Rekayasa Perangkat Lunak

Dr. Ratna Wardani, S.Si., M.T.

JURUSAN PENDIDIKAN TEKNIK ELEKTRONIKA
UNIVERSITAS NEGERI YOGYAKARTA
2012

DAFTAR ISI

Bab 1. Pendahuluan

1

1. Pengertian Perangkat Lunak

Perangkat lunak adalah seluruh perintah yang digunakan untuk memproses informasi. Perangkat lunak dapat berupa program atau prosedur. Program adalah kumpulan perintah yang dimengerti oleh komputer sedangkan prosedur adalah perintah yang dibutuhkan oleh pengguna dalam memprosesinformasi (O'Brien, 1999).

Perangkat Lunak (*software*) merupakan data elektronik yang disimpan sedemikian rupa oleh komputer itu sendiri, data yang disimpan ini dapat berupa program atau instruksi yang akan dijalankan oleh perintah, maupun catatan-catatan yang diperlukan oleh komputer untuk menjalankan perintah yang dijalankannya. Untuk mencapai keinginannya tersebut dirancanglah suatu susunan logika, logika yang disusun ini diolah melalui perangkat lunak, yang disebut juga dengan program beserta data-data yang diolahnya. Pengeloahan pada software ini melibatkan beberapa hal, diantaranya adalah sistem operasi, program, dan data. Software ini mengatur sedemikian rupa sehingga logika yang ada dapat dimengerti oleh mesin komputer.

2. Sejarah Perkembangan Perangkat Lunak.

2.1. Tahun 1950 - 1965 (*The Early Years*)

a. Batch Orientation : Berhubungan dengan hardware.
b. Limited Distribution :
c. Belum ada komputer PC (baru ada main frame) sehingga tidak semua orang dapat memakainya dan software terbatas untuk kepentingan tertentu.
d. Custom Software : Software seragam yaitu EDP (Electronic data Processing) dan word processing

2.2. Tahun 1965 - 1970 *(The Second Era*)

a. Multiuser : satu komputer untuk banyak user
b. Real Time : yaitu job now, print now.
c. Database : masih dalam bentuk magnetic
d. Product Software : Sudah munculnya pabrik software (ex.ISICALS, IBM)

2.3. Tahun 1976 - 1986 (T*he Third Era*)

a. Distributed system : Sistem operasi yang diperuntukkan jaringan komputer terdistribusi dimana pengalokasian kerja secara otomatis dilaksanakan sistem operasi (awal mula lahirnya LAN)
b. Embedded Intelligent : Penamaan Software di dalam prompt
c. Low cost Hardware
d. Custommer Impact : Pengkritikan Software.

2.4. Tahun 1987 – 200

a. Powerfull desktop system : Dikenal istilah 386 (karena lahir tahun 1986) mempunyai kapasitas dari 16 bit - 32 bit, Mulai dikenal windows, contoh AT, PS/2
b. Object Oriented Technologi
c. Expert System
d. Artificial Neural Network : syaraf buatan (robotic)
e. Parallel Computing

3. Krisis Perangkat Lunak

3.1. Problem

a. Perangkat lunak tidak berfungsi secara baik (kualitas yang kurang).
b. Ketidaktepatan penjadwalan proses dan biaya produksi.
c. Produktivitas yang belum dapat memenuhi tuntutan kebutuhan pemeliharaan yang sukar.
d. Pemeliharaan yang sukar.

3.2. Penyebab

Karakteristik perangkat lunak dan faktor manusia.

a. Perangkat lunak bersifat logical : kualitasnya ditentukan secara individual.
b. Perangkat lunak tidak lapuk : perbaikan komponen perangkat lunak biasanya melibatkan modifikasi/perubahan desain.
c. Pihak manajer : kurang mengetahui karakteristik perangkat lunak → proses pembuatannya tidak dapat dikelola dengan baik.
d. Pihak pemoggram perangkat lunak : pendekatan individual berdasarkan pengalaman masa lalu, padahal hanya sedikit yang menghasilkan perangkat lunak dengan kualitas yang baik → bagaikan rantai yang tak terputus.
e. Faktor manusia secara umum : sifat *ignorant* dan *resistant* terhadap metode-metode baru pembuatan perangkat lunak, serta kepercayaan terhadap mitos-mitos tertentu.

4. Pengertian Rekayasa Perangkat Lunak

4.1. Definisi

RPL menurut Fritz Bauer [NAU69]:

"Penerapan dan pemanfaatan prinsip-prinsip rekayasa untuk menghasilkan perangkat lunak yang ekonomis, andal dan bekerja secara efisien pada mesin-mesin yang nyata"

4.2. Tujuan Rekayasa Perangkat Lunak

Tujuan Rekayasa Perangkat Lunak dapat dijelaskan sebagai berikut:

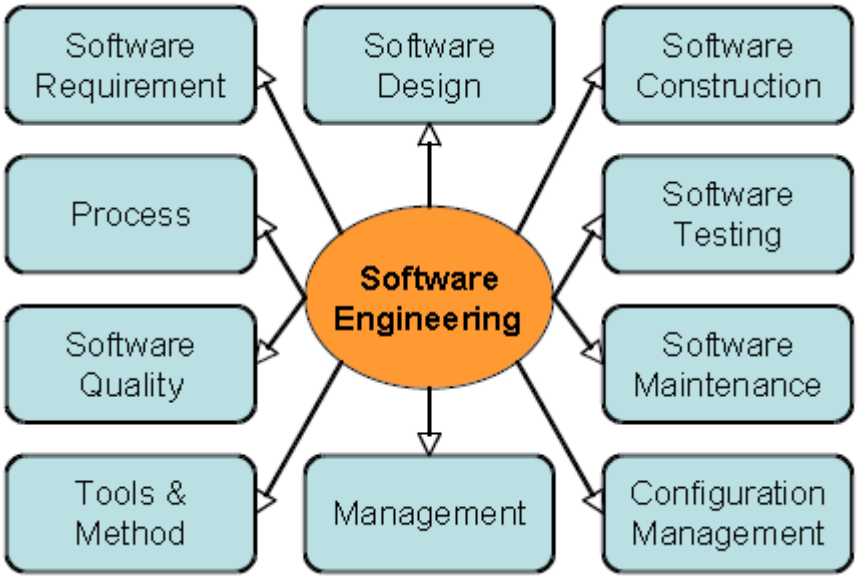
a. Memperoleh biaya produksi perangkat lunak yang rendah.
b. Menghasilkan perangkat lunak yang kinerjanya tinggi, andal dan tepatwaktu.
c. Menghasilkan perangkat lunak yang dapat bekerja pada berbagai jenis platform
d. Menghasilkan perangkat lunak yang biaya perawatannya rendah.

