### 2.1.1 Karakteristik dan Pengertian Sistem

Istilah sistem bukanlah hal yang asing bagi kebanyakan orang. Pada dasarnya, *sistem* adalah sekumpulan elemen yang saling terkait atau terpadu yang dimaksudkan untuk mencapai suatu tujuan. Sebagai gambaran, jika dalam sebuah sistem terdapat elemen yang tidak memberikan manfaat dalam mencapai tujuan yang sama, maka elemen tersebut dapat dipastikan bukanlah bagian dari sistem.

Dari keberadaan ini dapat ditentukan bahwa sebuah sistem memiliki karakteristik :

- Pertama , sebuah sistem memiliki komponen/elemen yang sering disebut *subsistem.*
- Kedua , memiliki batas sistem yang jelas.
- Ketiga , memiliki lingkungan luar sistem.
- Keempat , adanya suatu penghubung sistem.
- Kelima , memiliki input, proses dan output.
- Keenam , yang dapat kita jadikan pedoman apakah sebuah sistem berhasil atau tidak yakni adanya *sasaran atau tujuan.*

#### 2.1.1.1 Komponen / 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

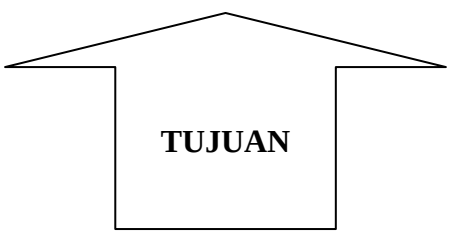
*Penghubung sistem* merupakan suatu media penghubung antara satu subsistem dengan subsistem lainnya yang membentuk satu kesatuan, sehingga sumber-sumber daya mengalir dari subsistem yang satu ke subsistem lainnya. Dengan kata lain melalui penghubung output dari subsistem akan menjadi input bagi subsistem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

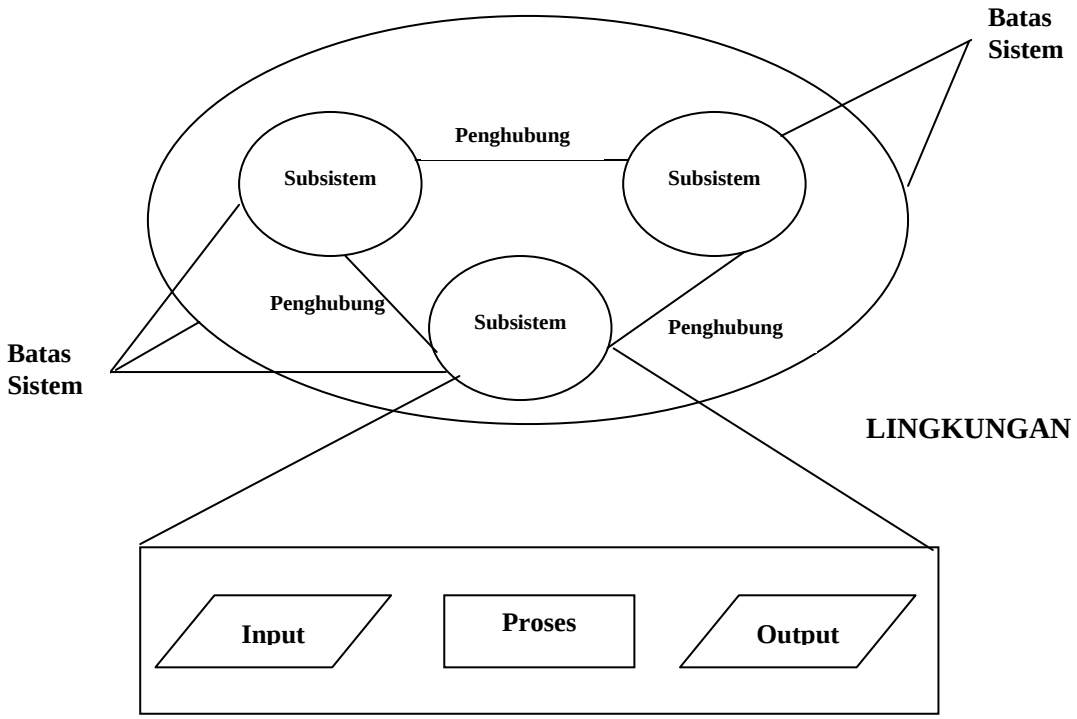
### 2.1.1.5 Input, Proses dan Output

*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.






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.

### 2.1.2 Pengertian Sistem Informasi

Ada beragam definisi sistem informasi, sebagaimana tercantum dalam tabel 2.1. Dari berbagai definisi tersebut, dapat disimpulkan bahwa sistem informasi mencakup *sejumlah komponen* (manusia, komputer, teknologi informasi

dan prosedur kerja), ada *sesuatu yang diproses* (data menjadi informasi) dan dimaksudkan untuk *mencapai tujuan atau sasaran* tertentu.

| SUMBER | DEFINISI |
|---|---|
| Alter (1992) | Sistem informasi adalah kombinasi antara prosedur kerja, informasi, orang dan teknologi informasi yang diorganisasikan untuk mencapai tujuan dalam sebuah organisasi. |
| Bodnar dan Hopwood (1993) | Sistem informasi adalah kumpulan perangkat keras dan perangkat lunak yang dirancang untuk mentransformasikan data ke dalam bentuk informasi yang berguna. |
| Gelinas, Oram dan Wiggins (1990) | Sistem informasi adalah suatu sistem buatan manusia yang secara umum terdiri dari sekumpulan komponen berbasis komputer dan manual yang dibuat untuk menghimpun, menyimpan dan mengelola data serta menyediakan informasi keluaran kepada para pemakai. |
| Hall (2001) | Sistem informasi adalah sebuah rangkaian prosedur formal dimana data dikelompokkan, diproses menjadi informasi dan didistribusikan kepada pemakai. |
| Turban, McLean dan Wetherbe (1999) | Sebuah sistem informasi mengumpulkan, memproses, menyimpan, menganalisis dan menyebarkan informasi untuk tujuan yang spesifik. |

| | |
|---|---|
| Wilkinson ( 1992) | Sistem informasi adalah kerangka kerja yang mengkoordinasikan sumber daya (manusia, komputer) untuk mengubah masukan (input) menjadi keluaran (informasi) guna mencapai sasaran-sasaran perusahaan. |

Tabel 2.1 Definisi Sistem Informasi

*Sistem Informasi Manajemen* adalah sekumpulan sistem informasi yang saling berinteraksi, yang memberikan informasi baik untuk kepentingan operasi atau kegiatan manajerial (Jogiyanto H.M., 1995)

*Sistem Informasi Manajemen* adalah keterpaduan sistem manusia-mesin untuk menyediakan informasi yang mendukung operasi dan menjamin pengambilan keputusan dalam sebuah organisasi (Gordon B. Davis 1974). Sistem ini menggunakan perangkat keras dan perangkat lunak komputer, prosedur pedoman, model untuk analisa, perencanaan, pengawasan dan pengambilan keputusan dan suatu database.

Yang terpenting dari aspek tersebut adalah keseluruhannya. Karena Sistem Informasi Manajemen akan melintasi seluruh sistem penyedia informasi di berbagai lapisan dalam organisasi. Oleh karena itu perlu ditekankan bahwa *Sistem Informasi Manajemen adalah kumpulan sistem informasi dan bukan sistem keseluruhan (total sistem).*

