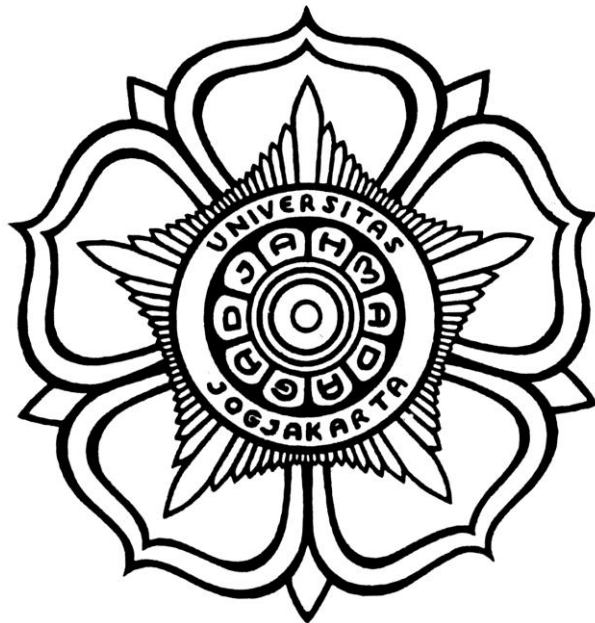


BUKU PETUNJUK (MODUL) PRAKTIKUM

**PENGEMBANGAN PERANGKAT LUNAK
(MII3502)**



**LABORATORIUM KOMPUTER DASAR
JURUSAN ILMU KOMPUTER DAN ELEKTRONIKA
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Gadjah Mada
Yogyakarta
2017**

TATA TERTIB PRAKTIKAN

1. Masuk sesuai jadwal/waktu yang telah ditentukan/disepakati bersama, terlambat lebih dari 15 menit dipersilahkan tidak memasuki ruang LABORATORIUM KOMPUTER DASAR.
2. Praktikan yang tidak mengikuti praktikum lebih dari 3 kali tanpa keterangan resmi, tidak berhak mengikuti responsi/ujian.
3. Tidak diadakan ujian susulan baik ujian tengah semester maupun akhir semester, kecuali atas persetujuan dosen/pengampu mata kuliah bersangkutan.
4. Memakai Pakaian rapi dan sopan:
 - **Pria** :
 - Kemeja Lengan panjang atau pendek
 - Celana panjang Rapi
 - Bersepatu
 - Tidak boleh memakai T-shirt tanpa krah atau tanpa lengan
 - **Wanita** :
 - Kemeja Lengan panjang/pendek (tidak ketat dan atau transparan)
 - Rok ATAU Celana panjang (tidak ketat dan atau transparan)
 - Bersepatu
 - Tidak boleh memakai T-shirt tanpa krah atau tanpa lengan
5. Mahasiswa tidak diperbolehkan merokok, makan dan minum pada saat kuliah praktikum.
6. Tas dan perlengkapan lain harus diletakkan pada tempat yang telah disediakan (hanya diperbolehkan membawa barang berharga, Ex: Handphone, dompet dan alat yang diperlukan untuk praktikum)
7. Barang berharga milik peserta kuliah praktikum menjadi tanggung jawab sendiri (laboran tidak bertanggungjawab atas kehilangan barang tersebut).
8. Dering HP harus dimatikan (silent) pada saat kuliah praktikum.
9. Selesai Praktikum, Komputer dimatikan dan kursi dirapikan kembali.
10. Mahasiswa diwajibkan menjaga kebersihan dan ketertiban serta ketenangan belajar.

11. Mahasiswa tidak diperbolehkan menggunakan komputer untuk bermain games.
12. Mahasiswa tidak diperkenankan men-install program/software tanpa petugas lab.
13. Mahasiswa tidak diperkenankan memindah posisi hardware (mouse, keyboard, monitor, CPU)
14. Mahasiswa tidak diperbolehkan membawa atau mengambil (secara sengaja atau tidak sengaja) perlengkapan praktikum yang ada di Laboratorium komputer).
15. Mahasiswa wajib menjaga keutuhan semua peralatan yang ada di Laboratorium Komputer serta tidak diperbolehkan memakai komputer Pengajar/Instruktur.
16. Mengisi daftar hadir dan mencatat nomor kursi/komputer yang digunakan selama praktikum berlangsung ke dalam form daftar hadir yang telah disediakan.
17. Melaporkan keadaan komputer dan atau peralatan yang digunakan (yang rusak/tidak berfungsi) sebelum, sesaat dan atau sesudah penggunaan ke Pengajar/Instruktur atau kepada laboran.
18. Praktikan wajib mematikan Komputer yang telah selesai digunakan dan merapikan kembali kursi, meja dan perlengkapan pendukung praktikum lainnya setelah praktikum selesai dilaksanakan/berakhir.
19. Laboran berhak mencatat, memberikan sanksi atau melakukan tindakan seperlunya terhadap praktikan yang melanggar tata tertib.

LABORATORIUM KOMPUTER DASAR

DIKE F.MIPA UGM

KATA PENGANTAR

Assalamu'alaikum Wr. Wb.

Alhamdulillah, atas berkat rahmat Allah Yang Maha Kuasa dengan didorongkan oleh keinginan luhur memperluas wawasan dalam pengembangan pengetahuan tentang Pengembangan Perangkat Lunak, maka buku petunjuk (modul) praktikum ini disusun/dibuat.

Buku petunjuk (modul) Praktikum Pengembangan Perangkat Lunak ini dibuat untuk membantu jalannya Praktikum Pengembangan Perangkat Lunak prodi S1 Ilmu Komputer. Buku petunjuk (modul) ini dibuat sedemikian rupa sehingga dapat dengan mudah dipahami dan dipelajari oleh mereka yang belum pernah mengenal Pengembangan Perangkat Lunak sekalipun, dan bagi mereka yang pernah mengenal membaca buku ini akan menyegarkan ingatan.

Terima kasih kami ucapkan kepada semua pihak yang telah membantu dan mendukung pembuatan buku ini.

Wassalamu'alaikum Wr. Wb.

LABORATORIUM KOMPUTER DASAR

DIKE F.MIPA UGM

DAFTAR ISI

PENGANTAR REKAYASA PERANGKAT LUNAK.....	1
A. Tujuan Praktikum.....	1
B. Indikator.....	1
C. MATERI.....	1
D. SOAL.....	3
MODEL-MODEL PROSES REKAYASA PERANGKAT LUNAK.....	4
A. Tujuan Praktikum.....	4
B. Indikator.....	4
C. MATERI.....	4
BISNIS PROSES DAN ENTITY RELATIONSHIP DIAGRAM	10
A. Tujuan Praktikum.....	10
B. Indikator.....	10
C. MATERI.....	10
BAGAN ALIR (FLOWCHART).....	15
A. Tujuan Praktikum.....	15
B. Indikator.....	15
C. MATERI.....	15
D. SOAL.....	21
DIAGRAM ALIRAN DATA (DAD)	22
A. Tujuan Praktikum.....	22
B. Indikator.....	22
C. MATERI.....	22
D. SOAL.....	29
E. TUGAS BESAR BAGIAN 1.....	29
UML (UNIFIED MODELLING LANGUAGE)	30
A. Tujuan Praktikum:.....	30
B. Indikator:.....	30
C. Materi	30
D. SOAL.....	34
MACAM-MACAM DIAGRAM UML	35
A. TujuanPraktikum :.....	35
B. Indikator:.....	35
C. Materi	35
D. SOAL.....	38
E. LATIHAN	61
F. TUGAS BESAR BAGIAN 2 (Lanjutan Bagian 1)	61

PERTEMUAN KE 1

MATERI : PENGANTAR REKAYASA PERANGKAT LUNAK

A. Tujuan Praktikum

1. Mengetahui definisi perangkat lunak dan Rekayasa Perangkat Lunak.
2. Mengetahui fase-fase Rekayasa Perangkat Lunak.
3. Mengetahui tahapan proses Rekayasa Perangkat Lunak

B. Indikator

1. Praktikan mengetahui apa yang dimaksud dengan perangkat lunak dan Rekayasa Perangkat Lunak.
2. Praktikan dapat memahami dan menjelaskan fase-fase Rekayasa Perangkat Lunak.
3. Praktikan dapat memahami dan menjelaskan tahapan-tahapan proses Rekayasa Perangkat Lunak.

C. MATERI

1. Definisi Perangkat Lunak dan Rekayasa Perangkat Lunak

Perangkat lunak adalah (1) *instruksi-instruksi (program komputer) yang ketika dieksekusi memberikan fungsi dan unjuk kerja yang diinginkan*; (2) *struktur data yang memungkinkan program-program memanipulasi informasi secara proporsional*, dan (3) *informasi deskriptif dalam bentuk virtual maupun fisik yang mendeskripsikan operasi dan kegunaan dari program*.

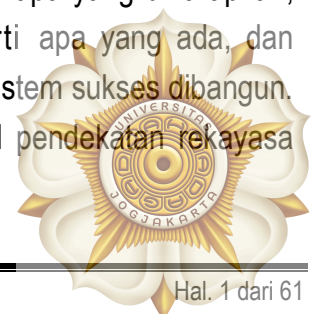
Rekayasa Perangkat Lunak adalah (1) *penerapan sebuah metode (pendekatan) yang sistematis, memenuhi aturan-aturan tertentu yang ada, dan terukur, pada pembangunan, pengoperasian, dan pemeliharaan perangkat lunak*. (2) *Disiplin ilmu yang mempelajari metode-metode (pendekatan) Rekayasa Perangkat Lunak*.

Dari definisi diatas dapat diketahui bahwa bahasan Rekayasa Perangkat Lunak meliputi seluruh aspek produksi perangkat lunak, mulai dari tahap awal analisa kebutuhan pengguna (*requirement capturing*), menentukan spesifikasi dari kebutuhan pengguna (*specification*), desain (*design*), penulisan kode (*coding*), pengetesan (*testing*), hingga pemeliharaan sistem setelah digunakan (*maintenance*).

2. Fase Rekayasa Perangkat Lunak

Pada Rekayasa Perangkat Lunak terdapat tiga fase, yaitu:

- ❖ Fase Definisi (*Definition Phase*) berfokus pada pertanyaan “apa” (*what*), dimana 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, antar muka seperti apa yang akan dibangun, batasan desain seperti apa yang ada, dan kriteria validasi apa saja yang dibutuhkan untuk mendefinisikan bahwa sistem sukses dibangun. Metode yang diterapkan selama fase definisi tergantung pada model pendekatan rekayasa perangkat lunak (atau kombinasi model pendekatan) yang diaplikasikan.



1. Fase Pengembangan (*Development Phase*) berfokus pada pertanyaan “bagaimana” (*how*), yaitu selama masa pengembangan perangkat lunak teknisi harus mendefinisikan bagaimana data dikonstruksikan, bagaimana detail prosedur akan diimplementasikan, bagaimana antar muka ditandai dan dibangun, dan lain sebagainya.
- ❖ Fase Pemeliharaan (*Maintenance Phase*) berfokus pada perubahan atau penyesuaian yang dibutuhkan perangkat lunak terkait dengan koreksi kesalahan, penyesuaian ketika lingkungan perangkat lunak berubah, serta penyesuaian sehubungan perubahan kebutuhan pelanggan/pengguna perangkat lunak. Empat tipe perubahan yang mungkin terjadi selama fase pemeliharaan, yaitu :
 1. Koreksi. Kekurangan atau cacat pada sebuah perangkat lunak mungkin saja baru ditemukan setelah perangkat lunak selesai dibangun. Pemeliharaan yang bersifat korektif bertujuan untuk memperbaiki kekurangan.
 2. Adaptasi. Pemeliharaan yang bersifat adaptasi dilakukan untuk mengakomodasi perubahan di lingkungan eksternal perangkat lunak, seperti misalnya perubahan CPU, sistem operasi dan lain sebagainya. Dengan adanya pemeliharaan ini, perangkat lunak yang ada tetap *adaptive* dan bisa digunakan sesuai dengan peningkatan yang terjadi di lingkungan eksternalnya.
 3. Peningkatan. Ketika, sebuah perangkat lunak digunakan pada lingkungan tertentu, pengguna ataupun customer mungkin saja mengharapkan fungsi atau fitur tambahan yang akan mempermudah/menguntungkan mereka ketika memanfaatkan perangkat lunak tersebut. Pemeliharaan yang bersifat peningkatan akan menghasilkan perangkat lunak yang jauh lebih fungsional dibandingkan dengan saat pertama kali dibuat.
 4. Pencegahan. Performa perangkat lunak mungkin saja memburuk akibat perubahan, oleh karena itu, pemeliharaan preventif harus dilakukan untuk memungkinkan perangkat lunak tetap melayani kebutuhan *end user*. Secara essensial, pemeliharaan yang bersifat pencegahan menghasilkan perubahan pada program computer sehingga program ini bisa secara lebih mudah dikoreksi, disesuaikan dengan keadaan, dan ditingkatkan.

3. Proses Rekayasa Perangkat Lunak

Proses dapat didefinisikan sebagai kumpulan aktifitas, kegiatan, dan tugas-tugas yang dilakukan ketika membuat sebuah produk atau dilakukan untuk mendapatkan hasil yang diinginkan. Proses Rekayasa Perangkat Lunak adalah kegiatan-kegiatan yang dilakukan ketika membangun sebuah perangkat lunak, mulai dari awal analisis kebutuhan hingga akhir pemeliharaan perangkat lunak. Sebuah proses dalam Rekayasa Perangkat Lunak mendefinisikan aktifitas dan keadaan berupa “siapa melakukan apa, kapan, dan bagaimana”.

Pada umumnya Proses Rekayasa Perangkat Lunak yang biasa juga disebut dengan *Software Development Life Cycle* (SDLC) terdiri dari beberapa tahapan yang dapat digolongkan berdasarkan aktifitas yang dilakukan, diantaranya adalah:

- Requirement analysis and specification

Pada tahap ini dilakukan pengumpulan data tentang perangkat lunak yang akan dibangun. Analisis kebutuhan dilakukan untuk mengetahui secara spesifik program seperti apa yang dibutuhkan oleh calon pengguna mulai dari unjuk kerja, kemampuan apa saja yang harus dimiliki perangkat lunak, antar muka seperti apa yang diharapkan, arsitektur secara umum, dan lain sebagainya. Informasi tentang perangkat lunak bias didapatkan dengan bermacam-macam cara seperti interaksi langsung dengan calon pengguna, memahami lingkungan/system dimana perangkat lunak akan dipakai, dan lain sebagainya. Setelah semua informasi terkumpul maka dibuat sebuah dokumen berisi spesifikasi program yang akan dibangun.

- Design

Setelah kebutuhan dan persyaratan perangkat lunak diketahui, selanjutnya dibangun sebuah desain yang menggambarkan bagaimana perangkat lunak akan dibangun dan bagaimana kira-kira hasil akhir dari perangkat lunak yang akan dibangun. Desain disini meliputi antar muka, arsitektur basis data, alat-alat (tools) yang akan digunakan untuk membangun perangkat lunak, hingga desain pengetesan akhir.

- Implementation (coding)

Tahap implementasi adalah tahapan dimana desain perangkat lunak yang telah dibuat direalisasikan kedalam kode-kode dan elemen-elemen pembangunan perangkat lunak.

- Validation (testing)

Pada tahap pengetesan dan validasi dilakukan pengecekan apakah ada kesalahan pada perangkat lunak yang sedang dibangun dan juga pengecekan apakah perangkat lunak yang dibangun telah memenuhi kebutuhan dan spesifikasi yang telah didokumentasikan pada tahap *requirement analysis and specification*.

- Maintenance

Tahapan terakhir adalah *maintenance*. Pada tahapan ini dilakukan pemeliharaan dan juga perubahan pada perangkat lunak apabila dibutuhkan. Proses pemeliharaan biasanya dilakukan untuk mengantisipasi dan memperbaiki kesalahan (*error*) yang baru diketahui setelah perangkat lunak digunakan. Perubahan pada perangkat lunak, baik itu penambahan, pengurangan, maupun pembaharuan bagian dari perangkat lunak menyesuaikan dengan kebutuhan pengguna terkait dengan perubahan lingkungan system.

Dari tahapan proses yang telah dijelaskan dapat diketahui bahwa dalam sebuah proses pengembangan perangkat lunak tahapan implementasi (*coding*) bukanlah satu-satunya tahapan terpenting dan bahkan pada umumnya bukan tahapan dengan durasi terlama. Seringkali tahapan yang memakan waktu paling banyak dan sangat krusial adalah tahapan analisis kebutuhan dan tahapan pemeliharaan (*maintenance*).

D. SOAL

1. Sebutkan pengertian rekayasa perangkat lunak dan apa pentingnya rekayasa perangkat lunak dalam membangun sebuah perangkat lunak.



PERTEMUAN KE 2

MATERI : MODEL-MODEL PROSES REKAYASA PERANGKAT LUNAK

A. Tujuan Praktikum

1. Menjelaskan apakah yang dimaksud dengan model proses Rekayasa Perangkat Lunak dan kenapa model proses dibutuhkan (fungsi dari model proses).
2. Mengenalkan dan menjelaskan macam-macam model proses Rekayasa Perangkat Lunak.
3. Mengetahui perbedaan antar model proses dan mengetahui kelebihan dan kekurangan masing-masing model proses.

B. Indikator

1. Praktikan dapat menjelaskan definisi dan fungsi model proses rekayasa perangkat lunak.
2. Praktikan dapat menyebutkan dan menjelaskan jenis-jenis model proses rekayasa perangkat lunak.
3. Praktikan dapat mengetahui perbedaan antar model proses serta kelebihan dan kekurangan masing-masing model proses.

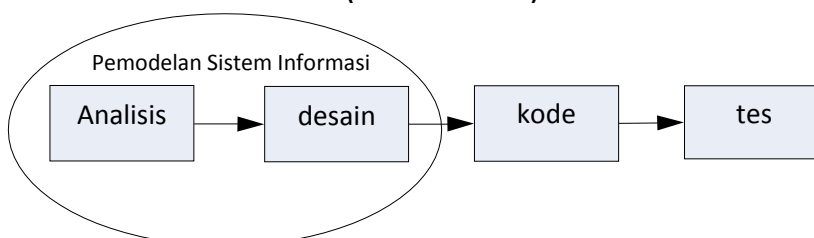
C. MATERI

2.1 Model Proses Perangkat Lunak

Untuk menyelesaikan masalah yang nyata dalam suatu hal, perekayasa perangkat lunak harus menggabungkan strategi pengembangan yang melingkupi lapisan proses, metode dan alat-alat bantu serta fase-fase generik (akan dijelaskan pada bab selanjutnya). Strategi yang dimaksud, sering diacukan sebagai *model proses* atau *paradigma rekayasa perangkat lunak*. Model proses untuk rekayasa perangkat lunak dipilih berdasarkan sifat aplikasi dan proyeknya, metode dan alat-alat bantu yang akan dipakai.

Pada bab ini selanjutnya akan didiskusikan bermacam-macam model proses yang berbeda pada perangkat lunak. Penting untuk diingat bahwa masing-masing model sudah ditandai dengan cara tertentu sehingga diharapkan bisa membantu di dalam kontrol dan koordinasi dari proyek perangkat lunak yang nyata. Dengan demikian, pada intinya semua model menunjukkan karakteristiknya.

2.1.1. MODEL SEKUENSIAL LINIER (WATERFALL)



Gambar Model Sekuensial Linier (Waterfall Model)

Gambar diatas menggambarkan *sekuensial linier* untuk rekayasa perangkat lunak, yang sering disebut dengan *siklus kehidupan klasik* atau *model air terjun*. Sekuensial linier menghasilkan sebuah

pendekatan kepada pengembangan perangkat lunak yang sistematis dan sekuensial yang mulai pada tingkat dan kemajuan system pada seluruh analisis, desain, kode, pengujian (tes), dan pemeliharaan. Tahapan model proses ini seperti yang telah disebutkan pada tahapan proses di BAB I, yaitu sebagai berikut:

- ❖ Analisis kebutuhan perangkat lunak

Proses pengumpulan kebutuhan difokuskan khususnya untuk perangkat lunak, pereayasa perangkat lunak (Analisis) harus memahami domain permasalahan (problem domain), tingkah laku, unjuk kerja dan antarmuka (interface) yang diperlukan. Kebutuhan baik untuk sistem maupun perangkat lunak didokumentasikan dan dilihat lagi dengan pelanggan.

- ❖ Desain

Desain perangkat lunak sebenarnya adalah proses multi langkah yang berfokus pada empat atribut sebuah program yang berbeda (struktur data, arsitektur perangkat lunak, representasi interface, dan detail (algoritma) prosedural. Proses desain menerjemahkansyarat/kebutuhanke dalam sebuah representasi perangkat lunak yang dapat diperkirakan demi kualitas sebelum dimulai pemunculan kode (*coding*). Sebagaimana analisis, desain ini juga didokumentasikan.

- ❖ Generasi kode

Desain harus diterjemahkan ke dalam bentuk mesin yang bisa dibaca. Langkah pembuatan kode meliputi pekerjaan dalam langkah ini, dan dapat dilakukan secara mekanis.

- ❖ Pengujian (Tes)

Sekali kode dibuat, pengujian program dimulai. Proses pengujian berfokus pada logika internal perangkat lunak, memastikan bahwa semua pernyataan sudah diuji, dan pada eksternal fungsional, yaitu mengarahkan pengujian untuk menemukan kesalahan-kesalahan dan memastikan bahwa input yang dibatasi akan memberikan hasil aktual yang sesuai dengan hasil yang dibutuhkan.

