

REKAYASA PERANGKAT LUNAK

Dany Pratmanto M.Kom

Pertemuan 1

PERANGKAT LUNAK
dan
REKAYASA PERANGKAT
LUNAK

1. PERANGKAT LUNAK

Definisi Perangkat Lunak (PL) adalah:

- Instruksi-instruksi program komputer yang ketika dijalankan menyediakan fitur-fitur, fungsi-fungsi dan kinerja yang dikehendaki
- Struktur data yang memungkinkan program-program memanipulasi informasi
- Informasi deskriptif pada salinan tercetak dan bentuk-bentuk maya yang menggambarkan pengoperasian dan penggunaan program

2. KARAKTERISTIK PERANGKAT LUNAK

- PL dikembangkan atau direkayasa, bukan diproduksi dalam konteks manufaktur
- PL tidak mengalami “kelelahan”
- PL dibuat berdasarkan spesifikasi yang diminta oleh pengguna

A. Kategori Perangkat Lunak

- PL Sistem (*System Software*)
- PL Aplikasi (*Application Software*)
- PL Rekayasa/Ilmiah (*Engineering/Scientific Software*)
- PL yang tertanam (*Embedded Software*)
- PL Lini Produk (*Product-Line Software*)
- PL Aplikasi Web (*Web/Mobile Applications*)
- PL Kecerdasan Buatan (*Artificial Intelligence Software*)

B. Jenis Perangkat Lunak Aplikasi

a. *Stand-Alone Applications*

adalah contoh aplikasi seperti aplikasi office pada PC, program CAD, software manipulasi foto, dll

b. *Interactive Transaction-Based Applications*

adalah aplikasi yang mengeksekusi pada komputer remote dan yang diakses oleh pengguna dari PC mereka sendiri atau terminal

c. *Batch Processing Systems*

adalah sistem bisnis yang dirancang untuk memproses data input yang besar untuk membuat output yang sesuai. Contoh: sistem penagihan telepon, dan sistem pembayaran gaji

Jenis PL Aplikasi (Lanjutan)

d. *Embedded Control Systems*

adalah sistem kontrol PL yang mengontrol dan mengelola perangkat keras, atau sistem yang tertanam pada jenis sistem lain. Contoh: PL yang mengontrol penggereman *anti-lock* mobil, dan *software* dalam *oven microwave* untuk mengontrol proses memasak.

e. *Entertainment Systems*

adalah sistem yang terutama untuk penggunaan pribadi dan yang dimaksudkan untuk menghibur pengguna.

f. *Systems for Modelling and Simulation*

adalah sistem yang dikembangkan untuk model proses fisik atau situasi, dengan banyak objek yang saling berinteraksi

Jenis PL Aplikasi (Lanjutan)

g. Data Collection Systems

adalah sistem yang mengumpulkan data dari lingkungan mereka menggunakan satu set sensor dan mengirim data ke sistem lain untuk diproses.

h. Systems of Systems

adalah sistem yang terdiri dari sejumlah sistem PL lain.

C. Perangkat Lunak Warisan

- PL warisan harus diadaptasikan sedemikian rupa sehingga memenuhi kebutuhan dari lingkungan atau teknologi komputasi yang baru
- PL warisan harus ditingkatkan kinerjanya supaya dapat menjalankan kebutuhan bisnis baru
- PL warisan harus diperluas sedemikian rupa agar dapat saling mengoperasikan dengan sistem/PL/basisdata modern lainnya
- PL harus dirancang ulang sehingga dapat hidup dalam lingkungan pengoperasian jaringan komputer

D. Kegagalan Perangkat Lunak

Faktor-faktor penyebab kegagalan PL:

- **Meningkatnya tuntutan**

RPL membangun sistem yang lebih besar, sistem yang lebih kompleks menyebabkan tuntutan berubah. Sistem harus dibangun dan disampaikan lebih cepat, lebih besar, dan lebih kompleks. Sistem harus memiliki kemampuan baru yang sebelumnya dianggap mustahil.

- **Harapan yang rendah**

Hal ini relatif mudah untuk menulis program komputer tanpa menggunakan metode dan teknik RPL. Banyak Pengusaha yang tidak menggunakan metode RPL, akibatnya PL lebih mahal dan kurang dapat diandalkan.

Stakeholder dalam RPL

- ***Users***: adalah orang-orang yang akan menggunakan PL.
- ***Customer (client)***: adalah orang-orang yang membeli atau memesan PL.
- ***Software Developer***: adalah orang-orang yang mengembangkan dan memelihara PL.
- ***Development Manager***: adalah orang-orang yang menjalankan organisasi yang mengembangkan PL, dan biasanya memiliki latar belakang pendidikan dalam administrasi bisnis.

3. REKAYASA PERANGKAT LUNAK (RPL)

- RPL adalah disiplin teknik yang berkaitan dengan semua aspek produksi PL dari tahap awal spesifikasi sistem sampai pemeliharaan.
- Aspek produksi RPL berkaitan dengan proses teknis dari pengembangan PL, manajemen proyek PL dan pengembangan alat-alat, metode, dan teori untuk mendukung produksi PL.
- RPL merupakan aplikasi dari suatu pendekatan yang semantik, disiplin, dan dapat diukur pada pengembangan, operasi, dan perawatan PL.

RPL (Lanjutan)

PL dalam segala bentuk aplikasinya harus direkayasa, dengan alasan:

- PL telah menyatu secara maya dengan setiap aspek dalam kehidupan
- Kebutuhan IT yang sudah banyak dituntut oleh individu, bisnis dan pemerintah bertambah kompleks
- Individu, bisnis, dan pemerintah mengandalkan PL untuk mengambil keputusan yang bersifat taktis dan strategis
- Nilai aplikasi terus bertambah, kemungkinan jumlah pengguna dan usia PL akan bertambah

4. PROSES PERANGKAT LUNAK

- Suatu **proses** merupakan sekumpulan aktivitas, aksi, dan tugas yang dijalankan ketika suatu produk kerja harus dibuat.
- Sebuah **proses PL** adalah urutan kegiatan yang mengarah ke produksi produk software.
- Empat kegiatan proses PL adalah:
 - a. Spesifikasi PL
 - b. Pengembangan PL
 - c. *Software validasi*
 - d. *Software evolusi*

Proses PL (Lanjutan)

- Suatu **aktivitas** berupaya mencapai tujuan umum dan diterapkan tanpa memperhatikan lingkungan aplikasi, tanpa memperhatikan ukuran proyek, tanpa memperhatikan kompleksitas dan usaha, dan tanpa memperhatikan kekakuan dari RPL saat diterapkan.
- Suatu **tugas** konsentrasi pada tujuan yang kecil tetapi terdefinisi dengan baik.

Proses PL (Lanjutan)

- **Kerangka kerja proses** membangun dasar bagi proses RPL yang lengkap dengan cara mengidentifikasi aktivitas kerangka kerja yang cocok untuk semua proses RPL.
- Kerangka kerja proses mencakup sekumpulan aktivitas yang berperan sebagai penyanga dan cocok dengan keseluruhan proses PL.
- Aktivitas kerangka kerja proses:
 - a. Komunikasi
 - b. Perencanaan
 - c. Pemodelan
 - d. Konstruksi
 - e. Penyerahan PL ke pelanggan/user

Proses PL (Lanjutan)

- Aktivitas kerangka kerja proses RPL disempurnakan oleh aktivitas yang bertindak sebagai penyangga.
- Kegiatan-kegiatan penyangga mencakup:
 - a. Penelusuran dan kendali proyek PL
 - b. Manajemen risiko
 - c. Penjaminan kualitas PL
 - d. Tinjauan teknis
 - e. Pengukuran
 - f. Manajemen konfigurasi PL
 - g. Manajemen penggunaan ulang
 - h. Persiapan produk kerja dan produksi

5. PRAKTEK RPL

Langkah-langkah RPL:

a. Memahami permasalahan

- Siapa yang terkait dalam pemecahan masalah?
- Apa saja yang tidak diketahui?
- Data, fungsi, dan fitur yang dibutuhkan
- Dapatkah masalah dikategorikan (dipecah menjadi masalah yang lebih kecil)?
- Dapatkah masalah diwakili dengan grafis?
- Dapatkah dibuat sebuah model analisis?

PRAKTEK RPL (Lanjutan)

b. Merancang solusi

- Pernahkah ada masalah serupa sebelumnya dan telah didapatkan pemecahan masalahnya?
- Dapatkah sub-masalah didefinisikan?
- Dapatkah menyusun solusinya?

PRAKTEK RPL (Lanjutan)

c. Menjalankan rancangan

- Apakah solusi cocok dengan masalah?
- Apakah kode program dapat dilacak secara langsung?
- Apakah komponen dari solusi sudah tepat?

d. Memeriksa hasil

- Uji setiap komponen dari solusi dengan menggunakan strategi pengujian
- Apakah solusi sesuai dengan data, fungsi dan fitur yang dibutuhkan?

Prinsip-Prinsip Umum RPL

- a. Alasan keberadaan PL
- b. Sederhana
- c. Pertahankan visi
- d. Apa yang dibuat, akan digunakan oleh konsumen/pengguna
- e. Membuka diri terhadap masa depan
- f. Merancang selangkah ke depan sehingga dapat digunakan kembali
- g. Review

6. MITOS-MITOS PL

A. Mitos Manajemen

Mitos-1: Kita sudah memiliki buku yang standar dan prosedur untuk membangun PL.

Realita: Apakah buku tersebut mencerminkan praktik RPL modern, lengkap, dan dapat beradaptasi dengan keadaan yang dihadapi saat ini?

Mitos-2: Jika kita tertinggal dari jadwal yang telah ditetapkan, kita dapat menambah jumlah programmer dan akan memenuhi jadwal dengan cepat.

Realita: Menambah orang baru untuk proyek PL yang tertunda menyebabkan penyelesaian proyek PL tersebut menjadi semakin terlambat.

A. Mitos Manajemen (Lanjutan)

Mitos-3: Jika memutuskan untuk menyewa orang ketiga untuk mengerjakan proyek PL, kita bisa sedikit lega karena PL dikerjakan oleh pihak ketiga.

Realita: Jika sebuah organisasi tidak dapat memahami cara mengelola dan mengendalikan proyek PL secara internal, maka organisasi tersebut akan bekerja lebih keras lagi ketika menyewa pihak ketiga.

B. Mitos Pelanggan

- Mitos-1: Pernyataan tujuan umum sudah cukup untuk mulai menulis program, dan kita dapat membuat rinciannya nanti.
- Realita: Pembuatan pernyataan kebutuhan yang komprehensif dan stabil tidak selalu dimungkinkan (tidak ambigu), tetapi perlu mengembangkan komunikasi yang efektif antara pengembang dan pelanggan.
- Mitos-2: Kebutuhan PL terus menerus berubah, tetapi perubahan-perubahan dapat dengan mudah diakomodasi karena PL bersifat fleksibel.
- Realita: Dampak perubahan beragam sesuai dengan waktu di mana perubahan diperkenalkan.

C. Mitos Praktisi

Mitos-1: Ketika kita menulis kode program dan menjalakannya, maka pekerjaan dianggap sudah selesai.

Realita: Semakin cepat kita mulai menulis „kode program“, semakin lama waktu yang dibutuhkan untuk menyelesaiakannya.

Mitos-2: Satu-satunya produk kerja untuk mencetak proyek PL yang berhasil adalah program yang sedang berjalan.

Realita: Sebuah produk kerja hanyalah sebagian kecil dari konfigurasi PL yang pada dasarnya mencakup banyak unsur RPL yang berhasil dan memberikan panduan bagi dukungan PL.

C. Mitos Praktisi (lanjutan)

Mitos-3: RPL akan memaksa kita membuat dokumentasi yang berlebihan dan terkesan tidak penting, dan akan selalu menghambat kemajuan.

Realita: RPL merupakan kegiatan yang bertujuan untuk meningkatkan kualitas produk. Kualitas yang baik mengarah pada berkurangnya pekerjaan yang berulang-ulang sehingga pengiriman ke pelanggan akan lebih cepat.

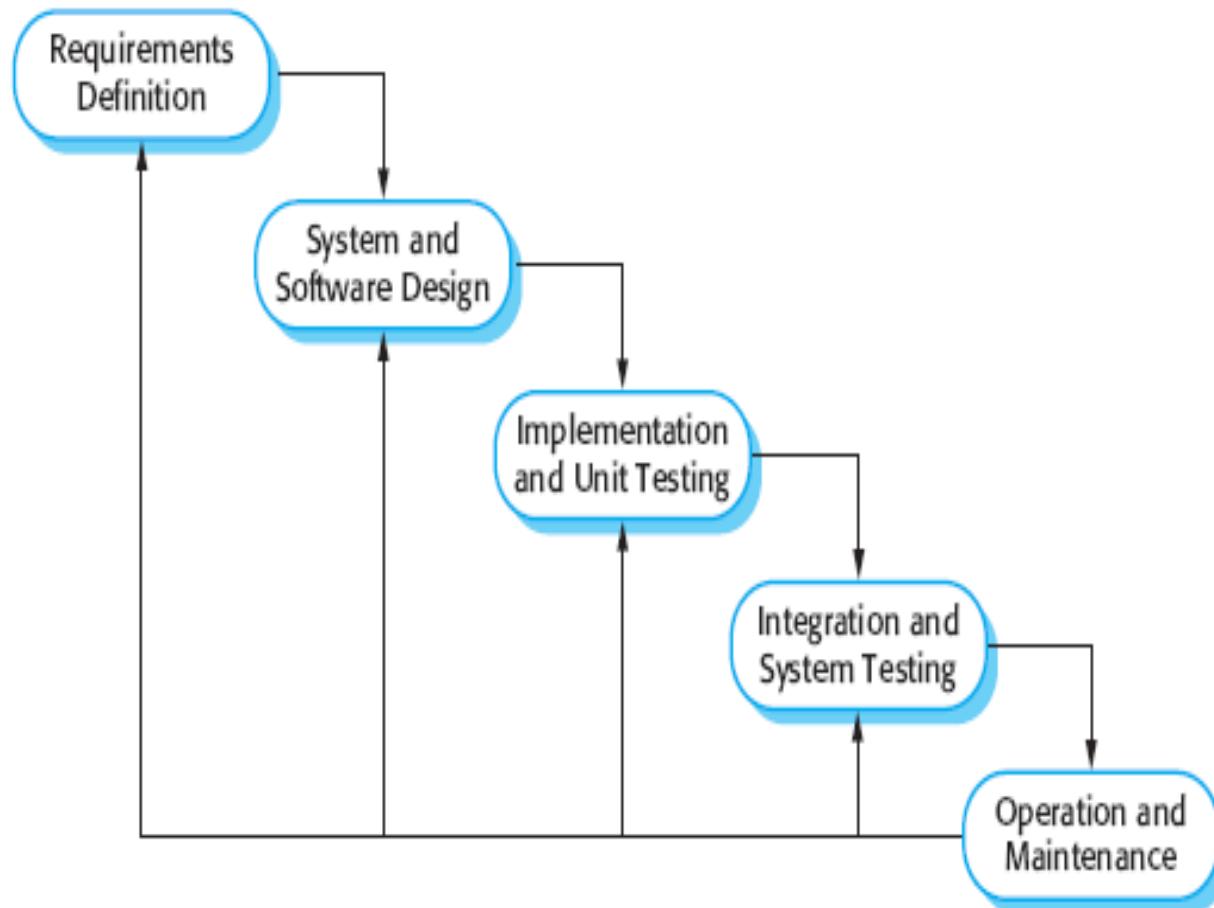
Pertemuan 2

**MODEL PROSES
PENGEMBANGAN
PERANGKAT LUNAK**

1. MODEL WATERFALL

- Model *Waterfall* juga disebut siklus hidup klasik (*Classic Life Cycle*).
- Merupakan pendekatan sistematis dan berurutan (*sequential*) pada PL yang dimulai dengan spesifikasi kebutuhan *user* dan berlanjut melalui beberapa tahapan, dan diakhiri dengan penyerahan sistem/ PL kepada pelanggan.
- Pada prinsipnya, hasil dari setiap tahap adalah satu atau lebih dokumen yang disetujui (*Sign off*).
- Tahap berikutnya tidak boleh dimulai sampai tahap sebelumnya selesai.

Model Waterfall



Sumber: Sommerville (2011: 30)

Tahapan Model Waterfall

1. *Requirements Analysis and Definition*

Langkah ini merupakan analisa kebutuhan sistem. Berisi layanan-layanan sistem, kendala, dan tujuan yang ditetapkan melalui konsultasi dengan pengguna sistem, kemudian didefinisikan secara rinci yang berfungsi sebagai spesifikasi sistem.

Requirements Analysis and Definition (Lanjutan)

Misalkan untuk aplikasi web:

Maka analisa kebutuhannya ditinjau dari kebutuhan Admin dan User. Jelaskan secara singkat kebutuhan apa yang bisa dilakukan oleh admin dan user.

Contoh untuk web: User dapat melakukan login, user dapat membeli produk, user dapat melakukan konfirmasi pembayaran, dll

Misalkan untuk aplikasi desktop:

Pada program penjualan, kebutuhannya adalah kasir.

Tahapan Model Waterfall

2. *System and Software Design*

Proses *design* mengalokasikan kebutuhan *hardware* dan *software* untuk membangun arsitektur sistem secara keseluruhan (struktur data, arsitektur PL, *interface*, dan detail/algoritma prosedural). Proses *design* akan menerjemahkan syarat kebutuhan perancangan PL yang dapat diperkirakan sebelum dibuat *coding*.

Contoh:

Aplikasi web ini dibangun dengan beberapa desain, diantaranya desain database menggunakan ERD dan LRS, desain struktur program menggunakan Struktur Navigasi, dan desain sistem menggunakan UML.

Tahapan Model Waterfall

3. *Implementation and Unit Testing*

Perancangan PL direalisasikan sebagai satu set program atau unit program (*coding*). *Coding* merupakan penerjemahan *design* dalam bahasa yang bisa dikenali oleh komputer yang dilakukan oleh *programmer*.

Kemudian dilanjutkan dengan testing terhadap pengujian unit dengan melibatkan verifikasi setiap unit agar memenuhi spesifikasinya.

Contoh:

Pembuatan code program menggunakan software Dreamweaver CS5, pengolahan database dengan MySQL, script untuk penjelajah web menggunakan Javascript, dst...

Tahapan Model Waterfall

4. *Integration and System Testing*

Program-program diintegrasikan dan diuji sebagai sistem yang lengkap untuk memastikan bahwa kebutuhan/persyaratan PL telah dipenuhi. Setelah pengujian, PL sistem dikirim ke pelanggan.

Contoh:

Setelah tahap pembuatan code program dilakukan pengujian program untuk menemukan kesalahan-kesalahan program. Pengujian program menggunakan *black box testing* untuk memastikan bahwa seluruh input yang memerlukan verifikasi data sudah benar.

Tahapan Model Waterfall

5. *Operation and Maintenance*

Operasi dan pemeliharaan adalah siklus hidup terlama.

Sistem ini dipasang dan digunakan oleh *user*.

Perawatan melibatkan koreksi kesalahan yang tidak ditemukan sebelumnya, meningkatkan implementasi sistem dan meningkatkan layanan sistem saat persyaratan baru ditemukan.

Contoh:

Pemeliharaan sistem akan terus dilakukan dengan korektif terhadap kesalahan yang mungkin terjadi, adaptif dengan peningkatan layanan sistem, dst...

Kelebihan model waterfall:

- a. Memiliki proses yang urut dan bertahap, sehingga kualitas sistem/PL yang dihasilkan akan baik.
- b. Setiap proses memiliki spesifikasinya sendiri, karena setiap tahap harus terselesaikan dengan lengkap sebelum melanjutkan ke tahap berikutnya.
- c. Setiap proses tidak dapat saling tumpang tindih.
- d. Metode ini akan lebih baik digunakan jika kebutuhan-kebutuhan sudah diketahui.

Kekurangan model *waterfall*:

- a. Proses yang dilakukan cenderung panjang dan lama, karena proses pengembangan tidak dapat dilakukan berulang sebelum menghasilkan produk.
- b. Kesalahan kecil pada satu tahapan akan menimbulkan masalah besar jika tidak diketahui sejak awal pengembangan, berakibat pada tahapan selanjutnya
- c. Biaya penggunaan metode yang cenderung mahal
- d. Membutuhkan banyak riset dan penelitian pendukung untuk mengembangkan sistem sehingga pelanggan harus sabar karena pembuatan PL baru dimulai pada tahap perancangan.
- e. Kenyataannya sulit untuk mengikuti aturan *sequential*, karena iterasi sulit dilakukan dan menyebabkan masalah baru.

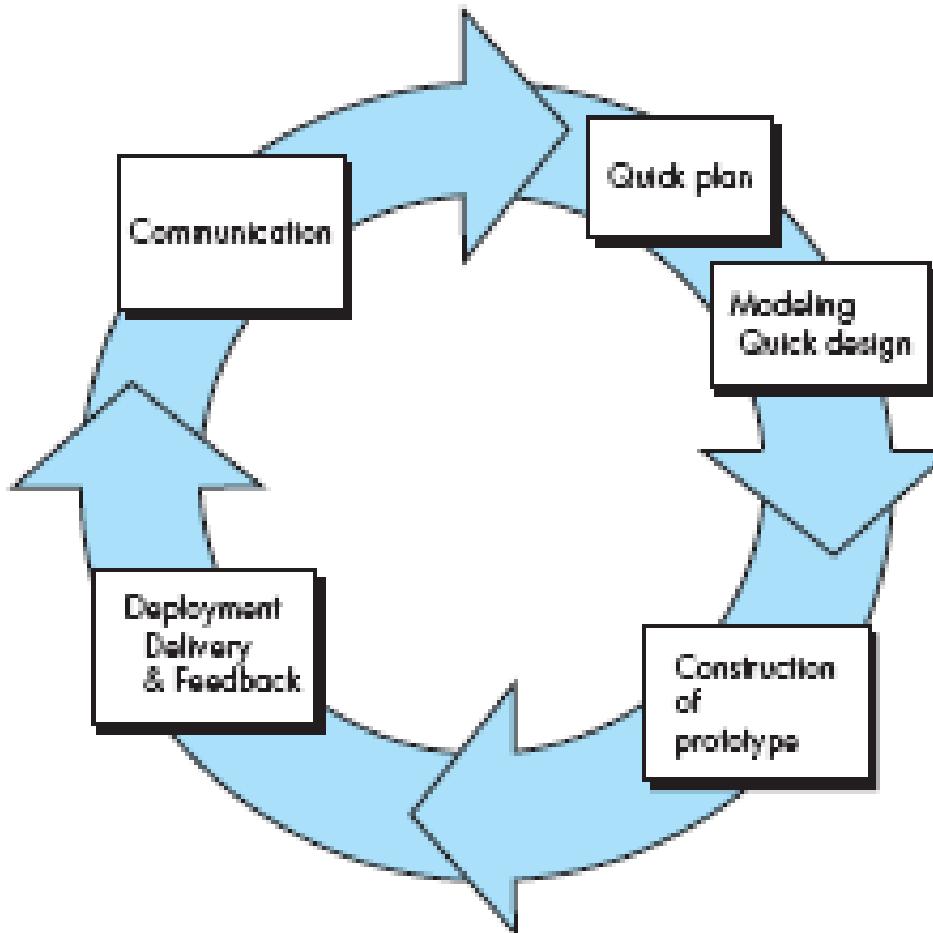
2. MODEL *PROTOTYPE*

- *Prototype* adalah pendefinisian sejumlah sasaran PL berdasarkan kebutuhan dan pemahaman secara umum, tetapi tidak bisa mengidentifikasi kebutuhan secara rinci untuk beberapa fungsi dan fitur-fitur.
- Tujuannya adalah untuk membantu dalam tahap analisis dan desain yang memungkinkan pengguna untuk melihat lebih awal apa yang akan dilakukan sistem, yaitu untuk memfasilitasi validasi.
- *Prototype* dapat digunakan sebagai model proses yang berdiri sendiri.

MODEL *PROTOTYPE* (Lanjutan)

- Pembuatan prototipe biasanya digunakan sebagai teknik yang dapat diimplementasikan di dalam konteks setiap model proses PL, sehingga membantu *stakeholder* untuk lebih memahami apa yang akan dikembangkan ketika spesifikasi kebutuhan belum jelas.

Model Prototype



Sumber: Pressman (2015: 45)

Tahapan dalam Model *Prototype*

1. Dimulai dengan dilakukannya komunikasi antara tim pengembang PL dengan pelanggan.
2. Tim pengembang bertemu dengan *stakeholder* untuk mendefinisikan sasaran keseluruhan PL, mengidentifikasi spesifikasi kebutuhan yang diketahui, dan menggambarkan definisi lebih jauh pada iterasi selanjutnya.
3. Pembuatan prototipe direncanakan dengan cepat, dan pemodelan dilakukan
4. Prototipe diserahkan kepada *stakeholder* untuk dievaluasi, dan memberikan umpan balik yang digunakan untuk persyaratan lebih lanjut
5. Iterasi akan terjadi saat prototipe diperbaiki

Tujuan Pengembangan Model *Prototype*:

- a. Membuat antarmuka pengguna yang dapat diterima
- b. Membuat sistem yang dapat berfungsi, meskipun terbatas, tetapi tersedia dengan cepat untuk menunjukkan kelayakan dan kegunaan dari aplikasi
- c. Dapat digunakan untuk melatih pengguna sebelum sistem yang lengkap dikirim ke pelanggan
- d. Untuk menjelaskan bahwa beberapa teknologi baru akan menyediakan fasilitas yang dibutuhkan

Manfaat Model *Prototype*

Model *prototype* dikembangkan dan didemonstrasikan pada awal proses pembangunan PL, sehingga dapat bermanfaat untuk:

- a. Menghindari kesalahpahaman antara pengembang PL dan pengguna
- b. Beberapa fasilitas yang hilang mungkin dapat terungkap
- c. Fasilitas yang sulit digunakan/membingungkan dapat diidentifikasi dan disempurnakan
- d. Pengembang PL mungkin menemukan persyaratan yang tidak lengkap atau tidak konsisten.

Masalah pada Model *Prototype*

1. *Stakeholder* hanya melihat tampilan PL yang akan dipakai tanpa mempedulikan bagaimana kerja sistem, dan pemeliharaan jangka panjang.
2. Perubahan yang dibuat selama pengembangan PL mungkin akan mengubah struktur arsitektur. Oleh karena itu mungkin sulit dan mahal untuk pemeliharaannya.
3. Karakteristik sistem yang penting seperti kinerja, keamanan dan keandalan, mungkin akan diabaikan selama pengembangan PL.
4. Selama tahap pengembangan, *prototype* akan diubah untuk memenuhi kebutuhan pengguna. kemungkinan perubahan yang dibuat akan tidak terkontrol dan tidak didokumentasikan dengan baik.

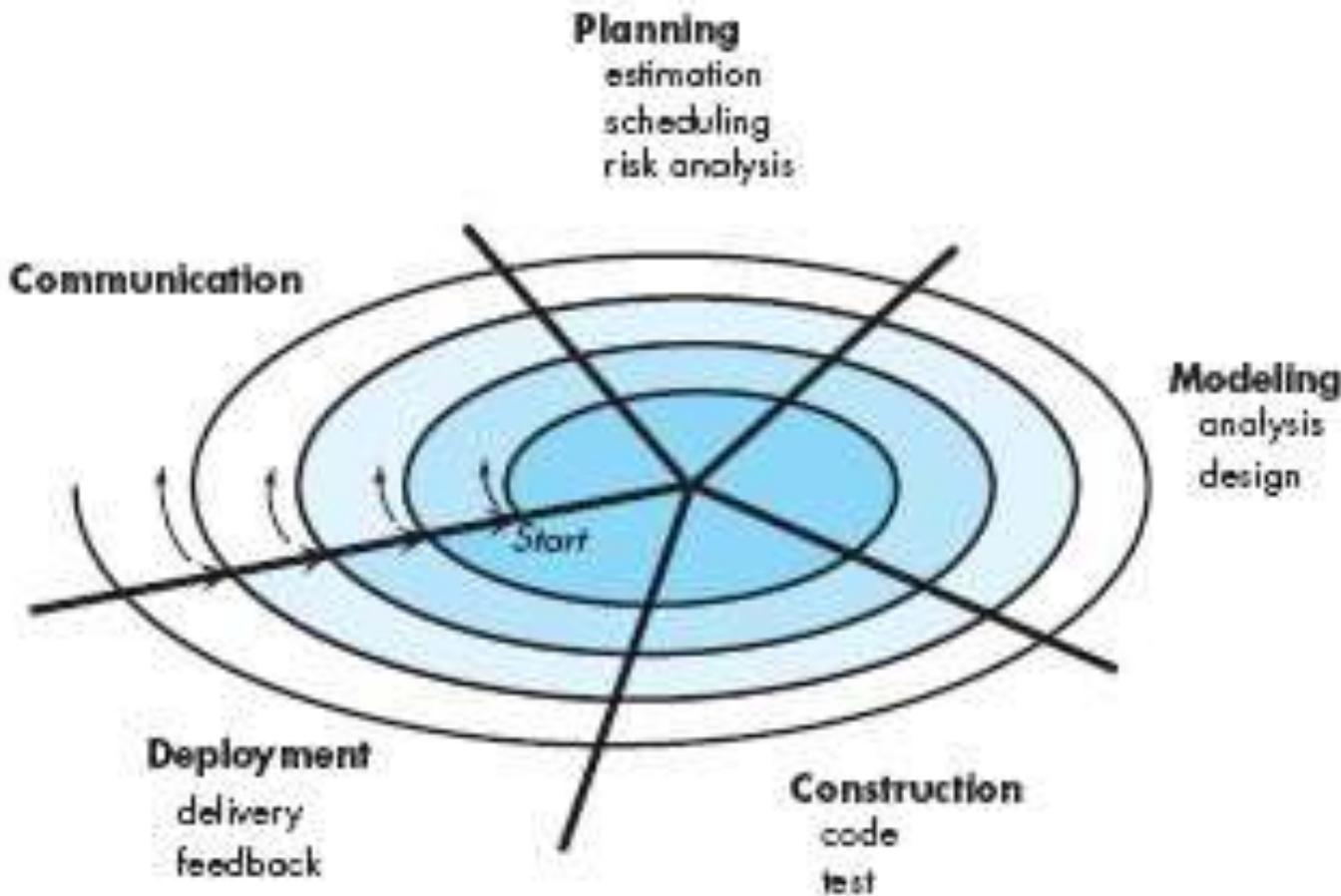
3. MODEL SPIRAL

- Model Spiral merupakan suatu model proses PL evolusioner yang menggabungkan pendekatan *prototyping* yang bersifat iteratif dengan aspek yang terkontrol dan sistematis pada model *waterfall*.
- Model pengembangan spiral adalah model proses PL yang dikendalikan risiko yang digunakan untuk memandu para *stakeholder* untuk secara bersamaan merekayasa sistem yang bernuansa PL.

MODEL SPIRAL (Lanjutan)

- Model spiral menggunakan prototipe sebagai mekanisme pengurangan risiko.
- Model spiral menggunakan pendekatan langkah demi langkah (*waterfall*) yang sistematis tetapi menggabungkannya ke dalam kerangka iteratif yang lebih realistik mencerminkan dunia nyata.
- Model spiral adalah pendekatan realistik untuk pengembangan sistem berskala besar.

Model Spiral



Sumber: Pressman (2015: 48)

Penjelasan gambar:

- a. Proses evolusioner ini dimulai dari titik tengah, ke bagian luar spiral searah jarum jam.
- b. Risiko PL akan dipertimbangkan saat masing-masing gerakan dibuat dan titik pengukuran dicatat setiap saat langkah-langkah evolusioner dilewati.
- c. Lintasan pertama di sekitar spiral dapat menghasilkan spesifikasi produk, putaran berikutnya di sekitar spiral mungkin digunakan untuk mengembangkan suatu prototipe dan pada lintasan berikutnya secara progresif bergerak ke versi PL yang semakin canggih.

Penjelasan gambar (lanjutan)

- d. Setiap melewati lintasan menghasilkan penyesuaian pada perencanaan proyek.
- e. Biaya dan jadwal disesuaikan berdasarkan umpan balik yang berasal dari pelanggan setelah pengiriman produk.
- f. Selain itu, dilakukan penyesuaian jumlah iterasi yang direncanakan untuk menyelesaikan produk PL.

Keuntungan Model *Spiral*

- a. Pendekatan yang dikendalikan risiko menghindari banyak kesulitan.
- b. Mengakomodasi persiapan untuk evolusioner siklus hidup, pertumbuhan, dan perubahan pada produk PL.
- c. Menyediakan mekanisme untuk tujuan kualitas dan PL gabungan ke pengembangan produk PL
- d. Mempunyai fokus untuk mengeliminasi kesalahan (*error*)
- e. Menyediakan pendekatan terpisah untuk pengembangan dan pemasangan PL
- f. Menyediakan kerangka kerja aktif untuk pengembangan sistem *hardware* dan *software* yang terintegrasi.

Kerugian Model *Spiral*

- a. Memerlukan penyesuaian dengan PL yang menitik beratkan pada kontrol dan titik permasalahan yang merupakan keunggulan model *waterfall*.
- b. Berdasarkan keahlian manajemen risiko yang memerlukan penaksiran risiko yang masuk akal, akan menimbulkan masalah yang lebih besar jika risiko mayor tidak ditemukan
- c. Memerlukan kebutuhan untuk penelitian lebih lanjut terhadap langkah-langkah spiral khususnya untuk area analisis risiko.

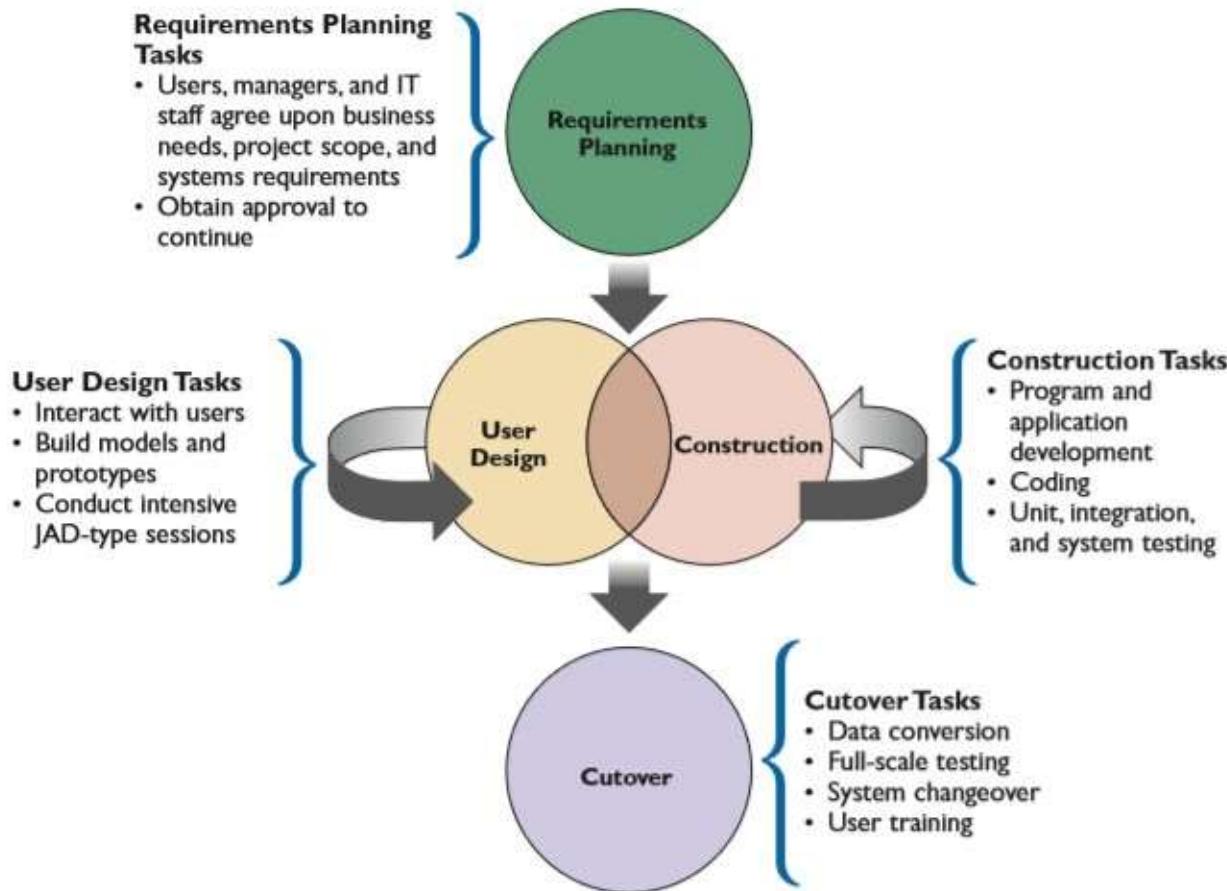
4. MODEL *Rapid Application Development* (RAD)

- RAD adalah teknik berbasis tim yang mempercepat pengembangan SI dan menghasilkan fungsi-fungsi SI.
- RAD menggunakan pendekatan kelompok
- Produk akhir RAD adalah sistem informasi baru.
- RAD adalah metodologi yang lengkap, dengan 4 fase siklus hidup yang sejajar dengan fase SDLC tradisional.
- Penggunaan RAD untuk mengurangi biaya dan waktu pengembangan, dan meningkatkan probabilitas keberhasilan

MODEL RAD (Lanjutan)

- RAD sangat bergantung pada prototipe dan keterlibatan pengguna.
- Berdasarkan input pengguna, prototipe dimodifikasi dan proses interaktif berlanjut sampai sistem benar-benar dikembangkan dan pengguna puas.
- Interaksi berkelanjutan antara fase desain dan konstruksi pengguna
- Tim proyek menggunakan *CASE tools* untuk membangun prototipe dan membuat aliran dokumentasi yang berkelanjutan.

Fase dan Aktivitas RAD



Sumber: Shelly (2012: 148)

Fase RAD (Lanjutan)

1. REQUIREMENTS PLANNING

- Fase ini menggabungkan elemen-elemen perencanaan sistem dan fase analisis SDLC.
- Pengguna, manajer, dan anggota staf IT mendiskusikan dan menyetujui kebutuhan bisnis, ruang lingkup proyek, kendala, dan persyaratan sistem.
- Fase ini berakhir ketika tim menyetujui masalah-masalah utama dan mendapatkan izin manajemen untuk melanjutkan.

Fase RAD (Lanjutan)

2. USER DESIGN

- Selama fase ini, pengguna berinteraksi dengan sistem analis kemudian mengembangkan model dan prototipe yang mewakili semua input, proses, output.
- Tim/subkelompok RAD biasanya menggunakan kombinasi teknik JAD (*Joint Application Development*) dan *CASE tools* untuk menerjemahkan kebutuhan pengguna ke dalam model.
- Desain pengguna adalah kontinyu, proses interaktif memungkinkan pengguna untuk memahami, memodifikasi, dan akhirnya menyetujui model kerja sistem yang memenuhi kebutuhan mereka.

Fase RAD (Lanjutan)

3. CONSTRUCTION

- Fase konstruksi berfokus pada tugas pengembangan program dan aplikasi yang mirip dengan SDLC.
- Pengguna terus berpartisipasi dan masih dapat menyarankan perubahan atau peningkatan saat tampilan atau laporan aktual dikembangkan.

Fase RAD (Lanjutan)

4. CUTOVER

- Merupakan fase peralihan, termasuk konversi data, pengujian, pergantian ke sistem baru, dan pelatihan pengguna.
- Dibandingkan dengan metode tradisional, seluruh proses dikompresi. Akibatnya, sistem baru dibangun, dikirim, dan ditempatkan dalam operasi yang lebih cepat (*agile*).

Tujuan RAD

- a. Mengurangi waktu dan biaya pengembangan dengan melibatkan pengguna dalam setiap fase pengembangan sistem.
- b. Membuat modifikasi yang diperlukan dengan cepat, seiring dengan perkembangan desain.
- c. Membatasi biaya perubahan yang biasanya terjadi dalam jadwal pengembangan yang berlarut-larut.
- d. Membutuhkan sistem informasi untuk mendukung fungsi bisnis baru, karena digerakkan oleh pengguna.
- e. Dengan input pengguna, membantu tim pengembangan merancang sistem yang membutuhkan antarmuka pengguna yang interaktif atau kompleks.

Keuntungan dan Kerugian RAD

Keuntungan utama:

- sistem dapat dikembangkan lebih cepat dengan penghematan biaya yang signifikan

Kerugian:

- Menekankan mekanisme sistem itu sendiri dan tidak menekankan kebutuhan strategis bisnis perusahaan.
- Baik untuk jangka pendek.
- Memungkinkan lebih sedikit waktu untuk mengembangkan kualitas, konsistensi, & standar desain.
- RAD dapat menjadi alternatif yang menarik, jika suatu organisasi memahami risiko yang mungkin terjadi

5. MODEL SCRUM

- Scrum adalah sebuah proses yang *agile* untuk menangani produk yang kompleks.
- Scrum digunakan untuk memandu kegiatan pengembangan dalam suatu proses yang mencakup kerangka kerja seperti: kebutuhan, analisis, desain, evolusi, dan pengiriman.
- Scrum menekankan penggunaan seperangkat pola proses PL yang telah terbukti efektif untuk proyek dengan jadwal yang ketat, perubahan kebutuhan, dan kekritisan bisnis

Tahapan kegiatan pengembangan Scrum

1. *Backlog*

- Daftar prioritas kebutuhan proyek/fitur yang menyediakan nilai bisnis bagi pelanggan.
- Item dapat ditambahkan ke *backlog* kapan saja (diperkenalkan adanya perubahan).

Tahapan kegiatan pengembangan Scrum (Lanjutan)

2. *Sprints*

- Unit kerja diperlukan untuk mencapai kebutuhan yang didefinisikan dalam *backlog* yang harus sesuai dengan *time box* yang telah didefinisikan sebelumnya (biasanya 30 hari).
- Time box adalah waktu yang telah dialokasikan untuk menyelesaikan beberapa tugas
- Tidak diperkenankan adanya perubahan, sehingga anggota tim bekerja dalam lingkungan jangka pendek tetapi stabil.

Tahapan kegiatan pengembangan Scrum (Lanjutan)

3. *Scrum Meetings*

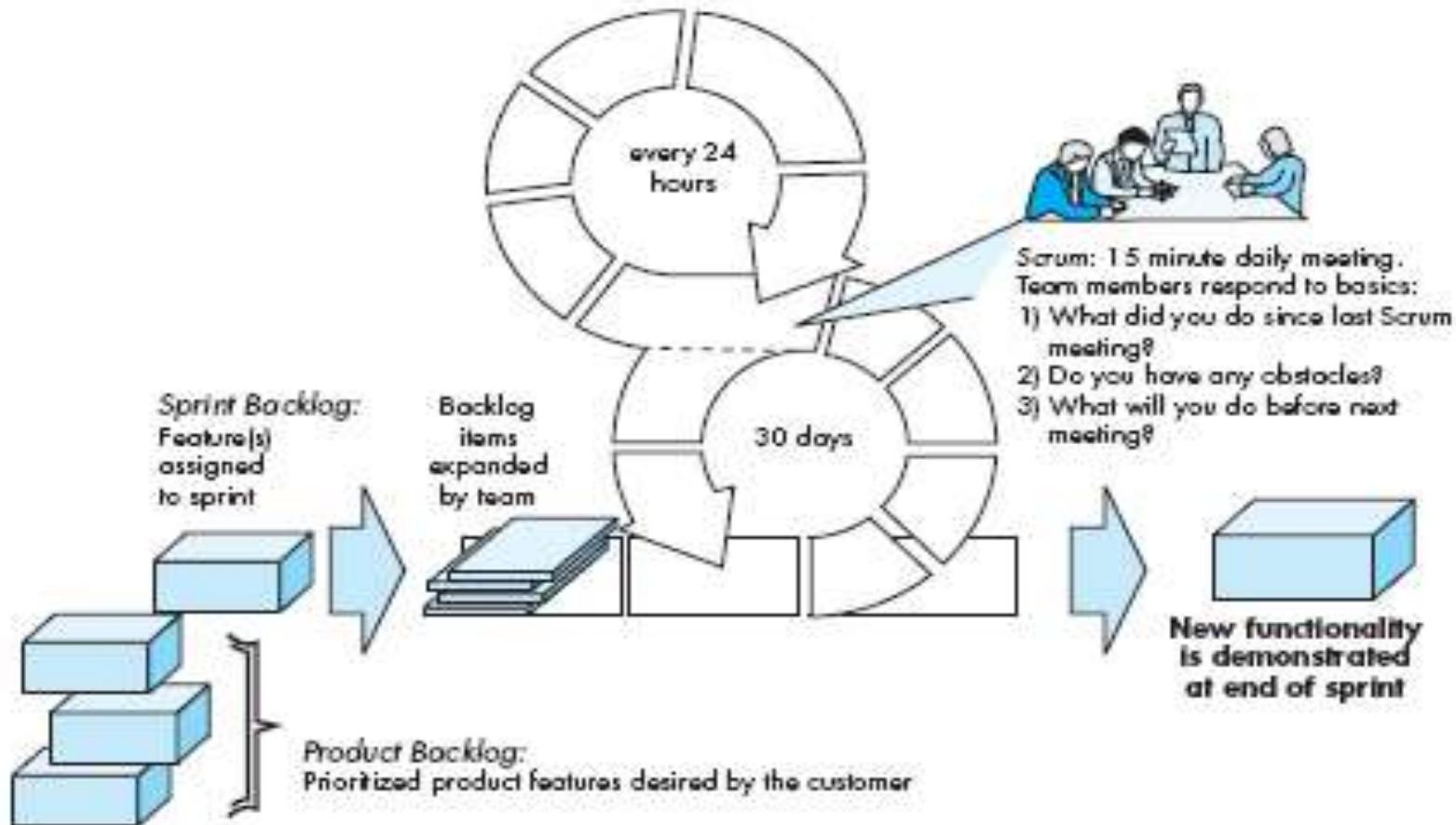
Waktunya pendek (biasanya 15 menit) tiap pertemuan yang diadakan setiap hari.

Pertemuan ini membantu tim untuk mengungkap potensi masalah sedini mungkin.

4. *Demo*

Memberikan peningkatan PL untuk pelanggan sehingga fungsionalitas yang telah diterapkan dapat didemonstrasikan dan dievaluasi oleh pelanggan.

Model Scrum



Sumber: Pressman (2015: 78)

Penjelasan gambar

- a. Gagasan awal adalah bahwa seluruh tim harus diberdayakan untuk membuat keputusan sehingga istilah 'Manajer Proyek', telah dengan sengaja dihindari.
- b. **Scrum Master** adalah seorang fasilitator yang mengatur
 - pertemuan harian
 - melacak *backlog* dari pekerjaan yang harus dilakukan
 - mencatat keputusan
 - mengukur kemajuan terhadap *backlog*
 - berkomunikasi dengan pelanggan dan manajemen di luar tim

Penjelasan gambar (lanjutan)

- c. Seluruh tim menghadiri pertemuan harian, agar tetap fokus pada tugas masing-masing.
- d. Selama pertemuan, semua anggota tim berbagi informasi, melaporkan kemajuan mereka sejak pertemuan terakhir, menjelaskan masalah yang muncul, dan apa yang direncanakan untuk hari berikutnya.
- e. Ini berarti bahwa semua orang di tim tahu apa yang sedang terjadi, dan jika masalah muncul, dapat saling membanttu.
- f. Semua orang berpartisipasi dalam perencanaan jangka pendek ini.

Keuntungan Model Scrum

- a. Produk dipecah menjadi beberapa sub produk sehingga dapat dikelola dan dimengerti.
- b. Persyaratan yang tidak stabil tidak menghambat kemajuan.
- c. Seluruh tim memiliki visibilitas sehingga dapat meningkatkan komunikasi tim.
- d. Pelanggan melihat pengiriman tepat waktu dan mendapatkan umpan balik tentang bagaimana produk tersebut bekerja.
- e. Kepercayaan antara pelanggan dan pengembang akan terjalin dimana semua orang mengharapkan proyek untuk berhasil.

Contoh penggunaan waterfall dalam proses pengembangan sistem:

Studio mengantikan Eclipse sebagai IDE resmi untuk mengembangkan aplikasi Android.

- b. Bahasa Pemrograman Java yang merupakan bahasa berorientasi objek untuk pengembangan aplikasi mandiri, aplikasi berbasis *internet* aplikasi untuk perangkat cerdas yang berkomunikasi lewat *Internet/jaringan* komunikasi (Hariyanto B. , 2014)

Model pengembangan sistem yang digunakan pada penelitian guna mendapatkan informasi adalah:

1. Teknik yang digunakan pada tahap pengembangan sistem (*System Development Live Cycle*) adalah teknik waterfall yang dilakukan dalam beberapa tahap (Pressman, 2012):
 - a. *Software requirements analysis*, merupakan tahap untuk mendapatkan spesifikasi kebutuhan pengguna dalam penggunaan aplikasi parenting yaitu smartphone yang berbasis android versi 6.0.

- b. *Design*. Pada tahap ini dibuat disain interface dan model sistem dengan UML (*Unified Modelling Language*) berdasarkan kebutuhan pengguna, perangkat lunak dan perangkat keras. Diagram yang dibuat berupa Usecase Diagram, Activity Diagram, dan Deployment Diagram.
- c. *Code generation*, merupakan tahap pembuatan program aplikasi dengan *software Android Studio* dan bahasa pemrograman *java* dan pemakaian aplikasi diimplementasikan pada *smartphone* berbasis Android.
- d. *Testing* dilakukan untuk mengetahui apakah aplikasi yang dibuat berfungsi dengan benar. Pengujian aplikasi ini menggunakan *black box testing*.

2. Untuk mendapatkan data dan informasi yang bervariasi dari beberapa sumber yang berbeda guna pembuatan aplikasi ini, dilakukan wawancara kepada ibu yang mempunyai anak usia 0-24 bulan. Hasil dari wawancara mudah

Tugas

- Mahasiswa membuat rancangan pengembangan PL menggunakan salah satu model proses yang telah dibahas, dengan menjelaskan tahapan-tahapan SDLC seperti pada contoh slide sebelumnya
- Studi kasus ditentukan oleh Dosen dan tidak diperkenankan mengutip atau menulis ulang dari jurnal/artikel yang ada di internet
- Dikumpulkan pada pertemuan berikutnya

Pertemuan 3

**KEBUTUHAN
PERANGKAT LUNAK**

1. REKAYASA KEBUTUHAN

- Kebutuhan untuk suatu sistem adalah deskripsi tentang apa yang harus dilakukan oleh sistem berupa layanan yang diberikan dan kendala dalam operasinya.
- Kebutuhan ini mencerminkan kebutuhan pelanggan untuk sistem yang melayani tujuan tertentu seperti mengontrol perangkat, menempatkan pesanan, atau mencari informasi.
- Proses mencari tahu, menganalisis, mendokumentasikan serta memeriksa layanan dan kendala ini disebut **Rekayasa Kebutuhan (*Requirement Engineering*)**.
- Rekayasa Kebutuhan harus disesuaikan dengan kebutuhan proses, proyek, produk, dan orang-orang yang melakukan pekerjaan.

Jenis Kebutuhan:

1. **Kebutuhan pengguna** adalah pernyataan, dalam bahasa alami ditambah diagram, dari layanan apa yang diharapkan sistem untuk diberikan kepada pengguna sistem dan kendala di mana ia harus beroperasi.
2. **Kebutuhan sistem** adalah deskripsi yang lebih rinci tentang fungsi, layanan, dan kendala operasional sistem perangkat lunak. Dokumen Kebutuhan sistem (kadang-kadang disebut spesifikasi fungsional) harus mendefinisikan secara tepat apa yang akan diimplementasikan. Ini mungkin menjadi bagian dari kontrak antara pembeli sistem dan pengembang perangkat lunak.

Kegiatan pada Rekayasa Kebutuhan:

1. Pengenalan Permasalahan (*Inception*)
2. Pengenalan Lanjutan (*Elicitation*)
3. Elaborasi (*Elaboration*)
4. Negosiasi
5. Spesifikasi (*Specification*)
6. Validasi (*Validation*)
7. Manajemen Kebutuhan (*Requirement Management*)

A. Pengenalan Permasalahan (*Inception*)

- Proyek PL dimulai ketika kebutuhan bisnis atau pasar atau layanan baru telah diidentifikasi atau ditemukan
- Menetapkan pemahaman dasar tentang masalah, siapa yang menginginkan solusi, sifat solusi, serta keefektifan komunikasi dan kolaborasi antara *stakeholder* dengan tim PL.

B. Pengenalan Lanjutan (*Elicitation*)

- Bagian penting dari elisitasi adalah untuk menetapkan tujuan bisnis.
- Masalah yang sering dijumpai:
 - ✓ Lingkup permasalahan: tentang batasan sistem tidak jelas atau rincian teknis yang membingungkan
 - ✓ Permasalahan yang berkaitan dengan pemahaman
 - ✓ Permasalahan yang berkaitan dengan kestabilan

Pengenalan Lanjutan (Lanjutan)

Kegiatan pada tahap ini adalah:

1. Kebutuhan penemuan

adalah proses pengumpulan informasi tentang sistem yang diperlukan dan sistem yang ada, filterisasi pengguna dan kebutuhan sistem.

Sumber informasi selama tahap penemuan kebutuhan termasuk dokumentasi, *stakeholder*, dan spesifikasi sistem.

Pengenalan Lanjutan (Lanjutan)

2. Klasifikasi Kebutuhan dan Organisasi

Kegiatan ini mengumpulkan kebutuhan yang tidak terstruktur dan kebutuhan kelompok yang bersifat koheren.

Cara pengelompokkan kebutuhan menggunakan model arsitektur sistem untuk mengidentifikasi sub-sistem dan untuk menghubungkan kebutuhan dengan masing-masing sub-sistem.

3. Kebutuhan Prioritas dan Negosiasi

Kegiatan ini berkaitan dengan memprioritaskan kebutuhan dan menemukan cara penyelesaian konflik melalui negosiasi.

C. Elaborasi (*Elaboration*)

- Elaborasi dilakukan dengan cara membuat dan penyempurnaan skenario pengguna yang menggambarkan bagaimana pengguna akhir (dan aktor lain) akan berinteraksi dengan sistem.

D. Negosiasi

- Konflik yang terjadi antara pelanggan, pengguna dan *stakeholder* harus didamaikan dengan pendekatan yang bersifat iteratif untuk menentukan skala prioritas kebutuhan, menilai biaya-biaya dan risiko masing-masing.

E. Spesifikasi (*Specification*)

- Spesifikasi kebutuhan adalah proses menuliskan kebutuhan pengguna dan kebutuhan sistem ke dalam dokumen kebutuhan.
- Spesifikasi dapat berupa dokumen tertulis, model grafis, model matematika formal, kumpulan skenario penggunaan sistem/PL, prototipe, atau kombinasi dari semuanya.
- Kebutuhan pengguna menggambarkan kebutuhan fungsional dan non-fungsional sehingga dapat dimengerti oleh pengguna sistem (perilaku eksternal dari sistem) yang tidak memiliki pengetahuan teknis.
- Dokumen kebutuhan tidak boleh menyertakan rincian arsitektur atau desain sistem.

1). Spesifikasi Bahasa

- Bahasa alami telah digunakan untuk menulis kebutuhan PL sejak awal RPL yang bersifat ekspresif, intuitif, dan universal.
- Panduan sederhana:
 1. Buat format standar dan pastikan bahwa semua definisi kebutuhan mematuhi format tsb.
 2. Gunakan bahasa secara konsisten
 3. Gunakan penjelasan teks (tebal, miring, atau warna) untuk memilih bagian-bagian penting dari kebutuhan.
 4. Jangan berasumsi bahwa pembaca memahami bahasa RPL, hindari penggunaan jargon, singkatan, dan akronim.
 5. Sebisa mungkin, harus mencoba mengaitkan alasan dengan setiap kebutuhan pengguna

2). Spesifikasi Struktur

- Bahasa alami terstruktur adalah cara penulisan kebutuhan sistem dimana kebebasan dalam penulisan terbatas dan semua kebutuhan ditulis dengan cara standar.
- Untuk menentukan kebutuhan fungsional, informasi berikut harus disertakan:
 1. Deskripsi fungsi atau entitas yang ditentukan.
 2. Penjelasan tentang input dan dari mana asalnya.
 3. Penjelasan tentang output dan kemana tujuannya.
 4. Informasi yang diperlukan untuk perhitungan atau entitas lain dalam sistem yang digunakan.
 5. Penjelasan tentang tindakan yang harus diambil.
 6. Penjelasan tentang efek samping dari operasi (jika ada)

F. Validasi (*Validation*)

- Validasi kebutuhan adalah proses pengecekan kebutuhan yang benar-benar menentukan sistem yang diinginkan oleh pelanggan.
- Validasi melakukan pemeriksaan untuk memastikan bahwa:
 - ✓ Semua kebutuhan PL telah dinyatakan dengan jelas
 - ✓ Inkonsistensi, kelalaian, dan kesalahan telah terdeteksi dan diperbaiki
 - ✓ Produk kerja sesuai dengan standar yang ditetapkan untuk proses, proyek, dan produk.

Validasi (Lanjutan)

Hal-hal yang perlu pemeriksaan:

1. Pemeriksaan Validitas

Suatu sistem diperlukan untuk melakukan fungsi-fungsi tertentu dan dapat mengidentifikasi fungsi tambahan.

2. Pemeriksaan Konsistensi

Kebutuhan dalam dokumen tidak boleh bertentangan. Artinya, tidak ada batasan yang bertentangan atau deskripsi yang berbeda dari fungsi sistem yang sama.

3. Pemeriksaan Kelengkapan Dokumen

Kebutuhan yang mendefinisikan semua fungsi dan batasan yang dimaksudkan oleh pengguna sistem.

Validasi (Lanjutan)

Hal-hal yang perlu pemeriksaan:

4. Pemeriksaan Realisme

Dengan menggunakan pengetahuan tentang teknologi yang ada, kebutuhan harus diperiksa untuk memastikan bahwa mereka benar-benar dapat diimplementasikan.

5. Verifiability

Kebutuhan sistem harus selalu ditulis sehingga dapat diverifikasi, artinya menulis serangkaian tes yang dapat menunjukkan bahwa sistem yang dikirimkan memenuhi setiap kebutuhan yang ditentukan.

G. Manajemen Kebutuhan *(Requirement Management)*

- Adalah serangkaian kegiatan yang membantu tim proyek untuk mengidentifikasi, mengontrol, dan melacak kebutuhan-kebutuhan dan melacak perubahan terhadap kebutuhan saat proyek berlangsung.
- Ada beberapa alasan mengapa perubahan tidak dapat dihindari:
 1. Lingkungan bisnis dan teknis sistem selalu berubah setelah instalasi.
 2. Pengguna sistem bukan orang yang sama.
 3. Sistem besar biasanya memiliki komunitas pengguna yang beragam

1). Kebutuhan Perencanaan Manajemen

Tahap perencanaan menetapkan tingkat detail manajemen kebutuhan yang diperlukan, dengan memutuskan:

1. Identifikasi Kebutuhan

Setiap kebutuhan harus diidentifikasi secara unik sehingga dapat direferensikan dengan kebutuhan lain dan digunakan dalam pelacakan.

2. Proses manajemen perubahan

Adalah serangkaian kegiatan yang menilai dampak dan biaya perubahan.

Kebutuhan Perencanaan Manajemen (Lanjutan)

3. Kebijakan Pelacakan

Menentukan hubungan antara setiap kebutuhan, kebutuhan dengan desain sistem, dan pemeliharaan catatan-catatan.

4. Dukungan alat

Kebutuhan manajemen melibatkan pemrosesan sejumlah besar informasi tentang kebutuhan. Alat yang dapat digunakan berkisar dari sistem manajemen kebutuhan khusus untuk *spreadsheet* dan sistem basis data sederhana.

2). Kebutuhan Manajemen Perubahan

- Kebutuhan manajemen perubahan harus diterapkan untuk semua perubahan yang diajukan pada kebutuhan sistem setelah dokumen kebutuhan disetujui.
- Ada tiga tahapan utama untuk proses manajemen perubahan:
 1. Analisis masalah dan spesifikasi perubahan
 2. Analisis dan biaya perubahan
 3. Perubahan implementasi

2. KEBUTUHAN FUNGSIONAL DAN NON-FUNGSIONAL

- **Kebutuhan Fungsional** adalah layanan yang harus disediakan sistem, bagaimana sistem harus bereaksi terhadap input tertentu, dan bagaimana sistem harus berperilaku dalam situasi tertentu.
- **Kebutuhan non-fungsional** adalah batasan pada layanan atau fungsi yang ditawarkan oleh sistem.

Termasuk:

- ✓ Kendala waktu
- ✓ Kendala pada proses pengembangan
- ✓ Kendala yang dikenakan oleh standar

A. Kebutuhan Fungsional

- Kebutuhan fungsional menggambarkan apa yang harus dilakukan oleh sistem.
- Kebutuhan ini tergantung pada jenis PL yang dikembangkan, pengguna, dan pendekatan umum yang diambil oleh organisasi saat menulis kebutuhan.
- Kebutuhan pengguna dijelaskan secara abstrak yang dapat dipahami oleh pengguna sistem, terutama untuk kebutuhan sistem yang lebih spesifik menggambarkan fungsi-fungsi sistem, input dan output, dan pengecualian secara rinci

Kebutuhan Fungsional (Lanjutan)

- Contoh: kebutuhan fungsional untuk sistem informasi tentang pasien:
 - ✓ User harus dapat mencari daftar janji untuk semua poliklinik sesuai dengan jadwal praktek dokter.
 - ✓ Setiap hari, untuk setiap poliklinik mencatat daftar pasien yang telah melakukan pendaftaran hari itu.
 - ✓ Setiap petugas yang menggunakan sistem harus diidentifikasi secara unik dengan nomor karyawan delapan digit miliknya.

B. Kebutuhan Non-Fungsional

- Kebutuhan non-fungsional adalah kebutuhan yang tidak secara langsung terkait dengan layanan spesifik yang disampaikan oleh sistem kepada penggunanya.
- Berhubungan dengan sifat sistem yang muncul seperti keandalan, waktu respon, dll.
- Dapat berupa kendala pada implementasi sistem seperti kemampuan perangkat I/O, representasi data yang digunakan dalam antarmuka dengan sistem lain.
- Kebutuhan non-fungsional muncul melalui kebutuhan pengguna, karena keterbatasan anggaran, kebijakan organisasi, kebutuhan interoperabilitas dengan *software/hardware*, atau faktor eksternal seperti peraturan keselamatan atau undang-undang privasi.

Kebutuhan Non-Fungsional (Lanjutan)

Implementasi kebutuhan ini dapat disebarluaskan di seluruh sistem, dengan alasan:

1. Kebutuhan non-fungsional dapat mempengaruhi keseluruhan arsitektur sistem daripada komponen individu. Misalnya, untuk memastikan bahwa terpenuhinya kebutuhan kinerja harus mengatur sistem untuk meminimalkan komunikasi antar komponen.
2. Kebutuhan non-fungsional tunggal, seperti kebutuhan keamanan, dapat menghasilkan sejumlah kebutuhan fungsional terkait yang menentukan layanan sistem baru yang diperlukan.

Karakteristik kebutuhan non-fungsional

1. Kebutuhan produk

Kebutuhan ini menentukan atau membatasi perilaku perangkat lunak.

- Kebutuhan kinerja tentang seberapa cepat sistem harus dijalankan dan berapa banyak memori yang dibutuhkan
- Kebutuhan keandalan yang menetapkan tingkat kegagalan yang dapat diterima
- Kebutuhan keamanan
- Kebutuhan kegunaan

Karakteristik kebutuhan non-fungsional

2. Kebutuhan Organisasi

Adalah kebutuhan sistem yang luas yang berasal dari kebijakan dan prosedur di organisasi pelanggan dan pengembang.

- Kebutuhan operasional yang menentukan bagaimana sistem akan digunakan
- Kebutuhan proses pengembangan yang menentukan bahasa pemrograman, lingkungan pengembangan atau proses standar yang akan digunakan
- Kebutuhan lingkungan yang menentukan lingkungan operasi sistem.

Karakteristik kebutuhan non-fungsional

3. Kebutuhan Eksternal

Mencakup semua kebutuhan yang berasal dari faktor eksternal ke sistem dan proses pengembangannya.

- Kebutuhan peraturan yang mengatur apa yang harus dilakukan untuk sistem yang akan disetujui untuk digunakan oleh regulator, seperti bank sentral
- Kebutuhan legislatif yang memastikan bahwa sistem beroperasi sesuai peraturan
- Kebutuhan etis yang memastikan bahwa sistem akan diterima oleh pengguna dan masyarakat umum

Contoh Requirement

Daftar Isi

Sejarah Revisi

1. Pendahuluan

- 1.1. Kegunaan
- 1.2. Konvensi-Konvensi dalam Dokumen
- 1.3. Pembaca yang Dituju
- 1.4. Lingkup Proyek
- 1.5. Rujukan-Rujukan

2. Deskripsi Keseluruhan

- 2.1. Sudut Pandang Produk
- 2.2. Fitur-Fitur Produk
- 2.3. Kelas-Kelas Pengguna dan Karakteristiknya
- 2.4. Lingkungan Operasional
- 2.5. Perancangan dan Implementasi Batasan-Batasan
- 2.6. Dokumentasi Pengguna
- 2.7. Asumsi dan Kebergantungan

3. Fitur-Fitur Sistem

- 3.1. Fitur Sistem-1
- 3.2. Fitur Sistem-2 (dan selanjutnya)

4. Kebutuhan-Kebutuhan Antarmuka Eksternal

- 4.1. Antarmuka Pengguna
- 4.2. Antarmuka Perangkat Keras
- 4.3. Antarmuka Perangkat Lunak
- 4.4. Komunikasi

5. Kebutuhan-Kebutuhan Non-Fungsional Lainnya

- 5.1. Kebutuhan-Kebutuhan Kinerja
- 5.2. Kebutuhan-Kebutuhan Keamanan
- 5.3. Kebutuhan-Kebutuhan Kualitas PL

6. Kebutuhan-Kebutuhan Lainnya

Lampiran A: Daftar Istilah

Lampiran B: Model-Model Analisis

Lampiran C: Daftar Permasalahan

TUGAS

- Mahasiswa menguraikan Kebutuhan Fungsional dan Non-Fungsional
- Kasus dapat berupa PL desktop/web/mobile yang ditentukan oleh Dosen
- Contoh pada Sistem Tiket Antrian di Bank:
 - A. Kebutuhan Fungsional
 1. Kebutuhan Nasabah
 2. Kebutuhan Bank
 - B. Kebutuhan Non-Fungsional
 1. Kebutuhan Kegunaan
 2. Kebutuhan Performa

LATIHAN

Soal No. 1

1. Definisi Kebutuhan sistem adalah:
 - a. proses pengumpulan informasi tentang sistem yang diperlukan dan sistem yang ada, filterisasi pengguna dan kebutuhan sistem
 - b. Deskripsi tentang apa yang harus dilakukan oleh sistem berupa layanan yang diberikan dan kendala dalam operasinya.
 - c. Proses menuliskan kebutuhan pengguna dan kebutuhan sistem ke dalam dokumen kebutuhan.
 - d. Kebutuhan sistem yang luas yang berasal dari kebijakan dan prosedur di organisasi pelanggan dan pengembang
 - e. Semua kebutuhan yang berasal dari faktor eksternal ke sistem dan proses pengembangannya

Soal No. 2

2. Kebutuhan eksternal yang memastikan bahwa sistem akan diterima oleh pengguna dan masyarakat umum disebut:
 - a. Kebutuhan Legislatif
 - b. Kebutuhan peraturan
 - c. Kebutuhan etis
 - d. Kebutuhan pengembang
 - e. Kebutuhan pengguna

Soal No. 3

3. Definisi Kebutuhan penemuan adalah:
 - a. proses pengumpulan informasi tentang sistem yang diperlukan dan sistem yang ada, filterisasi pengguna dan kebutuhan sistem
 - b. Deskripsi tentang apa yang harus dilakukan oleh sistem berupa layanan yang diberikan dan kendala dalam operasinya.
 - c. Proses menuliskan kebutuhan pengguna dan kebutuhan sistem ke dalam dokumen kebutuhan.
 - d. Kebutuhan sistem yang luas yang berasal dari kebijakan dan prosedur di organisasi pelanggan dan pengembang
 - e. Semua kebutuhan yang berasal dari faktor eksternal ke sistem dan proses pengembangannya

Soal No. 4

4. Layanan yang harus disediakan sistem, bagaimana sistem harus bereaksi terhadap input tertentu, dan bagaimana sistem harus berperilaku dalam situasi tertentu, merupakan definisi dari:
 - a. Kebutuhan Lingkungan
 - b. Kebutuhan Legislatif
 - c. Kebutuhan Sistem
 - d. Kebutuhan Fungsional
 - e. Kebutuhan Non Fungsional

Soal No. 5

5. Proses mencari tahu, menganalisis, mendokumentasikan serta memeriksa layanan dan kendala ini disebut:
 - a. Kebutuhan Sistem
 - b. Kebutuhan Legislatif
 - c. Kebutuhan Non Fungsional
 - d. Kebutuhan Fungsional
 - e. Rekayasa Kebutuhan

Pertemuan 4

KONSEP PERANCANGAN

1. PENDAHULUAN

- Perancangan PL merupakan tempat dimana aturan kreativitas (kebutuhan *stakeholder*, kebutuhan bisnis, dan pertimbangan teknis) semuanya secara bersamaan disatukan untuk membentuk sebuah produk atau sistem/PL.
- Perancangan menciptakan representasi atau model PL.
- Model perancangan memberikan detail tentang arsitektur PL, struktur data, antarmuka, dan komponen yang diperlukan untuk mengimplementasikan sistem.
- Tujuan perancangan PL adalah untuk menghasilkan model atau representasi PL yang memperlihatkan kekuatan, komoditi, dan kenyamanan.

Model Perancangan

1. Perancangan data/kelas

- Mengubah model kelas menjadi realisasi kelas perancangan dan struktur data yang diperlukan untuk mengimplementasikan PL.
- Objek, hubungan dan konten data rinci yang digambarkan oleh atribut kelas dan notasi lainnya memberikan dasar untuk aktivitas perancangan data.
- Bagian dari perancangan kelas dapat terjadi bersamaan dengan perancangan arsitektur PL.
- Perancangan kelas yang lebih rinci terjadi karena setiap komponen PL dirancang.

Model Perancangan

2. Perancangan arsitektur

- Mendefinisikan hubungan antara elemen struktural utama dari PL, gaya dan pola arsitektur yang dapat digunakan untuk mencapai kebutuhan yang ditentukan untuk sistem, dan kendala yang mempengaruhi cara dimana arsitektur dapat diimplementasikan.
- Mewakili perancangan kerangka kerja arsitektur sistem berbasis komputer berasal dari model kebutuhan.

Model Perancangan

3. Perancangan antarmuka

- Menggambarkan bagaimana PL berkomunikasi dengan sistem, dan dengan manusia yang menggunakannya.
- Antarmuka menyiratkan aliran informasi (misal: data atau kontrol) dan jenis perilaku tertentu.
- Perancangan antarmuka pada tingkat komponen mengubah elemen struktural dari arsitektur PL menjadi deskripsi prosedural komponen PL.
- Informasi yang diperoleh dari model berbasis kelas dan model perilaku berfungsi sebagai dasar untuk perancangan komponen.

2. PROSES PERANCANGAN

- Perancangan PL adalah proses yang bersifat iteratif dimana spesifikasi kebutuhan PL diterjemahkan menjadi "*blue print*" untuk membangun PL.
- *Blue print* menggambarkan pandangan holistik PL. Yaitu, perancangan diwakili pada tingkat abstraksi yang tinggi pada tingkat yang dapat langsung ditelusuri ke tujuan sistem dan data yang lebih rinci, fungsional, dan kebutuhan perilaku.
- Saat iterasi perancangan PL berlangsung, penghalusan lebih lanjut akan menggerakkan abstraksi yang lebih rendah.

A. Atribut-Atribut Kualitas PL

Tiga karakteristik umum yang berfungsi sebagai panduan untuk evaluasi perancangan yang baik:

1. Perancangan PL harus menerapkan semua spesifikasi kebutuhan yang secara eksplisit ada dalam model kebutuhan, dan mengakomodasi semua spesifikasi kebutuhan implisit yang diinginkan oleh *stakeholder*.
2. Perancangan PL harus menghasilkan produk kerja yang mudah dibaca dan dimengerti bagi mereka yang membuat kode-kode program dan yang akan melakukan pengujian untuk kemudian mendukung PL.
3. Perancangan PL seharusnya menyediakan gambaran lengkap tentang PL, menangani permasalahan data, fungsional, dan perilaku dari perspektif implementasi.

Panduan Kualitas PL

Tujuannya untuk mengevaluasi kualitas representasi perancangan.

1. Rancangan PL harus menunjukkan arsitektur yang:

- telah dibuat menggunakan gaya atau pola arsitektur yang dapat dikenali
- tersusun atas komponen-komponen yang menunjukkan karakteristik perancangan yang baik
- dapat diimplementasikan secara evolusioner, dan memfasilitasi implementasi dan pengujian.

2. Rancangan PL harus bersifat modular, artinya PL harus secara logis menjadi bagian dalam elemen-elemen atau subsistem.

Panduan Kualitas PL (Lanjutan)

3. Rancangan PL harus berisi representasi data, arsitektur, objek-objek, antarmuka, dan komponen yang berbeda.
4. Rancangan PL harus memuat struktur data yang sesuai untuk kelas yang akan diimplementasikan dan digambarkan dari pola-pola data yang dapat dikenali.
5. Rancangan PL harus mengarah pada komponen yang menunjukkan karakteristik fungsional yang bersifat independen.
6. Rancangan PL harus memuat antarmuka yang mengurangi kompleksitas hubungan antar komponen dan dengan lingkungan eksternal.

Panduan Kualitas PL (Lanjutan)

7. Rancangan PL harus diturunkan dari metode perulangan yang dikendalikan oleh informasi yang diperoleh selama analisis kebutuhan PL.
8. Sebuah perancangan harus direpresentasikan menggunakan notasi yang secara efektif mengkomunikasikan maknanya.

Atribut-Atribut Kualitas

1. Fungsionalitas

Dinilai dengan mengevaluasi sejumlah fitur dan kemampuan program, fungsi-fungsi umum yang disampaikan, dan keamanan sistem secara keseluruhan.

2. Peggunaan

Dinilai dengan mempertimbangkan faktor manusia, estetika, konsistensi, dan dokumentasi.

3. Keandalan

Dievaluasi dengan mengukur frekuensi dan tingkat keparahan kegagalan, akurasi hasil keluaran, waktu antar kegagalan, *recovery*, dan prediktabilitas program.

Atribut-Atribut Kualitas (Lanjutan)

4. Kinerja

Diukur menggunakan kecepatan pemrosesan, waktu tanggap, konsumsi sumber daya, *throughput*, dan efisiensi.

5. Daya dukung

Menggabungkan kemampuan program untuk dikembangkan, kemampuan beradaptasi, dan kemampuan melayani kebutuhan pengguna, kemampuan uji, kompatibilitas, konfigurabilitas (kemampuan untuk mengatur dan mengontrol elemen-elemen konfigurasi PL).

3. KONSEP-KONSEP PERANCANGAN

Konsep perancangan pada dasarnya akan menyediakan kebutuhan:

- Kriteria yang digunakan untuk membagi PL menjadi komponen yang bersifat mandiri
- Rincian fungsi/struktur data yang dapat dipisahkan dari representasi konseptual
- Kriteria yang digunakan untuk mendefinisikan kualitas teknis perancangan PL

A. Abstraksi

- Pada tingkat abstraksi tertinggi, penyelesaian masalah dinyatakan dalam istilah luas menggunakan bahasa pada lingkungan permasalahan.
- Pada tingkat abstraksi lebih rendah, deskripsi yang lebih rinci dari penyelesaian masalah harus diberikan.
 - Terminologi berorientasi masalah digabungkan dengan terminologi berorientasi implementasi dengan tujuan untuk lebih dapat menyelesaikan permasalahan
 - Pada tingkat abstraksi yang paling rendah, penyelesaian masalah dapat langsung diimplementasikan menggunakan bahasa pemrograman yang dipilih.

B. Arsitektur

- Arsitektur sistem/PL merupakan struktur keseluruhan PL dan cara bagaimana struktur tersebut memberikan integritas konseptual untuk suatu sistem/PL.
- Arsitektur sistem/PL juga merupakan struktur/organisasi dari komponen program (modul) yang menjelaskan bagaimana komponen-komponen tersebut berinteraksi, dan struktur data yang digunakan oleh komponen.

Arsitektur (Lanjutan)

Properti sebagai bagian dari perancangan arsitektur:

- **Properti Struktural**

Mendefinisikan komponen sistem (modul, objek, filter) dan menjelaskan bagaimana komponen-komponen tersebut dikemas dan berinteraksi satu sama lain.

- **Properti Fungsional Tambahan**

Deskripsi perancangan arsitektural seharusnya bisa menyelesaikan permasalahan tentang bagaimana arsitektur perancangan mencapai kebutuhan akan kinerja, kapasitas, keandalan, keamanan, kemampuan beradaptasi, dan karakteristik lainnya.

- **Keluarga sistem yang berhubungan**

Perancangan arsitektural seharusnya dibuat melalui pola perulangan/iterasi.

C. Pola (*Pattern*)

- Pola adalah suatu wawasan yang di dalamnya memuat esensi dari solusi yang terbukti untuk permasalahan perancangan PL dalam konteks tertentu [Brad Appleton].
- Tujuan dari setiap pola perancangan adalah untuk memberikan deskripsi yang memungkinkan perancang untuk menentukan:
 1. Apakah pola berlaku untuk pekerjaan saat ini
 2. Apakah pola dapat digunakan kembali
 3. Apakah pola dapat berfungsi sebagai panduan untuk mengembangkan pola yang serupa, tetapi berbeda secara fungsional atau struktural

D. Pemisahan Perhatian

- Pemisahan perhatian adalah konsep perancangan yang menunjukkan bahwa masalah yang kompleks dapat diselesaikan jika permasalahan itu dibagi menjadi bagian-bagian yang lebih kecil yang lebih mudah diselesaikan dan/atau dioptimalkan secara independen.
- Pemisahan perhatian dapat diimplementasikan menggunakan konsep perancangan PL yang saling berhubungan seperti: modularitas, aspek-aspek, kemandirian fungsional, dan penghalusan.

E. Modularitas

- Modularitas adalah atribut tunggal dari PL yang memungkinkan suatu program untuk dapat dikelola secara cerdas.
- PL monolitik adalah program besar yang terdiri dari satu modul.
- PL monolitik tidak dapat dengan mudah dipahami oleh rekayasaawan PL. Jumlah lintasan kendali, rentang referensi, jumlah variabel, dan kompleksitas keseluruhan hampir tidak mungkin dimengerti.

Modularitas (Lanjutan)

- Lebih banyak modul berarti ukuran modul semakin lebih kecil.
- Bertambahnya jumlah modul, maka upaya (biaya) yang terkait dengan pengintegrasian modul juga bertambah.
- Tujuan Modularitas:
 - a. Pengembangan PL dapat lebih mudah direncanakan
 - b. Versi-versi PL dapat didefinisikan
 - c. Perubahan-perubahan dapat dengan mudah diakomodasikan
 - d. *Testing* dan *debugging* dapat dilakukan secara lebih efisien
 - e. Mudah dalam pemeliharaan PL

F. Penyembunyian Informasi

- Maksudnya bahwa Modul PL harus dispesifikasi dan dirancang sedemikian rupa sehingga informasi (algoritma dan data) yang terdapat dalam modul tidak dapat diakses oleh modul lain yang tidak memerlukan informasi tersebut.
- Penyembunyian informasi (*Information hiding*) menyiratkan bahwa modularitas yang efektif dapat dicapai dengan mendefinisikan sejumlah modul independen yang berkomunikasi satu sama lain dalam hal informasi yang diperlukan untuk mencapai fungsi PL.
- Manfaatnya ketika modifikasi tertentu perlu dilakukan selama pengujian PL dan selama pemeliharaan PL.

G. Independensi Fungsional

- Konsep independensi fungsional adalah hasil langsung dari pemisahan masalah, modularitas, dan konsep abstraksi dan penyembunyian informasi.
- Independensi fungsional dicapai dengan cara mengembangkan modul yang memiliki fungsi tunggal (*single-minded*) dan memiliki interaksi yang bersifat tertutup dengan modul lain.
- Modul independen lebih mudah dikembangkan karena fungsi-fungsi di dalamnya dapat dilokalisasi dan antarmuka disederhanakan.
- Modul independen lebih mudah dipelihara dan diuji

Independensi Fungsional (Lanjutan)

Modul independensi mempunyai kriteria kualitatif:

1. Kohesivitas

- Adalah indikasi dari kekuatan fungsional suatu modul.
- Umumnya melakukan pekerjaan tunggal dan hanya memerlukan sedikit interaksi dengan komponen lainnya

2. Kebergantungan (*Coupling*)

- Adalah indikasi dari kemandirian antar modul.
- *Coupling* tergantung pada kompleksitas antarmuka yang menghubungkan modul-modul yang ada dalam program, yaitu titik masuk suatu modul, dan data apa yang melewati antarmuka modul.

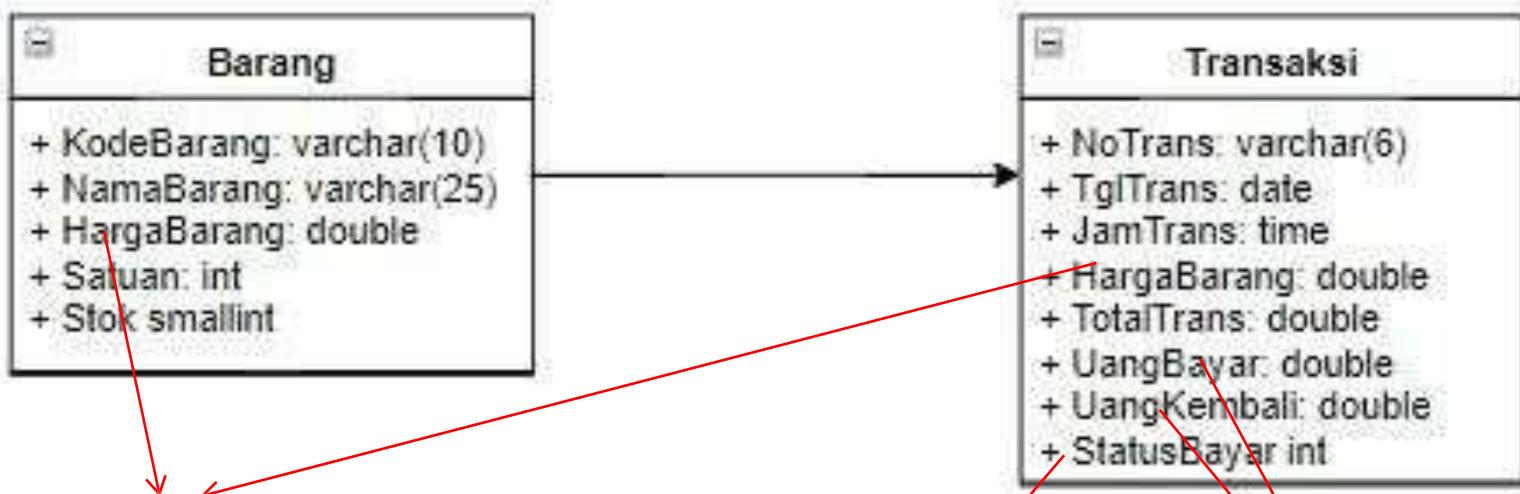
H. Penghalusan

- Penghalusan merupakan proses elaborasi yang dimulai dengan pernyataan fungsi (atau deskripsi informasi) yang didefinisikan pada tingkat abstraksi yang tinggi.
- Penghalusan langkah demi langkah menggunakan strategi perancangan *top-down*.
- Hirarki PL dikembangkan dengan melakukan dekomposisi pernyataan fungsi yang bersifat makro (abstraksi prosedural) secara bertahap hingga pernyataan dalam bahasa pemrograman dapat tercapai.
- Penghalusan membantu untuk mendapatkan rincian pada tingkat rendah saat perancangan berlangsung.

I. Refaktorisasi

- Refaktorisasi adalah teknik reorganisasi perancangan PL yang bertujuan untuk menyederhanakan perancangan (atau kode) dari komponen tanpa harus mengubah fungsi atau perilakunya.
- Refaktorisasi PL, diperiksa kembali untuk:
 - a. Menemukan redundansi
 - b. Menemukan elemen perancangan yang tidak digunakan
 - c. Menemukan algoritma yang tidak efisien (tidak perlu)
 - d. Menemukan struktur data yang konstruksinya buruk
 - e. Menemukan kegagalan perancangan lainnya yang dapat diperbaiki untuk menghasilkan perancangan yang lebih baik.

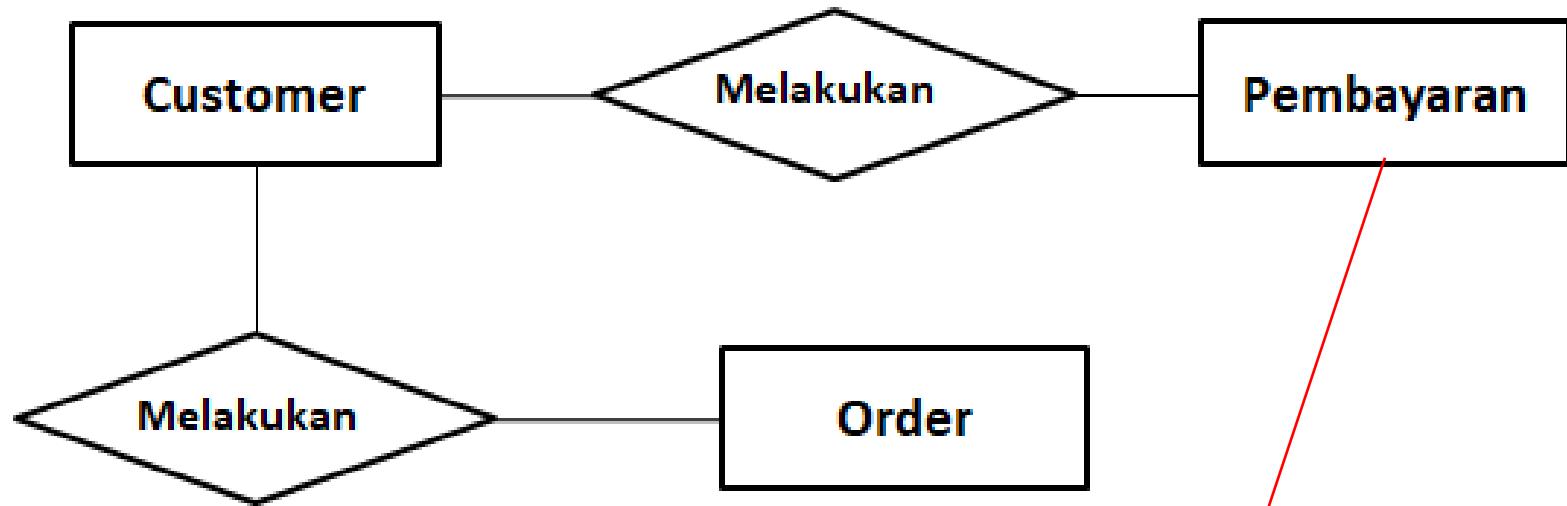
Contoh:



Pendefinisian data yang menyebabkan redundansi data pada tabel Transaksi.
Harga barang dituliskan pada kedua tabel, bisa saja dengan kode yang sama tetapi harga berbeda.

Elemen data yang tidak berfungsi di dalam sistem
Elemen data StatusBayar yang tidak perlu digunakan, menyebabkan ambigu

Contoh:



Konstruksi database yang tidak baik.
Customer dapat melakukan pembayaran padahal belum melakukan Transaksi (Order)

Pertemuan 5

PEMODELAN SISTEM dengan UML (bagian 1)

1. *Unified Modelling Language*

- Menurut Booch, et.al. UML adalah bahasa standar untuk menulis *blue print* PL. UML dapat digunakan untuk memvisualisasikan, menentukan, membuat, dan mendokumentasikan artefak dari sistem PL secara intensif.
- UML sesuai untuk sistem pemodelan mulai dari sistem informasi perusahaan, aplikasi berbasis web yang terdistribusi, bahkan sampai sistem *real time embedded* yang sulit.
- UML adalah proses yang independen, walaupun secara optimal harus digunakan dalam proses yang menggunakan *case driven, architecture-centric, iterative, dan incremental*.

UML adalah bahasa untuk:

a. *Visualizing*

Beberapa hal dimodelkan secara tekstual atau dengan model grafis. UML adalah bahasa grafis yang menggunakan sekelompok simbol grafis. Setiap simbol dalam notasi UML didefinisikan dengan baik secara semantik, sehingga pengembang dapat menulis model UML dan dapat menafsirkan model itu dengan jelas.

b. *Specifying*

UML dapat membangun model yang tepat, tidak ambigu, dan lengkap. UML membahas spesifikasi semua keputusan analisis, perancangan, dan implementasi penting yang harus dilakukan dalam mengembangkan dan menerapkan sistem PL yang intensif.

UML adalah bahasa untuk (lanjutan)

c. *Constructing*

UML bukan bahasa pemrograman visual, namun modelnya bisa langsung terhubung ke berbagai bahasa pemrograman, dan memungkinkan untuk memetakan ke bahasa pemrograman seperti Java, C ++, atau Visual Basic, atau bahkan ke tabel dalam basis data relasional atau penyimpanan database berorientasi objek yang tetap.

d. *Documenting*

UML membahas dokumentasi arsitektur sistem dan semua detailnya. UML menyediakan bahasa untuk mengekspresikan persyaratan dan tes. UML juga menyediakan bahasa untuk memodelkan kegiatan perencanaan proyek.

UML versi 1.0 dibagi menjadi 2 kelompok:

A. Diagram Struktur (*Structural Diagrams*)

1. Class Diagram
2. Object Diagram
3. Component Diagram
4. Deployment Diagram

B. *Behavioral Diagrams*

1. Use Case Diagram
2. Sequence Diagram
3. Activity Diagram
4. Statechart Diagram
5. Collaboration Diagram

Diagram-Diagram dalam UML

1. *Class Diagram*

Menunjukkan seperangkat kelas, antarmuka, dan kolaborasi dan hubungan di antara mereka. Class Diagram membahas desain statis dari suatu sistem.

2. *Object Diagram*

Menunjukkan satu set objek dan hubungan antara objek. Diagram objek memodelkan *instance* dari hal-hal yang terdapat dalam *Class Diagram*. Diagram objek digunakan untuk memodelkan desain statis suatu sistem, untuk memvisualisasikan, menentukan, dan mendokumentasikan model struktural, dan membangun aspek statis sistem melalui teknik maju (*forward*) dan mundur (*reverse*).

Diagram-Diagram dalam UML (Lanjutan)

3. *Component Diagram*

Menunjukkan organisasi dan ketergantungan antar sekumpulan komponen.

4. *Deployment Diagram*

Menunjukkan konfigurasi komponen dalam proses eksekusi aplikasi. Diagram ini terdiri dari node yang merupakan perangkat keras dan membungkus satu atau lebih komponen.

5. *Use Case Diagram*

Menunjukkan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Diagram ini sangat penting dalam mengatur dan memodelkan perilaku suatu sistem.

Diagram-Diagram dalam UML (Lanjutan)

6. *Sequence Diagram*

Diagram interaksi yang menekankan urutan waktu pada pesan. Menempatkan objek yang berpartisipasi dalam interaksi pada sumbu X dan menempatkan pesan antar objek sepanjang sumbu Y, sedangkan waktu digambarkan dari atas ke bawah.

7. *Activity Diagram*

Diagram yang menunjukkan arus dari aktivitas ke aktivitas dalam suatu sistem. *Activity Diagram* membahas pandangan dinamis suatu sistem, dan sangat penting dalam pemodelan fungsi suatu sistem dan menekankan aliran kontrol antar objek.

Diagram-Diagram dalam UML (Lanjutan)

8. *Statechart Diagram*

Menggambarkan perubahan status atau transisi status dari sebuah mesin atau sistem atau objek. *Statechart* sangat penting dalam memodelkan perilaku antarmuka, kelas, atau kolaborasi dan menekankan perilaku dari suatu objek, yang sangat berguna dalam memodelkan sistem reaktif.

9. *Collaboration Diagram*

Diagram interaksi yang menekankan struktur organisasi objek yang mengirim dan menerima pesan. Diagram ini merupakan perluasan dari diagram objek, yaitu memberikan tambahan asosiasi antara objek, dan menunjukkan objek mengirimkan *message* ke objek-objek yang lain.

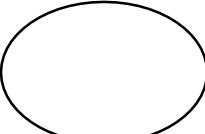
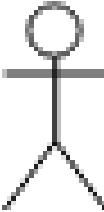
2. USE CASE DIAGRAM

- Use Case Diagram digunakan untuk menggambarkan serangkaian tindakan (*use cases*) bahwa sistem dapat melakukan interaksi di luar sistem (aktor) dengan sistem itu sendiri (abstraksi).
- Use Case juga digunakan untuk mengetahui fungsi apa saja yang ada dalam sebuah sistem dan siapa yang berhak menggunakan fungsi-fungsi itu.
- Nama Use Case didefinisikan semudah mungkin dan dapat dipahami.
- Dua hal utama pada Use Case yaitu pendefinisian aktor dan use case.

USE CASE DIAGRAM (Lanjutan)

- Use case digunakan untuk
 - a. Merepresentasikan interaksi sistem – pengguna
 - b. Mendefinisikan dan mengatur persyaratan fungsional dalam suatu sistem
 - c. Menentukan konteks dan persyaratan sistem
 - d. Memodelkan aliran dasar peristiwa dalam use case

Simbol-Simbol Use Case

No	Nama	Simbol	Keterangan
1.	Use Case		<ul style="list-style-type: none">• Digambarkan dengan elips horizontal• Nama Use case menggunakan kata kerja
2.	Aktor		<ul style="list-style-type: none">• Menggambarkan orang, system/external entitas yang menyediakan atau menerima informasi• Merupakan lingkungan luar dari sistem• Nama Aktor menggunakan Kata benda• Aktor utama digambarkan pada pojok kiri atas dari diagram

Simbol-Simbol Use Case

No	Nama	Simbol	Keterangan
3.	Asosiasi	—	<ul style="list-style-type: none">• Menggambarkan bagaimana aktor berinteraksi dengan use case• Bukan menggambarkan aliran data/informasi
4.	Generalisasi	↑	<ul style="list-style-type: none">• Gambarkan generalisasi antara use case atau antara aktor dengan panah tertutup yang mengarah dari <i>child</i> ke <i>parent</i>

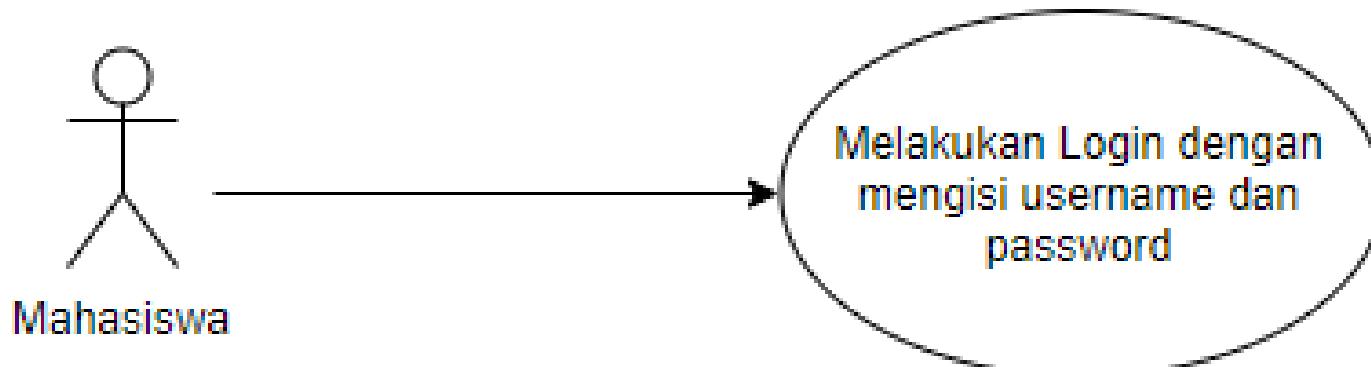
Simbol-Simbol Use Case

No	Nama	Simbol	Keterangan
5.	Relasi include		<ul style="list-style-type: none">Hubungan antara dua use case untuk menunjukkan adanya perilaku use case yang dimasukkan ke dalam perilaku dari <i>base use case</i>Tanda panah terbuka harus terarah ke sub use case
	Relasi extend		<ul style="list-style-type: none">Perluasan dari use case lain (<i>optional</i>)Tanda panah terbuka harus terarah ke base use case
6.	Boundary Boxes		<ul style="list-style-type: none">Untuk memperlihatkan batasan sistem dengan lingkungan luar sistem

Contoh penggunaan relasi *include* dan *extend*

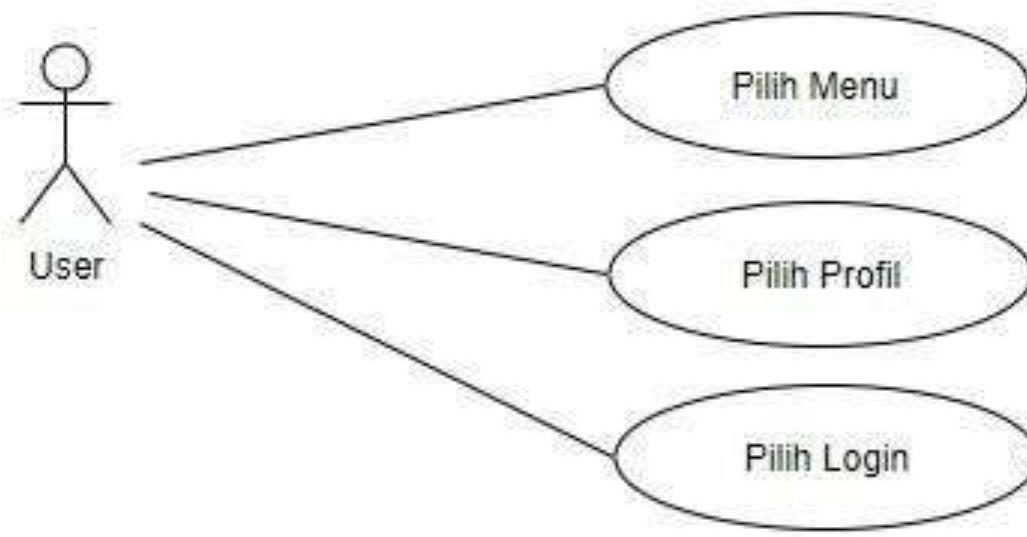


Contoh Penggambaran Use Case (1)



- Pada gambar di atas, penulisan nama use case **tidak dituliskan secara berlebihan**, seharusnya hanya bisnis proses saja yang dituliskan yaitu LOGIN
- Asosiasi aktor dengan use case sebaiknya tidak menggunakan tanda panah, kecuali untuk penggunaan relasi include/extend, dan generalisasi

Contoh Penggambaran Use Case (2)



Pada gambar di atas, penulisan nama use case **Pilih Menu**, **Pilih Profil**, **Pilih Login**, tidak seharusnya ditulis demikian. Use case dituliskan dengan bisnis proses, bukan aksi seperti dalam Activity Diagram

Contoh

Berikut contoh pembuatan Use Case untuk pembelian tiket online

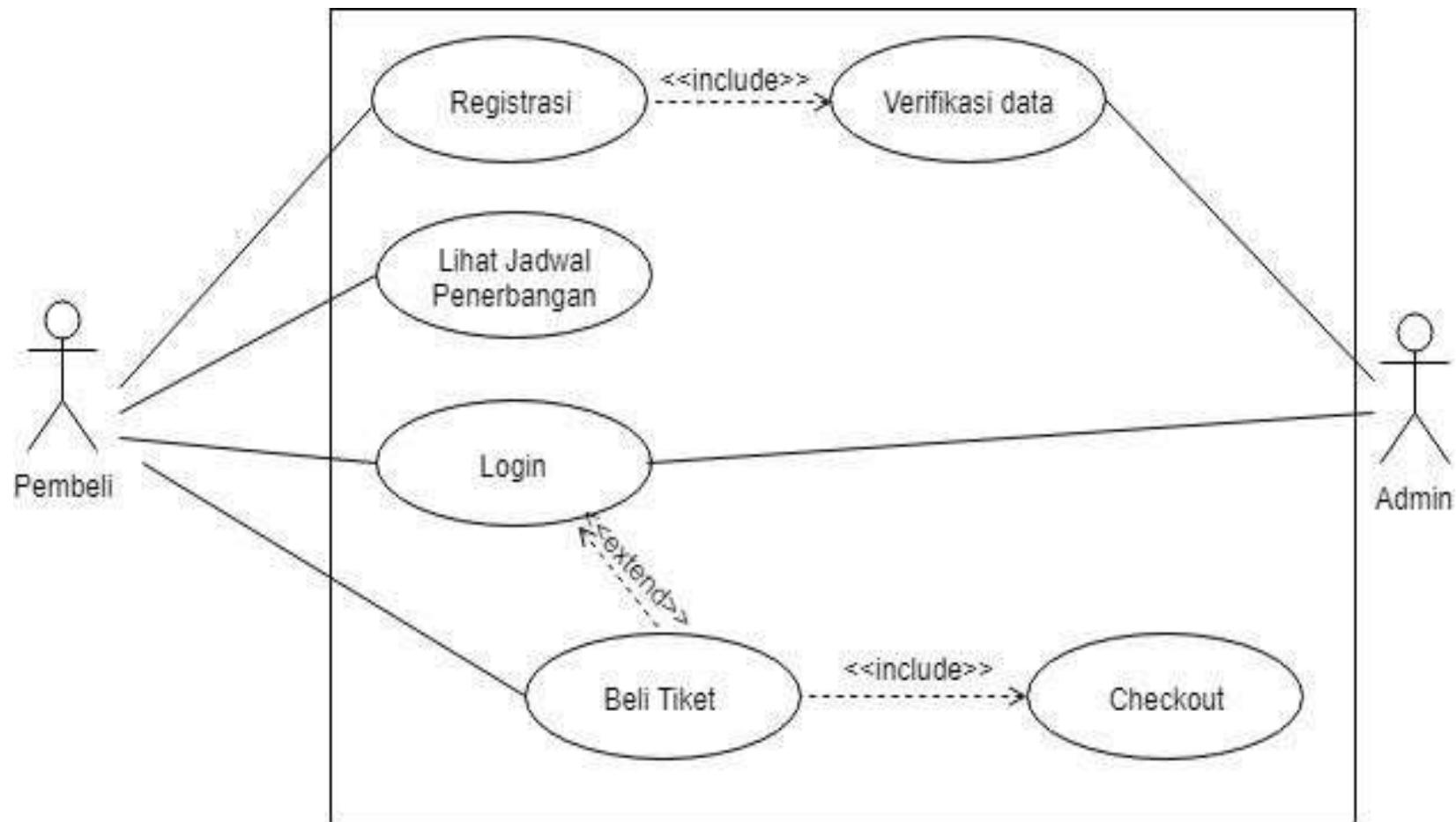
1.a. Daftar Kebutuhan Fungsional Use Case

No	Use Case	Aktor	Deskripsi
1	Registrasi	Pembeli	Use case ini berfungsi untuk proses pendaftaran sebagai pembeli
2	Verifikasi data	Admin	Use case ini berfungsi untuk melakukan pengecekan data pembeli
3	Lihat Jadwal Penerbangan	Pembeli	Use case ini berfungsi untuk melihat jadwal penerbangan oleh pembeli
4	Login	Pembeli, Admin	Use case ini berfungsi untuk masuk ke dalam sistem untuk pembeli dan admin
5	Beli Tiket	Pembeli	Use case ini berfungsi untuk membeli tiket penerbangan

1.b. Daftar Kebutuhan Fungsional Aktor

No	Aktor	Deskripsi
1	Pembeli	Pembeli adalah aktor yang dapat melakukan Registrasi, melihat jadwal penerbangan, melakukan login, membeli tiket, dan checkout
2	Admin	Admin adalah aktor yang dapat melakukan verifikasi data pembeli dan melakukan login

2. Use Case Pembelian Tiket Online



3. Spesifikasi Use Case

Nama Use Case	Beli Tiket
Deskripsi	Use case ini menyediakan layanan pembelian tiket penerbangan
Aktor	Pembeli
Pre-Condition	Login
Post-Condition	<ul style="list-style-type: none">• Pembeli melengkapi data penumpang• Pembeli memilih cara pembayaran• Pembeli melakukan checkout
Relasi	Entend dari Login

4. Skenario Use Case

Aksi Aktor	Reaksi Sistem
1. Pembeli melakukan registrasi	2. Sistem menyimpan data registrasi
3. Pembeli melihat jadwal penerbangan	4. Sistem akan menampilkan jadwal penerbangan berikut harga tiket yang ditawarkan
5. Pembeli membeli tiket penerbangan	6. Sistem menampilkan form data penumpang yang harus diisi oleh pembeli
7. Pembeli mengisi data penumpang	8. Sistem menampilkan halaman untuk memilih cara pembayaran

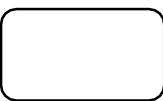
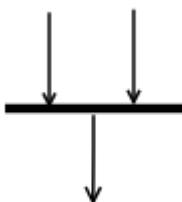
4. Skenario Use Case (Lanjutan)

Aksi Aktor	Reaksi Sistem
9. Pembeli memilih cara pembayaran	
	10. Sistem menampilkan halaman checkout 11. Sistem mengirimkan notifikasi pembelian tiket melalui email
12. Pembeli melakukan pembayaran 13. Pembeli melakukan konfirmasi pembayaran	
	14. Sistem mengirimkan e-tiket melalui email

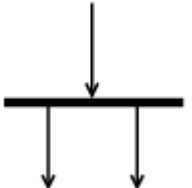
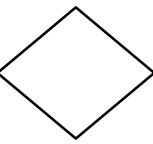
3. ACTIVITY DIAGRAM

- Activity Diagram adalah teknik untuk menggambarkan logika prosedural, proses bisnis, dan jaringan kerja antara pengguna dan sistem.
- Menggunakan notasi yang **mirip flowchart**, meskipun terdapat sedikit perbedaan notasi karena diagram ini mendukung behavior paralel.
- Activity diagram dibuat berdasarkan **sebuah atau beberapa use case** pada use case diagram
- Memungkinkan melakukan proses untuk memilih urutan dalam melakukannya atau hanya menyebutkan aturan-aturan rangkaian dasar yang harus diikuti, karena proses-proses sering muncul secara paralel.

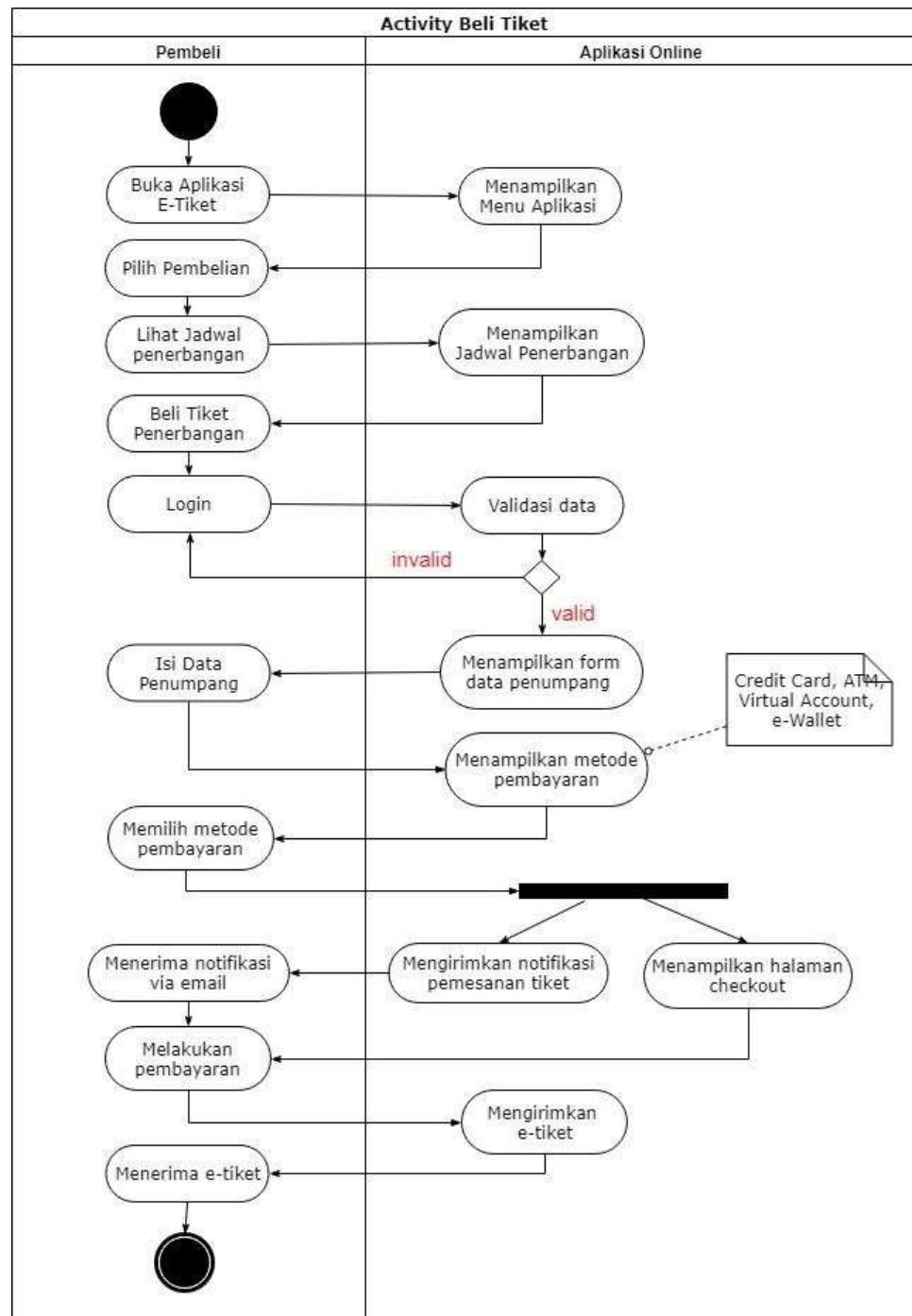
Simbol-Simbol Activity Diagram

No	Nama	Simbol	Keterangan
1.	Start		<ul style="list-style-type: none">Menjelaskan awal proses kerja dalam activity diagramHanya ada satu simbol start
2.	End		<ul style="list-style-type: none">Menandai kondisi akhir dari suatu aktivitas dan merepresentasikan penyelesaian semua arus prosesBisa lebih dari satu simbol end
3.	Activity		<ul style="list-style-type: none">Menunjukkan kegiatan yang membentuk proses dalam diagram
4.	Join		<ul style="list-style-type: none">Menggabungkan dua atau lebih aktivitas bersamaan dan menghasilkan hanya satu aktivitas yang terjadi dalam satu waktu

Simbol-Simbol Activity Diagram

No	Nama	Simbol	Keterangan
5.	Fork		<ul style="list-style-type: none">Membagi aliran aktivitas tunggal menjadi beberapa aktivitas bersamaan
6.	Decision		<ul style="list-style-type: none">Mewakili keputusan yang memiliki setidaknya dua jalur bercabang yang kondisinya sesuai dengan opsi pencabangan
7.	Connector		<ul style="list-style-type: none">Menunjukkan arah aliran atau aliran kontrol dari aktivitas
8.	Swimlane		<ul style="list-style-type: none">Cara untuk mengelompokkan aktivitas berdasarkan aktorMenggunakan garis vertikal

Contoh Activity Diagram



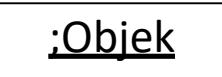
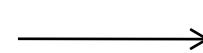
4. SEQUENCE DIAGRAM

- *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* menggambarkan interaksi yang fokusnya pada urutan pesan yang dipertukarkan, bersama dengan spesifikasi kemunculannya yang sesuai pada garis hidup.
- *Sequence diagram* terdiri atas dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).
- Diagram ini secara khusus berasosiasi dengan *use case* diagram

Fungsi Sequence Diagram:

- a. Menentukan detail dari *Use Case*.
- b. Memodelkan logika prosedur, fungsi, atau operasi yang terdapat dalam sistem.
- c. Untuk melihat bagaimana objek dan komponen saling berinteraksi satu sama lain untuk menyelesaikan suatu proses.
- d. Merencanakan dan memahami fungsionalitas secara rinci dari skenario yang ada atau yang akan datang.

Simbol Sequence Diagram

No	Simbol	Nama	Fungsi
1		Object	Komponen utama Sequence Diagram
2		Actor	Menggambarkan orang yang sedang berinteraksi dengan sistem
3		Entity Class	Menggambarkan hubungan kegiatan yang akan dilakukan
4		Boundary Class	Menggambarkan sebuah penggambaran dari form
5		Control Class	Menggambarkan penghubung antara boundary dengan tabel
6		Life Line	Menggambarkan tempat mulai dan berakhirnya sebuah message
7		Message	Menggambarkan pengiriman pesan

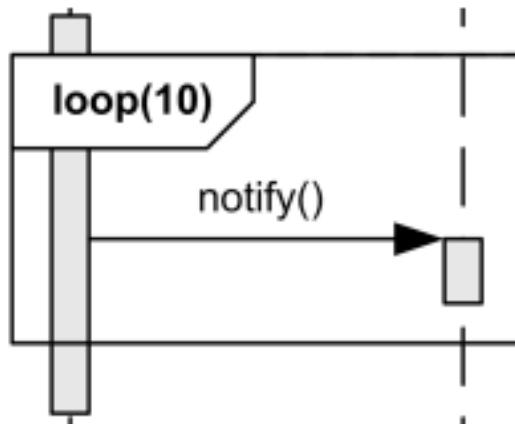
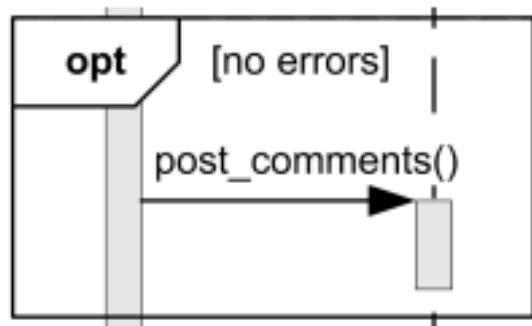
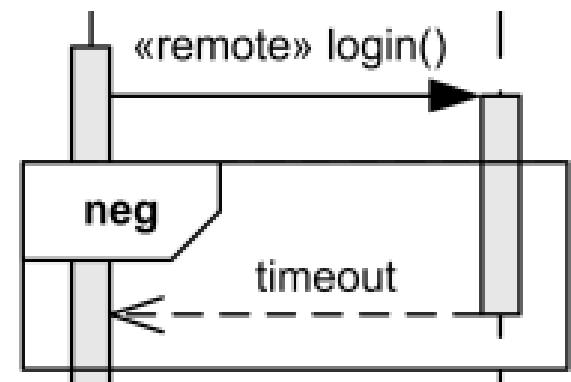
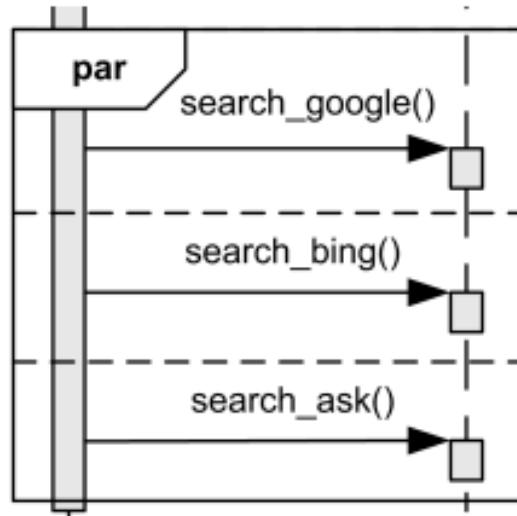
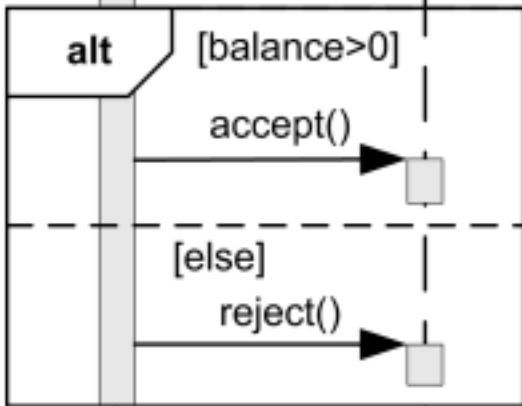
Loop dan Kondisi

- Loop dan kondisi menggunakan kerangka interaksi, yaitu cara penandaan sebuah bagian *sequence diagram*.
- Kerangka terdiri dari beberapa daerah yang dipisahkan menjadi beberapa ***fragmen***.
- Setiap kerangka memiliki sebuah ***operator***.
- Setiap *fragmen* memiliki sebuah ***guard***.
- *Guard* merupakan sebuah ekspresi kondisional dalam tanda kurung [], dan menunjukkan bahwa pesan akan dikirimkan jika nilai *guard* benar.

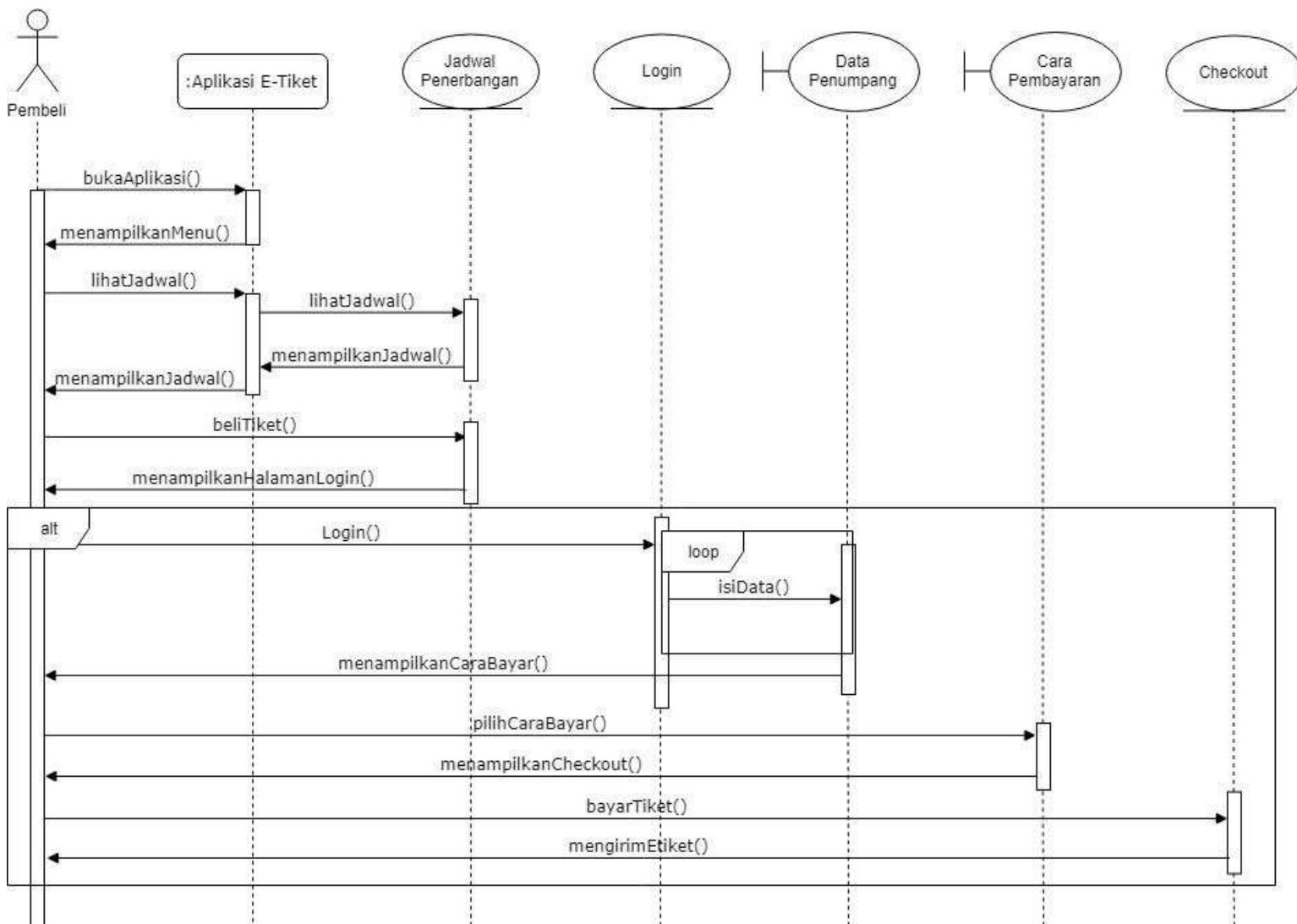
Operator Umum untuk Kerangka Interaksi

Operator	Keterangan
alt	Alternatif dari banyak fragmen. Hanya yang kondisinya true yang akan dijalankan
opt	Optional; fragmen akan dijalankan jika kondisi yang mendukungnya true
par	Paralel; setiap fragmen dijalankan secara paralel
loop	Looping, fragmen mungkin dijalankan berulang kali dan guard menunjukkan basis iterasi
region	Critical region; fragmen hanya dapat mempunyai satu thread untuk menjalankannya
neg	Negatif; fragmen menunjukkan interaction yang salah
ref	Reference; menunjukkan ke sebuah interaction yang didefinisikan pada diagram yang lain
sd	Sequence diagram

Contoh penggunaan operator:



Contoh Sequence Diagram



Latihan

Dosen memberikan contoh kasus, kemudian memodelkan kasus tersebut dengan diagram-diagram yang sudah dibahas

Pertemuan 6

PEMODELAN SISTEM dengan UML (bagian 2)

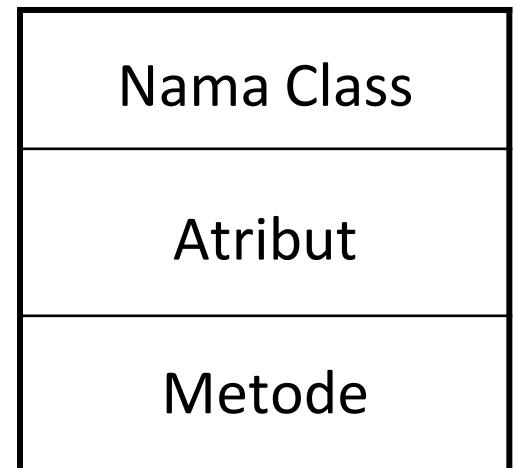
5. CLASS DIAGRAM

- Class Diagram adalah salah satu jenis diagram yang paling berguna dalam UML karena diagram tersebut secara jelas memetakan struktur sistem tertentu dengan memodelkan kelas, atribut, operasi, dan hubungan antar objek
- Mendeskripsikan jenis objek dalam sistem dan berbagai macam hubungan statis yang terdapat diantara objek.
- Menunjukkan properti dan operasi sebuah class dan batasan-batasan yang terdapat dalam hubungan objek tersebut.

CLASS DIAGRAM (Lanjutan)

A. Properti

- ✓ Merupakan sebuah konsep tunggal, tetapi tampak seperti dua notasi yang berbeda: atribut dan asosiasi.
- ✓ Mewakili fitur-fitur struktural dari sebuah class.
- Class berbentuk persegi panjang dengan tiga baris: nama class, atribut class, dan metode atau operasi yang mungkin digunakan class.
- Class dan subclass dikelompokkan bersama untuk menunjukkan hubungan statis antar objek



B. Atribut

Notasi atribut mendeskripsikan properti dengan sebaris teks di dalam kotak class

BU:

visibility name: type multiplicity = default (property-string)

Ket:

- Visibility: menandakan atribut public (+), private (-), dan protected (#)
- Name: nama atribut
- Type: batasan tentang objek yang dapat diletakkan dalam atribut
- default value: nilai dari objek
- (property-string): properti tambahan

C. Assosiasi

- Merupakan garis solid antara dua class, ditarik dari class sumber ke class target

D. Multiplicity

- Merupakan indikasi tentang berapa banyak objek yang bisa berhubungan dengan objek lain.
- Bentuk multiplicity:
 - $1 \rightarrow$ hanya satu
 - $0..1 \rightarrow$ nol atau satu
 - $* \rightarrow$ lebih dari satu

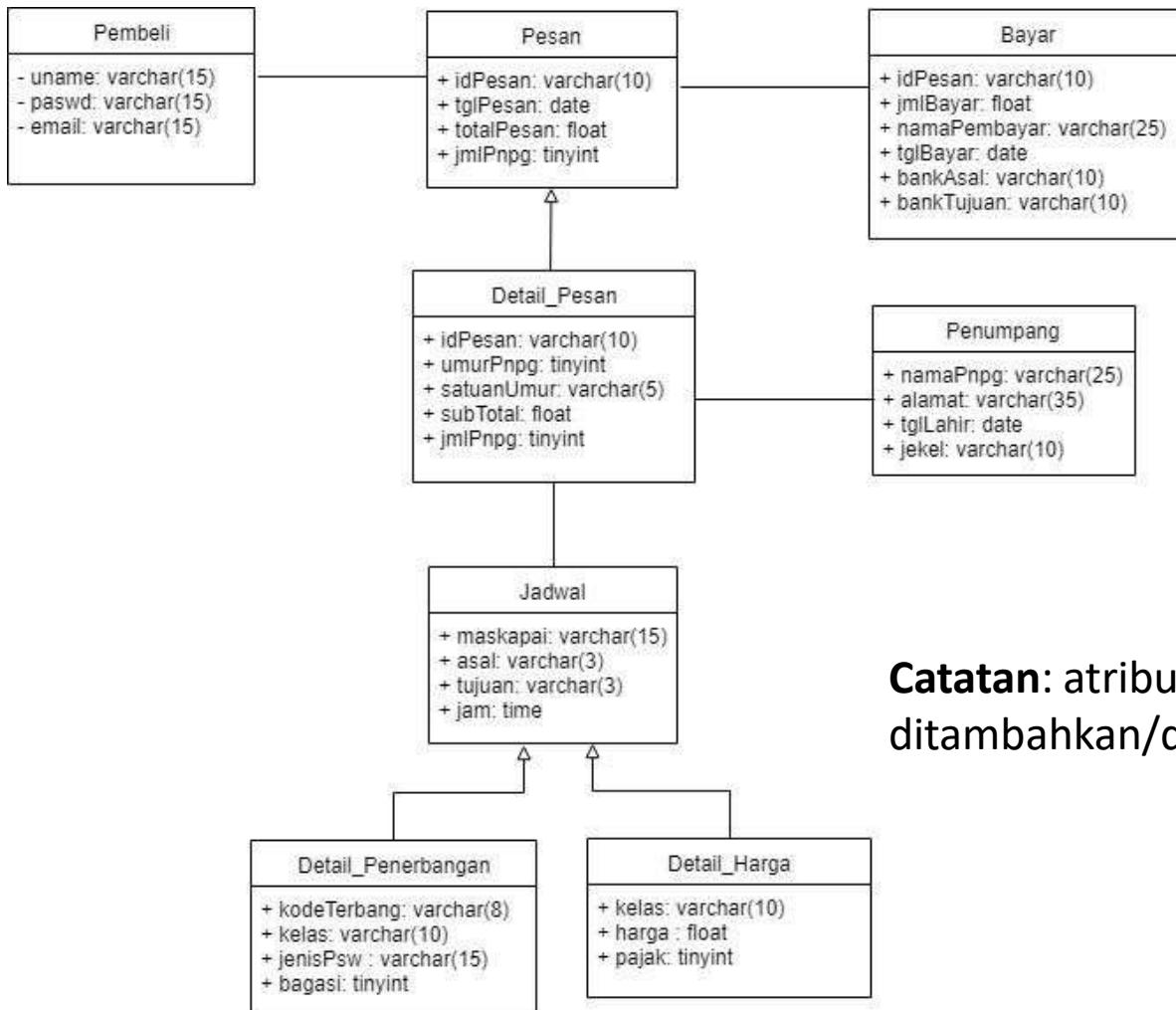
E. Operasi

- Suatu kegiatan yang akan dilakukan oleh sebuah class

Relasi pada Class Diagram

No	Simbol	Nama	Penjelasan
1	—	Asosiasi	Relasi antar class dengan makna umum, biasanya disertai dengan multiplicity
2	→	Asosiasi berarah	Relasi antar class dengan makna class yang satu digunakan oleh class yang lain, disertai dengan multiplicity
3	→	Generalisasi	Relasi antar class dengan makna generalisasi-spesialisasi
4	----->	<i>Dependency</i>	Relasi antar class dengan makna kebergantungan antar class
5	—◇	Agregasi	Relasi antar class dengan makna semua-bagian (<i>whole-part</i>)

Contoh Class Diagram



Catatan: atribut dapat ditambahkan/disesuaikan

6. COMPONENT DIAGRAM

- *Component diagram* merepresentasikan dunia riil item yaitu ***component software***.
- Diakses melalui ***interfacenya***.
- Relasi antara *component* dan *interface* disebut *realization*.
- Mewakili potongan-potongan yang independen yang bisa dipesan dan diperbaharui sewaktu-waktu.
- Digunakan saat memecah sistem menjadi komponen-komponen dan menampilkan hubungan dengan antarmuka (memecah sistem menjadi lebih rendah).
- Pembagian sistem ke dalam *component* lebih didorong untuk kepentingan **marketing** daripada teknis.

Component Diagram (Lanjutan)

- Tujuan dari *component diagram* adalah untuk menunjukkan hubungan antara berbagai komponen dalam suatu sistem.
- Komponen dalam UML dapat mewakili
 - ✓ Komponen logis (misal: komponen bisnis, komponen proses)
 - ✓ Komponen fisik (misal: komponen CORBA, Java, .NET, komponen WSDL),
- Diagram komponen umumnya berisi:
 - ✓ Komponen
 - ✓ Antarmuka
 - ✓ Ketergantungan, generalisasi, asosiasi, dan realisasi

Komponen (*Component*)

- Komponen adalah kelas yang mewakili bagian modular dari suatu sistem dengan konten yang di-enkapsulasi.
- Sebuah komponen memiliki perilaku yang didefinisikan dalam hal antarmuka.
- Bagian komponen yang lebih besar dari fungsi sistem dapat digunakan kembali, dan menghubungkan antarmuka.
- Sebuah komponen dimodelkan sepanjang siklus hidup pengembangan dan disempurnakan secara berturut-turut ke dalam *deployment* dan *run-time*.
- Komponen dapat dimanifestasikan oleh satu atau lebih artefak.

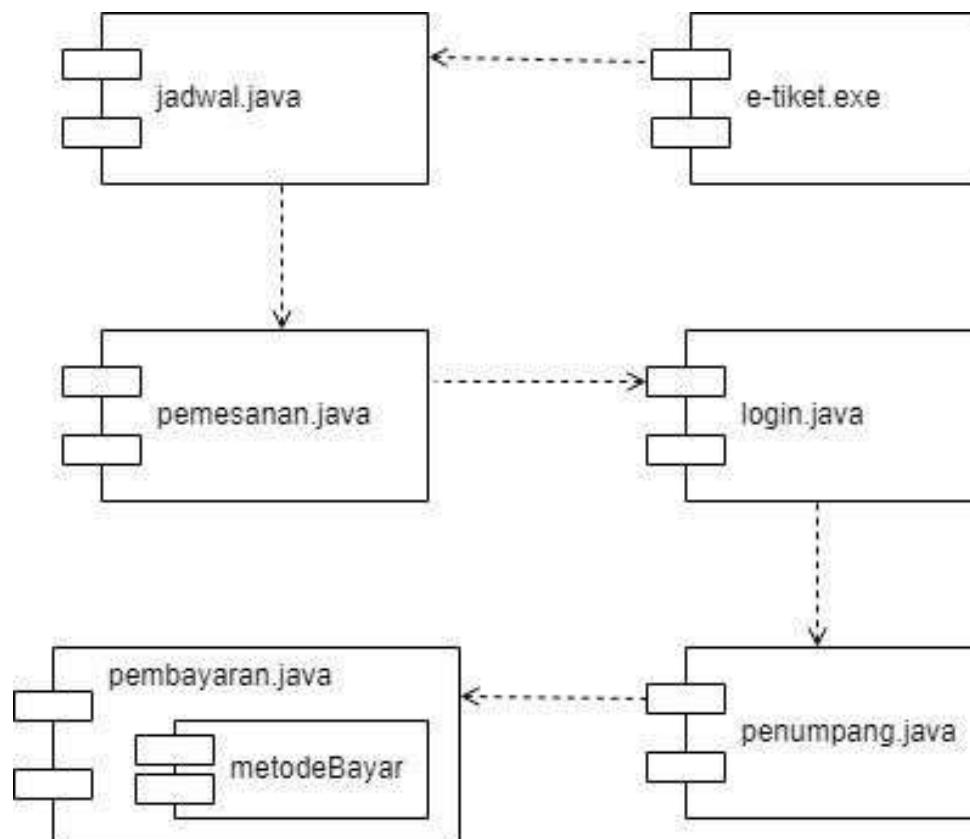
Diagram Komponen

Diagram Komponen dapat digambarkan dengan beberapa cara:

1. Memodelkan kode sumber (*source code*)
2. Memodelkan file eksekusi yang dirilis ke pengguna
3. Memodelkan basis data secara fisik
4. Memodelkan sistem yang dapat disesuaikan (adaptasi)

Contoh Component Diagram

Dengan model eksekusi



7. DEPLOYMENT DIAGRAM

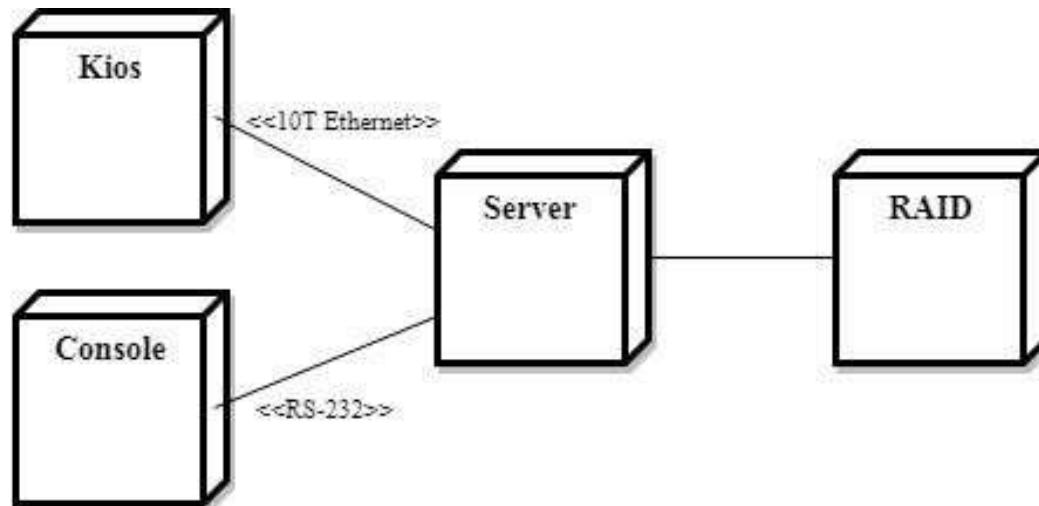
- *Deployment diagram* adalah diagram yang menunjukkan konfigurasi dari *node-node* pemrosesan waktu proses (*run time*) dan komponen-komponen yang ada di dalamnya.
- *Deployment diagram* menunjukkan tata letak sebuah sistem secara fisik, menampakkan bagian-bagian *software* yang berjalan pada bagian-bagian *hardware*.
- *Deployment diagram* umumnya mengandung:
 - ✓ *Node*: digambarkan dengan balok yang mewakili PL dasar atau elemen *hardware*, atau *node* dalam sistem.
 - ✓ Ketergantungan dan hubungan asosiasi yang digambarkan dengan garis solid dari *node* ke *node*.

Beberapa cara penamaan pada *node*:



Asosiasi

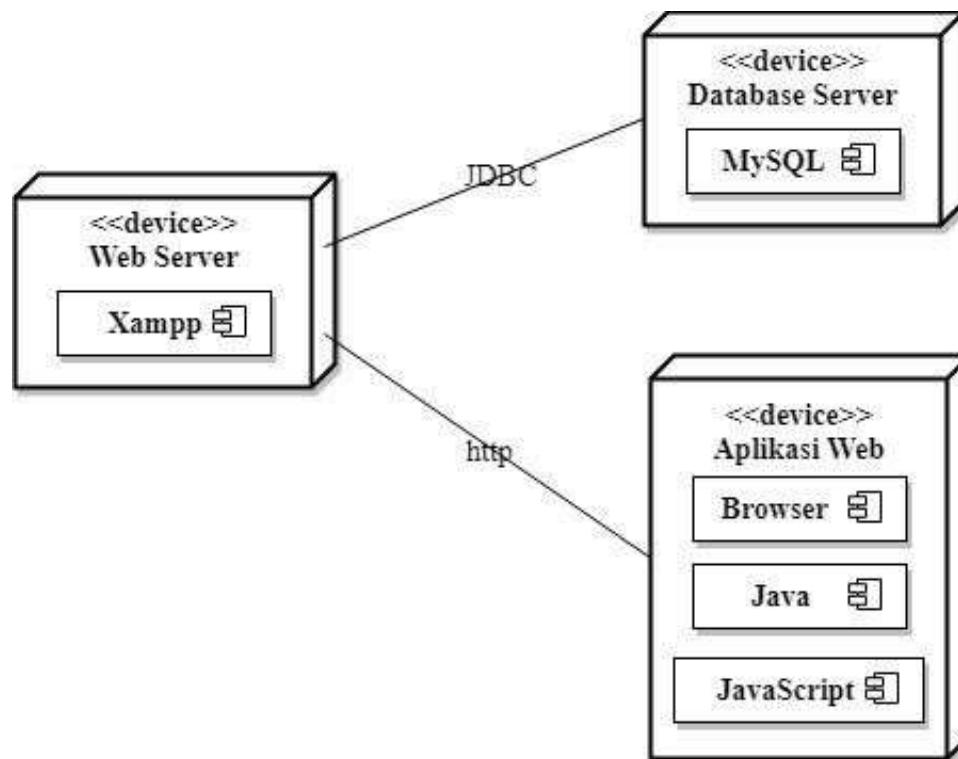
- Jenis hubungan yang paling umum adalah asosiasi yang merepresentasikan koneksi fisik antar *node* seperti koneksi *Ethernet*, saluran serial, atau *bus*.



Deployment Diagram (Lanjutan)

- *Deployment diagram* memiliki beberapa aplikasi yang dapat digunakan untuk:
 - ✓ Menampilkan elemen PL yang digunakan oleh elemen perangkat keras.
 - ✓ Mengilustrasikan pemrosesan *run time* untuk perangkat keras.
 - ✓ Memberikan pandangan tentang topologi sistem perangkat keras.
- *Deployment diagram* tidak hanya penting untuk memvisualisasikan, menentukan, dan mendokumentasikan *embedded*, *client/server*, dan sistem terdistribusi, tetapi juga untuk mengelola sistem yang dapat dieksekusi melalui teknik maju dan rekayasa ulang.

Contoh Deployment Diagram



Pertemuan 9

**PERANCANGAN
BERORIENTASI OBJEK**

1. PENDAHULUAN

- Sistem berorientasi objek terdiri dari **objek** yang berinteraksi yang mempertahankan keadaan (*state*) lokal dan menyediakan operasi pada *state* tersebut.
- Representasi *state* bersifat pribadi dan tidak dapat diakses langsung dari luar objek.
- Proses desain berorientasi objek melibatkan perancangan kelas objek dan hubungan antara kelas-kelas tersebut.

Pendahuluan (Lanjutan)

- Mengubah implementasi suatu objek atau menambahkan metode atau operasi tidak mempengaruhi objek lain dalam sistem.
- Hal-hal yang diperhatikan dalam desain berorientasi objek:
 1. Memahami dan mendefinisikan **konteks** dan **interaksi** eksternal dengan sistem.
 2. Desain arsitektur sistem.
 3. Identifikasi **objek utama** dalam sistem.
 4. Kembangkan model desain.
 5. Tentukan antarmuka.

Pendahuluan (Lanjutan)

- Desain berorientasi objek biasanya diimplementasikan dengan bahasa pemrograman berorientasi objek.
- Keuntungan utama dari desain OO adalah:
 - a. Sistem Analis dapat **menghemat waktu** dan menghindari kesalahan dengan menggunakan objek secara modular
 - b. Programer dapat menerjemahkan desain ke dalam kode. Objek yang baru dapat dibuat tanpa mengubah kode yang sudah ada
 - c. Bekerja dengan modul program yang dapat digunakan kembali (*reuse*) yang telah diuji dan diverifikasi.

A. Identifikasi Kelas Objek

Tujuan mengidentifikasi kelas objek dalam sistem berorientasi objek:

1. Gunakan analisis gramatikal dari deskripsi bahasa alami. Objek dan atribut adalah kata benda; operasi atau layanan adalah kata kerja.
2. Gunakan entitas nyata (benda) dalam domain aplikasi seperti mobil, peran seperti manajer atau dokter, acara seperti permintaan, interaksi seperti pertemuan, lokasi seperti kantor, unit organisasi seperti perusahaan.
3. Gunakan analisis berbasis skenario dimana berbagai skenario penggunaan sistem diidentifikasi dan dianalisis secara bergantian.

B. Istilah dalam Objek Oriented

1. OBJEK (*Object*)

- Objek adalah konsep atau abstraksi tentang sesuatu yang memiliki arti untuk aplikasi yang akan dikembangkan
- Objek diwakili dengan kata benda
- Objek dapat berupa:
 - Objek orang/manusia: Karyawan, Mahasiswa
 - Objek tempat: Kantor, Gedung, Toko
 - Objek abstrak: Transaksi, Jadwal, Peminjaman
 - Objek organisasi: Divisi-IT, HRD
 - Objek peralatan/benda: Mobil, Buku, Baju

2. ATRIBUT (*Attribute*)

- Suatu objek memiliki atribut tertentu yang merupakan **karakteristik** yang menggambarkan objek.
- Suatu atribut dapat mengambil sebuah nilai yang ditentukan berdasarkan *domain* yang dihitung.
- *Domain* merupakan satu himpunan nilai-nilai spesifik.
- Contoh: kelas MOBIL memiliki sebuah atribut WARNA. Domain nilai untuk warna adalah {putih, hitam, perak, abu-abu, biru, merah, kuning, hijau}.
- Objek dapat memiliki atribut khusus yang disebut *state*. Keadaan suatu objek adalah kata sifat yang menggambarkan status objek saat ini.
- Misalnya rekening bank dapat aktif, tidak aktif, tertutup, atau dibekukan

3. METODE (*Method*)

- Suatu metode mendefinisikan **tugas-tugas** spesifik yang dapat dilakukan oleh suatu objek
- Metode dituliskan dengan kata kerja yang menggambarkan apa dan bagaimana suatu objek melakukan sesuatu.
- Misalnya: objek PELANGGAN dapat melakukan tugas-tugas tertentu seperti melakukan pemesanan, membayar tagihan, dan mengubah alamatnya.

4. PESAN (*Message*)

- Pesan (*Message*) adalah perintah yang memberi tahu suatu objek untuk melakukan metode tertentu.
- Misalnya: pesan TAMBAHKAN SISWA mengarahkan kelas SISWA untuk menambahkan nomor siswa, nama, dan data lain tentang siswa itu. Demikian pula, pesan bernama HAPUS SISWA memberi tahu kelas SISWA untuk menghapus instance Siswa.
- Pesan yang sama untuk dua objek berbeda dapat menghasilkan hasil yang berbeda.

Contoh pesan



Objek INSTRUCTOR mengirim pesan ENTER GRADE ke instance kelas STUDENT RECORD.

Objek INSTRUCTOR dan kelas STUDENT RECORD dapat digunakan kembali dengan sedikit modifikasi, di sistem informasi sekolah lain dimana banyak atribut dan metode akan serupa

Contoh Pesan:

STUDENT
Attributes
Student Number
Name
Address
Telephone
Date of birth
Fitness record
Status
Methods
Add Student
Delete Student
Add fitness-class
Change address
Change telephone
Change status
Update fitness record

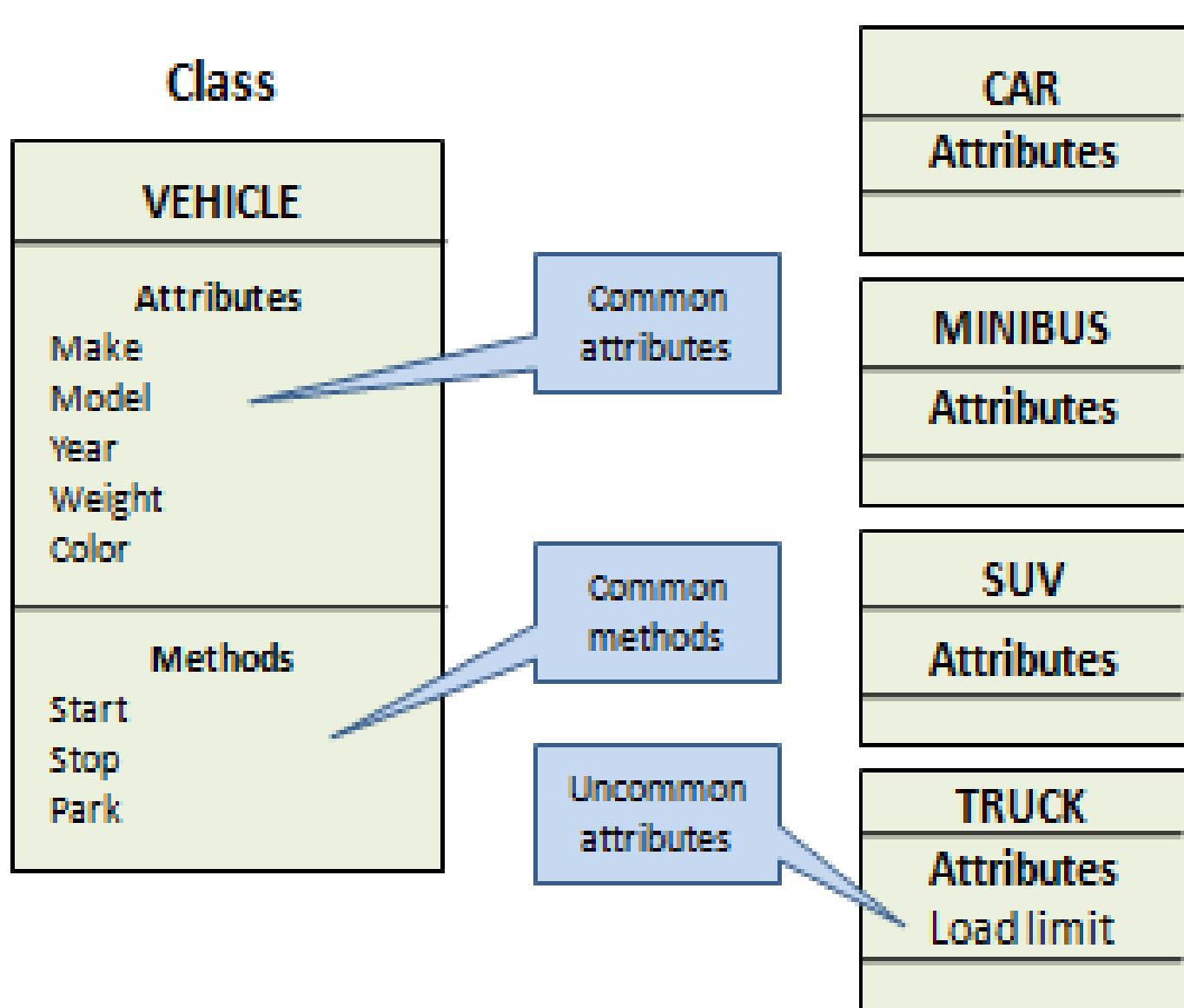
Message : ADD STUDENT
Tells the STUDENT class to perform all the steps needed to add a STUDENT instance

Message : DELETE STUDENT
Tells the STUDENT class to perform all the steps needed to delete a STUDENT instance.

5. KELAS (*Class*)

- Kelas adalah deskripsi umum yang menggambarkan sebuah kumpulan berisi objek-objek yang sama.
- Semua objek dalam kelas berbagi atribut dan metode yang sama, sehingga kelas seperti *blue print*, atau *template* untuk semua objek di dalam kelas.
- *Superclass* adalah generalisasi dari satu himpunan kelas-kelas yang berhubungan.
- *Subclass* adalah spesialisasi dari *superclass*.
- Contoh: superclass kendaraanBermotor adalah generalisasi dari kelas Truk, SUV, Minibus dan Car. Subclass Minibus mewarisi semua atribut kendaraanBermotor, tetapi juga menggabungkan atribut tambahan yang spesifik hanya untuk Minibus.

Contoh kelas dan subkelas



C. Hubungan Antara Objek dan Kelas

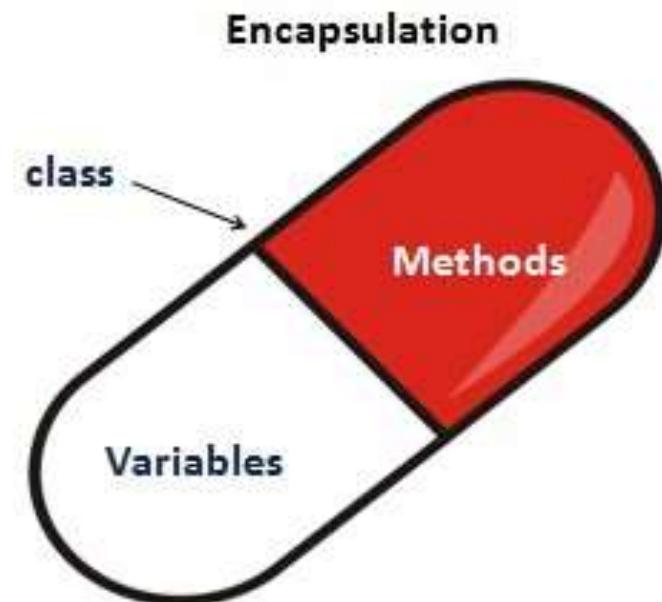
- Hubungan (*relationship*) memungkinkan objek untuk berkomunikasi dan berinteraksi ketika mereka melakukan fungsi bisnis dan transaksi yang diperlukan oleh sistem.
- Hubungan menggambarkan apa yang perlu diketahui objek satu sama lain, bagaimana objek merespon perubahan pada objek lain, dan efek keanggotaan dalam kelas, superclass, dan subclass.
- Beberapa hubungan lebih kuat daripada yang lain (seperti hubungan antara anggota keluarga lebih kuat dari satu hubungan antara kenalan biasa). Hubungan terkuat disebut **warisan**.

2. KARAKTERISTIK OBJEK

A. Enkapsulasi (*Encapsulation*)

- Data dan prosedur/fungsi dikemas bersama-sama dalam suatu objek, sehingga prosedur/fungsi lain dari luar tidak dapat mengaksesnya.
- Data terlindung dari prosedur atau objek lain kecuali prosedur yang berada dalam objek tersebut.
- Merupakan pembatasan ruang lingkup program terhadap data.
- Enkapsulasi memungkinkan objek untuk digunakan sebagai komponen modular di mana saja dalam sistem, karena objek mengirim dan menerima pesan tetapi tidak mengubah metode internal objek lain.

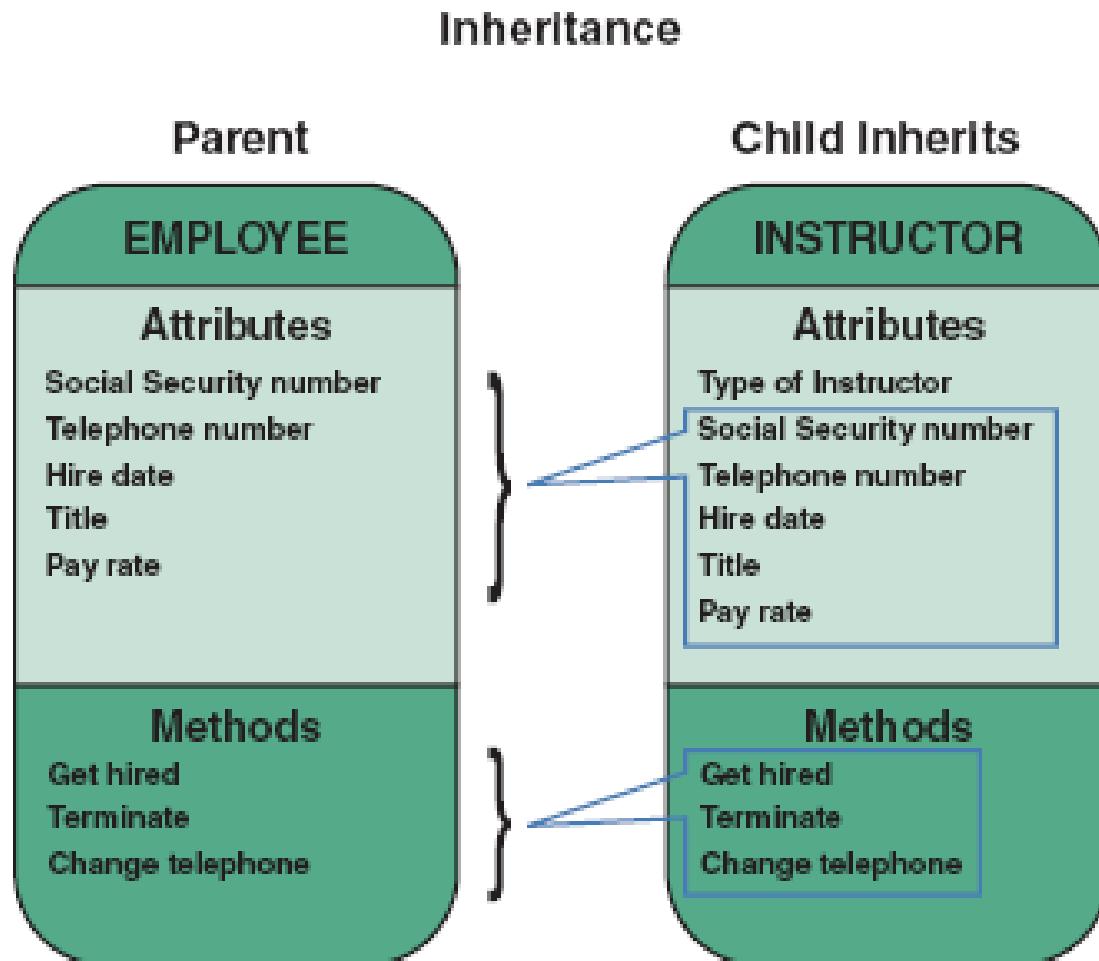
Contoh Enkapsulasi



B. Pewarisan (*Inheritance*)

- Pewarisan adalah salah satu pembeda utama antara sistem konvensional dan sistem berbasis objek.
- Subkelas Y mewarisi semua atribut dan operasi-operasi yang terkait dengan superkelas X. Ini berarti semua struktur dan algoritma data yang secara orisinal dirancang dan diimplementasikan untuk X segera tersedia untuk Y
- Perubahan apa pun pada atribut-atribut atau operasi-operasi yang dimuat ke dalam sebuah superkelas, akan diwarisi oleh semua subkelas.

Contoh pewarisan:

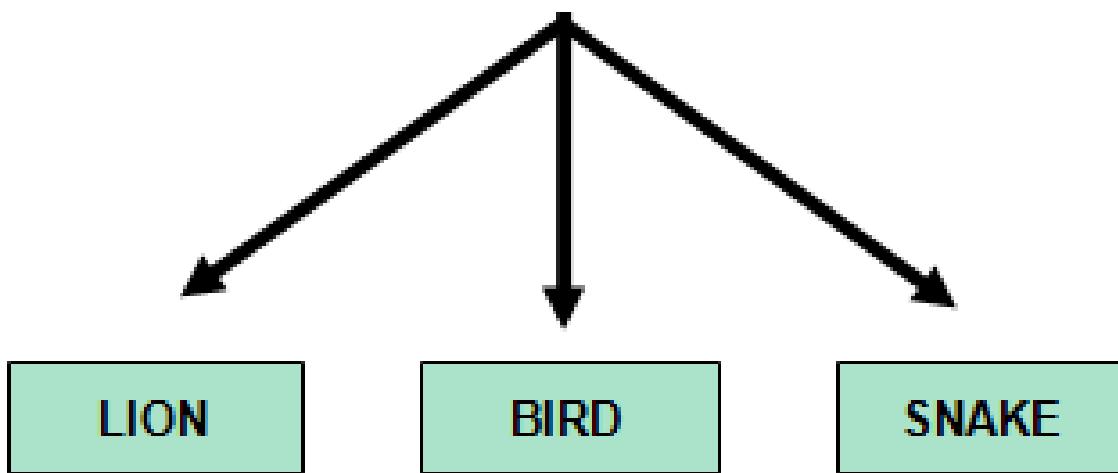


C. Polimorfis (*Polymorphism*)

- Merupakan konsep yang menyatakan bahwa sesuatu yang sama dapat mempunyai bentuk dan perilaku berbeda.
- Polimorfis juga mempunyai arti bahwa operasi yang sama mungkin mempunyai perbedaan dalam kelas yang berbeda.
- Polimorfis sangat mengurangi usaha yang diperlukan untuk memperluas perancangan sistem berorientasi objek

Contoh Polimorfis

Message: MOVE



3. KELAS-KELAS PERANCANGAN

- Model kebutuhan menentukan serangkaian kelas-kelas analisis yang masing-masing kelas menggambarkan beberapa elemen masalah yang fokus pada masalah yang dilihat oleh pengguna.
- Himpunan kelas-kelas perancangan adalah
 1. memperhalus kelas-kelas analisis dengan menyediakan detail perancangan yang memungkinkan kelas-kelas bisa diimplementasikan
 2. menciptakan suatu himpunan kelas-kelas perancangan yang baru, yang mengimplementasikan suatu infrastruktur PL yang mendukung solusi bisnis.

Kelas perancangan yang merepresentasikan lapisan berbeda dari perancangan arsitektur:

- a. **Kelas-kelas antarmuka.** Pengguna menentukan semua abstraksi yang diperlukan untuk interaksi manusia dengan komputer
- b. **Kelas-kelas bisnis.** Kelas-kelas mengidentifikasi atribut dan operasi/metode yang dibutuhkan untuk mengimplementasikan beberapa elemen ranah bisnis.
- c. **Kelas-kelas proses.** Mengimplementasikan abstraksi bisnis yang levelnya lebih rendah untuk sepenuhnya mengelola kelas-kelas ranah bisnis.
- d. **Kelas-kelas persisten.** Merepresentasikan *data store* yang akan terus ada setelah eksekusi PL.
- e. **Kelas-kelas sistem.** Mengimplementasikan manajemen PL dan mengendalikan fungsi-fungsi agar mampu mengoperasikan sistem dan berkomunikasi dengan dunia luar

A. Karakteristik Kelas Perancangan

a. Lengkap dan cukup

- Suatu kelas perancangan seharusnya menjadi enkapsulasi lengkap dari semua atribut dan metode yang dapat layak diharapkan.
- Cukup berarti memastikan bahwa kelas perancangan berisi hanya metode-metode yang cukup untuk mencapai tujuan kelas.
- Contoh: kelas Scene adalah lengkap hanya jika kelas ini berisi semua atribut dan metode yang dapat layak diasosiasikan dengan pembuatan suatu *scene video*.

Karakteristik Kelas Perancangan (Lanjutan)

b. Sederhana

- Metode-metode yang dihubungkan dengan sebuah kelas perancangan harus fokus ke pencapaian satu fungsi spesifik pada kelas.
- Contoh: kelas `VideoClip` memiliki atribut `StartPoint` dan `EndPoint` untuk mengindikasikan titik awal dan titik akhir.

Karakteristik Kelas Perancangan (Lanjutan)

c. Kohesi tinggi

- Kelas perancangan kohesif adalah *single minded*. Artinya kelas ini memiliki satu kumpulan kecil tanggung jawab yang fokus dan menerapkan atribut dan metode untuk menjalankan tanggung jawab tersebut.
- Contoh: kelas VideoClip dapat berisi satu kumpulan metode-metode untuk mengedit klip video. Kohesi dijaga asalkan setiap metode fokus semata-mata pada atribut-atribut yang diasosiasikan dengan klip video.

Karakteristik Kelas Perancangan (Lanjutan)

d. Keterhubungan rendah

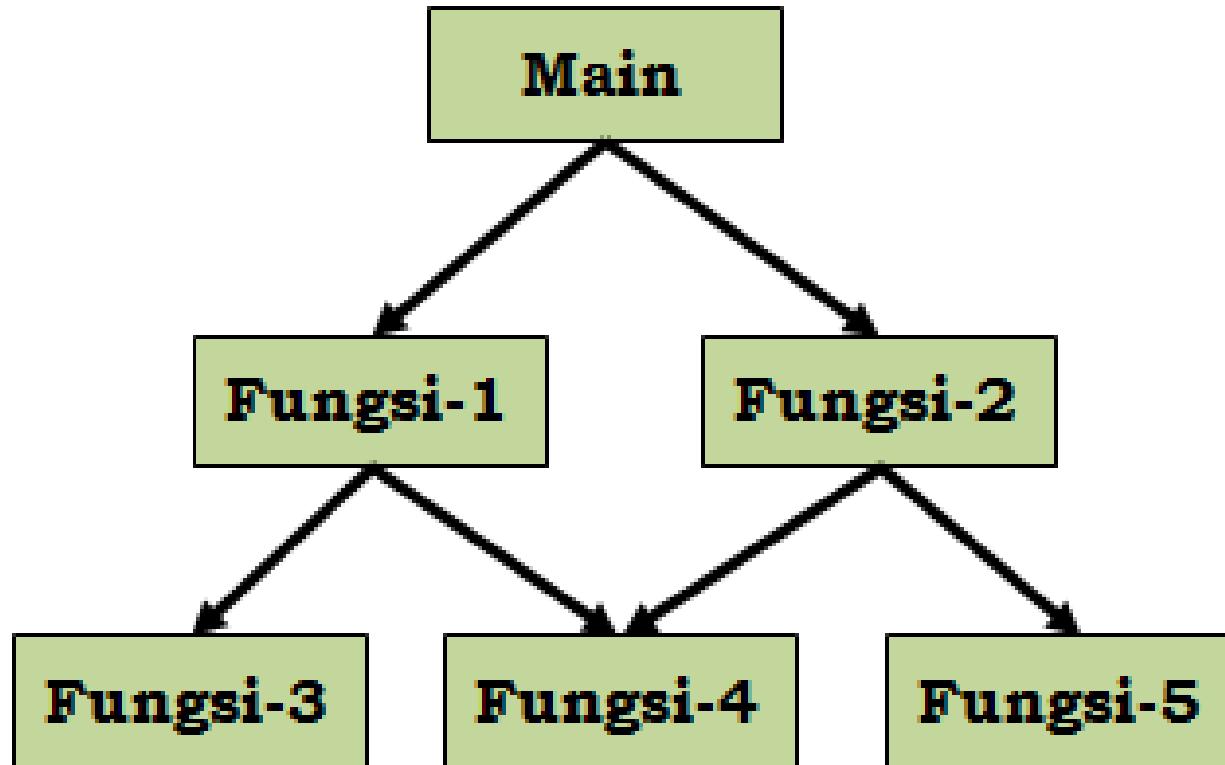
- Jika sebuah model perancangan memiliki keterhubungan tinggi (semua kelas perancangan berkolaborasi dengan semua kelas perancangan lainnya), sistem menjadi sulit diimplementasikan, diuji, dan dipelihara.
- Kelas perancangan pada subsistem memiliki hanya pengetahuan terbatas tentang kelas-kelas lain.
- Pembatasan ini dinamakan *Law of Demeter* yang menyatakan bahwa suatu metode seharusnya hanya mengirim pesan ke metode-metode pada kelas-kelas yang berdekatan.

4. PENDEKATAN PEMROGRAMAN TERSTRUKTUR

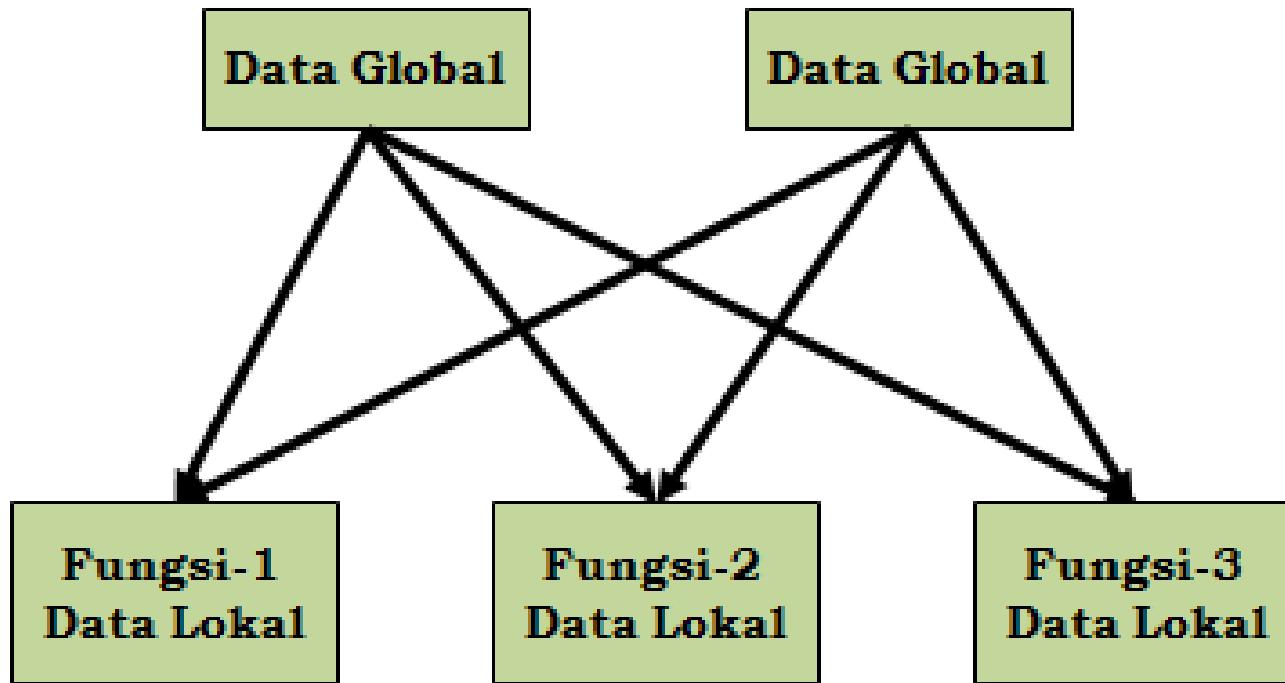
**Karakteristik Pendekatan Berorientasi
Prosedur/Fungsi:**

- a. Penekanan pada sesuatu yang harus dikerjakan (algoritma pemecahan masalah)
- b. Program berukuran besar dipecah menjadi program-program yang lebih kecil
- c. Kebanyakan fungsi/prosedur berbagi data global
- d. Data bergerak secara bebas dalam sistem dari satu fungsi ke fungsi yang lain yang terkait
- e. Fungsi-fungsi mentransformasi data dari satu bentuk ke bentuk yang lain
- f. Menggunakan pendekatan *top-down*

Struktur Umum Pemrograman Terstruktur



Hubungan Data dan Fungsi pada Pemrograman Terstruktur

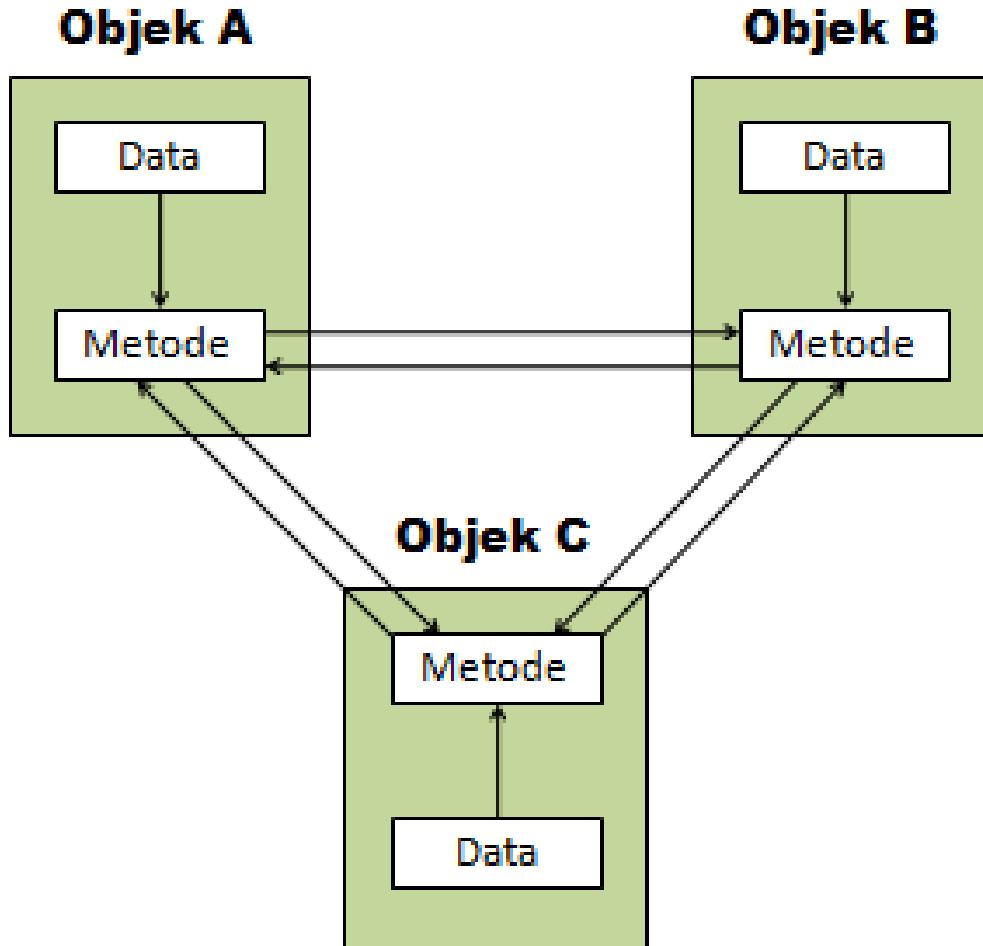


5. PENDEKATAN BERORIENTASI OBJEK

Karakteristik Pada Pendekatan Berorientasi Objek:

- Pendekatan lebih kepada data (bukan fungsi/prosedur)
- Program besar dibagi menjadi beberapa objek
- Struktur data dirancang dan menjadi karakteristik dari objek-objek
- Fungsi-fungsi yang mengoperasikan data tergabung dalam suatu objek yang sama
- Data tersebunyi dan terlindung dari fungs/prosedur yang ada di luar
- Objek-objek dapat saling berkomunikasi dengan saling mengirim *message* satu sama lain
- Menggunakan pendekatan *bottom-up*

Pengorganisasian Data dan Metode (Fungsi) pada Pendekatan Berorientasi Objek



6. Perbedaan Pemrograman Terstruktur dengan Pendekatan Berorientasi Objek

A. PEMROGRAMAN TERSTRUKTUR

- Permasalahan dilihat sebagai urutan sesuatu yang harus dikerjakan, seperti input – proses – output.
- Fokus utamanya pada fungsi atau prosedur
- Data global pada program yang sangat besar, sangat sulit untuk dilacak. Jika merevisi data global maka merevisi setiap fungsi yang menggunakan data global.
- Tidak menggambarkan kasus nyata dengan baik, karena fungsi-fungsi berorientasi pada aksi dan tidak berhubungan langsung dengan permasalahan.
- Kurang sempurna dalam menangkap kebutuhan *reusable components*, karena tidak ada standarisasi modul

B. PENDEKATAN BERORIENTASI OBJEK

- Penekanan pada “apa” yang dapat dilakukan oleh objek
- Pendekatan lebih kepada data
- Perubahan pada struktur data internal tidak mempengaruhi struktur data objek yang lain
- Penggunaan bersama untuk beberapa tingkat yang berbeda, seperti penggunaan bersama untuk disain dan kode
- Menggunakan pendekatan *bottom-up*
- Fungsi/prosedur berbagi data global

Pertemuan 10

**DESAIN
ARSITEKTUR**

1. PENDAHULUAN

- Perancangan arsitektur merupakan tahap pertama dalam proses perancangan PL, yang dimulai dengan perancangan data kemudian berlanjut pada penurunan satu atau lebih struktur arsitektural sistem.
- Arsitektur sistem/PL adalah struktur sistem/PL yang menggabungkan komponen PL, menggabungkan properti yang tampak dari komponen tersebut, dan mendeskripsikan hubungan antar komponen.
- *Output* dari perancangan arsitektur berupa model arsitektur yang menggambarkan bagaimana sistem diatur sebagai satu set komponen yang saling berkomunikasi.

2. ARSITEKTUR PL

Arsitektur mencakup:

- Komponen bangunan yang berbeda dapat diintegrasikan menjadi suatu bentuk keseluruhan yang bersifat kohesif
- Bangunan yang dibuat sesuai dengan lingkungannya
- Bangunan yang dibangun sesuai dengan kegunaannya
- Tekstur, warna dan material pembentuknya dikombinasikan untuk membuat tampilan yang bagus
- Perancangan pencahayaan, template, dan garis batas
- Merupakan suatu bentuk seni

Arsitektur PL merupakan representasi yang memungkinkan untuk:

1. Melakukan analisis terhadap efektivitas perancangan dan disesuaikan dengan kebutuhan yang dinyatakan sebelumnya
2. Melakukan pertimbangan alternatif arsitektural pada tahap dimana perubahan rancangan dapat dilakukan dengan cara yang relatif mudah
3. Mengurangi risiko yang berhubungan dengan konstruksi PL

Alasan arsitektur PL:

- Representasi arsitektur PL adalah sesuatu yang memungkinkan terjadinya komunikasi di antara semua pihak yang tertarik pada pengembangan sistem berbasis komputer
- Arsitektur yang dibuat di awal perancangan akan memiliki efek yang menentukan pada semua pekerjaan RPL selanjutnya
- Arsitektur menggambarkan model yang relatif kecil dan mudah dipahami, dan menggambarkan bagaimana sistem distrukturkan dan bagaimana komponen di dalamnya saling bekerja sama.

A. Deskripsi Arsitektural

Sasaran dari deskripsi arsitektural:

- Untuk menetapkan kerangka kerja konseptual dan kosa kata yang digunakan selama perancangan arsitektur PL
- Untuk menyediakan panduan yang rinci pada waktu merepresentasikan deskripsi arsitektural
- Untuk memandu praktik perancangan yang baik

B. Keputusan Arsitektural

Pola Deskripsi Keputusan Arsitektur

a. Permasalahan Perancangan

Deskripsikan permasalahan perancangan arsitektural yang akan diselesaikan.

b. Penyelesaian

Menentukan pendekatan yang dipilih untuk menyelesaikan permasalahan yang berkaitan dengan perancangan

c. Kategori

Spesifikasi kategori perancangan yang akan diselesaikan permasalahannya, seperti perancangan data, struktur isi dan komponen, integrasi, presentasi

Pola Deskripsi Keputusan Arsitektur (Lanjutan)

d. Asumsi-asumsi

Indikasikan asumsi saat menentukan keputusan. Misalnya standar teknologi, pola yang tersedia, permasalahan yang berkaitan dengan sistem/PL

e. Alternatif-alternatif

Secara singkat deskripsikan alternatif yang akan dipertimbangkan dan mengapa ditolak

f. Argumen

Jelaskan mengapa memilih penyelesaian di atas dan alternatif-alternatif lainnya

Pola Deskripsi Keputusan Arsitektur (Lanjutan)

g. Keputusan yang berhubungan

Keputusan terdokumentasi yang berhubungan dengan keputusan yang diambil

h. Implikasi

Indikasikan konsekuensi perancangan akibat penentuan keputusan. Apakah penyelesaian akan berakibat pada perancangan lainnya?

i. Perhatian yang berhubungan

Adakah kebutuhan lain yang berhubungan dengan keputusan yang diambil?

Pola Deskripsi Keputusan Arsitektur (Lanjutan)

j. Produk kerja

Indikasikan dimana keputusan yang diambil akan tercermin dalam deskripsi arsitektur

k. Catatan

Rujukan catatan tim lainnya yang sebelumnya telah digunakan untuk membuat keputusan

Beberapa pertimbangan dalam keputusan Arsitektur:

1. Adakah arsitektur aplikasi generik yang dapat bertindak sebagai *template* untuk sistem yang sedang dirancang?
2. Bagaimana sistem akan didistribusikan ke sejumlah perangkat keras?
3. Pola atau gaya arsitektur apa yang digunakan?
4. Pendekatan fundamental apa yang digunakan untuk menyusun sistem?
5. Bagaimana komponen struktural dalam sistem akan terdekomposisi menjadi sub-komponen?

Beberapa pertimbangan dalam keputusan Arsitektur:

6. Strategi yang akan digunakan untuk mengontrol pengoperasian komponen dalam sistem
7. Organisasi arsitektur apa yang terbaik untuk memberikan persyaratan sistem non-fungsional?
8. Bagaimana desain arsitektur akan dievaluasi?
9. Bagaimana arsitektur sistem didokumentasikan?

3. TAMPILAN ARSITEKTURAL

1. Tampilan Logis

Abstraksi dalam sistem sebagai objek atau kelas objek.

2. Tampilan Proses

Menunjukkan bagaimana (pada saat *run-time*) sistem terdiri dari proses yang saling berinteraksi.

3. Tampilan Pengembangan

PL diuraikan untuk pengembangan, yaitu menunjukkan *detail* dalam komponen yang akan diimplementasikan oleh pengembang tunggal atau tim pengembang.

4. Tampilan Fisik

Menunjukkan perangkat keras sistem dan bagaimana komponen PL didistribusikan di seluruh sistem.

4. GAYA ARSITEKTUR

Gaya arsitektur mendeskripsikan kategori sistem yang mencakup:

- **Kumpulan komponen**, seperti sistem basis data dan modul-modul yang melaksanakan fungsi tertentu yang diperlukan oleh sistem
- **Penghubung (konektor)** yang memungkinkan komunikasi, koordinasi, dan kerja antar komponen
- **Batasan** yang mendefinisikan bagaimana komponen dapat diintegrasikan untuk membentuk suatu sistem/PL
- **Model semantik** yang memungkinkan perancang sistem memahami properti keseluruhan sistem

Gaya dan Struktur Arsitektur (Persyaratan Non-Fungsional)

1. Kinerja (*Performance*)

Arsitektur harus dirancang agar semua komponen dapat digunakan pada berbagai komputer/prosesor, dan mendistribusikan di seluruh jaringan.

2. Keamanan (*Security*)

Menggunakan struktur berlapis untuk melindungi aset yang paling penting di lapisan terdalam, dengan tingkat validasi keamanan yang tinggi.

3. Keamanan (*Safety*)

Operasi yang terkait dengan keselamatan terletak di salah satu komponen tunggal atau komponen kecil.

Gaya dan struktur arsitektur (persyaratan non-fungsional)

4. Ketersediaan (*Availability*)

Arsitektur harus dirancang untuk menyertakan komponen redundan sehingga dimungkinkan saat mengganti dan memperbarui komponen tanpa menghentikan sistem.

5. Pemeliharaan (*Maintainability*)

Arsitektur sistem harus dirancang menggunakan komponen mandiri yang dapat diubah dengan mudah. Struktur data bersama harus dihindari.

Struktur Dasar Arsitektur

- Arsitektur PL merepresentasikan suatu struktur dimana beberapa kumpulan entitas (komponen) dihubungkan dengan sejumlah relasi (konektor).
- Komponen dan konektor dihubungkan dengan properti yang dapat membedakan jenis komponen dan konektor yang digunakan.

a. Struktur Fungsional

- Komponen merepresentasikan fungsi atau entitas.
- Konektor merepresentasikan antarmuka untuk melewatkkan data ke suatu komponen.
- Properti mendefinisikan sifat dari komponen dan mengorganisasikan antarmuka.

Struktur Dasar Arsitektur (Lanjutan)

b. Struktur Implementasi

- Komponen berbentuk paket, kelas, objek, prosedur, fungsi, metode, dll, yang merupakan sarana untuk mengemas fungsionalitas komponen pada berbagai peringkat abstraksi.
- Konektor meliputi kemampuan untuk melewatkkan data dan kendali, berbagi data, menggunakan, dan menginstansiasi.
- Properti pada komponen fokus pada karakteristik kualitas, seperti kemampuan untuk *maintenance* dan *reuse* yang dihasilkan saat struktur diimplementasikan.

Struktur Dasar Arsitektur (Lanjutan)

c. Struktur Konkurensi

- Komponen merepresentasikan unit-unit konkurensi yang terorganisasi sebagai pekerjaan paralel (*thread*).
- Konektor mencakup sinkronisasi, prioritas, mengirim data, dan menjalankan proses/fungsi.
- Properti mencakup prioritas, kemampuan untuk meramalkan, dan waktu eksekusi.

d. Struktur Fisik

- Komponen merupakan perangkat keras fisik.
- Konektor merupakan antarmuka antar komponen perangkat keras.
- Properti berkaitan dengan kapasitas, *bandwidth*, kinerja, dan atribut lainnya.

Struktur Dasar Arsitektur (Lanjutan)

e. Struktur Pengembangan

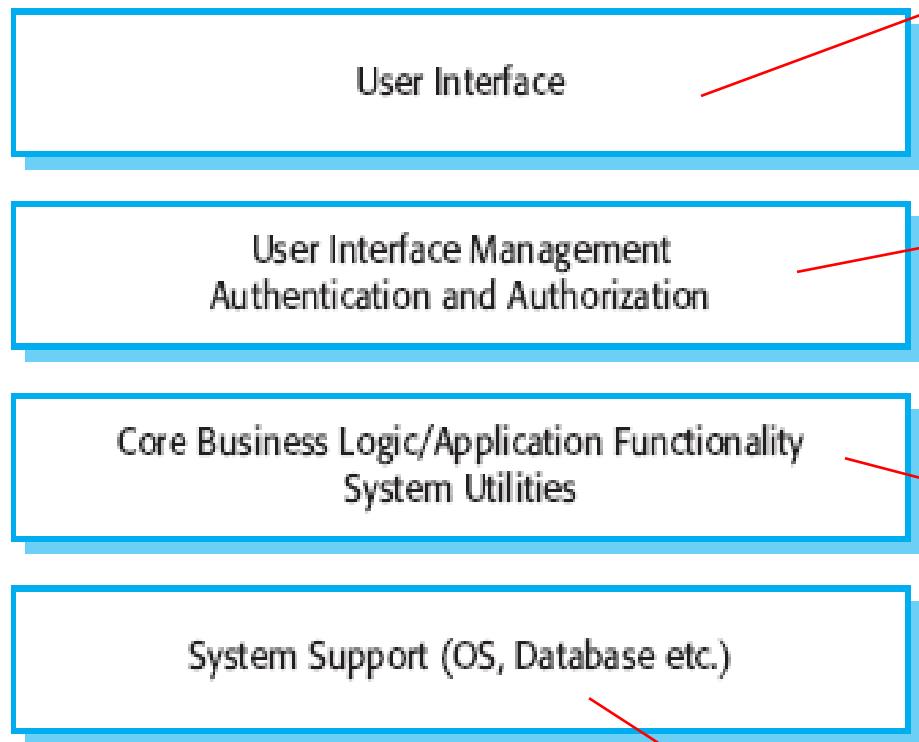
- Mendefinisikan komponen, produk kerja, dan sumber informasi lainnya.
- Konektor merepresentasikan relasi antar produk kerja.
- Properti mengidentifikasi karakteristik tiap-tiap *item*.

5. POLA ARSITEKTUR (*Architectural Patterns*)

A. Lapisan Arsitektur (*Layered Architecture*)

- Pemahaman tentang pemisahan dan independensi sangat penting untuk desain arsitektur karena memungkinkan perubahan secara lokal.
- Menambahkan tampilan baru atau mengubah tampilan yang ada dapat dilakukan tanpa perubahan apa pun pada data dalam model.

Gambar Generik Arsitektur Lapisan



Lapisan atas menyediakan fasilitas antarmuka pengguna

Lapisan aplikasi: komponen fungsionalitas aplikasi dan komponen utilitas

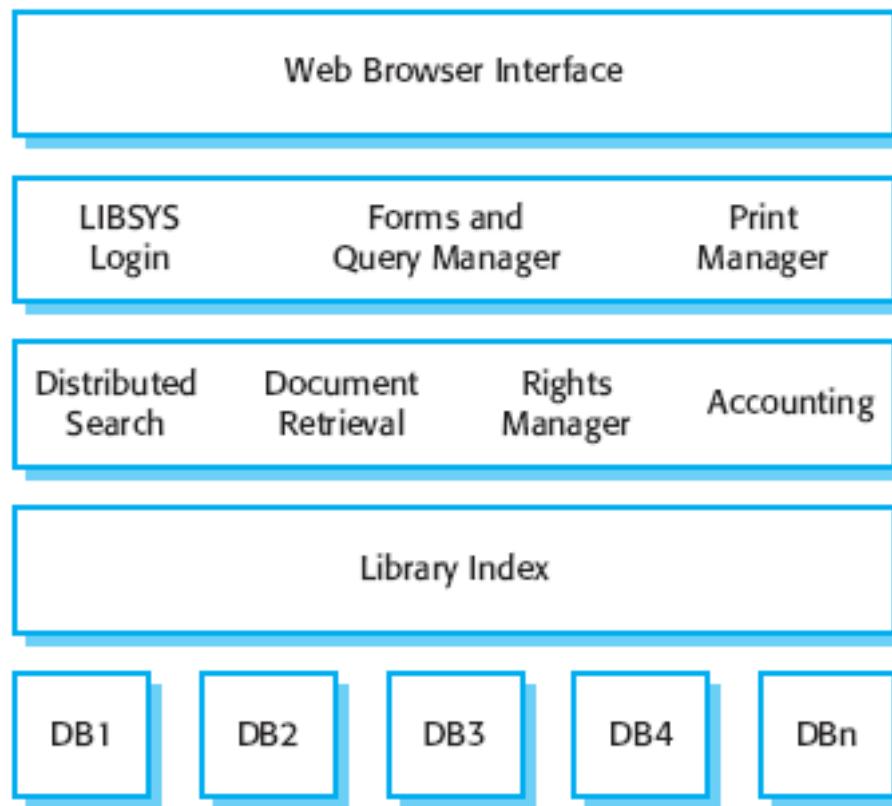
Lapisan ketiga: manajemen antarmuka pengguna dan menyediakan otentifikasi dan otorisasi pengguna

Lapisan terendah: PL pendukung sistem (basis data dan OS)

Pola Arsitektur Berlapis

Name	Layered Architecture
Deskripsi	Mengatur sistem ke dalam lapisan dengan fungsi terkait. Lapisan menyediakan layanan ke lapisan di atasnya sehingga lapisan tingkat terendah mewakili layanan inti yang kemungkinan akan digunakan di seluruh sistem.
Contoh	Sebuah model berlapis dari suatu sistem untuk berbagi dokumen hak cipta yang disimpan di media penyimpanan.
Saat digunakan	<ul style="list-style-type: none">• saat membangun fasilitas baru di atas sistem yang ada• ketika pengembangan tersebar di beberapa tim dengan tanggung jawab masing-masing tim• ketika ada persyaratan untuk keamanan multi-level
Keuntungan	Memungkinkan penggantian seluruh lapisan selama antarmuka dipertahankan. Fasilitas redundan (misal otentikasi) dapat disediakan di setiap lapisan untuk meningkatkan keandalan sistem.
Kerugian	Lapisan tingkat tinggi mungkin harus berinteraksi langsung dengan lapisan tingkat yang lebih rendah daripada melalui lapisan tepat di bawahnya. Kinerja dapat menjadi masalah karena beberapa tingkat interpretasi permintaan layanan diproses pada setiap lapisan.

Contoh arsitektur lapisan, dengan lapisan bawah menjadi basis data individual di setiap pustaka pada Sistem Perpustakaan



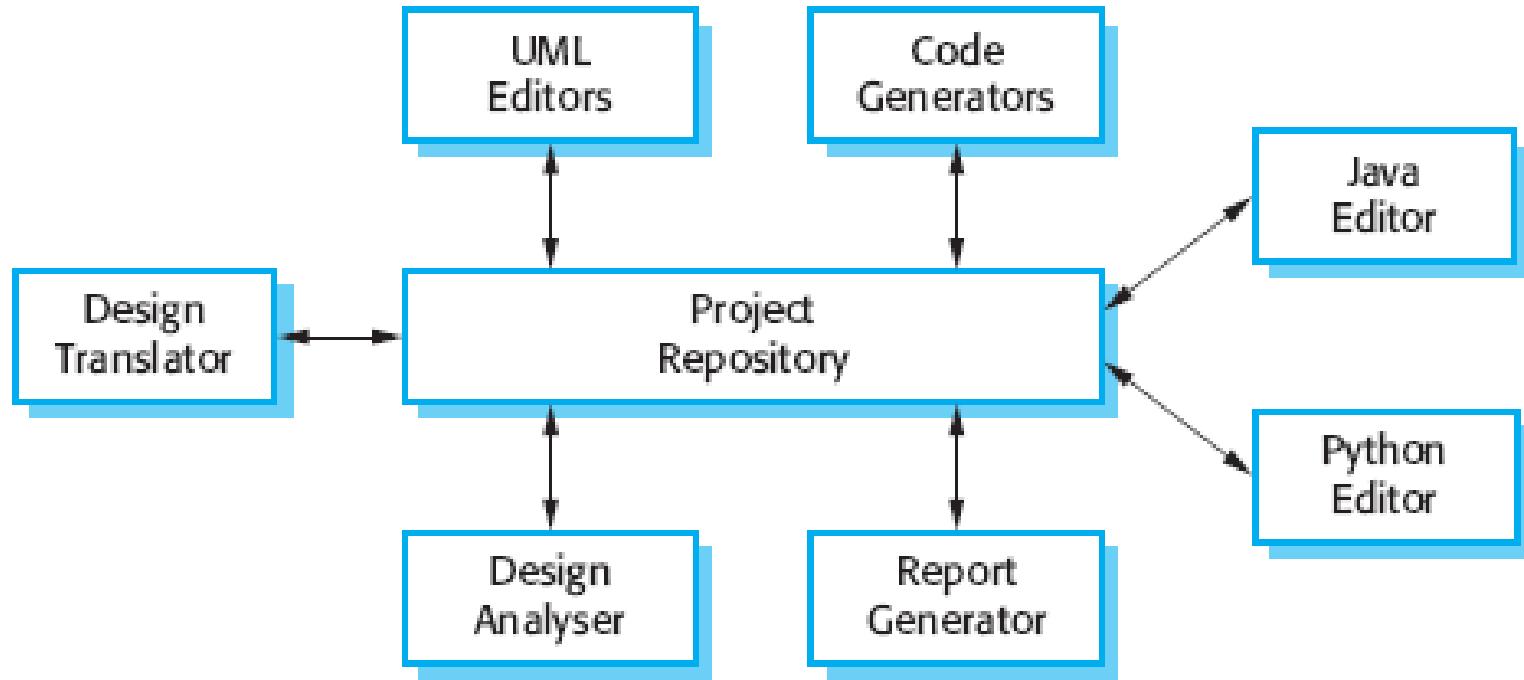
B. Arsitektur Repozitori (*Repository Architecture*)

- Bagaimana satu set komponen yang saling berinteraksi dapat berbagi data.
- Model ini cocok untuk aplikasi di mana data dihasilkan oleh satu komponen dan digunakan oleh yang lain
- Tidak perlu mentransmisikan data secara eksplisit dari satu komponen ke komponen lainnya. Tetapi komponen harus beroperasi di sekitar model data repositori yang disepakati.
- Pola repositori berkaitan dengan struktur statis dari suatu sistem dan tidak menunjukkan organisasi *run-time*.

Repositori Arsitektur

Nama	Repository
Deskripsi	Semua data dalam sistem dikelola di repositori pusat yang dapat diakses oleh semua komponen sistem
Contoh	Contoh dari IDE dimana komponen menggunakan repositori, dan setiap PL menghasilkan informasi yang kemudian tersedia untuk digunakan oleh alat lain.
Saat digunakan	Ketika sistem dengan sejumlah besar informasi yang dihasilkan disimpan untuk waktu yang lama.
Keuntungan	<ul style="list-style-type: none">• Komponen dapat mandiri, karena tidak perlu mengetahui keberadaan komponen lain.• Perubahan yang dilakukan oleh satu komponen dapat disebarluaskan ke semua komponen.• Semua data dapat dikelola secara konsisten karena semuanya ada di satu tempat.
Kerugian	Masalah dalam repositori mempengaruhi seluruh sistem.

Contoh repositori arsitektur untuk sebuah IDE



Menunjukkan IDE yang mencakup alat yang berbeda untuk mendukung pengembangan berbasis model. Repositori dalam kasus ini adalah lingkungan yang dikendalikan oleh versi yang melacak perubahan pada PL dan memungkinkan *rollback* ke versi sebelumnya.

C. Client–Server Architecture

- Sebuah sistem yang mengikuti pola *client-server* diatur sebagai satu set layanan *server*, dan *client* yang mengakses dan menggunakan layanan.
- Komponen utama dari model ini adalah:
 1. **Server** memberikan layanan ke komponen lain.
Contoh: *server* menawarkan layanan pencetakan, *server file* yang menawarkan layanan manajemen file, dan *server* kompilasi yang menawarkan layanan kompilasi bahasa pemrograman.
 2. *Client* yang menggunakan layanan yang ditawarkan oleh *server*.
 3. Jaringan yang memungkinkan *client* untuk mengakses layanan.

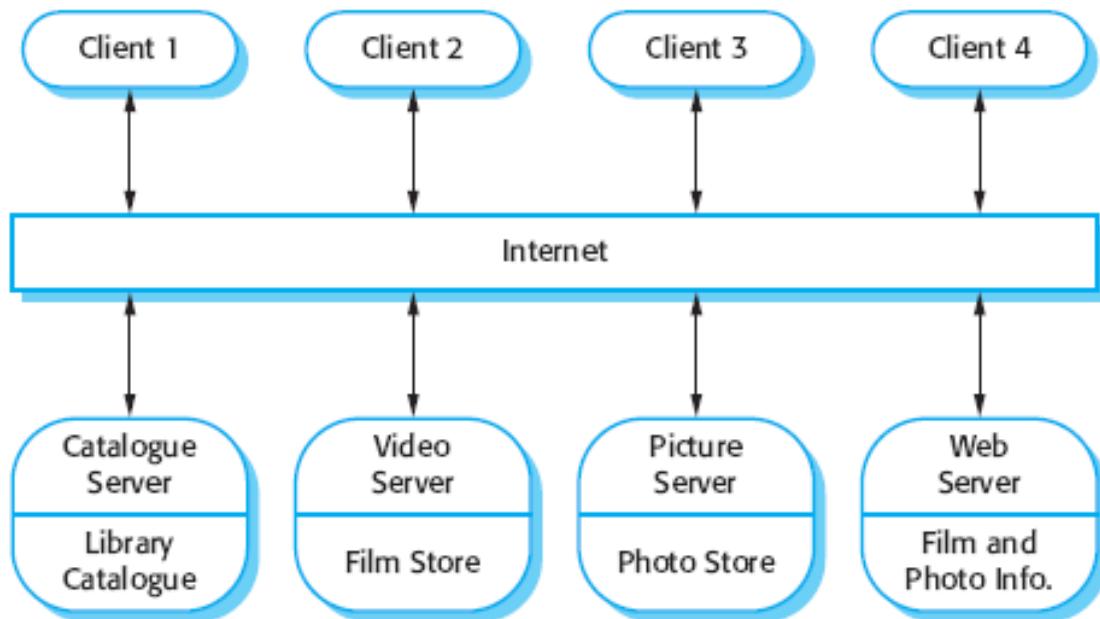
Client–Server Architecture (Lanjutan)

- Arsitektur *client-server* dianggap sebagai arsitektur sistem terdistribusi, tetapi model logis dari layanan independen yang berjalan pada *server* terpisah dapat diimplementasikan pada satu komputer
- Penggunaan yang efektif dapat dilakukan dari sistem jaringan dengan banyak prosesor terdistribusi.
- Sangat mudah untuk menambahkan server baru dan mengintegrasikannya dengan seluruh sistem atau meng-*upgrade server* secara transparan tanpa mempengaruhi bagian lain dari sistem.

Arsitektur ***Client-Server***

Name	Client-Server
Deskripsi	Fungsionalitas sistem diatur ke dalam layanan, dengan setiap layanan yang dikirim dari <i>server</i> terpisah
Contoh	Contoh dari perpustakaan film/video
Saat digunakan	Ketika data dalam database harus diakses dari berbagai lokasi.
Keuntungan	<ul style="list-style-type: none">• Server dapat didistribusikan melalui jaringan.• Fungsi umum dapat tersedia untuk semua <i>client</i> dan tidak perlu diterapkan di semua layanan.
Kerugian	<ul style="list-style-type: none">• Setiap layanan dapat terjadi kegagalan sehingga rentan terhadap penolakan layanan atau kegagalan <i>server</i>.• Kinerja tidak dapat diprediksi karena tergantung pada jaringan dan juga sistem.

Contoh sistem perpustakaan film/video



- Dalam sistem ini, beberapa server mengelola dan menampilkan berbagai jenis media.
- Server video dapat menangani kompresi dan dekompresi video dalam berbagai format.
- Katalog harus dapat menangani pertanyaan dan menyediakan tautan ke dalam sistem informasi web yang mencakup data tentang film dan klip video, dan e-commerce mendukung penjualan foto, film, klip video.

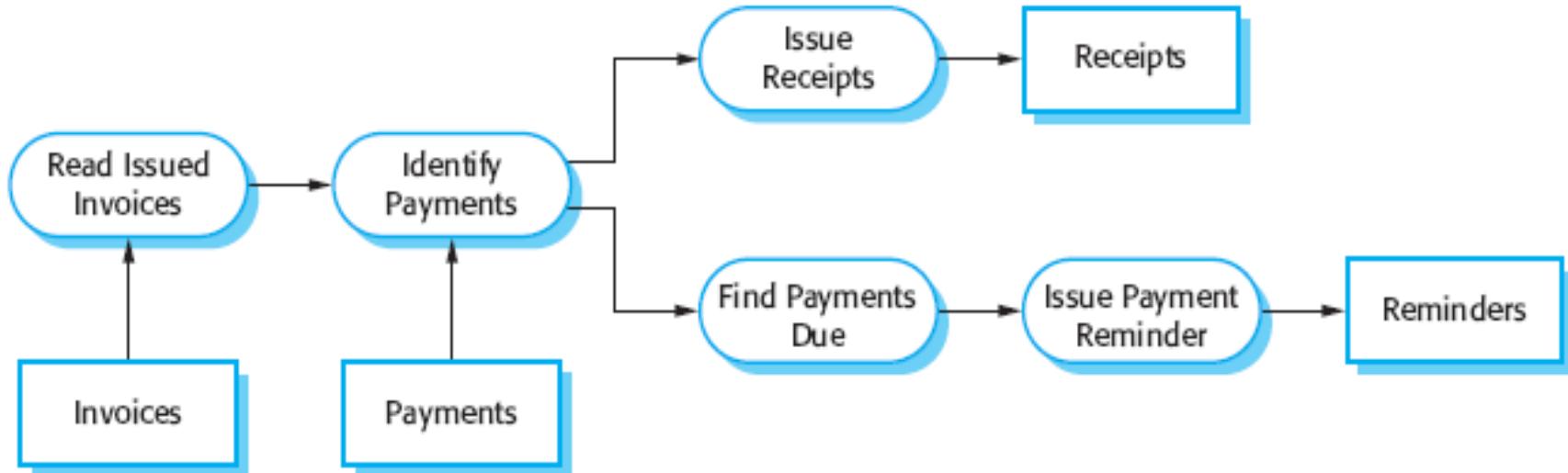
D. Pipe and Filter Architecture

- Model dari sistem *run-time* di mana transformasi secara fungsional memproses input dan menghasilkan output.
- Aliran data bergerak secara berurutan (seperti dalam pipa).
- Setiap langkah pemrosesan diimplementasikan sebagai transformasi.
- Transformasi dapat dilakukan secara berurutan/paralel.
- Data diproses oleh transformasi per-item-nya atau dalam satu *batch*.
- *Pipe* digunakan untuk melewati aliran teks dari satu proses ke proses lainnya.
- *Filter* digunakan pada transformasi untuk menyaring data.

Pipe and Filter Architecture

Name	Pipe and Filter
Deskripsi	Pengolahan data diatur dalam suatu sistem sehingga setiap komponen pemrosesan (<i>filter</i>) bersifat diskrit dan melakukan satu jenis transformasi data.
Contoh	Contoh pada sistem untuk memproses faktur.
Saat digunakan	Umumnya digunakan dalam aplikasi pemrosesan data (baik batch atau berbasis transaksi) di mana input diproses dalam tahap terpisah untuk menghasilkan output.
Keuntungan	<ul style="list-style-type: none">• Mudah dimengerti dan mendukung transformasi <i>reuse</i>.• Gaya alur kerja cocok dengan struktur proses bisnis.• Dapat diimplementasikan sebagai sistem sekuensial/konkuren.
Kerugian	<ul style="list-style-type: none">• Format transfer data harus disepakati di antara transformasi komunikasi.• Setiap transformasi harus memahami input dan tidak mempublikasikan outputnya ke bentuk yang tidak dipahami.• Meningkatkan <i>overhead</i> sistem, berarti bahwa tidak mungkin menggunakan kembali transformasi fungsional yang menggunakan struktur data yang tidak kompatibel.

Contoh pada sistem untuk memproses faktur



Suatu organisasi telah menerbitkan faktur kepada pelanggan. Seminggu sekali, pembayaran yang telah dilakukan direkonsiliasi dengan faktur.

Untuk faktur yang telah dibayarkan, diberikan tanda terima. Untuk faktur yang belum dibayar dalam waktu pembayaran yang ditentukan, diberikan pesan untuk mengingatkan

6. ARSITEKTUR APLIKASI

- Sistem aplikasi dimaksudkan untuk memenuhi kebutuhan bisnis yang memiliki banyak kesamaan dan menggunakan aplikasi tertentu.
- Arsitektur aplikasi dapat diimplementasikan kembali ketika mengembangkan sistem baru, tetapi untuk banyak sistem bisnis, penggunaan kembali aplikasi dimungkinkan tanpa implementasi ulang

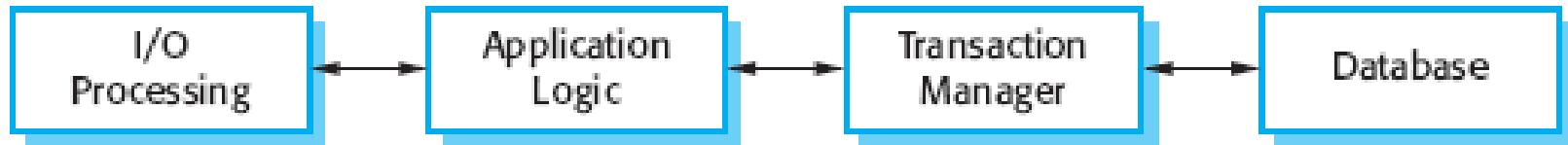
A. Sistem Pemrosesan Transaksi *(Transaction Processing Systems)*

- Aplikasi pemrosesan transaksi adalah aplikasi yang berpusat pada database yang memproses permintaan pengguna untuk informasi dan memperbarui informasi dalam basis data.
- Merupakan jenis sistem bisnis interaktif yang paling umum, di mana pengguna membuat permintaan *asynchronous* untuk layanan
- Transaksi basis data adalah urutan operasi yang diperlakukan sebagai unit tunggal, dan semua operasi dalam transaksi harus diselesaikan sebelum perubahan basis data dibuat permanen.

Aplikasi Pemrosesan Transaksi (Lanjutan)

- Dari perspektif pengguna, transaksi adalah setiap urutan operasi yang koheren yang memenuhi tujuan, seperti menemukan jadwal perkuliahan.
- Sistem pemrosesan transaksi dapat diatur sebagai arsitektur '*pipe and filter*' dengan komponen sistem sebagai input, pemrosesan, dan output.
- Misal: pelanggan menarik uang tunai dari ATM. Sistem ini terdiri dari dua komponen PL ATM dan PL pemrosesan akun di server basis data bank. Komponen I/O diimplementasikan sebagai PL di ATM dan komponen pemrosesan adalah bagian dari server database bank.

Contoh Aplikasi Pemrosesan Transaksi



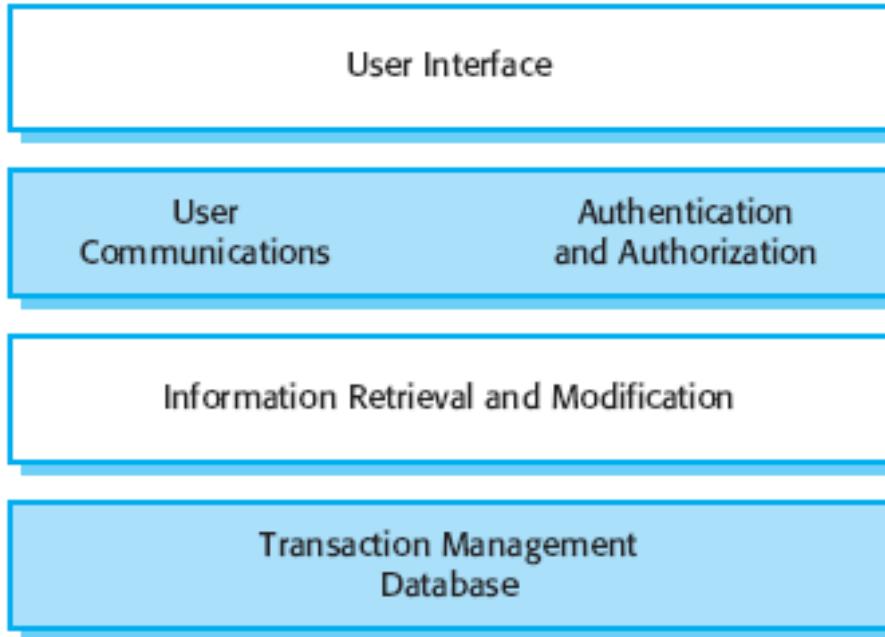
Penjelasan

- Pengguna membuat permintaan ke sistem melalui komponen pemrosesan I / O.
- Permintaan diproses oleh beberapa aplikasi logika.
- Transaksi dibuat dan diteruskan ke manajer transaksi, yang biasanya tertanam dalam sistem manajemen basis data.
- Setelah manajer transaksi memastikan bahwa transaksi sudah diselesaikan dengan benar, kemudian memberi sinyal ke aplikasi bahwa proses telah selesai

B. Sistem Informasi

- Semua sistem yang melibatkan interaksi dengan basis data dapat dianggap sebagai sistem informasi berbasis transaksi.
- Sistem informasi memungkinkan akses yang terkontrol ke basis informasi yang besar. Seperti katalog perpustakaan, jadwal penerbangan, atau catatan pasien di rumah sakit.
- Sebagai contoh dari instantiation model berlapis

Contoh Sistem Informasi

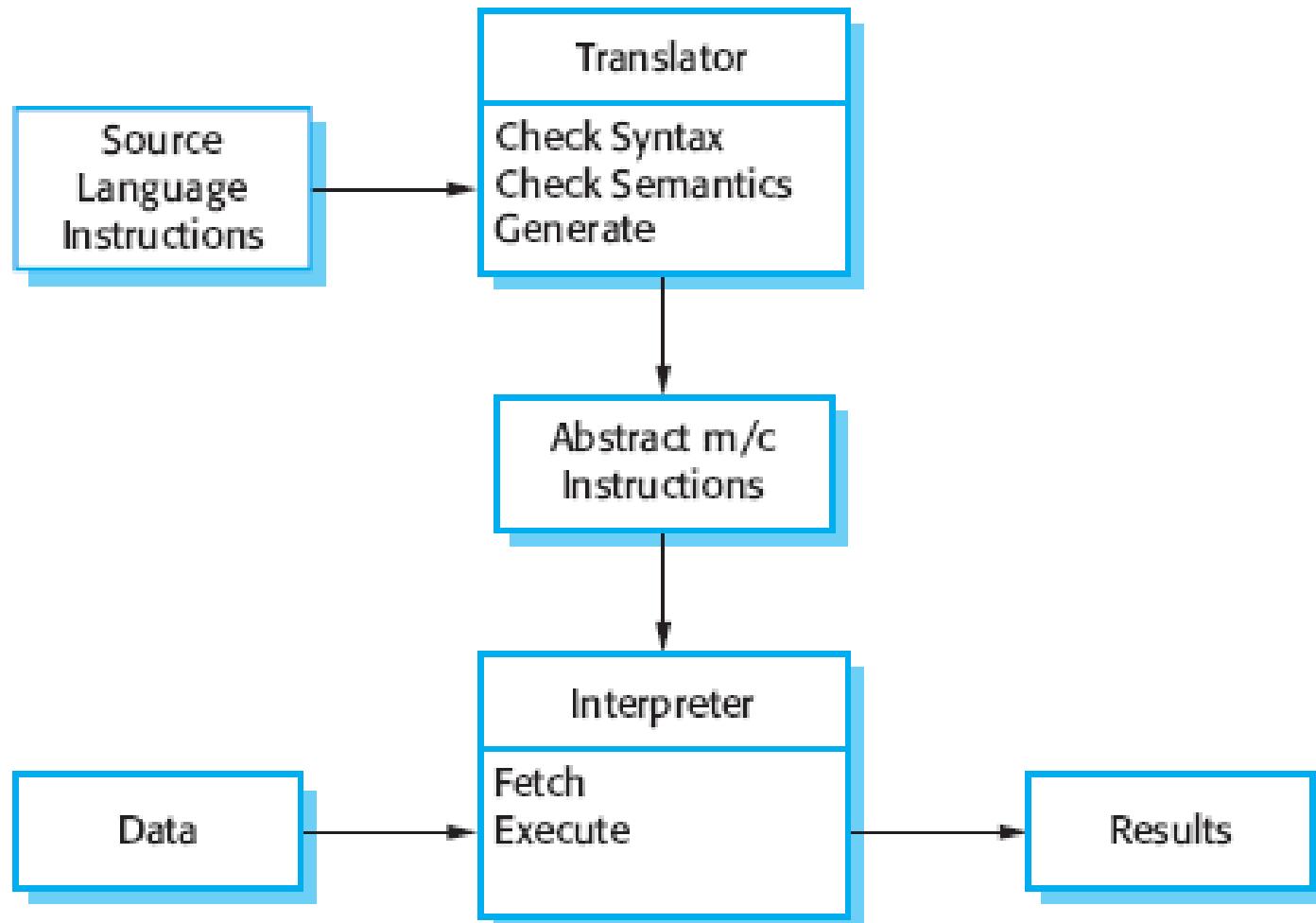


- Sistem dimodelkan menggunakan pendekatan berlapis di mana lapisan atas mendukung antarmuka pengguna dan lapisan bawah adalah database sistem.
- Lapisan komunikasi pengguna menangani semua I/O dari antarmuka pengguna, dan lapisan pencarian informasi untuk mengakses dan memperbarui database

C. Sistem Pemrosesan Bahasa (*Language Processing Systems*)

- Adalah sistem di mana maksud pengguna dinyatakan dalam bahasa formal (seperti Java).
- Memproses ke dalam bahasa formal, kemudian menafsirkan representasi secara internal.
- Sistem pemrosesan bahasa dengan *compiler*, yang menerjemahkan bahasa program tingkat tinggi ke dalam kode mesin.
- Sistem pemrosesan bahasa juga menerjemahkan bahasa alami atau buatan ke dalam representasi bahasa lain, dan bahasa pemrograman dapat mengeksekusi kode yang dihasilkan.

Contoh Sistem Pemrosesan Bahasa



Pertemuan 11

PERANCANGAN APLIKASI WEB

1. PENDAHULUAN

Perancangan aplikasi web memerlukan aktivitas teknis berupa:

1. Menetapkan tampilan pada web
2. Pembuatan rancangan estetika antarmuka pengguna
3. Pendefinisian struktur arsitektur aplikasi web secara keseluruhan
4. Pengembangan isi dan fungsional
5. Perencanaan navigasi

Pendahuluan (lanjutan)

Perancangan web sangat penting bagi *designer* karena:

1. Membuat model yang dapat dinilai kualitasnya dan dapat diperbaiki sebelum isi dan kode dibentuk
2. Membuat model sebelum pengujian dilakukan
3. Membuat model sebelum *end-user* yang berjumlah besar menggunakan aplikasi

Langkah-langkah *web design* dengan membuat:

1. Perancangan Isi

Dikembangkan selama tahapan analisis, dilakukan sebagai basis untuk penetapan objek-objek

2. Perancangan Estetika (Perancangan Grafis)

Membuat tampilan yang akan dilihat oleh *user*

3. Perancangan Arsitektural

Fokus pada struktur *hypermedia* untuk semua objek isi dan untuk semua fungsi pada aplikasi web

Langkah-langkah *web design* dengan membuat:

4. Perancangan Antarmuka

Menentukan tampilan dan mekanisme interaksi yang mendefinisikan *user interface*

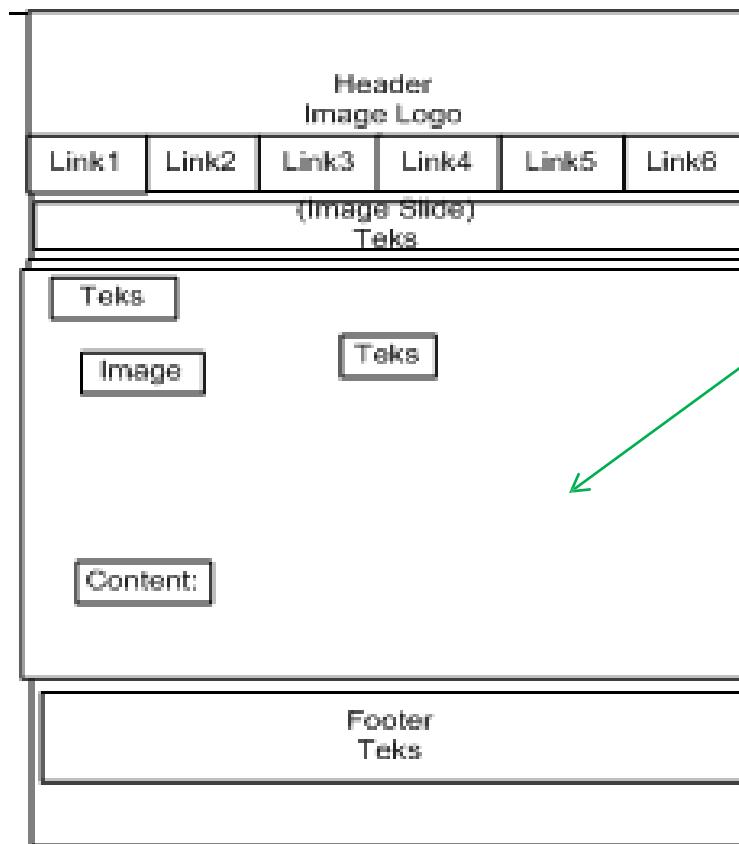
5. Perancangan Struktur Navigasi

Mendefinisikan bagaimana *end user* melakukan penelusuran untuk melintasi *hypermedia*

6. Perancangan Komponen

Merepresentasikan rincian struktur elemen-elemen fungsional aplikasi web

Contoh disain yang sulit dipahami, karena mengandung pernyataan yang umum berupa “Link”



Bagian ini tidak
menjelaskan
kebutuhan isi
tampilannya

2. SIFAT-SIFAT APLIKASI WEB

- a. Kepadatan jaringan
- b. Keserempakan
- c. Jumlah pengguna yang tidak dapat diprediksi
- d. Kinerja
- e. Ketersediaan
- f. Digerakkan oleh data
- g. Peka terhadap isi
- h. Evolusi yang berkesinambungan
- i. Kesegeraan
- j. Keamanan
- k. Estetika

3. KUALITAS PERANCANGAN APLIKASI WEB

Atribut Kualitas:

1. Keamanan

Kemampuan aplikasi web dan lingkungan server untuk mencegah akses yang tidak sah, dan mencegah serangan-serangan yang berasal dari luar.

2. Ketersediaan

Pengukuran atas persentase waktu yang tersedia bagi aplikasi web untuk dapat digunakan oleh *user*.

3. Skalabilitas

Aplikasi web mampu mengakomodasi kebutuhan terhadap jumlah *end user* yang semakin bertambah

4. Waktu untuk masuk ke pasar

Ditinjau dari sudut pandang bisnis

Contoh disain Login dari sudut pandang keamanan

Masukan Login Anda

The image shows a login form titled "Masukan Login Anda". It contains two input fields for time: "03:46:43 pm" and "17:12:16", which are highlighted with red arrows. Below these is a dropdown menu labeled "Pilih". To the left of the dropdown is a "User" label and below it is a "Password" label. At the bottom is a large "LogIn" button.

Penggunaan
2 waktu yang
berbeda jauh

Sebaiknya User tidak
memilih dari
beberapa data yang
disediakan

User tidak dapat
melakukan navigasi
lainnya kecuali
LOGIN

Kualitas Aplikasi Web

1. Kemudahan Penggunaan

- a. Kemudahan pemahaman situs global
- b. Umpulan balik dari *user* dan fitur-fitur bantuan
- c. *User interface* dan fitur-fitur estetika
- d. Fitur-fitur khusus

2. Fungsionalitas

- a. Kemampuan pencarian dan penerimaan
- b. Fitur-fitur navigasi dan perambahan (*browsing*)
- c. Fitur-fitur aplikasi yang berhubungan dengan lingkungan

3. Keandalan

- a. Pembetulan pemrosesan tautan (*link*)
- b. Pemulihan dari kesalahan
- c. Validasi dan pemulihan pada *user*

Sistem memberikan umpan balik kepada user jika melakukan kesalahan



Kualitas Aplikasi Web (Lanjutan)

4. Efisiensi

- a. Kinerja waktu tanggap aplikasi web
- b. Kecepatan pembentukan halaman-halaman
- c. Kecepatan penggambaran grafik-grafik

5. Kemudahan Pemeliharaan

- a. Kemudahan untuk dilakukan koreksi
- b. Keamanan aplikasi web untuk beradaptasi
- c. Kemudahan aplikasi web untuk dikembangkan

Sasaran perancangan web yang baik:

1. Kesederhanaan (*Simplicity*)

Fungsi-fungsi mudah digunakan dan mudah dipahami

2. Konsisten (*Consistency*)

Konstruksi perancangan isi dibuat secara konsisten.

Misalnya: jenis font yang sama pada semua dokumen teks, skema warna dan gaya yang konsisten

3. Identitas (*Identity*)

Estetika, *user interface*, dan perancangan navigasi harus konsisten dengan lingkungan aplikasi untuk apa aplikasi web itu dibuat atau dikembangkan.

4. Ketangguhan (*Robustness*)

User pada umumnya mengharapkan isi dan fungsi yang relevan terhadap kebutuhan *user*.

Sasaran perancangan web yang baik:

5. Kemudahan melakukan navigasi dalam aplikasi

Aplikasi web seharusnya dirancang sedemikian rupa sehingga tampilannya intuitif dan hasilnya dapat dengan mudah diramalkan.

6. Daya tarik visual (*Visual Appeal*)

Tampilan isi, rancangan *user interface*, pengaturan warna, keseimbangan yang harus terjadi di antara teks, grafik dan media lainnya, mekanisme navigasi sangat memiliki kontribusi pada daya tarik visual

7. Kompatibilitas (*Compatibility*)

Aplikasi web akan digunakan pada berbagai jenis lingkungan eksekusi aplikasi yang berbeda (*hardware*, OS, *browser*, dan koneksi internet)

Contoh desain web

HEADER																																																																																							
Home	Product	Sign In	Registration	Confirmation	How To Order	About Us																																																																																	
<table border="1"><tr><td colspan="8">[CONFIRMATION PAYMENT]</td></tr><tr><td>No. booking</td><td colspan="7"><input type="text"/></td></tr><tr><td>Sender's name</td><td colspan="7"><input type="text"/></td></tr><tr><td>Total transfer:</td><td colspan="7"><input type="text"/></td></tr><tr><td>Information</td><td colspan="7"><input type="text"/></td></tr><tr><td>Bank name</td><td colspan="7"><input type="text"/></td></tr><tr><td>Date of transfer</td><td colspan="7"><input type="text"/></td></tr><tr><td>Picture</td><td colspan="7"><input type="text"/></td></tr><tr><td colspan="8"><input type="button" value="Send"/></td></tr><tr><td colspan="8">Information Teks</td></tr></table>								[CONFIRMATION PAYMENT]								No. booking	<input type="text"/>							Sender's name	<input type="text"/>							Total transfer:	<input type="text"/>							Information	<input type="text"/>							Bank name	<input type="text"/>							Date of transfer	<input type="text"/>							Picture	<input type="text"/>							<input type="button" value="Send"/>								Information Teks							
[CONFIRMATION PAYMENT]																																																																																							
No. booking	<input type="text"/>																																																																																						
Sender's name	<input type="text"/>																																																																																						
Total transfer:	<input type="text"/>																																																																																						
Information	<input type="text"/>																																																																																						
Bank name	<input type="text"/>																																																																																						
Date of transfer	<input type="text"/>																																																																																						
Picture	<input type="text"/>																																																																																						
<input type="button" value="Send"/>																																																																																							
Information Teks																																																																																							
INFORMASI				About Us																																																																																			
				How To Order																																																																																			
				Contact us																																																																																			
FOOTER																																																																																							

4. PERANCANGAN ANTARMUKA

- Salah satu tantangan membuat *user interface* adalah bagaimana caranya *user* masuk ke aplikasi.
- Sasaran-sasaran *user interface* adalah untuk:
 1. Menetapkan suatu jendela yang konsisten untuk meletakkan isi-isi dan fungsionalitas yang disediakan oleh *user interface*
 2. Memandu *user* melalui serangkaian interaksi dengan aplikasi web yang dikembangkan
 3. Mengorganisasikan pilihan-pilihan navigasi dan isi-isi yang dapat dilihat *user* yang dapat berupa menu navigasi, *icon* grafis, dan gambar-gambar grafis

Contoh desain web berbasis android



5. PERANCANGAN ESTETIKA

Sering juga disebut Perancangan Grafis, yang merupakan tambahan artistik yang sering digunakan untuk melengkapi aspek-aspek teknis dari perancangan aplikasi web.

Tata letak yang baik pada perancangan *interface*:

1. Jangan mengisi bagian dari halaman web dengan informasi yang akhirnya sulit untuk mengidentifikasi informasi tersebut
2. Lakukan penekanan pada isi yang merupakan alasan utama bagi *user* untuk masuk ke aplikasi web
3. Lakukan pengelompokkan fitur navigasi, isi, dan fungsi
4. Jangan perluas bagian aplikasi dengan penggunaan *scrollbar*, sebaiknya kurangi isi yang jumlahnya banyak
5. Sesuaikan resolusi layar dan ukuran jendela *browser*

Contoh

Sebaiknya tidak menggabungkan semua kebutuhan penggunaan aplikasi, pisahkan antara pemesanan dengan pengembalian mobil, serta penggunaan warna teks yang seharusnya mudah dibaca

The screenshot shows a software interface for managing car rentals. At the top center is a digital clock displaying "01:39:02". The main title "PROSES PENGEMBALIAN" is centered above a table for returning vehicles. To the left, there's a section for "Data Penyewa" (Renter) with fields for "Kode Penyewa" and "Nama Penyewa". Below that is a "Data Mobil" (Vehicle Data) section with fields for "No. Polisi", "Merk Mobil", "Model Mobil", "Warna" (Color), "Tarif Sewa" (Daily Rate), "Denda Sewa" (Fines), "Status" (Status), and "Paket Sewa" (Rental Package). A "Lihat Data" (View Data) button is located at the bottom of this section. To the right of the main table, there's a "Data Supir" (Driver Data) section with fields for "Kode Supir", "Nama Supir", "No. SIM", and "Status". Below the table, there are several payment-related fields: "Uang Muka" (Down Payment), "Sisa Bayar" (Remaining Balance), "Total Bayar" (Total Payment), "Uang Bayar" (Payment Received), and "Kembali" (Change). On the far right, there are buttons for "TAMBAH" (Add), "SIMPAN" (Save), and "BATAL" (Cancel). A "Sewa Lagi" (Rent Again) button is located at the bottom right.

6. PERANCANGAN ISI

- Hubungan objek isi dengan objek isi lainnya adalah sebagai bagian dari suatu model kebutuhan untuk aplikasi web.
- Permasalahan yang terjadi pada perancangan isi jika jumlah objek isi yang digabungkan untuk membentuk halaman web tunggal merupakan fungsi dari kebutuhan *user*, yang dibatasi oleh kecepatan pengunduhan koneksi ke internet, juga dibatasi oleh besarnya ukuran jendela monitor yang digunakan *user*.

Contoh

Tampilan rancangan detail isi buku (objek)

A Web Page

http://uuuuybook.com

UuuuyBook.com Beranda Kategori Buku Konfirmasi Info email user

Semua Kategori > Nama Kategori > Detail buku

search

Judul

Harga

ISBN	:	qqqqqqqqqqqqqqqq
Nama Buku	:	xxxxxxxxxx
Penerbit	:	xxxxxxxxxx
Penulis	:	xxxxxxxxxx
Tanggal Terbit	:	dd/mm/yyyy
Kategori	:	xxxxxxxxxx
jumlah halaman	:	qqq
Jumlah buku	:	qqq

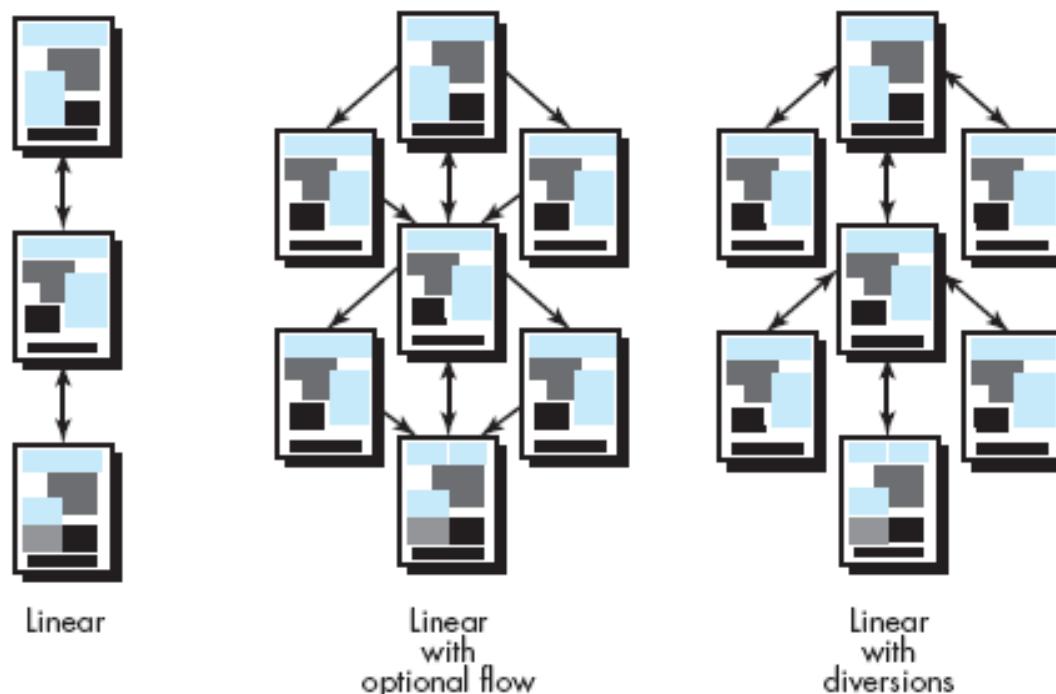
pesan sekarang tambah ke kranjang

7. PERANCANGAN ARSITEKTURAL

- Perancangan arsitektur web terkait dengan sasaran untuk aplikasi web, terkait dengan isi yang akan ditampilkan, *user*, dan navigasi.
- Arsitektur isi pada umumnya fokus pada bagaimana objek-objek isi distrukturkan agar layak untuk dipresentasikan kepada *user* dan menarik untuk ditelusuri.
- Aplikasi web distrukturkan untuk dapat mengelola interaksi *user* dengan aplikasi web, bagaimana menangani pekerjaan proses internal, bagaimana melakukan pengaturan navigasi, serta bagaimana menampilkan isinya.

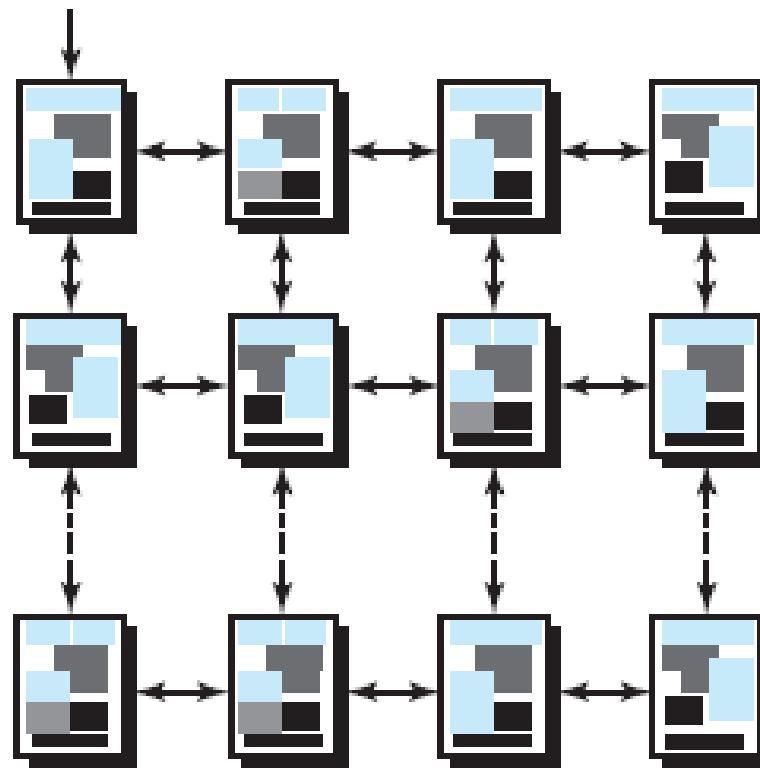
A. Arsitektur Isi

1. **Struktur Linier:** dilakukan saat interaksi *user* dengan aplikasi web secara umum memperlihatkan urutan yang dapat diramalkan. Contoh: urutan pemasukan pemesanan produk dimana informasi tertentu harus dalam urutan yang bersifat spesifik.



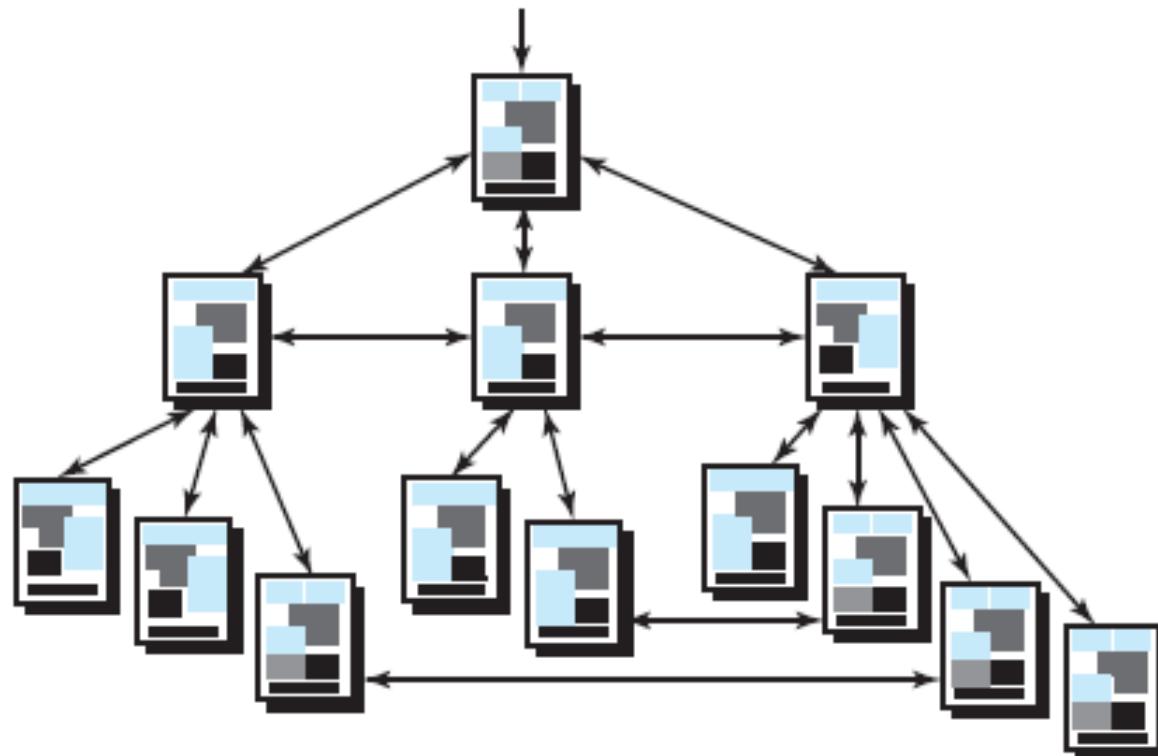
Arsitektur Isi (lanjutan)

2. **Struktur *Grid*:** arsitektur yang diterapkan saat isi aplikasi web dapat diorganisasikan menjadi 2 dimensi atau lebih. Dimensi horizontal merepresentasikan jenis-jenis produk, dan dimensi vertikal merepresentasikan penawaran yang disediakan oleh penjualnya.



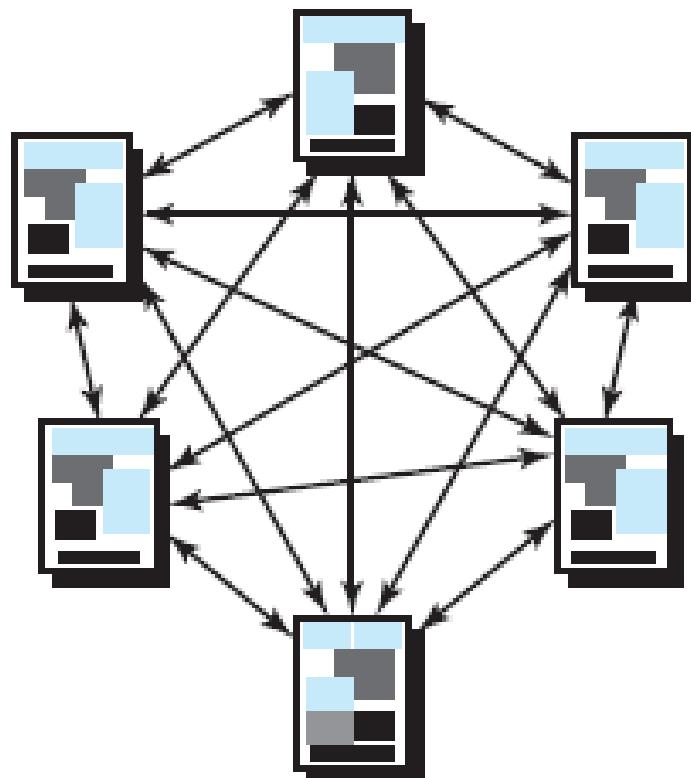
Arsitektur Isi (lanjutan)

3. **Struktur Hirarki:** rancangan dimungkinkan adanya pencabangan *hypertext* (aliran kendali) secara horizontal bergerak melintasi cabang-cabang vertikal pada struktur aplikasi web.



Arsitektur Isi (lanjutan)

4. **Struktur Jaringan (Web Murni):** struktur ini dapat digabungkan untuk membentuk struktur-struktur yang bersifat komposit.



B. Arsitektur Aplikasi Web

- Mendeskripsikan infrastruktur yang memungkinkan sistem atau aplikasi berbasis web untuk mencapai sasaran-sasaran bisnisnya.
- Arsitektur aplikasi web dibuat dalam 3 lapisan yang bertujuan untuk memisahkan antarmuka dari mekanisme navigasi dan dari perilaku yang dimiliki oleh aplikasi. Hal ini akan menyederhanakan implementasi aplikasi web dan akan meningkatkan penggunaan ulang komponen-komponennya.

Arsitektur Aplikasi Web (Lanjutan)

Aristektur MVC (*Model View-Controller*) secara fungsional dijelaskan sebagai berikut:

- a. **Pengendalian**: mengelola akses ke model, ke view dan melakukan koordinasi aliran data di antara model dan view.
- b. **Model**: memuat semua isi yang bersifat spesifik terhadap aplikasi dan memuat logika pemrosesan, termasuk di dalamnya semua objek isi.
- c. **View**: memuat semua fungsi yang bersifat spesifik terhadap *user interface* yang di dalamnya termuat presentasi isi dan logika pemrosesan, objek isi, akses ke data dan ke sumber informasi lainnya, dan semua fungsionalitas pemrosesan yang diperlukan *end user*.

8. PERANCANGAN NAVIGASI

- Merancang lintasan-lintasan navigasi yang memungkinkan *user* mengakses isi dan fungsi-fungsi aplikasi web.

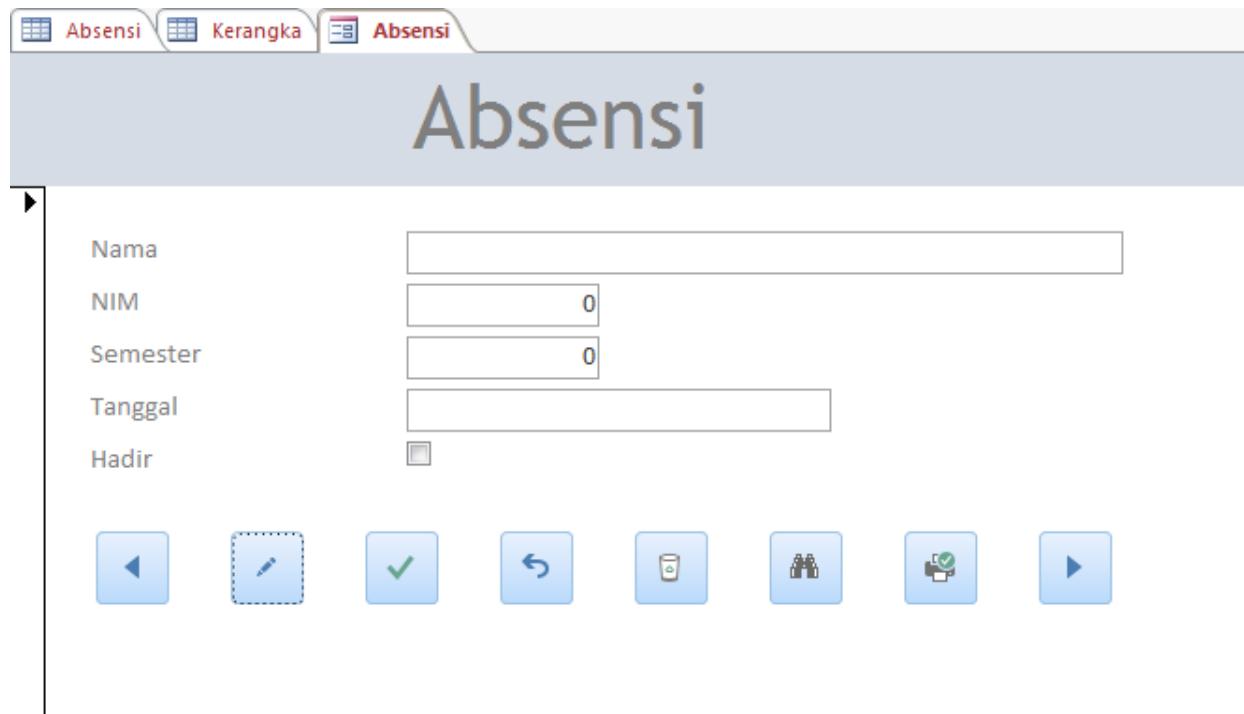
A. Semantik-Semantik Navigasi

- Unit semantik navigasi adalah sejumlah informasi dan struktur navigasi yang saling berkolaborasi untuk memenuhi kebutuhan *user* yang bersifat khusus.
- Contoh seorang pelanggan baru mungkin memiliki 3 *use case* yang berbeda yang semuanya menghasilkan akses ke informasi-informasi dan fungsi-fungsi aplikasi web yang berbeda. Semantik navigasi harus dibuat untuk masing-masing sasaran tersebut.

B. Sintak Navigasi

- **Link navigasi yang bersifat individual:** *link* berbasis teks, *icon*, tombol, pilihan, dan grafis harus sesuai dan konsisten dengan isi yang ditampilkan
- **Bar navigasi horizontal:** daftar kategori isi dan kategori fungsional utama yang berisi *link* yang sesuai
- **Kolom-kolom navigasi vertikal:**
 - Berisi daftar kategori utama dan kategori fungsional
 - Berisi semua objek isi utama
- **Tab-tab:** merepresentasikan kategori isi dan kategori fungsional sebagai *tab sheet* saat suatu *link* diperlukan
- **Peta situs:** menyediakan suatu tabel isi yang dapat digunakan untuk melakukan navigasi ke semua objek dan semua fungsionalitas

Contoh icon navigasi yang kurang jelas



Pertemuan 12

PENGUJIAN PERANGKAT LUNAK

1. DASAR-DASAR PENGUJIAN PL

- Pengujian perangkat lunak adalah proses menjalankan dan mengevaluasi sebuah PL secara manual maupun otomatis untuk menguji apakah PL sudah memenuhi persyaratan atau belum, atau untuk menentukan perbedaan antara hasil yang diharapkan dengan hasil sebenarnya.
- Pengujian bertujuan untuk mencari kesalahan.
- Pengujian yang baik adalah pengujian yang memiliki kemungkinan besar dalam menemukan kesalahan sebanyak mungkin dengan usaha sekecil mungkin.

A. Tujuan Pengujian

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

B. *Testability*

Testability adalah kemampuan PL untuk dapat diuji artinya seberapa mudah sebuah program komputer untuk bisa diuji.

Karakteristik *testability* PL:

- a. Kemampuan untuk bisa dioperasikan (*operability*)
- b. Kemampuan untuk bisa diobservasi (*observability*)
- c. Kemampuan untuk dapat dikontrol (*controllability*)
- d. Kemampuan untuk dapat disusun (*decomposability*)
- e. Kesederhanaan (*simplicity*)
- f. Stabilitas (*stability*)
- g. Kemampuan untuk dapat dipahami (*understandability*)

C. Karakteristik Pengujian

- a. Pengujian yang baik memiliki probabilitas tinggi untuk menemukan kesalahan
- b. Pengujian yang baik tidak berulang-ulang, waktu dan sumber daya pengujian terbatas
- c. Pengujian terbaik harus menjadi “bibit terbaik” yaitu pengujian yang memiliki kemungkinan tertinggi dalam mengungkap seluruh kelas kesalahan
- d. Pengujian yang baik tidak terlalu sederhana atau tidak terlalu rumit

2. PENGUJIAN WHITE BOX

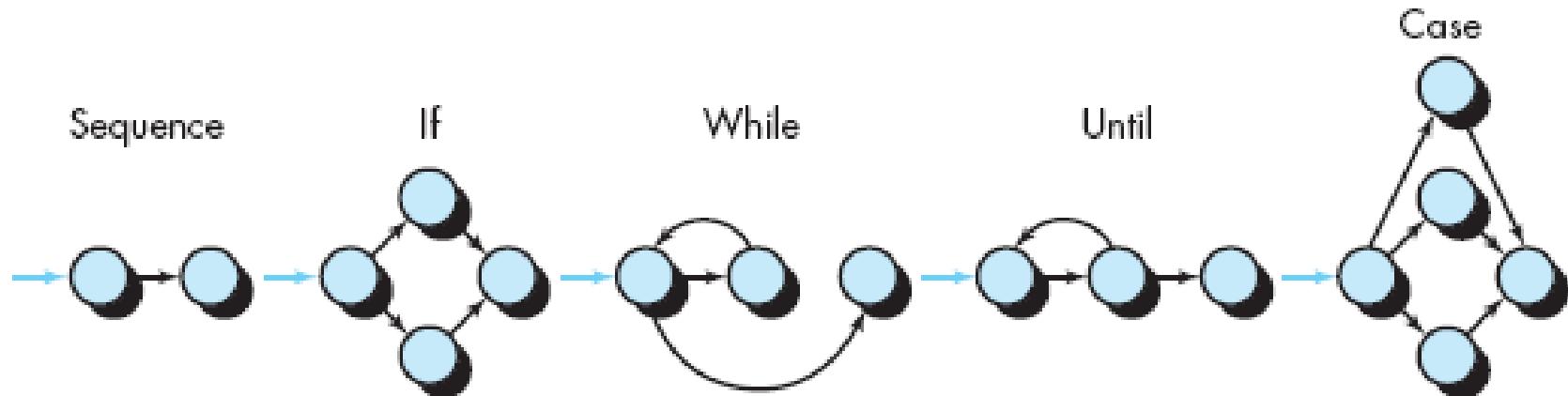
- Disebut juga pengujian kotak kaca (*glass box testing*).
- Merupakan sebuah filosofi perancangan *test case* yang menggunakan struktur kontrol.
- *Test case* pada *white box*:
 - a. Menjamin bahwa semua jalur independen di dalam modul telah dieksekusi sedikitnya satu kali
 - b. Melaksanakan semua keputusan logis pada sisi benar dan salah
 - c. Melaksanakan semua perulangan (*loop*) yang memenuhi semua batas operasional
 - d. Melakukan struktur data internal untuk memastikan kebenarannya

A. Pengujian Jalur Dasar (*Basis Path Testing*)

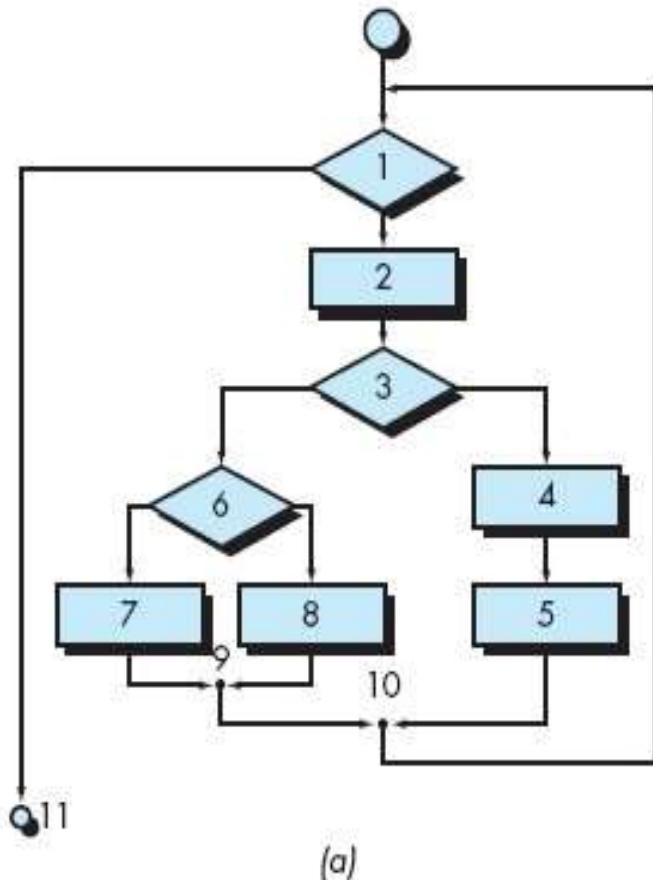
- Adalah teknik pengujian yang memungkinkan perancangan *test case* untuk menurunkan ukuran kompleksitas logis dari suatu rancangan prosedural dan menggunakan ukuran ini sebagai pedoman untuk menentukan rangkaian dasar jalur eksekusi.
- *Test case* diturunkan untuk menguji rangkaian dasar yang dijamin untuk mengeksekusi setiap pernyataan dalam program, setidaknya satu kali selama pengujian.
- Menggambarkan arus kontrol logis dengan menggunakan Notasi Grafik Alir (*Flow Graph*)

a. Notasi Grafik Alir (*Flow Graph*)

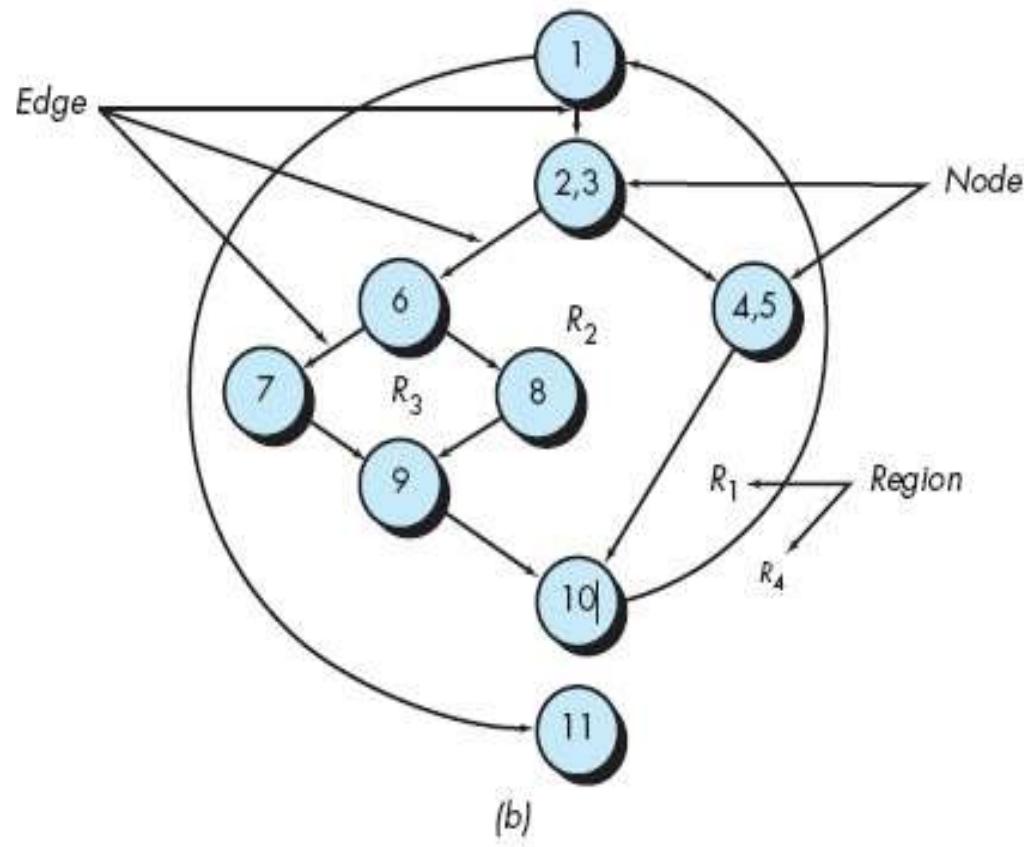
- Adalah notasi sederhana untuk merepresentasikan aliran kontrol logis.
- Lingkaran mewakili pernyataan kode program
- Notasi *flow graph* seperti gambar di bawah ini:



Untuk menggambarkan *flow graph*, dengan merepresentasikan perancangan prosedural seperti gambar berikut:



Notasi Flowchart

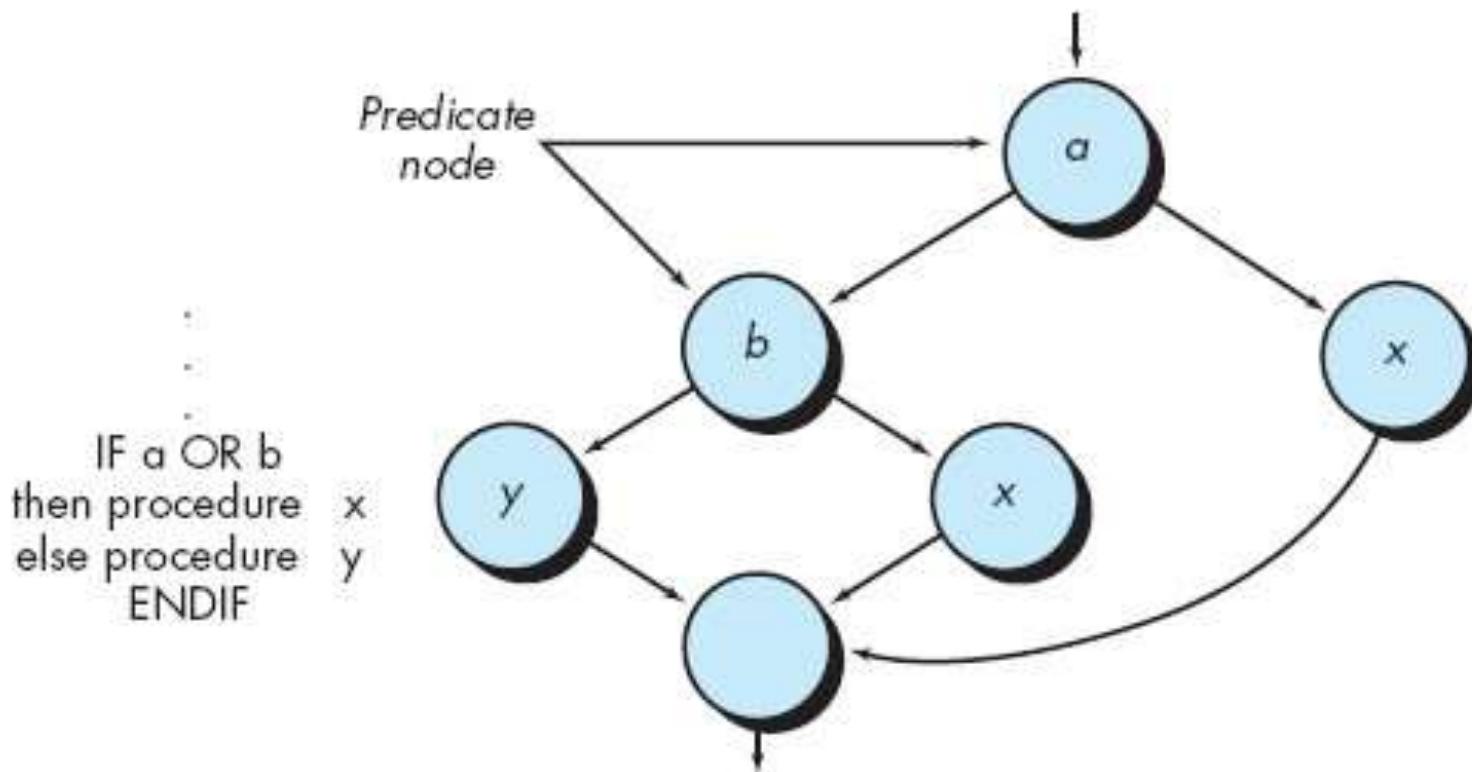


Flow Graph

b. Notasi *Flow Graph* (Lanjutan)

- Lingkaran menunjukkan **simpul** (*node*), merupakan satu atau lebih pernyataan-pernyataan prosedural
- Panah menunjukkan **edge** atau *link*, merupakan aliran kendali
- Area yang dibatasi oleh edge dan *node* disebut **region**.
- *Flow graph* menjadi rumit ketika adanya kondisi gabungan pada saat satu atau operator *boolean* ada dalam pernyataan bersyarat.
- *Node* yang berisi kondisi disebut **node predikat** dan ditandai oleh dua atau lebih *edge* yang berasal dari *node* tersebut.

Logika gabungan

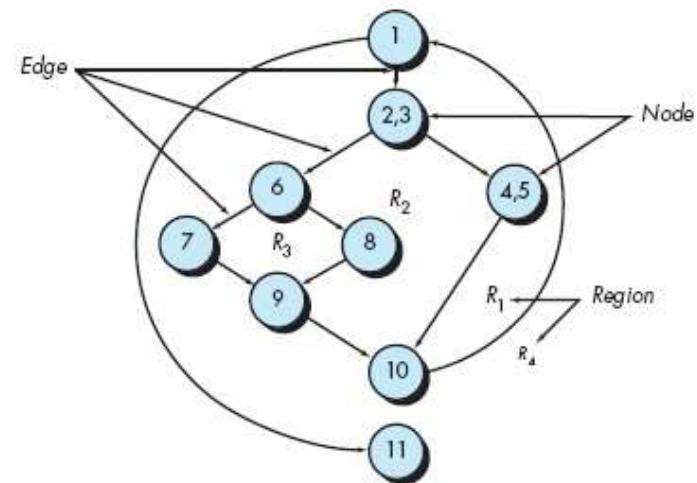


B. Jalur Independen

- Jalur Independen (*Independent Path*) adalah setiap jalur yang melalui program yang memperkenalkan setidaknya satu kumpulan pernyataan-pernyataan pemrosesan atau kondisi baru.

Jalur independen gambar disamping:

- **Path 1:** 1-11
- **Path 2:** 1-2-3-4-5-10-1-11
- **Path 3:** 1-2-3-6-8-9-10-1-11
- **Path 4:** 1-2-3-6-7-9-10-1-11
- Path 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11 tidak dianggap jalur independen karena tidak melintasi setiap *edge* baru
- Path 1 sampai 4 merupakan **basis set**.



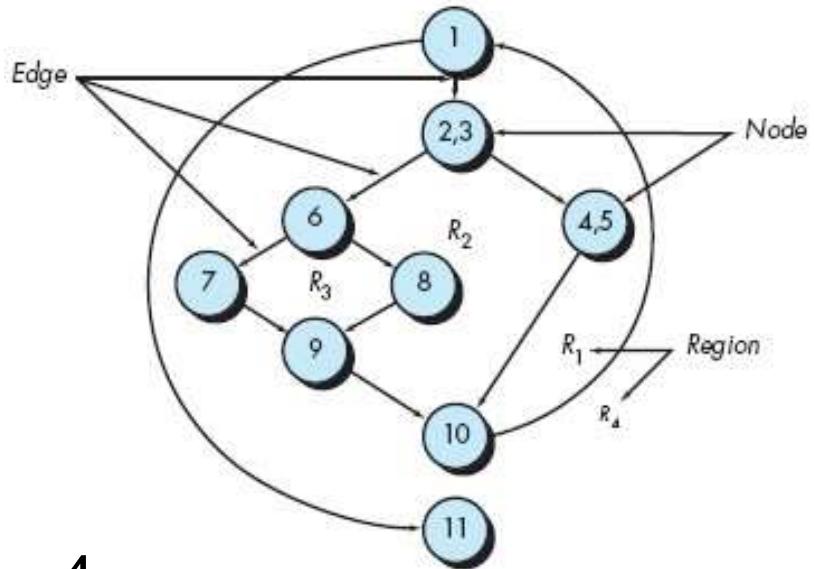
C. Kompleksitas Siklomatik

- Adalah metrik PL yang menyediakan ukuran kuantitatif dari kompleksitas logis suatu program.
- Perhitungan Kompleksitas Siklomatik:
 - a. Jumlah daerah-daerah (*region*) *flow graph* yang berhubungan dengan Kompleksitas Siklomatik
 - b. Kompleksitas Siklomatik $V(G) = E - N + 2$ dimana E adalah jumlah *edge*, N adalah jumlah *node*.
 - c. Kompleksitas Siklomatik $V(G) = P + 1$ dimana P adalah jumlah *node* predikat.

Kompleksitas Siklomatik (Lanjutan)

Dari kasus *independent path*:

- Jumlah *region* adalah 4
- $V(G) = 11 \text{ edge} - 9 \text{ node} + 2 = 4$
- $V(G) = 3 \text{ node predikat} + 1 = 4$
- Jadi Kompleksitas Siklomatiknya adalah 4



D. Menghasilkan *Test Case*

Diberikan Pseudocode sbb:

```
1 → do while record masih ada
      baca record
2 →   if record ke 1 = 0
3 →     then proses record
      simpan di buffer naikan
      counter
4 →     else if record ke 2 = 0
5 →       then reset counter
6 →     else proses record
      simpan pada file
7a →   endif
      endif
7b → enddo
8 → end
```

Langkah-langkah untuk menurunkan *basis set*:

1. Buat *flow graph*
2. Menentukan *independent path*
3. Menentukan kompleksitas siklomatik

Catatan: Dosen menjelaskan langkah-langkah di atas

E. Pengujian Struktur Kontrol

a. Pengujian Kondisi

- Pengujian kondisi adalah metode perancangan *test case* yang menguji kondisi logis yang terdapat dalam modul program.
- Kondisi sederhana adalah variabel *boolean* atau ekspresi relasional, kemungkinan didahului oleh satu operator NOT
- Jenis kesalahan dalam kondisi meliputi kesalahan operator *boolean*, kesalahan variabel *boolean*, kesalahan kurung *boolean*, kesalahan operator relasional, dan kesalahan ekspresi aritmatika.

b. Pengujian Perulangan

Adalah teknik pengujian *white box* yang fokus pada validitas konstruksi perulangan.

(1) Perulangan Sederhana

Pengujian dilakukan dengan mudah, dimana n jumlah maksimum yang diijinkan melewati perulangan:

- (a) Melewati perulangan secara keseluruhan
- (b) Hanya satu kali melalui perulangan
- (c) Dua kali melalui perulangan
- (d) Melalui perulangan sebanyak m dimana $m < n$
- (e) $n - 1, n, n + 1$ melalui perulangan

Contoh Perulangan Sederhana

Berikut diberikan pseudocode tentang perulangan sederhana, dan carilah hasilnya.

x : integer

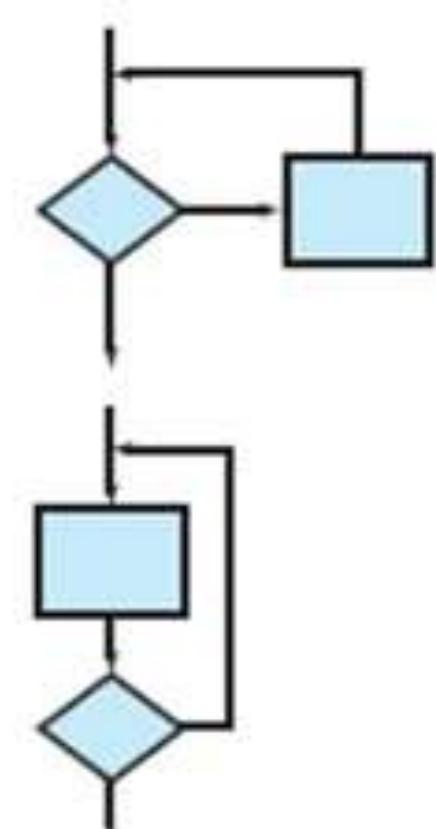
x = 1

WHILE (x < 5) DO

PRINT (x)

x = x + 1

ENDWHILE



(2). Perulangan Bersarang

Menggunakan pendekatan perulangan sederhana, sehingga jumlah pengujian akan meningkat.

Petunjuk pengujian:

- (a) Mulai dari perulangan terdalam dan atur semua perulangan ke nilai minimum
- (b) Lakukan pengujian perulangan sederhana untuk perulangan terdalam, sambil menjaga perulangan luar pada nilai minimum (misal *counter* perulangan)
- (c) Lanjutkan pada perulangan ke luar dan lakukan pengujian pada perulangan berikutnya
- (d) Lakukan sampai semua perulangan telah diuji

Contoh Perulangan Bersarang

Berikut diberikan pseudocode tentang perulangan bersarang, dan carilah hasilnya.

i, j : integer

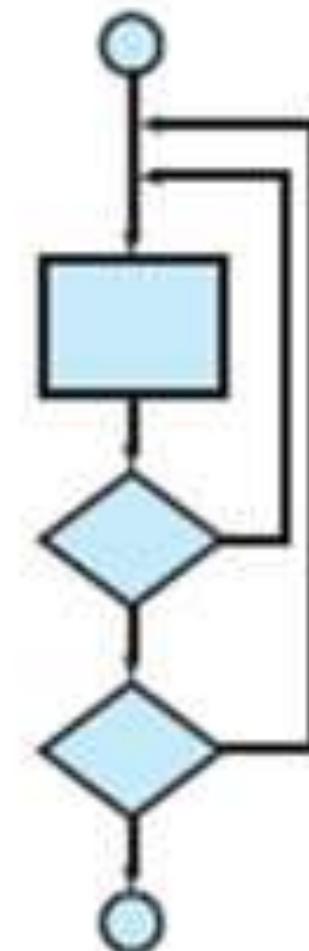
FOR i = 1 TO 4 DO

FOR j = 1 TO 3 DO

PRINT (i, j)

ENDFOR

ENDFOR



(3). Perulangan Terangkai

- Pengujian menggunakan pendekatan perulangan sederhana bila masing-masing perulangan independen.
- Tetapi bila dua perulangan dirangkai dan *counter* perulangan 1 digunakan sebagai harga awal perulangan 2 maka perulangan tersebut menjadi tidak independen, dan direkomendasikan ke perulangan tersarang

Contoh Perulangan Terangkai

x, y : integer

x = 0

y = 1

WHILE (x < 20) DO

PRINT (x)

x = x + 2

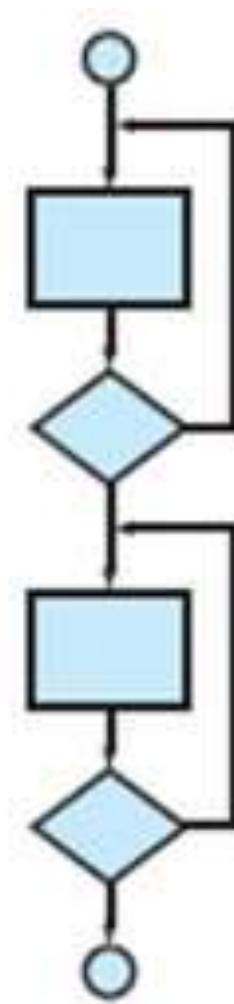
ENDWHILE

WHILE (y < 20) DO

PRINT (y)

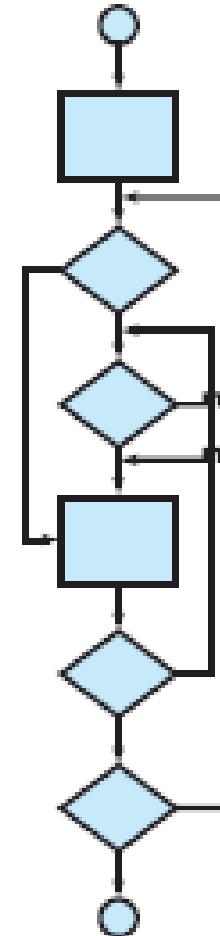
y = y + 2

ENDWHILE



(4). Perulangan Tak Terstruktur

Kapan saja memungkinkan, perulangan didisain kembali agar mencerminkan penggunaan konstruksi pemrograman terstruktur.



3. PENGUJIAN BLACK BOX

- Disebut juga pengujian perilaku.
- Pengujian *black box* memungkinkan untuk membuat beberapa kumpulan **kondisi input** yang akan melakukan semua kebutuhan fungsional untuk program.
- Kategori kesalahan pada pengujian *black box*:
 - a. Fungsi yang salah atau hilang
 - b. Kesalahan antarmuka
 - c. Kesalahan struktur data atau akses basis data eksternal
 - d. Kesalahan perilaku atau kinerja
 - e. Kesalahan inisialisasi dan penghentian

1. Metode Pengujian Berbasis Grafik

Langkah-langkah pengujian:

- Memahami objek-objek yang dimodelkan dalam PL dan penghubung yang menghubungkan objek-objek tersebut
- Menentukan serangkaian pengujian yang memastikan bahwa semua objek memiliki hubungan satu sama lain seperti yang diharapkan

Metode Pengujian Berbasis Grafik (Lanjutan)

- Node direpresentasikan sebagai lingkaran.
- Hubungan direpresentasikan dengan anak panah
- Hubungan satu arah (*directed link*) bahwa hubungan bergerak hanya satu arah.
- Hubungan dua arah atau hubungan simetris (*bidirection link*) bahwa hubungan berlaku dua arah.
- Hubungan paralel digunakan ketika ada sejumlah hubungan yang berbeda yang dibangun di antara *node-node* grafik.

2. Partisi Kesetaraan (*Equivalence Partitioning*)

- Adalah metode pengujian *black box* yang membagi daerah *input* program ke dalam kelas-kelas data dari *test case* yang dapat diturunkan.
- Sebuah kelas kesetaraan merepresentasikan keadaan valid atau tidak valid dari kondisi *input*.
- Contoh: kesalahan terhadap semua data karakter yang mungkin mengharuskan banyak *test case* sebelum kesalahan umum teramati.

Partisi Kesetaraan (Lanjutan)

Kelas kesetaraan dapat didefinisikan:

- Jika kondisi *input* menspesifikasikan range, satu kelas kesetaraan yang valid dan dua kelas kesetaraan yang tidak valid didefinisikan
- Jika kondisi *input* membutuhkan nilai tertentu, satu kelas kesetaraan yang valid dan dua kelas kesetaraan yang tidak valid didefinisikan
- Jika kondisi *input* menspesifikasikan anggota dari himpunan, satu kelas kesetaraan yang valid dan dua kelas kesetaraan yang tidak valid didefinisikan
- Jika kondisi *input* adalah *boolean*, satu kelas kesetaraan yang valid dan dua kelas kesetaraan yang tidak valid ditentukan

Contoh Pengujian *Equivalence Partitioning*

Ketentuan diskon 20% untuk pembelian barang sejenis sebanyak 2, dan diskon 30% untuk pembelian barang sejenis sebanyak 3 atau lebih

No	Test Case	Hasil yang diharapkan	Hasil yang diperoleh	Ket
1.	User mengisi jumlah barang = 0	Sistem tidak memproses transaksi dan menampilkan umpan balik	Sistem menampilkan pesan bahwa jumlah pembelian harus diisi	Berhasil
2.	User mengisi jumlah barang = 1	Sistem akan menghitung diskon dan subtotal	Sistem menghitung diskon sebesar 0 dan menghitung subtotal	Berhasil
3.	User mengisi jumlah barang = 2	Sistem akan menghitung diskon dan subtotal	Sistem menghitung diskon sebesar 20% dan menghitung subtotal	Berhasil
4.	User mengisi jumlah barang ≥ 3	Sistem akan menghitung diskon dan subtotal	Sistem menghitung diskon sebesar 30% dan menghitung subtotal	Berhasil

3. Analisis Nilai Batas (*Boundary Value Analysis*)

- Merupakan teknik perancangan *test case* yang melengkapi partisi kesetaraan dengan fokus pada kondisi *input*, dan juga akan menghasilkan *output*.
- Banyak kesalahan terjadi pada kesalahan *input*.
- BVA mengijinkan untuk menyeleksi kasus uji yang menguji batasan nilai input.
- BVA merupakan komplemen dari *equivalence partitioning*, lebih memilih pada elemen-elemen di dalam kelas ekivalen pada bagian sisi batas dari kelas

Pedoman BVA

- a. Jika kondisi *input* menspesifikasikan range yang dibatasi oleh nilai a dan b, *test case* harus dirancang dengan nilai a dan b dan hanya di atas dan di bawah nilai a dan b
- b. Jika kondisi *input* menspesifikasikan sejumlah nilai, *test case* harus dikembangkan untuk menguji jumlah-jumlah minimum dan maksimum.
- c. Terapkan pedoman 1 dan 2 untuk kondisi *input*.
- d. Jika struktur data program internal memiliki batas-batas yang telah ditentukan, pastikan untuk merancang *test case* untuk menguji struktur data pada batasnya.

Contoh Pengujian BVA

Jika sistem menampilkan jumlah stok barang adalah n, maka:

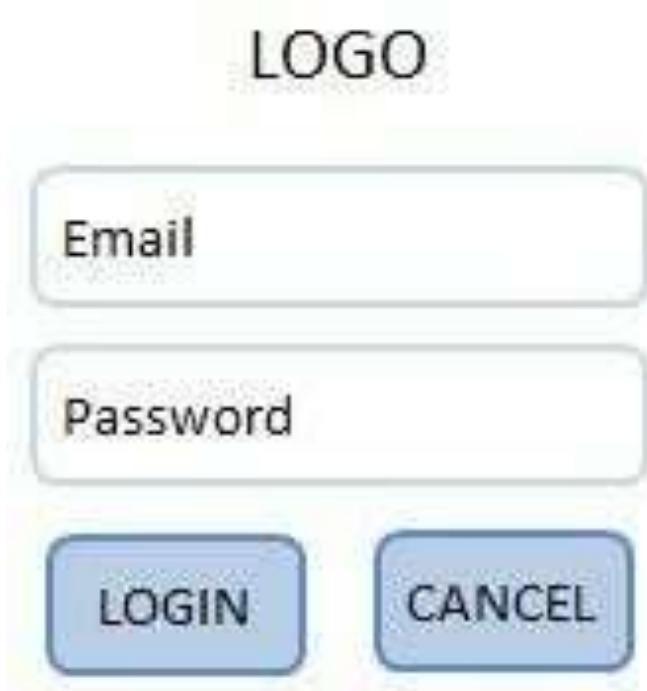
Noc	Test Case	Hasil yang diharapkan	Hasil yang diperoleh	Ket
1.	User mengisi stok = 0	Sistem tidak memproses transaksi dan memberikan umpan balik	Sistem menampilkan pesan bahwa jumlah pembelian harus diisi	Berhasil
2.	User mengisi stok = 1 sampai n	Sistem akan memproses transaksi dan mengurangi jumlah stok	Sistem menghitung subtotal dan mengurangi jumlah stok	Berhasil
3.	User mengisi stok > n	Sistem tidak memproses transaksi dan memberikan umpan balik	Sistem menampilkan pesan bahwa jumlah pembelian melebihi stok	Berhasil

Catatan: sebaiknya tidak menuliskan “sesuai harapan” pada kolom ini

4. Pengujian Larik Ortogonal

- Dapat diterapkan untuk masalah-masalah dimana *input domain* relatif kecil tapi terlalu besar untuk mengakomodasi pengujian yang lengkap.
- Bermanfaat dalam menemukan kesalahan yang terkait dengan logika yang salah dalam komponen PL

Contoh:



Jika rancangan aplikasi untuk login seperti gambar di atas, maka *black box testing* dapat dibuat berdasarkan beberapa kondisi input (5 kondisi)

Contoh pengujian keamanan pada User Login

No	Test Case	Hasil yang diharapkan	Hasil yang diperoleh	Ket
1.	User tidak mengisi <i>username</i> atau <i>password</i> atau keduanya	User tidak bisa masuk ke dalam sistem	Sistem tetap pada form Login dan menampilkan pesan untuk mengisi <i>username</i> dan <i>password</i>	Berhasil
2.	User mengisi <i>username</i> saja atau <i>password</i> saja	User tidak bisa masuk ke dalam sistem	Sistem akan menampilkan pesan untuk mengisi <i>username</i> atau <i>password</i>	Berhasil
3.	User salah mengisi <i>username</i>	User tidak bisa masuk ke dalam sistem	Sistem akan menampilkan pesan jika <i>username</i> atau <i>password</i> salah	Berhasil
4.	User salah mengisi <i>password</i>	User tidak bisa masuk ke dalam sistem	Sistem akan menampilkan pesan jika <i>username</i> atau <i>password</i> salah	Berhasil
5.	User salah mengisi <i>username</i> dan <i>password</i>	User tidak bisa masuk ke dalam sistem	Sistem akan menampilkan pesan jika <i>username</i> atau <i>password</i> salah	Berhasil

Pertemuan 13

PENGUJIAN APLIKASI WEB

1. PENGERTIAN

- Adalah serangkaian aktivitas yang tujuannya untuk menemukan kesalahan dalam isi, fungsi, kegunaan, kemampuan navigasi, kinerja, kapasitas dan keamanan aplikasi web sebelum aplikasi-aplikasi web yang dibuat dikirimkan ke *end user*.
- Hal ini penting karena jika *end user* menemukan kesalahan yang membuat mereka meragukan aplikasi web tersebut, mereka akan pergi ke web lain untuk mencari isi dan informasi
- Langkahnya dimulai dengan fokus pada aspek aplikasi web yang terlihat oleh *user* dan berlanjut pada pengujian yang terkait dengan teknologi dan infrastruktur.

2. KONSEP PENGUJIAN UNTUK APLIKASI WEB

A. Dimensi Kualitas

Kualitas dievaluasi dengan menerapkan serangkaian tinjauan teknis yang melihat berbagai elemen dari model perancangan dan dengan menerapkan proses pengujian.

Atribut Dimensi Kualitas

1. Isi (**content**)

Dievaluasi di tingkat sintak dan semantik. Pada tingkat sintak dokumen berbasis teks diuji dalam hal ejaan, tanda baca dan tata bahasa. Pada tingkat semantik aspek yang dinilai adalah kebenaran informasi yang disajikan, konsistensi di seluruh objek isi dan objek terkait, dan rendahnya ambiguitas

2. Fungsi

Diuji untuk menemukan kesalahan yang menunjukkan ketidaksesuaian dengan persyaratan *customer*

3. Struktur

Dinilai untuk memastikan bahwa aplikasi web benar-benar menyediakan isi dan fungsi aplikasi web

Atribut Dimensi Kualitas (Lanjutan)

4. Kegunaan

Diuji untuk memastikan bahwa setiap kategori *user* didukung oleh antarmuka yang *user friendly* serta menerapkan semua sintak dan semantik navigasi yang diperlukan

5. Kemampuan untuk dapat dinavigasi

Diuji untuk memastikan bahwa semua sintak dan semantik navigasi dilakukan untuk menemukan kesalahan, seperti *link* yang salah dan *dead link*

6. Kinerja

Diuji di bawah berbagai kondisi operasi, konfigurasi, dan *loading*

Atribut Dimensi Kualitas (Lanjutan)

7. Kompatibilitas

Diuji dengan menjalankan aplikasi web dalam berbagai konfigurasi *host* yang berbeda baik di sisi *server* maupun *client*

8. Interoperabilitas

Diuji untuk memastikan bahwa aplikasi web berantarmuka dengan benar dengan aplikasi lain dan *database*

9. Keamanan

Diuji dengan menilai kerentanan potensial

Kesalahan Atribut pada Pengujian Aplikasi Web

- a. Pengujian web mengungkap masalah yang didapatkan pertama kali di sisi *client*, melalui antarmuka yang implementasinya pada sebuah *browser* atau perangkat komunikasi pribadi
- b. Aplikasi web diimplementasikan pada beberapa konfigurasi dan lingkungan yang berbeda, kemungkinan sulit untuk menemukan kesalahan di luar lingkungan tempat kesalahan pertama kali ditemukan

Kesalahan Atribut pada Pengujian Aplikasi Web

- c. Kesalahan akibat kode program yang tidak tepat (misal HTML), seperti kesalahan penelusuran dokumen
- d. Kesalahan pada *client-server* sulit dilacak di tiga lapisan: *client*, *server*, atau jaringan
- e. Beberapa kesalahan yang berada di lingkungan operasi yang bersifat statis, sementara yang lain terkait dengan lingkungan operasi yang bersifat dinamis

B. Strategi Pengujian

Langkah-langkahnya:

1. Model konten untuk aplikasi web ditinjau untuk menemukan kesalahan
2. Model antarmuka ditinjau untuk memastikan bahwa semua *use-case* dapat diakomodasi
3. Model perancangan ditinjau untuk mengungkap kesalahan navigasi
4. Antarmuka pengguna diuji untuk mengungkap kesalahan dalam presentasi dan/atau mekanik navigasi
5. Komponen fungsional diuji untuk setiap unit

Langkah-Langkah Strategi Pengujian (Lanjutan)

6. Navigasi untuk seluruh arsitektur diuji
7. Aplikasi web diimplementasikan dalam berbagai konfigurasi lingkungan yang berbeda dan diuji kompatibilitasnya
8. Pengujian keamanan dilakukan dalam upaya untuk mengungkap kelemahan aplikasi web
9. Pengujian kinerja dilakukan
10. Aplikasi web diuji oleh *end user*, hasil interaksinya dievaluasi untuk menemukan kesalahan isi dan navigasi, keamanan, keandalan, dan kinerja aplikasi web

C. Perencanaan Pengujian

Sebuah rencana aplikasi web mengidentifikasikan:

- a. Himpunan tugas-tugas yang diterapkan ketika pengujian dimulai
- b. Produk kerja yang dihasilkan ketika setiap tugas pengujian dijalankan
- c. Cara dimana hasil pengujian dievaluasi, dicatat, dan digunakan kembali saat pengujian regresi dilakukan

3. PENGUJIAN ISI

Pengujian isi menggabungkan baik peninjauan maupun pembuatan *test case* yang dapat dilaksanakan

A. Tujuan Pengujian Isi

- Untuk mengungkap kesalahan sintak dengan memeriksa ejaan dan tata bahasa otomatis
- Untuk mengungkap kesalahan semantik yang fokus pada informasi pada setiap isi objek
- Untuk mencari kesalahan dalam pengaturan atau struktur isi dalam susunan dan hubungan yang tepat

Contoh:

KATEGORI
BAJU (4)
CELANA (3)
DOMPET (2)
ROMPI (2)
SWEETER (1)
TAS (4)
TOPI (3)

Penulisan sintak yang salah, dapat memberikan semantik yang berbeda



B. Pengujian Basis Data

Pengujian basis data menjadi sulit dikarenakan:

- a. Permintaan dari *client* jarang disajikan dalam bentuk yang dapat dimasukkan ke sistem manajemen basis data
- b. Basis data dapat berada jauh dari *server*
- c. Data mentah yang diperoleh dari basis data harus dikirim ke *server* aplikasi web dan diformat dengan benar untuk pengiriman selanjutnya kepada *client*
- d. Objek isi yang bersifat dinamis harus dikirim ke *client* dalam bentuk yang dapat ditampilkan kepada *end user*

Pengujian Basis Data (Lanjutan)

Pengujian basis data harus memastikan bahwa:

- a. Informasi yang valid dilewatkan antara *client* dan *server* dari lapisan antarmuka
- b. Proses aplikasi web menuliskan ekstraksi atau format data *user* dengan baik dan benar
- c. Data *user* diberikan dengan tepat untuk fungsi transformasi data pada sisi *server* yang membentuk *query* yang sesuai
- d. Query yang dilewatkan ke *layer* manajemen data yang berkomunikasi dengan rutin-rutin akses basis data terletak di komputer lain

4. PENGUJIAN ANTARMUKA

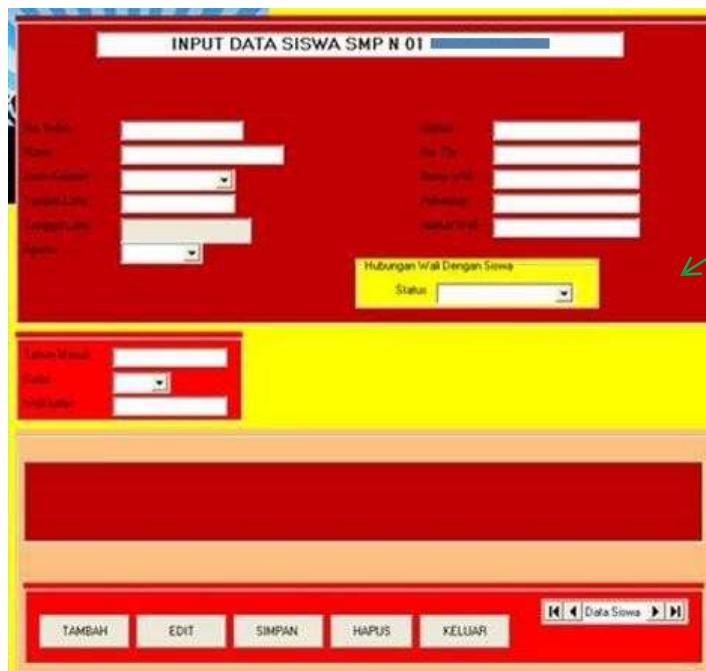
- Untuk memastikan bahwa *script* yang benar dibangun untuk setiap permintaan *user* dan benar dikirimkan ke *server*.
- Verifikasi dan validasi antarmuka *user* terjadi pada:
 - a. Model antarmuka memastikan bahwa telah sesuai dengan kebutuhan *stakeholder* dan elemen lain.
 - b. Model perancangan antarmuka ditinjau untuk memastikan bahwa kriteria kualitas generik telah ditetapkan untuk semua antarmuka.
 - c. Selama pengujian fokus pada interaksi *user*.

A. Strategi Pengujian Antarmuka

Langkah-langkahnya:

1. Fitur-fitur antarmuka diuji seperti jenis huruf, warna, gambar, *border*, tabel dll
2. Mekanisme antarmuka diuji dengan cara yang sama dengan pengujian unit, misalnya pengujian untuk keranjang belanja pada *e-commerce*, isi *streaming*, penulisan *script* dll
3. Mekanisme antarmuka diuji dalam konteks penggunaan *use case* untuk kategori *user* tertentu
4. Antarmuka lengkap diuji terhadap *test case* terpilih
5. Antarmuka diuji dalam berbagai lingkungan

Contoh penggunaan warna, teks dan gambar pada antarmuka



Teks sulit dibaca

Contoh pengujian pada antarmuka yang berbeda

Menu

Home Member Transaksi Cek Buku

Form Transaksi Peminjaman

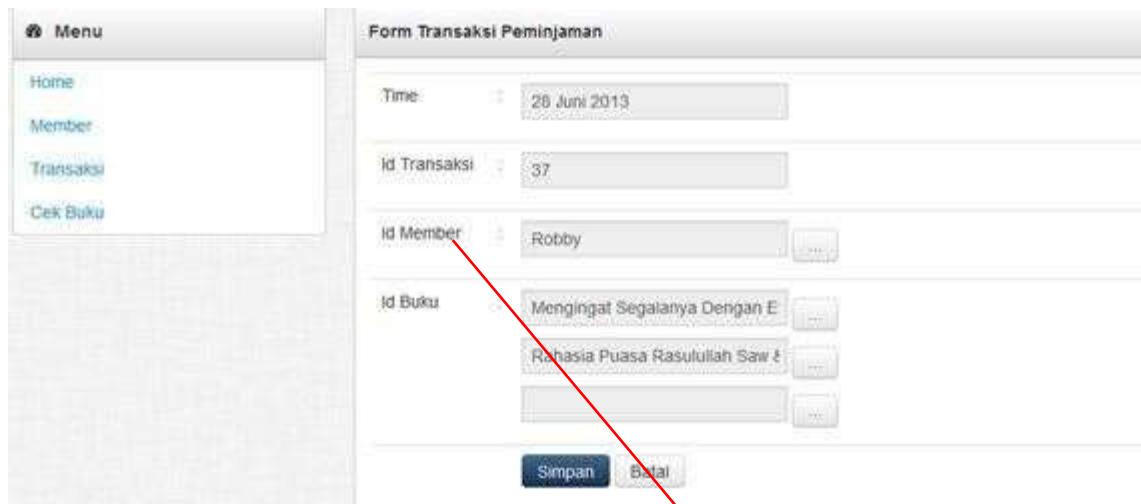
Time: 26 Juni 2013

Id Transaksi: 37

Id Member: Robby

Id Buku: Mengingat Segalanya Dengan E...
Rahasia Puasa Rasulullah Saw E...
...

Simpan Batal



Data tidak sama

Menu

Home Member Transaksi Cek Buku

Member

Pencarian: (ID Member atau Nama) Go

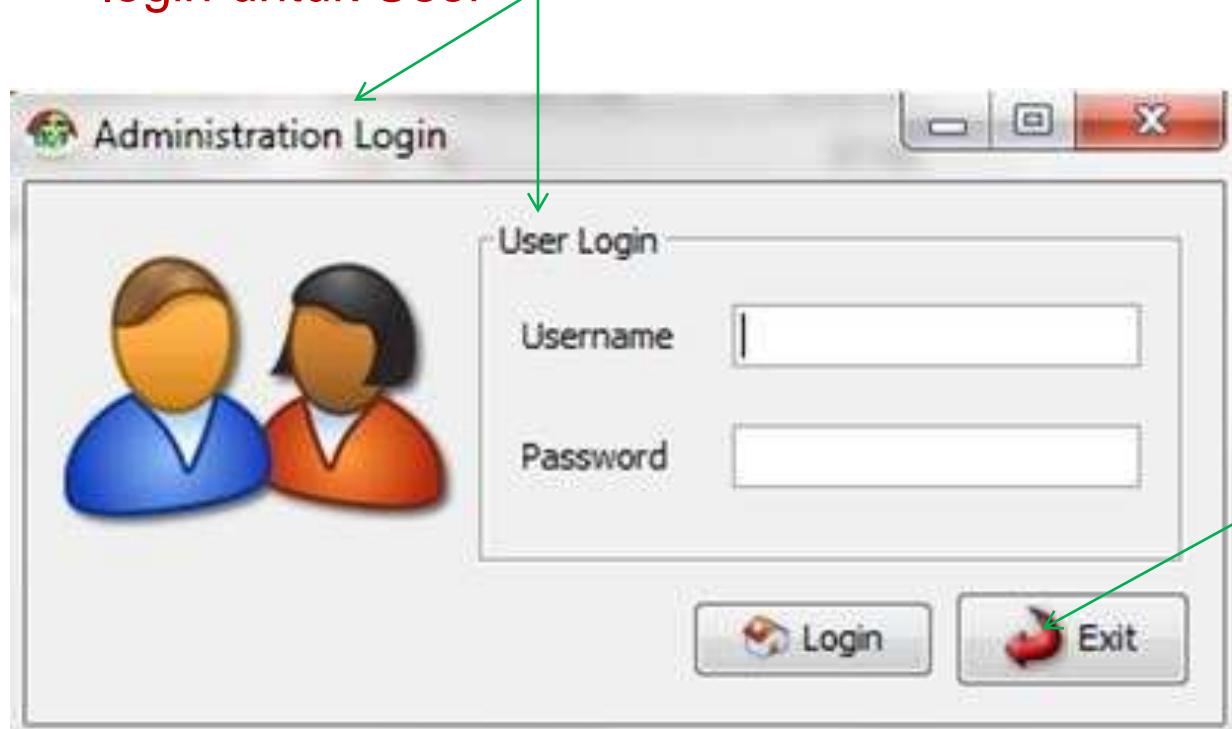
Registrasi Member

No	ID Member	Nama Member	Email	HP	Aksi
1	1306001	Robby	[REDACTED]	[REDACTED]	/EX
2	1306002	Andre	[REDACTED]	[REDACTED]	/EX

Contoh

Antarmuka diuji dalam berbagai lingkungan:

Lingkungan Administrator tetapi
login untuk User

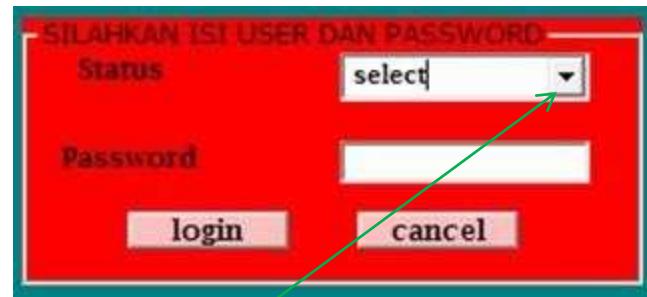


Gunakan simbol
icon yang umum
untuk exit

B. Mekanisme Pengujian Antarmuka

1. **Formulir.** Pengujian untuk memastikan:
 - a. Label formulir dapat diidentifikasi secara visual
 - b. Server menerima semua informasi form
 - c. *Default* yang tepat saat *user* tidak memilih dari menu *pull down* atau dari tombol
 - d. Fungsi-fungsi perambah seperti tanda panah *back* tidak merusak data yang diisikan ke dalam form
 - e. *Script* yang memeriksa kesalahan *input* data
 - f. Lebar kolom dan jenis data yang tepat
 - g. Mencegah *user* memasukkan *string text* lebih panjang dari jumlah max. yang telah ditetapkan
 - h. Menu *pull down* diurutkan dan dapat dipahami *user*
 - i. *Auto-fill* tidak mengarah ke kesalahan *input* data
 - j. *Key tab* memicu perpindahan di antara kolom

Contoh input data:



Sebaiknya user tidak
diberikan pilihan dengan
combo box

Contoh untuk user yang mengisi data sendiri

Aplikasi Tiket

Kode Kereta	KAIZO
Nama Kereta	
Jurusan	
Harga	
Jumlah Tiket	
<input type="button" value="HITUNG"/>	
Total Bayar	
<input type="button" value="Bersih"/> <input type="button" value="Keluar"/>	

User memasukkan
data sendiri,
seharusnya tampil
pada saat kode
kereta dipilih

Mekanisme Pengujian Antarmuka (Lanjutan)

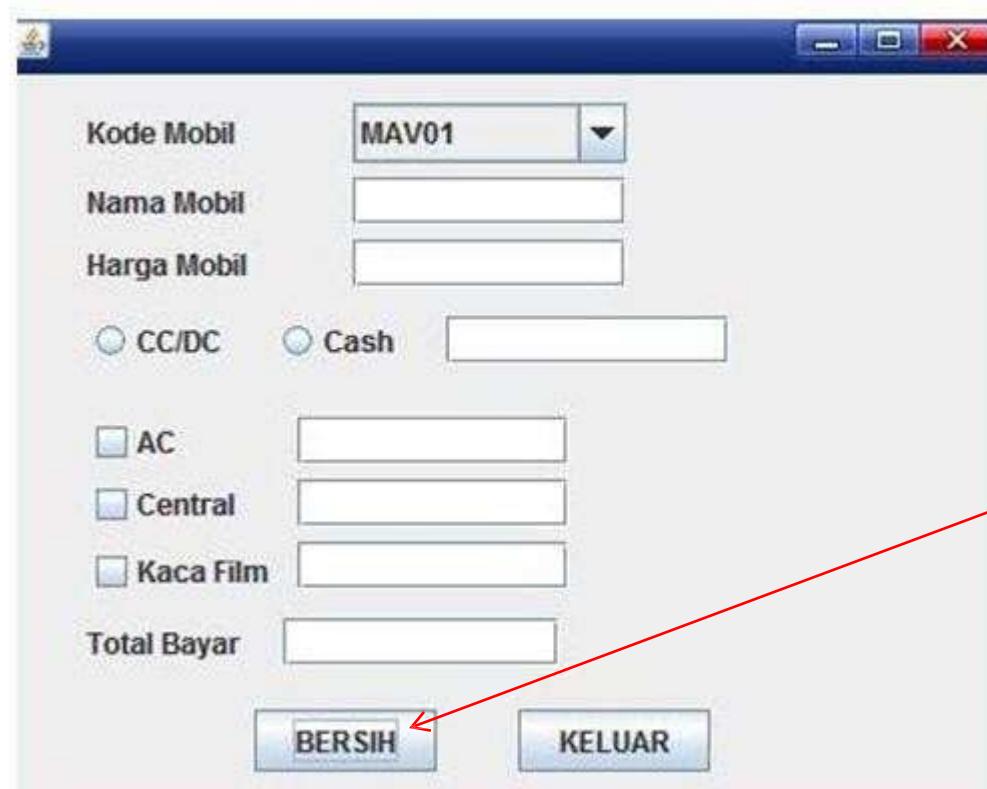
2. ***Link***. Setiap *link* navigasi diuji untuk memastikan bahwa objek isi atau fungsi yang tepat dapat dicapai
3. ***Client-side scripting***. Pengujian dilakukan untuk menemukan kesalahan saat *script* dijalankan
4. ***HTML dinamis***. Dijalankan untuk memastikan bahwa tampilan dinamis sudah benar
5. ***Pop-up windows***. Memastikan bahwa
 - a. *Pop-up* diukur dan diposisikan dengan benar
 - b. *Pop-up* tidak menutupi jendela aplikasi web asli
 - c. Perancangan *pop-up* konsisten dengan perancangan antarmuka
 - d. *Scroll bar* dan mekanisme kontrol lainnya yang ditambahkan ke *pop-up* diletakkan dengan benar

Mekanisme Pengujian Antarmuka (Lanjutan)

6. **Script CGI.** Pengujian kotak hitam dilakukan dengan penekanan pada integritas data saat dilewatkan ke *script CGI*
7. **Streaming content.** Pengujian menunjukkan bahwa data *streaming* terbarukan, ditampilkan dengan benar dan dapat dihentikan tanpa kesalahan dan *restart* tanpa kesulitan
8. **Mekanisme aplikasi antarmuka spesifik.** Pengujian sesuai dengan daftar fungsi dan fitur yang didefinisikan pada antarmuka

Contoh fitur yang tidak disediakan pada Input Data

Form Input Data Mobil



Command yang
membingungkan
dan tidak ada
command SIMPAN

C. Pengujian Kompatibilitas

Aplikasi web harus dapat dijalankan pada komputer yang berbeda, berupa:

- Perangkat tampilan
- Sistem Operasi
- *Browser*
- Kecepatan koneksi jaringan

Pengujian Kompatibilitas (Lanjutan)

Langkah-langkah uji kompatibilitas:

1. Mendefinisikan sekumpulan konfigurasi komputasi di sisi *client*, mengidentifikasi *platform*, perangkat layar, sistem operasi, *browser* yang tersedia, kecepatan koneksi internet, dll.
2. Melakukan serangkaian uji validasi kompatibilitas berupa pengujian navigasi, pengujian kinerja, dan pengujian keamanan

5. PENGUJIAN NAVIGASI

Tugas pengujian navigasi:

1. Memastikan bahwa semua mekanisme yang memungkinkan pengguna aplikasi web melakukan penelusuran melalui aplikasi web.
2. Untuk memvalidasi bahwa setiap unit semantik navigasi dapat dicapai oleh kategori pengguna yang tepat

Pengujian Sintaks Navigasi

1. *Link Navigasi*

Mekanisme menyertakan *link* internal dalam aplikasi web dan *link* eksternal ke aplikasi web lain, dan *anchor* pada halaman web tertentu

2. *Redirect*

Link beraksi saat *user* meminta URL yang tidak ada atau memilih sebuah *link* yang isinya telah dihapus atau namanya telah berubah

3. *Bookmark*

Memastikan bahwa judul halaman yang berarti dapat diekstraksi saat *bookmark* dibuat

Pengujian Sintaks Navigasi (Lanjutan)

4. ***Frame and frameset***:

Frameset berisi beberapa *frame* dan memungkinkan untuk menampilkan beberapa halaman web secara bersamaan. Oleh karena itu harus diuji dalam hal isi, tata letak layar dan ukuran yang tepat, kinerja *download*, dan kompatibilitas *browser*.

5. ***Site map***

Berisi daftar isi lengkap pada semua halaman web, setiap *entry* harus diuji untuk memastikan bahwa *link* membawa *user* membaca isi/fungsionalitas yang tepat

6. ***Search engine internal***

User mengetikkan kata kunci untuk menemukan isi yang diperlukan

6. PENGUJIAN KONFIGURASI

A. Masalah di bagian *Server*

- Aplikasi web sepenuhnya kompatibel dengan server OS
- Berkas sistem, direktori, dan data yang terkait dibuat dengan benar saat aplikasi web dioperasikan
- Keamanan sistem mengijinkan aplikasi web untuk berjalan dan melayani *user* tanpa gangguan atau penurunan kinerja
- Aplikasi web terintegrasi secara tepat dengan perangkat lunak basis data
- *Script* aplikasi web sisi *server* mengeksekusi dgn benar
- Jika *proxy server* yang digunakan, apakah perbedaan konfigurasi telah diatasi melalui pengujian

B. Masalah di bagian *Client*

Pengujian konfigurasi fokus pada kompatibilitas aplikasi web pada komponen berikut:

- *Hardware*: CPU, memori, penyimpanan, perangkat cetak
- Sistem operasi
- *Browser*: *Firefox*, *Safari*, *IE*, *Opera*, *Chrome*
- Komponen antarmuka: *Active-X*, *Java applet*
- *Plug in*: *Quick Time*, *RealPlayer*
- Konektivitas: kabel, DSL, Wifi

7. PENGUJIAN KEAMANAN

Untuk menyelidiki kerentanan lingkungan di sisi *client*, komunikasi jaringan yang terjadi saat data dilewatkan dari *client* ke *server*, dan di lingkungan *server* itu sendiri.

- Contoh kerentanan yang dapat terjadi:
 - *Buffer overflow*, seperti memasukkan URL yang lebih panjang dari ukuran *buffer*
 - Akses tidak sah
 - *Spoofing*
 - Serangan DOS
- Implementasi keamanan:
 - *Firewall*
 - *Authentication*
 - *Encryption*
 - *Authorization*

Pertemuan 14

**IMPLEMENTASI
dan
PEMELIHARAAN**

1. IMPLEMENTASI PL

IMPLEMENTASI

- Perancangan dan implementasi PL adalah tahap dalam proses RPL dimana dikembangkan sistem PL yang dapat dieksekusi.
- Implementasi adalah proses mewujudkan desain sebagai sebuah program.
- RPL mencakup semua kegiatan yang terlibat dalam pengembangan PL dari persyaratan awal sistem hingga pemeliharaan dan pengelolaan sistem yang digunakan.
- Implementasi dapat melibatkan pengembangan program atau menyesuaikan dan mengadaptasi sistem generik, *off-the-shelf* untuk memenuhi persyaratan khusus dari suatu organisasi.

IMPLEMENTASI (Lanjutan)

Aspek implementasi yang sangat penting untuk RPL:

1. *Reuse*

Sebagian besar PL modern dibangun dengan menggunakan kembali komponen atau sistem yang ada.

2. *Configuration Management*

Selama proses pengembangan, banyak versi yang berbeda dari setiap komponen PL.

3. *Host-Target Development*

Produksi PL biasanya tidak dijalankan pada komputer yang sama dengan lingkungan pengembangan PL. Pengembangan pada satu komputer (sistem *host*) dan dijalankan pada komputer yang terpisah (sistem *target*).

A. *Reuse*

Penggunaan ulang (*reuse*) PL dimungkinkan pada sejumlah level yang berbeda:

1. Tingkat Abstraksi.

Pada tingkat ini, tidak menggunakan *reuse software* secara langsung tetapi menggunakan pengetahuan abstraksi dalam desain PL menggunakan pola desain dan pola arsitektur.

2. Tingkat Objek.

Pada tingkat ini, langsung menggunakan *reuse objects* dari *library* daripada menulis kode sendiri.

Reuse (Lanjutan)

3. Tingkat Komponen.

Komponen adalah kumpulan objek dan kelas objek yang beroperasi bersama untuk menyediakan fungsi dan layanan terkait. Pada tingkat ini harus menyesuaikan dan memperluas komponen dengan menambahkan beberapa kode sendiri.

4. Tingkat Sistem.

Pada tingkat ini, menggunakan kembali seluruh sistem aplikasi. Biasanya melibatkan beberapa jenis konfigurasi sistem. Dapat dilakukan dengan menambahkan dan memodifikasi kode atau dengan menggunakan antarmuka konfigurasi sistem sendiri.

Reuse (Lanjutan)

Keuntungan menggunakan *reuse software* yang ada:

- a. Dapat mengembangkan sistem baru dengan lebih cepat
- b. Dengan risiko pengembangan yang lebih sedikit dan juga biaya yang lebih rendah
- c. Karena *reuse software* yang digunakan telah diuji dalam aplikasi lain, sehingga lebih dapat diandalkan daripada PL baru

Reuse (Lanjutan)

Biaya yang terkait dengan penggunaan kembali:

1. Biaya waktu yang dihabiskan dalam mencari PL untuk digunakan kembali dan menilai apakah sudah memenuhi kebutuhan atau tidak, dan menguji PL untuk memastikan bahwa dapat bekerja di lingkungan sistem.
2. Biaya membeli PL yang dapat digunakan kembali.
3. Biaya untuk adaptasi dan mengkonfigurasi komponen PL/sistem yang dapat digunakan kembali.
4. Biaya untuk mengintegrasikan elemen *reuse software* yang dapat digunakan satu sama lain (jika menggunakan PL dari sumber yang berbeda) dan dengan kode baru yang telah dikembangkan.

B. Manajemen Konfigurasi **(*Configuration Management*)**

- Manajemen konfigurasi merupakan proses rekayasa sistem untuk menetapkan dan mempertahankan konsistensi dari kinerja produk, fungsional, dan atribut fisik dengan persyaratan, desain, dan informasi operasional sepanjang hidupnya
- Tujuan dari manajemen konfigurasi adalah untuk mendukung proses integrasi sistem sehingga semua pengembang dapat mengakses kode dan dokumen dengan cara yang terkontrol, mencari tahu perubahan yang telah dibuat, dan mengkompilasi dan menghubungkan komponen untuk membuat sistem

Manajemen Konfigurasi (Lanjutan)

Tiga aktivitas dasar manajemen konfigurasi:

1. **Version Management**, dukungan diberikan untuk melacak berbagai versi komponen PL, mencakup fasilitas untuk mengkoordinasikan pengembangan oleh beberapa programmer.
2. **Integrasi sistem**, dukungan disediakan untuk membantu pengembang menentukan versi komponen yang digunakan untuk membuat setiap versi sistem.
3. **Pelacakan masalah**, dukungan diberikan untuk memungkinkan *user* melaporkan *bug* dan masalah lain, dan memungkinkan semua pengembang untuk melihat siapa yang bekerja pada masalah ini dan memperbaikinya

C. Host-Target Development

- PL dikembangkan pada satu komputer (*host*), tetapi berjalan pada mesin yang terpisah (*target*).
- *Platform* lebih dari sekedar perangkat keras, termasuk sistem operasi yang terinstal ditambah perangkat lunak pendukung lainnya seperti DBMS, *platform* pengembangan, dan lingkungan pengembangan interaktif.
- *Platform* pengembangan dan eksekusi adalah sama, sehingga memungkinkan untuk mengembangkan PL dan mengujinya di mesin yang sama. Tetapi terkadang sering berbeda sehingga perlu memindahkan PL yang dikembangkan ke *platform* eksekusi untuk menguji atau menjalankan simulator pada mesin untuk pengembangan

Host-Target Development (Lanjutan)

Platform pengembangan PL harus menyediakan berbagai alat untuk mendukung proses RPL, termasuk:

1. Sebuah kompilator terintegrasi dan sistem pengeditan yang dirancang secara sintaks yang memungkinkan untuk membuat, mengedit, dan mengkompilasi kode.
2. Sistem *debug* bahasa.
3. Alat pengeditan grafis, seperti alat untuk mengedit UML.
4. Alat pengujian, seperti JUnit yang dapat menjalankan serangkaian tes secara otomatis pada versi baru program.
5. Alat dukungan proyek yang membantu mengatur kode untuk berbagai proyek pengembangan.

Host-Target Development (Lanjutan)

- Diperlukan keputusan tentang bagaimana PL yang dikembangkan akan digunakan pada *platform* target.
- Pertimbangan dalam membuat keputusan adalah:
 1. Persyaratan perangkat keras dan perangkat lunak dari suatu komponen
 2. Ketersediaan persyaratan sistem
 3. Komunikasi komponen, jika ada tingkat lalu lintas komunikasi yang tinggi antar komponen

2. PEMELIHARAAN (MAINTENANCE)

PEMELIHARAAN (*MAINTENANCE*)

Pemeliharaan PL adalah suatu aktivitas yang sangat luas yang sering digambarkan mencakup semua pekerjaan yang dibuat di suatu sistem setelah PL beroperasi.

Aktivitas meliputi:

- a. Penambahan atau perbaikan program, seperti penambahan fungsi baru, dan perbaikan tampilan.
- b. Perbaikan terhadap kesalahan yang timbul
- c. Penghapusan kemampuan kualitas
- d. Peningkatan pencapaian & memperluas daya guna untuk memenuhi kebutuhan *user* yang semakin bertambah
- d. Menyesuaikan PL untuk memenuhi lingkungan yang berubah

A. Kategori Pemeliharaan PL

- **Korektif** adalah perbaikan program akibat adanya kesalahan
- **Adaptif** adalah penyesuaian dengan lingkungan yang baru, seperti penerapan pada *platform* di lingkungan yang baru, format tampilan printer, dll
- **Perfective** terjadi pada saat pengguna sistem atau *stakeholder* merubah *requirement* dari sistem yang dibangun
- **Preventif** berhubungan dengan prediksi yang akan datang, seperti penggunaan anti virus untuk keamanan data, *back-up* data dan program

Contoh Tugas-Tugas Pemeliharaan

	<p>Corrective Maintenance</p> <ul style="list-style-type: none">• Diagnose and fix logic errors• Replace defective network cabling• Restore proper configuration settings• Debug program code• Update drivers• Install software patch
	<p>Adaptive Maintenance</p> <ul style="list-style-type: none">• Add online capability• Create new reports• Add new data entry field to input screen• Install links to Web site• Create employee portal
	<p>Perfective Maintenance</p> <ul style="list-style-type: none">• Install additional memory• Write macros to handle repetitive tasks• Compress system files• Optimize user desktop settings• Develop library for code reuse• Install more powerful network server
	<p>Preventive Maintenance</p> <ul style="list-style-type: none">• Install new antivirus software• Develop standard backup schedule• Implement regular defragmentation process• Analyze problem report for patterns• Tighten all cable connections

a. Pemeliharaan Korektif (*Corrective Maintenance*)

- Pekerjaan pemeliharaan sistem harus dilakukan terlebih dahulu di lingkungan pengujian, dan kemudian dimigrasikan ke operasional sistem.
- Situasi terburuk adalah kegagalan sistem. Jika keadaan darurat terjadi, tim pemeliharaan mencoba memperbaiki masalah dengan segera, sementara permintaan sistem tertulis disiapkan dan ditambahkan ke *log* pemeliharaan.
- Ketika sistem beroperasi kembali, tim pemeliharaan menentukan penyebabnya, menganalisa masalah, dan mendesain solusi permanen. Kemudian memperbarui file data, menguji sistem secara menyeluruh, dan menyiapkan dokumentasi lengkap.

b. Pemeliharaan Adaptif (*Adaptive Maintenance*)

- Pemeliharaan adaptif menambahkan peningkatan pada operasional sistem dan membuat sistem lebih mudah digunakan berupa peningkatan fitur baru/perubahan.
- Misal: layanan baru, teknologi manufaktur baru, atau dukungan untuk operasi berbasis web baru.
- Pemeliharaan adaptif membutuhkan lebih banyak sumber daya departemen IT daripada pemeliharaan korektif.
- Pemeliharaan adaptif bisa lebih sulit daripada pengembangan sistem baru karena penyempurnaan harus bekerja dalam batasan sistem yang ada/baru.

c. Pemeliharaan Perfektif *(Perfective Maintenance)*

- Melibatkan perubahan operasional sistem agar lebih efisien, dapat diandalkan, dan dapat dipelihara.
- Permintaan untuk pemeliharaan korektif dan adaptif biasanya berasal dari pengguna, sedangkan departemen IT biasanya memulai pemeliharaan perfektif.
- Pemeliharaan perfektif dapat meningkatkan keandalan sistem. Misalnya, masalah *input* dapat menyebabkan program berhenti secara tidak normal, sehingga diperlukan program yang dapat menangani masalah tsb.
- Semakin banyak program berubah, semakin besar ketidakefisienan dan sulit dipertahankan.

d. Pemeliharaan Preventif (*Preventive Maintenance*)

- Untuk menghindari masalah, pemeliharaan preventif membutuhkan area analisis dimana masalah mungkin terjadi.
- Pemeliharaan preventif menghasilkan peningkatan kepuasan pengguna, *downtime* yang menurun, dan pengurangan biaya.
- Pemeliharaan harus dilayani oleh teknisi yang ahli sehingga kualitas pemeliharaan akan langsung mempengaruhi keberhasilan organisasi.

3. PEMELIHARAAN MANAJEMEN

A. TIM PEMELIHARAAN

1. Systems Administrator

Bertanggung jawab untuk pemeliharaan rutin dan berwenang mengambil tindakan pencegahan untuk menghindari keadaan darurat. Seperti kerusakan server, pemadaman jaringan, insiden keamanan, dan kegagalan perangkat keras.

2. Systems Analyst

Bertugas menyelidiki dan menemukan sumber masalah dengan menggunakan keterampilan analisis dan sintesis. Analisis: memeriksa keseluruhan unsur-unsur individu. Sintesis: mempelajari bagian-bagian untuk memahami keseluruhan sistem.

Pemeliharaan Manajemen (Lanjutan)

3. Programmer

- Programmer aplikasi bekerja pada pengembangan dan pemeliharaan sistem baru.
- Programmer sistem berkonsentrasi pada perangkat lunak dan utilitas sistem
- Programmer basis data fokus pada pembuatan dan dukungan sistem basis data skala besar.

Pemeliharaan Manajemen (Lanjutan)

B. PERMINTAAN PEMELIHARAAN

Pengguna mengirimkan sebagian besar permintaan untuk pemeliharaan korektif dan adaptif ketika sistem tidak berfungsi dengan baik, atau jika mereka menginginkan fitur baru.

1. Determinasi Awal

Ketika pengguna mengajukan permintaan pemeliharaan, administrator membuat penentuan awal, jika permintaan memerlukan perhatian segera, administrator akan mengambil tindakan sekaligus.

Pemeliharaan Manajemen (Lanjutan)

2. Komite Peninjau Sistem

Ketika suatu permintaan melebihi tingkat biaya yang telah ditentukan atau melibatkan perubahan konfigurasi utama, komite peninjau sistem akan menetapkan prioritas, atau menolaknya.

3. Penyelesaian Tugas

Administrator sistem bertanggung jawab untuk mempertimbangkan pengalihan tugas di antara staf IT atau membatasi tugas pemeliharaan kepada individu atau tim tertentu agar tugas dapat diselesaikan dengan baik.

Pemeliharaan Manajemen (Lanjutan)

4. User Notification

Pengguna yang memulai permintaan pemeliharaan mengharapkan tanggapan yang cepat, terutama jika situasi tersebut secara langsung mempengaruhi pekerjaan mereka. Bahkan ketika tindakan korektif tidak dapat terjadi dengan segera, pengguna akan menghargai umpan balik dari administrator sistem dan harus terus diberitahu tentang keputusan atau tindakan yang akan mempengaruhi pengguna.