4.3. Elemen Kunci RPL

a. Metode : 'how to' yang bersifat teknis. Meliputi bidang-bidang perencanaan proyek, estimasi, analisis persyaratan, perancangan, coding, pengujian dan pemeliharaan.
b. Tools : memberikan dukungan automasi bagi metode dikenal dengan CASE
c. Prosedur : mengintegrasikan metode dan tool untuk mendefinisikan kapan suatu metode akan digunakan, hasil yang diharapkan, pengendalian untuk menjamin kualitas hasil, dan milestone yang dapat digunakan untuk mengevaluasi kemajuan

4.4. Ruang Lingkup RPL

Ruang lingkup RPL dapat digambarkan sebagai berikut :



Gambar 1. Ruang lingkup RPL (Abran et al, 2004)

- *Software Requirment* merupakan kegiatan yang dilakukan untuk mengidentifikasi dan menganalisis kebutuhan perangkat lunak. Hasil akhir tahapan ini adalah spesifikasi dan model perangkat lunak.

- *Software Design* adalah tahapan perancangan arsitektur, komponen, antar muka, dan karakteristik lain dari perangkat lunak
- *Software Construction* berhubungan dengan detail pengembangan perangkat lunak, termasuk algoritma, pengkodean, pengujian dan pencarian kesalahan
- *Software Testing* meliputi pengujian pada keseluruhan perilaku perangkat lunak
- *Software Maintenance* mencakup upaya-upaya perawatan ketika perangkat lunak telah dioperasikan
- *Software Configuration* management berhubungan dengan usaha perubahan konfigurasi perangkat lunak untuk memenuhi kebutuhan tertentu
- *Software Engineering Management* berkaitan dengan pengelolaan dan pengukuran RPL, termasuk perencanaan proyek perangkat lunak
- *Software Engineering Tools and Methods* mencakup kajian teoritis tentang alat bantu dan metode RPL
- *Software Engineering Process* berhubungan dengan definisi, implementasi pengukuran, pengelolaan, perubahan dan perbaikan proses RPL
- *Software Quality* menitik beratkan pada kualitas dan daur hidup perangkat lunak

5. Metode Pengembangan dalam RPL

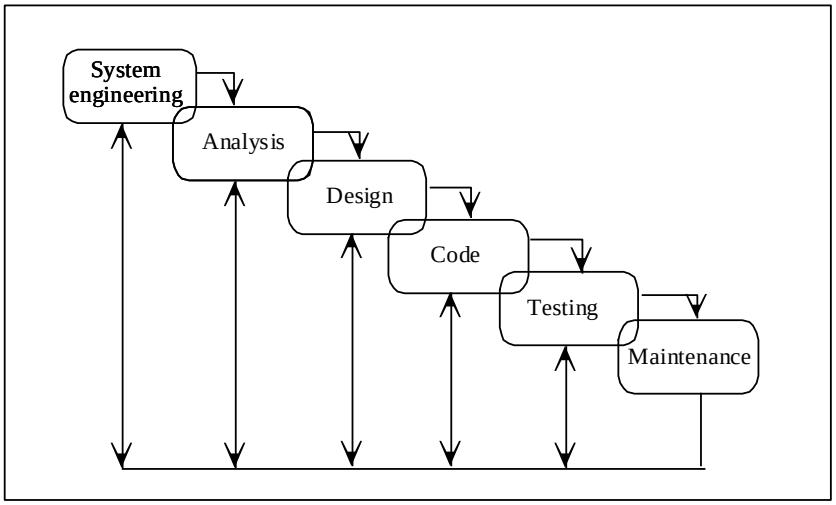
5.1. Siklus Klasik (Model Air terjun).

a. Berdasarkan siklus konvensional dalam bidang rekayasa lainnya → pendekatan sekuensial yang sistematis.

b. Tahapan-tahapan :

1. Analisis dan Rekayasa Sistem : Perangkat lunak merupakan bagian dari sebuah sistem → untuk mendapatkan bgambaran yang meluas pada aras sistem.

2. Analisis Persyaratan : Fakus lebih terarah ke perangkat lunak, berusaha mengetahui aspek 'what' → melibatkan pemakaian dan pengembangan.
3. Perancangan : Menerjemahkan persyaratan menjadi suatu bentuk representasi yang dapat dievaluasi kualitasnya sebelumnya coding dilakukan.



Gambar 2. Model Air Terjun

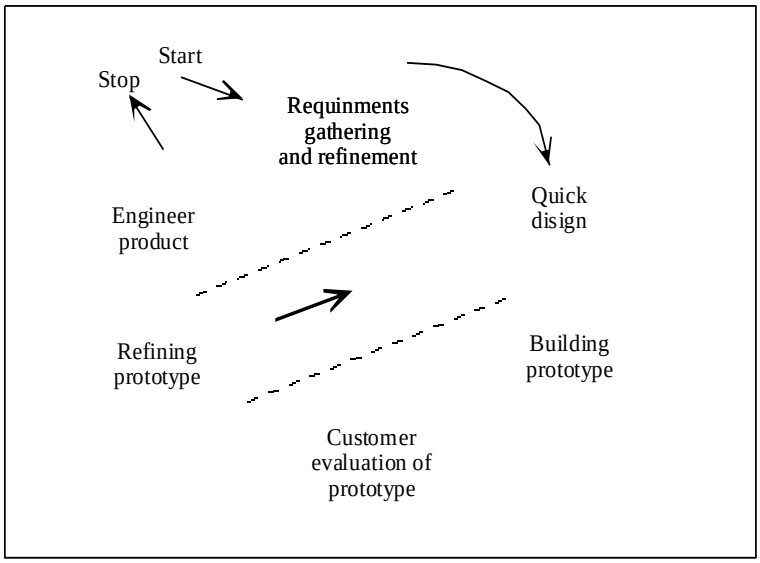
4. Coding (Penulisan Program) : Penerjemahan rancangan ke dalam bentuk yang dapat dimengerti komputer.
5. Pengujian. Berfokus pada rincian logikal dari perangkat lunak, bertujuan mengungkapkan dan menghilangkan kesalahan-kesalahan yang ada sehingga perangkat lunak bekerja sesuai dengan yang diharapkan.
6. Pemeliharaan. Meliputi kegiatan-kegiatan koreksi kesalahan dan penyesuaian perangkat lunak terhadap perubahan lingkungannya.

c. Problem :
   1. Proyek-proyek pengembangan perangkat lunak jarang yang mengikuti alur sekuensial secara ketat → banyak melibatkan proses iterasi.

