

Modul Rekayasa Perangkat Lunak STT-Garut 2016
TATA TERTIB & PETUNJUK PELAKSANAAN PRAKTIKUM

REKAYA PERANGKAT LUNAK
JURUSAN TEKNIK INFORMATIKA
SEKOLAH TINGGI TEKNOLOGI GARUT

A. TATA TERTIB

1. Praktikan diharapkan hadir tepat pada waktunya. Keterlambatan lebih dari 30 menit, praktikan tidak boleh mengikuti praktikum dan dianggap tidak hadir (alpa)
2. Praktikan wajib duduk dan menggunakan computer pada tempat /computer yang telah ditentukan (tidak boleh pindah dari satu computer ke computer lain selama periode pelaksanaan praktikum.
3. Mengisi daftar hadir sebagai bukti mengikuti kegiatan praktikum.
4. Praktikan wajib memelihara dan menjaga peralatan yang ada di ruang
5. komputer, Kerusakan/kehilangan yang disebabkan oleh praktikan merupakan tanggung jawab praktikan.
6. Praktikan harus membawa media penyimpana (CD/DVD/ media penyimpanan lainnya yang bebas dari virus)
7. Praktikan tidak diperkenankan memakai T-SHIRT dan sandal dalam jenis apapun selama praktikum berlangsung.
8. Selama praktikum berlangsung, praktikan tidak diperkenankan keluar dari ruang komputer, kecuali bila ada keperluan yang sangat penting dan tidak dapat ditunda.
9. Apabila Praktikan akan meninggalkan ruang praktikum maka praktikan dapat meminta izin terlebih dahulu kepada koordinator praktikum.
10. Selama Praktikum berlangsung tidak diperkenankan merokok, makan dan minum.
11. Praktikan harus memberikan laporan praktikum paling lambat 1 minggu setelah setelah praktikum modul terakhir selesai

12. Ketentuan lain yang belum tercantum, tetapi kemudian di pandang perlu akan ditetapkan kemudian.

B. PETUNJUK PELAKSANAAN PRAKTIKUM

1. Setiap praktikan harus membawa segala sesuatu keperluan praktikum, seperti Modul Praktikum, alat tulis, dan media penyimpanan.
2. Praktikan harus mengikuti keseluruhan praktikum, jika tidak hadir tanpa alasan yang jelas satu kali atau lebih dianggap mengundurkan diri.
3. Bagi praktikan yang tidak dapat mengikuti jadwal praktikum. yang telah ditentukan dengan alasan sakit (Melalui Surat Keterangan Dokter), harus memberitahukan kepada asisten/koordinator praktikum pada saat jadwal ,praktikumnya dilaksanakan.
4. Praktikan yang karena sesuatu hal tidak dapat mengikuti praktikum maka yang bersangkutan tetap diwajibkan untuk menyelesaikan tugas latihan dan modul yang telah ditetapkan secara mandiri
5. Sebelum praktikum dimulai, praktikan harus memahami teori dan petunjuk pelaksanaan praktikum.
6. Bila Praktikan mengalami kesulitan pada saat pelaksanaan praktikum, praktikan dapat bertanya kepada asisten/dosen pembimbing praktikum.
7. Setiap praktikan wajib membuat laporan praktikum dengan ketentuan
8. sebagai berikut:
 - a. Laporan dibuat dalam bentuk Hard copy dan Soft Copy dengan media penyimpanan pada CD/DVD yang bebas virus, diberi label Nama, NPM Keterlambatan penyerahan laporan praktikum akan mendapatkan pengurangan nilai.

PENDAHULUAN

A. TUJUAN

Praktikum Rekayasa Perangkat Lunak bertujuan untuk :

- Memberikan pengetahuan kepada praktikan tahapan-tahapan dalam pembuatan Rekayasa Perangkat Lunak.
- Praktikan bisa menyelesaikan proyek Perangkat Lunak.
- Praktikan bisa membiasakan diri untuk menyelesaikan Proyek Perangkat Lunak secara terstruktur baik dalam satu tim maupun individu.
- Praktikan bisa menerapkan metodologi Rekayasa pembuatan Perangkat Lunak pada suatu kasus tertentu yang diberikan.
- Praktikan dapat membuat dokumen-dokumen yang diperlukan dalam rekayasa perangkat lunak
- Menunjang mata kuliah Rekayasa Perangkat Lunak
- Memberikan wawasan kepada praktikan untuk menghadapi mata kuliah Kerja Praktek dan Tugas Akhir.
- Membiasakan praktikan untuk menyelesaikan tugas/pekerjaan tepat waktu sesuai yang telah dijadwalkan.

B. TEORI

1. PERANGKAT LUNAK

Perangkat Lunak adalah *“(1)perintah (program komputer) yang bila dieksekusi memberikan fungsi dan unjuk kerja seperti yang diinginkan. (2) struktur data yang memungkinkan program memanipulasi secara proporsional, dan (3) dokumen yang menggambarkan operasi dan kegunaan program”*.
(Pressman, Roger S, 2002)

1.1. PENGERTIAN REKAYASA PERANGKAT LUNAK

- **Pressman, Roger S, 2002**

- 1) Pembentukan dan penggunaan prinsip rekayasa (*engineering*) untuk mendapatkan perangkat lunak secara ekonomis namun andal dan dapat bekerja secara efisien pada komputer (dikutip dari Fritz Bauer, 1968).
- 2) Suatu disiplin, kaidah yang mengintegrasikan proses, metode, dan alat bantu (*tools*) untuk pembangunan perangkat lunak komputer.

- **IEEE Computer Society:**

- 1) *The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.*
- 2) *The study of approaches as in (1).*

1.2. MENGAPA RPL ?

- 1) Perangkat lunak sudah diselesaikan dan diserahkan (*delivered*) tetapi tidak pernah digunakan (47%).
- 2) Pemakai (*user*) sudah membayar untuk perangkat lunak tetapi tidak pernah jadi dan diserahkan (29,7%).
- 3) Perangkat lunak sudah digunakan tetapi kritis atau ditinggalkan (19%).
- 4) Perangkat lunak digunakan setelah dilakukan modifikasi (3%).
- 5) Hanya sebagian kecil perangkat lunak yang dapat digunakan sebagaimana mestinya (2%).

1.3. PERANGKAT LUNAK BERKUALITAS

- 1) Perangkat lunak yang dihasilkan sesuai dengan kebutuhan yang diinginkan.
- 2) Perangkat lunak dapat digunakan dan beroperasi dengan benar di lingkungan sebenarnya.

- 3) Perangkat lunak memberikan manfaat bagi pemakai yang menggunakannya.
- 4) Biaya yang dikeluarkan untuk membuatnya rendah (efisien), efektif dan sesuai dengan anggaran yang telah ditetapkan.
- 5) Tepat waktu, baik saat pembuatan, penyerahan ke pemakai, maupun instalasinya.
- 6) Setiap tahap pekerjaan terjamin kualitasnya, terdokumentasi, dan dapat dipertanggungjawabkan kebenarannya (ada proses verifikasi dan validasi).

1.4. KARAKTERISTIK PERANGKAT LUNAK

Perangkat Lunak lebih merupakan elemen logika dan bukan merupakan elemen sistem fisik. Dengan demikian perangkat lunak memiliki karakteristik diantaranya;

- a. Perangkat lunak dibangun dan dikembangkan, tidak dibuat dalam bentuk yang klasik.
- b. Perangkat lunak tidak pernah usang
- c. Sebagian besar perangkat lunak dibuat secara *custom-built*, serta tidak dapat dirakit dari komponen yang sudah ada, (Pressman, Roger S, 2002)

1.5. APLIKASI PERANGKAT LUNAK

Kategori umum untuk aplikasi perangkat lunak maka dapat digolongkan menjadi jenis perangkat lunak diantaranya;

1. Perangkat Lunak Sistem .Perangkat lunak sistem merupakan sekumpulan program yang ditulis untuk melayani program –program yang lain.
2. Perangkat Lunak *Real-Time*. Program-program yang memonitor, menganalisis atau mengontrol kejadian dunia nyata pada saat terjadi.
3. Perangkat Lunak Bisnis. Pemrosesan informasi bisnis merupakan area aplikasi perangkat lunak yang paling luas.

4. Perangkat Lunak Teknik dan Ilmu Pengetahuan

Perangkat lunak yang memiliki jangkauan aplikasi mulai dari astronomi sampai vulkanologi, dari analisis otomotif n sampai dinamika orbit pesawat ruang angkasa, dan dari biologi molekuler sampai pabrik yang sudah diotomatiskan.

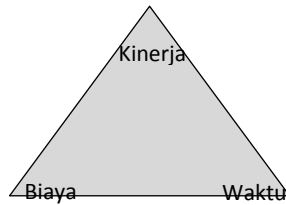
- a. Perangkat Lunak Tertanam (*Embedded Software*) .*Embedded Software* ada dalam *Read Only memory* dan dipakai untuk mengontrol hasil serta sistem untuk keperluan konsumen dan pasar industri.
- b. Perangkat Lunak Komputer Personal. Perangkat lunak yang di khususkan untuk kepentingan personal
- c. Perangkat Lunak Kecerdasan Buatan. Perangkat lunak yang menggunakan algoritma *non-numeris* untuk memecahkan masalah kompleks yang tidak sesuai untuk perhitungan atau analisis secara langsung. (Pressman, Roger S, 2002)

1.6. PERANAN PERANGKAT LUNAK

- 1) Menggantikan peran manusia, dengan otomasi terhadap suatu tugas atau proses.
- 2) Memperkuat peran manusia, dengan menyajikan informasi yang diperlukan manusia untuk menyelesaikan tugas atau proses.
- 3) Restrukturisasi peran manusia, dengan melakukan perubahan-perubahan terhadap sekumpulan tugas atau proses.
- 4) Hiburan dan Permainan, dengan menyajikan aplikasi interaktif yang semakin dekat dengan kenyataan, (Wahono, Romi Satria, 2007)

1.7. TUJUAN REKAYASA PERANGKAT LUNAK

Secara umum tujuan RPL tidak berbeda dengan bidang rekayasa yang lain. Hal ini dapat kita lihat pada Gambar di bawah ini.



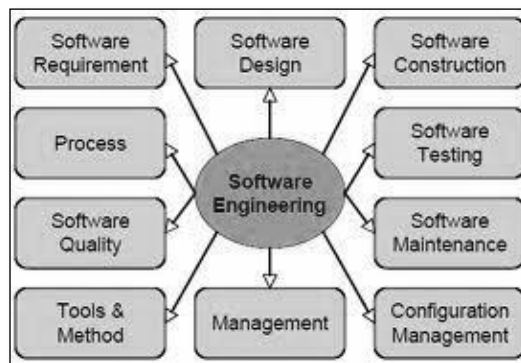
Gambar Tujuan RPL

Dari Gambar di atas dapat diartikan bahwa bidang rekayasa akan selalu berusaha menghasilkan output yang kinerjanya tinggi, biaya rendah dan waktu penyelesaian yang tepat. Secara lebih khusus kita dapat menyatakan tujuan RPL adalah:

- Memperoleh biaya produksi perangkat lunak yang rendah
- Menghasilkan perangkat lunak yang kinerjanya tinggi, andal dan tepat waktu
- Menghasilkan perangkat lunak yang dapat bekerja pada berbagai jenis platform
- Menghasilkan perangkat lunak yang biaya perawatannya rendah

1.8. RUANG LINGKUP REKAYASA PERANGKAT LUNAK

lingkup RPL dapat digambarkan sebagai berikut :



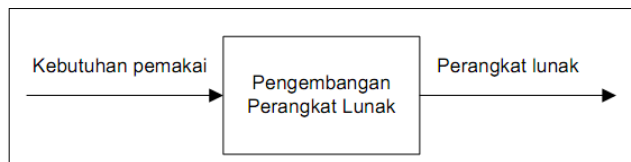
Gambar Ruang Lingkup RPL

- *Software requirements* berhubungan dengan spesifikasi kebutuhan dan persyaratan perangkat lunak.
- *Software design* mencakup proses penentuan arsitektur, komponen, antarmuka, dan karakteristik lain dari perangkat lunak.

- *Software construction* berhubungan dengan detail pengembangan perangkat lunak, termasuk algoritma, pengkodean, pengujian, dan pencarian kesalahan.
- *Software testing* meliputi pengujian pada keseluruhan perilaku perangkat lunak.
- *Software maintenance* mencakup upaya-upaya perawatan ketika perangkat lunak telah dioperasikan.
- *Software configuration management* berhubungan dengan usaha perubahan konfigurasi perangkat lunak untuk memenuhi kebutuhan tertentu.
- *Software engineering management* berkaitan dengan pengelolaan dan pengukuran RPL, termasuk perencanaan proyek perangkat lunak.
- *Software engineering tools and methods* mencakup kajian teoritis tentang alat bantu dan metode RPL.
- *Software engineering process* berhubungan dengan definisi, implementasi, pengukuran, pengelolaan, perubahan dan perbaikan proses RPL.
- *Software quality* menitikberatkan pada kualitas dan daur hidup perangkat lunak.

2. PENGEMBANGAN PERANGKAT LUNAK

Pengembangan perangkat lunak didefinisikan sebagai suatu proses dimana kebutuhan pemakai diterjemahkan menjadi produk perangkat lunak melalui suatu rangkaian aktivitas tertentu sesuai model proses yang digunakan (lihat Gambar).

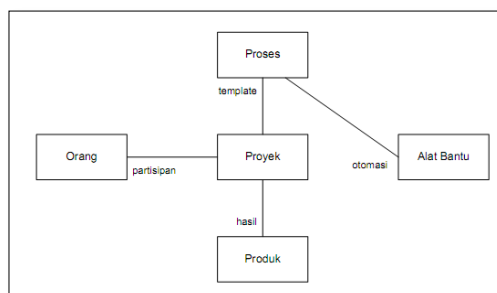


Gambar Proses Pengembangan Perangkat Lunak

Proses pengembangan perangkat lunak dilaksanakan setelah proses akuisisi dan pasokan. Dalam pelaksanaannya, proses pengembangan ini akan melibatkan:

- Orang (*people*) Individu, kelompok, atau bagian organisasi yang menjadi pelaksana pekerjaan, seperti analis sistem, pemrogram, penguji perangkat lunak, dan pihak-pihak lainnya termasuk pemakai dan pelanggan.
- Proyek (*project*) Pekerjaan pengembangan perangkat lunaknya sendiri, yang dikelola sesuai prinsip-prinsip manajemen proyek.
- Produk (*product*) Kode sumber (*source code*), *executable programs*, model-model, dan dokumen-dokumen yang dihasilkan sebagai produk selama pelaksanaan pengembangan.
- Proses (*process*) Kumpulan aktivitas yang digunakan untuk menghasilkan perangkat lunak. Setiap aktivitas dilaksanakan dengan menggunakan pendekatan atau metode teknis tertentu.
- Alat bantu Kumpulan perangkat bantu atau kakas otomatis dan semi-otomatis yang akan digunakan untuk mendukung aktivitas-aktivitas dari proses.

Hubungan kelima elemen yang terkait dengan proses pengembangan di atas ditunjukkan oleh Gambar berikut.



Gambar Hubungan Antar Elemen Proses Pengembangan

2.1. FASE PENGEMBANGAN PERANGKAT LUNAK

Kegiatan yang berhubungan dengan fase pengembangan perangkat lunak dapat dikategorikan ke dalam tiga fase umum, yaitu :

1) Fase Definisi (*Definiton phase*)

Pada fase ini pengembang perangkat lunak harus mengidentifikasi informasi apa yang akan diproses, fungsi dan unjuk kerja apa yang dibutuhkan, tingkah laku sistem seperti apa yang diharapkan, *interface* apa yang akan dibangun, batasan desain apa yang ada, dan kriteria validasi apa yang dibutuhkan untuk mendefinisikan sistem yang sukses.

2) Fase Pengembangan (*Development Phase*)

Pada fase ini pengembang mendefinisikan bagaimana data dikonstruksikan, bagaimana fungsi-fungsi diimplementasikan sebagai sebuah arsitektur perangkat lunak, bagaimana detail prosedur akan diimplementasikan, bagaimana *interface* ditandai (dikarakterisasi), bagaimana rancangan akan diterjemahkan ke dalam bahasa pemrograman, serta bagaimana pengujian akan dilakukan.

3) Fase Pemeliharaan (*Maintenance Phase*)

Fase ini berfokus pada perubahan (*change*) yang dihubungkan dengan koreksi kesalahan, penyesuaian yang dibutuhkan ketika lingkungan perangkat lunak berkembang, serta perubahan sehubungan dengan perkembangan yang disebabkan oleh perubahan kebutuhan pelanggan, (Pressman, Roger S., , 2002).