## 2.2 Pengembangan Sistem

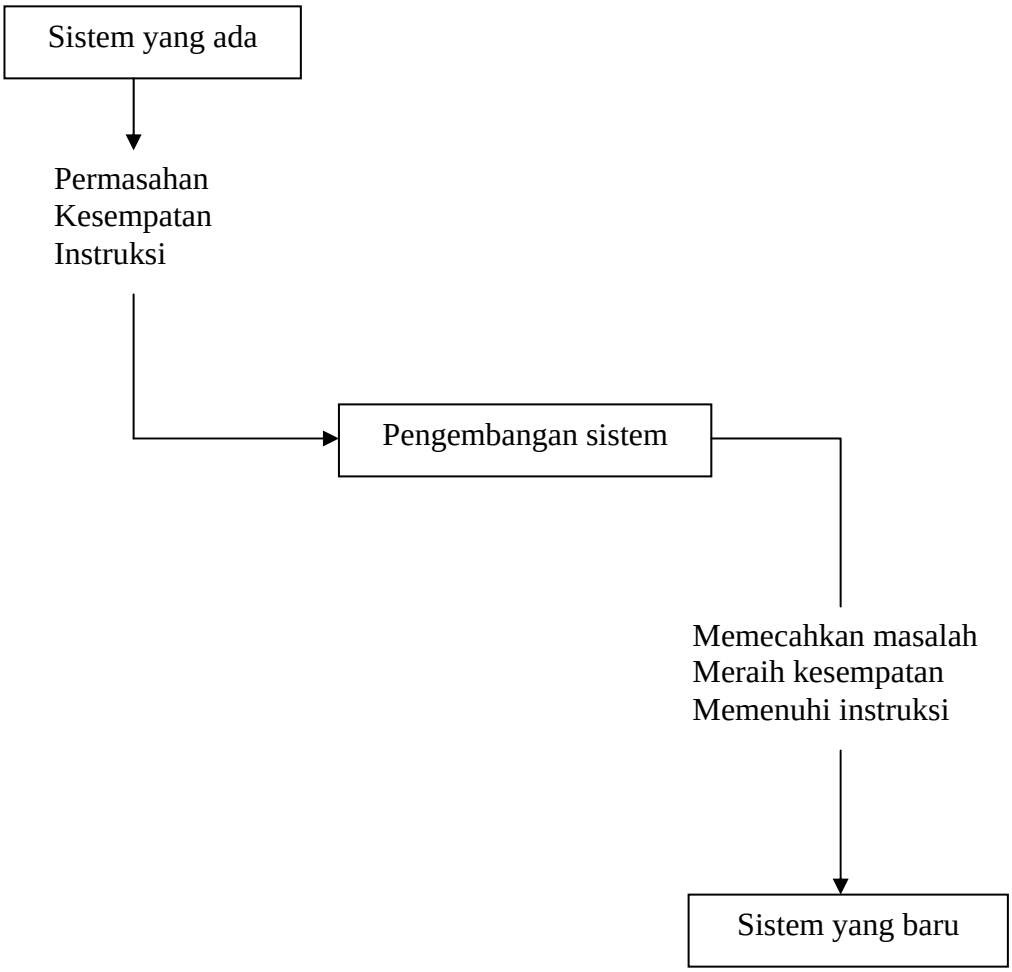
Pengembangan sistem *(sistem development)* dapat berarti menyusun sistem yang baru untuk menggantikan sistem yang lama secara keseluruhan atau

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/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.

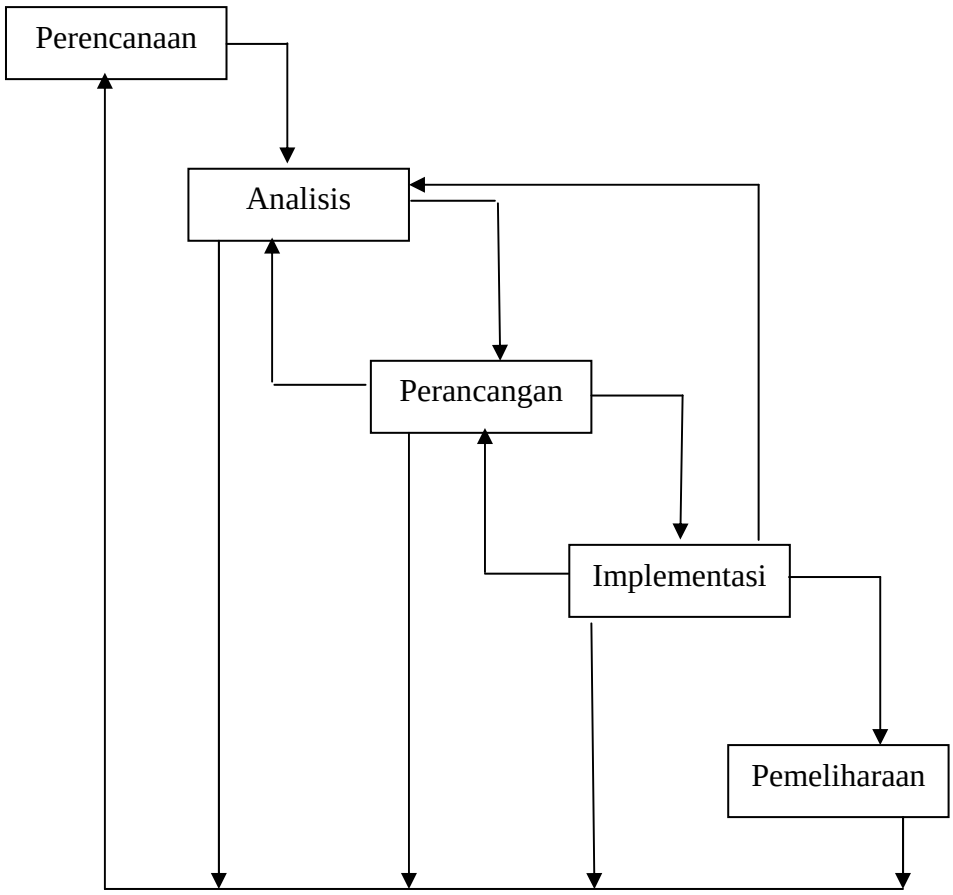
Karena adanya permasalahan, kesempatan atau instruksi, maka sistem yang baru perlu dikembangkan untuk emecahkan permasalahan-permasalahan yang

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/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 tool 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 / 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.

## 2.3 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 : manusia memiliki nama, pekerjaan, rumah, dan lain-lain. Mobil memiliki warna, merk, sejumlah roda, dan lain-lain. Komputer memiliki kecepatan, sistem operasi, dan lain-lain. Obyek dapat beraksi dan bereaksi. Manusia dapat berjalan, berbicara, makan, minum ; mobil dapat berjalan, mengerem ; 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 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/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.

### 2.4.1 Konsep Dasar dalam Object Oriented Analysis and Design

#### 2.4.1.1 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.

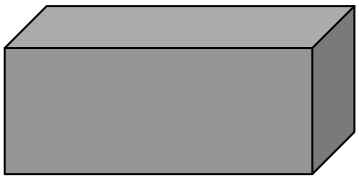
*Behavior* suatu obyek mendefinisikan bagaimana sebuah obyek bertindak (beraksi) dan memberikan reaksi. Behavior ditentukan oleh himpunan semua atau beberapa operation yang dapat dilakukan dalam obyek itu sendiri. Behavior dari sebuah obyek dicerminkan oleh *interface, service dan method* dari obyek 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 service nya mencetak apapun yang dia terima.

#### 2.4.1.2 Kelas (Class)

*Kelas (Class)* adalah definisi umum (pola, template atau cetak biru) untuk himpunan obyek sejenis. Kelas menetapkan spesifikasi perilaku (behaviors) 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

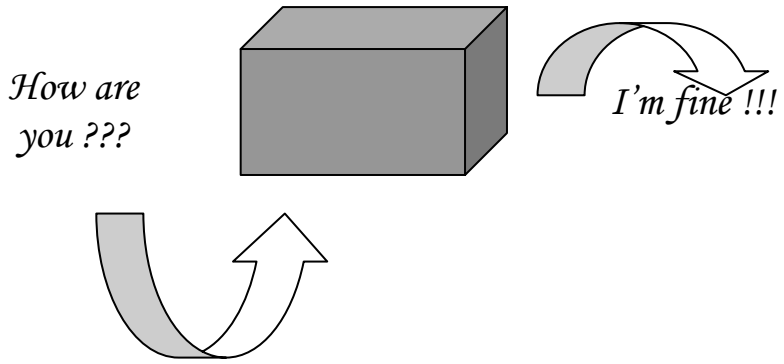
tidur. Contoh yang mungkin dari kelas binatang ini adalah kucing, gajah dan kuda.

### 2.4.1.3 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 statenya dari padam menjadi menyala. Obyek bola lampu menyalakan dirinya dan menunjukkan kepada obyek orang tersebut bahwa state barunya adalah menyala.

### 2.4.1.4 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.

### 2.4.2 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 pogrammer selalu membuat software dari dasar karena tidak adanya kode yang bisa digunakan ulang (reuse).

Kebanyakan perusahaan tersebut sebelumnya telah menggunakan pemrogramam structural. Metode structural menggunakan pendekatan dengan pendekatan top-down, yang mana pendekatan ini memecahkan masalah dengan menbagi 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.

### 2.4.3 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[1], encapsulation[2], polymorphism[3] dan inheritance[4].

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

[1] Memfokuskan perhatian pada karakteristik obyek yang paling penting dan paling dominan yang bisa digunakan untuk membedakan obyek tersebut dari obyek lainnya.
[2] Menyembunyikan banyak hal yang terdapat dalam obyek yang tidak perlu diketahui oleh obyek lain.
[3] Konsep yang menyatakan bahwa suatu fungsi yang sama dapat diterapkan dan dapat dimiliki oleh kelas-kelas yang berbeda.
[4] Berbagi atribut dan operasi antar kelas berdasarkan hirarki kelas.

## 2.5 Unified Modelling Language (UML)

### 2.5.1 Sejarah Singkat Unified Modelling Language (UML)

Gambar 2.6 Logo Unified Modelling Language

*Unified Modelling Language* (UML) adalah sebuah "bahasa" yang telah menjadi standar dalam industri untuk menentukan, visualisasi, merancang dan mendokumentasikan artifact[5] dari sistem software, untuk memodelkan bisnis dan sistem non software lainnya. UML merupakan suatu kumpulan teknik terbaik yang telah terbukti sukses dalam memodelkan sistem yang besar dan kompleks.

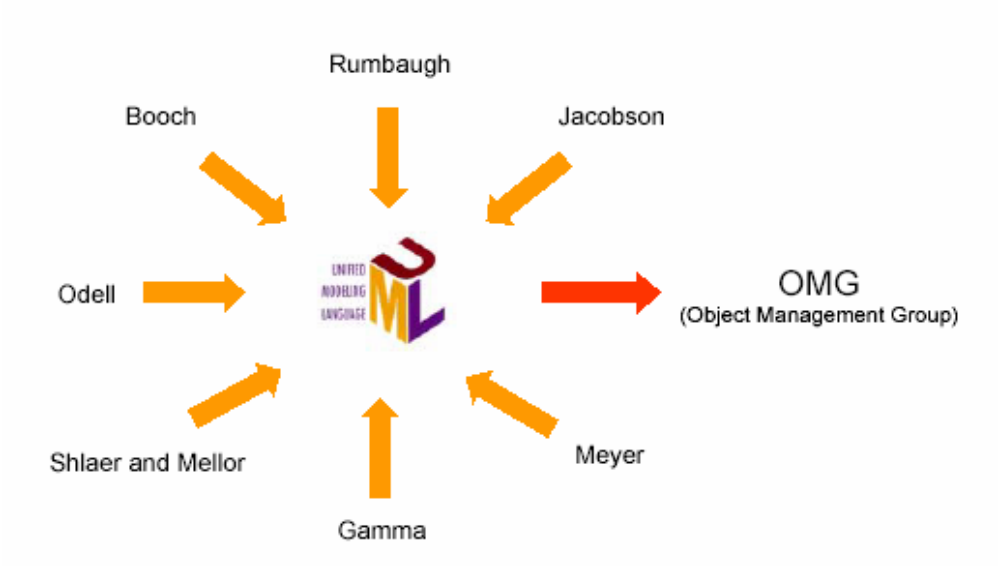
Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa-bahasa berorientasi objek seperti C++, Java, 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

[5] *Artifact* adalah sepotong informasi yang digunakan atau dihasilkan dalam suatu proses rekayasa software. *Artifact* dapat berupa model, deskripsi atau software.

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, dan sebagainya. 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 bekerja sama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan.




Gambar 2.7 Metodologi dalam UML

Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang boleh dikatakan 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.

*Object Management Group, Inc. (OMG)* adalah sebuah organisasi international yang dibentuk pada 1989, didukung lebih dari 800 anggota, terdiri dari perusahaan sistem informasi, software developer, dan pada user sistem

komputer. organisasi ini salah satunya bertugas membuat spesifikasi “manajemen objek” untuk menetapkan kerangka bersama dalam rekayasa software.

Sasaran OMG adalah membantu perkembangan *object-oriented technology* dan mengarahkannya dengan mendirikan Object Management Architecture (OMA). OMA menentukan infrastruktur konseptual yang didasarkan pada seluruh spesifikasi yang dikeluarkan OMG.

OMG kemudian mengeluarkan UML, dimana dengan adanya UML ini diharapkan dapat mengurangi kekacauan dalam bahasa pemodelan yang selama ini terjadi dalam lingkungan industri. UML diharapkan juga dapat menjawab masalah penotasian dan mekanisme tukar menukar model yang terjadi selama ini.

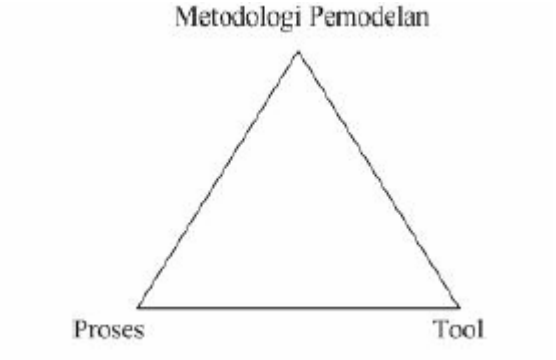
### 2.5.2 Tinjauan Mengenai UML

Saat ini piranti lunak semakin luas dan besar lingkupnya, sehingga tidak bisa lagi dibuat asal-asalan. Piranti lunak saat ini seharusnya dirancang dengan memperhatikan hal-hal seperti scalability, security, dan eksekusi yang robust walaupun dalam kondisi yang sulit. Selain itu arsitekturnya harus didefinisikan dengan jelas, agar bug mudah ditemukan dan diperbaiki, bahkan oleh orang lain selain programmer aslinya. Keuntungan lain dari perencanaan arsitektur yang matang adalah dimungkinkannya penggunaan kembali modul atau komponen untuk aplikasi piranti lunak lain yang membutuhkan fungsionalitas yang sama.

Pemodelan (*modeling*) adalah proses merancang piranti lunak sebelum melakukan pengkodean (*coding*). Model piranti lunak dapat dianalogikan seperti

pembuatan blueprint pada pembangunan gedung. Membuat model dari sebuah sistem yang kompleks sangatlah penting karena kita tidak dapat memahami sistem semacam itu secara menyeluruh. Semakin komplek sebuah sistem, semakin penting pula penggunaan teknik pemodelan yang baik.

Dengan menggunakan model, diharapkan pengembangan piranti lunak dapat memenuhi semua kebutuhan pengguna dengan lengkap dan tepat, termasuk faktor-faktor seperti scalability, robustness, security, dan sebagainya. Kesuksesan suatu pemodelan piranti lunak ditentukan oleh tiga unsur, yang kemudian terkenal dengan sebutan segitiga sukses (*the triangle for success*). Ketiga unsur tersebut adalah metode pemodelan (*notation*), proses (*process*) dan *tool* yang digunakan. Memahami notasi pemodelan tanpa mengetahui cara pemakaian yang sebenarnya (proses) akan membuat proyek gagal. Dan pemahaman terhadap metode pemodelan dan proses disempurnakan dengan penggunaan tool yang tepat.




Gambar 2.8 Kesuksesan suatu Model Piranti Lunak

### 2.5.2.1 Artifact UML

UML menyediakan beberapa *notasi dan artifact standar* yang bisa digunakan sebagai alat komunikasi bagi para pelaku dalam proses analisis dan desain. Artifact didalam UML didefinisikan sebagai informasi dalam bentuk yang digunakan atau dihasilkan dalam proses pengembangan perangkat. Contohnya adalah source code yang dihasilkan oleh proses pemrograman.

Yang harus diperhatikan untuk menjaga konsistensi antar artifact selama proses analisis dan desain adalah bahwa setiap perubahan yang terjadi pada satu artifact harus juga dilakukan pada atifact sebelumnya.

Untuk membuat suatu model, UML memiliki diagram grafis sebagai berikut :

- use case diagram
- class diagram
- behavior diagram
  - statechart diagram
  - activity diagram
  - interaction diagram
    - sequence diagram
    - collaboration diagram
- implementation diagram
  - component diagram
  - deployment diagram

Diagram-diagram tersebut diberi nama berdasarkan sudut pandang yang berbeda-beda terhadap sistem dalam proses analisis atau rekayasa.

Dibuatnya berbagai jenis diagram diatas karena :

- Setiap sistem yang kompleks selalu paling baik jika didekati melalui himpunan berbagai sudut pandang yang kecil yang satu sama lain hampir saling bebas *(independent).* Sudut pandang tunggal senantiasa tidak mencukupi untuk melihat sistem yang besar dan kompleks.
- Diagram yang berbeda-beda tersebut dapat menyatakan tingkatan yang berbeda-beda dalam proses rekayasa.
- Diagram-diagram tersebut dibuat agar model yang dibuat semakin mendekati realitas.

**2.5.2.2 Semantik dalam UML**

*OMG* telah menetapkan semantik (makna istilah) semua notasi UML dalam model struktural dan model behavior. Model struktural (model statis), menekankan stuktur obyek dalam sebuah sistem, menyangkut kelas-kelas, interface, atribut dan hubungan antar komponen. Model behavioral (model dinamis), menekankan perilaku obyek dalam sebuah sistem, termasuk metode, interaksi, kolaborasi dan state history.

### 2.5.2.3 Tujuan UML

Tujuan utama UML diantaranya untuk :

- Memberikan model yang siap pakai, bahasa pemodelan visual yang ekspresif untuk mengembangkan dan saling menukar model dengan mudah dan dimengerti secara umum.
- Memberikan bahasa pemodelan yang bebas dari berbagai bahasa pemrograman dan proses rekayasa.
- Menyatukan praktek-praktek terbaik yang terdapat dalam bahasa pemodelan.

### 2.5.2.4 Cakupan UML

*Pertama*, UML menggabungkan konsep Booch, OMT dan OOSE, sehingga UML merupakan suatu bahasa pemodelan tunggal yang umum dan digunakan secara luas oleh para user ketiga metode tersebut dan bahkan para user metode lainnya.

*Kedua*, UML menekankan pada apa yang dapat dikerjakan dengan metode-meode tersebut.

*Ketiga,* UML berfokus pada suatu bahasa pemodelan standar, bahkan pada proses standar. Meskipun UML harus diaplikasikan dalam konteks sebuah proses, dari pengalaman, bahwa organisasi dan masalah yang berbeda juga memerlukan proses yang berbeda pula.

UML *tidak mencakup* :

- Bahasa Pemrograman

  *UML adalah bahasa pemodelan visual,* bukan dimaksudkan untuk menjadi suatu bahasa pemrograman visual, tetapi UML memberikan arah untuk bergerak kearah kode.

- Tool (software aplikasi) pemodelan

  Membuat standar sebuah bahasa diperlukan oleh tool-tool dan proses. UML mendefinisikan semantik dan notasi, bukan sebuah tool. Contoh tool yang menggunakan UML sebagai bahasanya adalah *Rational Rose dan Enterprise Architect.*

- Proses rekayasa

  UML digunakan sebagai bahasa dalam proyek dengan proses yang berbeda-beda. UML bebas dari proses dan mendefinisikan sebuah proses standar bukan tujuan UML atau RFP dari OMG. Dalam pembahasan ini kita akan menggunakan sebuah proses yang dikeluarkan *Rational Software, yaitu Rational Unified Process (RUP)*

## 2.6 Notasi dalam UML

### 2.6.1 Actor

Gambar 2.9 Notasi Actor

*Actor* menggambarkan segala pengguna software aplikasi (user). Actor memberikan suatu gambaran jelas tentang apa yang harus dikerjakan software aplikasi. Sebagai contoh sebuah actor dapat memberikan input kedalam dan menerima informasi dari software aplikasi, perlu dicatat bahwa sebuah actor berinteraksi dengan use case, tetapi tidak memiliki kontrol atas use case. Sebuah actor mungkin seorang manusia, satu device, hardware atau sistem informasi lainnya.

### 2.6.2 Use Case

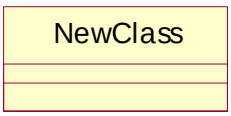
Gambar 2.10 Notasi Use Case

*Use case* menjelaskan urutan kegiatan yang dilakukan actor dan sistem untuk mencapai suatu tujuan tertentu. Walaupun menjelaskan kegiatan, namun use case hanya menjelaskan apa yang dilakukan oleh actor dan sistem bukan bagaimana actor dan sistem melakukan kegiatan tersebut.

- *Use-case Konkret* adalah use case yang dibuat langsung karena keperluan actor. Actor dapat melihat dan berinisiatif terhadapnya
- *Use-case Abstrak* adalah use case yang tidak pernah berdiri sendiri. Use case abstrak senantiasa termasuk didalam (include), diperluas dari (extend) atau memperumum (generalize) use case lainnya.

Untuk menggambarkannya dalam use case model biasanya digunakan association relationship yang memiliki stereotype include, extend atau generalization relationship. Hubungan *include* menggambarkan bahwa suatu use case seluruhnya meliputi fungsionalitas dari use case lainnya. Hubungan *extend* antar use case berarti bahwa satu use case merupakan tambahan fungsionalitas dari use case yang lain jika kondisi atau syarat tertentu terpenuhi.

### 2.6.3 Class




Gambar 2.11 Notasi Class

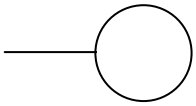
*Class* merupakan pembentuk utama dari sistem berorientasi obyek, karena class menunjukkan kumpulan obyek yang memiliki atribut dan operasi yang sama. Class digunakan untuk mengimplementasikan interface.

Class digunakan untuk mengabstraksikan elemen-elemen dari sistem yang sedang dibangun. Class bisa merepresentasikan baik perangkat lunak maupun perangkat keras, baik konsep maupun benda nyata.

Notasi class berbentuk persegi panjang berisi 3 bagian: persegi panjang paling atas untuk *nama class*, persegi panjang paling bawah untuk *operasi*, dan persegi panjang ditengah untuk *atribut.*

Atribut digunakan untuk menyimpan informasi. Nama atribut menggunakan kata benda yang bisa dengan jelas merepresentasikan informasi yang tersimpan didalamnya. Operasi menunjukkan sesuatu yang bisa dilakukan oleh obyek dan menggunakan kata kerja.

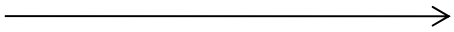
### 2.6.4 Interface



Gambar 2.12 Notasi Interface

*Interface* merupakan kumpulan operasi tanpa implementasi dari suatu class. Implementasi operasi dalam interface dijabarkan oleh operasi didalam class. Oleh karena itu keberadaan interface selalu disertai oleh class yang mengimplementasikan operasinya. Interface ini merupakan salah satu cara mewujudkan *prinsip enkapsulasi* dalam obyek.

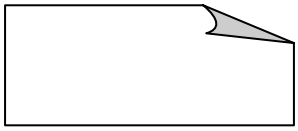
### 2.6.5 Interaction



Gambar 2.13 Notasi Interaction

*Interaction* digunakan untuk menunjukkan baik aliran pesan atau informasi antar obyek maupun hubungan antar obyek. Biasanya interaction ini dilengkapi juga dengan teks bernama operation signature yang tersusun dari nama operasi, parameter yang dikirim dan tipe parameter yang dikembalikan.

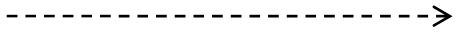
### 2.6.6 Note



Gambar 2.14 Notasi Note

*Note* digunakan untuk memberikan keterangan atau komentar tambahan dari suatu elemen sehingga bisa langsung terlampir dalam model. Note ini bisa disertakan ke semua elemen notasi yang lain.

### 2.6.7 Dependency



Gambar 2.15 Notasi Dependency

*Dependency* merupakan relasi yang menunjukan bahwa perubahan pada salah satu elemen memberi pengaruh pada elemen lain. Elemen yang ada di

bagian tanda panah adalah elemen yang tergantung pada elemen yang ada dibagian tanpa tanda panah.

Terdapat 2 stereotype dari dependency, yaitu include dan extend. *Include* menunjukkan bahwa suatu bagian dari elemen (yang ada digaris tanpa panah) memicu eksekusi bagian dari elemen lain (yang ada di garis dengan panah).

*Extend* menunjukkan bahwa suatu bagian dari elemen di garis tanpa panah bisa disisipkan kedalam elemen yang ada di garis dengan panah.

### 2.6.8 Association

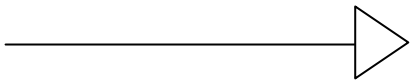
Gambar 2.16 Notasi Asociation

*Association* menggambarkan navigasi antar class (navigation), berapa banyak obyek lain yang bisa berhubungan dengan satu obyek (multiplicity antar class) dan apakah suatu class menjadi bagian dari class lainnya (aggregation).

*Navigation* dilambangkan dengan penambahan tanda panah di akhir garis. *Bidirectional navigation* menunjukkan bahwa dengan mengetahui salah satu class bisa didapatkan informasi dari class lainnya. Sementara *UniDirectional navigation* hanya dengan mengetahui class diujung garis association tanpa panah kita bisa mendapatkan informasi dari class di ujung dengan panah, tetapi tidak sebaliknya.

Aggregation mengacu pada hubungan “has-a”, yaitu bahwa suatu class memiliki class lain, misalnya Rumah memiliki class Kamar.

### 2.6.9 Generalization



Gambar 2.17 Notasi Generalization

*Generalization* menunjukkan hubungan antara elemen yang lebih umum ke elemen yang lebih spesifik. Dengan generalization, class yang lebih spesifik (subclass) akan menurunkan atribut dan operasi dari class yang lebih umum (superclass) atau *"subclass is superclass"*. Dengan menggunakan notasi generalization ini, konsep inheritance dari prinsip hirarki dapat dimodelkan.

### 2.6.10 Realization

Gambar 2.18 Notasi Realization

*Realization* menunjukkan hubungan bahwa elemen yang ada di bagian tanpa panah akan merealisasikan apa yang dinyatakan oleh elemen yang ada di bagian dengan panah. Misalnya class merealisasikan package, component merealisasikan class atau interface.

## 2.7 Mengenal Rational Rose

### 2.7.1 Dasar-dasar Pemodelan dengan Rational Rose

#### 2.7.1.1 Visual Modelling

*Visual Modelling* adalah proses menggambarkan cetak biru suatu sistem secara grafis, terdiri dari komponen-komponen, interfaces dan koneksi-koneksi yang ada dalam sistem tersebut, agar mudah dipahami dan dikomunikasikan.

Visual Modelling dapat membantu kita untuk menampilkan elemen-elemen yang penting secara detil dari suatu masalah yang kompleks dan menyaring untuk kemudian membuang elemen-elemen yang tidak penting.

Rational Rose menggunakan UML sebagai bahasa pemodelannya. Semua semantik dan notasi dalam UML dibuat untuk digunakan dalam visual modelling.

#### 2.7.1.2 Model dalam Rekayasa Software

Sebagaimana telah diketahui dalam mendesain sebuah model dalam proses rekayasa software sangat penting sebagaimana pentingnya memiliki cetak biru untuk membangun suatu bangunan yang besar. Agar model yang dibuat memenuhi kriteria sebagai model yang bagus, maka model yang dibuat harus dapat :

- Mengidentifikassi persyaratan-persyaratan *(requirements)* dan dapat menyampaikan informasi dengan jelas.
- Berfokus pada bagaimana komponen-komponen sistem berinteraksi.
- Membantu kita untuk melihat hubungan antar komponen.

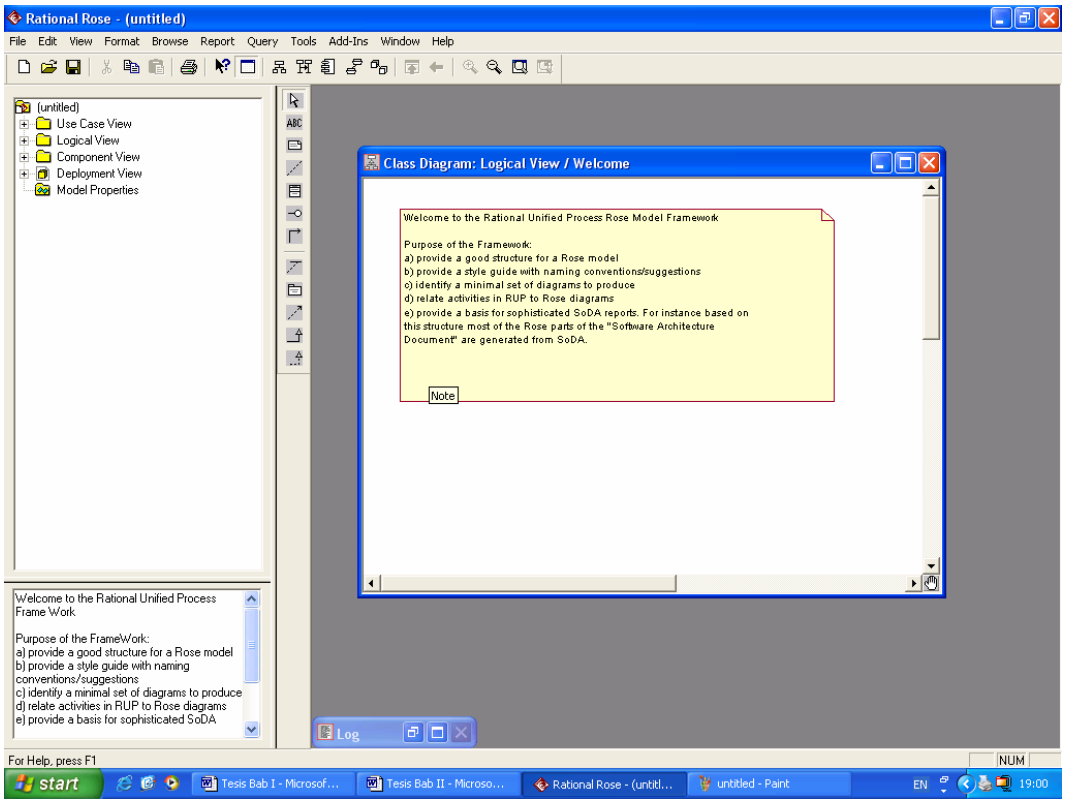
- Meningkatkan komunikasi antara anggota tim dengan menggunakan bahasa yang mudah dipahami, dalam hal ini bahasa grafis.

#### 2.7.1.3 Edisi Rational Rose

Ada tiga edisi Rational Rose, yaitu :

- Rose Modeller, tidak mendukung bahasa pemrograman apapun.
- Rose Profesional, mendukung satu bahasa pemrograman.
- Rose Enterprise, mendukung banyak bahasa pemrograman, yaitu CORBA, VC ++, Visual Basic, Java dan sebagainya.

### 2.7.2 Graphical User Interface (GUI) dalam Rational Rose




Gambar 2.19 Graphical User Interface dalam Rational Rose

Elemen-elemen dasar dalam Grapichal User Interface (GUI) dalam Rational Rose menurut A. Suhendar dan Hariman Gunadi (2002) adalah :

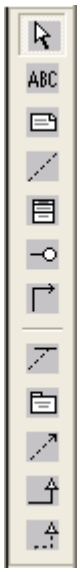
- Toolbar Standard

  *Toolbar standard* pada Rational Rose terdapat pada bagian atas jendela utama dan senantiasa ditampilkan dan tidak bergantung pada tipe diagram yang aktif.


Gambar 2.20 Toolbar Standard

- Toolbox Diagram

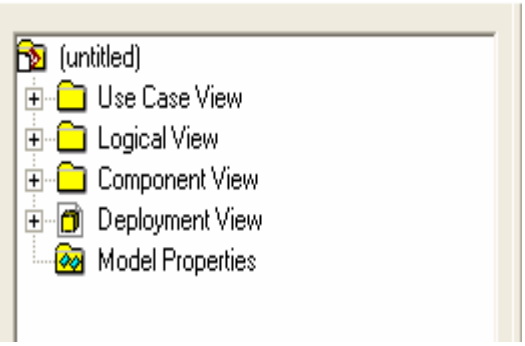
  *Toolbox diagram* pada Rational Rose berubah sesuai dengan jenis diagram yang aktif. Diagram yang aktif ditampilkan dengan title bar warna biru.