2. Sulit bagi pemberi pekerjaan untuk menyatakan semua keinginannya secara eksplisit di awal tahap pengembangan.
3. Hasil baru akan diketahui lama setelah proyek pengembangan dimulai

5.2. Prototyping

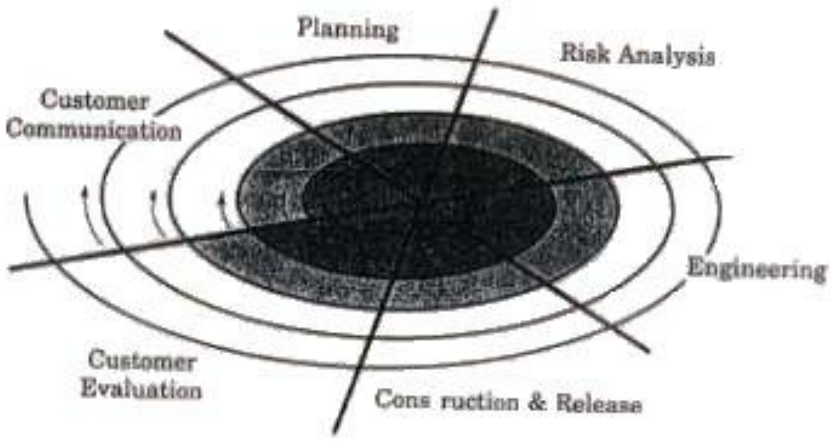
a. Bila pemakaian belum siap dengan persyaratan perangkat lunak secara lengkap (rinci).
b. Model Perangkat Lunak :
   1) Model 'kertas. Hanya agar pemakai mengerti interaksi antara dia dengan perangkat lunak.
   2) Model kerja. Mengimplementasikan beberapa fungsi perangkat lunak
   3) Program. Sebagian atau semua fungsi telah diimplementasikan, dan akan dikembangkan dalam proses pengembangan.
c. Prototype bukan perangkat lunak yang sesungguhnya.
d. Melibatkan suatu proses iterasi yang berfokus pada penyempurnaan prototype yang didasarkan pada persyaratan yang diminta oleh pemakai → pemakai terlibat intensif selama proses pengembangan.
e. Problem. Ketidaksadaran bahwa prototype bukan hasil yang diharapkan.
   1) Pemakai : tidak sabar untuk menunggu perangkat lunak yang sebenarnya.
   2) Pengembang : Pemakai teknik dan tools yang tidak optimal pada prototype yang akhirnya tetap digunakan pada perangkat lunak yang sesungguhnya.



Gambar 3. Prototype

5.3. Model Spiral

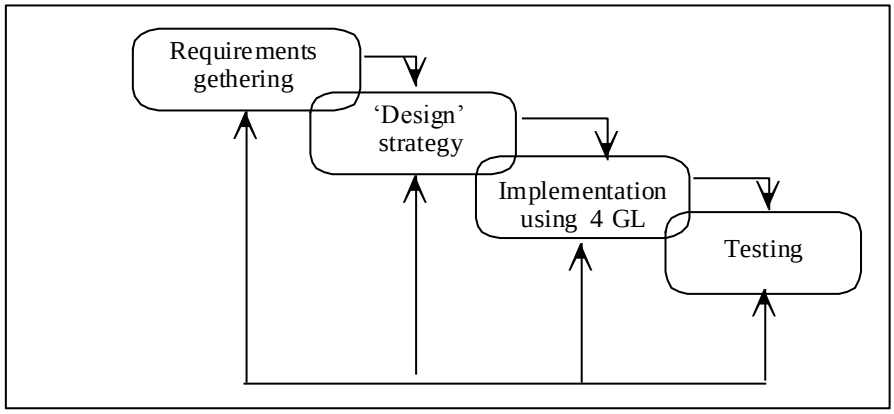
a. Menggabungkan keuntungan-keuntungan model air terjun dan prototyping, dan memasukkan aktivitas analisis resiko (risk analysis)



Gambar 4. Typical Spiral Model

b. Melibatkan proses iterasi, tiap iterasi bekerja pada satu ‘level produk’ (dari level prototype awal sampai pada level perangkat lunak yang diinginkan). Tiap perpindahan level didahului oleh analisis resiko.

c. Terdiri dari 4 aktivitas utama :



Gambar 5. Aktivitas Spiral Model

1) Perencanaan : Penentuan sasaran, alternatif solusi dan hambatan.
2) Analisis resiko : Analisis alternatif solusi dan identifikasi resiko
3) Perekayasaan : Pengembangan produk pada 'level berikutnya'
4) Evaluasi oleh pemakai : Diterapkan pada hasil proses perekayasaan.

d. Problem. Menuntut keahlian dalam bidang analisis resiko.

