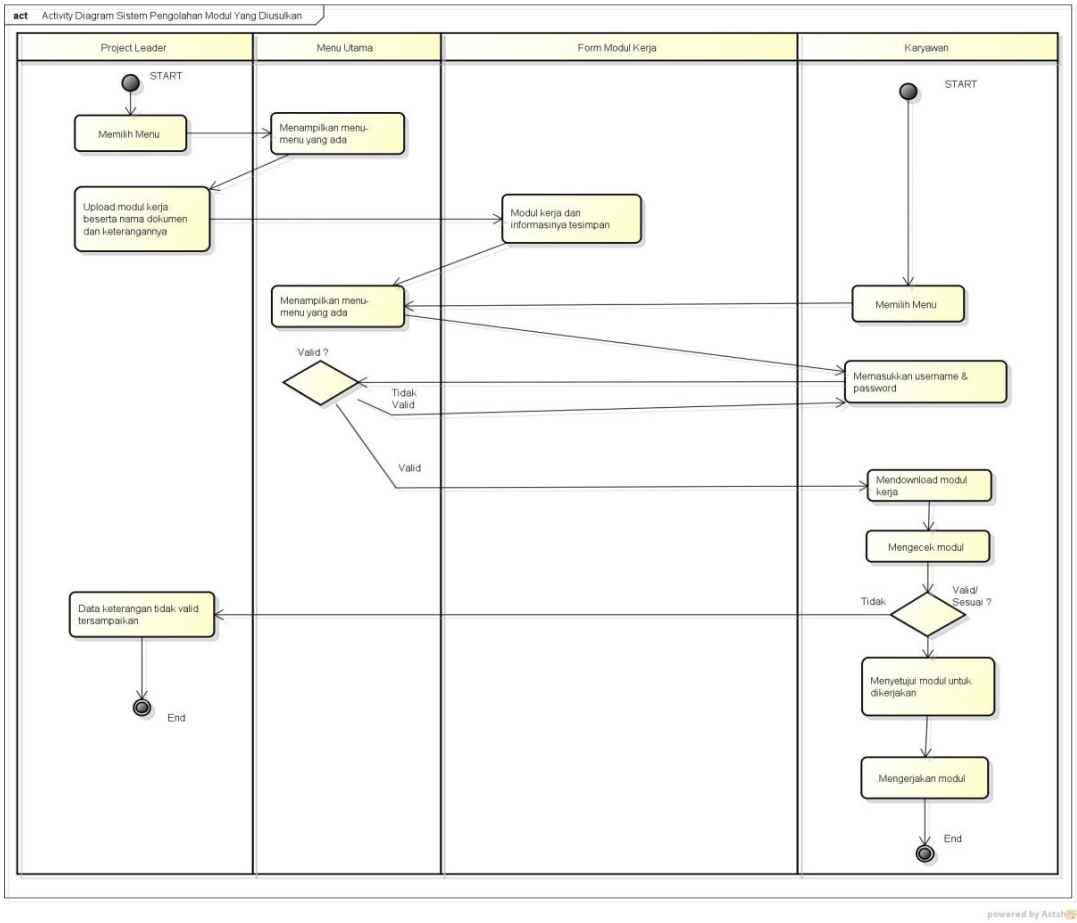


Gambar 2.9. Contoh *Use-Case Diagram.*

2.4.5.3.2. Diagram Aktivitas

Diagram aktivitas atau *activity diagram* menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Yang perlu diperhatikan disini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem.

Contoh:

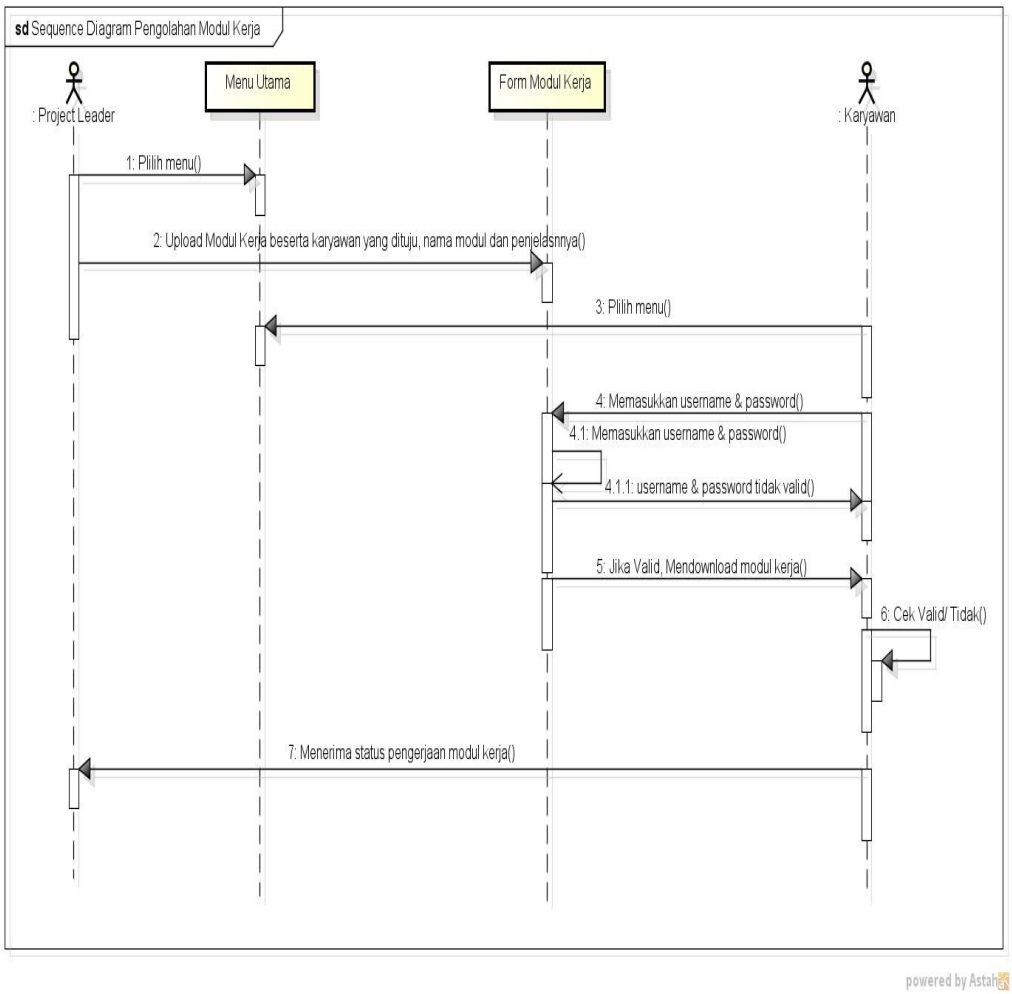


Gambar 2.10. Contoh *Activity Diagram*.

2.4.5.3.3. *Sequence Diagram*

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

Contoh:

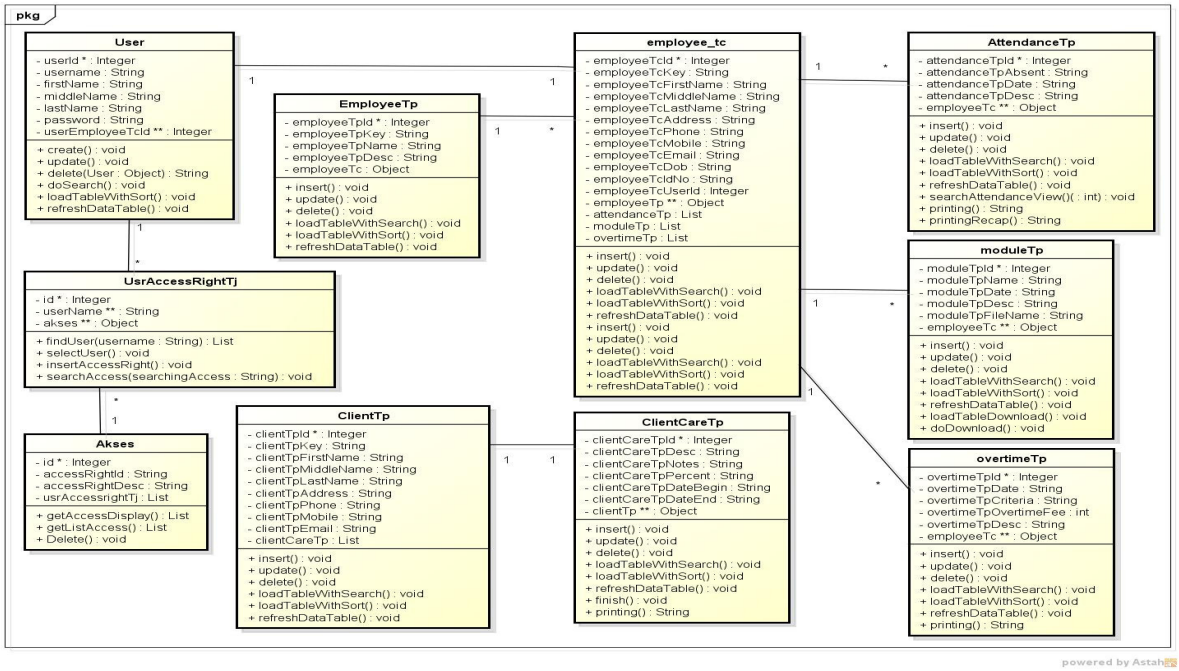


Gambar 2.11. Contoh *Sequence Diagram*.