2.2. SOFTWARE ENGINEERING PROCESS

- Seri aktivitas yang harus dilaksanakan selama siklus hidup perangkat lunak.
- IEEE/EIA 12207 Standard for Information Technology:
 1. Proses utama (*primary processes*)
 2. Proses pendukung (*supporting processes*)
 3. Proses organisasi (*organizational processes*)

PROSES UTAMA

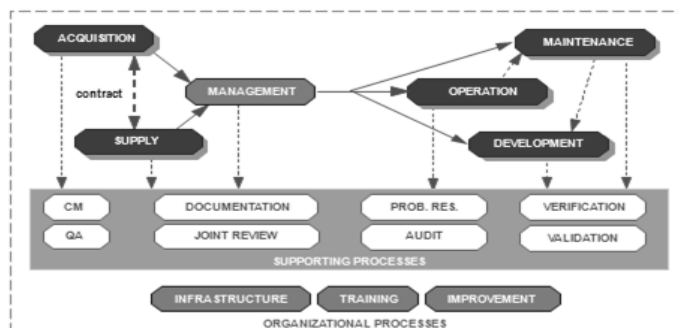
1. Akuisisi (*Acquisition*)
2. Pasokan (*Supply*)
3. Pengembangan (*Development*)
4. Pengoperasian (*Operation*)
5. Pemeliharaan (*Maintenance*)

PROSES PENDUKUNG

1. Documentation
2. Configuration Management
3. Quality Assurance
4. Verification
5. Validation
6. Joint Review
7. Audit
8. Problem Resolution

PROSES ORGANISASI

1. Management
2. Infrastructure
3. Improvement
4. Training

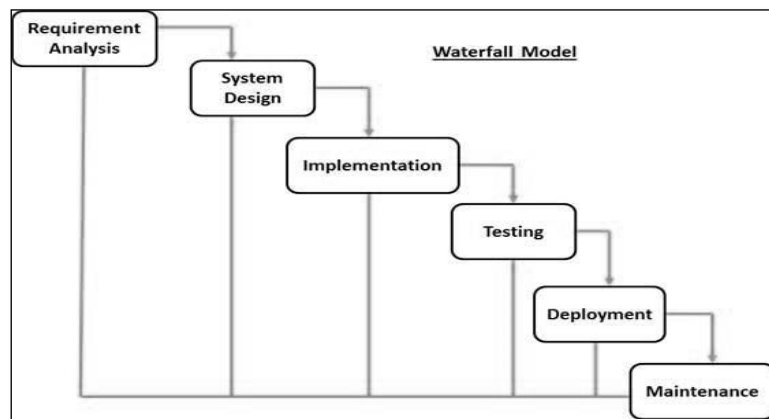


Gambar Software Engineering process IEEE/EIA 12207

2.3. Model Proses Pengembangan Perangkat Lunak

A. Linear Sequential Model

Linear sequential model (atau disebut juga “*classic life cycle*” atau “*waterfall model*”) adalah metode pengembangan perangkat lunak dengan pendekatan sekuensial dengan cakupan aktivitas :



Gambar *Waterfall Model*

1) Rekayasa sistem dan Analisis (*Sistem Engineering and Analysis*)

Karena perangkat lunak adalah bagian dari sistem yang lebih besar, pekerjaan dimulai dari pembentukan kebutuhan-kebutuhan untuk seluruh elemen sistem dan kemudian memilah mana yang untuk pengembangan perangkat lunak. Hal ini penting, ketika perangkat lunak harus berkomunikasi dengan hardware, orang dan basis data

2) Analisis kebutuhan perangkat lunak (*Software Requirements Analysis*)

Pengumpulan kebutuhan dengan fokus pada perangkat lunak, yang meliputi :

Domain informasi, fungsi yang dibutuhkan, unjuk kerja/performansi dan antarmuka. Hasilnya harus didokumentasi dan direview ke pelanggan

3) Perancangan (*Design*)

Ada 4 atribut untuk program yaitu : Struktur Data, Arsitektur perangkat lunak, Prosedur detil dan Karakteristik Antarmuka. Proses desain mengubah kebutuhan-kebutuhan menjadi bentuk karakteristik yang dimengerti perangkat lunak sebelum dimulai penulisan program. Desain ini harus terdokumentasi dengan baik dan menjadi bagian konfigurasi perangkat lunak.

4) Pembuatan kode (*Coding*)

Penterjemahan perancangan ke bentuk yang dapat dimengerti oleh mesin, dengan menggunakan bahasa pemrograman

5) Pengujian (*Testing*)

Setelah kode program selesai *testing* dapat dilakukan. *Testing* memfokuskan pada logika internal dari perangkat lunak, fungsi eksternal dan mencari segala kemungkinan kesalahan dan memeriksa apakah sesuai dengan hasil yang diinginkan.

6) Pemeliharaan (*Maintenance*)

Merupakan bagian paling akhir dari siklus pengembangan dan dilakukan setelah perangkat lunak dipergunakan. Kegiatan :

- *Corrective Maintenance* : Mengoreksi kesalahan pada perangkat lunak, yang baru terdeteksi pada saat perangkat lunak dipergunakan
- *Adaptive Maintenance* : Penyesuaian dengan lingkungan baru, misalnya sistem operasi atau sebagai tuntutan atas perkembangan sistem komputer, misalnya penambahan printer driver
- *Perfektive Maintenance* : Bila perangkat lunak sukses dipergunakan oleh pemakai. Pemeliharaan ditujukan untuk menambah kemampuannya seperti memberikan fungsi-fungsi tambahan, peningkatan kinerja dan sebagainya.

Kelemahan model *linear sequential*:

- 1) Proyek yang sebenarnya jarang mengikuti alur sekuensial seperti diusulkan, sehingga perubahan yang terjadi dapat menyebabkan hasil yang sudah didapat tim harus diubah kembali/iterasi sering menyebabkan masalah baru.
- 2) Linear sequential model mengharuskan semua kebutuhan pemakai sudah dinyatakan secara eksplisit di awal proses, tetapi kadang-kadang ini tidak dapat terlaksana karena kesulitan yang dialami pemakai saat akan mengungkapkan semua kebutuhannya tersebut.
- 3) Pemakai harus bersabar karena versi dari program tidak akan didapat sampai akhir rentang waktu proyek.
- 4) Adanya waktu menganggur bagi pengembang, karena harus menunggu anggota tim proyek lainnya menuntaskan pekerjaannya.

B. *Prototyping Model*

Pendekatan model *Prototyping* dipilih jika kebutuhan belum dapat didefinisikan secara jelas dan tegas. Pemakai hanya memberikan gambaran umum dari spesifikasi dan kegunaan perangkat lunak tanpa merinci seperti apa masukan, poses, dan keluarannya. Model proses *Prototyping* dilaksanakan secara berulang dengan diawali oleh aktivitas pengumpulan kebutuhan sistem dan berakhir jika produk perangkat lunak yang dihasilkan sudah sesuai dengan yang diharapkan oleh pemakai seperti ditunjukkan oleh Gambar dibawah. Cakupan aktivitas model *Prototyping* secara lengkap meliputi aktivitas:

- 1) Pengumpulan kebutuhan dan perbaikan

Pengembang bertemu dengan pemakai (pelanggan) untuk menentukan objektif perangkat lunak secara keseluruhan dan mengidentifikasi kebutuhan-kebutuhan yang sudah diketahui.

2) Perancangan cepat

Melakukan perancangan secara cepat dengan fokus pada hal-hal yang akan langsung terlihat oleh pemakai, seperti antarmuka pemakai dan fungsi-fungsi dasar.

3) Membangun *Prototype*

Model perancangan yang dihasilkan selanjutnya digunakan untuk membuat *Prototype* pertama, yang mungkin berbentuk:

➤ *Interactions Prototype*

Prototype perangkat lunak yang memungkinkan pemakai untuk memahami bagaimana berinteraksi dengan sistem perangkat lunak.

➤ *Subset function Prototype*

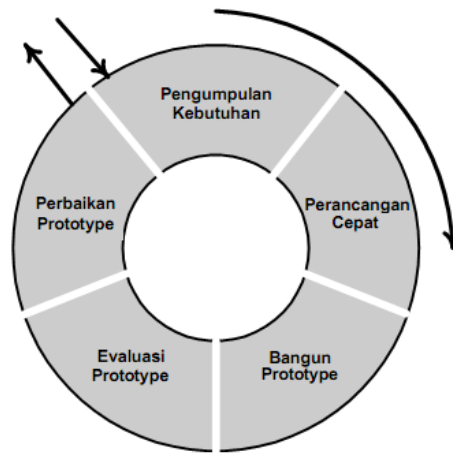
Prototype perangkat lunak yang sudah dapat digunakan tetapi baru mengimplementasikan sebagian dari fungsi-fungsi yang diinginkan.

➤ *Existing program*

Program sebenarnya yang mengimplementasikan sebagian besar atau seluruh fungsionalitas yang dibutuhkan, tetapi masih ada hal-hal utama lainnya yang harus disempurnakan pada pengembangan berikutnya.

4) Evaluasi *Prototype* Menguji coba dan mengevaluasi *Prototype* bersama-sama dengan pemakai untuk mendapatkan umpan balik yang dapat membantu pengembang memperbaiki *Prototype* yang sudah dibuat, atau membangun *Prototype* yang baru.

5) Perbaiki *Prototype* Melakukan penambahan dan perbaikan-perbaikan terhadap *Prototype* berdasarkan hasil evaluasi sampai diperoleh produk yang diinginkan.



Gambar Model *Prototyping*

C. RAD (*Rapid Application Development*) Model

Merupakan model proses pengembangan perangkat lunak secara linear sequential yang menekankan pada siklus pengembangan yang sangat singkat/pendek. Jika kebutuhan dipahami dengan baik, proses RAD memungkinkan tim pengembangan menciptakan “sistem fungsional yang utuh” dalam periode waktu yang sangat pendek (kira-kira 60-90 hari). Pendekatan RAD model menekankan cakupan :

1) Pemodelan bisnis (*Bussiness Modelling*)

Aliran informasi diantara fungsi-fungsi bisnis dimodelkan dengan suatu cara untuk menjawab pertanyaan-pertanyaan berikut : Informasi apa yang mengendalikan proses bisnis ? Kemana informasi itu pergi? Siapa yang memprosesnya ?

2) Pemodelan data (*Data Modelling*)

Aliran informasi yang didefinisikan sebagai bagian dari fase pemodelan bisnis disaring ke dalam serangkaian objek data yang dibutuhkan untuk menopang bisnis tersebut. Karakteristik/atribut dari masing-masing objek diidentifikasi dan hubungan antara objek-objek tersebut didefinisikan.

3) Pemodelan proses (*Process Modelling*)

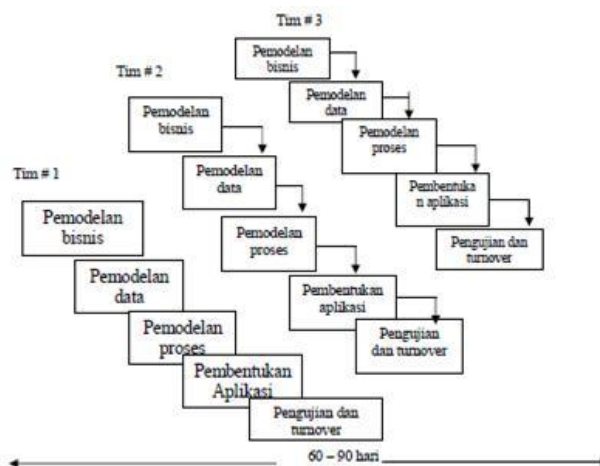
Aliran informasi yang didefinisikan dalam fase pemodelan data ditransformasikan untuk mencapai aliran informasi yang perlu bagi implementasi sebuah fungsi bisnis. Gambaran pemrosesan diciptakan untuk menambah, memodifikasi, menghapus atau mendapatkan kembali sebuah objek data.

4) Pembuatan aplikasi (*Application generation*)

Selain menciptakan perangkat lunak dengan menggunakan bahasa pemrograman generasi ketiga yang konvensional, RAD lebih banyak memproses kerja untuk memakai lagi komponen program yang telah ada atau menciptakan komponen yang bias dipakai lagi. Pada semua kasus, alat-alat Bantu otomatis dipakai untuk memfasilitasi konstruksi perangkat lunak.

5) Pengujian dan pergantian (*Testing and turnover*)

Karena proses RAD menekankan pada pemakaian kembali, banyak komponen yang telah diuji. Hal ini mengurangi keseluruhan waktu pengujian. Tapi komponen baru harus diuji.



Gambar Model RAD

Kelemahan model RAD :

- 1) Untuk proyek dengan skala besar, RAD membutuhkan sumber daya manusia yang cukup untuk membentuk sejumlah tim RAD yang baik.
- 2) RAD membutuhkan pengembang dan pemakai yang mempunyai komitmen dalam aktivitas *rapid-fire* untuk melaksanakan aktivitas melengkapi sistem dalam kerangka waktu yang singkat. Jika komitmen tersebut tidak ada, proyek RAD gagal.

Tidak semua aplikasi sesuai untuk RAD. bila system tidak dapat dimodulkan dengan teratur, pembangunan komponen penting pada RAD akan menjadi sangat problematic. RAD menjadi tidak sesuai jika risiko teknisnya tinggi.

D. Model Proses Perangkat Lunak Evolusioner

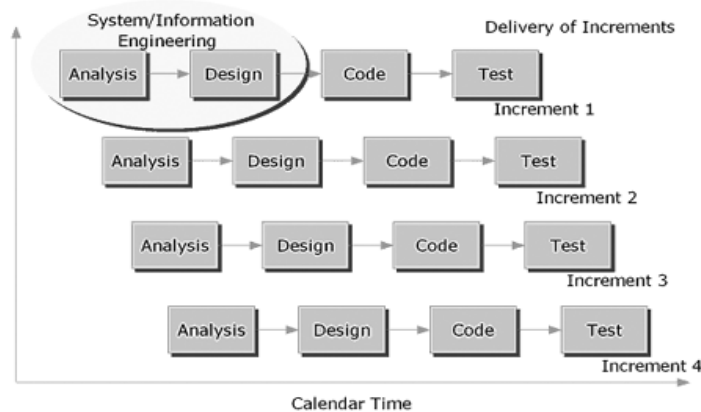
Model evolusioner adalah model iterative, ditandai dengan tingkah laku yang memungkinkan perekayasa perangkat lunak mengembangkan versi perangkat lunak yang lebih lengkap sedikit demi sedikit. Kebutuhan produk dan bisnis kadang-kadang berubah seiring dengan laju perkembangannya. Dalam situasi tersebut maupun lainnya, perekayasa perangkat lunak membutuhkan sebuah model proses yang sudah dirancang secara eksplisit untuk mengakomodasi produk perkembangan sepanjang waktu. Model ini bukan termasuk rekayasa perangkat lunak klasik. Model evolusioner meliputi :

- 1) Model penambahan

Model incremental menggabungkan elemen-elemen model sekuensial linier (diaplikasikan secara berulang) dengan filosofi *Prototype* iterative. Model ini memakai urutan-urutan linier di dalam model yang membingungkan, seiring dengan laju waktu kalender. Setiap urutan linier menghasilkan penambahan, perangkat lunak “yang bisa

Modul Rekayasa Perangkat Lunak STT-Garut 2016 disampaikan.” Contoh, perangkat lunak pengolah kata yang dikembangkan dengan menggunakan paradigma pertambahan akan menyampaikan manajemen file, editing, serta fungsi penghasilan dokumen pada pertambahan pertama, dan selanjutnya. Pertambahan pertama dapat disebut sebagai produk inti (*core product*).

Model ini berfokus pada penyampaian produk operasional dalam Setiap pertambahannya. Pertambahan awal ada di versi *stripped down* dari produk akhir, tetapi memberikan kemampuan untuk melayani pemakai dan juga menyediakan platform untuk evaluasi oleh pemakai.



Gambar Model Incremental

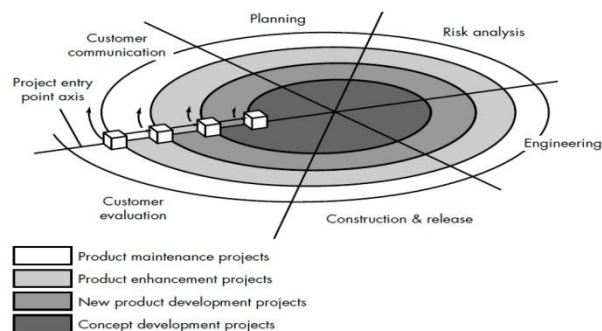
Perkembangan pertambahan, khususnya berguna pada staffing, tidak bisa dilakukan menggunakan implementasi lengkap oleh batas waktu bisnis yang sudah disepakati untuk proyek tersebut. Jika produk inti diterima dengan baik, maka staf tambahan bisa ditambahkan untuk mengimplementasi pertambahan selanjutnya.