5.4. 4th Generation Technique

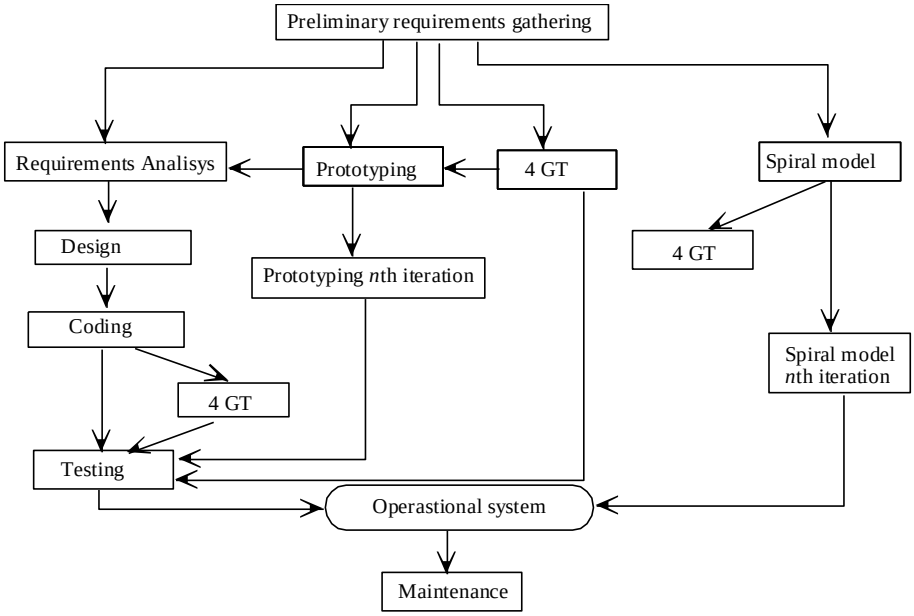
a. Merupakan kumpulan tool perangkat lunak
b. Memungkinkan pengembangan menyebutkan karakteristik perangkat lunak pada aras yang lebih tinggi, dan tool-tool tersebut akan membangkitkan kode program secara otomatis.
c. Beberapa bidang pengembangan perangkat lunak yang telah dapat diautomatisasikan :
   1) Query basis data (database query)
   2) Pembangkitan laporan (report generation)
   3) Manipulasi data
   4) Definisi layar

5) Kemampuan grafis aras tinggi

d. Proses mirip dengan model air terjun (diawali dengan pengumpulan persyaratan perangkat lunak).

5.5. Gabungan beberapa metode

Bertujuan mengambil kelebihan masing-masing paradigma



Gambar 6. Menggabungkan beberapa paradigma

Bab 2. Manajemen Proyek

2

1. Pendahuluan

Manajemen proyek merupakan layer yang mendasari tahap pengembangan selanjutnya. Tahap ini meletakan dasar-dasar perencanaan berupa :

a. Sasaran
b. Ruang Lingkup
c. Alternatif-alternatif Solusi
d. Kekangan-kekangan Manajerial dan Teknis

Aspek yang perlu disusun adalah:

a. Estimasi Biaya
b. Penjadwalan Proyek
c. Resiko-resiko yang muncul

2. Pengukuran Perangkat Lunak

a. Meskipun perangkat lunak bersifat abstrak, tetap diperlukan pengukuran yang berkaitan dengannya.
b. Diperlukan untuk melakukan estimasi, dan berkaitan dengan aspek produktivitas :
    1. Sebagai indikasi kualitas produk yang dihasilkan.
    2. Untuk mengevaluasi produktivitas kerja.
    3. Untuk mengakses kemanfaatan pemakaian metode-metode dan teknik-teknik RPL.
    4. Sebagai dasar untuk melakukan estimasi biaya, beban kerja, maupun penjadwalan.
    5. Sebagai justifikasi untuk memperoleh tool-tool baru atau pelatihan tambahan.

3. Estimasi

a. Dilakukan untuk mendapatkan besaran-besaran kuantitatif yang diperlukan dalam perencanaan proyek.
b. Estimasi I : ruang lingkup perangkat lunak :
    1. Fungsi
    2. Kinerja
    3. Kekangan
    4. Antarmuka
    5. Keandalan
    6. Harus diberikan secara jelas, *bounded*, dan *eksplisit*
c. Estimasi II : resource → perangkat keras, perangkat lunak dan manusia.
d. Teknik-teknik dekomposisi : LOC & FP, estimasi usaha.
e. Model-model estimasi empiris : COCOMO, Putnam.

4. Analisis Resiko

a. Analisis resiko untuk menentukan ‘go’ atau ‘no go’
b. Empat tahap aktivitas :
    1. Identifikasi resiko : Mengidentifikasi keberadaan resiko resiko sebagai berikut :
        i. Resiko proyek (problem-problem anggaran, pendwalan, personil, resource, pelanggan)
        ii. Teknis (problem-problem dalam perancangan, implementasi, verifikasi dan pemeliharaan - kekaburan, ketidakpastian, ketertinggalan)
        iii. Bisnis (problem-problem pemasaran, ketidaksesuaian kebutuhan, dukungan manajemen, dukungan anggaran)
    2. Proyeksi / Estimasi Resiko : Kemungkinan terjadinya resiko-resiko dan konsekuensi yang harus dialami bila resiko tersebut muncul
    3. Penafsiran Resiko : Mengevaluasi lebih jauh estimasi kemungkinan munculnya resiko-resiko tersebut dan mencari cara pengendalian bila resiko-resiko tersebut terjadi

4. Manajemen Resiko : Langkah-langkah pengendalian resiko.

5. Penjadwalan

a. Kadang-kadang lebih penting daripada masalah pembiayaan, terutama untuk produk-produk kunci dengan jumlah konsumen yang sangat banyak.
b. Hubungan Personil dan beban kerja : semakin banyak personil belum tentu menyebabkan pekerjaan lebih cepat diselesaikan → komunikasi antar personil menyita waktu kerja.
c. Metode : Program Evaluation and Review Technique (PERT) dan Critical Path Method (CPM) :
    1. Mendefinisikan jalur kritis (rangkaian aktivitas yang menentukan lama waktu pekerjaan)
    2. Memperkirakan waktu penyelesaian untuk setiap aktivitas beserta batas-batasnya.
d. Setelah jadwal dibuat, pelaksanaannya perlu dimonitor dan dikendalikan dengan cara : *meeting*, evaluasi *milestones*, evaluasi hasil yang diperoleh.

6. Rencana Proyek

a. Berisi rangkuman dari semua yang dikerjakan pada tahap perencanaan :
    1. Introduksi
        i. Sasaran Dokumen
        ii. Sasaran Proyek
            a. Sasaran
            b. Fungsi-fungsi utama
            c. Kinerja
            d. Kekangan
    2. Estimasi
        i. Data Historis yang digunakan untuk estimasi
        ii. Teknik Estimasi yang digunakan

   iii. Hasil estimasi
3. Resiko resiko Proyek
   i. Analisis resiko
   ii. Manajemen resiko
4. Penjadwalan.
   i. Struktur pekerjaan
   ii. Diagram Gantt
   iii. Diagram resource
5. Resource proyek
   i. Manusia
   ii. Perangkat keras
   iii. Perangkat lunak
6. Organisasi Tim
7. Mekanisme Pengendalian Pekerjaan
8. Lampiran-lampiran

Bab 3. Proses Rekayasa Perangkat Lunak

1. Pendahuluan

3

Secara umum, proses rekayasa perangkat lunak dibagi menjadi 3 tahapan utama, yaitu Definisi, Pengembangan dan Pemeliharaan.

a. Definisi yang berfokus pada '**What'**
   1) Analisa sistem
   2) Perencanaan Pengembangan
   3) Analisis Kebutuhan

b. Pengembangan yang berfokus pada '**How'**
   1) Perancangan/Desain
   2) Implementasi/Coding
   3) Pengujian

c. Pemeliharaan yang berfokus pada '**Change'**
   1) Koreksi
   2) Adaptasi
   3) Penyempurnaan

Elemen-elemen yang tercakup dalam poses rekayasa perangkat lunak mencakup : perangkat lunak, perangkat keras, manusia, basis data, dokumentasi, dan prosedur. Elemen-elemen ini saling berinteraksi dalam berbagai cara untuk mentransformasi informasi.