2.4.5.3.4. Diagram Kelas

Diagram kelas atau class diagram menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. *Class Diagram* adalah sebuah spesifikasi yang jika di instansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class Diagram* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaam tersebut (metoda/fungsi). *Class Diagram* menggambarkan struktur dan deskripsi *Class*, *Package*, dan dan *Object* beserta hubungan satu sama lain seperti containment, pewarisan, asosiasi, dan lain-lain.

Contoh:



Gambar 2.12. Contoh *Class Diagram*.

2.5. *Flowmap*

2.5.1. Pengertian *Flowmap*

Flowmap adalah penggambaran secara grafik dari langkah-langkah dan urut-urutan prosedur dari suatu program.

2.5.2. Langkah Dalam Membuat *Flowmap*

Seorang analis dan programmer akan membuat *flowmap* ada beberapa petunjuk yang harus diperhatikan, seperti :

1. *Flowmap* digambarkan dari halaman atas ke bawah dan dari kiri ke kanan.
2. Aktivitas yang digambarkan harus didefinisikan secara hati-hati dan definisi ini harus dapat dimengerti oleh pembacanya.
3. Kapan aktivitas dimulai dan berakhir harus ditentukan secara jelas.
4. Setiap langkah dari aktivitas harus diuraikan dengan menggunakan deskripsi kata kerja, misalkan MENGHITUNG PAJAK PENJUALAN.
5. Setiap langkah dari aktivitas harus berada pada urutan yang benar.
6. Lingkup dan range dari aktifitas yang sedang digambarkan harus ditelusuri dengan hati-hati. Percabangan-percabangan yang memotong aktivitas yang sedang

digambarkan tidak perlu digambarkan pada *flowmap* yang sama. Simbol konektor harus digunakan dan percabangannya diletakan pada halaman yang terpisah atau hilangkan seluruhnya bila percabangannya tidak berkaitan dengan sistem.

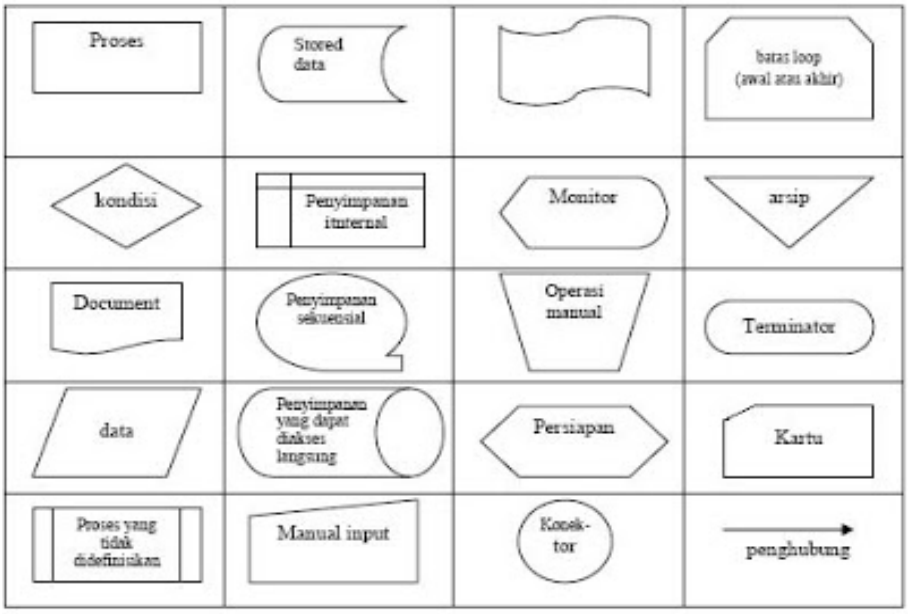
7. Gunakan simbol-simbol *flowmap* yang standar.

2.5.3. Jenis – jenis *Flowmap*

1. *Flowmap* Sistem
2. *Flowmap* Paperwork atau *Flowmap* Dokumen
3. *Flowmap* Skematik
4. *Flowmap* Program
5. *Flowmap* Proses

2.5.4. Simbol – simbol *Flowmap*

Dibawah ini adalah simbol-simbol yang sering digunakan pada pembuatan *flowmap*.