2) Model spiral

Awalnya diusulkan oleh Boehm (BOE88), adalah model proses perangkat lunak yang evolusioner, merangkai sifat iterative dari *Prototype* dengan cara control dan aspek sistematis dari model sekuensial linier. Model yang berpotensi untuk pengembangan versi pertambahan perangkat lunak secara cepat. Model spiral dibagi

Modul Rekayasa Perangkat Lunak STT-Garut 2016
menjadi sejumlah aktifitas kerangka kerja atau wilayah tugas, antara lain :

- Komunikasi pelanggan, tugas-tugas yang dibutuhkan untuk membangun komunikasi yang efektif diantara pengembang dan pelanggan.
- Perencanaan, tugas-tugas yang dibutuhkan untuk mendefinisikan sumber-sumber daya, ketepatan waktu, dan proyek informasi lain yang berhubungan.
- Analisis resiko, tugas-tugas yang dibutuhkan untuk menaksir resiko-resiko, baik manajemen maupun teknis.
- Perekayasaan, tugas-tugas yang dibutuhkan untuk membangun satu atau lebih representasi dari aplikasi tersebut.
- Konstruksi dan peluncuran, tugas-tugas yang dibutuhkan untuk mengkonstruksi, menguji, memasang (instal), dan memberikan pelayanan kepada pemakai (contohnya pelatihan dan dokumentasi)
- Evaluasi pelanggan, tugas-tugas untuk memperoleh umpan balik dari pelanggan dengan didasarkan pada evaluasi representasi perangkat lunak, yang dibuat selama masa perekayasaan, dan dimplementasikan selama masa pemasangan.



Gambar Model spiral

Model spiral menjadi pendekatan yang realistis bagi perkembangan system dan perangkat lunak skala besar. Karena perangkat lunak terus bekerja selama proses bergerak, pengembang dan pemakai memahami, dan bereaksi lebih baik terhadap resiko dari Setiap tingkat evolusi. Model spiral menggunakan *Prototype* sebagai mekanisme pengurangan resiko.

Model spiral membutuhkan keahlian penafsiran resiko yang masuk akal, dan sangat bertumpu pada keahlian ini untuk mencapai keberhasilan. Jika sebuah resiko tidak dapat ditemukan dan diatur, pasti akan terjadi masalah. Model ini membutuhkan waktu bertahun-tahun sampai kehandalannya bisa dipertimbangkan dengan kepastian absolute.

3) Model rakitan komponen

Model ini menggabungkan beberapa karakteristik model spiral. Bersifat evolusioner, sehingga membutuhkan pendekatan iterative untuk menciptakan perangkat lunak. Tetapi model ini merangkai aplikasi dari komponen perangkat lunak sebelum dipaketkan (kadang disebut kelas). Aktivitas perangkat lunak dimulai dengan identifikasi calon kelas. Dipenuhi dengan mengamati data yang akan dimanipulasi oleh aplikasi dan algoritma-algoritma yang akan diaplikasikan. Data dan algoritma yang berhubungan dikemas ke dalam kelas. Kelas-kelas tersebut disimpan dalam *class library* (tempat penyimpanan).

Model ini membawa pada penggunaan kembali perangkat lunak, dan kegunaan kembali itu memberi sejumlah keuntungan yang bisa diukur pada rekayasa perangkat lunak.

4) Model perkembangan konkrue

Representasi aktivitas dalam model ini, meliputi aktivitas analisis, bisa berada dalam salah satu dari keadaan-keadaan yang dicatat pada saat tertentu. Dengan cara yang sama, aktivitas yang lain (desain atau komunikasi pelanggan) dapat direpresentasikan dalam sebuah sikap

Modul Rekayasa Perangkat Lunak STT-Garut 2016
yang analog. Semua aktifitas ada secara konkruen tetapi dia tinggal didalam keadaan yang berbeda. Model ini sering digunakan sebagai paradigme bagi pengembangan aplikasi klien/server.

Kenyataanya model proses konkruen bisa diaplikasikan ke dalam semua tipe perkembangan perangkat lunak, dan memberikan gambaran akurat mengenai keadaan tertentu dari sebuah proyek. Selain membatasi aktifitas perekayasa ke dalam sederetan kejadian, model proses juga mendefinisikan jaringan aktivitas.

E. Model *Rational Unified Process* (RUP)

RUP adalah proses pengembangan perangkat lunak berbasis UML (*Unified Modeling Language*) yang mempunyai karakteristik:

1) Berulang (*iterative*)

Tahap pengembangan untuk setiap produk yang diserahkan (*release*) dilaksanakan secara berulang.

2) *Architecture centric*

Menggunakan arsitektur sistem sebagai artifak utama untuk konseptualisasi, konstruksi, pengelolaan, dan penyusunan sistem selama pengembangan.

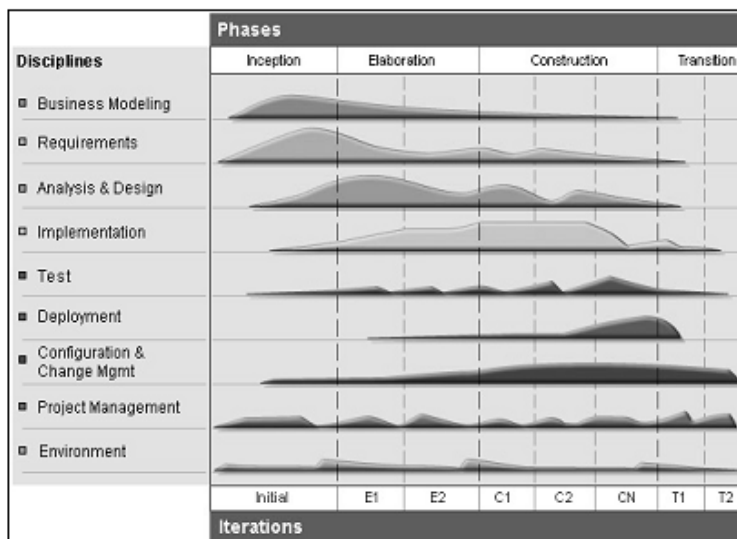
3) *Use case-driven*

Menggunakan use case sebagai artifak utama untuk menetapkan perilaku sistem yang diinginkan dan untuk mengkomunikasikan perilaku sistem tersebut kepada para *stakeholder* sistem.

4) *Risk-driven*

Menghilangkan atau mengurangi resiko-resiko yang dapat menghambat kesuksesan proyek.

Sasaran RUP adalah menghasilkan perangkat lunak yang berkualitas tinggi, sesuai kebutuhan pemakai, dengan jadwal dan biaya sesuai dengan yang direncanakan



Gambar Arsitektur *Rational Unified Process*

Proses pengembangan pada RUP dinyatakan dalam dua dimensi, atau dua sumbu (lihat Gambar).

Sumbu horizontal (sumbu x) merepresentasi waktu dan menunjukkan aspek dinamis dari proses, yaitu siklus, tahap, iterasi, dan milestone.

- Sumbu vertikal (sumbu y) merepresentasikan aspek statis dari proses, yaitu aktivitas, artifak, pelaksana kerja (worker) dan aliran kerja (workflow).

Tahapan RUP

Berdasarkan Gambar sebelumnya, tahap (phases) pelaksanaan pengembangan pada RUP meliputi:

1) Permulaan (inception)

Tahap inception fokus pada penentuan manfaat perangkat lunak yang harus dihasilkan, penetapan proses-proses bisnis (business case), dan perencanaan proyek.

2) Pemerincian (elaboration)

Tahap untuk menentukan use case (set of activities) dari perangkat lunak berikut rancangan arsitekturnya.

3) Konstruksi (construction)

Membangun produk perangkat lunak secara lengkap yang siap diserahkan kepada pemakai.

4) Transisi (transition)

Menyerahkan perangkat lunak kepada pemakai, mengujinya di tempat pemakai, dan memperbaiki masalah-masalah yang muncul saat dan setelah pengujian.

Aliran Kerja RUP

Ada dua jenis aliran kerja (workflow) pada RUP, yaitu aliran kerja utama dan aliran kerja pendukung.

➤ Aliran Kerja Utama

1) Pemodelan bisnis (business modeling)

Mendeskripsikan struktur dan proses-proses bisnis organisasi.

2) Kebutuhan (*requirements*)

Mendefinisikan kebutuhan perangkat lunak dengan menggunakan metode use case.

3) Analisis dan perancangan (analysis and design)

Mendeskripsikan berbagai arsitektur perangkat lunak dari berbagai sudut pandang.

4) Implementasi (implementation)

Menulis kode-kode program, menguji, dan mengintegrasikan unit-unit programnya.

5) Pengujian (testing)

Mendeskripsikan kasus uji, prosedur, dan alat ukur pengujian.

6) Deployment

Menangani konfigurasi sistem yang akan diserahkan.

➤ Aliran Kerja Pendukung

- 1) Manajemen konfigurasi dan perubahan (configuration and change management) Mengendalikan perubahan dan memelihara artifak-artifak proyek.
- 2) Manajemen proyek (project management) Mendeskripsikan berbagai strategi pekerjaan dengan proses yang berulang.
- 3) Lingkungan (environment) Menangani infrastruktur yang dibutuhkan untuk mengembangkan sistem.

2.4. Metode Pengembangan Perangkat Lunak

Yang dimaksud dengan metode pengembangan perangkat lunak disini adalah pendekatan, sudut pandang, atau kumpulan aturan yang harus diikuti untuk menyelesaikan tahap-tahap pekerjaan saat melaksanakan pengembangan perangkat lunak. Ada beberapa pendekatan atau metode yang sudah dikenal, dan berikut adalah penjelasan untuk metode-metode yang sering digunakan.

2.4.1. Konvensional atau Tradisional

Sudut pandang pengembangan pada metode ini ditujukan pada sistem fisik (prosedur kerja) yang ada dalam suatu organisasi. Tahap pengembangan biasanya diawali dengan mengamati dokumen apa yang menjadi media data atau informasi, bagaimana dokumen tersebut terbentuk, bagaimana dokumen tersebut mengalir dari satu bagian ke bagian yang lain, proses apa yang terjadi pada dokumen tersebut, dan seterusnya. Hasil setiap tahap pengembangan dimodelkan dengan menggunakan alat bantu yang disebut bagan alir (*flowchart*) yang menggunakan simbol-simbol tertentu yang sudah dibakukan oleh ANSI (*American National Standard Institute*). Pemodelan yang dibuat pada umumnya adalah:

1) Peta aliran kerja (*flowmap*)

Menggambarkan prosedur kerja secara fisik, baik yang masih manual atau yang sudah menggunakan komputer, dilihat berdasarkan aliran dokumen yang digunakan.

2) *System Flowchart*

Menggambarkan kerangka proses program dilihat dari masukan, proses dan keluarannya.

3) *Program Flowchart*

Menggambarkan urutan logika proses dari program.

2.4.2. Berorientasi Aliran Data atau Fungsi

Sudut pandang pengembangan dengan metode ini adalah aspek fungsional dan perilaku laku sistem. Pengembang harus mengetahui fungsi-fungsi atau proses-proses apa saja yang ada dalam sistem, data apa yang menjadi masukannya, dimana data tersebut disimpan, transformasi apa yang akan dilakukan terhadap data tersebut, dan apa yang menjadi hasil transformasinya. Selain itu, pengembang harus mengetahui keadaan, perubahan, kondisi, dan aksi dari sistem.

Salah satu teknik yang paling populer untuk metode ini adalah Teknik Terstruktur (*Structured Technique*) yang meliputi analisis, perancangan, dan pemrograman. Pada teknik ini, hasil analisis dan perancangan dimodelkan dengan menggunakan alat bantu pemodelan seperti:

1) Diagram Aliran Data (*Data Flow Diagram* atau DFD)

Digunakan untuk menggambarkan aliran data dalam sistem, sumber dan tujuan data, proses yang mengolah data tersebut, dan tempat penyimpanan datanya.

2) Kamus Data (*Data Dictionary*)

Digunakan untuk mendeskripsikan struktur dari data atau informasi yang mengalir (ada) dalam sistem.

3) Spesifikasi Proses (*Process Specification* atau *P-Spec*)

Digunakan untuk menggambarkan deskripsi dan spesifikasi dari setiap proses yang ada pada sistem, biasanya untuk proses-proses atomik, dengan menggunakan notasi yang disebut *Structured English*.

4) Diagram Transisi Keadaan (*State Transition Diagram* atau STD)

Digunakan untuk menggambarkan perilaku sistem, dan biasanya untuk sistem waktu nyata (*real time*).

5) Diagram E-R

Digunakan untuk menggambarkan hubungan (*relationship*) antara entitas-entitas dilihat dari aspek datanya (atau hubungan antar tempat penyimpanan).

6) Bagan Terstruktur (*Structure Chart*)

Digunakan untuk menggambarkan struktur atau arsitektur program.

7) Pseudo-code

Digunakan untuk menuliskan algoritma setiap modul program.

2.4.3. Berorientasi Data

Sudut pandang pengembangan pada metode ini adalah struktur data dari dokumen masukan/keluaran yang digunakan dalam sistem. Tahap pelaksanaan pengembangannya pada umumnya mengikuti urutan sebagai berikut:

- 1) Mengidentifikasi entitas-entitas atau item-item yang menjadi *objek informasi kunci* berikut operasi-operasinya.
- 2) Menyatakan struktur informasi (dari dokumen) secara hirarki dengan menggunakan konstruksi *sequence*, *selection* dan *repetition*.
- 3) Memetakan hirarki struktur informasi menjadi struktur program.

data ini diantaranya adalah:

- Data Structured System Development (DSSD) Diperkenalkan pertama kali oleh J.D. Warnier (1974) dan kemudian oleh Ken Orr (1977) sehingga sering disebut juga teknik Warnier-Orr. Teknik ini menggunakan perangkat *entity diagram*, *assembly line diagram* dan *Warnier-Orr diagram* untuk memodelkan hasil analisis dan perancangannya.
- Jackson System Development (JSD) Dikembangkan oleh M.A. Jackson (1975) dengan menggunakan perangkat pemodelan yang disebut *structure diagram* dan *system specification diagram*.

2.4.4. Berorientasi Objek

Berbeda dengan pendekatan-pendekatan sebelumnya, metode berorientasi objek memandang perangkat lunak yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata. Pada metode ini, informasi dan proses yang dimiliki oleh suatu objek “dienkapsulasi” (dibungkus) dalam satu kesatuan. Gambar berikut menunjukkan cara pandang metode berorientasi objek dibandingkan dengan metode berorientasi fungsi.



Gambar Sudut Pandang Metode Berorientasi Objek vs Fungsi

Beberapa teknik pengembangan perangkat lunak yang berorientasi

objek ini diantaranya adalah:

- *Object Oriented Analysis (OOA)* dan *Object Oriented Design (OOD)* dari Peter Coad dan Edward Yourdon (1990).
- *Object Modeling Technique (OMT)* dari James Rumbaugh, Michael Blaha, William Premerlan, Frederick Eddy dan William Lorensen
- (1991). *Object Oriented Software Engineering (OOSE)* dari Ivar Jacobson
- (1992). Metode Booch dari Grady Booch (1994). Syntropy dari Steve Cook dan John Daniels (1994).

2.5. Alat Bantu Pengembangan Perangkat Lunak

Pada table di bawah ini akan di jelaskan beberapa alat bantu dalam pengembangan perangkat lunak

Konvensional	Tahap Pengembangan	Alat Bantu
	Analisis	Flowmap
	Perancangan	System Flowchart, Program Flowchart
	Implementasi	Bahasa pemrograman
Fungsi	Tahap Pengembangan	Alat Bantu
	Analisis	DFD, DD, p-spec, ERD
	Perancangan	Structure chart, pseudo-code
	Implementasi	Bahasa pemrograman prosedural
Objek	Tahap Pengembangan	Alat Bantu
	Analisis dan Perancangan	UML
	Implementasi/Pemrograman	Bahasa pemrograman objek

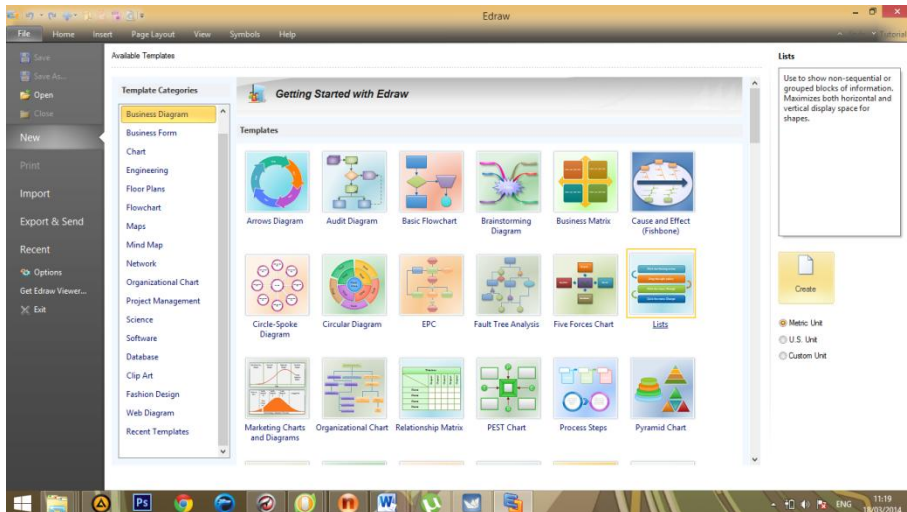
2.5.1. Case Tools untuk membantu Pengembangan Perangkat Lunak

CASE (*Computer-Aided Software Engineering*) Tools merupakan perangkat komputer yang berbasis produk yang memiliki tujuan untuk mendukung satu atau lebih rekayasa perangkat lunak dalam proses pengembangan software (perangkat lunak). Contoh: Visio, Edraw, Argo UML, DIA Diagram, Rational Rose, dsb.