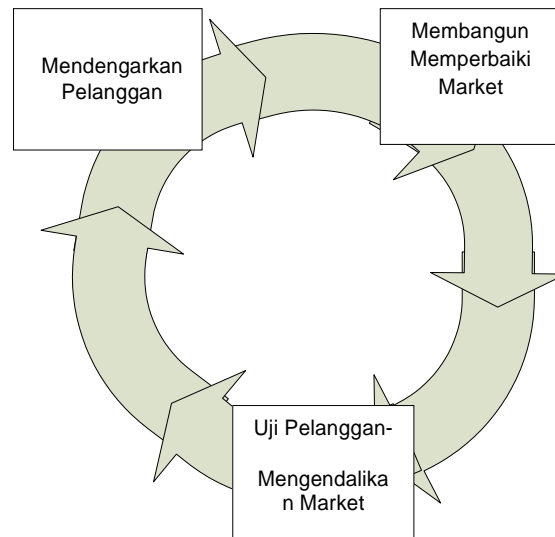
- ❖ Pemeliharaan

Perangkat lunak akan mengalami perubahan setelah disampaikan kepada pelanggan (perkecualian yang mungkin adalah perangkat lunak yang dilekatkan). Perubahan akan terjadi karena kesalahan-kesalahan karena perangkat lunak harus disesuaikan untuk mengakomodasi perubahan-perubahan di dalam lingkungan eksternalnya atau karena pelanggan membutuhkan perkembangan fungsional atau unjuk kerja. Pemeliharaan perangkat lunak mengaplikasikan lagi setiap fase program sebelumnya dan tidak membuat yang baru lagi.



2.1.2 MODEL PROTOTIPE

Model prototipe ini dimulai dengan pengumpulan kebutuhan. Pengembang dan pelanggan bertemu dan mendefinisikan obyektif keseluruhan dari perangkat lunak, dan mengidentifikasi segala kebutuhan yang diketahui.



Gambar. Model Prototipe

Secara ideal prototipe berfungsi sebagai sebuah mekanisme untuk mengidentifikasi kebutuhan perangkat lunak. Prototipe bisa menjadi paradigma yang efektif bagi rekayasa perangkat lunak. Kuncinya adalah mendefinisikan aturan-aturan main pada saat awal, yaitu pelanggan dan pengembang keduanya harus setuju bahwa prototipe dibangun untuk berfungsi sebagai mekanisme pendefinisian kebutuhan. Prototipe kemudian disingkirkan dan perangkat lunak actual direkayasa dengan tertuju kepada kualitas dan kepada kemampuan pemeliharaan.

2.1.3. MODEL RAD

Rapid Application Development (RAD) adalah sebuah model proses perkembangan perangkat lunak sekuensial linier yang menekankan siklus perkembangan yang sangat pendek. Model RAD ini merupakan sebuah adaptasi “kecepatan tinggi” dari model sekuensial linier dimana perkembangan cepat dicapai dengan menggunakan pendekatan konstruksi berbasis komponen. 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 melingkupi fase-fase sebagai berikut :



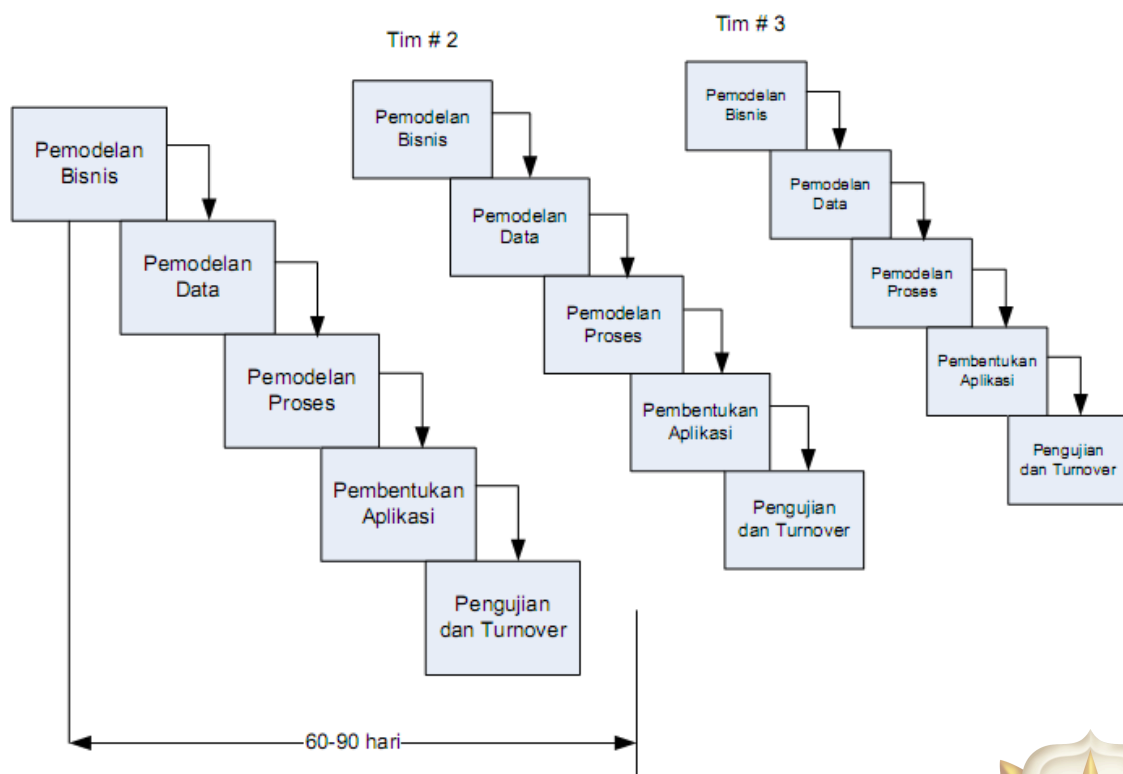
Pemodelan Bisnis. Aliran informasi diantara fungsi-fungsi bisnis dimodelkan dengan suatu cara untuk menjawab pertanyaan-pertanyaan berikut : Informasi apa yang mengendalikan proses bisnis? Informasi apa yang dimunculkan? Siapa yang memunculkannya? Kemana informasi itu pergi? Siapa yang memprosesnya?

Pemodelan Data. Aliran informasi yang didefinisikan sebagai bagian dari fase business modelling disaring kedalam serangkaian objek data yang dibutuhkan untuk mendukung bisnis tersebut.

Pemodelan Proses. Aliran informasi yang didefinisikan di dalam fase data modelling 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.

Pembentukan Aplikasi. RAD mengasumsikan pemakaian teknik generasi keempat. Selain menciptakan perangkat lunak dengan menggunakan bahasa pemrograman generasi ketiga yang konvensional, RAD lebih banyak memproses kerja untuk memakai lagi komponen program yang ada atau menciptakan komponen yang bisa dipakai lagi.

Pengujian dan Turnover. Karena proses RAD menekankan pada pemakaian kembali, banyak komponen program telah diuji. Hal ini mengurangi keseluruhan waktu pengujian. Tetapi komponen baru harus diuji dan semua interface harus dilatih secara penuh.



Gambar. Model RAD (Rapid Application Development)

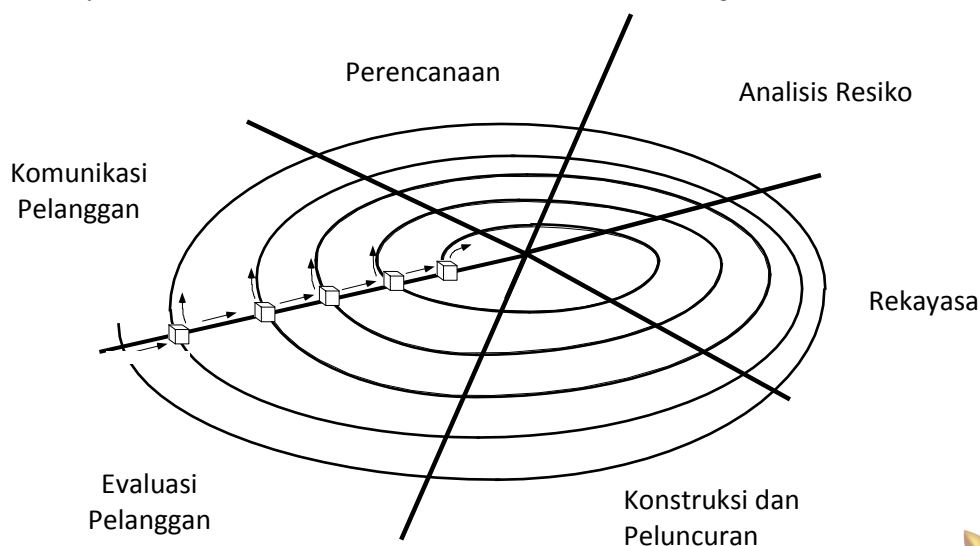


2.1.4. MODEL SPIRAL

Model spiral (*spiral model*) yang pada awalnya diusulkan oleh Boehm adalah model proses perangkat lunak yang evolusioner yang merangkai sifat iteratif dari prototipe dengan cara kontrol dan aspek sistematis dari model sekuensial linier. Di dalam model spiral, perangkat lunak dikembangkan di dalam suatu deretan pertambahan.

Selama awal iterasi, rilis inkremental bisa merupakan sebuah model atau prototipe kertas. Selama iterasi berikutnya, sedikit demi sedikit dihasilkan versi sistem rekayasa yang lebih lengkap. Model spiral dibagi menjadi sejumlah aktifitas kerangka kerja, disebut juga wilayah tugas, diantara tiga sampai enam wilayah tugas :

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



Gambar. Model Spiral



Model spiral menjadi sebuah 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 prototipe sebagai mekanisme pengurangan resiko. Tetapi yang lebih penting lagi, model spiral memungkinkan pengembang menggunakan pendekatan prototipe pada setiap keadaan di dalam evolusi produk. Model spiral menjaga pendekatan langkah demi langkah secara sistematis seperti yang diusulkan oleh siklus kehidupan klasik, tetapi memasukkannya ke dalam kerangka kerja iterative yang secara realistis merefleksikan dunia nyata.



PERTEMUAN KE 3

MATERI : BISNIS PROSES DAN *ENTITY RELATIONSHIP DIAGRAM*

A. Tujuan Praktikum

1. Memperkenalkan dan menjelaskan bisnis proses untuk perancangan perangkat lunak.
2. Menjelaskan *Entity Relationship Diagram*.
3. Memberikan studi kasus implementasi bisnis proses suatu perangkat lunak.

B. Indikator

1. Agar praktikan dapat menjelaskan definisi dan fungsi bisnis proses.
2. Agar praktikan dapat menyebutkan dan menjelaskan *Entity Relationship Diagram* beserta kegunaannya.
3. Agar praktikan dapat bisnis proses untuk perancangan suatu sistem perangkat lunak.

C. MATERI

3.1 Pengertian Bisnis Proses

Proses bisnis (*business process*) dapat didefinisikan sebagai kumpulan dari proses dan berisi kumpulan aktifitas (*tasks*) yang saling berelasi satu sama lain untuk menghasilkan suatu keluaran yang mendukung pada tujuan dan sasaran strategis dari organisasi.

3.2 Penggunaan Bisnis Proses

Sebelum menentukan bisnis proses, dibutuhkan penjabaran mengenai deskripsi permasalahan, serta *goal* suatu sistem yang akan dibuat. Pada kasus ini, akan diberikan contoh kasus yang berhubungan dengan sistem informasi penerbangan pesawat. Ilustrasi deskripsi permasalahan dijabarkan sebagai berikut.

3.2.1 Deskripsi permasalahan

Pada setiap bandara sudah mempunyai jadwal penerbangan untuk semua maskapai baik penerbangan domestik maupun internasional. Dimana pihak bandara telah melakukan penjadwalan terhadap pesawat dan rute penerbangan baik keberangkatan serta kedatangan. Petugas yang mengatur lalu lintas udara yang biasa disebut air traffic controller (ATC) memiliki peranan yang penting terutama pada pesawat udara untuk mencegah antar pesawat tidak terlalu dekat satu sama lain, mencegah tabrakan antar pesawat udara dan pesawat udara dengan rintangan yang ada di sekitarnya selama beroperasi. Selain itu ATC mempunyai peran dalam pengaturan kelancaran arus lalu lintas, membantu pilot dalam mengendalikan keadaan darurat, memberikan informasi yang dibutuhkan pilot (seperti informasi cuaca, informasi navigasi penerbangan, dan informasi lalu lintas udara). ATC adalah rekan terdekat pilot selama di udara, peran ATC sangat besar dalam tercapainya tujuan penerbangan. Dalam sistem pelayanan penerbangan mempunyai beberapa pelayanan yang ditawarkan melalui web diantaranya sebagai berikut.



1. Pihak bandara dapat melihat keadaan cuaca saat ini, baik atau buruk untuk melakukan penerbangan, status penerbangan (landing, takeoff, delay), jadwal kedatangan, penerbangan dan tujuan pesawat.
2. Maskapai dapat melihat jadwal tugas penerbangan dari dan ke mana, siapa pilot, pramugari dan kru-krunya.
3. Agen bisa menjual, booking, memantau harga-harga pesawat dan jadwal penerbangan.
4. Customer dapat melihat jadwal penerbangan, rute penerbangan, melihat harga tiket pesawat, booking dan pesan tiket.

Dari deskripsi di atas, diharapkan agar penjadwalan penerbangan bandara, jadwal pilot, jadwal kru, dan pembelian tiket pesawat dapat dilakukan dengan lebih cepat dan efisien melalui media berbasis web.

3.3 Bisnis Proses

Proses bisnis dan operasional suatu bandara penerbangan dapat melibatkan banyak pihak. Pihak pengatur otoritas bandara berfungsi untuk mengatur pengendalian takeoff atau landing setiap pesawat, mengatur lalu lintas udara pada saat takeoff atau landing (ATC). Untuk informasi cuaca petugas ATC akan memberikan informasi cuaca kepada pilot, disini petugas ATC memberitahukan apakah pilot harus menunggu untuk landing, takeoff bahkan cancel penerbangan. Petugas ATC juga mempunyai wewenang terhadap pilot untuk takeoff dan landing. Pihak bandara dapat melihat semua jadwal keberangkatan atau kedatangan pesawat. Pihak bandara dapat menampilkan semua jadwal penerbangan, beserta dengan jadwal kedatangan atau keberangkatan, status (delay, landed, call waiting room, takeoff), serta jadwal keberangkatan atau kedatangan tambahan. BMKG akan terintegrasi dengan pihak menara untuk meminta data perkiraan cuaca. BMKG memberikan semua informasi cuaca kepada pihak menara yang menjadi otoritas petugas bandara.

Manajemen maskapai penerbangan menjadwalkan tim pilot dan kru untuk penerbangan semua rute yang sudah di jadwalkan. Pilot dan kru mempunyai id karyawan untuk mengecek jadwal penerbangannya untuk hari ini, besok, lusa, seminggu, dan bulan kedepan. Pihak pemilik maskapai penerbangan yang menawarkan bisnis jasa penerbangan kepada customer seperti memberikan informasi jadwal penerbangan, tujuan, dan harga kepada customer. Customer dapat melakukan pembelian langsung ke bandara atau dapat secara online melalui web. Customer akan mendapatkan kode booking dan kode pembayaran. Setelah melakukan pembayaran customer akan di kirim e-mail tiket.

Untuk pihak agen, agen akan menyediakan informasi tiket ke customer dari semua jadwal maskapai penerbangan dan agen akan memberikan harga terbaik dari semua penerbangan. Agen dapat memberikan informasi semua itu karena agen memiliki hak akses dan memiliki kode akses agen sebagai member di setiap maskapai. Karena agen telah menjadi member maka agen dapat melihat semua info yang dibutuhkan oleh customer dan dapat melakukan pembelian tiket pesawat.

Customer yang akan melakukan penerbangan dihari dan waktu yang berbeda, bisa memesan tiket dan memilih asal-tujuan penerbangan dengan informasi harga dan list jadwal pada hari-hari

tersebut. Untuk booking customer membayar perbatas waktu yang sudah ditentukan dan ketika batas waktu berakhir maka kode booking akan hangus. Setelah booking dan melakukan pembayaran selanjutnya customer melakukan cek in untuk mendapatkan tempat duduk. Cek in bisa dilakukan secara online dan bisa langsung di dalam bandara dengan memasukan kode booking, firstname dan lastname. Jika secara online customer dapat memilih kursi sesuai dengan yang diinginkan tapi jika di bandara makan petugas yang akan memilihkan nomor kursi yang masih kosong dengan menunjukan kode booking dan selanjutnya petugas akan mencetak boarding passs untuk penumpang.

Agar keamanan dalam penerbangan makan untuk setiap pesawat sebelum melakukan penerbangan harus di periksa terlebih dahulu. Pemeriksaan dilakukan oleh kru mekanik dan kru mekanik memiliki jadwal periksa masing-masing agar tiap hari pesawat rutin dilakukan pengecekan (fan, ac, fungsi dikabin kopilot, mesin, ban, dll) supaya menghindari hal-hal yang tidak diinginkan.

Setelah selesai mendeskripsikan bisnis proses dapat dilakukan analisis fungsional untuk menentukan proses besar sistem.

3.4 Rancangan Proses

Dalam sistem penjadwalan penerbangan bandara, jadwal pilot-kru dan pembelian tiket pesawat terdapat proses-proses sebagai berikut.

- **Proses penjadwalan tugas pilot dan kru.** Pihak Maskapai menentukan jadwal penugasan pilot dan kru. Sehingga pilot dan kru dapat melihat informasi jadwal penugasannya.
- **Proses penjadwalan jam dan rute penerbangan.** Pihak maskapai menentukan penjadwalan jam dan rute penerbangan pesawat. Dimana rute meliputi asal dan tujuan penerbangan. Sedangkan penjadwalan jam meliputi jam keberangkatan dan kedatangan pesawat. Jadwal jam dan rute dapat dilihat juga oleh pihak bandara dan customer. Informasi Jam dan Rute yang dibuat Maskapai akan terhubung dengan bandara, sehingga customer juga dapat melihat informasi tersebut di Masakapai maupun Bandara.
- **Proses pemesanan tiket pesawat.** Pihak agen yang telah bekerja sama dengan beberapa atau seluruh maskapai penerbangan dapat melakukan login atau mengakses website yang telah disediakan oleh maskapai masing-masing. Agen login, melakukan booking untuk customer, mencatat data-data customer dan melakukan pencetakan tiket untuk customer.
- **Proses pemantauan cuaca.** Pihak bandara (bagian ATC) akan meminta data dari BMKG tentang kondisi cuaca terkini. Kemudian nntinya ATC akan memberikan informasi kepada pilot apakah pilot dapat mendarat (*landing*) ataupun berangkat (*takeoff*) di bandara tersebut. Hasil dari pemantauan ini adalah status penerbangan sudah *ready*, *waiting* atau *cancel*.
- **Proses pemantauan penerbangan pesawat.** Pihak bandara mencatat, dan memantau status keberangkatan atau kedatangan pesawat. Bilamana terjadi *delay*, sudah *takeoff*, sudah *landing* atau *cancel* penerbangan. Hasil dari pemantauan ini akan disimpan dan diproses untuk dapat ditampilkan dalam status penerbangan seperti no pesawat, nama maskapai, jadwal keberangkatan, jadwal kedatangan, serta status.

- **Proses pembelian tiket pesawat.** Customer dapat melakukan pembelian tiket pada Agen. Agen pun dapat melayani pembelian tiket lebih dari satu maskapai. Customer akan mendapatkan nota pembayaran dari agen, beserta tiketnya. Serta Customer dapat membeli tiket atau *booking* melalui sistem yang disediakan oleh maskapai masing-masing.
- **Proses pemesanan kursi (*check in*).** Customer ataupun petugas dapat melakukan *checkin* untuk mendapatkan nomor kursi di pesawat. Dimana akan diinputkan *first name* ataupun *lastname* dan kode *booking* tiket. Nantinya dapat dipilih kursi yang *tersedia*, kemudian mendapatkan *boardingpass*.

