Rangkuman UTS RPL

Mari nguli 484 halaman slide (yes I counted)

My reaction to that information:

Disusun oleh M Farrel Danendra Rachim

Garis Besar Materi

- Intro to RPL (Definition, categories, layers, software process, system engineering)
- Rekayasa Kebutuhan (Definisi & elemen kebutuhan, karakteristik pengguna, sifat kebutuhan, keterkaitan antar kebutuhan, proses rekayasa kebutuhan)
- Analisis Terstruktur (Data Flow Diagram, Diagram konteks, PSPEC, Data dictionary, State Transition Diagram)
- Penulisan SKPL (Spesifikasi kebutuhan, penulisan SKPL yang baik → kalimat, sifat, kesalahan umum, dll, dokumen kebutuhan)
- Perancangan Terstruktur (Perancangan, arsitektur P/L, structure chart, partisi arsitektur, abstraksi, modularitas, independensi, coupling, kohesi, info hiding, refactoring, perancangan antarmuka)
- Scenario-based modeling (use-case, jalur dasar, extend/include, generalisasi, diagram aktivitas)

Intro to RPL

Software (P/L)

> *Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.*
>
> (IEEE Standard Glossary of Software Engineering Terminology, 1990)

Characteristics:

- Both a product and vehicle for delivering a product
- Engineered, not manufactured
- Deteriorates, doesn't wear out
- Most software is still custom-built, even though industry is moving towards component-based software construction

Software must be...

- Adapted to meet the needs of new computing environment/technology
- Enhanced to implement new business requirements
- Extended sehingga bisa interoperable dengan sistem/database lain
- Re-architected agar bisa berjalan dlm environment komputer

Software (P/L)

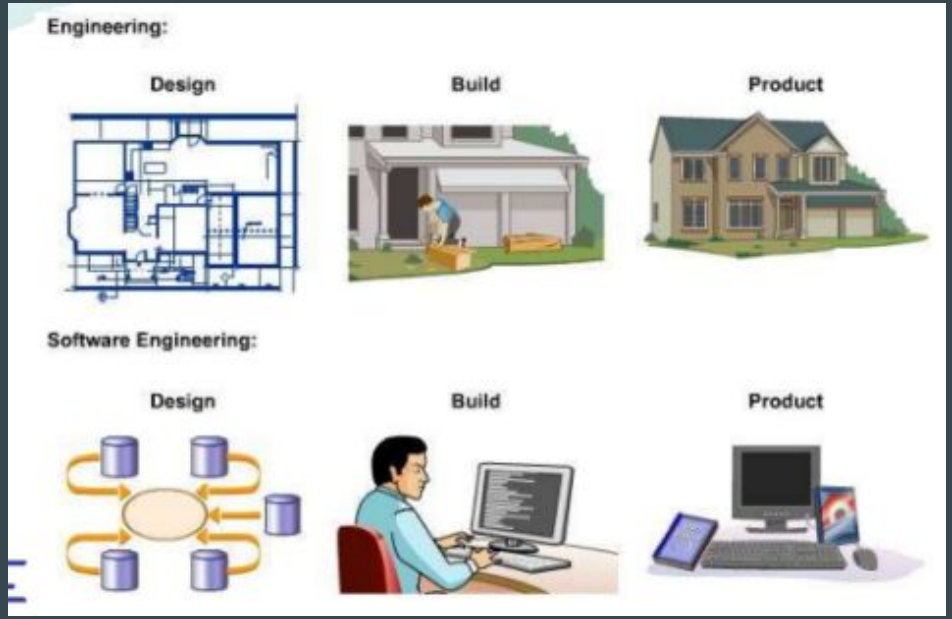
- Software engineers: Build reliable software that doesn't harm people and view computer software as programs, documents, and data required to design & build the system
- Software users: Only concerned with → Are expectations met? Are tasks easier to complete?

Software Application Domains

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line (w/ entertainment) software
- Web-applications
- Mobile-based applications
- AI software

Software Engineering (S/E)

- Establishing engineering principles to get reliable & efficient software in an economical manner
- Pendekatan sistematis, disiplin, quantifiable dalam development, operation, maintenance software
- Encompasses process, management techniques, technical methods, use of tools



Software Categories

1. Web-based systems and applications

- Augmentation of HTML by development tools (XML, Java) causes computing capability & informational content
- Web 3.0 → Semantic databases that provide new functionality that requires web linking, flexible data rep, external access APIs

2. Mobile applications

- Didesain secara spesifik agar terletak di platform mobile
- User interface dengan mekanisme interaksi unik yg dimiliki platform mobile
- Interoperabilitas dengan web-based resources

Software Categories

3. Cloud Computing

4. Product Line Software

- Kumpulan sistem software intensif yang memiliki beberapa fitur sama untuk memenuhi kebutuhan segmen pasar spesifik dan dikembangkan dari aset inti yang sama dan telah ditentukan
- Termasuk: Requirements, architecture, design patterns, reusable components, test cases, etc.

Lapisan RPL

- Process: Foundation for S/E. Men-define framework yang harus di-*establish* agar delivery teknologi efektif
- S/E methods: Pendekatan cara membangun software (technical)
  - Model descriptions/system models: Describes graphical models to be produced
  - Notations
  - Rules: Constraints to system models
  - Recommendations: On good design practice
  - Process guidance: What activities to follow?
- S/E tools: Dukungan auto/semi-auto untuk proses & method

Tiap lapisan memiliki ketergantungan antar-lapisan, tidak bisa berdiri sendiri

Lapisan RPL

- Tools: CASE Tool (Rational Rose), IDE (VisualStudio, Eclipse, NetBeans), versi software (CVS, SVN, Github)
- Metode: Pengumpulan kebutuhan pengguna (goal oriented, viewpoints), analisis & perancangan (terstruktur/OO), pengujian (black/white box)
- Proses: Waterfall/incremental/spiral model, agile development, rapid app development
- Quality focus: Six Sigma, Total Quality Management, CMM, ISO/IEC 9126

Umbrella Activities

- Software project tracking & control: Assess progress against project plan, take action to maintain schedule
- Risk management: Assess risks that may affect outcome of project/quality of product
- Software quality assurance: Conducts activities to ensure software quality
- Technical reviews: Assess products to uncover & remove errors
- Measurements: Defines process, project, product measures to meet stakeholders' needs
- Software configuration management: Manages effects of change
- Reusability management: Criteria for product reuse, establishes mechanisms to achieve reusable components
- Work product preparation & production: Activities to create work products → models, logs, documents, forms, lists

Software Process

Generic Software Process Framework:

- Communication: System analyst vs user/programmer
- Planning: Cost, time, human resources
- Modeling: Structured/OO approach
- Construction: Coding, testing
- Deployment: Software delivery to customer

Essence of S/E practice

- Understand the problem (analysis/communication)
- Plan a solution (modeling/design)
- Carry out the plan (coding)
- Examine result (testing, quality assurance)

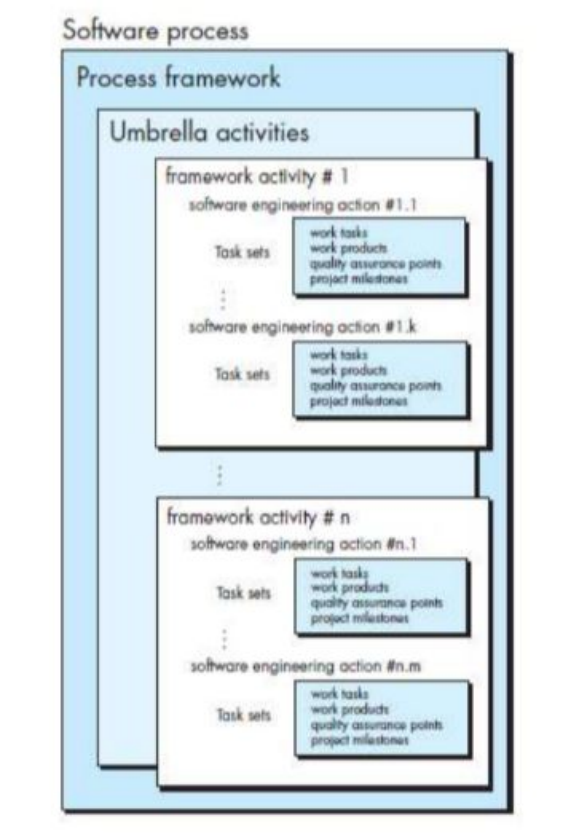
Process adaptation

S/E process should be agile and adaptable to the problem, project, team, organizational culture. Proses di sebuah projek mungkin beda jauh dengan proses di projek lain.

Software Process

Software Practice Core Principles

- Reason it all exist : Provide value to users
- Keep it simple stupid (KISS)
- Maintain the vision
- We produce, others will consume: Others understand what you've done to carry out tasks
- Open to the future changes
- Plan for reuse: Reduces cost & increases value of reusable components and systems
- Think first

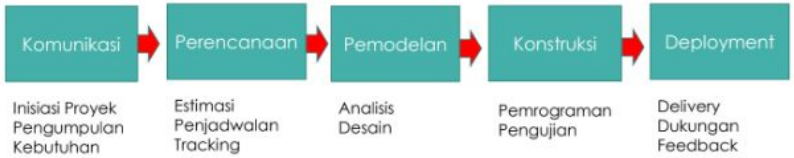


Software Development Life Cycle

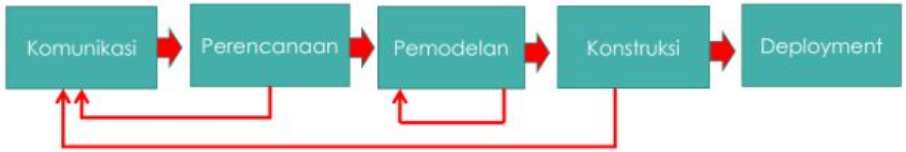
1. Analisis: Dasar dari perancangan. Analisis dilakukan terhadap semua kebutuhan perangkat lunak dan ditetapkan sebagai Spesifikasi Kebutuhan Perangkat Lunak (SKPL). Terjadi masukan dari customer dan persyaratan melakukan quality assurance
2. Perancangan: Menghasilkan sistem berkualitas tinggi. Prinsip dari perancangan adalah untuk memandu pengembang apa pekerjaan yang harus dilakukannya.
3. Implementasi: Produk dibangun dalam bentuk kode sesuai spesifikasi dan rancangan
4. Pengujian: Cacat produk dilaporkan, dilacak, diperbaiki dan diuji ulang, sampai produk mencapai standar kualitas yang ditentukan dalam spesifikasi
5. Perawatan/Pemeliharaan: Produk akan dirilis secara resmi di pasar yang sesuai. Produk dirilis kembali dengan penambahan yang disarankan pasar

Process Flow

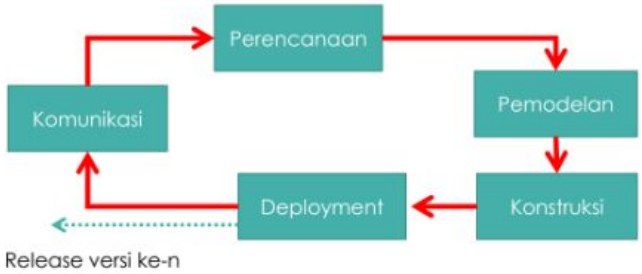
Alur Proses Linear ***(Linear Process Flow)***



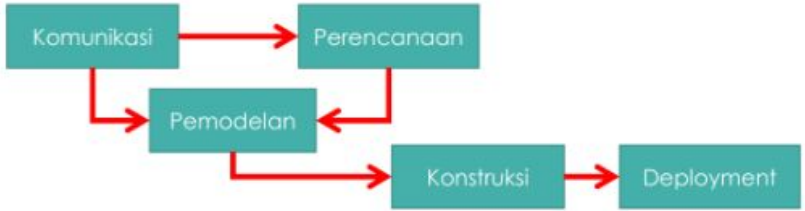
Alur Proses Iteratif ***(Iterative Process Flow)***



Alur Proses Berevolusi ***(Evolutionary process flow)***



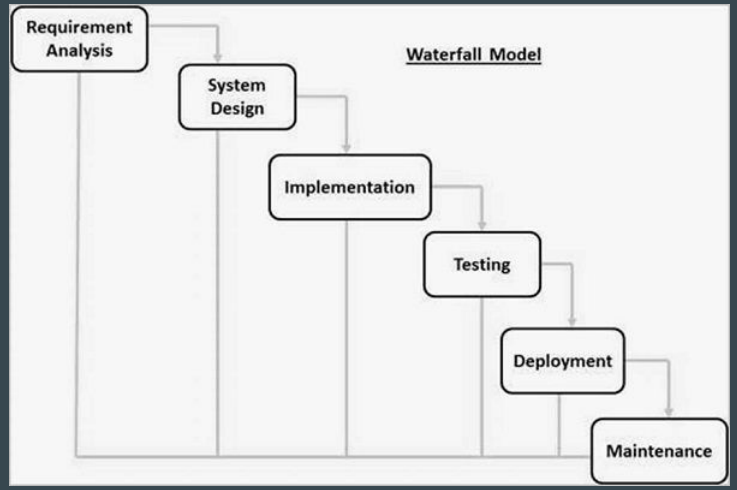
Alur Proses Paralel ***(Parallel process flow)***



Prescriptive Process Models

Waterfall model:

- Tiap fase harus diselesaikan sebelum lanjut ke fase berikutnya, tidak ada overlap antarphase
- Digunakan ketika suatu produk telah terdefinisi dan terdokumentasi secara rinci dan domain-domainnya telah diketahui
- Cocok untuk sistem terdefinisi baik atau sistem yg mengutamakan keselamatan (auto-pilot)
- Mudah dicari kesalahan saat awal perancangan, namun sulit jika diinginkan revisi ke tahap sebelumnya