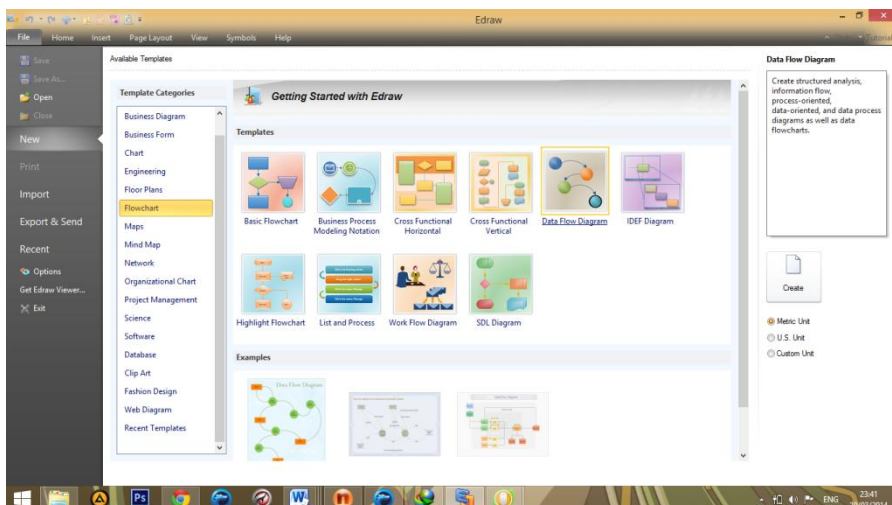
1) Penggunaan Edraw

Edraw adalah aplikasi pembuat diagram yang biasa di gunakan untuk membuat *mind mapping* dan diagram presentasi. Edraw memiliki fitur menarik seperti fasilitas *template diagram* yang unik, *background slide* dll.

Contoh pembuatan DFD dengan Edraw. Buka aplikasi edraw maka akan muncul halaman seperti berikut:



Pilih Flowchat pada colom Template Categories dan pilih DFD

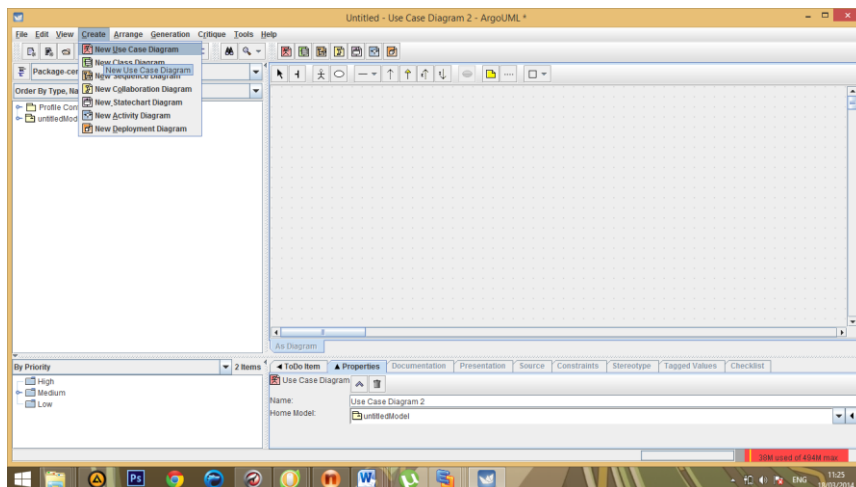


2) Penggunaan Argo UML

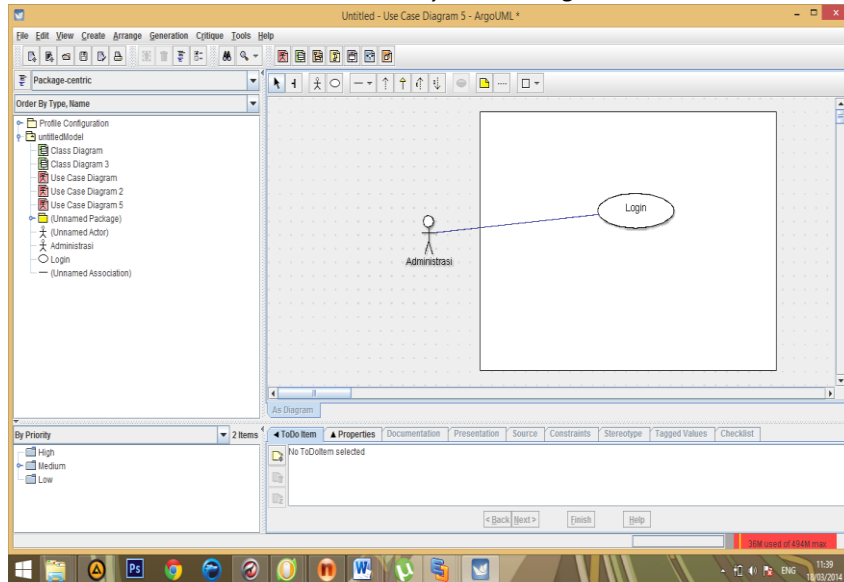
Argo UML merupakan salah satu Case tools yang dapat digunakan untuk mendesain UML , class diagram ataupun diagram-diagram lainnya. Yang tak kalah pentingnya Argo UML adalah salah satu tools yang bersifat *open source*.



Gambar halaman awal ArgoUML



Membuat UseCase diagram dengan argoUML



Membuat Actor dan UseCase di ArgoUML

3. REFERENSI

Referensi-referensi yang dapat digunakan selain modul praktikum dan materi mata kuliah Rekayasa Perangkat Lunak diantaranya :

- Pressman, Roger S., *"Software Engineering : A Practitioner's Approach 4th Edition"*, Mc-Graw Hill, 1997.
- Yourdon, Edward, *"Modern Structured Analysis"*, Prentice Hall, 1989.
- Davis, Allan M., *"Software Requirements : Analysis & Specification"*, Prentice
- Hall. Sommerville, Ian, *"Software Engineering 6th Edition"*, Addison-Wesley, 2001.
- Wahono, Romi S., *"Requirements Engineering: Mari Pecahkan Masalah Batu!"*, <http://ilmukomputer.org>, 2006.
- IBM Corporation. 2007. *The IBM Rational Unified Process for System z*. New York. IBM Corporation, (diunduh dari <http://www.redbooks.ibm.com/>).
- Kumpulan materi kuliah RPL
- Internet

MODUL 0

KONSEP ANALISIS DAN DESAIN BERORIENTASI OBJEK

1. Tujuan

- 1) Mahasiswa memiliki pemahaman yang baik mengenai segala sesuatu yang berkenaan dengan paradigma rekayasa perangkat lunak berorientasi object termasuk didalamnya konsep pembungkusan (encapsulation) pewarisan (inheritance), dan sebagainya'
- 2) Mahasiswa mengetahui penggunaan paradigma rekayasa perangkat lunak berorientasi objek dalam tahapan analisis dan perancangan perangkat lunak.
- 3) Membedakan perancangan perangkat lunak berdasarkan rekayasa perangkat lunak prosedural dengan perancangan berdasarkan rekayasa perangkat lunak objek.
- 4) Mahasiswa mengenal keunggulan paradigma rekayasa perangkat lunak berorientasi objek.
- 5) Mahasiswa mengenal metodologi dalam rekayasa perangkat lunak berorientasi objek'.

2. TEORI

2.1 Konsep Dasar Pendekatan Objek

- Suatu teknik atau cara pendekatan baru dalam melihat permasalahan dari sistem (sistem perangkat lunak, sistem informasi, atau sistem lainnya).
- Pendekatan berorientasi objek akan memandang sistem yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata.
- Ada banyak cara untuk mengabstraksikan dan memodelkan objek-objek tersebut, mulai dari abstraksi objek, kelas, hubungan antar kelas sampai abstraksi sistem.

- Saat mengabstraksikan dan memoderkan objek ini, data dan proses - proses yang dipunyai oleh objek akan dienkapsulasi (dibungkus) menjadi satu kesatuan. contoh: Tinjau aktivitas kuliah pada suatu sistem akademik sebagai berikut:



- Dari aktivitas kuliah tersebut, secara eksplisit ada 3 objek yang langsung dapat dikenali yaitu Dosen yang memberikan kuliah, Mahasiswa yang mengikuti kuliah, dan Materi Kuliah yang disampaikan. secara implisit, ada 2 objek lain yang bisa dikenali lagi yaitu Jadwal kapan kuliah diadakan dan Nilai yang didapat mahasiswa dari kuliah yang sudah diikutinya. Abstraksi dan pemodelan untuk salah satu dari kelima objek tersebut, misalnya untuk objek Dosen adalah:



Gambar Objek Dosen

- Dalam rekayasa perangkat lunak, konsep pendekatan berorientasi objek dapat diterapkan pada tahap analisis, perancangan, pemrograman, dan pengujian perangkat lunak.
- Ada berbagai teknik yang dapat digunakan pada masing-masing tahap tersebut, dengan aturan dan alat bantu pemodelan tertentu.

2.2 Objek dan Kelas

Objek

- Objek adalah abstraksi dari sesuatu yang mewakili dunia nyata seperti benda, manusia, satuan organisasi, tempat kejadian, struktur, status atau hal-hal lain yang bersifat abstrak.
- Suatu entitas yang mampu menyimpan informasi (status) dan mempunyai operasi (kelakuan) yang dapat diterapkan atau dapat berpengaruh pada status objeknya.
- Dalam konteks OOP, objek adalah instansiasin(yang dibentuk secara seketika) dari kelas pada saat eksekusi (seperti halnya deklarasi variabel pada pemograman prosedural). Jadi semua objek adalah instan dari kelas.
- Objek mempunyai siklus hidup: diciptakan, dimanipulasi, dan dihancurkan.

Kelas

- Kelas adalah kumpulan dari objek-objek dengan karakteristik yang sama.
- Kelas adalah definisi statik dari himpunan objek yang sama yang mungkin lahir atau diciptakan dari kelas tersebut.
- Sebuah kelas akan mempunyai sifat(atribut), kelakuan (operasi), hubungan (relotionship) dan arti.
- Suatu kelas dapat diturunkan dari kelas yang lain, dimana atribut dari kelas semula dapat diwariskan ke kelas yang baru.

1) Property Objek

Sebuah objek pada dasarnya mempunyai foperty sebagai berikut:

- Atribut
 - Nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek.
 - Merupakan ciri dari sebuah objek.
 - Dipunyai secara individual oleh sebuah obiek.
 - Contoh : berat, warna, jenis, nama, dsb.

- Layanan (service)
 - Metode atau operasi yang berfungsi untuk memanipulasi objek itu sendiri.
 - Fungsi atau transformasi yang dapat dilakukan terhadap objek atau dilakukan oleh objek.
 - Dapat berasal dari: model objek, event, aktivitas atau aksi keadaan, fungsi, kelakuan dunia nyata.
- Contoh: Read, Write, Move, Copy dan sebagainya.

2.3 Sistem Berorientasi Objek

1) Definisi

- Sebuah sistem yang dibangun dengan berdasarkan metode berorientasi objek adalah sebuah sistem yang komponennya dibungkus (dienkapsulasi) menjadi kelompok data dan fungsi.
- Setiap komponen dalam sistem tersebut dapat mewarisi atribut dan sifat dari komponen lainnya, dan dapat berinteraksi satu sama lainnya.

2) Karakteristik Sistem Berorientasi Objek

Karakteristik atau sifat-sifat yang dimiliki sebuah sistem berorientasi objek adalah:

- Abstraksi
Prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.
- Enkapsulasi
Pembungkusan atribut data dan layanan (operasi-operasi) yang dimiliki objek, untuk menyembunyikan implementasi dari objek sehingga objek lain tidak mengetahui cara kerjanya.

- **Pewarisan (*inheritance*)**
Mekanisme yang memungkinkan satu objek (baca : kelas) mewarisi sebagian atau seluruh definisi dari objek lain sebagai bagian dari dirinya.
- ***Reusability***
Pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan pada permasalahan lainnya yang melibatkan objek tersebut.
- **Generalisasi dan Spesialisasi**
Menunjukkan hubungan antara kelas dan objek yang umum dengan kelas objek yang khusus.
- **Komunikasi antar Objek**
Komunikasi antar objek dilakukan lewat pesan (message) yang dikirim dari satu objek ke objek lainnya.
- ***Polymorphism***
Kemampuan suatu objek untuk digunakan di banyak tujuan yang berbeda dengan nama yang sama sehingga menghemat baris program.

2.4 Analisis Berorientasi Objek

- Investigasi masalah untuk menemukan (mengidentifikasi) dan mendefinisikan objek-objek atau konsep-konsep yang ada di ruang masalah.
- Proses untuk menentukan objek-objek potensial yang ada dalam sistem dan mendeskripsikan karakteristik dan hubungannya dalam sebuah notasi formal.
- Aplikasi konsep berorientasi objek untuk memodelkan permasalahan dan sistem, baik untuk lingkup perangkat lunak maupun nonperangkat lunak.

Metode Analisis secara umum

Pada prinsipnya semua metode analisis berorientasi objek adalah sama, perbedaan hanya terletak pada sudut pandang dan teknis pelaksanaannya. Secara umum, metode analisis berorientasi objek mencakup representasi kelas dan hirarki kelas, model hubungan objek, dan model perilaku objek. Tahap atau skema pelaksanaan analisis berorientasi objek :

- Tentukan kebutuhan pemakai.
- Identifikasi kelas dan objek.
- Identifikasi atribut dan layanan untuk setiap objek.
- Definisikan struktur dan hirarki.
- Buat model hubungan objek.
- Buat model perilaku objek.

2.5 UML

UML adalah sebuah bahasa untuk menentukan, memvisualisasi, membangun dan mendokumentasikan artifacts (bagian dari informasi yang digunakan atau dihasilkan oleh proses pembuatan perangkat lunak, artifact tersebut dapat berupa model, deskripsi atau perangkat lunak) dari sistem perangkat lunak, seperti pada pemodelan bisnis dan sistem non perangkat lunak lainnya [HAN98].

Selain itu, UML adalah bahasa pemodelan yang menggunakan konsep orientasi object. UML dibuat oleh Grady Booch , James Rumbaugh , dan Ivar Jacobson di bawah bendera Rational Software Corp [HAN98]. UML menyediakan notasi-notasi yang membantu memodelkan sistem dari berbagai perspektif. UML tidak hanya digunakan dalam pemodelan perangkat lunak, namun hampir dalam semua bidang yang membutuhkan pemodelan. UML juga merupakan suatu kumpulan teknik terbaik yang telah terbukti sukses dalam memodelkan sistem yang besar dan kompleks.

Tujuan Penggunaan UML:

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

MODUL 1

USE CASE DIAGRAM

Tujuan Praktikum

- 1) Praktikan mampu membuat sebuah skenario suatu sistem yang nantinya dapat diimplementasikan menjadi sebuah perangkat lunak'
- 2) Praktikan bisa memahami alur dari setiap tahap yang digunakan dalam perancangan perangkat lunak menggunakan UML'
- 3) Praktikan dapat memahami hubungan antara actor dengan usecase diagram.
- 4) Praktikan mampu membuat Usecase diagram dari skenario yang telah ada.

TEORI

Kelakuan Sistem :

- 1) Kebutuhan sistem adalah fungsionalitas apa yang mesti disediakan oleh sistem, apakah didokumentasikan pada model use case yang menggambarkan fungsi sistem yang diharapkan (usecase) yang mengelilinginya (actor) dan hubungan antara actor dengan usecase (usecase diagram).
- 2) Usecase model dimulai pada tahap inception dengan mengidentifikasi actor dan usecase utama pada sistem. Kemudian model ini diolah lebih matang ditahap elaboration untuk memperoleh lebih detail informasi yang ditambahkan Pada use case.