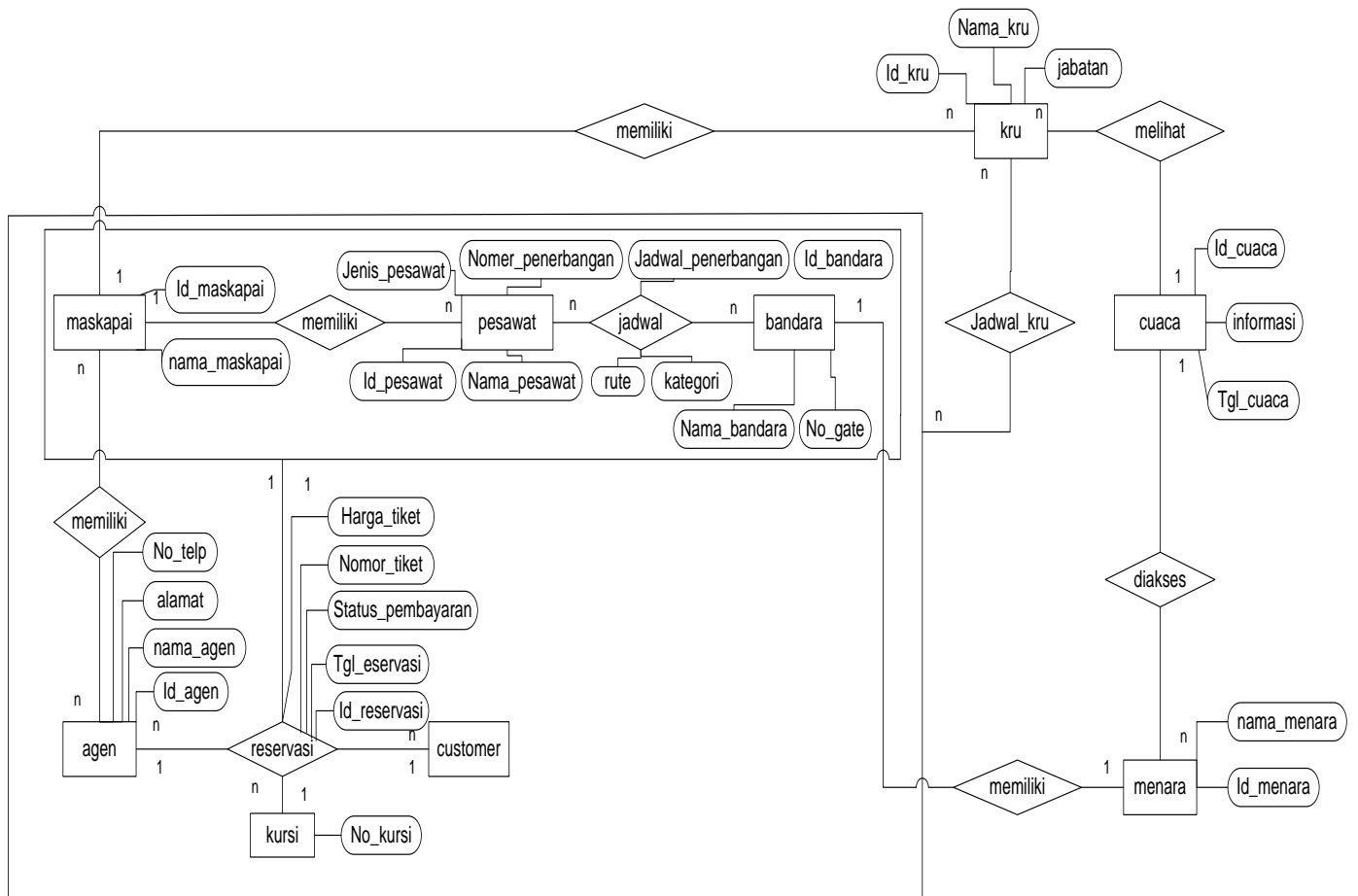
3.5 Spesifikasi User

Berdasarkan bisnis proses dan rancangan proses, dapat ditentukan spesifikasi *user* yang nanti dapat digunakan untuk pemodelan data yang digambarkan dengan menggunakan *Entity Relationship Diagram* (ERD).

- **Agen** dapat terhubung dengan maskapai untuk melakukan pemesanan tiket. Selain itu juga terdapat layanan yang secara otomatis mengenai jadwal penerbangan, rute pesawat, harga tiket, dan informasi ketersediaan tempat duduk.
- **Maskapai** adalah pihak yang menentukan jadwal-jadwal mengenai tugas pilot dan kru, jam pemberangkatan dan kedatangan, serta rute asal dan tujuan penerbangan. Maskapai juga melayani pemesanan kursi setelah customer memiliki tiket.
- **Bandara** yang terhubung pihak Maskapai dapat menampilkan Jadwal penerbangan beserta rute penerbangannya. Pemantauan cuaca pun dapat diperlihatkan oleh Bandara yang terhubung dengan BMKG.
- **Customer** dapat melihat jadwal penerbangan, dimana customer melakukan akses langsung ke sistem maskapai untuk mendapatkan informasi penerbangan sesuai dengan rute serta jadwal yang diinginkan. Customer yang ingin melakukan pemesanan tiket akan berhubungan dengan Agen atau Maskapai. Proses pemesanan tiket dapat secara online.



3.6 Pemodelan data menggunakan ERD



PERTEMUAN KE 4

MATERI : BAGAN ALIR (FLOWCHART)

A. Tujuan Praktikum

1. Memperkenalkan dan menjelaskan Flowchart sebagai media perancangan dan dokumentasi suatu sistem.
2. Menjelaskan jenis-jenis Flowchart dan kegunaannya.
3. Memberikan studi kasus penggunaan Flowchart.

B. Indikator

1. Agar praktikan dapat menjelaskan definisi dan fungsi Flowchart.
2. Agar praktikan dapat menyebutkan dan menjelaskan jenis-jenis Flowchart serta kegunaannya.
3. Agar praktikan dapat menggunakan Flowchart untuk merancang atau mendokumentasikan prosedur/algoritma dari suatu sistem.

C. MATERI

4.1 Definisi dan Fungsi *Flowchart*

Flowchart adalah diagram yang merepresentasikan proses dan algoritma pada sebuah sistem secara visual. *Flowchart* berbeda dengan DAD. Perbedaan yang paling jelas antara Flowchart dan DAD adalah dari tahapan/langkah alur data serta simbol-simbol yang digunakan. Perbedaan lainnya adalah DAD menunjukkan aliran data dan tidak memperdulikan urutan kejadian, sedangkan *Flowchart* bersifat sekuensial, runtut menunjukkan proses, prosedur, dan operasi-operasi dalam suatu sistem.

Flowchart tersusun atas gambar/diagram yang mempunyai aliran satu atau dua arah secara sekuensial sekuensial dan biasanya digambarkan dari atas ke bawah. *Flowchart* digunakan untuk merepresentasikan maupun mendesain program. Oleh karena itu flowchart harus bisa merepresentasikan komponen-komponen dalam bahasa pemrograman. *Flowchart* dapat dibuat sebelum maupun setelah pembuatan program. *Flowchart* yang dibuat sebelum membuat program digunakan untuk mempermudah pembuat program untuk menentukan alur logika program, sedangkan yang dibuat setelah pembuatan program digunakan untuk menjelaskan alur program kepada orang lain atau sebagai dokumentasi.

Berikut adalah macam-macam flowchart:

1. Bagan Alir Sistem (*System Flowchart*)

Bagan alir sistem merupakan bagan yang menunjukkan arus pekerjaan secara keseluruhan dari sistem. Bagan ini menjelaskan urutan dari prosedur-prosedur dan menunjukkan apa yang dikerjakan di sistem. Contoh bagan alir sistem adalah bagan alir pendaftaran pasien di RS, bagan alir pendaftaran matakuliah praktikum, dan lain sebagainya.



2. Bagan Alir Dokumen (*Document Flowchart*)






Bagan alir dokumen atau disebut juga bagan alir formulir (*form flowchart/paperwork flowchart*) merupakan bagan alir yang menunjukkan arus laporan, formulir, dan bermacam bentuk dokumen lain termasuk tembusan-tembusannya pada suatu sistem. Bagan alir dokumen ini menggunakan simbol-simbol yang sama dengan yang digunakan di dalam bagan alir sistem. Contoh. Pelaporan bulanan Perusahaan.

3. Bagan Alir Skematik (*Schematic Flowchart*)

Bagan alir skematik (*schematic flowchart*) merupakan bagan alir yang mirip dengan bagan alir sistem, yaitu untuk menggambarkan prosedur di dalam sistem. Perbedaannya adalah, bagan alir skematik selain menggunakan simbol-simbol bagan alir sistem, juga menggunakan gambar-gambar komputer dan peralatan lainnya yang digunakan. Maksud penggunaan gambar-gambar ini adalah untuk memudahkan komunikasi kepada orang yang kurang paham dengan simbol-simbol bagan alir. Penggunaan gambar-gambar ini memudahkan untuk dipahami, tetapi sulit dan lama menggambaranya.. Contoh. Bagan alir proses robot, bagan alir proses pencetakan dokumen.

4. Bagan Alir Proses (*Process Flowchart*)

Bagan alir proses (*process flowchart*) merupakan bagan alir yang banyak digunakan di teknik industri. Bagan alir ini juga berguna bagi analis sistem untuk menggambarkan proses dalam suatu prosedur. Bagan alir proses mcnggunakan lima buah simbol tersendiri.





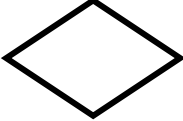

	menunjukkan suatu operasi (operation)
	menunjukkan suatu pemindahan (movement)
	menunjukkan suatu simpanan (storage)
	menunjukkan suatu inspeksi (inspection)
	menunjukkan suatu penundaan (delay)

5. Bagan Alir Program (*Program Flowchart*)

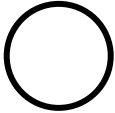
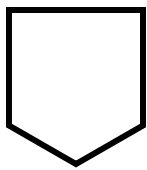
Bagan alir program (*program flowchart*) merupakan bagan yang menjelaskan secara rinci langkah-langkah dari proses sebuah program. Bagan alir inilah yang paling sering digunakan

oleh kalangan IT. Simbol yang digunakan pada bagan alir ini dapat dilihat pada gambar 4.1. Contoh penggunaan bagan alir: Bagan alir untuk program penghitungan faktorial, bagan alir untuk proses penghitungan luas sebuah benda.

Bagan alir program dapat terdiri dari dua macam, yaitu bagan alir logika program (*program logic flowchart*) dan bagan alir program komputer terinci (*detailed computer program flowchart*). Bagan alir logika program digunakan untuk menggambarkan tiap-tiap langkah di dalam program komputer secara logika. Bagan alir logika program ini dipersiapkan oleh analis sistem. Bagan alir program komputer terinci (*detailed computer program flow-chart*) digunakan untuk menggambarkan instruksi-instruksi program komputer secara terinci. Bagan alir ini dipersiapkan oleh pemrogram.

Nama	Simbol	Keterangan
Oval		Menunjukkan notasi untuk awal dan akhir bagan alir
Aliran data		Menunjukkan petunjuk dari aliran fisik pada program
Data		Menunjukkan suatu operasi input atau suatu operasi output
Rectangle		Menunjukkan suatu proses yang akan digunakan
Diamond		Menotasikan suatu keputusan (atau cabang) yang akan dibuat. Pro
Predefined process (Sub program)		Permulaan sub program/ proses menjalankan subprogram



On page connector		Penghubung bagian-bagian flowchart yang berada oada satu halaman
Off page connector		Penghubung bagian-bagian flowchart yang berada pada halaman berbeda

Gambar 4.2. Simbol-simbol pada Program Flowchart

Keterangan tambahan :

1. Tanda Oval, biasanya disebut sebagai *start and end symbol*. Biasanya diberi nama “Mulai” dan “Selesai”, atau “Start” dan “End” ataupun frase lain yang menunjukkan bahwa program/bagan alir tersebut mulai dan selesai.
2. Penunjuk aliran data berupa panah menunjukkan alur dari suatu operasi/peroses ke operasi/proses selanjutnya.
3. Jajaran genjang digunakan untuk menandai proses masukan (input) dan keluaran (output). Biasanya dipakai kata “Baca” untuk menandai masukan dan kata “Cetak” untuk menandai keluaran.
4. Persegi panjang menjelaskan proses yang terjadi. Proses dijelaskan dengan menuliskan kata atau kalimat dan dapat juga dengan notasi matematika yang mengindikasikan adanya suatu proses kerja yang bukan merupakan proses masukan atau keluaran.
5. Diamond digunakan sebagai tanda pemilihan yang bersifat kondisional atau suatu keputusan. Simbol ini mengandung pertanyaan Yes/No atau Ya/Tidak atau Benar/Salah. Simbol ini memiliki dua simbol aliran data yang keluar, biasanya dari sudut yang bawah dan sudut yang kanan, satunya menyatakan “True” (benar) dan satunya menyatakan “False” (salah).
6. Simbol penghubung digunakan untuk menunjukkan sambungan dari bagan alir yang terputus baik di halaman yang masih sama atau di halaman yang berbeda.

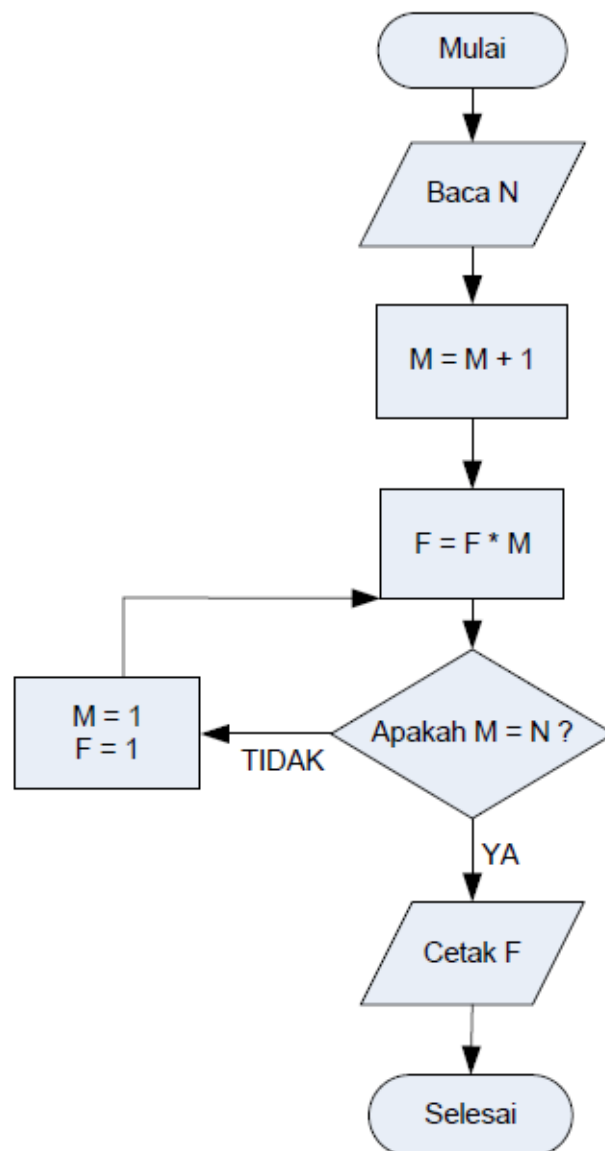


Gambar 4.3 Simbol penghubung pada Flowchart



Contoh kasus penggunaan Flowchart :

1. Berikut ini adalah contoh dari bagan alir untuk proses menghitung Faktorial N ($N!$), dimana $N! = 1 * 2 * 3 * 4 * 5 * \dots * N$. fungsi dari bagan alir untuk menghitung N Faktorial ini akan mempermudah kita dalam menuliskan pada program komputer, karena dari bagan alir ini, kita akan mudah untuk memahami algoritma yang digunakan, karena flowchart ini merupakan tahapan suatu instruksi seperti halnya algoritma suatu program. N = bilangan faktorial yang dicari, M = bilangan 1, 2, 3, 4...N.



Flowchart penghitungan N factorial



Yang perlu diperhatikan saat akan membuat diagram alir proses peminjaman

VCD yaitu :

Investigasi data

Wawancara/Observasi sistem

Narasi

Investigasi Data

Kartu anggota

Fotokopi KTP/SIM

Lembar identitas

Nota peminjaman

Data VCD

Laporan bulanan

Wawancara

Hasil Wawancara

- a. Orang yang terlibat: Anggota, admin, manager
- b. Calon anggota harus mendaftar dengan membawa identitas diri

Hasil Quesioner

- a. Keterlambatan tidak didenda didenda
- b. Kesulitan dalam mencari data vcd pada arsip vcd

Narasi

Calon anggota penyewaan vcd mengisi lembar identitas dan memberikan kartu pengenalan (SIM/KTP)

Petugas memeriksa apakah data tersebut sudah ada pada arsip anggota

Jika tidak ada petugas akan membuat kartu anggota baru dan memberikannya pada anggota

Pada saat meminjam anggota harus menunjukkan kartu anggota dan memberikan data vcd yang akan dipinjam

Petugas akan mencari dari arsip vcd

- i. Jika vcd tidak ada atau sedang dipinjam maka petugas akan memberitahukan status kosong ke anggota
- ii. Jika ada maka petugas akan membuat nota peminjaman dan memberikannya ke anggota



D. SOAL

1. Buatlah Flowchart program untuk menghitung dan menampilkan volume tabung dengan rumus ($\phi \times \text{jari-jari} \times \text{tinggi}$) dan luas selimut tabung dengan rumus $2 \times \phi \times \text{jari-jari} \times \text{tinggi}$ dengan panjang diameter ($\text{jari-jari} = 1/2 \times \text{diameter}$) dan tinggi tabung diinputkan melalui keyboard pada saat program dieksekusi.
2. Akan dibuat sebuah aplikasi sederhana untuk menghitung program diskon belanja di sebuah pusat perbelanjaan. Ketentuan yang berlaku untuk diskon ini, yaitu :
 - A. Jika total nilai belanja minimal Rp. 700.000 maka diskon yang diberikan adalah 30% dari total belanja.
 - B. Jika total nilai belanja diantara Rp. 300.000 dan Rp. 700.000 maka diskon yang diberikan adalah 20% dari total belanja.
 - C. Jika total nilai belanja kurang Rp. 300.000, maka customer tidak akan mendapatkan diskon total belanja. Namun, jika di dalam daftar belanja customer terdapat minimal tiga item pakaian dengan merk yang sama, maka untuk item dengan merk tersebut, customer hanya perlu membayar sejumlah total item kurang satu (contoh, jika di dalam daftar belanja pengguna terdapat lima item merk A, maka hanya customer perlu membayar empat item untuk merk A tersebut).
 - D. Customer yang memiliki total nilai belanja minimal Rp. 700.00, selain mendapatkan diskon, data pribadinya juga akan disimpan diikutkan dalam program pengundian mobil.
3. Perhatikan kembali studi kasus pada soal pada bab 3 no 3. Buatlah flowchart untuk :
 1. Pencarian sebuah buku
 2. Transaksi peminjaman buku



PERTEMUAN KE 5

MATERI : DIAGRAM ALIRAN DATA (DAD)

A. Tujuan Praktikum

1. Memperkenalkan Diagram Aliran Data sebagai alat (*tool*) perancangan sistem.
2. Memahami level-level yang ada pada Diagram Aliran Data
3. Mempelajari simbol-simbol dan teknik pembuatan Diagram Aliran Data.

B. Indikator

1. Praktikan mengetahui dan dapat menjelaskan tentang Diagram Aliran Data.
2. Praktikan dapat menggunakan Diagram Aliran Data untuk merancang sistem.
3. Praktikan memahami dan dapat menerapkan level-level yang ada pada Diagram Aliran Data.





C. MATERI

5.1 Definisi

DAD (Diagram Aliran Data) atau yang juga dikenal dengan sebutan **DFD** (Data Flow Diagram) merupakan alat perancangan sistem yang berorientasi pada aliran data dengan konsep dekomposisi. Merupakan representasi grafik aliran data yang terjadi di dalam sebuah system mulai dari input hingga output. DAD dapat digunakan baik untuk penggambaran analisa, dokumentasi, maupun untuk perancangan sistem. Pada umumnya, DAD digunakan untuk merancang sistem yang menggunakan penyimpanan data (data store) dalam mengelola informasi dalam sistem. Pada sebuah DFD, item data mengalir dari sebuah sumber data eksternal atau penyimpanan data internal ke penyimpanan data internal atau ke item eksternal, melalui satu atau lebih proses internal. Oleh karena DAD digunakan untuk merepresentasikan sebuah system atau perangkat lunak pada berbagai level abstraksi, dimungkinkan untuk mempartisi DAD menjadi beberapa level untuk menerangkan secara lebih detail fungsi dan peningkatan aliran informasi di dalam sistem.

Diagram Aliran Data dibangun dengan menggunakan simbol-simbol yang merepresentasikan entitas luar sistem/perangkat lunak, proses, aliran data, dan tempat penyimpanan data. Untuk lebih jelasnya dapat dilihat gambar 3.1.



Simbol	Keterangan
	Entitas Luar/Terminator
	Proses
	Aliran Data (Data Flow)
	Penyimpanan Data (Data Store)

Gambari 3.1 simbol-simbol DAD

Keterangan :

- ✓ **Entitas Luar:** kesatuan diluar sistem yang akan berinteraksi dengan sistem baik dalam bentuk pemberian input/informasi ke sistem, menerima output dari sistem, ataupun keduanya. Dapat berupa orang atau organisasi sumber informasi lain, penerima akhir dari suatu laporan, ataupun system lain.

Contoh :



- ✓ **Proses** adalah transformasi input menjadi output, merupakan kegiatan atau pekerjaan yang dilakukan oleh orang atau mesin komputer, dimana aliran data masuk ditransformasikan ke aliran data keluar. Sebuah proses harus memiliki sebuah nama yang mendeskripsikan secara representatif proses yang akan dilakukan pada proses tersebut dan diusahakan tidak terlalu panjang.

Contoh :



Ada beberapa hal yang perlu diperhatikan tentang proses :

1. Proses harus memiliki input dan output.
2. Proses dapat dihubungkan dengan komponen **entitas luar**, **data store (penyimpanan data)** atau **proses** melalui alur data.
3. Sistem/bagian/divisi/departemen yang sedang dianalisis oleh profesional sistem digambarkan dengan komponen proses.



1. **Aliran data/Arus data** digunakan untuk menjelaskan perpindahan data atau paket data dari satu bagian ke bagian lain. Arus data menghubungkan baik entitas luar, proses, ataupun penyimpanan data.
2. Aliran data dapat berbentuk sebagai berikut :
3. Formulir atau dokumen yang digunakan perusahaan
4. Laporan tercetak yang dihasilkan sistem
5. Output dilayar computer
6. Masukan untuk komputer
7. Komunikasi ucapan
8. Surat atau memo
9. Data yang dibaca atau direkam di file
10. Suatu isian yang dicatat pada buku agenda
11. Transmisi data dari suatu komputer ke komputer lain

Ada beberapa interkoneksi yang tidak boleh dihubungkan secara langsung melalui arus data yaitu :

- iii. Koneksi antara penyimpanan data dan penyimpanan data lainnya. Beberapa informasi pada sebuah penyimpanan data dimungkinkan secara independen untuk dikirimkan ke sebuah data penyimpanan lain. Secara praktikal proses ini harus melibatkan sebuah proses (tidak secara langsung).
- iv. Koneksi antara entitas luar dan sebuah penyimpanan data. Untuk dapat menulis atau membaca dari penyimpanan tetap diperlukan sebuah proses.