Aktivitas-aktivitas dan ruang lingkupnya :

1) Rekayasa sistem komputer : bersifat umum, meliputi semua aspek yang berkaitan dengan penggunaan komputer dalam proses produksi.
2) Rekayasa perangkat keras : perangkat keras (komputer dan segala sesuatu yang berhubungan dengannya).
3) Rekayasa perangkat lunak.
4) Rekayasa faktor manusia (*human engineering*) : penyiapan faktor manusia dan hubungannya dengan sistem komputer.

5) Rekayasa basis data : analisis, perancangan dan implementasi basis data yang diterapkan begitu domain informasi basis data selesai ditentukan.
6) Rekayasa dokumen dan prosedur

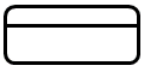
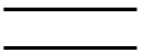
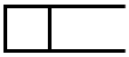
2. Analisis Kebutuhan

Analisis kebutuhan adalah tahapan rekayasa perangkat lunak yang digunakan untuk proses pemecahan masalah. Proses ini menguraikan sebuah sistem menjadi komponen-komponennya dengan tujuan mempelajari seberapa bagus komponen-komponen tersebut bekerja dan berinteraksi untuk meraih tujuan yang ditetapkan.

Analisis adalah bagian yang penting dari proses rekayasa perangkat lunak, karena semua proses lanjutan akan sangat bergantung pada baik tidaknya hasil analisis. Ada satu bagian penting yang biasanya dilakukan dalam tahapan analisis yaitu pemodelan proses bisnis.

Model proses adalah model yang memfokuskan pada seluruh proses di dalam sistem yang mentransformasikan data menjadi informasi (Harris, 2003). Model proses juga menunjukkan aliran data yang masuk dan keluar pada suatu proses. Biasanya model ini digambarkan dalam bentuk Diagram Arus Data (Data Flow Diagram / DFD). DFD meyajikan gambaran apa yang manusia, proses dan prosedur lakukan untuk mentransformasi data menjadi informasi. Penggunaan DFD sebagai Modeling Tool dipopulerkan Oleh Demacro & Yordan (1979) dan Gane & Sarson (1979) dengan menggunakan pendekatan Metoda Analisis Sistem Terstruktur.

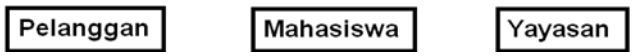
DFD menggambarkan arus data dari suatu sistem informasi, baik sistem lama maupun sistem baru secara logika tanpa mempertimbangkan lingkungan fisik dimana data tersebut berada.

Demacro & Yordan	Keterangan	Gane & Sarson
	External Entity	
	Process	
	Data Flow	
	Data Store	

Tabel 1. Simbol dalam DFD

2.1 External Entity

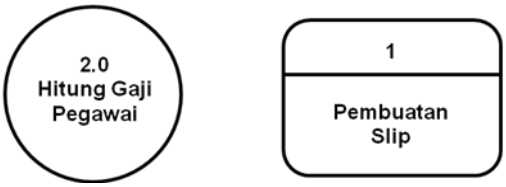
External Entity adalah merupakan entitas diluar sistem yang berinteraksi dengan sistem yang akan dikembangkan. External Entity ini dapat berupa orang, organisasi, maupun sistem yang lain. Contoh dari External Entity ini adalah Pelanggan, Mahasiswa, Yayasan, dan lain sebagainya.



Gambar 7. Simbol External Entity

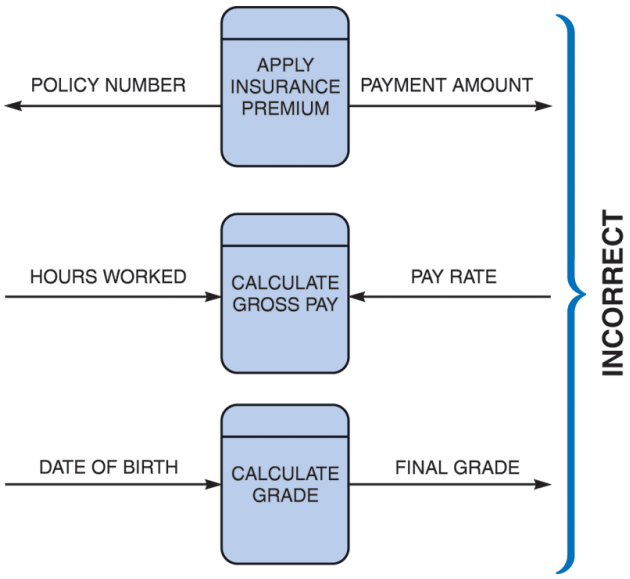
2.2 Process

Merupakan kegiatan atau pekerjaan yang dilakukan oleh orang atau mesin komputer, dimana aliran data masuk kemudian ditranformasikan keluar (aliran data keluar).