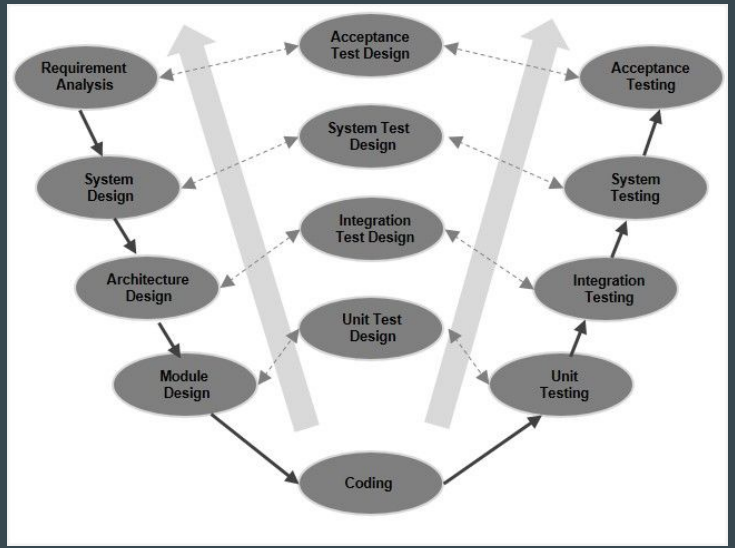


Source: https://www.tutorialspoint.com/sdlc/index.htm

Prescriptive Process Models

V-Model:

- Dalam tiap fase di daur pengembangan, akan ada fase testing.
- Merupakan ekstensi dari waterfall model.
- Digunakan ketika suatu produk telah terdefinisi dan terdokumentasi secara rinci dan domain-domainnya telah diketahui (mirip seperti waterfall model)
- Mudah dimengerti dan dilakukan konsepnya, namun tidak fleksibel terhadap perubahan

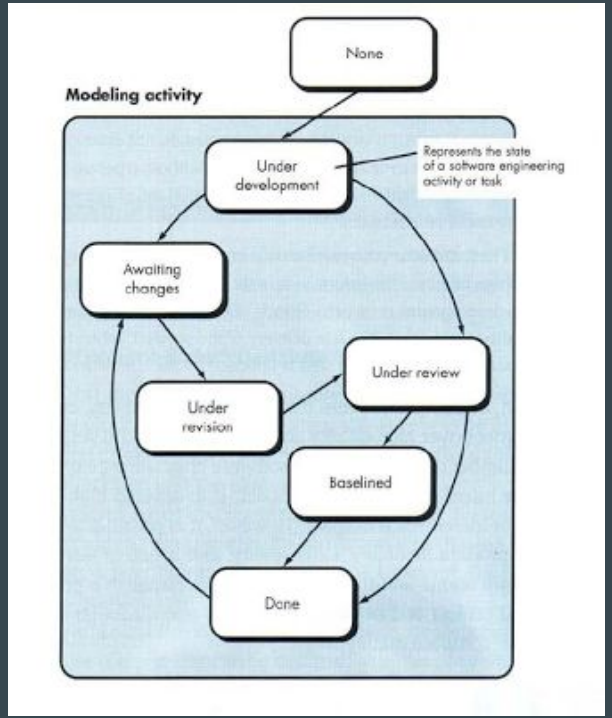


Source: https://www.tutorialspoint.com/sdlc/index.htm

Prescriptive Process Models

Concurrent Model

- Aktivitas kerja dilakukan secara bersamaan, setiap proses kerja memiliki beberapa pemicu kerja dari aktivitas yang saling berhubungan
- Ada dua dimensi:
  - Sistem: Design, Perakitan, Penggunaan
  - Komponen: Design dan Realisasi
- Perancangan terjadi secara besar dan terencana secara matang
- Memungkinkan terjadinya perubahan besar-besaran, maka akan membuat biaya dan waktu yang diperlukan membengkak



Source: https://www.materi-it.com/2014/11/mengenal-concurrent-development-model.html

Prescriptive Process Models

Incremental Model

- Dipecah menjadi beberapa fungsi atau bagian sehingga model pengenbangannya secara bertahap
- Digunakan jika spesifikasi sangat besar dan projek memiliki waktu banyak untuk pengembangan
- Manajemen yang sederhana, resiko kegagalan proyek secara keseluruhan lebih rendah
- Kemungkinan tiap bagian tidak dapat diintegrasikan, total harga tinggi

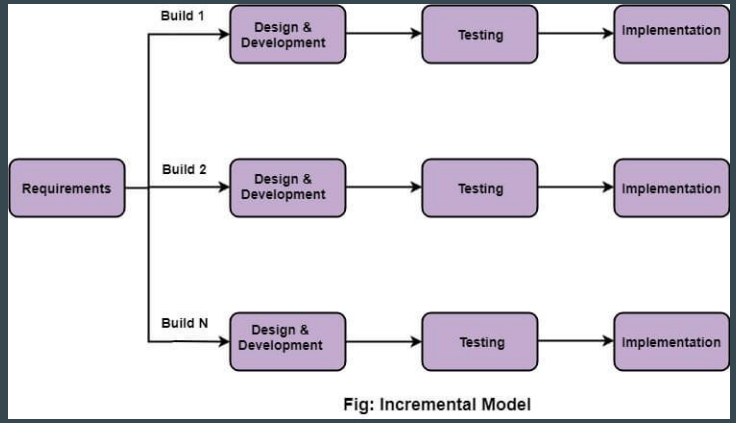


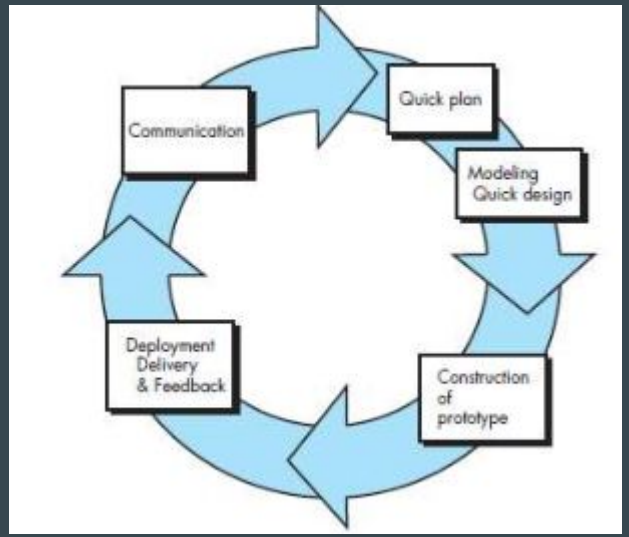
Fig: Incremental Model

Source: https://www.javatpoint.com/software-engineering-incremental-model

Prescriptive Process Models

Evolutionary Prototyping

- Prototipe secara bertahap di-refine berdasarkan feedback pengguna sampai P/L diterima pengguna
- Digunakan saat tidak semua kebutuhan perangkat lunak dipahami ketika modeling sehingga sistem dapat diperbaiki dan dibangun ulang
- Risiko salah implementasinya lebih kecil karena selalu dapat feedback dari pengguna
- Sulit untuk direncanakan timelinenya (tidak tahu pasti berapa kali butuh perulangan).

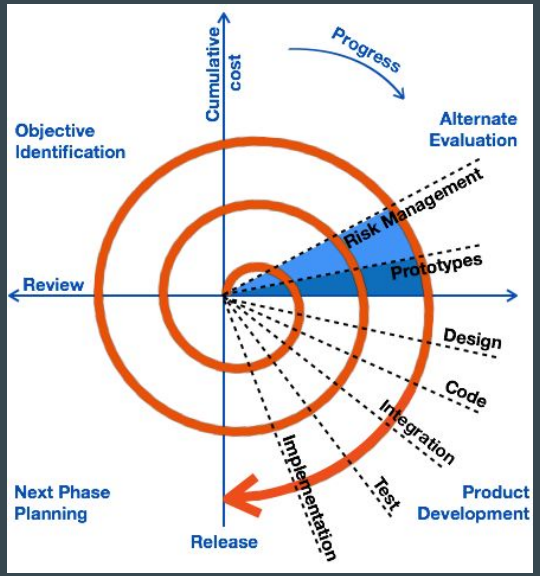


Source: https://www.geeksforgeeks.org/software-engineering-prototyping-model/

Prescriptive Process Models

Spiral Model

- Memudahkan penambahan fungsionalitas/perbaikan setiap kali iterasi spiral
- Digunakan jika risiko besar, spesifikasi tidak jelas dan kompleks, dan perubahan sering terjadi
- Cocok untuk analisis risiko berkelanjutan dengan tujuan risiko gagal akan berkurang setiap iterasi
- Mudah mengubah spesifikasi, penggunaan prototipe yang ekstensif
- Siklus pengembangannya cukup kompleks karena berbasis analisis risiko, durasi proyek tidak bisa diketahui secara cepat



Source: https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm

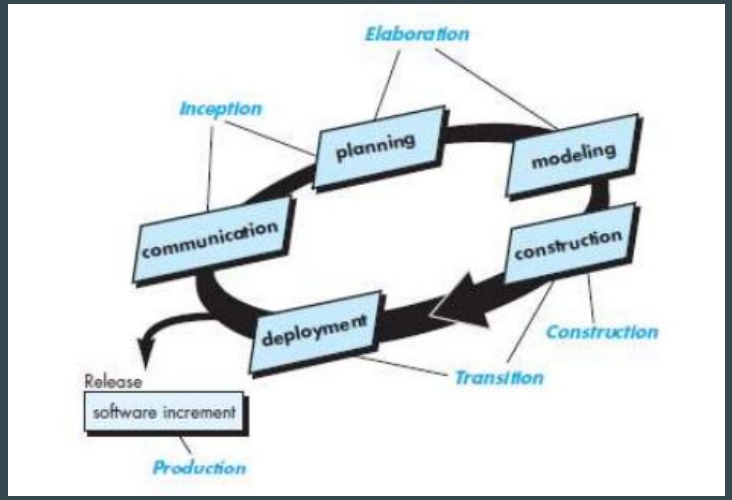
Specialized Process Models

- Component-based development: Apps from prepackaged software components
- Formal methods model
  - Aktivitas yg mengarah ke spek matematikal software komputer
  - Meng-apply notasi math untuk mengembangkan computer-based system
  - Safety-critical software (aircraft, medical)
- Aspect-oriented software development (AOP)
  - Paradigma baru, pendekatan yang fokus kepada aspek2

Unified Process

- Use-case driven, architecture-centric, iterative, incremental
- UML: Unified modeling, bahasa yang mengandung banyak notasi untuk modeling dan pengembangan sistem object-oriented

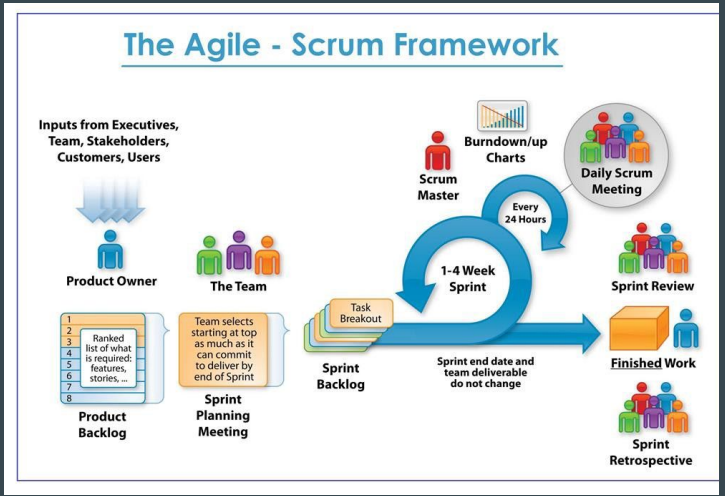
1. Inception: Create use cases, define the scope and potential risk
2. Elaboration: Analyze the problem, design & refine architecture, refine & create requirements
3. Construction: Develop the components and other features of the system
4. Transition: Transit the system from development into production, making the system available to end user for understand and use



Source: https://medium.com/@mojamcpds/software-development-process-models-4168b305303c#:~:text=Linear%20process%20model%20follow%20a,or%20no%20feedback%20or%20refinement.