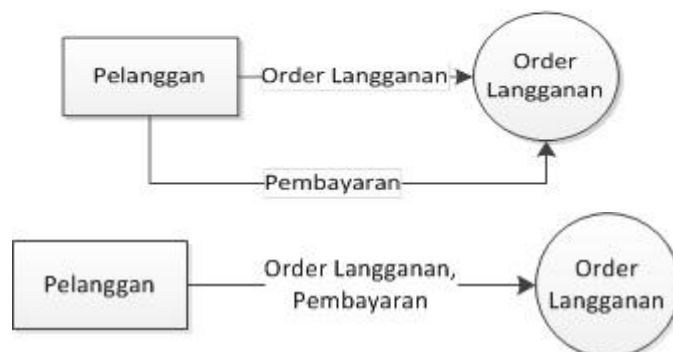
Penyimpanan data/data store

Data store adalah tempat dimana data disimpan, baik dapat secara temporary ataupun permanen. Jumlah data store yang disertakan pada DFD bergantung pada analisis data yang dilakukan pada tahap analisis kebutuhan dan rancangan penstrukturan data.

5.2 Konsep Arus Data :

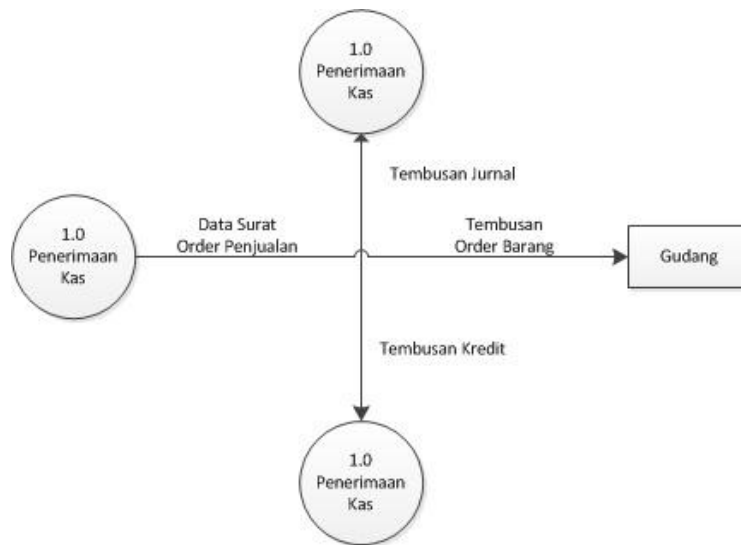
5.2.1 Packet of Data (Paket Data)

Bila dua data mengalir dari suatu sumber yang sama menuju ke tujuan yang sama, maka harus dianggap sebagai suatu arus data yang tunggal



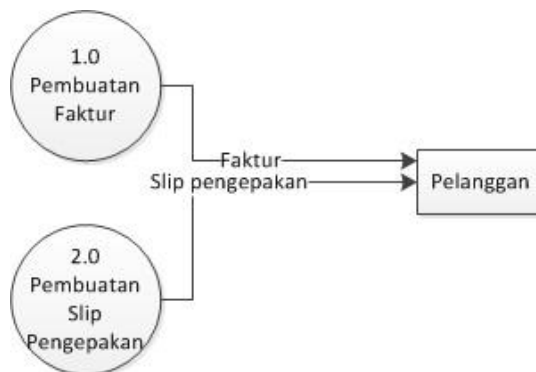
Diverging Data Flow (Arus Data Menyebar)

Arus data yang menyebar menunjukkan sejumlah tembusan dari arus data yang sama dari sumber sama ke tujuan berbeda.



Convergen Data Flow (Arus Data Mengumpul)

Arus data yang mengumpul yaitu Arus data yang berbeda dari sumber yang berbeda mengumpul ke tujuan yang sama.



Sumber dan Tujuan

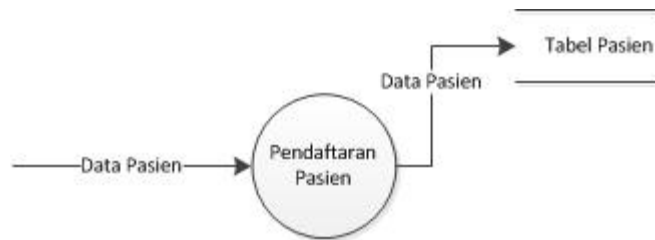
Arus data harus dihubungkan pada proses, baik dari maupun yang menuju proses.

1. **Data Storage** adalah tempat dimana data disimpan, baik dapat secara temporary ataupun permanen. Merupakan komponen untuk membuat model sekumpulan data, dapat berupa suatu file atau suatu sistem database dari suatu komputer, suatu arsip/dokumen, suatu agenda/buku. Jumlah data store yang disertakan pada DFD bergantung pada analisis data yang dilakukan pada tahap analisis kebutuhan dan rancangan penstrukturan data.

Yang perlu diperhatikan tentang data store :

- Alur data dari proses menuju data store, hal ini berarti data store berfungsi sebagai tujuan/tempat penyimpanan dari suatu proses (proses write).

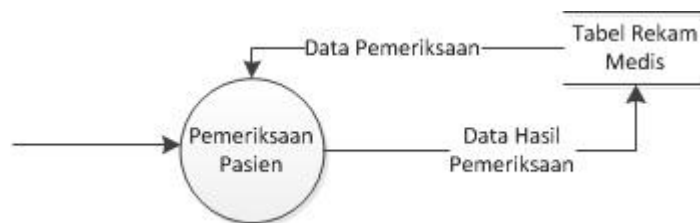




- Alur data dari data store ke proses, hal ini berarti data store berfungsi sebagai sumber/proses yang memerlukan data (proses read)



- Alur data dari proses menuju data store dan sebaliknya berarti berfungsi sebagai sumber dan tujuan.



Langkah-langkah Pembuatan DAD:

- ❖ Identifikasi semua kesatuan luar yang terlibat dengan sistem
- ❖ Identifikasi input dan output yang berhubungan dengan kesatuan luar
- ❖ Buatlah gambaran diagram konteksnya.

5.3 Jenis Diagram dalam DAD:

1. Diagram Konteks

Disebut juga diagram tingkat atas, merupakan diagram sistem yang menggambarkan aliran-aliran data yang masuk dan keluar dari sistem dan yang masuk dan keluar dari entitas luar. Diagram konteks merepresentasikan gambaran/batasan sistem, input apa saja yang diinputkan, serta informasi apa saja yang dihasilkan oleh sistem.

Hal yang harus diperhatikan:

- ✓ Memberikan gambaran tentang seluruh sistem
- ✓ Terminal yang memberikan masukan ke sistem disebut *source*
- ✓ Terminal yang menerima keluaran disebut *sink/destination*
- ✓ Hanya ada satu proses
- ✓ Tidak boleh ada data store

2. Diagram Level 1

Setelah pembuatan DAD Level Konteks, selanjutnya adalah pembuatan DAD Level 1, dimana pada DAD Level 1 adalah penggambaran dari Diagram Konteks yang lebih rinci (Overview Diagram) atau biasanya disebut sebagai **dekomposisi**. Hal yang harus diperhatikan:



- ❖ Perlihatkan data store yang digunakan
 - ❖ Setiap aliran data dari atau ke entitas eksternal yang ditampilkan pada diagram konteks tetap ditampilkan pada DAD Level1 (relevansi aliran data)
 - ❖ Setiap proses memiliki paling tidak satu aliran data input dan aliran data output
 - ❖ Pada proses yang tidak dirinci lagi, tambahkan tanda * pada akhir penomoran proses
 - ❖ Keseimbangan antara diagram konteks dan diagram level 1 harus dipelihara
3. Diagram Level 2
 4. Diagram Level 3, dst
 5. Penomoran Proses

Nama Level	Nama Diagram	Nomor Proses
0	Konteks	0
1	Diagram Level 1	1.0, 2.0, 3.0
2	Diagram Rinci 1.0	1.1, 1.2, 1.3
2	Diagram Rinci 2.0	2.1, 2.2, 2.3
2	Diagram Rinci 3.0	3.1, 3.2, 3.3
3	Diagram Rinci 1.1	1.1.1, 1.1.2, 1.1.3
3	Diagram Rinci 1.2	1.2.1, 1.2.2, 1.2.3
3	Diagram Rinci 1.3	1.3.1, 1.3.2, 1.3.2
Dst		

5.4 Penggambaran DAD

Tidak ada aturan baku untuk menggambarkan DFD, tapi dari berbagai referensi yang ada, secara garis besar:

5.3.1 Membuat Diagram Konteks

Diagram ini adalah diagram level tertinggi dari DAD yg menggambarkan hubungan sistem dengan lingkungan luarnya.

Cara:

1. Tentukan nama sistemnya. Misal, Sistem Informasi Rumah Sakit, Aplikasi Pengenalan Jenis Ikan, Aplikasi Pemilihan Jamur berkualitas dsb.
2. Tentukan batasan sistemnya.
Misalnya: kemampuan-kemampuan yg dimiliki system, fasilitas/layanan yang diberikan
3. Tentukan terminator/entitas luar apa saja yg ada dalam sistem.
Maksudnya : tentukan *user* yang menggunakan system tersebut
4. Tentukan apa yg diterima/diberikan terminator dari/ke sistem.
Maksudnya: tentukan data-data yang digunakan oleh sistem
5. Gambarkan diagram konteks.



5.3.2 Buat Diagram Level 1

Diagram ini adalah dekomposisi dari diagram Context.

Cara:

1. Tentukan proses utama yang ada pada sistem.
2. Tentukan apa yang diberikan/diterima masing-masing proses ke/dari sistem sambil memperhatikan konsep keseimbangan (alur data yg keluar/masuk dari suatu level harus sama dengan alur data yg masuk/keluar pada level berikutnya)
3. Apabila diperlukan, munculkan data store sebagai sumber maupun tujuan alur data.
4. Gambarkan diagram level satu.
5. Hindari perpotongan arus data.
6. Beri nomor pada proses utama (nomor tidak menunjukkan urutan proses). Misal. 1.0, 2.0, 3.0 dst

5.3.3 Buat Diagram Level 2

Diagram ini merupakan dekomposisi dari diagram level satu.

Cara:

1. Tentukan proses yang lebih kecil (sub-proses) dari proses utama yang ada di level satu.
2. Tentukan apa yang diberikan/diterima masing-masing sub-proses pada/dari sistem dan perhatikan konsep keseimbangan.
3. Apabila diperlukan, munculkan data store (transaksi) sebagai sumber maupun tujuan alur data.
 - Gambarkan DFD level Dua
 - Hindari perpotongan arus data.
 - Beri nomor pada masing-masing sub-proses yang menunjukkan dekomposisi dari proses sebelumnya. Contoh : 1.1, 1.2, 1.3 dst

5.3.4 DFD Level Tiga, Empat, dst

Diagram ini merupakan dekomposisi dari level sebelumnya. Proses dekomposisi dilakukan sampai dengan proses siap dituangkan ke dalam program. Aturan yang digunakan sama dengan level dua.

Pengembangan Diagram Alir Data

Bagaimana mengembangkan Diagram Alir Data. Berikut ini beberapa aktivitas yang dibutuhkan untuk pengembangan DAD:

- Mengidentifikasi aliran data kunci
- Mengidentifikasi semua entitas eksternal
- Mengidentifikasi semua fungsi
- Mengidentifikasi semua jalur aliran data
- Mengidentifikasi batas sistem
- Mengidentifikasi semua proses
- Mengidentifikasi semua data store
- Mengidentifikasi antara proses dan data store
- Pada Level yang lebih rendah, mengisi tiap detail proses.



D. SOAL

Tugas:

1. Sebutkan pengertian DFD dan tujuan pembuatan DFD.
2. Apa perbedaan antara diagram konteks dengan diagram di level bawahnya dan apa saja yang saja yang menjadi perhatian penting ketika membuat DFD sebuah sistem.
3. Pemilik sebuah toko penyewaan buku skala kecil ingin membuat sebuah sistem/software yang nantinya akan membantu mengefisienkan proses bisnis pada toko tersebut, karena metode transaksi yang berjalan pada toko tersebut selama ini adalah transaksi pembukuan manual, dimana dirasa kurang efisien. Sistem yang akan dibangun tersebut nantinya akan digunakan untuk :
 1. Memproses transaksi penyewaan
 2. Menyimpan data-data buku berdasarkan kategori buku
 3. Menyimpan data penyewa
 4. Menyimpan data-data transaksi penyewaan.
 5. Menyampaikan laporan keuangan dalam skala waktu tertentu

Penguna yang akan bersentuhan langsung dengan perangkat lunak ini adalah penjaga dan pemilik toko.

Prosedur penyewaan yang berjalan pada toko tersebut yaitu :

Untuk dapat meminjam buku pengguna harus menjadi anggota terlebih dahulu. Untuk dapat menjadi seorang anggota, petugas toko akan mecalon anggota penyewaan buku mengisi lembar identitas dan memberikannya kepada petugas agar petugas dapat menyimpan data peminjam di komputer.

Setiap kali akan meminjam buku, seorang peminjam diminta untuk memberikan data buku yang akan dipinjam dan kartu pengenalan (SIM/KTP/KTM). Petugas akan memeriksa apakah data tersebut sudah ada pada arsip anggota. Jika tidak ada, petugas akan meminta calon peminjam mengisi formulir keanggotaan. Namun, jika data peminjam ada pada arsip maka transaksi peminjaman akan langsung diproses.

Petugas akan mencari arsip buku.

- ❖ Jika buku tidak ada atau sedang dipinjam maka petugas akan memberitahukan status kosong ke anggota
- ❖ Jika ada maka petugas akan membuat nota peminjaman dan memberikannya ke anggota

Lama waktu peminjaman buku ditentukan oleh banyaknya buku yang dipinjam dan pada toko tersebut diberlakukan sistem denda akibat adanya keterlambatan pengembalian.

E. TUGAS BESAR BAGIAN 1

Buatlah sistem non sistem informasi sesuai dengan yang Anda inginkan. Buatlah dengan menggunakan langkah-langkah pembuatan *software* yang pernah dijelaskan pada pertemuan sebelumnya (mulai dari analisis sistem, bisnis proses, aturan bisnis, perancangan DFD level 0-level 3, dan perancangan antarmuka)! Pada tahap ini, Anda harus menjabarkan ide Anda seperti mengembangkan Bab IV di skripsi. Presentasikan hasil kerja Anda!



PERTEMUAN KE 6

MATERI : UML (UNIFIED MODELLING LANGUAGE)

A. Tujuan Praktikum:

- b. Mengenalkan Pengertian UML (Unified Modelling Language)
- c. Menjelaskan Konsep Dasar UML
- d. Menjelaskan konsep objek

B. Indikator:

- ✓ Agar praktikan bisa memahami pengertian UML beserta kegunaannya
- ✓ Agar praktikan memahami prinsip pemodelan berorientasi objek
- ✓ Agar praktikan bisa menjelaskan Konsep Dasar UML dan berbagai macam jenis diagram UML

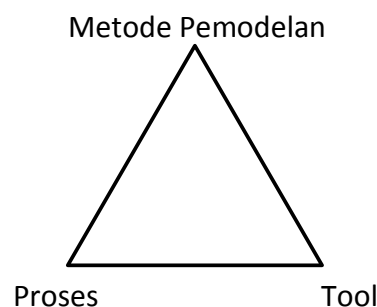
C. Materi

5.1 Pendahuluan

Teknologi perancangan perangkat lunak telah berkembang pesat, dari perancangan struktur prosedural menuju perancangan berorientasi objek yang didukung oleh perkembangan pemodelan visual dengan cara mengimplementasikan pemrograman berbasis/berorientasi-objek. Meyer (1997) menyatakan bahwa pemodelan berorientasi objek merupakan suatu komplemen dan dalam banyak kasus telah menggantikan kedudukan pemodelan terstruktur sebagai teknologi tinggi yang bagus ("good").

Dalam suatu proses pengembangan software, analisa dan rancangan telah merupakan terminologi yang sangat tua. Pada saat masalah diidentifikasi, dapat dikatakan kita berada pada tahap perancangan. Merancang adalah menemukan suatu cara untuk menyelesaikan masalah. Salah satu tool / model untuk merancang pengembangan perangkat lunak yang berorientasi objek adalah UML.

Kesuksesan suatu pemodelan perangkat lunak ditentukan oleh tiga unsur, yang kemudian terkenal dengan sebuah segitiga sukses (*the triangle for success*). Ketiga unsure tersebut adalah metode pemodelan (*notation*), proses (*process*) dan *tool* yang digunakan.



5.2 Konsep Objek

Obyek dalam analisis dan perancangan perangkat lunak merupakan suatu konsep (*concept*), benda (*thing*). Secara sederhana, obyek adalah mobil, manusia, alarm. Namun, obyek dapat pula merupakan sesuatu yang abstrak yang ada didalam sistem, seperti: table; database, event; aktivitas.

Obyek dapat dikenali dari keadaan maupun juga aktivitasnya. Alasan mengapa saat ini pendekatan dalam pengembangan perangkat lunak dengan menggunakan konsep object-oriented adalah:

1. *Scalability*

Artinya, konsep *object-oriented* lebih mudah dipakai untuk menggambarkan sistem yang besar dan kompleks.

2. *Dynamic modeling*,

Artinya, konsep object-oriented dapat digunakan untuk pemodelan sistem dinamis dan *real time*.

5.3 Teknik Dasar OOA/D (Object-Oriented Analysis/Design)

Dalam pemodelan, meskipun metodologi implementasi obyek terikat pada aturan standar, namun teknik pemilihan obyek tidak dapat dipisahkan pada subyektifitas analis perangkat lunak dan desainer. Beberapa obyek bisa saja akan diabaikan, dan obyek lain yang menjadi perhatian akan diimplementasikan ke dalam sistem. Menurut Alhir (2002), terdapat 4 konsep dasar dalam OOA/D, yaitu:

1. *Abstraction*

Abstraction/abstraksi merupakan konsep atau ide. Sedangkan teknik yang digunakan untuk mengidentifikasi suatu abstraksi dinamakan *abstracting*. Suatu abstraksi dapat berupa konsep suatu hubungan yang memiliki hubungan dengan konsep lainnya.

2. *Encapsulation*

Encapsulation/enkapsulasi melibatkan pengelompokan ide berdasarkan prinsip-prinsip lokalisasi, lalu menyembunyikan informasi yang ada. Dengan kata lain, konsep enkapsulasi merupakan suatu tindakan menyembunyian informasi.

3. *Generalization*

Generalization/generalisasi melibatkan pengelompokan dari ide-ide berdasarkan persamaan dan perbedaan yang dimiliki.

4. *Polymorphisme*.

Polymorphisme melibatkan hubungan dari kumpulan ide yang memberikan spesifikasi, namun memiliki ciri khas yang unik maupun implementasi yang spesifik. Dengan kata lain, setiap obyek mempunyai ruang lingkup berupa kelas, atribut, dan metode yang dibatasi.

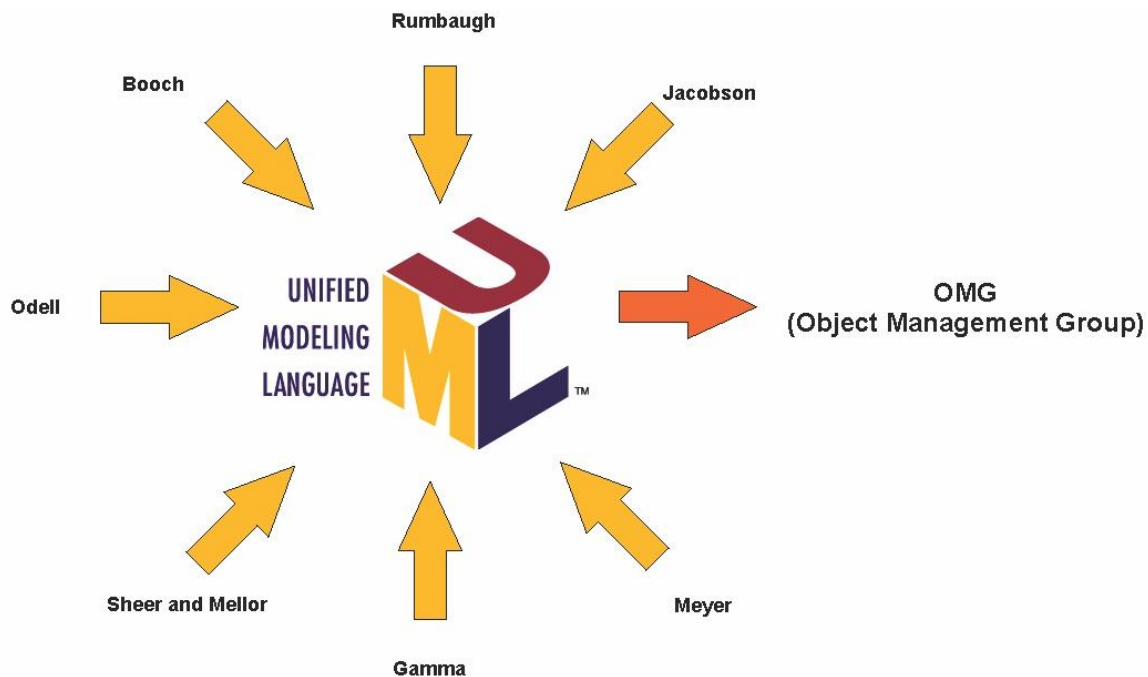


Pengertian

UML (*Unified Modeling Language*) adalah sebuah bahasa yang berdasarkan grafik/gambar untuk memvisualisasi, menspesifikasi, membangun, dan mendokumentasikan suatu sistem pengembangan *software* berbasis obyek. UML sendiri juga memberikan standar penulisan sebuah *blue print* suatu sistem, yang meliputi konsep bisnis proses, penulisan kelas-kelas dalam bahasa pemrograman yang spesifik, skema database, dan komponen-komponen yang diperlukan dalam sistem *software* (<http://www.omg.org>). UML menawarkan sebuah standar untuk merancang model sebuah sistem.