Komponen-komponen yang terlibat dalam usecase diagram :

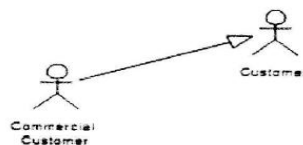
1.1 Actor

Pada dasarnya actor bukanlah bagian dari usecase diagram, namun untuk dapat terciptanya suatu use case diagram diperlukan beberapa actor dimana actor tersebut mempresentasikan seseorang atau sesuatu (seperti perangkat, sistem lain) yang berinteraksi dengan sistem. Sebuah actor

mungkin hanya memberikan informasi inputan pada sistem, hanya menerima informasi dari sistem atau keduanya menerima dan memberi informasi pada sistem, actor hanya berinteraksi dengan use case tetapi tidak memiliki kontrol atas use case. Actor digambarkan dengan stick man.

Actor dapat digambarkan secara umum atau spesifik, dimana untuk membedakannya kita dapat menggunakan relationship

Contoh :



Gambar Actor

Ada beberapa kemungkinan yang menyebabkan actor tersebut terkait dengan sistem antara lain:

- Yang berkepentingan terhadap sistem dimana adanya arus informasi baik yang diterimanya maupun yang dia inputkan ke sistem.
- Orang ataupun pihak yang akan mengelola sistem tersebut.
- External resource yang digunakan oleh sistem.
- Sistem lain yang berinteraksi dengan sistem yang akan dibuat.

1.2 Use Case

use case adalah gambaran fungsionalitas dari suatu sistem, sehingga customer atau pengguna sistem paham dan mengerti mengenai kegunaan sistem yang akan dibangun.

Catatan:

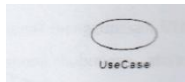
use case diagram adalah penggambaran sistem dari sudut pandang pengguna sistem tersebut (user), sehingga pembuatan use case lebih dititikberatkan pada

Modul Rekayasa Perangkat Lunak STT-Garut 2016

fungsionalitas yang ada pada sistem, bukan berdasarkan alur atau urutan kejadian.

a. Cara menentukan usecase dalam suatu sistem:

- Pola perilaku perangkat lunak aplikasi.
- Gambaran tugas dari sebuah actor.
- Sistem atau “benda” yang memberikan sesuatu yang bernilai kepada actor.
- Apa yang dikerjakan oleh suatu perangkat lunak (*bukan bagaimana cara mengerjakannya).



b. Relasi dalam Usecase

Ada beberapa relasi yang terdapat pada usecase diagram:

- 1) Association, menghubungkan link antar element.
- 2) Generalization, disebut juga inheritance (pewarisan), sebuah elemen dapat merupakan spesialisasi dari elemen lainnya.
- 3) Dependency, sebuah element bergantung dalam beberapa cara ke element lainnya.
- 4) Aggregation, bentuk association dimana sebuah element berisi elemen lainnya.

Tipe relasi/ stereotype yang mungkin terjadi pada usecase diagram:

- 1) <<include>> , yaitu kelakuan yang harus terpenuhi agar sebuah event dapat terjadi, dimana pada kondisi ini sebuah usecase adalah bagian dari use case lainnya.
- 2) <<extends>>, kelakuan yang hanya berjalan dibawah kondisi tertentu seperti menggerakkan alarm.

- 3) <<communicates>>, mungkin ditambahkan untuk asosiasi yang menunjukkan asosiasinya adalah communicates association. Ini merupakan pilihan selama asosiasi hanya tipe relationship yang dibolehkan antara actor dan use case.

1.3 Use Case Diagram

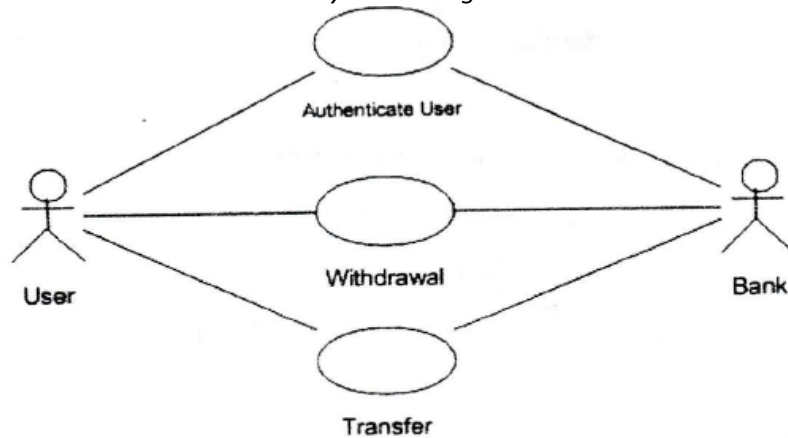
Adalah gambaran graphical dari beberapa atau semua actor, use case, dan interaksi diantaranya yang memperkenalkan suatu sistem.

1.4 Cara Membuat Usecase Diagram

- 1) Buka software aplikasi ArgoUML dan buatlah use case Diagram Dengan kasus sebagai berikut:

Sebuah bank mengoperasikan ATM dan mengelola banyak tabungan, setiap nasabah memiliki setidaknya satu rekening tabungan pada satu bank tertentu. setiap tabungan dapat diakses melalui kartu debit. proses utama sistem ATM berkomunikasi dengan pusat komputer dan didesain untuk menangani beberapa transaksi. setiap transaksi menunjuk sebuah tabungan tertentu. suatu transaksi akan menghasilkan satu dari dua hal berikut: transaksi diterima atau mengeluarkan pesan penolakan transaksi'. Untuk melakukan sebuah transaksi akan melalui dua tahap: pengecekan tabungan dan pemroses transaksi. proses pengecekan tabungan akan menetapkan persetujuan untuk proses transaksi. Jika persetujuan ditolak, ATM akan mengeluarkan pesan penolakan, namun jika diterima, transaksi akan diproses dengan menggunakan nomor rekening tabungan dan ATM membaca dari kartu debit.

Pengecekan tabungan dilakukan bersamaan pada saat ATM memvalidasi kartu debit dari bank yang bersangkutan. Jika kartu valid, password akan dicek dengan nasabah.



Use Case Diagram Kasus ATM

- 2) Setelah usecase diagram terbentuk, buatlah scenario use case seperti contoh berikut:

Skenario use case

Nama use case : Authenticate user
 Actor : User, bank
 Type : Primary
 Tujuan : verifikasi user

ACTOR	SISTEM
1. <u>User</u> memasukkan <u>kartu debit</u>	
	2. <u>ATM</u> meminta <u>PIN</u> dari <u>user</u>
3. <u>User</u> memasukkan <u>PIN</u> dan menekan OK	
	4. <u>ATM</u> memverifikasi dengan <u>Bank</u> bahwa <u>kartu</u> dan <u>PIN</u> adalah legal dari <u>Rekening</u> yang benar
	5. <u>ATM</u> meminta jenis <u>transaksi</u>

Nama use case : Withdrawal
 Actors : User, bank
 Type : Primary
 Tujuan : Penarikan uang secara cash
 Deskripsi : User datang ke ATM dengan kartu debit untuk melakukan penarikan tunai. User memasukkan kartu ke ATM. ATM meminta user untuk memasukkan PIN. User memasukkan PIN dan sistem mengotorisasi penarikan tunai. ATM mengeluarkan uang dan mengeluarkan nota. ATM mengirim transaction record ke bank untuk meng-update saldo tabungan. Setelah selesai, user meninggalkan ATM dengan membawa uang dan nota tadi.

ACTOR	SISTEM
1. User memilih menu withdrawl	
	2. ATM meminta jumlah uang yang akan ditarik
3. User memasukkan jumlah uang yang akan ditarik	
	4. ATM mengecek jumlah uang yang akan ditarik dengan saldo minimal yang diperbolehkan pada bank tersebut.
	5. Update saldo
	6. ATM mengeluarkan uang
	7. ATM mencetak nota dan mengeluarkan kartu

TUGAS

- 1) Buat *Use Case diagram* beserta skenarionya dengan kasus **ATM setor tunai**
- 2) Beri kesimpulan dan pemahaman anda mengenai Use Case Diagram

MODUL II

CANDIDATE CLASS & INTERACTION DIAGRAM

Tujuan

- Praktikan dapat menentukan *candidate class* dari skenario yang telah ada.
- Praktikan dapat menggambarkan interaction diagram baik dengan sequence maupun collaboration diagram.
- Praktikan dapat membedakan antara sequence diagram dengan collaboration diagram dan menggunakannya dengan perancangan perangkat lunak dengan UML.

TEORI

2.1 Definisi Object dan Class

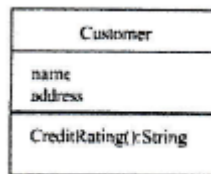
Object adalah gambaran dari entity, baik dunia nyata atau konsep dengan batasan batasan dan pengertian yang tepat. object bisa mewakili sesuatu yang nyata seperti komputer, mobil atau dapat berupa konsep seperti proses kimia, transaksi bank, permintaan pembelian, dll. setiap object dalam sistem memiliki tiga karakteristik yaitu state (status), Behaviour (sifat) dan Identity (identitas).

Cara mengidentifikasi object:

- 1) Pengelompokan berdasarkan kata/frase benda pada skenario.
- 2) Berdasarkan daftar kategori object, antara lain:
 - Object fisik, contoh: pesawat telepon.
 - Spesifikasi/ rancangan/ deskripsi, contoh: deskripsi pesawat.
 - Tempat, contoh: gudang.
 - Transaksi, contoh: penjualan.
 - Butir yang terlibat pada transaksi, contoh: barang jualan.
 - Peran, contoh: pelanggan
 - Wadah, contoh: pesawat terbang.
 - Piranti, contoh: PABX.

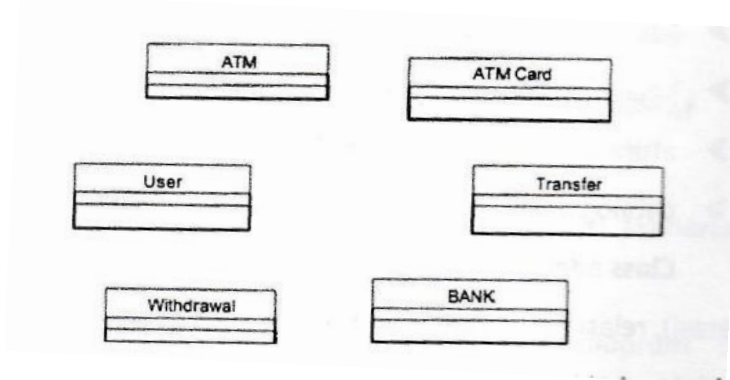
- Kata benda abstrak, contoh: kecanduan
- Kejadian, contoh: pendaratan.
- aturan atau kebijakan, contoh: aturan diskon.
- catalog atau rujukan, contoh: daftar pelanggan.

Class adalah deskripsi sekelompok object dari property {atribut}, sifat (operasi), relasi antar object dan semantik yang umum. class merupakan template untuk membentuk object. setiap object merupakan contoh dari beberapa class dan object tidak dapat menjadi contoh lebih dari satu class. Penamaan class menggunakan kata benda tunggal yang merupakan abstraksi yang terbaik. Pada UML, class digambarkan dengan segi empat yang dibagi. Bagian atas merupakan nama dari class. Bagian yang tengah merupakan struktur dari class (atribut) dan bagian bawah merupakan sifat dari class (operasi).



Dari skenario pada modul sebelumnya untuk studi kasus pada ATM, kita dapat mendefinisikan candidate class, dimana candidate class secara kasar dapat diambil dari kata benda yang ada, atau sesuai dengan apa yang telah dijelaskan diatas.

No	Kategori Object	Nama Object	Perlu/tidak
1.	Object Fisik	ATM (Mesin), ATM card	Perlu
2.	Transaksi	Withdrawal, Transfer	Perlu
3.	Butir yang terlibat pada transaksi
4.	Peran	User(Pemegang ATMCARD) Bank	Perlu Perlu
5.	Piranti	ATM Komputer	Perlu Tidak perlu
6.	Proses	Withdrawal Update	Perlu Tidak perlu
7.	Katalog	Daftar Account	Perlu



Untuk memahami Class lebih lanjut akan kita bahas pada modul selanjutnya.

2.2 Interaction Diagram

2.2.1 Use Case Realization

Fungsionalitas Usecase direpresentasikan dengan aliran peristiwa-peristiwa. Skenario digunakan untuk menggambarkan bagaimana use case-use case direalisasikan sebagai interaksi antara object-object.

Use case realization menggambarkan bagaimana realisasi dari setiap use case yang ada pada use case model. Untuk menggambarkan bagaimana realisasi dari suatu use case dapat menggunakan beberapa diagram, diantaranya adalah Class Diagram owned by Use Case Realization serta Interaction Diagram.

- **Interaction Diagram** merupakan model yang menjelaskan bagaimana sejumlah object bekerjasama dengan beberapa kelakuan. Interaction diagram menerangkan kelakuan dari suatu usecase. Diagram ini menggambarkan sejumlah object dan pesan yang dijalankan antara object dengan usecase.

Ketika kita memberikan pesan, aksi yang dihasilkan adalah sebuah pernyataan tereksekusi yang membentuk abstraksi dari prosedur komputasi. Sebuah aksi mungkin menghasilkan perubahan kondisi.

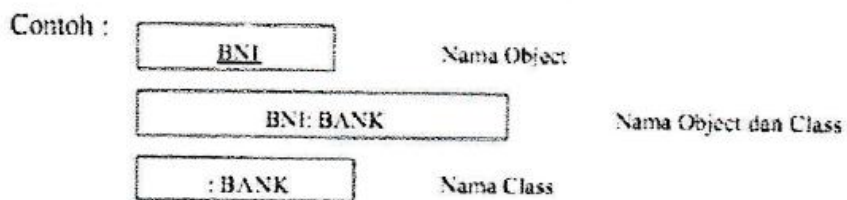
Dalam UML, kita dapat memodelkan beberapa jenis aksi, yaitu:

- Call: memanggil operasi yang ada pada object , object mungkin mengirim kedirinya sendiri, menghasilkan pemanggilan lokal dari operasi.
- Return: mengembalikan nilai dari caller.
- Send : Mengirimkan sinyal ke object.
- Create: membuat sebuah object.
- Destroy: mematikan sebuah object, object mungkin saja mematikan dirinya sendiri.

Ada dua macam Interaction Diagram yaitu : Sequence Diagram dan Collaboration Diagram.

2.2.2 Sequence Diagram

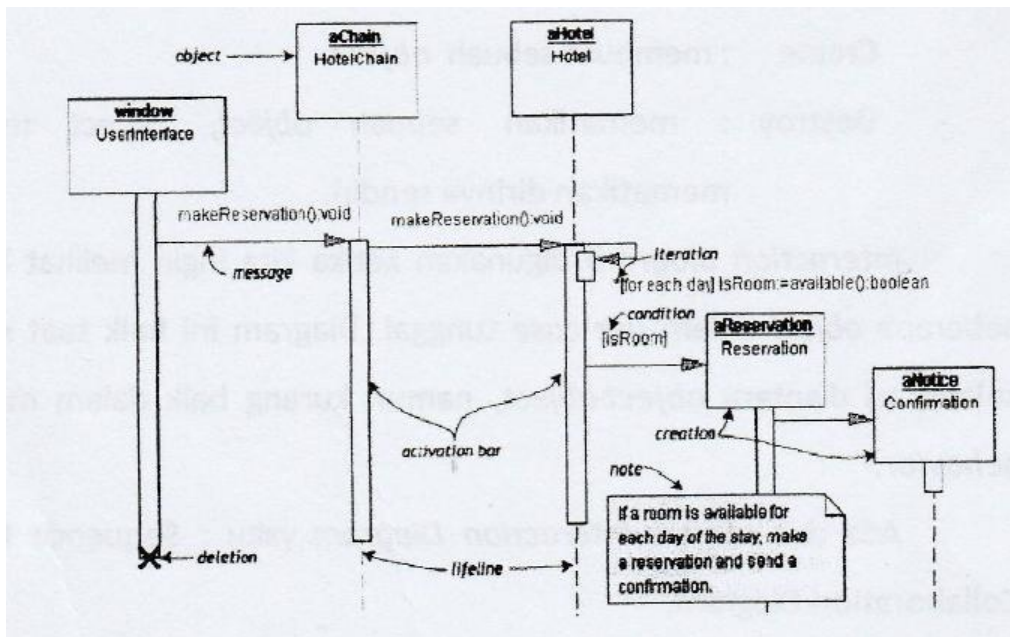
Sequence Diagram menggambarkan interaksi antara sejumlah object dalam urutan waktu. Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara object juga interaksi antar object yang terjadi pada titik tertentu dalam eksekusi sistem. Dalam UML, object pada diagram sequence digambarkan dengan segi empat yang berisi nama dari object yang digarisbawahi. Pada object terdapat 3 cara untuk menamainya yaitu : nama object, nama object dan class serta nama class.



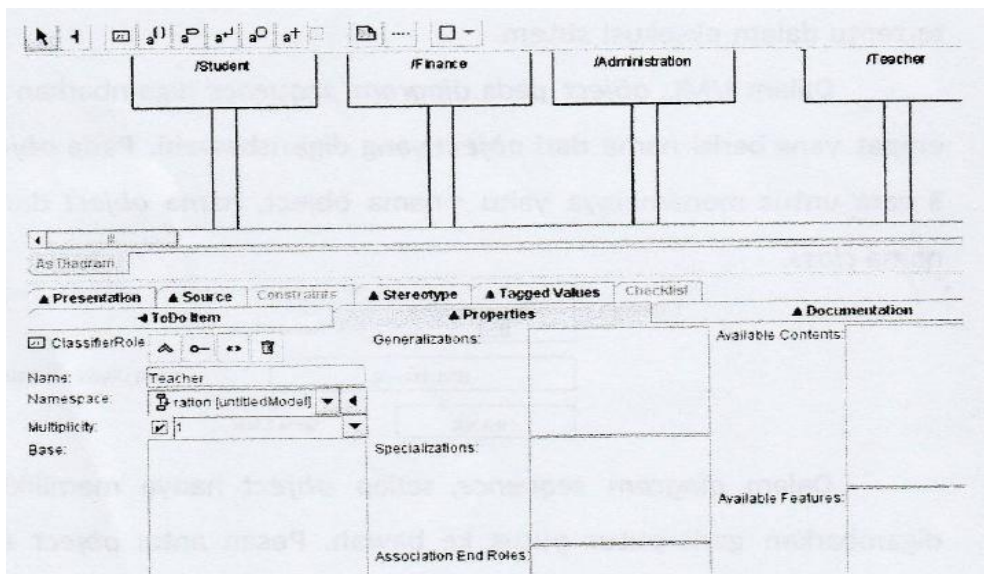
Dalam diagram sequence, setiap object hanya memiliki garis yang digambarkan garis putus putus ke bawah. Pesan antar object digambarkan dengan anak panah dari object yang mengirimkan pesan ke object yang menerima pesan.

Cara membuat sequence diagram di ArgoUML:

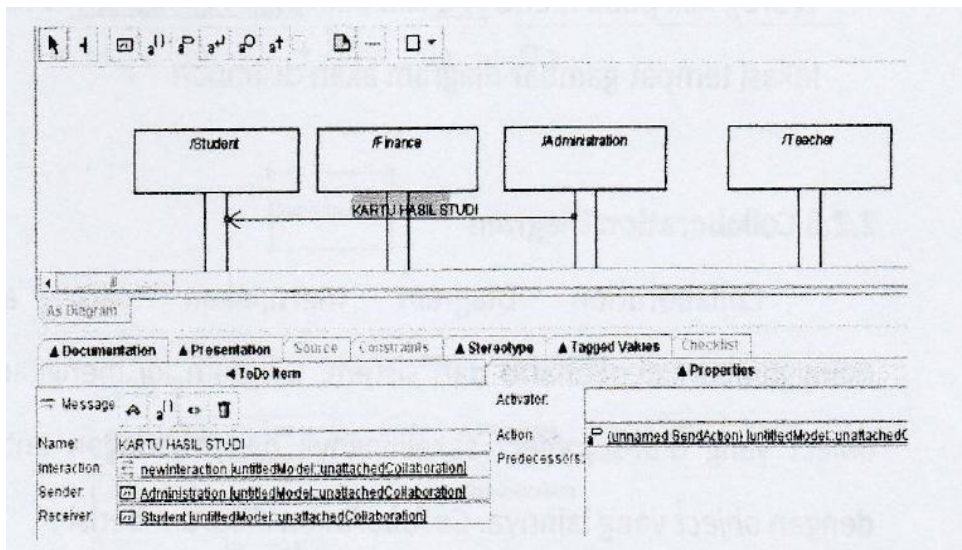
1) Contoh:



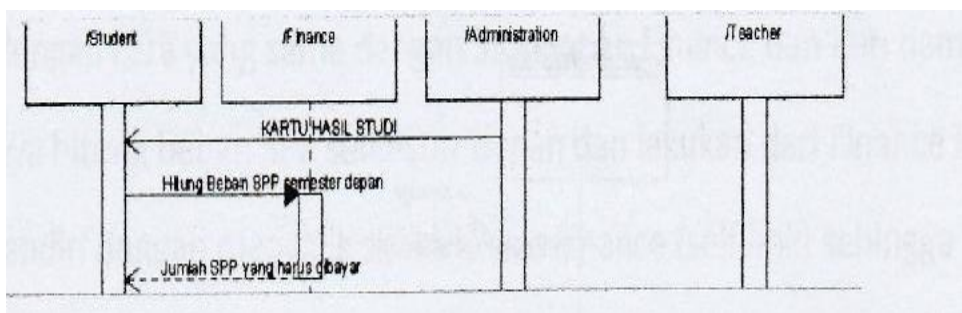
2) Jalankan menu create > New Sequence Diagram dan drag icon New classifier Role sebanyak 4 buah ke Editing Pane dan beri name dengan Student, Finance, Administration dan Teacher sehingga muncul tampilan sbb:



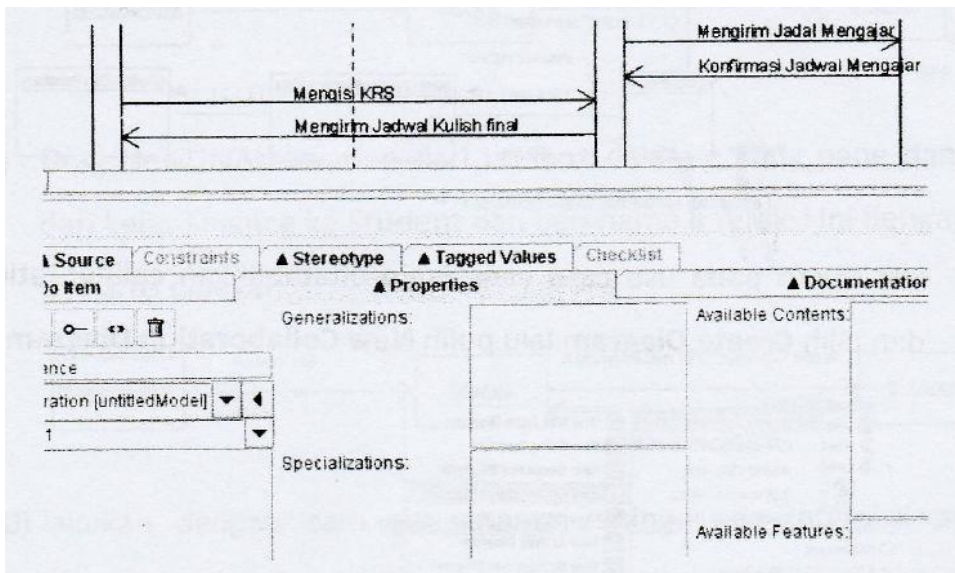
- 3) Drag New Send Action dari Administration ke Student, dan beri name dengan Kartu Hasil Studi.



- 4) Misal Student mau menghitung pembayaran SPP semester depan maka drag New Call Action (semacam menjalankan menu hitung SPP) dari Student ke Finance dan beri name Hitung beban SPP semester depan. Return Value dapat diberi name engan SPP yang harus dibayar seperti tampilan sbb:



- 5) Lengkapi sehingga anda menambahkan 4 garis New Send Action lagi dengan name seperti di gambar sbb :



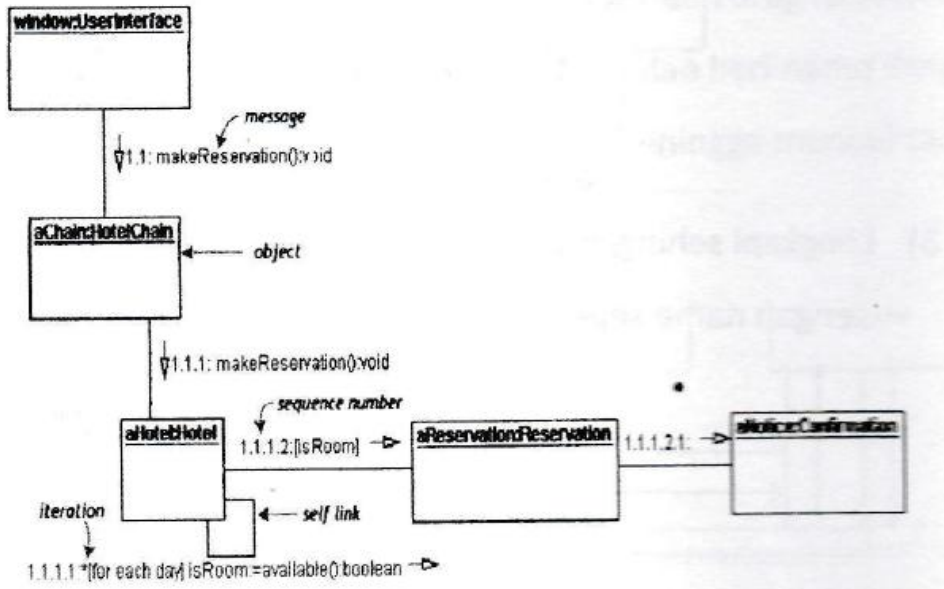
- 6) Untuk merubah atau menyalin diagram ke file dokumen lain (misal MS. Word), klik pada menu File lalu pilih Export Diagram setelah itu tentukan lokasi tempat gambar diagram akan disimpan.

2.2.3 Collaboration Diagram

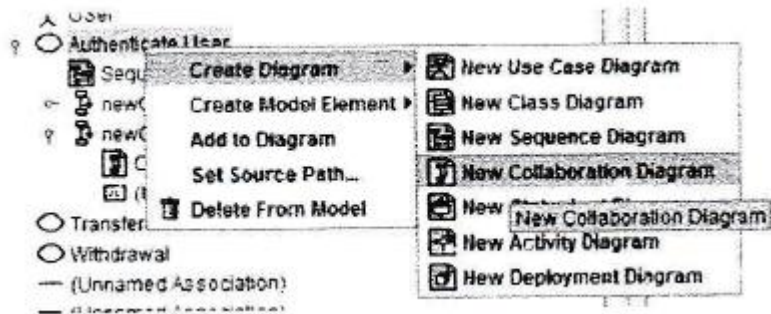
Collaboration Diagram merupakan cara alternatif untuk menggambarkan skenario dari sistem. Diagram ini menggambarkan interaksi object yang diatur object sekelilingnya dan hubungan antara setiap object dengan object yang lainnya. Collaboration diagram berisi :

- object yang digambarkan dengan segiempat.
- hubungan antara object yang digambarkan dengan garis penghubung.
- pesan yang digambarkan dengan teks dan panah dari object yang mengirim pesan ke penerima pesan.

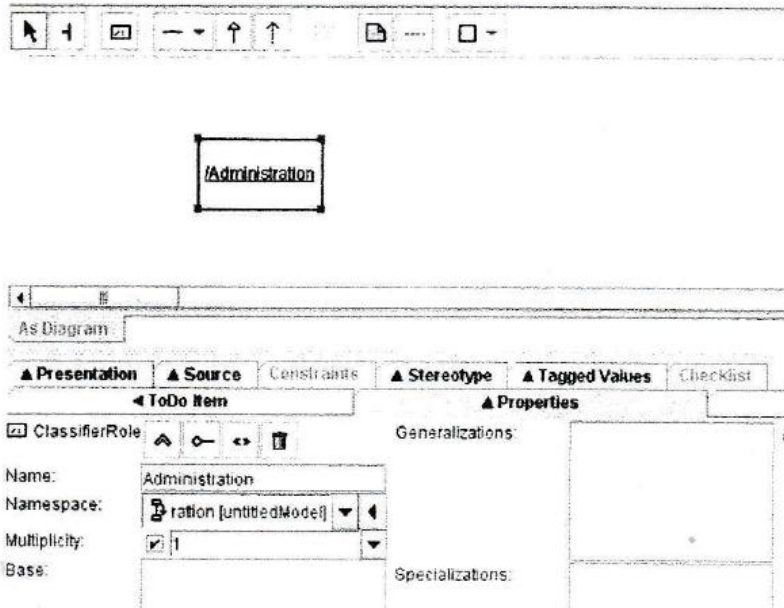
Untuk membuat collaboration diagram, contoh:



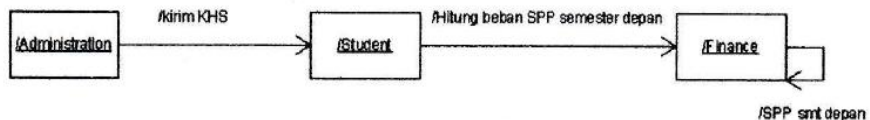
- 1) Klik kanan pada usecase yang ingin ditambahkan collaboration diagram dan pilih Create Diagram lalu pilih New collaboration Diagram.



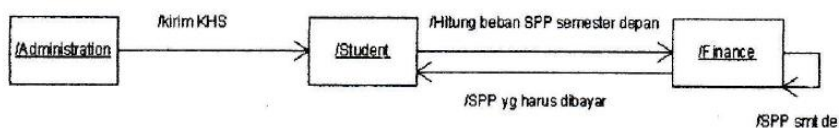
- 2) Maka akan muncul jendela untuk membuat collaboration diagram, kemudian dari jendela Editing Pane, drag Ico New Classifier Role sebanyak satu buah, beri nama Administrator.



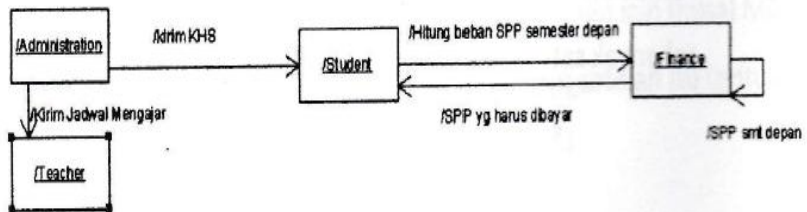
- 3) Letakkan kursor pada sisi kanan kelas Administrator dan klik sehingga terbentuk panah interaksi dengan kelas lain. Beri nama kelas ini dengan name Student dan beri nama interaksinya dengan name kirim KHS.
- 4) Lakukan dengan cara yang sama dengan Student ke Finance dan beri nama interaksinya hitung beban SPP semester depan dan lakukan dari Finance ke Finance sendiri dengan mengklik sisi kiri kelas Finance (self link) sehingga terbentuk relasi sbb :



- 5) Drag NewUniAssociation dari toolbox dalam editing pane dan tempelkan dari kelas Finance ke Student dan beri nama interaksi ini dengan nama spp yg harus dibayar.



- 6) Lakukan dengan cara yang sama sehingga ada interaksi antara kelas Administration dengan Teacher dengan name Kirim Jadwal Mengajar.



TUGAS

- 1) Buat activity diagram dan collaboration diagram untuk kasus ATM.
- 2) Beri kesimpulan dan penjelasan mengenai pemahaman anda tentang collaboration diagram dan activity diagram.

MODUL III

CLASS DIAGRAM

TUJUAN

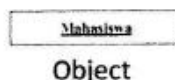
- Praktikan dapat menggambarkan *class diagram* dari *candidate class* yang telah ada.
- Praktikan bisa memberikan atribut dan operasi pada masing-masing *class* yang didefinisikan.
- Praktikan dapat menggambarkan relasi antar *class* dan tipe -tipenya.

Pada modul sebelumnya kita sudah mengupas sedikit tentang *object* dan *candidate class*, dan pada modul ini kita akan membahas lebih dalam tentang bagaimana suatu *class* dapat dibentuk, hubungan anatar *class* beberapa *class* turunan.

TEORI

3.1. Status(State), Behaviour dan Identify

- **Status** dari *object* adalah satu kondisi yang mungkin ada. Status dari *object* akan berubah setiap waktu dan ditentukan oleh sejumlah *property* (atribut) dengan nilai dari properti, ditambah relasi *object* dengan *object* lainnya.
- **Sifat (Behaviour)** menentukan bagaimana *object* merespon permintaan dari *object* lain dan melambangkan setiap *object* yang dapat dilakukan. Sifat ini diimplementasikan dengan sejumlah operasi untuk *object*.
- **Identitas (Identify)** artinya setiap *object* yang unik Pada UML, *object* digambarkan dengan segiempat dan nama dari object diberi garis bawah.



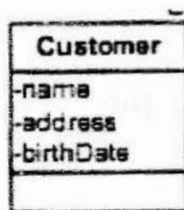
Mendefinisikan class

Seperti telah dijelaskan sebelumnya, stereotypes memberikan kemampuan untuk membuat pemodelan yang baru. Beberapa stereotype untuk class adalah entity, boundry, control, utility dan exception.

3.2. Atribut dan Operasi pada Class

3.2.1. Atribut

Atribut adalah salah satu property yang dimiliki oleh class yang menggambarkan batasan dari nilai yang dapat dimiliki oleh property tersebut. sebuah class mungkin memiliki beberapa atribut atau tidak memilikinya sama sekali. sebuah atribut merepresentasikan beberapa property dari sesuatu yang kita modelkan, yang dibagi dengan semua object dari semua class yang ada. contohnya, setiap tembok memiliki tinggi, lebar dan ketebalan. Atribut dalam implementasinya akan digambarkan sebagai sebuah daftar (list) yang diletakkan pada kotak dibawah nama class. Ia seperti halnya nama class merupakan teks. Biasanya huruf pertama dari tiap kata merupakan huruf kapital, terkecuali untuk huruf awal. Sebagai contohnya : birthDate.



Contoh atribut dari class

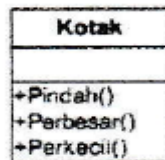
Untuk lebih lanjut kita pun bisa menspesifikasikan atribut beserta jenis data yang kita gunakan untuk atribut tersebut.



Contoh lain dari attribute

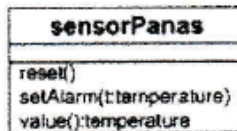
3.2.2. Operasi

Sebuah operasi adalah sebuah implementasi dari layanan yang dapat diminta dari beberapa *object* dari *class* , yang mempengaruhi *behaviour*. Dengan kata lain operasi adalah abstraksi dari segala sesuatu yang dapat kita lakukan pada sebuah *object* dan ia berlaku untuk semua *object* yang terdapat dalam *class* tersebut. *Class* mungkin memiliki beberapa operasi atau tanpa operasi sama sekali. contohnya adalah sebuah *class* "kotak" dapat dipindahkan, diperbesar atau diperkecil. Biasanya (namun tidak selalu), memanggil operasi pada sebuah *object* akan mengubah data atau kondisi dari *object* tersebut. Operasi ini dalam implementasinya digambarkan dibawah atribut dari sebuah *class*.



contoh dari operasi

Untuk lebih lanjut kitapun bisa menspesifikasikan semua parameter yang terlibat dalam operasi tersebut.



contoh lain dari operasi

3.2.3 Pengorganisasian atribut dan operasi.

Ketika menggambarkan sebuah *class* kita tidak perlu menampilkan seluruh atribut atau operasi. Karena dalam sebagian besar kasus kita tidak dapat menampilkannya dalam sebuah gambar, karena terlalu banyaknya atribut atau operasinya bahkan terkadang tidak perlu karena kurang relevannya atribut atau operasi tersebut untuk ditampilkan. Sehingga kita dapat menampilkan hanya sebagian atau bahkan tidak sama sekali atribut dan operasinya.

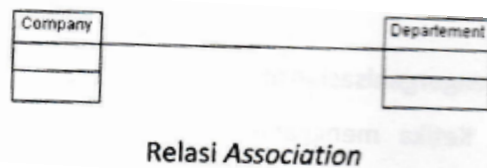
Kosongnya kotak tempat pengisian bukan berarti tidak ada. Karena itu kita dapat menambahkan tanda ("...") pada akhir daftar yang menunjukkan bahwa masih ada atribut atau operasi yang lain.

3.3 Relasi dalam Obiek

Semua sistem terdiri dari class. Class dan object. Kelakuan sistem dicapai melalui kerjasama antar object, contohnya seorang mahasiswa ditambahkan dalam daftar class, jika daftar class memperoleh message untuk menambahkan mahasiswa. Interaksi antar object disebut object relationship. Dua tipe relationship yang ditemukan pada saat analisis adalah association dan aggregation.

3.3. 1 Association Relationships

Association adalah hubungan semantik bidirectional diantara class-class. Ini bukan aliran data sebagaimana pada permodelan desain dan analisa terstruktur, data diperbolehkan mengalir dari kedua arah. Asosiasi diantara class-class artinya ada hubungan antara object-object pada class-class yang berhubungan. Banyaknya object yang terhubung tergantung dengan multiplicity pada asosiasi, yang akan dibahas nanti.



3.3.2 Aggregation relationships

Aggregation relationships adalah bentuk khusus dari asosiasi dimana induk terhubung dengan bagian-bagiannya. Notasi UML untuk relasi agregasi adalah sebuah asosiasi dengan diamond putih melekat pada class yang menyatakan induk. Contoh, Course Tugas akhir terdiri atas CourseOffering Tugas Akhir 1 dan courseoffering Tugas akhir 2.



Relasi Agregasi

Pertanyaan-pertanyaan di bawah dapat digunakan untuk menentukan apakah asosiasi seharusnya menjadi agregasi:

- 1) Apakah klausa *has-a* ("bagian dari") digunakan untuk menggambarkan relasi?
- 2) Apakah beberapa operasi di induk secara otomatis dapat dipakai pada bagian-bagiannya? Sebagai contoh, *delete* sebuah *course*, maka akan *men-delete course offering*-nya.

3.3.3 Indikator Multiplicity

Walaupun *multiplicity* ditentukan untuk *class*, *multiplicity* menentukan banyaknya *object* yang terlibat dalam relasi. *Multiplicity* menentukan banyaknya *object* yang terhubung satu dengan yang lainnya. Indikator *multiplicity* terdapat pada masing-masing akhir garis relasi, baik pada asosiasi maupun agregasi. Beberapa contoh *multiplicity* adalah :

1	Tepat satu
0..*	Nol atau lebih
1..*	Satu atau lebih
0..1	Nol atau satu
5..8	range 5 s.d. 8
4..6,9	range 4 s.d. 6 dan 9

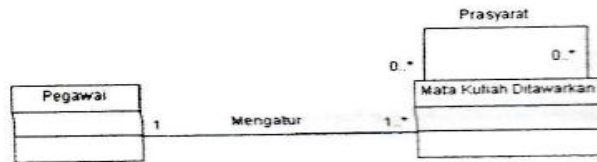


Multiplicity

3.3.4 Reflexive Relationships

Multiple object pada *class* yang sama dapat saling berkomunikasi satu dengan yang lainnya. Hal ini ditunjukkan pada *class diagram* sebagai

reflexive association atau aggregation . Penamaan role lebih disukai untuk digunakan pada reflexive relationships dari pada penampilan association relationship.



Reflexive relationship

- **Menemukan Relationships**

Untuk menemukan relationships class-class yang ada dapat dilakukan dengan memeriksa skenario dan pertukaran message diantara class-class yang ada. Pada tahap analisa, dua relationship yang ditemukan adalah asosiasi dan agregasi. Dikarenakan metode yang digunakan merupakan iterative maka relationship akan berubah seiring dengan fase analisis dan desain.

- **Menambahkan Behavior dan Struktur (Atribut)**

Perhatian: salah satu metode untuk mengetahui behavior pada class adalah dengan memetakan message pada interaction diagram (Modul 2) menjadi operasi pada class tujuan! Baca juga membuat class pada pembahasan sebelumnya pada modul ini (Modul 2).

Sebuah class mempunyai sekumpulan kewajiban yang menentukan kelakuan object-object dalam class. Kewajiban ini diwujudkan dalam operasi-operasi yang didefinisikan untuk class tersebut. Struktur dari suatu class didefinisikan oleh atribut-atribut class tersebut. Setiap atribut adalah definisi data yang pada object dalam class nya. Object yang didefinisikan dalam class mempunyai sebuah nilai untuk setiap atribut dalam class.

Message dalam interaction diagram {Modul 2}, pada umumnya dipetakan menjadi operasi pada class tujuan. Namun ada beberapa kasus dimana message tidak menjadi operasi, antara lain message dari atau menuju actor yang merepresentasikan orang/individu dan message menuju boundary class yang

merepresentasikan class GUI. Namun jika actor merepresentasikan external entity maka message dari atau menuju actor dapat menjadi operasi pada class.

Operasi dapat juga dibuat tanpa tergantung (*independen*) dari *interaction diagram* (Modul 2), karena tidak semua skenario direpresentasikan dalam diagram. Hal yang sama juga berlaku untuk operasi yang dibuat dengan tujuan untuk membantu operasi lain.

Kebanyakan dari atribut dari sebuah *class* ditemukan pada definisi masalah, kebutuhan perangkat lunak dan aliran dokumentasi kejadian. Atribut juga dapat ditemukan ketika mendefinisikan sebuah *class*. Sebuah *relationship* mungkin juga dapat memiliki struktur dan *behavior*, hal ini terjadi jika informasi berhubungan dengan sebuah *link* di antara dua *object* dan bukan dengan salah satu *object* diantaranya. Struktur dan *behavior* dalam sebuah *relationship* disimpan dalam *class association*.

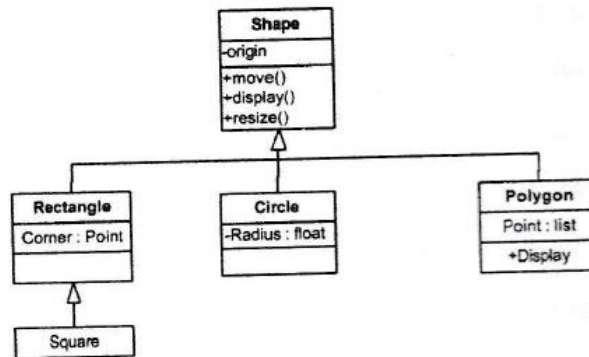
3.4 Inheritance

Inheritance merupakan kemampuan untuk membuat hierarki yang terdiri atas *class-class*, dimana terdapat struktur dan atau *behavior* (kelakuan) diantara *class-class*. Istilah *superclass* digunakan oleh *class* yang menyimpan informasi umum. Keturunan dari *superclass* disebut *subclass*.

Sebuah *subclass* mewarisi semua atribut, operasi dan *relationship* yang dipunyai oleh semua *superclass-superclassnya*. *Inheritance* disebut juga hierarki *is-a* (adalah sebuah) atau *kind-of* (sejenis). *Subclass* dapat menggunakan atribut dan operasi tambahan yang hanya berlaku pada level hierarkinya. Karena *inheritance relationship* bukan sebuah *relationship* diantara *object* yang berbeda, maka *relationship* ini tidak pernah diberi nama, penamaan role juga tidak digunakan dan *multiplicity* tidak digunakan. Terdapat dua cara untuk menemukan *inheritance*, yaitu *generalization* dan *specialization*

3.4.1 Generalization

Generalization menjamin kemampuan untuk membuat superclass yang melingkupi struktur dan behaviour yang umum pada beberapa class dibawahnya (subctass).



Generalization

3.4.2 Specialization

Specialization menjamin kemampuan untuk membuat subclass yang berfungsi untuk menambah atribut dan operasi superclass. Operasi pada superclass dapat di-override (konsep polymorphism). Tetapi, sebuah subclass seharusnya tidak boleh membatasi sebuah operasi yang didefinisikan dalam superclass-nya, dengan kata lain subclass seharusnya tidak boleh menyediakan lebih sedikit behaviour atau struktur daripada superclas-nya.

3.5 Package

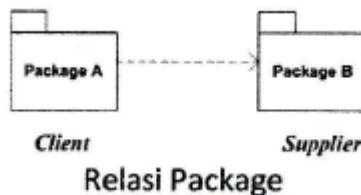
Jika sistem hanya memiliki sedikit class, kita dapat mengaturnya dengan mudah. sebagian besar sistem dibuat dari banyak class sehingga kita memerlukan suatu mekanisme untuk mengelompokkannya bersama untuk memudahkan dalam hal penggunaan, perawatan dan penggunaan kembali. Package adalah kumpulan dari package atau class yang berelasi. Dengan mengelompokkan class dalam package, kita bisa melihat level yang lebih tinggi dari model kita atau kita bisa menggali model dengan lebih dalam dengan melihat apa yang ada di dalam package.

Jika sistemnya kompleks, package mungkin dibuat di awal fase elaborasi sebagai fasilitator komunikasi. Untuk sistem yang lebih sederhana, class-class yang didapat pada tahap analisa mungkin dikelompokkan dalam suatu package. Di UML. Package digambarkan sebagai folder.



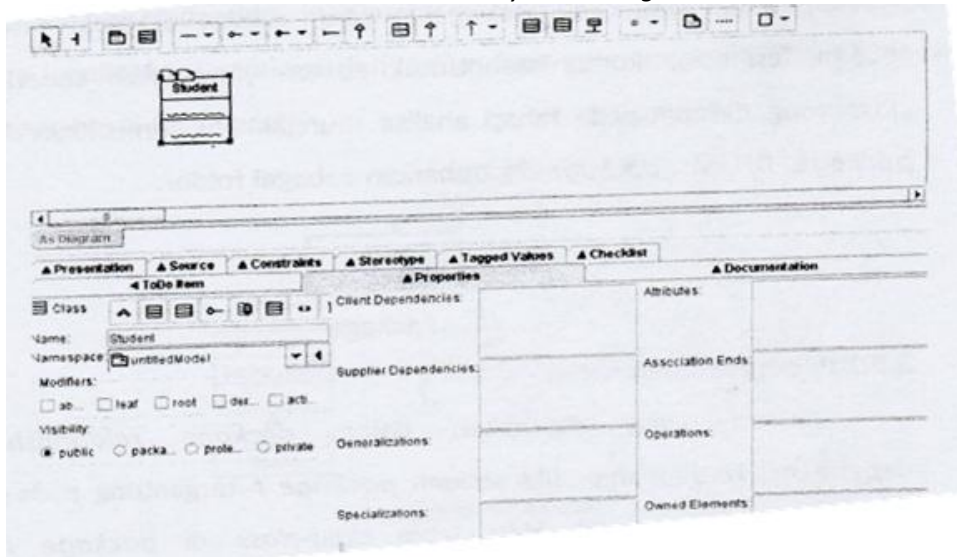
3.5.1 Package Relationship

Relasi yang digunakan dalam package relationship adalah dependency relationship. Jika sebuah package A tergantung pada package B, hal ini berakibat satu atau lebih class-class di package A memulai berkomunikasi dengan satu atau lebih public class di package B. Package A disebut client package dan package B disebut supplier package.

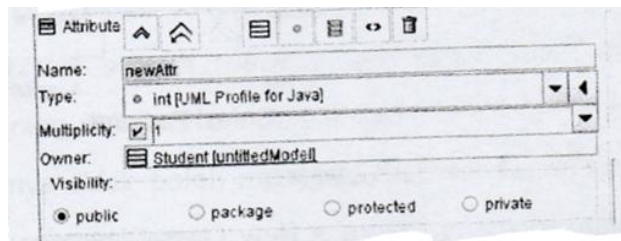


3.6 Cara membuat Class Diagram

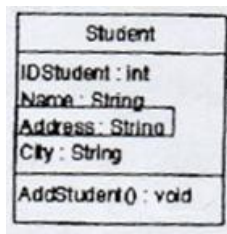
- 1) Jalankan menu Create > New Class Diagram, drag icon Class ke editing pane dan berilah name pada Details pane dengan student sehingga terbentuk gambar seperti di bawah ini :



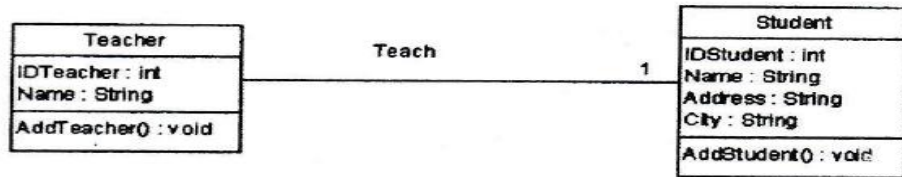
- 2) Klik kotak atribut dibawah nama kelas pada Editing pane dan isikan atribut IDStudent dengan tipe int, name dengan tipe String, Address dengan tipe String dan City bertipe String pula.



- 3) Klik kotak operation dibawah atribut kelas student di editing pane dan berilah name AddStudent dengan tipe return value void.



- 4) Buat kelas Teacher dengan langkah yang sama dengan kelas student dan lengkapi dengan atribut IDTeacher int, Name string dan tambahkan Operation AddTeacher. Drag icon association dan drag dari kelas Teacher ke kelas Student dan berilah name Teach.



- 5) Simpanlah dulu proyek anda dengan file > Save Project As > beri nama bebas.
- 6) Untuk menghasilkan coding kelas, jalankan menu generate > Generate All Class pada direktori jika diperlukan dan klik pilih hasil codingnya misal java dan muncul tampilan sbb:



Hasil codenya adalah sbb :

```
public class Student {
    public int IDStudent;
    public String Name;
    public String Address;
    public String City;
    public Teacher Teach;
    public void AddStudent() {
    }
}
```

TUGAS

- 1) Bagaimana membedakan antara *actor* sebagai orang/individu dengan *external entity*.
- 2) Perbaiki candidate Class diagram system ATM lengkapi diagram dengan attribute dan operasinya, jika ada generalisasi terapkan pada diagram tersebut.

MODUL IV

STATE TRANSITION DIAGRAM DAN ACTIVITY DIAGRAM

TUJUAN

- Praktikan dapat menentukan *object-object* dinamis dari suatu *class* dan menggambarkan *state diagram* dari *object-object* tersebut.
- Praktikan dapat menggambarkan *activity diagram* dan membedakannya dengan *state diagram*.

TEORI

4.1 State Transition Diagram

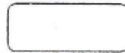
Use case dan skenario menyediakan cara untuk menggambarkan kelakuan sistem yakni interaksi antara *object-object* di dalam sistem. Kadang-kadang diperlukan untuk melihat kelakuan di dalam *object*. *State transition diagram* menunjukkan *state-state* dari *object* tunggal, *event-event* atau pesan yang menyebabkan transisi dari satu *state* ke *state* yang lain, dan *action* yang merupakan hasil dari perubahan sebuah *state*.

State transition diagram tidak akan dibuat untuk setiap *class* di sistem. *State transition diagram* hanya dibuat untuk **class yang berkelakuan dinamis**. *Interaction diagram* dapat dipelajari untuk menentukan *dynamic object* di sistem, yaitu *object* yang menerima dan mengirim beberapa pesan. *State transition diagram* juga sangat berguna untuk meneliti kelakuan dari sebuah kumpulan *whole class* dan *control class*.

4.1.1 States

State adalah sebuah kondisi selama kehidupan sebuah *object* ketika *object* memenuhi beberapa kondisi, melakukan beberapa *action*, atau menunggu sebuah *event*. *State* dari sebuah *object* dapat dikarakteristikan oleh nilai dari satu atau lebih atribut-atribut dari *class*.

State-state dari sebuah object ditemukan dengan pengujian/pemeriksaan atribu-atribut dan hubungan-hubungan dari object, Notasi UML untuk state adalah empat persegi panjang/bujur sangkar dengan ujung yang dibulatkan, seperti ditunjukkan pada gambar.



Notasi UML untuk State

State transition diagram meliputi seluruh pesan dari object yang dapat mengirim dan menerima. Skenario merepresentasikan satu jalur yang melewati sebuah state transition diagram. Jarak waktu antara dua pesan yang dikirim oleh sebuah object merepresentasikan sebuah state . Oleh karena itu, sequence diagram ditentukan untuk menemukan state-state sebuah object (lihat pada ruang antara garis-garis yang merepresentasikan pesan-pesan diterima oleh object).

4. 1.2 State Transitions

State transition merepresentasikan sebuah perubahan dari state awal ke sebuah state berikutnya (yang mungkin dapat sama dengan state awal). Sebuah action dapat menyertai sebuah state transition.

Ada dua cara untuk membuat transisi sebuah state - otomatis dan tidak otomatis. State transition yang otomatis terjadi ketika activity dari state awal telah lengkap - tidak ada event yang terasosiasi dengan state transition yang belum bernama. State transition yang tidak otomatis disebabkan oleh sebuah event bernama (salah satu dari object atau dari luar sistem). Kedua tipe dari state transition dipertimbangkan untuk membuat waktu nol dan tidak dapat diinterupsi. Sebuah state transition direpresentasikan oleh sebuah panah yang menunjuk dari state awal ke state berikutnya.

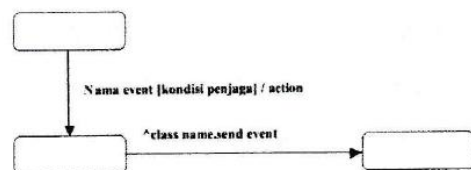
4.1.3 Special States

Ada dua state khusus yang ditambahkan di state transition diagram. Pertama adalah start state. Masing-masing diagram harus mempunyai satu dan hanya satu start state ketika object mulai dibuat. Notasi UML untuk start state ditunjukkan gambar dibawah khusus berikutnya adalah stop state. Sebuah object boleh mempunyai banyak stop state. Notasi UML untuk stop state ditunjukkan gambar.



4.1.4 State Transition Details

Sebuah state transition dapat mempunyai sebuah action dan/atau sebuah kondisi penjaga (guard condition) yang terasosiasi dengannya, dan mungkin juga memunculkan sebuah event. Sebuah action adalah kelakuan yang terjadi ketika state transition terjadi. Sebuah event adalah pesan yang dikirim ke object lain di sistem. Kondisi penjaga adalah ekspresi boolean dari nilai atribut-atribut yang mengijinkan sebuah state transition hanya jika kondisinya benar. Kedua action dan penjaga adalah kelakuan dari object dan secara tipikal menjadi operasi. Seringkali operasi-operasi ini adalah tersendiri hal itu, mereka digunakan hanya oleh object dirinya sendiri. Notasi UML untuk state transition details ditunjukkan gambar dibawah ini.

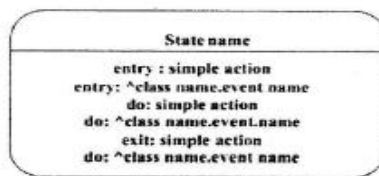


Notasi UML untuk state transition detail

4.1.5 State Details

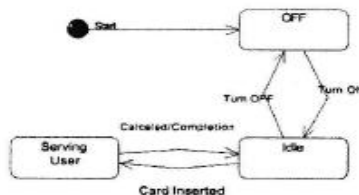
Action-action yang mengiringi seluruh state transition ke sebuah state mungkin ditempatkan sebagai sebuah entry action dalam state. Demikian juga, action-action yang mengiringi seluruh state transition keluar dari sebuah state mungkin ditempatkan sebagai sebuah aksi keluar dalam state. Kelakuan yang terjadi dalam state disebut activity. Sebuah activity dimulai ketika state dimasukkan dan salah satu dari melengkapi atau diinterupsi oleh sebuah state transition yang keluar.

Kelakuan mungkin sebuah action yang sederhana, atau kelakuan merupakan sebuah event yang terkirim ke object lain. sesuai dengan action-action dan guard -guard, kelakuan ini secara tipikal dipetakan ke operasi-operasi dalam object. Notasi UML untuk state detailed information ditunjukkan gambar dibawah.



State Details

Contoh state diagram untuk kasus ATM



State diagram ATM

Untuk membuat State diagram klik pada menu **Create** -> lalu pilih

New StateChart Diagram atau dengan meng klik icon 

4.2 Activity Diagram


Activity diagram memodelkan workflow proses bisnis dan urutan aktivitas dalam sebuah proses. Diagram ini sangat mirip dengan flowchart karena memodelkan workflow dari satu aktivitas ke aktivitas lainnya atau dari aktivitas ke status. Menguntungkan untuk membuat activity diagram pada awal pemodelan proses untuk membantu memahami keseluruhan proses.

Activity diagram juga bermanfaat untuk menggambarkan parallel behaviour atau menggambarkan interaksi antara beberapa use case.

Elemen-elemen activity diagram :

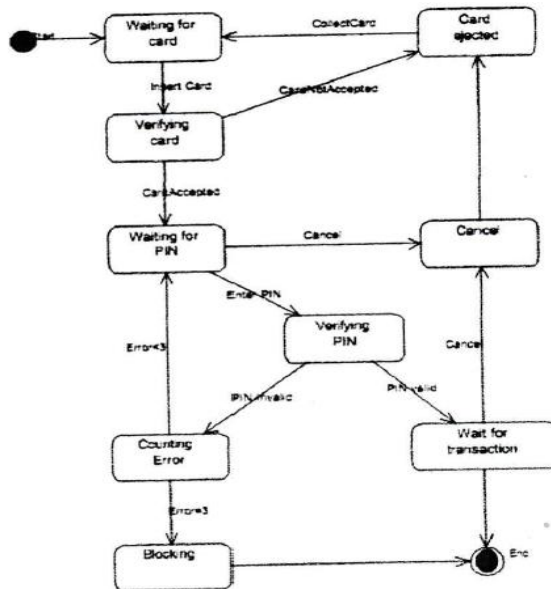
- 1) Status start (mulai) dan end (akhir).
- 2) Aktivitas yang merepresentasikan sebuah langkah dalam workflow.
- 3) Transition menunjukkan terjadinya perubahan status aktivitas (Transitions show what state follows another)
- 4) Keputusan yang menunjukkan alternatif dalam workflow.
- 5) Synchronization bars yang menunjukkan subflow parallel. Synchronization bars dapat digunakan untuk menunjukkan concurrent threads pada workflow proses bisnis.
- 6) Swimlanes yang merepresentasikan role bisnis yang bertanggung jawab pada aktivitas yang berjalan.

Untuk membuat activity diagram klik pada menu Create -> lalu pilih

New Activity Diagram atau dengan meng klik icon 

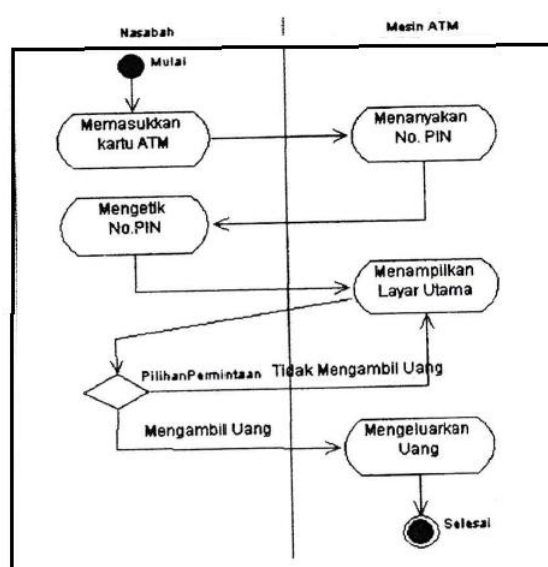
TUGAS

- 1) Buat State diagram untuk diagram untuk tiap Class pada kasus ATM, contoh 1



StateCard Diagram for class ATMCARD

- 2) Buat Activity diagram lengkap dengan swimline untuk system ATM
Berikut contoh activity diagram untuk pengambilan uang, anda boleh membagi system ATM perbagian transaksi atau digabung dalam satu transaksi.



- 3) Jelaskan perbedaan antara state diagram dan activity diagram

COMPONEN DIAGRAM

TUJUAN

- 1) Mahasiswa dapat lebih mengetahui dan memahami pengertian dari Component Diagram.
- 2) Mahasiswa dapat lebih mengetahui dan memahami penggunaan dari Component Diagram
- 3)

TEORI

Component Diagram

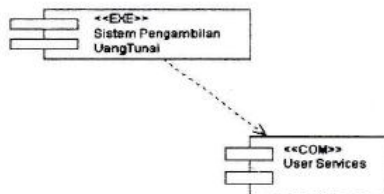
Component Diagram menggambarkan alokasi semua kelas dan objek ke dalam komponen-komponen dalam desain fisik system software. Diagram ini memperlihatkan pengaturan dan kebergantungan antara komponen-komponen software, seperti source code, binary code, dan komponen tereksekusi (executable components). Sebuah komponen berisi informasi tentang logic class atau class yang diimplementasikan sehingga membuat pemetaan dari logical view ke component view.

Component diagram dapat dibuat satu atau lebih untuk menggambarkan komponen dan paket, atau menerangkan isi dari tiap-tiap paket komponen.

Untuk membuat component diagram di ArgoUML, klik pada menu **Create ->** lalu pilih **New Deployment Diagram** atau dengan meng klik icon

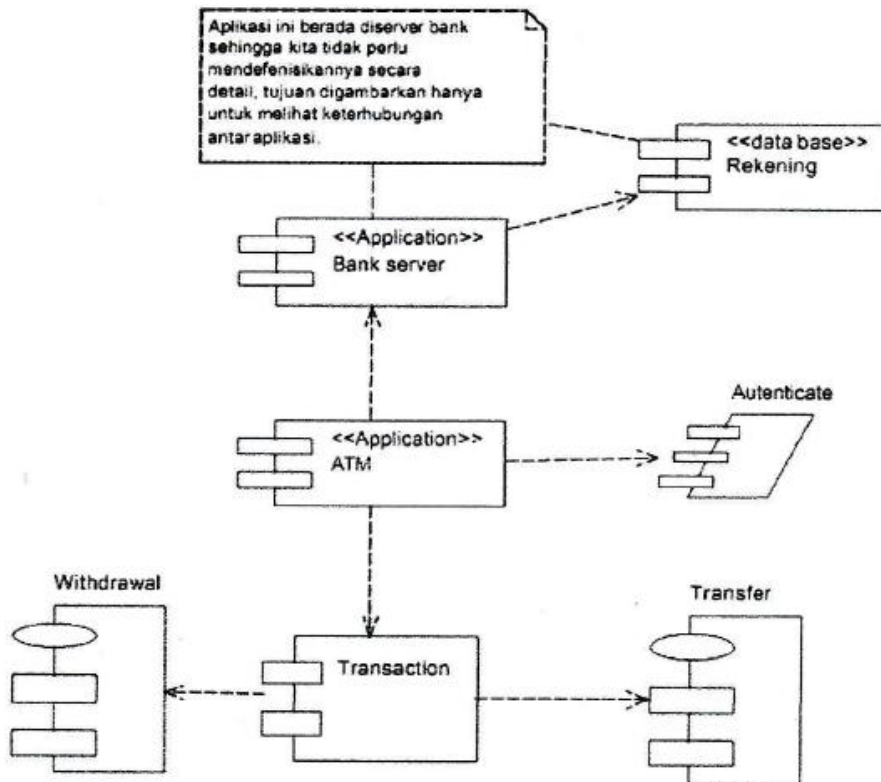


, contoh component diagram



TUGAS

- 1) Sebutkan dan jelaskan fungsi dari component diagram pada UML
- 2) Buat component diagram untuk kasus ATM seperti gambar dibawah ini



- 3) Buat 1 contoh komponen diagram dan berikan deskripsi dari component diagram tersebut

MODUL VI

DEPLOYMENT DIAGRAM

TUJUAN


- 1) Mahasiswa dapat lebih mengetahui dan memahami pengertian dari Deployment Diagram.
- 2) Mahasiswa dapat lebih mengetahui dan memahami penggunaan dari Deployment Diagram.

TEORI

Deployment Diagram memperlihatkan pemetaan software kepada hardware. setiap model hanya memiliki satu deployment diagram.

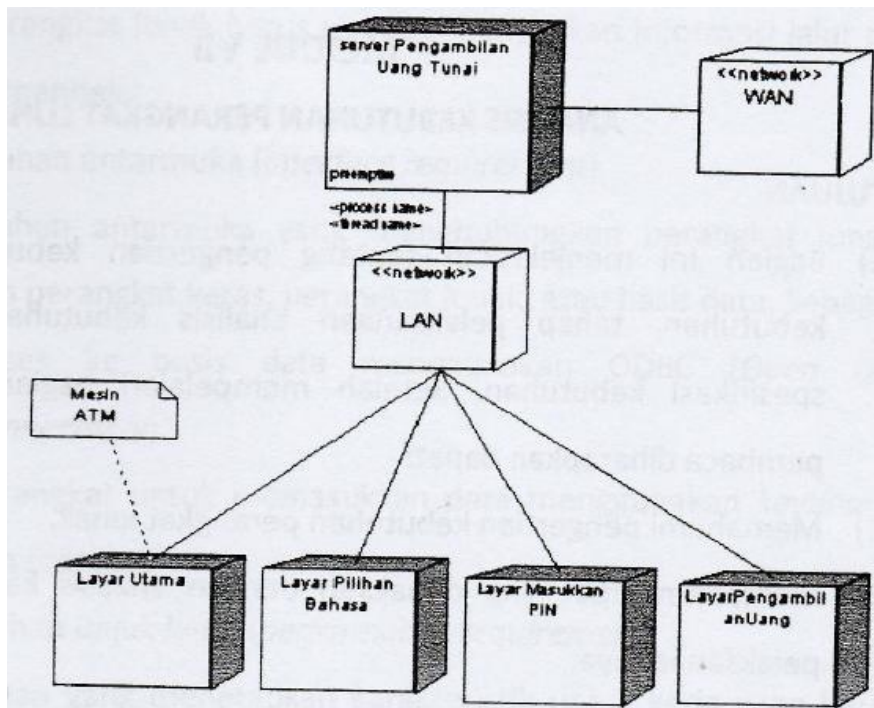
Deployment diagram menggambarkan arsitektur fisik dari perangkat keras dan perangkat lunak sistem, menunjukkan hubungan komputer dengan perangkat (nodes) satu sama lain dan jenis hubungannya. Di dalam nodes, executable component dan object yang dialokasikan untuk memperlihatkan unit perangkat lunak yang dieksekusi oleh node tertentu dan ketergantungan komponen.

Cara membuat deployment diagram adalah sebagai berikut :

Klik pada menu create -> lalu pilih New Deployment Diagram atau dengan mengklik , contoh Deployment diagram.

TUGAS

- 1) Buat Deployment diagram untuk kasus ATM seperti berikut:



- 2) Buat contoh lain minimal 2 buah, dan beri deskripsinya.