Gambar 8. Simbol Process

Beberapa penggambaran process yang tidak benar:



Gambar 9. Penggambaran Process yang salah

2.3 Data Flow

Data Flow merupakan aliran data yang masuk maupun keluar dari suatu Process dan disimbolkan dengan anak panah. Aliran data dapat berbentuk sebagai berikut :

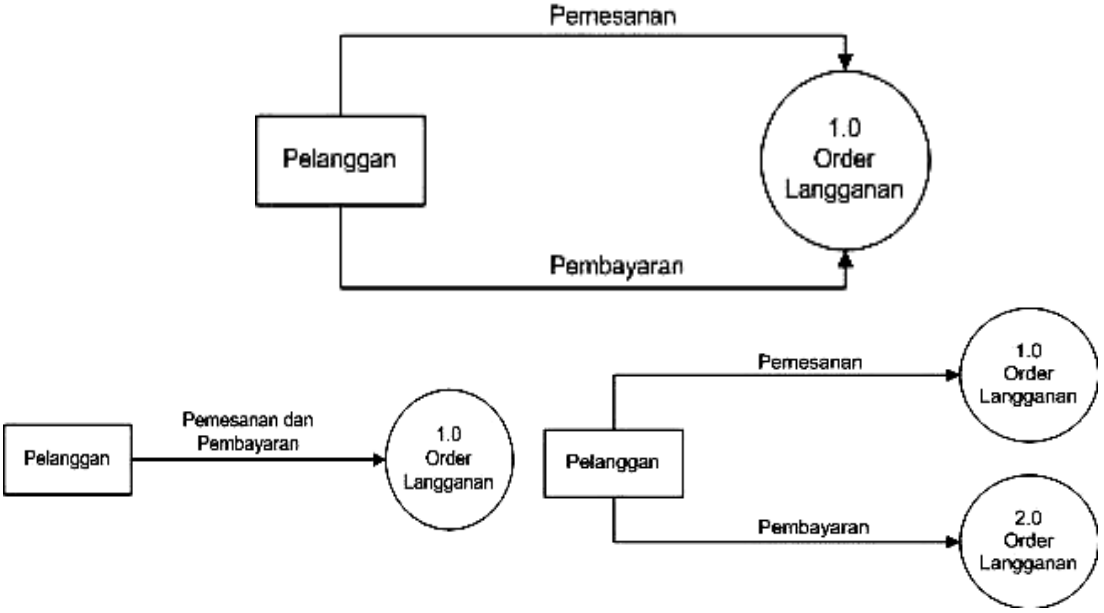
1. Formulir atau dokumen yang digunakan perusahaan
2. Laporan tercetak yang dihasilkan sistem
3. Output dilayar komputer
4. Masukan untuk komputer
5. Komunikasi ucapan
6. Surat atau memo
7. Data yang dibaca atau direkam di file
8. Suatu isian yang dicatat pada buku agenda
9. Transmisi data dari suatu komputer ke komputer lain

Data ada tiga kemungkinan, antara lain :

2.3.1. *Packet of Data*

Bila dua data mengalir dari suatu sumber yang sama ke tujuan

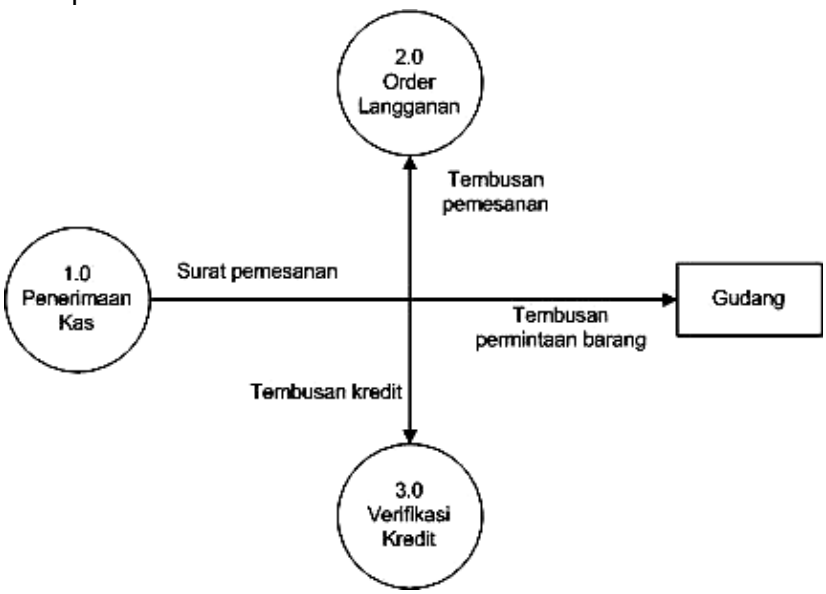
yang sama, maka harus dianggap sebagai suatu aliran data yang tunggal.



Gambar 10. Packet of Data

2.3.2. *Diverging Data Flow*

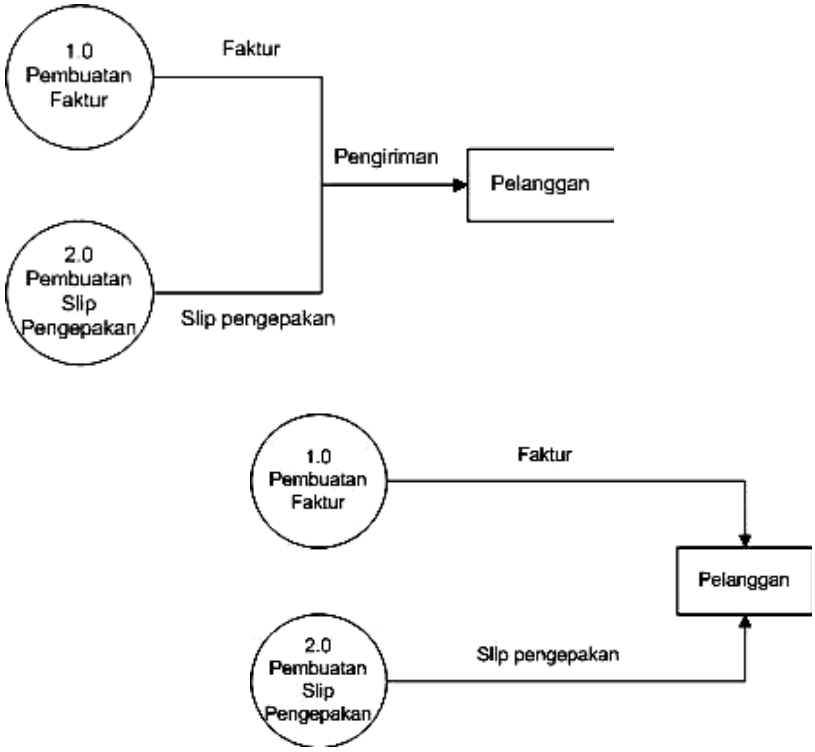
Bila dari suatu sumber mengalir data yang menyebar ke tujuan yang berbeda, menunjukan bahwa aliran data tersebut merupakan tembusan dari aliran data.