Gambar 2.21 Toolbox Diagram

- Browser

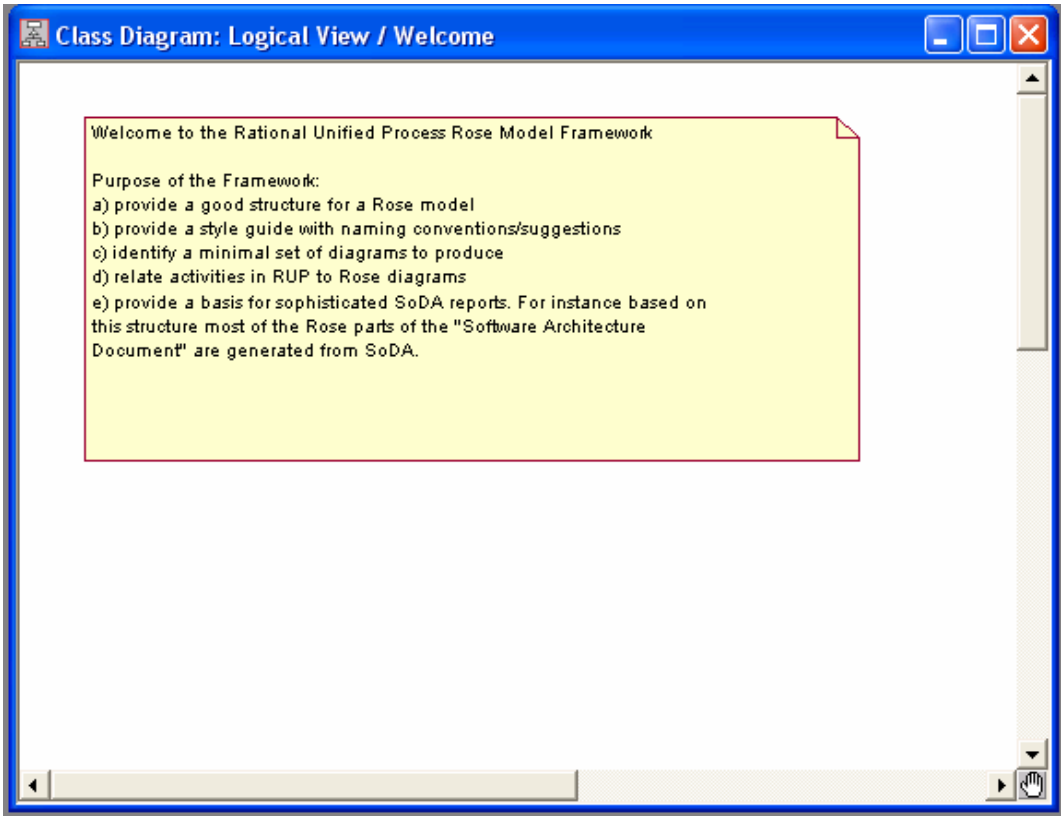
  *Browser* pada Rational Rose membantu kita untuk melihat secara hirarkis elemen-elemen model. Tanda positif (+) menandakan bahwa elemen tersebut mengandung informasi tambahan didalamnya. Dengan menekan tanda (+), informasi tersebut diperluas. Sebaliknya tanda negatif (-) menandakan informasi terbuka secara penuh.




Gambar 2 22 Browser

- Jendela Diagram

  *Jendela Diagram* menampilkan atau memodifikasi diagram dalam jendela diagram. Jika terdapat beberapa diagram yang dibuka pada saat yang sama, dapat ditampilkan secara berlapis (cascade) atau berkotak-kotak (tiled).

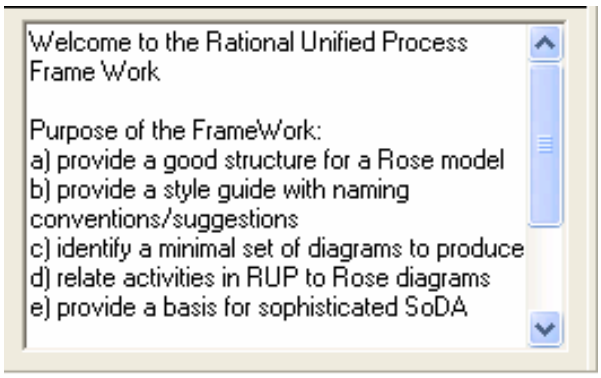



Gambar 2.23 Jendela Diagram

- Jendela Dokumentasi

  *Jendela dokumentasi* digunakan untuk membuat dokumentasi elemen-elemen model. Kita dapat membuat, melihat atau memodifikasi dokumentasi dalam jendela ini atau dalam jendela dokumentasi yang terdapat dalam spesifikasi. Jika jendela dokumentasi tidak muncul, pilih documentation dari menu view. Jika terdapat tanda ceklis pada documentation dan jendela ini belum muncul,

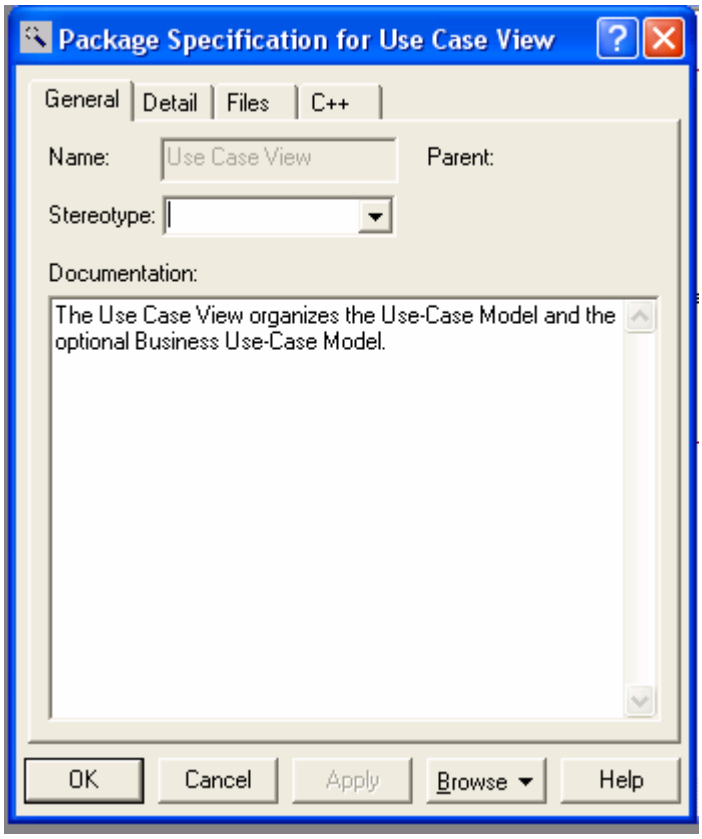
gerakkan kursor ke bagian bawah browser. Saat pointer berubah menjadi kursor pembagi (biasanya tanda panah pada kedua ujungnya) tarik keatas untuk memunculkan jendela dokumentasi. Isi dokumentasi pada jendela ini adalah dokumentasi dari elemen yang kita sorot pada jendela diagram.




Gambar 2.24 Jendela Dokumentasi

- Spesification

*Spesification* adalah kotak dialog yang digunakan untuk membuat atau mengubah properties elemen model.




Gambar 2.25 Spesification

### 2.7.3 View dalam Rational Rose

#### 2.7.3.1 Use Case View

*Use case view* membantu kita untuk memahami dan menggunakan sistem yang kita modelkan. View ini melihat pada bagaimana actor[6] dan use case[7] berinteraksi.

Terdapat beberapa diagram yang digunakan dalam use case view, yaitu :

- Use Case Diagram.
- Sequence Diagram.
- Collaboration Diagram.
- Activity Diagram.

#### 2.7.3.2 Logical View

*Logical view*, mengarah pada persyaratan *(requirements)* fungsional sistem. View ini melihat pada kelas-kelas dan hubungan antar kelas-kelas tersebut. Diagram dalam view ini adalah :

- Class Diagram.
- Sequence Diagram.
- Collaboration Diagram.
- Statechart Diagram.

[6] Actor menggambarkan pengguna (user) sistem. Actor membantu membatasi sistem dan memberi gambaran yang jelas mengenai apa yang harus dilakukan oleh sistem. Penting untuk dicatat bahwa actor berinteraksi dengan use-case, tetapi tidak mengendalikan use-case.

[7] Sebuah use-case dapat digambarkan sebagai suatu cara tertentu untuk menggunakan sistem dari sudut pandang satu pengguna (an actor)

#### 2.7.3.3 Component View

Mengarah pada pengaturan software. View ini mengandung informasi mengenai komponen-komponen software, komponen tereksekusi (executable) dan library untuk sistem yang kita modelkan. Hanya ada satu diagram dalam view ini, yaitu component diagram.

