Universitas Gunadarma

Rekayasa Perangkat Lunak

Bahan Ajar

Nelly Sofi

2012

KATA PENGANTAR

Puji Syukur kehadirat Tuhan Yang Maha Esa karena atas limpahan rahmat-Nya sehingga dapat menyelesaikan modul Rekayasa Perangkkat Lunak untuk mahasiswa. Modul dilengkapi dengan latihan soal untuk menguji pemahaman siswa terkait dengan materi yang terdapat pada modul. Modul ini membahas pembelajaran terkait dengan Rekayasa Perangkat Lunak dari awal memulai merekayasa perangkat lunak sampai dengan bagaimana melakukan uji coba dan pemeliharaan.

Penulis menyadari masih banyak kekurangan dalam penyusunan modul ini. Oleh karena itu penulis mengharap saran, kritikan, maupun ide konstruktif yang dapat penulis gunakan untuk membuat versi berikutnya yang lebih baik dan lebih lengkap serta lebih ter struktur.

Jakarta, Mei 2012

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Bab 1
Software Engineering

Sofware Engineering (SE) atau yang sering dikenal dengan Rekayasa Perangkat Lunak dimana merupakan ilmu yang mempelajari tehnik pembuatan software yang baik dengan pendekatan tehnik (*Engineering approach*.

Ada beberapa cara / fase dalam engineering approach :
1. Fase Perencanaan
2. Fase Pengembangan
3. Fase Pemeliharaan

Tantangan Software Engineering

CSE memiliki tantangan yang akan dihadapi dalam software engineering adalah mengembangkan hardware komputer yang dapat mengurangi biaya pengolahan dan penyimpanan data serta bagaimana cara untuk mengurangi biaya dan memperbaiki kualitas solusi berbasis komputer

Solusi yang dapat diterapkan untuk tantangan tersbut adalah software merupakan faktor kunci dalam keberhasilan suatu usaha, software dapat membedakan satu perusahaan dari perusahan saingannya, sehingga dalam membuat sebuah software harus memperhatikan saingan-saingan perusahaan tersbut, jangan sampai membuat sebuah software sama dengan software yang ada di perusahaan saingannya.

Software

Software adalah sebuah instruksi atau program komputer yang ketika dieksekusi akan memberi fungsi dan hasil yang diinginkan. Software juga memiliki struktur data, memungkinkan program memanipulasi informasi serta dokumen yang menggambarkan operasi dan penggunaan program.

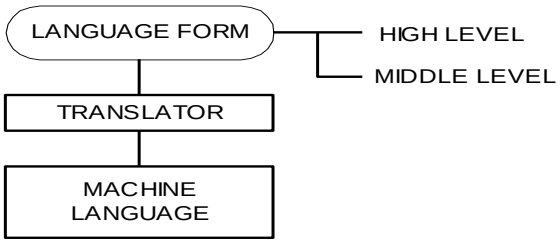
Sifat dan Karakteristik Software

- Software merupakan elemen sistem logik dan bukan elemen sistem fisik seperti hardware
- Elemen itu tidak aus, tetapi bisa rusak.

- Elemen software itu direkayasa atau dikembangkan dan bukan dibuat di pabrik seperti hardware
- Software itu tidak dapat dirakit.

Sofware terdiri dari

- Bentuk Bahasa
- Bentuk Translator
- Bentuk Mesin :



Gambar 1.1. Bentuk Mesin

Contoh-contoh Aplikasi Software

1. Sistem Software
2. Real Time Software

   Perlu dicatat bahwa istilah real time berbeda dari istilah interactive atau time sharing. Sistem real time harus memberikan respons pada waktu yang ditentukan, sedangkan pada sistem interactive atau time sharing respons time biasanya melebihi batas waktu yang ditentukan tanpa merusak hasil.
3. Business Software
4. Engineering and Sciencetific Sofware
5. Emdebed Software
6. PC Software
7. Artificial Intelegent Software

Pengembangan Software

Pengembangan software (software engineering) menurut IEEE adalah : Aplikasi sistematik, disiplin, pendekatan kuantitatif untuk pengembangan, operasi dan pemeliharaan dari software, dengan kata lain software engineering merupakan sebuah metodologi pengembangan perangkat lunak (software) yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal spesifikasi sistem hingga pada tahap pemeliharaan

sistem setelah digunakan dengan tujuan untuk membuat perangkat lunak yang tepat dengan metode yang tepat.

Hal yang perlu dipertimbangkan dalam pengembangan software, yaitu :

1. Produk dan software (terdiri dari program, dokumen dan data)
2. Porses pengembangannya (terdiri dari proses manajemen dan proses terkait)

System Development Life Cycle (SDLC) merupakan beberapa tahap pengembangan pemantauan produk dari perangkat lunak. Contoh dari SDLC antara lain model waterfall, model V, model spiral, prototyping dan lain-lain. Sedangkan proses manajemen dalam pengembangan software lunak terdiri atas manajemen proyek, configuration management dan quality assurance management dan proses teknikal merupakan metode yang diaplikasikan pada tahap tertentu dalam pengembangan software, yang didalamnya termasuk metode analisis, metode desain, metode pemrograman, dan metode testing.

Dalam membuat software yang baik, ada beberapa cara :

1. *Fase Perencanaan (Planning) :*
   a. Rencana software
   b. Analisa kebutuhan software
   c. Analisa *cost banefit* (Salah satu bagian dari studi kelayakan)
2. *Fase Pengembangan (Development) :*
   a. Coding
   b. Testing

   Macam-macam test program :
   i. Unit test (Test per modul)
   ii. Integreated test (Test penggabungan dari modul-modul yang telah diuji)
   iii. Validated test (Diuji dengan data sebenarnya)
   iv. System test (Test dilakukan dengan lingkungan sebenarnya)
   v. Topdown test (Test gabungan dari atas ke bawah)
   vi. Bottom up test (Test gabungan dari bawah ke atas)
3. *Fase Pemeliharaan (Maintenance) :*

   Jenis-jenis maintenance
   a. Koreksi (Corection)
   b. Adaptasi (Adaptive)

      Software dikembangkan sesuai dengan tuntutan perkembangan zaman
   c. Adaptasi yang berkembang pada dewasa ini terbagi atas :

i. Sistem Operasi
   - ◊ Pengarahan sistem operasi yang bersifat multi user. Contoh : UNIX
   - ◊ Sistem operasi yang bersifat jaringan. Contoh : NOVELL

ii. RDBMS - Relational DataBase Management System
   - ◊ Berkembang dalam bentuk bahasa SQL (Structure Query Language).

iii. Bahasa

   Mengarah pada perkembangan bahasa generasi ke empat (4GL - Fourth Generation Language). Bahasa 4GL adalah suatu bahasa yang dibuat untuk meningkatkan produktifitas programmer dan end user. Contoh :

   a. INFORMIX - Dapat dijalankan pada PC dengan minimum RAM 4MB + 640KB dan disk storage > 40MB
   b. ORACLE
   c. INGRES
   d. AS / SET - Digunakan pada IBM AS 400
   e. POWER HOUSE - digunakan pada HR 3000

iv. Perfective

   Menyempurnakan software yang ada biasanya dilakukan karena permintaan / saran / kritik user.

Model Software Engineering

Krisis software tidak dapat hilang dalam satu satu malam, di mana tidak ada suatu pendekatan yang baik dalam mengatasi krisis software, namun gabungan dari metode untuk semua fase dalam pengembangan siftware seperti peralatan yang lebih baik untuk mengautomatisasi metode-metode ini, tehnik yang lebih baik untuk mengontrol kualitas, dan filosofi untuk koordinasi kontrol, serta manajemen dipelajari dalam suatu disiplin ilmu yang kita sebut *software engineering*.

Menurut **Fritz Bauer**, software engineering adalah disiplin ilmu yang menerapkan prinsip-prinsip engineering agar mendapatkan software yang ekonomis yang dapat dipercaya dan bekerja lebih efisien pada mesin yang sebenarnya.

Software engineering terdiri dari 3 elemen kunci, yaitu :

1. Metode,
2. Peralatan (tools),
3. Prosedur, yang memungkinkan manajer mengontrol proses pengembangan software dan memberikan praktisi dasar yang baik untuk pembentukan software berkualitas tinggi.

1. Metode Software Enginnering

Metode software engineering memberikan tehnik-tehnik bagaimana membentuk software. Metode ini terdiri dari serangkaian tugas :

a. Perencanaan & estimasi proyek
b. Analisis kebutuhan sistem dan software
c. Desain struktur data
d. Arsitektur program dan prosedur algoritma
e. Coding
f. Testing dan pemeliharaan

2. Peralatan Software Engineering

Peralatan software engineering memberikan dukungan atau semiautomasi untuk metode. Contohnya :

a. CASE (Case Aided Software Engineering), yaitu suatu software yang menggabungkan software, hardware, dan database software engineering untuk menghasilkan suatu lingkungan software engineering.
b. Database Software Engineering, adalah sebuah struktur data yang berisi informasi penting tentang analisis, desain, kode dan testing.
c. Analogi dengan CASE pada hardware adalah : CAD (Computer Aided Design), CAM (Computer Aided Manufacturing), CAE (Computer Aided Engineering)

3. Prosedur Software Engineering

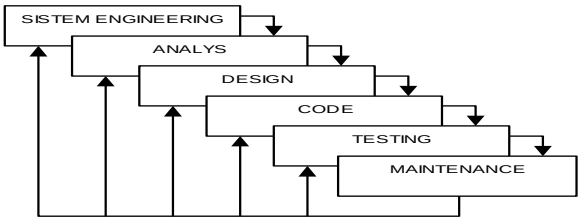
Terdiri dari :

a. urut-urutan di mana metode tersebut diterapkan
b. dokumen
c. laporan-laporan
d. formulir-formulir yang diperlukan
e. mengontrol kualitas software
f. mengkoordinasi perubahan yang terjadi pada software

Software Engineering Paradigma

Dalam penguasaan atas model software engineering atau software engineering paradigma, dikenal ada beberapa metode yang luas dipergunakan, yaitu :

1. Classic Life Cycle Pradigm - Model Water Fall - Model Siklus Hidup Klasik



Gambar 1.2. Model CLC

Keterangan gambar :

a. System Engineering and Analysis

   Karena software merupakan bagian terbesar dari sistem, maka pekerjaan dimulai dengan cara menerapkan kebutuhan semua elemen sistem dan mengalokasikan sebagian kebutuhan tersebut ke software Pandangan terhadap sistem adalah penting, terutama pada saat software harus berhubungan dengan elemen lain, seperti : Hardware, Software, Database

b. Analisis kebutuhan software

   Suatu proses pengumpulan kebutuhan software untuk mengerti sifat-sifat program yang dibentuk software engineering, atau analis harus mengerti fungsi software yang diinginkan, performance dan interface terhadap elemen lainnya. Hasil dari analisis ini didokumentasikan dan direview / dibahas / ditinjau bersama-sama customer.

c. Design

   Desain software sesungguhnya adalah *proses multi step* (proses yang terdiri dari banyak langkah) yang memfokuskan pada 3 atribut program yang berbeda, yaitu :

   - Struktur data
   - Arsitektur software
   - Rincian prosedur

   Proses desain menterjemahkan kebutuhan ke dalam representasi software yang dapat diukur kualitasnya sebelum mulai coding. Hasil dari desain ini didokumentasikan dan menjadi bagian dari konfigurasi software.

d. Coding

   Desain harus diterjemahkan ke dalam bentuk yang dapat dibaca oleh mesin

e. Testing

   Segera sesudah objek program dihasilkan, pengetesan program dimulai. Proses testing difokuskan pada logika internal software. Jaminan bahwa semua pernyataan

atau statements sudah dites dan lingkungan external menjamin bahwa definisi input akan menghasilkan output yang diinginkan.

f. Maintenance

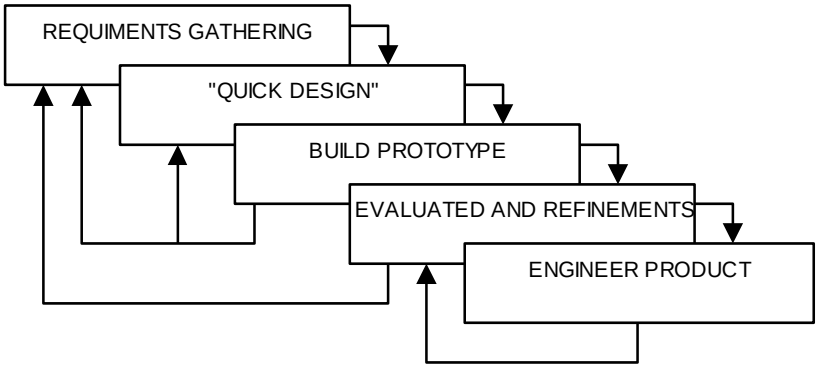
Software yang sudah dikirim ke customer data berubah karena :

- Software mengalami error
- Software harus diadaptasi untuk menyesuaikan dengan lingkungan external, misalnya adanya sistem operasi baru atau peripheral baru.
- Software yang lebih disempurnakan karena adanya permintaan dari customer.

Masalah yang dihadapi dari model siklus hidup klasik adalah :

a. Proyek yang sebenarnya jarang mengikuti aliran sequential yang ditawarkan model ini. Iterasi (Pengulangan) selalu terjadi dan menimbulkan masalah pda aplikasi yang dibentuk oleh model ini.
b. Seringkali pada awalnya customer sulit menentukan semua kebutuhan secara explisit (jelas).
c. Customer harus sabar karena versi program yang jalan tidak akan tersedia sampai proyek software selesai dalam waktu yang lama.

2. Prototype Paradigma



Gambar 1.3. Model Prototype

Seringkali seorang customer sulit menentukan input yang lebih terinci, proses yang diinginkan dan output yang diharapkan. Tentu saja ini menyebabkan developer tidak yakin dengan efisiensi alogoritma yang dibuatnya, sulit menyesuaikan sistem operasi, serta interaksi manusia dan mesin yang harus diambil. Dalam hal seperti ini, pendekatan prototype untuk software engineering merupakan langkah yang terbaik. *Prototype*

sebenarnya adalah suatu proses yang memungkinkan developer membuat sebuah model software.

Ada 2 bentuk dari model ini, yaitu :

a. Paper Prototype

   Menggambarkan interaksi manusia dan mesin dalam sebuah bentuk yang memungkinkan user mengerti bagaimana interaksi itu terjadi.

b. Working Prototype

   Adalah prototype yang mengimplementasikan beberapa bagian dari fungsi software yang diinginkan seperti pada pendekatan pengembangan software. Model ini dimulai dengan :

   - Pengumpulan kebutuhan developer dan customer
   - Menentukan semua tujuan software
   - Mengidentifikasi kebutuhan-kebutuhan yang diketahui

Hasil dari pengumpulan kebutuhan diteruskan pada **Quick Design**. Quick Design ini memfokuskan pada representasi aspek-aspek software yang dapat dilihat oleh user, misalnya format input dan output, selanjutanya dari desain cepat diteruskan pada pembentukan prototype (langkah ke 3). Prototype ini dievaluasi oleh customer / user dan digunakan untuk memperbaiki kebutuhan-kebutuhan software. Proses iterasi terjadi agar prototype yang dihasilkan memenuhi kebutuhan customer, juga pada saat yang sama developer mengerti lebih baik tentang apa yang harus dikerjakan.

Masalah yang dihadapi oleh prototyping paradigm ini adalah :

a. Customer hanya melihat pada apa yang dihasilkan oleh software, tidak peduli pada hal-hal yang berhubungan dengan kualitas software dan pemeliharaan jangka panjang.
b. Developer seringkali menyetujui apa yang diterangkan oleh customer agar prototype dapat dihasilkan dengan cepat. Akibatnya timbul pemilihan sistem operasi / bahasa pemrograman yang tidak tepat.

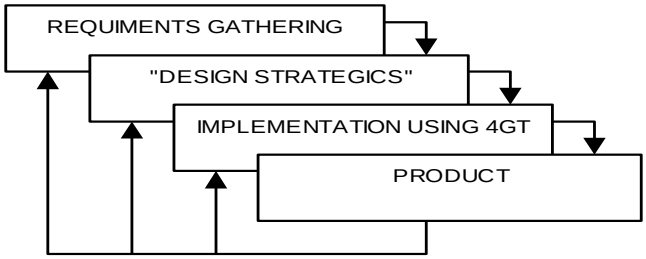
3. Fourth Generation Technique Paradigma

Istilah Fourth Generation Technique (4GT) meliputi seperangkat peralatan software yang memungkinkan seorang developer software menerapkan beberapa karakteristik software pada tingkat yang tinggi, yang kemudian menghasilkan *source code* dan *object code* secara otomatis sesuai dengan spesifikasi yang ditentukan developer.

Saat ini peralatan / tools 4GT adalah bahasa non prosedur untuk :

- DataBase Query

- Pembentukan laporan (Report Generation)
- Manipulasi data
- Definisi dan interaksi layar (screen)
- Pembentukan object dan source (Object and source generation)
- Kemampuan grafik yang tinggi, danKemampuan spreadsheet



Gambar 1.4. Model 4GT

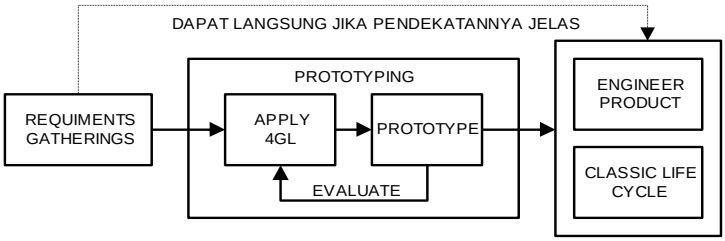
Keterangan gambar :

- Model 4GT untuk software engineering dimulai dengan rangkaian pengumpulan kebutuhan. Idealnya, seorang customer menjelaskan kebutuhan-kebutuhan yang selanjutnay diterjemahkan ke dalam prototype. Tetapi ini tidak dapat dilakukan karena customer tidak yakin dengan apa yang diperlukan, tidak jelas dalam menetapkan fakta-fakta yang diketahui dan tidak dapat menentukan informasi yang diinginkan oleh peralatan 4GT.
- Untuk aplikasi kecil adalah mungkin bergerak langsung dari langkah pengumpulan kebutuhan ke implementasi yang menggunakan bahasa non prosedur fourth generation (generasi ke 4). Tetapi untuk proyek besar, pengembangan strategi desain sistem tetap diperlukan, sekalipun kita menggunakan 4GL. Penggunaan 4GT tanpa desain untuk proyek besar akan menyebabkan masalah yang sama yang ditemui dalam pengembangan software yang menggunakan pendekatan konvensional.
- Implementasi yang menggunakan 4GL memungkinkan developer software menjelaskan hasil yang diinginkan yang kemudian diterjemahkan ke dalam bentuk source code dan object code secara otomatis.
- Langkah yang terakhir adalah mengubah implementasi 4GT ke dalam sebuah product. Selanjutnya developer harus melakukan pengetesan, pengembangan dokumentasi dan pelaksanaan semua aktifitas lainnya yang diwujudkan dalam model software engineering.

Masalah yang dihadapi dalam model 4GT adalah adanya sebagian orang yang beranggapan bahwa :

a. peralatan 4GT tidak semudah penggunaan bahasa pemrograman,
b. source code yang dihasilkan oleh peralatan ini tidak efisien,
   pemeliharaan sistem software besar yang dikembangkan dengan 4GT masih merupakan tanda tanya

4. Model Kombinasi



Gambar 1.5. Model Kombinasi

Model ini menggabungkan keuntungan-keuntungan dari beberapa model sebelumnya. Seperti pada model sebelumnya, model kombinasi ini dimulai dengan langkah pengumpulan kebutuhan Pendekatan yang dapat diambil adalah pendekatan siklus hidup klasik (Analisis sistem dan analisis kebutuhan software) atau dapat juga menggunakan pendekatan seperti prototyping jika definisi masalahnya tidak terlalu formal.

Jika kebutuhan untuk fungsi dan performance software diketahui dan dimengerti, pendekatan yang dianjurkan adalah model siklus hidup klasik. Sebaliknya, jika aplikasi software menuntut interaksi yang sering antara manusia dan mesin, membutuhkan algoritma yang tidak dapat dibuktikan, atau membutuhkan tehnik output / kontrol, maka pendekatan yang dianjurkan adalah model prototyping.

Pada kasus seperti ini, 4GL dapat digunakan untuk mendapat prototype dengan cepat. Segera sesudah prototype dievaluasi dan disempurnakan, langkah desain dan implementasi dalam siklus hidup klasik diterapkan.

Proses Pengembangan Software

Dari model yang disebut di atas dapat diambil suatu kesimpulan, bahwa proses pengembangan software terdiri dari 3 fase, yaitu :

1. Fase Definisi
2. Fase Pengembangan (Development)
3. Fase Pemeliharaan (Maintenance)

Fase Definisi

Fase definisi memfokuskan pada "*What*". Selama definisi ini, developer software berusaha untuk :

a. Mengidentifikasi informasi apa yang dikerjakan proses
b. Fungsi dan performance apa yang diinginkan
c. Interface apa yang dibutuhkan
d. Hambatan desain apa yang ada, dan
e. Kriteria validasi apa yang dibutuhkan untuk menetapkan keberhasilan sistem.

- Sistem Analis

  Sistem analis menetapkan peranan dari setiap elemen dalam sistem berbasis komputer, terutama mengalokasikan peranan software.
- Sistem Software Planning

  Dalam sistem ini, setelah lingkungan software dialokasikan, maka langkah dari sistem software planning ini adalah :
  - ✓ Pengalokasian sumber / resource
  - ✓ Estimasi biaya
  - ✓ Penetapan tugas pekerjaan dan jadual.

- Requirement Analysis

  Penetapan lingkup untuk software memberikan petunjuk / arah. Namun definisi yang lebih rinci dari informasi dan fungsi software diperlukan sebelum pekerjaan dimulai.

Fase Pengembangan

Fase pengembangan berfokus pada "*How*". Selama pengembangan, developer software berusaha menjelaskan :

a. Bagaimana struktur data dan arsitektur software yang didesain
b. Bagaimana rincian prosedur diimplementasikan (diterapkan)
c. Bagaimana desain diterjemahkan ke dalam bahasa pemrograman atau bahasa non prosedur, dan
d. Bagaimana pengetesan akan dilaksanakan.

- Desain software (Software Design)

  Desain menterjemahkan kebutuhan-kebutuhan software ke dalam sekumpulan representasi (grafik, tabel, diagram, atau bahasa yang menjelaskan struktur data, arsitektur software dan prosedur algoritma).

- Coding

  Representasi desain harus diterjemahkan ke dalam bahasa tiruan / artificial language yang menghasilkan perintah-perintah yang dapat dieksekusi oleh komputer.
- Software Testing

  Segera sesudah software diimplementasikan dalam bentuk yang dapat dieksekusi oleh mesin, software perlu ditest untuk menemukan kesalahan (merupakan fungsi logika dan implementasi).

Fase Pemeliharaan

Fase pemelihaaan berfokus pada "*Change*" atau perubahan. Ini dapat disebabkan :

a. Perubahan karena software error (Corective Maintenance)
b. Perubahan karena software disesuaikan / diadaptasi dengan lingkungan external, misalnya munculnya CPU baru, sistem operasi baru (Adaptive Maintenance)
c. Perubahan software yang disebabkan customer / user meminta fungsi tambahan, misalnya fungsi grafik, fungsi matematik, dan sebagainya (Perfective Maintenance)

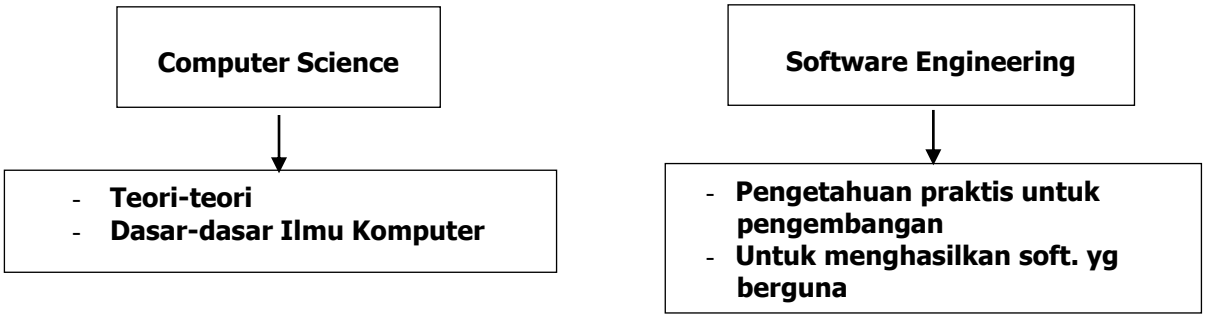
Analogi

Sebuah Analogi untuk Software Engineering

- Gas Oksigen dapat dibuat oleh :
  - Sebuah Laboratorium
  - Industri Pabrik Gas
- Perangkat lunak dapat dikembangkan oleh :
  - Sekelompok programmer
  - Sebuah organisasi yang mengembangkannya melalui rekayasa
- Apa yang membedakan antara gas oksigen dengan perangkat lunak yang dikembangkan ?

Perbedaan Sofware Engineering vs Computer Science

- Apa perbedaan antara software engineering (rekayasa perangkat lunak) dengan computer science (ilmu komputer)



Gambar 1.6. Perbedaan Software Engineering dan Computer Science

- Computer science terkait dgn teori-teori & dasar-dasar dari ilmu komputer, sedangkan software engineering terkait pada pengetahuan & penyerahan perangkat lunak yg berguna.
- Teori-teori ilmu komputer biasanya tidak cukup digunakan sebagai pendukung yang lengkap dari software engineering.

Semakin banyaknya kebutuhan disertai banyaknya masalah dengan kompleksitas yang tinggi, maka dibutuhkan untuk menyelesaikannya dengan menggunakan *Software Engineering.* Selanjutnya dimulailah :

Software Engineering adalah bagaimana mengelola kompleksitas tersebut dan dapat bekerja dalam satu Tim Work

Latihan soal :

1. Berikut ini, Mana yang bukan fase dalam engineering approach?
   a. Fase Pemeliharaan
   b. Fase Perencanaan
   c. Fase Pembaharuan
   d. Fase Pengembangan
2. Yang termasuk dalam fase definisi dalam pengembangan software adalah?
   a. Berapa harga pembuatannya?
   b. Untuk siapa software tersebut dibuat?
   c. Fungsi dan performance apa yang diinginkan?
   d. Butuh berapa lama pengerjaannya?
3. Pada Prototype Paradigma, bentuk modelnya yaitu?
   a. Hard Prototype dan Soft Prototype
   b. Working Prototype dan Paper Prototype
   c. Digital Prototype dan Paper Prototype
   d. Working Prototype dan Product Prototype
4. Masalah yang dihadapi dari 4GT paradigma adalah?
   a. Peralatan yang tidak semudah penggunaan bahasa pemrograman dan source code yang dihasilkan tidak efisien.
   b. Proyek yang sebenarnya jarang mengikuti aliran sequential yang ditawarkan model ini. Iterasi (Pengulangan) selalu terjadi dan menimbulkan masalah pda aplikasi yang dibentuk oleh model ini.
   c. Seringkali pada awalnya customer sulit menentukan semua kebutuhan secara explisit (jelas).
   d. Customer harus sabar karena versi program yang jalan tidak akan tersedia sampai proyek software selesai dalam waktu yang lama.
5. Berikut ini yang bukan serangkaian tugas dari metode software engineering adalah?
   a. Coding dan Testing
   b. Desain struktur data
   c. Dokumentasi program
   d. Perencanaan dan estimasi proyek
6. Seperangkat peralatan software yang memungkinkan seorang developer software menerapkan beberapa karakteristik software pada tingkat tinggi, yang kemudian menghasilkan source code dan object code secara otomatis. Merupakan pengertian dari?

a. Model Kombinasi

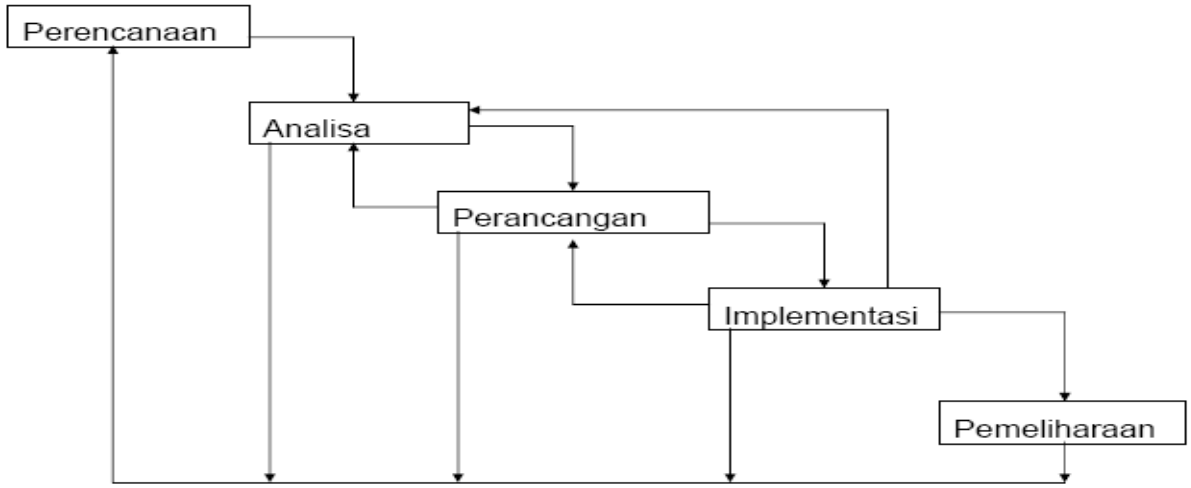
b. Fourth Generation Technique Paradigma

c. Classic Life Cycle Paradigma

d. Prototype Paradigma.

Bab 2
KEBUTUHAN DAN SPESIFIKASI SOFTWARE

Sistem yang baik adalah yang selalu menyesuaikan dengan perubahan lingkungan yang terjadi disekitarnya atau sistem tersebut harus dinamis menuju keadaan yang lebih baik.



Gambar 2.1. Model Waterfall

Keterangan gambar :

- **Tahap Perencanaan** : menyangkut studi kebutuhan user, studi kelayakan baik secara teknis maupun teknologi serta penjadwalan pengembangan suatu proyek sistem informasi
- **Tahap Analisis** : yaitu tahap dimana kita berusaha mengenali segenap permasalahan yang muncul pada pengguna, mengenali komponen-komponen sistem, obyek-obyek, hubungan antar obyek dan sebagainya
- **Tahap Perancangan** : yaitu tahap dimana kita mencoba mencari solusi permasalahan yang didapat dari tahap analisa
- **Tahap Implementasi** : tahap dimulainya pemilihan perangkat keras, penyusunan perangkat lunak aplikasi, melihat apakah sistem yang dibuat sudah sesuai dengan kebutuhan user atau belum.
- **Tahap Pemeliharaan** : mulai melakukan pengoperasian sistem dan jika diperlukan dapat melakukan perbaikan-perbaikan kecil

Teknologi Object Oriented

Teknik Object Oriented merupakan paradigma baru dalam rekayasa software yang didasarkan dengan obyek dan kelas. Teknik object oriented memandang software bagian per bagian dan menggambarkannya dalam satu obyek. Teknologi obyek menganalogikan sistem aplikasi seperti kehidupan nyata yang didominasi obyek, objek dapat beraksi dan bereaksi, contoh :

- Manusia adalah obyek yang memiliki atribut (nama, pekerjaan, rumah, dll.)
- Manusia dapat berjalan, makan, minum, dll.

Empat Prinsip Dasar dari OOP

- **Abstraksi** : memfokuskan pada karakteristik obyek
- **Enkapsulasi** : menyembunyikan banyak hal yang terdapat dalam obyek yang tidak perlu diketahui obyek lain
- **Modularitas** : membagi sistem yang rumit menjadi bagian-bagian yang lebih kecil
- **Hirarki** : berhubungan dengan abstraksi dan modularitas yaitu pembagian berdasarkan urutan dan pengelompokan tertentu.

Bagi Software Engineer, Teknik Pemodelan Object Oriented berpengaruh dalam bahasa pemrograman, metodologi rekayasa, manajemen proyek, hardware, dsb

Analisis dan perancangan berorientasi obyek adalah suatu metode analisis yang memeriksa requirements (syarat-syarat/keperluan yang harus dipenuhi suatu sistem) dari sudut pandang kelas-kelas dan obyek-obyek dalam lingkup permasalahan

Konsep Dasar Object Oriented Analysis dan Design

- Obyek adalah benda secara fisik atau konseptual yang memiliki keadaan (state) dan perilaku (behavior).
- Kelas (Class) adalah definisi umum (pola, template atau cetak biru) untuk himpunan obyek sejenis.
- Kotak hitam dan Interface sebuah obyek digambarkan sebagai kotak hitam untuk mengakses obyek melalui interface. Kotak hitam berisi Kode (himpunan instruksi dengan bahasa yang dipahami komputer) dan Data.
- Association dan Aggregation Association adalah hubungan antar obyek yang saling membutuhkan. Aggregation adalah menggambarkan seluruh bagian dari obyek.

Perbedaan Antara Metode Struktural dan OOAD

Perbedaan antara metode structural dan OOAD dapat dilihat pada bagaimana data dan fungsi disimpan. Seperti metode struktural, untuk data dan fungsi disimpan terpisah, biasanya semua ditempatkan sebelum fungsi ditulis. Sedangankan metode OOAD untuk data dan fungsi yang berhubungan dalam satu obyek disimpan bersamaan dalam satu kesatuan.

Pemrograman Berorientasi Objek

Pemrograman berorientasi objek merupakan kelanjutan dari proses analisa dan desain berorientasi obyek yang kemudian diimplementasikan dengan bahasa pemrograman berorientasi obyek. misal : C++, Java, Visual Basic, dsb

UML

Tools yang digunakan untuk pemrograman yang berorientasi objek dapat menggunakan UML (Unified Modelling Language). UML merupakan bahasa yang dapat membuat model untuk semua jenis aplikasi perangkat lunak yang dapat berjalan pada perangkat keras.

UML menyediakan beberapa notasi dan artifact standar sebagai alat komunikasi bagi pelaku dalam proses analisa dan desain.

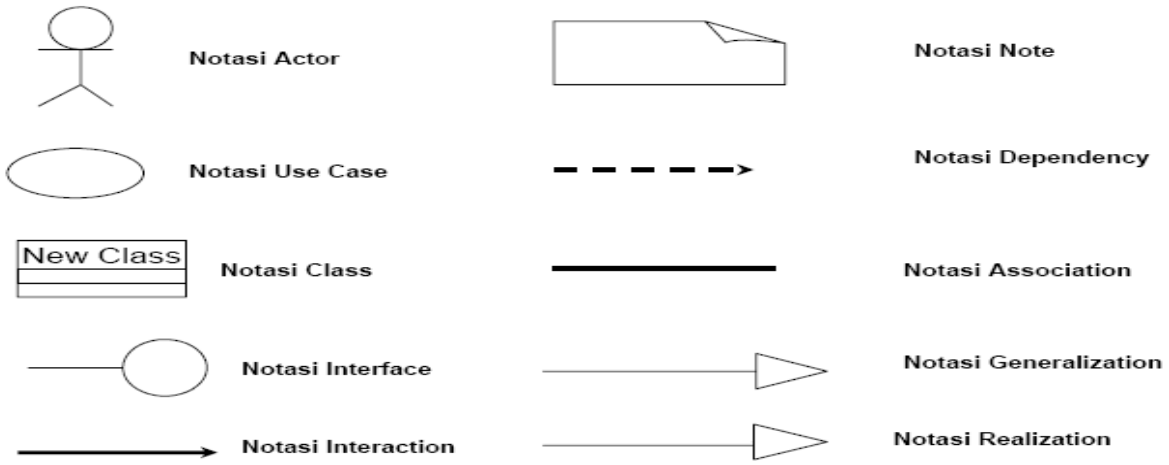
Diagram-diagram UML

- Use Case Diagram
- Class Diagram
- Behavior Diagram :
  - state chart diagram
  - activity diagram
  - interaction diagram : sequence diagram, collaboration diagram
- Implementation Diagram
- Component Diagram
- Deployment Diagram

Cakupan UML

UML merupakan penggabungan antara konsep Booch, OMT, dan OOSE, serta menekankan pada apa yang dapat dikerjakan dengan metode-metode tersebut. UML berfokus pada bahasa Pemodelan Standar, bahkan pada Proses Standar

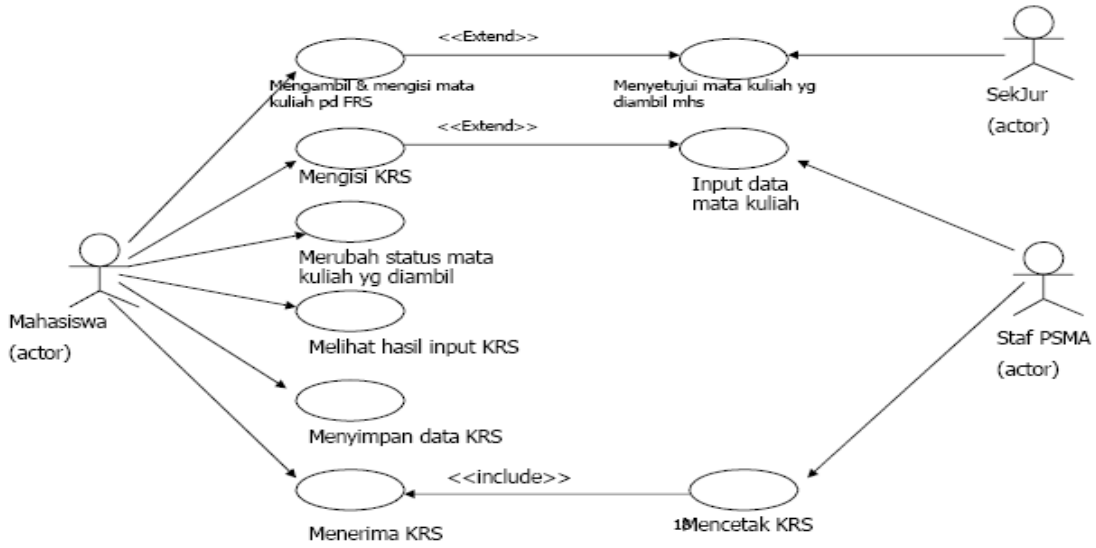
Notasi dalam UML



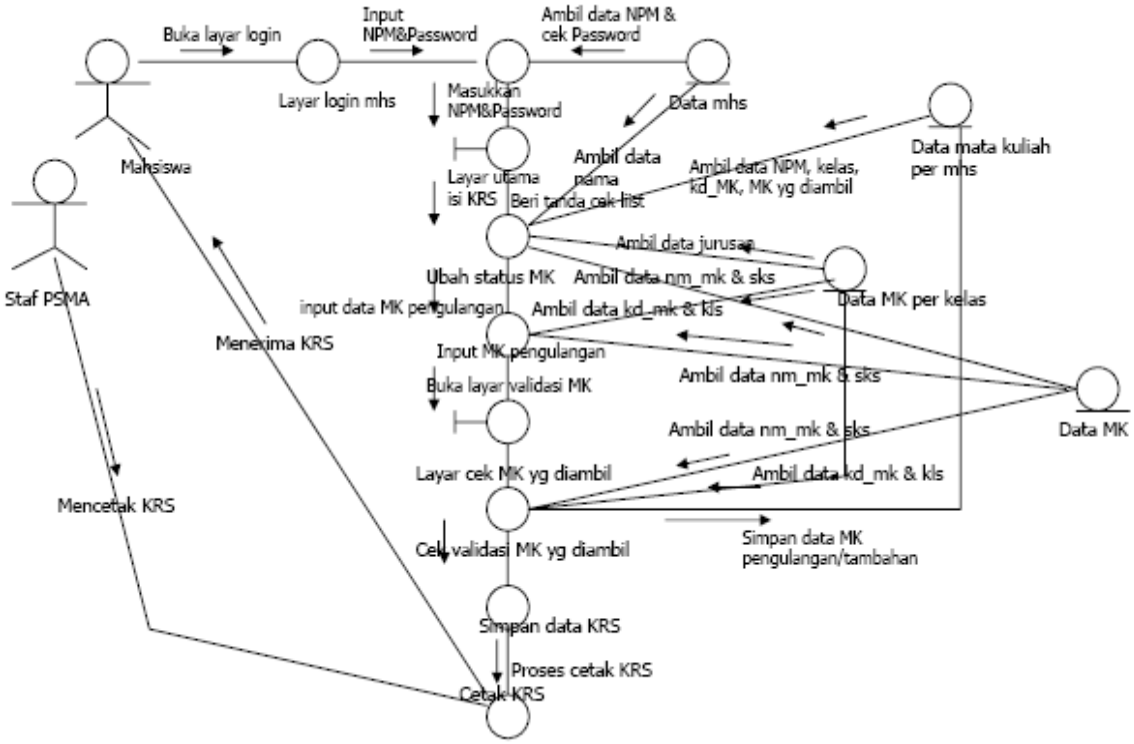
Gambar 2.2. Notasi UML

Contoh Prosedur pengisian KRS

1. Buat alur dr prosedur pengisian KRS
2. Buat alur ketika pengisian KRS
3. Use Case
4. Collaboration



Gambar 2.3. Use Case Diagram



Gambar 2.4. Collaboration Diagram

Pembahasan lebih lanjut UML akan dibahas di bab lain dalam modul ini.

Latihan soal :

1. Model Waterfall adalah suatu proses perangkat lunak yang berurutan, dibawah ini yang termasuk urutan tahapan dari model waterfall adalah..
   a. Perencanaan, perancangan, implementasi, Analisa, pemeliharaan
   b. Perencanaan, analisa, perancangan, implementasi, pemeliharaan
   c. Perencanaan, Analisa, implementasi, perancangan, pemeliharaan
   d. Perencanaan, perancangan, Analisa, implementasi, pemeliharaan
2. Berikut ini yang termasuk empat prinsip dasar dari OOP, kecuali..
   a. Polymorphism
   b. Hirarki
   c. Enkapsulasi
   d. Modularitas

3. Berikut ini yang termasuk dari Konsep Dasar Object Oriented Analysis dan Design, kecuali..
   a. Kotak hitam
   b. Kelas
   c. Obyek
   d. Abstraksi
4. Benda secara fisik atau konseptual yang memiliki keadaan (state) dan perilaku (behavior) adalah pengertian dari..
   a. Aggregation
   b. Interface
   c. Obyek
   d. Association
5. menyembunyikan banyak hal yang terdapat dalam obyek yang tidak perlu diketahui obyek lain adalah pengertian dari..
   a. Hirarki
   b. Enkapsulasi
   c. Abstraksi
   d. Modularitas
6. UML adalah singkatan dari..
   a. Unified Modelling Language
   b. Unified Monitoring Language
   c. Unity Modelling Language
   d. UnityMonitoring Language
7. Yang bukan termasuk dari penggabungan konsep UML adalah..
   a. Konsep OOSE
   b. Konseo OOAD
   c. Konsep OMT
   d. Konsep Booch
8. Berikut ini yang merupakan pengertian dari Teknik Object Oriented adalah..
   a. Kelanjutan dari proses analisa dan desain berorientasi obyek yang kemudian diimplementasikan dengan bahasa pemrograman berorientasi obyek.
   b. Paradigma baru dalam rekayasa software yang didasarkan dengan obyek dan kelas.
   c. Suatu metode analisis yang memeriksa requirements dari sudut pandang kelas-kelas dan obyek-obyek dalam lingkup permasalahan

d. Data dan fungsi yang berhubungan dalam satu obyek disimpan bersamaan dalam satu kesatuan.

9. Berikut ini yang merupakan fungsi dari metode OOAD adalah..
   a. Kelanjutan dari proses analisa dan desain berorientasi obyek yang kemudian diimplementasikan dengan bahasa pemrograman berorientasi obyek.
   b. Paradigma baru dalam rekayasa software yang didasarkan dengan obyek dan kelas.
   c. Suatu metode analisis yang memeriksa dari sudut pandang kelas-kelas dan obyek-obyek dalam lingkup permasalahan
   d. Data dan fungsi yang berhubungan dalam satu obyek disimpan bersamaan dalam satu kesatuan.
10. Berikut ini yang merupakan pengertian dari Analisis dan perancangan berorientasi obyek adalah..
   a. Kelanjutan dari proses analisa dan desain berorientasi obyek yang kemudian diimplementasikan dengan bahasa pemrograman berorientasi obyek.
   b. Paradigma baru dalam rekayasa software yang didasarkan dengan obyek dan kelas.
   c. Suatu metode analisis yang memeriksa requirements dari sudut pandang kelas-kelas dan obyek-obyek dalam lingkup permasalahan
   d. Data dan fungsi yang berhubungan dalam satu obyek disimpan bersamaan dalam satu kesatuan.

Bab 3
PERENCANAAN PROYEK PERANGKAT LUNAK

Observasi pada estimasi

Proses manajemen proyek perangkat lunak dimulai dengan planning dan perkiraan (estimasi). Yang mempengaruhi estimasi Yang pertama dari aktifitas ini adalah estimation (perkiraan). Estimasi membawa resiko yang inheren (dari diri sendiri) dan resiko inilah yang membawa ketidakpastian.

Yang mempengaruhi estimasi :

- Project Complexity (kompleksitas proyek).
- Project Size (ukuran proyek).
- Structural Uncertainty (ketidakpastian struktural)

Tujuan Perencanaan Proyek Perangkat Lunak

Menyediakan sebuah kerangka kerja yang memungkinkan manajer membuat estimasi yang dapat dipertanggungjawabkan terhadap sumber daya, biaya dan jadual pada awal proyek yang dibatasi oleh waktu.

Aktifitas Perencanaan Proyek Perangkat Lunak :

- Menentukan Ruang Lingkup Perangkat Lunak.
- Mengestimasi Sumber Daya yang Dibutuhkan.

Ruang lingkup PL menggambarkan :

- Fungsi untuk memberikan awalan yang lebih detail pada saat dimulai estimasi.
- Kinerja melingkupi pemrosesan dan kebutuhan waktu respon.
- Batasan mengidentifikasi batas yang ditempatkan pada PL oleh hardware eksternal, memori dan sistem lain.
- Konsep sebuah interface diinterpretasi untuk menentukan :
    - Hardware yang mengeksekusi PL dan device yang dikontrol secara langsung oleh PL.
    - Software yang sudah ada dan harus dihubungkan dengan PL yang baru.
    - Manusia yang menggunakan PL melalui perangkat I/O.
    - Prosedur.

Sumber Daya

- Manusia.
- Perangkat Lunak.
    - Memiliki kategori yang diusulkan oleh BEUNATAN :
    - Komponen Off-the self.
    - Komponen Full-Experience.
    - Komponen Partial-Experience.
    - Komponen Baru.
- Lingkungan (Software Engineering Environment – SEE), menggabungkan PL dan hardware.

Estimasi Proyek Perangkat Lunak

Akurasi estimasi proyek PL didasarkan pada :

- Tingkat dimana perencana telah dengan tepat mengestimasi ukuran produk yang akan dibuat.
- Kemampuan mengestimasi ukuran ke dalam kerja manusia, waktu kalender,dan dolar.
- Tingkat dimana rencana proyek mencerminkan kemampuan tim PL.
- Stabilitas syarat produk serta lingkungan yang mendukung usaha pengembangan PL.

Putnam dan Myers mengusulkan 4 masalah penentuan dari ukuran :

- Fuzzy-logic sizing (logika kabur).
  Perencana harus mengidentifikasi tipe aplikasi, membuat besarannya dalam skala kuantitatif kemudian dibandingkan dengan rentang orisinil.
- Function point sizing.
  Perencana mengembangkan estimasi berdasarkan karakteristik domain informasi.

- Standard component sizing.
  PL dibangun dari sejumlah 'komponen standar' yg umum (subsistem, modul, laporan, program interaktif).

- Change sizing.
- Digunakan jika PL yang ada harus dimodifikasi dengan banyak cara sebagai bagian dari proyek.

COCOMO

Barry Boehm memperkenalkan hirarki model estimasi PL dengan nama COCOMO (**CO**nstructive **CO**st **MO**del = Model Biaya Konstruktif).

COCOMO adalah sebuah model yang didesain oleh Barry Boehm untuk memperoleh perkiraan dari jumlah orang-bulan yang diperlukan untuk mengembangkan suatu produk perangkat lunak. Satu hasil observasi yang paling penting dalam model ini adalah bahwa motivasi dari tiap orang yang terlibat ditempatkan sebagai titik berat. Hal ini menunjukkan bahwa kepemimpinan dan kerja sama tim merupakan sesuatu yang penting, namun demikian poin pada bagian ini sering diabaikan.

1. Model COCOMO Dasar.

   Model COCOMO dapat diaplikasikan dalam tiga tingkatan kelas :

   - *Proyek organik* (organic mode) adalah proyek dengan ukuran relatif kecil, dengan anggota tim yang sudah berpengalaman, dan mampu bekerja pada permintaan yang relatif fleksibel.
   - *Proyek sedang* (semi-detached mode)Merupakan proyek yang memiliki ukuran dan tingkat kerumitan yang sedang, dan tiap anggota tim memiliki tingkat keahlian yang berbeda
   - *Proyek terintegrasi* (embedded mode). Proyek yang dibangun dengan spesifikasi dan operasi yang ketat

   Model COCOMO dasar ditunjukkan dalam persamaan 1, 2, dan 3 berikut ini :

$$E = a_b (KLOC)^b b \quad (1)$$

$$D = c_b (E)^d b \quad (2)$$

$$P = E/D \quad (3)$$

   Dimana :

   E : besarnya usaha (orang-bulan)

   D : lama waktu pengerjaan (bulan)

   KLOC : estimasi jumlah baris kode (ribuan)

   P : jumlah orang yang diperlukan.

   Sedangkan koefisien ab, bb, cb, dan db diberikan pada Tabel 1 berikut :

Tabel 3.1. Koefisien untuk Model COCOMO

Proyek Perangkat Lunak	a_b	b_b	c_b	d_b
Organik	2.4	1.05	2.5	0.38
Sedang	3.0	1.12	2.5	0.35
Terintegrasi	3.6	1.20	2.5	0.32

2. Model COCOMO Lanjut (Intermediate COCOMO)

Pengembangan model COCOMO adalah dengan menambahkan atribut yang dapat menentukan jumlah biaya dan tenaga dalam pengembangan perangkat lunak, yang dijabarkan dalam kategori dan subkatagori sebagai berikut:

1. Atribut produk (product attributes)
   - Reliabilitas perangkat lunak yang diperlukan (RELY)
   - Ukuran basis data aplikasi (DATA)
   - Kompleksitas produk (CPLX)

2. Atribut perangkat keras (computer attributes)

   - Waktu eksekusi program ketika dijalankan (TIME)
   - Memori yang dipakai (STOR)
   - Kecepatan mesin virtual (VIRT)
   - Waktu yang diperlukan untuk mengeksekusi perintah (TURN)

3. Atribut sumber daya manusia (personnel attributes)
   - Kemampuan analisis (ACAP)
   - Kemampuan ahli perangkat lunak (PCAP)
   - Pengalaman membuat aplikasi (AEXP)
   - Pengalaman penggunaan mesin virtual (VEXP)
   - Pengalaman dalam menggunakan bahasa pemrograman (LEXP)

4. Atribut proyek (project attributes)
   - Penggunaan sistem pemrograman modern(MODP)
   - Penggunaan perangkat lunak (TOOL)

- Jadwal pengembangan yang diperlukan (SCED)
- Masing-masing subkatagori diberi bobot seperti dalam tabel 3.2 dan kemudian dikalikan.

Tabel 3.2. Nilai Bobot Sub-Kategori

Driver Type	Code	V. Low	Low	Nominal	High	V. High	Ex.High
Product Attributes	RELY	0,75	0,88	1,00	1,15	1,40	
	DATA		0,94	1,00	1,08	1,16	
	CPLX	0,70	0,85	1,00	1,15	1,30	1,65
Computer Attributes	TIME			1,00	1,11	1,30	1,66
	STOR			1,00	1,06	1,21	1,56
	VIRT	0,87	1,00	1,15	1,30		
	TURN		0,87	1,00	1,07	1,15	
Personnel Attributes	ACAP	1,46	1,19	1,00	0,86	0,71	
	AEXP	1,29	1,13	1,00	0,91	0,82	
	PCAP	1,42	1,17	1,00	0,86	0,70	
	VEXP	1,21	1,10	1,00	0,90		
	LEXP	1,14	1,07	1,00	0,95		
Project Attributes	MODP	1,24	1,10	1,00	0,91	0,82	
	TOOL	1,24	1,10	1,00	0,91	0,83	
	SCED	1,24	1,08	1,00	0,91	1,10	

Pada aplikasi PL, dari segi biaya sering lebih efektif membeli daripada mengembangkan sendiri. Pada keputusan make-buy dengan pilihan :

- PL dapat dibeli (atau lisensi) off-the-self.
- Komponen PL full-experiencedan partial-experience.
- PL dapat dibuat custom-built oleh kontraktor luar.

$$\mathbf{E}=a_i(\mathbf{KLOC})^{(b_i)}.\mathbf{EAF} \qquad (4)$$

Dimana :

E : besarnya usaha (orang-bulan)
KLOC : estimasi jumlah baris kode (ribuan)
EAF : faktor hasil penghitungan dari sub-katagori di atas.

Koefisien ai dan eksponen bi diberikan pada tabel berikut.

Tabel 3.3. Koefisien Model COCOMO Lanjut

Proyek Perangkat Lunak	a_i	b_i
Organik	3.2	1.05
Sedang	3.0	1.12
Terintegrasi	2.8	1.20

2.1 Persamaan Perangkat Lunak

Persamaan perangkat lunak merupakan model variabel jamak yang menghitung suatu distribusi spesifik dari usaha pada jalannya pengembangan perangkat lunak. Persamaan berikut ini diperoleh dari hasil pengamatan terhadap lebih dari 4000 proyek perangkat lunak :

$$E = [LOC \times B^{0.333}/P]^3 \times (1 / t^4) \qquad (5)$$

Dimana :

E = usaha yang dilakukan (orang-bulan atau orang-tahun)
t = durasi proyek dalam (bulan atau tahun)
B = faktor kemampuan khusus
P = parameter produktivitas

Nilai B diambil berdasarkan perkiraan. Untuk program berukuran kecil (0.5 < KLOC < 5), B = 0.16. Untuk program yang lebih besar dari 70 KLOC, B = 0.39.

Sedangkan besarnya nilai P merefleksikan :

- Kematangan proses dan praktek manajemen
- Kualitas rekayasa perangkat lunak
- Tingkat bahasa pemrograman yang digunakan
- Keadaan lingkungan perangkat lunak
- Kemampuan dan pengalaman tim pengembang
- Kompleksitas aplikasi

Berdasarkan teori, diperoleh P = 2000 untuk sistem terapan, P = 10000 untuk perangkat lunak pada sistem informasi dan sistem telekomunikasi, dan P = 28000 untuk sistem aplikasi bisnis.

2.2 Konversi Waktu Tenaga Kerja

Konversi waktu tenaga kerja ini diperoleh dari angka pembanding yang digunakan pada perangkat lunak ConvertAll, dengan hubungan persamaan antara orang-bulan (OB), orang-jam (OJ), orang-minggu (OM), dan orang-tahun (OT) adalah sebagai berikut :

OM = 40 OJ (6)

OT = 12 OB (7)

OT = 52 OM (8)

Dari persamaan di atas, diperoleh konversi orang-bulan ke orang-jam sebagai berikut :

OB = (40 OJ x 52) / 12 (9)

OB = 173,33 OJ

3. Model COCOMO II

Model COCOMO II, pada awal desainnya terdiri dari 7 bobot pengali yang relevan dan kemudian menjadi 16 yang dapat digunakan pada arsitektur terbarunya.

Tabel 3.4. COCOMO II Early Design Effort Multipliers

CODE	Effort Modifier
RCPX	Product reliability and complexity
REUSE	Required reusability
PDIF	Platform difficulty
PERS	Personnel capability
PREX	Personel experience
FCIL	Facilities available
SCED	Schedule pressure

Tabel 3.5. COCOMO II Post Architecture Effort Multipliers

Modifier Type	Code	Effort Modifier
Product Attributes	RELY	Required software reliability
	DATA	Database size
	DOCU	Documentation match to life-cycle needs
	CPLX	Product complexity
	REUSE	Required reuseable
Computer Attributes	TIME	Execution time constraint
	STOR	Main storage constraint
	PVOL	Platform volatility
Personnel Attributes	ACAP	Analyst capabilities
	AEXP	Application experience
	PCAP	Programmer capabilities
	PEXP	Platform experience
	LEXP	Programming language experience
	PCON	Personnel continuity
Project Attributes	TOOL	Use of software tools
	SITE	Multisite development
	SCED	Schedule pressure

Sama seperti COCOMO Intermediate, masing-masing sub katagori dapat digunakan untuk aplikasi tertentu pada kondisi very low, low, manual, nominal, high maupun very high. Masing-masing kondisi memiliki nilai bobot tertentu. Nilai yang lebih besar dari 1 menunjukkan usaha pengembangan yang meningkat, sedangkan nilai di bawah 1 menyebabkan usaha yang menurun. Kondisi Laju nominal (1) berarti bobot pengali tidak berpengaruh pada estimasi. Maksud dari bobot yang digunakan dalam COCOMO II, harus dimasukkan dan direfisikan di kemudian hari sebagai detail dari proyek aktual yang ditambahkan dalam database.

Latihan soal :

1. Berikut ini yang tidak dapat mempengaruhi estimasi adalah..
   a. Project size
   b. Project complexity
   c. Standard Component Sizing
   d. Structural Uncertainty
2. Pengembangan model COCOMO adalah dengan menambahkan atribut yang dapat menentukan jumlah biaya dan tenaga dalam pengembangan perangkat lunak, yang dijabarkan dalam beberapa kategori dan subkatagori. Yang termasuk dari atribut proyek adalah..

a. SCED
b. VEXP
c. STOR
d. VIRT

3. Yang bukan termasuk dari tingkatan kelas Model COCOMO adalah..
   a. Proyek organic
   b. Proyek sedang
   c. Proyek besar
   d. Proyek terintegrasi
4. Putnam dan Myers mengusulkan 4 masalah penentuan dari ukuran, Perencana mengembangkan estimasi berdasarkan karakteristik domain informasi disebut..
   a. Change sizing.
   b. Standard component sizing.
   c. Fuzzy-logic sizing (logika kabur).
   d. Function point sizing.
5. COCOMO merupakan singkatan dari..
   a. Cost Constructive Modal
   b. Constructive Cost Modal
   c. Cost Constructing Modal
   d. Constructing Cost Modal
6. Konsep sebuah interface diinterpretasi untuk menentukan..
   a. Stabilitas syarat produk serta lingkungan yang mendukung usaha pengembangan PL.
   b. Software yang sudah ada dan harus dihubungkan dengan PL yang baru.
   c. Perencana harus mengidentifikasi tipe aplikasi, membuat besarannya dalam skala kuantitatif kemudian dibandingkan dengan rentang orisinil.
   d. Batasan mengidentifikasi batas yang ditempatkan pada PL oleh hardware eksternal, memori dan sistem lain.
7. Dasar akurasi estimasi proyek PL adalah..
   a. Stabilitas syarat produk serta lingkungan yang mendukung usaha pengembangan PL.
   b. Software yang sudah ada dan harus dihubungkan dengan PL yang baru.
   c. Perencana harus mengidentifikasi tipe aplikasi, membuat besarannya dalam skala kuantitatif kemudian dibandingkan dengan rentang orisinil.
   d. Batasan mengidentifikasi batas yang ditempatkan pada PL oleh hardware eksternal, memori dan sistem lain.

Bab 4
PERMODELAN ANALISIS

Perbedaan antara metoda analisis dan perancangan sistem dengan paradigma konvensional dibandingkan dengan paradigma berorientasi objek (OOAD) :

1. Paradigma Konvensional
   - Fokus pada Proses (Input-Proses-Output);
   - Data terpisah dari Prosedur;
   - Dekomposisi Fungsional

     Sistem diurai menjadi sejumlah fungsi (prosedur, logika) dengan sistem tersentralisasi (struktur terhirarki) dimana data dapat dibagi dan digunakan secara bersama.

2. Paradigma Berorientasi Objek
   - Fokus pada Domain Objek, tidak pada prosedur;
   - Data dan Prosedur disimpan dalam Objek;
   - Dekomposisi Data

Sistem diurai ke dalam sejumlah obyek (konsep, abstrak, benda) dalam dunia nyata yang saling berkomunikasi dan melaksanakan sejumlah pelayanan secara desentralisasi.
Setiap obyek membungkus (*encapsulate*) sejumlah prosedur dan data yang berinteraksi dengan obyek lainnya melalui suatu pesan (*message*).

Dapat dilihat pada table di bawah ini :

Tabel 4.1. Perbandingan Structure dan Object-Oriented Paradigm

Structured Paradigm	Object-OrientedParadigm
1. Requirements phase	1. Requirements phase
2. Specification (analysis) phase	2′. Object-oriented analysis phase
3. Design phase	3′. Object-oriented design phase
4. Implementation phase	4′. Object-oriented programming phase
5. Integration phase	5. Integration phase
6. Maintenance phase	6. Maintenance phase
7. Retirement	7. Retirement

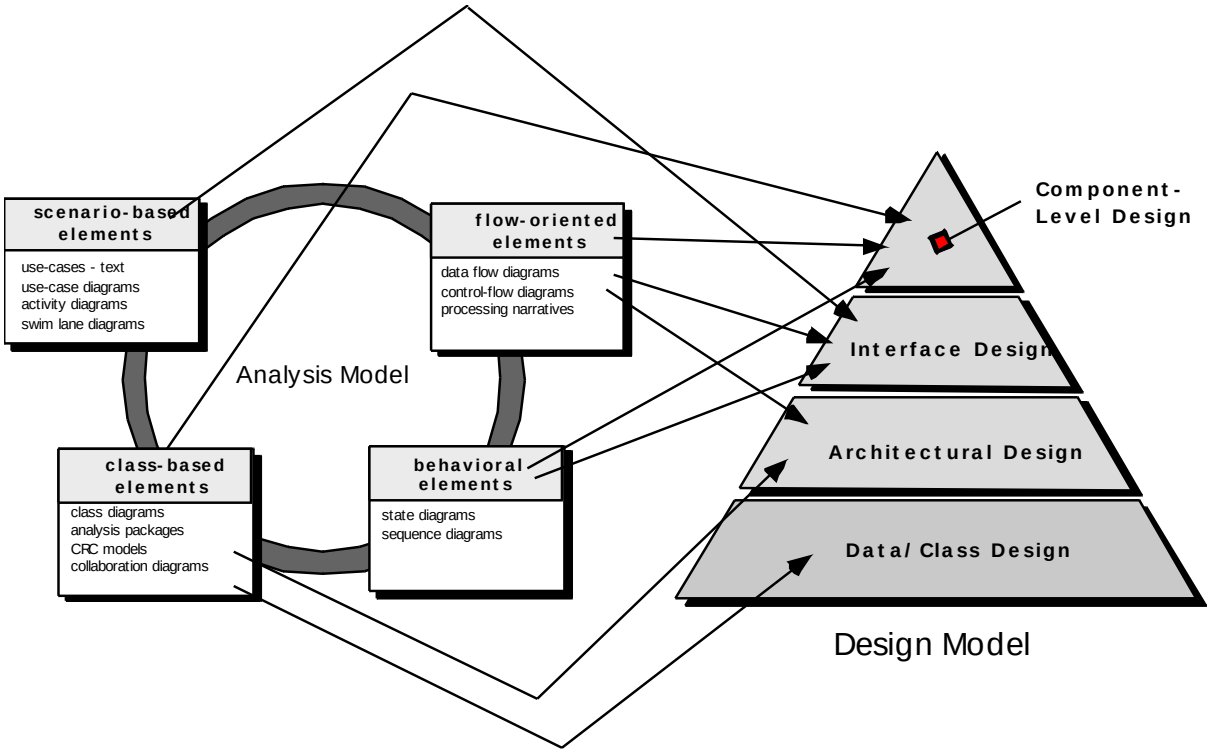
1. Analisis Model ke Design Model

Software desain adalah software tindakan rekayasa lalu dalam aktivitas modeling dan set panggung untuk konstruksi (pembuatan kode dan pengujian).

Aliran informasi selama perancangan perangkat lunak diilustrasikan pada Gambar di bawah ini. Model analisis, yang diwujudkan oleh, unsur-unsur aliran yang berorientasi dan perilaku berbasis kelas berbasis skenario, pakan tugas desain.

Desain arsitektur mendefinisikan hubungan antara elemen struktural dari perangkat lunak, gaya arsitektur dan pola desain yang dapat digunakan untuk mencapai persyaratan yang ditetapkan untuk sistem, dan kendala yang mempengaruhi cara di mana desain arsitektur dapat diimplementasikan.

Desain arsitektur dapat berasal dari Sistem Specs, model analisis, dan interaksi subsistem didefinisikan dalam model analisis.



Gambar 4.1. Analisis ke Design Model

Desain antarmuka menjelaskan bagaimana software berkomunikasi dengan sistem yang interpolasi dengan manusia yang menggunakannya. Sebuah antarmuka menyiratkan arus informasi (data, dan atau kontrol) dan jenis tertentu dari perilaku.

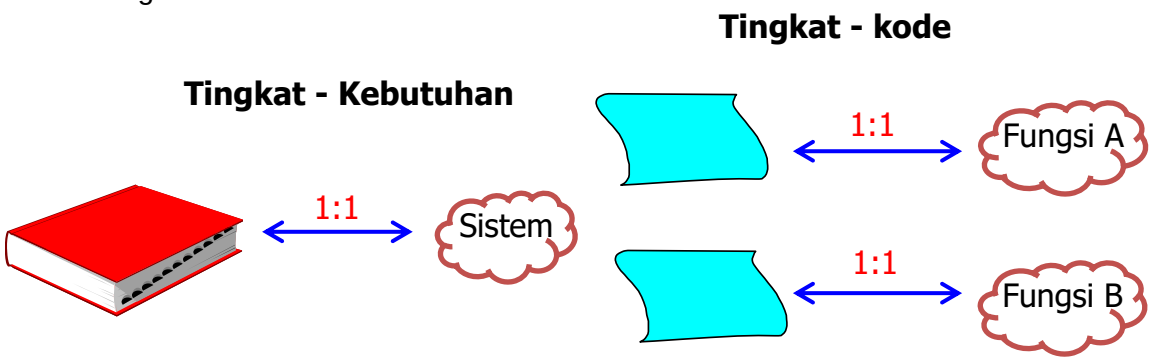
Desain komponen-tingkat mengubah elemen struktural dari arsitektur perangkat lunak menjadi deskripsi prosedur dari komponen software.

Pentingnya desain perangkat lunak dapat dinyatakan dengan satu kata - berkualitas. Desain adalah tempat di mana kualitas dipupuk dalam rekayasa perangkat lunak. Design memberikan representasi perangkat lunak yang dapat dinilai untuk kualitas. Desain adalah satu-satunya cara secara akurat yang dapat menerjemahkan kebutuhan pelanggan menjadi produk jadi atau sistem perangkat lunak.

Permodelan Analisis

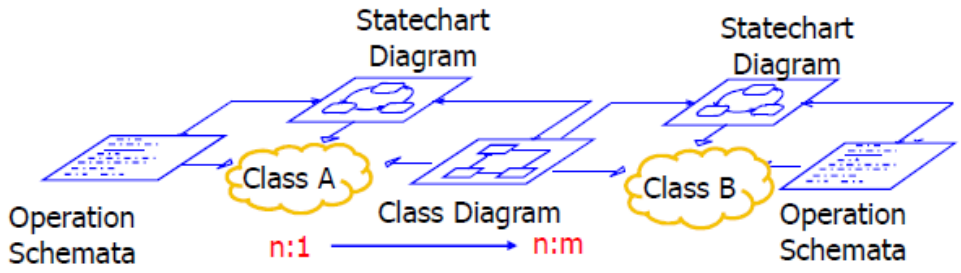
Dalam permodelan analisis ini terdapat dua macam paradigma :

1. Paradigma Konvensional



Gambar 4.2. Paradigma Konvensional

2. Paradigma Berorientasi Objek



Gambar 4.3. Paradigma Berorientasi Objek

Keuntungan menggunakan *object-oriented* adalah

- Reusability
- Modularity
- Maintainability

Konsep utama dari *object-oriented* adalah

- Objects

- Attributes
- Methods
- Encapsulation
- Polymorphism
- Classes and class hierarchies
- Instances
- Inheritance
- Abstraction and hiding
- Messages

2. Objek

Objek adalah suatu abstraksi dari sesuatu dalam suatu domain masalah, menyatakan kemampuan sistem untuk :

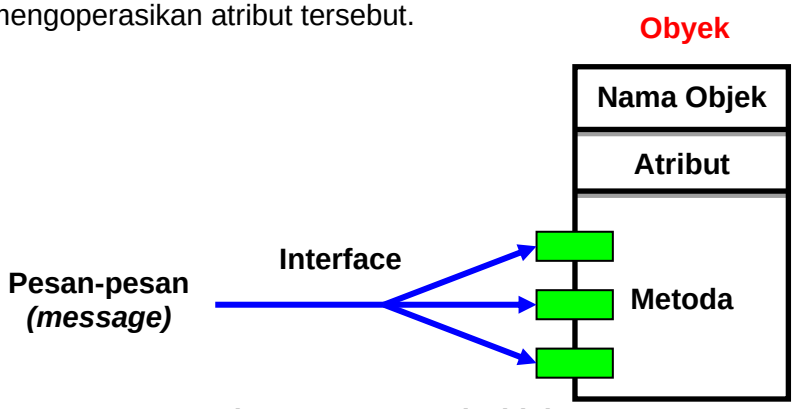
- menyimpan informasi tentang objek tersebut,
- berinteraksi dengan objek tersebut,
- atau keduanya.

Objek merupakan entitas didalam sebuah sistem perangkat lunak yang meyajikan contoh nyata dan entitas-entitas sistem.

Objek biasanya berupa benda atau sesuatu kejadian

- Benda konkrit : pesawat, lampu, buku, …
- Konsepsi : terbang, terang, kuliah, …
- Abstraksi : perusahaan, bisnis, sekolah, …

Objek adalah sekumpulan atribut (data) bersama dengan gabungan metoda (fungsi) yang digunakan untuk mengoperasikan atribut tersebut.



Gambar 4.4. Anatomi Objek

Keterangan gambar :

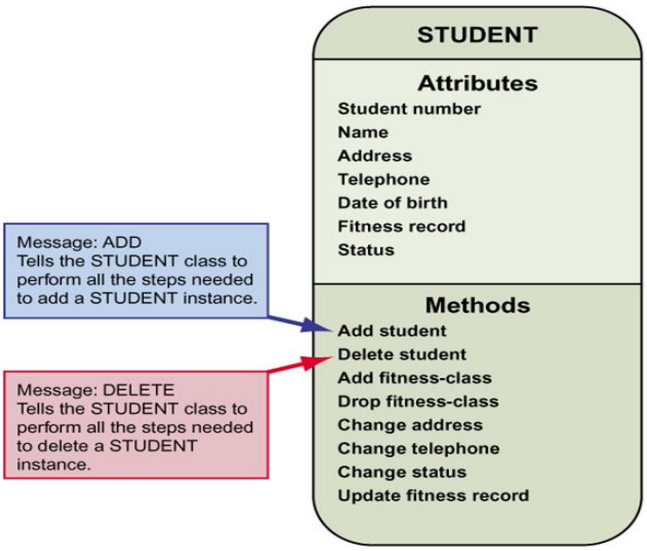
Dunia luar berkomunikasi ke obyek dengan mengirimkan pesan (message).

Atribut adalah nilai internal atau data terkait pada suatu objek yang menunjukan :

- Ciri-ciri atau sifat-sifat dari obyek
- Penggambaran keadaan (state) obyek

Methods (operations, behavior)

- Behavior mendifinisikan bagaimana suatu objek bertindak dan bereaksi, dan berhubungan dengan fungsi diterapkan pada suatu atribut.
- Behavior objek disebut metoda atau operasi pelayanan (service).



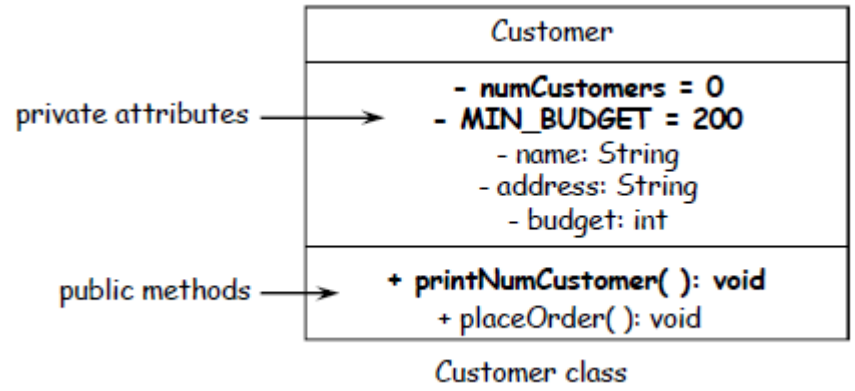
Gambar 4.5. Contoh Objek

Encapsulation

- Pengkapsulan berarti mengemas beberapa item bersama-sama menjadi satu unit yang tertutup dalam rangka menyembunyikan struktur internal suatu obyek dari lingkungan/dunia luar.
- Pengapsulan seringkali dianggap sebagai “penyembunyian informasi”.

Tiga Metode Enkapsulasi yaitu :

- *Private* : attributes dan methods dienkapsulasi didalam class dan hanya dapat diakses oleh member class tersebut.
- *Public*: metode mendefinisikan inteface sebagai sarana mengakses class dari client-nya.Dapat diakses oleh object manapun.
- *Protected* : hanya dapat diakses oleh object-class turunannya

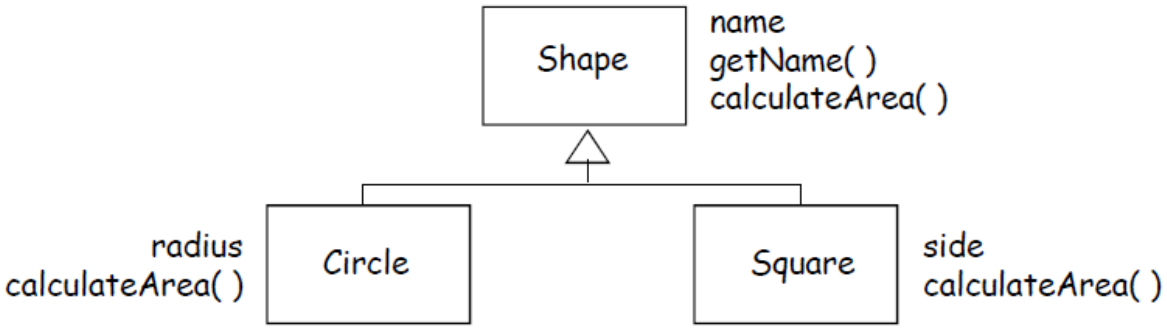


Gambar 4.6. Metode Enkapsulasi dalam sebuah Objek

Polymorphism

- Kemampuan object yang berbeda untuk menjalankan method yang sesuai untuk merespon ke pesan yg sama
- Pemilihan method yang sesuai tergantung pada class yg digunakan untuk membuat object

Contoh: suatu kelas segi-empat dan kelas segi-tiga dapat melakukan suatu metode 'hitungLuas' tetapi dengan menggunakan rumus perhitungan luas yang berbeda.



Gambar 4.7. Contoh Polymorphism terhadap perhitungan luas

Menghitung luas dapat dilakukan dengan banyak cara, contoh seperti gambar di atas yang menggunakan rumus circle atau square dalam perhitungan luasnya.

3. Keterikatan antar Objek : *Object Cohesion dan Coupling*

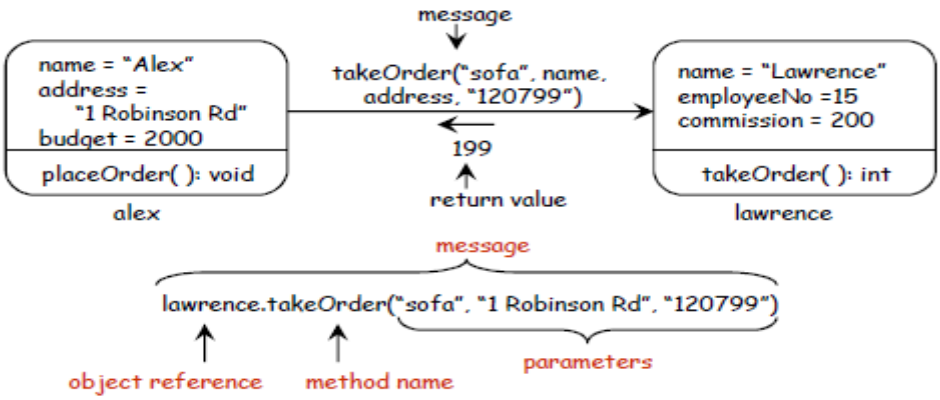
Cohesion suatu komponen adalah ukuran tentang hubungan antara komponen suatu object class. Setiap operasi menyediakan fungsi untuk mengubah, melihat, atau menggunakan atribut object sebagai layanan dasar,

Coupling adalah suatu indikasi kekuatan interkoneksi antara program units. Sistem dengan coupling yg kuat memiliki interkoneksi yang kuat sehingga setiap program unit sangat

ketergantungan dengan yang lainnya (mis.: shared variables, interchange control function). Sistem dengan couple yang lemah tidak memiliki ketergantungan yang kuat antar program units.

Komunikasi dalam Objek

- Object berkomunikasi dengan object lain melalui pengiriman pesan (messages)
- Suatu pesan adalah suatu metode call dari suatu object pengirim-pesan ke suatu object penerima pesan
- Suatu pesan terdiri dari: Object referensi yang mengindikasikan penerima pesan, nama method dan parameter (argumen dari method)
- Object penerima pesan disebut server ke object pengirim pesan, dan objek pengirim pesan adalah client dari server.



Gambar 4.8. Contoh Komuniasi dalam Objek

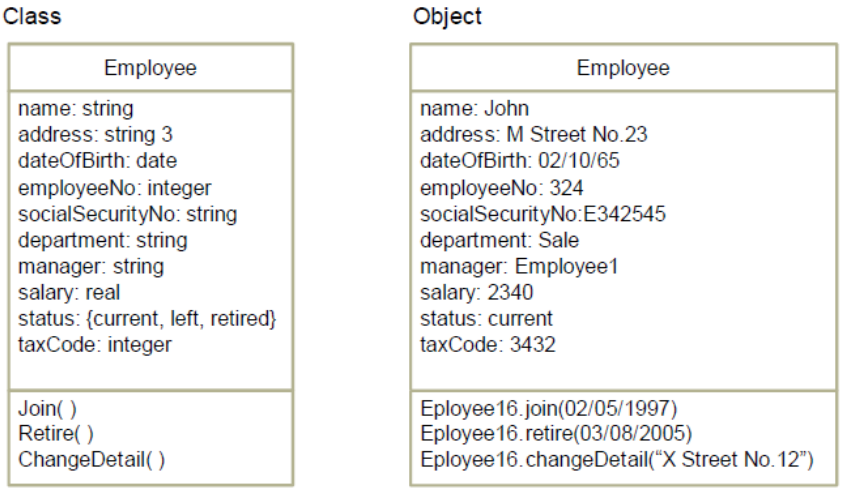
4. Kelas (Class)

Seperti di dunia nyata, objek-objek dapat dikelompokkan atau diklasifikasikan kedalam suatu kelas.

Kelas adalah adalah definisi umum (pola, template, blueprint) untuk menghimpun objek sejenis yaitu koleksi dari objek-objek yang memiliki anggota-anggota yang sama (ciri-ciri, struktur dan perilakunya).

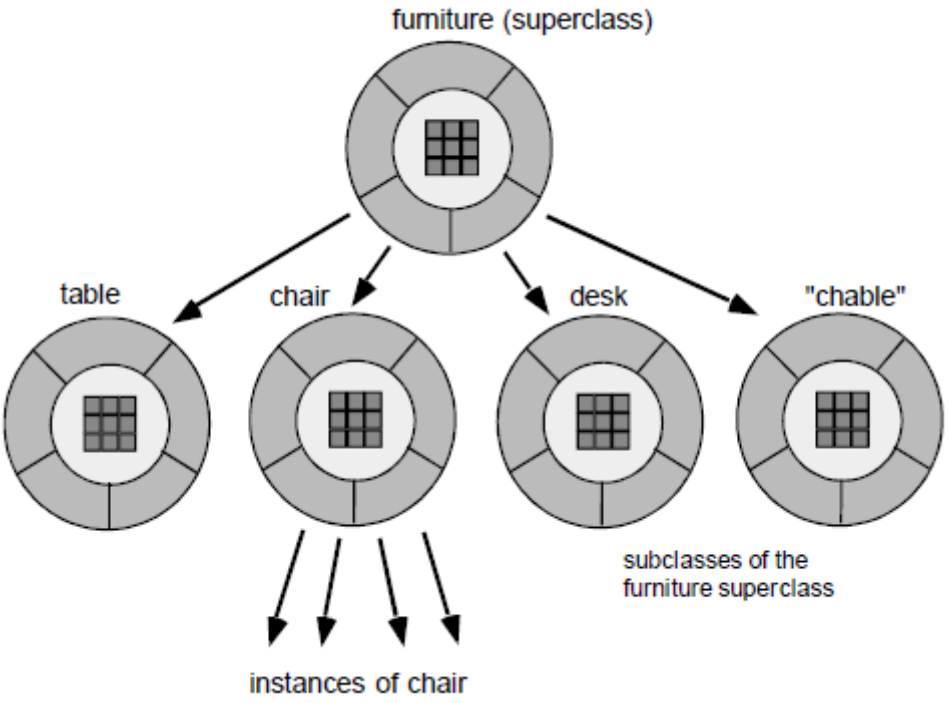
Kelas menetapkan spesifikasi atribut dan perilaku objek-objek tersebut. Kelas adalah abstraksi entitas dalam dunia nyata, objek adalah “contoh kejadian” (instance) dari sebuah kelas.

Objek dapat juga turunan (inheritance) suatu Kelas, dimana Kelas adalah kategori umum suatu objek dan Objek adalah kejadian spesifik dari suatu kelas.



Gambar 4.9. Contoh Class dan Object

Class Hierarchies

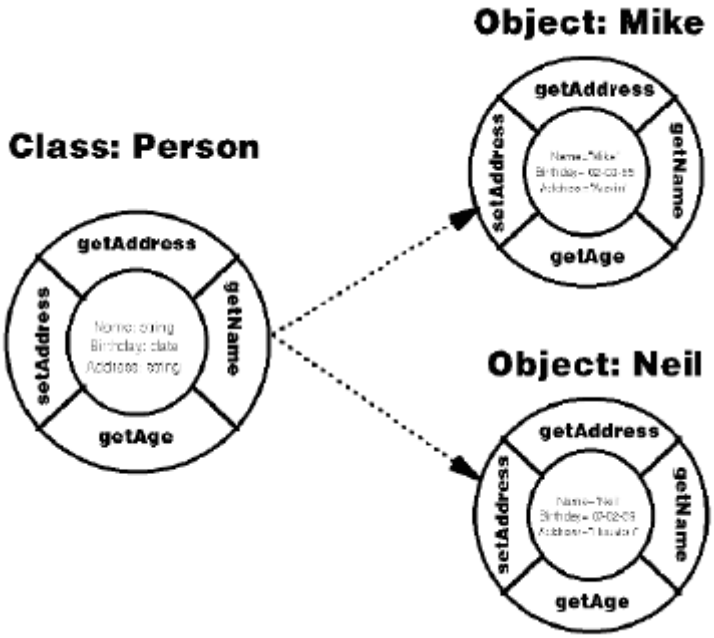


Gambar 4.10. Contoh Class Hierarchies

Instance

- Object didefinisikan oleh Class, setiap Object adalah instance dari suatu Class.
- Contoh: Atribut untuk kelas binatang adalah berkaki empat dan memiliki ekor. Perilakunya adalah tidur dan makan.

Instance yang mungkin adalah: kucing, kuda, …



Gambar 4.11. Contoh Instance

Inheritance

Hirarki klasifikasi memungkinkan kelas-kelas obyek mewarisi atribut-atribut dari kelas-kelas yang lebih umum.

Pewarisan adalah suatu mekanisme menciptakan kelas-kelas baru (sub-kelas) dari kelas-kelas yang sudah ada. Kelas turunannya adalah sebuah subkelas atau subtype dari kelas sebelumnya. Sub-kelas tersebut dapat diperluas perilakunya dengan menambah metode-metode baru atau struktur-sturktur data baru.

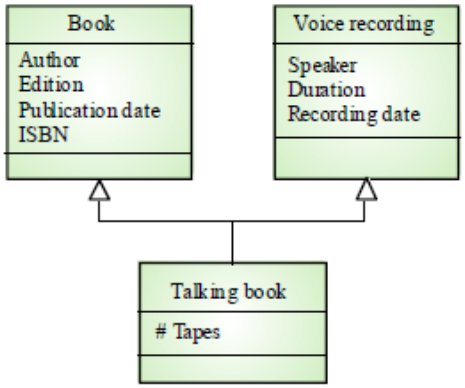
Keuntungan Inheritance:

- Merupakan mekanisme abstraksi yang dapat digunakan untuk mengklasifikasikan entitas
- Merupakan mekanisme re-use pada tahap perancangan dan pemrograman
- Grafik Inheritance adalah suatu bentuk gambaran tetang organisasi pada suatu domain dan sistem

Multiple Inheritance

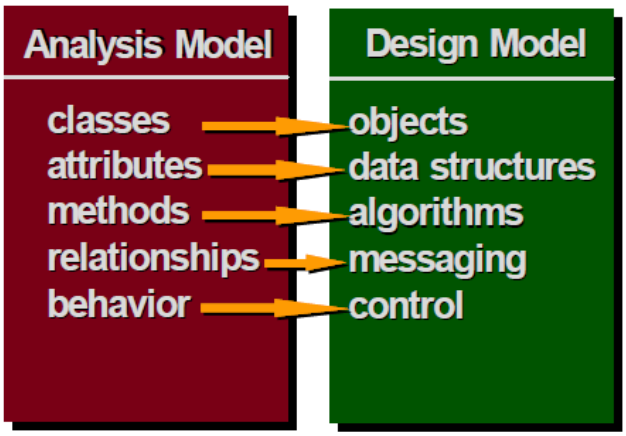
- Suatu object class dapat pula dibentuk dari turunan beberapa super-class,
- Akan memberikan dampak konflik semantic dimana atribut/service dengan nama yang sama pada super-class yang berbeda memiliki semantic yang berbeda

- Membentuk hierarchy yang lebih kompleks



Gambar 4.12. Contoh Multiple Inheritance

Keterkaitan antara OOA dan OOD



Gambar 4.13. Keterkaitan OOA dan OOD

UML

UML singkatan dari Unified Modeling Language adalah bahasa pemodelan visual dalam rekayasa perangkat lunak yaitu untuk menggambarkan, menspesifikasikan, membangun, dan mendokumentasikan sistem perangkat lunak.

UML mengkombinasikan:

- Metoda Booch
- Metoda OMT (Object Modeling Technique)
- Metoda OOSE (Object Oriented Software Engineering)
- Data Modeling concepts (Entity Relationship Diagrams)
- Business Modeling (work flow)

Metode UML menggunakan tiga bangunan dasar untuk mendeskripsikan sistem atau perangkat lunak yang akan dikembangkan yaitu :

1. Sesuatu (Things)
    - Structural things : classes, interfaces, collaborations, use cases, active classes, components, nodes.
    - Behavioral things : interactions, state machines.
    - Grouping things : packages.
    - Annotational things : notes.
2. Relasi (Relationship)
3. Diagram

a. Things (sesuatu)

Structural things, bagian yang relatif statis dapat berupa elemen-elemen yang bersifat fisik maupun konseptual seperti :

- Kelas (*Class*) adalah himpunan dari objek-objek yang berbagi atribut serta operasi yang sama.
- Antarmuka (*Interface*) adalah kumpulan dari operasi-operasi yang menspesifikasi layanan suatu kelas atau komponen atau objek.
- Kolaborasi (*Collaboration*) yang didefinisikan dengan interaksi dan jumlah kumpulan/kelompok dari kelas-kelas/elemen-elemen yang bekerja secara bersama-sama.
- *Use case* adalah rangkaian/uraian sekelompok yang saling terkait dan membentuk sistem secara teratur yang dilakukan atau diawasi oleh sebuah aktor. ‘use case’ digunakan untuk membentuk tingkah-laku benda/ things dalam sebuah model serta direalisasikan oleh sebuah kolaborasi.
- Kelas aktif (*Active Class*) adalah kelas dimana objek yang dimilikinya memiliki satu atau lebih proses dan lebih jauh menginisialisasi suatu aktifitas kendali.
- Komponen (*Component*) adalah bagian fisik dan bagian yang dapat digantikan pada suatu sistem, dapat berupa berkas ActiveX, COM+ ataupun komponen Java Beans.
- Simpul (*Node*) merupakan fisik dari elemen-elemen yang ada pada saat dijalankannya sebuah sistem.

Behavioral things, bagian yang dinamis biasanya merupakan kata kerja dari model UML yang mencerminkan perilaku sepanjang ruang dan waktu seperti :

- Interaksi adalah suatu perilaku yang mencakup himpunan pesan-pesan yang diperlukan untuk menyelesaikan suatu fungsi tertentu. Perilaku kumpulan objek-objek atau operasi individual bisa dispesifikasikan dengan interaksi.
- State adalah perilaku yang menspesifikasi urutan kedudukan suatu objek atau interaksi-interaksi sepanjang waktu dalam menanggapi event-event yang terjadi.

Grouping things, bagian pengorganisasian dalam UML. Dalam penggambaran model UML yang rumit diperlukan penggambaran paket yang menyederhanakan model. Paket-paket ini kemudian dapat didekomposisi lebih lanjut. Paket berguna bagi pengelompokan sesuatu, misalnya model-model serta subsistem-subsistem.

An notational things, merupakan bagian yang memperjelas model UML. Dapat berisi komentar yang menjelaskan fungsi serta cirri-ciri tiap elemen dalam model UML.

b. Relasi (Relation)

Ada empat relationship (hubungan) dalam UML yaitu :

1. Ketergantungan (Dependency) adalah hubungan dimana perubahan yang terjadi pada suatu elemen independent akan mempengaruhi elemen yang bergantung padanya.
2. Asosiasi adalah apa dan bagaimana yang menghubungkan antara objek satu dengan yang lainnya. Suatu bentuk asosiasi adalah agregasi yang menampilkan hubungan suatu objek dengan bagian-bagiannya.
3. Generalisasi adalah hubungan dimana objek anak berbagi perilaku dan struktur data dari objek yang ada di atasnya (objek induk). Arah dari objek induk ke objek anak dinamakan spesialisasi sedangkan arah sebaliknya dinamakan generalisasi.

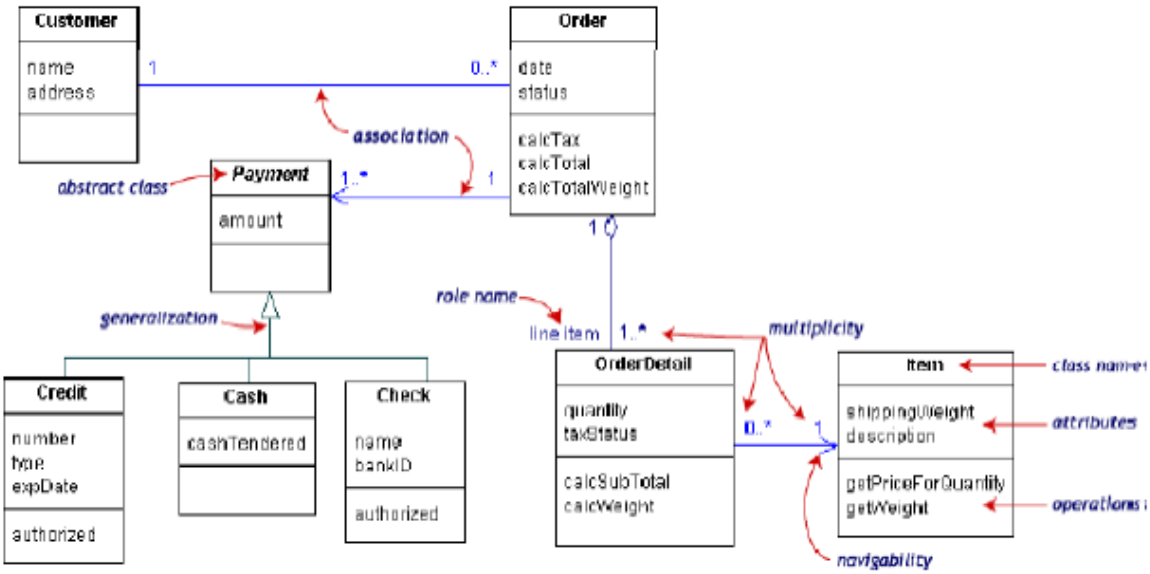
Realisasi adalah operasi yang benar-benar dilakukan oleh suat

c. Diagram

UML menyediakan beberapa jenis diagram yang dapat dikelompokkan berdasarkan sifatnya (statis atau dinamis) yaitu :

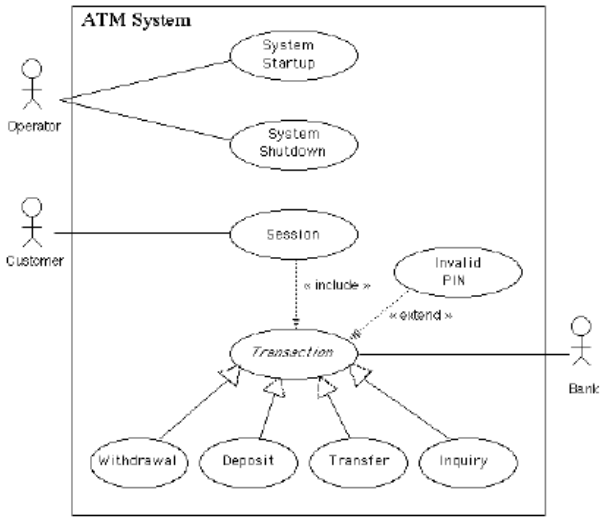
Bersifat statis :

- Class Diagram, memperlihatkan himpunan kelas-kelas, antarmuka-antarmuka, kolaborasi-kolaborasi serta relasi-relasi.



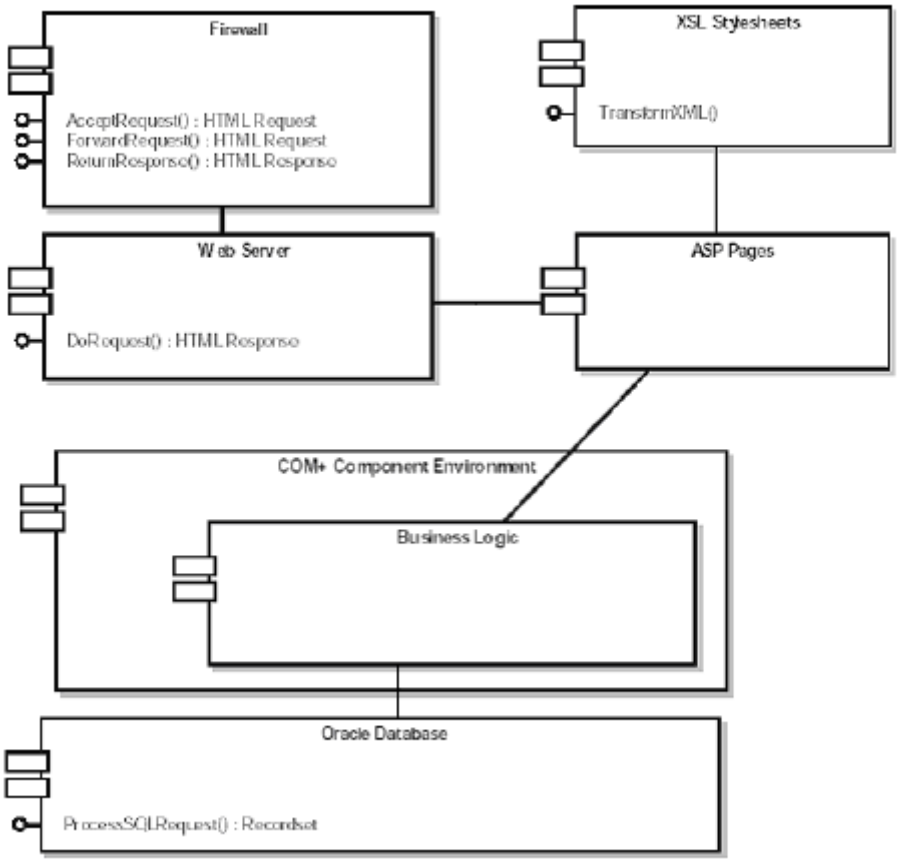
Gambar 4.14. Class Diagram

- Diagram Objek, memperlihatkan objek-objek serta relasi-relasi antar objek.
- Use Case Diagram, memperlihatkan himpunan use case dan aktor-aktor.



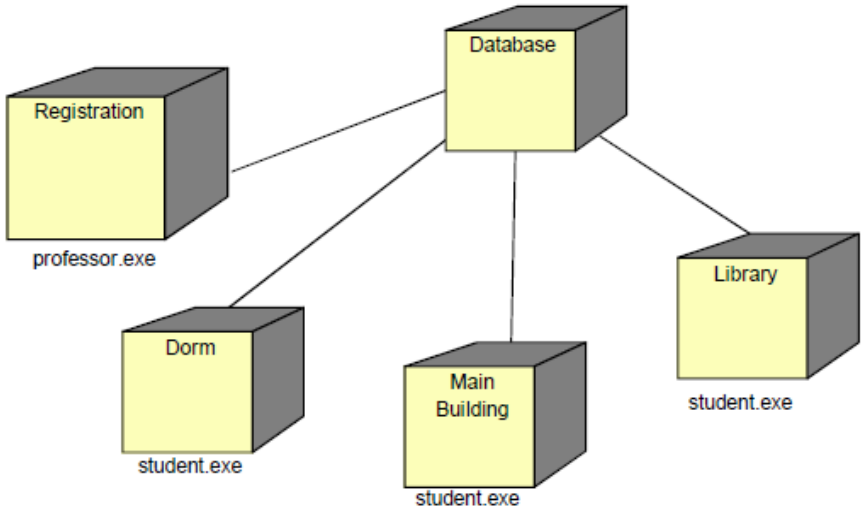
Gambar 4.15. Use Case Diagram

- Component Diagram, memperlihatkan organisasi serta ketergantungan pada komponen-komponen yang telah ada sebelumnya.



Gambar 4.16. Diagram Component

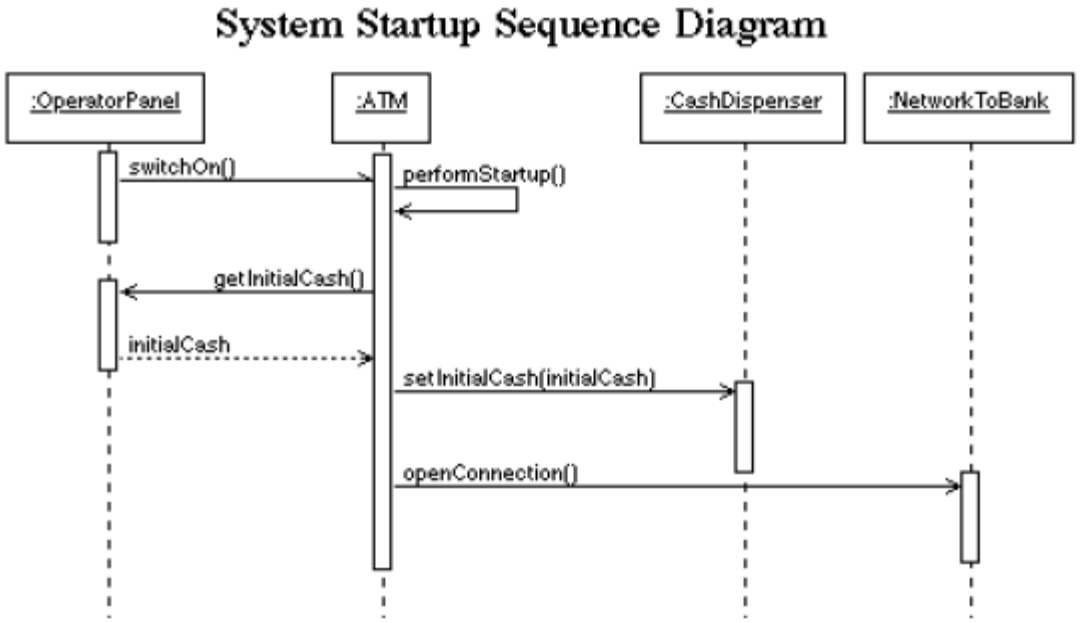
- Deployment Diagram, memperlihatkan konfigurasi saat aplikasi dijalankan.



Gambar 4.17. Deployment Diagram

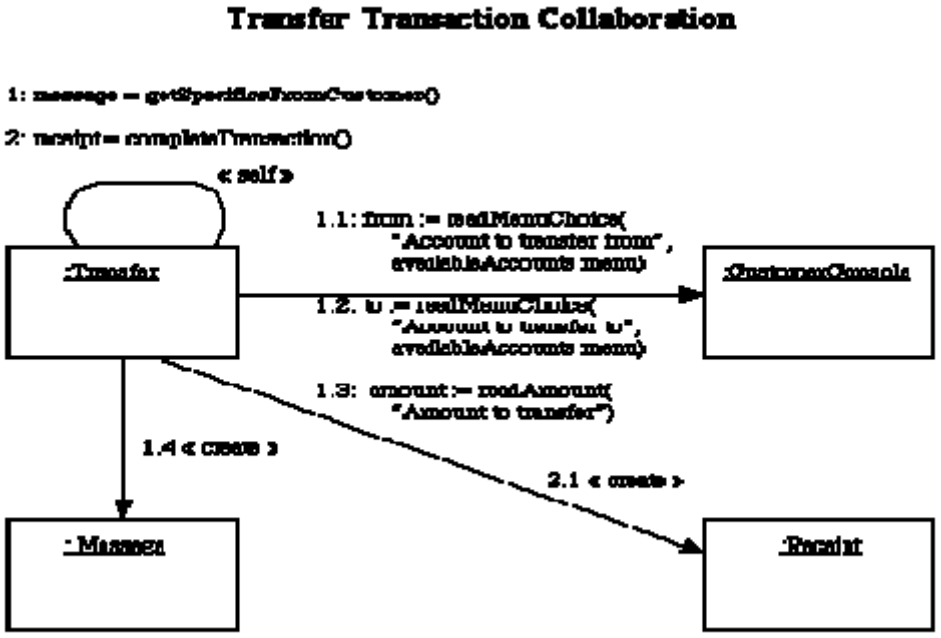
Beberapa diagram di bawah ini dianggap sebagai diagram yang bersifat dinamis

- Sequence Diagram, diagram interaksi yang menekankan pada pengiriman pesan dalam suatu waktu tertentu.



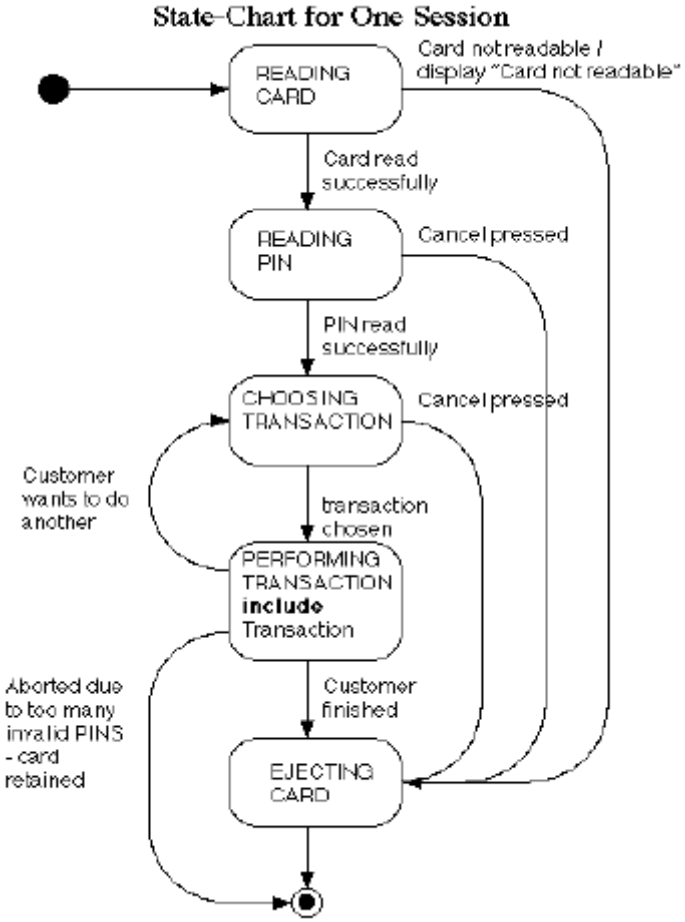
Gambar 4.18. Sequence Diagram

- Collaboration Diagram, diagram interaksi yang menekankan organisasi struktural dari objek-objek yang menerima serta mengirim pesan.



Gambar 4.19. Collaboration Diagram

- Statechart Diagram, memperlihatkan state-state pada sistem; memuat state, transisi, event serta aktifitas.



Gambar 4.20. Statechart Diagram

- Activity Diagram adalah tipe khusus dari diagram state yang memperlihatkan aliran dari suatu aktifitas ke aktifitas lainnya dalam suatu sistem.

Latihan soal :

1. Berikut merupakan keuntungan menggunakan *object – oriented*, kecuali …
   a. Reusability
   b. Modularity
   c. Maintainability
   d. Easily

2. Entitas didalam sebuah sistem perangkat lunak yang biasanya berupa benda atau suatu kejadian yaitu …
   a. Class
   b. Object
   c. Method
   d. Attribute
3. Suatu *class* dapat diturunkan dari *class* yang lain, dimana *attribute* dari *class* semula dapat diwariskan ke *class* yang baru adalah prinsip dari …
   a. Encapsulation
   b. Instance
   c. Inheritance
   d. Polymorphism
4. Prinsip penyederhanaan dari sesuatu yang kompleks dengan cara memodelkan *class* sesuai dengan masalahnya adalah …
   a. Abstraction
   b. Encapsulation
   c. Polymorphism
   d. Properties
5. Yang merupakan bentuk keterkaitan antara OOA dan OOD yaitu …
   a. Classes $\rightarrow$ Control
   b. Methods $\rightarrow$ Objects
   c. Behavior $\rightarrow$ Algorithms
   d. Attributes $\rightarrow$ Data Structures
6. Pada perangkat lunak yang akan kita kembangkan terdapat metode pemodelan visual yang menggunakan 3 bangunan dasar untuk mendeskripsikan sistem, metode tersebut yaitu …
   a. Messages, Things, Diagram
   b. Things, Relationship, Diagram
   c. Relationship, Message, Things
   d. Things, Relationship, Message

Bab 5
KONSEP DAN PRINSIP ANALISIS

Pemahaman lengkap mengenai persyaratan perangkat lunak sangat penting bagi keberhasilan usaha pengembangan perangkat lunak. Tidak peduli bagaimana perangkat lunak dirancang atau dikodekan, program yang dianalisis dan ditentukan secara tidak baik akan mengecewakan pemakainya dan akan membawa kegagalan bagi pengembangnya.
Analisis merupakan sebuah :

- Penemuan
- Perbaikan
- Pemodelan
- Spesifikasi (baru)

Pihak yang terlibat dalam Analisis Sistem

- Pengembang maupun klien harus berperan aktif
- Klien berusaha memformulasikan kembali konsep yang tidak jelas dari fungsi perangkat lunak dan kinerja kedalam detail yang konkret.
- Pengembang bertindak sebagai integrator, konsultan dan pemecah masalah
- Akuntan
- Auditor eksternal

Tujuan Analilsis Sistem

- Mendefinisikan masalah secara tepat
- Menyusun alternatif penyelesaian
- Memilih dan mempertimbangkan satu dari alternatif tersebut
- Menyusun spesifikasi logis untuk penyelesaian
- Menyusun persyaratan fisik untuk penyelesaian
- Menyusun anggaran untuk fase desain sistem pengkodean dan implementasi sistem

Tahapan awal dalam menganalisa dapat mempertanyakan beberapa pertanyaan yang dapat menjelaskan:

- Pemahaman dasar dari masalah
- Orang yang membutuhkan solusi
- Keadaan dari solusi yang diinginkan

- Efektivitas komunikasi dan kolaborasi awal antara konsumen dengan developer
- Perolehan→memperoleh kebutuhan dari semua stakeholder
- Pelanggan hanya memiliki ide yang samar-samar apa yang dibutuhkan→pengembang akan menghasilakn sesuatu yang mengacu terhadap ide yang samar-samar tersebut.
- Pelanggan akan terus mengikuti perubahan, sehingga merugikan untuk si pengembang

Spesifikasi—salah satu dari berikut ini:

- Dokumen tertulis
- Sekelompok model
- Matematika formal
- Sekumpulan scenario user (use-cases)
- Prototipe

Validasi—memeriksa mekanisme yang memuat

- Kesalahan isi atau interpretasi
- Area dimana klarifikasi dibutuhkan
- Informasi yang hilang
- inkonsistensi(masalah utama ketika produk atau sistem besar direkayasa)
- Kebutuhan yang konflik atau tidak realistis.

Manajemen Kebutuhan—mengatur kebutuhan

1. Teknik Komunikasi

Sebelum memulai semuanya maka dibutuhan sebuak teknik komunikasi untuk mendapatkan semua yang dibutuhkan, dengan cara sebagai berikut :

- Kenali stakeholder
  "who else do you think I should talk to?"
- Kenali beberapa sudut pandang
- Berusahalah menuju kolaborasi
- Pertanyaan pertama
  - Siapa di belakang permintaan atas pekerjaan ini ?
  - Siapa yang akan menggunakan solusi ini?
  - Apa keuntungan ekonomi dari solusi yang sukses ?
  - Apakah ada sumber solusi lain yang anda butuhkan?

Bagaimana cara untuk mendapatkan Requirement ?

- Pertemuan diadakan dan dihadiri baik oleh software engineer maupun konsumen
- Aturan persiapan dan partisipasi dibuat
- Agenda ditawarkan
- Seorang fasilitator (seperti konsumen, developer atau orang luar) mengendalikan pertemuan
- Mekanisme definisi digunakan(bisa berupa kertas kerja, grafik, bulletin board elektronik, forum virtual dsb

Hasil dari pertemuan-pertemuan yang sudah diagendakan sebelumnya adalah sebagai berikut :

- Menemukan permasalahan
- Mengajukan elemen-elemen solusi
- Negosiasi pendekatan yang berbeda
- Menentukan sekelompok kebutuhan solusi awal

Beberapa jenis pertemuan

a. FAST (Facilitated Application Specification Techniques)

Tujuannya adalah

- Menemukan permasalahan
- Mengajukan elemen-elemens olusi
- Negosiasi pendekatan yang berbeda
- Menentukan sekelompok kebutuhan solusi awal

Panduan dari FAST

J. Wood dan D. Silver menyarankan beberapa panduan umum FAST yang dapat digunakan yaitu :

- Peserta harus menghadiri semua rapat
- Semua peserta adalah sama
- Persiapan harus sama pentingnya dengan rapat yang sebenarnya
- Semua dokumen sebelum rapat harus dikaji ulang
- Lokasi rapat diluar ruangan terkadang diperlukan
- Tentukan agenda dan jangan sampai mengalami perubahan
- Jangan sampai terbawa dalam hal-hal teknis yang terlalu rinci

b. Penyebaran Fungsi Kualitas (Quality Function Deployment = QFD)

- Teknik manajemen kualitas yang menterjemahkan kebutuhan pelanggan ke dalam kebutuhan teknis untuk perangkat lunak
- Pertama kali diperkenalkan di Jepang untuk memaksimalkan kepuasan pelanggan
- Menekankan pemahaman tentang apa yang berguna kepada pelanggan dan kemudian menyebarkan nilai-nilai tersebut melalui proses rekayasa

Gambaran Konsep QFD

- Penyebaran fungsi menemukan “nilai” (dalam persepsi konsumen) dalam setiap fungsi yang diperlukan sistem
- Penyebaran Informasi menentukan event dan objek data
- Penyebaran Tugas memeriksa perilaku sistem
- Analisis Nilai menentukan prioritas relatif dari kebutuhan

2. Beberapa Prinsip Analisa

***Prinsip Analisa 1* :** Data Domain Model

- Menetapkan objek data
- Menggambarkan atribut data
- Menetapkan hubungan data

Prinsip Analisa 2 : Fungsi Model

- Mengidentifikasi fungsi yang (dapat) merubah objek data
- Mengindikasikan berapa data yang melalui system
- Mewakili data produsen dan konsumen

Prinsip Analisa 3 : Model Kebiasaan

- Mengindikasikan states yang berbeda dari system
- Menetapkan kejadian yang mungkin menyebabkan perubahan pada state

Prinsip Analisa 4 : Partisi Model

- Menyaring setiap model untuk mewakili level yang lebih rendah dari abstraksi
- Menyaring objek data
- Membuat hirarki fungsi
- Mewakili kebiasaan pada tingkatan yang berbeda tiap detil

: Membuat partisi horizontal dan vertikal

Prinsip Analisa 5 : Intisari :

- Memulai focus intisari masalah tanpa memperhatikan rincian implementasi

Beberapa prinsip di atas dapat

1. Mengerti masalah sebelum kita memulai menciptakan model analisa
2. Membangun protipe yang memungkinkan pelanggan untuk mengerti bagaimana pelanggan mengerti interaksi manusia dan mesin dapat terjadi
3. Mencatat hal-hal yang baru dan alasan untuk setiap kebutuhan
4. Menggunakan gambaran bertingkat setiap kebutuhan
5. Memprioritaskan kebutuhan
6. Bekerja untuk menghilangkan keragu-raguan

3. **Negosiasi Kebutuhan**

Merupakan wadah untuk mennegosiasikan segala hal yang terkait dengan apa yang akan dikerjakan, sejauh mana atau adakah perubahan-perubahan yang akan dilakukan. Di negosiasi kebutuhan ini dapat :

- Mengenali stakeholder kunci
- Orang-orang ini yang akan dilibatkan negosiasi
- Menentukan “kondisi menang” setiap stakeholder
- Kondisi kemenangan tidak selalu jelas
- Negosiasi
- Bekerja menuju sekumpulan kebutuhan yang merupakan win-win solution

4. **Validasi Requirement**

Menghasilkan apa saja yang dibutuhkan dari hasil kesepakatan yang sudah dibicarakan sebelumnya (negosisasi kebutuhan).

Validasi Requirement 1, dilihat dari beberapa pertanyaan berikut :

- Apakah setiap kebutuhan konsisten dengan tujuan keseluruhan sistem/produk?
- Apakah semua kebutuhan telah dispesifikasikan pada tingkat abstraksi yang tepat ? Apakah beberapa kebutuhan pada tingkatakan detail teknis tidak tepat pada level ini ?
- Apakah kebutuhan benar-benar diperlukan ataukan dia hanya merupakan fitur tambahan yang tidak esensial bagi tujuan sistem ?
- Apakah setiap kebutuhan terbatasi dengan baik dan tidak ambigu ?
- Apakah setiap kebutuhan mempunyai atribut ? Apakah sebuah sumber tercatat untuk setiap kebutuhan ?

- Apakah setiap kebutuhan konflik dengan kebutuhan lain ?

Validasi Requirement 2, dilihat dari beberapa pertanyaan berikut :

- Apakah setiap kebutuhan dapat diterima dalam lingkungan teknik yang menjadi rumah bagi sistem/produk?
- Apakah setiap kebutuhan dapat diuji, setelah diimplementasi ?
- Apakah model kebutuhan mencerminkan informasi, fungsi, dan perilaku sistem yang dibangun dengan baik.
- Apakah model kebutuhan telah dipartisi sedemikian sehingga menampilkan secara progresif informasi yang lebih detail tentang sistem ?
- Apakah pola kebutuhan telah digunakan untuk mempermudan model kebutuhan ? Apakah semua pola telah divalidasi? Apakah pola konsisten dengan kebutuhan konsumen ?

Analisis persyaratan adalah langkah teknis pertama pada proses rekayasa perangkat lunak. Analisis harus berfokus pada domain informasi, fungsional dan tingkah laku dari masalah. Dalam beberapa kasus tidaklah mungkin untuk secara lengkap memspesifikasi suatu masalah pada tahap awal. Spesifikasi persyaratan perangkat lunak dikembangkan sebagai akibat dari analisis.

Latihan soal :

1. Quality Function Deployment merupakan …
   a. Teknik spesifikasi aplikasi yang digunakan untuk mengumpulkan kebutuhan yang di aplikasikan selama masa awal analisis dan spesifikasi
   b. Teknik manajemen kualitas yang menterjemahkan kebutuhan pelanggan ke dalam kebutuhan teknis untuk perangkat lunak
   c. Sebuah tugas rekayasa perangkat lunak yang menjembatani jurang antara alokasi
   d. Langkah teknis pertama pada proses rekayasa perangkat lunak
2. Analisis persyaratan merupakan …
   a. Teknik spesifikasi aplikasi yang digunakan untuk mengumpulkan kebutuhan yang di aplikasikan selama masa awal analisis dan spesifikasi
   b. Teknik manajemen kualitas yang menterjemahkan kebutuhan pelanggan ke dalam kebutuhan teknis untuk perangkat lunak
   c. Sebuah tugas rekayasa perangkat lunak yang menjembatani jurang antara alokasi

d. Langkah teknis pertama pada proses rekayasa perangkat lunak

3. Berikut merupakan cara untuk mendapatkan *Requirement*, kecuali …
   a. Pertemuan diadakan dan dihadiri baik oleh software engineer maupun konsumen
   b. Aturan persiapan dan partisipasi dibuat
   c. Menentukan sekelompok kebutuhan solusi awal
   d. Seorang fasilitator (seperti konsumen, developer atau orang luar) mengendalikan pertemuan
4. Salah satu dari tujuan analisis sistem yaitu …
   a. Menyusun alternatif penyelesaian
   b. Mengajukan elemen-elemen solusi
   c. Menentukan sekelompok kebutuhan solusi awal
   d. Mengenali stakeholder kunci
5. Prinsip Analisa yang mengindikasikan states yang berbeda dari system yaitu …
   a. Data Domain Model
   b. Fungsi Model
   c. Model Kebiasaan
   d. Partisi Model
6. Prinsip Analisa yang mengindikasikan banyaknya data yang melalui system yaitu …
   a. Data Domain Model
   b. Fungsi Model
   c. Model Kebiasaan
   d. Partisi Model

Bab 6
PRINSIP & KONSEP DESAIN

Prinsip dan Konsep Desain disini memliki tujuan menghasilkan suatu model atau representasi dari entitas yang kemudian akan dibangun.

Fase Pengembangan dan Desain Perangkat Lunak

1. Fase pengembangan terdiri dari 3 langkah :
   - Design
   - Code Generation (manual or automatic)
   - Testing

   Setiap langkah melakukan transformasi informasi dalam suatu cara yang akhirnya menghasilkan software komputer yang valid.

2. Software Requirement

   Software Requirement dijelaskan dengan “Information Domain”, “Functional and Performance Requirments”, “Feed the design step”.

   Dalam software requirement ini dapat menggunakan metodelogi *data design* lebih difokuskan pada definisi dari struktur data, *architectural design* yang mendefinisikan hubungan antara elemen struktur utama dari program dan *procedural design* yang mengubah struktur elemen ke dalam prosedur software.

3. Proses Design

   Software design -> Suatu proses yang melawati serangkaian kebutuhan yang membentuk sebuah perangkat lunak

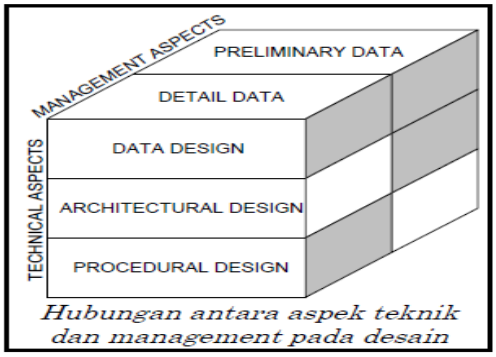
   Software design dibagi dalam 2 tahap :

   a. *Preliminary Design*

      Pada tahap ini difokuskan dengan transformasi dari keperluan / kebutuhan ke dalam data dan arsitektur software

b. Detail Design

Difokuskan pada penghalusan representasi arsitektur yang berisi struktur data detail dan algoritma untuk software



Gambar 6.1. Hubungan antara aspek teknik dan management pada desain

Kualitas Desain & Software

Beberapa tuntunan dalam melakukan agar dihasilkan desain dengan kriteria yang baik, yaitu suatu desain haruslah :

1. Memperlihatkan organisasi hirarki yang mengontrol elemen-elemen software
2. Berkenaan dengan modul. Software secara logika terbagi dalam elemenelemen yang membentuk fungsi dan sub fungsi
3. Berisi representasi yang berbeda dan terpisah dari data dan prosedur
4. Membentuk modul (contoh subroutine dan procedure) yang memperlihatkan karakteristik fungsi yang tidak saling bergantung
5. Diturunkan dengan menggunakan metode perulangan yang didukung oleh informasi yang ada selama analisa kebutuhan software

Untuk kualitas software akan dibahas di bab lain dalam modul ini.

Evolusi Desain Software

Evolusi dari desain software merupakan proses yang berkelanjutan terus selama 3 dekade. Beberapa metodologi telah tumbuh, dan secara umum memiliki karakteristik sebagai berikut :

1. Sebuah mekanisme untuk menterjemahkan representasi domain informasi ke dalam representasi desain
2. Notasi untuk merepresentasikan fungsi komponen-komponen dan interfaces-nya
3. Heuristics bagi penyaringan dan partisi
4. Petunjuk untuk penaksiran kualitas

Dasar-dasar Desain

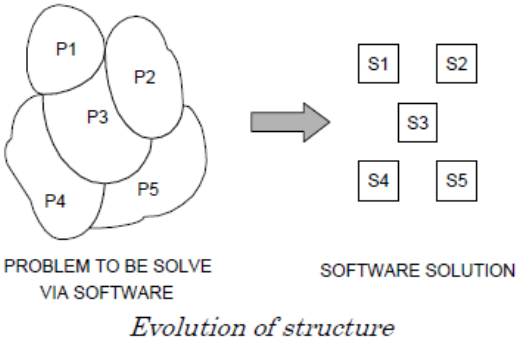
Untuk membantu software engineer dalam menjawab pertanyaan-pertanyaan berikut :

- Apakah kriteria yang dapat dipakai untuk mempartisi software menjadi sejumlah komponen ?
- Bagaimana fungsi atau struktur data dipisahkan dari suatu representasi konseptual software ?
- Apakah ada kriteria yang seragam yang menetapkan kualitas tehnik dari suatu software desain ?

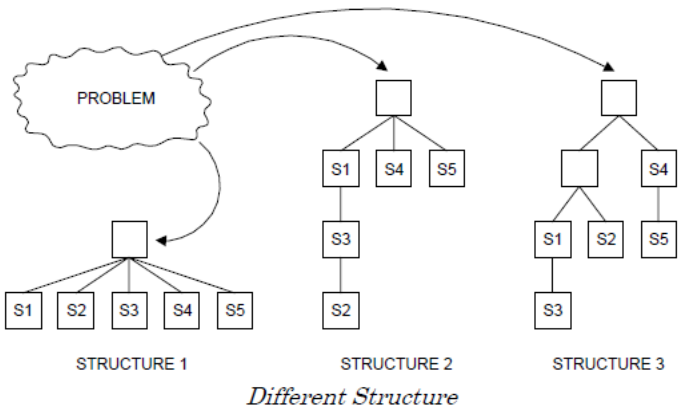
Arsitektur Software

Arsitektur perangkat lunak menyinggung 2 karakteristik penting dari sebuah program komputer :

1. Hirarki struktur dari komponen-komponen prosedural (modul)
2. Struktur data



Gambar 6.2. Evolution of Structure

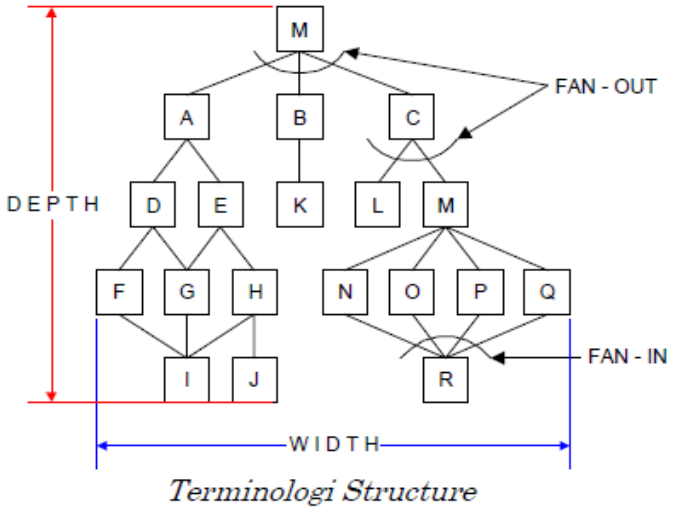


Gambar 6.3. Different Structure

Program Structure

Program structure menampilkan / menyajikan organisasi (seringkali organisasi hirarki) dari komponen-komponen program (modul-modul) dan mengandung arti hirarki dari kontrol program

Notasi yang digunakan adalah diagram tree yang biasanya dinamakan dengan structure chart.



Gambar 6.4. Terminologi Structure

Data Structure

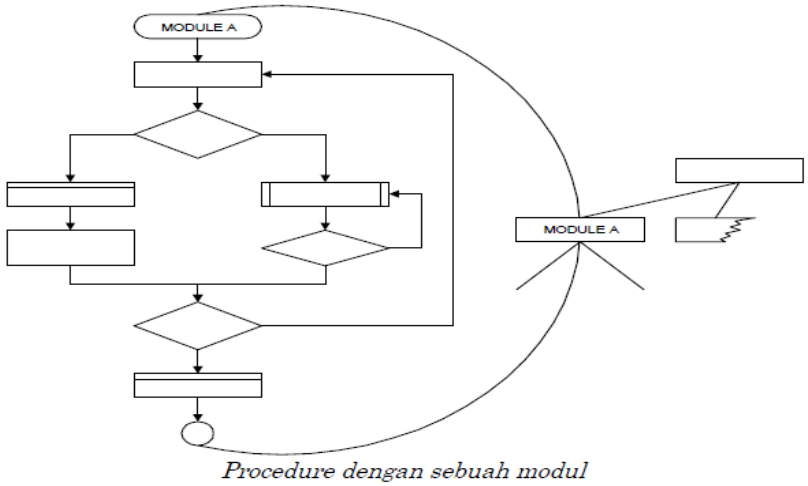
Menggambarkan relasi logikal antara sejumlah elemen..

Contoh :

```
type G = array [1..100] of integer;
...
...
Procedure S ( var T : G ; n : integer ; sum : integer );
Var
        I : integer;
begin
        sum := 0;
        for I:=1 to n do
                sum := sum + t[i];
end;
```

Software Procedure

Difokuskan pada detail pemrosesan dari setiap modul secara individu. Prosedur harus mengandung spesifikasi yang benar / tepat dari pemrosesan, termasuk : sequence of events, decision points, repetitive operations, dan struktur data.



Gambar 6.5. Procedure dalam sebuah modul

Modularitas

Software dibagi kedalam nama-nama yang terpisah dan elemen-elemen yang dapat dipanggil, yang disebut dengan modul, yang termasuk kedalam memenuhi syarat-syarat permasalahan

Misalkan :

C(x) = fungsi kompleksitas dari suatu masalah

E(x) = fungsi usaha/waktu yang diperlukan untuk memecahkan suatu masalah

P1 ,P2 = masalah 1, masalah 2

Jika : C(P1) > C(P2) maka : E(P1) > E(P2)

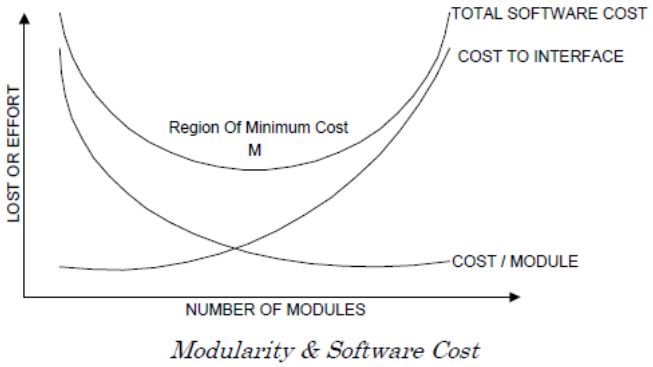
Berdasarkan penelitian :

1. C (P1 + P2) > C (P1) + C (P2)
2. E (P1 + P2) > E (P1) + E (P2)

Konklusi :

1. Kompleksitas suatu masalah gabungan P1 dan P2 akan berkurang jika masalah tersebut dipisahkan

2. Akan lebih mudah menyelesaikan suatu masalah jika dipecah / dipartisi



Gambar 6.6. Modularity dan Software Cost

Abstraksi

- Jika kita menggunakan suatu solusi modular untuk beberapa masalah, maka beberapa level / tingkat abstrasi dapat ditampilakn / diperlihatkan.
- Pada level tertinggi, suatu solusi berada pada term yang umum dengan menggunakan bahasa natural
- Level yang lebih rendah lebih berorientasi pada prosedur-prosedur

Contoh

- Abstraksi 1

The software will incorporate a computer graphics interface that will enable visual communication with the drafts person and a digitizer interface that replace the drafting board and square. All line and curve drawing, all geometric computation, and all sectioning and auxiliary views will be performed by the CAD Comp.

Contoh

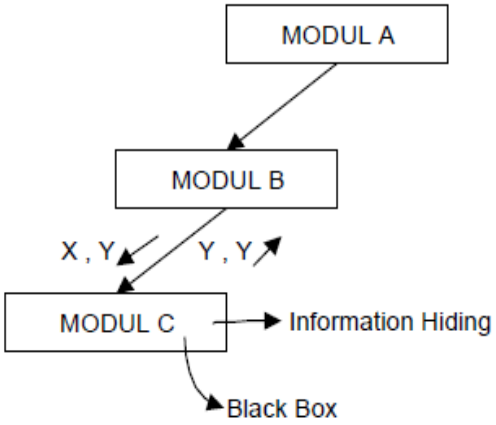
- Abstraksi 2

CAD Software tasks :

user interaction task ;
2-D drawing creation task ;
graphics display task ;
drawing file management task ;

end.

Penyembunyian Informasi



Gambar 6.7. Penyembunyian Informasi

- Black Box --> input, output, & proses dike-tahui tetapi proses detail tidak diketahui.
- Bagi Modul B, Modul C adalah Black Box

Keuntungan :

- Jika diperlukan modifikasi selama testing dan maintenance →data & prosedur disembunyikan dari bagian lain, dari program / software secara keseluruhan.
- Kesalahan-kesalahan yang terjadi selama modifikasi tidak merambat pada bagian lain.

Desain Modular Efektif

Modular design →mereduksi komplesitas masalah, menyediakan fasilitas untuk melakukan perubahan (dalam hal pemeliharaan), dan memudahkan implementasi dengan pengembangan paralel dari bagian-bagian yang berbeda dalam suatu sistem

- Module Type
  Abstraksi dan penyembunyian informasi dipakai untuk mendefinisikan modul-modul di dalam lingkungan software architecture.
- Di dalam structure software, suatu modul mungkin dikategorikan sebagai berikut :
  1. *Sequential module* yang dieksekusi tanpa interupsi yang dilakukan software aplikasi
  2. *Incremental module* yang dapat diinterupsi oleh program aplikasi dan kemudian kembali ke titik semula setelah interupsi selesai
  3. *Parallel module* yang dieksekusi secara simultan dengan modul lain dalam lingkungan Concurrent multiprocessor

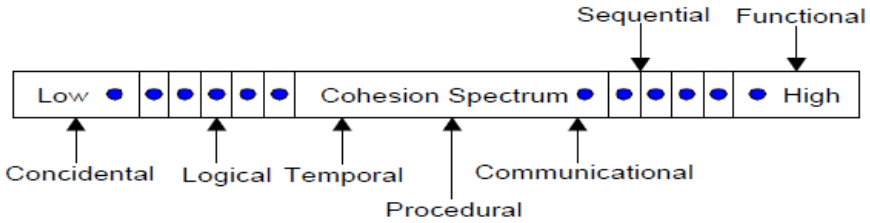
Independensi Fungsional

Konsep functional independence berkembang dari modularitas dan konsep abstraksi serta information hiding. Independence diukur dengan menggunakan 2 kriteria kualitatif, yaitu

1. Cohesion
2. Coupling

Cohesion (Keterpautan)

Perluasan / kelanjutan dari information hiding. Suatu modul kohesif membentuk sebuah tugas tunggal di dalam suatu software prosedur dan memerlukan sedikit interaksi dengan prosedur yang dibuat dalam bagian lain dari suatu program



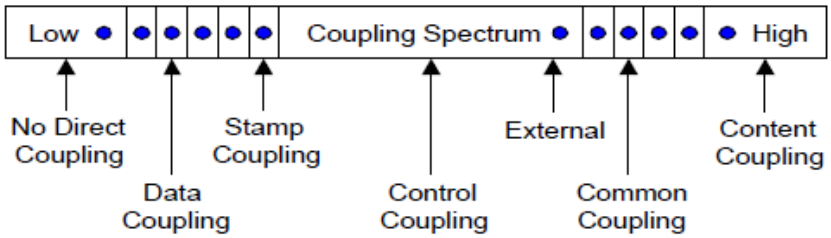
Gambar 6.8. Cohesion (Keterpaduan)

Jenis-jenis Cohesion (Keterpautan)

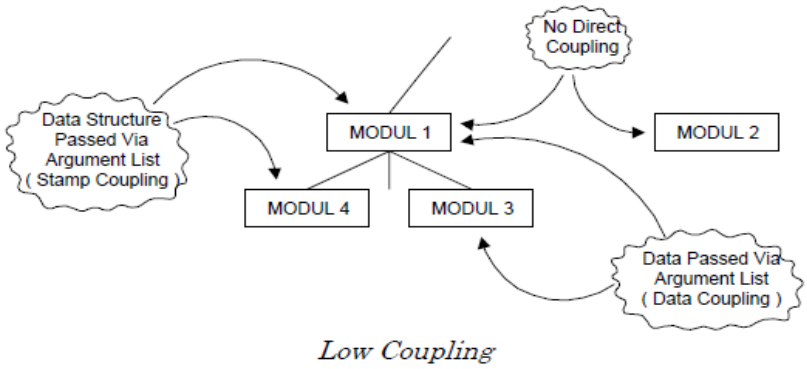
- Coincidental cohesion :
  sebuah modul yang membentuk sejumlah tugas yang berhubungan satu sama lain dengan longgar
- Logically cohesion :
  sebuah modul yang membentuk tugas-tugas yang dihubungkan secara logical
- Temporal cohesion :
  jika sebuah modul berisi sejumlah tugas yang dihubungkan dengan segala yang harus dieksekusi di dalam waktu yang bersamaan.
- Procedural cohesion :
  jika pemrosesan elemen-elemen dari suatu modul dihubungkan dan harus dieksekusi dalam urutan spesifik
- Communication cohesion :
  jika pemrosesan elemen-elemen dikonsentrasikan pada satu area dari suatu struktur data

Coupling (Bergandengan)

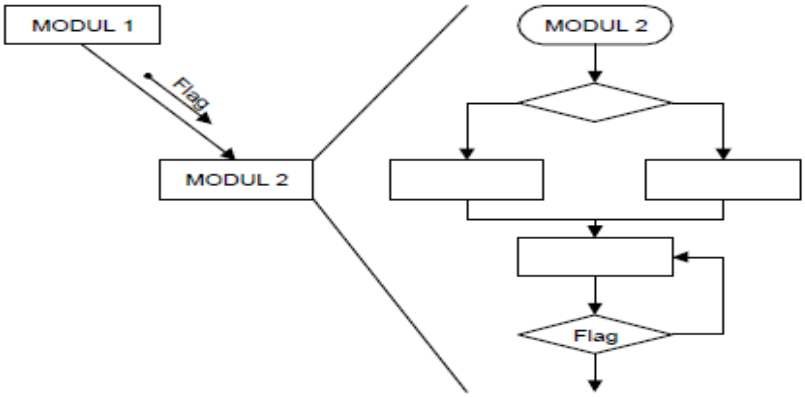
Merupakan suatu pengukuran dari keterkaitan / keterhubungan antara sejumlah modul dalam struktur program. Coupling tergantung pada kompleksitas interface antar modul



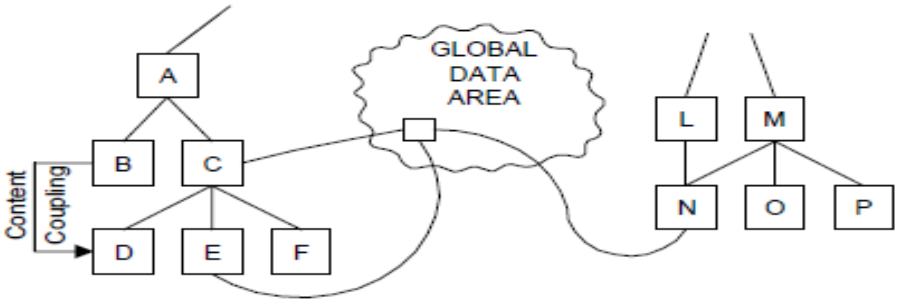
Gambar 6.9 Coupling (Bergandengan)



Gambar 6.10 Low Coupling



Gambar 6.11. Control coupling terjadi ketika modul 1 mengirimkan kontrol data ke modul2



High Coupling

Gambar 6.12 High Coupling

- Modul C, E, dan N menunjukkan common-coupling

Heuristik Design Bagi Modularitas yang Efektif

- Evaluasi “iterasi pertama” dari struktur program untuk mengurangi perangkaian dan meningkatkan kohesi.
- Usahakan meminimalkan struktur dengan fan-out yang tinggi; usahakan untuk melakukan fan-in pada saat kedalaman (depth) bertambah.
- Jagalah supaya lingkup efek dari suatu model ada dalam lingkup control.
- Evaluasi interface modul untuk mengurangi kompleksitas dan redundansi.
- Tetapkan modul-modul yang fungsinya dapat diprediksi, tetapi hindari modul yang terlalu restriktif.
- Usahakan modul-modul “entri kontrol" dengan menghindari “hubungan patalogis”
- Kemaslah software berdasarkan batasan desain dan persyaratan probabilitas.

Modul Design

Prinsip dan konsep desain yang dibicarakan pada bab ini membangun sebuah fondasi untuk pembuatan model desain yang mencakup representasi data, arsitektur, interface dan prosedur. Hal tersebut dapat dibuatkan dalam dokumentasi design berikut :

Outline spesifikasi desain :

I. Ruang Lingkup

A. Sasaran Sistem

B. Persyaratan utama software

C. Batasan-batasan dan pembatasan desain

II. Desain Data

A. Obyek data dan struktur data resultan

B. Struktur file dan database

1. Struktur file eksternal

a. struktur logis

b. deskripsi record logis

c. metode akses

2. data global

3. file dan referensi lintas data

III. Desain Arsitektural

A. Kajian data dan aliran control

B. Struktur program yang diperoleh

IV. Desain Interface

A. Spesifikasi interface manusia-mesin

B. Aturan desain interface manusia-mesin

C. Desain interface eksternal

1. Interface untuk data eksternal

2. Interface untuk sistem atau peralatan eksternal

V. Desain Prosedural

Untuk masing-masing modul :

A. Naratif pemrosesan

B. Deskripsi Interface

C. Deskripsi bahasa (atau lainnya) desain

D. Modul-modul yang digunakan

E. Struktur data internal

F. Keterangan/larangan/pembatasan

VI. Persyaratan Lintas-Referensi

VII. Ketetentuan pengujian

1. Panduan pengujian

2. Strategi integrasi

3. Pertimbangan Khusus

VIII. Catatan Khusus

IX. Lampiran

Latihan soal :

1. Fase pengembangan terdiri dari..
   a. Design, testing, Code generation
   b. Testing, code generation, design.
   c. Design, code generation, testing
   d. Testing, design, code generation
2. Software design dibagi dalam dua tahap, Preliminary Design dan Detail Design. Di bawah ini yang merupakan pengertian dari Preliminary Design adalah..
   a. Suatu proses yang melawati serangkaian kebutuhan yang membentuk sebuah perangkat lunak
   b. Difokuskan dengan transformasi dari keperluan / kebutuhan ke dalam data dan arsitektur software
   c. Penghalusan representasi arsitektur yang berisi struktur data detail dan algoritma untuk software
   d. Proses yang berkelanjutan terus selama 3 dekade.
3. Salah satu tuntunan untuk menghasilkan design dnegan kriteria yang baik adalah..
   a. Membentuk modul yang memperlihatkan karakteristik fungsi yang tidak saling bergantung
   b. Menerjemahkan representasi domain informasi ke dalam representasi desain
   c. Penghalusan representasi arsitektur yang berisi struktur data detail dan algoritma untuk software
   d. Hirarki struktur dari komponen-komponen procedural
4. Notasi yang biasa digunakan dalam program structure adalah..
   a. Diagram tree
   b. Diagram activity
   c. Diagram use case
   d. Diagram sequence
5. Berikut ini yang bukan termasuk dari modul structure software adalah..
   a. Incremental module
   b. Sequential module
   c. Tree module
   d. Parallel module
6. Sebuah modul yang membentuk tugas-tugas yang dihubungkan secara logical termasuk dalam cohesion jenis..
   a. Coincidental cohesion

b. Procedural cohesion
c. Temporal cohesion
d. Logically cohesion

7. Heuristik Design Bagi Modularitas yang Efektifa adalah, kecuali..
   a. Evaluasi interface modul untuk mengurangi kompleksitas dan redundansi.
   b. Tetapkan modul-modul yang fungsinya dapat diprediksi, tetapi hindari modul yang terlalu restriktif.
   c. Evaluasi “iterasi pertama” dari struktur program untuk mengurangi perangkaian dan meningkatkan kohesi.
   d. Akan lebih mudah menyelesaikan suatu masalah jika dipecah / dipartisi
8. Mereduksi komplesitas masalah, menyediakan fasilitas untuk melakukan perubahan dan memudahkan implementasi dengan pengembangan paralel dari bagian-bagian yang berbeda dalam suatu system adalah pengertian dari..
   a. Modular design
   b. Modular system
   c. Modular type
   d. Modular parallel
9. Yang bukan termasuk dari Prosedur yang mengandung spesifikasi benar / tepat dari pemrosesan adalah..
   a. sequence of events
   b. repetitive operations
   c. Information Domain
   d. struktur data
10. Yang bukan termasuk jenis-jenis cohesion adalah..
   a. Procedural cohesion
   b. Sequence cohesion
   c. Coincidental cohesion
   d. Communication cohesion

Bab 7
Metode Desain

Kenapa Arsitektur ?

Arsitektur bukanlah operasional perangkat lunak, namun merupakan representasi yang memungkinkan pengembang PL untuk :

a. Menganalisa efektivitas desain dalam memenuhi kebutuhan,
b. Mengetahui alternatif2x arsitektur pada keadaan dimana membuat perubahan desain masih relatif lebih mudah, dan
c. Mengurangi resiko terkait dengan konstruksi PL.

Arsitektur dianggap penting karena merupakan :

- Representasi dari arsitektur PL adalah enabler bagi komunikasi antar pihak(stakeholder) yang tertarik dengan pengembangan sistem berbasisk omputer.
- Arsitektur menyoroti keputusan desain awal yang akan mempunyai pengaruh yang sangat besar pada pekerjaan RPL yang mengikutinya, dan keberhasilan pada entitas sistem operasional.
- Arsitektur membangun model yang relatif kecil dan mudah digenggam secara intelektual tentang bagaimana sistem distrukturkan dan bagaimana komponen-komponen yang bekerjasama.

Desain Data

Dilihat dari desain data untuk level arsitektur dan level komponen dapat diurakan di bawah ini :

Pada level arsitektur

- *D*esain satu atau lebihd atabase untuk mendukung arsitektur aplikasi.
- Desain method untuk„mining‟ isi dari berbagai database.
- Navigasi melalui database-database yang ada dalam usaha untuk mengambilin formasi level bisnis yang sesuai.
- Desain sebuah data warehouse—sebuah database besar, independen yang mempunyai akses pada data yang disipan dalam database yang melayani sekelompok aplikasi yang dibutuhkan bisnis

Pada level komponen

- Mengambil objek-objek data dan mengembangkan satus et abstraksi data
- Implementasi atribut-atribut objek data sebagai satu atau lebih struktur data
- Review struktur data untuk memastikan bahwa relasi yang tepat sudah dibuat
- Sederhanakan struktur data sesuai dengan kebutuhan

Desain Data – Level Komponen

1. Prinsip-prinsip analisis semantik yang diterapkan pada fungsi dan perilaku harus juga dapat berjalan pada data.
2. Seluruh struktur data dan operasi yang akan dilakukan harus dapat diidentifikasi.
3. Sebuah data dictionary harus dibuat dan digunakan untuk menentukan desain program dan data.
4. Keputusan desain data level rendah harus ditunda hingga akhir proses desain.
5. Representasi struktur dara harus diketahui oleh modul yang menggunakannya langsung dalam struktur tersebut (enkapsulasi).
6. Sebuah pustaka struktur data dan operasi yang memungkinkan untuk diterapkan harus dikembangkan.
7. Desain PL dan bahasa pemrograman harus mendukung spesifikasi dan realisasi dari tipe data abstrak.

Gaya Arsitektur

Masing-masing menggambarkan kategori sistem yang menunjukkan:

1. Sekumpulan komponen(mis database, modul komputasi) yang menunjukkan fungsi yang dibutuhkan sistem,
2. Sekumpulan connector yang memungkinkan komunikasi, koordinasi dan kerjasama antar komponen,
3. Batasan yang menentukan bagaimana komponen dapat diintegrasikan untuk membentuk sistem,
4. Model semantikyang memungkinkan desainer untuk memahami properti keseluruhan dari sistem dengan menganlisa properti dalam bagian2x didalamnya.

- Data-centered architectures
- Data flow architectures
- Call and return architectures
- Object-oriented architectures
- Layered architectures

Analisis Desain Arsitektur

1. Kumpulkan semua skenario.
2. Dapatkan kebutuhan-kebutuhan, batasan-batasan, dan gambaran lingkungan.
3. Gambarkan pola/gaya arsitektur yang telah dipilih untuk menangani scenario-skenario dan kebutuhan-kebutuhan :
    - module view
    - process view
    - data flow view
4. Evaluasi kualitas atribut-atribut dengan melihat setiap atribut dalam isolasi.
5. Kenali kualitas atribut untuk setiap atribut arsitektur untuk masing-masing gaya arsitektur yang spesifik.
6. Lakukkan kritik pada arsitektur-arsitektur kandidat(yg dikembangkan pada langkah 3 menggunakan analisis pada langkah 5.

Arsitektur Terpartisi

Ada 2 partisi Arsitektur, yaitu partisi vertical dan partisi horizontal.

Mengapa dipartisi ??

- Hasilnya adalah PL yang mudah diuji
- Membawa kepada PL yang lebih mudah dikelola
- Hasilnya efek samping yang semakin sedikit
- Hasilnya adalah PL yang lebih mudah dikembangkan

Desain Terstruktur

Tujuan desain terstruktur adalah untuk mendapatkan arsitektur program yang terpartisi dengan pendekatan : DFD dipetakan ke arsitektur program, PSPEC dan STD digunakan untuk mengindikasikan setiap modul serta notasi (diagram struktur).

Latihan soal :

1. Arsitektur merupakan representasi yang memungkinkan pengembang software untuk?
    a. Menjadikan software yang kurang baik menjadi lebih baik.
    b. Menganalisa efektivitas desain dalam memenuhi kebutuhan.
    c. Mengembangkan suatu software yang telah dibuat.
    d. Menganalisa cara kerja suatu software.

2. Arsitektur terpatisi ada 2 , yaitu?
   a. Partisi Terstruktur dan Tidak Terstruktur.
   b. Partisi Dinamis dan Statis.
   c. Partisi Sekuensial dan Kombinasional.
   d. Partisi Vertikal dan Horizontal.
3. Pada analisis desain arsitektur, gambar pola/gaya arsitektur yang dibutuhkan adalah?
   a. Module view, Process view, dan Prototype view.
   b. Prototype view, Data Flow view, dan Process view.
   c. Module view, Data Flow view, dan Prototype view.
   d. Module view, Data Flow view, dan Process view.
4. Berikut yang membuat Arsitektur dianggap penting, kecuali?
   a. Arsitektur membangun model yang relatif kecil dan mudah digenggam secara intelektual tentang bagaimana sistem distrukturkan dan bagaimana komponen-komponen yang bekerjasama.
   b. Representasi dari arsitektur software adalah enabler bagi komunikasi antar pihak(stakeholder) yang tertarik dengan pengembangan sistem berbasis komputer.
   c.. Arsitektur berperan dalam pengembangan suatu software karena untuk mengembangkan software membutuhkan arsitektur yang baik.
   d. Arsitektur menyoroti keputusan desain awal yang akan mempunyai pengaruh yang sangat besar pada pekerjaan RPL yang mengikutinya, dan keberhasilan pada entitas sistem operasional.
5. Berikut ini, mana yang bukan dari gaya arsitektur?
   a. Sebuah pustaka struktur data dan operasi yang memungkinkan untuk diterapkan harus dikembangkan.
   b. Model semantik yang memungkinkan desainer untuk memahami properti keseluruhan dari sistem dengan menganlisa properti dalam bagian-bagian di dalamnya.
   c. Sekumpulan connector yang memungkinkan komunikasi, koordinasi dan kerjasama antar komponen.
   d. Batasan yang menentukan bagaimana komponen dapat diintegrasikan untuk membentuk sistem.

Bab 8
Teknik Pengujian Perangkat Lunak

Pengujian Perangkat Lunak

Setelah desain selesai biasanya aan dilakukan pengujian terhadap produk yang dibuat, untuk mengetahui apakah sudah sesuai dengan keinginan customer atau sudahkah produk berjalan dengan benar dan baik ? Pengujian program harus dilakukan pertama kali oleh pemrogram itu sendiri, setelah itu baru diserahkan dan dilakukan pengujian oleh penguji *(tester)* program. Umumnya, pengujian yang dilakukan oleh pengembang dilakukan pada tahap akhir dari proses pengembangan, setelah semua modul aplikasi dikembangkan.

Persiapan Pengujian Perangkat Lunak

Pengujian *software (software testing)* membutuhkan persiapan, sebelum pengujian dilakukan. Mengapa? Karena proses testing harus dilakukan secara sistematis, tidak bisa secara sembarang, karena *software* yang dihasilkan harus bebas dari *error*, untuk mengurangi resiko kerugian yang akan diderita oleh penggunanya. Produk *software* harus menguntungkan penggunanya pada saat digunakan.

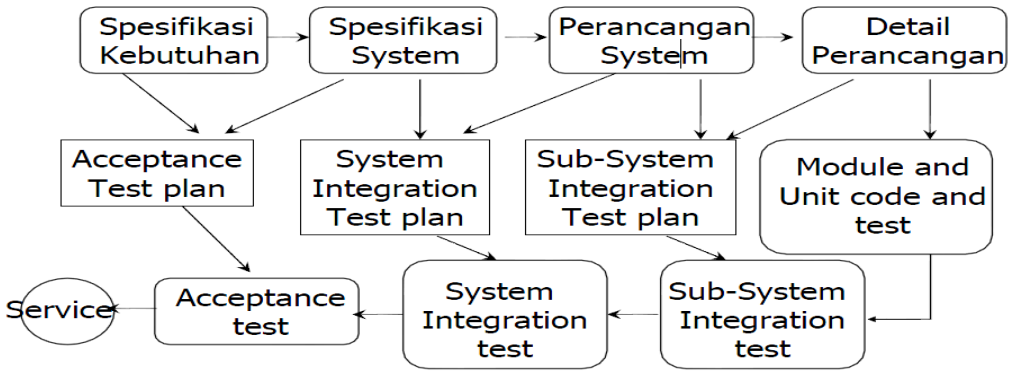
Untuk itu sebelum melakukan pengujian maka dibuatlah rencana untuk pengujian tersebut. Rencana pengujian dapat dilihat di bawah ini :

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

Berikut persiapan yang dapat dilakukan untuk dapat melakukan proses testing:

- membuat *checklist*
    - *list* yang akan ditest
    - *list requirement*
    - *list* rancangan
    - *list* spesifikasi
    - *list* manual, jika sudah ada - biasanya diperlukan untuk pengujian oleh *user*
- pembuatan *test case*
    - merupakan elemen dasar yang harus ditesting
    - merupakan *list* yang *independent*
- pembuatan grup *test case*
    - kumpulan dari beberapa *test case*
    - merupakan *list* yang akan memiliki status hasil test
- pembuatan modul test
    - pembuatan skenario *testing*
    - terdiri atas beberapa grup *test case*
    - diasosiasikan dengan fungsionalitas modul
    - mengacu kepada dokumen *requirement* dan desain/*spec* program
- pembuatan *package testing*
- pembuatan produk test

Dengan dimilikinya *checklist*, dapat mengetahui progress dari kegiatan testing itu sendiri. Mana yang sudah selesai dilakukan test, mana yang belum. Mana yang sudah dilakukan test pun, dapat diketahui mana yang benar modulnya sudah selesai, dan mana yang belum selesai. Jadi tidak sekedar mengetahui mana yang sudah dan mana yang belum.



Gambar 8.1. Hubungan Rencana Pengujian dan Proses Pengembangan Sistem

User Acceptance Test (UAT) atau Uji Penerimaan Pengguna adalah suatu proses pengujian oleh pengguna yang dimaksudkan untuk menghasilkan dokumen yang dijadikan bukti bahwa software yang telah dikembangkan telah dapat diterima oleh pengguna, apabila hasil pengujian *(testing)* sudah bisa dianggap memenuhi kebutuhan dari pengguna. Hasil dari UAT adalah dokumen yang menunjukkan bukti pengujian, berdasarkan bukti pengujian inilah dapat diambil kesimpulan, apakah software yang diuji telah dapat diterima atau tidak.

Bahan untuk pengujian harus disiapkan oleh perancang aplikasi atau pengguna. Bahan untuk pengujian suatu modul program akan terdiri atas banyak data dan prosedur. Setiap data dan prosedur disebut sebagai kasus uji *(test case)*.

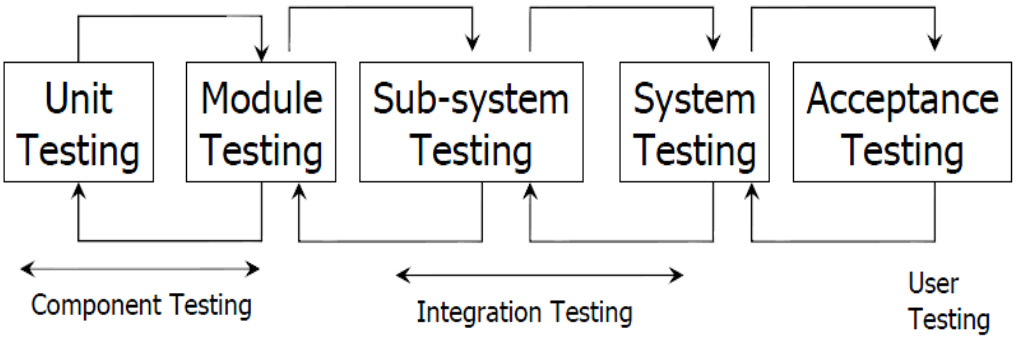
Satu modul akan memiliki banyak kasus uji. Mengapa? Karena di dalam suatu program atau modul, pada umumnya, di dalamnya akan terdapat lebih dari satu prosedur atau fungsi. Setiap prosedur dan fungsi akan memiliki kegunaan sendiri, maka sudah seharusnya setiap fungsi atau prosedur harus diuji. Pengujian pada level prosedur atau fungsi disebut sebagai pengujian pada level unit.

Aspek pengujian meliputi:

- pemeriksaan *error*, masih ada atau tidak
- pemeriksaan apakah *software* telah sesuai dengan *requirement* atau belum

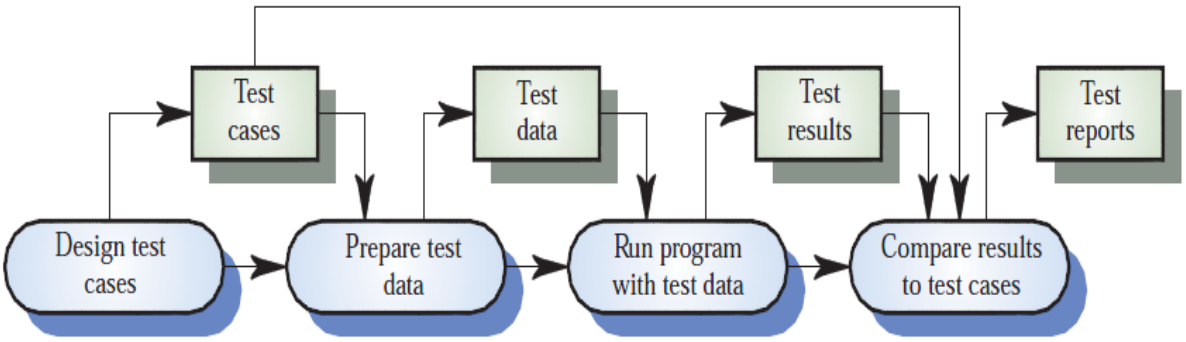
Untuk dapat melakukan pengujian dengan benar dan dapat dipertanggungjawabkan, maka harus disusun skenario untuk pengujian terlebih dahulu. Skenario pengujian adalah dokumen yang berisi persiapan dan langkah-langkah yang harus dilakukan untuk menguji *software* secara terinci. Langkah-langkah ini menunjukkan aspek-aspek *software* yang harus diuji.

Proses Testing



Gambar 8.2. Proses Testing

Proses Defect Testing



Gambar 8.3. Proses Defet Testing

Prioritas Testing

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

Test Data dan Kasus Test

Test data adalah input yang yang direncankan digunakan oleh sistem, sedangan Test cases: Input yang digunakan untuk menguji sistem dan memprediksi output dari input jika sistem beroperasi sesuai dengan spesifikasi.

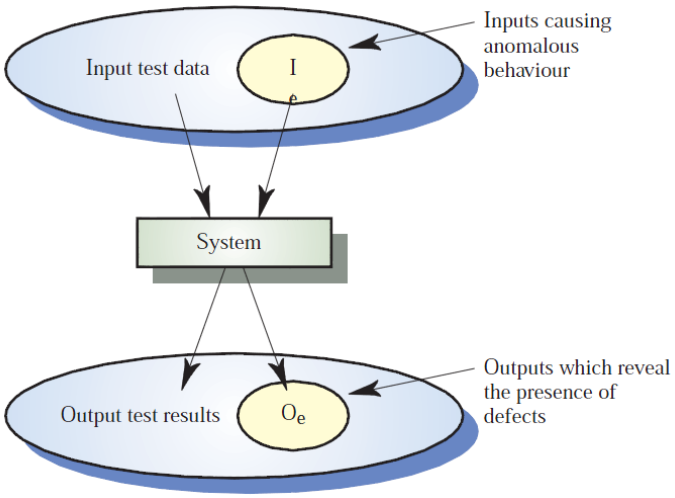
1. Component Testing

Component testing merupakan pengujian komponen-komponen program, yang biasanya dilakukan oleh component developer (kecuali untuk system kritis).

Terdiri dari :

1. Unit Testing digunakan untuk menguji setiap modul untuk menjamin setiap modul menjalankan fungsinya dengan baik. Metode yang dilakukan : Black Box Testing dan White – Box Testing.
2. Module Testing digunakan untuk mengecek apakah struktur kendali sudah memetakan kinerja kesuluruhan modul secara tepat.

a. Black Box Testing



Gambar 8.4. Black Box Testing

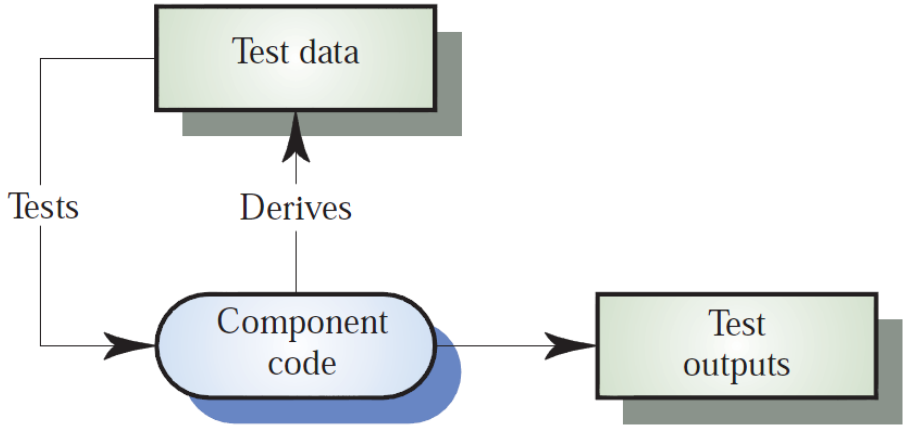
Pengujian dengan black box ini lebih terfokus pada apakah unit program memenuhi kebutuhan (requirement) yang disebutkan dalam spesifikasi.

Pada pengujian black box juga merupakan cara pengujian yang hanya dilakukan dengan menjalankan atau mengeksekusi unit atau modul kemudian diamati apakah hasil dari unit itu sesuai dengan proses bisnis yang diinginkan. Jika ada unit yang tidak sesuai outputnya maka untuk menyelesaikannya, diteruskan pada pengujian yang kedua yaitu White – Box Testing.

Dalam pengujian black box berusaha menemukan kesalahan dalam kategori :

- Fungsi-fungsi yang tidak benar atau hilang
- Kesalahan interface
- Kesalahan dalam struktur data atau akses database eksternal
- Kesalahan kinerja
- Inisialisasi dan kesalahan terminasi

b. White – Box Testing



Gambar 8.5. White Box Testing

Pengujian white box merupakan cara pengujian dengan melihat ke dalam modul untuk meneliti kode – kode program yang ada, dan menganalisis apakah ada kesalahan atau tidak. Jika ada modul yang menghasilkan output yang tidak sesuai dengan proses bisnis yang dilakukan maka baris – baris program, variabel dan parameter yang terlibat pada unit tersebut akan dicek satu persatu dan diperbaiki kemudian di-compile ulang.

c. Path Testing

Pengujian Path memiliki tujuan untuk meyakinkan bahwa himpunan test case akan menguji setiap path pada suatu program paling sedikit satu kali. Dimana titik awal untuk penhujian path adalah suatu program flow graph yang menunjukkan node – node yang menyatakan program decisions (misalnya.: if-then-else condition) dan busur menyatakan alur control,, serta statements dengan conditions adalah node – node dalam flow graf.

Program Flow Graph

Filow Graph menggambarkan alur kontrol. Setiap cabang ditunjukkan oleh path yg terpisah dan loop ditunjukkan oleh arrows looping kembali ke loop kondisi node. Digunakan sebagai basis untuk menghitung cyclomatic complexity. Rumus dari cyclomatic complexity adalah sebagai berikut :

Cyclomatic complexity = Jumlah edges – Jumlah Node +2

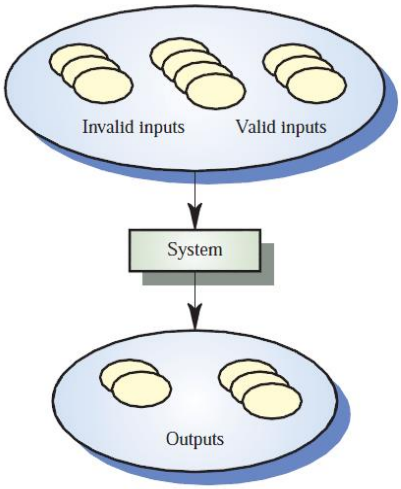
Dimana *cyclomatic complexity* menyatakan jumlah test untuk menguji control statements.

Independent Path

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

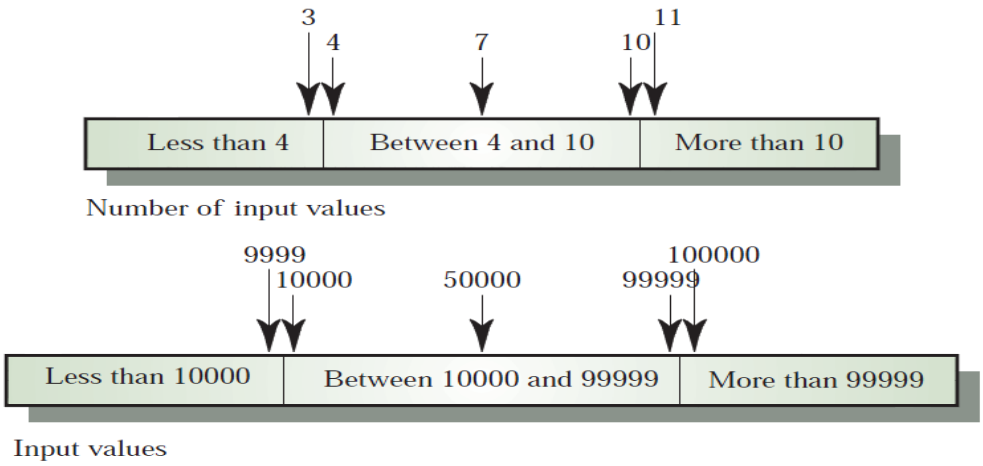
d. Partisi Ekivalensi (Equivalensi Partition)

Partisi Ekivalensi merupakan input data dan output hasil terdapat di klas yang berbeda yang sesuai dengan klas inputnya, dimana masing-masing klas equivalensi partition diprosres dengan program yang akan memproses anggota klas-klas tersebut secara equivale. Sedangkan untuk test cases dipilih dari masing-masing partisi.



Gambar 8.6. Partisi Ekivalensi

- Partition system inputs and outputs into ‘equivalence sets’
- If input is a 5-digit integer between 10000 and 99999, equivalence partitions are <10000, 10000-99999 and > 100000
- Choose test cases at the boundary of these sets
- 00000, 9999, 10000, 99999, 100001



Gambar 8.7. Input Partisi Ekivalensi

2. Integration Testing

Pengujian kelompok komponen-komponen yang terintegrasi untuk membentuk sub-system ataupun system. Dilakukan oleh tim penguji yang independent. Pengujian ini berdasarkan spesifikasi sistem. Pengujian terintegrasi terdiri dari serangkaian test sebagai berikut :

1. Ujicoba antarmuka : ujicoba setiap fungsi dari antarmuka.
2. Ujicoba skenario pengguna : Pastikan setiap skenario berjalan dengan baik.
3. Ujicoba aliran data : uji setiap proses dalam langkah per langkah.
4. Ujicoba sistem antarmuka : pastikan data mengalir antar proses.

System Testing merupakan pengujian terhadap integrasi sub-system, yaitu keterhubungan antar sub-system

3. User Testing

Pengujian user biasanya dikenal dengan istilah Acceptance Testing., merupakan pengujian terakhir sebelum sistem dipakai oleh user. Dimana melibatkan pengujian dengan data dari pengguna sistem yang biasa dikenal sebagai “alpha test” (“beta test” untuk software komersial, dimana pengujian dilakukan oleh potensial customer).

Dalam user testing seringkali ditemukan istilah-istilah seperti di bawah ini :

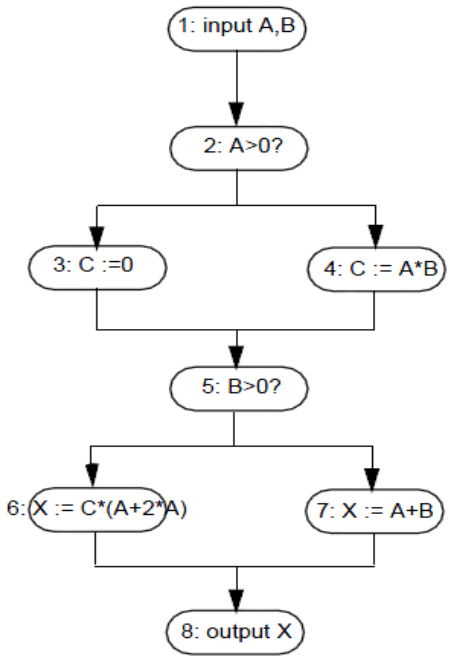
- *Fault*: kesalahan dalam source code yang mungkin menimbulkan failure ketika code yang fault tersebut dijalankan.
- *Error* : kesalahan dalam logika yang mungkin menimbulkan failure ketika program sedang dijalankan.

- *Failure*: output yang tidak benar/tidak sesuai ketika sistem dijalankan.

Tabel 8.1 Failure Class

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

Contoh :



- Suppose node 6 should be X:= C*(A+2*B)
  - Failure-less fault:
    - » executing path (1,2,4,5,7,8) will not reveal this fault because 6 is not executed
    - » nor will executing path (1,2,3,5,6,8) because C = 0
- Need to make sure proper test cases are selected
  - the definitions of C at nodes 3 and 4 both affect the use of C at node 6
    - » executing path (1,2,4,5,6,8) will reveal the failure, but only if B /= 0

Gambar 8.8. Contoh Fault, Error dan Failure

Latihan soal :

1. Output yang tidak benar/ tidak sesuai ketika sistem dijalankan, merupakan pengertian dari?
   a. Error.
   b. Failure.

c. Fault.

d. Incorrect.

2. Pengujian yang hanya dilakukan dengan menjalankan atau mengeksekusi unit atau modul kemudian diamati apakah hasilnya sesuai proses bisnis yang diinginkan, merupakan pengertian dari?

a. Black Box Testing.

b. Partisi Ekivalensi.

c. Path Testing.

d. White Box Testing.

3. Pengujian kelompok komponen-komponen yang membaur untuk membentuk sub-system ataupun system, merupakan pengertian dari?

a. User Testing.

b. Component Testing.

c. Path Testing.

d. Integration Testing.

4. Pada path testing, terdapat program flow graph yang menggambarkan alur kontrol, digunakan sebagai basis untuk menghitung Cyclomatic Complexity. Rumus dari Cyclomatic Complexity adalah?

a. Cyclomatic complexity = Jumlah edges – Jumlah Node +1

b. Cyclomatic complexity = Jumlah edges – Jumlah Node +2

c. Cyclomatic complexity = Jumlah Node – Jumlah edges +1

d. Cyclomatic complexity = Jumlah Node – Jumlah edges +2

5. Pada Componen Testing, terdiri dari 2 jenis yaitu?

a. Program Testing dan System Testing.

b. Unit Testing dan Module Testing.

c. Unit Testing dan System Testing.

d. Module Testing dan System Testing.

6. Kenapa dalam satu modul akan memiliki banyak kasus uji?

a. Karena di dalam suatu program atau modul, pada umumnya, di dalamnya akan terdapat lebih dari satu prosedur atau fungsi.

b. Karena di dalam suatu program atau modul, pada umumnya, di dalamnya akan terdapat lebih dari satu Objek atau variabel utama.

c. Karena di dalam suatu program atau modul, pada umumnya, di dalamnya akan terdapat lebih dari satu method atau fungsi.

d. Karena di dalam suatu program atau modul, pada umumnya, di dalamnya akan terdapat lebih dari satu class atau superclass.

Bab 9
STRATEGI PENGUJIAN PERANGKAT LUNAK

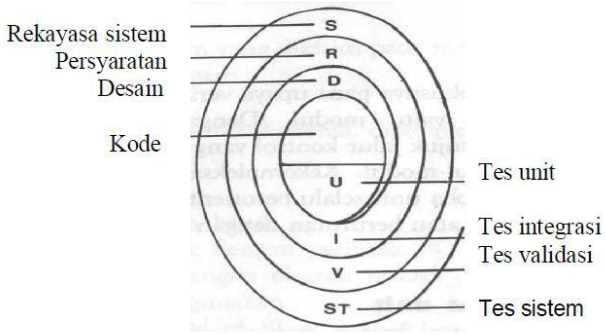
Pendekatan Strategis ke Pengujian Perangkat Lunak

Pengujian PL adalah elemen kritis dari jaminan kualitas PL dan mepresentasikan spesifikasi, desain dan pengkodean. Dalam melakukan uji coba ada 2 masalah penting yang akan dibahas, yaitu :

- Teknik uji coba PL.
- Strategi uji coba PL.

Teknik Uji Coba Perangkat Lunak

Pada dasarnya, pengujian merupakan suatu proses rekayasa perangkat lunak yang dapat dianggap (secara psikologis) sebagai hal yang destruktif daripada konstruktif.



Gambar 9.1. Siklus Rekayasa Perangat Lunak

Sasaran Pengujian (Glen Myers)

Menurut Glen Myers, pengujian adalah proses eksekusi suatu program dengan maksud menemukan kesalahan. Test case yang baik adalah test case yang memiliki probabilitas tinggi untuk menemukan kesalahan yang belum pernah ditemukan sebelumnya. Pengujian yang sukses adalah pengujian yang mengungkap semua kesalahan yang belum pernah ditemukan sebelumnya.

Prinsip Pengujian (Davis)

Menurut Davis, pengujian :

- Semua pengujian harus dapat ditelusuri sampai ke persyaratan pelanggan.

- Pengujian harus direncanakan lama sebelum pengujian itu dimulai.
- Prinsip Paretoberlaku untuk pengujian PL.
- Pengujian harus mulai “dari yang kecil”dan berkembang ke pengujian “yang besar”.
- Pengujian yang mendalam tidak mungkin.
- Paling efektif, pengujian dilakukan oleh pihak ketiga yang independen.

Strategi Pengujian Perangkat Lunak

Strategi uji coba mempunyai karaktiristik sebagai berikut :

- Pengujian mulai pada tingkat modul yang paling bawah, dilanjutkan dengan modul di atasnya kemudian hasilnya dipadukan.
- Teknik pengujian yang berbeda mungkin menghasilkan sedikit perbedaan (dalam hal waktu).
- Pengujian dilakukan oleh pengembang PL dan (untuk proyek yang besar) suatu kelompok pengujian yang independen.
- Pengujian dan debugging merupakan aktivitas yang berbeda, tetapi debugging termasuk dalam strategi pengujian.
- Point tambahan, pengujian PL adalah satu elemen dari 84ontr yang lebih luas yang sering diacu sebagai *verifikasi* dan *validasi* (V & V).
    - Verifikasi : Kumpulan aktifitas yang menjamin penerapan PL benar-benar sesuai dengan fungsinya.
    - Validasi :Kumpulan aktifitas yang berbeda yang memastikan bahwa PL yang dibangun dapat memenuhi keperluan pelanggan.

Pengujian Unit

Unit testing (uji coba unit) fokusnya pada usaha verifikasi pada unit terkecil dari desain PL, yakni modul.

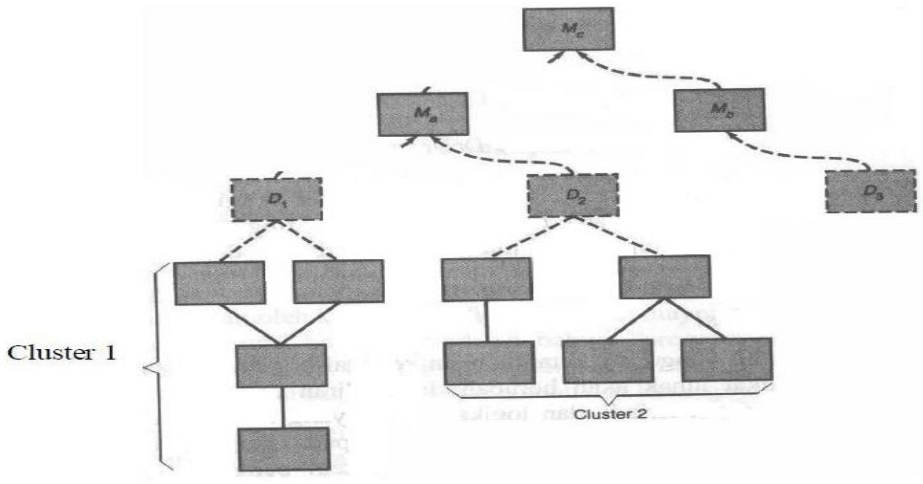
- Pertimbangan Pengujian Unit :
    - Myers mengusulkan checklist untuk pengujian interface.
    - Bila suatu modul melakukan I/O eksternal, maka pengujian interface tambahan harus dilakukan.
- Prosedur Pengujian Unit :
    - **Driver** adalah program yang menerima data untuk test case dan menyalurkan ke modul yang diuji dan mencetak hasilnya.
    - **Stub** melayani pemindahan modul yang akan dipanggil untuk diuji.

Pengujian Integrasi

Pengujian Integrasi dapat dilakukan dengan beberapa cara, yaitu :

- **Top Down Integration**
  - Modul utama digunakan sebagai test driver dan stub yang menggantikan seluruh modul yang secara langsung berada di bawah modul 85ontrol utama.
  - Tergantung pada pendekatan perpaduan yang dipilih (depth / breadth).
  - Uji coba dilakukan selama masing-masing modul dipadukan.
  - Pada penyelesaian masing-masing uji coba stub yang lain dipindahkan dengan modul sebenarnya.
  - Uji coba regression (pengulangan pengujian) dilakukan untuk mencari kesalahan lain yang mungkin muncul.

- **Bottom Up Integration**
  - Modul tingkat bawah digabungkan ke dalam cluster yang memperlihatkan subfungsi PL.
  - Driver (program 85ontrol pengujian) ditulis untuk mengatur input test case dan output.
  - Cluster diuji.
  - Driver diganti dan cluster yang dikombinasikan dipindahkan ke atas pada struktur program.



Gambar 9.2. Contoh Pengujian Integrasi

Pengujian Validasi

Pengujian validasi terdiri beberapa pengujian sebelum masuk ke pasaran atau digunakan oleh customer, dua diantaranya seperti di bawah ini :

- Pengujian Alpha

  Dilakukan pada sisi pengembang oleh seorang pelanggan. PL digunakan pada setting yang natural dengan pengembang “yang memandang” melalui bahu pemakai dan merekam semua kesalahan dan masalah pemakaian.
- Pengujian Beta

  Dilakukan pada satu atau lebih pelanggan oleh pemakai akhir PL dalam lingkungan yang sebenarnya, pengembang biasanya tidak ada pada pengujian ini. Pelanggan merekam semua masalah (real atau imajiner) yang ditemui selama pengujian dan melaporkan pada pengembangpada interval waktu tertentu.

Pengujian Sistem

Pengujian yang dilakuakan dalam sistem adalah sebagai berikut :

- Recovery Testing

  Sistem testing yang memaksa PL mengalami kegagalan dalam bermacam-macam cara dan memeriksa apakah perbaikan dilakukan dengan tepat.
- Security Testing

  Pengujian yang akan melakukan verifikasi dari mekanisme perlindungan yang akan dibuat oleh sistem, melindungi dari hal-hal yang mungkin terjadi.
- Strees Testing

  Dirancang untuk menghadapi situasi yang tidak normal pada saat program diuji. Testing ini dilakukan oleh sistem untuk kondisi seperti volume data yang tidak normal (melebihi atau kurang dari batasan) atau frekuensi.

Latihan soal :

1. Kumpulan aktifitas yang menjamin penerapan PL benar-benar sesuai dengan fungsinya yaitu …
   a. Verifikasi
   b. Pengujian
   c. Strategi
   d. Validasi
2. Pengujian Sistem yang memaksa PL mengalami kegagalan dalam bermacam-macam cara dan memeriksa apakah perbaikan dilakukan dengan tepat yaitu …
   a. Stress Testing
   b. Cluster Testing

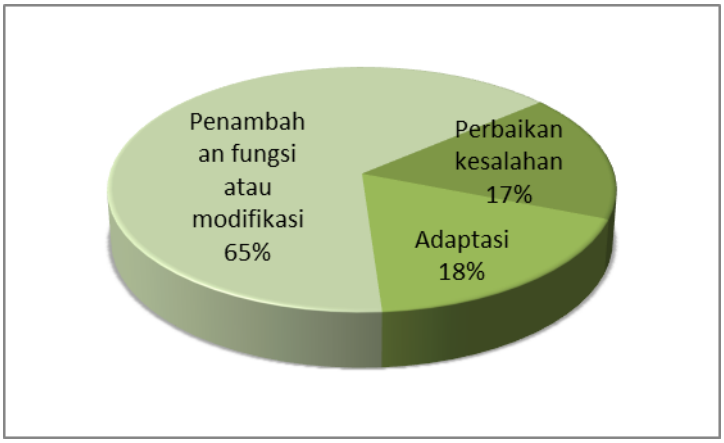
c. Network Testing
d. Recovery Testing

3. Pengujian validasi yang dilakukan pada satu atau lebih pelanggan oleh pemakai akhir PL dalam lingkungan yang sebenarnya yaitu …
   a. Pengujian Alpha
   b. Pengujian Beta
   c. Pengujian Integrasi
   d. Pengujian Unit
4. Pengujian integrasi dimana modul utama digunakan sebagai test driver dan stub yang menggantikan seluruh modul yang secara langsung berada di bawah modul control utama yaitu …
   a. Pengujian Alpha
   b. Pengujian Beta
   c. Top Down Integration
   d. Bottom Up Integration
5. Berikut merupakan stratefi pengujian perangkat lunak, kecuali …
   a. Pengujian mulai pada tingkat modul yang paling bawah, dilanjutkan dengan modul di atasnya kemudian hasilnya dipadukan.
   b. Teknik pengujian yang berbeda mungkin menghasilkan sedikit perbedaan (dalam hal waktu).
   c. Pengujian dilakukan oleh pengembang PL dan (untuk proyek yang besar) suatu kelompok pengujian yang independen.
   d. Pengujian harus mulai “dari yang kecil” dan berkembang ke pengujian “yang besar”.
6. Pengujian adalah proses eksekusi suatu program dengan maksud menemukan kesalahan, merupakan pengertian pengujian menurut …
   a. Davis
   b. Glen Myers
   c. Pressman
   d. Beizer

Bab 10
Pemeliharaan Perangkat Lunak

Definisi Pemeliharaan

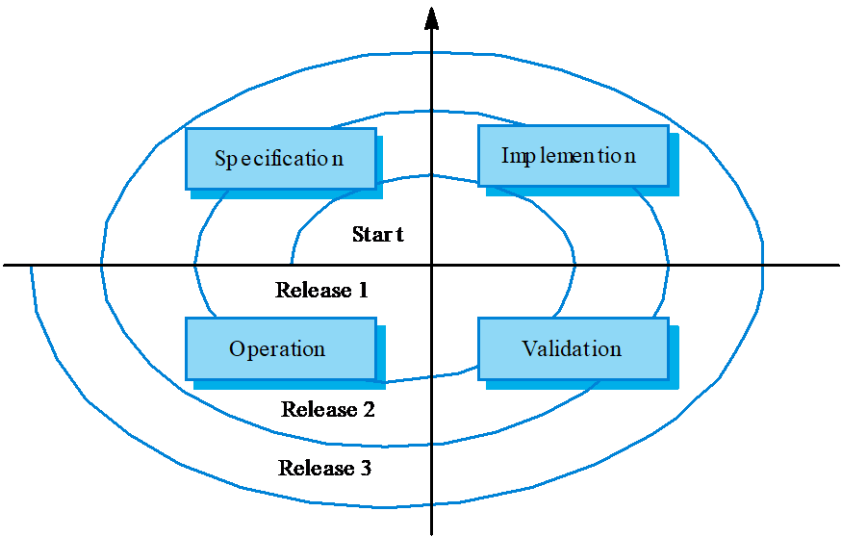
Modifikasi produk perangkat lunak setelah di *realest* untuk memperbaiki kesalahan (*faults*), untuk meningkatkan performa atau atribut lainnya (*reliable*, *maintainable*, …), dan untuk adaptasi produk perangkat lunak terhadap lingkungan baru.” (IEEE)



Gambar 10.1. Persentase Modifikasi, perbaikan kesalahan dan adaptasi

Pemeliharaan tidak dapat dielakkan karena :

- Kebutuhan sistem biasanya berubah ketika sistem sedang dikembangkan dikarenakan lingkungannya yang berubah. Oleh karena itu sistem yang dikirimkan tidak akan sesuai dengan kebutuhannya.
- Sistem sangat berhubungan erat dengan lingkungannya. Ketika suatu sistem terpasang pada lingkungan maka sistem tersebut akan mengubah lingkungannya dan karenanya terjadi perubahan kebutuhan sistem.
- Sistem harus dapat dipelihara jika sistem tetap ingin berguna di lingkungannya.



Gambar 10.2. Model Spiral dan Evolusi

Alasan kesulitan pemeliharaan perangkat lunak diantaranya adalah:

- Rendahnya kualitas perangkat lunak yang berjalan (yang sudah ada).
- Sistem tidak dirancang untuk memperhatikan konsep pemeliharaan.
- Pemeliharaan bukan merupakan bagian yang dirasakan perlu pada suatu perangkat lunak.

Biaya pemeliharaan

- Biaya pemeliharaan biasanya lebih besar dari biaya pengembangan yaitu sekitar 2 sampai 100 kali tergantung dari aplikasinya.
- Dipengaruhi oleh faktor teknikal dan non teknikal.
- Peningkatan biaya setelah perangkat lunak dipelihara. Kesalahan pada proses pemeliharaan struktur perangkat lunak dapat menyebabkan pemeliharaan kedepan yang lebih sulit.
- Perangkat lunak yang sudah lama dapat memiliki biaya dukungan yang cukup tinggi (misalnya bahasa pemograman yang lama, *compilers* dan lain sebagainya).

Biaya pemeliharaan dipengaruhi oleh beberapa faktor di bawah ini :

- Team stability

  Biaya pemeliharaan akan berkurang jika yang terlibat dalam proses pemeliharaan adalah staff yang sama dalam beberapa waktu pemeliharaan.
- Contractual responsibility

Pengembang sistem mungkin tidak memiliki tanggung jawab kontrak untuk pemeliharaan sehingga tidak ada keinginan untuk merancang sistem yang nantinya akan dapat berubah.

- Staff skills

  Staff pemeliharaan sering kali tidak berpengalaman dan memiliki pengetahuan yang terbatas.

- Program age and structure

  Sesuai dengan usia program, strukturnya akan menua dan akan sulit untuk dimengerti dan diubah.

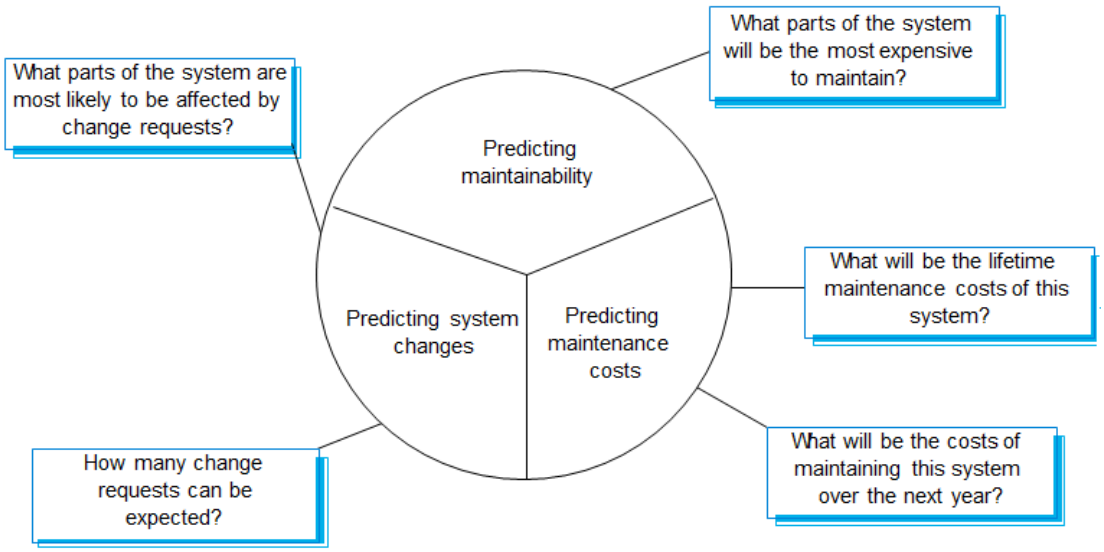
Prediksi Pemeliharaan

Prediksi pemeliharaan sangat memperkirakan bagian mana dari sistem yang akan dapat menjadi masalah dan memiliki biaya pemeliharaan yang tinggi

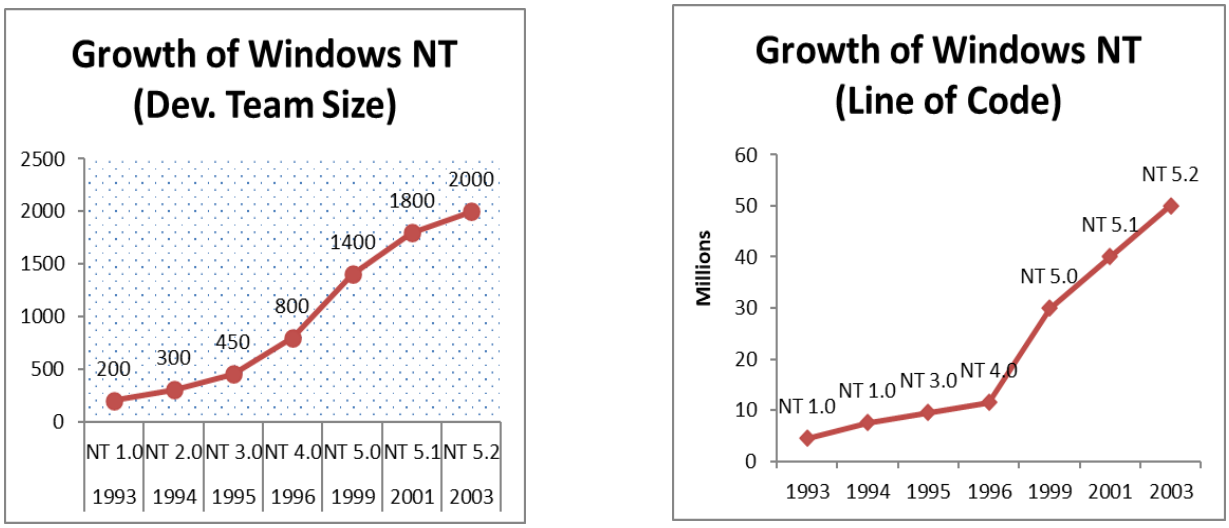
Memprediksi beberapa perubahan membutuhkan pemahaman tentang hubungan antara sistem dan lingkungannya.

Faktor yang mempengaruhi hubungan ini adalah

- Jumlah dan kompleksitas dari antar muka sistem;
- Jumlah dari kebutuhan sistem yang terpengaruhi secara mendasar;
- Bisnis proses dimana tempat sistem tersebut digunakan.
-



Gambar 10.3. Prediksi Pemeliharaan



Gambar 10.4. Diagram Perkembangan Modifikasi Windows NT

- Microsoft Windows NT, 30 juta baris code ditambahkan selama 6 tahun.
- Telecom switch software, 5.2 juta modifikasi sepanjang satu dekade.
- Web-based applications, 73% dari biaya pembuatan e-commerce digunakan untuk *re-design* web site setelah implementasi pertama

Kategori pemeliharaan menurut ISO/IEC 14764 (2006) adalah :

1. **Preventive maintenance**

   Perubahan untuk mendeteksi dan memperbaiki kesalahan yang baru sebelum menjadi kesalahan yang fatal.

2. **Corrective maintenance**

   Perubahan untuk mengatasi kegagalan atau kerusakan yang ditemukan selama masa waktu *preventive maintenance*.

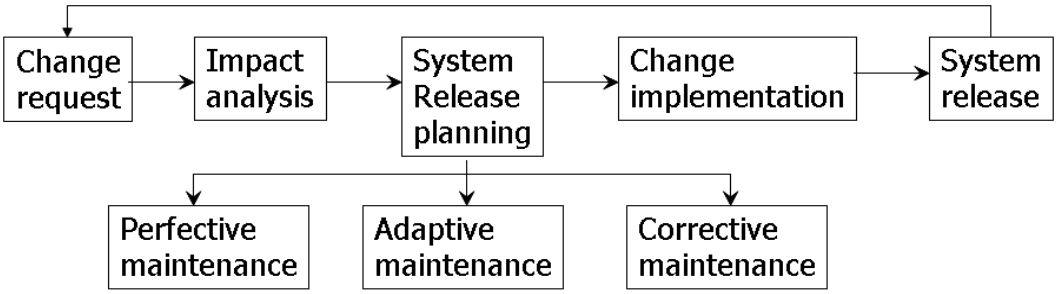
3. **Adaptive maintenance**

   Perubahan untuk menjaga agar perangkat lunak dapat digunakan pada lingkungan yang berbeda.

4. **Perfective maintenance**

   Perubahan untuk meningkatkan *performance* ataupun *maintainability*.

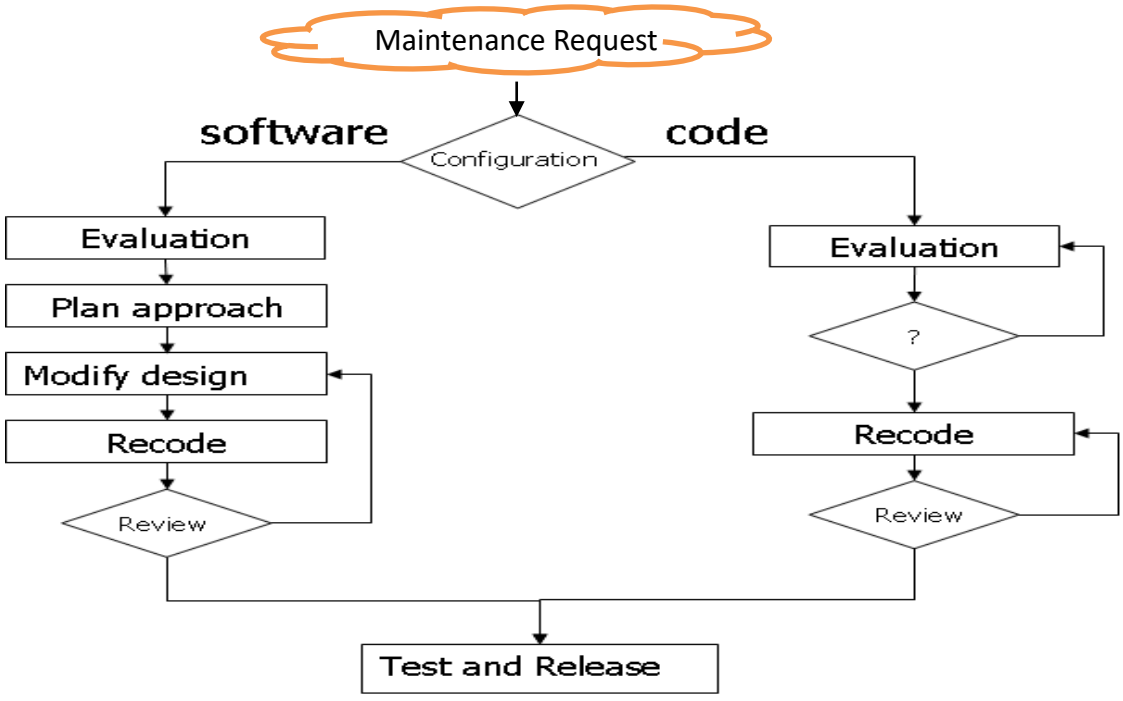
   Menurut E. B. Swanson pada bukunya yang terbit pada tahun 1976 kategori pemeliharaan dibagi menjadi *corrective*, *adaptive*, dan *perfective*.



Gambar 10.5. Proses Pemeliharaan

Permintaan perubahan

- Perubahan yang diminta oleh *users*, *customers* atau *management*
- Pada kenyataannya, semua perubahan memerlukan analisis yang hati-hati
- Pada kenyataan, perubahan perangkat lunak dirasakan perlu untuk
    - Memperbaiki kesalahan
    - Perubahan lingkungan sistem
    - Kebutuhan yang mendesak dari perubahan bisnis



Gambar 10.6. Perminataan Perubahan Pemeliharaan

Latihan soal :

1. Salah satu yang menyebabkan pemeliharaan tidak dapat dielakkan adalah..
   a. Sistem harus dapat dipelihara jika sistem tetap ingin berguna di lingkungannya.
   b. Pemeliharaan bukan merupakan bagian yang dirasakan perlu pada suatu perangkat lunak.
   c. Dipengaruhi oleh faktor teknikal dan non teknikal.
   d. Sesuai dengan usia program, strukturnya akan menua dan akan sulit untuk dimengerti dan diubah.
2. Salah satu yang menyebabkan kesulitan pemeliharaan perangkat lunak adalah..
   a. Sistem harus dapat dipelihara jika sistem tetap ingin berguna di lingkungannya.
   b. Pemeliharaan bukan merupakan bagian yang dirasakan perlu pada suatu perangkat lunak.
   c. Dipengaruhi oleh faktor teknikal dan non teknikal.
   d. Sesuai dengan usia program, strukturnya akan menua dan akan sulit untuk dimengerti dan diubah.
3. Faktor-faktor yang mempengaruhi hubungan antara sistem dan lingkungannya, kecuali..
   a. Bisnis proses dimana tempat sistem tersebut digunakan.
   b. Jumlah dari kebutuhan sistem yang terpengaruhi secara mendasar
   c. Bisnis proses dimana tempat sistem tersebut tidak digunakan
   d. Jumlah dan kompleksitas dari antar muka sistem
4. Dibawah ini yang bukan termasuk dari Kategori pemeliharaan menurut ISO/IEC 14764 (2006) adalah..
   a. Adaptive maintenance
   b. Preventive maintenance
   c. Corrective maintenance
   d. Constructive maintenance
5. Faktor-faktor yang mempengaruhi biaya pemeliharaan, kecuali..
   a. Sequence of events
   b. Staff skills
   c. Contractual responsibility
   d. Team stability

REFERENSI

Adi Nugroho, Rekayasa Perangkat Lunak Menggunakan UML dan Java, Andi Publisher, 2010.

Hanif Al Fatta, Analisis dan PErancangan SIstem Informasi, Penerbit Andi, 2007.

Ian Sommerville, *Software Engineering*, 7th Addison Wesley Publishing Company, 2003

Janner Simamarta, Rekayasa Perangkat Lunak, Andi Publisher, 2010.

Julius Hermawan, Analisa Desain & Pemrograman Berorientasi Obyek dengan UML dan Visual Basic.NET, Andi Publisher, 2000

Paul Ammann, Jeff Offutt, Introduction To Software Testing, 2013

Ron Patton, Software Testing Second Edition, Sam Publisher, 2006

Rosa A.S, Modul Pembelajaran Rekayasa Perangkat Lunak (Terstruktur & Berorientasi Objek), Penerbit Informatika, Juni 2011

Stephen H. Khan, Metrics and Models in Software Quality Engineering Second Edition, Pearson Education, India, 2007.

Tavri D. Mahyuzir, *Pengantar Rekayasa Perangkat Lunak*, Elexmedia Komputindo, 1997