Gambar 11. Diverging DFD

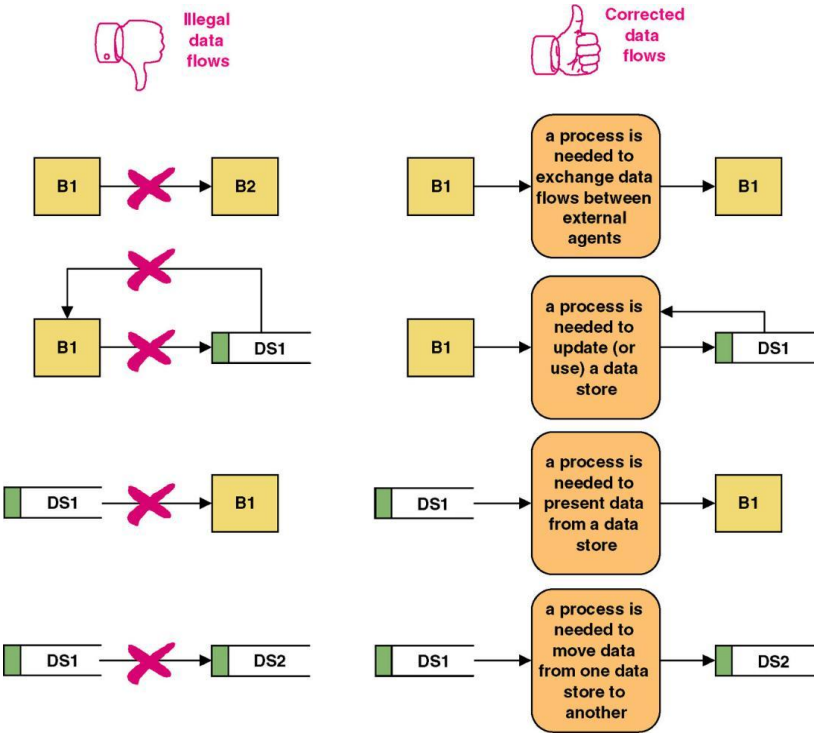
2.3.3. *Convergen Data Flow*

Data yang mengalir dari sumber yang berbeda menuju ke tujuan yang sama.



Gambar 12. Convergen DFD

Beberapa hal yang perlu diperhatikan



2.4 Data Store

Merupakan tempat penyimpanan data yang dapat berupa suatu file atau suatu sistem database dari suatu komputer, suatu arsip/dokumen, suatu agenda/buku.

3. Desain Perangkat Lunak

Desain perangkat lunak adalah tugas, tahapan atau aktivitas yang difokuskan pada spesifikasi detil dari solusi berbasis computer (Whitten et al, 2004).

Disain perangkat lunak sering juga disebut sebagai *physical design*. Jika tahapan analisis sistem menekankan pada masalah bisnis (*business rule*), maka sebaliknya disain perangkat lunak fokus pada sisi teknis dan implementasi sebuah perangkat lunak (Whitten et al, 2004).

Output utama dari tahapan disain perangkat lunak adalah spesifikasi disain. Spesifikasi ini meliputi spesifikasi disain umum yang akan disampaikan kepada *stakeholder* sistem dan spesifikasi disain rinci yang akan digunakan pada tahap implementasi. Spesifikasi disain umum hanya berisi gambaran umum agar *stakeholder* sistem mengerti akan seperti apa perangkat lunak yang akan dibangun. Biasanya diagram DFD tentang perangkat lunak yang baru merupakan

point penting dibagian ini. Spesifikasi disain rinci atau kadang disebut disain arsitektur rinci perangkat lunak diperlukan untuk merancang sistem sehingga memiliki konstruksi yang baik, proses pengolahan data yang tepat dan akurat, bernilai, memiliki aspek *user friendly* dan memiliki dasar-dasar untuk pengembangan selanjutnya.

Desain arsitektur ini terdiri dari desain database, desain proses, desain *user interface* yang mencakup desain input, output *form* dan *report*, desain *hardware*, *software* dan jaringan. Desain proses merupakan kelanjutan dari pemodelan proses yang dilakukan pada tahapan analisis.

4. Implementasi

Implementasi adalah tahapan menerjemahkan hasil disain logis dan fisik ke dalam kode-kode program komputer.

Implementasi atau pengkodean adalah proses menterjemahkan dokumen hasil desain menjadi baris-baris perintah bahasa pemrograman komputer. Semakin baik hasil analisis dan disain yang dilakukan, maka proses pengkodean ini akan lebih mudah dilakukan.

5. Testing dan Evaluasi

Pengujian software melibatkan semua kelompok pengguna yang telah direncanakan pada tahap sebelumnya. Pengujian tingkat penerimaan terhadap perangkat lunak akan berakhir ketika dirasa semua kelompok pengguna menyatakan bisa menerima perangkat lunak tersebut berdasarkan kriteria-kriteria yang telah ditetapkan.

Pengujian software adalah proses untuk memastikan apakah semua fungsi sistem bekerja dengan baik, dan mencari apakah masih ada kesalahan pada sistem.

Pengujian atau testing software sangat penting untuk dilakukan. Pengujian ini bertujuan untuk menjamin kualitas software, dan juga menjadi peninjauan terakhir terhadap spesifikasi, disain dan pengkodean.

Terdapat dua pendekatan dalam melakukan pengujian software , yaitu :

1. Pendekatan *black-box testing*

   Pendekatan ini melakukan pengujian terhadap fungsi operasional software. Pendekatan ini biasanya dilakukan oleh penguji yang tidak ikut serta dalam pengkodean software.

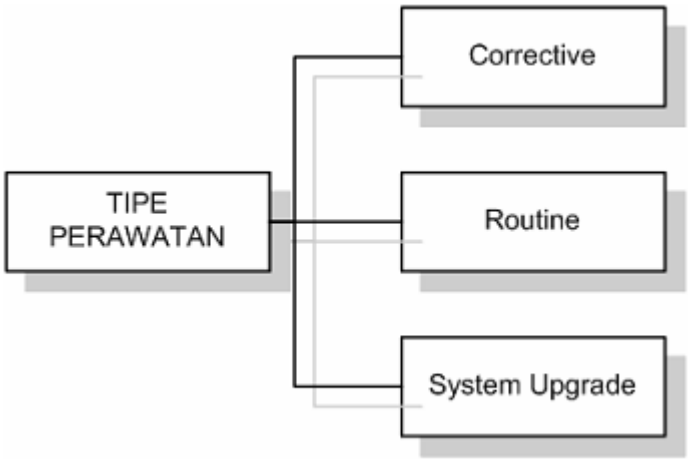
2. Pendekatan *white-box testing*

   Metoda ini dilakukan oleh orang yang memahami cara kerja operasi internal software yang membentuk keseluruhan operasi software.