#### 2.7.3.4 Deployment View

Memperlihatkan pemetaan setiap proses kedalam hardware. View ini paling bermanfaat ketika kita membuat model suatu sistem yang diterapkan dalam lingkungan arsitektur yang terdistribusi dimana kita menerapkan aplikasi dan server pada lokasi yang berbeda. Hanya ada satu diagram dalam view ini, yaitu deployment diagram.

### 2.7.4 Diagram dalam Rational Rose

#### 2.7.4.1 Use Case Diagram

Menjelaskan manfaat sistem jika dilihat menurut pandangan orang yang berada diluar sistem (actor). Diagram ini menunjukkan fungsionalitas suatu sistem atau kelas dan bagaimana sistem berinteraksi dengan dunia luar. Use Case diagram dapat digunakan selama *proses analisis untuk menangkap requirements sistem* dan untuk memahami bagaimana sistem seharusnya bekerja.

### 2.7.4.2 Class Diagram

Memperlihatkan hubungan antar kelas dan penjelasan detil tiap-tiap kelas didalam model desain (dalam logical view) dari suatu sistem. Selama proses analisis, class diagram memperlihatkan aturan-aturan dan tanggung jawab entitas yang menentukan perilaku sistem. Selama tahap desain, class diagram berperan dalam menangkap struktur dari semua kelas yang membentuk arsitektur sistem yang dibuat. Merupakan fondasi untuk component diagram dan deployment diagram.

### 2.7.4.3 Statechart Diagram

Memperlihatkan urutan keadaan sesaat yang dilalui sebuah obyek, kejadian yang menyebabkan sebuah transisi dari satu state atau aktivitas kepada yang lainnya, dan aksi yang menyebabkan perubahan satu state atau aktivitas

### 2.7.4.4 Activity Diagram

Memodelkan alur kerja (workflow) sebuah proses bisnis dan urutan aktivitas dalam suatu proses. Diagram ini sangat mirip dengan sebuah flowchart karena kita dapat memodelkan sebuah alur kerja dari satu aktivitas ke aktivitass lainnya atau dari satu aktivitas ke keadaan sesaat (state). Juga sangat berguna ketika ingin menggambarkan perilaku paralel atau menjelaskan bagaimana perilaku dalam berbagai use case berinteraksi.

### 2.7.4.5 Sequence Diagram

Menjelaskan interaksi obyek yang disusun dalam suatu urutan tertentu. Sequence diagram memperlihatkan tahap demi tahap apa yang seharusnya terjadi untuk menghasilkan sesuatu didalam use case.

### 2.7.4.6 Collaboration Diagram

Melihat pada interaksi dan hubungan terstruktur antar obyek. Tipe diagram ini menekankan pada hubungan *(relationship)* antar obyek, sedangkan sequence diagram menekankan pada urutan kejadian. Collaboration diagram digunakan sebagai alat untuk menggambarkan interaksi yang mengungkapkan keputusan mengenai perilaku sistem.

### 2.7.4.7 Component Diagram

Menggambarkan alokasi semua kelas dan obyek kedalam komponen-komponen dalam desain fisik sistem software. Diagram ini mmperlihatkan pengaturan dan kebergantungan antara komponene-komponen software seperti source code, binary code dan komponen tereksekusi.

### 2.7.4.8 Deployment Diagram

Diagram ini memperlihatkan pemetaan software kepada hardware.Diagram ini mengambarkan detail bagaimana komponen di-*deploy* dalam infrastruktur sistem, dimana komponen akan terletak, bagaimana kemampuan jaringan pada

lokasi tersebut dan hal lain yang bersifat fisik (Sri Dharwiyanti dan Romi satria Wahono, 2003)

### 2.7.5 Use-Case Modelling

*Use-case Modelling* adalah teknik paling sederhana dan paling efektif untuk memodelkan kebutuhan sistem berdasarkan pandangan user (Bennet, Simon, Steve Mc Robb dan Ray Farmer, 2000). Use-case modelling digunakan untuk bagaimana sistem atau kerja nyata dari suatu sistem atau bagaimana user ingin sistem itu bekerja. Use-case pada dasarnya adalah langkah awal dari analisis berdasarkan obyek dengan UML.

Use-case model terdiri dari actor dan use-case. Actor merepresentasikan user dan sistem lain yang berinteraksi dengan sistem. Use-case model sesungguhnya merepresentasikan tipe dari user, bukan suatu hal dari user. Use-case merepresentasikan karakteristik sistem, skenario dari tujuan sistem kedalam reaksi untuk menggerakkan actor.

Use-case adalah skenario untuk memahami kebutuhan user. Use-case model dapat menjadi kerangka dalam proyek pengembangan, perencanaan dan dokumentasi dari kebutuhan sistem. Use-case adalah interaksi antara user dan sistem, menggambarkan tujuan dari sistem dan tanggapan sistem untuk user. Use-case model mencoba untuk mensistematiskan identifikasi dari kegunaan sistem dan tanggapan dari sistem. Use-case model juga dapat mencakup kelas-kelas dan hubungan yang dimiliki oleh sub-sistem dari sistem.

Setiap use-case atau skenario merepresentasikan apa yang user ingin lakukan. Setiap use-case harus mempunyai nama dan deskripsi teks pendek, hanya sedikit paragraph.

### 2.7.5.1 Identifikasi Actor

Mengidentifikasikan actor adalah sama pentingnya dengan mengidentifikasikan kelas, struktur, hubungan, atribut dan karakteristik. Ketika menentukan actor, sangat penting untuk berfikir mengenai aturan rata-rata dari orang atau jenis pekerjaan. User mungkin memainkan lebih dari satu aturan. Actor harus merepresentasikan user tunggal.

Harus mengidentifikasi actor dan mengerti bagaimana mereka akan berguna dan berinteraksi dengan sistem. Kandidat untuk actor dapat ditemukan dengan menjawab pertanyaan-pertanyaan berikut :

- *Siapa yang menggunakan sistem ?*

  Kelompok mana yang membutuhkan pertolongan dari sistem untuk melakukan suatu pekerjaan.

- *Kelompok pengguna mana yang membutuhkan penampilan fungsi sistem ?*

  Fungsi ini dapat merupakan fungsi utama dan fungsi sekunder, seperti administrasi.

- *Apa masalah dari penyelesaian aplikasi (untuk siapa) ?*

- Dan terakhir, *bagaimana user menggunakan sistem (use-case) ?*

  Apa yang mereka lakukan dengan sistem ?

### 2.7.5.2 Menemukan Use-Case

Menurut Bennet, Simon, Steve Mc Robb dan Ray Farmer (2000), langkah-langkah untuk menemukan use-case adalah sebagai berikut:

- Untuk setiap actor, temukan tugas dan fungsi actor harus dapat melakukan apa yang sistem butuhkan dari actor untuk melakukannya. Use-case harus merepresentasikan kegiatan dari kejadian untuk menghasilkan tujuan.
- Beri nama untuk setiap use-case.
- Deskripsikan masing-masing use-case.

Berapa banyak use-case yang dibutuhkan ? Untuk sepuluh orang per tahun proyek, actor mungkin mendapatkan dua puluh use-case (tidak terhitung hubungan uses dan extend). Dalam penelitian lain, mungkin mencapai seratus use-case untuk sebuah proyek. Tidak ada formula tertentu, hanya dibutuhkan kedinamisan dan kerja yang menemukan kenyamanan.

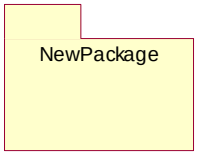
### 2.7.5.3 Penamaan Use-Case

Penamaan use-case harus menguntungkan deskripsi global dari fungsi use-case. Nama harus mendeskripsikan apa yang terjadi ketika bagian dari kinerja use-case bekerja. Penamaan dalam kata kerja atau kata benda, harus dilakukan dengan hati-hati karena deskripsi dari use-case harus merupakan gambaran dan sifatnya tetap.

Use-case adalah alat utama dalam menggambarkan kebutuhan. Menggambarkan use-case adalah salah satu langkah pertama yang penting untuk

melakukan sesuai dengan kebutuhan. Setiap use-case adalah kebutuhan yang berpotensi. Setiap use-case atau skenario merepresentasikan apa yang user ingin lakukan.