UML tidak hanya merupakan sebuah bahasa pemrograman visual saja, namun juga dapat dihubungkan secara langsung ke berbagai bahasa pemrograman, seperti JAVA, C++, Visual Basic, atau bahkan dihubungkan secara langsung ke dalam sebuah object-oriented database.

UML adalah standar dunia yang dibuat oleh *Object Management Group* (OMG), sebuah badan yang bertugas mengeluarkan standar-standar teknologi object-oriented dan software component.



5.4 Konsep Dasar UML

UML 2.x menyediakan 13 tipe diagram, yang dibagi menjadi 3 kategori, yang terdiri dari, 6 tipe diagram yang digunakan untuk struktur aplikasi statis; 3 tipe diagram yang digunakan untuk tipe umum suatu behavior; dan 4 tipe diagram yang digunakan untuk interaksi aspek yang berbeda. Dari berbagai penjelasan yang terdapat di dokumen dan buku-buku UML, sebenarnya konsepsi dasar UML bisa kita rangkumkan dalam tabel dibawah.



Kategori Diagram	Diagram
<i>Structure Diagrams</i>	Class Diagram, Object Diagram, Component Diagram, Composite Structure diagram, Package Diagram, Deployment Diagram
<i>Behavior Diagrams</i>	Use case Diagram, Activity Diagram, State Machine Diagram
<i>Interaction Diagrams</i>	Sequence Diagram, Communication Diagram, Timing Diagram, Interaction Overview Diagram

Keunggulan UML

❖ *Uniformity*

Pengembang cukup menggunakan satu metodologi dari tahap analisis hingga perancangan. Memungkinkan merancang komponen antar muka secara terintegrasi bersama perancangan PL dan perancangan struktur data

❖ *Understandability*

Kode yang dihasilkan dapat dikelolamenjadi kelas-kelas yangberhubungan dengan masalah sesungguhnya sehingga lebih mudah untuk dipahami.

❖ *Stability*

Kode program yang dihasilkan relatif stabil sepanjang waktu, karena mendekati permasalahan yang sesungguhnya.

❖ *Reusability*

Dengan metodologi berorientasi objek, dimungkinkan penggunaan ulang kode, sehingga pada akhirnya akan sangat mempercepat waktu pengembangan perangkat lunak (atau sistem informasi).

Seperti yang telah tercantum di halaman sebelumnya, beberapa macam diagram UML antara lain adalah:

1. Use case diagram
2. Class diagram
3. Activity diagram
4. Statechart diagram
5. Sequence diagram
6. Communication diagram
7. Object diagram
8. Component diagram
9. Timing diagram
10. Deployment diagram
11. Interaction overview diagram
12. Package diagram
13. Composite structure diagram



D. SOAL

Tugas:

1. Sebut dan jelaskan perbedaan pemrograman berorientasi objek dengan pemrograman procedural!
2. Ilustrasikan keempat konsep dasar dalam Object Oriented Analysis/Design!
3. Sebut dan jelaskan kelebihan pemodelan perangkat lunak menggunakan UML jika dibandingkan dengan menggunakan Diagram Aliran Data (DAD)!



PERETEMUAN KE 7

MATERI : MACAM-MACAM DIAGRAM UML

A. TujuanPraktikum :

1. Mengetahui contoh-contoh diagramUML beserta penjelasannya
2. Menjabarkan langkah-langkah membuat 3 diagram yang paling sering digunakan dalam object oriented design (*use case*, *activity diagram*, *class diagram*)
3. Mengidentifikasi diagram yang sesuai berdasarkan permasalahan yang dihadapi

B. Indikator:

1. Praktikan bisa menjelaskan pengertian masing-masing jenis diagram dalam UML
2. Praktikan bisa mengimplementasikan diagram dalam UML berdasarkan permasalahan yang dihadapi

C. Materi

6.1 Use Case Diagram

Use case diagram adalah diagram deskripsi fungsi dari sebuah sistem dari perspektif/sudut pandang para pengguna sistem. *Use case* mendefinisikan "apa" yang dilakukan oleh sistem dan elemen-elemennya, bukan "bagaimana" sistem dan elemen-elemennya saling berinteraksi (Alhir, 2002). *Use case* bekerja dengan menggunakan *scenario*, yaitu deskripsi urutan-urutan langkah yang menerangkan apa yang dilakukan penggunaan terhadap sistem maupun sebaliknya.

Use case diagram mengidentifikasi fungsionalitas yang dimiliki oleh sistem (*use-case*), user yang berinteraksi dengan sistem (*aktor*), dan asosiasi/keterhubungan antara user dengan fungsionalitas sistem. Diagram *use case* sangat membantu bila kita sedang menyusun kebutuhan dalam suatu sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua keunggulan yang ada pada sistem.

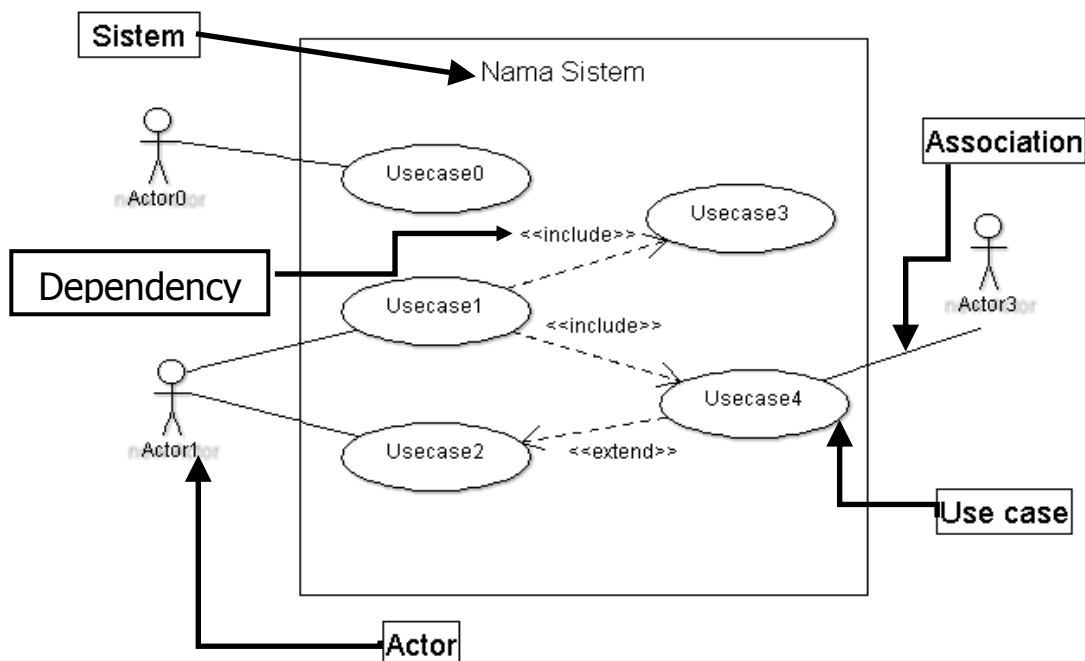
Sebuah *use case* dapat menyertakan (*include*) fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang umum (*common*). Sebuah *use case* juga dapat menambahkan langkah atau fungsionalitas tambahan dengan cara melakukan *extend use case* lain.

Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

Use Case Diagram menggambarkan interaksi antara actor dengan proses atau sistem yang dibuat. *Use Case Diagram* mempunyai beberapa bagian penting seperti: *Actor*, *Use Case*, *Unidirectional Association*, dan *Generalization*.



Komponen Use Case



1. Sistem

Menyatakan batasan sistem dalam relasi dengan aktor-aktor yang menggunakannya (di luar sistem) dan fitur-fitur yang harus disediakan (dalam sistem). Sistem Digambarkan dengan segi empat yang membatasi semua use case dalam sistem terhadap pihak mana sistem akan berinteraksi. Sistem disertai label yang menyebutkan nama dari sistem, tapi umumnya tidak digambarkan karena tidak terlalu memberi arti tambahan pada diagram.

2. Actor

Aktor adalah pengguna sistem. Aktor tidak terbatas hanya manusia saja, jika sebuah sistem berkomunikasi dengan aplikasi lain dan membutuhkan input atau memberikan output, maka aplikasi tersebut juga bisa dianggap sebagai aktor. Aktor Bisa berupa manusia, sistem, atau device yang memiliki peranan dalam keberhasilan operasi dari sistem. Seluruh entitas di luar perangkat lunak/aplikasi yang berinteraksi dengan sistem dapat dipandang sebagai aktor.

3. Use case

Mengidentifikasi tugas utama sistem. Tanpa tugas, sistem tidak akan memenuhi permintaan user/aktor. Setiap use case mengekspresikan tujuan/goal dari sistem yang harus dicapai. Use case diberi nama sesuai dengan goal-nya dan digambarkan dengan elips dengan nama di dalamnya. Fokus tetap pada goal bukan bagaimana mengimplementasikannya walaupun use case berimplikasi pada prosesnya nanti.

4. Association

Mengidentifikasi adanya interaksi antara setiap aktor/pengguna tertentu dengan use case tertentu. Digambarkan sebagai garis antara aktor terhadap use case yang bersangkutan.



5. Stereotype

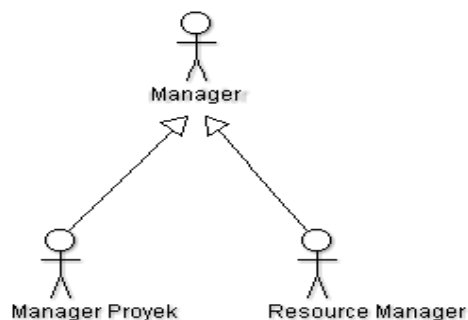
Stereotype adalah sebuah model khusus yang terbatas untuk kondisi tertentu. Untuk menunjukkan *stereotype* digunakan symbol "<<" diawalnya dan ditutup dengan ">>" diakhirnya. Menyediakan informasi lebih banyak mengenai peranan dari elemen tanpa menyebutkan implementasinya. *Stereotype* digunakan untuk menggambarkan:

- ❖ Use case Dependency
- ❖ Class-class
- ❖ Package-package
- ❖ Classifier

6. Dependency

- ❖ Dependency <<include>>
 - ✓ Menggambarkan hubungan antar dua use case di mana yang satu memanggil yang lain.
 - ✓ Jika pada beberapa use case terdapat bagian yang memiliki aktivitas yang sama, maka bagian aktivitas tersebut biasanya dijadikan use case tersendiri dengan relasi dependensi setiap use case semula ke use case yang baru.
 - ✓ Digambarkan dengan garis putus-putus bermata panah dengan notasi <<include>> pada garis.
 - ✓ Arah mata panah sesuai dengan arah pemanggilan.
 - ✓ Use case yang di-include tidak dapat berdiri sendiri dan use case yang melakukan *include* tidak dapat berjalan dengan sempurna tanpa use case yang dipanggil.
- ❖ Dependency <<extend>>
 - ✓ Jika pemanggilan memerlukan adanya kondisi tertentu maka berlaku dependensi <<extend>>
 - ✓ Digambarkan serupa dengan dependensi <<include>> namun arah panah berlawanan.
 - ✓ Use case yang melakukan extend tidak selalu menjalankan fungsionalitas use case yang di-extend. Contohnya peringatan pin salah pada ATM tidak akan muncul jika pin yang dimasukkan benar, jadi use case transaksi tidak selalu menjalankan fungsi penanganan pin salah (use case pin salah).
- ❖ Generalization
 - ✓ Mendefinisikan relasi antara dua actor atau dua use case apabila salah satu actor/use case mewarisi sifat dari yang lainnya.

Contoh:



Arah panah relasi pada use case mengarah pada *use case* yang lebih besar kontrolnya atau yang dipakai.



Petunjuk untuk menyusun diagram use case

- ✓ Tentukan target sistem.
- ✓ Identifikasi semua actor.
- ✓ Identifikasi semua use case.
- ✓ Definisikan asosiasi antara setiap actor dan setiap use case.
- ✓ Cek setiap actor dan setiap use case untuk mendapatkan kemungkinan adanya perbaikan.
- ✓ Cek setiap use case untuk dependensi <<include>>.
- ✓ Cek setiap use case untuk dependensi <<extend>>.
- ✓ Evaluasi setiap actor dan setiap use case untuk generalisasi.

D. SOAL

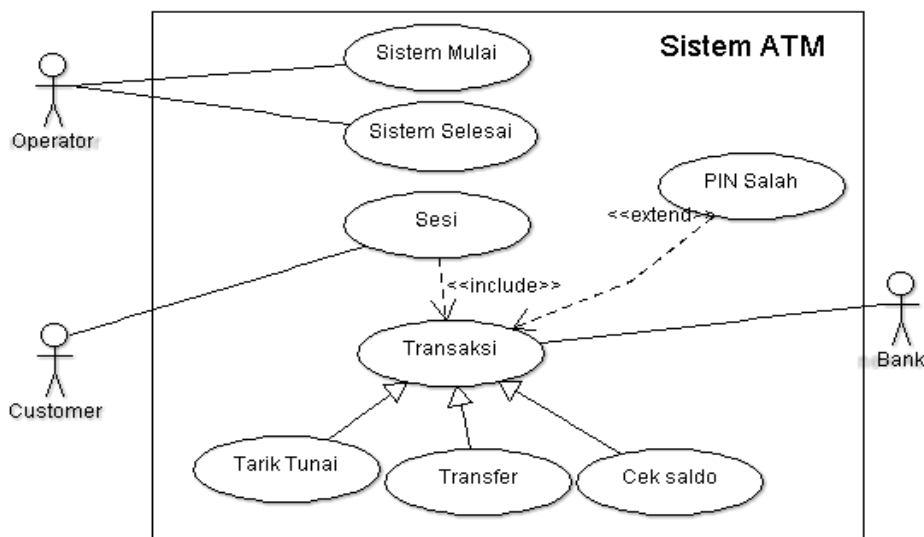
Contoh kasus: Sistem ATM Bank

Sistem ATM yang akan dibuat adalah sistem ATM sederhana dengan fungsi paling sederhana yaitu: mengambil uang tunai, melakukan transfer ke rekening tertentu pada bank yang sama, dan memeriksa saldo. Sistem yang akan dimodelkan tidak mencakup beberapa fungsi ATM yang bergantung fasilitas tambahan dari bank, misalnya membayar kartu kredit.

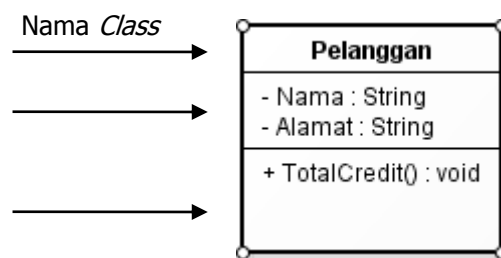
1. Identifikasi actor yang terlibat: operator, customer, dan bank
2. Identifikasi use case yang digambarkan dalam system:
 - Use case Sistem Mulai
 - Use case Sistem Selesai
 - Use case Sesi
 - Use case Transaksi
 - Use case Transaksi Tarik Tunai
 - Use case Transaksi Transfer
 - Use case Cek Saldo
 - Use case PIN Salah
3. Kelompokkan use case yang telah diidentifikasi berdasarkan skenario :
 - Use case utama: Mulai, Sesi, Selesai, Transaksi
 - Use case transaksi: terbagi menjadi 3 tipe dengan tambahan sifat spesifik yang mewarisi sifat dari use case transaksi, yaitu: Tarik Tunai, Transfer, dan Cek Saldo.
 - Perluasan use case utama yang merupakan perilaku terhadap kejadian khusus (*extend*) : PIN salah



Hasil akhir dari *use case* yang dirancang adalah:



Classdiagram



Class merupakan spesifikasi yang apabila class tersebut dipanggil akan menghasilkan sebuah objek. *Class* merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) dari suatu sistem, sekaligus memberikan layanan untuk mengorganisasi/manipulasi keadaan (metode/fungsi) tersebut.

Dalam pemodelan statis dari sebuah sistem, diagram kelas biasanya digunakan untuk memodelkan salah satu dari tiga hal di bawah ini:

1. Perbendaharaan dari sistem
1. Kolaborasi
1. Skema *logical database*

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

1. Nama, merupakan nama dari sebuah *class*.
2. Atribut, merupakan properti dari sebuah *class*. Atribut melambangkan batas nilai yang mungkin ada pada objek dari *class*.
3. Operasi, adalah sesuatu yang bisa dilakukan oleh sebuah class atau yang dapat dilakukan oleh class lain terhadap sebuah class.

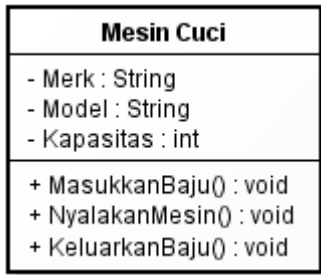


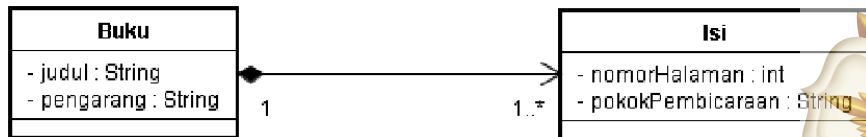



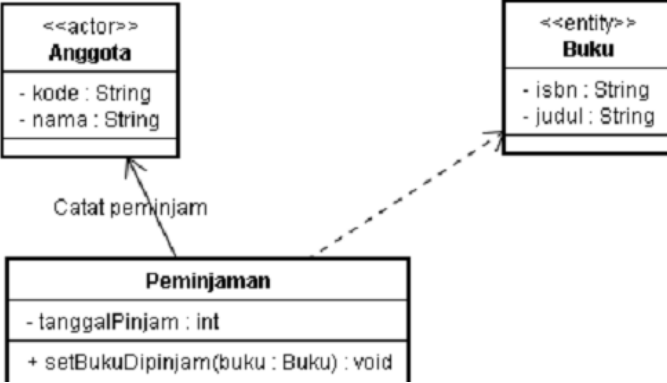



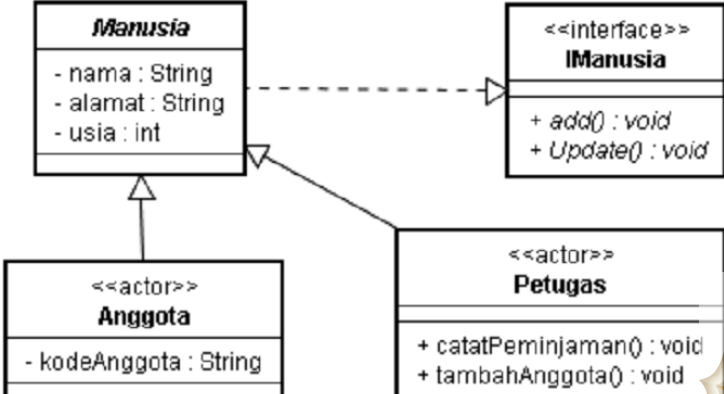
Atribut dan metoda dapat memiliki salah satu sifat berikut:

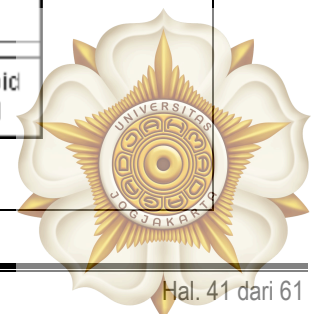
- ✓ *Private*, tidak dapat dipanggil dari luar class yang bersangkutan.
- ✓ *Protected*, hanya dapat dipanggil oleh class yang bersangkutan dan anak-anak yang mewarisinya.
- ✓ *Public*, dapat dipanggil oleh siapa saja.
- ✓ *Package*, hanya dapat dipanggil oleh instance sebuah class pada paket yang sama.

Class diagram dilambangkan dengan notasi-notasi sebagai berikut:

Notasi dalam Class Diagram

Notasi	Gambar	Keterangan
Class		Class adalah blok-blok pembangun pada pemrograman berorientasi obyek. Sebuah class digambarkan sebagai sebuah kotak yang terbagi atas 3 bagian. Bagian atas adalah bagian nama dari class. Bagian tengah mendefinisikan property/atribut class. Bagian akhir mendefinisikan method-method dari sebuah class.
Assosiation	<p>Sebuah asosiasi merupakan sebuah <i>relation ship</i> paling umum antara 2 class, dan dilambangkan oleh sebuah garis yang menghubungkan antara 2 class. Garis ini bisa melambangkan tipe-tipe <i>relation ship</i> dan jugadapat menampilkan hukum-hukum multiplisitas pada sebuah <i>relationship</i> (Contoh: <i>One-to-one</i>, <i>one-to-many</i>, <i>many-to-many</i>).</p> <p>Simbol: <u>1..n Owned by 1</u></p> <p>Contoh:</p>  <p>Directional Association atau Relasi 1 arah antara Class Petugas dan Anggota</p>	
Composition	<p>Jika sebuah class tidak bias berdiri sendiri dan harus merupakan bagian dari class yang lain, maka class tersebut memiliki relasi <i>Composition</i> terhadap class tempat dia bergantung tersebut. Sebuah <i>relationship composition</i> digambarkan sebagai garis dengan ujung berbentuk jajaran genjang berisi/solid.</p> <p>Simbol: </p> <p>Contoh:</p>  <p>Relasi Composition antara buku dan isi</p>	

<p><i>Dependency</i></p>	<p>Kadang kala sebuah <i>class</i> menggunakan <i>class</i> yang lain. Hal ini disebut <i>dependency</i>. Umumnya penggunaan <i>dependency</i> digunakan untuk menunjukkan operasi pada suatu <i>class</i> yang menggunakan <i>class</i> yang lain. Sebuah <i>dependency</i> dilambangkan sebagai sebuah panah bertitik-titik.</p> <p>Simbol: </p> <p>Contoh:</p>  <p>Relasi <i>Dependency</i> antara Class Peminjaman dan Buku</p>
<p><i>Aggregation</i></p>	<p><i>Aggregation</i> mengindikasikan keseluruhan bagian <i>relationship</i> dan biasanya disebut sebagai relasi "mempunyai sebuah" atau "bagian dari". Sebuah <i>aggregation</i> digambarkan sebagai sebuah garis dengan sebuah jajaran genjang yang tidak berisi/tidak solid.</p> <p>Simbol: </p> <p>Contoh:</p>  <p>Relasi <i>Aggregation</i> antara buku dan daftar pustaka</p>
<p><i>Generalization</i></p>	<p>Sebuah relasi <i>generalization</i> sepadan dengan sebuah relasi <i>inheritance</i> pada konsep berorientasi obyek. Sebuah <i>generalization</i> dilambangkan dengan sebuah panah dengan kepala panah yang tidak solid yang mengarah ke kelas "parent"-nya/induknya.</p> <p>Simbol: </p> <p>Contoh:</p>  <p>Generalization dari kelas manusia</p>



Kelas dapat merupakan implementasi dari sebuah *interface*, yaitu class abstrak yang hanya memiliki metode. *Interface* tidak dapat langsung diwujudkan, tetapi harus diubah dahulu menjadi sebuah class.

Petunjuk Mendefinisikan Class

Untuk mendefinisikan kelas dari suatu sistem, cara yang cukup mudah adalah memulainya dari model *use case* yang sudah dibuat dengan cara menetapkan objek apa saja yang dibutuhkan di sistem. Objek dapat berupa orang, benda, kejadian, tempat, dll. Setiap objek mempunyai atribut-atribut, serta method/operasi-nya.

Membuat Class Diagram

Misalkan terdapat deskripsi seperti dibawah ini:

Class buku memiliki beberapa atribut yang terdiri dari :

- ✓ Judul Buku
- ✓ Pengarang

Class buku memiliki method:

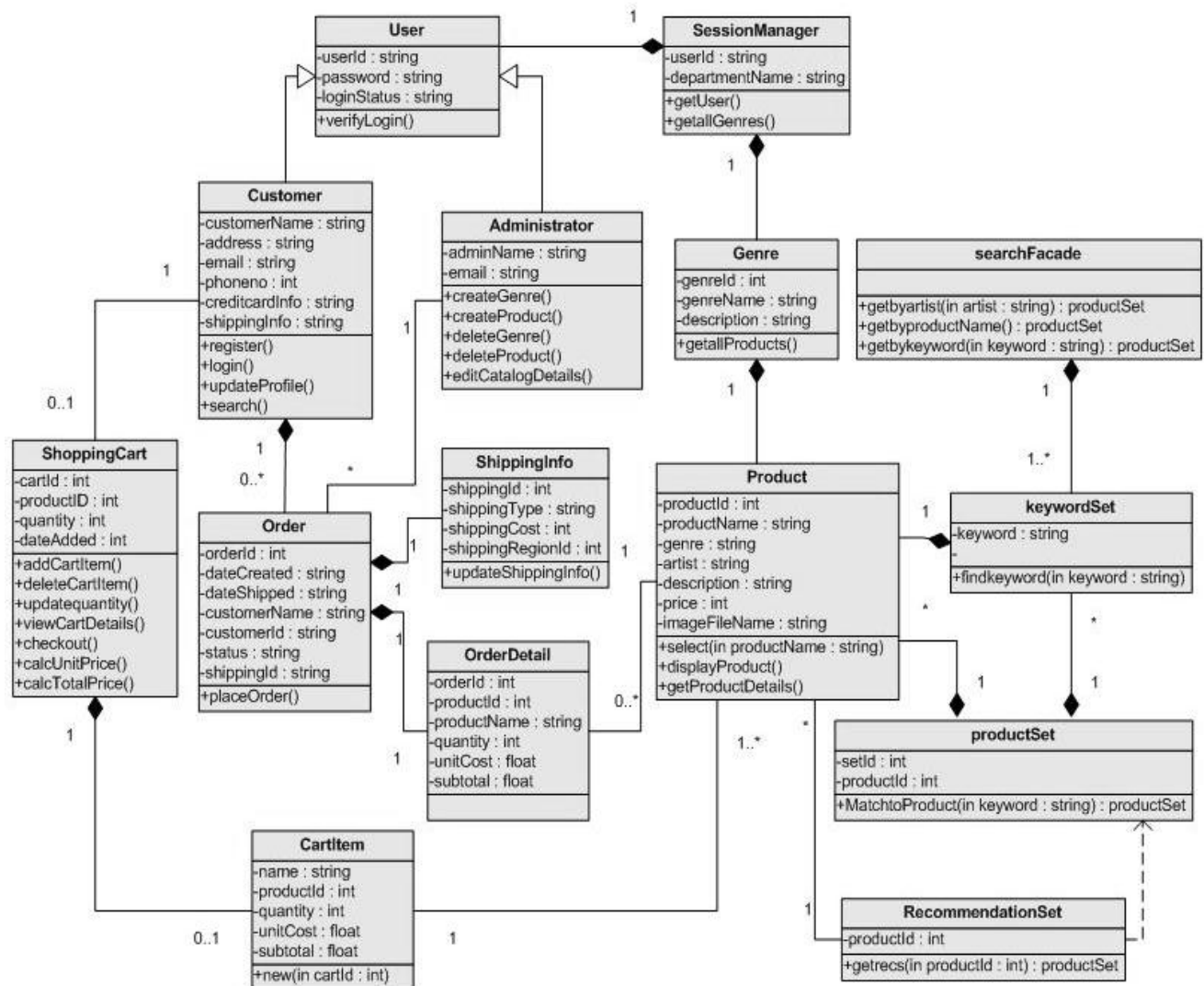
- ✓ `setJenis()`
- ✓ `ambilHalaman()`

Penggambaran dari *class diagram* adalah

Buku
- judul : String - pengarang : String
+ setJenis(String tipe : int) : void + ambilHalaman() : int



Contoh Class Diagram:



Contoh Implementasi Class Diagram

Buku
+judul : String +pengarang : String +idJenis : byte
+ambilJenis() +ambilHalaman() : int



Implementasi class diagram buku ke dalam program

```
Buku.java
package buku;
public class Buku {
    public String judul;
    public String pengarang;
    public byte idJenis;

    public void ambilJenis() {
        switch(idJenis) {
            case 1: {
                System.out.println("Pemrograman Komputer");
                break;
            }
            case 2: {
                System.out.println("Aplikasi Terapan");
                break;
            }
        }
    }

    public int ambilHalaman() {
        if(judul=="Pemrograman Java") {
            return 100;
        }
        else {
            return 0;
        }
    }

    public static void main(String[] args) {
        Buku Java = new Buku();
        Java.judul = "Pemrograman Java";
        Java.pengarang = "Stevani Al-Manahe";
        Java.idJenis = 1;
        Java.ambilJenis();
        System.out.print(Java.ambilHalaman());
    }
}
```

Contoh 2

Contoh class diagram dengan konstruktor

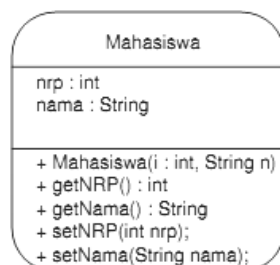
mahasiswa
npm : String nama : String
+mahasiswa(String newNpm, String newNama) +getNpm() : String +getNama() : String



Implementasi class diagram Mahasiswa ke dalam program

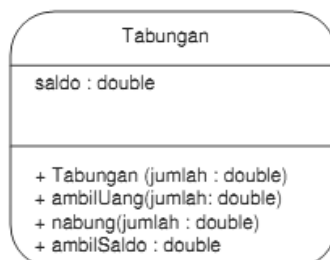
```
Mahasiswa.java X
public class Mahasiswa {
    String npm;
    String nama;
    public Mahasiswa(String newNpm, String newNama) {
        this.npm = newNpm;
        this.nama = newNama;
    }
    public String getNpm() {
        return npm;
    }
    public String getNama() {
        return nama;
    }
    public static void main(String[] args) {
        Mahasiswa siswa = new Mahasiswa("7006301051", "Stevani Al-Manahe");
        System.out.println("NPM " + siswa.getNpm());
        System.out.println("Nama " + siswa.getNama());
    }
}
```

Tugas 1:Implementasikan Perluasan Class Diagram Mahasiswa seperti berikut :



Buat program untuk menguji Class yang telah dibuat!

Tugas 2:Implementasikan Class Diagram Tabungan seperti berikut :



Buat program untuk menguji Class yang telah dibuat.



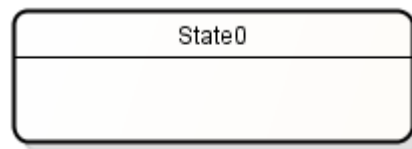
6.2 Statechart Diagram

Use case dan skenario memberikancara untuk mendeskripsikan kinerja sistem, yakni interaksi antara objek-objek di dalam sistem. 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.

Notasi yang digunakan pada state transition diagram

State

Lambang:



State adalah sebuah kondisi selama ketika objek berada dalam kondisi tertentu, melakukan beberapa aksi, atau menunggu sebuah *event*. State dari suatu objek dapat dibedakan oleh nilai dari satu atau lebih atribut-atribut dari *class*. State dari suatu objek ditentukan dengan pengujian/pemeriksaan atribut-atribut dan hubungan-hubungan dari objek. Pada umumnya *statechart diagram* menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu *state chart diagram*).

State Transitions

State transition diagram meliputi seluruh pesan dari objek yang dapat mengirim dan menerima. Suatu skenario merepresentasikan satu jalur yang melewati sebuah *state transition diagram*. Rentang waktu antara dua pesan yang dikirim oleh sebuah objek merepresentasikan sebuah *state*.

Lambang:



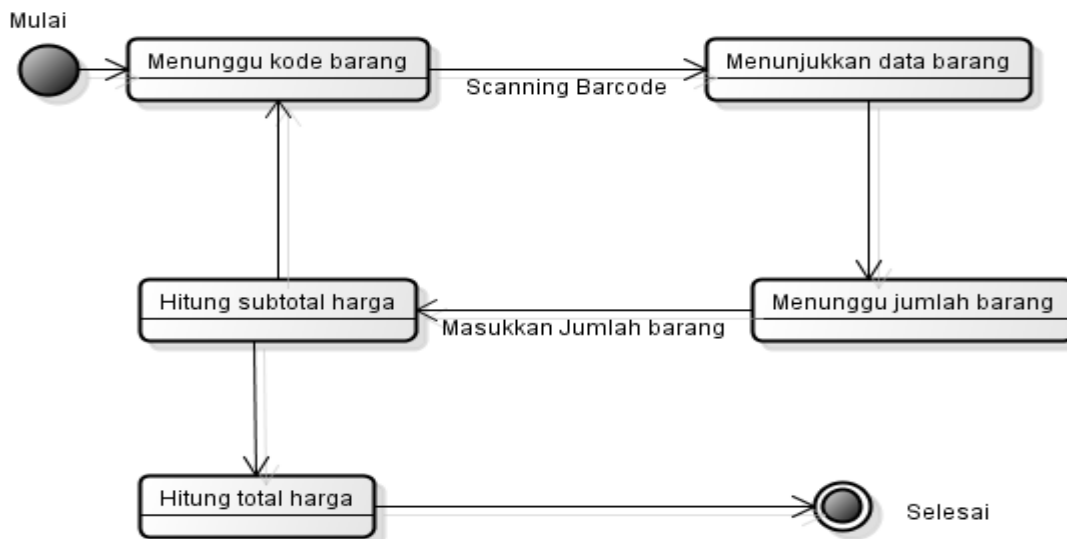
Special States

Terdapat dua *state* khusus yang ditambahkan di *state transition diagram*, yaitu *start state* dan *stop state*. Tiap-tiap diagram harus memiliki satu, dan hanya satu *start state* ketika obyek dibuat. Sedangkan untuk *stop state*, sebuah object boleh memiliki banyak *stop state*. Berikut ini merupakan gambar notasi *start state* (gambar kiri) dan *stop state* (gambar kanan).



Implementasi Start Transition Diagram

Contoh kasus: mesin kasir




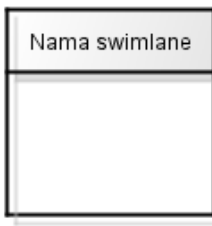


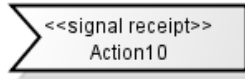
6.3 Activity Diagram

Diagram aktivitas (*activity diagram*) digunakan untuk mendokumentasikan alur kerja pada sebuah sistem yang dimulai dari pandangan level bisnis hingga ke level operasional. Diagram aktivitas mempunyai peran seperti halnya *flowchart*, akan tetapi perbedaannya dengan *flowchart* adalah diagram aktivitas bisa mendukung perilaku paralel sedangkan *flowchart* tidak bisa. Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut:

- ✓ Rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
- ✓ Urutan atau pengelompokan tampilan dari sistem/user interface, yaitu setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan.
- ✓ Rancangan pengujian yaitu setiap aktivitas dianggap memerlukan sebuah pengujian yang terdefinisi.

Tabel berikut memberikan contoh simbol-simbol yang sering digunakan digunakan untuk membuat suatu diagram aktivitas:

Notasi	Gambar	Keterangan
Status awal		Status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah status awal.
Status akhir		Status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki sebuah status akhir.
Aktivitas		Aktivitas yang dilakukan sistem, biasanya diawali dengan kata kerja
Decision		Asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu, bisa juga pilihan untuk mengambil keputusan
Fork/Join		Fork, digunakan untuk menunjukkan kegiatan yang dilakukan secara paralel, atau untuk menggabungkan dua kegiatan paralel menjadi satu.

Rake		Menunjukkan adanya dekomposisi
Swimlane		Memisahkan objek/pelaku yang bertanggung jawab terhadap aktivitas yang terjadi
Aliran akhir		Aliran akhir (<i>flow final</i>)
Pengiriman		Tanda pengiriman/ <i>signal sending</i>
Penerimaan		Tanda penerimaan/ <i>signal receipt</i>

Langkah-langkah pembuatan *activity diagram*

Diagram aktivitas dibaca dari atas ke bawah, mungkin bercabang untuk menunjukkan kondisi, keputusan, dan/atau memiliki kegiatan paralel. Berikut adalah langkah-langkah membuat diagram aktivitas:

1. Buat simbol status awal ketika mengawali diagram
2. Gambarkan aksi pertama dan seterusnya sesuai aliran kegiatan sistem. Gunakan sebuah *fork* ketika membagi alur menjadi dua buah aktivitas atau lebih yang terjadi secara bersamaan, lalu gabungkan lagi dengan simbol *join*.
3. Apabila terdapat lebih dari satu objek/pelaku, maka tempatkan aktivitas sesuai dengan pelakunya. Penggunaan *swimlane* untuk memisahkan aktivitas-aktivitas dengan objek pelaku yang berbeda.
4. Cabang *decision* digunakan untuk menunjukkan suatu kegiatan yang memenuhi kondisi tertentu.
5. Akhiri diagram dengan simbol status akhir.

Contoh *activity diagram*

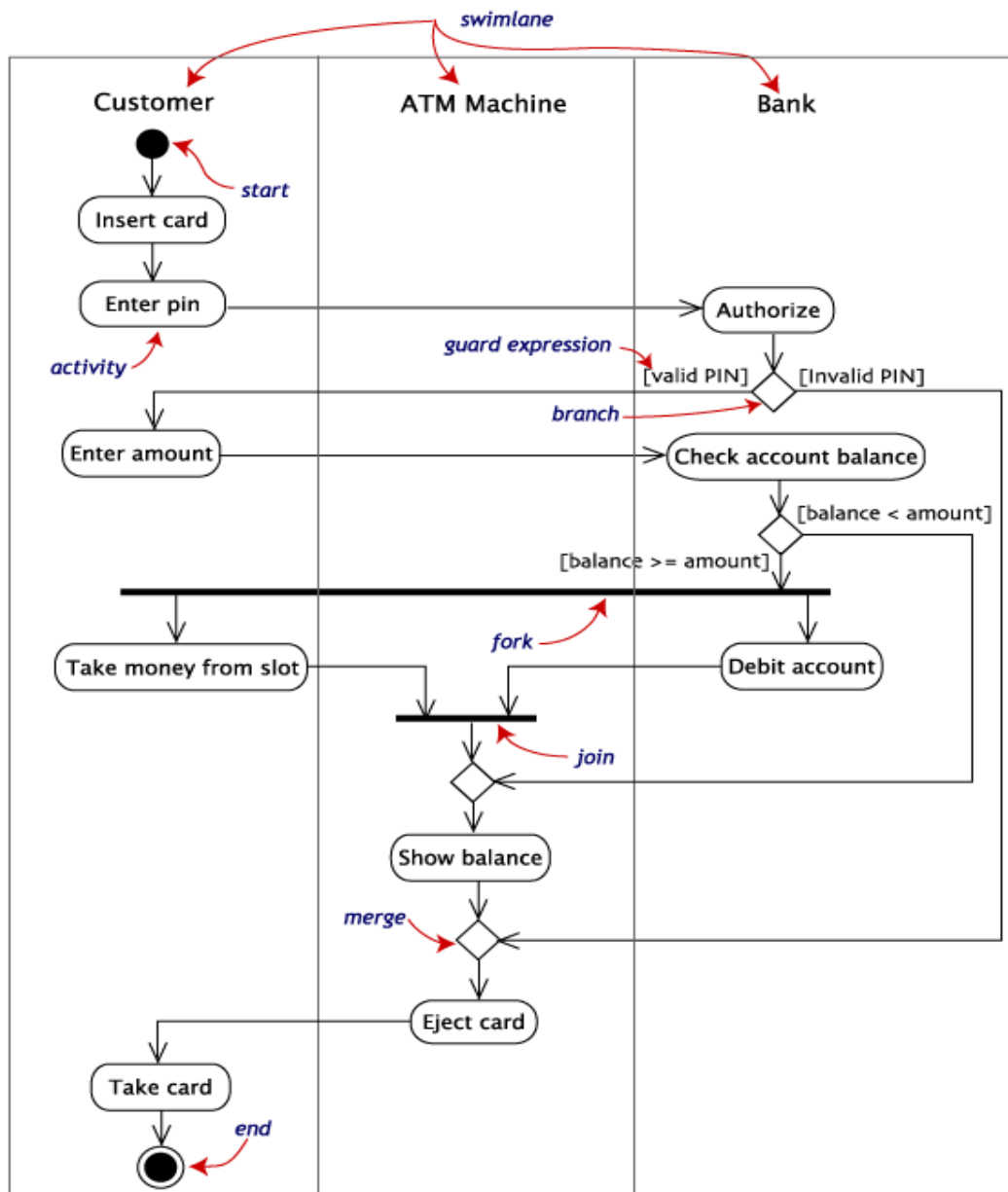
Gambar berikut ini menunjukkan sebuah contoh sederhana dari diagram aktivitas menggunakan *swimlane*. Pada sistem, terdapat 3 pengguna sistem beserta aktivitasnya, yaitu:

Nama Swimlane	Aktivitas
Pelanggan (customer)	<ul style="list-style-type: none"> ✓ Masukkan kartu ATM ✓ Masukkan PIN ✓ Masukkan nominal ✓ Ambil uang dari slot ✓ Ambil kartu ✓ Selesai



Mesin ATM	<ul style="list-style-type: none"> ✓ Perlihatkan saldo ✓ Tolak kartu
Bank	<ul style="list-style-type: none"> ✓ Lakukan otentikasi ✓ Cek saldo ✓ Debet tabungan

Hasil diagram aktivitas tersebut adalah:

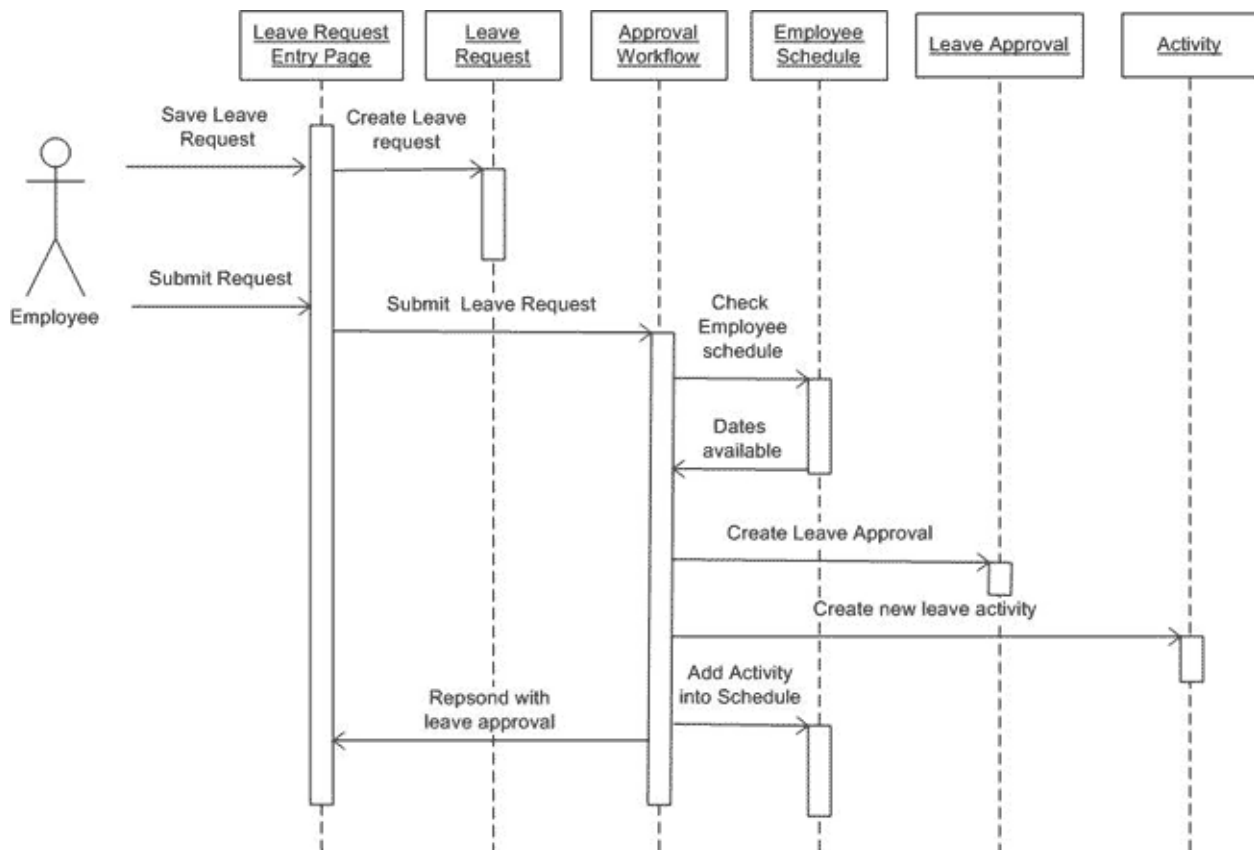


6.4 Sequence Diagram

Sequence diagram menggambarkan interaksi antara sejumlah objek dalam urutan waktu. Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara objek juga interaksi antar objek yang terjadi pada titik tertentu dalam eksekusi sistem.



