Yulian Findawati, ST., M.MT.
;;'
Cindy Taurusta, S.ST., M.T.

BUKU AJAR
REKAYA PERANGKAT LUNAK
Terstruktur Dan Berorientasi Objek

I

UNIVERSITAS MUHAMMADIYAH SIDOARJO

IDENTITAS BUKU

Rekayasa perangkat lunak (software engineering) bukanlah berbicara tentang bahasa pemrograman untuk membangun sebuah perangkat lunak. Rekayasa Perangkat Lunak atau RPL adalah suatu bidang profesi yang mendalami cara-cara pengembangan perangkat lunak, termasuk pembuatan, pemeliharaan, manajemen organisasi pengembangan perangkat lunak, dan sebagainya, atau sebuah profesi dari seorang perekayasa perangkat lunak yang berkaitan dengan pembuatan dan pemeliharaan aplikasi perangkat lunak dengan menerapkan teknologi dan praktik dari ilmu komputer, manajemen proyek, dan bidang-bidang lainnya. Buku ini mudah dipahami kerena disertai studi kasus.

Pembahasan dalam buku ini mencakup:

* Pengantar Rekayasa Perangkat Lunak,
* Evolusi Strategi Pengembangan Perangkat Lunak,
* Model-Model Proses Pengembangan Perangkat Lunak,
* Kebutuhan Perangkat Lunak,
* Metode dan Tool Rekayasa Perangkat Lunak,
* Organisasi Model Kematangan dari CMM,
* Manajemen Proyek untuk Perangkat Lunak,
* Asumsi Manajemen Proyek,
* Tim Perangkat Lunak,
* Kualitas Perangkat Lunak,
* Pengujian Perangkat Lunak,
* Pemeliharaan Perangkat Lunak, dan
* Komponen Perangkat Lunak.

PRAKATA

Puji syukur kepada Tuhan Yang Maha Esa, sehingga Usulan Buku Rekayasa Perangkat Lunak ini dapat disusun dengan baik dan selesai pada waktu yang telah ditentukan oleh Universitas Muhammadiyah Sidoarjo khususnya LP3IK yang bersedia untuk mengeluarkan dana dalam penulisan buku ajar ini. Penulisan buku ajar ini ditulis dalam 5 bab secara garis besar, berdasarkan pengalaman penulis didunia kerja serta diambil dari hasil penelitian baik dari hasil penelitian mandiri maupun dari pendanaan Kemenristekdikti yang penulis dapatkan . Tak lupa kami juga mengucapkan terima kasih kepada:

1. Dr. Drs. Hidayatullah, M.Si selaku Rektor Universitas Muhammadiyah Sidoarjo yang telah memberikan dan memfasilitasi dalam penulisan buku ajar ini.
2. LP3iK Universitas Muhammadiyah Sidoarjo yang telah memfasilitasi dan mengkoordinasi dalam penulisan buku ajar ini.
3. Izza Anshori, ST.,MT. selaku Dekan Fakultas Teknik, Universitas Universitas Muhammadiyah Sidoarjo yang telah memberikan dukungan untuk mengikuti penulisan buku ajar ini.
4. Dosen-dosen Teknik Informatika Universitas Universitas Muhammadiyah Sidoarjo yang telah memberikan dukungan untuk mengikuti penulisan buku ajar ini.
5. Para narasumber yang penulis tidak dapat sebutkan satu persatu yang telah banyak membantu, atas pengetahuan dan keterampilan yang diberikan dalam penyusunan penulisan buku ajar ini.
6. Aslab Teknik Informatika dan mahasiswa Teknik Informatika yang penulis tidak dapat sebutkan satu persatu yang telah banyak membantu, atas pengetahuan dan keterampilan yang diberikan dalam penyusunan penulisan buku ajar ini

Akhir kata, kritik dan saran sangat diharapkan untuk penyempurnaan buku ajar ini. Harapan kami semoga buku ajar ini dapat digunakan sebagai tambahan informasi dan bermanfaat bagi aktivitas pembelajaran mata kuliah Rekayasa Perangkat Lunak di Program Studi Teknik Informatika , Fakultas Teknik, Universitas Muhammadiyah Sidoarjo dan pembaca lainnya.

Penulis

DAFTAR ISI

f. Mahasiswa dapat memahami Spesifikasi dan kajian spesifikasi

BAB IV : Pemodelan Analisis

a. Mahasiswa dapat memahami Elemen Model Analisis

b. Mahasiswa dapat memahami Pemodelan kebutuhan Fungsional

c. Mahasiswa dapat memahami Pemodelan Proses

d. Mahasiswa dapat memahami Pemodelan Data

e. Mahasiswa dapat memahami Kamus Data

f. Mahasiswa dapat memahami CRUD Matriks

BAB V : Prinsip dan Konsep Desain

a. Mahasiswa dapat memahami Desain perangkat lunak dan rekayasa Perangkat lunak

b. Mahasiswa dapat memahami Prinsip Desain

c. Mahasiswa dapat memahami Konsep Desain

d. Mahasiswa dapat memahami Desain Modular Efektif

e. Mahasiswa dapat memahami Model Desain

f. Mahasiswa dapat memahami Dokumentasi Desain

BAB VI : Metode Desain

a. Mahasiswa dapat memahami Desain Data

b. Mahasiswa dapat memahami Desain Arsitektur

c. Mahasiswa dapat memahami Proses Desain Arsitektur

d. Mahasiswa dapat memahami Pasca Pemrosesan Desain

e. Mahasiswa dapat memahami Optimasi Desain Arsitektur

f. Mahasiswa dapat memahami Desain Interface

g. Mahasiswa dapat memahami Desain Interface Manusia-Mesin

h. Mahasiswa dapat memahami Desain Prosedural

i. Mahasiswa dapat memahami Coding

BAB VII : Teknik Pengujian Perangkat Lunak

a. Mahasiswa dapat memahami Dasar – dasar pengujian Perangkat Lunak

b. Mahasiswa dapat memahami Desain Test Case

c. Mahasiswa dapat memahami Pengujian White Box

d. Mahasiswa dapat memahami Pengujian Struktur Kontrol

e. Mahasiswa dapat memahami Pengujian Black Box

BAB VIII : Strategi Pengujian Perangkat Lunak

a. Mahasiswa dapat memahami Pendekatan strategis ke pengujian perangkat lunak

b. Mahasiswa dapat memahami Pengujian Unit

c. Mahasiswa dapat memahami Pengujian Integrasi

d. Mahasiswa dapat memahami Pengujian Validasi

e. Mahasiswa dapat memahami Pengujian Sistem

f. Mahasiswa dapat memahami Debugging

g. Mahasiswa dapat memahami Quality Assurance

KULIAH REKAYASA PERANGKAT LUNAK

1. BAB I. PENGENALAN REKAYASA PERANGKAT LUNAK

a. Definisi Rekayasa Perangkat Lunak

   Ilmu yang mempelajari tehnik pembuatan software yang baik dengan pendekatan tehnik (*Engineering approach)*

b. Definisi Perangkat Lunak

   Perangkat lunak adalah (1) Instruksi (program komputer) yang ketika dijalankan memberikan fitur, fungsi dan kinerja yang diinginkan, (2) Struktur data yang memungkinkan program untuk memanipulasi informasi secara memadai, dan (3) informasi deskriptif baik dalam bentuk hard copy maupun virtual yang menggambarkan pengoperasian dan penggunaan program. (Roger S. Pressman). Perangkat lunak adalah program komputer dan dokumentasi terkait seperti persyaratan, model desain, dan manual pengguna. (Ian Sommervile)

1.1 Karakteristik perangkat lunak

a. Software merupakan elemen sistem logik dan bukan elemen sistem fisik seperti hardware
b. Elemen itu tidak aus, tetapi bisa rusak.
c. Elemen software itu direkayasa atau dikembangkan dan bukan dibuat di pabrik seperti hardware

1.2 Komponen perangkat lunak

Tiga komponen perangkat lunak, antara lain adalah sebagai berikut :

1. Sistem Operasi : Merupakan komponen utama perangkat lunak sistem. Sistem Operasi (disebut juga platform software) terdiri dari program utama dan program low-level yang mengatur operasi dasar komputer. Memungkinkan perangkat lunak aplikasi untuk berinteraksi dengan komputer dan Membantu komputer untuk mengelola sumber daya baik itu internal maupun eksternal Secara khusus, sistem operasi menangani control dan penggunaan sumber daya perangkat keras, termasuk ruang disk, memori, alokasi CPU time, dan perangkat peripheral.

2. Device Driver : Membantu komputer mengontrol perangkat peripheral. Driver artinya adalah pemacu yang maksudnya adalah

dengan dipasangnya suatu device ke komputer sementara operating sistem kita atau komputer tidak mengenalinya maka driver tadi yang akan memperkenalkan bahwa device yang dipasang itu adalah benar adanya dan bisa digunakan karena Device Driver adalah program komputer yang mengawal jenis-jenis peranti yang dipasangkan (install) pada komputer. Program ini adalah spesifik untuk peranti yang tertentu saja dan tidak boleh digunakan pada peranti yang lain , contoh: mesin pencetak(printer) memerlukan driver untuk berfungsi

3. Program Utilitas : Adalah sebuah program yang digunakan untuk Meningkatkan kapabilitas program komputer yang telah ada pada computer. Perangkat lunak utilitas merupakan perangkat lunak komputer yang didisain untuk membantu proses analisis, konfigurasi, optimasi, dan membantu pengelolaan sebuah komputer ataupun sistem. Perangkat lunak utilitas harus dibedakan dengan perangkat lunak aplikasi yang memungkinkan pengguna melakukan berbagai hal dengan komputer seperti mengetik, melakukan permainan, merancang gambar, dan lain-lain. Perangkat lunak utilitas lebih memfokuskan penggunaannya pada pengoptimasian fungsi dari infrastruktur yang terdapat dalam sebuah komputer. Karena fungsinya, perangkat lunak utilitas umumnya tidak ditujukan untuk pengguna secara umum, melainkan ditujukan untuk pengguna yang memiliki pemahaman atas cara kerja sistem komputer yang cukup baik.

1.3 Aplikasi perangkat lunak

1. Perangkat Lunak Sistem

   Adalah sekumpulan program yang ditulis untuk melayani atau menunjang program lainnya. Beberapa sistem software seperti compiler, editor, komponen-komponen sistem operasi, driver dan prosesor telekomunikasi.

2. Perangkat Lunak Real Time software

   Software yang mengukur, menganalisis dan mengontrol kejadian yang sesungguhnya terjadi di dunia. Elemen-elemen real time software terdiri dari :

   a. Komponen pengumpul data : yang mengumpulkan dan menyusun informasi dari lingkungan external.

b. Komponen analisis : Yang mentransformasikan informasi yang diperlukan oleh aplikasi
c. Komponen kontrol :Yang memberikan respon kepada lingkungan external
d. Komponen monitor : Yang mengkoordinasi semua komponen-komponen lainnya, sehingga respons real time yang berkisar 1 milisecond sampai 1 menit dapat dipertahankan.

Perlu dicatat bahwa istilah real time berbeda dari istilah interactive atau time sharing. Sistem real time harus memberikan respons pada waktu yang ditentukan, sedangkan pada sistem interactive atau time sharing respons time biasanya melebihi batas waktu yang ditentukan tanpa merusak hasil.

3. Perangkat Lunak Bisnis

Perangkat Lunak yang palinmg banyak digunakan dalam bidang aplikasi Perangkat Lunak . Perangkat Lunak ini digunakan oleh manajemen untuk mengambil kepitusan (Decision Making) dalam bidang bisnis. Contoh : DAC EASY ACCOUNTING, FINANCE MANAJER

4. Engineering and sciencetific software

Perangkat Lunak yang dicirikan dengan algoritma numerik, aplikasinya berkisar dari astronomi sampai vulkanologi, dari analis ketegangan otomotif sampai dinamika orbit ruang angkasa. Software ini banyak digunakan dalam bidang engineering dan science. Contoh :CAD / CAM (Computer Aided Design / Computer Aided Manufacture - Ssimulasi sistem)

5. Emdebed software

Suatu software disimpan dalam memori tetap - ROM - Read Only Memory, dan digunakan untuk mengontrol product dan sistem software ini dijalankan dengan berbagai fungsi terbatas.

6. PC software (Personal Computer)

Software yang banyak digunakan di komputer pribadi (PC). Contoh :

Word Processing	:	WS, WP
Spreadsheet	:	Lotus, Supercalc
Computer Graphics	:	Printshop, Print Magic
Games	:	Paoman, Load Runner
DBMS	:	Dbase III+, Foxbase, Clipper
Network	:	LAN, Novell

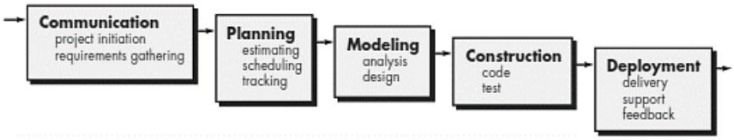
g. Artificial Intelegence Perangkat Lunak

Perangkat Lunak yang banyak menggunakan algoritma non numerik dalam memecahkan masalah kompleks yang tidak dapat dianalisis dengan analisis komputasi biasa. Saat ini bidang AI yang paling aktif adalah expert system atau knowledge base system. Bidang aplikasi lain dari Perangkat Lunak AI adalah pengenalan citra dan suara (image and voice pattern recognition), teorema pembuktian dan permainan / games.

1.4 Model perangkat lunak

a. Model Waterfall

Kadang-kadang disebut siklus kehidupan klasik, menunjukkan pendekatan sistematis berurutan pengembangan perangkat lunak yang dimulai dengan spesifikasi pelanggan persyaratan dan berlangsung melalui perencanaan, pemodelan, konstruksi, dan penyebaran, yang berpuncak pada keberlangsungan perangkat lunak



Gambar 2 . Model Waterfall

Keterbatasan:

a. Sifat persyaratan tidak akan banyak berubah selama pengembangan;
b. Model menyiratkan bahwa harus berusaha menyelesaikan tahap tertentu sebelum melanjutkan ke tahap berikutnya.
c. Tidak menjelaskan fakta bahwa persyaratan terus berubah.
d. Ini juga berarti bahwa pelanggan tidak dapat menggunakan apa pun hingga seluruh sistem selesai.

e. Model menyiratkan bahwa begitu produk selesai, yang lainnya adalah pemeliharaan.
f. Sangat mahal
g. Beberapa tim duduk ideal untuk tim lain selesai
h. Oleh karena itu, model ini hanya sesuai jika persyaratannya dipahami dengan baik dan perubahan akan terjadicukup terbatas selama proses desain.

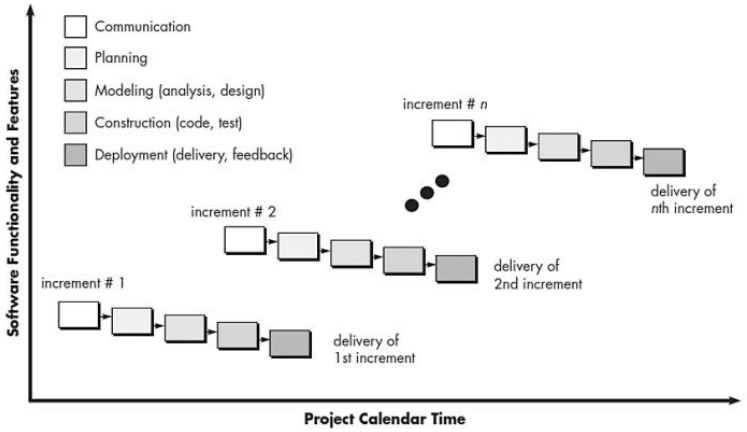
Kapan menggunakan model waterfall?

a. Persyaratan sangat dikenal, jelas dan pasti
b. Definisi produk stabil
c. Teknologi dipahami
d. Tidak ada persyaratan yang ambigu
e. Banyak sumber daya dengan keahlian yang dibutuhkan tersedia secara bebas
f. Proyek ini singkat

b. Model Incremental sebagai contoh salah satunya model RAD

Model inkremental menggabungkan elemen aliran proses linier dan paralel.

a. Model inkremental menerapkan urutan linear dengan cara berjalan bergantian seiring berjalannya waktu kalender.
b. Setiap urutan linier menghasilkan "peningkatan" yang dapat dihasilkan dari perangkat lunak.
c. Sebagai contoh, perangkat lunak pengolah kata yang dikembangkan menggunakan paradigma inkremental mungkin memberikan dasar manajemen file, mengedit, dan fungsi produksi dokumen dalam kenaikan pertama; lebih mutakhir mengedit dan mendokumentasikan kemampuan produksi dalam peningkatan kedua; memeriksa ejaan dan tata bahasa
d. kenaikan ketiga; dan kemampuan tata letak halaman lanjutan dalam peningkatan keempat.



Gambar 3. Model incremental

Keuntungan:

a. Menghasilkan perangkat lunak yang bekerja dengan cepat dan lebih awal selama siklus hidup perangkat lunak.
b. Model ini lebih fleksibel - lebih murah untuk mengubah ruang lingkup dan persyaratan.
c. Lebih mudah untuk menguji dan men-debug selama iterasi yang lebih kecil.
d. Dalam model ini pelanggan dapat merespons setiap yang dibangun.
e. Menurunkan biaya pengiriman awal.
f. Lebih mudah untuk mengelola risiko karena potongan-potongan yang berisiko diidentifikasi dan ditangani selama iterasi.

Kekurangan:

a. Membutuhkan perencanaan dan desain yang baik.
b. Memerlukan definisi yang jelas dan lengkap tentang keseluruhan sistem sebelum dapat dipecah dan dibangun secara bertahap.
c. Biaya total lebih tinggi dari air terjun.

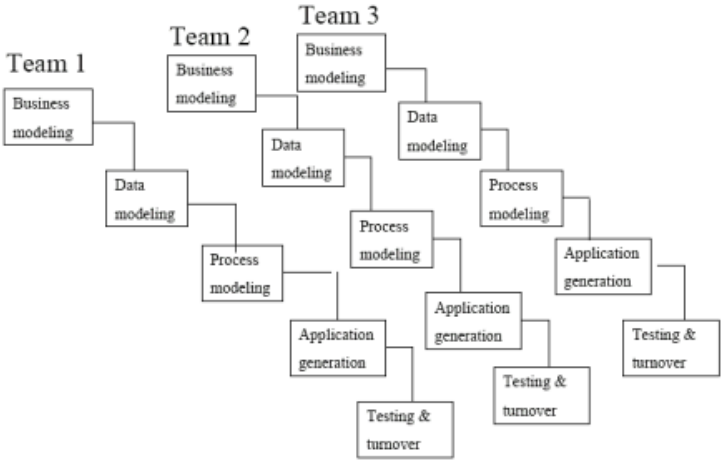
Kapan menggunakan model incremental?

a. Model ini dapat digunakan ketika persyaratan sistem lengkap didefinisikan dengan jelas dan dipahami.
b. Persyaratan utama harus ditentukan; Namun, beberapa detail dapat berevolusi seiring berjalannya waktu.
c. Ada kebutuhan untuk mendapatkan produk ke pasar lebih awal.
d. Teknologi baru sedang digunakan.
e. Sumber daya dengan keahlian yang dibutuhkan tidak tersedia.
f. Ada beberapa fitur dan tujuan berisiko tinggi.

RAD (Rapid Application Development) Process Model

Ini adalah tipe model inkremental. Dalam model RAD komponen atau fungsi dikembangkan secara paralel sebagai jika mereka adalah proyek mini.

a. Perkembangannya adalah waktu kemas, dikirim dan kemudian dirangkai menjadi prototipe kerja.
b. Ini dapat dengan cepat memberi pelanggan sesuatu untuk dilihat dan digunakan dan untuk memberikan umpan balik mengenai pengiriman dan persyaratan mereka.



Gambar 4. Model RAD

Keuntungan:

a. Mengurangi waktu pengembangan.
b. Meningkatkan reusabilitas komponen
c. Ulasan awal cepat terjadi
d. Mendorong umpan balik pelanggan

e. Integrasi dari awal menyelesaikan banyak masalah integrasi.

Kekurangan:

a. Untuk proyek-proyek besar tetapi terukur, RAD membutuhkan sumber daya manusia yang cukup.
b. Proyek gagal jika pengembang dan pelanggan tidak berkomitmen dalam kerangka waktu yang jauh lebih singkat.
c. Bermasalah jika sistem tidak dapat dimodulasi.
d. Tidak sesuai ketika risiko teknis tinggi (penggunaan teknologi baru yang berat).

Kapan menggunakan model RAD?

a. RAD harus digunakan ketika ada kebutuhan untuk membuat sistem yang dapat dimodulasi dalam waktu 2-3 bulan
b. Djika ada ketersediaan tinggi desainer untuk pemodelan dan anggaran cukup tinggi untuk membayar biaya mereka bersama dengan biaya alat pembuat kode otomatis.
c. Model RAD SDLC harus dipilih hanya jika sumber daya dengan pengetahuan bisnis yang tinggi tersedia dan
d. ada kebutuhan untuk menghasilkan sistem dalam rentang waktu singkat (2-3 bulan).

c. Model Prototyping

Ide dasar di sini adalah bahwa alih-alih membekukan persyaratan sebelum desain atau pengkodean dapat dilanjutkan, prototipe sekali pakai dibangun untuk memahami persyaratan.

a. Prototipe ini dikembangkan berdasarkan persyaratan yang diketahui saat ini.
b. Dengan menggunakan prototipe ini, klien bisa mendapatkan "nuansa sebenarnya" dari sistem, karena interaksi denganprototipe dapat memungkinkan klien untuk lebih memahami persyaratan sistem yang diinginkan.
c. Prototyping adalah ide yang menarik untuk sistem yang rumit dan besar yang tidak ada proses manual atau sistem yang ada untuk membantu menentukan persyaratan.
d. Prototipe biasanya bukan sistem yang lengkap dan banyak rinciannya tidak dibangun dalam prototipe

e. Tujuannya adalah untuk menyediakan sistem dengan fungsi keseluruhan.

2. BAB II. PERENCANAAN PROYEK PERANGKAT LUNAK

2.1 Metrik Perangkat Lunak

a. Terminologi

i. **Measure** : Indikasi kuantitatif dari tingkat, jumlah, dimensi, atau ukuran dari beberapa atribut produk atau proses.

- **Metrik**: Sejauh mana suatu sistem, komponen, atau proses memiliki atribut tertentu. Berhubungan beberapa ukuran (mis. rata-rata jumlah kesalahan yang ditemukan per jam orang)
- **Indikator**: Kombinasi metrik yang memberikan wawasan ke dalam proses perangkat lunak, proyek atau produk
- **Metrik Langsung**: Atribut yang langsung terukur (misalnya baris kode, kecepatan eksekusi, cacat dilaporkan)
- **Metrik Tidak Langsung**: Aspek yang tidak dapat dihitung secara segera (misalnya fungsi, kuantitas, keandalan)
- **Faults**:
  - ✓ Error: Kesalahan yang ditemukan oleh praktisi selama pengembangan perangkat lunak
  - ✓ Cacat: Kesalahan ditemukan oleh pelanggan setelah rilis

b. Klasifikasi Metrik

- **Proses :**Kegiatan yang terkait dengan produksi perangkat lunak
- **Produk :**Hasil eksplisit dari aktivitas pengembangan perangkat lunak, Hasil kerja, dokumentasi, berdasarkan produk
- **Proyek :** Masukan ke dalam kegiatan pengembangan perangkat lunak**,** perangkat keras, pengetahuan, SDM

c. Metrik Proses

- Metrik proses dikumpulkan di semua proyek dan dalam jangka waktu yang lama. Niat mereka adalah menyediakan serangkaian indikator proses yang mengarah pada perbaikan proses perangkat lunak jangka panjang.
- Fokus pada kualitas yang dicapai sebagai konsekuensi dari proses yang dapat diulang atau dikelola.
- Strategi dan Jangka Panjang.
- Peningkatan Statistik Proses Perangkat Lunak (SSPI).

Kategorisasi dan Analisis Kesalahan:

- ✓ Semua kesalahan dan cacat dikategorikan berdasarkan originilitas
- ✓ Biaya untuk memperbaiki setiap kesalahan dan kecacatan dicatat
- ✓ Jumlah kesalahan dan cacat di setiap kategori dihitung
- ✓ Data dianalisis untuk menemukan kategori yang menghasilkan biaya tertinggi untuk organisasi
- ✓ Rencana dikembangkan untuk memodifikasi proses

d. Metrik Proyek

- Metrik proyek memungkinkan manajer proyek perangkat lunak untuk menilai status proyek yang sedang berlangsung, melacak
- potensi risiko, mengungkap area masalah sebelum mereka "kritis," menyesuaikan alur kerja atau tugas, dan mengevaluasi
- kemampuan tim proyek untuk mengontrol kualitas produk kerja perangkat lunak.
- Metrik proyek pada sebagian besar proyek perangkat lunak terjadi selama estimasi.
- Metrik yang dikumpulkan dari proyek-proyek sebelumnya digunakan sebagai dasar dari upaya dan perkiraan waktu yang dibuat untuk pekerjaan perangkat lunak saat ini.
- Tingkat produksi diwakili dalam hal model yang dibuat, ulasan jam, poin fungsi, dan diukur.
- Metrik ini digunakan untuk meminimalkan jadwal pengembangan dengan membuat penyesuaian yang diperlukan untuk menghindari penundaan dan kurangi potensi masalah dan risiko.
- Metrik proyek digunakan untuk menilai kualitas produk secara berkelanjutan dan, bila perlu, memodifikasi pendekatan teknis untuk meningkatkan kualitas.

e. Metrik Produk

- Fokus pada kualitas kiriman
- Metrik produk digabungkan di beberapa proyek untuk menghasilkan metrik proses
- Metrik untuk produk:
  - ✓ Ukuran Model Analisis
  - ✓ Kompleksitas Model Desain
  - ✓ Kerumitan algoritmik internal
  - ✓ Kompleksitas arsitektur
  - ✓ Kompleksitas aliran data

- Metrik kode

2.2 Pengukuran Perangkat Lunak

- Tindakan Langsung (atribut internal)
    - Biaya, usaha, LOC, kecepatan, memori
- Tindakan Tidak Langsung (atribut eksternal)
    - Fungsionalitas, kualitas, kompleksitas, efisiensi, keandalan, pemeliharaan

Matrik berorientasi ukuran

- Metrik perangkat lunak berorientasi ukuran diturunkan dengan menormalkan kualitas dan / atau ukuran produktivitas dengan mengingat ukuran perangkat lunak yang telah diproduksi.
- Metrik sederhana yang berorientasi ukuran untuk setiap proyek:
    - ✓ Kesalahan per KLOC (ribuan baris kode)
    - ✓ Cacat per KLOC
    - ✓ $ per KLOC
    - ✓ Halaman dokumentasi per KLOC
    - ✓ Kesalahan per orang per bulan
    - ✓ KLOC per orang per bulan
    - ✓ $ per halaman dokumentasi

Matriks berorientasi fungsi

- Metrik perangkat lunak berorientasi fungsi menggunakan ukuran fungsionalitas yang disampaikan oleh aplikasi sebagai nilai normalisasi.
- Metrik berorientasi fungsi yang paling banyak digunakan adalah functipn point (FP).
- Komputasi titik fungsi didasarkan pada karakteristik domain informasi perangkat lunak dan kompleksitas

Matrik Berorientasi Objek

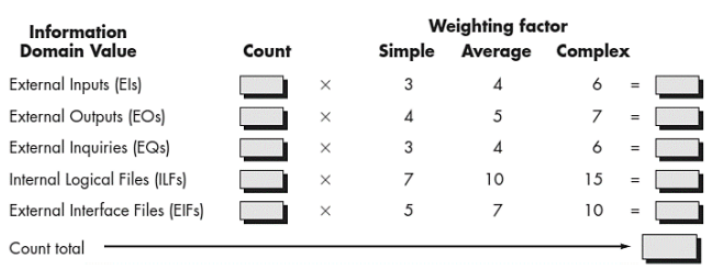
- Metrik Berorientasi Objek Metrik proyek perangkat lunak konvensional (LOC atau FP) dapat digunakan untuk memperkirakan proyek perangkat lunak berorientasi objek.
- Namun, metrik ini tidak memberikan perincian yang cukup untuk penyesuaian jadwal dan upaya itu diperlukan saat melakukan iterasi melalui proses evolusi atau inkremental.

Matrik berorientasi Use Case

- Seperti FP, use case didefinisikan pada awal proses perangkat lunak, sehingga memungkinkan untuk digunakan untuk estimasi sebelum pemodelan dan kegiatan konstruksi dimulai.
- Use case menggambarkan (secara tidak langsung, setidaknya) fungsi dan fitur yang terlihat oleh pengguna yang merupakan persyaratan dasar untuk sebuah sistem. Use case tidak bergantung pada bahasa pemrograman.
- Karena Use case dapat dibuat pada tingkat abstraksi yang sangat berbeda, tidak ada "ukuran" standar untuk use case.
- Tanpa ukuran standar tentang apa itu use case, aplikasinya menggunakan ukuran normalisasi (misalnya, usaha yang dikeluarkan per use case)

2.3 Metrik function point

- Metrik function point menyediakan metode standar untuk mengukur berbagai fungsi aplikasi perangkat lunak
- Metrik function point, mengukur fungsionalitas dari sudut pandang pengguna, itu permintaan pengguna dan menerima sebagai imbalan.
- Nilai domain informasi:
    - ✓ Jumlah input pengguna - Masukan berbeda dari pengguna
    - ✓ Jumlah keluaran pengguna - Laporan, layar, pesan kesalahan, dll.
    - ✓ Jumlah pertanyaan pengguna - Masukan on line yang menghasilkan beberapa hasil
    - ✓ Jumlah file - File logis (basis data)
    - ✓ Jumlah antarmuka eksternal - File data / koneksi sebagai antarmuka ke sistem lain
    - ✓ Formula untuk menghitung FP adalah
      FP = Jumlah Total * [0,65 + 0,01 * ∑ (Fi)]
    - ✓ Di mana, Jumlah total adalah semua jumlah kali faktor pembobotan yang ditentukan untuk setiap organisasi melalui data empiris. Fi (i = 1 hingga 14) adalah nilai penyesuaian kompleksitas



Information Domain Value	Count		Weighting factor Simple	Average	Complex		
External Inputs (EIs)		×	3	4	6	=	
External Outputs (EOs)		×	4	5	7	=	
External Inquiries (EQs)		×	3	4	6	=	
Internal Logical Files (ILFs)		×	7	10	15	=	
External Interface Files (EIFs)		×	5	7	10	=	
Count total							

Nilai Adjustment Factors (Fi):

F1. Data Communication
F2. Distributed Data Processing
F3. Performance
F4. Heavily Used Configuration
F5. Transaction Role
F6. Online Data Entry
F7. End-User Efficiency
F8. Online Update
F9. Complex Processing
F10. Reusability
F11. Installation Ease
F12. Operational Ease
F13. Multiple Sites
F14. Facilitate Change

Contoh Sederhana:

inputs

3 simple X 3 = 9
4 average X 4 = 16
1 complex X 6 = 6

outputs

6 average X 5 = 30
2 complex X 7 = 14

files

5 complex X 15 = 75

inquiries

8 average X 4 = 32

interfaces

3 average X 7 = 21
4 complex X 10 = 40

Unadjusted function points 243

F09. Complex internal processing = 3
F10. Code to be reusable = 2
F03. High performance = 4
F13. Multiple sites = 3
F02. Distributed processing = 5
Project adjustment factor = 17

Adjustment calculation:
Adjusted FP = Unadjusted FP X [0.65 + (adjustment factor X 0.01)]
= 243 X [0.65 + (17 X 0.01)]
= 243 X [0.82]
= 199.26 Adjusted function points

2.4 Estimasi Proyek Perangkat Lunak

- Estimasi proyek perangkat lunak adalah bentuk pemecahan masalah, dan dalam banyak kasus, masalah yang harus dipecahkan (yaitu, mengembangkan perkiraan biaya dan usaha untuk proyek perangkat lunak) terlalu rumit untuk dipertimbangkan dalam satubagian.
- Estimasi sumber daya, biaya, dan jadwal untuk upaya rekayasa perangkat lunak membutuhkan pengalaman, akses untuk informasi historis yang baik (metrik), dan keberanian untuk berkomitmen terhadap prediksi kuantitatif berdasarkan informasi kualitatif yang ada
- Estimasi proyek perangkat lunak adalah bentuk pemecahan masalah, dan dalam banyak kasus, masalah yang harus dipecahkan adalah terlalu rumit untuk dipertimbangkan dalam satu bagian.
- Untuk alasan ini, harus menguraikan masalah, mendifinisikannya sebagai satu set yang lebih sehingga lebih mudah dikelola

Ukuran perangkat Lunak

Ukuran "logika fuzzy":

- Pendekatan ini menggunakan teknik penalaran perkiraan yang merupakan landasan logika fuzzy.
- Untuk menerapkan pendekatan ini, perencana harus mengidentifikasi jenis aplikasi, menetapkan besarnya pada skala kualitatif, dan kemudian perbaiki besaran dalam kisaran aslinya.