Gambar 2.13. Simbol-simbol *Flowmap*.

1. Simbol Dokumen

Digunakan untuk menggambarkan input-output berupa dokumen yang printable dan berujud, contohnya, barang yang dibeli, KTP, KTM, email (karena printable), kwitansi dan lainnya. Simbol yang digunakan:



Simbol ini ada juga yang mengartikan sebagai simbol computer screen. Jadi kalo ada display di monitor, bisa pake simbol ini. Tapi kebanyakan gak dipake buat hal seperti itu, karena untuk tampilan monitor digunakan simbol display (lihat di simbol lain, di bagian bawah)

2. Simbol Data

Sama halnya dengan dokumen, digunakan untuk menggambarkan *input-output*, hanya saja yang non-printable, contohnya data diri ketika registrasi email, atau hendak memasukkan komen ke sistem, pakailah simbol ini. Simbolnya:



3. Simbol Proses Komputer

Untuk menggambarkan jalannya sistem yang ada. Contohnya, jikalau hendak menggunakan excel, atau menggunakan website buat nyimpen/ insert/ update/ delete data, pakailah simbol ini.



4. Simbol *Storage*

Simbol ini merupakan simbol *input-output*, tapi biasanya harus punya panah masuk dan keluar dari proses yang sifatnya computerized (yang petak segi empat). Simbol yang digunakan adalah:



Simbol ini merupakan simbol storage data secara umum. Terus, jikalau ada yang umum, pasti ada yang khusus bukan? Yah ada. Simbol yang digunakan buat yang khusus yaitu:



Digunakan buat database (jikalau sistemnya make dbms semacam MySql, Oracle, Access dll)



Jikalau anda menyimpan di hard-drive (secondary storage), pakailah simbol ini. Jadi penyimpanannya berupa file. Jangan sampe timbul pertanyaan, “bukannya database juga nyimpennya di hard-drive?”. Jangan sampe yah. Beda kasus itu. Yang ini jika menyimpan dalam bentuk file seperti XML, txt, atau format dokumen lain.



Simbol di atas digunakan jikalau menggunakan internal memori pada sistem, contohnya kalo anda ingi melakukan penyimpanan data di array, pakailah simbol ini.

Karena setiap simbol storage data sudah menunjukkan sifatnya yang repository, jangan sampe anda menaruh header di *flowmap* dengan tulisan repository lalu menjejerkan simbol-simbol di atas. Karena, sekali lagi, header bertujuan untuk menggambarkan PIC (*Person In Charge)*, bukan klasifikasi kegunaan sistem.

5. Simbol Proses Manual:

Digunakan dalam melakukann sesuatu yang tidak ditangani sistem. Contohnya, menulis kwitansi tapi tidak dari computer, tapi pake buku kwitansi. Atau mau nyatet di buku catatan, maka pakailah simbol ini. Simbol yang digunakan:



Simbol ini kadang digabung ama proses computer. Karena ada beberapa mazhab yang gak menganggap eksistensi keberadaan nih simbol proses manual untuk sebuah diagram *flowmap*.

6. Simbol Konektor

Simbol konektor digunakan jika ada suatu proses yang putus di 1 halaman terus pengen disambung ke halaman 2 karena kertasnya gak cukup (singkatnya, tidak bisa disambungin pake garis karena masalah ruang penggambaran). Atau agar garis yang ditarik dari simbol yang satu dengan yang lain tidak saling silang, bisa juga menggunakan konektor. Atau bisa juga untuk melihat hubungan antar proses yang tidak terhubung tapi berkaitan.

Simbol konektor sendiri ada 2. Jika hendak menyambungkan proses di halaman yang berbeda karena kertasnya tidak cukup, maka pakai simbol berikut:



Sedangkan kalo masih dalam halaman yang sama, atau hendak menghubungkan proses tidak berkaitan tapi berhubungan, gunakan simbol ini:



7. Simbol *Decision*

Digunakan jika melakukan pemilihan. Biasanya terkait dengan sudah valid atau belum? Berkas sudah lengkap atau belum? Dan lain-lain. Buat pemilihan ini, anda bisa menggunakan simbol:



8. Simbol Manual *Input*

Biasa juga disebut simbol *keyboard*. Dari namanya kita sudah mengetahui, bahwa di sistem berkaitan dengan memasukkan data menggunakan *keyboard*, pakailah simbol ini. Tapi tidak hanya *keyboard*, *mouse, webcam* dan manual *input* lain, bisa menggunakan simbol ini. Biasanya, simbol ini digunakan dengan panah arah keluar ke petak segi empat, dan arah panah masuk dari simbol dokumen. Simbolnya adalah:

