

Pertemuan 1

PENGENALAN REKAYASA PERANGKAT LUNAK

Pokok Bahasan dalam RPL :

- ❖ RPL sebagai produk dan sebagai produk
- ❖ Konsep manajemen proyek
- ❖ Proses pembangunan PL dan metrik proyek
- ❖ Perencanaan proyek PL(Perangkat Lunak)
- ❖ Manajemen resiko dlm pelaksanaan proyek
- ❖ Penjadwalan dan penelusuran proyek pembangunan PL
- ❖ Jaminan kualitas PL(Perangkat Lunak)
- ❖ Manajemen konfigurasi PL
- ❖ Rekayasa sistem ke arah CB

Pokok Bahasan dalam RPL (lanjutan)

- ❖ Konsep dan prinsip analisis
- ❖ Pemodelan analisis
- ❖ Konsep dan prinsip desain
- ❖ Metode desain
- ❖ Implementasi pembangunan
- ❖ Teknik pengujian perangkat
- ❖ Strategi perancangan PL
- ❖ CASE tool pembangunan PL

Buku Referensi :

- Pressman, RS., 2008, Software Engineering: A Practitioner's Approach, New York: McGraw-Hill
- Sommerville, I, 2007, Software Engineering, Addison Wesley

Rekayasa Perangkat Lunak

- ☐ Perangkat Lunak? (Software??)
- ☐ Rekayasa Perangkat lunak-RPL?
- ☐ (Software engineering-SE??)
- ☐ Rekayasa sistem-RS? (system engineering-SyE??)
- ☐ Evolusi Perangkat Lunak
- ☐ Computer Science vs RPL
- ☐ RPL vs RS ??
- ☐ Pelaku yang berhubungan dengan Rekayasa Perangkat Lunak
- ☐ Mitos yang ada berkembang
- ☐ Tantangan dalam Pengembangan Perangkat Lunak

Definisi Perangkat Lunak (PL)

- ☐ IEEE-Standar Glossary of Software Engineering Terminology, 1990:

“Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.”

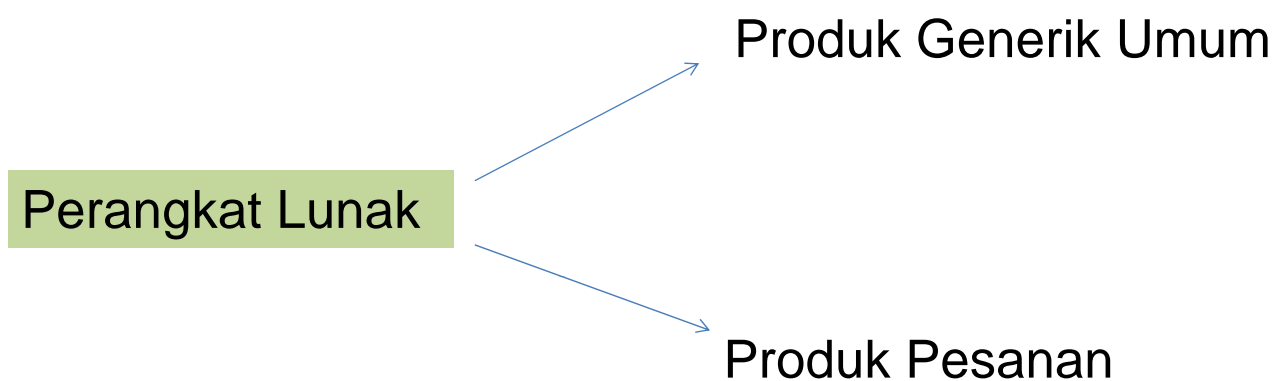
Maksudnya :

Perangkat lunak merupakan kumpulan dari program, prosedur, dan dokumen data lain yang saling berhubungan yang merepresentasikan masalah di dunia nyata yang dikonfigurasi dalam sebuah bentuk aplikasi yang harus dikerjakan komputer

Produk Perangkat Lunak

- Perangkat lunak <> produk perangkat keras
- Produk perangkat lunak dikembangkan (developed) atau direkayasa (engineered).
- Tidak dipabrikkan seperti Perangkat keras.
- Perangkat lunak secara pemakaian tidak pernah AUS layaknya perangkat keras

Produk Perangkat Lunak (2)



- ❑ Perbedaan PENTING antara 2 bentuk perangkat lunak :

Produk Generik	Produk Pesanan
Pada produk generik, organisasi yang mengembangkan perangkat lunak mengontrol spesifikasi perangkat lunak	Pada produk pesanan, spesifikasi biasanya dikembangkan dan dikontrol oleh organisasi yang membeli perangkat lunak tersebut.
diproduksi oleh organisasi pengembang dan dijual ke pasar terbuka → biasa disebut software	dipesan oleh pelanggan tertentu & dikembangkan khusus bagi pelanggan

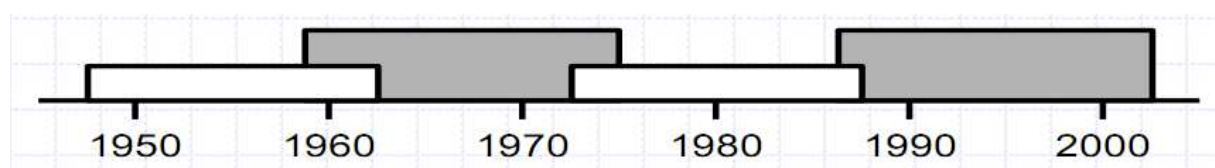
- ❑ Karakteristik perangkat lunak yang baik:
- ❖ Usability
 - ❖ be reliable
 - ❖ maintainability
 - ❖ Efficiency
 - ❖ eye cathcing user interface
 - ❖ long life time
 - ❖ Mempunyai kinerja sesuai fungsi yang dibutuhkan pemakai

Jenis-jenis aplikasi Perangkat Lunak

- ☐ Perangkat Lunak Sistem (System software)
- ☐ Perangkat lunak waktu nyata (Realtime Software)
- ☐ Perangkat Lunak Teknik Dan Ilmu Pengetahuan (Scientific & Engineering Software)
- ☐ Embeded System (yang ditanamkan ke sistem)
- ☐ Perangkat Lunak Pengolah Data (Data Processing)
- ☐ Perangkat Lunak Sistem Informasi (Information System)
- ☐ Perangkat Lunak Sensor
- ☐ Perangkat Lunak Komunikasi (Communicaion Software)
- ☐ Perangkat Lunak Pengolah Grafis
- ☐ Perangkat Lunak Kecerdasan

Evolusi Perangkat Lunak

- ☐ Perangkat lunak pertama kali diciptakan tahun 1945
- ☐ Fokus utama pembuatannya untuk mengembangkan praktik dan teknologi dalam meningkatkan produktivitas para praktisi pengembang PL dan kualitas aplikasi yg dapat digunakan oleh pemakai
- ☐ Evolusi dipicu adanya tuntutan bisnis dan lingkungan kerja yang berkembang sangat dinamis



Evolusi Perangkat Lunak (2)

❖ Era I (1945 – 1960):

- ☐ Munculnya teknologi perangkat keras di tahap awal
- ☐ Penggunaan perangkat lunak yg berorientasi batch
- ☐ Distribusi perangkat lunak masih terbatas
- ☐ Didominasi perangkat lunak model custome
- ☐ Munculnya istilah software engineering (akhir 1950-an/awal 1960-an)
- ☐ Belum didefinisikan secara jelas tentang aspek–software engineering

Evolusi Perangkat Lunak (3)

❖ Era II (1960 – 1970)

- ☐ Disebut era krisis perangkat lunak (software crisis).
- ☐ Penggunaan perangkat lunak sudah meluas
- ☐ Telah hadir perusahaan yang membangun software (software house)
- ☐ Perangkat lunak sdh mengenal multiprogram, multiuser, real-time, dan penggunaan database.

Evolusi Perangkat Lunak (4)

- ❖ Era II (Lanjutan)
 - ❑ Banyak project PL yg gagal
 - ✓ Over budget/anggaran
 - ✓ Berakibat rusak fisik dan kematian
 - ✓ Meledaknya Roket Ariane , kesalahan perintah dlm PL
 - ❑ Dua konferensi ttg software engineering:
 - ✓ Disponsori Komite Sains NATO
 - ✓ Tahun 1968 dan 1969
 - ✓ Profesi resmi bidang software engineering

Evolusi Perangkat Lunak (5)

- ❖ Era III (1975 – 1985)
 - ❑ Pengembangan sistem mengarah ke konsep sistem terdistribusi.
 - ❑ Penerapan sistem embeded intelligence
 - ❑ Harga perangkat keras sudah jauh lebih murah sehingga pemakaian meluas
 - ❑ Pemanfaatan jaringan global dan lokal serta sudah diperkenalkan komunikasi digital

Evolusi Perangkat Lunak (6)

- ❖ Era IV (1985 – 2000)
 - ☐ Kemampuan PC sudah setara dengan komputer mainframe
 - ☐ Penerapan teknologi yang berorientasi pada objek
 - ☐ Implementasi sistem pakar
 - ☐ Jaringan saraf tiruan
 - ☐ Komputasi paralel
 - ☐ Jaringan komputer sudah semakin canggih

Evolusi Perangkat Lunak (7)

- ❖ Era V (2000 – sekarang)
 - ☐ Penggunaan media digital
 - ☐ Media web berkembang pesat
 - ☐ Wireless sudah meluas
 - ☐ Teknologi meluas hingga di mobile computing, mobile programming
 - ☐ Perangkat keras sudah semakin kecil namun powerfull
 - ☐ Dilakukan berbagai penelitian yang menghasilkan model proses/paradigma pengembangan PL utk mengatasi krisis PL

Krisis Perangkat Lunak

❖ Masalah yang muncul:

- Estimasi jadwal dan biaya yang seringkali tidak tepat
- Produktivitas orang-orang software yang tidak dapat mengimbangi permintaan software
- Kualitas software yang kurang baik.
- Kurangnya pengetahuan tentang: Bagaimana mengembangkan software, memelihara software yang ada.
- Bagaimana mengimbangi permintaan software yang makin besar.

Mitos Dalam Perangkat Lunak (Management)

❖ Mitos1:

- ☐ Kita tidak perlu mengubah pendekatan terhadap pengembangan software, karena jenis pemrograman yang kita lakukan sekarang ini sudah kita lakukan 10 tahun yang lalu.
- ☐ Realitasnya : Walau hasil program sama, produktivitas dan kualitas software harus ditingkatkan dengan menggunakan pendekatan software developments

Mitos Dalam Perangkat Lunak (Management) (2)

❖ Mitos2:

- ☐ Kita sudah mempunyai buku yang berisi standarisasi dan prosedur untuk pembentukan software.
- ☐ Realitasnya : Memang buku tersebut ada, tetapi apakah buku tersebut sudah dibaca atau buku tersebut sudah ketinggalan jaman (out of date).

❖ Mitos3:

- ☐ Jika kita tertinggal dari jadwal yang ditetapkan, kita menambah beberapa programmer saja. Konsep ini sering disebut Mongolian harde concept.

Mitos dalam perangkat lunak (Customer)

❖ Mitos1:

- ☐ Pernyataan tujuan umum sudah cukup untuk memulai penulisan program. Penjelasan yang lebih rinci akan menyusul kemudian.
- ☐ Realitasnya : Definisi awal yang buruk adalah penyebab utama kegagalan terhadap usaha-usaha pem-bentukan software. Penjelasan yang formal dan terinci tentang informasi fungsi performance interface, hambatan desain dan kriteria validasi adalah penting. Karakteristik di atas dapat ditentukan hanya setelah adanya komunikasi antara customer dan developer.

Mitos dalam perangkat lunak (Customer)

❖ Mitos2:

- ☐ Kebutuhan proyek yang terus menerus berubah dapat dengan mudah diatasi karena software itu bersifat fleksibel.
- ☐ Realitasnya : memang benar bahwa kebutuhan software berubah, tetapi dampak dari peru-bahan berbeda dari waktu ke waktu.

Mitos Dalam Perangkat Lunak (Praktisi)

❖ Mitos1:

- ☐ Tidak ada metode untuk analisis disain dan testing terhadap suatu pekerjaan, cukup menuju ke depan terminal dan mulai coding.
- ☐ Realitasnya : Metode untuk analisis desain dan testing diperlukan dalam pengembangan software.

❖ Mitos2:

- ☐ Segera setelah software digunakan, pemeliharaan dapat diminimalisasikan dan diatasi dengan cara "CATCH AS CATCH CAM".
- ☐ Realitasnya : Diperlukan budget yang besar dalam maintenance software. Pemeliharaan software harus diorganisir, direncanakan dan dikontrol seolah-olah sebagai suatu proyek besar dalam sebuah organisasi.

Mitos dalam perangkat lunak (Management)

❖ Mitos1:

- ❑ Kebutuhan proyek yang terus menerus berubah dapat dengan mudah diatasi karena software itu bersifat fleksibel.
- ❑ Realitasnya : memang benar bahwa kebutuhan software berubah, tetapi dampak dari peru-bahan berbeda dari waktu ke waktu.

Definisi Rekayasa Perangkat Lunak (RPL)

RPL atau *Software Engineering (SE)* → Disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal spesifikasi sistem sampai pemeliharaan sistem setelah digunakan. Perangkat Lunak yang dibuat harus mampu:

- ✓ Tepat waktu
- ✓ Tepat anggaran
- ✓ Meningkatkan kinerja
- ✓ Mengoperasikan prosedur sistem dengan benar

Perbedaan RPL dengan Computer science

Computer Science	Software Engineering
Computer science lebih memperhatikan teori & metode komputerisasi, sedangkan software engineering menyangkut masalah praktikal pembuatan dan delivery perangkat lunak	Software engineering merupakan bagian dari system engineering, dimana sistem engineering memperhatikan semua aspek pembuatan sistem berbasis komputer termasuk perangkat keras, perangkat lunak & proses

Perbedaan RPL dengan Rekayasa Sistem (RS)?

Rekayasa Sistem	Rekayasa Perangkat Lunak
Rekayasa Sistem (RS) berkaitan dengan semua aspek dalam pembangunan sistem berbasis komputer termasuk hardware, rekayasa PL dan proses.	❖ RPL adalah bagian dari rekayasa sistem yang meliputi pembangunan PL, infrasktruktur, kontrol, aplikasi dan database pada sistem.

Tantangan dalam Rekayasa Perangkat Lunak

- ☐ Tantangan warisan
- ☐ Tantangan heterogenitas
- ☐ Tantangan pengiriman

Pelaku Dalam RPL

- ❖ **Manajer**
(Manajer proyek, Manajer konfigurasi, Manajer penjamin kualitas PL, Manajer bidang lainnya sesuai kebutuhan)
- ❖ **Software Developer**
(Analis sistem, Desainer, Programmer, Inspektur PL, Pengontrol perubahan)
- ❖ **Pendukung**
(Staf administrasi, Humas, Pencatat teknis, Administrator database, Administrator jaringan)

Pertemuan 2

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

POKOK BAHASAN

- ☐ Biaya PL
- ☐ Software Quality Attribute
- ☐ Standar kualitas
- ☐ Takaran Jaminan Kualitas
- ☐ CASE TOOLS
- ☐ Siklus Hidup Perangkat Lunak (SWDLC/Software Development Life Cycle)

BIAYA PERANGKAT LUNAK (SOFTWARE COST)

- ✓ Terkadang mendominasi biaya sistem secara keseluruhan
- ✓ Biaya terbesar untuk perangkat lunak terletak pada proses perawatan (maintenance) dibanding biaya pembuatannya (develop)
- ✓ Biaya pengadaan perangkat lunak yang di pasang pada PC sering lebih besar dibandingkan dengan harga perangkat kerasnya kec. Di negara-negara yang tidak menghargai HAKI
- ✓ Biaya perangkat lunak secara kasar sebesar 60% dari biaya untuk pembangunan dan 40% untuk pengujian
- ✓ Secara umum besarnya biaya bervariasi tergantung pada tipe sistem yang dibangun dan kebutuhan sistem seperti kinerja dan kehandalan sistem
- ✓ Biaya distribusi tergantung pada model pembangunan yang digunakan

SOFTWARE QUALITY ATTRIBUTE (1)

Ciri-ciri kualitas menurut lembaga penjamin mutu PL (ISO, ANSI, IEEE, dll):

- ❖ Correctness (kebenaran)
- ❖ Reliability (tahan uji)
- ❖ User Friendliness
- ❖ Maintainability (mudah dirawat)
- ❖ Portability (mudah di distribusikan)

UKURAN JAMINAN KUALITAS (1)

❖ Ukuran membangun (constructive measures)

- ☐ Aplikasi yg konsisten pada metode di seluruh fase proses pembangunan
- ☐ Penggunaan peralatan/ tools yang memadai
- ☐ Pembangunan PL pd basis kualitas yg tinggi di akhir tahapan
- ☐ Perawatan yang konsisten pada dokumentasi pengembangan

❖ Ukuran analitik (analytical measures)

- ☐ Analisis program yang statis
- ☐ Analisis program yang dinamis
- ☐ Pemilihan test case yang sistematis
- ☐ Pencatatan yang konsisten pada analisis produk

UKURAN JAMINAN KUALITAS (2)

❖ Ukuran Organisasi (Organization Measures)

- ☐ Pengalaman pengembang (developer) dalam mempelajari strategi dan teknik yang tepat dalam membangun PL

KRISIS PERANGKAT LUNAK

- ❖ Masalah yang muncul:
 - ☐ Estimasi jadwal dan biaya yang seringkali tidak tepat
 - ☐ Produktivitas orang-orang software yang tidak dapat mengimbangi permintaan software
 - ☐ Kualitas software yang kurang baik.
- ❖ Kurangnya pengetahuan tentang:
 - ☐ Bagaimana mengembangkan software
 - ☐ Bagaimana memelihara software yang ada, yang berkembang dalam jumlah besar
 - ☐ Bagaimana mengimbangi permintaan software yang makin besar.

KODE ETIK PROFESI

- Konfidensialitas (menghormati klien)
- Tidak boleh menerima pekerjaan di luar
- kompetensinya
- Hak kekayaan intelektual (HaKI)
- Penyalahgunaan komputer, hack, crack,

KODE ETIK INTERNASIONAL

- ❖ Digagas oleh masyarakat profesional di Amerika (1999) yang tergabung dalam ACM/IEEE
- ❖ Makna yang terkandung:
 - ☐ Prinsip-prinsip kesepakatan yang dihubungkan dengan tingkah laku dan keputusan yang dibuat oleh para ahli profesional
 - ☐ Masyarakat profesional: praktisi, pengajar, manajer, supervisor, pengambil kebijakan.

CASE TOOLS

- ❖ CASE (Computer Aided Software Engineering)
 - ☐ Suatu peralatan baik HW maupun SW komputer yang digunakan untuk menyediakan pendukung otomatis dalam aktivitas pembangunan PL.
 - ☐ Tujuan
meningkatkan produktivitas dalam proses pembangunan PL secara signifikan

❖ Dikelompokkan dalam 2 kategori:

1. Upper-CASE

Mendukung aktivitas proses pembangunan tahap awal (tahap analisis kebutuhan dan desain)

2. Lower-CASE

Mendukung aktivitas pembangunan di tahap akhir programming, debugging, dan testing)

CASE TOOLS (2)

Penggunaan CASE tools:

- ☐ Graphical Editors
- ☐ Data Dictionaries
- ☐ GUI Builders
- ☐ Debugger
- ☐ Automated Translators
- ☐ Compiler Integrated
- ☐ Instalator Kit

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

Proses Generik

☐ Spesifikasi

Apa yang harus dilakukan oleh perangkat lunak dan batasan/kendala pengembangannya

☐ Pengembangan

Proses memproduksi sistem perangkat lunak

☐ Validasi

Pengujian perangkat lunak terhadap keinginan pengguna

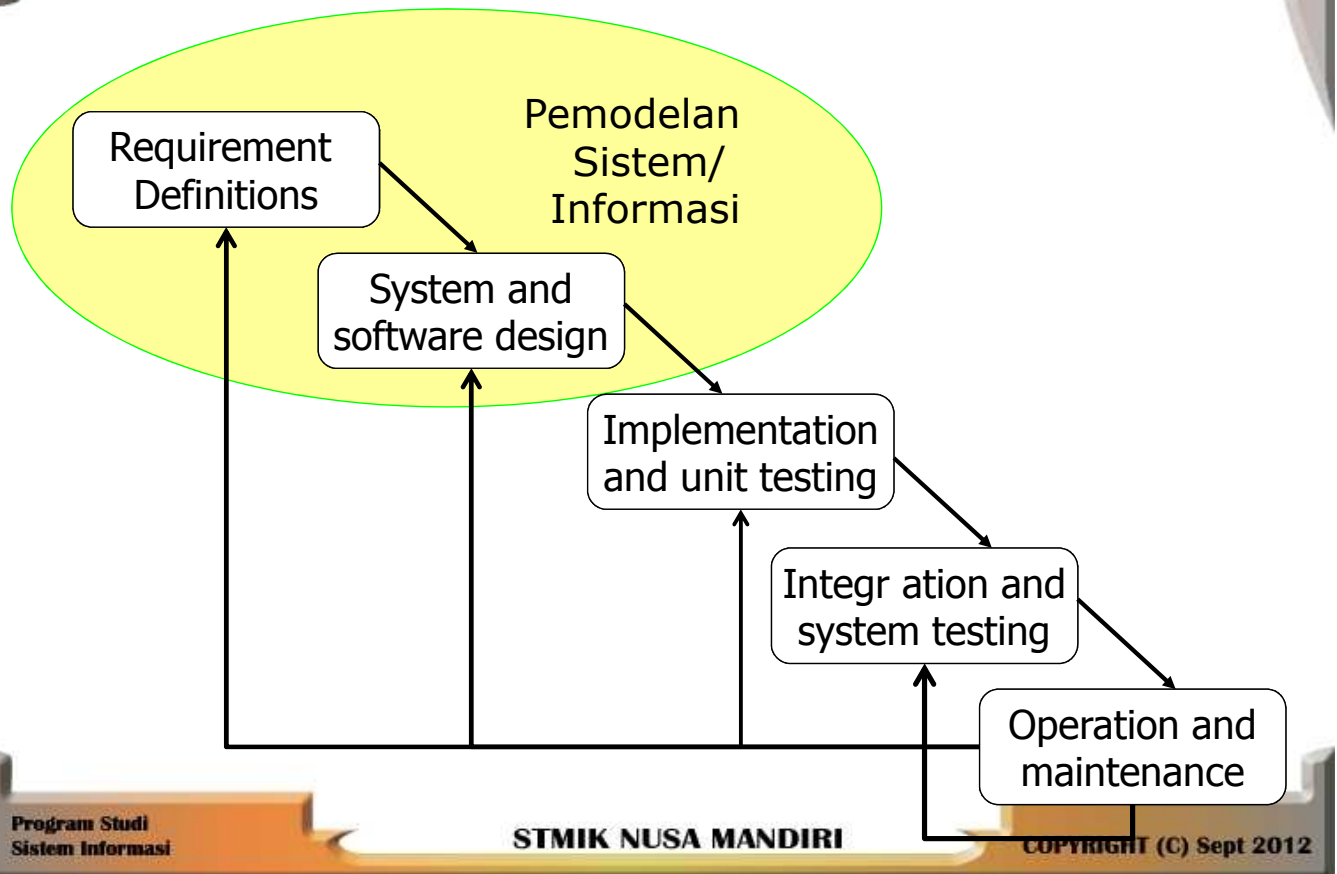
☐ Evolusi

Perubahan perangkat lunak berdasarkan perubahan keinginan.

MODEL PROSES RPL

- ❖ Model Waterfall,
- ❖ Model Prototyping,
- ❖ Model Evolutionary
- ❖ Model Spiral
- ❖ Reuse Based Development

WATERFALL MODEL



WATERFALL MODEL (2)

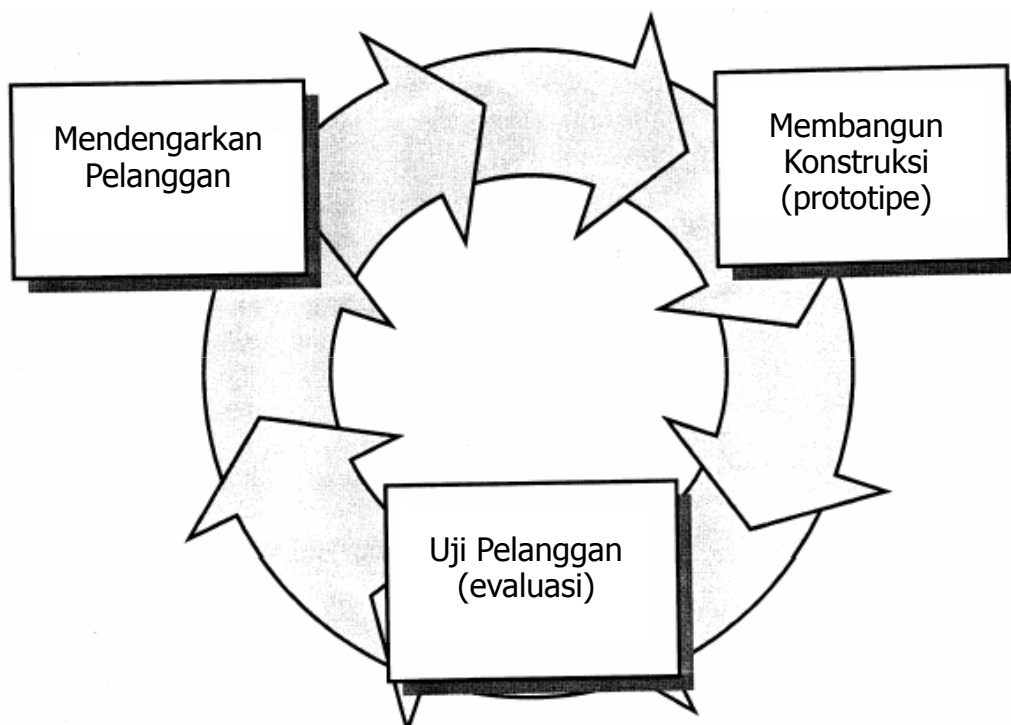
- ❖ Requirements Analysis And Definition
- ❖ System And Software Design
- ❖ Implementation And Unit Testing
- ❖ Integration And System Testing
- ❖ Operation And Maintenance

WATERFALL MODEL (3)

Problems Model Waterfall

1. Jarang sekali proyek yang prosesnya bisa dilakukan secara sequential.
2. Sukar bagi customer untuk secara eksplisit mengemukakan semua kebutuhannya.
3. Customer harus sabar.
4. Developer sering menunda pekerjaan. Anggota tim harus menunggu anggota lainnya
5. menyelesaikan tugasnya.

PROTOTYPE MODEL



PROTOTYPE MODEL (2)

- ☐ Prototype Paradigm dimulai dengan mengumpulkan kebutuhan-kebutuhan customer.
- ☐ Developer dan customer bertemu dan mendefinisikan obyektif software secara menyeluruh, mengidentifikasi kebutuhan-kebutuhan yang diketahui dari area pekerjaan.
- ☐ Setelah itu dibuat Quick Design.
- ☐ Quick Design difokuskan pada representasi aspek software yang bisa dilihat customer/user (misal: format input dan output).
- ☐ Quick Design cenderung ke pembuatan prototipe.
- ☐ Prototipe dievaluasi customer/user dan digunakan untuk menyempurnakan kebutuhan software yang akan dikembangkan.

PROTOTYPE MODEL (2)

- ☐ Sering terjadi customer menjabarkan obyektif umum mengenai software yang diminta, tetapi tidak bisa mendefinisikan input, proses, output yang diminta secara detail.
- ☐ Disisi lain, developer menjadi tidak yakin terhadap efisiensi algoritma, kemampuan adaptasi terhadap sistem operasi, atau bentuk interaksi mesin dengan orang.
- ☐ Untuk mengatasi situasi tersebut, bisa digunakan pendekatan Prototype Paradigm.

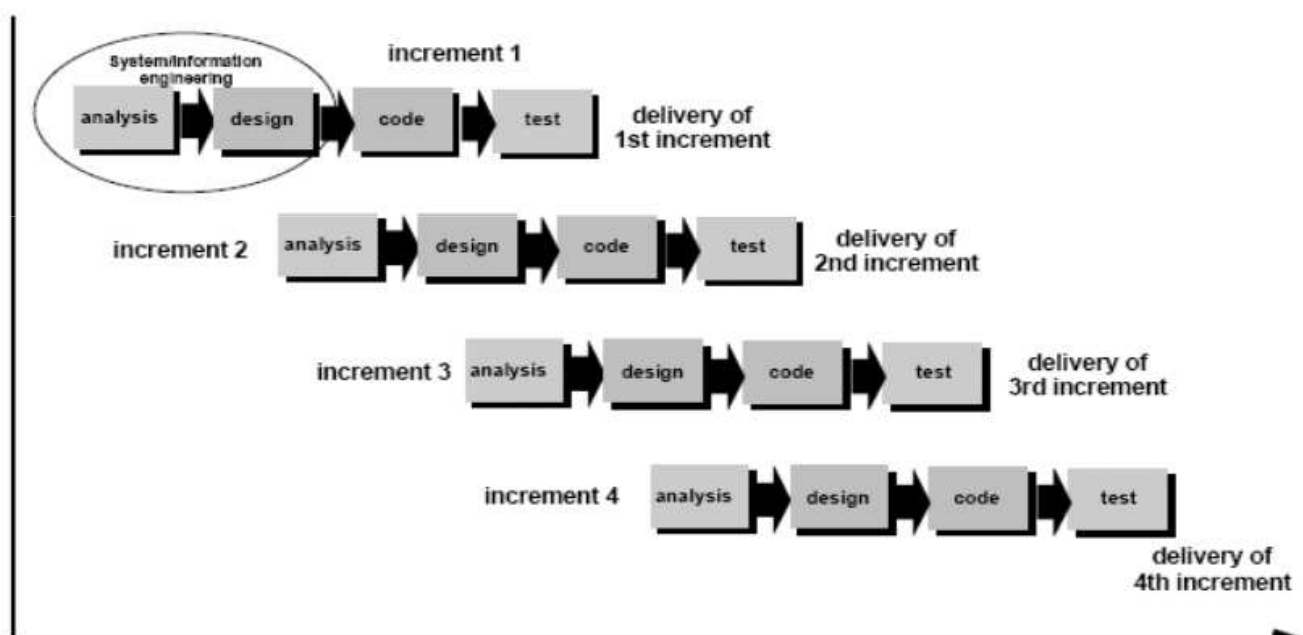
PROTOTYPE MODEL (3)

Problems Prototyping Model

- ❖ Customer melihat prototipe tersebut sebagai versi dari software.
 - ❑ Pada saat produk tersebut harus dibangun ulang supaya level kualitas bisa terjamin,
 - ❑ Customer akan mengeluh dan meminta sedikit perubahan saja supaya prototipe tersebut bisa berjalan.
- ❖ Development membuat implemetasi yang kompromitas dengan tujuan untuk memperoleh prototipe pekerjaan secara cepat.
 - ❑ Dampaknya adalah sistem operasi atau bahasa pemrograman yang dipergunakan tidak tepat, algoritma tidak efisien.

EVOLUTIONARY MODEL

Evolutionary Model (Incremental)



EVOLUTIONARY MODEL INCREMENTAL (2)

Penjelasan :

1. Kombinasikan elemet-element dari waterfall dengan sifat iterasi/perulangan.
2. Element-element dalam waterfall dikerjakan dengan hasil berupa produk dengan
3. Spesifikasi tertentu, kemudian proses dimulai dari fase pertama hingga akhir dan menghasilkan produk dengan spesifikasi yang lebih lengkap dari yang sebelumnya.
4. Demikian seterusnya hingga semua spesifikasi memenuhi kebutuhan yang ditetapkan oleh pengguna.

EVOLUTIONARY MODEL INCREMENTAL (3)

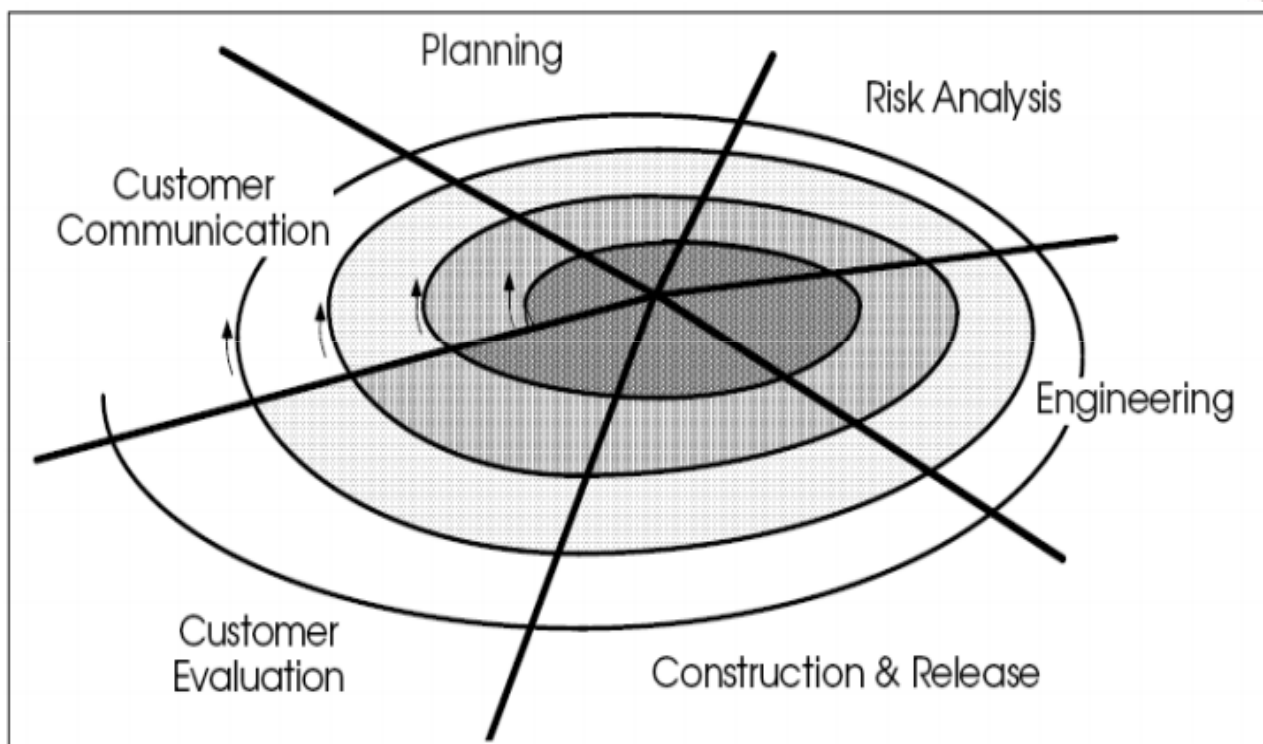
5. Produk hasil increment pertama biasanya produk inti (core product). Produk tersebut digunakan oleh pengguna atau menjalani review/ pengecekan detil. Hasil review tsb menjadi bekal untuk pembangunan pada increment berikutnya, sampai produk yang komplit dihasilkan.
6. Model ini cocok jika jumlah anggota tim pengembang/pembangun PL tidak cukup.
7. Mampu mengakomodasi perubahan secara fleksibel.
8. Produk yang dihasilkan pada increment pertama bukanlah prototype, tapi produk yang sudah bisa berfungsi dengan spesifikasi dasar.

EVOLUTIONARY MODEL INCREMENTAL (4)

Kekurangan Incremental Model:

- ☐ Hanya cocok untuk proyek berukuran kecil (tidak lebih dari 200.000 baris coding)
- ☐ Mungkin terjadi kesulitan untuk memetakan kebutuhan pengguna ke dalam rencana spesifikasi masing-masing hasil increment

EVOLUTIONARY MODEL SPIRAL



EVOLUTIONARY MODEL SPIRAL (2)

Penjelasan :

❖ Customer Comunication

Membangun komunikasi yang baik dengan pelanggan

❖ Planning

Mendefinisikan sumber, batas waktu, informasi-informasi lain seputar proyek

❖ Risk Analyst

Identifikasi resiko management dan teknis

❖ Engineering

Pembangunan contoh-contoh aplikasi misalnya prototype

❖ Construction and release

Pembangunan, test, install dan report

❖ Customer Evaluation

Mendapatkan feedback dari pengguna berdasarkan evaluasi pada fase engineering dan fase instalasi

EVOLUTIONARY MODEL SPIRAL (3)

- ☐ Pada model spiral, resiko sangat dipertimbangkan. Resiko adalah sesuatu yang mungkin mengakibatkan kesalahan.
- ☐ Model spiral merupakan pendekatan yang realistik untuk Perangkat Lunak berskala besar.
- ☐ Pengguna dan pembangun bisa memahami dengan baik software yang dibangun karena setiap kemajuan yang dicapai selama proses dapat diamati dengan baik. Namun demikian, waktu yang cukup panjang mungkin bukan pilihan bagi pengguna, karena waktu yang lama sama dengan biaya yang lebih besar.

REUSE BASED

A. Software Re-engineering

☐ Apakah itu?

Restrukturisasi atau menulis ulang sebagian atau keseluruhan dari sistem yang telah ada tanpa merubah fungsionalitasnya.

☐ Kapan?

Ketika sebagian tetapi tidak semua sub sistem yg besar membutuhkan perawatan yg sering

Ketika HW dan SW sudah lama hampir tak berfungsi

☐ Bagaimana?

Sistem bisa di restrukturisasi dan didokumentasi ulang untuk membuat menjadi mudah dalam perawatan

REUSE BASED (2)

❖ Software Re-engineering (lanjutan)

☐ Mengapa?

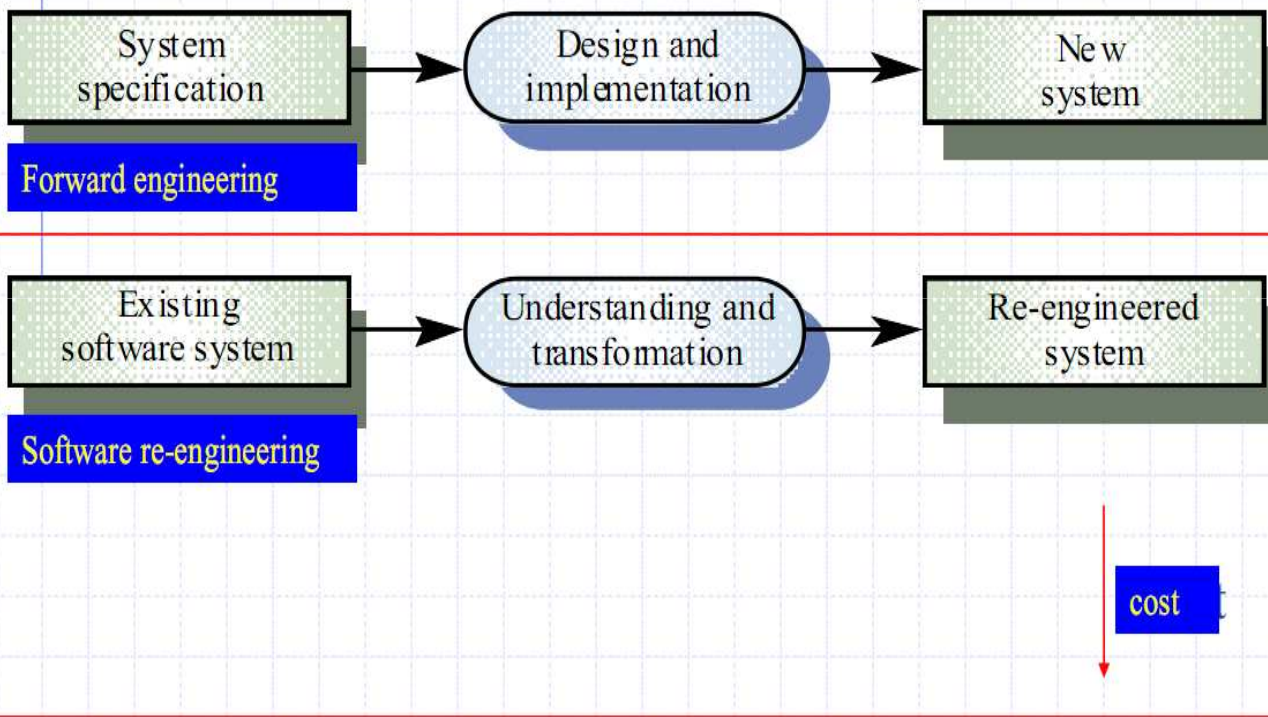
➤ Mengurangi resiko

SW yang baru dibangun membawa resiko yg tinggi

➤ Mengurangi biaya

Biaya untuk re-engineering sering lebih kecil dibanding membangun SW baru.

REUSE BASED (3)



REUSE BASED (3)

B. Reverse Engineering

- ☐ Analisis SW kembali dalam tahap pemahaman dlm desain dan spesifikasinya
- ☐ Bisa sebagian proses re-engineering atau sebagian spesifikasi sistem untuk diimplementasi ulang
- ☐ Membangun database dan bangkitkan program informasi dari proses ini
- ☐ Mengapa?
 - ✓ Kode aslinya telah dalam keterbatasan
 - ✓ Perawatan terbentur pada struktur dan program yang rusak sehingga membutuhkan kerja yg sangat keras
 - ✓ Program secara otomatis distrukturisasi ulang untuk menghilangkan beberapa bagian yang tidak beres dalam kondisi yang sangat kompleks.

Tugas: II (10%)

Diskusi Teori Konsep Dasar Perangkat Lunak hasil dan hasil diskusi dibacakan berkelompok dan dikumpulkan

Pertemuan 3

Manajemen Proyek Perangkat Lunak

Proses Dalam Manajemen PL

Manajemen proyek merupakan lapisan pertama dalam proses rekayasa perangkat lunak skala besar. Untuk menuju pada proyek yang berhasil, perlu dimengerti tentang :

- ❖ Lingkup pekerjaan
- ❖ Resiko yang dapat ditimbulkan
- ❖ Sumber-sumber yang diperlukan
- ❖ Tugas yang harus dilaksanakan
- ❖ Patokan yang harus diikuti
- ❖ Usaha atau biaya yang dikeluarkan
- ❖ Dan Penjadwalan

Langkah Awal dalam Manajemen Perangkat Lunak

Untuk mengestimasi biaya, pembagian tugas, dan penjadwalan, sebelum sebuah proyek direncanakan :

- Memastikan tujuan dan ruang lingkup
- Memperhatikan alternatif-alternatif solusi
- Identifikasi batasan teknik dan manajerial

Fokus Manajemen Proyek

Manajemen proyek terfokus pada 4P, yaitu :

1. **People**
2. **Product** (Perangkat lunak yang dihasilkan)
3. **Process**
4. **Project**

Faktor-faktor yang mempengaruhi hasil akhir proyek Perangkat Lunak

- ☐ Ukuran (*size*)
- ☐ Batas waktu pengiriman (*Delivery Deadline*)
- ☐ Pembiayaan dan anggaran (*Budgets & Costs*)
- ☐ Bidang aplikasi (*Application Domain*)
- ☐ Implementasi Teknologi (*Technology Can Be Implemented*)
- ☐ Batasan-batasan sistem (*System Constrains*)
- ☐ Kebutuhan pengguna (*User Requirements*)
- ☐ Sumber daya yang tersedia (*Available Resource*)

Permasalahan Dalam Manajemen Proyek

- ❖ Bagaimana kualitas produk yang akan dihasilkan
- ❖ Perkiraan / beban resiko yang timbul
- ❖ Ukuran perangkat lunak
- ❖ Estimasi / perkiraan dana
- ❖ Penjadwalan proyek
- ❖ Komunikasi dengan pelanggan
- ❖ Tim perancang
- ❖ Sumber daya lainnya
- ❖ Proses monitoring proyek

Fokus Dalam RPL

- ❖ Analisa Resiko
- ❖ Estimasi Biaya
- ❖ Penjadwalan
- ❖ Manajemen proyek
- ❖ Pengecekan Kualitas hasil terkait dengan kualitas yang diinginkan bersama
- ❖ Manajemen Sumber Daya Manusia

Pengukuran Perangkat Lunak

Pengukuran dan satuan ukuran akan membantu untuk mengerti proses-proses dalam pengembangan dan produk itu sendiri. Proses dan produk diukur usaha untuk meningkatkan kualitasnya.

Pengukuran Perangkat Lunak (2)

1. Pengukuran Langsung

Terkait dengan biaya dan usaha yang diaplikasikan, misalnya yang menyangkut deretan kode program, kecepatan eksekusi, ukuran memori yang dibutuhkan dan cacat pada produk, yang dilaporkan pada sejumlah periode waktu

2. Pengukuran tidak Langsung

Terkait dengan fungsionalitas, kualitas, kompleksitas, efisiensi,abilitas, kemampuan pemeliharaan dan lain-lain

Pengukuran Perangkat Lunak (3)

Mengapa perangkat Lunak Harus Diukur?

1. Untuk mengetahui karakteristik Perangkat Lunak
2. Proses evaluasi Perangkat Lunak
3. Prediksi kebutuhan Perangkat Lunak
4. Pengembangan Perangkat Lunak

Pengukuran Perangkat Lunak (4)

Kualitas Pengukuran Perangkat Lunak :

- ❖ *Correctness*
- ❖ *Maintability*
- ❖ *Integrity*
- ❖ *Usability*

Estimasi

Dalam aktifitas utama proyek yaitu perencanaan, dilakukan:

- ☐ Sumber daya manusia (ukuran orang/bulan)
- ☐ Jangka waktu kronologis (Ukuran waktu kalender)
- ☐ Biaya (Ukuran uang Rp)

Analisis Resiko

- ❖ Analisis resiko merupakan serangkaian langkah untuk menyiasati resiko
- ❖ Analisis resiko sangat penting dalam manajemen proyek perangkat lunak. Beberapa hal yang harus diperhatikan berkaitan dengan resiko adalah: Masa yang akan datang, Perubahan, Pilihan.
- ❖ Menyiasati Resiko
- ❖ Pengendalian Resiko

Tujuan Pengukuran Perangkat Lunak

- ☐ Indikasi kualitas produk
- ☐ Perkiraan produktivitas orang-orang yang menghasilkan produk
- ☐ Perkiraan manfaat dari penerapan metode dan tools
- ☐ Membentuk dasar dari estimasi
- ☐ Menegaskan (Justify) permintaan tools baru dan pelatihan

Ukuran Kualitas Perangkat Lunak

- ☐ Kualitas perangkat lunak dihitung pada saat proses rekayasa perangkat lunak ataupun setelah diserahkan kepada pemakai.
- ☐ Satuan ukuran kualitas perangkat lunak pada saat proses rekayasa :
 1. Kompleksitas program
 2. Modularitas yang efektif
 3. Besarnya program

Penyebab Kegagalan (PL)

Penyebab kegagalan sebuah proyek PL :

- ☐ Batas waktu pengerjaan proyek yang tidak realistis
- ☐ Perubahan keinginan pelanggan
- ☐ Meremehkan pekerjaan
- ☐ Munculnya resiko yang dapat diperkirakan dan resiko yang diluar perkiraan
- ☐ Kesulitan secara teknis
- ☐ Kesalahpahaman antara anggota tim proyek
- ☐ Kesalahan dalam manajemen proyek

Komponen Dalam Proyek PL

- ☐ Manager Senior
- ☐ Manager (Teknis) Proyek
- ☐ Pelaksana
- ☐ Pelanggan
- ☐ Pemakai Akhir (end-user)

Komponen Dalam Proyek PL (1)

Faktor Pertimbangan dalam menyeleksi tim pelaksana proyek :

1. Tingkat kesulitan dari masalah yang akan dikerjakan
2. Ukuran program yang dihasilkan yang terkait dengan jumlah fungsi yang digunakan
3. Waktu yang dibutuhkan oleh tim untuk bekerja secara bersama- sama
4. Tingkatan dimana masalah dapat dimodularisasi / dibuat dalam bentuk modul
5. Kualitas yang diperlukan serta keandalan sistem yang dibangun
6. Kepastian tanggal penyampaian ke pelanggan
7. Memiliki kemampuan sosialisasi (komunikasi) yang dibutuhkan dalam proyek

Definisi Masalah dalam RPL

1. Menetapkan Ruang Lingkup Permasalahan :

- Konteks
- Tujuan Informasi
- Fungsi dan Unjuk Kerja

2. Dekomposisi masalah

Menetapkan pembagian fungsi / aktivitas kerja pada 2 area utama, yaitu ;

- a. Fungsionalitas yang harus disampaikan
- b. Proses yang akan dipakai untuk menyampaikannya

Tugas Pertemuan II:

- Cara pencarian jurnal

Contoh Link Bedah jurnal yang telah disediakan

1. jurnal.ump.ac.id/index.php/juita/article/download/89/83
2. www.journal.uad.ac.id/index.php/JIFO/article/download/329/152
3. ee.uad.ac.id/?download=03-Sistem%20pakar%20THT.pdf
4. <http://journal.uui.ac.id/index.php/Snati/article/viewFile/951/910>
5. http://file.upi.edu/Direktori/FPMIPA/PRODI._ILMU_KOMPUTER/196601011991031-WAWAN_SETIAWAN/12._Optimalisasi_Flash.pdf
6. <http://www.scribd.com/doc/66909360/Jurnal-IT-Vol-1-STMIK-Handayani>

Tugas Lanjutan

Bobot 15 %

Buatlah

1. Kelompok yang terdiri dari 3- 4 orang
2. Kelompok tersebut membahas Tugas Akhir berupa program yang pernah dibuat oleh salah satu anggota kelompok dan carilah jurnal yang sesuai dengan program tersebut
3. Buatlah rencana pengembangan dari program yang telah dipilih tersebut

1. Sebutkanlah metode perencanaan anggaran!
2. Sebutkanlah hal-hal yang berkaitan dengan metode ABC!
3. Sebutkanlah cara untuk menyiasati resiko!
4. Sebutkanlah strategi pengendalian resiko!
5. Apakah yang dimaksud Segitiga proyek (Proyek Triangle)

ARTIFACT UML

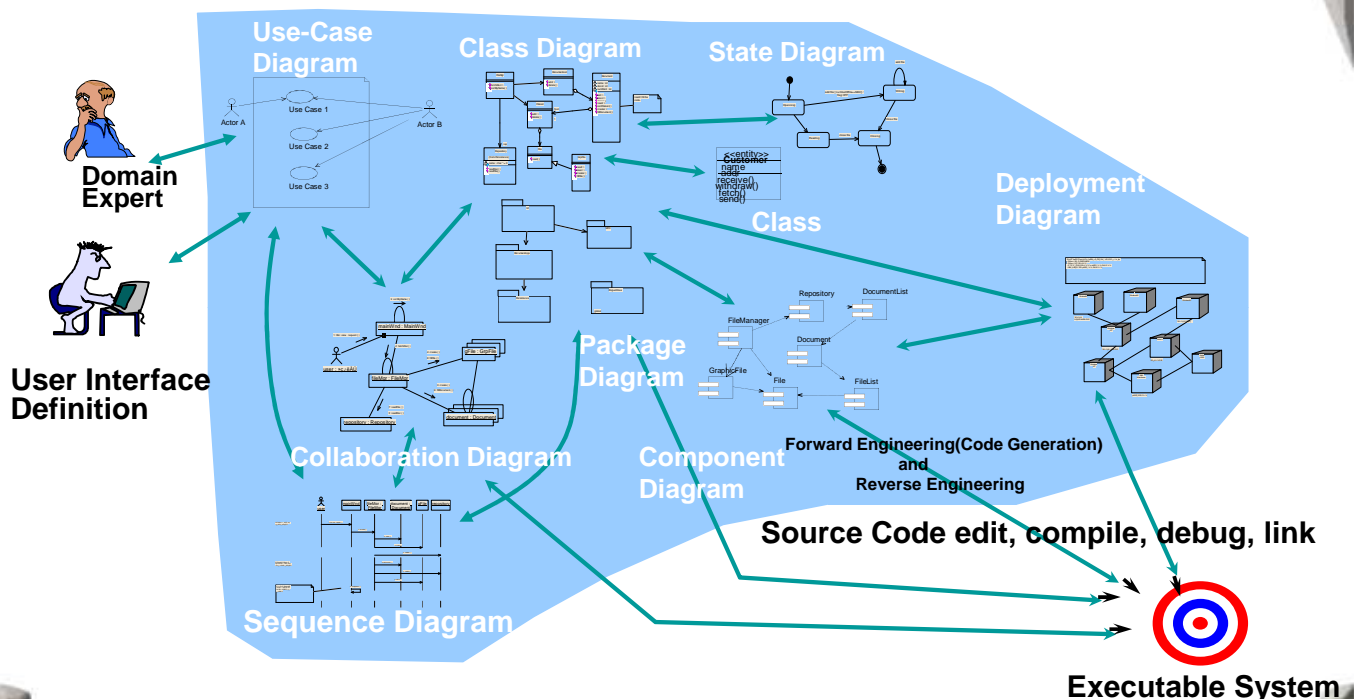
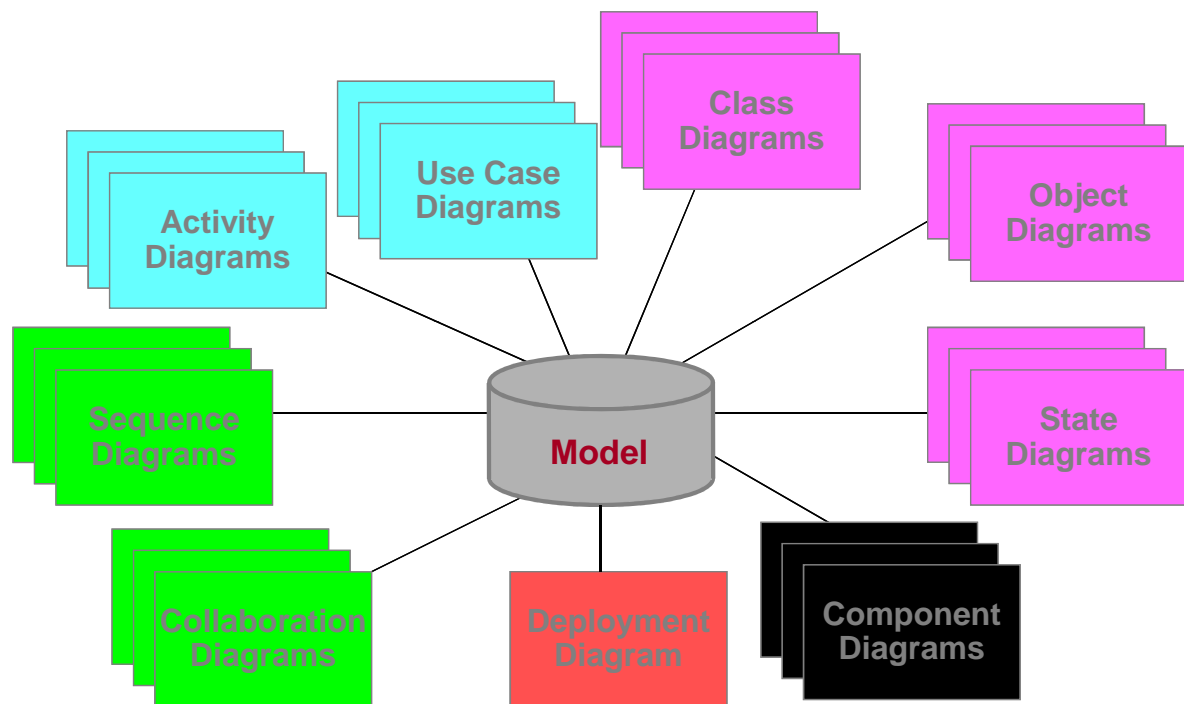


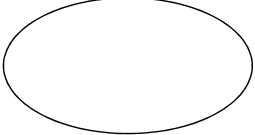
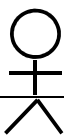
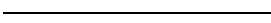
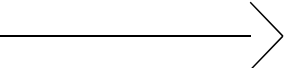
DIAGRAM-DIAGRAM DI UML



USE CASE DIAGRAM

- Menggambarkan fungsionalitas yang diharapkan dari sebuah sistem.
- Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”.
- Menggambarkan kebutuhan sistem dari sudut pandang user
- Mengfokuskan pada proses komputerisasi (automated processes)
- Menggambarkan hubungan antara use case dan actor

- Use case menggambarkan proses system (kebutuhan sistem dari sudut pandang user)
- Secara umum use case adalah:
 - Pola perilaku sistem
 - Urutan transaksi yang berhubungan yang dilakukan oleh satu actor
- Use case diagram terdiri dari
 - a. Use case
 - b. Actors
 - c. Relationship
 - d. System boundary boxes (optional)
 - e. Packages (optional)

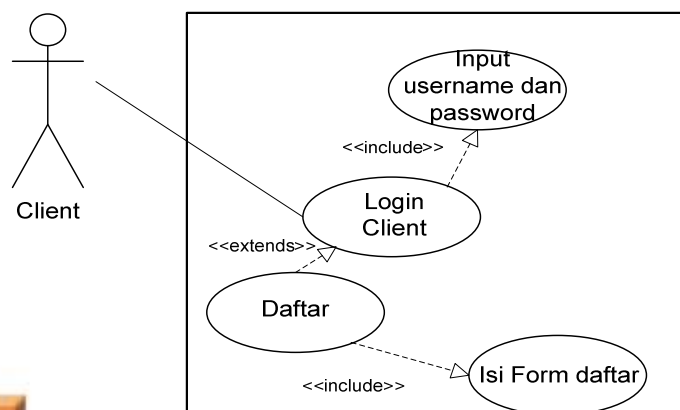
Use case	Use case dibuat berdasar keperluan actor, merupakan “apa” yang dikerjakan system, bukan “bagaimana” system mengerjakannya Use case biasanya menggunakan kata kerja	
Actor	Tidak boleh ada komunikasi langsung antar actor . Actor menggambarkan sebuah tugas/peran dan bukannya posisi sebuah jabatan Actor memberi input atau menerima informasi dari system. Actor biasanya menggunakan Kata benda	
Assosiation Garis tanpa panah	Ujung panah pada association antara actor dan use case mengindikasikan siapa/apa yang meminta interaksi dan bukannya mengindikasikan aliran data Sebaiknya gunakan Garis tanpa panah untuk association antara actor dan use case	
Assosiation Panah Teerbuka	association antara actor dan use case yang menggunakan panah terbuka untuk mengindikasikan bila actor berinteraksi secara pasif dengan system anda	

Association

- *Associations* bukan menggambarkan aliran data/informasi
- *Associations* digunakan untuk menggambarkan bagaimana actor terlibat dalam use case
- Ada 4 jenis relasi yang bisa timbul pada use case diagram
 1. Association antara actor dan use case
 2. Association antara use case
 3. Generalization/Inheritance antara use case
 4. Generalization/Inheritance antara actors

Association antara use case

- `<<include>>` termasuk didalam use case lain (required) / (diharuskan)
 - Pemanggilan use case oleh use case lain, contohnya adalah pemanggilan sebuah fungsi program
 - Tanda panah terbuka harus terarah ke sub use case
 - Gambarkan association include secara horizontal

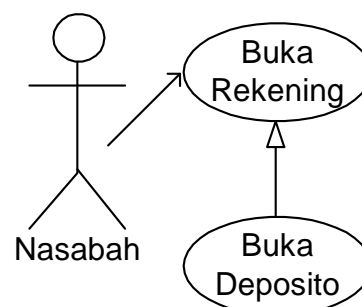


Association antara use case (Lanjut)

- <<extend>> perluasan dari use case lain jika kondisi atau syarat terpenuhi
 - Kurangi penggunaan association Extend ini, terlalu banyak pemakaian association ini membuat diagram sulit dipahami.
 - Tanda panah terbuka harus terarah ke parent/base use case
 - Gambarkan association extend secara vertical

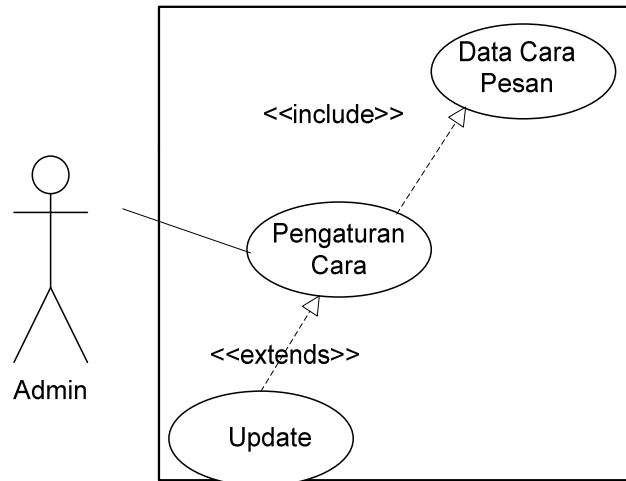
Generalization/inheritance antara use case

- Generalization/inheritance digambarkan dengan sebuah garis berpanah tertutup pada salah satu ujungnya yang menunjukkan lebih umum
- Gambarkan generalization/inheritance antara use case secara vertical dengan inheriting use case dibawah base/parent use case
- Generalization/inheritance dipakai ketika ada sebuah keadaan yang lain sendiri/perlakuan khusus (*single condition*)



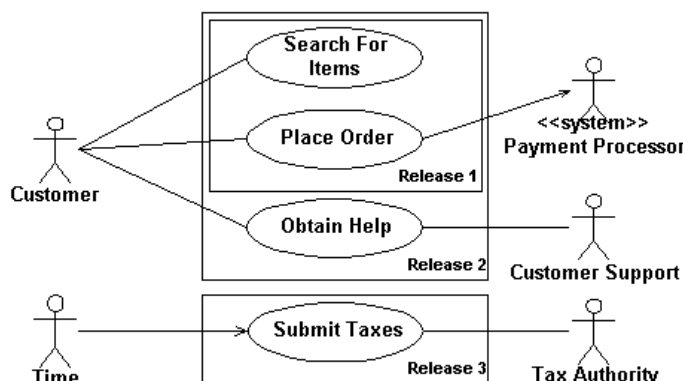
Generalization/inheritance antara actor

- Gambarkan generalization/inheritance antara actors secara vertical dengan inheriting actor dibawah base/parent use case



Use case System boundary boxes

- Digambarkan dengan kotak disekitar use case, untuk menggambarkan jangkauan system anda (scope of of your system).
- Biasanya digunakan apabila memberikan beberapa alternative system yang dapat dijadikan pilihan
- System boundary boxes dalam penggunaannya optional



CLASS DIAGRAM

CLASS DIAGRAM

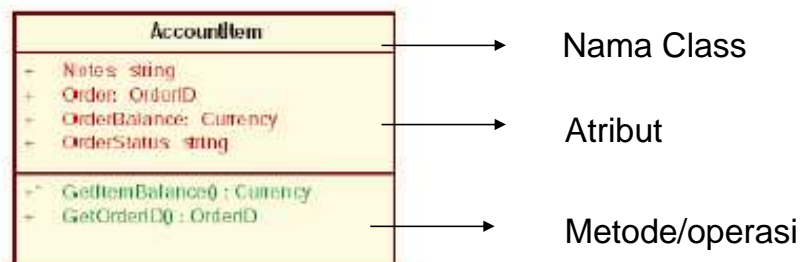
- Adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek.
- *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).
- *Class diagram* menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

Class memiliki tiga area pokok :

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

CLASS DIAGRAM (LANJUTAN)

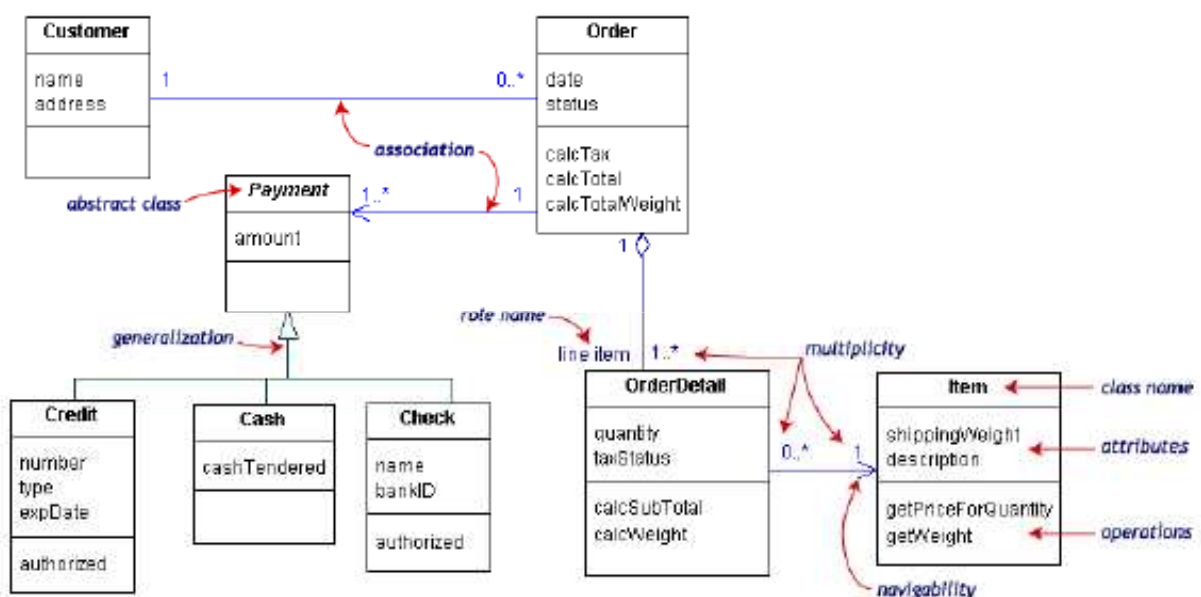
- Atribut dan metoda dapat memiliki salah satu sifat berikut :
 - *Private*
 - *Protected*
 - *Public*
 - *Package*



HUBUNGAN ANTAR CLASS

1. Asosiasi, yaitu hubungan statis antar *class*.
2. Agregasi, yaitu hubungan yang menyatakan bagian ("terdiri atas..").
3. Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru.
4. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-*passing* dari satu *class* kepada *class* lain.

CONTOH CLASS DIAGRAM



MULTIPLICITY

- Unspecified
- Exactly one
- Zero or more (many, unlimited)
- One or more
- Zero or one (optional scalar role)
- Specified range
- Multiple, disjoint ranges

1

0..*

*

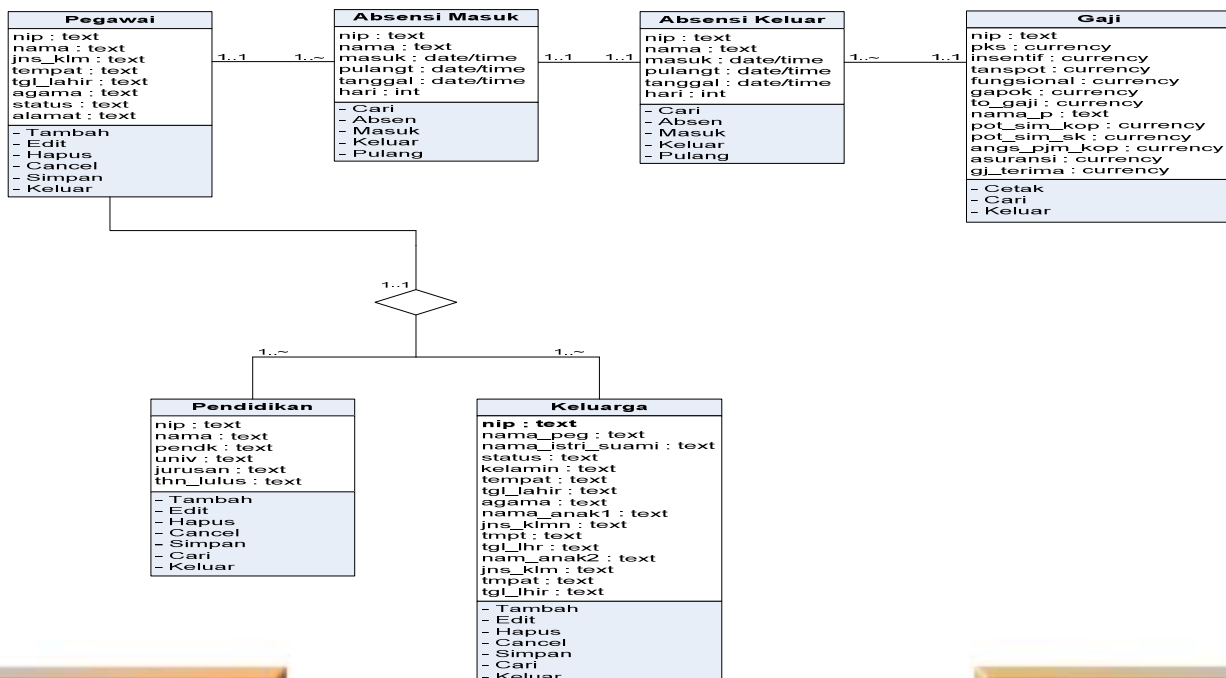
1..*

0..1

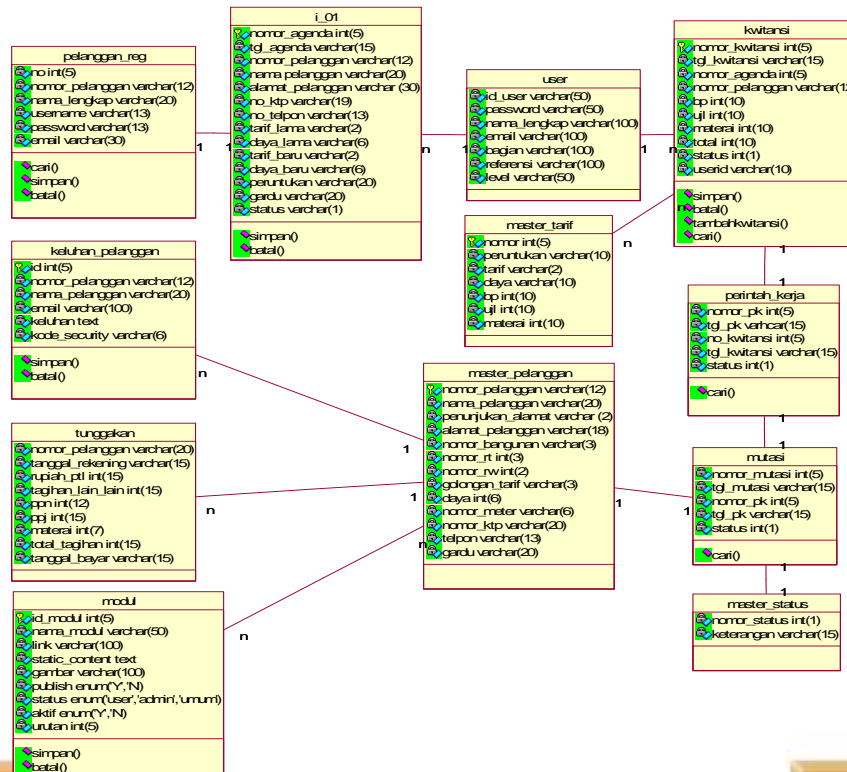
2..4

2, 4..6

Class Diagram diperoleh berdasarkan dari database
Contoh Kasus (Acknowledgments Evi Lutfi Muktar)




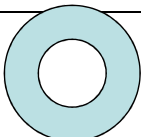


Contoh Kasus (Acknowledgments Toeko triyanto)



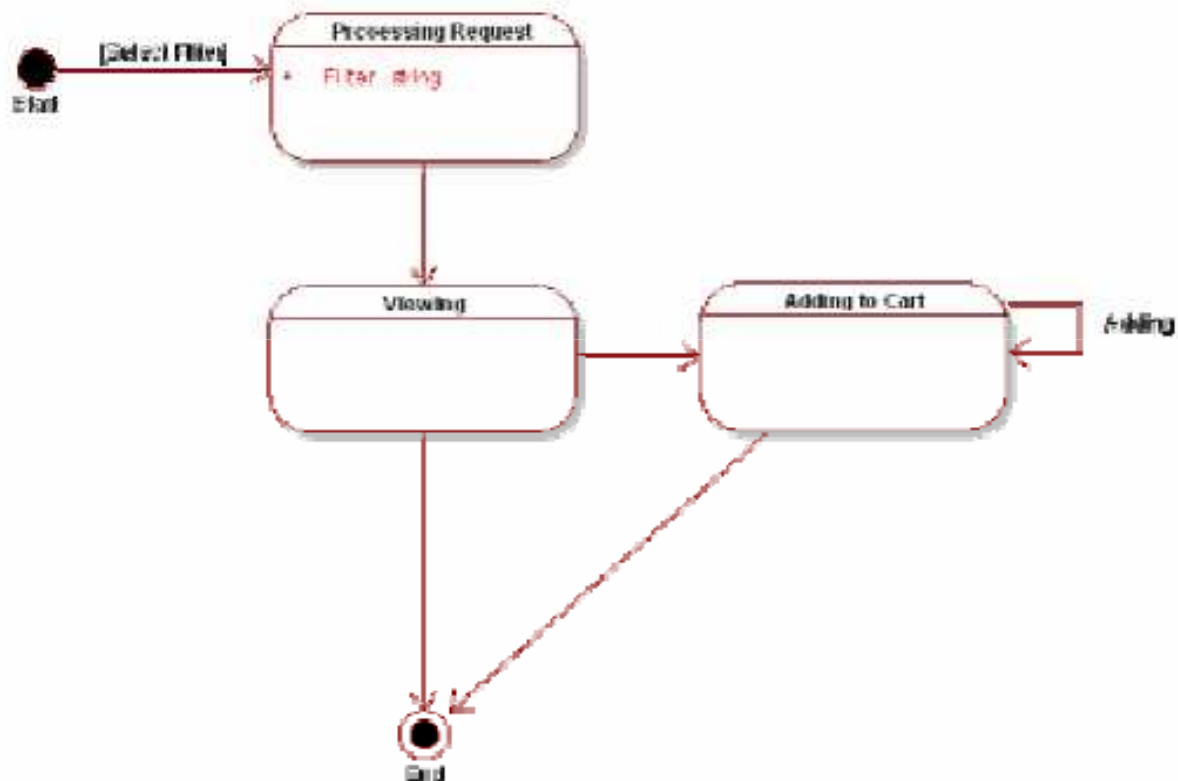
Statechart Diagram.

- *Statechart diagram/ state diagram* digunakan untuk mendokumentasikan beragam kondisi/keadaan yang bisa terjadi terhadap sebuah *class* dan kegiatan apa saja yang dapat merubah kondisi/keadaan tersebut.

State	Notasi State menggambarkan kondisi sebuah entitas, dan digambarkan dengan segiempat yang pinggirnya tumpul dengan nama state didalamnya	
Transition	Sebuah Transition menggambarkan sebuah perubahan kondisi objek yang disebabkan oleh sebuah event. Transition digambarkan dengan sebuah anak panah dengan nama event yang ditulis di atasnya, dibawahnya atau sepanjang anak panah tersebut.	
Initial State	sebuah kondisi awal sebuah object sebelum ada perubahan keadaan. Initial State digambarkan dengan sebuah lingkaran solid. Hanya satu Initial State yang diizinkan dalam sebuah diagram	
Final State	menggambarkan ketika objek berhenti memberi respon terhadap sebuah event. Final State digambarkan dengan lingkaran solid didalam sebuah lingkaran kosong.	

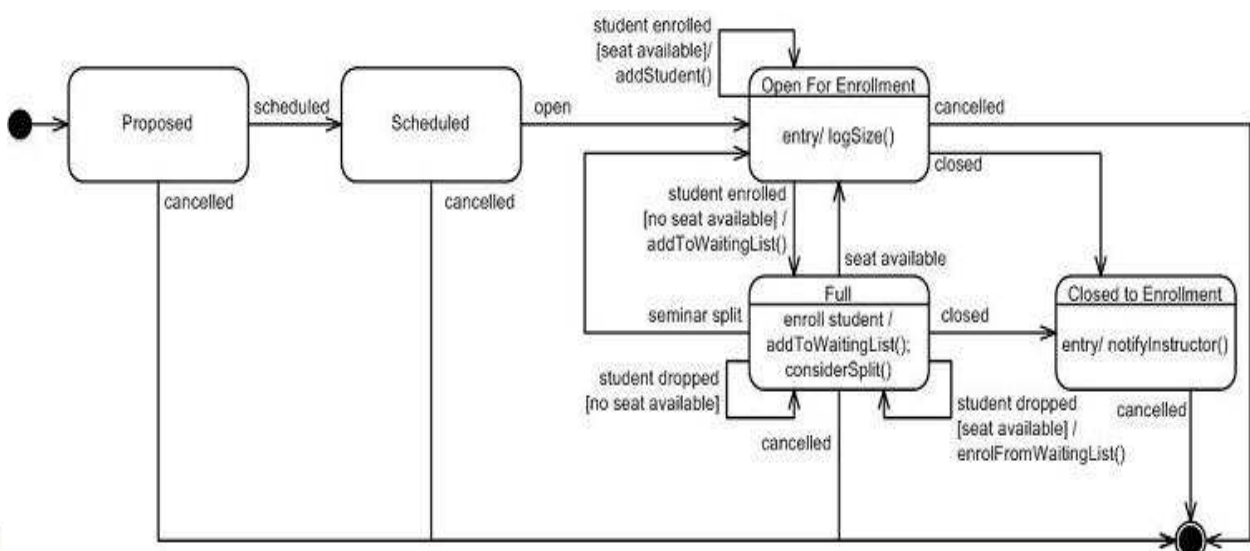
- *Statechart diagram* menggambarkan transisi dan perubahan keadaan (dari satu *state* ke *state* lainnya)
- Pada umumnya *statechart diagram* menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu *statechart diagram*).
- *State* digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya
- Transisi antar *state* umumnya memiliki kondisi *guard* yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku.
- *Action* yang dilakukan sebagai akibat dari *event* tertentu dituliskan dengan diawali garis miring.
- Titik awal dan akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah.

Contoh State Diagram



State Machine Diagram (Statechart diagram in versi 1.x)

- Untuk memodelkan behavior/methode (lifecycle) sebuah kelas atau object
- Menunjukkan urutan kejadian sesaat (state) yang dilalui sebuah object, transisi dari sebuah state ke state lainnya



State Machine Diagram (Statechart diagram in versi 1.x)

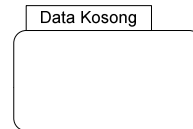
Sebuah state machine diagram mempunyai :

- state (kejadian sesaat) are represented by the values of attributes of an object

- State digambarkan dengan bentuk Data Kosong

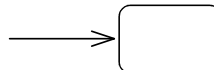


atau



- “Black Hole” states

is state has transitions into it but none out



- Miracle states

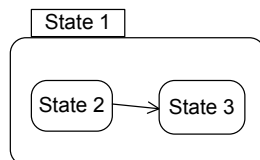
is state has transitions out of it but none into it



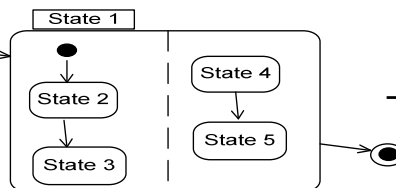
State Machine Diagram (Statechart diagram in versi 1.x)

- Composite State

- Kumpulan dari beberapa states yang setidaknya dalam sebuah region



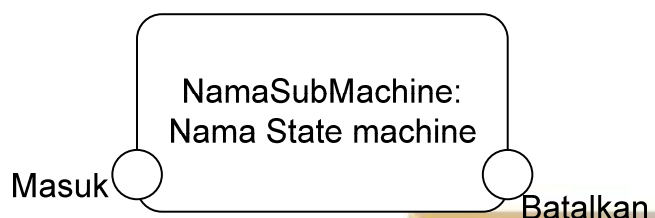
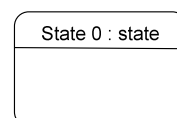
- Orthogonal State, jenis composite state lebih dari 1 region



- Digunakan untuk mendukung konsep encapsulation
- Sebuah state tidak boleh mempunyai region dan submachine secara bersamaan
- Nama state mempunyai sintaks :
nama submachine state :
referenced state machine

- Submachine State

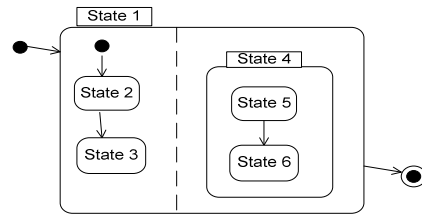
- Sejenis composite state yang isinya didefinisikan oleh state machine lain
- State Machine yang berisi submachine state disebut “Containing state machine”
- Sebuah state yang dihubungkan ke state machine lainnya
- Dihubungkan ke satu/lebih entry point dan satu/lebih exit point



State Machine Diagram (Statechart diagram in versi 1.x)

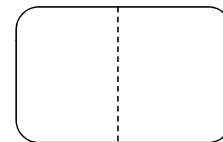
Sub States

- Sebuah state yang ada dalam sebuah region
 - Direct Substate, Sub state yang tidak berisi state lain
 - Indirect Substate, Sub state yang berisi state lain



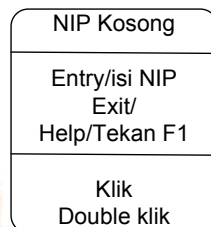
Region (kelompok state)

- Dipisahkan dengan garis terputus, yang setiap region boleh mempunyai nama sebagai optional
- Sebuah state tidak boleh mempunyai region dan submachine secara bersamaan



State terpisah menjadi 3 bagian yaitu

- Activity label bisa berupa Entry, Exit atau do
- Dimana Activity expression adalah penggunaan atribut



Nama State

Internal Activity, kegiatan yang dilakukan dalam state
sintaks : Activity label/activity expression

Internal transition

State Machine Diagram (Statechart diagram in versi 1.x)

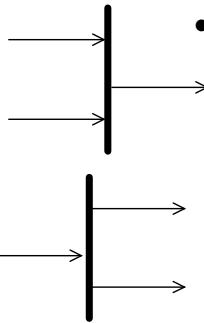
label on transition is in the format

event [guard][/]methode list()

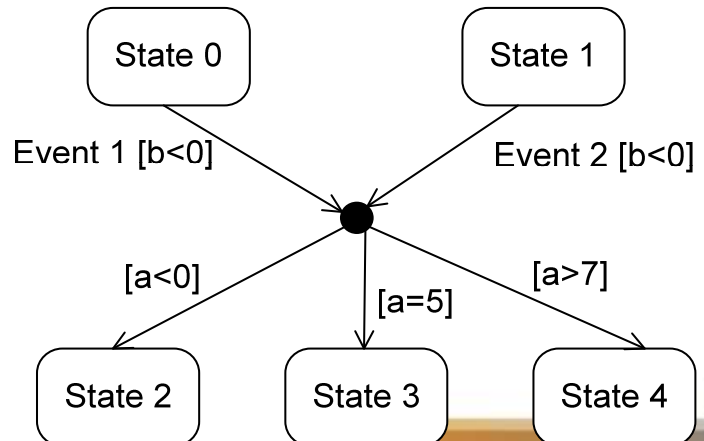
- event biasa dituliskan dengan past tense
- event menyebabkan sebuah object berpindah dari satu state ke state lain
- Guard, condition that must be true for the transition to be triggered
- Guard harus konsisten dan tidak overlap
Contoh: $X < 0$, $X = 0$ dan $X > 0$ konsisten
 $X \leq 0$ dan $X \geq 0$ tidak konsisten
- Guards harus lengkap logikanya
Contoh: $X < 0$ dan $X > 0$, bagaimana jika $X = 0$?
- Methode dijalankan
 - ketika object memasuki state diindikasikan dengan methode bernama entry()
 - ketika object keluar state diindikasikan dengan methode bernama exit()
- Methode menyebabkan perubahan di sebuah state bisa juga tidak

State Machine Diagram (Statechart diagram in versi 1.x)

- Join, menggabungkan beberapa transition menjadi sebuah transition
- Fork, memecah sebuah transition menjadi beberapa transition yang berkondisi AND (transition harus dilewati semuanya).
- Junction, Menggabungkan sebuah/beberapa transition dan memecahnya menjadi sebuah/beberapa transition yang berkondisi AND (transition harus dilewati semuanya). Digunakan tanda lingkaran hitam kecil

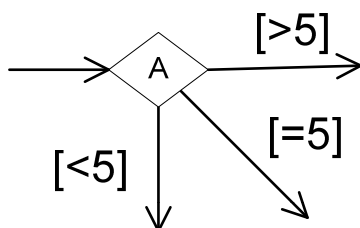


- Dimungkinkan transition ke sebuah state yang berisi beberapa state yang disebut state list

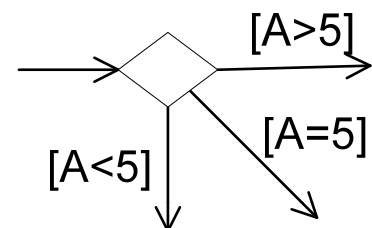


- Choice, Mengkondisikan sebuah transition menjadi sebuah/beberapa transition, yang hanya dipilih salah satu transition(choice).
 - Digunakan lambang diamond
 - Operand dapat diletakkan didalam diamond atau pada transition

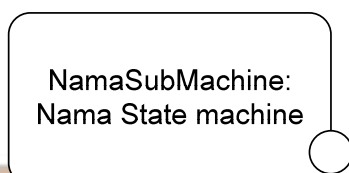
Contoh:



atau

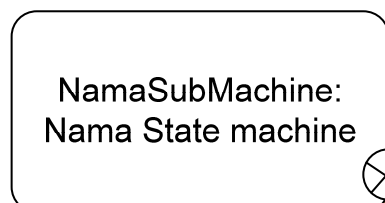


- Entry point
Dilambangkan sebuah lingkaran kecil yang ditaruh pada pinggir state(bisa juga didalam atau diluar), dan berguna sebagai submachine state



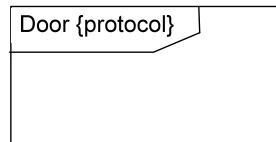
lagi

- Exit point
Dilambangkan sebuah lingkaran kecil bersilang yang ditaruh pada pinggir state (bisa juga didalam atau diluar), dan berguna sebagai submachine state



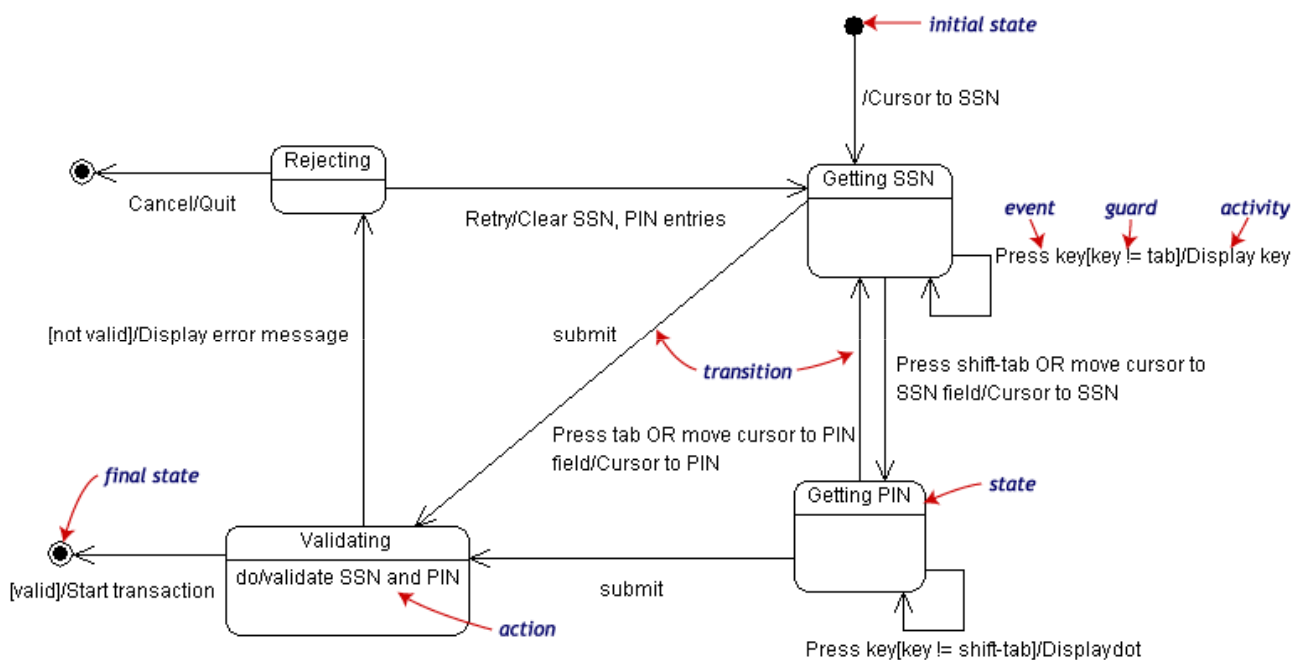
batalan

- State Machine Diagram ada 2 jenis
 - Behavioral State Machines
 - Protocol State Machines
- Tidak adanya internal activity seperti entry, exit, do
- Transition pada Protocol State Machines harus menggunakan Protocol Transition



- Protocol Transition
 - Sintaks : [pre condition] event / [post condition]
 - precondition atau postcondition adalah guard (Guard is condition that must be true for the transition to be triggered)
 - Precondition, kondisi sebelum transition
 - Postcondition, kondisi setelah transition

Contoh State Diagram



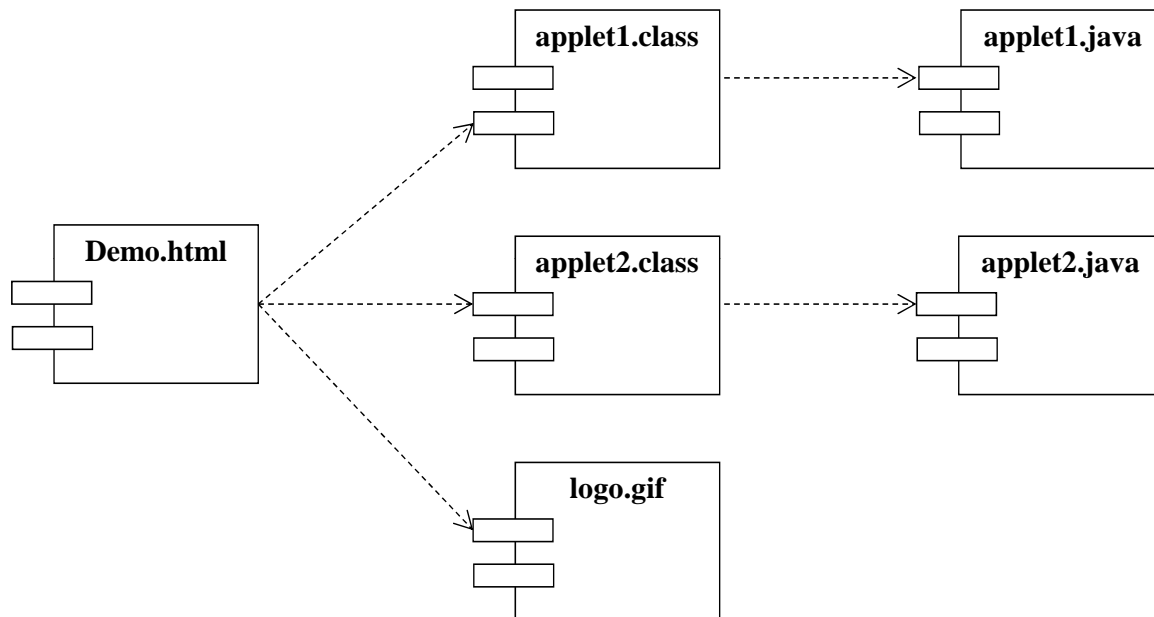
Deployment Diagram

- **Deployment/physical diagram** menggambarkan detail bagaimana komponen di-*deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, server atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisikal
- **Node** adalah server, *workstation*, atau piranti keras lain yang digunakan untuk men-*deploy* komponen dalam lingkungan sebenarnya. Hubungan antar *node* (misalnya TCP/IP) dan *requirement* dapat juga didefinisikan dalam diagram ini.

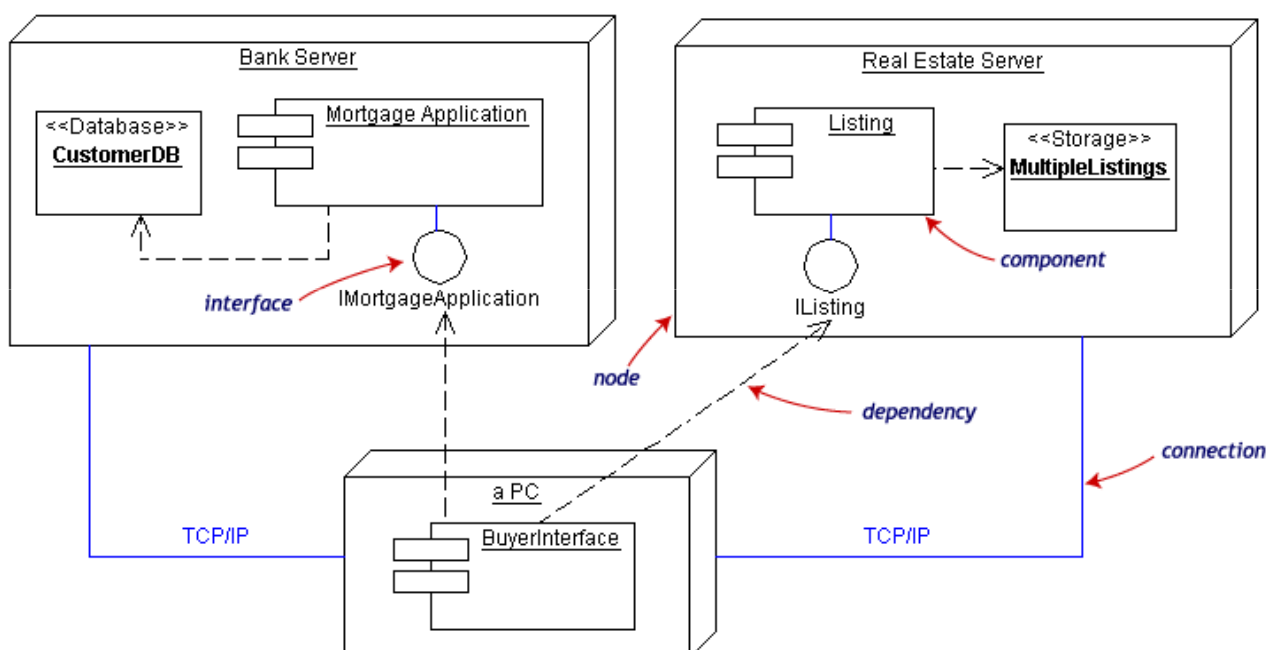
Component Diagram

- **Component diagram** menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) di antaranya.
- Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*, baik yang muncul pada *compile time*, *link time*, maupun *run time*.
- Pada umumnya komponen terbentuk dari beberapa *class* dan/atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil.
- Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.

Contoh : Component Diagram



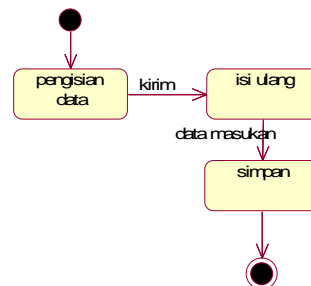
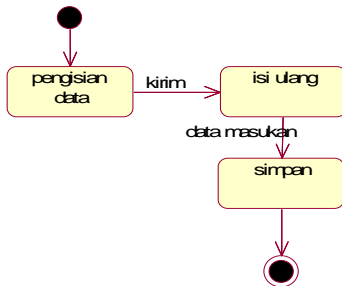
Contoh : Component & Deployment Diagram



Contoh kasus (Acknowledgments Toeko triyanto)

state chart diagram pendaftaran

statechart diagram pengisian data kwitansi.



Bobot 30% (design)

Tugas : - Berdasarkan tugas pada pertemuan sebelumnya
(Pengembangan dari program yang pernah dibuat)

Buatlah design UML dari sistem usulan dengan
apakah itu berupa program desktop, web, animasi
atau sistem pakar (pilih salah satu)

- (untuk pertemuan 4, 5 dan 6) buatlah
rancangannya dengan menggunakan Tools,
misalnya : Enterprise Architect , Rational Rose,
Argo UML, Visual Paradigm dan lain-lain, sesuai
dengan Diagram yang telah dipelajari diatas.

-Dikumpulkan berupa laporan




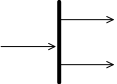
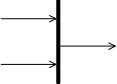

Pertemuan 5

ACTIVITY DIAGRAM

ACTIVITY DIAGRAM

- Menggambarkan proses bisnis dan urutan aktivitas dalam sebuah proses
- Dipakai pada business modeling untuk memperlihatkan urutan aktifitas proses bisnis
- Struktur diagram ini mirip flowchart atau Data Flow Diagram pada perancangan terstruktur
- Sangat bermanfaat apabila kita membuat diagram ini terlebih dahulu dalam memodelkan sebuah proses untuk membantu memahami proses secara keseluruhan
- Activity diagram dibuat berdasarkan sebuah atau beberapa use case pada use case diagram

Simbol Activity Diagram

Simbol	Keterangan
	Start Point
	End Point
	Activities
	Fork (Percabangan)
	Join (Penggabungan)
	Decision
Swimlane	Sebuah cara untuk mengelompokkan activity berdasarkan Actor (mengelompokkan activity dalam sebuah urutan yang sama)

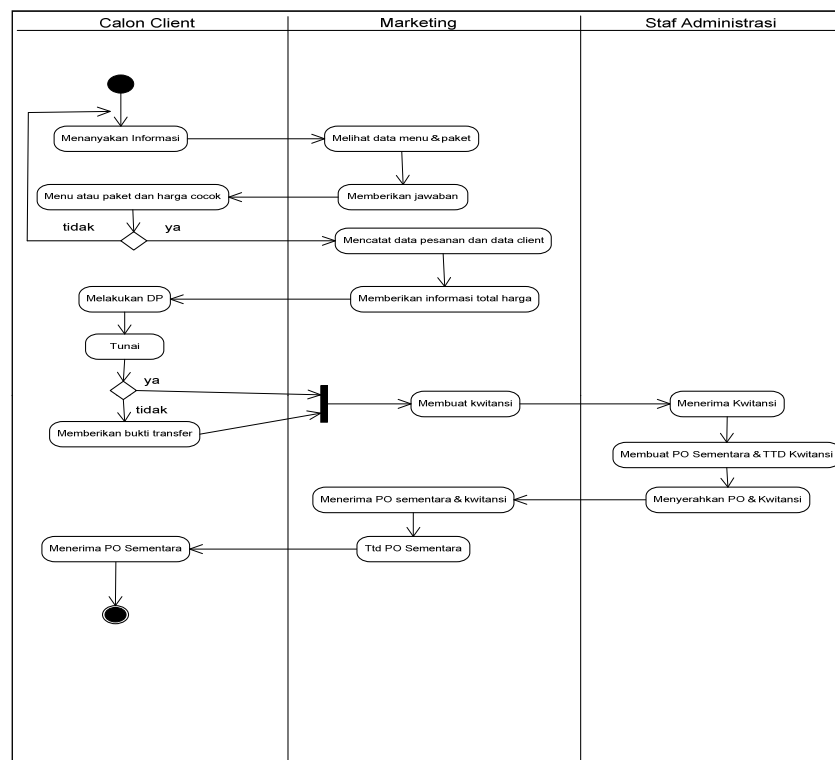
Sumber : Rational rose
Program Studi
Sistem Informasi

STMIK NUSA MANDIRI

COPYRIGHT (C) Sept 2012

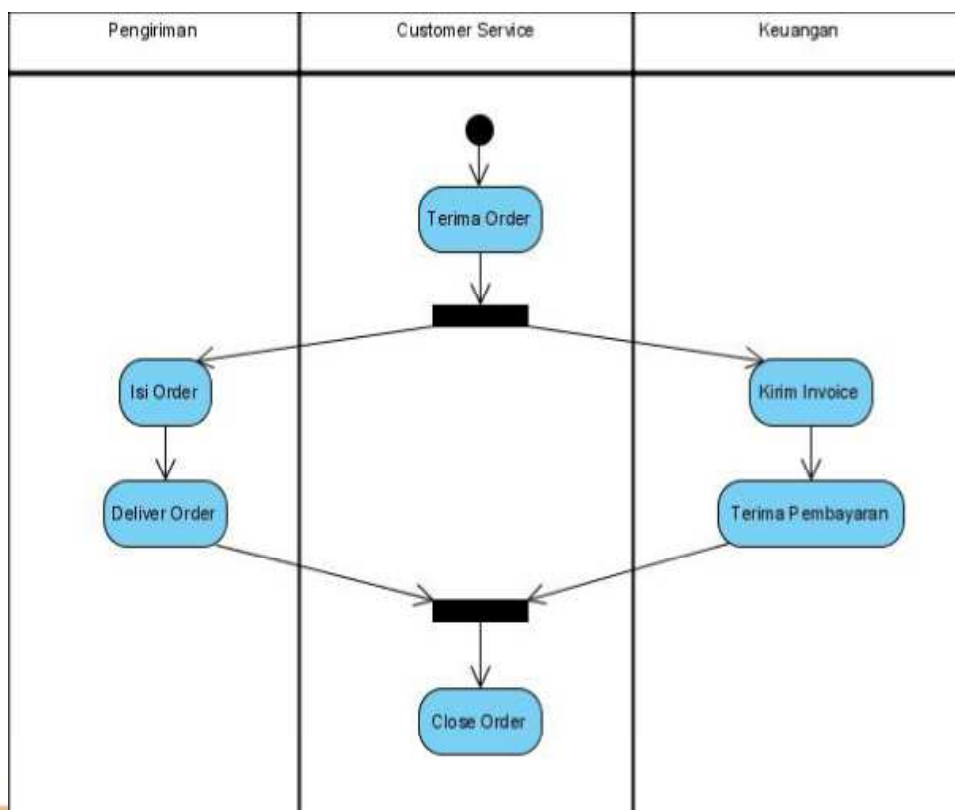
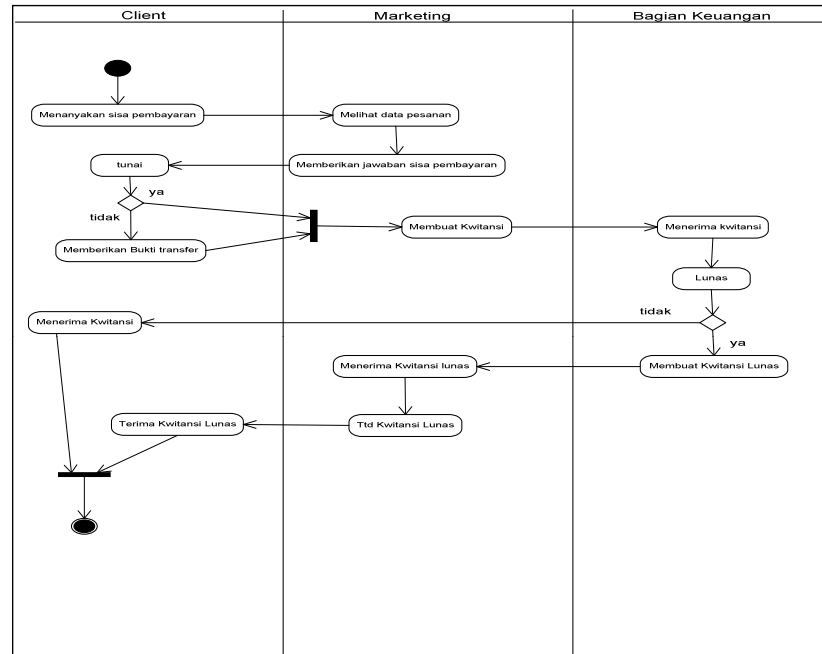
CONTOH ACTIVITY DIAGRAM

Activity Diagram Pembuatan PO Sementara

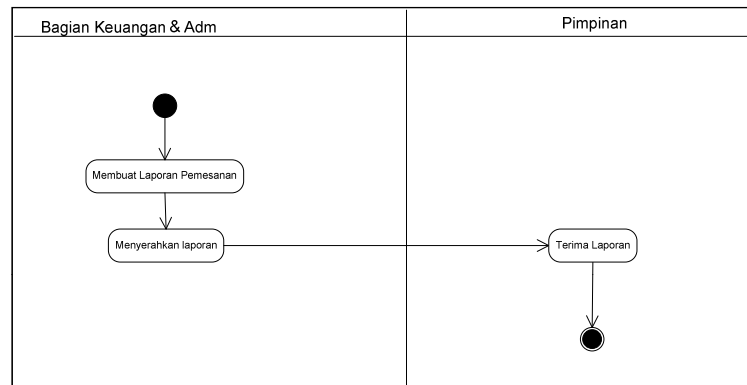


CONTOH ACTIVITY DIAGRAM

Activity
Diagram
Pembayaran



Activity Diagram Laporan



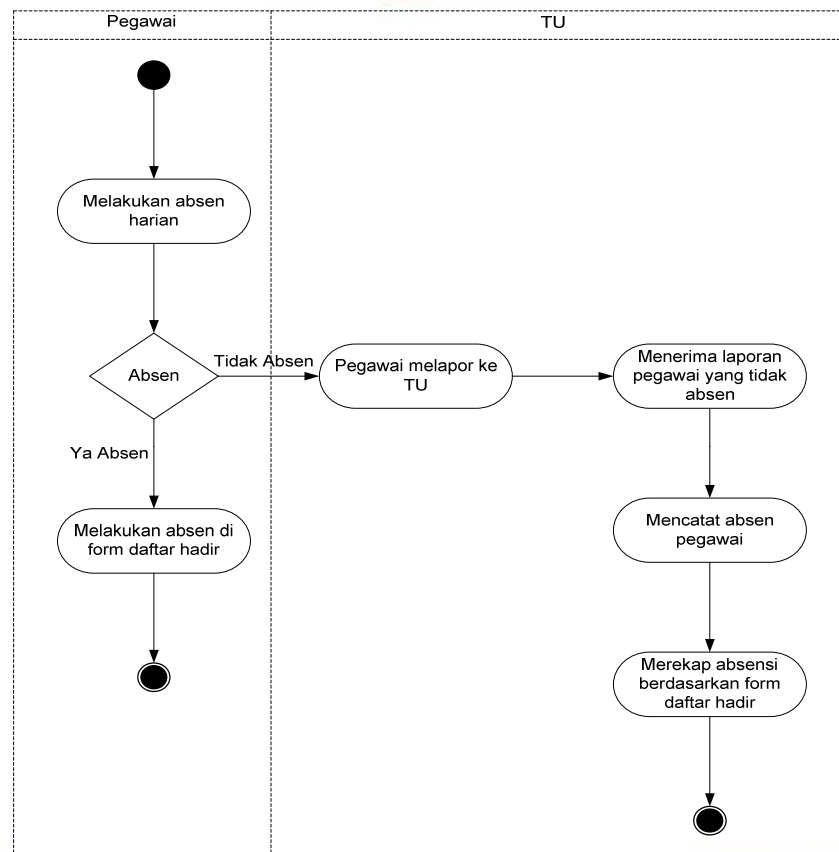
Procedure Berjalan (Acknowledgments Evi Lutfi Muktar)

Proses pembuatan Daftar Data Pegawai dan Gaji pada SMP PGRI 1 Depok adalah sebagai berikut :

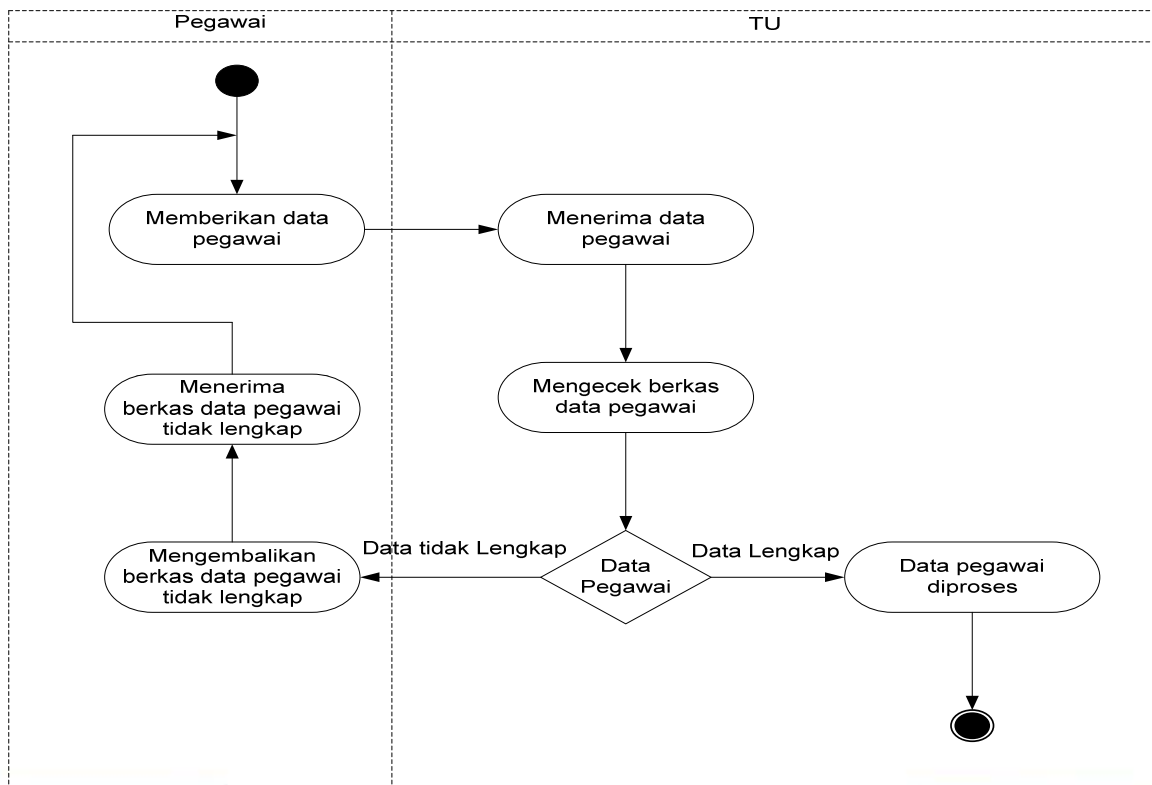
1. **Proses Absensi**
Pegawai melakukan absensi harian melalui form daftar hadir pegawai. Berdasarkan form daftar hadir pegawai tersebut bagian Tata Usaha (TU) akan membuat Rekap Absen (RA) harian untuk diserahkan kepada Administrasi.
2. **Proses Pemberian Rekap Biodata Pegawai (RBP)**
Pegawai memberikan data pribadi pegawai, data pendidikan, data keluarga yang dijadikan satu menjadi data pegawai kepada bagian Tata Usaha yang kemudian diarsipkan menjadi Rekap Biodata Pegawai (RBP). Lalu Rekap Biodata Pegawai (RBP) diserahkan kepada bagian administrasi untuk proses pengolahan Daftar Data Pegawai Dan Gaji (DDPG).

3. **Proses Pengolahan Daftar Data Pegawai dan Gaji (DDPG)**
Setelah bagian administrasi menerima Rekap Biodata Pegawai (RBP) dan Rekap Absen (RA) akan mengolah kedua data tersebut untuk dibuatkan menjadi Daftar Data Pegawai dan Gaji (DDPG) yang kemudian diserahkan kepada Kepala Sekolah untuk ditanda tangani atau di Acc.
4. **Proses Pembuatan Laporan**
Daftar Data Pegawai dan Gaji (DDPG) yang sudah diterima dan ditanda tangani oleh Kepala Sekolah akan diserahkan kembali kepada bagian Administrasi untuk dibuatkan Laporan Data Pegawai (LDP) dan Laporan Gaji Pegawai (LGP).
Setelah bagian administrasi menerima Daftar Data Pegawai dan Gaji yang sudah di Acc akan membuat Laporan Data Pegawai (LDP) dan Laporan Gaji Pegawai (LGP) yang nantinya akan diserahkan kepada Kepala Sekolah. selain itu bagian Administrasi akan membuatkan slip gaji untuk diserahkan kepada pegawai.

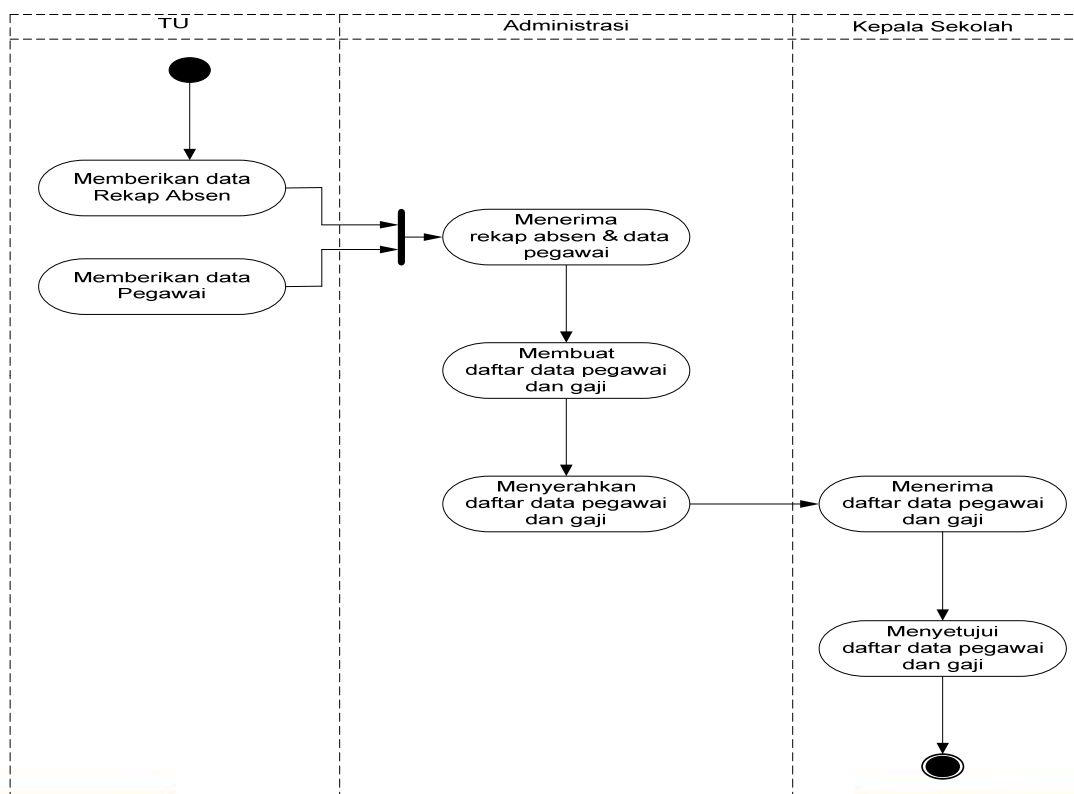
Proses Absensi



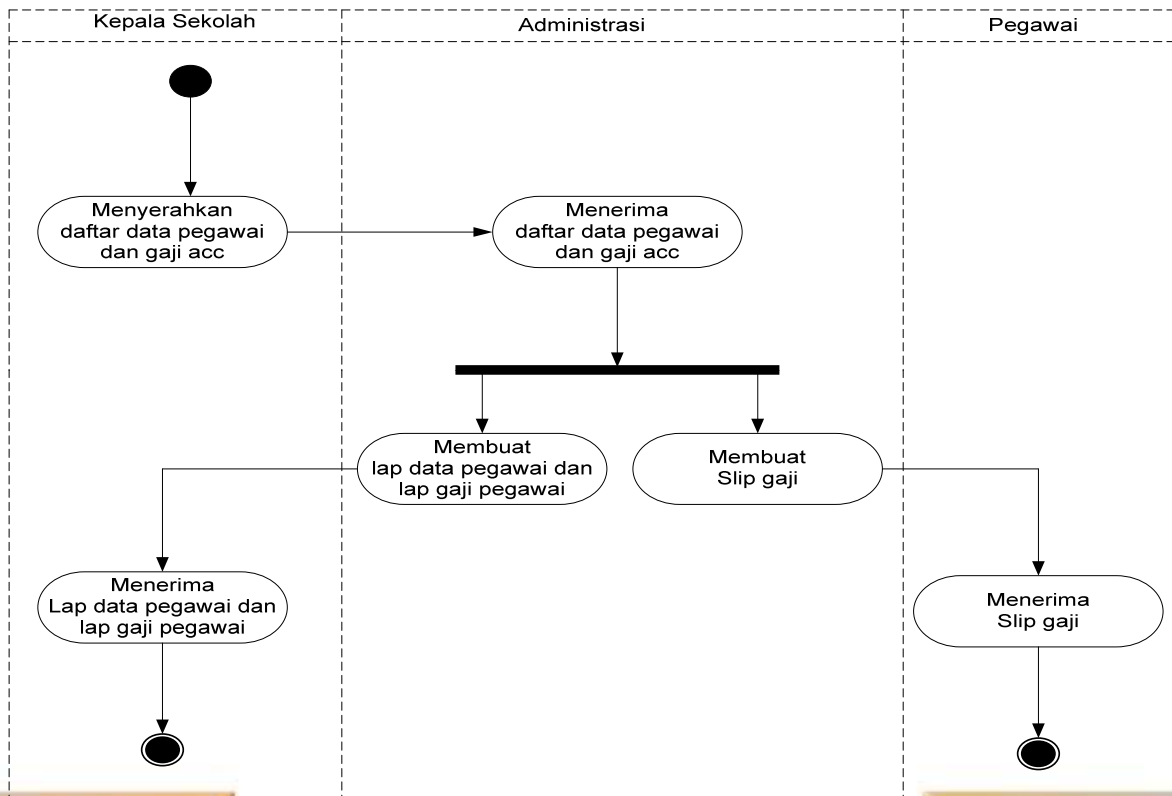
Activity Diagram Rekap Biodata Pegawai (RBP)



Activity Diagram Pembuatan Daftar Data pegawai dan Gaji (DDPG)



Activity Diagram Proses Laporan



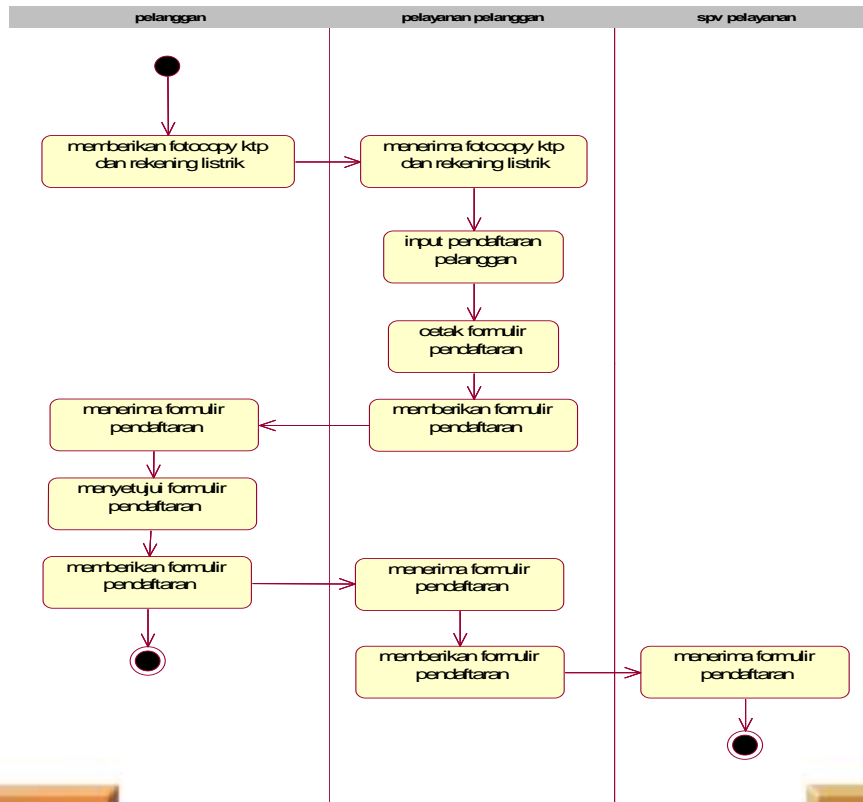
(Acknowledgments Toeko triyanto)

Proses bisnis pelayanan pelanggan perubahan daya pada PT PLN adalah sebagai berikut :

•Pendaftaran perubahan daya

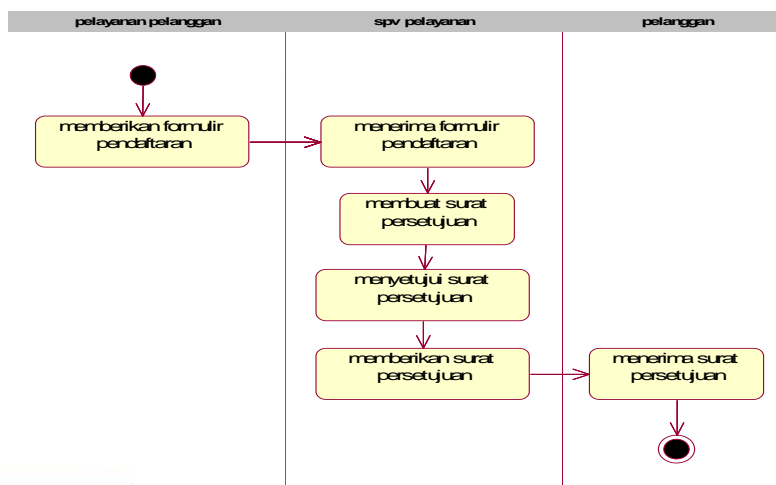
Konsumen datang ke kantor PT PLN(Persero) dengan membawa fotocopy KTP dan kwitansi pembayaran rekening bulan terakhir kemudian diserahkan dibagian pelayanan pelanggan. Pegawai pelayanan pelanggan akan menginput berdasarkan data dari konsumen , setelah diinput maka akan dicetak formulir pendaftaran perubahan daya untuk kemudian ditandatangani oleh pelanggan. Satu rangkap untuk pelanggan sebagai tanda bukti. Lainnya disimpan oleh bagian pelayanan pelanggan untuk diteruskan ke supervisor untuk proses persetujuan

Activity diagram pendaftaran perubahan daya



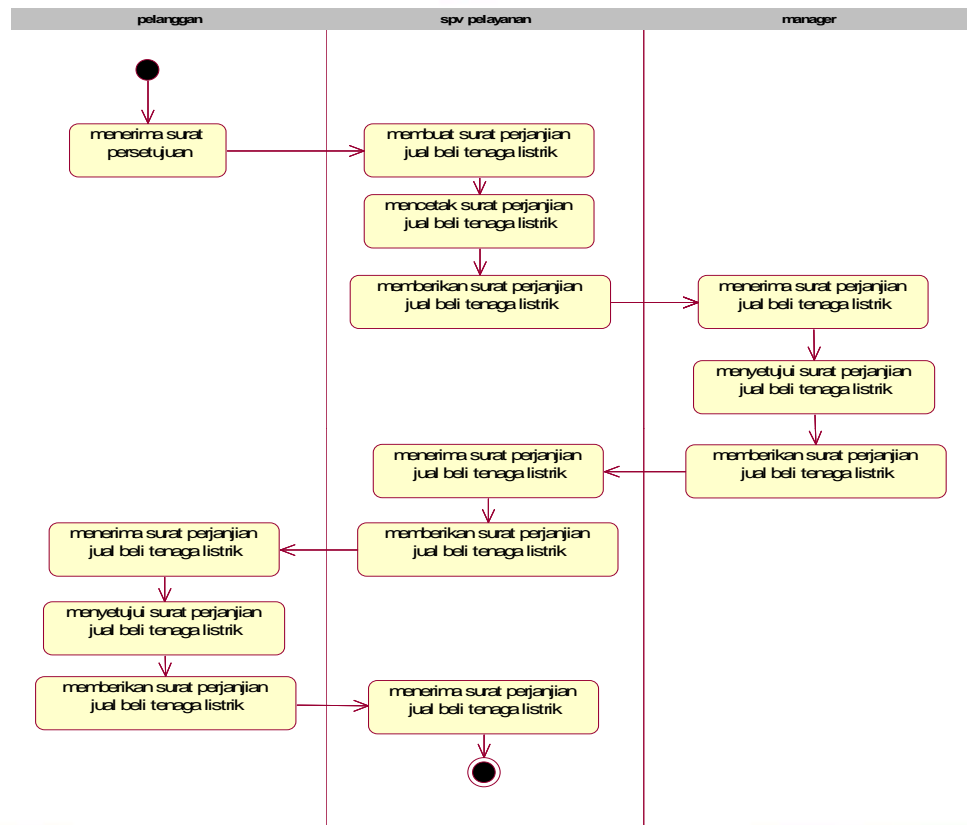
• Persetujuan perubahan daya

Rangkap formulir pendaftaran yang disimpan oleh bagian pelayanan pelanggan kemudian dibuatkan surat jawaban persetujuan yang kemudian ditandatangani oleh supervisor pelayanan pelanggan dicetak menjadi dua rangkap, rangkap pertama diberikan kepada pelanggan, sedangkan rangkap yang kedua disimpan oleh bagian pelayanan pelanggan sebagai arsip.



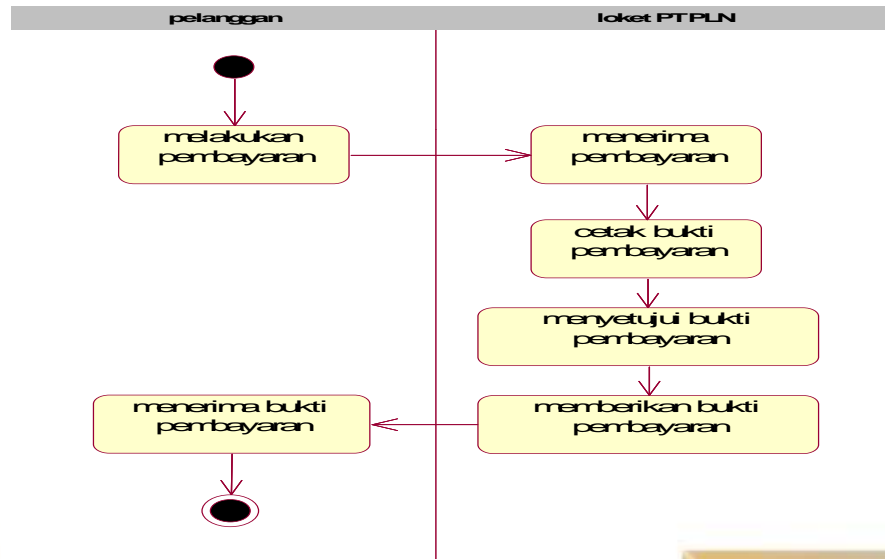
• Perjanjian jual beli tenaga listrik

Setelah pelanggan menerima surat jawaban persetujuan dari PT. PLN (Persero) maka sipelanggan akan datang ke kantor PT PLN untuk menandatangani surat perjanjian jual beli tenaga listrik sesuai dengan daya listrik yang baru yang akan dipasang. Surat perjanjian jual beli tenaga listrik tersebut juga ditandatangani oleh manager.



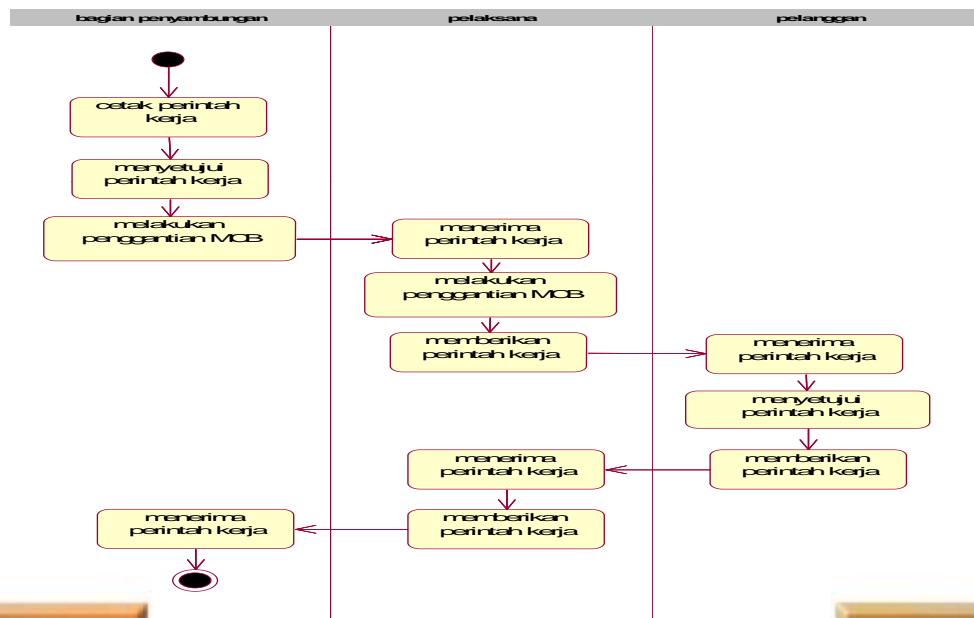
• Pembayaran

Setelah menandatangani surat perjanjian jual beli tenaga listrik maka sipelanggan tinggal membayar sejumlah yang tertera pada surat perjanjian jual beli tenaga listrik ke loket pembayaran perubahan daya, pelanggan akan mendapatkan kwitansi pembayaran sebagai bukti bahwa si pelanggan telah melaksanakan kewajibannya.



• Perintah kerja

Saat si pelanggan membayar kewajibannya maka perintah kerja terbit dan siap untuk di cetak, untuk diberikan kepada pelaksana sebagai perintah kerja untuk pelaksanaan penggantian MCB pelanggan.



Latihan STUDI KASUS ACTIVITY DIAGRAM

- Koperasi STMIK Nusa Mandiri adalah sebuah koperasi yang mengelola simpan pinjam bagi para anggotanya, berikut ini adalah kegiatan yang dilakukan oleh bagian Kredit dalam menangani pemberian pinjaman bagi para anggotanya.
- Setiap kali bagian kredit akan memberikan pinjaman kepada Anggota maka Anggota diharuskan mengisi Formulir Permohonan Pinjaman yang berisi *Nomor FPP, Tanggal Permohonan, Nomor Anggota, Nama Anggota, Jumlah Permohonan dan Keperluan*. Yang kemudian oleh Bagian Kredit dicatat dan disimpan kedalam Arsip FPP. Berdasarkan Arsip FPP tersebut Bagian Kredit membuat Bukti Peminjaman yang diberikan kepada Anggota yang berisi No. BP, tgl BP, Nomor Anggota, Nama Anggota, Jumlah Realisasi, Lama Angsuran, Jumlah Angsuran dan Bunga.

- Setiap Bulan Anggota diharuskan membayar Angsuran sejumlah Angsuran yang disepakati pada saat Peminjaman yang kemudian oleh bagian Kredit dicatat dan direkam kedalam Arsip Angsuran. Berdasarkan Arsip Angsuran tersebut bagian Kredit membuat Bukti Angsuran yang diberikan kepada Anggota yang berisi No. BA, Tanggal BA, No. BP, Jumlah Angsur dan Bunga
- Pada akhir bulan Bagian Kredit selalu membuat Laporan Peminjaman dan Laporan Angsuran yang diberikan Kepada Ketua Koperasi.

Sequence Diagram

Sequence Diagram

- *Sequence diagram* menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atas dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

- *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang *men-trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan.
- Diagram ini secara khusus berasosiasi dengan use case diagram
- Memperlihatkan tahap demi tahap apa yang seharusnya terjadi untuk menghasilkan sesuatu didalam use case

Simbol Sequence Diagram

a. An Actor



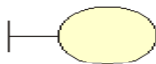
Menggambarkan orang yang sedang berinteraksi dengan sistem

b. Entity Class



Menggambarkan hubungan kegiatan yang akan dilakukan

c. Boundary Class



Menggambaran sebuah penggambaran dari form

d. Control Class



Menggambarkan penghubung antara boundary dengan tabel

e. A focus Of Control & A life line



Menggambarkan tempat mulai dan berakhirnya sebuah message

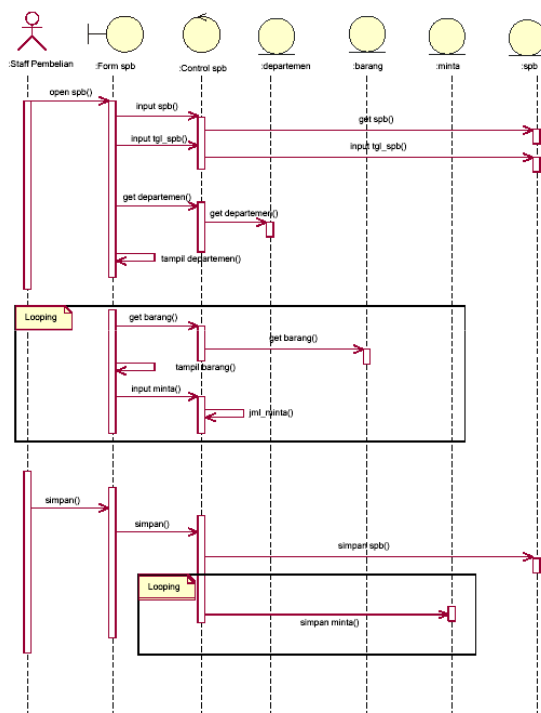
f. A message

A Message{}



Menggambarkan Pengiriman Pesan

Contoh Sequence Diagram



Form spb

ASTRA CREDIT COMPANIES
Jl. R. Fatmawati No.9 Jakarta – Selatan 14240

ENTRI SPB

Spb: Nomor SPB [auto number] Tanggal SPB dd-mm-yyyy

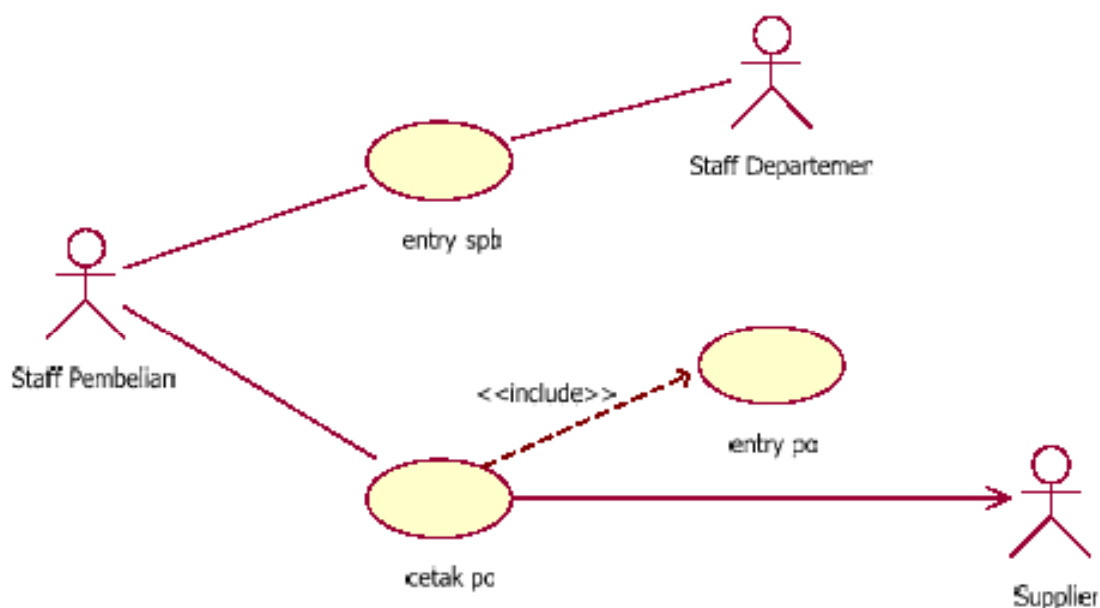
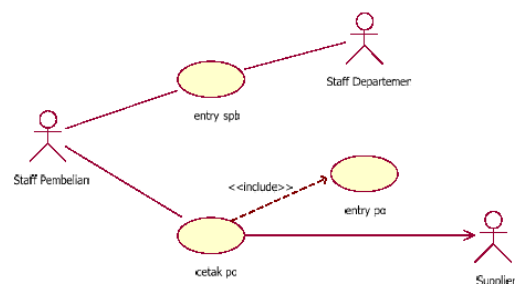
Departemen: Kode Departemen [pilih] Nama Departemen [tampil]

Barang: Kode Barang [pilih] Saluran [tampil]

Nama Barang [tampil] Jumlah Minta [99]

Kode Barang	Nama Barang	Saluran	Jumlah Minta
[tampil]	[tampil]	[tampil]	[tampil]

[Simpan] [Batal] [Keluar]



Form spb



ASTRA CREDIT COMPANIES
Jl. R's Fatmawati No.9 Jakarta – Selatan 14240

ENTRY SPB

Spb

Nomor SPB

[auto number]

Tanggal SPB

dd-mm-yyyy

Departemen

Kode Departemen

[pilih]

Nama Departemen

[tampil]

Barang

Kode Barang

[pilih]

Satuan

[tampil]

Nama Barang

[tampil]

Jumlah Minta

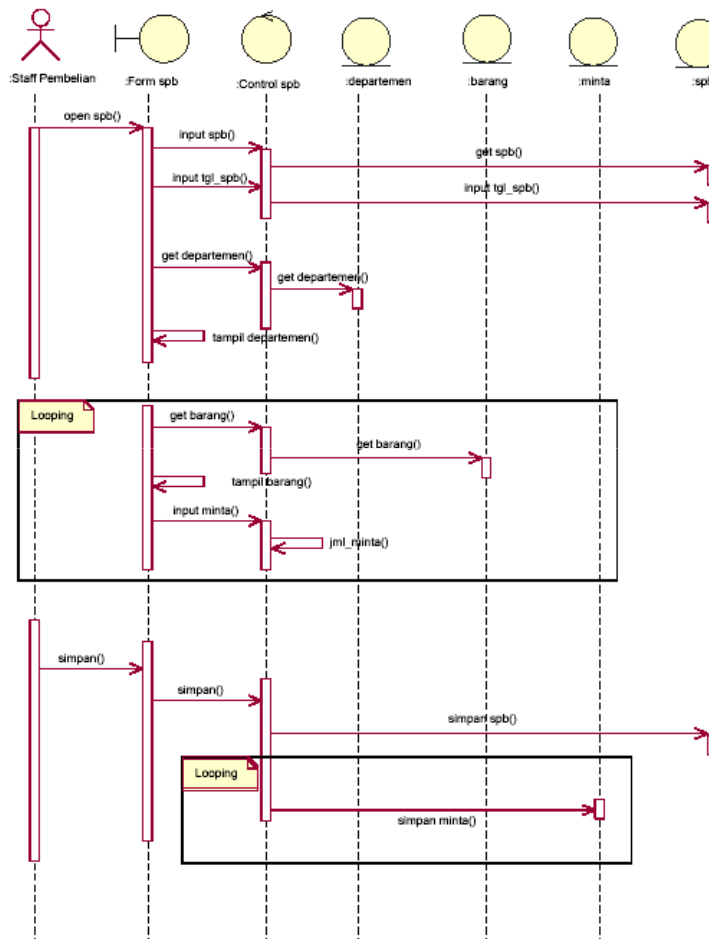
[95]

Kode Barang	Nama Barang	Satuan	Jumlah Minta
[tampil]	[tampil]	[tampil]	[tampil]

Simpan

Batal

Keluar



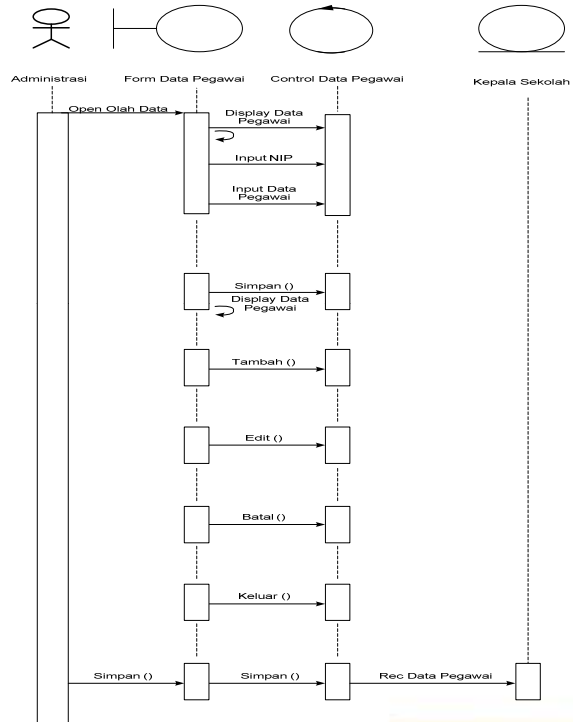
SEQUENCE DIAGRAM INPUT DATA PEGAWAI

Laporan Data Pegawai

DATA PEGAWAI PGRI 1 DEPOK

NIP	Nama	Jenis Kelamin	Tgl Lahir	Tempat
1001	Rida Fahrudin	Laki-laki	01/01/1960	Tasikmalaya
1002	Sutomo	Laki-laki	08/04/1957	Sleman

Cetak Hapus Keluar



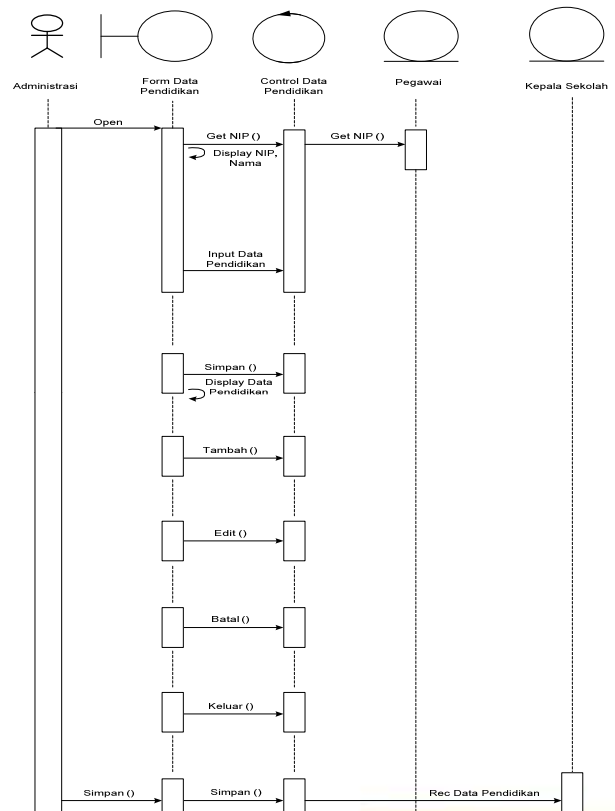
SEQUENCE DIAGRAM INPUT DATA PENDIDIKAN

Laporan Data Pendidikan

DATA PENDIDIKAN PEGAWAI PGRI 1 DEPOK

NIP	Nama	Pendidikan	Universitas	Jurusan
1001	Rida Fahrudin	S2	UNPAK	Keguruan Penc B. Inggris
1002	Sutomo	S1	UT	

Cetak Hapus Keluar

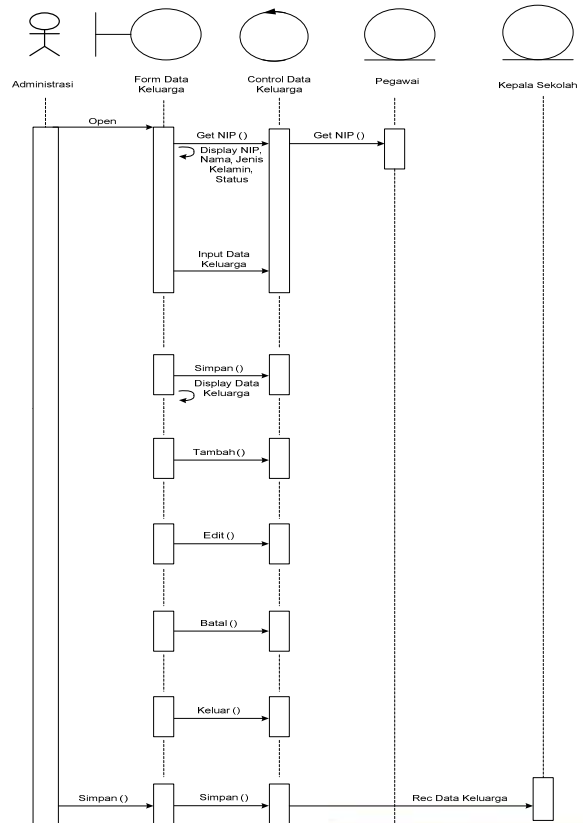


SEQUENCE DIAGRAM INPUT DATA KELUARGA

Laporan Data Keluarga

NIP	Nama Pegawai	Nama Istri/suami	Status	Jenis Kelamin
1001	Rida Fahrudin	Aan Rahmah	Istri	Laki-Laki
1002	Sutomo	Nuraini	Istri	Perempuan

Cetak Hapus Keluar

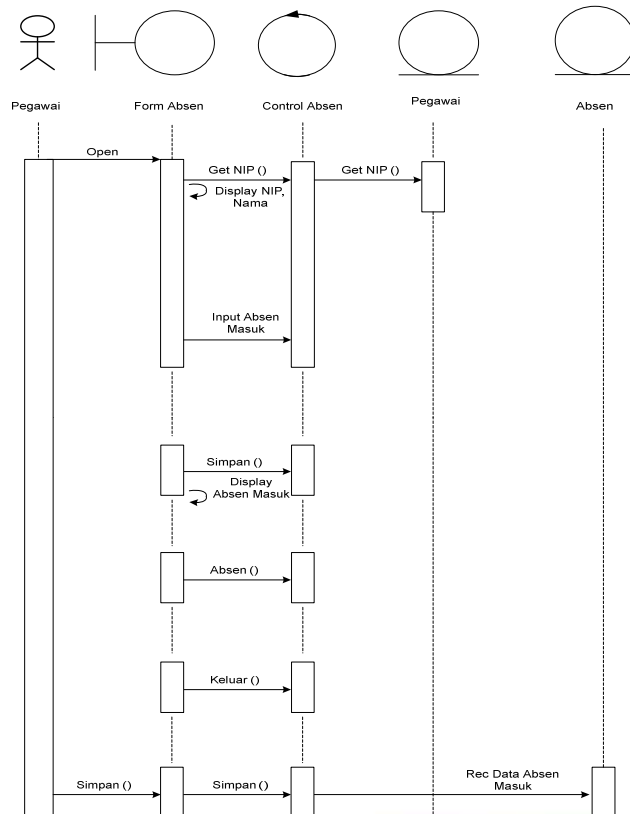


SEQUENCE DIAGRAM ABSEN MASUK

Absens Masuk

NIP	Nama	Masuk	Tanggal	Hari
1001	Rida Fahrudin	17:18:38	30/01/2009	30
1002	Sutomo	13:07:52	05/02/2009	5

Cetak Hapus Keluar



Contoh kasus PLN (Acknowledgments Toeko triyanto)

PT PLN (Persero)
DISTRIBUSI JAWA BARAT DAN BANTEN

[Home] [Manajemen User] [Manajemen Modul] [Profil] [Pendaftaran] [cetak dokumen] [Kontas] [perintah kerja] [Mutasi] [Manajemen Kontrol] [Data Pelanggan] [Informasi Tagihan] [Guestbook] [Logout]

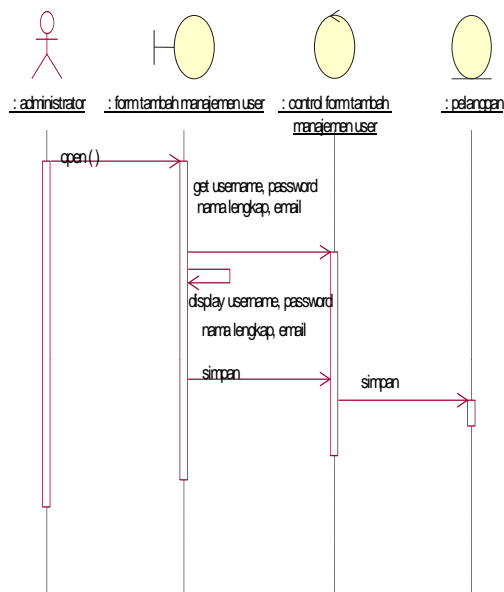
Pendaftaran Perubahan Daya

Tambah Data

NO	AGENDA	TGL	ID PELANGGAN	NAMA	DAYA LAMA	DAYA BARU	STATUS	Aksi
1	33	26 July 2009	538750011111	TO EKO TRIYANTO	1300	2200	4	Sdh dicetak
2	32	26 July 2009	538750011111	TO EKO TRIYANTO	1300	900	0	Cetak
3	31	26 July 2009	538750011111	TO EKO TRIYANTO	1300	900	2	Sdh dicetak
4	30	26 July 2009	538750011111	TO EKO TRIYANTO	900	1300	2	Sdh dicetak
5	29	23 July 2009	538750011111	TO EKO TRIYANTO	900	900	2	Sdh dicetak
6	28	23 July 2009	538750011111	TO EKO TRIYANTO	900	1300	5	Sdh dicetak

Ket Status

- 0 = Sdh dicetak
- 1 = Pendaftaran
- 2 = Cetak Pendaftaran
- 3 = Surat jawaban persetujuan
- 4 = Surat perjanjian jual beli tenaga listrik



PT PLN (Persero)
DISTRIBUSI JAWA BARAT DAN BANTEN

[Home] [Manajemen User] [Manajemen Modul] [Profil] [Pendaftaran] [cetak dokumen] [Kontas] [perintah kerja] [Mutasi] [Manajemen Kontrol] [Data Pelanggan] [Informasi Tagihan] [Guestbook] [Logout]

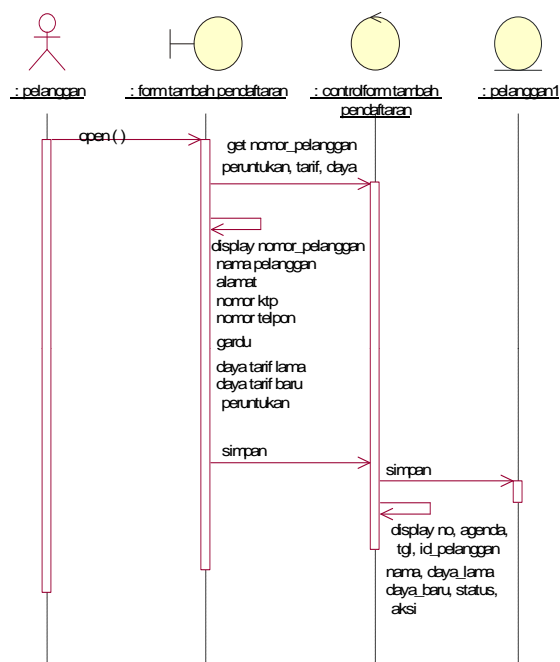
Pendaftaran Perubahan Daya

Tambah Data

NO	AGENDA	TGL	ID PELANGGAN	NAMA	DAYA LAMA	DAYA BARU	STATUS	Aksi
1	33	26 July 2009	538750011111	TO EKO TRIYANTO	1300	2200	4	Sdh dicetak
2	32	26 July 2009	538750011111	TO EKO TRIYANTO	1300	900	0	Cetak
3	31	26 July 2009	538750011111	TO EKO TRIYANTO	1300	900	2	Sdh dicetak
4	30	26 July 2009	538750011111	TO EKO TRIYANTO	900	1300	2	Sdh dicetak
5	29	23 July 2009	538750011111	TO EKO TRIYANTO	900	900	2	Sdh dicetak
6	28	23 July 2009	538750011111	TO EKO TRIYANTO	900	1300	5	Sdh dicetak

Ket Status

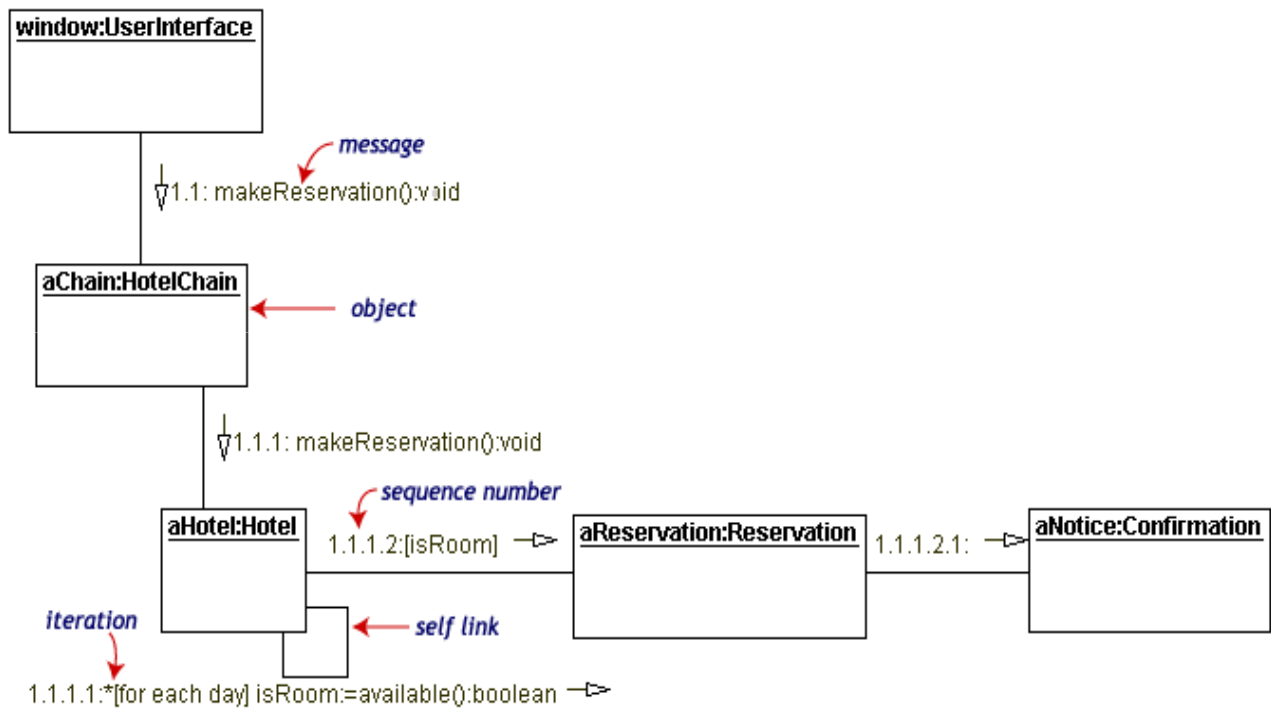
- 0 = Sdh dicetak
- 1 = Pendaftaran
- 2 = Cetak Pendaftaran
- 3 = Surat jawaban persetujuan
- 4 = Surat perjanjian jual beli tenaga listrik



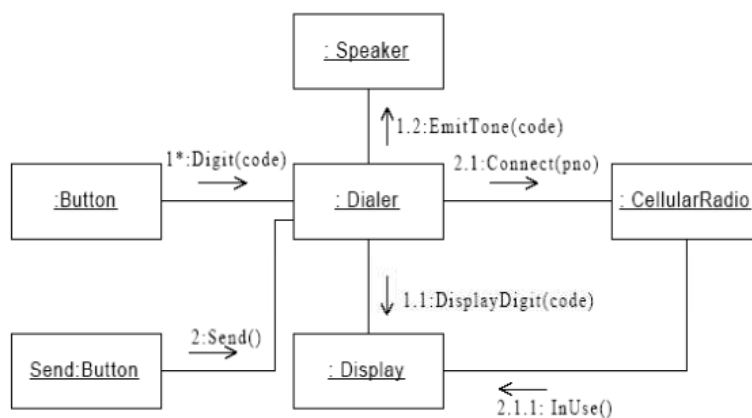
Collaboration Diagram

- *Collaboration diagram* juga menggambarkan interaksi antar objek seperti *sequence diagram*, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu Penyampaian *message*.
- Setiap *message* memiliki *sequence number*, di mana *message* dari level tertinggi memiliki nomor 1. Messages dari level yang sama memiliki prefiks yang sama.

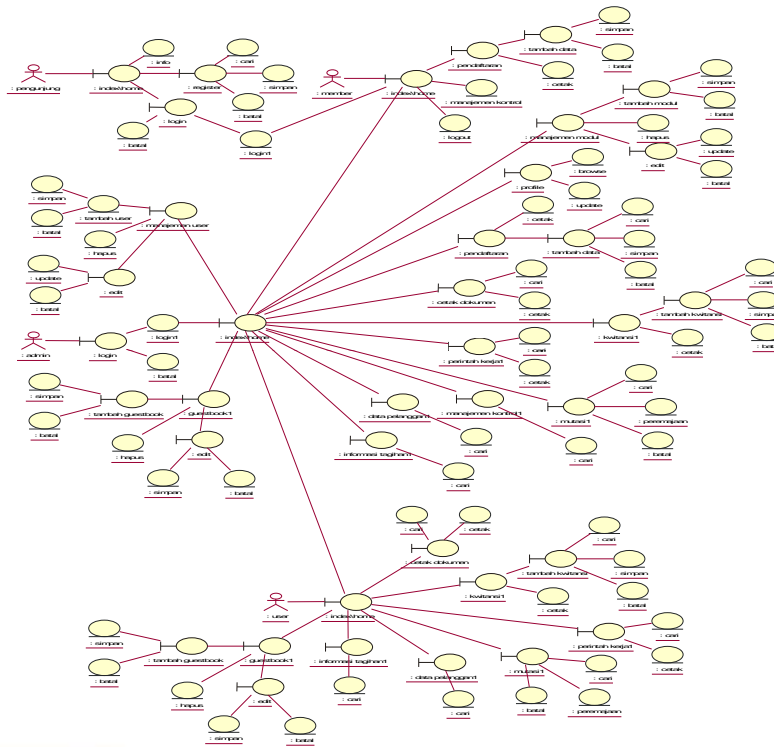
Contoh Collaboration Diagram



Berikut adalah sebuah contoh *collaboration diagram* yang mengilustrasikan sebuah sistem telepon genggam (*handphone*) :



Collaboration Diagram (Acknowledgments Toeko triyanto)



Tugas : Buatlah rancangannya dengan menggunakan Tools , misalnya :

- Enterprise Architect
- Rational Rose
- Argo UML
- Visual Paradigm

Sesuai dengan diagram yang telah dibahas pada pertemuan sebelumnya.

Lanjutkan kasus yang telah dibuat pada pertemuan sebelumnya (Pertemuan IV)

Tugas

Bobot 50%

Buatlah program aplikasi dari pengembangan design UML yang telah di buat sebelumnya serta dipresentasikan (Program dan Design UML) agar terlihat kesesuaian program dgn design.

Dipresentasikan setelah MID TEST perkelompok

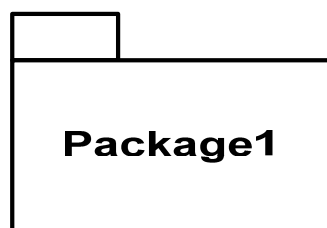
PERTEMUAN 6

Package Diagram

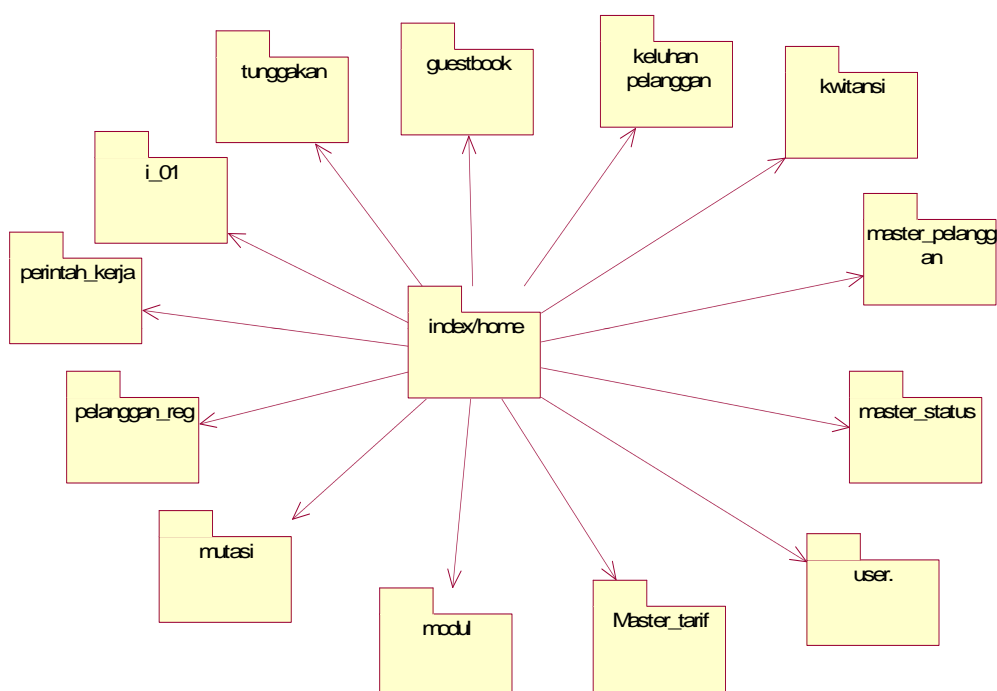
Sebuah bentuk pengelompokan yang memungkinkan untuk mengambil sebuah bentuk di UML dan mengelompokkan elemennya dalam tingkatan unit yang lebih tinggi. Kegunaan *package* yang paling umum adalah untuk mengelompokkan *class*.

•Package Diagram

Menggambarkan pengelompokan dari suatu *class-class*



Contoh package diagram (Acknowledgments Toeko triyanto)



Entity Relationship Diagram


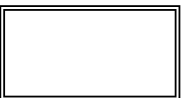
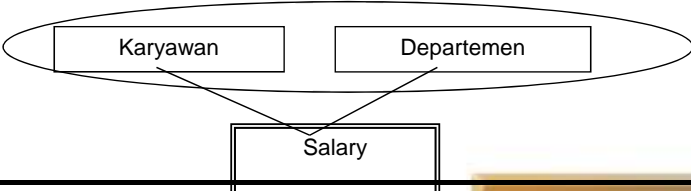
ERD adalah :

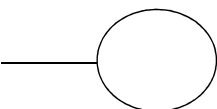
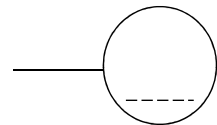
Model untuk menjelaskan hubungan antar data dalam basis data berdasarkan suatu persepsi bahwa real word terdiri dari objek-objek dasar yang mempunyai hubungan atau relasi antara objek-objek tersebut


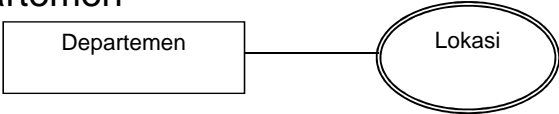
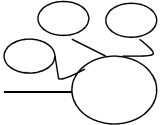
TAHAP MEMBUAT ERD

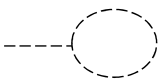
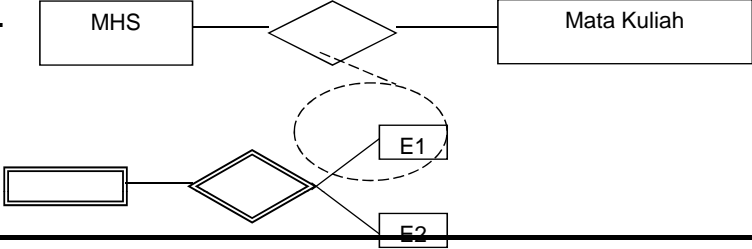
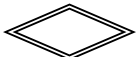
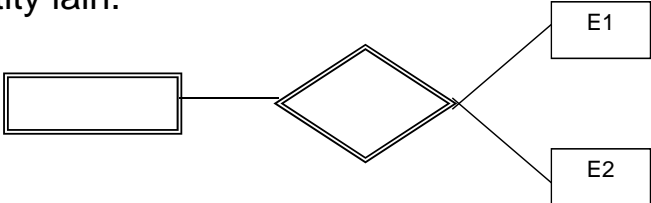
1. Keluarkan semua atribut yang dimiliki oleh dokumen sumber
2. Tentukan Atribut yang dapat menjadi Primary Key jika **Tidak ada** boleh **dibuat baru** lalu tentukan ketergantungan atribut terhadap primary key nya
3. Tentukan nama entitas dari kelompok atribut yang telah bergantung terhadap primary keynya.
4. Gambarkan hubungan masing-masing entitas beserta atribut – atributnya.
5. Tentukan Cardinality/tingkat hubungan dari masing-masing Entitas yang telah terhubung.

Notasi dan Penamaan Untuk Konstruksi Skema Diagram ER

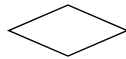
No	Simbol	Keterangan
1.		<p>Entity Type</p> <p>Suatu yang ada (secara eksplisit ada) namun keberadaannya dapat nyata dapat virtual, serta perbedaan antar entity harus jelas.</p> <p>Ex. Pegawai, Departemen</p>
2.		<p>Weak entity Type</p> <p>Suatu entity yang tidak punya key atribut keberadaannya tidak perlu berdiri sendiri / diluar system. Didalam weak dimungkinkan 1 weak memiliki banyak entity. Setidaknya-tidaknya memiliki 1 relasi.</p> <p>Ex.</p> 

3.		<p>Attribute</p> <p>Keterangan yang dimiliki entity / sifat-sifat yang melekat pada entity yang perlu dicatat.</p> <p>Ex. Pegawai: Nopeg, Nama, Alamat, Jenis Kel, tgl. Masuk</p>									
4.		<p>Key Attribute</p> <p>Bila didalam attribute terdapat nilai sama, maka kita perlu membuat "Key attribute" sehingga dipastikan tidak akan terjadi nilai/record sama.</p> <p>Ex. Pegawai : sebagai key adalah NoPeg</p> <table border="1"> <thead> <tr> <th>NoPeg</th><th>Nama</th><th>Alamat</th></tr> </thead> <tbody> <tr> <td>P01</td><td>Bella</td><td>Malang</td></tr> <tr> <td>P02</td><td>Bella</td><td>Batu</td></tr> </tbody> </table>	NoPeg	Nama	Alamat	P01	Bella	Malang	P02	Bella	Batu
NoPeg	Nama	Alamat									
P01	Bella	Malang									
P02	Bella	Batu									

5.		<p>Multivalued Attribute</p> <p>Satu entity yang memiliki 2 attribute sama</p> <p>Ex. Departemen yang memiliki 2 lokasi pabrik</p> <p>Departemen</p>  <p>Hal ini bukan berarti bias untuk orang yang mempunyai 2 nama atau 2 alamat</p>
6.		<p>Composite Attribute</p> <p>Attribute yang mempunyai nilai attribute lebih dari Satu</p> <p>Ex. Nama : Nama Depan, Nama Tengah, Nama Belakang</p> <p>Alamat : Jalan, Nomer, Kota</p>

7.		<p>Derived Attribute</p> <p>Merupakan kombinasi dari attribute-attribute dimana keberadaannya tidak perlu disimpan.</p> <p>Ex.</p> 
8.		<p>Identifying Relationship Type</p> <p>Bila entity mempunyai hubungan lebih dari satu entity lain.</p> 

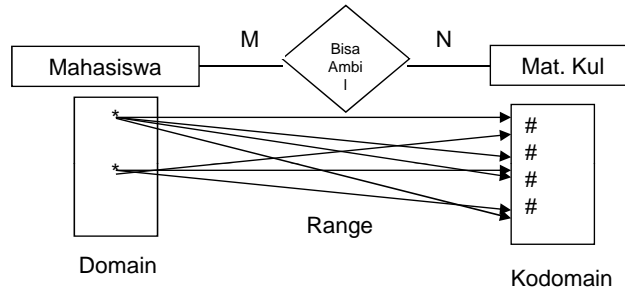
9.



Relationship Type

Menyatakan hubungan antar attribute sehingga terjadi pemetaan.

Ex.



Hasil Dari Relasi :

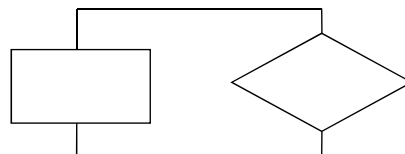
One To One (1:1)

One To Many (1:N)

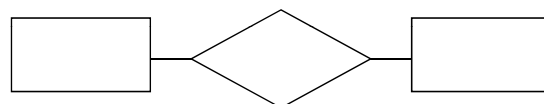
Many To Many (1:M)

Derajat Relationship

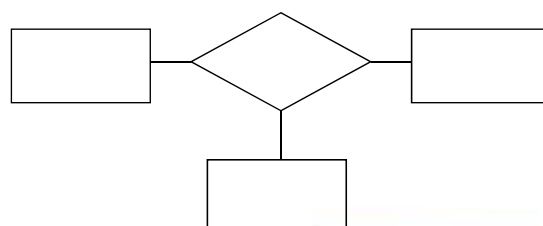
★ UNARY RELATIONSHIP



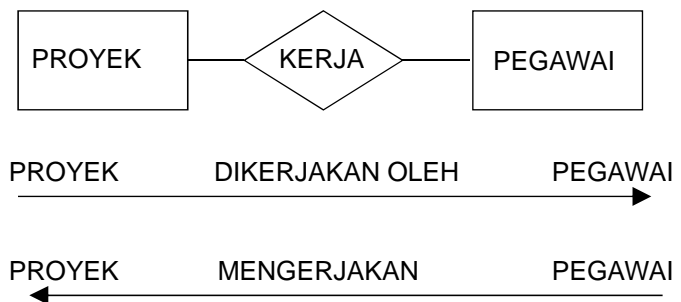
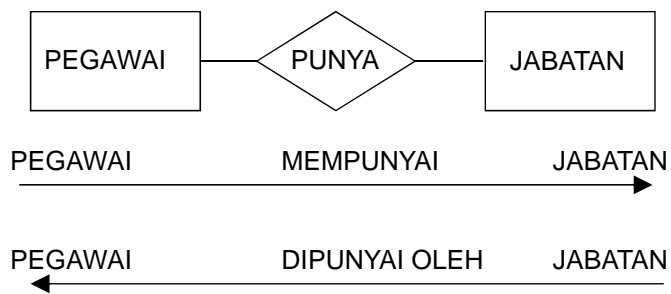
★ BINARY RELATIONSHIP



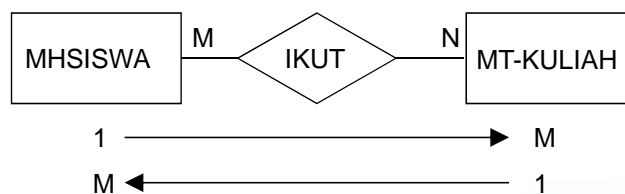
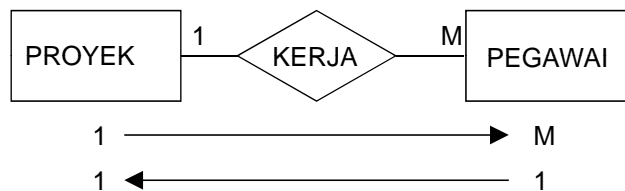
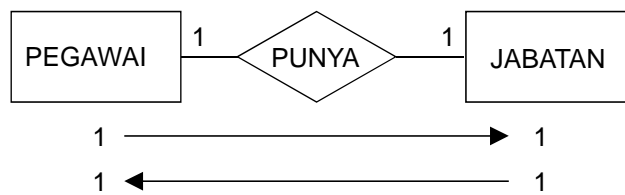
★ N-ARY RELATIONSHIP



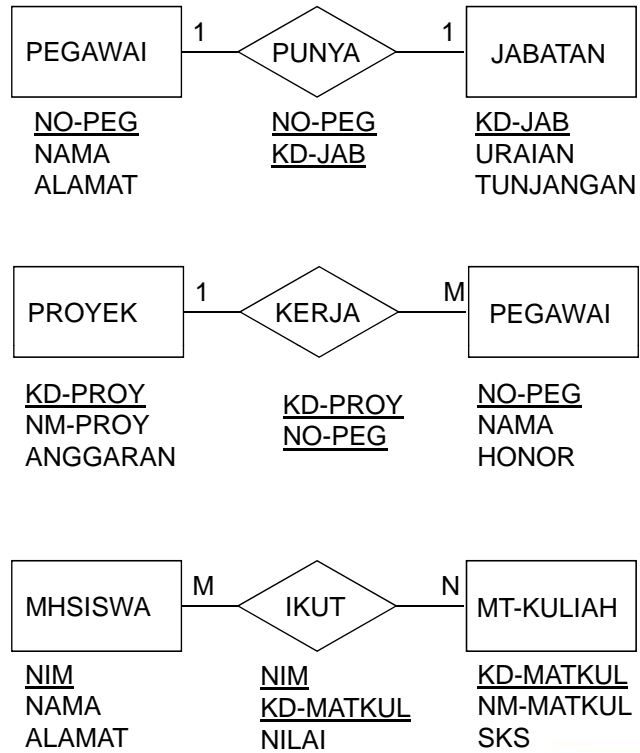
ENTITY-RELATIONSHIP DIAGRAM



ENTITY-RELATIONSHIP DIAGRAM

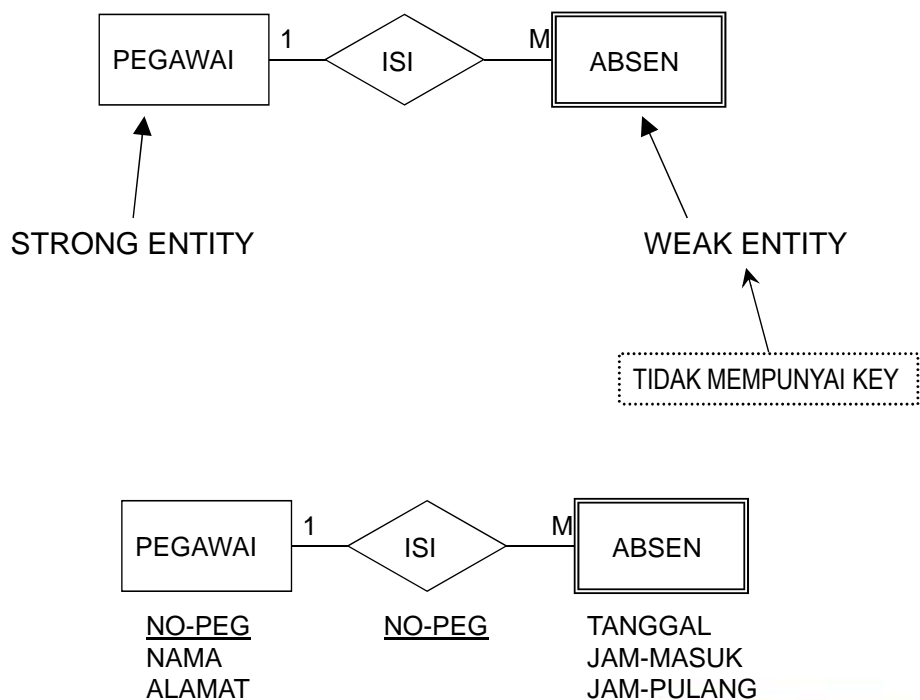


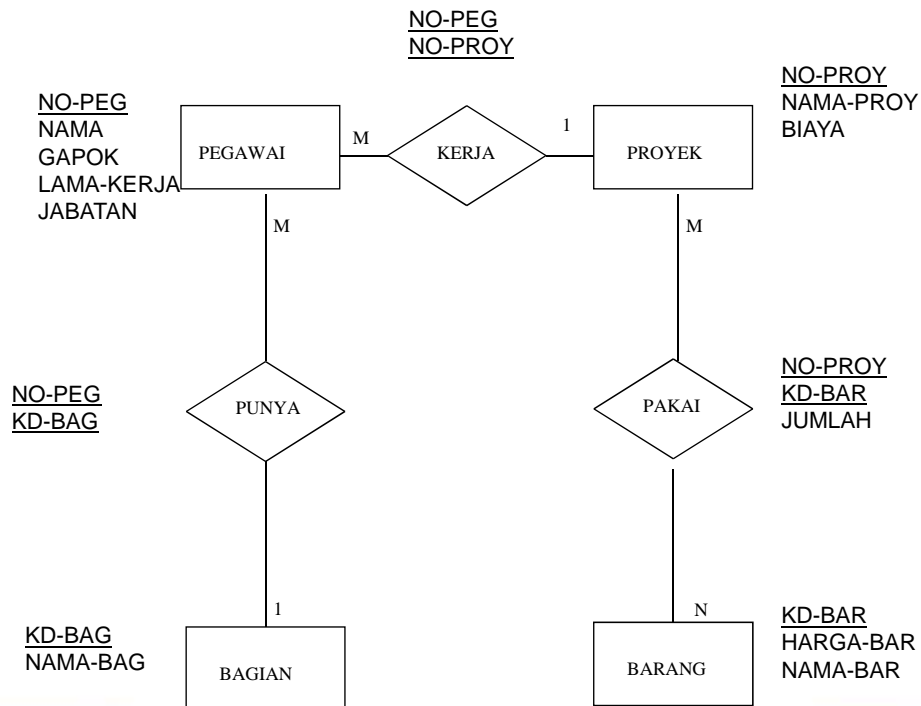
ENTITY-RELATIONSHIP DIAGRAM



ENTITY-RELATIONSHIP DIAGRAM

★ JENIS ENTITY





Tugas : Buatlah rancangannya dengan menggunakan Tools,

misalnya :

Enterprise Architect

Rational Rose

Argo UML

Visual Paradigm sesuai dengan diagram yang telah dipelajari diatas
(Melanjutkan kasus yang telah dibuat pada pertemuan sebelumnya
(Pertemuan IV dan V))

Pertemuan 9

PRINSIP DAN KONSEP DESAIN

Pokok Bahasan dalam RPL :

- ❖ Desain PL dan Rekayasa PL
- ❖ Prinsip Desain
- ❖ Konsep Desain
- ❖ Desain Modular Afektif
- ❖ Model Desain
- ❖ Dokumentasi Desain

Buku Referensi :

- Pressman, RS., 2008, Software Engineering: A Practitioner's Approach, New York: McGraw-Hill
- Sommerville, I, 2007, Software Engineering, Addison Wesley

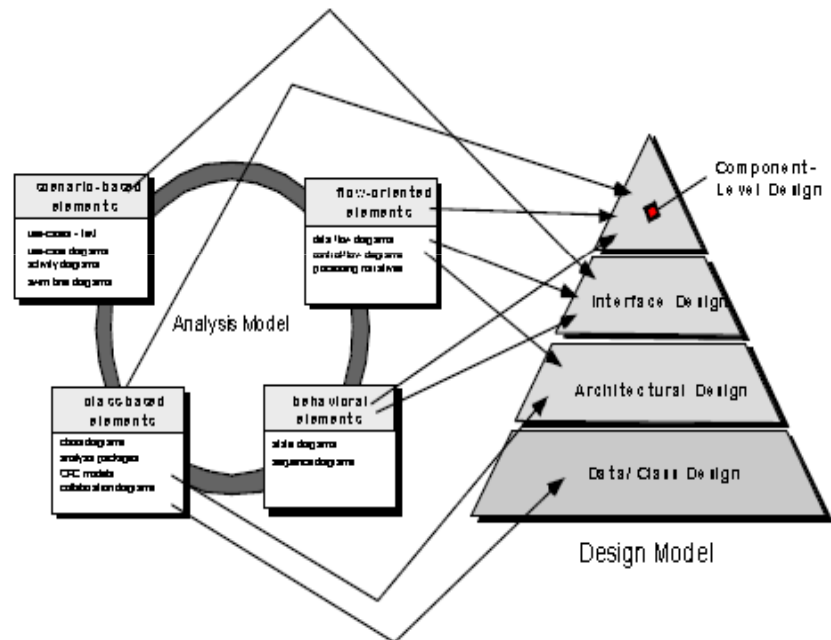
TUJUAN PRINSIP DAN KONSEP DESAIN

- Memahami konsep dan prinsip desain PL
- Mengerti desain secara modular dapat mengurangi kompleksitas program dan mudah diimplementasikan
- Memahami model desain
- Membuat dan mengetahui isi dari dokumentasi

Desain dan Rekayasa Perangkat Lunak

Hal yang harus diperhatikan :

- Desain Data
- Desain Arsitektur
- Desain Interface
- Desain Prosedural



PROSES DESAIN

3 karakteristik pedoman bagi evaluasi Desain :

- Desain mengimplementasikan semua kebutuhan eksplisit yang ada dalam model analisis, dan mengakomodasi semua kebutuhan implisit yang diinginkan oleh konsumen.
- Desain harus dapat berupa panduan yang dapat dibaca dan dipahami oleh orang-orang yang akan membuat kode, dan mereka yang menguji serta nantinya mendukung PL
- Desain harus menyediakan gambaran utuh dari PL, menggambarkan domain data, fungsional, dan perilaku dari perspektif implementasi.

Evolusi Desain Perangkat Lunak

Karakteristik Umum :

1. Mekanisme penerjemahan suatu model analisis ke dalam representasi desain.
2. Notasi untuk merepresentasikan komponen-komponen fungsional dan interfacenya.
3. Heuristik bagi penyaringan dan partisi.
4. Pedoman bagi penilaian kualitas.

KONSEP – KONSEP DESAIN

Konsep desain PL fundamental memberikan kerangka kerja untuk mendapatkan program yang berfungsi dengan benar.

- Abstraksi
- Penyaringan
- modularitas
- Arsitektur perangkat Lunak
- Hirarki Kontrol / struktur Program
- Partisi Struktural
- Struktur data
- Prosedur perangkat lunak
- Penyembunyian Informasi

Penyaringan

- Penyaringan sebenarnya adalah proses elaborasi . Dimulai dengan suatu statemen fungsi pada suatu tingkat abstraksi tinggi.
- Statemen fungsi adalah statemen yang menggambarkan fungsi atau informasi secara konseptual.
- Penyaringan membantu desainer untuk mengungkapkan detail tingkat rendah ketika desain berjalan.

Modularitas

5 kriteria mengevaluasi metode desain menurut Meyer :

- a. Dekomposabilitas Modular → dekomposisi
- b. Komposabilitas Modular
- c. Kemampuan Pemahaman Modular
- d. Kontinuitas Modular
- e. Ptoreksi Modular

Arsitektur Perangkat Lunak

Sekumpulan properti sebagai bagian dari desain arsitektural (Shaw dan Garlan) :

1. Properti Struktural
2. Properti Ekstra Fungsional
3. Keluarga dari sistem yang berhubungan

Patern / Pola

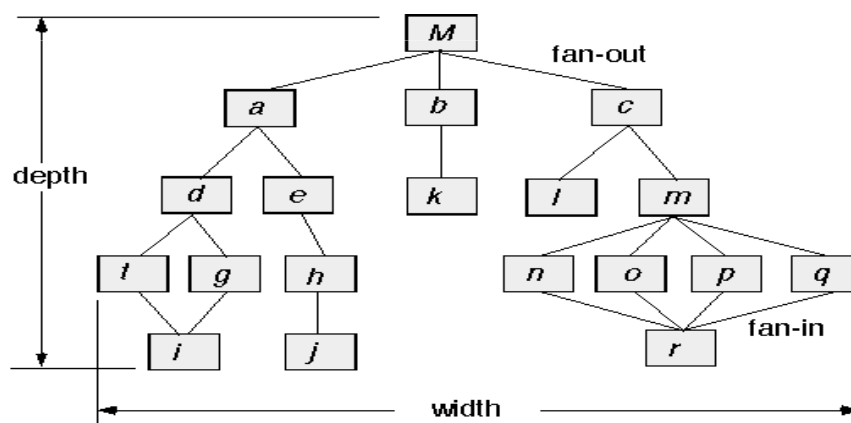
- Design Pattern
adalah sebuah istilah (English) dalam Rekayasa Perangkat Lunak (Software Engineering) yang mengacu kepada solusi umum yang dapat digunakan secara berulang kali untuk menyelesaikan masalah-masalah umum yang ditemukan dalam disain perangkat lunak.

Hierarki Kontrol

- Yang paling umum digunakan adalah diagram pohon
- Depth dan width mengindikasikan jumlah modul yang dikontrol dan rentang keseluruhan kontrol
- Fan-out pengukuran jumlah modul yang dikontrol secara langsung oleh modul yang lain.
- Fan-in mengindikasikan berapa banyak modul yang secara langsung mengontrol sebuah modul yang diberikan.
- Hubungan kontrol diantara kontrol :
 - Superordinat (modul yang mengontrol modul lain).
 - Subordinat (modul yang dikontrol modul lain)

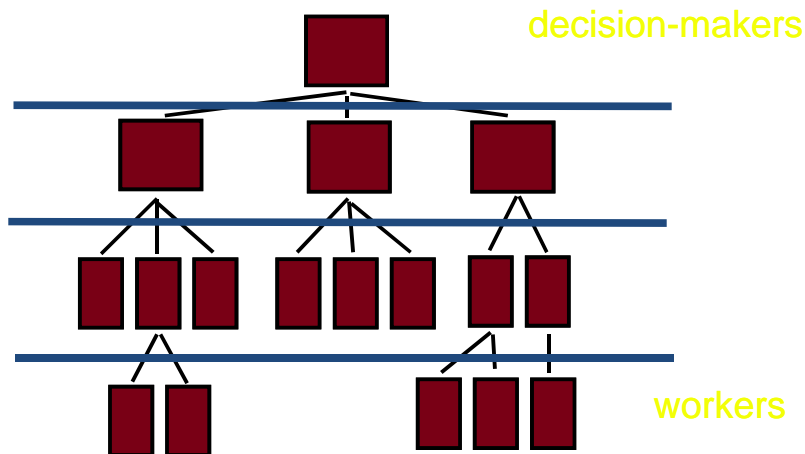
Hierarki Kontrol (2)

- Visibilitas (komponen program yang dapat dipakai sebagai data oleh komponen lainnya)
- Konektivitas (Komponen yang dipakai secara tidak langsung oleh sebuah modul yang ditetapkan)



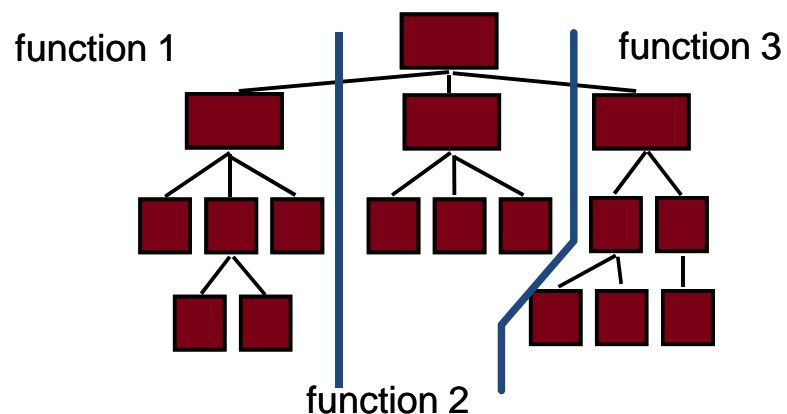
Partisi Struktural

- Partisi Vertikal
- Didesain sehingga pengambilan keputusan dan pekerjaan distratifikasi
- Modul pengambilan keputusan tetap ada di puncak arsitektur



Partisi Struktural (lanjutan)

- Partisi Horizontal
- Tentukan cabang yang terpisah pada hierarki modul untuk setiap fungsi utama
- Gunakan modul kontrol untuk koodinasi komunikasi antar fungsi2x



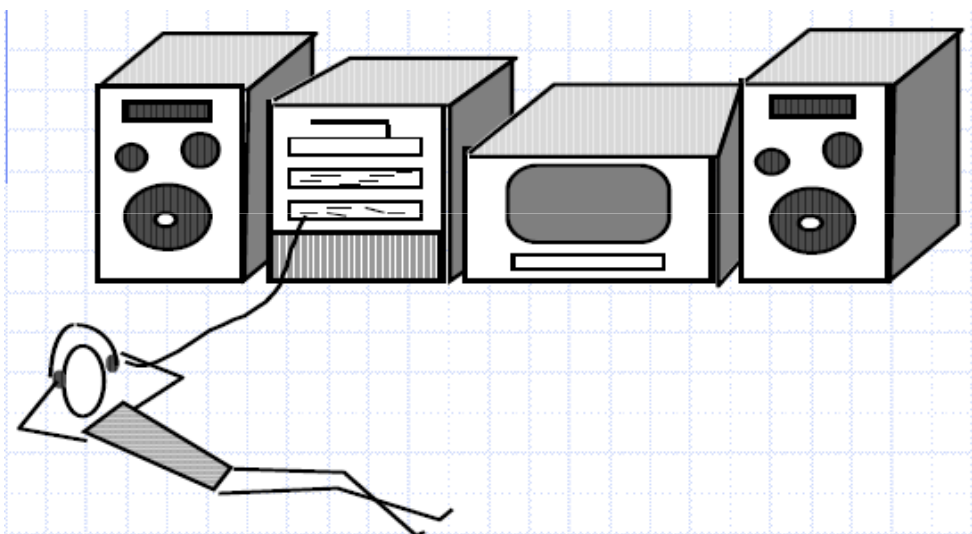
Struktur Data

Struktur Data menentukan :

- Organisasi dan kompleksitas
- Item Skalar
- Metode akses
- Vektor Sekuensial

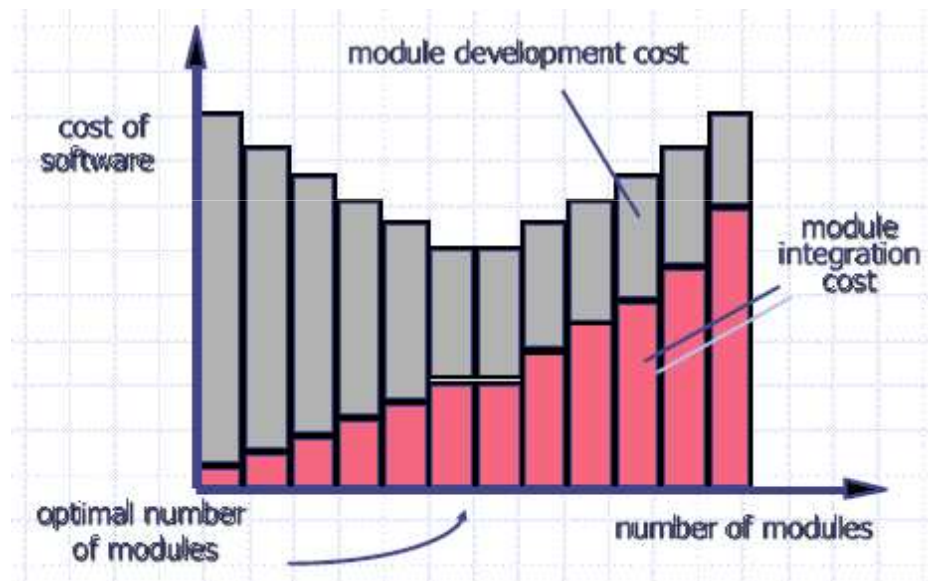
DESAIN MODULAR AFEKTIF

- Mudah untuk dibangun, mudah untuk dirubah dan mudah untuk ditetapkan...

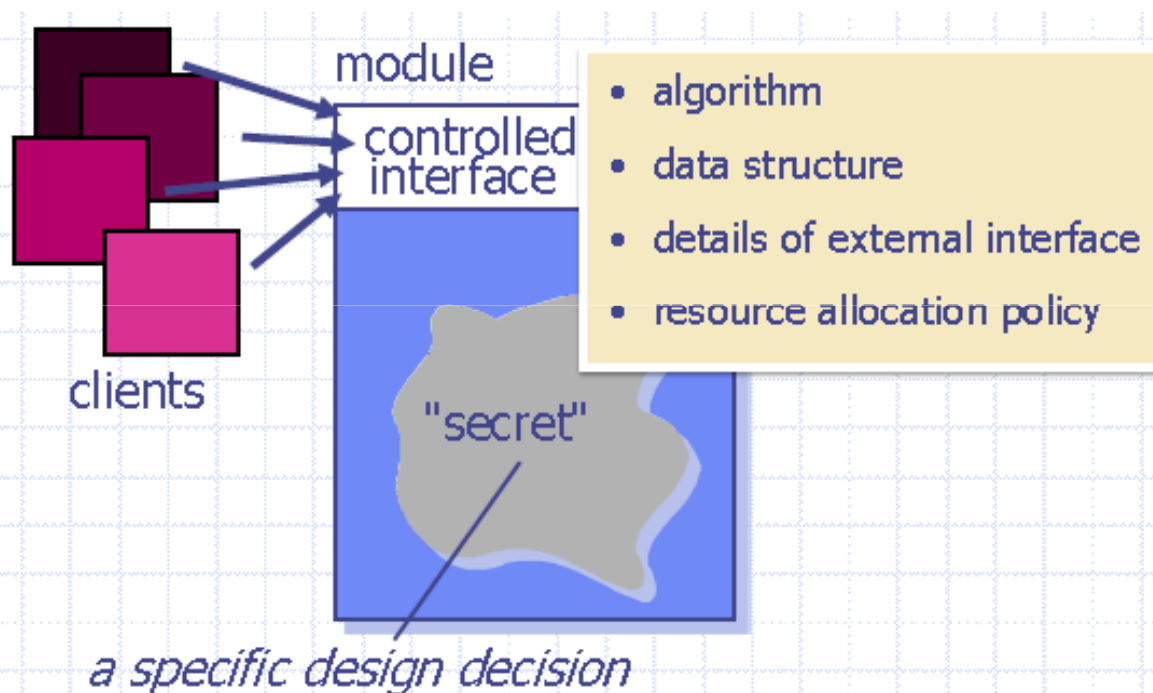


Modularitas

- Berapakah jumlah modul yang pas untuk desain PL tertentu?



Penyembunyian Informasi



Mengapa Informasi disembunyikan?

1. Mengurangi “efek samping”
2. Membatasi pengaruh global dari keputusan desain lokal
3. Menekankan komunikasi melalui interface yang terkendali
4. Mengurangi penggunaan data global
5. Merujuk pada enkapsulasi—sebuah atribut dari desain kualitas tinggi
6. Menghasilkan PL dengan kualitas tinggi

Indepedensi Fungsi

Indepedensi diukur dengan menggunakan 2 kriteria kualitatif

1. Kohesi
2. Coupling

Macam-macam Kohesi

- Coincidnetal
- Logical
- Temporal
- Procedural
- Communicational
- Sequential
- Functional

Jelek/lemah



baik/kuat

Indepedensi Fungsi (lanjutan)

Modul melakukan tugas :

1. Menghitung data suplemen yang didasarkan pada data yang dihitung secara orisinil.
2. Menghasilkan laporan kesalahan pada workstation pemakai.
3. Melakukan kalkulasi follow up yang diminta oleh pemakai.
4. Memperbaharui basis data.
5. Memungkinkan pemilihan menu untuk pemesanan berikutnya.

- **Kohesif Prosedural**

Elemen pemrosesan dari suatu modul dihubungkan dan harus dieksekusi dalam suatu urutan yang spesifik.

- **Kohesi Komunikasional**

Semua elemen pemrosesan berkonsentrasi pada satu area dari suatu struktur data.

Perangkaian

Merupakan : Pengukuran interkoneksi diantara modul-modul pada sebuah struktur program

Heuristik Desain

1. Evaluasi "iterasi pertama" dari struktur program untuk mengurangi perangkaian dan meningkatkan kohesi.
2. Usahakan meminimalkan struktur dengan fan-out yang tinggi ; usahakan untuk melakukan fan-in pada saat kedalaman bertambah.
3. Jaga lingkup efek dari suatu modul ada dalam lingkup kontrol dari modul itu.
4. Evaluasi interface modul untuk mengurangi kompleksitas dan redudansi dan meningkatkan konsistensi

Perangkaian (lanjutan)

5. Tetapkan modul-modul yang fungsinya dapat diprediksi, tetapi hindari modul yang terlalu restriktif.
6. Usahakan modul–modul "entri terkontrol" menghindari "hubungan patologis" dengan
7. Kemaslah PL berdasarkan batasan desain dan persyaratan.

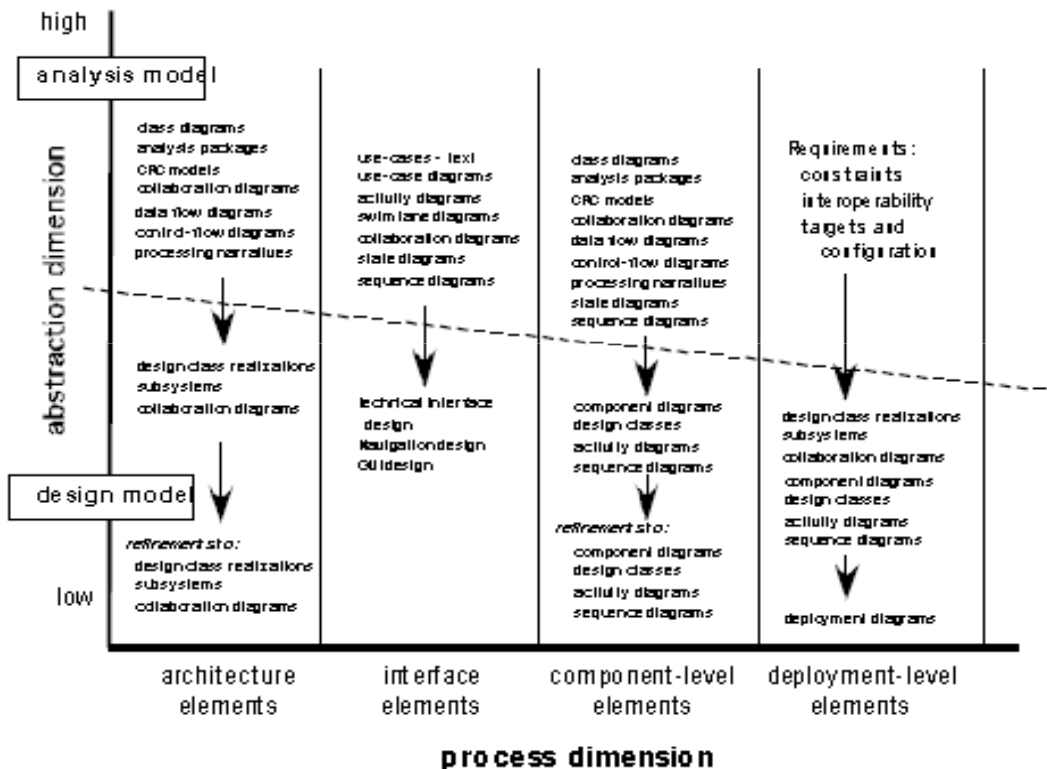
MODEL DESAIN

Direpresentasikan sebagai sebuah piramid.

Konsep Desain OO

- Desain Class
 - Entity classes
 - Boundary classes
 - Controller classes
- Inheritance—semua tanggung jawab superclass akan diwarisi oleh semua subclassnya
- Messages—stimulasi beberapa perilaku yang dapat terjadi pada objek penerima pesan
- Polymorphism—sebuah karakteristik yang mengurangi usaha yang dibutuhkan untuk memperluas desain

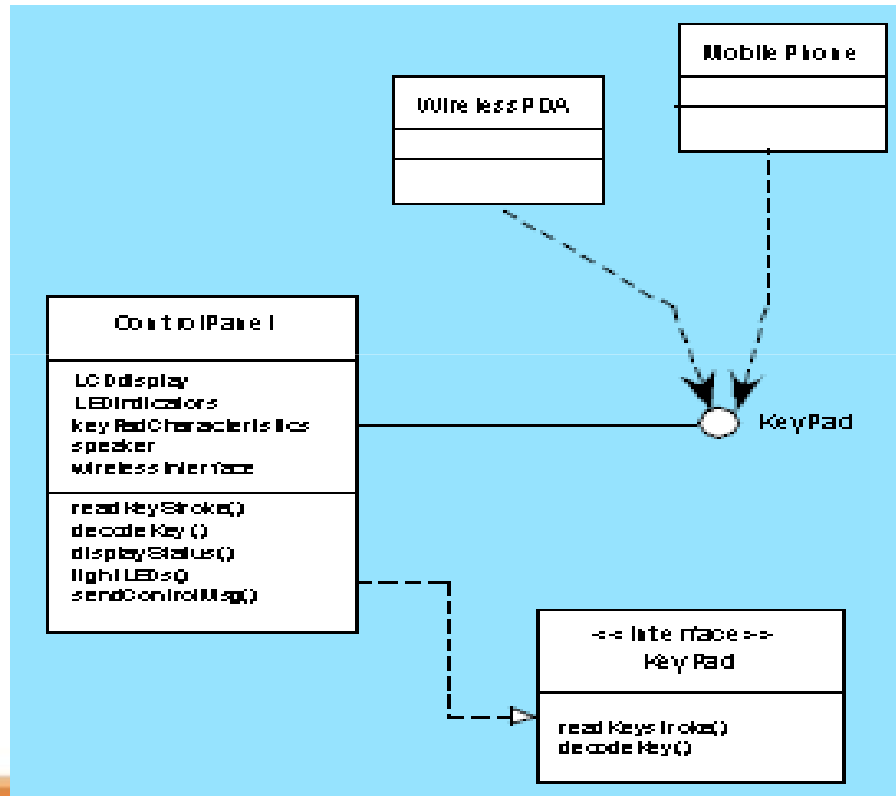
Model Desain



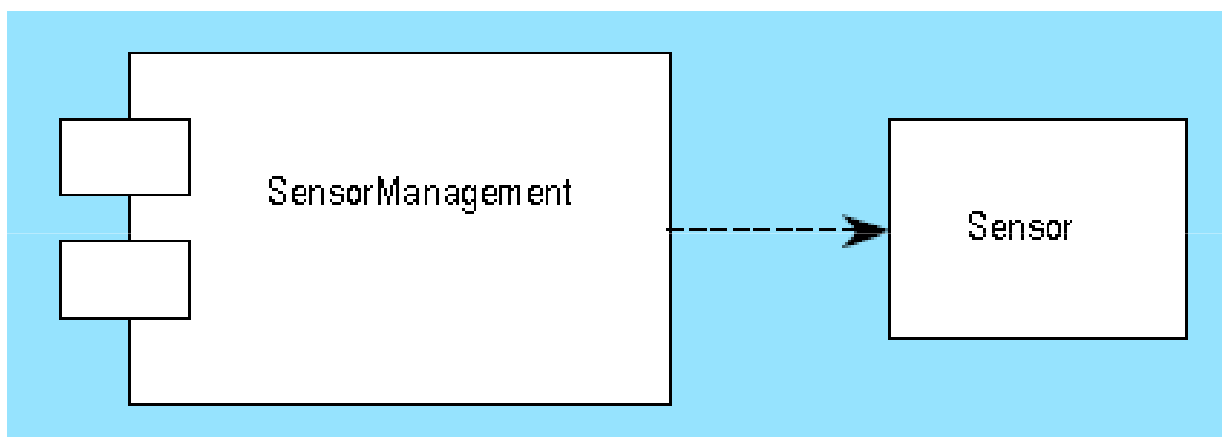
Elemen Model Desain

- Elemen-elemen Data
 - Data model --> struktur data
 - Data model --> arsitektur database
- Elemen-elemen arsitektur
 - Domain aplikasi
 - Class-class analisis, relasinya, kolaborasi dan perilaku diubah menjadi realisasi desain
 - Patterns dan "styles" (Chapter 10)
- Elemen-elemen interface
 - user interface (UI)
 - Interface external pada sistem lain, piranti-piranti, jaringan-jaringan atau produsen maupun konsumen informasi lainnya
 - Interface internal antara komponen-komponen desain.
- Elemen-elemen komponen
- Elemen-elemen deploy

Elemen Interface



Elemen Komponen



Frameworks

- Sebuah framework bukan merupakan pattern arsitektur, namun lebih merupakan kerangka dengan sekumpulan “plug points” (yang juga disebut hooks dan slots) yang memungkinkannya untuk beradaptasi dengan domain permasalahan tertentu.
- Gamma et al mencatat bahwa:
 - ☐ Design patterns adalah elemen-elemen arsitektural yang lebih kecil daripada frameworks
 - ☐ Design patterns lebih umum daripada frameworks

DOKUMENTASI DESAIN

- Ruang lingkup
 - a. sasaran sistem
 - b. persyaratan utama PL
 - c. batasan dan pembatasan desain
- Desain Data
 - a. Obyek dan struktur data resultan
 - b. Struktur file dan database
 - 1. struktur file eksternal
 - 2. data global
 - a. struktur logis
 - b. deskripsi record logis
 - c. metode akses
 - 3. file dan referensi lintas data

DOKUMENTASI DESAIN (lanjutan)

- Desain arsitektural
 - a. Kajian data dan aliran kontrol
 - b. Struktur program yang diperoleh
- Desain interface
 - a. Spesifikasi interface manusia – mesin
 - b. Aturan desain interface manusia – mesin
 - c. Desain interface eksternal
 - 1. Interface untuk data eksternal
 - 2. Interface untuk sistem atau peralatan eksternal
- Desain prosedural
 - Untuk masing-masing model
 - a. Naratif pemrosesan
 - b. Deskripsi interface
 - c. Deskripsi bahasa (atau lainnya) desain

DOKUMENTASI DESAIN (lanjutan)

- c. Deskripsi bahasa (atau lainnya) desain
- d. Modul yang digunakan
- e. Struktur data internal
- f. Keterangan / larangan / pembatasan
- Persyaratan lintas referensi
- Ketentuan Pengujian
 - Panduan pengujian
 - Strategi integrasi
 - Pertimbangan khusus
- Catatan Khusus
- Lampiran

Pertemuan 10

METODE DESAIN (1)

Pokok Bahasan dalam RPL :

- ❖ Desain Data
- ❖ Desain Arsitektur
- ❖ Proses Desain Arsitektur
- ❖ Pasca Pemrosesan Desain
- ❖ Optimasi Desain Arsitektur

Buku Referensi :

- Pressman, RS., 2008, Software Engineering: A Practitioner's Approach, New York: McGraw-Hill
- Sommerville, I, 2007, Software Engineering, Addison Wesley

Tujuan Metode Desain

- Menjelaskan maksud dari arsitektur PL dan kenapa sangat penting.
- Memahami model data, struktur data, database, data warehouse, desain data pada level komponen

DESAIN DATA

- Aktivitas pertama (dan beberapa sering mengatakan yang terpenting) dari 4 aktivitas desain yang dilakukan selama RPL.
- Prinsip-prinsip :
 1. Prinsip analisis sistematis yang diaplikasikan pada fungsi dan perilaku seharusnya diaplikasikan juga pada data.
 2. Semua struktur data dan operasi yang dilakukan pada masing-masing struktur data harus diidentifikasi.
 3. Kamus data harus dibangun dan digunakan untuk menentukan baik data maupun desain program.
 4. Keputusan desain data tingkat rendah harus ditunda sampai akhir proses desain.
 5. Representasi struktur data hanya boleh diketahui oleh modul-modul yang harus menggunakan secara langsung data yang diisikan didalam struktur tersebut.
 6. Pustaka struktur data dan operasi yang berguna yang dapat diaplikasikan pada struktur data tersebut harus dikembangkan.
 7. Desain PL dan bahasa pemrograman harus mendukung spesifikasi dan realisasi dari tipe-tipe data abstrak.

DESAIN ARSITEKTUR

- Untuk mengembangkan struktur program modular dan merepresentasikan hubungan kontrol antar modul.
- Membentuk struktur program dan struktur data dengan menentukan interface yang memungkinkan data mengalir melalui program.
- Kontributor
Perintis desain PL yang didasarkan pada aliran data melalui sebuah sistem.
- Area Aplikasi
luasnya aplikasi dimana aplikasi dapat diaplikasikan.

PROSES DESAIN ARSITEKTUR

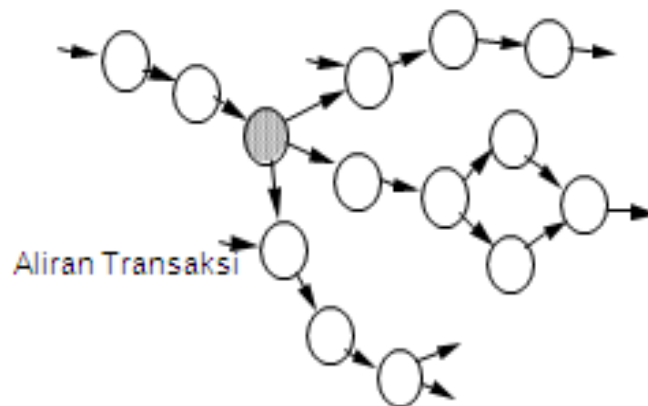
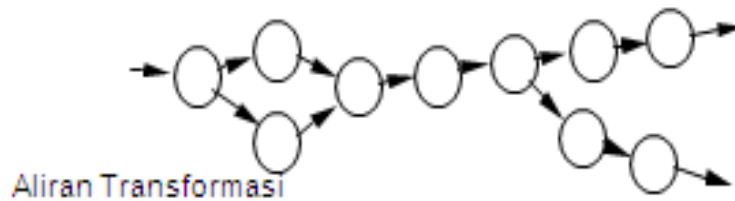
Langkahnya :

1. Tipe aliran informasi dibangun
2. Batas aliran diindikasikan.
3. DFD dipetakan kedalam struktur program.
4. Hierarki kontrol ditentukan dengan pemfaktoran.
5. Struktur resultan disaring/diperhalus dengan menggunakan pengukuran desain dan heuristik.

PROSES DESAIN ARSITEKTUR (lanjutan)

- Aliran Transformasi
Keseluruhan aliran data terjadi dalam cara yang berurutan dan mengikuti satu atau hanya beberapa jalur “garis lurus”, bila segmen dari aliran data menunjukkan karakteristik tersebut, maka disitu ada aliran transformasi.
- Aliran transaksi
Ditandai dengan pergerakan data sepanjang jalur masuk yang mengkonversikan informasi dunia eksternal ke dalam suatu transaksi.

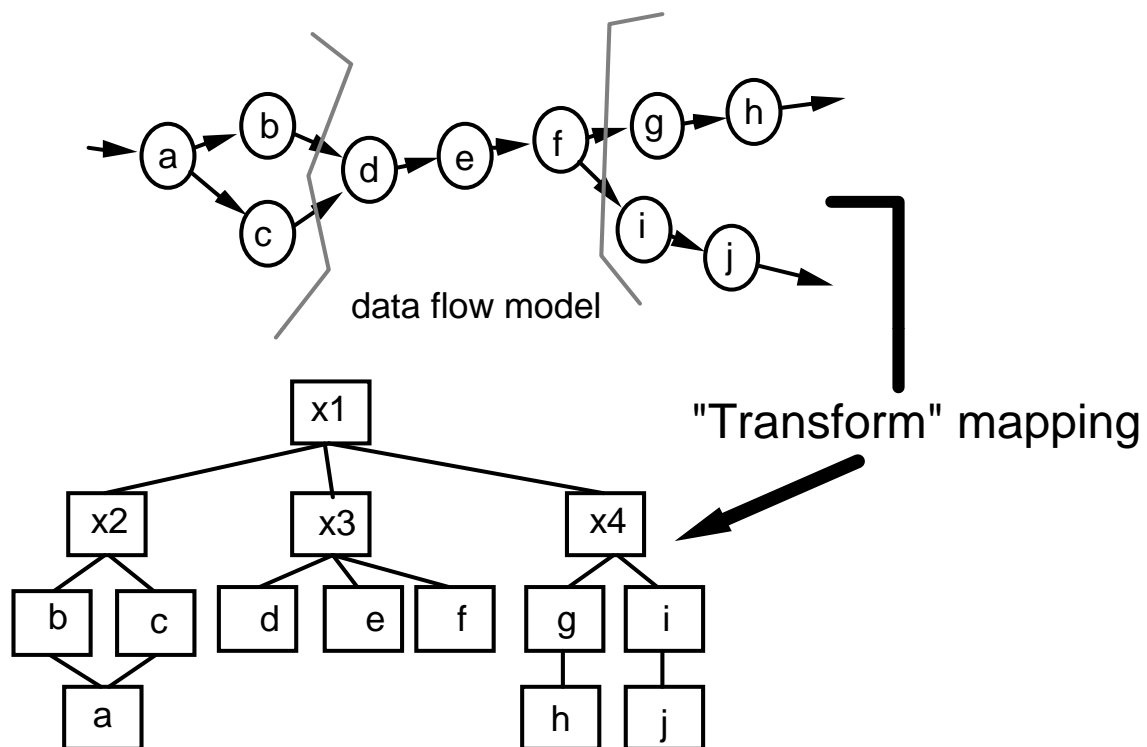
Karakteristik Aliran



PEMETAAN TRANSFORMASI

- Serangkaian langkah desain yang memungkinkan sebuah DFD dengan karakteristik aliran transformasi untuk dipetakan ke dalam tempalte yang telah ditentukan untuk struktur program.
- Langkah-langkah :
 1. Kaji model sistem fundamental.
 2. Kaji dan saring diagram aliran data untuk PL.
 3. Tentukan apakah DFD memiliki karakteristik aliran transformasi atau transaksi.
 4. Isolasi pusat transformasi dengan mengkhususkan batas aliran masuk dan keluar.
 5. Lakukan “pemfaktoran tingkat pertama”.
 6. Lakukan “pemfaktoran tingkat kedua”.
 7. Saringlah struktur program iterasi pertama dengan menggunakan heuristik desain bagi kualitas perangkat lunak yang telah ditingkatkan.

PEMETAAN TRANSFORMASI (LANJUTAN)

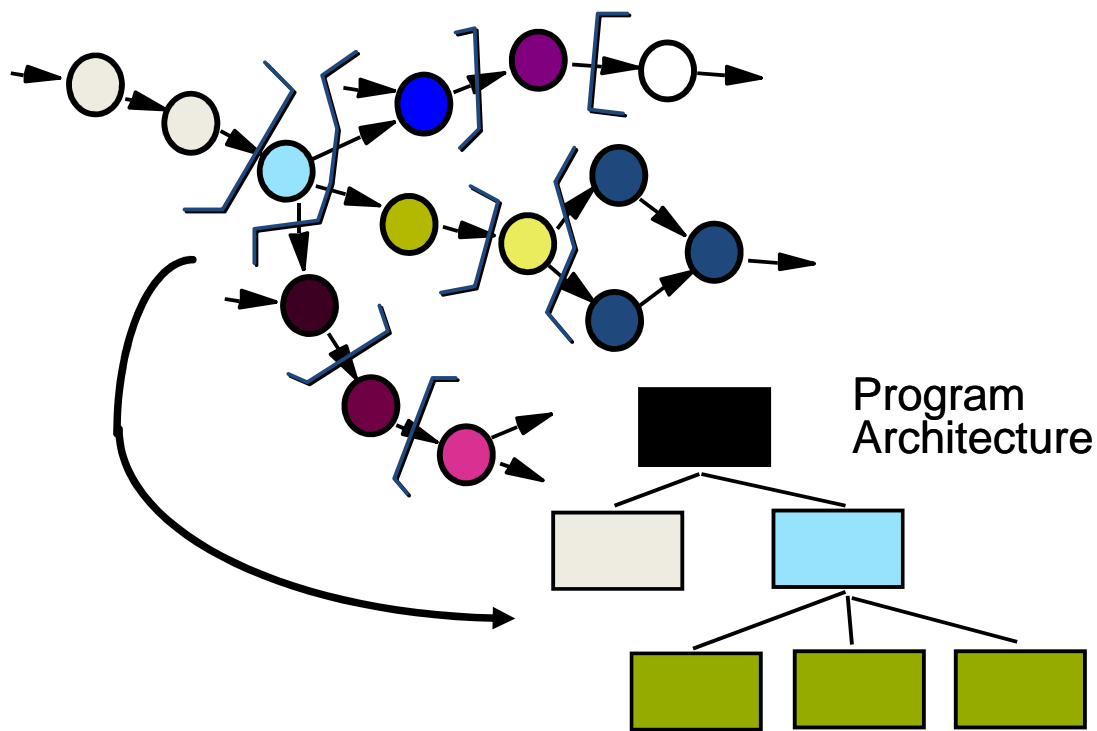


PEMETAAN TRANSAKSI

Langkah-langkah desain :

1. Kaji model sistem fundamental.
2. Kaji dan saring diagram aliran data untuk PL.
3. Tentukan apakah DFD memiliki karakteristik aliran transformasi atau transaksi.
4. Identifikasi pusat transaksi dan karakteristik aliran sepanjang masing-masing jalur aksi.
5. Petakan DFD pada sebuah struktur program yang sesuai dengan pemrosesan transaksi.
6. Faktorkan dan saringkan struktur transaksi dan struktur masing-masing jalur aksi.
7. Saring struktur program iterasi pertama dengan menggunakan heuristik desain untuk kualitas PL yng dikembangkan.

PEMETAAN TRANSAKSI (lanjutan)



PASCA PEMROSESAN DESAIN

Tugas yang harus dilakukan :

- Mengembangkan narasi pemrosesan untuk masing-masing modul.
- Menyediakan deskripsi interface untuk masing-masing modul.
- Menentukan struktur data lokal dan global.
- Mencatat semua batasan desain.
- Mengkaji desain.
- Mempertimbangkan “optimasi” (bila diperlukan dan dibenarkan).

Pendekatan :

1. Kembangkan dan saring struktur program tanpa memperhatikan optimasi kinerja-kritis.
2. Gunakan Peranti CASE yang mensimulasi kinerja run-time untuk mengisolasi area inefisiensi.
3. Selama iterasi desain selanjutnya, pilihlah judul yang dicurigai “time-hot” dan dengan hati-hati kembangkanlah prosedur (algoritma) untuk efisiensi waktu.
4. Kodekan sebuah bahasa pemrograman yang sesuai.
5. Instrumentasikan PL untuk mengisolasi modul yang menjelaskan utilisasi proses yang berat.
6. Bila perlu, desain ulang atau kodekan kembali bahasa yang tergantung pada mesin untuk meningkatkan efisiensi.

Pertemuan 11

METODE DESAIN (2)

Pokok Bahasan dalam RPL :

- ❖ Desain Interface
- ❖ Desain Interface Manusia – Mesin
- ❖ Desain Prosedural
- ❖ Coding

Buku Referensi :

- Pressman, RS., 2008, Software Engineering: A Practitioner's Approach, New York: McGraw-Hill
- Sommerville, I, 2007, Software Engineering, Addison Wesley

Tujuan Metode Desain

- Menjelaskan maksud dari arsitektur PL dan kenapa sangat penting.
- Memahami model data, struktur data, database, data warehouse, desain data pada level komponen

DESAIN INTERFACE

Memfokuskan diri pada 3 area perhatian :

1. Desain interface antara modul-modul PL.
2. Desain interface antara PL dan prosedur dan konsumen informasi, bukan manusia lainnya (yakni entitas eksternal lainnya).
3. Desain interface antara seorang manusia (user) dan komputer.

- **Desain interface pemakai internal**

(desain interface inter-modular) dikendalikan oleh data yang harus mengalir diantara modul-modul dan karakteristik bahasa pemrograman dimana PL akan diimplementasikan.

- **Desain interface pemakai eksternal**

Dimulai dengan evaluasi terhadap masing-masing entitas eksternal yang direpresentasikan pada DFD model analisis.

- **Desain Interface Pemakai**

Berkaitan dengan studi terhadap manusia juga terhadap isu-isu teknologi

DESAIN INTERFACE MANUSIA MESIN

Dimulai dengan membuat model-model fungsi sistem yang berbeda-beda. Kemudian digambarkan tugas yang berorientasi pada manusia dan komputer yang dibutuhkan untuk mencapai fungsi sistem.

Model-Model Desain Interface

1. Model Desain

Menggabungkan data, arsitektur, interface, dan representasi prosedural dari PL

1. Model Pemakai
2. Persepsi Sistem
3. Cara Sistem

DESAIN INTERFACE MANUSIA MESIN (lanjutan)

2. Model Pemakai

Menggambarkan para pemakai akhir dari sistem, meliputi profil, usia, jenis kelamin, kemampuan fisik, pendidikan, latar belakang etnis dan kultural, motivasi, tujuan dan kepribadian.

3. Persepsi Sistem

Citra sistem yang ada dikepala seorang pemakai akhir.

4. Cara Sistem

merangkai manifestasi bagian luar dari sistem berbasis komputer, dengan semua informasi yang mendukung, yang menggambarkan siteksis dan semantik sistem.

DESAIN INTERFACE MANUSIA MESIN (lanjutan)

Pemodelan dan Analisis Tugas-

Proses desain interface :

1. Petakan tujuan untuk serangkaian aksi khusus.
2. Tentukan urutan aksi saat tindakan akan dieksekusi pada tingkat interface.
3. Indikasikan keadaan sistem.
4. Tentukan mekanisme kontrol.
5. Perlihatkan bagaimana mekanisme kontrol mempengaruhi keadaan sistem.
6. Indikasikan bagaimana pemakai menginterpretasi keadaan sistem dari informasi yang diberikan melalui interface.

DESAIN INTERFACE MANUSIA MESIN (lanjutan)

Masalah desain :

1. Apakah help dapat diperoleh untuk semua fungsi sistem .
2. Bagaimana pemakai memperoleh help
3. Bagaimanan help akan direpresentasikan
4. Bagaimana pemakai kembali ke interaksi normal.
5. Bagaimana informasi help distruktur.