Ukuran Function Point:

- Perencana mengembangkan perkiraan karakteristik domain informasi.

Ukuran komponen standar:

- Perangkat lunak terdiri dari sejumlah "komponen standar" yang berbeda yang umum untuk yang khusus area aplikasi.

Ukuran pengubah:

- Pendekatan ini digunakan ketika proyek mencakup penggunaan perangkat lunak yang ada yang harus dimodifikasi beberapa cara sebagai bagian dari proyek.
- Perencana memperkirakan jumlah dan jenis (mis., Menggunakan kembali, menambahkan kode, mengubah kode, dan menghapus kode) modifikasi yang harus diselesaikan.

Estimasi Berbasis Masalah

- Mulailah dengan Ruang lingkup terbatas
- Merombak perangkat lunak menjadi masalah masing-masing yang dapat diperkirakan secara individual
- Hitung nilai LOC atau FP untuk setiap fungsi
- Turunkan perkiraan biaya atau upaya dengan menerapkan nilai LOC atau FP ke metrik awal produktivitas. (mis., LOC / orang-bulan atau FP / orang-bulan)
- Gabungkan estimasi fungsi untuk menghasilkan perkiraan keseluruhan untuk keseluruhan proyek

2.5 COCOMO Model (Model Proses Empiris)

- Singkatan dari **onstructive Cost Model**
- Seperti halnya semua model estimasi, ia membutuhkan ukuran informasi dan menerimanya dalam tiga bentuk: objek poin, poin fungsi, dan garis source code
- Model komposisi aplikasi - Digunakan selama tahap awal rekayasa perangkat lunak dan hal-hal penting yang dibutuhkan yaitu
    - ✓ Prototyping antarmuka pengguna
    - ✓ Pertimbangan perangkat lunak dan interaksi sistem
    - ✓ Penilaian kinerja
    - ✓ Evaluasi kematangan teknologi
- Awal tahap desain model - Digunakan ketika persyaratan telah stabil dan arsitektur perangkat lunak dasar telah ditetapkan
- Model tahap pasca-arsitektur - Digunakan selama pembangunan perangkat lunak

Proyek perangkat lunak organik, Semidetached dan Embedded

- Organik: Suatu proyek pengembangan dapat dianggap tipe organik, jika proyek berkaitan dengan pengembangan program aplikasi dipahami dengan baik, ukuran tim pengembangan cukup kecil, dan anggota tim berpengalaman dalam mengembangkan jenis proyek serupa.

- **Semidetached**: Sebuah proyek pengembangan dapat dianggap tipe semidetached, jika pengembangan terdiri dari campuran staf berpengalaman dan tidak berpengalaman. Anggota tim mungkin terbatas pengalaman pada sistem terkait tetapi mungkin tidak familiar dengan beberapa aspek dari sistem yang sedang dikembangkan.
- **Embedded**: Sebuah proyek pengembangan dianggap tipe tertanam, jika perangkat lunak sedang dikembangkan dengan kuat digabungkan ke perangkat keras yang kompleks, atau jika peraturan ketat pada operasional prosedur ada.

Model COCOMO memberikan perkiraan perkiraan parameter proyek. COCOMO dasar model estimasi diberikan oleh ekspresi berikut:

- **Upaya = a1 x (KLOC) a2 PM (orang Bulan)**
- **Waktu Pengembangan = b1 x (Usaha) b2 Bulan**
  Di mana, a1, a2, b1, b2 adalah konstanta untuk setiap kategori produk perangkat lunak
- **Estimasi Upaya**
  - ✓ Organik: Upaya = 2,4 (KLOC) ^{1,05} PM
  - ✓ Semi-terpisah: Upaya = 3.0 (KLOC) ^{1,12} PM
  - ✓ Tertanam: Upaya = 3,6 (KLOC) ^{1,20} PM
- **Estimasi Waktu Pengembangan**
  - ✓ Organik: Waktu Pengembangan = 2,5 (Usaha) ^{0,38} Bulan
  - ✓ Semi-terpisah: Waktu Pengembangan = 2,5 (Usaha) ^{0,35} Bulan
  - ✓ Tertanam: Waktu Pengembangan = 2,5 (Usaha) ^{0,32} Bulan
- Contoh :
  Asumsikan bahwa ukuran s / w produk organik telah diperkirakan menjadi 32.000 baris source code . Asumsikan bahwa gaji rata-rata perangkat lunak menjadi Rs. 15.000 / - bulan. Tentukan upaya yang dibutuhkan untuk pengembangan produk perangkat lunak dan nominal waktu pengembangan.
  Upaya = 2,4 x (32) 1,05 = 91 PM
  Waktu pengembangan = 2,5 x (91) 0,38 = 14 bulan
  Biaya = 14 x 15.000 = Rs. 210,000 /

2.6 Perencanaan Proyek Perangkat Lunak

- Tujuan perencanaan proyek perangkat lunak adalah menyediakan kerangka kerja yang memungkinkan manajer untuk membuat perkiraan sumber daya, biaya, dan jadwal yang wajar.
- Selain itu, perkiraan untuk menentukan skenario kasus terbaik dan terburuk sehingga hasil proyek dapat dibatasi. Meskipun ada tingkat

ketidakpastian yang melekat, tim perangkat lunak memulai rencana yang telah ditetapkan sebagai konsekuensi dari tugas-tugas ini.

- Oleh karena itu, rencana harus disesuaikan dan diperbarui saat proyek berlangsung.

2.7 Penjadwalan Proyek

- Penjadwalan proyek perangkat lunak adalah tindakan yang mendistribusikan perkiraan upaya di seluruh durasi proyek yang direncanakan dengan mengalokasikan upaya untuk tugas rekayasa perangkat lunak tertentu.
- Penting untuk dicatat, bagaimanapun, bahwa jadwal berkembang seiring waktu.
- Selama tahap awal perencanaan proyek, jadwal makroskopik dikembangkan.
- Jenis jadwal ini mengidentifikasi semua kegiatan kerangka kerja proses utama dan fungsi produk yang mana mereka diterapkan.

Prinsip Penjadwalan

- Kompartementalisasi
  Produk dan proses harus diuraikan menjadi sejumlah kegiatan dan tugas yang dapat dikelola
- Interdependensi
  Tugas yang dapat diselesaikan secara paralel harus dipisahkan dari tugas yang diselesaikan secara serial
- Alokasi waktu
  Setiap tugas memiliki tanggal mulai dan selesai yang memperhitungkan interdependensi tugas ke dalam pembayaran
- Upaya validasi
  Manajer proyek harus memastikan bahwa pada hari tertentu ada anggota staf yang cukup yang ditugaskan menyelesaikan tugas-tugas dalam waktu yang diperkirakan dalam rencana proyek
- Mendefinisikan tanggung jawab
  Setiap tugas yang dijadwalkan perlu ditugaskan ke anggota tim tertentu
- Mendefinisikan Outcome
  Setiap tugas dalam jadwal harus memiliki hasil yang ditentukan (biasanya produk kerja atau yang dapat dikirimkan)
- Mendefinisikan milestone
  Milestone tercapai ketika satu atau lebih produk kerja dari tugas rekayasa dijamin berkualitas

Penjadwalan

- Penjadwalan proyek perangkat lunak tidak berbeda jauh dengan penjadwalan teknik multitask lainnya.
- Oleh karena itu, alat dan teknik penjadwalan proyek umum dapat diterapkan dengan sedikit modifikasi untuk proyek perangkat lunak.
- Evaluasi program dan teknik tinjauan (PERT) dan metode jalur kritis (CPM) adalah dua metode penjadwalan proyek yang dapat diterapkan untuk pengembangan perangkat lunak.
- Kedua teknik ini didorong oleh informasi yang sudah dikembangkan dalam kegiatan perencanaan proyek sebelumnya: perkiraan usaha, dekomposisi fungsi produk, pemilihan model proses yang tepat dan set tugas, dan dekomposisi dari tugas yang dipilih.
- Baik PERT dan CPM menyediakan alat kuantitatif yang memungkinkan Anda untuk
  (1) Tentukan jalur kritis — rantai tugas yang menentukan durasi proyek
  (2) Menetapkan perkiraan waktu "paling mungkin" untuk tugas individu dengan menerapkan model statistik
  (3) Hitung "batas waktu" yang menentukan "jendela" waktu untuk tugas tertentu.

2.8 Pelacakan Proyek

- Jadwal proyek menyediakan peta jalan untuk manajer proyek perangkat lunak.
- Ini mendefinisikan tugas dan milestone(tonggak waktu).
- Beberapa cara untuk melacak jadwal proyek:
  - ✓ melakukan pertemuan status proyek secara berkala
  - ✓ mengevaluasi hasil tinjauan dalam proses perangkat lunak
  - ✓ menentukan apakah pencapaian proyek formal telah tercapai
  - ✓ membandingkan tanggal mulai aktual dengan tanggal mulai yang direncanakan untuk setiap tugas
  - ✓ pertemuan informal dengan praktisi
- Manajer proyek mengambil kendali jadwal dalam aspek:
  - ✓ kepegawaian proyek
  - ✓ masalah proyek
  - ✓ sumber daya proyek
  - ✓ ulasan
  - ✓ anggaran proyek

2.9 Manajemen risiko

- Risiko adalah masalah potensial - mungkin terjadi dan mungkin tidak
- Definisi konseptual risiko
    - Risiko menyangkut kejadian masa depan
    - Risiko melibatkan perubahan dalam pikiran, opini, tindakan, tempat, dll.
    - Risiko melibatkan pilihan dan ketidakpastian pilihan yang diperlukan
- Dua karakteristik risiko
    - Ketidakpastian - risiko mungkin atau tidak mungkin terjadi, yaitu, tidak ada risiko 100
    - Kerugian - risiko menjadi kenyataan dan konsekuensi atau kerugian yang tidak diinginkan terjadi

Kategori risiko

- Risiko proyek
    - Mengancam rencana proyek
    - Jika menjadi nyata, ada kemungkinan bahwa jadwal proyek akan tergelincir dan biaya akan meningkat
- Risiko teknis
    - Mengancam kualitas dan ketepatan waktu dari perangkat lunak yang akan diproduksi
    - Jika menjadi nyata, implementasi mungkin menjadi sulit atau tidak mungkin
- Risiko bisnis
    - Mengancam kelayakan perangkat lunak yang akan dibangun
    - Jika menjadi nyata, mereka mengancam proyek atau produk
- Resiko yang diketahui
    - Risiko-risiko yang dapat ditemukan setelah evaluasi yang cermat dari rencana proyek, bisnis dan
    - Lingkungan teknis di mana proyek sedang dikembangkan, dan informasi terpercaya lainnya sumber (mis., tanggal pengiriman yang tidak realistis)
- Risiko yang dapat diprediksi
    - Risiko yang dideduksi dari pengalaman proyek sebelumnya (misalnya, perputaran terakhir)

- Risiko yang tidak dapat diprediksi
    - Risiko-risiko yang dapat dan memang terjadi, tetapi sangat sulit untuk diidentifikasi sebelumnya

Sub Kategori risiko

- Risiko pasar - membangun produk atau sistem yang sangat baik yang tidak diinginkan oleh siapa pun
- Risiko strategis - membangun produk yang tidak lagi sesuai dengan strategi bisnis keseluruhan untuk perusahaan
- Risiko penjualan - membangun produk yang tidak dimengerti oleh penjual
- Risiko manajemen - kehilangan dukungan manajemen senior karena perubahan fokus atau perubahan orang-orang
- Risiko anggaran - kehilangan komitmen anggaran atau personil

Identifikasi resiko

- Satu metode untuk mengidentifikasi risiko adalah dengan membuat daftar periksa item risiko.
- Daftar periksa dapat digunakan untuk identifikasi risiko dan berfokus pada beberapa bagian yang diketahui dan dapat diprediksi risiko dalam subkategori umum berikut:
    - Ukuran produk — risiko yang terkait dengan ukuran keseluruhan perangkat lunak yang akan dibangun atau dimodifikasi.
    - Dampak bisnis — risiko yang terkait dengan batasan yang dikenakan oleh manajemen atau pasar.
    - Karakteristik pemangku kepentingan — risiko yang terkait dengan kecanggihan para pemangku kepentingan dan kemampuan pengembang untuk berkomunikasi dengan para pemangku kepentingan secara tepat waktu.
    - Definisi proses - risiko yang terkait dengan sejauh mana proses perangkat lunak telah ditetapkan dan diikuti oleh organisasi pengembangan.
    - Lingkungan pengembangan — risiko yang terkait dengan ketersediaan dan kualitas alat yang akan digunakan membangun produk.
    - Teknologi yang akan dibangun - risiko yang terkait dengan kompleksitas sistem yang akan dibangun dan "kebaruan" teknologi yang dikemas oleh sistem.

- Ukuran dan pengalaman staf — risiko yang terkait dengan keseluruhan pengalaman teknis dan proyek dari insinyur perangkat lunak yang akan melakukan pekerjaan.

2.10. Estimasi Risiko / Proyeksi Risiko

Langkah-Langkah Proyeksi Risiko

- Tentukan skala yang mencerminkan kemungkinan risiko yang dirasakan (mis., 1-rendah, 10-tinggi)
- Menjelaskan konsekuensi dari risiko
- Perkirakan dampak risiko pada proyek dan produk
- Perhatikan akurasi keseluruhan dari proyeksi risiko sehingga tidak akan ada kesalahpahaman

Mitigasi Risiko, Pemantauan, dan Manajemen

- Mitigasi risiko (perencanaan proaktif untuk menghindari risiko)
- Pemantauan risiko (menilai apakah risiko yang diprediksi terjadi atau tidak, memastikan langkah-langkah penghindaran risiko sedang terjadi diterapkan dengan benar, mengumpulkan informasi untuk analisis risiko masa depan, mencoba untuk menentukan risiko yang ditimbulkan)
- Manajemen risiko dan perencanaan kontingensi (tindakan yang harus diambil jika ada langkah-langkah mitigasi gagal dan risiko yang harus diambil)
- Tujuan dari mitigasi risiko, pemantauan dan rencana manajemen adalah mengidentifikasi kemungkinan potensi risiko.
- Ketika semua risiko telah diidentifikasi, selanjutnya akan dievaluasi untuk menentukan probabilitas kejadian
- Kemudian perencanaan dibuat untuk menghindari setiap risiko, untuk melacak setiap risiko apakah mungkin terjadi
- Merupakan tanggung jawab organisasi untuk melakukan mitigasi risiko, pemantauan, dan manajemen untuk menghasilkan produk yang berkualitas
- Semakin cepat risiko dapat diidentifikasi dan dihindari, semakin kecil kemungkinan harus menghadapi konsekuensi risiko tertentu.
- Semakin sedikit konsekuensi yang diderita akibat rencana RMMM yang baik, semakin baik produk, dan memperlancar proses pengembangan.

Mitigasi risiko

Untuk memitigasi risiko ini, Anda akan mengembangkan strategi untuk mengurangi perputaran. Di antara langkah-langkah yang mungkin terjadi diambil adalah:

- Bertemu dengan staf untuk menentukan penyebab turnover (misalnya, kondisi kerja yang buruk, upah rendah, dan pasar kerja yang kompetitif).
- Mitigasi penyebab yang ada di bawah kendali sebelum proyek dimulai.
- Setelah proyek dimulai, asumsikan turnover akan terjadi dan kembangkan teknik untuk memastikan kontinuitas ketika pegawai pergi.
- Mengorganisir tim proyek sehingga pengembangan informasi kegiatan tersebar luas.
- Menetapkan standar produk kerja dan menetapkan mekanisme untuk memastikan bahwa semua model dan dokumen dikembangkan secara tepat waktu.
- Lakukan peer review atas semua pekerjaan (sehingga lebih dari satu orang “up to speed”).
- Tetapkan anggota staf cadangan untuk setiap teknologi kritis.

Monitoring Risiko

- Manajer proyek memonitor faktor-faktor yang dapat memberikan indikasi apakah risiko akan terjadi.
- Dalam kasus pergantian staf yang tinggi, sikap umum anggota tim berdasarkan tekanan proyek, hubungan antar-pribadi di antara anggota tim, potensi masalah dengan kompensasi dan manfaat, dan ketersediaan pekerjaan di dalam perusahaan dan di luar itu semuanya dipantau.
- Selain memantau faktor-faktor ini, seorang manajer proyek harus memantau efektivitas risiko langkah-langkah mitigasi.
- Manajer proyek harus memantau produk kerja dengan hati-hati untuk memastikan bahwa masing-masing dapat berdiri sendiri dan masing-masing menanamkan informasi yang akan diperlukan jika seorang pendatang baru dipaksa bergabung tim perangkat lunak di tengah-tengah proyek.

Manajemen risiko

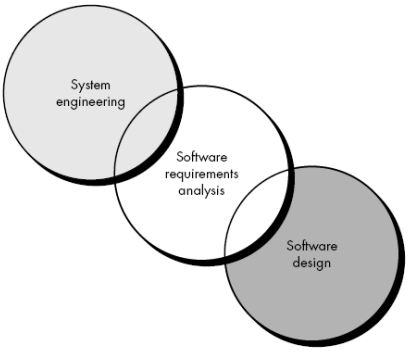
- Manajemen risiko dan perencanaan kontinjensi mengasumsikan bahwa upaya mitigasi telah gagal dan risiko telah menjadi kenyataan.
- Jika strategi mitigasi telah diikuti, cadangan tersedia, informasi didokumentasikan, dan pengetahuan telah tersebar di seluruh tim.
- Selain itu, dapat memfokuskan ulang sumber daya untuk sementara (dan menyesuaikan kembali jadwal proyek) ke fungsi-fungsi tersebut yang sepenuhnya memiliki staf, memungkinkan pendatang baru yang harus ditambahkan ke tim untuk "mendapatkan kecepatan."
- individu yang meninggalkan diminta untuk menghentikan semua pekerjaan dan menghabiskan minggu-minggu terakhir mereka dalam "transfer pengetahuan mode."

BAB III. KONSEP DAN PRINSIP ANALISIS

a. Analisis Kebutuhan Perangkat Lunak

Merupakan proses untuk menetapkan layanan-layanan (services) yang dibutuhkan customer dari sebuah system serta batasan-batasan (constraints) dalam pengoperasian sistem dan pengembangannya

Merupakan tugas dari rekayasa perangkat lunak yang menjembatani gap antara alokasi perangkat lunak di tingkat sistem dengan perancangan perangkat lunak.



Gambar 1. Analisis Kebutuhan Perangkat Lunak

Manfaat Analisa Kebutuhan

1. Membantu software engineers untuk memahami masalah dengan lebih baik.

2. Menghasilkan pemahaman tertulis (dokumentasi) dari masalah pelanggan.
3. Dimulai dengan kegiatan komunikasi berlanjut sampai pada kegiatan pemodelan.
4. Membuat kebutuhan jadi spesifik, jelas dan tidak ambigu
5. Menghasilkan Spesifikasi Kebutuhan Perangkat Lunak
6. Memberikan model yang digunakan sebagai dasar untuk tahap perancangan

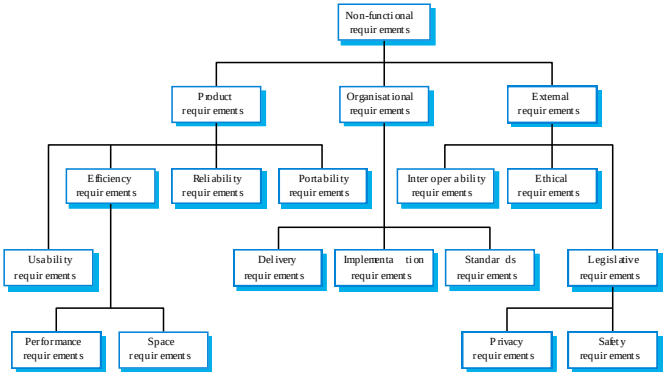
Pengertian Kebutuhan (Requirement)

Kebutuhan (requirement) merupakan Deskripsi/pernyataan dari layanan-layanan (services) yang dibutuhkan customer dari sebuah system serta batasan-batasan (constraints) dalam pengoperasian sistem dan pengembangannya. Kebutuhan perangkat lunak terdiri dari 2 jenis yaitu :

1. Kebutuhan fungsional (Functional Requirement)
   a. Aktivitas atau layanan yang diberikan oleh sistem. Berdasarkan pada produre atau fungsi bisnis.
   b. Menjelaskan fungsionalitas atau layanan system (system services).
   c. Tergantung pada type software, harapan user dan type dari system yang akan menggunakan software tersebut.
   d. Functional user requirements dapat berupa pernyataan secara umum mengenai apa yang harus dikerjakan system.
   e. Functional system requirements harus menjelaskan layanan system secara detail.

2. Kebutuhan Non fungsional (Non-Functional Requirement)
   Lingkungan operasional, performansi. Usability, reliability, dan security.
   - Product requirements
     Requirements yang menentukan bahwa produk yang dihasilkan harus berjalan sesuai dengan cara yang telah ditetapkan. Seperti : kecepatan eksekusi, reliability, dll.
   - Organisational requirements
     Requirements yang merupakan konsekuensi dari kebijakan dan prosedur organisasi seperti standar proses yang digunakan, kebutuhan implementasi , dll.
   - External requirements

Requirements yang muncul dari faktor-faktor di luar system dan proses pengembangannya seperti interoperability requirements, dan kebutuhan legislatif, dll.



Aktivitas dalam analisa kebutuhan yaitu :

a. Mengumpulkan informasi
b. mendefinisikan persyaratan sistem
c. Memprioritasken kebutuhan
d. Prototipe untuk kelayakan dan penemuan
e. menghasilkan dan mengevaluasi alternatif
f. Tinjau rekomendasi dengan manajemen

Kesulitan dalam analisa kebutuhan

a. Stakeholder sering tidak tahu apa yang dibutuhkan dari sistem komputer kecuali yang bersifat umum
b. Stakeholder sering menyatakan kebutuhan dengan istilah mereka sendiri dan pengetahuan yang implisit tentang kerja mereka sendiri
c. Stakeholder yang berbeda sering mengekspresikan kebutuhan dengan cara yang berbeda pula
d. Analisis bisa terjadi dalam konteks organisasi. Banyak faktor-faktor berpengaruh.
e. Lingkungan (ekonomi dan bisnis) yang senantiasa berubah.

Teknik Pengumpulan Kebutuhan Perangkat Lunak

a. Interview

Tim pengembang dan klien bertemu

Tipe

- Terstruktur → spesific, closed-ended question
- Tidak terstruktur → open-ended question

Perlu perencanaan yang baik

- Daftar kandidat yang akan diinterview
- Waktu interview
- Rencana interview untuk setiap kandidat

Tahapan interview

1. Memilih pihak yang akan diinterview (interviewee)
    - Idealnya semua pihak (stake holder) di interview.
    - Baik untuk melihat dari berbagai perspektif (sudut pandang)
2. Merancang pertanyaan interview
    - Terstruktur $\rightarrow$ spesific, pre-planned, closed-ended question
    - Tidak terstruktur $\rightarrow$ open-ended question
3. Persiapan interview
    - Mempersiapkan rencana interview (daftar pertanyaan, antisipasi jawaban dan follow-up).
    - Penetapan prioritas jika waktunya tidak cukup
    - Persiapan untuk interviewee (jadwal, alasan interview dan area diskusi untuk interview)
4. Melakukan interview
    - Profesional dan tidak bias.
    - Mencatat semua informasi.
    - Memastikan memahami semua issue
    - Pisahkan fakta dan opini.
    - Beri kesempatan interviewee untuk bertanya.
    - Pastikan mengucapkan terima kasih pada interviewee
    - Lakukan atau selesaikan tepat waktu.
5. Pasca interview (follow-up)
    - Mempersiapkan catatan
    - Mempersiapkan laporan

b. Kuisioner

Apabila banyak opini atau pendapat dibutuhkan.
Untuk mendapatkan informasi tertentu dari stake holder.
Tahapan

- Memilih partisipan.
- Merancang pertanyaan.
- Melakukan kuesioner.
- Melakukan Follow-up

c. Observasi

d. Analisis Dokumen

Memberikan petunjuk mengenai sistem yang ada (“as-is”)

Tipikal dokumen

- Forms
- Laporan
- Program komputer
- Prosedur dan user manual

Contoh:

Persyaratan Fungsional untuk Sistem Manajemen Hotel

1. Reservasi / Pemesanan

- Sistem harus mencatat reservasi.
- Sistem harus mencatat nama depan pelanggan.
- Sistem harus mencatat nama belakang pelanggan.
- Sistem harus mencatat jumlah penghuni.
- Sistem harus mencatat nomor kamar.
- Sistem akan menampilkan tarif kamar standar.
- Sistem harus mencatat nomor telepon pelanggan.
- Sistem akan menampilkan apakah ruangan itu dijamin atau tidak.
- Sistem harus menghasilkan nomor konfirmasi unik untuk setiap reservasi.
- Sistem harus mencatat tanggal dan waktu check-in yang diharapkan.
- Sistem harus mencatat tanggal dan waktu checkout yang diharapkan.
- Sistem akan memeriksa pelanggan.
- Sistem harus memungkinkan pemesanan dimodifikasi tanpa harus masuk kembali semua informasi pengguna.
- Sistem akan memeriksa pelanggan.
- Sistem akan menagih pelanggan untuk malam ekstra jika mereka keluar setelah jam 11:00 pagi
- Sistem akan menandai kamar yang dijamin sebagai "harus membayar" setelah pukul 18:00 pada saat check-in
- tanggal.
- Sistem harus mencatat umpan balik pelanggan.

2. Pemesanan makanan

- Sistem harus melacak semua makanan yang dibeli di hotel (restoran dan layanan kamar).
- Sistem harus mencatat pembayaran dan jenis pembayaran untuk makan.
- Sistem akan menagih kamar saat ini jika pembayaran tidak dilakukan pada saat layanan.

- Sistem akan menerima reservasi untuk restoran dan layanan kamar.

3. Manajemen

- Sistem akan menampilkan hunian hotel untuk jangka waktu tertentu (hari; termasuk masa lalu, sekarang, dan masa depan).
- Sistem harus menampilkan proyeksi hunian selama periode waktu (hari).
- Sistem akan menampilkan pendapatan ruangan untuk jangka waktu tertentu (hari).
- Sistem akan menampilkan pendapatan makanan untuk jangka waktu tertentu (hari).
- Sistem akan menampilkan laporan pengecualian, menunjukkan di mana harga kamar dan makanan default telah ditimpa.
- Sistem akan memungkinkan penambahan informasi, mengenai kamar, tarif, item menu, harga, dan profil pengguna.
- Sistem akan memungkinkan penghapusan informasi, mengenai kamar, tarif, item menu, harga, dan profil pengguna.
- Sistem akan memungkinkan untuk modifikasi informasi, mengenai kamar, tarif, menu item, harga, dan profil pengguna.
- Sistem akan memungkinkan manajer untuk menetapkan kata sandi pengguna

Persyaratan Non-Fungsional untuk Sistem Manajemen Hotel

- Waktu buka untuk layar antarmuka pengguna tidak lebih dari dua detik.
- Informasi login harus diverifikasi dalam lima detik.
- Query akan mengembalikan hasil dalam lima detik.
- Sistem Manajemen Hotel adalah sistem yang berdiri sendiri yang berjalan di lingkungan Windows.
- Sistem harus dikembangkan menggunakan platform Java.
- Harus ada konsistensi dalam nama-nama variabel dalam sistem.
- Antarmuka pengguna grafis harus memiliki tampilan dan nuansa yang konsisten.
- Tentukan faktor-faktor yang diperlukan untuk menetapkan keandalan sistem perangkat lunak yang diperlukan pada saat itupengiriman.
- Sistem harus tersedia selama jam operasional hotel normal.
- Perwakilan Layanan Pelanggan dan Manajer akan dapat masuk ke Sistem Manajemen Hotel.

- Perwakilan Layanan Pelanggan akan memiliki akses ke reservasi / pemesanan dan subsistem makanan.
- Manajer akan memiliki akses ke subsistem Manajemen serta Reservasi / Pemesanan dan Makanan subsistem.
- Akses ke berbagai subsistem akan dilindungi oleh layar login pengguna yang membutuhkan nama pengguna dan kata sandi.

BAB IV. Pemodelan Analisis

Merancang sistem komputerisasi adalah tugas pokok dari seorang *Systems Analyst*. Hasil rancangan tersebut selanjutnya akan ditindaklanjuti dengan pembuatan program aplikasi oleh *programmer.* Sistem komputerisasi yang telah dibuat selanjutnya akan diimplementasikan oleh *user*.

Pada kenyataannya, banyak sekali pertimbangan yang harus dilakukan seseorang dalam membuat sistem komputerisasi, misalkan spesifikasi *hardware* dan *software* (teknologi) apa saja yang dibutuhkan, berapa anggaran yang disediakan, siapa saja yang terlibat dan harus di*training*, waktu yang tersedia, dan sebagainya.

Karenanya, perancangan sistem komputerisasi akan melibatkan banyak orang di dalamnya. Hal ini mengharuskan dibuatnya '*master plan*,' '*blue print*,' atau skenario umum yang harus disepakati bersama terlebih dulu.

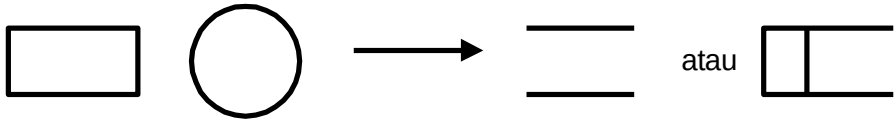
Catatan ini hanya memberikan sedikit gambaran dari perancangan sistem komputerisasi yang sangat rumit, yaitu hanya membahas tentang *Data Flow Diagram*, *Entity Relationship Diagram*, dan Normalisasi Data.

Pengantar DFD

DFD merupakan salah satu komponen dalam serangkaian pembuatan perancangan sebuah sistem komputerisasi. DFD menggambarkan aliran data dari sumber pemberi data (input) ke penerima data (output). Aliran data itu perlu diketahui agar si pembuat sistem tahu persis kapan sebuah data harus disimpan, kapan harus ditanggapi (proses), dan kapan harus didistribusikan ke bagian lain.

Komponen-komponen DFD

Komponen-komponen DFD terdiri atas :



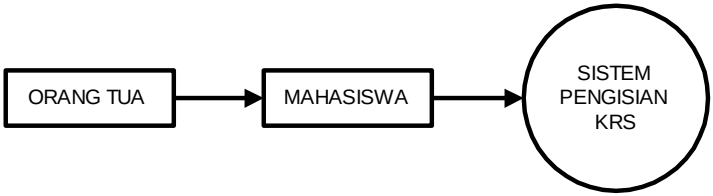
Terminator Proses Alur Data Penyimpan Data (*data store*)

Gambar 1. Komponen-komponen DFD

(1). Terminator

Terminator dapat disebut juga 'Kesatuan Luar,' yaitu suatu unit kerja/ jabatan, atau sejenisnya yang berada di luar sistem tetapi memberi andil atas pemberian atau penerimaan data dari sistem secara langsung. Terminator dapat pula disebut dengan 'Sumber Pemberi Data (input),' maupun 'Tujuan Pemberian Data (output).'

Pemberi data dan penerima data yang dimaksud adalah pihak yang sangat dekat dan memiliki hubungan langsung dengan sistem. Adapun pihak luar yang berhubungan dengan pihak luar lainnya tidak boleh digambarkan. Misalkan, dalam pengisian KRS, mahasiswa berhubungan dengan sistem. Orang tua berhubungan dengan mahasiswa, tetapi tidak berhubungan dengan sistem, karenanya, kesatuan luar 'orang tua', tidak boleh digambarkan.



Gambar 2. Contoh Hubungan Terminator yang Salah

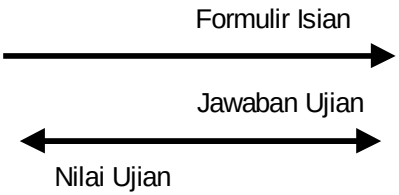
(2).Proses

Proses adalah suatu tindakan yang akan diambil terhadap data yang masuk. Karena proses adalah tindakan, maka proses berisi kata kerja, Proses diberikan identifikasi (nomor) agar mempermudah sekuen untuk diagram detilnya.

Gambar 3. Contoh Proses

(3). Alur Data

Alur data menggambarkan data yang mengalir dari terminator ke proses atau dari proses ke proses lainnya. Data yang dibawa oleh alur data harus disebutkan dan diletakkan di atas lambang alur data dan bila alur data digambar panjang, sebaiknya penulisan data mendekati lambang anak panahnya.



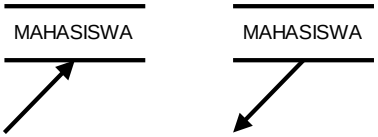
Gambar 4. Contoh Alur Data Searah dan Dua Arah

Data yang menempati alur data dapat berupa elemen data tunggal, maupun kumpulan elemen data. Misalkan, pada kumpulan elemen data : 'Jawaban Ujian', dapat ditulis secara lengkap dengan menyebutkan setiap elemen data yang ada di sana, yaitu : 'Lembar Jawaban', dan 'Naskah Soal'.

(4). Penyimpan Data (*Data Store*)

Data yang akan disimpan perlu ditempatkan ke satu tempat penyimpanan data. Data yang disimpan dapat berupa data manual maupun data digital. Untuk data digital, penyimpan data tersebut kelak akan dijadikan *file* data di komputer. Alur data yang anak panahnya menuju penyimpan data, kegiatannya adalah 'menulis/ merekam' data, sehingga isi *file* data akan berubah karenanya. Sedangkan alur data yang anak panahnya menuju ke proses dari penyimpan data, kegiatannya adalah 'membaca' data, sehingga isi *file* data tidak akan berubah karenanya.

Penyimpan data harus diberi nama, misalkan data yang berisi biodata mahasiswa diberi nama 'MAHASISWA'.



Gambar 5. Menulis dan Membaca data di Penyimpan Data

LEVELISASI DFD

DFD digambarkan secara bertingkat, dari tingkat yang global berturut-turut hingga tingkat yang sangat detil. Tingkat yang global (umum) disebut dengan 'Diagram Konteks' atau '*Context Diagram'*. Ini termasuk *level* 0.

Selanjutnya, dari diagram konteks, prosesnya dijabarkan lebih rinci lagi di 'Diagram Nol' atau '*Zero Diagram.*' Ini disebut *level* 1. Pada diagram nol ini yang berkembang hanya proses dan alur data yang menghubungkan proses-prosesnya, sedangkan jumlah terminator dan alur data yang masuk atau keluar dari terminator, tetap.

Bila, masih dirasakan perlu memerinci proses berikutnya, maka diagram selanjutnya disebut dengan 'Diagram Detil' atau 'Diagram primitif.' Ini disebut dengan *level* 2. Dalam diagram detil, yang digambar cukup proses (nomor berapa) yang perlu didetilkan saja, selain itu (proses lainnya, atau terminatornya)

tidak perlu digambarkan. Bila masih dapat lebih didetilkan lagi, maka *level* 3, dan seterusnya bisa dibuat.

Contoh Kasus

Di sebuah tempat penyewaan *Video Compact Disk* (VCD), masih dilakukan pencatatan manual untuk Penyewaan dan pengembalian VCD oleh Penyewa. Dalam kasus ini, akan dirancang sistem komputerisasi Penyewaan (saja) VCD tersebut.

Analisis

1. Pihak-pihak yang terkait :
   a. Penyewa;
   b. Pemilik usaha;
   c. Petugas.

   Petugas berada di dalam sistem (yang menjalankan sistem), sehingga tidak perlu digambarkan. Dari sini, terdapat 2 terminator, yaitu a dan b.

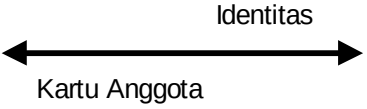
   1.a. Penyewa

   Data apa saja yang akan diberikan oleh Penyewa kepada sistem, dan data apa saja yang diberikan sistem kepada penyewa ?. Analisis ini bertujuan untuk menentukan data apa saja yang akan mengalir di alur data dari terminator Penyewa ke sistem (proses), dan sebaliknya.

   1.a.1. Penyewa Baru

   Penyewa baru (di kasus ini) harus membuat Kartu Anggota terlebih dulu. Pembuatan Kartu Anggota tidak dipungut biaya tetapi si Penyewa harus menunjukkan identitas diri (contoh : KTP).

   Petugas akan mencatat identitas Penyewa, membuatkan Kartu Anggota, dan bersama dengan KTP tersebut diserahkan kembali ke Penyewa.

   Proses manual bahwa KTP tersebut dikembalikan ke Penyewa tidak harus digambarkan di dalam arus data.

   1.a.2. Prosedur Penyewaan oleh Penyewa

   Penyewa yang akan meminjam film dipersilakan mencari sendiri filmnya, namun, bila mereka enggan mencarinya (tidak ketemu), mereka dapat langsung bertanya ke petugas. Petugas akan mengecek data film yang dicari dan akan dipinjam tersebut ke

file di komputer. Hasil pengecekan itu diinformasikan kepada Penyewa.

Bila film dicari ada dan mereka mau meminjamnya, maka si Penyewa harus menyerahkan Kartu Anggotanya (di lapangan, bisa saja hanya dengan menyebutkan identitasnya saja), dan uang sewanya

Adakalanya, petugas yang tidak yakin akan keanggotaan si Penyewa, dia melakukan cek keanggotaan ke *file* komputer. Bila ternyata data keanggotaannya tidak ada, maka si Petugas akan melakukan penolakan (pembatalan transaksi).

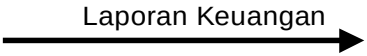
Bila benar anggota, maka Petugas akan mencatat data *film* yang dipinjam si Penyewa tersebut (transaksi) dan akan menyerahkan kembali Kartu Anggota dan film yang akan dipinjam tersebut ke Penyewa.

[Film | Informasi Penolakan] bisa ditulis : Film, Informasi Penolakan.

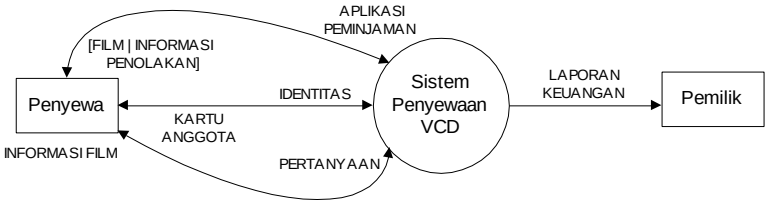
. 1.b. Pemilik Usaha (disingkat dengan Pemilik).

Apa saja data yang dibutuhkan oleh pemilik atas sistem, dan data apa saja yang diberikan oleh pemilik kepada sistem, perlu di analisis. Analisis ini akan menghasilkan alur data apa saja yang mengalir dari Terminator ke sistem dan sebaliknya.

Pada kasus ini, dicontohkan bahwa Pemilik hanya butuh laporan keuangan harian.



Dari analisis di atas, dapat dirancang DFD konteksnya :

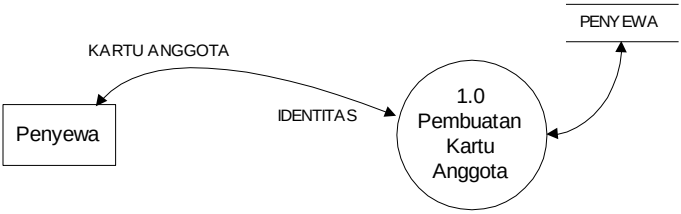


Gambar 6. DFD Konteks Kasus di Atas

"Aplikasi Peminjaman" yang tergambar di atas bisa saja ditulis secara detil, misalkan Bukti Keanggotaan, Uang Sewa, dan Daftar Film yang akan Disewa. "Identitas" boleh saja ditulis [KTP|SIM].
Sekali lagi, yang mengalir adalah data yang akan mempegaruhi proses komputerisasi, sedangkan untuk proses manualnya tidak perlu digambarkan. Misalkan, sewaktu akan meminjam film, Penyewa menyerahkan Kartu Anggota dan sewaktu menerima film, Kartu Anggota tersebut dikembalikan. Hal itu tidak perlu digambarkan.

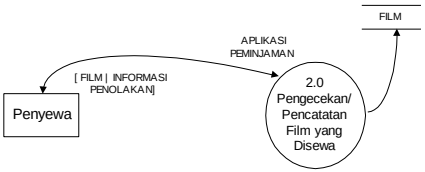
2. Pembuatan Diagram Nol (*Level* 1)
Diagram Nol adalah pengembangan proses yang lebih mendetil dari proses (sistem) yang ada di konteksnya. Jadi, jumlah terminator dan alur data yang masuk dan keluar dari terminator harus tetap.

2.1. Proses Pembuatan Kartu Anggota
Lihat poin 1.a.1. di atas. Gambar DFD-nya :

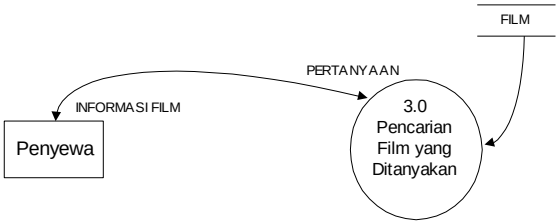


Gambar 7. Penggalan Diagram Nol

2.2. Proses Penyewaan VCD
Lihat poin 1.a.2. di atas. DFD-nya akan digambarkan sebagai :

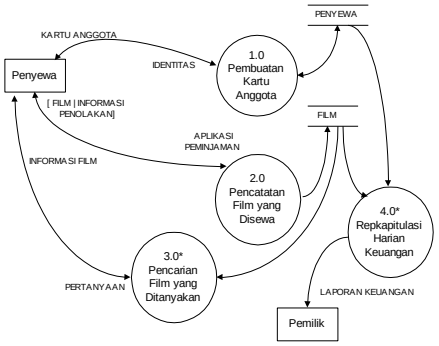


Gambar 8. Penggalan Diagram Nol
Proses Permintaan Informasi Keberadaan Film

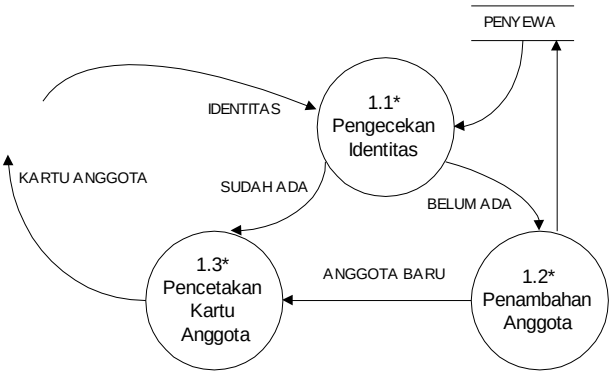


Gambar 9. Penggalan Diagram Nol

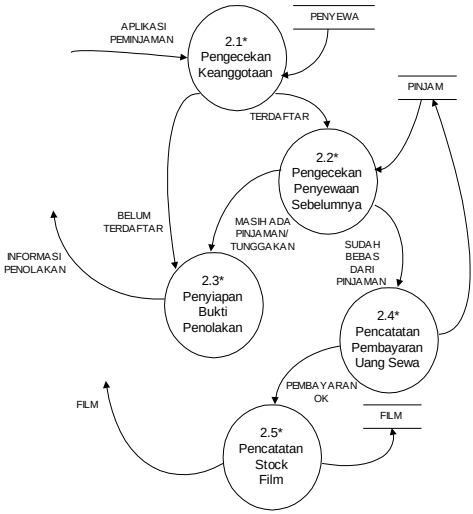
2.4. Gambar DFD *Zero* (*level* 1) Lengkapnya



Gambar 10. DFD *Level* 1 Kasus di Atas



Gambar 11. Diagram 1.0 Level 2



Gambar 12. Diagram 2.0 *Level* 2

Beberapa catatan tambahan :

(1) Pembuatan rancangan DFD harus sesuai dengan prosedur yang berlaku di tempat penelitian (jadi harus ada pembahasan mengenai prosedur yang berlaku, dan prosedur tersebut bukan penguji yang menentukan);

(2) Penggambaran DFD hendaknya dibuat sebaik mungkin (mudah ditelusuri, dan tidak rumit, misalkan dengan tidak adanya alur data yang bersilangan).

(3) Bila akan terjadi persilangan alur di penyimpan data, maka penyimpan data tersebut dapat digambar kembali dan diberi tanda '*' yang menandakan bahwa penyimpan data tersebut sama dengan nama penyimpan data sebelumnya (*copy*).

(4) Tanda '*' di nomor proses berarti proses tersebut tidak perlu didetilkan lagi.

3. Pembuatan Diagram Detil (*level* 2)
   Diagram detil perlu digambarkan bila masih ada suatu proses yang bisa dirinci lebih lanjut. Di sini dimisalkan penggambaran dari proses 1.0 (Pembuatan Kartu Anggota).

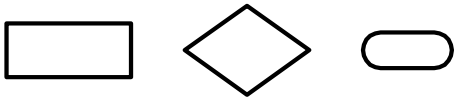
ENTITY RELATIONSHIP DIAGRAM (ERD)

ERD adalah gambaran mengenai berelasinya antarentitas. Sistem adalah kumpulan elemen yang setiap elemen memiliki fungsi masing-masing dan secara bersama-sama mencapai tujuan dari sistem tersebut. 'Kebersama-sama'-an dari sistem di atas dilambangkan dengan saling berelasinya antara satu entitas dengan entitas lainnya.

Entitas (*entity/ entity set*), memiliki banyak istilah di dalam ilmu komputer, seperti tabel (*table*), berkas (*data file*), penyimpan data (*data store*), dan sebagainya.

Komponen-komponen ERD

ERD memiliki komponen-komponen



Entitas Relasi Atribut

Gambar 13. Komponen-komponen ERD.

1. Entitas dan atribut.

   Seperti telah dijelaskan di atas, entitas adalah tempat penyimpan data, maka entitas yang digambarkan dalam ERD ini merupakan *data store* yang ada di DFD dan akan menjadi *file* data di komputer.

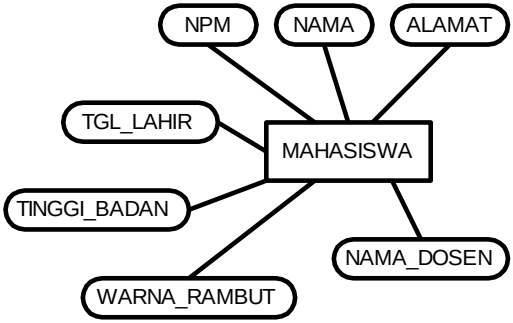
   Entitas adalah suatu objek dan memiliki nama. Secara sederhana dapat dikatakan bahwa jika objek ini tidak ada di suatu *enterprise* (lingkungan tertentu), maka *enterprise* tersebut tidak dapat berjalan normal. Contoh, entitas 'MAHASISWA' harus ada di lingkungan perguruan tinggi, begitu juga dengan entitas 'DOSEN', 'MT_KULIAH', dan sebagainya.

   Di dalam entitas 'MAHASISWA' berisi elemen-elemen data (biodata mahasiswa) yang terdiri atas NPM, NAMA, KELAS, ALAMAT, dan sebagainya. NPM, NAMA, KELAS, dan ALAMAT disebut dengan atribut (*field*).

   Apa saja atribut yang bisa menjadi ciri dari entitas, secara sederhana dapat dilakukan dengan melakukan pertanyaan logis. Misalkan, di entitas MAHASISWA ada atribut NILAI. Tanyakan ke mahasiswa, 'berapa nilai anda ?.' Tentu si mahasiswa akan berbalik tanya : 'nilai apa ?,' karena pertanyaan tersebut tidak dapat dijawab langsung, maka NILAI bukanlah atribut dari mahasiswa.

   Jika, pertanyaan yang diajukan dapat dijawab entitas secara logis dan benar, maka ia merupakan atributnya. Misalkan 'berapa tinggi badan anda ?,' maka tinggi badan adalah atribut dari mahasiswa, meskipun hal itu tidak perlu digunakan karena tidak berfungsi apa-apa dalam kemahasiswaannya.

   Jadi, ada atribut yang harus ada, ada atribut yang boleh ada, dan ada atribut yang tidak boleh ada, di dalam sebuah entitas. Contoh penggambaran entitas dan atributnya.



Gambar 14. Hubungan Atribut dan Entitasnya

Gambar 14. memperlihatkan bahwa atribut-atribut NPM, NAMA, ALAMAT, dan TGL_LAHIR harus ada di dalam biodata seorang mahasiswa. Atribut-atribut TINGGI_BADAN, dan WARNA_RAMBUT adalah atribut-atribut yang boleh tidak ada di dalam biodata mahasiswa (karena tidak penting). Sedangkan atribut NAMA_DOSEN adalah atribut yang tidak boleh ada di entitas mahasiswa.

Pada akhirnya, entitas ini akan menjadi *file* data (yang bersifat *master file*) di dalam komputer. *Master file* adalah *file* utama (yang harus ada, dan sifatnya jarang berubah).

2. Relasi

   Relasi adalah penghubung antara satu entitas (*master file*) dengan entitas lain di dalam sebuah sistem komputer. Pada akhirnya, relasi akan menjadi *file* transaksi (*transaction file*) di komputer.

   Secara kalimat logis, contoh relasi yang terjadi di sebuah perpustakaan adalah : "Anggota meminjam buku," atau "Anggota mengembalikan buku." Dalam hal ini, Anggota dan Buku adalah entitas, meminjam dan mengembalikan adalah transaksi (relasi antara anggota dan buku).

3. Derajat Kardinalitas (*Cardinality Degree*)

   Hubungan antarentitas ditandai pula oleh derajat kardinalitas. Fungsi dari derajat kardinalitas ini adalah untuk menentukan entitas kuat dan entitas lemah. Tiga jenis derajat kardinalitas adalah :

   (1) *One to one*, dilambangkan dengan 1 : 1
   (2) *One to many*, dan sebaliknya, yang dilambangkan dengan 1 : M dan sebaliknya *Many to man*
   (3) *Many to many*, dilambangkan dengan M : M atau M : N

Entitas dengan derajat kardinalitas 1 adalah entitas lemah sehingga entitas tersebut boleh digabung saja dengan entitas yang kuat (derajat kardinalitas M).

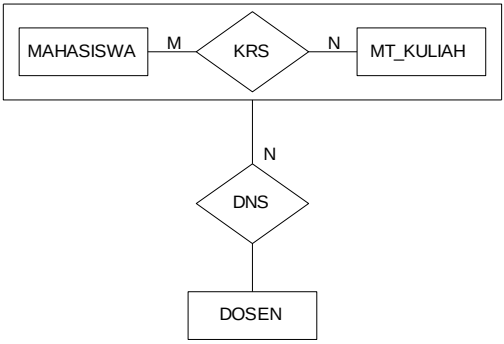
Misalkan kalimat bolak-balik berikut ini :

"Satu mahasiswa memiliki satu kelas"

"Satu kelas memiliki lebih dari satu (banyak) mahasiswa"

Kata (entitas) "KELAS" selalu disebut dengan kata "satu", sedangkan kata (entitas) "MAHASISWA" pernah disebut dengan lebih dari satu (banyak). Maka, di *file* MAHASISWA boleh berisi atribut KELAS, dan KELAS tidak perlu menjadi *file* sendiri.

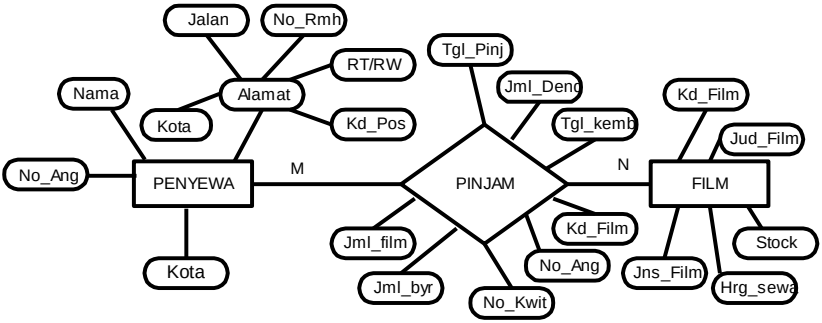
(4) Gabungan/ kombinasi ketiga bentuk di atas, misalkan *many to many to many*. Tapi, di sini tidak akan dibahas secara lebih lanjut. Contoh :



Gambar 15. Relasi *many to many to many*

Penggambaran ER dari kasus di atas (lanjutan dari DFD) dilakukan dengan cara :

(1) *Data Store* (penyimpan data) yang ada di DFD akan menjadi *Entity* di dalam ERD;
(2) Tentukan atribut-atribut (secara logika) yang harus ada di dalam setiap entitasnya;
(3) Tentukan serajat kardinalitasnya sesuai dengan peraturan yang berlaku di rental tersebut (dalam hal ini, setiap Penyewa boleh menyewa film lebih dari satu);
(4) Tentukan kunci atribut di setiap entitasnya.

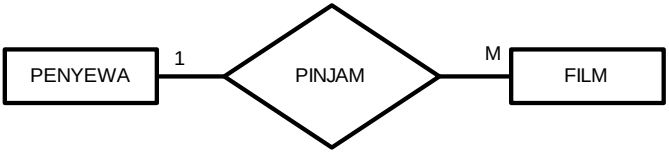


Gambar 16. ERD dari Kasus di Atas

Derajat kardinalitas yang terjadi adalah M : N (*many to many).* Mengapa tidak M : M, karena belum tentu "10 orang penyewa pasti menyewa 10 film." Karena tidak selalu M = M, maka dipilih M : N saja, jadi, suatu saat M boleh = N, dan di saat lain boleh M $\neq$ N.

Cara menentukan *many to many*-nya adalah dengan membuat dua kalimat bolak-balik. Derajat kardinalitas kalimat pertama diletakkan di atas, dan derajat kardinalitas kalimat kedua diletakkan di bawah.

Kalimat pertama : "satu orang penyewa boleh pinjam satu atau lebih (judul) film."



Kalimat kedua : "satu (judul) film boleh dipinjam oleh satu atau lebih penyewa"

4. Penentuan *Primary Key*

   Di setiap entitas di dalam ERD (di gambar 12 di atas), seharusnya ada atribut (*field*) yang dipilih untuk dijadikan kunci utama atribut (*primary key*/ *key field*), yaitu atribut yang dijadikan identitas yang menjamin keunikan (tidak ada yang sama) isi datanya.

   Misalkan, untuk entitas mahasiswa dipilih atribut NPM sebagai kunci utama atributnya karena tidak ada satupun mahasiswa yang memiliki NPM yang sama.

   Penulisan kunci utama atribut di dalam ERD harus dibedakan dengan atribut lainnya, misalkan dengan pemberian tanda '*' di depan nama atributnya, atau digarisbawahi atributnya.

   Secara logika, memang mudah menentukan sebuah atribut kunci, namun sesungguhnya, kunci utama diperoleh dari kunci kandidat, dan kunci kandidat diperoleh dari kunci super.

SUPER KEY

Super key adalah satu atau lebih *field* yang dapat dipilih untuk membedakan (mengkarakteristikkan) antara satu *record* dengan *record* lainnya. Bila *file*nya adalah MAHASISWA, maka satu atau lebih *field* yang dipilih agar dapat membedakan antara satu orang mahasiswa dengan mahasiswa lainnya.

NPM jelas bisa membedakan, NAMA juga bisa, namun dengan syarat tidak ada nama yang sama, gabungan NPM dan NAMA pasti bisa membedakan, apalagi gabungan NPM, NAMA dan TGL_LAHIR. Sehingga, *super key* bisa merupakan kombinasi dari satu atau gabungan *field* yang dapat mencirikan suatu *record*.

*Super key*nya : NPM

NAMA (dengan syarat tidak ada nama yang sama)
ALAMAT (dengan syarat alamat tidak ada yang sama)
TGL_LAHIR (dengan syarat tidak ada tanggal lahir yang sama)
NPM+NAMA
NPM+NAMA+ALAMAT
NPM+TGL_LAHIR
NPM+ALAMAT+TGL_LAHIR
dan berbagai kombinasi lainnya

CANDIDATE KEY

Kunci kandidat adalah kunci super dengan jumlah *field* paling sedikit, maka diperoleh : NPM, NAMA, ALAMAT, TGL_LAHIR (karena masing-masing hanya terdiri dari 1 *field* saja).

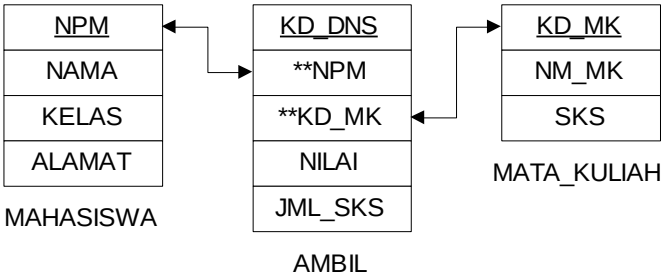
PRIMARY KEY

Kunci utama adalah kunci kandidat yang dipilih dengan kemungkinan kepemilikan nilai data *field* yang berbeda antara satu *record* dengan *record* lainnya. Maka dipilih NPM karena tidak ada mahasiswa yang memiliki NPM yang sama. Jelaslah, kunci utama pastilah merupakan kunci kandidat dan juga kunci super, tetapi sebaliknya, kunci super dan kunci kandidat belum tentu merupakan kunci utama.

ALTERNATE KEY

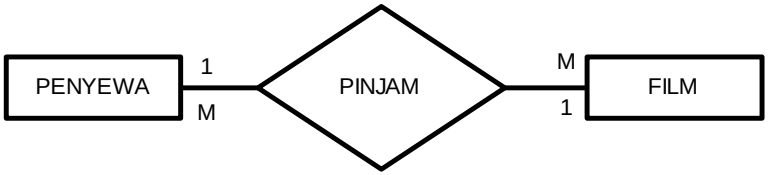
Kunci kandidat yang tidak terpilih menjadi kunci utama disebut dengan kunci alternatif.

Berikut, akan digambarkan di mana atribut (*field*) NILAI dimasukkan ke dalam suatu entitas (*file*) dan apa yang disebut dengan kunci tamu (*foreign key*).



Gambar 18. Hubungan Antarentitas

Di entitas MAHASISWA, *key field* yang dipilih adalah NPM;
Di entitas AMBIL, *key field* yang dipilih adalah KD_DNS;
Di entitas MATA_KULIAH, *key field* yang dipilih adalah KD_MK;
Di entitas AMBIL, yang merupakan *transaction file*, dimasukkan pula atribut NPM dan atribut KD_MK yang merupakan kunci-kunci utama dari entitas-entitas lain. Karenanya, NPM dan KD_MK di entitas AMBIL merupakan kunci tamu (*foreign key*)



Gambar 17. Penentuan *Many to Many*

Di antara derajat kardinalitas yang berada di atas dan bawah, dicari yang terbesar (yang kecil dihapus), maka akan didapatkan M : M yang pada akhirnya dilambangkan dengan M : N.

5. Penentuan *Primary Key*

   Di setiap entitas di dalam ERD (di gambar 12 di atas), seharusnya ada atribut (*field*) yang dipilih untuk dijadikan kunci utama atribut (*primary key*/ *key field*), yaitu atribut yang dijadikan identitas yang menjamin keunikan (tidak ada yang sama) isi datanya.

   Misalkan, untuk entitas mahasiswa dipilih atribut NPM sebagai kunci utama atributnya karena tidak ada satupun mahasiswa yang memiliki NPM yang sama.

Penulisan kunci utama atribut di dalam ERD harus dibedakan dengan atribut lainnya, misalkan dengan pemberian tanda '*' di depan nama atributnya, atau digarisbawahi atributnya.

Secara logika, memang mudah menentukan sebuah atribut kunci, namun sesungguhnya, kunci utama diperoleh dari kunci kandidat, dan kunci kandidat diperoleh dari kunci super.

SUPER KEY

Super key adalah satu atau lebih *field* yang dapat dipilih untuk membedakan (mengkarakteristikkan) antara satu *record* dengan *record* lainnya. Bila *file*nya adalah MAHASISWA, maka satu atau lebih *field* yang dipilih agar dapat membedakan antara satu orang mahasiswa dengan mahasiswa lainnya.
NPM jelas bisa membedakan, NAMA juga bisa, namun dengan syarat tidak ada nama yang sama, gabungan NPM dan NAMA pasti bisa membedakan, apalagi gabungan NPM, NAMA dan TGL_LAHIR. Sehingga, *super key* bisa merupakan kombinasi dari satu atau gabungan *field* yang dapat mencirikan suatu *record*.
*Super key*nya :

NPM
NAMA (dengan syarat tidak ada nama yang sama)
ALAMAT (dengan syarat alamat tidak ada yang sama)
TGL_LAHIR (dengan syarat tidak ada tanggal lahir yang sama)
NPM+NAMA
NPM+NAMA+ALAMAT
NPM+TGL_LAHIR
NPM+ALAMAT+TGL_LAHIR
dan berbagai kombinasi lainnya

CANDIDATE KEY

Kunci kandidat adalah kunci super dengan jumlah *field* paling sedikit, maka diperoleh : NPM, NAMA, ALAMAT, TGL_LAHIR (karena masing-masing hanya terdiri dari 1 *field* saja).

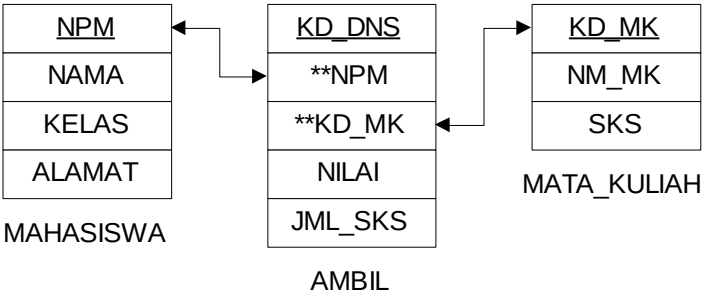
PRIMARY KEY

Kunci utama adalah kunci kandidat yang dipilih dengan kemungkinan kepemilikan nilai data *field* yang berbeda antara satu *record* dengan *record* lainnya. Maka dipilih NPM karena tidak ada mahasiswa yang memiliki NPM yang sama. Jelaslah, kunci utama pastilah merupakan kunci kandidat dan juga kunci super, tetapi sebaliknya, kunci super dan kunci kandidat belum tentu merupakan kunci utama.

ALTERNATE KEY

Kunci kandidat yang tidak terpilih menjadi kunci utama disebut dengan kunci alternatif.

Berikut, akan digambarkan di mana atribut (*field*) NILAI dimasukkan ke dalam suatu entitas (*file*) dan apa yang disebut dengan kunci tamu (*foreign key*).



Gambar 18. Hubungan Antarentitas

Di entitas MAHASISWA, *key field* yang dipilih adalah NPM;
Di entitas AMBIL, *key field* yang dipilih adalah KD_DNS;
Di entitas MATA_KULIAH, *key field* yang dipilih adalah KD_MK;

Di entitas AMBIL, yang merupakan *transaction file*, dimasukkan pula atribut NPM dan atribut KD_MK yang merupakan kunci-kunci utama dari entitas-entitas lain. Karenanya, NPM dan KD_MK di entitas AMBIL merupakan kunci tamu (*foreign key*).

NORMALISASI DATA (ND)

Normalisasi data adalah suatu proses/ prosedur/ cara yang menjamin sebuah data menjadi valid, dan efisien. Di dalam sistem basis data, ND juga berfungsi untuk meniadakan kerangkapan data (*redundancy*).

Banyak tahapan dalam ND, namun di sini hanya diperkenalkan 3 tahapan yang disebut dengan 1NF (*first normal form*), 2NF (*second normal form*), dan 3NF (*third normal form*).

1. *1NF/ First Normal Form* (Bentuk Normal Pertama)
   Pada tahap ini, setiap atribut diperiksa apakah sudah bersifat 'atomik', atau apakah atribut tersebut dalam penggunaannya kelak tidak perlu dibagi-bagi lagi. Contoh : Untuk penulisan atribut NAMA yang isi datanya adalah "George Washington".

   Apakah nama "George Washington" selamanya akan ditulis dengan "George Washington ? ", bila ya, maka atribut NAMA sudah atomik. Tetapi, adakalanya, "George Washington" akan dituliskan sebagai "Washington, George". Kalau demikian, bagaimana caranya mencetak nama "Washington, George" bila data nama yang disimpan adalah "George Washington" ?.

   Untuk itu, atribut NAMA harus dibagi lagi (karena belum atomik), menjadi NAMA1 yang berisi "George", dan NAMA2 yang berisi "Washington", sehingga untuk mencetak "Washington, George", cetak dulu NAMA2, baru NAMA1.

   Lihat gambar 16 di atas, atribut ALAMAT dibagi-bagi lagi menjadi JALAN, RT/RW, NO_RUMAH, dan KD_POS.

2. *2NF/ Second Normal Form* (Normalisasi Bentuk Kedua)
   Normalisasi bentuk kedua mensyaratkan bahwa 1NF sudah terpenuhi dan setiap atribut yang bukan merupakan kunci harus tergantung sepenuhnya dengan atribut kuncinya.

   Hal ini merupakan kelanjutan dari bahasan di ERD tentang entitas dan atribut. Misalkan, untuk data DOSEN, bila dipilih KD_DOSEN sebagai kunci atributnya, maka semua atribut yang ada harus tergantung secara fungsional dengan atribut KD_DOSENnya. Artinya, bila isi KD_DOSEN berganti, maka dosen (orangnya) harus merupakan orang yang lainnya.

   Contoh atribut yang salah (bila ada) di dalam entitas DOSEN yaitu NAMA_MHS. NAMA_MHS tidak tergantung pada KD_DOSEN.

3. *3NF/ Third Normal Form* (Normalisasi Bentuk Ketiga)
   Normalisasi bentuk ketiga mensyaratkan bahwa 2NF sudah terpenuhi dan setiap atribut yang bukan merupakan kunci tidak boleh tergantung dengan

atribut yang bukan kunci lainnya. Atau “menghilangkan ketergantungan transitif”.

Contoh kalimat ketergantungan transitif adalah : “Bila A tergantung B, dan B tergantung C, maka A akan tergantung pula oleh C”. Ini harus ditiadakan, menjadi A tergantung B, dan C juga tergantung B.

Contoh : dalam sebuah *file* ada *field* NPM, NAMA, TGL_LAHIR, KD_POS, KOTA dengan catatan KD_POS dan KOTA merupakan alamat dari mahasiswa.

NAMA, TGL_LAHIR, KD_POS, KOTA tergantung pada NPM-nya. Tetapi, KOTA tergantung pada KD_POS, jadi, susunan atribut ini tidak memenuhi syarat 3NF. Entitas ini harus dibagi menjadi 2, yaitu entitas MAHASISWA yang terdiri dari NPM, NAMA, TGL_LAHIR, KD_POS, dan entitas KODEPOS yang terdiri dari KD_POS dan KOTA.

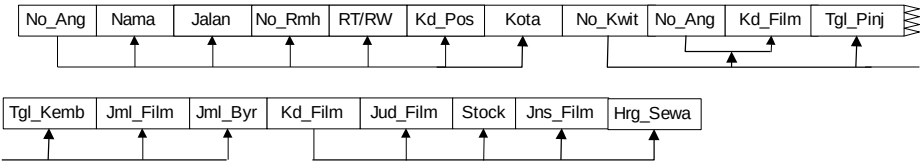
KD_POS di entitas MAHASISWA merupakan kunci tamu, dan KD_POS di entitas KODEPOS merupakan kunci utama.

Penyelesaian dari kasus di atas :

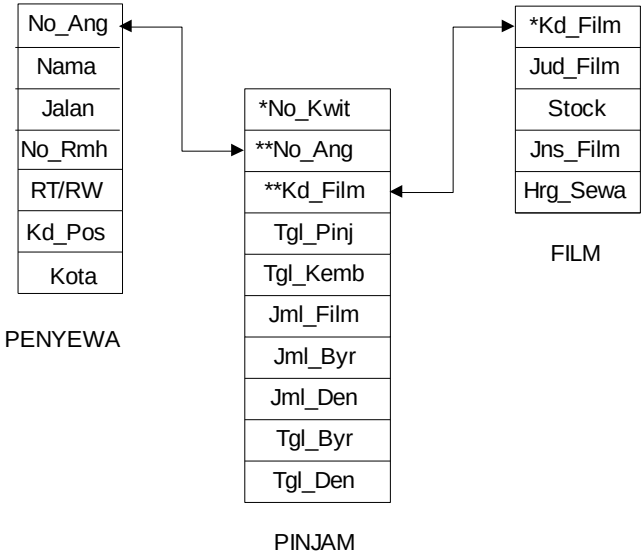
Belum normalisasi :

Hasil proses normalisasi bentuk pertama :

Proses normalisasi bentuk kedua :



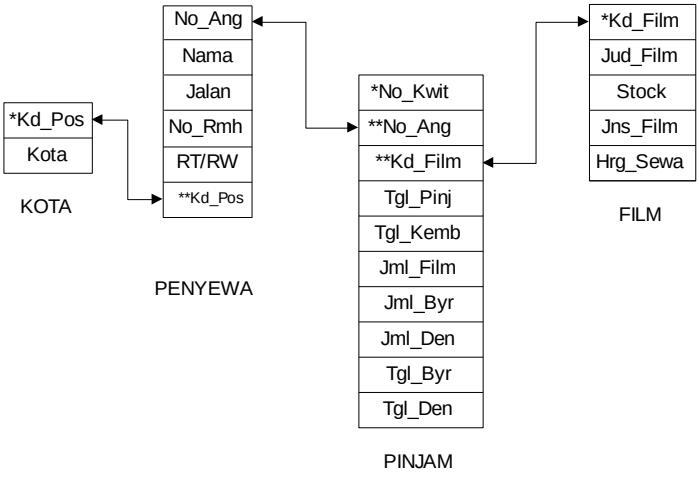
Hasil proses normalisasi bentuk kedua :



Gambar 19. Hasil Normalisasi Data Tingkat kedua

Hasil proses normalisasi bentuk ketiga :

Atribut ‘Kota’ di entitas Penyewa tergantung transitif kepada ‘Kd_Pos’. Karenanya, atribut ‘Kota’ dijadikan *file* sendiri dengan kunci atribut ‘Kd_Pos’.



Gambar 20. Hasil Normalisasi Data Tingkat Ketiga

4. *File-file* data yang digunakan

Dari hasil proses normalisasi data di atas, maka terbentuk tabel-tabel yang di komputer akan menjadi *file-file* data yang akan digunakan sebagai tempat menyimpan data. *File-file* data yang terbentuk tadi akan disesuaikan formatnya dengan bahasa pemrograman yang akan digunakan.

Dalam data base, pembuatan tabel tersebut dilakukan dengan DDL (*data definition language*) untuk melakukan *create table*, dan pemasukan datanya dilakukan dengan DML (*Data Manipulation Language*) untuk melakukan *insert* data. Contoh bahasa yang digunakan adalah SQL atau DB2.

Contoh bila dalam bahasa pemrograman *dBase* :

1. Nama *File* : PENYEWA.DBF

Nama *Field*	Jenis *Field*	Panjang	*Index*
NO_ANG	*Characters*	6	Y
NAMA	*Characters*	20	
JALAN	*Characters*	15	
NO_RUMAH	*Characters*	5	
RT_RW	*Characters*	8	
Kd_Pos	*Characters*	5	

2. Nama *File* : PINJAM.DBF

Nama *Field*	Jenis *Field*	Panjang	*Index*
NO_KWIT	*Characters*	6	Y
NO_ANG	*Characters*	6	
KD_FILM	*Characters*	6	
TGL_PINJ	*Characters*	10	
TGL_KEMB	*Characters*	10	
JML_FILM	*Numeric*	2	
JML_BYR	*Numeric*	6	
JML_DEN	*Numeric*	6	
TGL_BYR	*Characters*	10	
TGL_DEN	*Characters*	10	

3. Nama *File* : FILM.DBF

Nama *Field*	Jenis *Field*	Panjang	*Index*
KD_FILM	*Characters*	6	Y
JUD_FILM	*Characters*	20	
STOCK	*Numeric*	2	
JNS_FILM	*Characters*	15	
HRG_SEWA	*Numeric*	5	

4. Nama *File* : KOTA.DBF

Nama *Field*	Jenis *Field*	Panjang	*Index*
KD_POS	*Characters*	5	Y
KOTA	*Characters*	15	

5. Tambahan : Kamus Data (*Data Dictionary*)

Pendiagraman (DFD, ERD, Normalisasi) yang dilakukan *System Analyst* akan dibuatkan programnya oleh *programmer* dan selanjutnya akan diimplementasikan oleh para *user*. Untuk membuat persepsi yang sama atas sebuah data dari seluruh orang yang terlibat dalam sistem

komputerisasi, maka perlu dibuatkan kamus data, yaitu data yang menjelaskan tentang data (meta data).

Data yang dijelaskan dalam kamus data dapat berupa : setiap elemen data, alur data, penyimpan data, dan sebagainya yang ada di perancangan sistem.

Lambang-lambang yang digunakan dalam kamus data :

a. *Assignment*, contoh :A = 5000
b. *Concatenation*, contoh :NAME + ADDR
c. *Selection*, contoh : [MALE | FEMALE]
d. *Repetition*, contoh : $_{10}${player-name}^{15}
e. *Optional*, contoh : (MASCOT)
f. *Data Stores*, contoh :LEAGUE = {TEAM-NAME + MANAGER + COACH}

g. *Comments*, contoh : RULES = *Explanation provided in narrative text.

Contoh kamus data :

DOSEN : (KD_DOSEN + NM_DOSEN + JNS_KMIN + TGL_LHR)
JNS_KMIN : ["P" | "W"]
KD_DOSEN : * Kode Dosen*
NM_DOSEN : * Nama Dosen*
TGL_LAHIR: * Tanggal Lahir, format : dd/mm/yyyy *

KESIMPULAN :

Mata kuliah Perancangan Sistem ini bertujuan untuk membuat sistem komputerisasi, di mana di dalamnya terdapat *file-file* yang merupakan hasil dari perancangan (*data store* di DFD, dan *Entity* di ERD).

Adapun proses atau prosedur manualnya tidak perlu digambarkan, misalkan, bagaimana memilih orang-orang yang akan duduk di jabatan-jabatan perancangan sistem, proses pengawasan dari pimpinan ke "anak buah"nya, dan semacamnya tidak perlu dijelaskan, hal itu akan dipelajari di mata kuliah Manajemen Proyek Sistem Informasi, dan sejenisnya.

Di pelajaran ini yang perlu dipikirkan hanyalah, bagaimana membentuk *file-file* dalam satu kesatuan (sistem), bagaimana agar *file-file* tersebut dapat berdaya-guna secara efisien dan memenuhi aturan-aturan sebuah perancangan yang baik.

UML (*Unified Modelling Language*)

UML (*Unified Modelling Language*) adalah sebuah bahasa standart dalam industri yang biasa digunakan sebagai visualisasi, perancangan, dan pendokumentasian sebuah sistem perangkat lunak. Ketika *Chnoles (2003)* menyatakan bahwa UML adalah bahasa, maka UML memiliki *sintax* dan semantik. Hal ini yang menandakan bahwa jika dalam pemvisualisasian atau perancangan dan pendokumentasian perangkat lunak menggunakan konsep UML, maka harus mengikuti aturan – aturannya. Konsep UML mengatur bagaimana elemen – elemen pada setiap model yang kita buat saling berhubungan satu sama lain. Bagaimana cara kerja perangkat lunak, siapa saja yang memiliki hak akses terhadap perangkat lunak, bagaimana sistem bekerja jika terdapat error, bagaimana keamanan datanya, dimana datanya akan tersimpan, semua itu diatur, divisualisasikan, dirancang, didokumentasikan dalam UML. UML dikembangkan sebagai suatu alat yang digunakan sebagai analisis dan desain yang berorientasi pada *object* oleh Grady Booch, James Rumbaugh, dan Ivar Jacobson.

Adapun fungsi dari UML ini adalah :

1. Termasuk dalam tahap awal peancangan perangkat lunak
2. Metode yang digunakan oleh Seorang Sistem Analis mendesain dan menganalisa sebuah sistem atau perangkat lunak
3. Merupakan sarana komunikasi yang menguhubungkan perangkat lunak dengan proses bisnis
4. Digunakan untuk menjabarkan secara rinci sebuah sistem perangkat lunak
5. Mendokumentasi segala sesuatu yang berhubungan dengan sistem perangkat lunak

Tidak ada batasan yang jelas antara aneka ragam konsep dan kosntruksi di dalam UML, tapi untuk pemahaman yang lebih mudah, UML dibagi menjadi beberapa *view* atau pandangan. *View* atau pandangan adalah bagian yang simpel dari konstruksi pemodelan UML yang merepresentasikan aspek dari sebuah sistem. Pembagian menjadi *view* atau pandangan yang berbeda bukanlah sesuatu yang baku tergantung dari kebutuhan, tapi diharapkan dengan adanya *view* akan memudahkan kosntruksi UML. Satu atau lebih diagram merepresentasikan konsep notasi visual pada setiap *view* atau padangan. Pada level atas, view atau pandangan dapat dibagi menjadi tiga area :

1. Klasifikasi struktural (structural clasification) : mendeskripsikan hubungan segala hal yang ada di dalam system

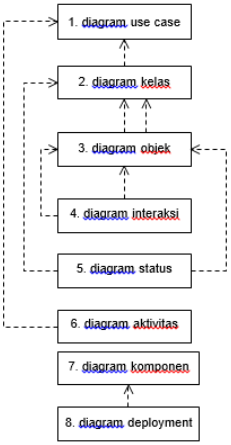
2. Kelakuan dinamik (dynamic behavior) : Mendeskripsikan kelakuan sistem, atau urutan perubahan yang dialami system
3. Pengelolaan model (model management) : mendeskripsikan keterkaitan organisasi dengan hirarki unit yang ada di dalam sistem

Berikut adalah keterkaitan antara *view* dan diagram di dalam UML:

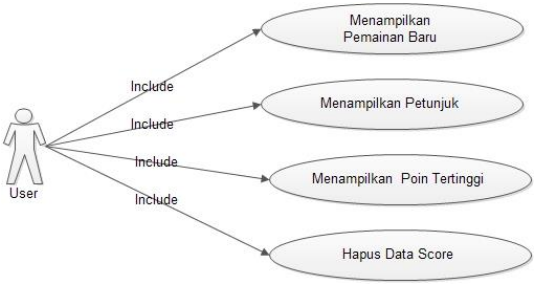
Area Mayor	**View**	**Diagram**
struktural	static view view atau pandangan yang tidak bergantung pada waktu	**diagram kelas**
	use case view view atau pandangan dari segi fungsionalitas sistem	**diagram use case**
	implementation view View atau pandangan dari segi komponen implementasi sistem	**diagram komponen**
	deployment view view atau pandangan dari segi node tempat komponen di- deploy	**diagram deployment**
dinamik	state status yang dialami sistem berdasmachine view view atau pandangan dari segi arkan objek-objek sistem	**diagram status**
	activity view view atau pandangan dari segi aktivitas yang dilakukan oleh sistem	**diagram aktivitas**
	Diagram interaksi	**diagram sekuen** **diagram kolaborasi**
pengelolaan model (model-management)	**model-management view** **view atau pandangan dari segi pengelolaan model sistem**	**diagram kelas**

Langkah-langkah pembuatan UML :

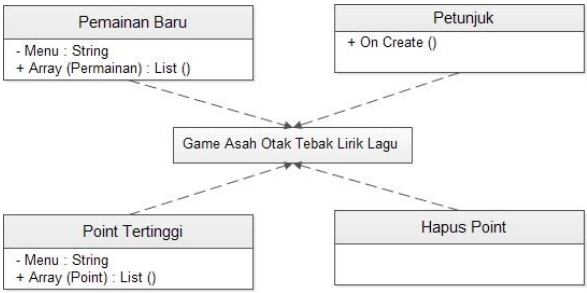
UML merupakan diagram yang saling terkait oleh karena itu perlu adanya kekonsistenan rancangan diagram yang satu dengan lainnya, bukan asal menggambar.
Berikut adalah keterkaitan diagram-diagram pada UML beserta urutan pembuatannya.



Dari tanda panah pada gambar diatas menjelaskan keterkaitan antar diagram, seperti contoh : pembuatan Diagram Class (*Class Diagram*) terkait dengan Diagram *Use Case* (*Use Case Diagram*), Diagram Objek (*Object Diagram*) terkait dengan Diagram Class. Sebagai studi kasus :
Sebuah aplikasi *game* atau permainan dapat menampilkan apa saja yang dibutuhkan oleh penggunanya,mulai dari menampilkan layar permainan baru, menampilkan petunjuk permainan, menampilkan point tertinggi dari seluruh pengguna hingga menghapus point. Berikut tampilan *use case diagram* sebuah *game.*



Berdasarkan *use case diagram* diatas, maka pembuatan *class diagram* yang merupakan *break down* atau turunan *uses case diagram* adalah sebagai berikut :



Penjelasan mengenai *use case diagram* dan *class diagram* serta diagram UML lainnya akan dibahas pada dibawah ini.

Class Diagram

Diagram kelas atau class diagram menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem . Kelas memiliki apa yang disebut atribut dan metode atau operasi.
Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas
Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas

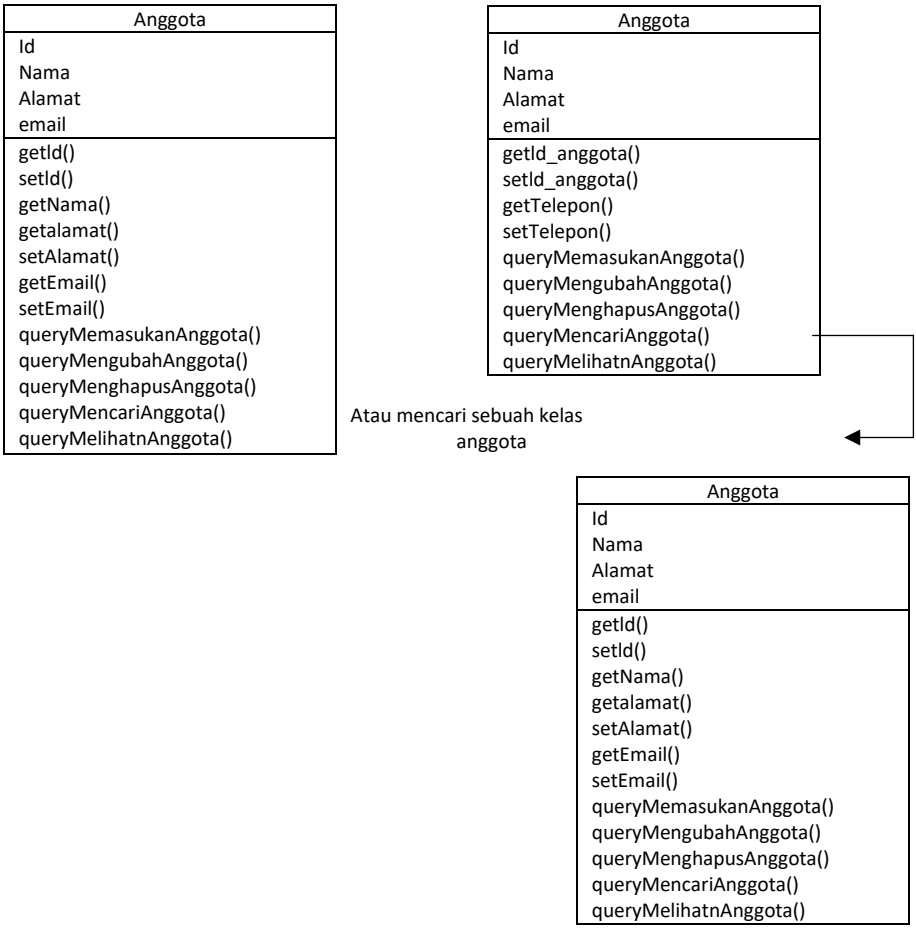
Diagram kelas dibuat agar pembuat agar program atau programer membuat kelas-kelas sesuai rancagan di dalam diagram kelas agar anatara dokumentasi perancangan dan perangkat lunak sinkron. Banyak berbagai kasus, perancangan kelas yang dibuat tidak sesuai dengan kelas-kelas yang dibuat pada perangakat lunak, sehingga tidaklah ada gunanya lagi sebuah perancangan karena apa yang dirancang dan hasil jadinya tidak sesuai.

Kelas-kelas yang ada struktur sistem harus dapat melakukan fungsi-fungsi sesuai dengan kebutuhan sistem sehinga pembuat perangkat lunak atau programmer dapat membuat kelas-kelas di dalam program perangkat lunak sesuai dengan perancangan digram kelas. Susunan struktur kelas yang baik pada diagram kelas sebaiknya memiliki jenis-jenis kelas berikut :

- Kelas main
  Kelas yang memiliki fungsi awal dieksekusi ketika sistem dijalankan
- Kelas yang menangani tampilan sistem (view)
  Kelas yang mendefinisikan dan mengatur tampilan ke pemakai
- Kelas yang diambil dari pendefinisi use case (controller)
  Kelas yang menangani fungsi-fungsi yang harus diamabil dari definisian use case , kelas ini biasanya disebut dengan kelas proses yang menangani proses bisnis pada perangakat lunak
- Kelas yang diambil dari pendefinisian data (model)

Kelas yang digunakan untuk memegang atau membungkus data menjadi sebuah kesatuan yang diambil maupun akan disimpan ke basis data . Semua tabel yang dibuat di basis data dapat dijadikan kelas , namun untuk tabel dari hasil relasi atau atribut multivalue pada ERD dapat dijadikan kelas tersendiri dapat juga tidak asalkan pengaksesanya dapat dipertanggungjawabkan atau tetap di dalam perancangan kelas . Misalkan dalam tabel Telepon dan Anggota pada studi kasus maka perancangan kelas hanya terdiri dari kelas anggota di mana di dalamnya ada sebuah atribut berupa larik (array) bertipe string dengan nama telepon.

Ilustrasinya dapat dilihat pada gambar berikut :



Hal terpenting adalah kolom telepon tetap dapat diakses dari perancangan kelas . Kelasdata biasanya adalah kelas yang terkait dengan pengaksessan tabel pada basis data sesuai dengan nama kelasnya , misalnya kelas anggota maka akan digunakan untuk mengakses tabel yang menyimpan data anggota .

Jenis-jenis anggota di atas juga dapat dihubungkan satu sama lain sesuai dengan pertimbangan yang dianggap baik asalkan fungsi-fungsi yang sebaiknya ada pada struktur kelas tetap ada. Susunan kelas juga dapat ditambahkan kelas utilitas (utility class) seperti koneksi ke basis data , membaca file teks , dan lain sebagainya sesuai kebutuhan .

Dalam mendefinisikan metode yang ada di dalam kelas perlu memperhatikan apa yang di sebut dengan cohesion dan coupling . Cohesion adalah ukuran berapa dekat keterkaitan intruksi di dalam sebuah metode terkait satu sama lain sedangkan coupling adalah ukuran seberapa dekat keterkaitan intruksi antara metode yang satu dengan metode yang lain dalam sebuah kelas . Sebagai aturan secara umum maka sebuah metode yang dibuat harus memiliki kadar cohesion yang kuat dan kadar coupling yang lemah .

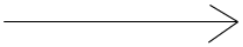
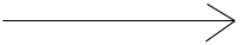
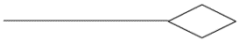
Perhatian jugak jika sebuah kelas hanya terdiri dari atribut saja , atau hanya metode . Berikut pertimbangan dalam membuat kelas :

Pertimbangan	Keterangan
Sebuah kelas yang hanya berisi sebuah atribut	Kelas seperti ini kurang efisien sehingga perlu di pikirkan kembali perancangan yang dilakukan , karena secara fiisik kelas dengan satu atribut adalah sebagai berikut : Sebuah berkas atau file hanya berisi seperti di atas tentu tidak efisien. Cara memperbaiki rancangan adalah dengan menggabungkan satu atribut tersebut ke kelas lain yang memiliki keterkaitan lebih dekat
Sebuah kelas yang hanya berisi atribut	Kelas seperti ini kurang efisien sehingga perlu di pikirkan kembali perancangan yang dilakukan , karena secara fisik kelas dengan hanya berisi atribut saja adalah sebagai berikut : Sebuah berkas atau file berisi seperti di atas tentu saja tidak efisien.

	Cara memperbaiki rancangan adalah dengan menggabungkan atribut tersebut ke kelas lain yang memiliki keterkaitan lebih dekat .
Sebuah kelas yang hanya berisi sebuah metode	Kelas seperti ini kurang efisien sehingga perlu dipikirkan kembali perancangan yang dilakukan , karena secara fisik kelas dengan hanya berisi atribut saja adalah sebagai berikut : Sebuah berkas atau file hanya berisi seperti di atas tentu saja tidak efisien Cara memperbaiki rancanagan adalah dengan menggabungkan atribut tersebut ke kelas lain yang memiliki keterkaitan lebih dekat .
Sebuah kelas yang hanya berisi sebuah metode	Kelas seperti ini kurang efisien sehingga penuh dipikirkan kembali perancangan yang dilakukan , karena secara fisik kelas dengan hanya berisi sebuah metode saja adalah sebagai berikut : Sebuah berkas atau file hanya berisi seperti di atas tentu saja tidak efisien Cara memperbaiki rancangan adalah menggabungkan atribut dan metode tersebut ke kelas lain yang memiliki keterkaitan lebih dekat.

Berikut adalah simbol-simbol yang ada pada diagram kelas :

Simbol	Deskripsi
Kelas **nama_kelas** +atribut +operasi	Kelas pada struktur sistem
Antarmuka / interface  **nama_interface**	Sama dengan konsep interface dalam pemrogrman berorientasi objek
Asosiasi / association	Relasi antar kelas dengan makna umum , asosiasi biasanya juga disertai dengan multiplicity
Asosiasi berarah / directed association	Relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain , asosisasi biasanya juga disertai dengan multiplicity

	
Generalisasi 	Relasi antar kelas dengan makna genelisasi-spesialisasi (umum khusus)
Kebergantungan / dependency 	Relasi antar kelas dengan makna kebergantungan antar kelas
Agregasi / aggregation 	Relasi antarkelas dengan makna semua-bagian (whole-part)

Object Diagram

Diagram objek menggambarkan struktur sistem dari segi penamaan objek dalam sistem. Pada diagram objek harus dipastikan semua kelas yang sudah didefinisikan pada diagram kelas harus di pakai objeknya, karena jika tidak, pendefinisian kelas itu tidak dapat dipertanggungjawabkan. Diagram objek juga berfungsi untuk mendefinisikan contoh nilai atau isi dari atribut tiap kelas.

Untuk apa mendefinisikan sebuah kelas sedangakan pada jalannya sistem, objeknya tidak pernah di pakai. Hubungan link pada diagram objek merupakan hubungan memakai dan dipakai dimana dua buah objek akan dihubungkan oleh link jika ada objek yang dipakai oleh objek lainya.

Berikut adalah simbol-simbol yang ada pada diagram objek :

Simbol	**Deskripsi**
nama_objek : nama_kelas Atribut = nilai Objek	Objek dari kelas yang berjalan saat sistem dijalankan
Link	Relasi antar objek

Penulisan nilai sama dengan penulisan pada kamus data DFD. Misalkan jika sebuah atribut dapat berisi lebih dari satu string maka akan ditulis denan lambang {nilai1, ..., nilain}, atau misalnya ingin menulis bahwa nilai sebuah atribut string dapat diisi nilai1 atau niali2 maka akan ditulis [naiali1| nilai2].

Component Diagram

Diagram komponen atau component diagram dibuat untuk menunjukan organisasi dan ketergantungan diantara kumpulan komponen dalam sebuah sistem. Diagram komponen fokus pada komponen sistem yang dibutuhkan dan ada di dalam sistem. Diagram komponen juga dapat digunakan untuk memodelkan hal-hal berikut:

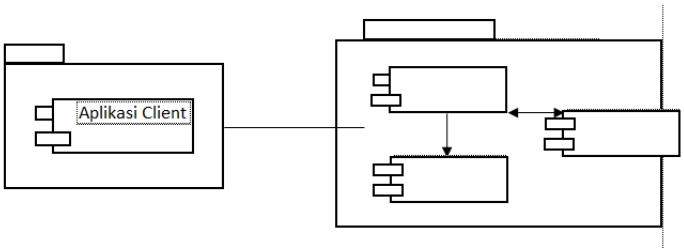
Source code program perangkat lunak

Komponen executable yang di lepas ke user

Basis data secara fisik

Sistem yang harus beradaptasi dengan sistem lain

Framework sistem, framework pada perangakat lunak merupakan kerangka kerja yang dibuat untuk memudahkan pengembangkan dan pemeliharaan aplikasi , contohnya seperti Struts dari Apache yang menggunakan prinsip desain Model-View-Controller (MVC) dimana source code program dikelompokan berdasarkan fungsinya seperti pada gambar berikut :



Gambar Ilustrasi Farmework

Dimana controller berisi source code yang menangani request dan validasi, model berisi source code yang menangani manipulasi data dan business logic, dan view berisi berisi source code yang menangani tampilan .

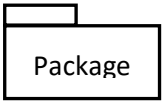
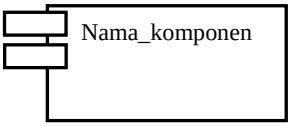
Komponen dasar yang biasanya ada dalam suatu sistem adalah sebagai berikut :

Komponen user interface yang menangani tampilan

Komponen bussines processing yang menangani fungsi-fungsi proses bisnis

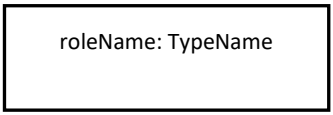
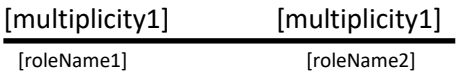
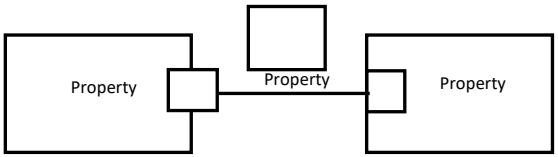
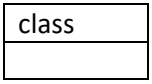
Komponen data yang menangani manipulasi data

Komponen security yang menangani keamanan sistem
Komponen lebih terfokus pada penggolongan secara umum fungsi-fungsi yang diperlukan.
Berikut adalah simbol-simbol yang ada pada digram komponen:

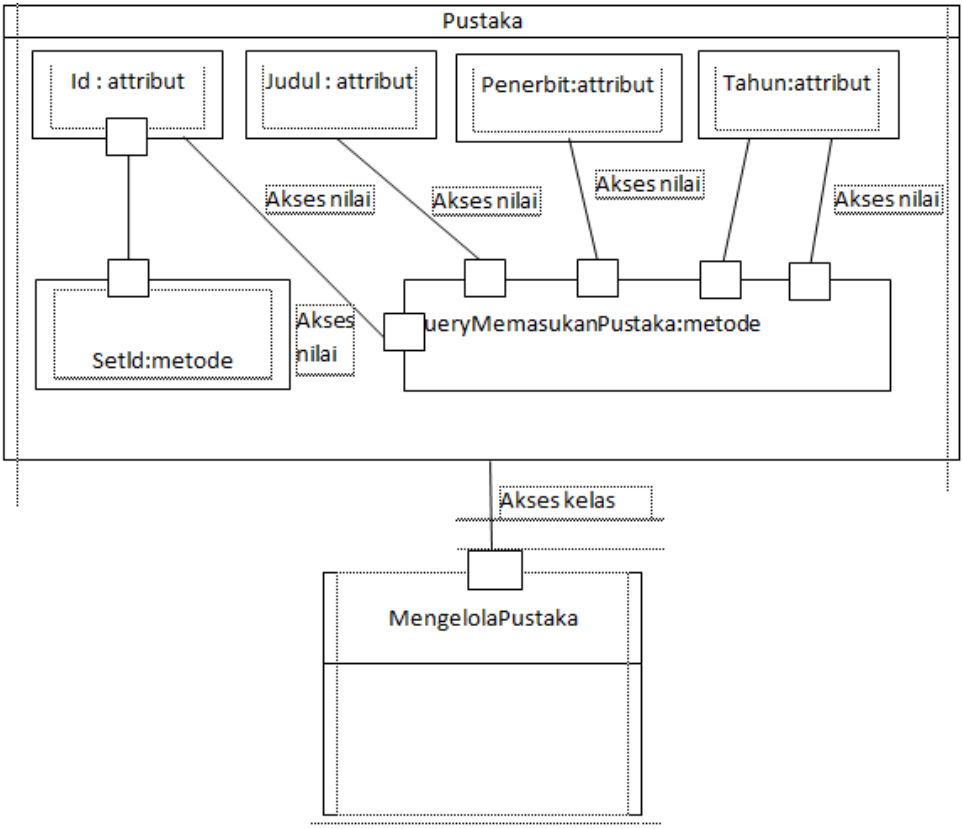
Simbol	**Deskripsi**
Package 	Package merupakan sebuah bungkusan dari satu atau lebih komponen
Komponen 	Komponen sistem
Kebergantungan / dependency 	Kebergantungan anatar komponen , arah panah mengarah pada komponen yang di pakai
Antarmuka / interface  **nama_interface**	Sama dengan konsep interface pada pemrograman berorientasi objek , yaitu sebagai anatar muka komponen agar tidak mengakses langsung komponen
Link	Relasi antar komponen

Composite Structure Diagram
Composite Structure Diagram baru mulai pada UML versi 2.0 , pada versi 1.x diagram ini belum muncul . Diagram ini dapat digunakan untuk menggabarkan struktur dari bagian-bagian yang saling terhubung maupun mendeskripsikan strukutur pada saat berjalan (runtime) dari instance yang saling terhubung . Dapat menggabarkan struktur didalam kelas atau kolaborasi . Contoh penggunaan diagram ini misalnya untuk menggambarkan deskripsi dari setiap bagian mesin yang saling terkait untuk menjalankan fungsi mesin tersebut , menggambarkan aliaran data router pada jaringan komputer , dan lain-lain .

Berikut adalah simbol-simbol yang ada pada diagram Composite Structure :

Simbol	Deskripsi
Property  	Property adalah satu set dari suatu instance. RoleName : peran / nama / identitas dari property (opsional) TypeName : type kelas dari property (harus ada)
Connector  Nama connector : tipe connector	Connector adalah cara komunikasi data dari 2 buah instance ConnName : nama connector (opsional) connType : tipe connector (opsional)
Port  Pemakainnya adalah sebagai berikut :	Port adalah cara yang digunakan dalam diagram composite structure tanapa menampilkan detail internal dari suatu sistem . Port digambarkan dalam bentuk kotak kecil yang menempel atau didalama suatu property. Port digambarkan menempel property jika fungsi tersebut dapat diakses public. Sedangakan port digambarkan di dalam suatu property jika fungsi tersebut bersifat protected.
Class  	Kelas; jika yang akan dijabarkan strukturnya adalah sebuah kelas.

Contoh diagram composite adalah sebagai berikut :

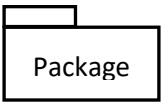


Gambar diagram composite

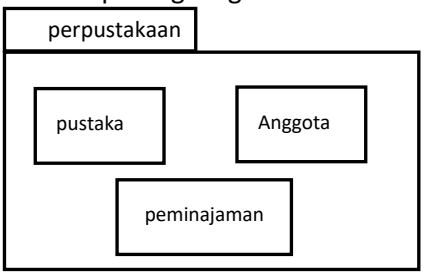
Maksud dari diagram di atas adalah bahwa atribut id, judul, penerbit, dan tahun diakses nilainya metode query Memasuakn pustaka , atribut id di akses pada metode setId dengan memberikan nilai kepada atribut id . Kelas Pustaka diakses oleh kelas Mengelola Pustaka

Package Diagram

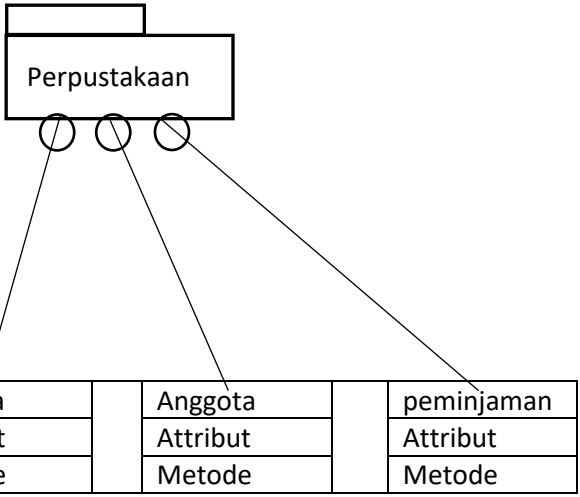
Package Diagram menyediakan cara mengumpilkan eleme-elemen yang saling terkait dalam diagram UML . Hampir semua diagram dalam UML dapat dikelompokan menggunakan packagediagram. Berikut ini simbol-simbol yang digunakan dalam :

Simbol	Deskripsi
Package 	Package merupakan sebuah bungkusan dari satu atau lebih kelas atau elemen diagram UML lainya

Elemen dalam package digambarkan di dalam package



Elemen dalam *package* digambarkan di luar *package*



Deployment Diagram

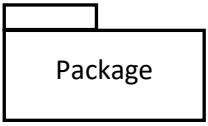
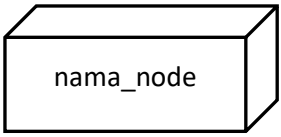
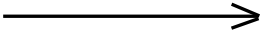
Diagram deployment atau deployment diagram menunjukan konfigurasi komponen dalam proses eksekusi aplikasi . Diagram deyployment juga dapat digunakan untuk memodelkan hal-hal berikut :

Sistem client / server misalanya seperti gambar berikut :

Diagram Deployment sistem client / server

Sistem terdistribusi murni

Rekayasa ulang aplikasi

Simbol	**Deskripsi**
Package 	Package merupakan sebuah bungkusan dari satu atau lebih node
Noode 	Biasanya mengacu pada perangkat keras (hardware) perangkat lunak yang tidak di buat sendiri (software) jika dalam node disertakan komponen untuk mengkonsistenkan rancangan maka komponen yang di ikutsertakan harus sesuai dengan komponen yang telah didefinisikan sebelumnya pada diagram komponen
Kebergantungan / dependency 	Kebergantungan antar node arah panah mengarah pada node yang dipakai
Link	Relasi antar node

Use Case Diagram

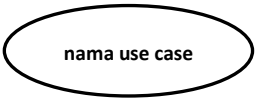
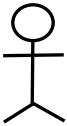
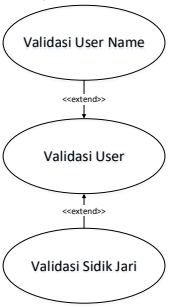
Use case atau diagram use case merupakan pemodelan untuk kelakuan (behavior) sistem informasi yang akan dibuat. Use case mendeskripsi sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, use case digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan funsgsi-fungsi itu.

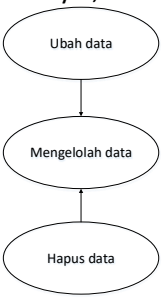
Syarat penamaan pada use case adalah nama didefinisikan sesimpel mungkin dan dapat dipahami . Ada dua hal utama pada use case yaitu pendefinisian apa yang disebut aktor dan use case.

Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang .

Use case merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor

Berikut adalah simbol-simbol yang ada pada diagram use case:

Simbol	Deskripsi
Use case 	Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor , biasanya dinyatakan dengan menggunakan kata kerja di awal di awal frase nama use case
Aktor / actor 	Orang, proses, atau sistem lain yang beriteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri , jadi walaupun simbol aktor adalah gambar orang , tapi aktor belum tentu merupakan orang , biasanya dinyatakan menggunakan kata benda di awal frase nama aktor
Asosiasi / association	Komunikasi antara aktor dan use case yang berpartisipasi pada use case atau use case memiliki interaksi dengan aktor
Ekstensi / extend 	Relasi use case tambahan ke sebuah use case dimana use case yang ditambahkan dapat berdiri sendiri walau tanpa use case tambahan itu, mirip dengan prinsip *inheritance* pada pemograman berorientasi objek , biasanya use case tambahan memiliki nama depan yang sama dengan use case yang ditambahkan , misal 

	Arah panah mengarah pada use case yang di tambahkan , biasanya use case yang menjadi extend-nya merupakan jenis yang sama dengan use case yang menjadi induknya
Generalisasi / generalization	Hubungan generalisasi dan spesialisasi (umum-khususu) antara dua buah use case dimana fungsi yang satu adalah fungsi yang lebih umum dari lainya , misalnya  Arah pana mengarah pada use case yang menjadi generalisasinya (umum)
Menggunakan / include / uses	Realisasi use case tambahan ke sebuah use case di mana use case yang ditambahkan memerlukan use case ini untuk menjalankan fungsinya atau sebagai syarat dijalnkan use case ini Ada dua sudut pandang yang cukup besar mengenai include di use case : Include berarti use case yang ditambahakan akan selalu dipanggil saat use case tambahan dijalanakan, misal pada kasus berikut : Include berarti use case yang tambahan akan selalu melakukan pengecekan apakah use case use case yang ditambakan telah dijalankan sebelum use case tambahan dijalankan , misal pada kasus berikut : Kedua interpretasi di atas tepat dianut salah

	satu atau keduanya tergantung pada pertimbangan dan interprestasi yang dibutuhkan .

Use case nantinya akan menjadi kelas prose pada digram kelas sehingga perlu di pertimbangkan penamaan yang dilakukan apakah sudah layak menjadi kelas atau belum sesuai dengan aturan pendefinisian kelas yang baik . Berikut adalah aturan perubahan use case yang layak menjadi kelas proses sehingga layak sebagai di jadikan use case :

Hubungan	Keterangan
Exstensi / extend	Pada hubungan ekstensi maka dapat hanya diambil use case induknya yang dijadikan kelas dengan metode berupa use case ekstensinya
Generalisasi / generalization	Pada hubungan generalisasi maka dapat hanya diambil use case umumnya yang dijadikan kelas dengan metode berupa use case khususnya
Use case yang berdiri sendiri	Metode yang munkin bisa ada di dalam kelas proses login adalah sebagai berikut :
Use case yang kurang tepat sebagai sebuah use case yang berdiri sendiri	Kurang tepat karena kelasnya akan menjadi : Kelas yang hanya terdiri dari satu metode sebenarnya kurang efisien

Setiap use case dilengkapi dengan skenario . Skenario use case adalah alur jalannya proses use case dari sisi aktor dan sistem . Berikut adalah format tabel skenario use case :

Aksi aktor	Relasi Sistem
Skenario Normal	
Skenario Alternatif	

Skenariouse case dibuat per use case terkecil, misalkan untuk generalisasi maka skenario yang dibuat adalah use case yang lebih khusus. Skenario normal adalah skenario bila sistem berjalan normal tanpa terjadi kesalahan atau error . Sedangkan skenario alternatif adalah skenario bila sistem tidak berjalan normal , atau mengalami error . Skenario normal dan skenario alternatif dapat lebih dari satu . Alur dari skenario yang nantinya menjadi dasar pembuatan diagram

sekuen.

Activity Diagram

Diagram aktivitas atau activity diagram menggabarkan workflow (aliran kerja) atau aktifitas dari sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak . Yang perlu diperhatikan disini adalah bahwa diagram aktivitas menggabarkan sistem bukan apa yang dilakuakn aktor , jadi aktivitas yang dapat dilakukan oleh sistem .

Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut :

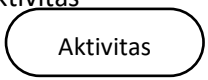
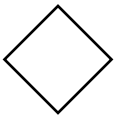
Rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis yang didefinisikan

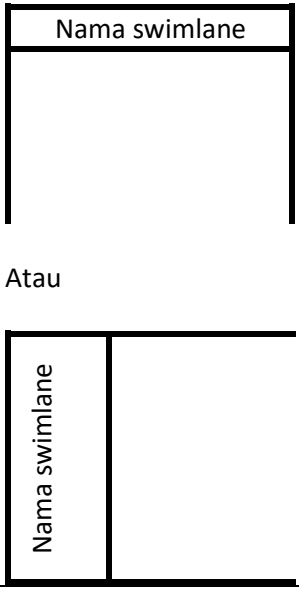
Urutan atau pengelompokan tampilan dari sistem / user interface dimana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan

Rancangan pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya

Rancangan menu yang ditampilkan pada perangkat lunak

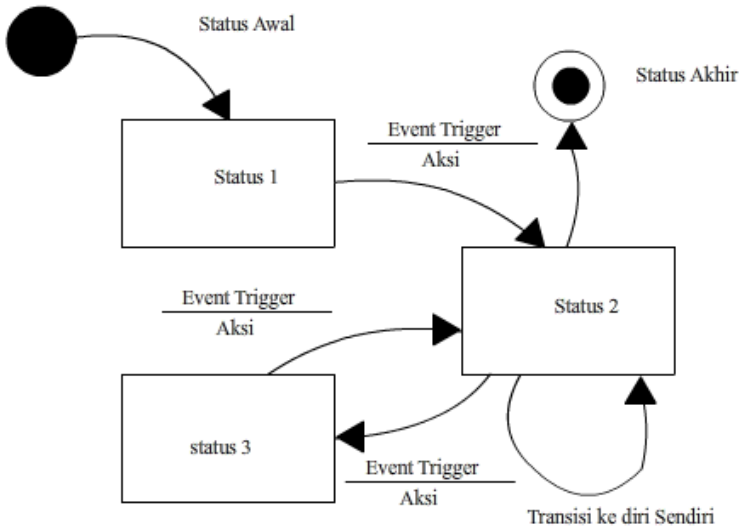
Berikut adalah simbol-simbol yang ada pada diagram aktivitas :

Simbol	Deskripsi
Status awal 	Status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah status awal
Aktivitas 	Aktivitas yang dilakuakan sistem, aktivitas biasanya diawali dengan kata kerja
Percabangan / decision 	Asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu
Penggabungan / join	Asosiasi penggabungan dimana lebih dari satu aktivitas digabunkan menjadi satu
Status akhir 	Status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki sebuah status akhir

Swimlane  	Memisahkan organisasi bisnis yang bertanggung jawab terhadap aktivitas yang terjadi

State Machine Diagram

State machine diagram atau statechart diagram atau dalam bahasa Indonesia disebut diagram mesin status atau sering juga disebut diagram status digunakan untuk menggabarkan perubahan status atau transisi status dari sebuah mesin atau sistem atau objek. Jika diagram sekuen digunakan untuk interaksi antar objek maka diagram status digunakan untuk interaksi di dalam sebuah objek. Perubahan tersebut digambarkan dalam suatu graf berarah. State machine diagram merupakan pengembangan dari diagram Finite State Automata dengan penambahan beberapa fitur dan konsep baru . Diagram Finite State Automata (FSA) ini biasanya diajarkan dalam mata kuliah Automata. State machine diagram cocok digunakan untuk menggabarkan alur interaksi pengguna dengan sistem. Berikut ini adalah contoh gambar diagram mesin status.

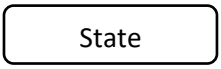


Contoh

state machine diagram

Berikut ini komponen-komponen dasar yang ada dalam state machine diagram :

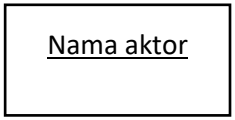
Simbol	Deskripsi
Start / Status Awal (Intial State) 	Start atau initial state adalah state atau keadaan awal pada saat sistem mulai hidup
End / Status Akhir (Final State) 	End atau final state adalah state keadaan akhir dari daur hidup suatu sistem
Event 	Event adalah kegiatan yang menyebabkan berubahannya status mesin.

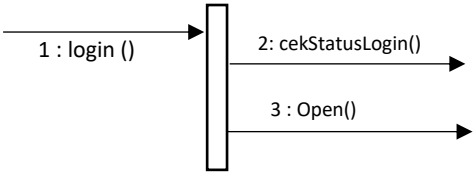
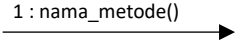
State 	State atau status adalah keadaan sistem pada waktu tertentu . State dapat berubahan jika ada event tertentu yang memicu perubahan tersebut

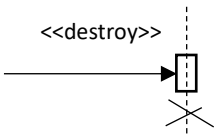
Sequence Diagram

Diagram sekuen menggabarkan kelakuan objek pada use case dengan mendeskripsikan waktu hidup objek dan massage yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram sekuen maka harus diketahui objek - objek yang terlibat dalam sebuah use case beserta metode - metode yang dimiliki kelas yang diinstansiasi menjadi objek itu. Membuat diagram sekuen juga dibutuhkan untuk melihat skenario yang ada pada use case.

Banyaknya diagram sekuen yang harus digambarkan adalah minimal sebanyak pendefinisian use case yang memiliki proses sendiri atau yang penting semua use case yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram sekuen sehingga semakin banyak use case yang didefinisikan maka diagram sekuen yang harus dibuat juga semakin banyak.
Berikut adalah simbol-simbol yang ada pada diagram sekuen :

Simbol	**Deskripsi**
Aktor  Atau  Tanpa waktu Aktif	Orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat diluar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal frase nama aktor.
Garis Hidup / *lifeline*	Menyatakan kehidupan suatu objek
Objek	Menyatakan objek yang berinteraksi pesan

Nama objek : nama kelas	
Waktu Aktif 	Menyatakan objek dalam keadaan aktif dan berinteraksi, semua yang terhubung dengan waktu aktif ini adalah sebuah tahapan yang dilakukan di dalamnya, misalnya  Maka cekStatusLogin() dan open() dilakukan didalam metode login() Aktor tidak memiliki waktu aktif.
Pesan Tipe Create	Menyatakan suatu objek membuat objek yang lain, arah panah mengarah pada objek yang dibuat.
Pesan Tipe Call 	Menyatakan suatu objek memanggil operasi atau metode yang ada pada objek lain atau dirinya sendiri, Arah panah mengarah pada objek yang memiliki operasi atau metode, karen ini memanggil operasi atau metode maka operasi atau metode yang dipanggil harus ada pada diagram kelas sesuai dengan kelas objek yang berinteraksi
Pesan Tipe Send	Menyatakan bahwa suatu objek mengirimkan data/masukan/informasi ke objek lainnya, arah panah mengarah pada objek yang dikirim
Pesan Tipe Return	Menyatakan bahwa suatu objek yang telah menjalankan suatu operasi atau metode mengghasilkan suatu kembalian ke objek tertentu, arah panah mengarah pada objek yang menerima kembalian
Pesan Tipe Destroy	Menyatakan suatu objek mengakhiri hidup objek yang

	lain, arah panah mengarah pada objek yang diakhiri, sebaiknya jika ada create maka ada destroy.

Penomoran pesan berdasarkan urutan interaksi pesan, penggamnbaran letak pesan harus berurutan, pesan yang lebih atas dari lainnya adalah pesan yang berjalan lebih dahulu.

Semua metode ini di dalam kelas harus ada di dalam diagram kolaborasi atau sekuen jika tidak ada berarti perancangan metode di dalam kelas itu kurang baik. Hal ini dikarenakan ada metode yang tidak dapat dipertanggungjawabkan kegunaannya.

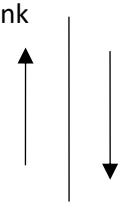
Communication Diagram

Communication Diagram atau diagram komunikasi pada UML versi 2.x adalah penyerderhanaa dari diagram kolaborasi (*collaborattion diagram)* pada UML Versi 1.x *Collaborattion Diagram* sudah tidak muncul lagi pada UML Versi 2.x. Diagram komunikasi sebenarnya adalah diagram kolaborasi tetapi dibuat untuk tiap sekuen.

Diagram komunikasi menggabarkan interaksi anatar objek / bagian dalam bentuk urutan pengiriman pesan . Diagram komunikasi mempersentasikan informasi yang diperoleh dari diagram kelas, Diagram sekuen dan diagram use case untuk mendeskripsikan gabunagn anatara struktur statis dan tingkah laku dinamis dari suatu sistem

Diagram komunikasi melompokan message pada kumpulan diagram sekuen menjadi sebuah diagram. Dalam diagram komunikasi yang dituliskan adalah operasi / metode yang dijalankan antara objek yang satu dan objek linya secara keseluruhan, oleh karena itu dapat diambil dari jalannya interaksi pada semua diagram sekuen. Penomoran metode dapat dilakukan berdasarkan urutan dijalanakannya metode / operasi diantaranya objek yang satu dengan objek lainya atau objek itu sendiri.

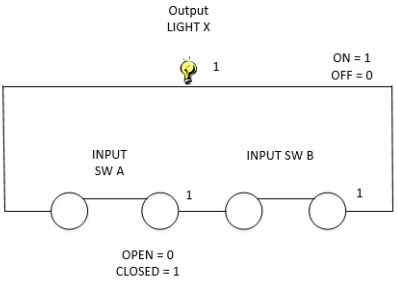
Berikut adalah simbol-simbol yang ada pada diagram kolaborasi :

Simbol	Deskripsi
Objek Nama_objek : nama_kelas	Objek yang melakukan interaksi pesan
Link	Relasi antar objek yang menghubungkan objek satu dengan lainya atau dengan dirinya sendiri
Arah pesan / stimulus	Arah pesan yang terjadi , jika pada suatu link ada dua arah pesan yang berbeda maka arah juga di gambarkan dua arah pada dua sisi link 

Timing Diagram

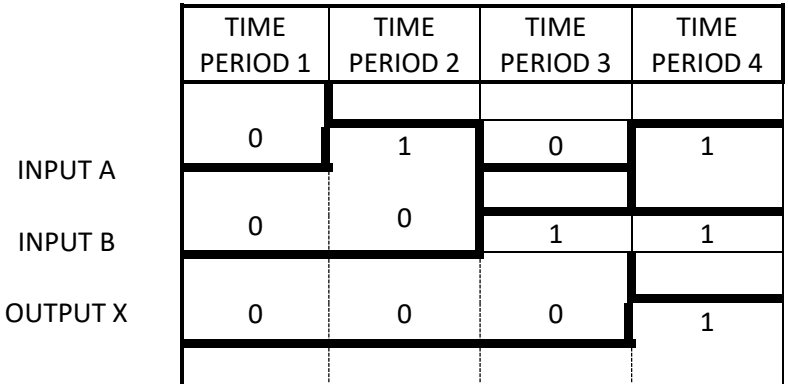
Timing diagram merupakan diagram yang fokus pada penggambaran terkait batasan waktu. Timing diagram digunakan untuk menggabarkan tingkah laku sistem dalam periode waktu tertentu. Timing diagram biasanya digunakan untuk mendeskripsikan operasi dari alat digital karena penggambaran secara visual akan lebih mudah dipahami dari pada dengan kata-kata

Berikut ini adalah contoh aliran sirkuit yang menggabarkan operasi timing diagram . Gambar di bawah menyatakan aliran listrik. Status OPEN = 0 artinya switch dalam posisi terbuka (tidak terhubung), sedangakan status CLOSED = 1 adalah posisi switch terhubung. Lampu akan menyala , status ON = 1 , jika switch terhubung.



Gambar contoh aliran sirkuit

Berikut ini contoh gambar pemanfaatan timingdiagram dalam menggabarkan seluruh status dalam gambar aliran sirkuit di atas.



Gambar contoh timing diagram

Aliran waktu pada timing diagram dibaca dari kiri ke kanan. Pada gambar timing diagram di atas dapat terlihat bahwa lampu akan menyala jika kedua switch atau INPUT A dan INPUT B dalam posisi CLosed = 1. Jika salah satu switch tidak terhubung maka lampu tidak akan menyala.

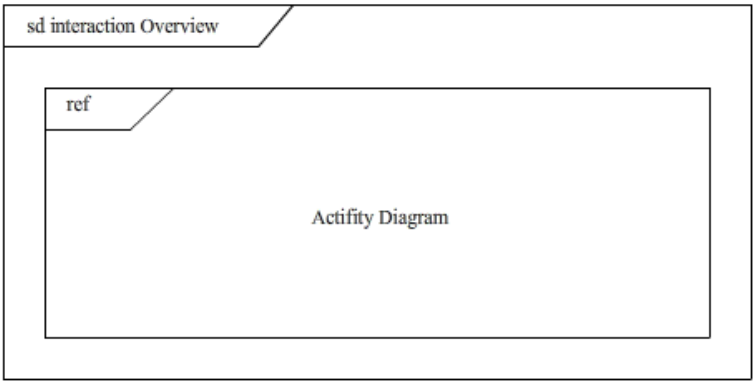
Interaction Overview Diagram

Interaction overview diagram mirip dengan diagram aktivitas yang berfungsi untuk menggambarkan sekumpulan urutan aktivitas. Interaction Overview Diagram adalah bentuk aktivitas diagram yang setiap titik mempresentasikan diagram interaksi. Interaksi diagram dapat meliputi diagram sekuen, diagram komunikasi, interaction overview diagram, dan timing diagram.

Hampir semua notasi pada interaction overview diagram sama dengan notasi pada diagram aktivitas. Sebagai contoh initial, final, decision, merge, fork, dan join nodes sama seperti pada diagram interaction occurence dan interaction element.

Interaction Occurrence

Interaction Occurrence atau kejadian interaksi adalah refensi untuk diagram interaksi yang ada. Sebuah interaction occurrence ditunjukan sebagai frame refensi (frame dengan tulisan refdi pojok kiri atas). Nama diagram yang sedang direfensikan ditunjukan pada tengah frame. Berikut adalah gambar contoh dari interaction occurence :

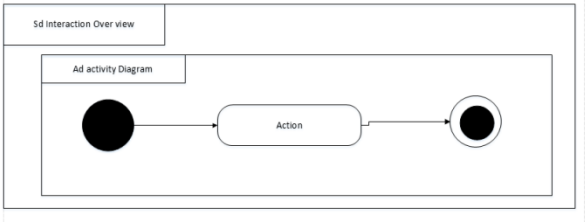


Contoh interaction occurence

sd di atas kependekan dari sequence diagram.

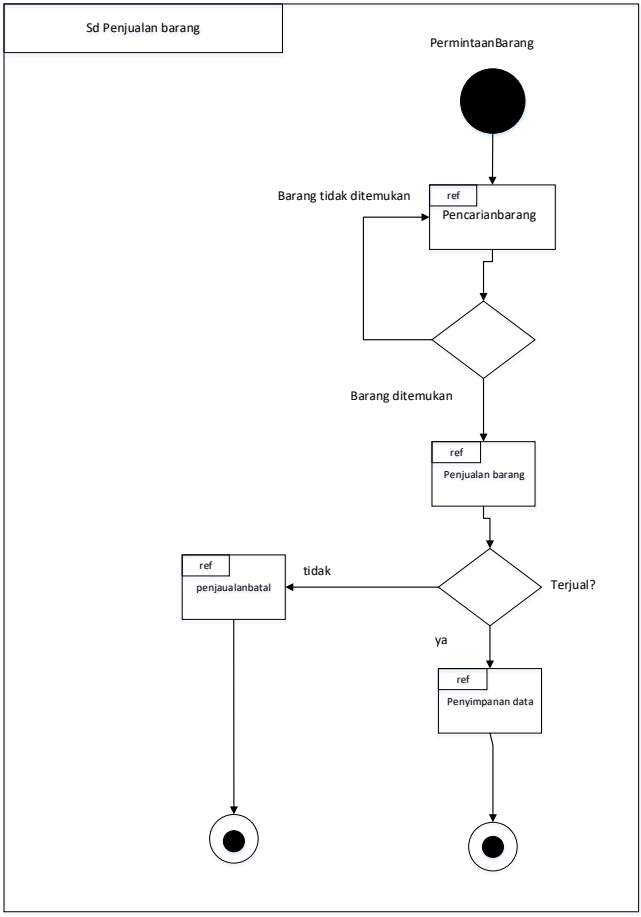
Interaction Element

Interaction element atau elemen interaksi mirip interaction occurrence. Perbedaanya adalah di dalam interaction element menampilkan isi diagram yang direfensikan secara langsung sedangkan interaction occurrence hanya menampilkan nama diagram yang direfensikan. Berikut adalah gambar contoh dari interction element:



Contoh Intersction Element

Berikut ini adalah contoh interaction overview diagram yang menampilkan kontrol-kontrol pada diagram aktifitas (fork, join, merge dan lain-lain) dengan abstraksi subproses digambarkan menggunakan interaction occurrence



Studi Kasus UML

Berikut ini adalah fungsi-fungsi yang dibutuhkan oleh perangkat lunak sistem:

Customer Function

1. Menangani proses penampilan informasi penerbangan yang terdiri dari info fare, info Schedule, info available, dan Special Offer berdasarkan inputan yang diinginkan oleh customer.
2. Menangani proses pendaftaran customer untuk melakukan reservasi tiket pesawat secara online.
3. Menangani proses login customer untuk masuk ke halaman reservasi tiket Pesawat.
4. Menangani proses reservasi tiket Pesawat
5. Menangani proses Pembayaran tiket Pesawat melalaui bank yang bersangkutan.
6. Menangani update data reservasi tiket Pesawat yang dilakukan oleh customer.
7. Menangani proses pembatalan tiket pesawat .

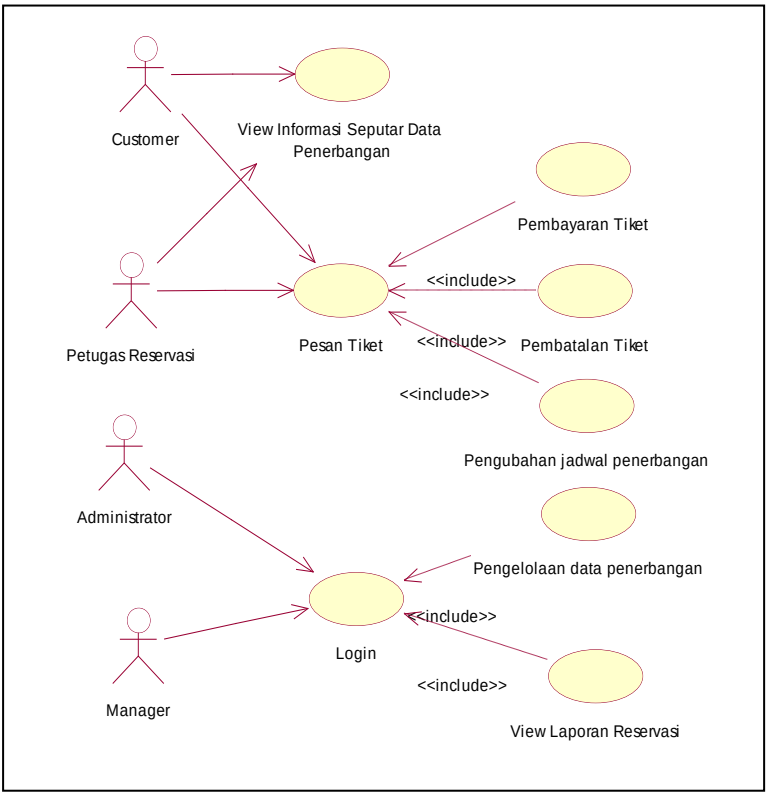
Kebutuhan fungsionalitas yang ada pada admin:

1. Menangani proses insert,update,delete halaman about, faq, tips n trip, office, kebijakan, contact us dan news.
2. Menangani data insert,update,delete yang dibutuhkan untuk data reservasi.
3. Menangani pemesanan tiket secara offline.
4. Menangani pemrosesan pemesanan tiket yang dilakukan oleh customer secara online.
5. Menangani pemrosesan data reservasi.
6. Menangani proses update profile dari admin.
7. Menangani pemrosesan ganti password oleh admin.
8. Menangani pengelompokan admin grop.

Kebuthan fungsionalitas yang ada pada manager:

1. Menangani pemrosesan tampilan laporan sesuai dengan inputan customer.

1.Diagram Use Case



Gambar Error! No text of specified style in document.**-1 Use Case Diagram**

Keterangan Use Case:

No.	Nama Use Case		Fungsi Use Case
1.	View Informasi Seputar Data Penerbangan	:	Memberikan informasi tentang jadwal penerbangan, ketersediaan kursi, harga tiket, dan kebijakan – kebijakan PESAWAT
2.	Pesan Tiket	:	Proses pemesana tiket
3.	Pembayaran Tiket	:	Proses pembayaran tiket jika tiket telah dipesan sebelumnya
4.	Pembatalan Tiket	:	Proses pembatalan tiket jika tiket telah dipesan sebelumnya
5.	Pengubahan Jadwal	:	Proses pengubahan jadwal penerbangan jika

	Penerbangan		tiket dipesan sebelumnya
6	Login	:	Autintifikasi User
7	Pengelolaan Data Penerbangan	:	Mengelola data penerbangan yang meliputi : • Input data penerbangan baru • Update data penerbangan • Delete data penerbangan
8	View Laporan Reservasi	:	Melihat laporandata hasil reservasi tiket

2. Flow of Events

No.	Nama Flow	Basic Flow	Alternate Flow
1.	Flow of Event View Informasi Seputar Data Penerbangan	• Customer memilih jenis informasi apa yang ingin dilihat	• Jika data tidak ada atau kosong maka sistem akan menampilkan pesan bahwa data tersebut belum ada
2.	Flow of Event Pesan Tiket	• Customer memilih jadwal penerbangan yang diinginkan • Customer memilih menu layanan pesan tiket • Customer menerima kode booking	• Jika jadwal penerbangan tidak ada maka sistem tidak akan menampilkan menu layanan pesan tiket
3.	Flow of Event Pembayaran Tiket	• Customer menginputkan kode booking yang dimilikinya • Sistem menampilkan data booking untuk divalidasi oleh customer • Customer memilih menu layanan bayar tiket • Customer menginputkan data kartu kreditnya	• Sistem akan memvalidasi kebenaran kode booking yang diinputkan customer • Sistem akan mencek data kartu kredit yang diinputkan customer sebelum melakukan proses transaksi

		• Sistem menampilakn data kartu kredit customer • Customer memilih menu ok untuk proses pembayaran tiket	
4.	Flow of Event Pembatalan Tiket	• Customer menginputkan kode booking yang dimilikinya • Sistem menampilkan data booking untuk divalidasi oleh customer • Customer memilih menu layanan batal tiket	• Sistem akan memvalidasi kebenaran kode booking yang diinputkan customer
5.	Flow of Event Pengubahan Jadwal Penerbangan	• Customer menginputkan kode booking yang dimilikinya • Sistem menampilkan data booking untuk divalidasi oleh customer • Customer memilih menu layanan pengubahan jadwal penerbangan tiket • Customer memilih jadwal penerbangan baru • Sistem melakukan perhitungan selisih harga tiket • Sistem menyimpan jadwal penerbangan baru	• Sistem akan memvalidasi kebenaran kode booking yang diinputkan customer • Jika jadwal penerbangan baru tidak ada maka sistem tidka akan memproses pengubahan jadwal baru dan kemudian menampilkan pesan jadwal penerbangan belum bisa diganti ke layar web
6.	Flow of Event	• User menginputkan	• Jika username dan

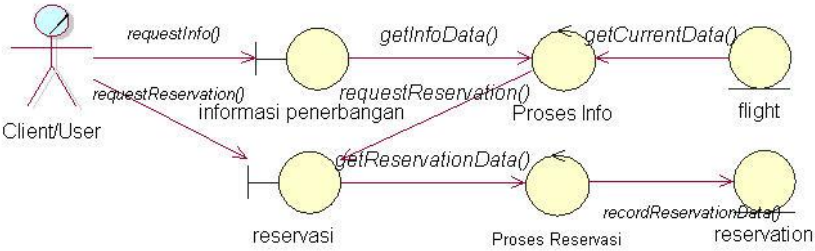
	Login	usernam dan password	password tidak valid maka sistem akan memapilkan pesan error ke layar
7.	Flow of Event Pengelolaan Data Penerbangan	• User memilih menu Pengelolaan Data Penerbangan yang terdiri dari input data baru, update data penerbangan, dan delete data penerbangan • User meginputkan data penerbangan baru jika memilih menu layanan input penerbangan baru • User memilih data penerbangan yang akan diupdate atau didelete jika memilih menu layanan update atau delete data penerbangan	• Sistem akan melakukan validasi format data inputan pada saat user melakukan proses input atau update data penerbangan • Sebelum menghapus data penerbangan, sistem akan melakukan verifikasi ke user apakah data penerbangan tersebut dihapus atau tidak
8.	Flow of Event View Laporan Reservasi	• User memilih menu view laporan reservasi	• Sistem akan melakukan verifikasi pakah yang login tersebut manager atau tidak

3.Diagram Interaksi

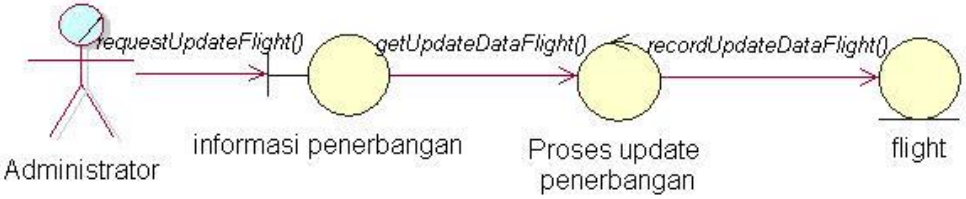
Diagram Interaksi terdiri atas Diagram Kolaborasi dan Sequence diagram

3.1Diagram Kolaborasi

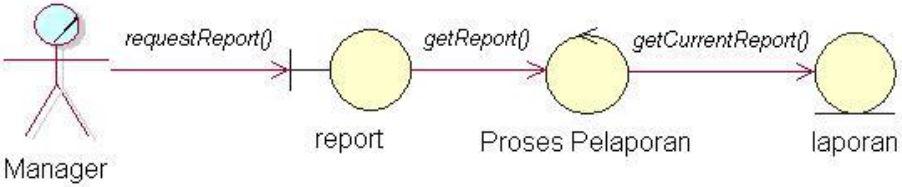
- Pada sisi Client atau Petugas reservasi



- Pada sisi Administrator

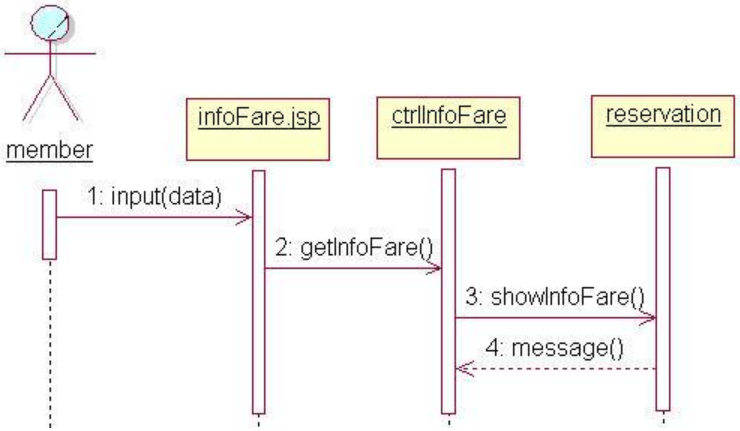


- Pada sisi Manager

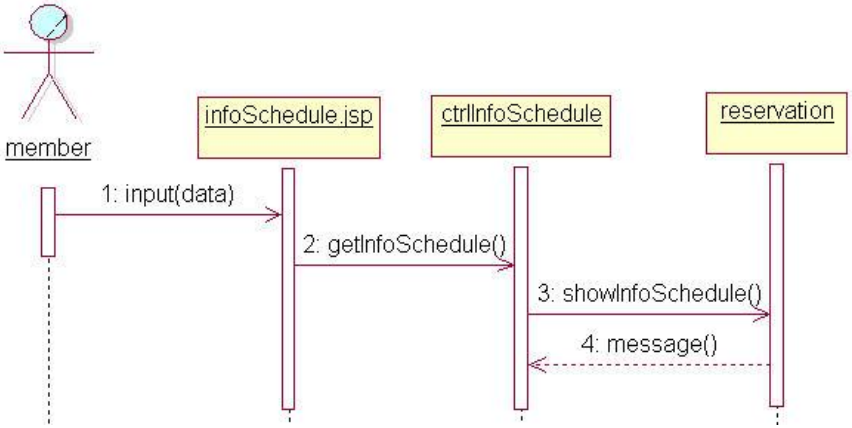


3.2 Sequence Diagram

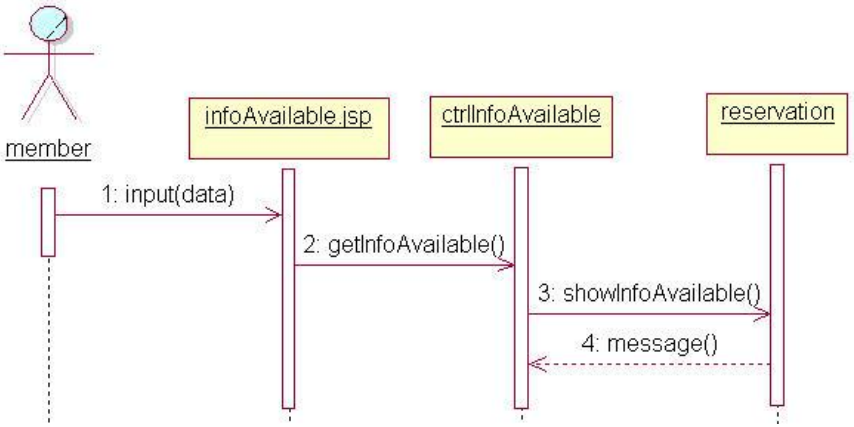
Info Fare



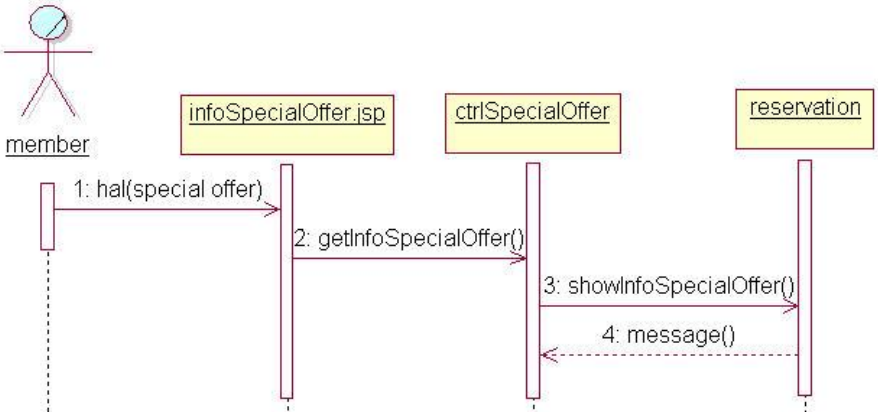
Info Schedule



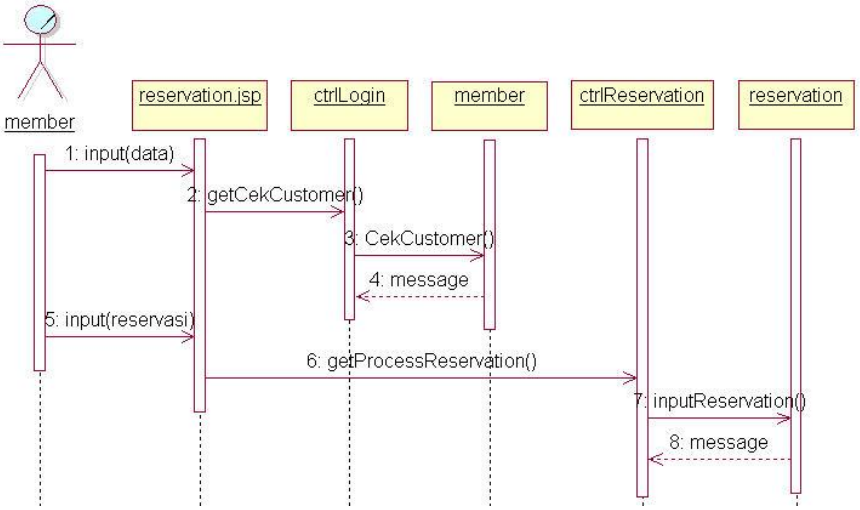
Info Available



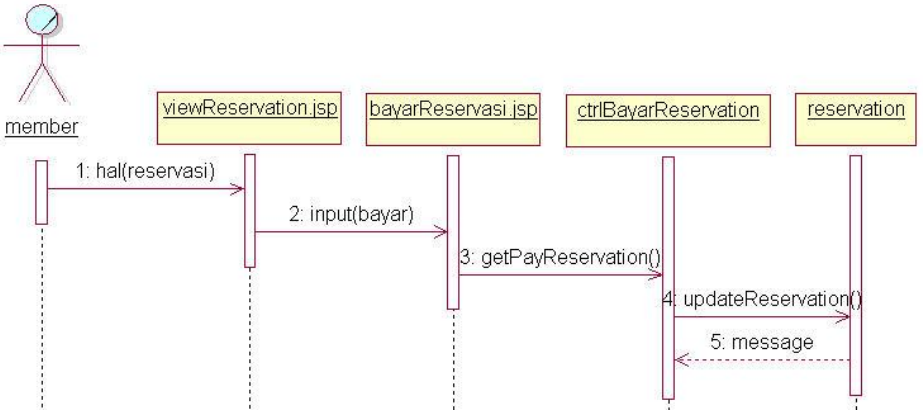
Info Special Offer



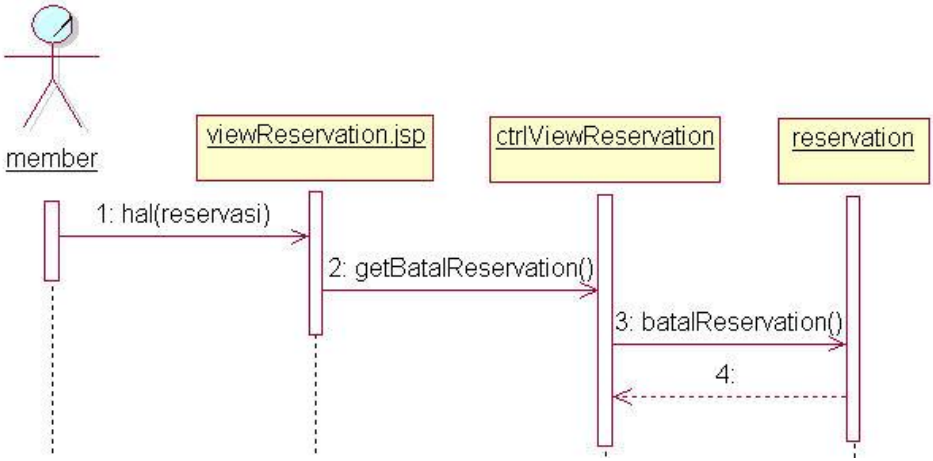
Pesan tiket



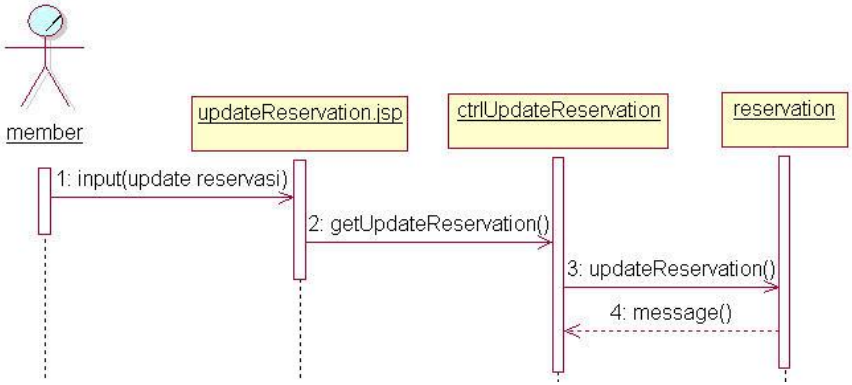
Bayar Tiket



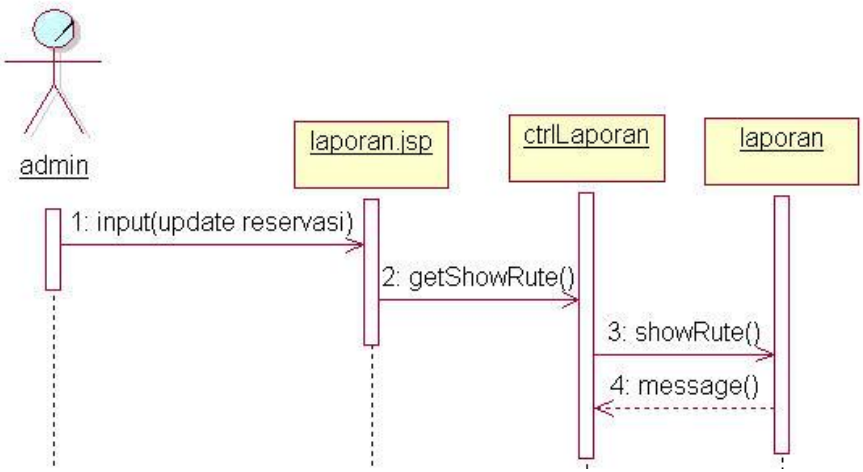
Batal Tiket



Ubah Jadwal Penerbangan

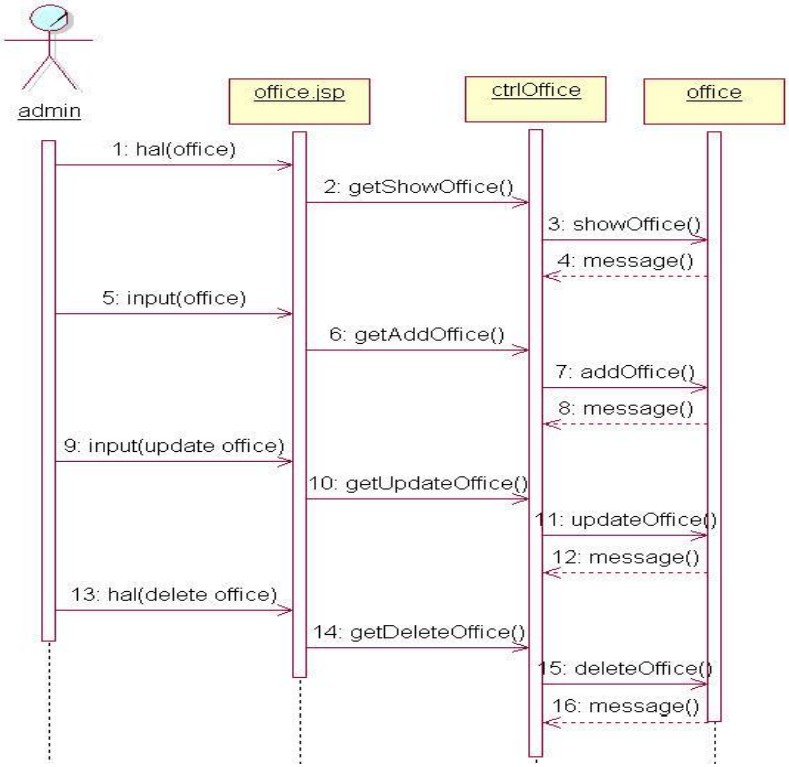


View Laporan Reservasi

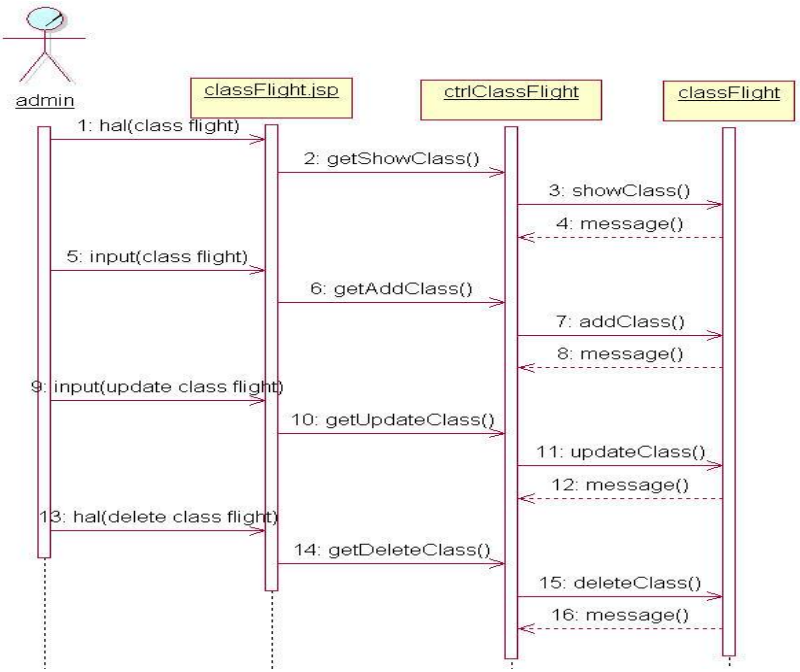


Kelola Data Terbang

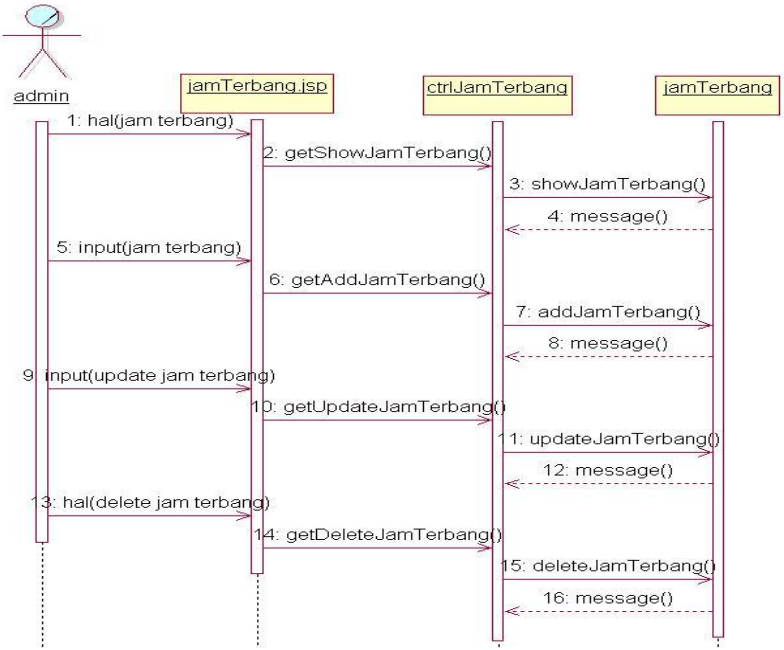
Office



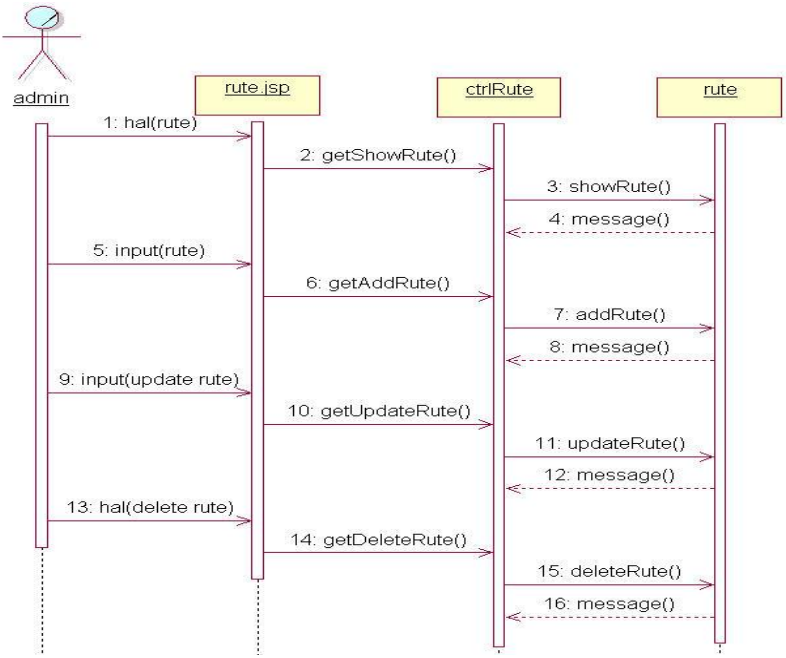
Kelas Tiket


admin
classFlight.jsp
ctrlClassFlight
classFlight
1: hal(class flight)
2: getShowClass()
3: showClass()
4: message()
5: input(class flight)
6: getAddClass()
7: addClass()
8: message()
9: input(update class flight)
10: getUpdateClass()
11: updateClass()
12: message()
13: hal(delete class flight)
14: getDeleteClass()
15: deleteClass()
16: message()

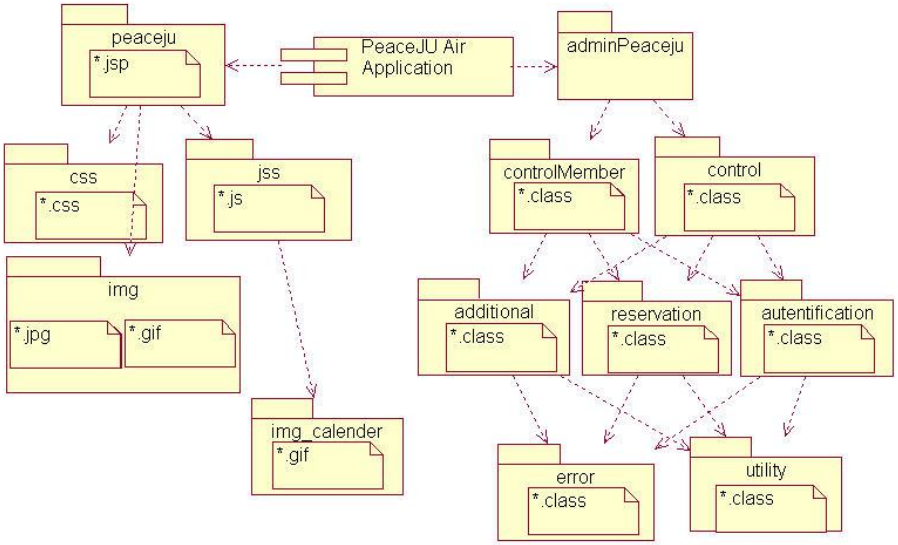
Jam Terbang


admin
jamTerbang.jsp
ctrlJamTerbang
jamTerbang
1: hal(jam terbang)
2: getShowJamTerbang()
3: showJamTerbang()
4: message()
5: input(jam terbang)
6: getAddJamTerbang()
7: addJamTerbang()
8: message()
9: input(update jam terbang)
10: getUpdateJamTerbang()
11: updateJamTerbang()
12: message()
13: hal(delete jam terbang)
14: getDeleteJamTerbang()
15: deleteJamTerbang()
16: message()

Rute

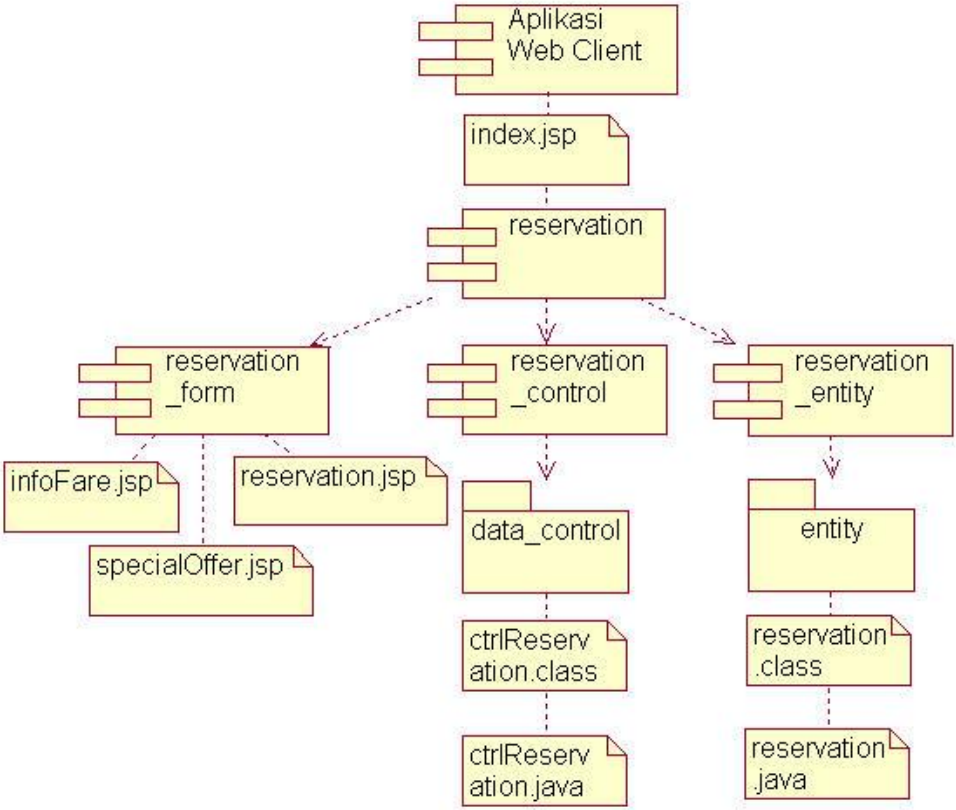


Desain arsitektur

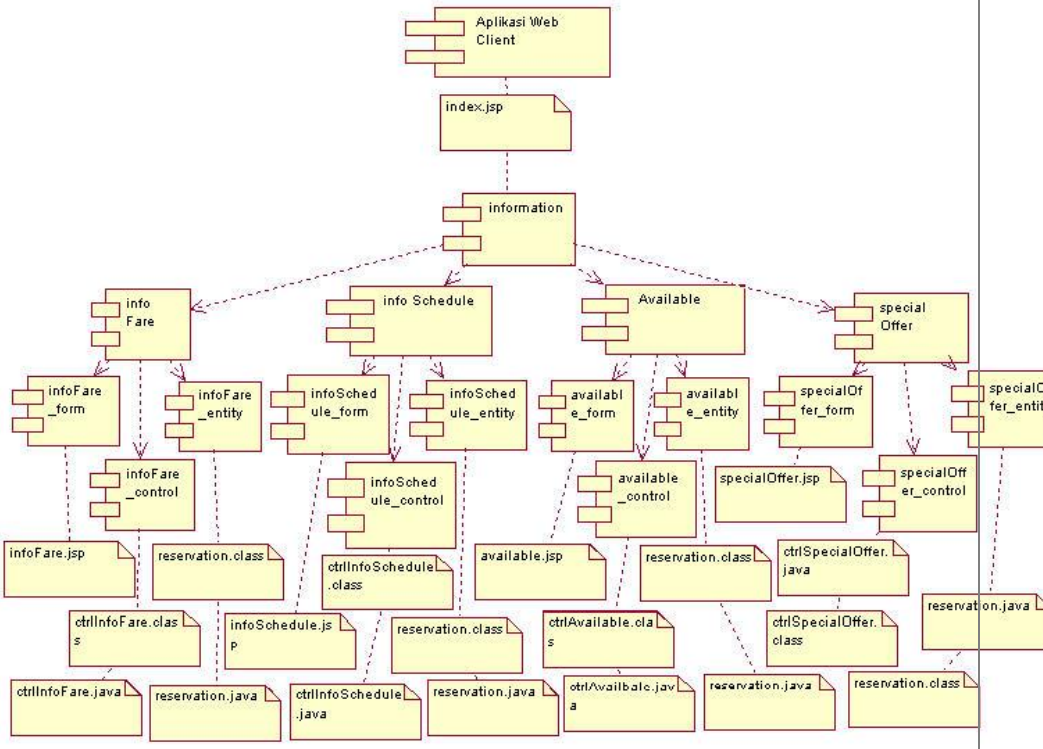


Gambar arsitektur fisik PL

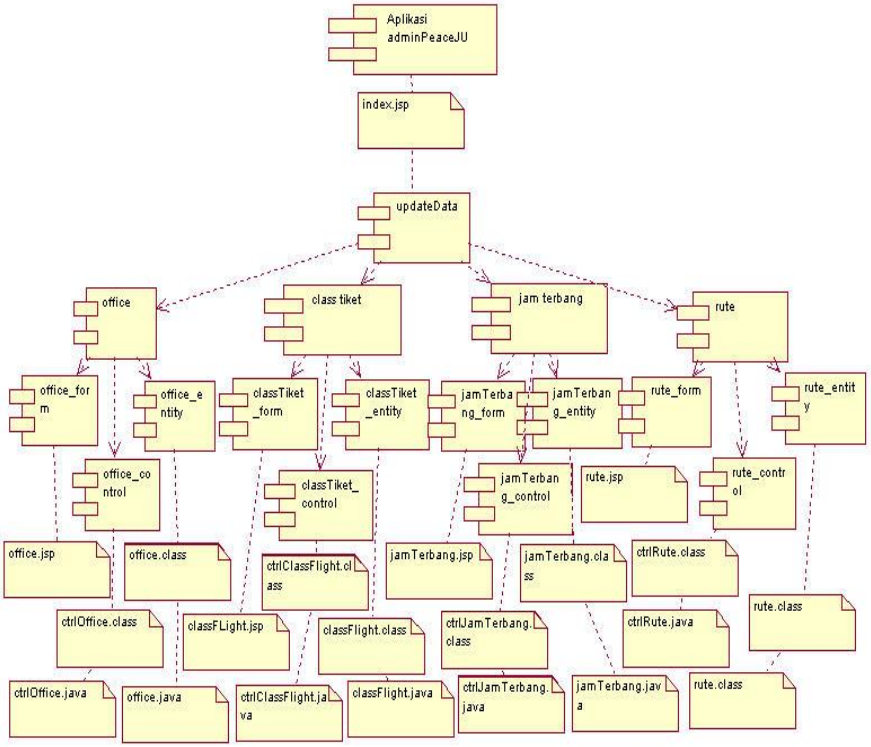
4.Aplikasi web untuk client (proses reservasi)



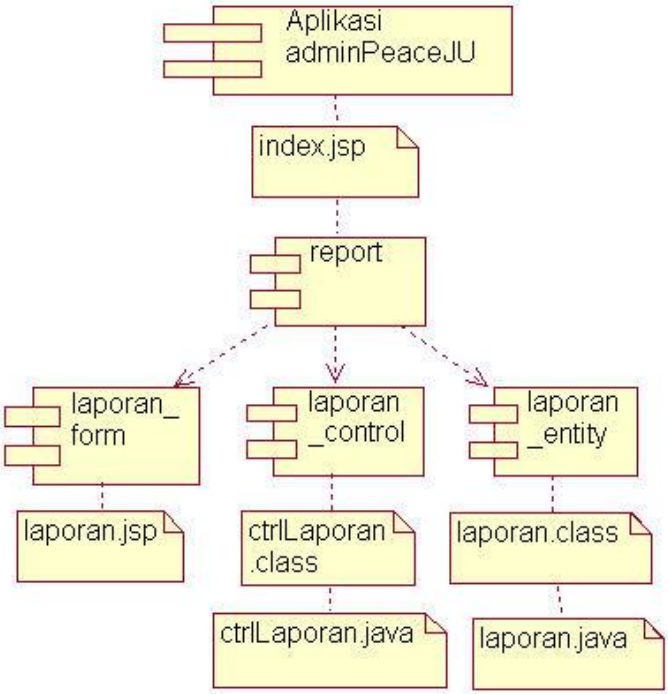
5. Aplikasi web untuk client (proses melihat informasi cont.)



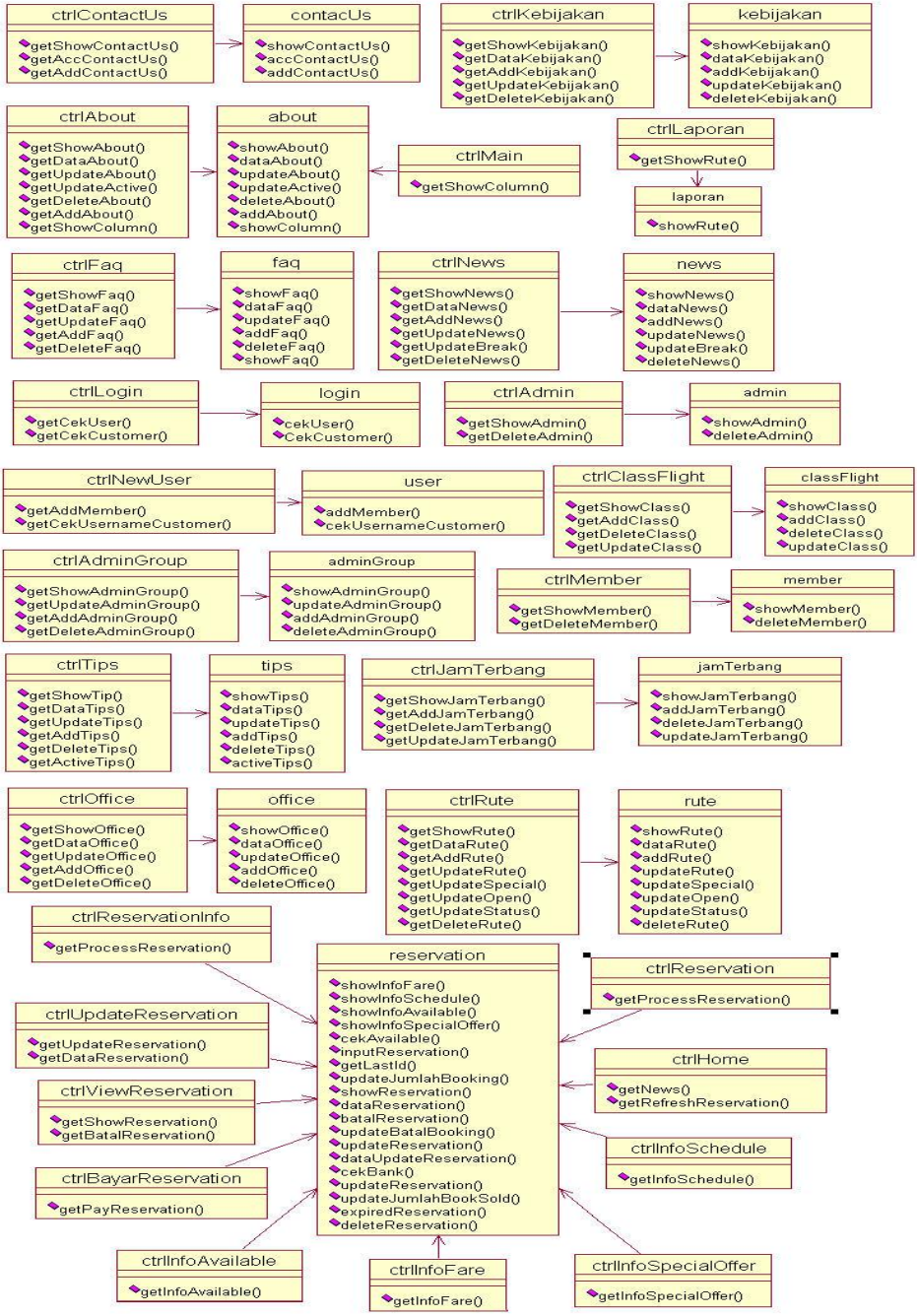
- 6. Aplikasi web untuk admin proses update data



7.Proses laporan



8.Class Diagram



BAB V. METODE DESAIN

5.1 Konsep dan Prinsip Desain

- Desain perangkat lunak duduk di inti teknis rekayasa perangkat lunak dan diterapkan terlepas dari model proses perangkat lunak yang digunakan.
- Tugas desain menghasilkan desain data, desain arsitektur, desain antarmuka, dan komponen desain.

Desain Data

- Desain data mengubah model domain informasi yang dibuat selama analisis ke dalam struktur data yang akan diperlukan untuk mengimplementasikan perangkat lunak.
- Objek data dan Relasi data didefinisikan dalam diagram hubungan entitas.
- Bagian dari desain data dapat terjadi dalam kombinasi dengan desain arsitektur perangkat lunak.

Desain arsitektur

- Desain arsitektur mendefinisikan hubungan antara elemen struktural utama dari perangkat lunak.
- Representasi desain arsitektur — kerangka kerja sistem berbasis komputer — dapat diturunkan dari spesifikasi sistem, model analisis, dan interaksi subsistem yang didefinisikan dalam model analisis.

Desain antarmuka

- Desain antarmuka menggambarkan bagaimana perangkat lunak berkomunikasi dalam dirinya, dengan sistem itu berinteraksi dengan user yang menggunakannya
- Antarmuka menyiratkan aliran informasi (mis., Data dan / atau kontrol) dan jenis perilaku tertentu. Oleh karena itu, data dan diagram aliran kontrol menyediakan banyak informasi yang diperlukan untuk desain antarmuka.

Desain Tingkat Komponen

- Desain tingkat komponen mengubah elemen struktural dari arsitektur perangkat lunak menjadi deskripsi prosedural komponen perangkat lunak.

Prinsip desain

1. Proses desain seharusnya tidak mengalami "tunnel vision".
2. Desain harus dapat dilacak ke model analisis.
3. Desain tidak harus menemukan kembali roda.
4. Desain harus "meminimalkan jarak intelektual" antara perangkat lunak dan masalah yang ada di dunia nyata.

5. Desain harus menunjukkan keseragaman dan integrasi.
6. Desain harus terstruktur untuk mengakomodasi perubahan.
7. Desain harus terstruktur untuk menurunkan secara perlahan, bahkan ketika data, kejadian, atau operasi di dalam kondisi abnormal ditemui.
8. Desain tidak coding, coding bukan desain.
9. Desain harus dinilai kualitasnya karena dibuat, bukan setelah fakta.
10. Desain harus ditinjau untuk meminimalkan kesalahan konseptual (semantik).

5.2 Arsitektur perangkat lunak dan desain perangkat lunak

- Desain arsitektur merepresentasikan struktur data dan komponen program yang dibutuhkan untuk membangun sistem berbasis komputer.
- Ini mempertimbangkan gaya arsitektur yang akan diambil sistem, struktur dan komponen sifat-sifatnya yang membentuk sistem, dan hubungan timbal balik yang terjadi di antara semua komponen arsitektur sistem.
- Representasi arsitektur perangkat lunak merupakan faktor yang memungkinkan komunikasi antar semua pihak (stakeholder) tertarik dengan pengembangan sistem berbasis komputer.
- Arsitektur menyoroti keputusan desain awal yang akan memiliki dampak reflektif pada semua rekayasa perangkat lunak yang mengikuti dan, yang penting pada keberhasilan akhir sistem sebagai kesatuan operasional
- Arsitektur "merupakan model yang relatif kecil, secara intelektual dapat dipahami bagaimana sistem itu terstruktur dan bagaimana komponennya bekerja bersama ”

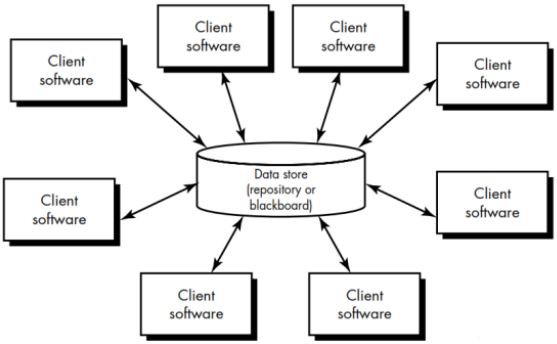
Gaya Arsitektur

Perangkat lunak yang dibangun untuk sistem berbasis komputer juga menunjukkan salah satu dari banyak gaya arsitektur.Setiap gaya menggambarkan kategori sistem yang mencakup

1. Seperangkat komponen (misalnya, basis data, modul komputasi) yang menjalankan fungsi yang dibutuhkan oleh sistem.
2. Satu set konektor yang memungkinkan "komunikasi, koordinasi dan kerja sama" di antara komponen.
3. Batasan yang menentukan bagaimana komponen dapat diintegrasikan untuk membentuk sistem.
4. Model semantik yang memungkinkan seorang desainer untuk memahami sifat keseluruhan dari sistem oleh menganalisis sifat-sifat yang dikenal dari bagian-bagian penyusunnya.

Gaya arsitektur yang berpusat pada data

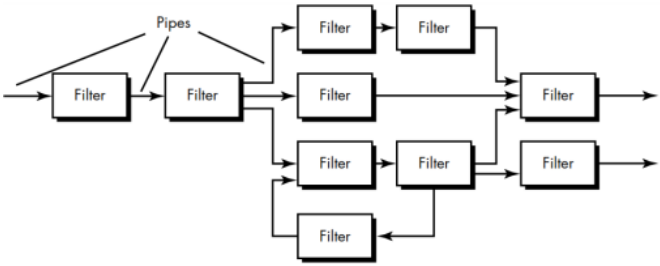
- Penyimpanan data (mis., File atau basis data) berada di pusat arsitektur ini dan sering diakses oleh komponen lain yang memperbarui, menambah, menghapus, atau memodifikasi data di dalam toko.
- Perangkat lunak klien mengakses repositori pusat.
- Dalam beberapa kasus, repositori data bersifat pasif.
- Artinya, perangkat lunak klien mengakses data independen dari setiap perubahan pada data atau tindakan perangkat lunak klien lainnya



Gambar 5.1 Gaya arsitektur yang berpusat pada data

Arsitektur aliran data

- Arsitektur ini diterapkan ketika data input harus diubah melalui serangkaian komputasi atau komponen manipulatif menjadi data output.
- Pola pipa dan filter memiliki seperangkat komponen, yang disebut filter, dihubungkan oleh pipa yang mengirimkan data dari satu komponen ke komponen berikutnya.
- Setiap filter bekerja secara independen dari komponen-komponen hulu dan hilir, dirancang untuk input data dari bentuk tertentu, dan menghasilkan output data (ke filter berikutnya) dari bentuk yang ditentukan.
- Namun, filter tidak memerlukan pengetahuan tentang pengerjaan filter tetangganya.



Gambar 5.2. Arsitektur Aliran Data

Call dan Return arsitektur

- Gaya arsitektur ini memungkinkan perancang perangkat lunak (arsitek sistem) untuk mencapai struktur program yang relatif mudah dimodifikasi dan diskalakan.

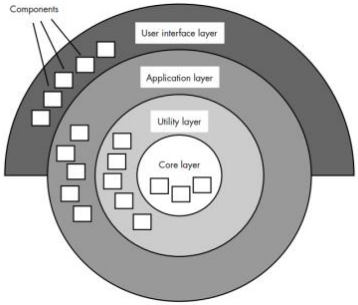
  Sejumlah gaya sub ada dalam kategori ini:
- Arsitektur program / subprogram utama. Struktur program klasik ini mengurai fungsi menjadi kontrol hirarki di mana program "utama" memanggil sejumlah komponen program, yang pada gilirannya mungkin mengaktifkan komponen lain.
- *Remote procedure call architectures* Komponen-komponen dari program utama / arsitektur subprogram adalah didistribusikan di beberapa komputer di jaringan.

Arsitektur Berorientasi objek

- Komponen-komponen suatu sistem merangkum data dan operasi yang harus diterapkan untuk memanipulasi data.
- Komunikasi dan koordinasi antar komponen diselesaikan melalui pengiriman pesan

Arsitektur berlapis

- Sejumlah lapisan yang berbeda didefinisikan, masing-masing menyelesaikan operasi yang semakin dekat menjadi ke set instruksi mesin.
- Pada lapisan luar, komponen melayani operasi antarmuka pengguna.
- Pada lapisan dalam, komponen melakukan interfacing sistem operasi.
- Lapisan perantara menyediakan layanan utilitas dan fungsi aplikasi perangkat lunak



Gambar 5.3 Arsitektur Berlapis

5.3 Desain Data

Bagian ini mendeskripsikan desain data pada level arsitektural dan komponen. Di tingkat arsitektur, desain data adalah proses menciptakan model informasi yang diwakili pada tingkat abstraksi yang tinggi (menggunakan tampilan data pelanggan).

Desain Data di Tingkat Arsitektur

- Tantangannya adalah mengekstrak informasi yang berguna dari lingkungan data, terutama ketika informasi yang diinginkan bersifat lintas fungsional.
- Untuk mengatasi tantangan ini, komunitas bisnis TI telah mengembangkan teknik penambangan data, juga disebut Knowledge discovery in database(KDD), yang menavigasi melalui database yang ada dalam upaya untuk ekstrak informasi tingkat bisnis yang sesuai.
- Namun, keberadaan beberapa basis data, strukturnya yang berbeda, dan tingkat detailnya terkandung dengan database, dan banyak faktor lainnya membuat data mining sulit dalam yang ada lingkungan basis data.
- Solusi alternatif, yang disebut data warehouse, menambahkan pada lapisan tambahan ke arsitektur data.
- Sebuah data warehouse adalah lingkungan data terpisah yang tidak terintegrasi secara langsung dengan aplikasi yang mencakup semua data yang digunakan oleh bisnis.

Desain Data di Tingkat Komponen

Pada tingkat komponen, desain data berfokus pada struktur data tertentu yang diperlukan untuk mewujudkan objek data dimanipulasi oleh komponen.

- ✓ Sempurnakan objek data dan kembangkan satu set abstraksi data
- ✓ Menerapkan atribut objek data sebagai satu atau lebih struktur data
- ✓ Tinjau struktur data untuk memastikan bahwa hubungan yang tepat telah ditetapkan

Kumpulan prinsip untuk spesifikasi data:

1. Prinsip analisis sistematis yang diterapkan pada fungsi dan perilaku juga harus diterapkan pada data.
2. Semua struktur data dan operasi yang harus dilakukan pada masing-masing harus diidentifikasi.
3. Kamus data harus dibuat dan digunakan untuk menentukan data dan desain program.
4. Keputusan desain data tingkat rendah harus ditunda sampai akhir proses desain.
5. Representasi struktur data seharusnya hanya diketahui oleh modul yang harus langsung digunakan dari data yang terkandung dalam struktur.
6. Sebuah pustaka struktur data yang berguna dan operasi yang dapat diterapkan padanya harus dikembangkan.
7. Perancangan perangkat lunak dan bahasa pemrograman harus mendukung spesifikasi dan realisasi abstrak tipe data.

5.4 Desain Tingkat Komponen atau Desain Prosedural

- Desain tingkat komponen, juga disebut desain prosedural, terjadi setelah data, arsitektur, dan antarmuka desain telah ditetapkan.
- Desain tingkat komponen menentukan struktur data, algoritme, karakteristik antarmuka, dan mekanisme komunikasi yang dialokasikan untuk setiap komponen perangkat lunak.
- Maksudnya adalah menerjemahkan model desain ke dalam perangkat lunak operasional.
- Tetapi tingkat abstraksi model desain yang ada relatif tinggi, dan tingkat abstraksi dari program operasionalnya rendah.
- Komponen bagian modular, dapat digunakan, dan dapat diganti dari sistem yang merangkum implementasi dan memperlihatkan satu set antarmuka.

Pendekatan Berorientasi Fungsi

Berikut ini adalah fitur yang menonjol dari pendekatan desain fungsi-berorientasi khas:

1. Suatu sistem dipandang sebagai sesuatu yang melakukan serangkaian fungsi. Mulai dari tampilan tingkat tinggi sistem , setiap fungsi secara berturut-turut disempurnakan menjadi fungsi yang lebih terperinci. Sebagai contoh, pertimbangkan fungsi buat-anggota baru-perpustakaan yang pada dasarnya membuat catatan untuk anggota baru, memberikan nomor keanggotaan unik kepadanya, dan mencetak tagihan ke rekeningnya biaya keanggotaan. Fungsi ini dapat terdiri dari sub-fungsi berikut:

   - ✓ assign-membership-number

- ✓ Rekam Anggota
- ✓ cetak tagihan

Masing-masing sub-fungsi ini dapat dibagi menjadi sub-fungsi yang lebih rinci dan seterusnya.

2. Status sistem dipusatkan dan dibagikan di antara fungsi yang berbeda, mis. data seperti keanggotaan kata sandi tersedia untuk referensi dan memperbarui ke beberapa fungsi seperti:

- ✓ buat anggota baru
- ✓ hapus anggota
- ✓ update Keanggotaan

Pendekatan Berorientasi Objek

- Dalam pendekatan desain berorientasi objek, sistem dilihat sebagai kumpulan objek (yaitu entitas). Sebagai satuan yang terdesentralisasi di antara objek dan setiap objek mengelola informasi sendiri.
- Sebagai contoh, dalam Perangkat Lunak Otomasi Perpustakaan, setiap anggota perpustakaan dapat menjadi objek terpisah dengan memiliki data dan fungsi untuk beroperasi pada data ini. Bahkan, fungsi yang ditetapkan untuk satu objek tidak bisa merujuk atau mengubah data dari objek lain.
- Objek memiliki data internal mereka sendiri yang mendefinisikan negara mereka. Objek serupa merupakan kelas.
- Dengan kata lain, setiap objek adalah anggota dari beberapa kelas. Kelas dapat mewarisi fitur dari kelas super. Secara konseptual, objek berkomunikasi melalui pengiriman pesan

5.5 Kohesi dan Kopling

- Kohesi merupakan indikasi kekuatan fungsional relatif dari suatu modul.
- Modul yang kohesif melakukan satu tugas, membutuhkan sedikit interaksi dengan komponen lain di bagian lain bagian dari suatu program. Secara sederhana, modul yang kohesif seharusnya (idealnya) hanya melakukan satu hal.
- Kohesi adalah ukuran kekuatan fungsional dari sebuah modul.
- Sebuah modul yang memiliki kohesi tinggi dan kopling rendah dikatakan independen secara fungsional dari modul lain. Dengan istilah kemandirian fungsional, kita berarti bahwa modul kohesif melakukan satu tugas atau fungsi.
- Coupling merupakan indikasi interdependensi relatif antar modul.
- Coupling tergantung pada kompleksitas antarmuka antar modul, titik di mana entri atau referensi berada dibuat untuk modul, dan data apa yang

melewati antarmuka.

- Sebuah modul yang memiliki kohesi tinggi dan kopling rendah dikatakan independen secara fungsional dari modul lain. Jika dua modul pertukaran data dalam jumlah besar, maka mereka sangat saling bergantung.
- Tingkat kopling antara dua modul tergantung pada kompleksitas antarmuka

Klasifikasi Kohesi

Kohesi Coincidental

- ✓ Sebuah modul dikatakan memiliki kohesi Coincidental, jika ia melakukan serangkaian tugas yang berhubungan satu sama lain sangat longgar, jika sama sekali.
- ✓ Dalam hal ini, modul berisi kumpulan fungsi acak. Sangat mungkin bahwa fungsi-fungsi tersebut dimasukkan ke dalam modul dari kebetulan murni tanpa pemikiran atau desain apa pun.
- ✓ Misalnya, dalam sistem pemrosesan transaksi (TPS), get-input, print-error, dan summarizemembers fungsi dikelompokkan ke dalam satu modul.

Kohesi logis

- ✓ Sebuah modul dikatakan logis secara kohesif, jika semua elemen dari modul melakukan operasi serupa, mis. penanganan kesalahan, input data, output data, dll.
- ✓ Contoh kohesi logis adalah kasus di mana satu set fungsi cetak menghasilkan output yang berbeda laporan disusun ke dalam satu modul.

Kohesi temporal

- ✓ Ketika modul berisi fungsi yang terkait dengan fakta bahwa semua fungsi harus dijalankan rentang waktu yang sama, modul dikatakan menunjukkan kohesi temporal.
- ✓ Kumpulan fungsi yang bertanggung jawab untuk inisialisasi, start-up, penutupan beberapa proses, dll. Pameran kohesi temporal.

Kohesi prosedural

- ✓ Sebuah modul dikatakan memiliki kohesi prosedural, jika himpunan fungsi modul adalah bagian dari prosedur (algoritma) di mana urutan langkah tertentu harus dilakukan untuk mencapai suatu obyektif, mis. algoritma untuk memecahkan kode pesan.

Kohesi komunikatif

- ✓ Sebuah modul dikatakan memiliki kohesi komunikasi, jika semua fungsi modul mengacu pada atau memperbarui struktur data yang sama, mis. set fungsi yang didefinisikan pada array atau stack.

Kohesi sekuensial

- ✓ Sebuah modul dikatakan memiliki kohesi berurutan, jika elemen-elemen suatu modul membentuk bagian-bagian dari urutan, di mana output dari satu elemen urutan adalah input ke yang berikutnya.
- ✓ Misalnya, dalam TPS, fungsi input-masukan, validasi-masukan, pengurutan-masukan dikelompokkan ke dalam satu modul

Kohesi fungsional

- ✓ Kohesi fungsional dikatakan ada, jika elemen yang berbeda dari sebuah modul bekerja sama untuk mencapai satu fungsi. Misalnya, modul yang berisi semua fungsi yang diperlukan untuk mengelola daftar gaji karyawan menunjukkan kohesi fungsional.
- ✓ Misalkan modul menunjukkan kohesi fungsional dan kami diminta untuk mendeskripsikan apa yang dilakukan modul, maka kita akan bisa menggambarkannya menggunakan satu kalimat.

Klasifikasi Coupling

Data coupling

Dua modul digabungkan data, jika mereka berkomunikasi melalui parameter. Contohnya adalah item data dilewatkan sebagai parameter antara dua modul, mis. integer, pelampung, karakter, dll. Item data ini harus terkait masalah dan tidak digunakan untuk tujuan kontrol

Stamp coupling

Dua modul dicap digabungkan, jika mereka berkomunikasi menggunakan item data gabungan seperti catatan dalam PASCAL atau struktur dalam C.

Control coupling

Kontrol kopling ada antara dua modul, jika data dari satu modul yang digunakan untuk mengarahkan urutan eksekusi instruksi di tempat lain. Contoh kopling kontrol adalah flag yang diatur dalam satu modul dan diuji dalam modul lain.

Common coupling

Dua modul umumnya digabungkan, jika mereka berbagi data melalui beberapa item data global

Content coupling

kopling konten ada antara dua modul, jika mereka berbagi kode, misalnya cabang dari satu modul ke dalam modul lain.

5.6 Desain Antarmuka Pengguna

- Desain antarmuka pengguna menciptakan media komunikasi yang efektif antara manusia dan komputer.
- Mengikuti serangkaian prinsip desain antarmuka, desain mengidentifikasi objek dan tindakan antarmuka dan kemudian menciptakan tata letak layar yang membentuk dasar untuk prototipe antarmuka pengguna.

Aturan Desain untuk Antarmuka Pengguna

1. Tempatkan Pengguna dalam Kontrol
    - Selama sesi pengumpulan-permintaan untuk sistem informasi baru yang besar, seorang pengguna kunci ditanyai. Berikut ini adalah prinsip desain yang memungkinkan pengguna untuk mempertahankan kontrol:
    - Tentukan mode interaksi dengan cara yang tidak memaksa pengguna menjadi tidak perlu atau tindakan yang tidak diinginkan
    - Modus interaksi adalah keadaan antarmuka saat ini. Misalnya, jika pemeriksaan ejaan dipilih dalam menu pengolah kata, perangkat lunak bergerak ke mode pemeriksaan ejaan. Tidak ada alasan untuk memaksakan pengguna untuk tetap berada dalam mode pemeriksa ejaan jika pengguna ingin membuat pengeditan teks kecil di sepanjang jalan.
    - Menyediakan untuk interaksi fleksibel.
    - Karena pengguna yang berbeda memiliki preferensi interaksi yang berbeda, pilihan harus disediakan. Untuk
    - Misalnya, perangkat lunak mungkin memungkinkan pengguna untuk berinteraksi melalui perintah keyboard, gerakan mouse, digitizer pen, layar multi sentuh, atau perintah pengenalan suara.
    - Memungkinkan interaksi pengguna menjadi interruptible
    - Bahkan ketika terlibat dalam serangkaian tindakan, pengguna harus dapat mengganggu urutan yang harus dilakukan sesuatu yang lain.
    - Perampingkan interaksi saat tingkat keterampilan maju dan biarkan interaksi disesuaikan.
    - Pengguna sering menemukan bahwa mereka melakukan urutan interaksi yang sama berulang kali.
    - Sembunyikan internal teknis dari pengguna biasa.
    - Antarmuka pengguna harus memindahkan pengguna ke dunia maya dari aplikasi. Pengguna tidak seharusnya sadar akan sistem operasi, fungsi manajemen file, atau komputasi teknologi lainnya
    - Desain untuk interaksi langsung dengan objek yang muncul di layar.
    - Pengguna merasakan kendali ketika mampu memanipulasi objek yang perlu dilakukan sebuah tugas dengan cara yang mirip dengan apa yang akan terjadi jika objek itu adalah benda fisik.

2.Kurangi Beban Memori Pengguna

- Semakin user harus ingat, semakin rawan kesalahan interaksi dengan sistem.
- Berikut ini adalah prinsip desain yang memungkinkan antarmuka untuk mengurangi beban memori pengguna:

- Mengurangi permintaan pada memori jangka pendek.
- Ketika pengguna terlibat dalam tugas-tugas kompleks, permintaan pada memori jangka pendek dapat menjadi signifikan. Itu antarmuka harus dirancang untuk mengurangi persyaratan mengingat tindakan, masukan, dan hasil.
- Tentukan default yang berarti.
- Set default awal harus masuk akal untuk rata-rata pengguna, tetapi pengguna harus dapat menentukan preferensi individu. Namun, opsi "reset" harus tersedia, memungkinkan redefinisi nilai default asli.
- Tentukan pintasan yang intuitif.
- Ketika mnemonik digunakan untuk menyelesaikan suatu fungsi sistem, mnemonic harus diikat ke tindakan dengan cara yang mudah diingat.
- Tata letak visual antarmuka harus didasarkan pada metafora dunia nyata.
- Hal ini memungkinkan pengguna untuk mengandalkan isyarat visual yang dipahami dengan baik, daripada menghafal sebuah rahasia urutan interaksi.
- Mengungkapkan informasi secara progresif.
- Antarmuka harus diatur secara hierarkis. Yaitu, informasi tentang tugas, objek, atau beberapa perilaku harus disajikan pertama pada tingkat abstraksi yang tinggi.