Agile - Scrum

- Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. Each iteration is considered as a short time "frame".
- SCRUM is an agile development process focused primarily on ways to manage tasks in team-based development conditions.
- Digunakan saat ukuran projek kecil dan diperlukan perubahan yang sering
- Mengurangi waktu development dan mudah menerima perubahan
- Kurangnya dokumen formal sehingga dapat tercipta kebingungan dan kesulitan mengelola produk saat selesai

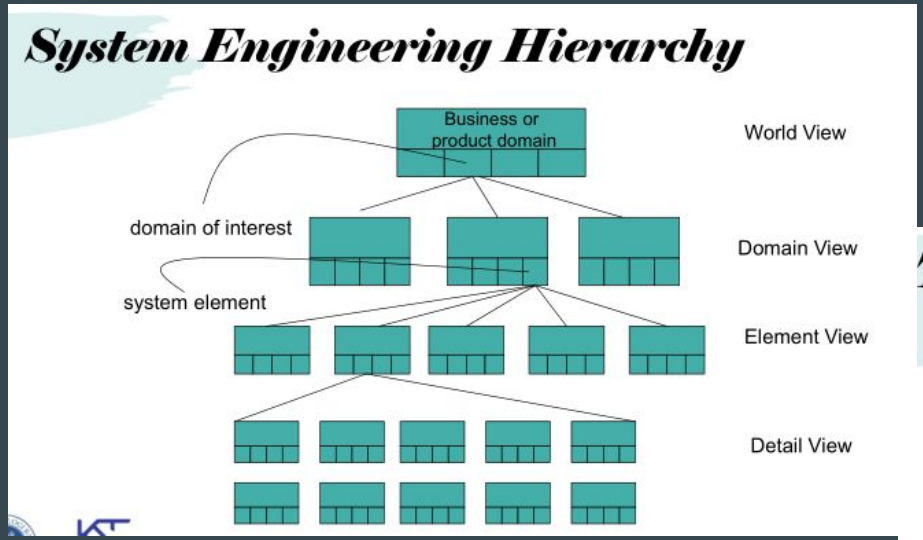


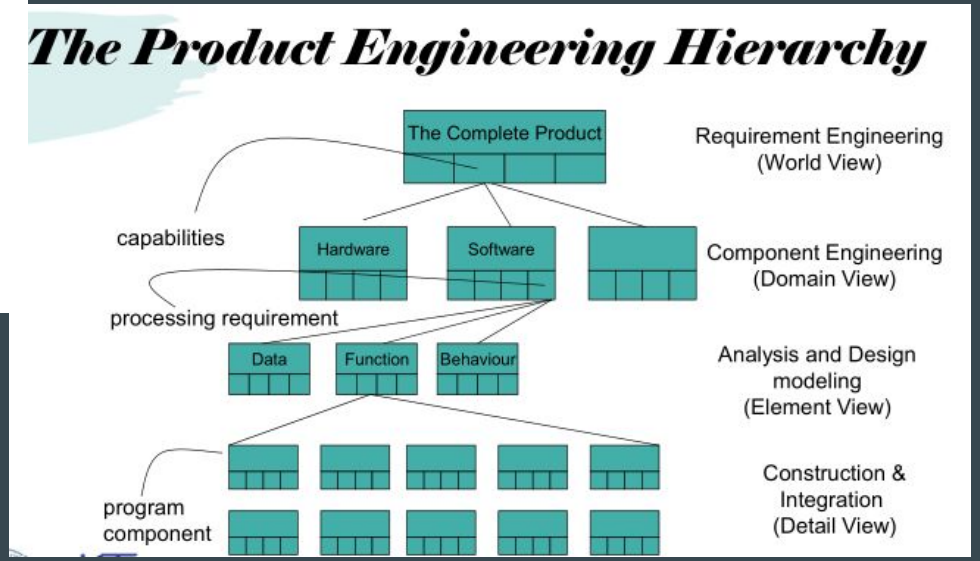
Source: https://www.javatpoint.com/software-engineering-agile-model

Computer-Based Systems

- Definisi: Kumpulan elemen sistem (software, hardware, people, data, documentation, procedures) bervariasi yang memproses informasi untuk meraih tujuan mendapat keuntungan bisnis
- System Engineering Hierarchy
  - World view: Set of domains (D_i), can be a system or system of systems
  - Domain view: Set of elements (E_i), serves role to obtain goals
  - Element view: Elements implemented by specifying technical component (C_i)
  - Detail view
- Product Engineering: Mengubah keinginan pengguna untuk beberapa kemampuan menjadi product yang berfungsi
  - Requirements engineering (world view)
  - Component engineering (domain view)
  - Analysis & Design modeling (element view)
  - Construction & integration (detailed view)

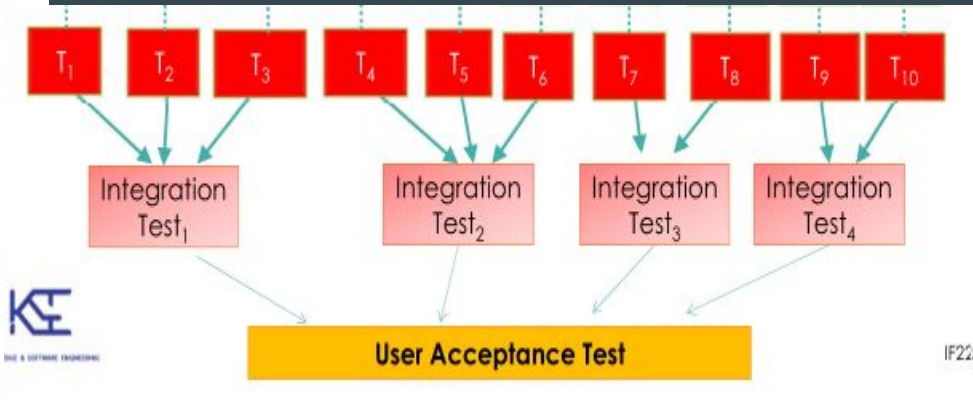
Computer-Based Systems





Aktivitas Pengembangan P/L

Pengumpulan Kebutuhan → Pembuatan Model Rancangan (dari umum hingga rinci) → Coding → Testing kode program → Integrasi unit program dan hasil penggabungannya diuji lagi → User Acceptance Testing di depan calon pengguna



Rekayasa Kebutuhan

Rekayasa Kebutuhan

- Definisi: Proses membentuk layanan yang dibutuhkan pelanggan dari suatu sistem dan batasan sistem akan beroperasi dan dikembangkan.
- Kebutuhan: Deskripsi layanan sistem dan batasan sistem yang dihasilkan selama proses rekayasa kebutuhan
- Requirements Engineering: Membantu pengembang mengerti masalah agar dapat dibuat solusi dalam P/L. Bagian dari proses "Komunikasi" yg berlanjut pada "Pemodelan"
- Manfaat Pernyataan Kebutuhan
  - Dasar membuka/mengikuti tender pekerjaan: Open to interpretation
  - Dasar pembuatan kontrak pekerjaan: Jaminan pengguna dapat P/L sesuai keinginan
- Abstraksi Kebutuhan: Agar kebutuhan perusahaan yang mengembangkan kontrak cukup fleksibel untuk calon peserta tender. Sesudah ada pemenang, kontraktor memberikan rinci dari apa yang akan dikembangkan untuk klien → kontrak

Rekayasa Sistem

- Rekayasa yg melibatkan bidang antar disiplin berbeda, fokus pada perancangan aktivitas pengembangan sistem kompleks
- Computer-based system elements: Software, hardware, people, data, documentation, procedures (SOP, UU, dll)

Kebutuhan pengguna

- Bahasa natural
- Kebutuhan akan dituliskan untuk diyakinkan kembali ke pengguna
- Carry out the plan (coding)
- Examine result (testing, quality assurance)

Untuk contoh ini bisa dilihat di slide rekayasa kebutuhan dari halaman 18 :)

Kebutuhan sistem

- Bahasa teknis
- Dokumen terstruktur: deskripsi rinci fungsi, layanan, batasan operasional sistem
- Definisi hal yang diimplementasikan, mungkin jadi bagian kontrak?
- Dokumentasi kebutuhan sistem

Karakteristik Pengguna

- Berbagai karakteristik pengguna
  - Memiliki suatu hak akses tertentu (Tamu, admin jaringan/sistem)
  - Hanya mengerti satu bagian operasional
  - Hanya mengerti satu sistem (cth: Hanya mengerti windows)
  - Terbiasa berkomunikasi dengan bahasa ibu
  - Berhubungan dengan pelanggan
  - Sibuk
  - Karakter beda-beda (sok tahu, emosi)
- Sistem analis harus sabar dan tidak mudah emosi agar mendapat kebutuhan pengguna
- Pernyataan kebutuhan menjadi:
  - Terlalu umum/terlalu rinci
  - Tidak konsisten
  - Perbedaan aturan/tanggungjawab dalam satu sistem
  - Kebutuhan seperti di luar anggaran, tidak realistis

Kebutuhan Fungsional & Non-Fungsional

- Kebutuhan Fungsional: Layanan yang harus diberikan sistem → respon sistem terhadap suatu input
- Kebutuhan Non-fungsional: Batasan fungsi yang ditawarkan sistem (cth: timing constraint)
  - Kategori:
    - Kebutuhan produk: Batas perilaku (memori, security, usability)
    - Kebutuhan organisasional: Diturunkan dari aturan lingkungan pengguna/pengembang
    - Kebutuhan eksternal: Luar sistem (mengikuti UU)
  - Kebutuhan NF ditulis secara kuantitas agar mudah diuji secara objektif
  - Pengukuran: Kecepatan, ukuran (MB/GB), kemudahan pakai, keandalan, robustness (waktu restart setelah failure, peluang kegagalan), portabilitas

Kebutuhan Harus...

- Mudah dimengerti
- Jangan tidak presisi: Interpretasi berbeda antara pengembang dan pengguna
  - Contoh: “Pengguna harus dapat mencari buku”
- Lengkap (semua deskripsi yang diperlukan sudah terungkapkan)
- Konsisten (tidak ada konflik atau kontradiksi pada deskripsi)

Rekayasa Kebutuhan

- Perlu mengenali:
  - Domain aplikasi yang memiliki fokus kualitas solusi dan kata kunci berbeda-beda (cth: Domain E-commerce, logistik, perbankan, dll)
  - Stakeholders
  - Organisasi pengembang kebutuhan
- Proses secara iteratif:
  - Requirements elicitation & analysis: Interview, dibutuhkan skenario, bekerja dengan pengguna untuk mengerti domain, layanan, & batasan, melibatkan stakeholders
    - Masalah: Stakeholder tidak tahu apa yang mereka inginkan
  - Requirement Validation: After requirement specifications developed, the requirements discussed in this document are validated
  - Requirement Management: Managing changing requirements during the requirements engineering process and system development.

Analisa Kebutuhan P/L Pendekatan Terstruktur

Structured Approach & Functional Decomposition

Structured approach

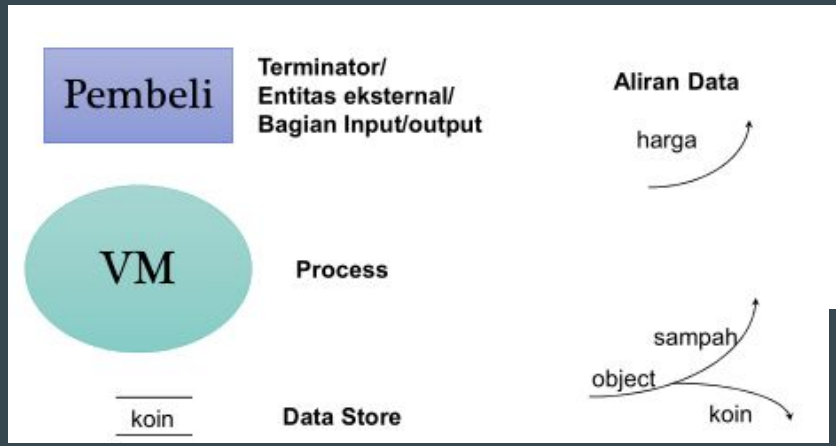
- Program relatif skala menengah ke atas, jumlah individu relatif banyak, butuh waktu tepat dan usaha besar, kompleksitas persoalan tinggi, melibatkan banyak elemen terkait

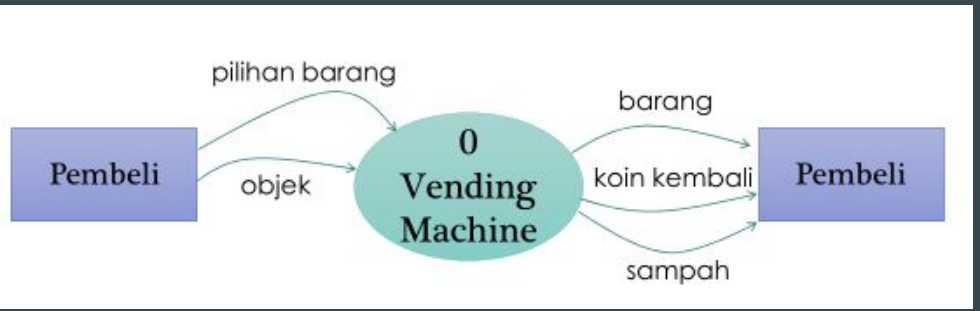
Dekomposisi Fungsional

- Bergantung pada fungsi/aksi PL
- Cara: Tulis dalam satu kalimat apa yang dilakukan PL. Jika PL melakukan beberapa fungsi, tulis setiap fungsi dalam satu baris.
- Contoh:
  - Robot dapat mengambil cangkir
  - Robot dapat menyeduh kopi
  - Robot dapat mengantar kopi

Data Flow Analysis - Diagram Konteks

- Menjelaskan keterhubungan sistem yang dikembangkan dengan sistem lain/external entity
- TEPAT satu lingkaran proses, MINIMAL satu input, satu output, satu entitas eksternal/terminator
- Dibentuk secara iteratif
- Usahakan jumlah input ke proses bedanya maksimal satu dengan output dari proses



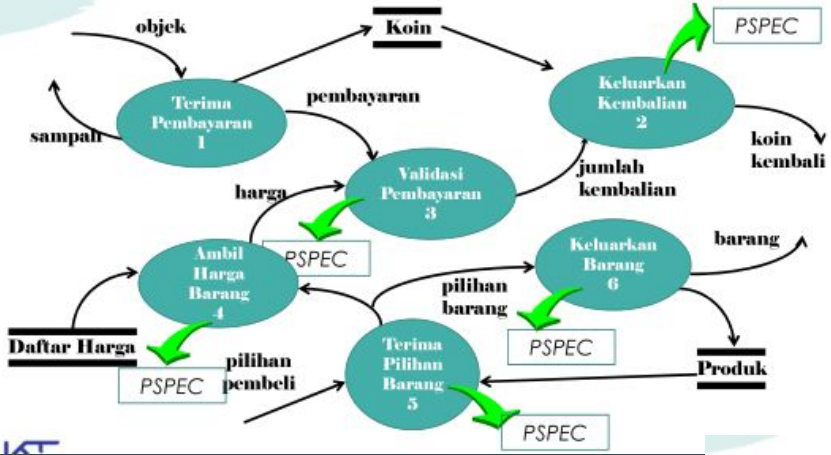


Data Flow Analysis - Data Flow Diagram (DFD)

- Dekomposisi fungsional → Tiap fungsi hanya dideskripsikan sekali saja. Fungsi yang terkait disatukan dalam kelompok, yang tidak terkait dipisahkan
- Tiap lingkaran = proses, tiap proses akan didekomposisi jadi proses lain ke DFD/algoritma
- Aturan 7 ± 2 → 5 - 9 proses dalam satu gambar diagram
- Proses2 adalah elemen yang aktif dalam model
- Proses melakukan transformasi jika tersedia semua informasi keluaran
- Data store : Tempat penyimpanan informasi
- Balancing: Input/output tiap proses dekomposisi harus sesuai induknya
- Leveling: Proses dekomposisi
- Saat proses tidak bisa dipecah lagi, akan menjadi primitif fungsional → diperjelas dengan PSPEC: Sepanjang ½ halaman, menunjukkan hubungan input proses dan output, dapat berupa gambar/persamaan math/bahasa sehari-hari

Data Flow Analysis - Data Flow Diagram (DFD)

Spesifikasi Proses (PSPEC – Process Specification)



PSPEC 3: Validasi Pembayaran

Inputs:

pembayaran	: data in
harga	: data in

Outputs:

jumlah kembali	: data out

Body:

```
If (pembayaran >= harga)
        jumlah kembalian= payment – harga
else
        jumlah kembalian = 0
```

PSPEC 1.1: Validasi Koin

Periksa Objek sesuaikan dengan data dari Parameter Koin.
Jika benar maka terima objek sebagai koin, jika tidak maka keluarkan objek sebagai sampah

PSPEC 1.2: Akumulasi Pembayaran

Tambahkan nilai koin ke data pembayaran.

Data Flow Analysis - Kamus Data

- Berisi sekumpulan nama data dan definisi
- Tiap aliran data harus ada kamusnya
- Nama-nama kelompok harus dipecah sampai paling elementer
- Atribut elemen primitif: Unit, range, akurasi
- Kamus data menjadi bagian basis data

Balancing Kebutuhan:

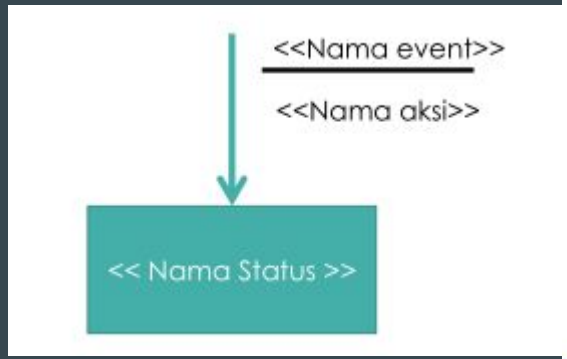
- Tiap DFD harus seimbang dengan induknya
- Tiap PSPEC harus seimbang dgn proses primitif fungsi
- Tiap aliran data & data store harus terdefinisi dan terdekomposisi jadi elemen primitif di kamus data

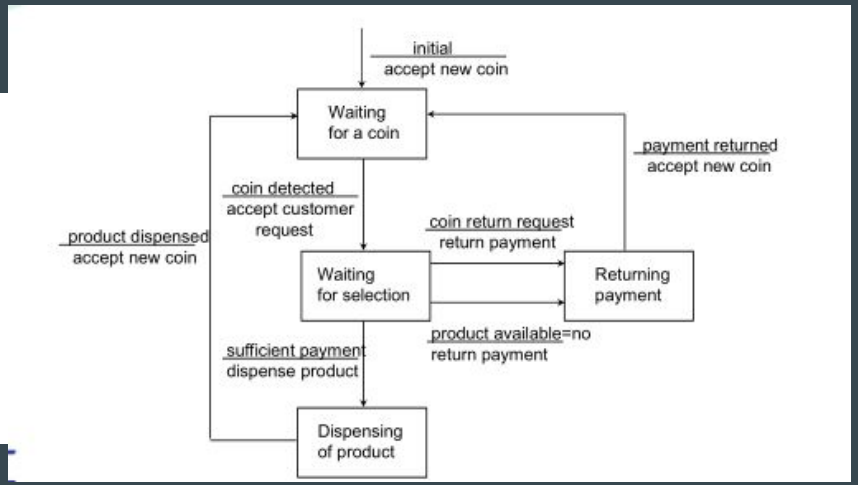
=	terdiri dari
+	digabungkan dengan
{ }	pengulangan
[..|..]	terdiri dari
( )	optional item
" "	literal
* *	memberikan insight tambahan tentang arti

Nama	Deskripsi
Objek	= [koin | sampah]
Barang	= [soda | permen | chips]
koin	= [100an | 200an | 500an | 1000an]
100an	= *uang koin 100 rupiah*

State Transition Diagram (STD)

- Menjelaskan perilaku dinamika sistem:
  - Sistem memberi respon terhadap suatu stimulus
  - Transisi status dari suatu entitas
  - Trigger yang menyebabkan transisi status dari suatu entitas
  - Contoh: Menekan keyboard dan tampil huruf





Inti

- DFD → Proses, aliran data yang berpindah antar proses
- ERD → Hubungan antar entitas
- STD → Urutan perubahan status secara dinamis jika diberi stimulus
- DFD/ERD/STD adalah alat diskusi antara developer dan customer, karena diagram mudah dimengerti customer serta membantu developer merancang P/L, serta bentuk berevolusi
- Jangan takut untuk mulai dari awal.

Penulisan SKPL

Penulisan SKPL yang baik

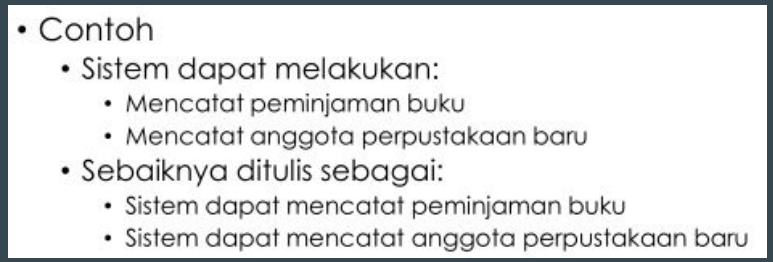
- Jelas/Lengkap → Mencakup semua aspek, hanya satu kebutuhan, mudah dibaca
- Konsisten → Tiap kebutuhan tidak ada yg konflik
- Benar → Akurat dalam identifikasi situasi dan keterbatasannya. Kebutuhan mungkin benar di satu masa, dan tidak benar di masa lain
- Mudah diubah → Pengelompokan dan strukturisasi rapi
- Terurut → Sesuai prioritas. Makin kompleks, makin susah pengurutannya. Salah urut, inefficient development
- Dapat diuji → Dinilai secara kuantitatif, definisikan sistem yang mudah digunakan (cth: “sistem memiliki S/O Windows”, “sistem memiliki fasilitas...”)
- Dapat ditelusuri → Spesifikasi diberi nomor identifikasi unik, konsisten, struktur logis
- Tidak ambigu → Tidak ada interpretasi ganda, jangan pakai struktur bahasa yang jelek (cth: kata “dan”, “atau”, “seminimum mungkin”, “user-friendly”)

Penulisan SKPL yang baik

- Hindari:
  - “Kecuali”, “tetapi”, “jika diperlukan”, “harus”, “dan lain-lain”
  - Kata-kata buzzword (cth: Big Data)
  - Penulisan singkatan (cth: PL)
  - Kata jumlah tidak terukur (cth: “kira2”, “mungkin”, “sebaiknya”)
  - Kata ganda/tidak tegas (cth: “mendukung”
  - Istilah berbeda untuk mengacu kepada sesuatu yang sama
  - Penulisan kebutuhan yang belum jelas
- Struktur: Subjek - predikat - objek
- Gunakan kalimat positif, bukan negatif
- Rule of thumb: <PL/Sistem/Nama> <kata kerja> <objek> dengan konsistensi penggunaan nama pada subjek
- Kalimat atomik: Jangan tulis lebih dari satu kebutuhan dalam satu kalimat

Penulisan SKPL yang baik

- Jangan:
  - Membuat asumsi jelek → biasanya karena tidak ada info cukup dan seputar anggaran
  - Tulis “HOW”, seharusnya “WHAT” → Untuk menghindari pemaksaan suatu solusi desain, “HOW” menyebabkan rancangan tidak fleksibel (harusnya Kebutuhan bukan Operasi)
  - Jangan pakai istilah salah
- Struktur kalimat:
  - PL mampu melakukan <aksi>
  - Aplikasi dapat <kata kerja>
  - Jangan pakai penggunaan daftar
  - Subjek: Selalu sistem, jangan ke hal lain seperti database



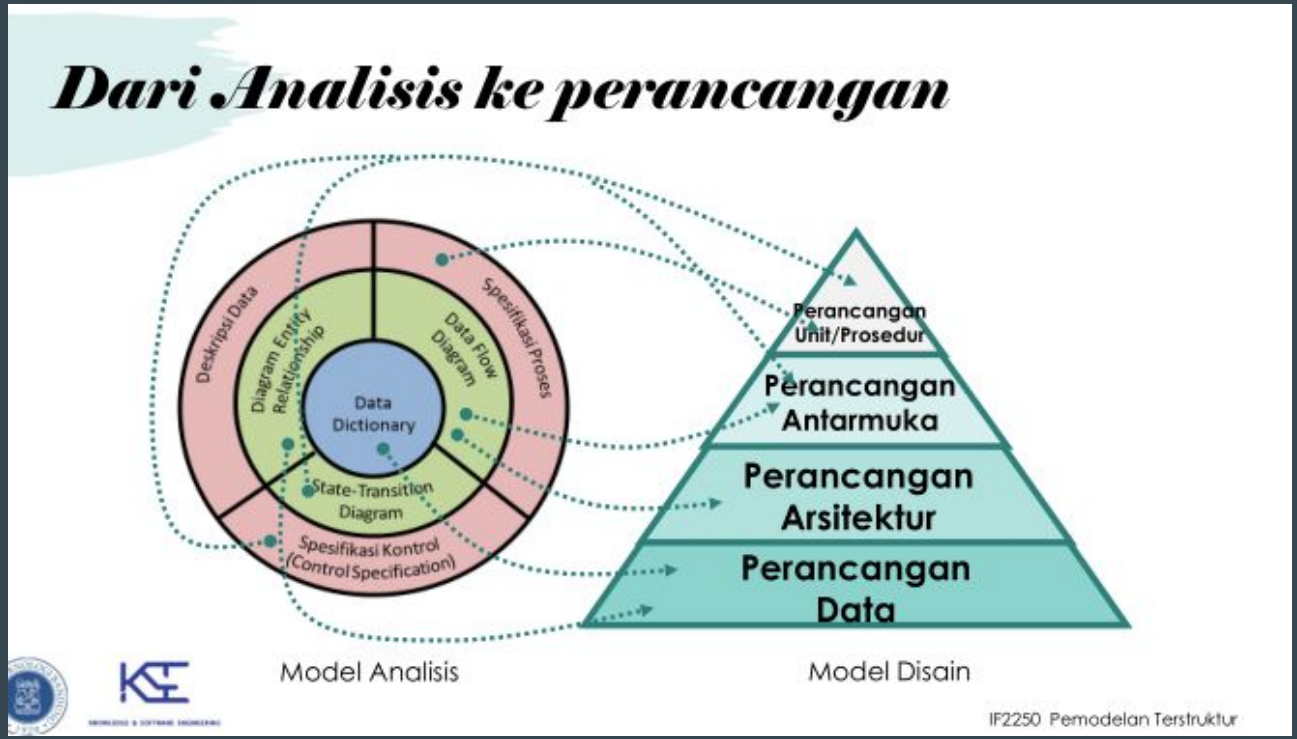
Penulisan SKPL yang baik

- Penulisan referensi: Jelas dituliskan acuannya, serta referensi itu sendiri
  - “Sistem dapat memiliki fungsi khusus yang didefinisikan di [DEF100]”
  - [DEF100] “Data Management Company”, by Ali Budi, 2017
  - “Sistem menggunakan fungsi yang dijelaskan di bab 3, paragraph ke 4”
- Penggunaan tabel/gambar: Diberi nomor unik, daftar judul tabel/gambar dicantum pada daftar isi.
- Aturan umum:
  - <NamaPerangkatLunak atau Sistem>
  - <mampu|dapat|akan>
  - <predikat>
  - <obyek>

Pemodelan Kebutuhan

- Pendekatan:
  - Pendekatan terstruktur: Tradisional/konvensional → konteks-DFD, ER
  - Pendekatan Objek → Diagram UML: Use case, kelas/objek, sekuens
- DK dan use case menggambarkan interaksi sistem dengan entitas terkait
  - DK: Hubungan entitas dan sistem (input-proses-output). Harus dilengkapi dengan proses dan aliran data yang lebih rinci.
  - Use case: Suatu aktor mampu apa saja terhadap sistem. Hanya fokus kepada kebutuhan pada sistem.

Perancangan Terstruktur



Perancangan P/L

- Prinsip perancangan harus dipahami sebelum dilakukan implementasi
- Perancangan memberikan berbagai variasi representasi PL
- Perancangan adalah aktivitas pemodelan terakhir
- Spesifikasi
  - Perancangan data → Transformasi analisis model kamus data dan diagram ER menjadi struktur data
  - Perancangan arsitektur → hubungan elemen struktural utama, model analisis proses dan keterhubungannya (dari DFD)
  - Perancangan antarmuka → Bagaimana elemen software saling terhubung dan berkomunikasi (dari DFD, STD)
  - Perancangan unit/prosedur → transformasi elemen struktural arsitektur PL menjadi deskripsi unit/komponen PL (dari PSPEC, CSPEC, STD)

Perancangan P/L

- Perancangan PL adalah proses iteratif: Mengubah kebutuhan menjadi blueprint
- Saat iterasi perancangan terjadi, hasil kebutuhan mengarah ke perancangan yang makin rinci (tingkat abstraksi lebih rendah)
- Syarat perancangan baik → Semua kebutuhan eksplisit dari model analisis diimplementasikan; dapat dimengerti; jadi gambaran lengkap dari sudut implementasi
- Atribut kualitas untuk perancangan (FURPS)
    - Fungsionalitas: Sekumpulan fungsi & fitur & kemampuan program
    - Usability: Faktor manusia (estetika, konsistensi, dokumentasi)
    - Reliability: Keandalan, frekuensi dan kerugian terjadinya kegagalan
    - Performansi: Kecepatan proses, waktu respon, waktu keseluruhan, efisiensi
    - Supportability - maintainability

Panduan Umum Perancangan P/L

- Perancangan harus melihat dari berbagai sudut pandang (teknologi, kemampuan pengguna, ketersediaan infrastruktur, dll)
- Semua elemen hasil analisis harus muncul sebagai elemen perancangan
- Perancangan tidak dikembangkan dari awal - Reuse
- Meminimisasi “gap” antara software dan dunia nyata
- Perancangan bersifat seragam dan mengandung kesatuan (interaksi konsisten)
- Hasil perancangan distrukturkan dengan baik agar tidak rusak karena ada incomplete data
- Perancangan bukan koding, koding bukan perancangan
- Perancangan dinilai kualitasnya saat proses pembentukan, BUKAN sesudahnya
    - Program mungkin bagus, namun pembuatan rancangan belum tentu bagus sehingga tidak reusable
- Perancangan sebaiknya direview

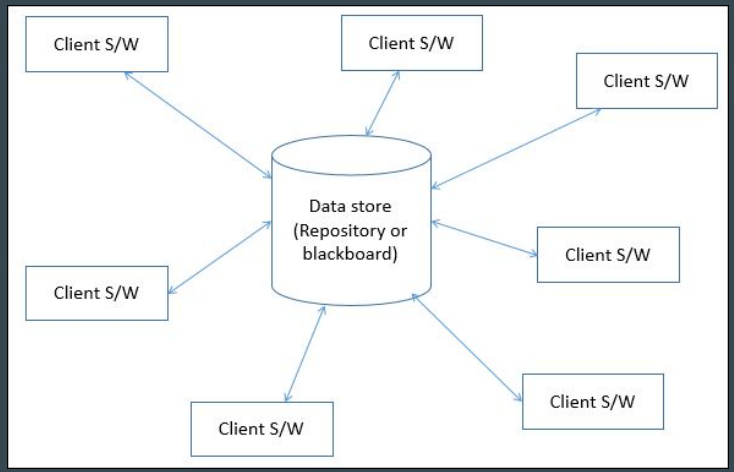
Arsitektur PL

- Properti struktural → Design melibatkan komponen2 yang saling terhubung dan berinteraksi dalam sistem
- Properti fungsi-ekstra → Pengaruh kebutuhan arsitektur terhadap kualitas sistem (performance, capacity, keamanan dll)
- Kumpulan sistem terkait → Pola2 berulang pada sistem yang mirip (kemampuan reuse)
- Karakterisitik setiap bentuk arsitektur:
    - Kumpulan komponen yang melakukan fungsi yang dibutuhkan sistem
    - Kumpulan penghubung antar komponen
    - Batasan/constraint, bagaimana komponen berintegrasi membentuk sistem
    - Model “semantik” : Perencana mengerti properti keseluruhan suatu sistem

Bentuk Arsitektur

Data-centered architecture → Cth: Database

- The data is centralized and accessed frequently by other components, which modify data.
- The client software access a central repository, data can be passed among clients using blackboard mechanism.
- The components access a shared data structure and are relatively independent, in that, they interact only through the data store.
- Menyediakan backup dan reusability, namun lebih rentan terhadap kegagalan dan perubahan data sangat mempengaruhi klien

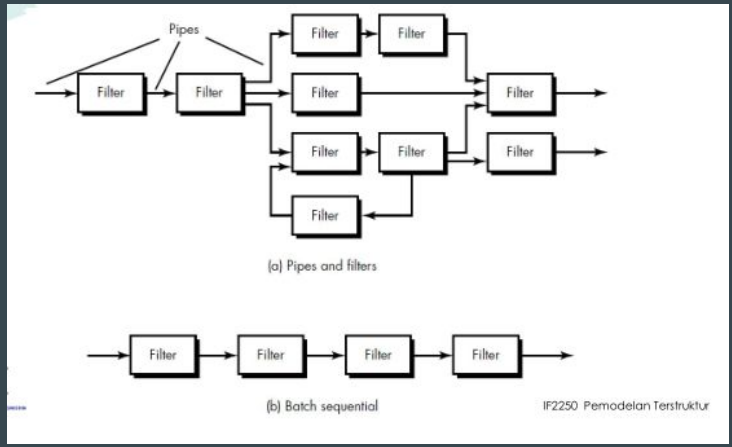


Source: https://www.geeksforgeeks.org/software-engineering-architectural-design/

Bentuk Arsitektur

Data Flow architecture

- Data input yang diubah oleh serangkaian komponen komputasi menjadi data yang bisa di-output.
- Pipes are used to transmit data.
- Each filter will work independently, take data input of a certain form and produces data output to the next filter
- The main objective of this approach is to achieve the qualities of reuse and modifiability.
- Pembagian subsistem lebih sederhana, namun memberikan latensi tinggi dan throughput rendah

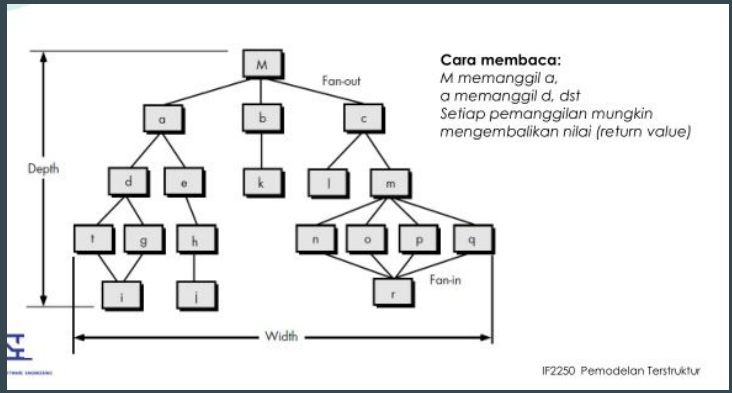


Source: https://www.geeksforgeeks.org/software-engineering-architectural-design/

Bentuk Arsitektur

Call and return architecture

- It is used to create a program that is easy to scale and modify.
- Terdapat macam-macam substyle:
  - Remote procedure call architecture: Komponen ini digunakan dalam arsitektur program utama atau sub program
  - Main program or Subprogram architectures: Struktur program yang klasik dan dapat mendekomposisi sebuah fungsi menjadi hirarki



Source: https://www.geeksforgeeks.org/software-engineering-architectural-design/

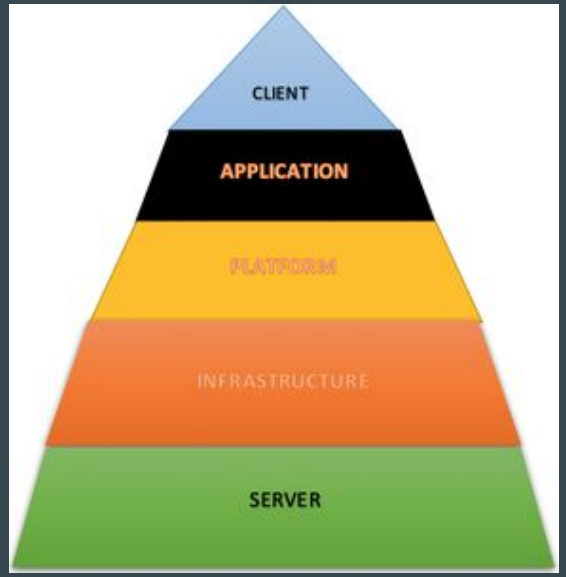
Bentuk Arsitektur

Object-oriented architecture

- The components of a system encapsulate data and the operations that must be applied to manipulate the data (Basically object-oriented concept)

Layered architecture

- Each layer will do some operations that becomes closer to machine instruction set progressively
- Outer layer: Components will receive the user interface operations
- Inner layer: Components will perform the operating system interfacing
- Intermediate layer: utility services, software functions
- Contoh: OSI-ISO Communication system



Source: https://www.geeksforgeeks.org/software-engineering-architectural-design/

Structured Chart (menghubungkan tiap modul)

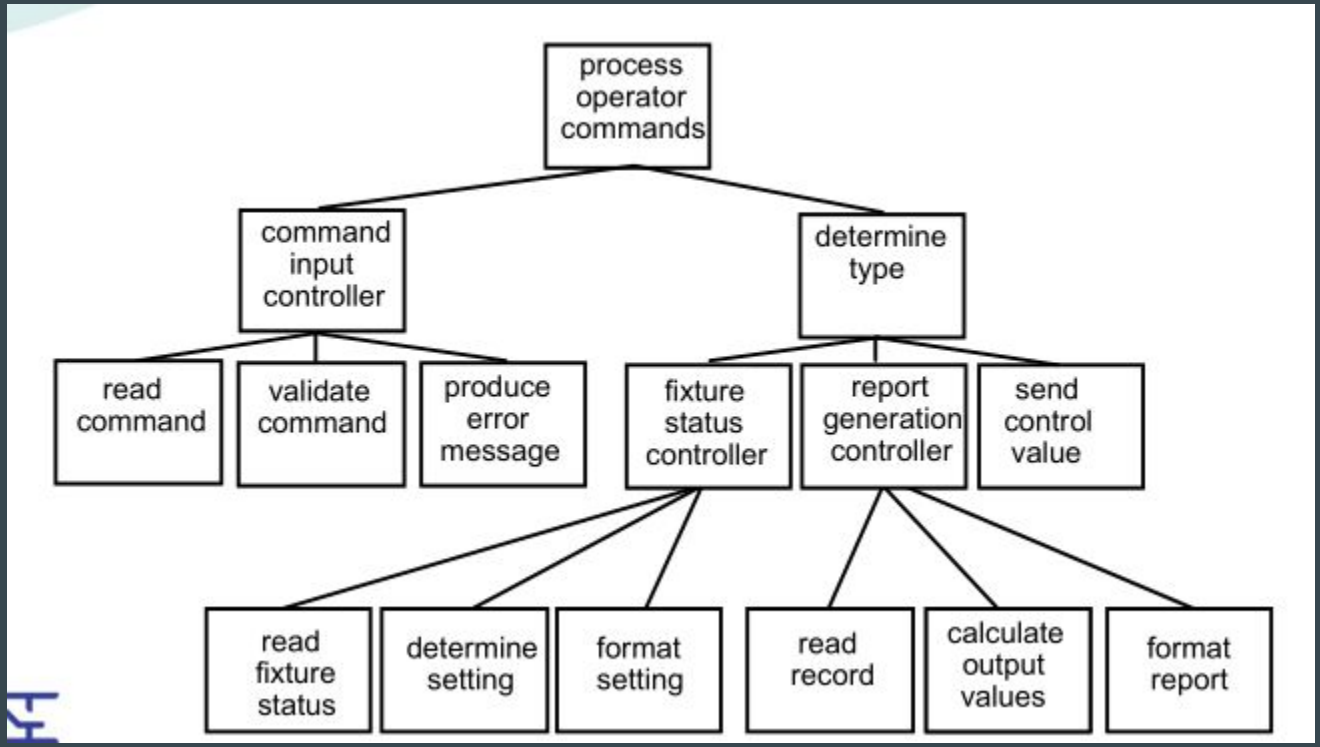
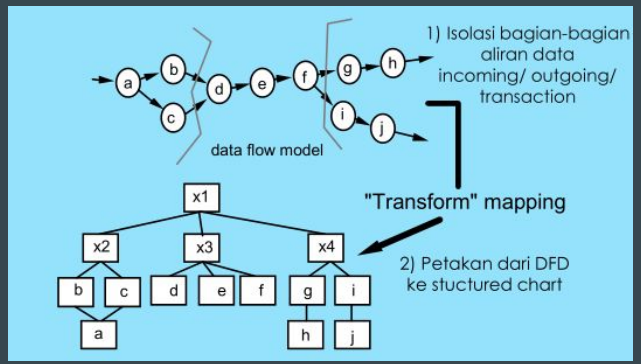
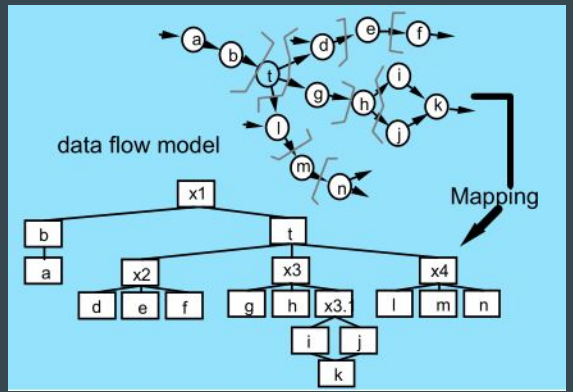
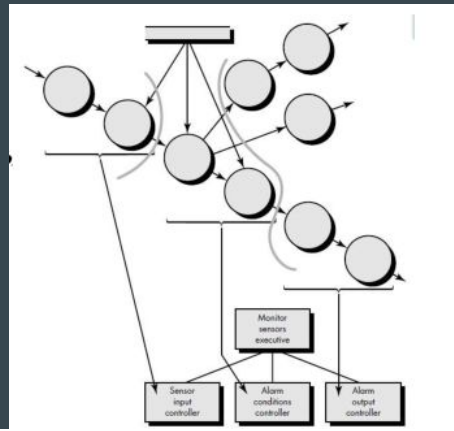


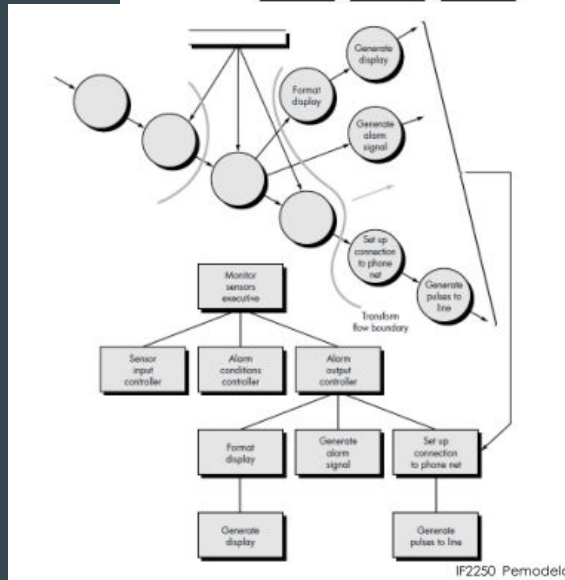
Diagram Terstruktur

- Isolasi incoming flow dan outgoing flow, berikan batas
- Dari batas tadi, petakan DFD, transformasikan menjadi modul yang sesuai
- Tambahkan modul ‘antara’ jika diperlukan
- Perbaiki hasil program struktur dengan memperhatikan modularitas dari hierarki modul



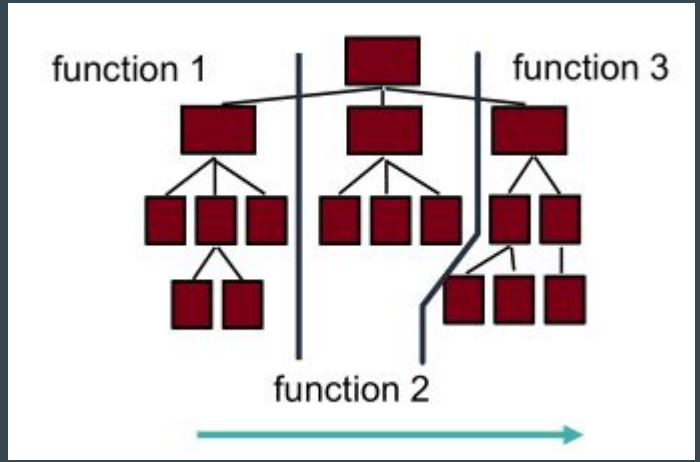


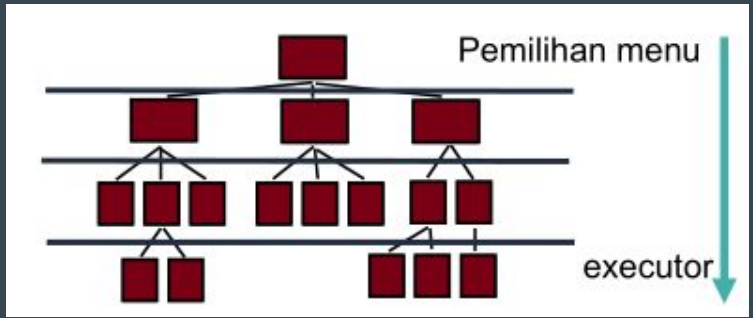




Konsep Perancangan - Partisi Struktural

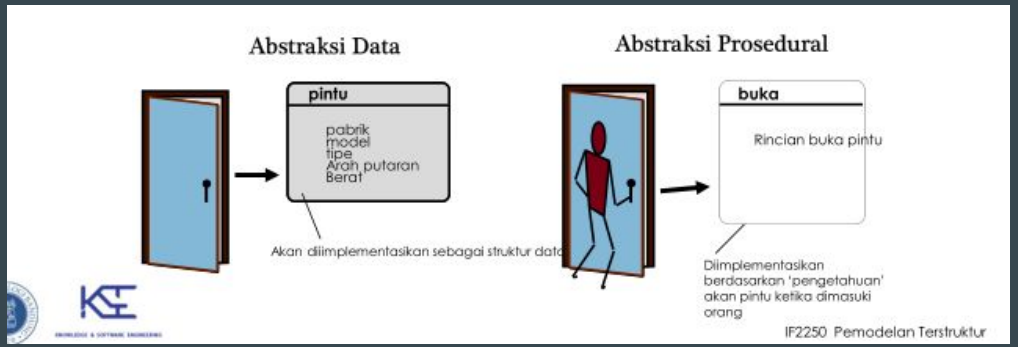
- Arsitektur dipartisi agar software mudah diuji dan dirawat, propagasi kesalahan berkurang, software mudah ditambah kurang modulnya
- Horizontal: Cabang terpisah dari hirarki modul untuk tiap fungsi utama
- Vertikal/Factoring: Bagian atas biasanya berbentuk modul-modul yang sifatnya untuk pengambilan keputusan dan bagian bawah bagian pekerjanya





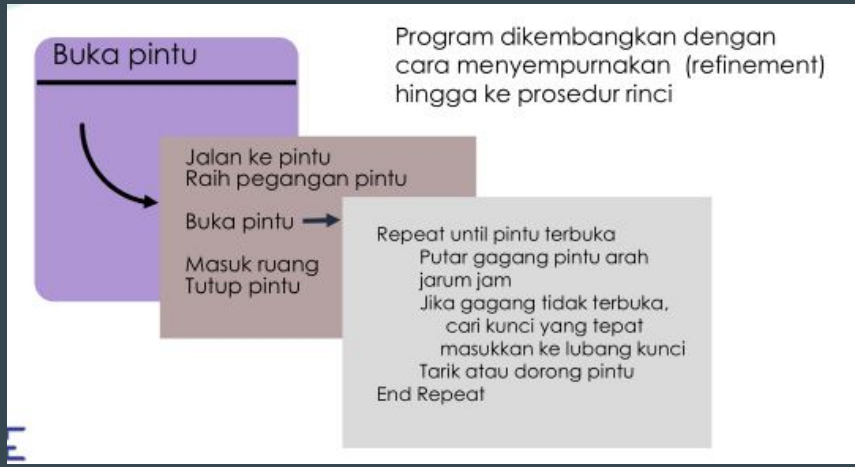
Konsep Perancangan - Abstraksi

- Abstraksi: Suatu cara untuk mengatur kompleksitas pada suatu sistem komputer
- Setiap level dalam suatu sistem komputer memiliki tingkat kompleksitas yang berbeda-beda
- Pada setiap lapisan kompleksitas, level di atas menggunakan abstraksi dari level di bawahnya. Paling tinggi: Solusi dalam lingkungan masalah. Paling rendah: Deskripsi solusi paling rinci
- Contoh level-level abstraksi:
  - Penggunaan email oleh user
  - Pengaturan/administrasi email
  - Pemrograman aplikasi



Konsep Perancangan - Stepwise Refinement

- Stepwise Refinement: Process of breaking down a programming problem into a series of steps..
  - You start with a general set of steps to solve the problem, defining each in turn.
  - Once you have defined each of the steps you then break the problem down into a series of smaller sub-steps.
  - Once this is complete you keep on going until you have described the problem in such a level of detail that you can code a solution to the problem.



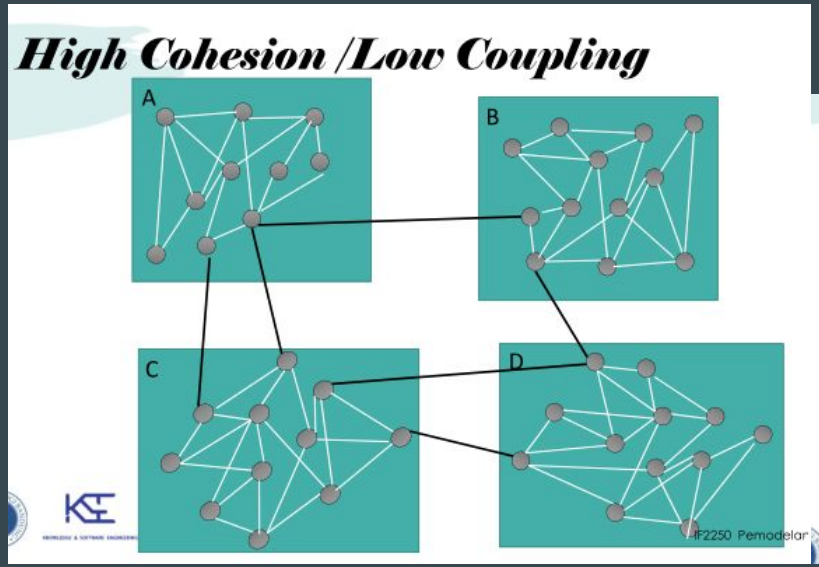
Source: https://learnlearn.uk/alevelcs/stepwise-refinement/

Konsep Perancangan - Modularitas

- Modul: Berbentuk kumpulan prosedur yang saling terkait secara fungsional atau file yang berisi kumpulan prosedur yang terkait secara prosedural
- PL dibagi menjadi modul yang terpisah dengan nama yang baru.
- Modularitas: Memecahkan permasalahan yang besar menjadi bagian-bagian yang kecil, sehingga pengembang dapat menyelesaikan permasalahan yang kecil terlebih dahulu untuk menyelesaikan masalah yang lebih besar.
- Kriteria evaluasi perancangan modular: Decomposability, composability (reuse), understandability (sebagai individu tunggal), continuity (perubahan kecil pada komponen, dampak minimal), protection (efek samping kesalahan dapat diminimumkan)

Konsep Perancangan - Modularitas

- Independensi fungsional
  - Kohesi tinggi (ketergantungan fungsional unit dalam satu modul)
  - Coupling rendah (ketergantungan fungsional antar modul)



Rancangan Modul A
Fungsi Cetak Ke Printer
Fungsi Cetak Ke Layar
Fungsi Cetak ke Projektor

Rancangan Modul B
Fungsi Cetak Ke Printer
Fungsi Cetak Ke Layar
Fungsi Cetak ke Projektor
Fungsi Baca Data Mahasiswa

Kohesi A lebih tinggi dari Kohesi B atau dapat dibaca: **"Rancangan modul A lebih baik daripada Rancangan Modul B"**

Rancangan Modul C
Fungsi Baca NIM Mahasiswa
Fungsi Baca Nama Mahasiswa
Fungsi Baca Alamat Mahasiswa
Fungsi Baca MataKuliah
Fungsi Baca Nama Dosen

Rancangan Modul D
Fungsi Baca NIM Mahasiswa
Fungsi Baca Nama Mahasiswa
Fungsi Baca Alamat Mahasiswa

Coupling C lebih tinggi dari *coupling* D atau dapat dibaca:
"Rancangan modul D lebih baik daripada Rancangan Modul C"

Konsep Perancangan - Modularitas

- Bentuk coupling (dari paling dihindari sampai paling ditoleransi):
  - Mengganti kode komponen lain (hanya bisa assembly)
  - Bercabang ke lokasi kode lain (instruksi Go-to dalam C/Assembly)
  - Akses data komponen lain (kesalahan bisa terjadi karena keterlibatan komponen lain)
  - Global/shared data
  - Pemanggilan prosedur/fungsi dengan switch sebagai parameter
  - Pemanggilan prosedur/fungsi dengan data parameter biasa (seminimal mungkin)
  - Memindahkan data stream antar komponen (output satu prosedur dibaca prosedur lain jadi tidak ada interaksi langsung antar modul)

Konsep Perancangan - Modularitas

- Bentuk cohesion:
  - Coincidental: Elemen metode terbentuk secara kebetulan (tidak terhubung)
  - Logical: Sekumpulan fungsi yang sama secara logis
    - Contoh: Prosedur multi-fungsi
    - Prosedur melakukan beberapa aksi. Jika ingin mengubah salah satu aksi, maka cenderung juga harus memeriksa elemen lain juga (maintenance kompleks)
  - Temporal: Sekelompok aksi yang berhubungan hanya karena dilakukan bersama-sama
    - Semua aksi tidak berhubung fungsional, sehingga harus diinisialisasi secara berurutan. Dalam OOP, inisialisasi bersamaan dengan penciptaan objek.
  - Communicational: Aksi pada data yang sama dikelompokkan bersama
    - Kohesi komunikasi dijelaskan dengan dilakukannya beberapa aksi pada benda
  - Functional: Bagian modul beda dengan task yang terdefinisi secara jelas dikelompokkan bersama, Ini bentuk kohesi paling baik → 1 aksi dan 1 objek yang dikenakan aksi

Konsep Perancangan - Struktur Data & Procedure

Struktur Data

- Representasi hubungan logis antar elemen-elemen data
- Sudah digunakan untuk merepresentasikan banyak masalah: Siswa (nim, nama, alamat, dll)
- Dalam array, linked list, stack, queue, dll
- Mendefinisikan struktur kendali tanpa melihat urutan pemrosesan dan titik pencabangan

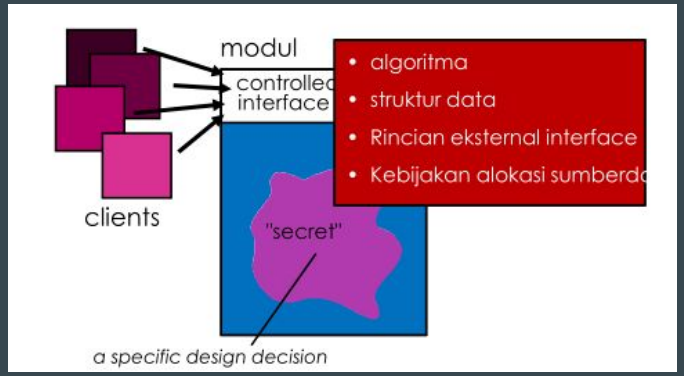
Procedure

- Fokus pada rincian pemrosesan untuk setiap modul
- Memberikan spesifikasi rinci untuk suatu pemrosesan (Kumpulan event, titik percabangan, operasi yang berulang, organisasi data & struktur)

Konsep Perancangan - Information Hiding

Mengapa?

- Mengurangi kemungkinan efek samping
- Menekankan komunikasi melalui manajemen interface
- Mengurangi pemakaian data global
- Mengarah ke ide ‘encapsulation
- Gunakan data lokal
  - Mudah dipelajari dalam lingkup satu modul tunggal
  - Modul mudah dipisahkan untuk di-reuse
- Hindari data global
  - Kita harus pelajari semua modul yang mengakses suatu variabel global



Konsep Perancangan - Information Hiding

- Struktur Umum → ADT/Class
  - Dalam satu modul/komponen berisi struktur data tunggal (menjelaskan modul atau atribut modul), statement pengubah dan pengakses struktur data)
  - Data dalam modul tidak dapat diakses secara langsung (cth: stack)
- Keuntungan
  - Mudah diperbaiki (hanya dalam satu unit saja)
  - Pengembangan independen (tidak tergantung modul lain, komunikasi antar modul lewat interface)
  - Mudah dimengerti untuk perancangan, pengujian, perawatan untuk modul tunggal

Konsep Perancangan - Pola, SoC, Aspek

- Pola perancangan (design pattern) menjelaskan struktur perancangan untuk memecahkan suatu masalah perancangan pada suatu konteks
  - Memungkinkan perancang menentukan apa suatu pola cocok diaplikasikan pada kasus tertentu, dapat di-reuse, & dapat menjadi panduan pengembangan pola mirip tapi beda struktur pola fungsional/struktural
- Separation of Concern: setiap masalah yang kompleks dapat dibagi menjadi lebih kecil agar dapat dipecahkan secara independen dan usaha dan waktu yang minimal.
  - Concern: Kebutuhan, use-case, fitur, struktur data, dll
- Aspek adalah representasi dari suatu concern yang crosscutting. Kebutuhan akan dijadikan representasi perancangan yang lebih modular saat perancangan dimulai.
  - Jika kebutuhan B tidak bisa dipenuhi saat kebutuhan A tidak dilibatkan, A crosscuts B

Konsep Perancangan - Refactoring

- Proses mengubah sistem PL dimana tidak mengubah perilaku eksternal kode, namun memperbaiki struktur internal.
- Diperiksa terhadap: Redundansi (duplikasi), elemen desain yang tidak digunakan, algoritma tidak efisien, strukdat yang jelek konstruksinya
- Hilangkan Goto (di C dan turunannya) → Pemrograman terstruktur, perhatikan struktur kendali seperti loop dan kondisi pencabangan
- Metode panjang akan sulit diubah dan di-reuse → Pecah menjadi lebih dari satu metode
- Perintah switch: Kadang berisi logika untuk instansi beda untuk class sama → Buat subkelas baru, bagian blok dalam case dipindahkan ke metode baru dalam subkelas baru
- Refactoring dilakukan jika ada duplikat, kode terlalu panjang, coupling tinggi atau kohesi rendah, ada loop dalam loop, terlalu banyak parameter, perubahan satu tempat mempengaruhi tempat lain, level abstraksi tidak konsisten, dll

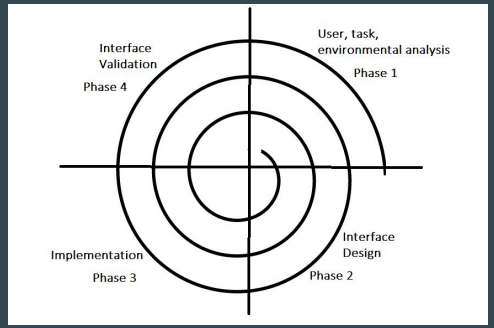
Perancangan Antarmuka

- GOLDEN RULES:
  - Pengguna harus menjadi pengendali
    - Interaksi tidak memaksa pengguna melakukan aksi yang tidak diinginkan
    - Interaksi pengguna dapat di-interupt/dibatalkan
    - Interaksi dibuat fleksibel, khususnya saat kemampuan makin meningkat
    - Hindari pengguna terbiasa mengerti dengan masalah teknis
  - Kurangi hal-hal yang mengharuskan pengguna harus mengingat2
    - Buat desain praktis & shortcut intuitif
    - Informasi ditampilkan secara progresif
  - Buat tampilan konsisten
    - Memperhatikan konteks yang dilakukan pengguna
    - Hati2 dalam perubahan jika model interaksi sebelumnya telah membentuk apa yang diinginkan pengguna

Perancangan Antarmuka

- Model Perancangan Interaksi Pengguna
  - Persepsi sistem: melihat sistem dari sudut pandang pengguna (end-user)
  - Model pengguna: Buat profil (pemahaman, skill, tipe pengguna) end-user
  - System image: “Presentasi” sistem dengan interface lengkap
  - Design model: Representasi PL dalam bentuk data design, arsitektural, interface, prosedural
- Masalah perancang antarmuka
  - Waktu respon sistem
  - Fasilitas panduan pengguna (terintegrasi)
  - Penanganan terjadinya kesalahan (Pesan menjelaskan masalah dengan rinci dan memberikan solusi)
  - Nama istilah perintah (Penamaan menggunakan istilah yang dimengerti pengguna, & penggunaan singkatan konsisten)

Perancangan Antarmuka

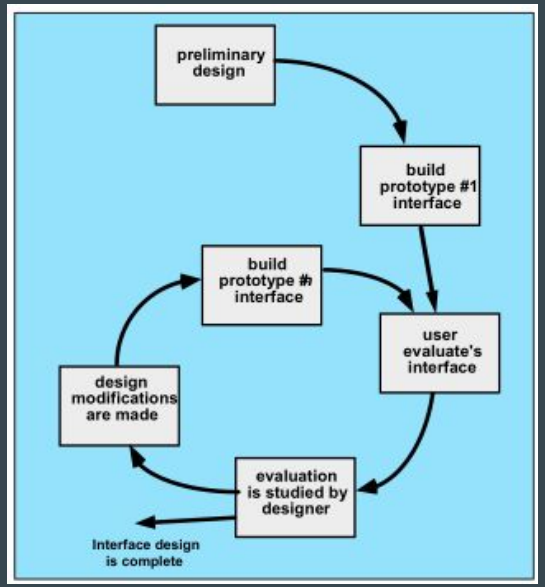


- Proses Perancangan Antarmuka Pengguna
  - User, task, environmental analysis
    - Fokus didasarkan pada profil pengguna yang menggunakan sistem
    - Dari kebutuhan pengguna dilakukan analisis di mana task pengguna berisi objek & aksi untuk meraih tujuannya diidentifikasi secara jelas
    - User environment: Interface berlokasi di mana? Apa hardware interface menyediakan constraint ruang/cahaya? Dll.
  - Interface Design: Mendefinisikan mekanisme pengguna, skenario pengguna, dll
  - Interface Construction: Buat prototipe/model secara iteratif sehingga tercipta User Interface toolkit
  - Interface Validation: Menguji interface, harus memenuhi kebutuhan pengguna

Perancangan Antarmuka

- Aktivitas Perancangan Antarmuka

1. Buat tujuan tiap task
2. Petakan tujuan menjadi sekumpulan aksi
3. Tentukan urutan aksi untuk tiap task/subtask (user-scenario)
4. Tentukan state sistem (interface yang ditampilkan sesuai user-scenario)
5. Definisikan mekanisme kendali (objek/aksi yang ada saat state berubah)
6. Tunjukkan bagaimana mekanisme kendali berefek pada state
7. Beri indikasi bagaimana pengguna mengartikan state melalui interface



Daur evaluasi perancangan antarmuka

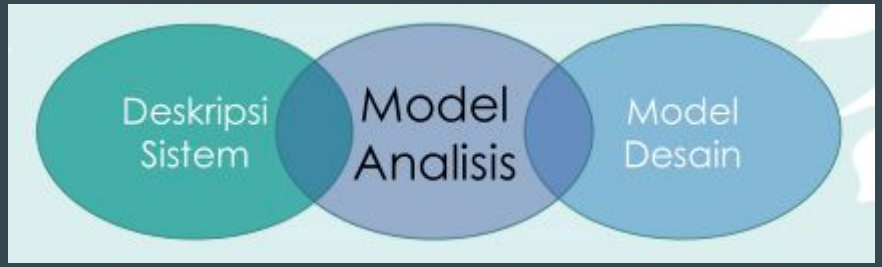
Perancangan Antarmuka

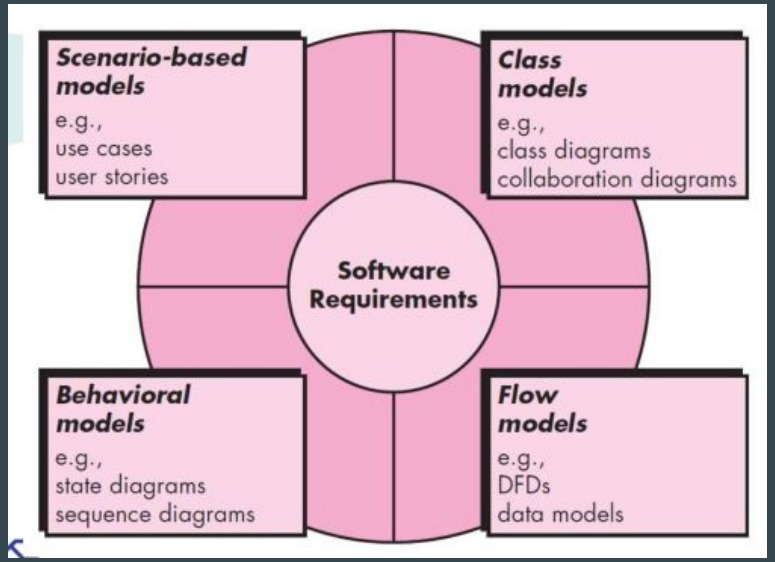
- Kriteria Evaluasi Perancangan Antarmuka
  - Spesifikasi antarmuka yang kompleks memberikan indikasi kompleksitas yang dipelajari pengguna
  - Jumlah task pengguna dan rata-rata aksi per task memberikan indikasi waktu interaksi dan efisiensi sistem
  - Jumlah task, aksi, dan state memberikan indikasi jumlah hal yang diingat pengguna
  - Gaya antarmuka, fasilitas panduan, penanganan error memberikan indikasi kompleksitas sistem dan tingkat acceptance user

Scenario-based Modeling

Model Analisis

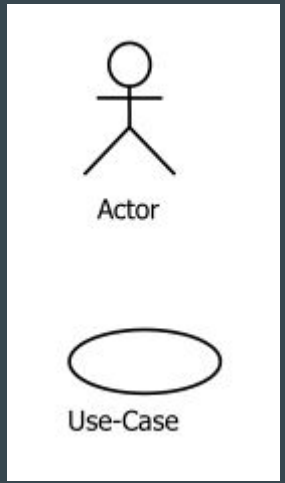
- Jembatan antara deskripsi sistem dan model desain
- Difokuskan pada kebutuhan yang ada dalam domain masalah/bisnis
- Tiap elemen model untuk memahami kebutuhan sistem
- Model sederhana agar dipahami





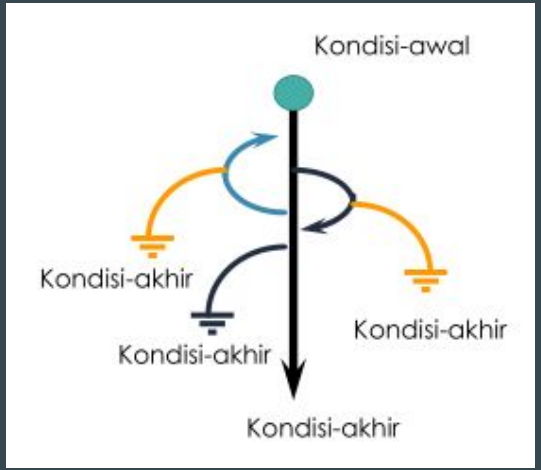
Scenario-Based Modeling - Use Case

- Use-case: Urutan aksi-aksi dalam sistem yang memberikan suatu hasil untuk aktor tertentu
- Use-case menjadi penghubung/komunikasi antara pengguna dan pengembang
- Aktor: Bukan bagian sistem, mungkin aktif atau pasif, bisa mewakilkan manusia/sistem lain
- Use-case: Dialog antara aktor dengan sistem, flow of event yang bermakna bagi sistem
    - Tiap use-case disertai dengan penjelasan:
        - Kondisi awal, bagaimana & kapan use-case dimulai
        - Interaksi sistem dengan aktor
        - Penggunaan objek, penjelasan apa yang dilakukan sistem
        - Instansiasi dari use-case akan memiliki perubahan antar state
        - Flow of event bisa dimulai dari basic path
    - Use-case dapat berupa paket



Aliran Event

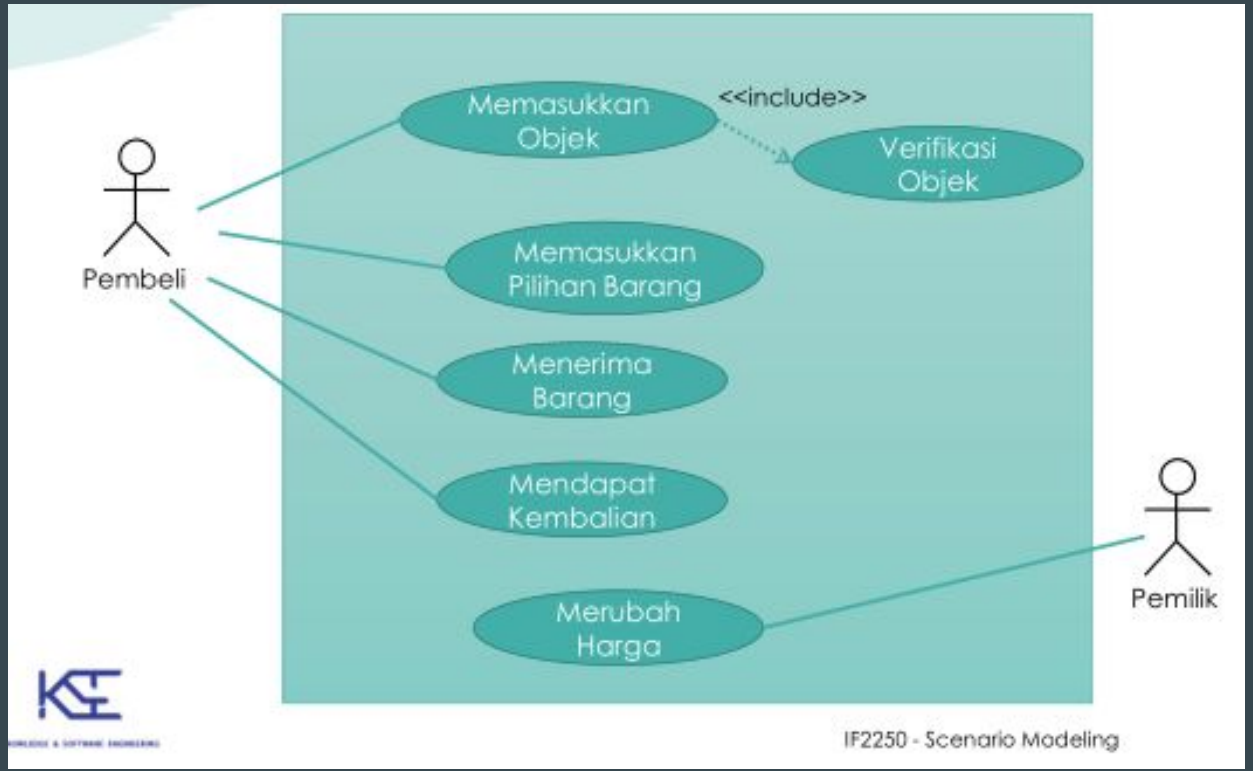
- Aliran normal: Apa yang terjadi umumnya kalau use-case dilakukan
- Aliran alternatif: Penanganan kasus khusus seperti error
- Aliran event use-case bisa dibagi menjadi sub-flow agar flow dasar lebih jelas
- Use-case harus mencakup semua flow: normal, alternatif, pengecualian
- Skenario: Tiap use case memiliki sekumpulan flow-event dengan skenario sebagai bagian instansiasi aliran event → dapat memudahkan test case



Langkah Pemodelan Berbasis Skenario

- Cari aktor → Siapa pengguna yang terkait dengan sistem (pelanggan/operator)
  - Ada aktor menggunakan sistem dan aktor melakukan perawatan
  - Peran aktor harus berbeda → Kebutuhan & tanggung jawab aktor, nama relevan (misal bedakan pengguna sistem dan pelanggan sistem)
- Cari use-case →Aktivitas oleh aktor untuk melakukan kegiatan
  - Lengkap/independen, memberi hasil utk aktor, dlm bentuk cerita, beri nama
  - Review setiap skenario utama
- Gambarkan diagram use case → tiap use case terhubung dengan minimal satu aktor
  - Jika >1 aktor, perjelas aktor siapa yang mentrigger use case pertama kali
- Buat skenario → Dilihat dari sudut pandang aktor
- Buat paket jika perlu → mengelompokkan elemen yang terkait secara semantik
  - Agar use-case lebih terstruktur
  - Menjadi batasan lingkup satu atau beberapa use-case

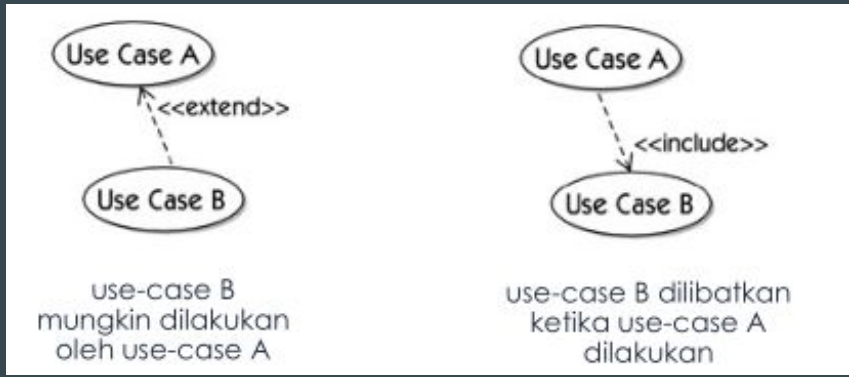
Langkah Pemodelan Berbasis Skenario

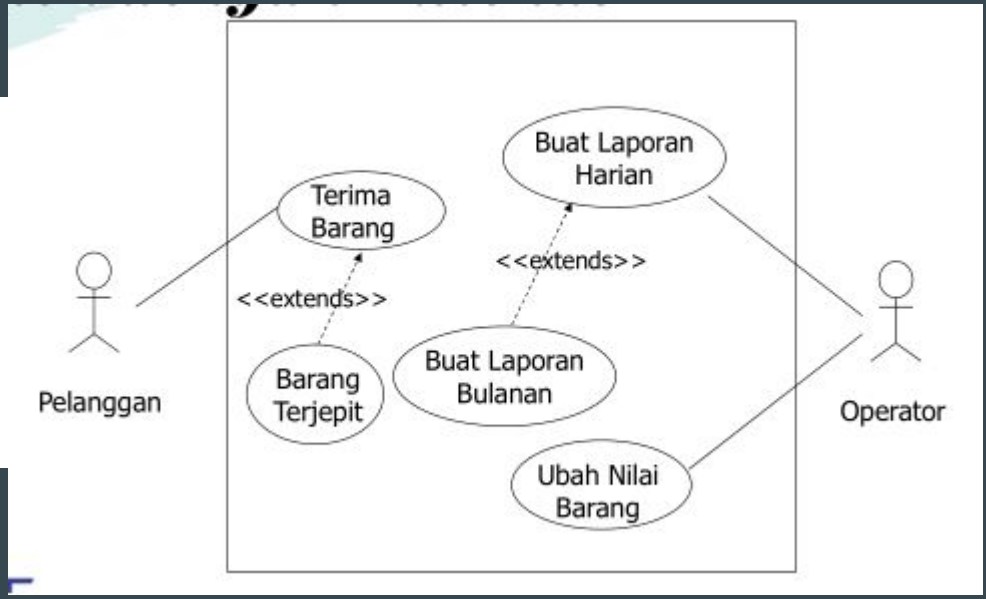


Untuk beragam contohnya bisa dicek di slide 6 mulai dari halaman 30

Ketergantungan antar use-case

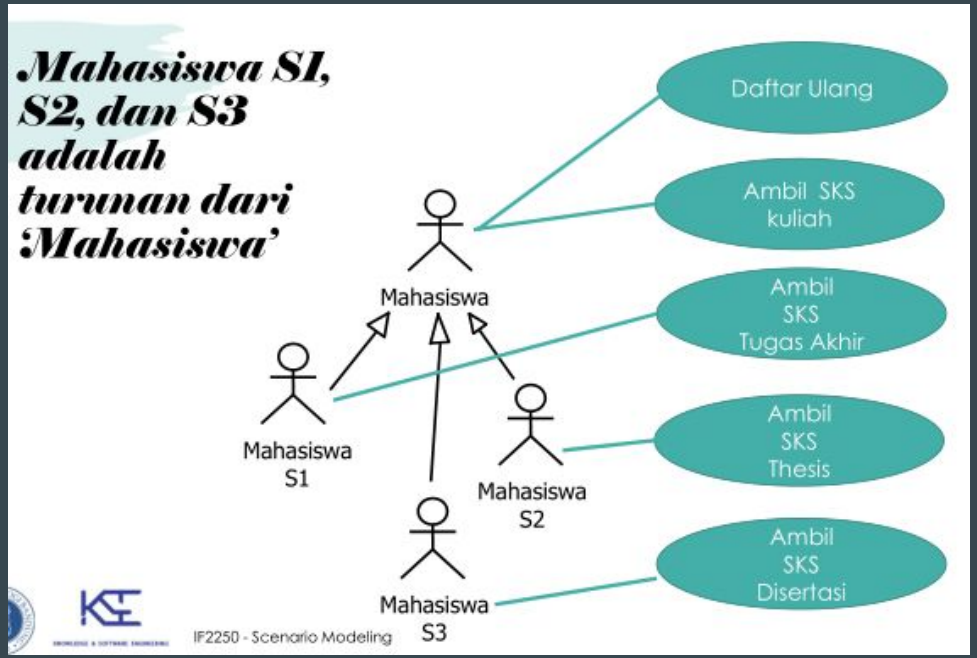
- Extend: Adanya perilaku tambahan (alternatif use-case)
- Include: Hubungan langsung dua use-case (use-case yang harus dilakukan)

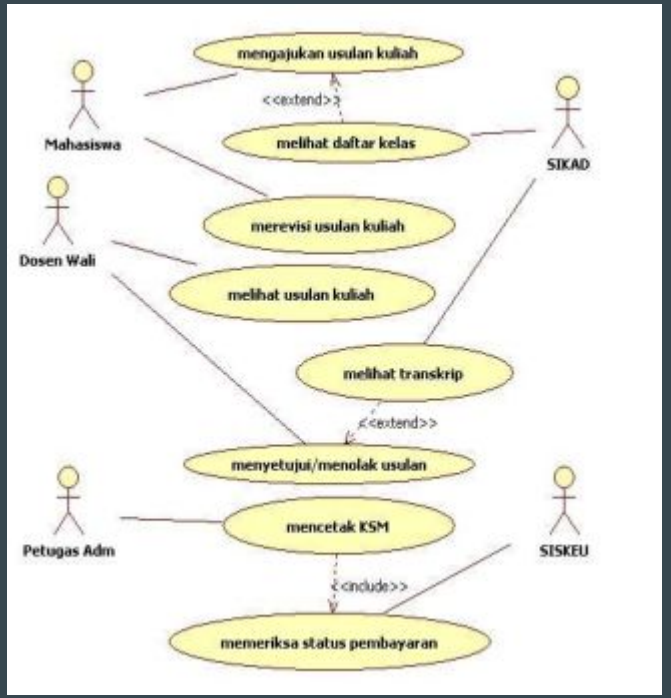




Generalisasi Aktor

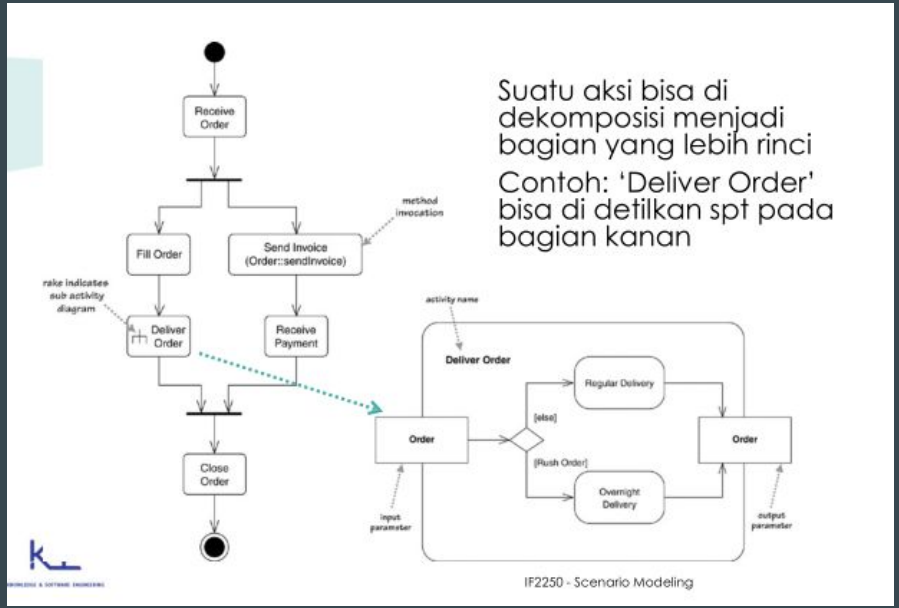
Beberapa aktor dapat memiliki peran yang sama pada suatu use-case (untuk latihan kasus dapat dilihat slide 6 halaman 45)





Deskripsi Model & Activity Diagram

- Jumlah use-case besar dengan berbagai alternatif → Penulisan menjadi tidak praktis
- Gunakan diagram:
  - Statechart: Untuk use-case kompleks, berisi penjelasan state dan transisi use-case
  - Aktivitas: Menjelaskan urutan proses prosedural, bisnis proses, workflow, seperti STD dan flowchart
  - Interaksi: Menjelaskan interaksi antar instansiasi dari aktor dan use-case



Activity Diagram (normal vs swimlane/partisi berdasarkan siapa yang melakukan apa)