DESAIN INTERFACE MANUSIA MESIN (lanjutan)

Peranti Implementasi

User Interface Development :

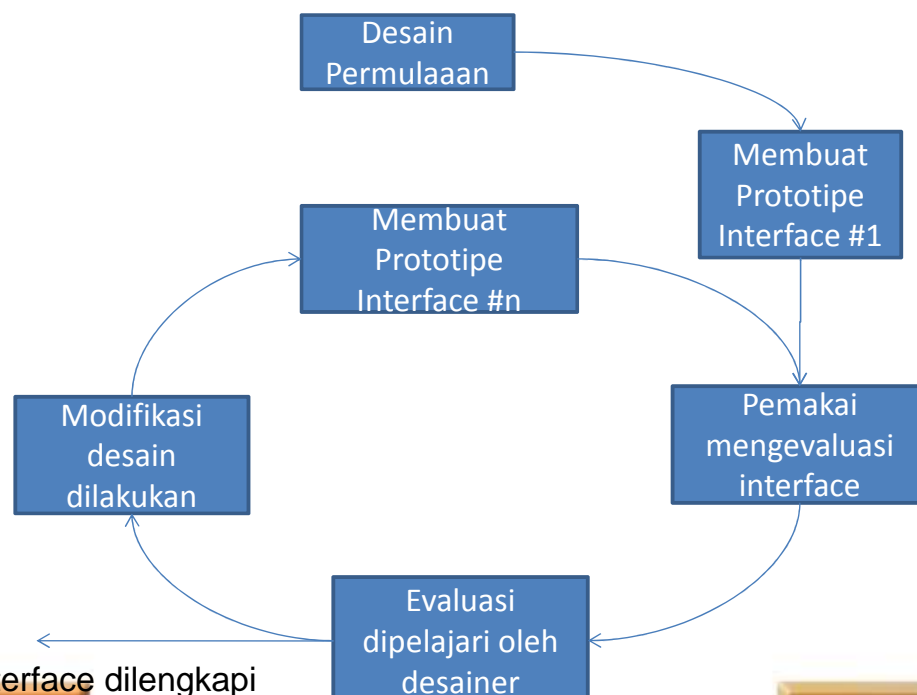
- Mengatur perangkat input (mouse / keyboard)
- Menvalidasi input pemakai.
- Menangani kesalahan dan menampilkan pesan kesalahan.
- Memberikan umpan balik.
- Menyediakan help dan promt.
- Penanganan jendela dan field, scrolling paa jendela.
- Membangun koneksi antara PL aplikasi dan interface.
- Mengisolasi aplikasi dari fungsimanajemen interface.
- Memungkinkan pemakai mengkostumasi interface.

Evaluasi Desain

- Panjang dan kompleksitas spesifikasi tertulis dari sistem dan interfacenya, mengindikasikan jumlah waktu belajar yang dibutuhkan para pemakai sistem.
- Jumlah perintah atau aksi yang ditentukan dan jumlah rata-rata argumen per perintah atau operasi individual per aksi, megindikasikan waktu interaksi dan efisiensi keseluruhan dari sistem tersebut.
- Jumlah aksi, perintah, dan keadaan sistem yang diindikasikan oleh model desain, menunjukkan beban memori pada pemakai sistem.
- Gaya interface, fasilitas help dan protokol penanganan kesalahan memberikan suatu indikasi umum mengenai kompleksitas interface dan tingkat dimana interface akan diterima oleh pemakai.

DESAIN INTERFACE MANUSIA MESIN (lanjutan)

- Siklus evaluasi desain interface



PEDOMAN DESAIN INTERFACE

1. Interaksi Umum

- Konsisten
- berikan umpan balik
- Verifikasi terhadap aksi destruktif yang signifikan.
- kemudahan pembatalan sebagian besar aksi.
- kurangi jumlah informasi yang harus diingat diantara aksi-aksi.
- Adanya efisiensi dalam dialog, gerakan dan pemikiran.
- Memaafkan kesalahan
- Kategorikan aktivitas menurut fungsinya dan atur geografi layar.
- Sediakan fasilitas help yang sensitif.
- Gunakan verbal aksi yang sederhana untuk menerima perintah.

PEDOMAN DESAIN INTERFACE (LANJUTAN)

2. Tampilan Informasi

- Hanya menampilkan informasi yang relevan dengan konteks yang ada.
- Jangan membanjiri pemakai dengan data.
- Gunakan label yang konsisten, penyingkatan standar dan warna yang dapat diprediksi.
- Ijinkan pemakai untuk memelihara konteks visual.
- Hasilkan pesan kesalahan yang berarti.
- Gunakan huruf besar dan kecil, indentasi dan pengelompokkan teks untuk membantu pemahamannya kelompokkan tipe informasi
- Gunakan jendela untuk mengelompokkan tipe informasi yang berbeda.
- Gunakan tampilan “analog” untuk merepresentasikan informasi yang lebih mudah diasimilasikan dengan bentuk representasi ini.
- Pertimbangkan ketersediaan geografi layar tampilan dan gunakan secara efisien.

3. Input Data

- Minimalkan jumlah aksi input yang dibutuhkan dari pemakai.
- Jaga konsistensi diantara tampilan informasi dan input data.
- Ijinkan pemakai mengkustomasi input.
- Interaksi harus fleksibel, tetapi juga diatur ke mode input yang disukai pemakai.
- Nonaktifkan perintah yang tidak sesuai di dalam konteks aksi yang sedang berlangsung.
- Biarkan pemakai mengontrol aliran interkatif.
- Sediakan help untuk membantu semua aksi input.
- Hilangkan input “mickey mouse”.

DESAIN PROSEDURAL

- Terjadi setelah data, desain arsitektur dan interface dibangun.

Pemrograman Terstruktur

Urutan : (langkah pemrosesan yang penting dalam spesifikasi sembarang algoritma).

Kondisi :(fasilitas bagi pemrosesan yang dipilih berdasarkan beberapa kejadian logis).

Pengulangan : (menyediakan looping)

DESAIN PROSEDURAL (lanjutan)

Notasi Desain Grafis

Peranti grafis memberikan bentuk gambar yang bagus yang telah menggambarkan detail prosedural.

Bagan Alir merupakan representasi grafis yang paling luas dipakai.

Notasi Desain Berbentuk Tabel

Tabel keputusan memberikan sebuah notasi yang menerjemahkan aksi-aksi dan kondisi ke dalam bentuk tabel.

DESAIN PROSEDURAL (lanjutan)

Langkah untuk mengembangkan tabel keputusan :

1. Daftarkan semua aksi yang dapat diasosiasikan dengan sebuah prosedur tertentu (atau modul).
2. Daftarkan semua kondisi (atau keputusan yang dibuat) selama eksekusi prosedur.
3. Hubungkan serangkaian kondisi tertentu dengan aksi tertentu, dengan mengeliminasi kombinasi dari kondisi yang mungkin.
4. Tentukan aturan-aturan dengan menunjukkan aksi, apa yang terjadi bagi serangkaian kondisi.

DESAIN PROSEDURAL (lanjutan)

Bahasa Desain Program (PDL) atau pseudocode :

“bahasa pasar” yang menggunakan kosakata dari satu bahasa dan keseluruhan sintaks dari yang lain.

Karakteristik bahasa desain :

1. Sintaks kata kunci (keyword) tersedia untuk semua gagasan terstruktur, deklarasi data dan karakteristik modularitas.
2. Sintaks bebas dari bahasa natural yang menggambarkan ciri-ciri pemrosesan.
3. Fasilitas deklarasi data yang harus meliputi struktur data kompleks (linked list) dan sederhana (skalar, array).
4. Definisi subprogram dan teknik pemanggilan yang mendukung berbagai mode deskripsi interface.

CODING

Pertimbangan Coding

Ada beberapa hal yang perlu dipertimbangkan dalam pelaksanaan coding :

- Rancangan yang dihasilkan : seberapa mudah menerjemahkannya ke dalam bahasa pemrograman.
- Efisiensi komputer (kecepatan dan kebutuhan memori).
- Portabilitas kode program.
- Keberadaan tool-tool pengembangan.
- Kemudahan untuk dipelihara/dirawat.

Pertemuan 12

IMPLEMENTASI

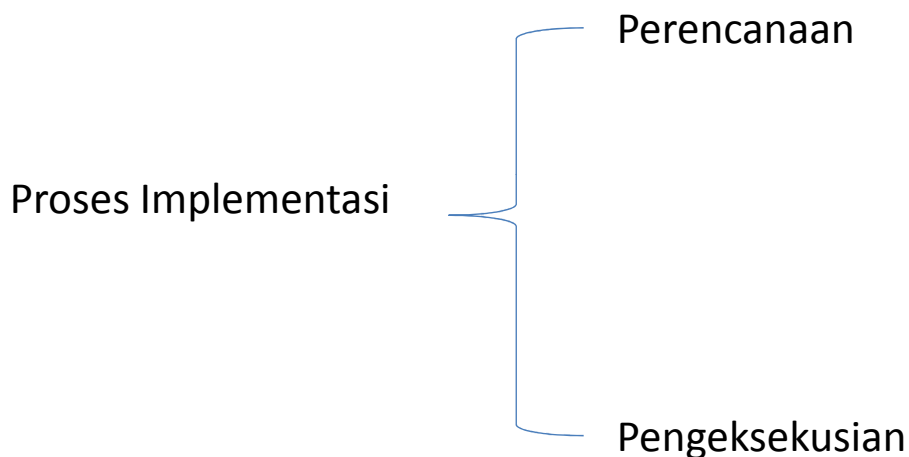
POKOK BAHASAN

- ❖ Makna & Tujuan Implementasi
- ❖ Perencanaan Implementasi
- ❖ Hal Penting Dalam Implementasi
- ❖ Persiapan Dokumentasi
- ❖ Pemasangan Atau Konversi Sistem Baru Ke Sistem Lama
- ❖ Evaluasi Sistem Baru
- ❖ Lingkungan Pemrograman
- ❖ Programming Style
- ❖ Prinsip Portability & Reusable (Kemudahan & Penggunaan
- ❖ Ulang Komponen)
- ❖ CASE Tools

Makna & Tujuan Implementasi (1)

- ☐ Merupakan tahap besar di akhir produksi PL
- ☐ Tahap ini merupakan proses pembuatan kode program berdasarkan platform dan kesepakatan dengan customer.
- ☐ Merupakan tahap transformasi dari hasil desain ke dalam program yang dpt dijalankan pada komputer yang akan digunakan di dalam sistem.
- ☐ Baik buruknya implementasi sangat tergantung pada baik buruknya hasil final dari tahap desain
- ☐ Melibatkan pengintegrasian semua komponen rancangan sistem termasuk PL, konversi ke sistem operasi.

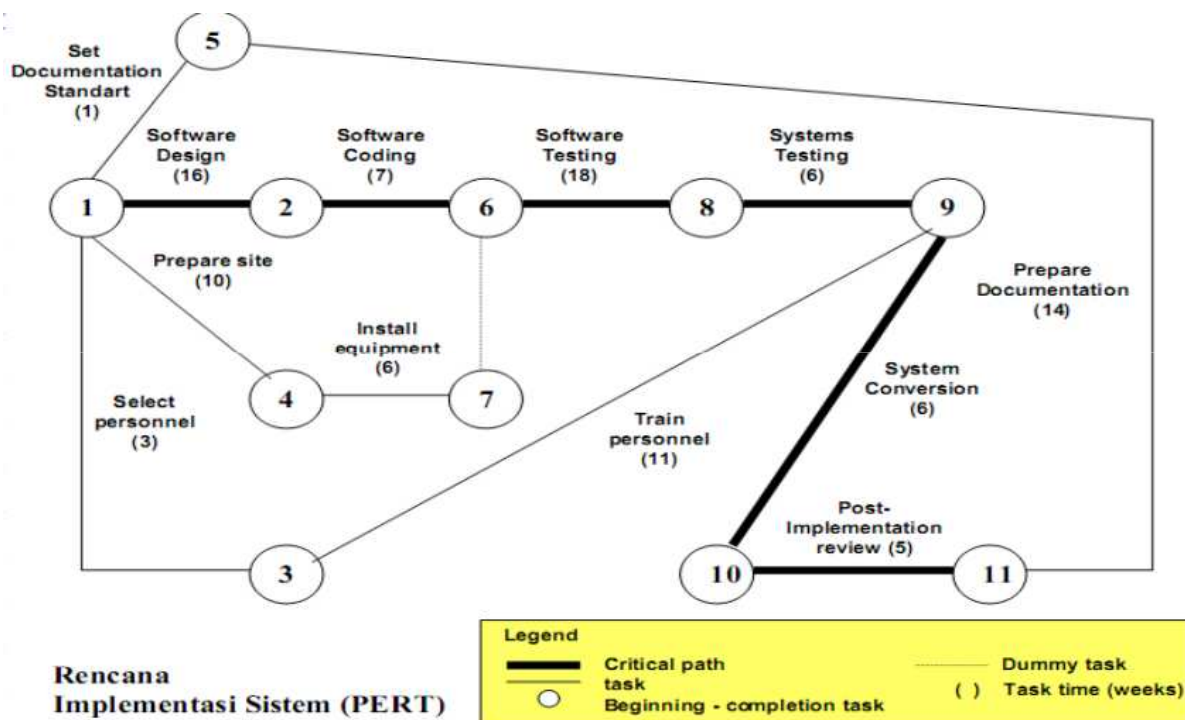
- ☐ Proses implementasi melibatkan:



Rencana Implementasi

- ❖ Merupakan formulasi rinci dan representasi grafik mengenai cara pencapaian implementasian sistem yang akan dilaksanakan
- ❖ Tim implementasi yg terlibat:
 - Manajer dan beberapa staff
 - Profesional sistem yang merancang sistem
 - Perwakilan Vendor
 - Pemakai Primer
 - Pengcode/programmer
 - Teknisi

Contoh Rencana Implementasi



Hal Penting Dalam Implementasi

1. Persiapan Tempat

- ❖ Diperlukan dokumentasi, yang perlu dipersiapkan : (Ruang, listrik , pengujian burn in / simulasi pada vendor)

2. Pelatihan Personil

3. Cakupan Pelatihan

4. Program Pelatihan

5. Teknik dan Alat Bantu Pelatihan

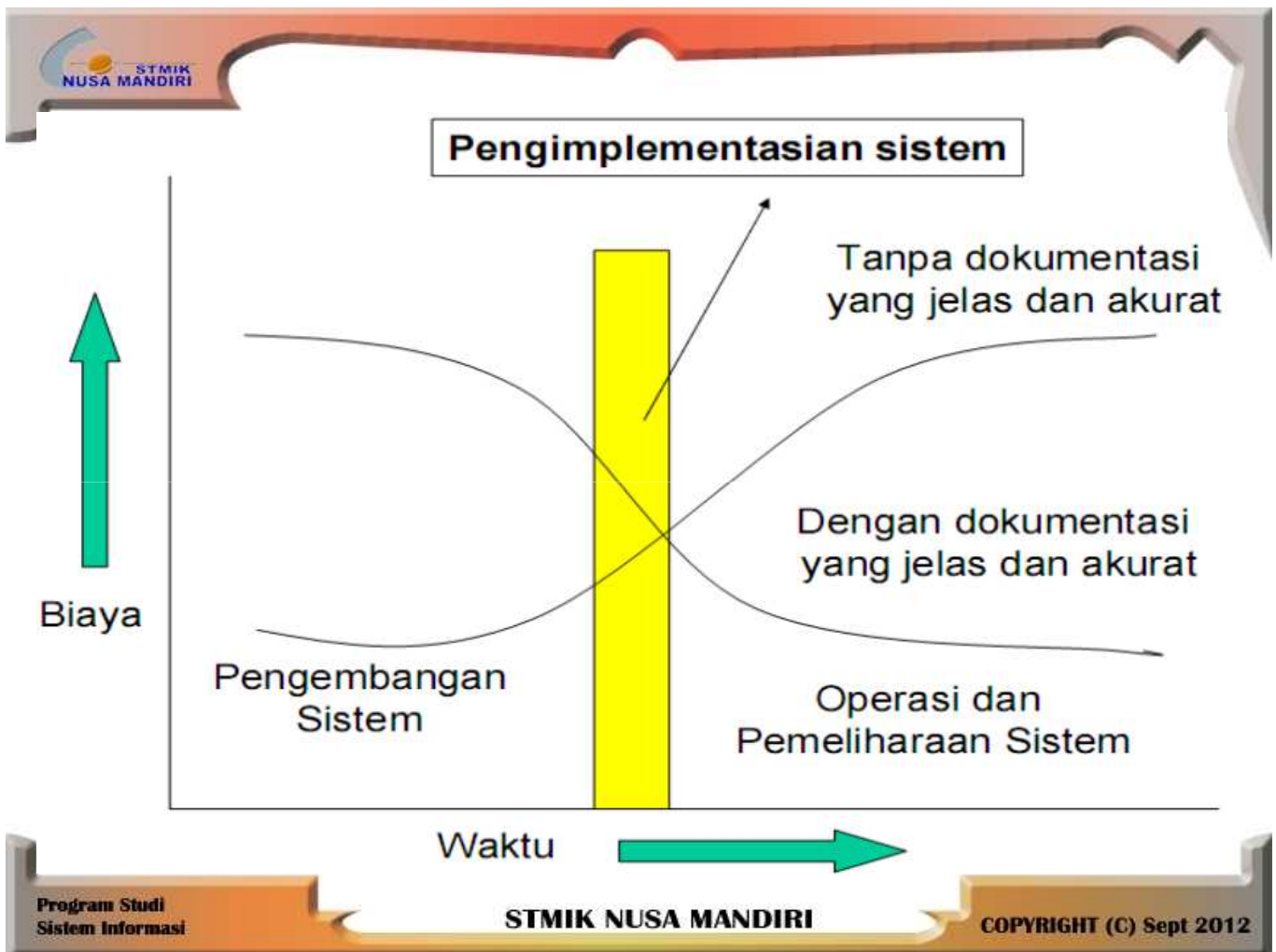
6. Software untuk pelatihan interaktif (cth : CBT, ABT, VBT, VOD)

7. Persiapan / pembuatan dokumen

8. Konversi File & Sistem

Dokumentasi

- ☐ Tujuan dokumentasi:
(Pelatihan, Penginstruksian, Pengkomunikasian, Penetapan standar kinerja, Pemeliharaan sistem ,Referensi historis)
- ☐ Empat area utama dokumentasi:
 1. Dokumen pemakai
 2. Dokumen Sistem
 3. Dokumen Perangkat lunak
 4. Dokumen operasi



Konversi

1. Konversi Langsung

Sistem yang lama langsung digantikan dengan sistem yang baru

2. Konversi Paralel

Sistem lama masih dijalankan sambil menjalankan sistem baru.

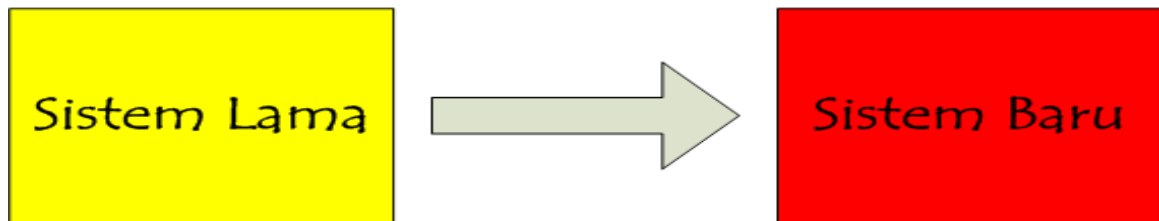
3. Konversi Phase-in

Sistem lama digantikan secara berangsur angsur sedikit demi sedikit.

4. Konversi Pilot

Dilakukan secara segmentasi bagian per bagian

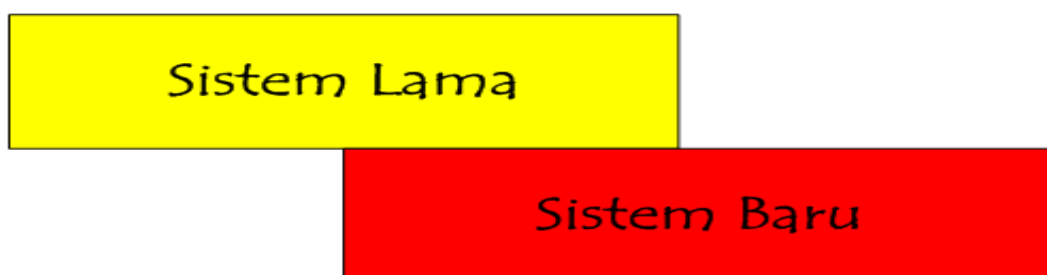
Metode Konversi Langsung



Konversi ini baik dilakukan jika :

- ☑ Sistem baru tidak menggantikan sistem lama
- ☑ Sistem lama sepenuhnya tidak bernilai
- ☑ Sistem baru bersifat kecil/sederhana
- ☑ Rancangan sistem baru sangat berbeda dari sistem lama

Metode Konversi Pararel



- ☒ Memberikan derajat proteksi yang tinggi dari kegagalan sistem baru
- ☒ Biaya yang dibutuhkan cukup besar karena keduanya harus jalan bersama-sama

Metode Konversi Phase-in

Sistem Lama

Sistem Baru

- ☐ Sistem baru di implementasikan sedikit demi sedikit untuk menggantikan sistem lama
- ☐ Sistem harus disegmentasi
- ☐ Perlu biaya tambahan utk membangun interface temporer dg sistem lama
- ☐ Proses implementasi membutuhkan waktu yang panjang

Metode Konversi Pilot

Sistem Lama	Sistem Lama	Sistem Baru
Sistem Lama	Sistem Baru	Sistem Baru

- ☒ Perlunya segmentasi organisasi
- ☒ Resiko lebih rendah dibandingkan metode konversi langsung
- ☒ Biaya lebih rendah dibanding metode paralel
- ☒ Cocok digunakan apabila adanya perubahan prosedur, HardWare, dan SoftWare

Konversi File Data

- ❖ Keberhasilan konversi sistem sangat tergantung pada seberapa jauh profesional sistem menyiapkan konversi file data yang diperlukan di dalam sistem baru
- ❖ **Konversi/Modifikasi Meliputi:**
 - ☐ Format file
 - ☐ Isi file
 - ☐ Media penyimpanan

❖ Metode Dasar Konversi:

1. Konversi File Total

Dpt digunakan pada ke-4 metode konversi sistem

2. Konversi File Gradual

- ☒ Lebih banyak digunakan utk metode paralel dan phase-in
- ☒ Selama konversi file perlu diperhatikan prosedur kendali untuk memastikan integrasi data
- ☒ Prosedur kendali utk masing2 file berbeda
- ☒ Suatu transaksi diterima dan dimasukkan ke dalam sistem
- ☒ Program mencari file master baru untuk record yang akan diupdate oleh transaksi tsb.jika record tsb ada maka pengupdatean record selesai

- ❖ Klasifikasi file:
 - ☐ File Master
 - ☐ File Transaksi
 - ☐ File Index
 - ☐ File Tabel
 - ☐ File Backup

Tahapan Implementasi

- ☒ Struktur dekomposisi, struktur data, dan identitas dipilih dan di kerjakan sampai prosedur desain mudah untuk ditata ulang dalam sebuah implementasi
- ☒ Level abstraksi pada desain, misal class, modul, algoritma, struktur data, dan tipe data harus diwujudkan dalam implementasi
- ☒ Antarmuka antara komponen sistem perangkat lunak harus diwujudkan secara jelas pada tahap implementasi
- ☒ Kode program tersebut harus dapat di cek konsistensinya pada setiap objek dan operasinya secara langsung menggunakan kompilator.

Kriteria Lingkungan Pemrograman

1. Modularity (Modularitas)
2. Dokumentasi Nilai Pada Bahasa Pemrograman
3. Struktur Data Dalam Bahasa Pemrograman
4. Struktur Aliran Pengendali
5. Efisiensi
6. Integritas
7. Portability (multiplatform)
8. Dukungan Dialog
9. Quality
10. Ketersediaan pustaka (Library)
11. Ketersediaan Tool Pembangunan
12. Kebijakan Instansi
13. Kebutuhan Eksternal

Programming Style

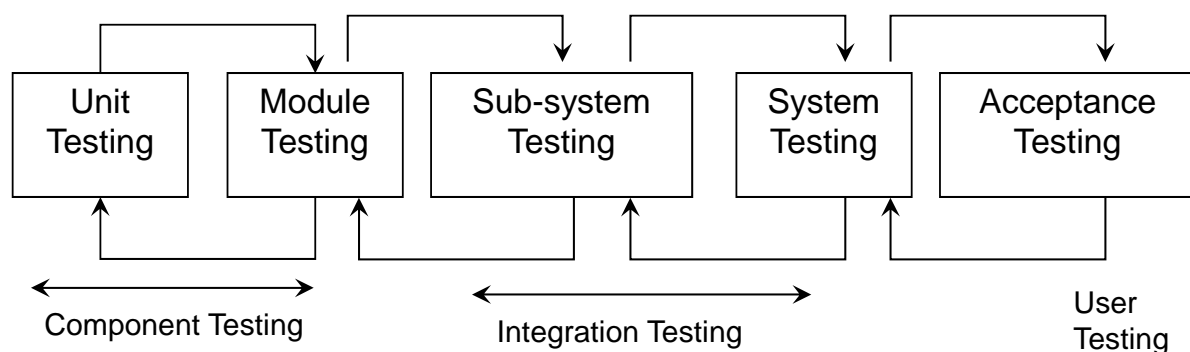
- ❖ Menulis sebuah program adalah seni dan merupakan proses yang kreatif
- ❖ Gaya pemrograman pada programmer mempengaruhi tingkat kemudahan pembacaan program yang dibuatnya
- ❖ Buku Code Complete
Mengulas tuntas suatu gaya pemrograman bahkan di dalamnya diberikan contoh variasi yang cukup banyak.
- ❖ Gaya pemrograman yang baik sangat didukung dari tahap desain dan perencanaan implementasi yang baik

PERTEMUAN 13

STRATEGI PENGUJIAN PERANGKAT LUNAK

Strategi uji coba perangkat lunak dilakukan untuk memudahkan para perancang untuk menentukan keberhasilan system yang telah dikerjakan

Proses Testing



Proses Testing

❖ Unit testing

Pengujian masing-masing unit komponen program untuk meyakinkan bahwa sudah beroperasi secara benar

❖ Module Testing

Pengujian terhadap koleksi unit-unit komponen yang saling berhubungan.

❖ Sub-system Testing

Pengujian terhadap koleksi module-module yang membentuk suatu sub-system (aplikasi)

Proses Testing

❖ System Testing

Pengujian terhadap integrasi sub-system, yaitu keterhubungan antar sub-system

❖ Acceptance Testing

- ☐ Pengujian terakhir sebelum sistem dipakai oleh user.
- ☐ Melibatkan pengujian dengan data dari pengguna sistem.
- ☐ Biasa dikenal sebagai “alpha test” (“beta test” untuk software komersial, dimana pengujian dilakukan oleh potensial customer)

Rencana Pengujian

- ❖ Proses testing
 - Deskripsi fase-fase utama dalam pengujian
- ❖ Pelacakan Kebutuhan
 - Semua kebutuhan user diuji secara individu
- ❖ Item yg diuji
 - Menspesifikasi komponen sistem yang diuji
- ❖ Jadwal Testing
- ❖ Prosedur Pencatatan Hasil dan Prosedur
- ❖ Kebutuhan akan Hardware dan Software
- ❖ Kendala-kendala
 - Mis: kekurangan staff, alat, waktu dll.

Failure and Faults

- ☐ Failure: output yang tidak benar/tidak sesuai ketika sistem dijalankan
- ☐ Fault: kesalahan dalam source code yang mungkin menimbulkan failure ketika code yang fault tersebut dijalankan

Failure Class	Deskripsi
Transient	Muncul untuk input tertentu
Permanent	Muncul untuk semua input
Recoverable	Sistem dapat memperbaiki secara otomatis
Unrecoverable	Sistem tidak dapat memperbaiki secara otomatis
Non-corrupting	Failure tidak merusak data
Corrupting	Failure yang merusak sistem data

Prioritas Testing

- ❖ Hanya test yang lengkap yang dapat meyakinkan sistem terbebas dari kesalahan, tetapi hal ini sangat sulit dilakukan.
- ❖ Prioritas dilakukan terhadap pengujian kemampuan sistem, bukan masing-masing komponennya.
- ❖ Pengujian untuk situasi yang tipikal lebih penting dibandingkan pengujian terhadap nilai batas.

Test Data Dan Kasus Test

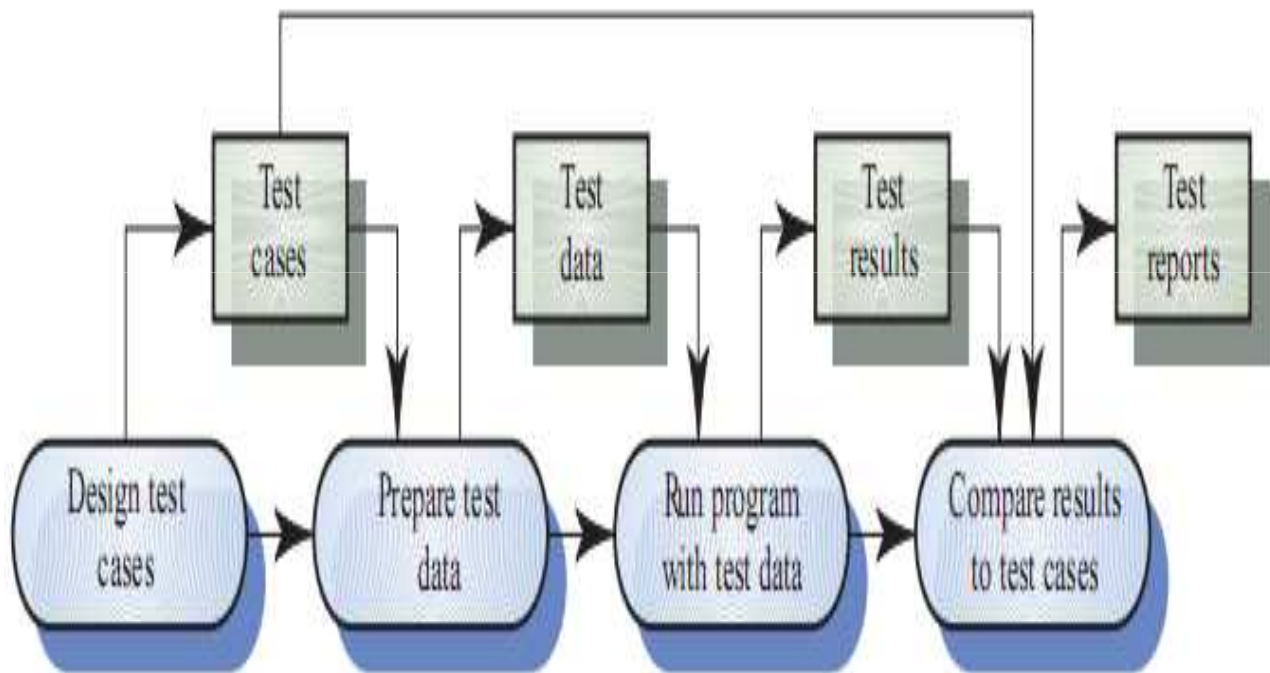
☐ Test Data

Input yang direncanakan digunakan oleh sistem.

☐ Test Cases

Input yang digunakan untuk menguji sistem dan memprediksi output dari input jika sistem beroperasi sesuai dengan spesifikasi.

Proses defect testing



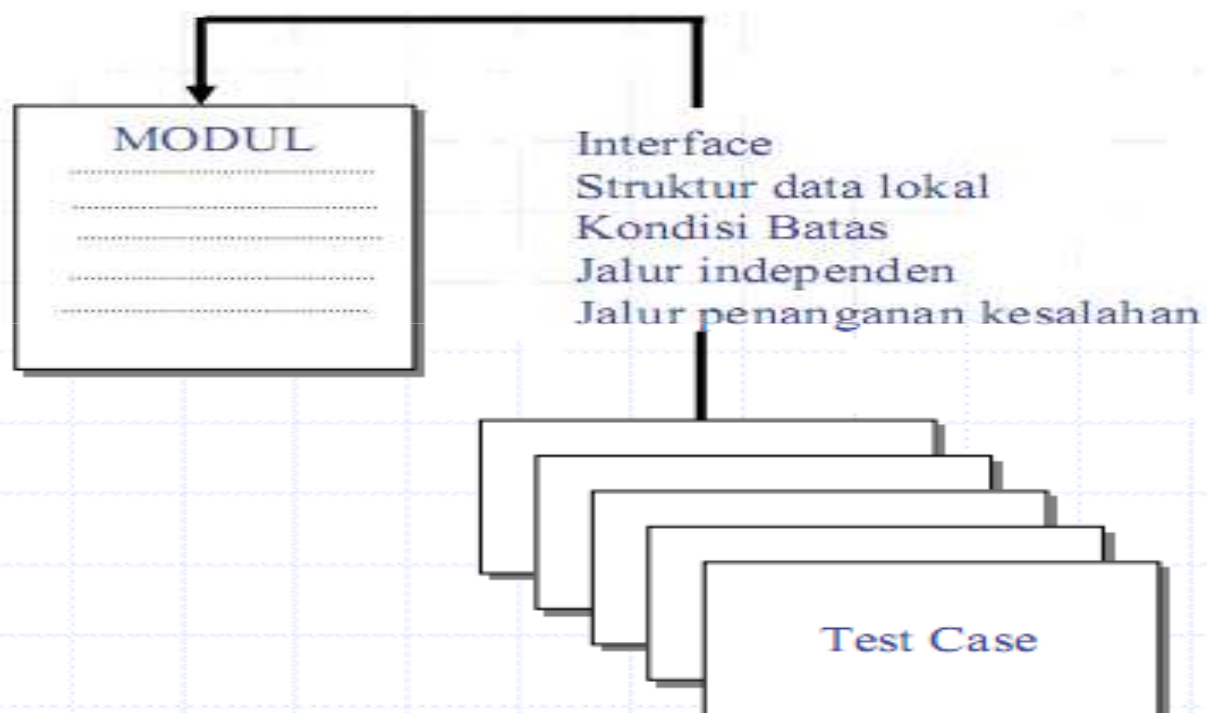
Pendekatan Strategis Pengujian Perangkat Lunak

- ❖ Pengujian Unit
- ❖ Pengujian Integrasi
- ❖ Pengujian Validasi
- ❖ Pengujian Sistem

Pengujian Unit

- ☐ Berfokus pada inti terkecil dari desain perangkat lunak yaitu modul
- ☐ Uji coba unit selalu berorientasi pada white box testing
- ☐ Dapat dikerjakan paralel atau beruntun dengan modul lainnya.

Pengujian Unit (2)



Pengujian Unit (3)

- ☐ Apakah jumlah parameter input sama dengan jumlah argumen?
- ☐ Apakah antara atribut dan parameter argumen sudah cocok?
- ☐ Apakah antara sistem satuan parameter dan argumen sudah cocok?
- ☐ Apakah jumlah argumen yang ditransmisikan ke modul yang dipanggil sama dengan atribut parameter?
- ☐ Apakah atribut dari argumen yang ditransmisikan ke modul yang dipanggil sama dengan atribut parameter?
- ☐ Apakah sistem unit dari argumen yang ditransmisikan ke modul yang dipanggil sama dengan sistem satuan parameter?

Pengujian Unit (4)

- ☐ Apakah jumlah atribut dan urutan argumen ke fungsi-fungsi built-in sudah benar?
- ☐ Adakah referensi ke parameter yang tidak sesuai dengan poin entri yang ada?
- ☐ Apakah argumen input only diubah?
- ☐ Apakah definisi variabel global konsisten dengan modul ?
- ☐ Apakah batasan yang dilalui merupakan argumen?

Pengujian Unit (5)

- ☐ Test case harus didesain untuk mengungkap kesalahan dalam kategori
 1. Pengetikan yang tidak teratur dan tidak konsisten inisialisasi yang salah atau nilai-nilai default
 2. Nama variabel yang tidak benar
 3. Tipe data yang tidak konsisten
 4. Underflow, overflow dan pengecualian pengalamatan

Seberapa Baik Sistem Yang Sudah di Bangun

❖ Dua Aspek yang dipertimbangkan:

- ☐ Apakah implementasi sudah sesuai dengan spesifikasi ?
- ☐ Apakah spesifikasi sesuai dengan kebutuhan user ?

❖ Validasi

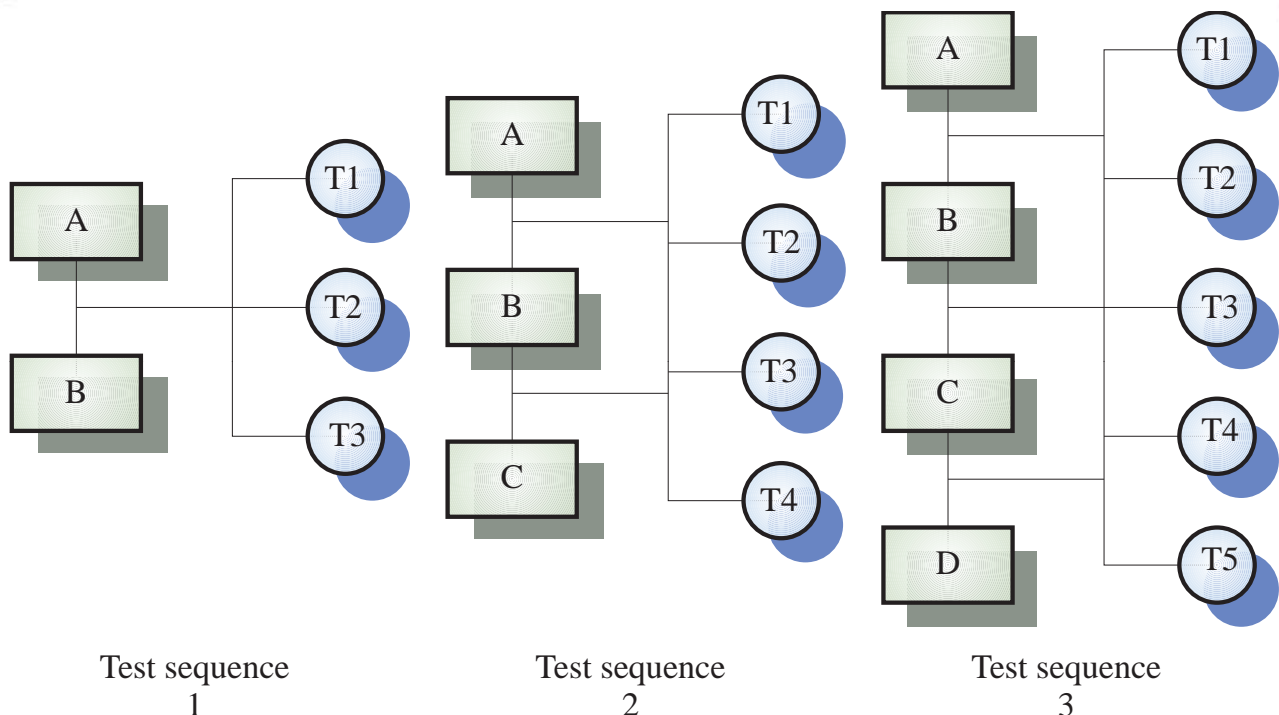
- ☐ Apakah sistem yang dikembangkan sudah benar?
- ☐ Pengujian dimana sistem ketika diimplementasikan sesuai dengan yang diharapkan

❖ Verifikasi

- ☐ Apakah sistem dikembangkan dengan cara yang benar ?
- ☐ Pengujian apakah sistem sudah sesuai dengan spesifikasi

Integration Testing

- ❑ Pengujian keseluruhan system atau sub-system yang terdiri dari komponen yang terintegrasi.
- ❑ Test integrasi menggunakan black-box dengan test case ditentukan dari spesifikasi.
- ❑ Kesulitannya adalah menemukan/melokasikan
- ❑ Penggunaan Incremental integration testing dapat mengurangi masalah tersebut.



Pendekatan Integration Testing

❖ Top-down Testing

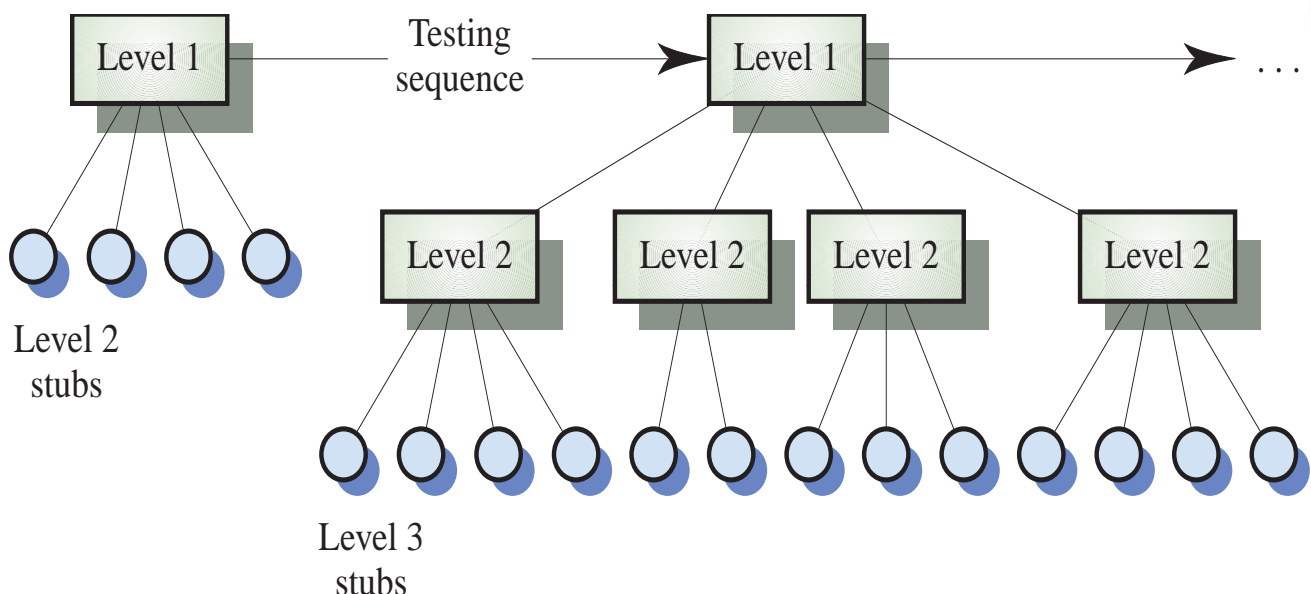
Berawal dari level-atas system dan terintegrasi dengan mengganti masing-masing komponen secara top-down dengan suatu stub (program pendek yg generate input ke sub-system yang diuji).

❖ Bottom-up Testing

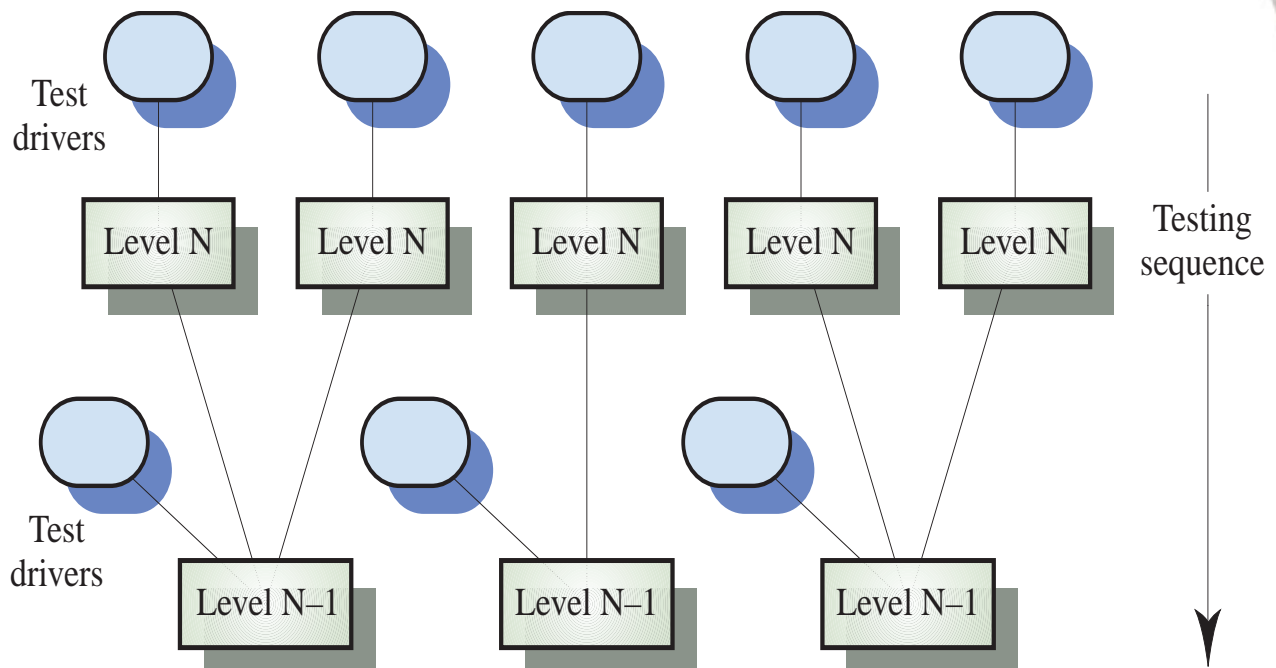
Integrasi components di level hingga sistem lengkap sudah teruji.

Pada prakteknya, kebanyakan test integrasi menggunakan kombinasi kedua strategi pengujian tsb.

Top Down Testing



Bottom Up Testing



Pendekatan Testing

❖ Architectural Validation

Top-down integration testing lebih baik digunakan dalam menemukan error dalam sistem arsitektur.

❖ System Demonstration

Top-down integration testing hanya membatasi pengujian pada awal tahap pengembangan system.

❖ Test Implementation

Seringkali lebih mudah dengan menggunakan bottom-up integration testing

Interface Testing

- ☐ Dilakukan kalau module-module dan sub-system terintegrasi dan membentuk sistem yang lebih besar
- ☐ Tujuannya untuk medeteksi fault terhadap kesalahan interface atau asumsi yang tidak valid tentang interface tersebut.
- ☐ Sangat penting untuk pengujian terhadap pengembangan sistem dengan menggunakan pendekatan object-oriented yang didefinisikan oleh object-objectnya

Pengujian Validasi

❖ Kajian Konfigurasi (audit)

- ☐ Elemen dari proses validasi
- ☐ Memastikan apakah semua elemen konfigurasi perangkat lunak telah dikembangkan dengan tepat

❖ Pengujian Alpha dan Beta

- ☐ Pengujian Alpha
 - ✓ Usability labs
 - ✓ Usability factors checklist
- ☐ Pengujian Beta

Pengujian Sistem :

- ☐ Pengujian Perbaikan
- ☐ Pengujian Keamanan
- ☐ Pengujian Stress
- ☐ Pengujian Kinerja

Pengujian Aplikasi Server :

- ☐ Volume Testing
- ☐ Stress Testing
- ☐ Performance Testing
- ☐ Data Recovery Testing
- ☐ Data Backup and Restore Testing
- ☐ Data Security Testing

Volume Testing

- ☐ Menemukan kelemahan sistem selama melakukan pemrosesan data dalam jumlah yang besar dalam periode waktu yang singkat.
- ☐ Tujuan: meyakinkan bahwa sistem tetap melakukan pemrosesan data antar batasan fisik dan batasan logik.
- ☐ Contoh:
Menguji proses antar server dan antar partisi hardisk pada satu server.

Stress Testing

- ☐ Tujuan: mengetahui kemampuan sistem dalam melakukan transaksi selama periode waktu puncak proses.
- ☐ Contoh periode puncak: ketika penolakan proses login on-line setelah sistem down atau pada kasus batch, pengiriman batch proses dalam jumlah yang besar dilakukan setelah sistem down.
- ☐ Contoh: Melakukan login ke server ketika sejumlah besar workstation melakukan proses menjalankan perintah sql database.

Performance Testing

- ☐ Dilakukan secara paralel dengan Volume dan Stress testing untuk mengetahui unjuk kerja sistem (waktu respon, throughput rate) pada beberapa kondisi proses dan konfigurasi.
- ☐ Dilakukan pada semua konfigurasi sistem perangkat keras dan lunak. Misal : pada aplikasi Client-Server diujikan pada kondisi korporate ataupun lingkungan sendiri (LAN vs. WAN, Laptop vs. Desktop)
- ☐ Menguji sistem dengan hubungannya ke sistem yang lain pada server yang sama.
- ☐ Load Balancing Monitor
- ☐ Network Monitor

Data Recovery Testing

- ☐ Investigasi dampak kehilangan data melalui proses recovery ketika terjadi kegagalan proses.
- ☐ Penting dilakukan karena data yg disimpan di server dapat dikonfigurasi dengan berbagai cara.
- ☐ Kehilangan Data terjadi akibat kegagalan sistem, hardisk rusak, penghapusan yang tidak sengaja, kecelakaan, virus dan pencuri.

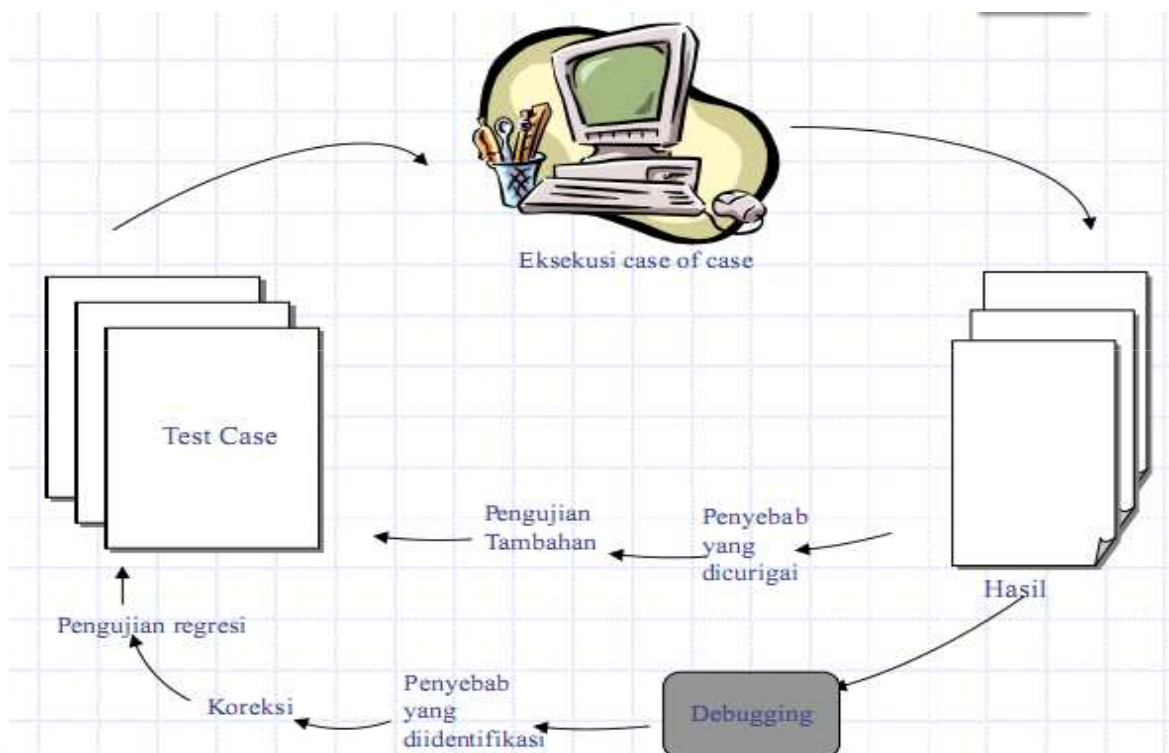
Data Backup dan Restore Testing

- ☐ Dilakukan untuk melihat prosedur back-up dan recovery.
- ☐ Diakukan dengan mensimulasikan beberapa kesalahan untuk menguj i proses backup dan recovery.
- ☐ Pengujian dilakukan terhadap strategi backup: frekuensi , medium, waktu, mekanisme backup (manual/ otomatis), personal, ? Berapa lama backup akan disimpan.
- ☐ Switching antara live dan backup server ketika terjadi kerusakan (load log transaction pada back-up kemudian melakukan recovery).

Data Security Testing

- ❑ *Privilege access* terhadap *database* diujikan pada beberapa user yang tidak memiliki *privilege access* ke *database*.
- ❑ *Shutdown database engine* melalui *operating system* (dengan beberapa perintah OS) yang dapat mematikan aplikasi *database*.

Debugging



PERTEMUAN 14

TEKNIK PENGUJIAN PERANGKAT LUNAK

TESTING

- ☐ Pengujian perangkat lunak adalah proses menjalankan dan mengevaluasi sebuah perangkat lunak secara manual maupun otomatis untuk menguji apakah perangkat lunak sudah memenuhi persyaratan atau belum, atau untuk menentukan perbedaan antara hasil yang diharapkan dengan hasil sebenarnya.
- ☐ Pengujian merupakan suatu tahapan pengerjaan yang bertujuan mencari kesalahan program. Kesalahan yang terjadi selama proses pengembangan perangkat lunak akan mengakibatkan bertambahnya waktu untuk menyelesaikan pekerjaan tersebut.
- ☐ Pengujian hendaknya dilakukan pada setiap tahap pengembangan yaitu mulai dari tahap analisis kebutuhan sampai dengan tahap perawatan.

KENAPA HARUS DIUJI ?

- Kita bukan seorang programmer yang cukup baik
- Kita mungkin tidak dapat cukup berkonsentrasi untuk menghindari kesalahan
- Kita terkadang lupa menggunakan pemrograman terstruktur secara penuh, perancangan atas-bawah untuk mendapatkan solusi
- Kita kadang buruk dalam mengerjakan sesuatu
- Kita seharusnya dapat membedakan apa yang dikatakan programmer lain atau pelanggan dan apa yang sebenarnya mereka pikirkan
- Kita seharusnya merasa bersalah apabila seseorang harus menguji koding kita
- Pengujian merupakan suatu perizinan terhadap kesalahan

PRINSIP PENGUJIAN

Beberapa prinsip pengujian yang harus diperhatikan (diusulkan oleh Davis):

1. Semua pengujian harus dapat ditelusuri sampai ke persyaratan pelanggan.
2. Pengujian harus direncanakan lama sebelum pengujian itu dimulai.
3. Prinsip Pareto berlaku untuk pengujian PL.
4. Pengujian harus mulai "dari yg kecil" s/d "yang besar".
5. Pengujian yg mendalam tidak mungkin.
6. Paling efektif, pengujian dilakukan oleh pihak ketiga yang independen

SASARAN PENGUJIAN

- Glen Mayers menyatakan sejumlah aturan yang dapat dipandang sebagai sasaran dari pengujian
 - ❖ Pengujian perangkat lunak adalah suatu proses pengeksekusian program dengan tujuan menemukan kesalahan (*error*)
 - ❖ Pengujian (*Test case*) yang baik adalah yang mempunyai probabilitas yang tinggi untuk menemukan error yang tak diketemukan
 - ❖ Pengujian yang sukses adalah pengujian yang dapat menemukan kesalahan (*error*) yang tidak ditemukan sebelumnya

TUJUAN PENGUJIAN

Tujuan yang diinginkan dari pelaksanaan pengujian perangkat lunak adalah :

- Menilai apakah perangkat lunak yang dikembangkan telah memenuhi kebutuhan pemakai.
- Menilai apakah tahap pengembangan perangkat lunak telah sesuai dengan metodologi yang digunakan.
- Membuat dokumentasi hasil pengujian yang menginformasikan kesesuaian perangkat lunak yang diuji dengan spesifikasi yang telah ditentukan.

TEST ABILITAS

Testabilitas Perangkat Lunak adalah seberapa mudah sebuah program komputer dapat diuji. Karena pengujian sangat sulit, perlu diketahui apa yang dapat dilakukan untuk membuatnya menjadi mudah.

Karakteristik PL yang Di Uji

- ☐ **OPERABILITAS**
- ☐ **OBSERVABILITAS**
- ☐ **KONTROLABILITAS**
- ☐ **DEKOMPOSABILITAS**
- ☐ **KESEDERHANAAN**
- ☐ **STABILITAS**
- ☐ **KEMAMPUAN DIPAHAMI**

ATRIBUT PENGUJIAN YANG BAIK

- ❖ Memiliki probabilitas yg tinggi menemukan kesalahan.
- ❖ Tidak redundan.
- ❖ Harusnya 'jenis terbaik'.
- ❖ Tidak boleh terlalu sederhana atau terlalu kompleks

DESAIN TEST CASE

Metode Desain Test case menyediakan pendekatan sistematis untuk uji coba. Dan menyediakan kemungkinan yang cukup tinggi untuk menemukan kesalahan.

2 macam test case :

- 1. Pengetahuan tentang fungsi yang spesifik dari produk yang telah dirancang untuk diperlihatkan,** test dapat dilakukan untuk menilai masing-masing fungsi apakah telah berjalan sebagaimana yang diharapkan.
- 2. Pengetahuan tentang cara kerja dari produk,** test dapat dilakukan untuk memperlihatkan cara kerja dari produk secara rinci sesuai dengan spesifikasinya.

PERANCANGAN TEST CASE

Dua macam pendekatan test yaitu :

1. Black Box Testing

Test case ini bertujuan untuk menunjukkan fungsi Perangkat Lunak tentang cara beroperasinya, apakah pemasukan data keluaran telah berjalan sebagaimana yang diharapkan dan apakah informasi yang disimpan secara eksternal selalu dijaga kemutakhirannya.

2. White Box Testing (Structural Testing)

Adalah meramalkan cara kerja perangkat lunak secara rinci, karenanya logikal path (jalur logika) perangkat lunak akan dites dengan menyediakan test case yang akan mengerjakan kumpulan kondisi dan atau pengulangan secara spesifik. Secara sekilas dapat diambil

WHITE BOX TESTING

1. Basis Path Testing

- Metode ini memungkinkan perancang test case mendapatkan ukuran kekompleksan logikal dari perancangan prosedural dan menggunakan ukuran ini sebagai petunjuk untuk mendefinisikan basis set dari jalur pengerjaan.
- Test case yang didapat digunakan untuk mengerjakan basis set yang menjamin pengerjaan setiap perintah minimal satu kali selama uji coba.
- Contoh dari Basis Path Testing :
 - ☐ Notasi Diagram Alir
 - ☐ Cyclomatic Complexity
 - ☐ Graph Metrix

WHITE BOX TESTING (2)

2. Loop Testing

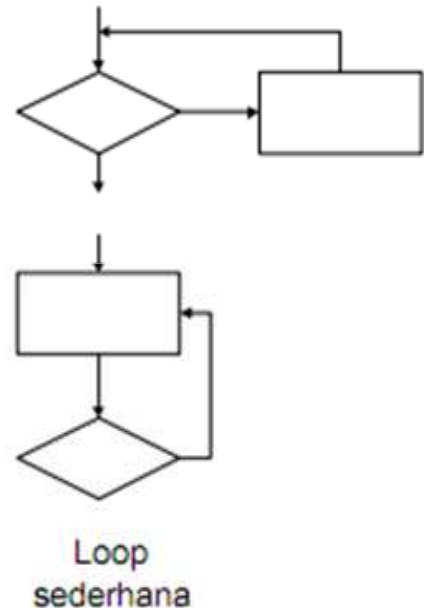
☐ Loop Sederhana

Pengujian loop sederhana dilakukan dgn mudah, dimana n jumlah maksimum yg diijinkan melewati loop tsb.

1. Lewati loop secara keseluruhan
2. Hanya satu yg dapat melewati loop
3. m dapat melewati loop dimana $m < n$

☐ Loop terangkai

☐ Loop tidak terstruktur

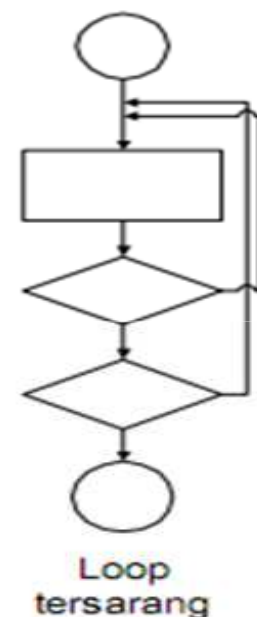


WHITE BOX TESTING (3)

☐ Loop Tersarang

Pengujian loop ini menggunakan pendekatan loop sederhana. Petunjuk pengujian loop tersarang :

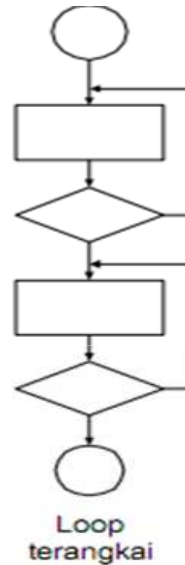
- Dimulai dari loop paling dalam dan atur semua loop ke nilai minimum.
- Kerjakan dgn prinsip loop sederhana untuk loop yg paling dalam sementara tahan loop yg di luar pada parameter terkecil (nilai kounter terkecil)
- Kemudian lanjutkan untuk loop yg di atasnya.
- Teruskan sampai semua loop selesai di uji.



WHITE BOX TESTING (4)

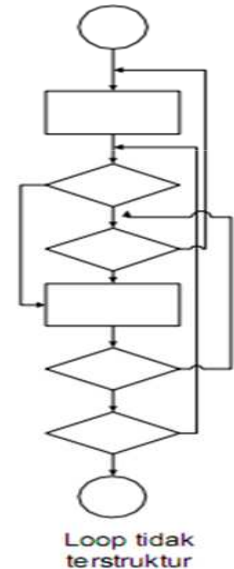
□ Loop Terangkai

Pengujian loop ini menggunakan pendekatan loop sederhana. Bila masing-masing loop independen, tetapi bila dua loop dirangkai dan pencacah loop 1 digunakan sebagai harga awal loop 2 maka loop tsb jadi tidak independen, dan di rekomendasikan ke loop tersarang



□ Loop Tidak Terstruktur

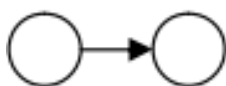
Kapan saja memungkinkan, loop ini didisain kembali agar mencerminkan penggunaan komposisi pemrograman terstruktur.



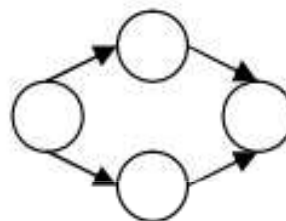
BASIS PATH TESTING

1. Notasi Diagram Alir (Program Flow Graph)

Sequence



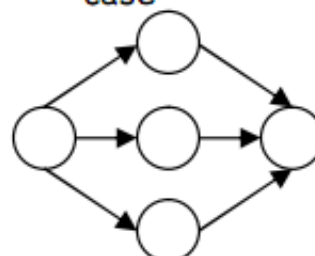
if



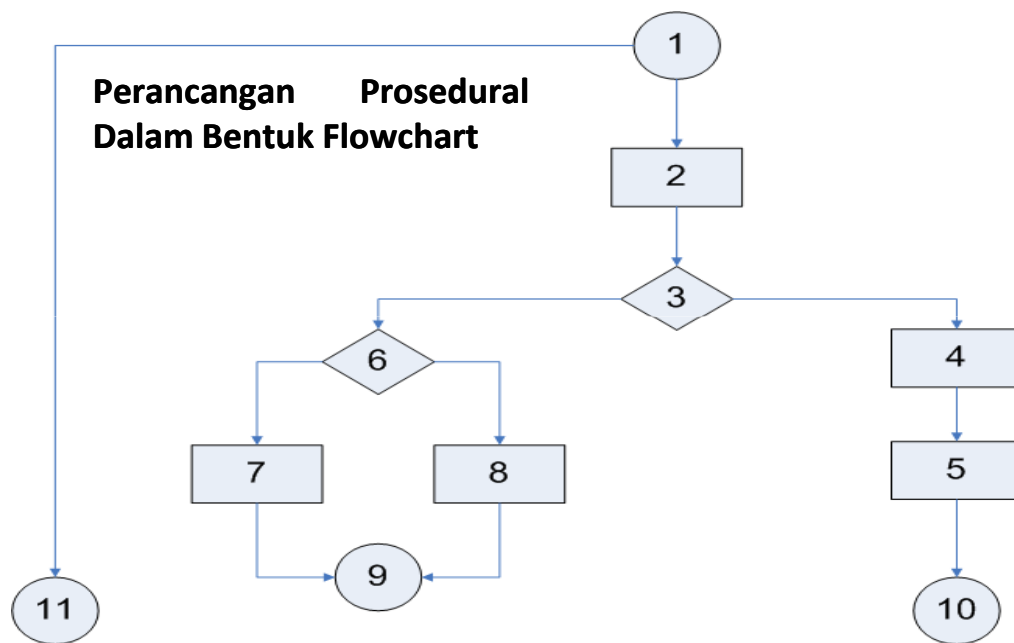
while



case

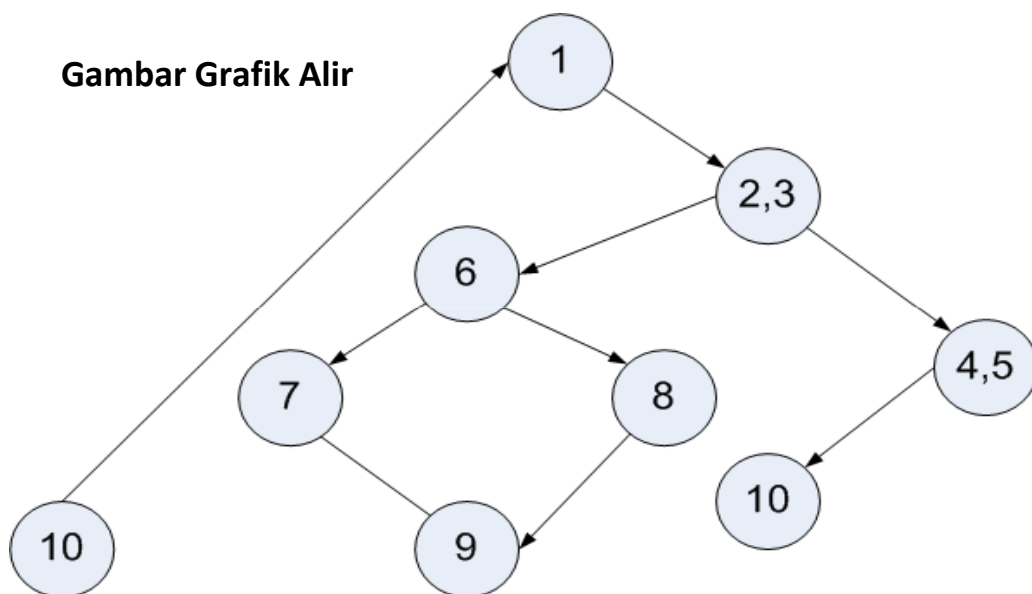


BASIS PATH TESTING (3)



BASIS PATH TESTING (4)

Gambar Grafik Alir



❑ **Lingkaran/node :**

Menggambarkan satu/lebih perintah prosedural. Urutan proses dan keputusan dapat dipetakan dalam satu node.

❑ **Tanda panah/edge :**

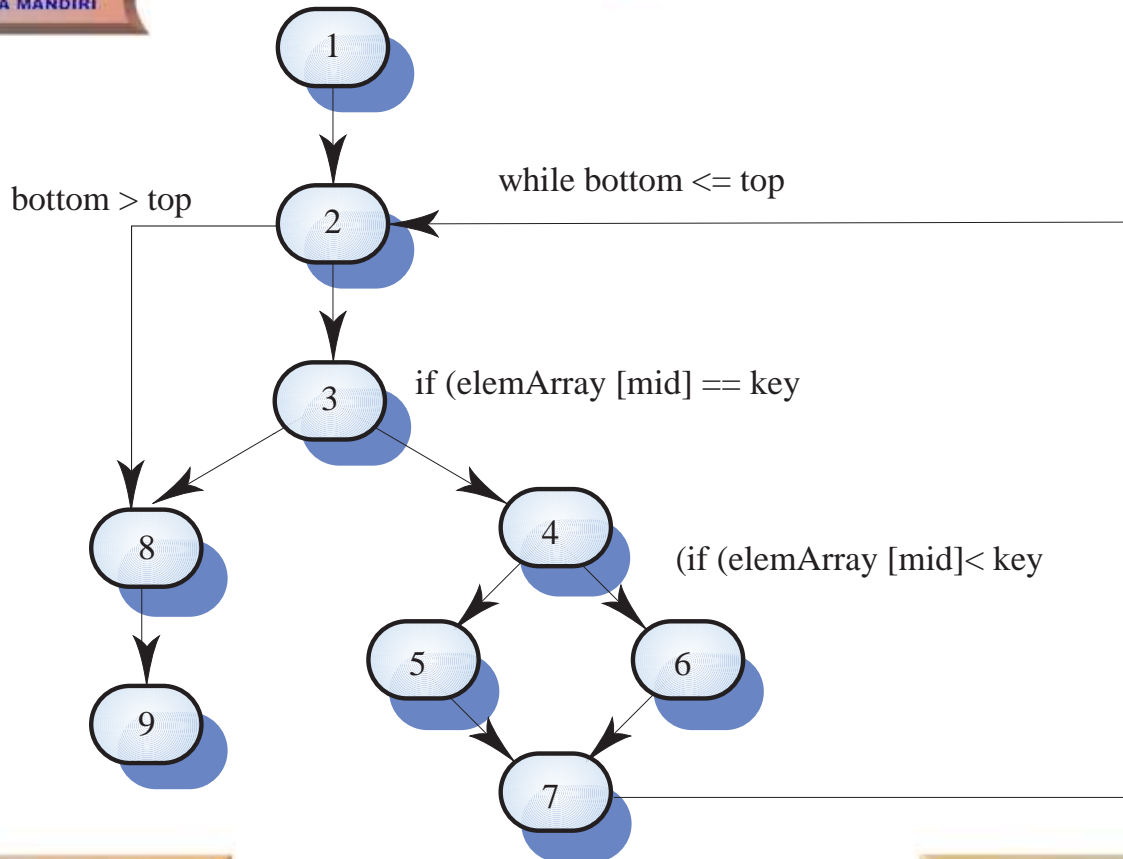
Menggambarkan aliran kontrol. Setiap node harus mempunyai tujuan node

❑ **Region :**

Adalah daerah yg dibatasi oleh edge dan node. Termasuk daerah diluar grafik alir.

Program Flow Graphs

- Menggambarkan alur kontrol. Setiap cabang ditunjukkan oleh path yang terpisah dan loop ditunjukkan oleh arrows looping kembali ke loop kondisi node.
- Digunakan sebagai basis untuk menghitung cyclomatic complexity
- Cyclomatic complexity
= Jumlah edges – Jumlah Node +2
- Cyclomatic complexity menyatakan jumlah test untuk menguji control statements



Independent Paths

- 1, 2, 3, 8, 9
- 1, 2, 3, 4, 6, 7, 2
- 1, 2, 3, 4, 5, 7, 2
- 1, 2, 3, 4, 6, 7, 2, 8, 9
- Test cases harus ditentukan sehingga semua path tersebut tereksekusi.

BLACK BOX TESTING

- Pendekatan pengujian dimana program dianggap sebagai suatu 'black-box' ('kotak hitam')
- Program test case berbasiskan spesifikasi
- Test planning dapat dimulai sejak awal proses pengembangan sistem
- Merupakan metode pelengkap *White Box Testing*. Berfokus pada kebutuhan fungsional dari PL.
- Memungkinkan perancang untuk memperoleh sekumpulan kondisi2 input yang secara penuh menguji semua kebutuhan fungsional suatu program

BLACK BOX TESTING (2)

- Black Box Testing berusaha menemukan kesalahan yang termasuk kategori di bawah ini
 - ☐ Fungsi2 yg hilang atau tidak benar
 - ☐ Kesalahan pada antarmuka
 - ☐ Kesalahan pada struktur data atau pengaksesan database eksternal
 - ☐ Kesalahan pada performance
 - ☐ Kesalahan pada inisialisasi dan terminasi

BLACK BOX TESTING (3)

Contoh **Black Box Testing** adalah

1. **Equivalence Partitioning**

2. **Boundary Value Analysis (BVA)**

- ☐ Melengkapi Equivalence Partitioning, dengan melakukannya dari domain output BVA merupakan pilihan test case yang mengerjakan nilai yang telah ditentukan, dengan teknik perancangan test case
- ☐ Melengkapi test case equivalence partitioning yg fokusnya pada domain input.

Hitung CC

- 1. Flowgraph mempunyai 4 region
- 2. $V(G) = 11 \text{ edge} - 9 \text{ node} + 2 = 4$
- 3. $V(G) = 3 \text{ predicate node} + 1 = 4$
- Perlu disiapkan 4 test case untuk masing2 path.
- Buat test case yang akan mengerjakan masing-masing path pada basisset. Data yang dipilih harus tepat sehingga setiap kondisi dari predicate node dikerjakan semua.

Latihan

■ Tugas 15 %

■ PROCEDURE RATA-RATA

- INTERFACE RESULT rata, total, input, total.valid
- INTERFACE RESULT nilai, minim, max
- TYPE NILAI (1:100) IS SCALAR ARRAY;
- TYPE rata, total, input, total.valid, max.minim, jumlah IS SCALAR;
- TYPE I IS INTEGER;
- I = 1;
- total, input = total, valid = 0;
- jumlah = 0;
- DO WHILE nilai(i) <> -999 .and. total.input < 100
 - tambahkan total.input dengan 1;
 - IF nilai(i) >= minimum .and. nilai(i) <=max;
 - THEN tambahkan total.valid dengan I;
 - jumlah=jumlah + nilai(i);
 - ELSE skip;
 - END IF
 - tambahkan i dengan 1;
- ENDDO
- IF total, valid> 0
- THEN rata =jumlah/total, valid;
- ELSE rata = -999;
- ENDIF
- END

Latihan

■ Susun Flowchart,Flowgraph,hitung CC

```

00: int bsearch(int x, const int A[], int N) {
01:   int begin = 0;
02:   int end = N;
03:   while (begin < end) {
04:     int middle = begin + (end - begin) / 2;
05:     if (x <= A[middle])
06:       end = middle;
07:     else
08:       begin = middle + 1;
09:   }
10:   return begin;
11: }

```