3.Buat Antarmuka Konsisten

Antarmuka harus menyajikan dan memperoleh informasi secara konsisten. Berikut adalah prinsip desain yang membantu membuat antarmuka konsisten:

- Memungkinkan pengguna untuk memasukkan tugas saat ini ke dalam konteks yang bermakna. Banyak antarmuka mengimplementasikan lapisan kompleks interaksi dengan puluhan gambar layar. Ini penting untuk menyediakan indikator yang memungkinkan pengguna untuk mengetahui konteks pekerjaan yang sedang dikerjakan.
- Menjaga konsistensi di seluruh bagian aplikasi.
- Seperangkat aplikasi semua harus menerapkan aturan desain yang sama sehingga konsistensi dipertahankan untuk semua interaksi.
- Jika model interaktif yang lalu telah menciptakan harapan pengguna, jangan membuat perubahan kecuali jika ada alasan kuat untuk melakukannya. Setelah urutan interaktif tertentu telah menjadi standar de facto, pengguna mengharapkan ini di setiap aplikasi yang dia temui.

BAB VII. TEKNIK PENGUJIAN PERANGKAT LUNAK

5.1 Dasar-dasar Pengujian Perangkat Lunak

Karakteristik berikut ini mengarah ke perangkat lunak yang dapat diuji.

1. Operabilitas.

- "Semakin baik kerjanya, semakin efisien itu dapat diuji."
- Jika suatu sistem dirancang dan dilaksanakan dengan kualitas dalam pikiran, relatif sedikit bug akan memblokir eksekusi tes, memungkinkan pengujian untuk maju tanpa cocok dan dimulai.

2. Observability .

- "Apa yang Anda lihat adalah apa yang Anda uji."
- Masukan yang disediakan sebagai bagian dari pengujian menghasilkan output yang berbeda.
- Status dan variabel sistem terlihat atau dapat dipulihkan selama eksekusi. Output yang salah mudah diidentifikasi. Kesalahan internal secara otomatis terdeteksi dan dilaporkan. Kode sumber dapat diakses.

3. Controlability

- **"**Semakin baik kita dapat mengontrol perangkat lunak, semakin banyak pengujian dapat diotomatisasi dan dioptimalkan."
- Semua output yang mungkin dapat dihasilkan melalui beberapa kombinasi input, dan format I / O berada konsisten dan terstruktur.
- Semua kode dapat dieksekusi melalui beberapa kombinasi input. Status perangkat lunak dan perangkat keras dan variabel dapat dikontrol langsung oleh insinyur uji.
- Tes dapat dengan mudah ditentukan, diotomatisasi, dan direproduksi.

4. Dekomposisi.

- “Dengan mengendalikan ruang lingkup pengujian, kita dapat lebih cepat mengisolasi masalah dan melakukan lebih pintar pengujian ulang. "
- Sistem perangkat lunak dibangun dari modul independen yang dapat diuji secara independen.

5.Simplicity.

- “Semakin sedikit ada tes, semakin cepat kita bisa mengujinya.”

- Program harus memperlihatkan kesederhanaan fungsional (misalnya, set fitur adalah persyaratan minimum yang harus dipenuhi Persyaratan); kesederhanaan struktural (misalnya, arsitektur dimodulasikan untuk membatasi penyebaran kesalahan), dan kode kesederhanaan (misalnya, standar pengkodean diadopsi untuk kemudahan pemeriksaan dan pemeliharaan).

5. Stability.

- "Semakin sedikit perubahan, semakin sedikit gangguan untuk menguji."
- Perubahan pada perangkat lunak jarang terjadi, terkontrol ketika terjadi, dan tidak membatalkan yang ada.
- Perangkat lunak pulih dari kegagalan.

3. *Understandability*

- "Semakin banyak informasi yang kami miliki, semakin pintar kami akan menguji."
- Desain arsitektur dan ketergantungan antara komponen internal, eksternal, dan bersama adalah dimengerti.
- Dokumentasi teknis dapat diakses dengan cepat, terorganisir dengan baik, spesifik dan rinci, serta akurat.Perubahan pada desain dikomunikasikan kepada penguji.

5.2 White Box Testing and Black Box Testing

- White-box testing, kadang-kadang disebut glass-box testing, adalah filosofi desain pengujian yang menggunakan struktur kontrol yang digambarkan sebagai bagian dari desain tingkat komponen untuk menurunkan kasus uji. Dengan menggunakan metode pengujian white-box, dapat menurunkan kasus uji dengan cara :
    a. Menjamin bahwa semua jalur independen dalam modul telah dilakukan setidaknya sekali.
    b. Melatih semua keputusan logis di sisi mereka yang benar dan salah.
    c. Melatih semua loop dalam batas operasional.
    d. Melatih struktur data internal untuk memastikan validitasnya.

Metode White Box Testing berlaku untuk tingkat pengujian perangkat lunak Ini terutama diterapkan pada pengujian Unit dan pengujian Integrasi

a. Pengujian Unit: Untuk menguji jalur dalam satu unit.

b. Pengujian Integrasi: Untuk menguji jalur antar unit.
c. Pengujian Sistem: Untuk menguji jalur antar subsistem.

Keuntungan white Box Testing

a. Pengujian dapat dimulai pada tahap awal. Satu tidak perlu menunggu GUI akan tersedia.
b. Pengujian lebih teliti, dengan kemungkinan mencakup sebagian besar jalur.

Kelemahan pengujian kotak putih

a. Karena tes bisa sangat kompleks, diperlukan sumber daya yang sangat terampil, dengan pengetahuan mendalam tentang pemrograman dan implementasi.
b. Pemeliharaan skrip uji dapat menjadi beban jika implementasi terlalu sering berubah.
c. Karena metode pengujian ini terkait erat dengan aplikasi yang sedang diuji, alat untuk memenuhi setiap jenis implementasi / platform mungkin tidak tersedia.

- **Black-box Testing, juga disebut pengujian perilaku, berfokus pada persyaratan fungsional perangkat lunak.**
    a. Artinya, teknik pengujian black-box memungkinkan Anda untuk menentukan set kondisi input yang akan sepenuhnya sesuai dengan
    b. Semua persyaratan fungsional untuk suatu program.
    c. Pengujian black-box bukan merupakan alternatif teknik white-box. Sebaliknya, itu adalah pendekatan pelengkap yang kemungkinan akan mengungkap kelas kesalahan yang berbeda dari metode kotak putih.
    d. Pengujian Black Box berupaya menemukan kesalahan dalam kategori berikut:
        - Fungsi salah atau tidak ada
        - kesalahan Antarmuka
        - Kesalahan dalam struktur data atau akses database eksternal
        - Perilaku atau kesalahan kinerja
        - Inisialisasi dan terminasi error.
    e. Metode Pengujian Kotak Hitam berlaku untuk tingkat pengujian perangkat lunak berikut:
        - Ini terutama diterapkan pada pengujian Sistem dan pengujian Penerimaan
        - Pengujian integrasi

- Pengujian Sistem
- Acceptance testing

- Semakin tinggi level, dan karenanya semakin besar dan kompleksnya kotak,maka semakin banyak pengujian black box digunakan.

Kerugian Black Box Testing

- Hanya sejumlah kecil input yang mungkin dapat diuji dan banyak jalur program akan dibiarkan belum teruji.
- Tanpa spesifikasi yang jelas, yang merupakan situasi di banyak proyek, kasus pengujian akan sulit untuk dirancang.
- Tes dapat berlebihan jika perancang / pengembang perangkat lunak sudah menjalankan test case.
- Pernah bertanya-tanya mengapa seorang peramal menutup mata ketika meramalkan peristiwa? Jadi hampir terjadi di Black Box Testing.

BAB VIII. Strategi Pengujian Perangkat Lunak

Strategi Pengujian

Pengujian perangkat lunak adalah elemen penting dari jaminan kualitas perangkat lunak dan mewakili tinjauan akhir spesifikasi, desain, dan pembuatan kode. Tidak biasa bagi organisasi pengembangan perangkat lunak untuk membayar antara 30 dan 40 persen dari total upaya proyek dalam pengujian. Insinyur menciptakan serangkaian uji kasus yang dimaksudkan untuk "mengalahkan" perangkat lunak yang telah dibangun. Bahkan, pengujian adalah satu langkah dalam proses perangkat lunak yang dapat dilihat (secara psikologis, setidaknya) sebagai destruktif daripada konstruktif.

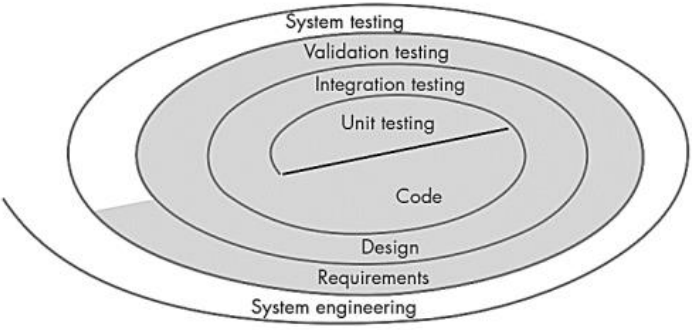
Jenis Pendekatan Pengujian

1. Verifikasi dan pendekatan Validasi:
2. Verifikasi mengacu pada serangkaian aktivitas yang memastikan bahwa perangkat lunak mengimplementasikan fungsi spesifik dengan benar.
3. Validasi mengacu pada serangkaian aktivitas yang berbeda yang memastikan bahwa perangkat lunak yang telah dibangun dapat dilacak ke persyaratan pelanggan.
4. Verifikasi: "Apakah kita membangun produk dengan benar?"
5. Validasi: "Apakah kita membangun produk yang tepat?

Strategi Pengujian

Awalnya, rekayasa sistem mendefinisikan peran perangkat lunak dan mengarah ke analisis persyaratan perangkat lunak, di mana domain informasi, fungsi, perilaku, kinerja, batasan, dan kriteria validasi untuk

perangkat lunak yang dibuat.



Gambar Strategi pengujian

Bergerak ke dalam sepanjang spiral, mulai dari merancang dan akhirnya untuk coding. Untuk mengembangkan perangkat lunak komputer, Bergerak spiral ke dalam (berlawanan arah jarum jam) sepanjang garis arus yang mengurangi tingkat abstraksi pada masing-masing belok.

Pengujian Unit

1. Unit adalah bagian terkecil dari sistem perangkat lunak yang dapat diuji itu mungkin termasuk file kode, kelas dan metode yang dapat diuji individu untuk kebenaran.
2. Unit adalah proses validasi blok bangunan kecil dari sistem yang kompleks, jauh sebelum menguji suatu modul besar terintegrasi atau sistem secara keseluruhan.
3. Driver dan / atau perangkat lunak rintisan harus dikembangkan untuk setiap unit tes driver tidak lebih dari " program utama "yang menerima data uji coba, melewatkan data tersebut ke komponen, dan mencetak hasil yang relevan.
4. Stub(Rintisan) berfungsi untuk menggantikan modul yang lebih rendah (dipanggil oleh) komponen yang akan diuji.
5. Sebuah rintisan(stub) atau " dummy subprogram " menggunakan antarmuka modul bawahan.

Pengujian terintegrasi:

1. Integrasi didefinisikan sebagai satu set integrasi antar komponen.
2. Menguji interaksi antara modul dan interaksi dengan sistem lain secara eksternal disebut pengujian integrasi.
3. Pengujian modul terintegrasi untuk memverifikasi fungsi gabungan setelah integrasi.

4. Pengujian integrasi membahas masalah yang terkait dengan masalah ganda verifikasi dan program konstruksi.
5. Modul biasanya adalah modul kode, aplikasi individual, aplikasi klien dan server pada jaringan, dll. Jenis pengujian ini sangat relevan dengan klien / server dan sistem terdistribusi.
6. Jenis pengujian integrasi adalah:
    a. Integrasi top-down
    b. Integrasi bottom-up
    c. Pengujian regresi
    d. Pengujian Smoke

Pengujian Validasi

1. Proses evaluasi perangkat lunak selama proses pengembangan atau di akhir pengembangan proses untuk menentukan apakah memenuhi persyaratan bisnis yang ditentukan.
2. Pengujian Validasi memastikan bahwa produk benar-benar memenuhi kebutuhan klien. Ini juga dapat didefinisikan sebagai
3. menunjukkan bahwa produk memenuhi penggunaan yang dimaksudkan ketika digunakan pada lingkungan yang sesuai.
4. Pengujian validasi memberikan jaminan akhir bahwa perangkat lunak memenuhi semua informasi, fungsional, perilaku dan persyaratan kinerja.
5. Pengujian alfa dilakukan di situs pengembang oleh sekelompok pengguna akhir yang representatif.
6. Perangkat lunak ini digunakan dalam pengaturan alami dengan pengembang "melihat ke balik bahu" dari pengguna dan kesalahan rekaman dan masalah penggunaan.
7. Tes alfa dilakukan di lingkungan yang terkontrol. Tes beta dilakukan di satu atau beberapa situs pengguna akhir.
8. Tidak seperti pengujian alfa, pengembang umumnya tidak ada.
9. Oleh karena itu, pengujian beta adalah aplikasi "langsung" dari perangkat lunak dalam lingkungan yang tidak bisa dikendalikan oleh pengembang.

Pengujian Sistem

1. Dalam pengujian sistem perangkat lunak dan elemen sistem lainnya diuji secara keseluruhan.

2. Untuk menguji perangkat lunak komputer, berputar ke arah searah jarum jam sepanjang garis arus yang meningkatkan ruang lingkup pengujian dengan setiap giliran.
3. Pengujian sistem memverifikasi bahwa semua elemen terhubung dengan benar dan keseluruhan fungsi / kinerja sistem tercapai.
4. **Recovery Testing** adalah tes sistem yang memaksa perangkat lunak untuk gagal dalam berbagai cara dan memverifikasi pemulihan yang dilakukan dengan benar.
5. Jika pemulihan dilakukan secara otomatis (dilakukan oleh sistem itu sendiri), inisialisasi ulang, mekanisme penunjuk cek, pemulihan data, dan restart dievaluasi untuk kebenaran. Jika pemulihan membutuhkan intervensi manusia, maka mean-time-to-repair (MTTR) dievaluasi untuk menentukan apakah itu dalam batas yang dapat diterima.
6. **Security Testing** mencoba untuk memverifikasi bahwa mekanisme perlindungan yang dibangun ke dalam suatu sistem akan, pada kenyataannya, melindunginya dari penetrasi yang tidak tepat.
7. Selama pengujian keamanan, tester memainkan peran dari individu yang ingin menembus sistem.
8. **Stress testing** mengeksekusi suatu sistem dengan cara yang menuntut sumber daya dalam jumlah, frekuensi, abnormal, atau volume.
9. Variasi stress testing adalah teknik yang disebut tes sensitivitas.
10. **Performance Testing** dirancang untuk menguji kinerja run-time perangkat lunak dalam konteks suatu sistem terintegrasi.
11. Pengujian kinerja terjadi di seluruh langkah dalam proses pengujian.
12. Bahkan di tingkat unit, kinerja modul individu dapat dinilai saat tes dilakukan.
13. **Deployment Testing**, terkadang disebut pengujian konfigurasi, menjalankan perangkat lunak di setiap lingkungan di mana itu beroperasi.
14. Selain itu, pengujian penyebaran memeriksa semua prosedur instalasi dan perangkat lunak instalasi khusus yang akan digunakan oleh pelanggan, dan semua dokumentasi yang akan digunakan untuk memperkenalkan perangkat lunak untuk mengakhiri pengguna.

Acceptance Testing

Acceptance Testing adalah tingkat pengujian perangkat lunak di mana suatu sistem diuji untuk dapat diterima.

1. Tujuan dari tes ini adalah untuk mengevaluasi kepatuhan sistem dengan persyaratan bisnis dan menilai apakah itu dapat diterima untuk pengiriman.
2. Ini adalah pengujian formal sehubungan dengan kebutuhan pengguna, persyaratan, dan proses bisnis yang dilakukan untuk menentukan apakah suatu sistem memenuhi kriteria penerimaan dan untuk memungkinkan pengguna, pelanggan atau entitas resmi lainnya untuk menentukan apakah atau tidak untuk menerima sistem.
3. **Acceptance Testing** dilakukan setelah Pengujian Sistem dan sebelum membuat sistem secara aktual digunakan

Verifikasi dan Validasi

Verifikasi	**Validasi**
Verifikasi adalah praktik verifikasi statis dokumen, desain, kode dan program.	Validasi adalah mekanisme validasi dinamis dan menguji produk yang sebenarnya
Ini tidak melibatkan mengeksekusi kode	Itu selalu melibatkan mengeksekusi kode.
Ini adalah pemeriksaan dokumen berdasarkan manusia dan file	Ini adalah eksekusi program berbasis komputer.
Verifikasi menggunakan metode seperti inspeksi, ulasan, penelusuran, dan pemeriksaan dll	Validasi menggunakan metode seperti kotak hitam (fungsional) pengujian, pengujian kotak abu-abu, dan putih kotak (struktural) pengujian dll.
Verifikasi adalah untuk memeriksa apakah perangkat lunak sesuai dengan spesifikasi	Validasi adalah untuk memeriksa apakah perangkat lunak bertemu harapan dan persyaratan pelanggan
Ini dapat menangkap kesalahan yang tidak dapat ditangkap oleh validasi. Ini adalah latihan tingkat rendah	Ini dapat menangkap kesalahan yang tidak dapat ditangkap oleh verifikasi. Ini adalah Latihan Tingkat Tinggi.
Target adalah spesifikasi kebutuhan, aplikasi dan arsitektur perangkat lunak, tingkat tinggi, lengkap desain, dan desain database dll	Target adalah produk aktual - unit, modul, bengkok modul terintegrasi, dan produk final yang efektif .
Verifikasi dilakukan oleh tim QA untuk memastikan itu perangkat lunak ini sesuai spesifikasi	Validasi dilakukan dengan keterlibatan tim penguji.

dalamDokumen SRS.	
Ini umumnya dilakukan pertama kali dilakukan sebelum validasi.	Biasanya mengikuti setelah verifikasi

Jaminan Software Quality Assurance (SQA)

Software Quality Assurance atau Jaminan kualitas perangkat lunak (sering disebut manajemen kualitas) adalah kegiatan payung yang diterapkan sepanjang proses perangkat lunak.

- Ini adalah pola kegiatan yang direncanakan dan sistematis yang diperlukan untuk memberikan tingkat kepercayaan yang tinggi pada kualitas suatu produk.
- Jaminan kualitas perangkat lunak (SQA) meliputi
  - ✓ Proses SQA.
  - ✓ Jaminan kualitas khusus dan tugas-tugas pengendalian mutu.
  - ✓ Praktik rekayasa perangkat lunak yang efektif.
  - ✓ Kontrol semua produk kerja perangkat lunak dan perubahan yang dibuat untuk mereka.
  - ✓ Prosedur untuk memastikan kepatuhan dengan standar pengembangan perangkat lunak.
  - ✓ Pengukuran dan mekanisme pelaporan.

Pentingnya SQA

- Kontrol kualitas dan jaminan adalah kegiatan penting untuk setiap bisnis yang menghasilkan produk yang akan digunakan oleh orang lain.
- Sebelum abad ke-20, kontrol kualitas adalah tanggung jawab dari pengrajin yang membangun sebuah produk.
- Seiring waktu berlalu dan teknik produksi massal menjadi hal yang biasa, kontrol kualitas menjadi aktivitas yang dilakukan oleh orang lain selain orang yang membangun produk.
- Kualitas perangkat lunak adalah salah satu aspek penting dari perusahaan pengembangan perangkat lunak.
- Jaminan kualitas perangkat lunak dimulai dari awal proyek, langsung dari tahap analisis.
- SQA memeriksa kepatuhan terhadap standar, proses, dan prosedur produk perangkat lunak.

- SQA mencakup proses sistematis untuk memastikan bahwa standar dan prosedur ditetapkan dan benar diikuti sepanjang siklus hidup pengembangan perangkat lunak dan siklus pengujian juga.
- Kepatuhan yang dibangun dengan standar dan prosedur yang disepakati dievaluasi melalui proses pemantauan, evaluasi produk, manajemen proyek dll.
- Alasan utama melibatkan jaminan kualitas perangkat lunak dalam proses produk perangkat lunak pengembangan adalah untuk memastikan bahwa produk akhir yang dibangun adalah sesuai spesifikasi kebutuhan dan mematuhi standar

Kegiatan SQA

Siapkan rencana SQA untuk sebuah proyek

- Rencana ini dikembangkan sebagai bagian dari perencanaan proyek dan ditinjau oleh semua pemangku kepentingan.
- Tindakan jaminan kualitas yang dilakukan oleh tim rekayasa perangkat lunak dan kelompok SQA diatur dengan rencananya.
- Rencana mengidentifikasi evaluasi yang harus dilakukan, audit dan tinjauan yang harus dilakukan, standar yang berlaku untuk proyek, prosedur untuk pelaporan kesalahan dan pelacakan, produk kerja yang dihasilkan oleh grup SQA, dan umpan balik yang akan diberikan kepada tim perangkat lunak.

Berpartisipasilah dalam pengembangan deskripsi proses perangkat lunak proyek

- Tim perangkat lunak memilih proses untuk pekerjaan yang akan dilakukan.
- Kelompok SQA meninjau deskripsi proses untuk kepatuhan dengan kebijakan organisasi, internal standar perangkat lunak, standar yang ditetapkan secara eksternal, dan bagian lain dari rencana proyek perangkat lunak.

Tinjau aktivitas rekayasa perangkat lunak untuk memverifikasi kepatuhan dengan proses perangkat lunak yang ditetapkan.

- Kelompok SQA mengidentifikasi, mendokumentasikan, dan melacak penyimpangan dari proses dan memverifikasi itu koreksi telah dilakukan.

Audit produk kerja perangkat lunak yang ditunjuk untuk memverifikasi kepatuhan dengan yang didefinisikan sebagai bagian dari proses perangkat lunak

- Kelompok SQA meninjau produk kerja yang dipilih; mengidentifikasi, mendokumentasikan, dan melacak penyimpangan; memverifikasi itu
- koreksi telah dilakukan; dan secara berkala melaporkan hasil kerjanya kepada manajer proyek.

Pastikan penyimpangan dalam pekerjaan perangkat lunak dan produk kerja didokumentasikan dan ditanganimenurut prosedur terdokumentasi.

- Penyimpangan mungkin ditemukan dalam rencana proyek, deskripsi proses, standar yang berlaku, atau produk kerja rekayasa perangkat lunak.
- Rekam semua ketidakpatuhan dan laporan kepada manajemen senior
- Item ketidakpatuhan dilacak sampai mereka terselesaikan

Teknik SQA

Pengumpulan data

Statistik Jaminan kualitas terdiri dari langkah-langkah berikut:

1. Informasi tentang cacat perangkat lunak dikumpulkan dan dikategorikan.
2. Upaya dilakukan untuk melacak setiap cacat ke penyebab dasarnya (misalnya, ketidaksesuaian dengan spesifikasi, kesalahan desain, pelanggaran standar, komunikasi yang buruk dengan pelanggan).
3. Menggunakan prinsip Pareto (80 persen cacat dapat dilacak sampai 20 persen dari semua kemungkinan penyebab), mengisolasi 20 persen ("beberapa yang penting").
4. Setelah beberapa penyebab penting telah diidentifikasi, bergerak untuk memperbaiki masalah yang telah menyebabkan cacat.

Sebuah organisasi rekayasa perangkat lunak mengumpulkan informasi tentang cacat untuk jangka waktu satu tahun.

- Beberapa cacat ditemukan karena perangkat lunak sedang dikembangkan.
- Lainnya ditemui setelah perangkat lunak dirilis ke pengguna akhir. Meski ratusan kesalahan yang berbeda ditemukan, semua dapat dilacak ke satu (atau lebih) dari penyebab berikut:
    1. spesifikasi yang tidak lengkap atau salah (IES)
    2. salah tafsir komunikasi pelanggan (MCC)
    3. penyimpangan yang disengaja dari spesifikasi (IDS)
    4. pelanggaran standar pemrograman (VPS)
    5. kesalahan dalam representasi data (EDR)
    6. antarmuka komponen tidak konsisten (ICI)
    7. kesalahan dalam logika desain (EDL)
    8. pengujian tidak lengkap atau salah (IET)
    9. dokumentasi tidak akurat atau tidak lengkap (IID)
    10. kesalahan dalam penerjemahan bahasa pemrograman desain (PLT)
    11. antarmuka manusia / komputer yang ambigu atau tidak konsisten (HCI)
    12. miscellaneous (MIS)

BATANG TUBUH

Fakultas	:	Teknik
Program Studi	:	Informatika
Mata Kuliah (MK)	:	Rekayasa Perangkat Lunak
Kode MK	:	TI00413
SKS	:	3 (tiga)
Semester	:	V (Lima)
Mata Kuliah Prasyarat	:	Sistem Informasi
StandarKompetensi (SK)	:	Mahasiswa memiliki pengetahuan terhadap metodologi pembangunan perangkat lunak melalui pendekatan struktur dan obyek, memahami teknik pengujian perangkat lunak, dan teknik perawatan perangkat lunak.

NO	Pokok Bahasan Dan TIU	Sub Pokok Bahasan dan TIK
BAB I	**Konsep dasar Rekayasa Perangkat** Mahasiswa dapat mengerti dan memahami konsep dasar rekayasa perangkat lunak	a. Mahasiswa mampu memahami Definisi perangkat lunak b. Mahasiswa mampu memahami Karakteristik perangkat lunak c. Mahasiswa mampu memahami Komponen perangkat lunak d. Mahasiswa mampu memahami Aplikasi perangkat lunak e. Mahasiswa mampu memahami Model perangkat lunak
BAB II	**Perencanaan Proyek Perangkat Lunak** Mahasiswa dapat memahami maksud dari perencanaan proyek perangkat lunak	a. Mahasiswa mampu memahami Observasi pada Estimasi b. Mahasiswa mampu memahami Tujuan Perencanaan Proyek c. Mahasiswa mampu memahami Ruang Lingkup Perangkat Lunak d. Mahasiswa mampu memahami Sumber Daya e. Mahasiswa mampu memahami Estimasi Proyek Perangkat Lunak
BAB III	**Konsep dan Prinsip Analisis** Mahasiswa dapat memahami konsep dan prinsip analisis	a. Mahasiswa dapat memahami Analisis Kebutuhan Perangkat Lunak b. Mahasiswa dapat memahami Teknik Komunikasi c. Mahasiswa dapat memahami Prinsip-prinsip analisis d. Mahasiswa dapat memahami Dokumen Analisis e. Mahasiswa dapat memahami Prototyping perangkat lunak f. Mahasiswa dapat memahami

		Spesifikasi dan kajian spesifikasi
BAB IV	**Pemodelan Analisis** Mahasiswa dapat memahami model yang digunakan dalam analisis	a. Mahasiswa dapat memahami Elemen Model Analisis b. Mahasiswa dapat memahami Pemodelan kebutuhan Fungsional c. Mahasiswa dapat memahami Pemodelan Proses d. Mahasiswa dapat memahami Pemodelan Data e. Mahasiswa dapat memahami Kamus Data f. Mahasiswa dapat memahami CRUD Matriks
BAB V	**Prinsip dan Konsep Desain** Mahasiswa dapat memahami prinsip dan Konsep desain perangkat lunak	a. Mahasiswa dapat memahami Desain perangkat lunak dan rekayasa Perangkat lunak b. Mahasiswa dapat memahami Prinsip Desain c. Mahasiswa dapat memahami Konsep Desain d. Mahasiswa dapat memahami Desain Modular Efektif e. Mahasiswa dapat memahami Model Desain f. Mahasiswa dapat memahami Dokumentasi Desain
BAB VI	**Metode Desain** Mahasiswa dapat memahami dan mengerti desain data dan arsitektur perangkat lunak	a. Mahasiswa dapat memahami Desain Data b. Mahasiswa dapat memahami Desain Arsitektur c. Mahasiswa dapat memahami Proses Desain Arsitektur

	Mahasiswa dapat memahami dan membuat spesifikasi desain Mahasiswa dapat memahami dan mengerti komponen-komponen interface	d. Mahasiswa dapat memahami Pasca Pemrosesan Desain e. Mahasiswa dapat memahami Optimasi Desain Arsitektur f. Mahasiswa dapat memahami Desain Interface g. Mahasiswa dapat memahami Desain Interface Manusia-Mesin h. Mahasiswa dapat memahami Desain Prosedural i. Mahasiswa dapat memahami Coding
BAB VII	**Teknik Pengujian Perangkat Lunak** Mahasiswa dapat memahami dan mengerti teknik-teknik pengujian perangkat lunak	a. Mahasiswa dapat memahami Dasar – dasar pengujian Perangkat Lunak b. Mahasiswa dapat memahami Desain Test Case c. Mahasiswa dapat memahami Pengujian White Box d. Mahasiswa dapat memahami Pengujian Struktur Kontrol e. Mahasiswa dapat memahami Pengujian Black Box
BAB VIII	**Strategi Pengujian Perangkat Lunak**	a. Mahasiswa dapat memahami Pendekatan strategis ke pengujian perangkat lunak b. Mahasiswa dapat memahami Pengujian Unit c. Mahasiswa dapat memahami Pengujian Integrasi d. Mahasiswa dapat memahami Pengujian Validasi e. Mahasiswa dapat memahami Pengujian Sistem f. Mahasiswa dapat memahami

		Debugging g. Mahasiswa dapat memahami Quality Assurance

DAFTAR PUSTAKA

a. Pressman, R. S., Software Engineering: A Practitioner's Approach, 8th Edition, McGraw-Hill, 2008
b. Sommerville, I., Software Engineering 8th edition, Addison-Wesley, 2007.
c. Stephen R. Schach: Object-Oriented and Classical Software Engineering, 7th Edition, 2007 Pustaka penunjang :
d. Meyer, B., Object-Oriented Software Construction, 2nd Edition, Prentice-Hall, 1997.
e. Pfleeger, S. L., Software Engineering Theory and Practice, 2nd Edition, Prentice Hall, 2001

	Yulian Findawati, Lahir di Sidoarjo tanggal 25 Juli 1983. Setelah menyelesaikan SLTP, dan SLTA di Sidoarjo, melanjutkan Pendidikan ke ITTELKOM Bandung. Meraih gelar sarjana (ST) pada Prodi Informatika tahun 2007 dan meraih gelar Magister (M.MT) pada jurusan Manajemen Teknologi Informasi ITS. Aktifitas keseharian Sebagai Ketua Prodi Teknik Informatika Universitas Muhammadiyah Sidoarjo Sejak 2010 sampai dengan sekarang. Mata Kuliah yang di ampuh penulis yaitu Rekayasa Perangkat Lunak, Pemrograman Berorientasi Objek, Kecerdasan Buatan , Algoritma Struktur Data dan Teknik Kompilasi.

	Cindy Taurusta, Lahir di Surabaya tanggal 25 April 1990. Setelah menyelesaikan SLTP, dan SLTA di Surabaya, melanjutkan Pendidikan Ahli Madya (D3) kemudian Lanjut Jenjang Sarjana (D4/S1) di Politeknik Elektronika Negeri Surabaya. Meraih gelar sarjana (S.ST) pada Prodi Informatika tahun 2013 dan meraih gelar Magister Teknik (M.T) pada jurusan Teknik Elektro Divisi Jaringan Cerdas Multmedia konsentrasi Game Technology pada Tahun 2015. Aktifitas keseharian Sebagai Staff Kemahasiswaan dan Keuangan Fakultas Teknik Universitas Muhammadiyah Surabaya Sejak Maret 2017 sampai dengan sekarang. Mata Kuliah yang di ampuh penulis yaitu Rekayasa Perangkat Lunak, Pemodelan Game, Pemrograman Berorientasi Objek, Kecerdasan Buatan , Algoritma Struktur Data, Basis Data, Pemrograman Web, Pengembangan Aplikasi Berbasis Web, Interaksi Manusia dan Komputer.

PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH SIDOARJO