Contoh sequence diagram



Sequence diagram memiliki ciri yang berbeda dengan diagram interaksi pada *communication diagram*. Perbedaan tersebut adalah:

2. Pada *sequence diagram* terdapat *lifeline* objek. *Lifeline* objek adalah garis tegas vertikal yang mencerminkan keberadaan sebuah objek sepanjang periode waktu. Sebagian besar objek-objek yang tercakup dalam diagram interaksi akan ada selama waktu tertentu, dan objek-objek itu diletakkan di bagian atas diagram dengan lifeline yang tergambar dari atas hingga bagian bawah diagram. Selain itu, objek lain dapat diciptakan. Dalam kasus ini, garis hidup dimulai pada saat pesan *create* diterima suatu objek. Suatu objek juga dapat dimusnahkan dengan pesan *destroy*. Apabila kasus ini terjadi, maka lifeline objek tersebut juga berakhir.
3. Terdapat fokus kendali (*Focus of Control*), berupa empat persegi panjang yang menampilkan aksi suatu objek secara langsung atau sepanjang subordinat. Puncak dari empat persegi panjang adalah permulaan aksi, bagian bawah adalah akhir dari suatu aksi (dan dapat ditandai dengan pesan *return*).



Contoh Kasus:

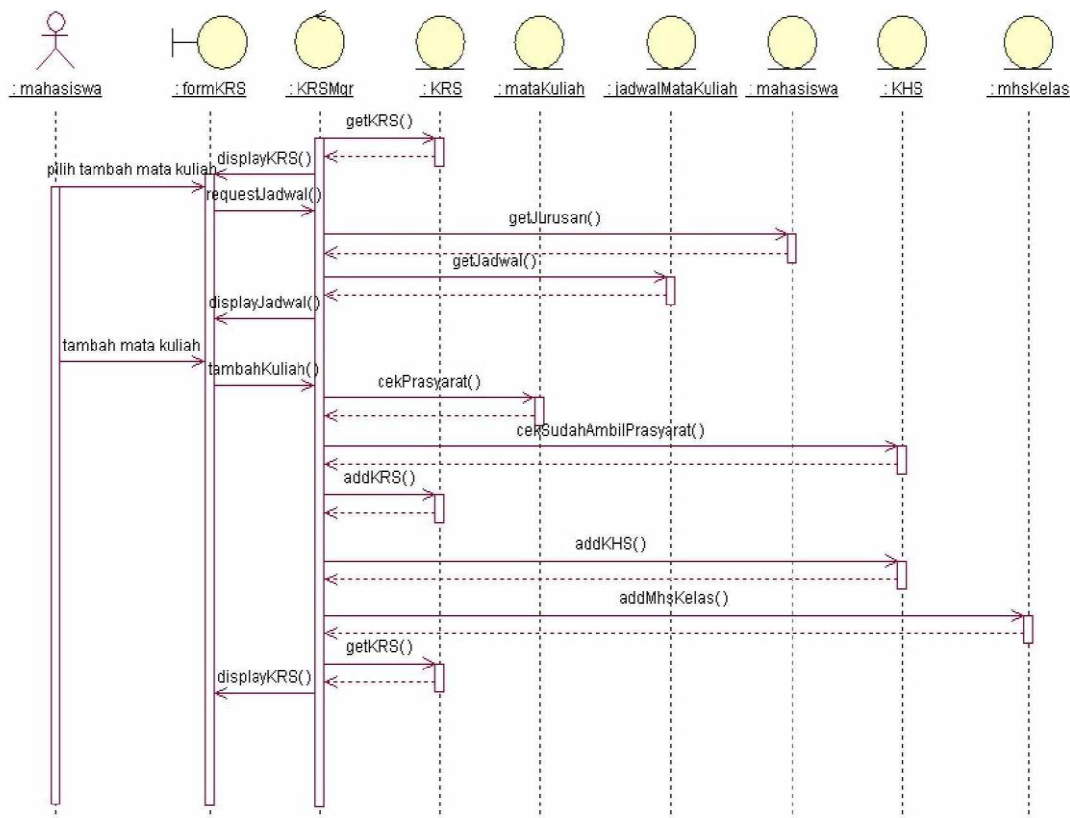
Sequence diagram untuk pasangan aktor mahasiswa dan use case KRS

Gambar 1 menunjukkan potongan use case diagram untuk pasangan mahasiswa dan use case KRS.



Gambar 1 Pasangan mahasiswa dan use case KRS

Gambar 3a dan 3b menunjukkan sequence diagram untuk pasangan mahasiswa dan use case KRS. Karena keterbatasan area gambar, maka sequence diagram untuk use case KRS dibagi menjadi dua bagian, yaitu sequence diagram yang menunjukkan class yang terlibat pada proses penambahan mata kuliah pada KRS (gambar 3a) dan sequence diagram yang menunjukkan class yang terlibat pada proses penghapusan mata kuliah dari KRS (gambar 3b)

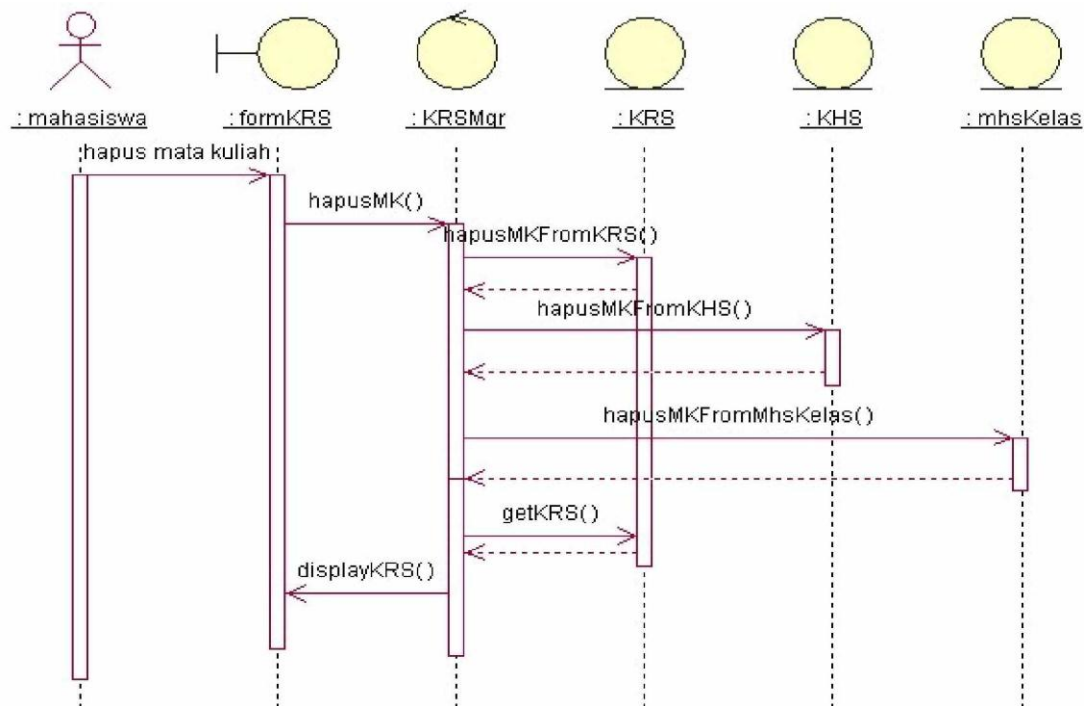


Gambar 3a sequence diagram untuk penambahan mata kuliah pada KRS

Pada gambar 3a ditunjukkan sequence diagram untuk penambahan mata kuliah pada KRS. Sebelum formKRS ditampilkan, maka class controller KRSMgr berusaha mendapatkan KRS mahasiswa dengan memanggil method getKRS pada class KRS. Jika mahasiswa belum mengisi KRS, hasilnya adalah KRS kosong. Jika mahasiswa telah mengisi KRS sebelumnya, maka hasilnya adalah KRS yang telah dibuat oleh mahasiswa.



KRS tersebut akan ditampilkan pada formKRS. Mahasiswa kemudian memilih menu tambah mata kuliah. Jika menu tersebut dieksekusi, maka class KRSMgr akan menampilkan mata kuliah yang dapat dipilih beserta jadwalnya (melalui query lewat classjadwalMataKuliah). Jika salah satu mata kuliah dipilih, maka diadakan pengecekan dahulu apakah mahasiswa tersebut sudah mengambil prasyarat mata kuliah. Class KRSMgr akan melakukan pengecekan tersebut dengan melibatkan class mataKuliah dan KHS. Jika telah mengambil prasyarat, maka mata kuliah tersebut akan ditambahkan ke KRS dan KHS mahasiswa yang bersangkutan.



Gambar 3b sequence diagram untuk penghapusan mata kuliah pada KRS

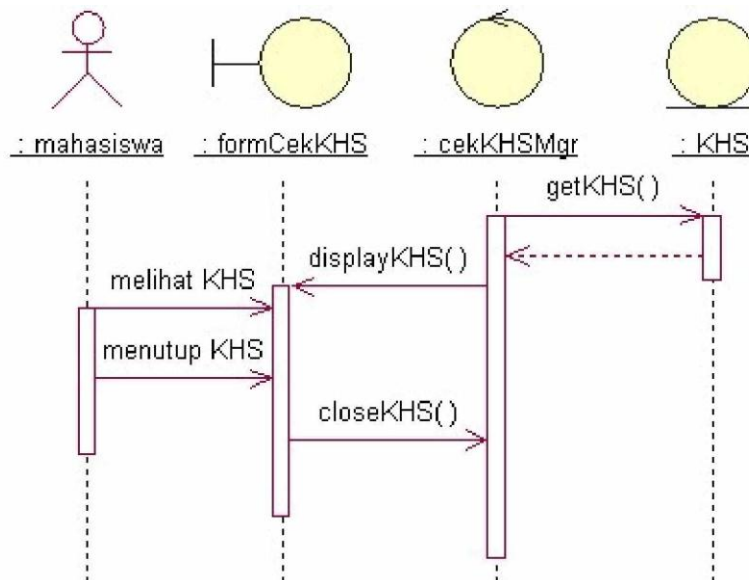
Jika mahasiswa menghapus mata kuliah tertentu pada KRSnya, maka dilakukan penyesuaian pada KRS dan KHSnya. Class KRSMgr akan melakukan penghapusan pada database melalui class KRS, KHS dan mhsClass. Setelah dilakukan penghapusan, daftar mata kuliah yang masih tersisa pada KRS ditampilkan kembali pada formKRS. Sequence diagram untuk pasangan aktor mahasiswa dan use case cek KHS Gambar 4 menunjukkan potongan use case diagram untuk pasangan mahasiswa dan use case cek KHS.



Gambar 4 Pasangan mahasiswa dan use case cek KHS



Gambar 5 menunjukkan sequence diagram untuk pasangan mahasiswa dan use case cek KHS. Jika mahasiswa akan melakukan pengecekan nilai, sebelum formCekKHS ditampilkan, class cekKHSMgr akan melakukan query melalui class KHS untuk mendapatkan data KHS untuk ditampilkan pada formCekKHS



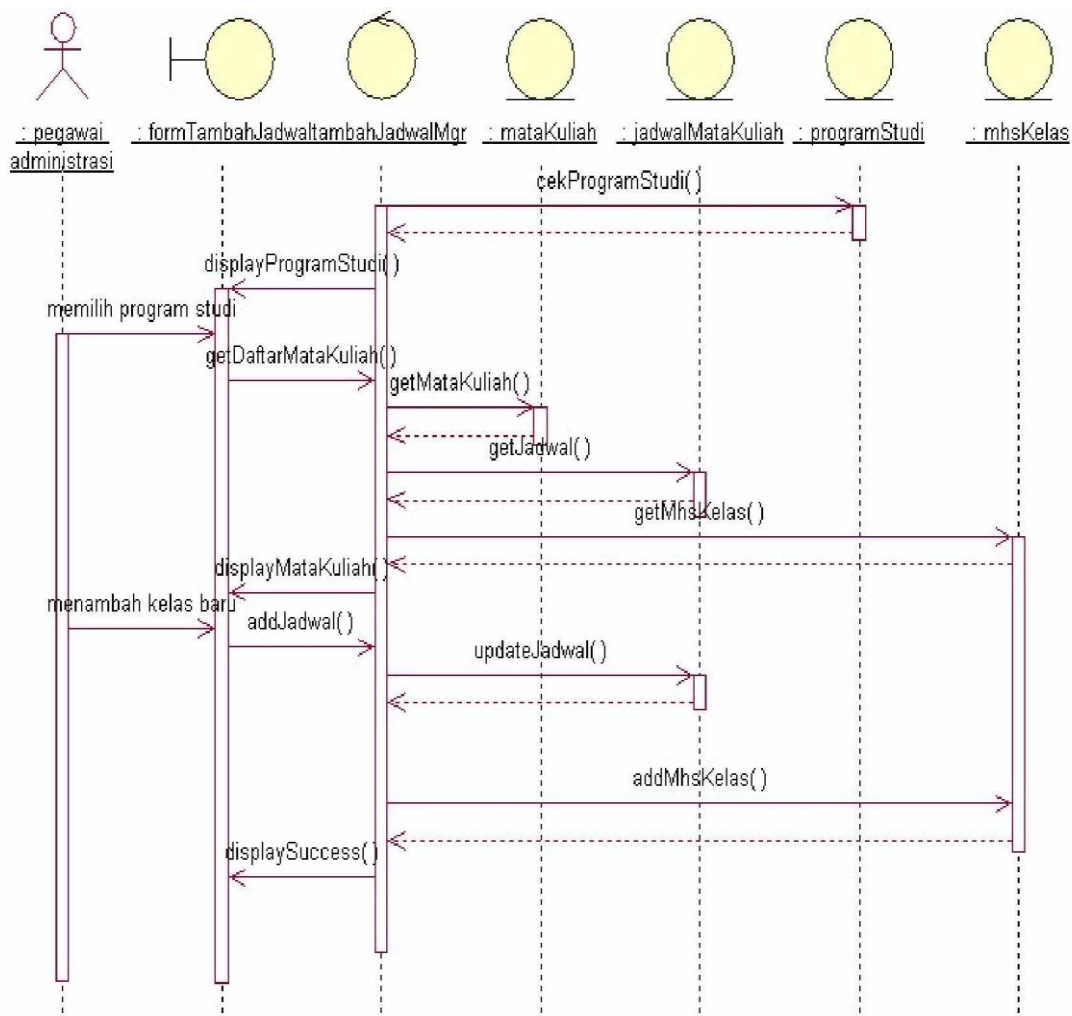
Gambar 5 sequence untuk pasangan mahasiswa dan use case cek KHS

Sequence diagram untuk pasangan aktor pegawai adminitrasi dan use case memasukkan mata kuliah dan jadwal Gambar 6 menunjukkan potongan use case diagram untuk pasangan actor pegawai adminitrasi dan use case memasukkan jadwal kuliah



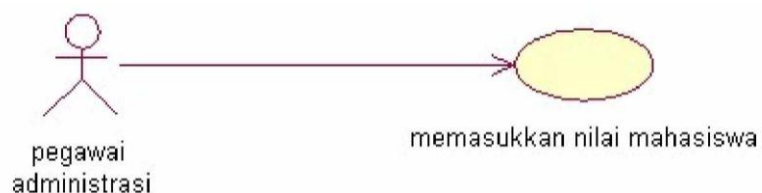
Gambar 6 Pasangan pegawai administrasi dan use case memasukkan jmata kuliah dan jadwal

Gambar 7 menunjukkan sequence diagram untuk pasangan aktor pegawai adminitrasi dan use case memasukkan jadwal kuliah. Sebelum formTambahJadwal ditampilkan, class tambahJadwalMgr akan melakukan query melalui class programStudi untuk mendapatkan daftar seluruh program studi yang akan ditampilkan pada formTambahJadwal. Pegawai administrasi mula ± mula akan memilih program studi yang akan ditambahkan jadwal mata kuliahnya (sehingga mahasiswa prodi tersebut dapat mengambil mata kuliah tesebut pada saat KRS). Dari prodi yang dipilih, maka ditampilkan seluruh mata kuliah yang dimiliki oleh prode tersebut. Proses query akan melibatkan class tambahJadwalMgr, mataKuliah, jadwalMataKuliah dan mhsClass. Petugas administrasi kemudian membuat class dan jadwal baru untuk mata kuliah tertentu. Bila penambahan class tersebut berhasil, maka ditampilkan pada formTambahJadwal bahwa pembuatan class baru telah berhasil. Class barutersebut dapat dipilih oleh mahasiswa pada saat proses KRS



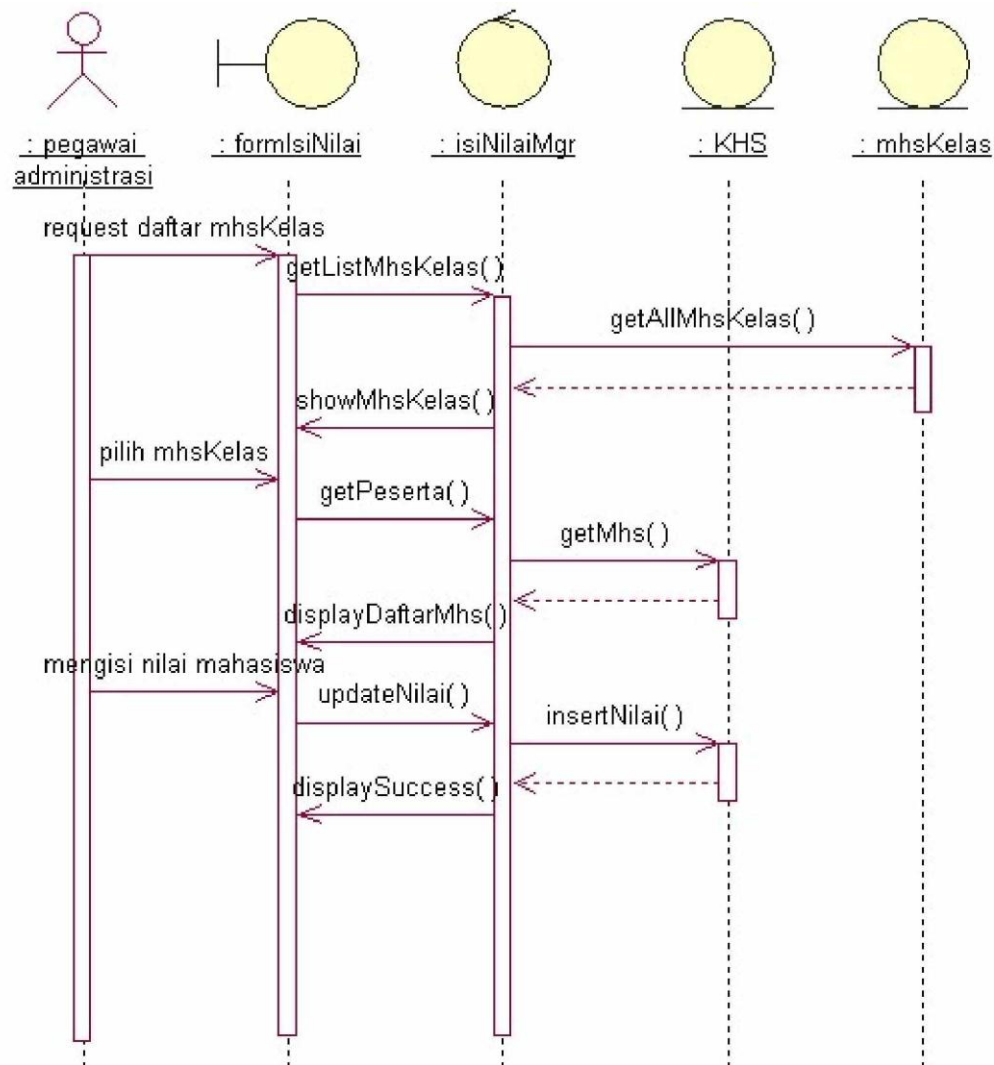
Gambar 7 sequence diagram untuk pasangan pegawai administrasi dan use case memasukkan mata kuliah dan jadwal

Sequence diagram untuk pasangan aktor pegawai administrasi dan use case memasukkan nilai mahasiswa. Gambar 8 menunjukkan potongan use case diagram untuk pasangan actor pegawai administrasi dan use case memasukkan nilai mahasiswa



Gambar 8 Pasangan pegawai administrasi dan use case memasukkan nilai mahasiswa





Gambar 9 sequence diagram untuk pasangan pegawai administrasi dan use case memasukkan nilai mahasiswa

Gambar 9 menunjukkan sequence diagram untuk pasangan aktor pegawai administrasi dan use case memasukkan nilai mahasiswa. Untuk mengisi nilai mata kuliah class tertentu petugas administrasi akan melakukan request untuk menampilkan seluruh class yang ada. Request tersebut ditangani oleh class isiNilaiMgr dan mhsClass. Dari class yang dipilih akan ditampilkan daftar peserta dan isian nilainya pada formIsiNilai. Petugas administrasi mengisi nilai mahasiswa pada form tersebut. Setelah mengisi nilai, maka nilai tersebut akan disimpan pada database. Class yang bertanggung jawab untuk proses tersebut adalah class isiNilaiMgr dan KHS.