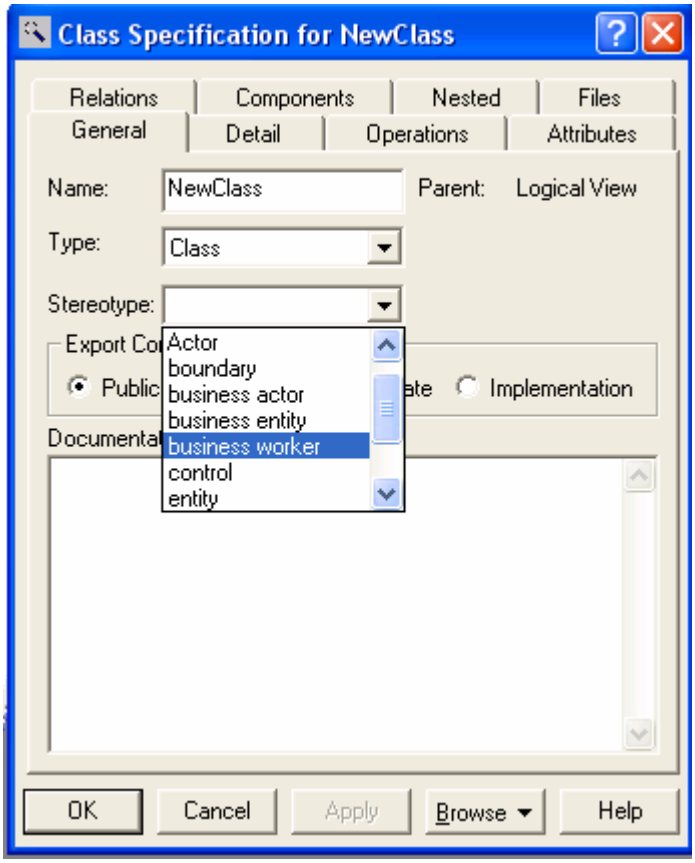
### 2.7.6 Package




Gambar 2.26 Notasi Package (Paket)

*Paket* adalah mekanisme pengelompokkan yang dgunakan untuk menandakan pengelompokkan elemen-elemen model. Sebuah paket dapat mengandung beberapa paket lain didalamnya. Paket digunakan untuk memudahkan mengorganisasikan elemen-elemen model.

### 2.7.7 Stereotype




Gambar 2.27 Ruang Stereotype dalam Jendela Spesification

Sebuah stereotype menerangkan subklasifikasi dari sebuah elemen model.

### 2.7.8 Relationship

Adalah koneksi antar model elemen.


Gambar 2.28 Association Relationship

*Association Relationship*, memodelkan koneksi antar obyek dari kelas yang berbeda.

Interaksi antara actor dan use-case dalam use-case model biasanya digunakan association relationship yaitu :

- *<<uses>>*

  Hubungan uses menunjukkan bahwa prosedur dari use-case merupakan bagian dari prosedur yang menggunakan use-case. Tanda panah menunjukkan keadaan tidak mengakibatkan pemanggilan prosedur dalam menggunakan use-case. Relasi uses antara use-case ditunjukkan dengan panah generalisasi dari use-case. Use-case yang dilakukan secara berulang, digunakan untuk meminimalkan pekerjaan.

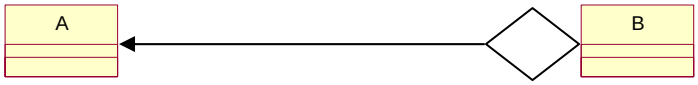
- *<<extend>>*

  Hubungan extend antar use-case berarti bahwa suatu use-case merupakan tambahan kegunaan dari use-case yang lain jika kondisi atau syarat tertentu dipenuhi. Jika prosedur dari use-case merupakan alternatif untuk menjelaskan use-case lain. Use-case akan dikerjakan apabila salah satu syarat terpenuhi.

Hubungan generalisasi antar use-case menunjukkan bahwa use-case yang satu merupakan spesialisai dari yang lain.

- *<<include>>*

  Hubungan include menggambarkan suatu use-case seluruhnya meliputi kegunaan dari use-case lainnya. 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 dieksekusi secara normal. Sebuah use-case dapat di-include oleh lebih dari use-case lain, sehingga duplikasi fungsional dapat dihindari.




Gambar 2.29 Aggregation Relationship

*Aggregation Relationship*, adalah bentuk khusus assosiasi yang memodelkan hubungan keanggotaan antara 2 kelas, yakni satu kelas disusun oleh kelas lainnya. Gambar diatas memperlihatkan bahwa B (kelas aggregate) yang secara fisik dibentuk oleh A atau secara logis B mengandung A.

## 2.8 Model Analisis

*Model analisis* menggambarkan realisasi dari use case - use case dalam use case model, dan bertindak sebagai abstraksi dari model desain. Tujuan akhir dari sebuah model analisis adalah untuk membuat pemetaan awal mengenai perilaku yang diisyaratkan dalam sistem aplikasi kedalam elemen-elemen pemodelan.

Model analisis merupakan transisi kedalam model desain, dan kelas-kelas analisis secara langsung berkembang menjadi elemen-elemen dalam model desain.

### 2.8.1 Kelas dalam Model Analisis

Elemen model yang terdapat dalam model analisis disebut kelas analisis. Kelas analisis adalah kelas berstereotype *boundary, control atau entity* yang menggambarkan sebuah konsep awal mengenai benda dalam sistem aplikasi yang memiliki tanggung jawab dan perilaku. Kelas analisis akhirnya berkembang menjadi kelas didalam model desain.

#### 2.8.1.1 Boundary Class


Gambar 2.30 Boundary Class

*Kelas boundary* adalah kelas yang memodelkan interaksi antara satu atau lebih actor dengan sistem. Kelas boundary memodelkan bagian dari sistem yang bergantung pihak lain disekitarnya dan merupakan pembatas sistem dengan dunia luar. Lebih lanjut lagi user interface class sering disamakan dengan form yang digunakan sebagai interface antara sistem dengan user.

Kelas boundary dapat berupa :

- *User interface,* yang merupakan sarana komunikasi antara sistem dengan user, misalnya jendela (window) dalam GUI.
- *Sistem interface,* yang merupakan sarana komunikasi antara sistem dengan sistem informasi lainnya misalnya communication protocol.
- *Device interface,* yang merupakan sarana komunikasi antara sistem dengan device *(alat),* seperti printer, sensor dan sebagainya.

### 2.8.1.2 Control Class


Gambar 2.31 Control Class

*Kelas control* adalah kelas yang mengkoordinasikan aktivitas dalam sistem. Kelas ini menghubungkan kelas boundary dengan kelas entity. Kelas *control* digunakan untuk memodelkan *"perilaku mengatur"*, khusus untuk satu atau beberapa *use-case* saja. Kelas *control* tidak dipengaruhi perubahan disekelilingnya. Kelas ini menggunakan atau membuat isi dari kelas *entity* dan biasanya memasangkan kelas *boundary* dengan kelas *entity*.

### 2.8.1.3 Entity Class


Gambar 2.32 Entity Class

*Kelas entity* adalah kelas yang berhubungan data dan informasi yang digunakan oleh sistem. Kelas entity ini adalah kelas yang menyimpan dan mengolah data. Kelas entity memodelkan informasi yang harus disimpan oleh sistem. Kelas *entity* memperlihatkan data dari sebuah sistem. Oleh karena itu, kelas *entity* membantu untuk memahami apa yang kira-kira ditawarkan oleh sistem kepada user.

*Entity object* (instance dari kelas entity) biasanya bersifat pasif dan tetap (tidak berubah-ubah). Tanggung jawab utama objek ini adalah untuk menyimpan dan mengatur informasi dalam sistem.

### 2.8.1.4 Aturan yang Well-formed dalam Model Analisis

Berikut ini tabel mengenai *association relationship* beserta *stereotype*-nya yang diperbolehkan dalam model analisis.[8]

| TO<br>FROM | ***ACTOR*** | ***BOUNDARY*** | ***ENTITY*** | ***CONTROL*** |
|---|---|---|---|---|
| ***Actor*** | | communicate | | |
| ***Boundary*** | communicate | communicate | communicate<br>subscribe | communicate |
| ***Entity*** | | | communicate<br>subscribe | |
| ***Control*** | | Communicate | communicate<br>subscribe | communicate |

Tabel 2.2 Aturan dalam Model Analisis

[8] (UML Standard Profiles, hal. 8)