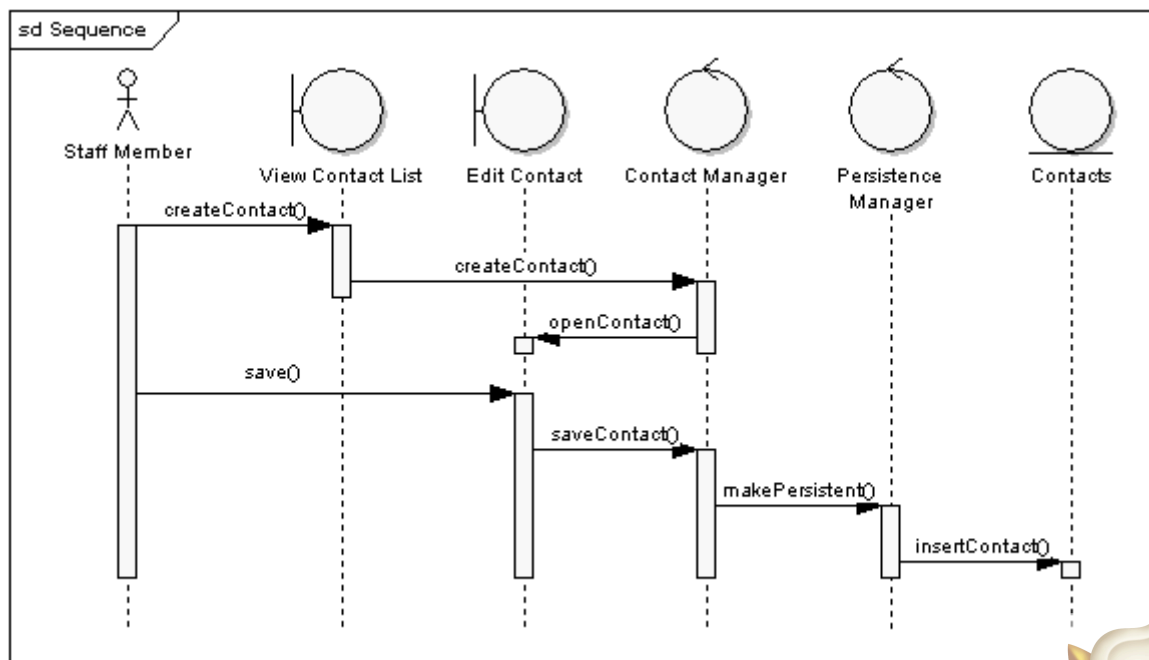
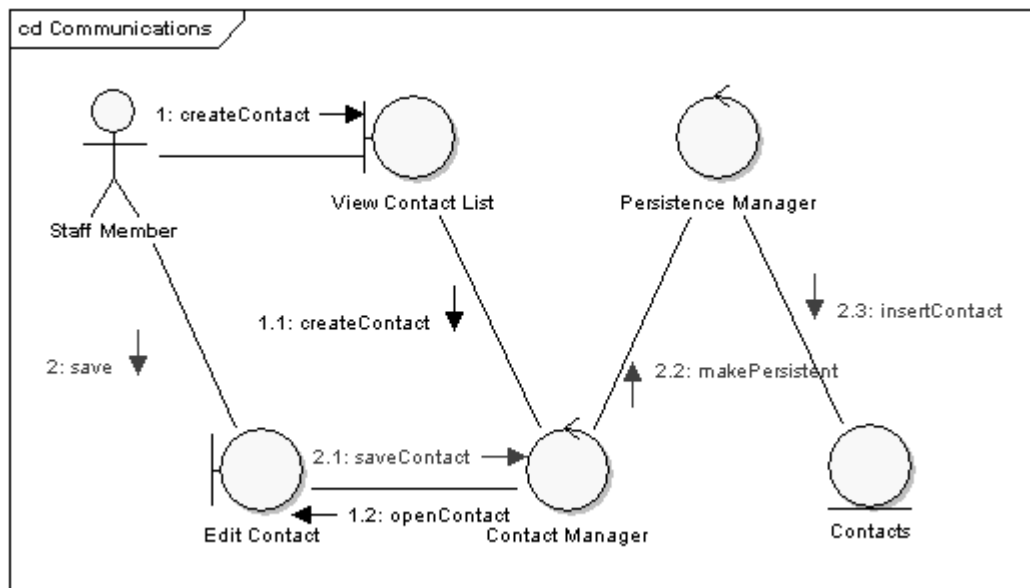
6.5 CommunicationDiagram

Communication diagram, yang sebelumnya disebut dengan *collaboration diagram*, merupakan diagram interaksi yang menunjukkan informasi yang hampir mirip dengan *sequence diagram*. Akan tetapi, *communication diagram* lebih berfokus pada hubungan antar objek.



Pada *communication diagram*, objek ditunjukkan dengan asosiasi penghubung ditengah-tengah. Pesan yang disampaikan ditambahkan ke asosiasi dan digambarkan dengan anak panah pendek yang menunjukkan arah aliran pesan. Urutan pesan ditunjukkan dengan skema penomoran.

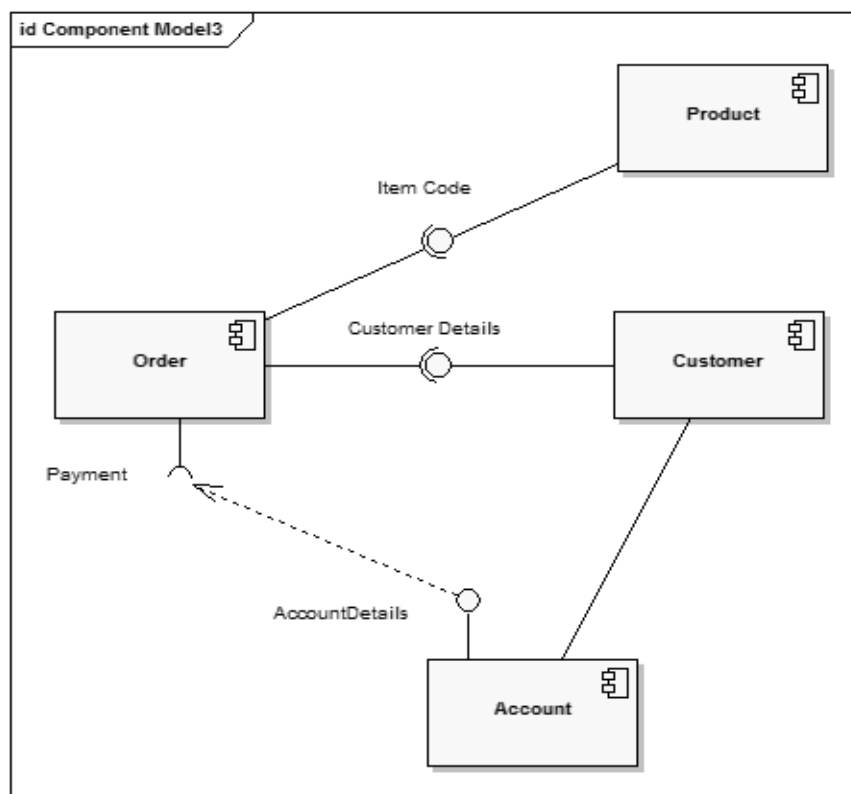
Gambar di bawah ini menunjukkan *communication diagram* dan *sequence diagram* yang memberikan informasi yang sama. Perolehan urutan pesan dari *communication diagram* dengan menggunakan skema penomoran masih bisa dilihat. Akan tetapi, perolehan urutan pesan ini tidak tampak secara langsung. Apa yang ditunjukkan oleh *communication diagram* cukup jelas, yaitu rangkaian pesan yang dilewatkan di antara objek yang berdekatan.



6.6 ComponentDiagram

Component diagram mengilustrasikan struktur perangkat lunak, *embedded controller*, *code*, baik *source code* maupun *binarycode*, *library* maupun *executable code* yang muncul pada *compile time*, *linktime*, maupun *run time*, yang berperan dalam pembuatan sistem. Suatu *component diagram* memiliki level yang lebih tinggi daripada *class diagram* (biasanya suatu komponen diimplementasikan oleh satu/lebih *class/objek* pada *runtime*). Komponen diagram digambarkan dengan *building blocks* sehingga komponen penyusunnya dapat mencakup sebagian besar dari suatu sistem. *Component diagram* menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) diantaranya.

Komponen piranti lunak adalah modul. Pada umumnya komponen terbentuk dari beberapa *class* dan/atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain. Berikut ini merupakan contoh dari *component diagram*.



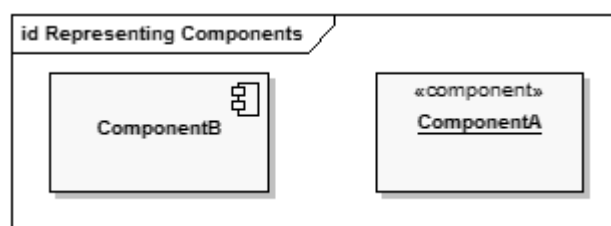
Contoh Component Diagram

Diagram di atas menunjukkan beberapa komponen dan *inter-relationship* diantaranya. Pertemuan antar konektor menunjukkan *interface* yang disediakan oleh "Product" dan "Customer", serta digunakan untuk *interface* yang ditentukan oleh "Order". Hubungan *dependency* memetakan detail akun (*account*) yang berasosiasi dengan pelanggan (*customer*) kepada *interface* yang dibutuhkan; "Payment", ditandai dengan "Order".

Dalam implementasinya, *component diagram* memiliki persamaan dengan *package diagram*, karena keduanya mendefinisikan batasan-batasan dan digunakan untuk mengelompokkan elemen menjadi struktur logis. Perbedaan antara *package diagram* dan *component diagram* adalah, *component diagram* menawarkan mekanisme pengelompokan yang lebih bersifat semantik. Artinya dengan menggunakan *component diagram*, seluruh model elemen dianggap *private*, sedangkan *package diagram* hanya menampilkan *item public*.

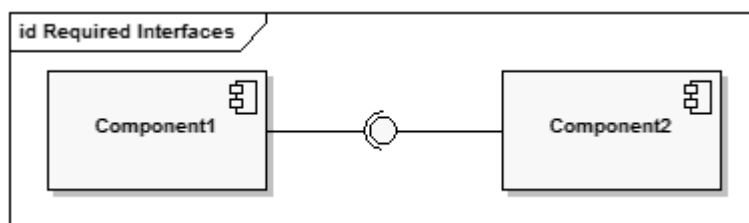
Menggambaran *Component Diagram*

Component digambarkan dengan persegi panjang dengan keterangan <<component>>. Selain itu, *component* bisa digambarkan dengan persegi panjang dengan *icon component* di sebelah pojok kanan atas. Gambar berikut merupakan contoh cara menggambar *component*.



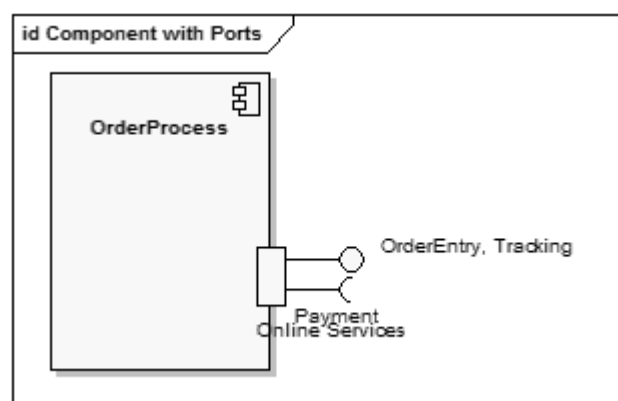
Assembly Connector

Assembly connector menghubungkan *interface* yang dibutuhkan oleh suatu *component* (Component1) dengan *interface component* lain (Component2), sehingga satu komponen dapat memberikan layanan yang dibutuhkan oleh komponen lain.



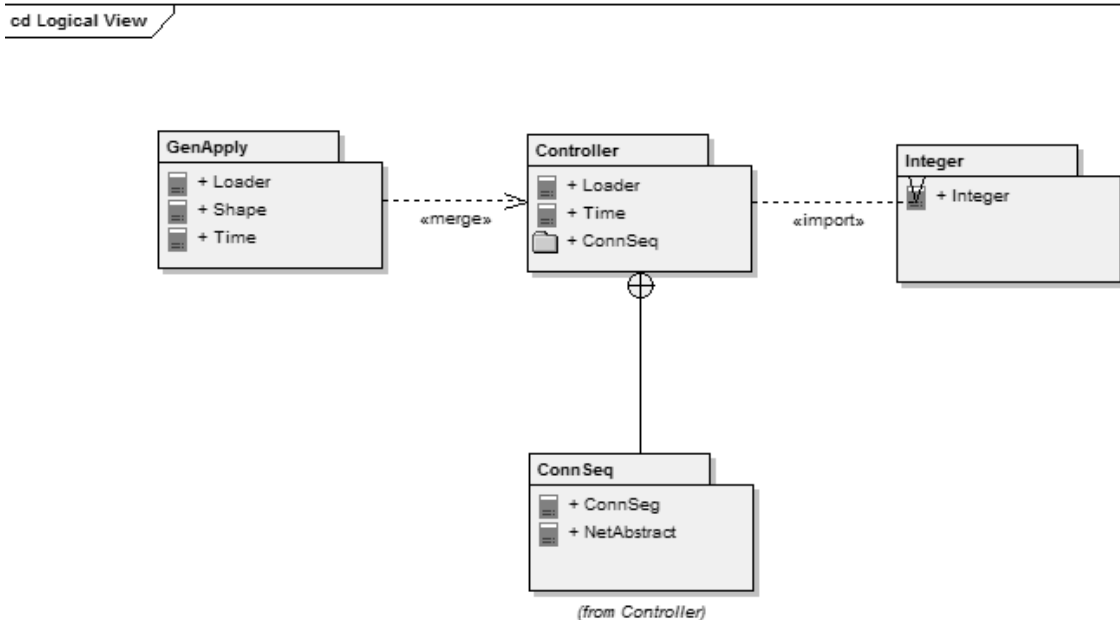
Component dengan Menggunakan Port

Port dapat menentukan *input* dan *output*, dengan kata lain, port bersifat dua arah. Gambar berikut ini merupakan contoh *component diagram* yang menggunakan port, dan digunakan untuk kasus layanan pembayaran online.



6.7 Package Diagram

Package diagram digunakan untuk merefleksikan susunan *package* beserta elemen-elemennya. Jika digunakan untuk menggambarkan elemen *class*, *package diagram* menyediakan visualisasi *namespace*. Diagram ini paling sering digunakan untuk mengorganisasi *use case diagram* dan *class diagram*, meskipun penggunaan *package diagram* tidak hanya terbatas untuk elemen-elemen UML. Gambar berikut ini merupakan contoh *package diagram*.



Elemen yang terdapat di suatu *package* memiliki *namespace* yang sama. Oleh karena itu, elemen yang terdapat di dalam suatu *namespace* khusus harus memiliki nama yang unik. *Package* dapat dibangun untuk merepresentasikan hubungan, fisik maupun logis. Saat memilih menyertakan *class* ke dalam suatu *package*, menempatkan *class* yang memiliki heirarki *inheritance* yang sama, ke dalam *package* yang sama merupakan cara yang sangat berguna.

Package Merge

Konektor «merge» antara dua *package* mendefinisikan generalisasi implisit antara elemen-elemen yang ada dalam suatu *source package*, dan elemen dengan nama yang sama yang ada di *package* yang menjadi target.

Package Import

Konektor «import» artinya, jika elemen-elemen dalam *package* target, merupakan *class* tunggal, gunakan nama umum apabila digunakan sebagai referensi dari *source package*. *Namespace* dari *source package* memberikan akses ke *class* yang menjadi target, sehingga *namespace* target tidak terpengaruh.

Nesting Connector

Konektor ini digunakan apabila *source package* telah terdapat di dalam *package* yang menjadi target.



6.8 Penggunaan UML

Langkah-langkah penggunaan UML:

1. Buatlah daftar *business process* dari level tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul.
2. Petakan *use case* untuk tiap *business process* untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh sistem. Kemudian perhalus *use case diagram* dan lengkapi dengan *requirement*, *constraints* (batasan-batasan), dan catatan-catatan lain.
3. Buatlah *deployment diagram* secara kasar untuk mendefinisikan arsitektur fisik sistem.
4. Definisikan persyaratan lain (non-fungsional, *security*, dan sebagainya) yang juga harus disediakan oleh sistem.
5. Berdasarkan *use case diagram*, mulailah membuat *activity diagram*.
6. Definisikan objek-objek level atas (*package* atau *domain*) dan buatlah *sequence* dan/atau *collaboration* diagram untuk tiap alir pekerjaan. Jika sebuah *use case* memiliki kemungkinan alir normal dan error, buatlah satu diagram untuk masing- masing alir.
7. Buarlah rancangan *user interface* model yang menyediakan antarmuka bagi pengguna untuk menjalankan skenario *use case*.
8. Berdasarkan model-model yang sudah ada, buatlah *class diagram*. Setiap *package* atau *domain* dipecah menjadi hirarki *class* lengkap dengan atribut dan metodenya. Akan lebih baik jika untuk setiap *class* dibuat *unit test* untuk menguji fungsionalitas *class* dan interaksi dengan *class* lain.
9. Setelah *class diagram* dibuat, kita dapat melihat kemungkinan pengelompokan *class* menjadi komponen-komponen. Karena itu buatlah *component diagram* pada tahap ini. Juga, definisikan tes integrasi untuk setiap komponen meyakinkan ia berinteraksi dengan baik.
10. Perhalus *deployment diagram* yang sudah dibuat. Detilkan kemampuan dan *requirement* piranti lunak, sistem operasi, jaringan, dan sebagainya. Petakan komponen ke dalam *node*.
11. Mulailah membangun sistem. Ada dua pendekatan yang dapat digunakan :
 - ✓ Pendekatan *use case*, dengan meng-assign setiap *use case* kepada tim pengembang tertentu untuk mengembangkan unit code yang lengkap dengan tes.
 - ✓ Pendekatan komponen, yaitu meng-assign setiap komponen kepada tim pengembang tertentu.
12. Lakukan uji modul dan uji integrasi serta perbaiki model berserta kodenya. Model harus selalu sesuai dengan kode yang aktual.
13. Piranti lunak siap di-*release*.



E. LATIHAN

1. Buatlah use case diagram untuk kasus di bawah ini :
 - a. Rental CD/DVD
 - b. Penjualan barang di Mirota Kampus
 - c. Perpustakaan FMIPA UGM
 - d. Apotek
 - e. Pendaftaran praktikum
 - f. Reservasi tiket pesawat
 - g. Pendaftaran dan penjadwalan ujian tugas akhirDengan asumsi batasan yang Anda buat.
2. Carilah informasi tentang *free software* yang mendukung pengembangan berbasis UML/software yang digunakan untuk membuat diagram-diagram UML!
3. Apakah class diagram dan activity diagram dapat digunakan untuk pengembangan sistem apapun ataupun hanya sistem-sistem tertentu? Jelaskan!
4. Akan dibuat sebuah sistem perpustakaan sederhana yang mempunyai fungsi inti pencarian buku, pengelolaan data anggota perpustakaan, pengaturan data buku, dan peminjaman. Buatlah use case dan activity diagram sederhana yang merepresentasikan sistem perpustakaan tersebut!
5. Bagaimanakah kedudukan (juga kelebihan dan kekurangan) UML dibandingkan dengan DAD dan flowchart? Diagram apa sajakah yang diperlukan untuk membangun suatu sistem?

F. TUGAS BESAR BAGIAN 2 (Lanjutan Bagian 1)

Buatlah sistem yang bukan sistem informasi sesuai dengan yang Anda inginkan. Pada tahap ini, Anda WAJIB :

1. Mempresentasikan sistem yang Anda kembangkan!
2. Menjabarkan sistem ke 4 diagram UML!
3. Mempresentasikan hasil sistem yang Anda kembangkan!