6. Pemeliharaan

Perangkat lunak akan mengalami perubahan setelah diterapkan di lingkungan pengguna. Perubahan yang terjadi bisa disebabkan oleh kesalahan-kesalahan, atau perangkat lunak harus disesuaikan untuk mengakomodasi perubahan-perubahan di dalam lingkungan eksternalnya, atau karena pelanggan membutuhkan perkembangan fungsional atau unjuk kerja. Pemeliharaan perangkat lunak mengaplikasikan lagi setiap fase program sebelumnya dan tidak membuat yang baru lagi

Ada beberapa tipe pemeliharaan yang biasa dikenal dalam dunia perangkat lunak seperti terlihat pada diagram di Gambar di bawah ini :



Gambar 13. Tipe-tipe pemeliharaan.

- Tipe pemeliharaan *corrective* dilakukan jika terjadi kesalahan atau biasa dikenal sebagai *bugs*. Pemeliharaan bisa dilakukan dengan memperbaiki kode program, menambah bagian yang dirasa perlu atau malah menghilangkan bagian-bagian tertentu.
- Tipe pemeliharaan *routine* biasa juga disebut *preventive maintenance* dilakukan secara rutin untuk melihat kinerja perangkat lunak ada atau tidak ada kesalahan.
- Tipe pemeliharaan sistem *upgrade* dilakukan jika ada perubahan dari komponen-komponen yang terlibat dalam perangkat lunak tersebut. Sebagai contoh perubahan platform sistem operasi dari versi lama ke versi baru menyebabkan perangkat lunak harus di-*upgrade*.

Bab 4. Pemodelan Perangkat Lunak

4

Pemodelan perangkat lunak digunakan untuk memberi gambaran kepada *stake holder* (pengembang, pengguna, manajemen) tentang perangkat lunak yang akan dikembangkan.

Ada 3 alasan melakukan pemodelan sistem :

- Dapat fokus pada aspek-aspek yang penting dalam pengembangan perangkat lunak tanpa harus terlibat terlalu jauh
- Dapat mendiskusikan perubahan dan koreksi terhadap kebutuhan pengguna dengan resiko dan biaya yang minimal
- Dapat menunjukkan pemahaman analis terhadap kebutuhan pengguna dan membantu desainer serta programer dalam implementasi perangkat lunak.

Terdapat sejumlah cara yang dapat digunakan untuk merepresentasikan perangkat lunak melalui diagram, misalnya *flowchart*, HIPO (*Hierarchy Input Process Output*), *Decision Tables*, DFD (*Data Flow Diagam*), *State Transition Diagram*, UML dan lain-lain. Bagian ini akan menjelaskan tentang DFD.

1. Data Flow Diagram

Data Flow Diagram (DFD) adalah representasi grafik dari sebuah sistem. DFD menggambarkan komponen-komponen sebuah sistem, aliran-aliran data di mana komponen-komponen tersebut, dan asal, tujuan, dan penyimpanan dari data tersebut.

Ada 3 (tiga) jenis DFD, yaitu ;

- Context Diagram (CD)
- DFD Fisik
- DFD Logis

Context Diagram

Context Diagram adalah data flow diagram tingkat atas (DFD Top Level), yaitu diagram yang paling tidak detail, dari sebuah sistem atau perangkat lunak yang menggambarkan aliran-aliran data ke dalam dan ke luar sistem atau perangkat lunak dan ke dalam dan ke luar entitas-entitas eksternal. (CD menggambarkan sistem atau perangkat lunak dalam satu lingkaran dan hubungan dengan entitas luar. Lingkaran tersebut menggambarkan keseluruhan proses dalam sistem atau perangkat lunak).

Beberapa hal yang harus diperhatikan dalam menggambar CD:

- Terminologi sistem :
    - Batas Sistem adalah batas antara "daerah kepentingan sistem".
    - Lingkungan Sistem adalah segala sesuatu yang berhubungan atau mempengaruhi sistem tersebut.
    - *Interface* adalah aliran yang menghubungkan sebuah sistem dengan linkungan sistem tersebut.
- Menggunakan satu simbol proses

  Yang masuk didalam lingkaran konteks (simbol proses) adalah kegiatan pemrosesan informasi (Batas Sistem). Kegiatan informasi adalah mengambil data dari file, mentransformasikan data, atau melakukan filing data, misalnya mempersiapkan dokumen, memasukkan, memeriksa, mengklasifikasi, mengatur, menyortir, menghitung, meringkas data, dan melakukan filing data (baik yang melakukan secara manual maupun yang dilakukan secara terotomasi).
- Nama/keterangan di simbol proses tersebut sesuai dengan fungsi sistem tersebut,
- Antara Entitas Eksternal/Terminator tidak diperbolehkan komunikasi langsung
- Jika terdapat termintor yang mempunyai banyak masukan dan keluaran, diperbolehkan untuk digambarkan lebih dari satu sehingga mencegah penggambaran yang terlalu rumit, dengan memberikan tanda asterik (*) atau garis silang (#).
- Jika Terminator mewakili individu (personil) sebaiknya diwakili oleh

peran yang dipermainkan personil tersebut.

- Aliran data ke proses dan keluar sebagai output keterangan aliran data berbeda

DFD Fisik

Adalah representasi grafik dari sebuah perangkat lunak yang menunjukan entitas-entitas internal dan eksternal dari sistem tersebut, dan aliran-aliran data ke dalam dan keluar dari entitas-entitas tersebut. Entitas-entitas internal adalah personel, tempat (sebuah bagian), atau mesin (misalnya, sebuah komputer) dalam sistem tersebut yang mentransformasikan data. Maka DFD fisik tidak menunjukkan apa yang dilakukan, tetapi menunjukkan dimana, bagaimana, dan oleh siapa proses-proses dalam sebuah sistem dilakukan.

Perlu diperhatikan didalam memberikan keterangan di lingkaran-lingkaran (simbol proses) dan aliran-aliran data (simbol aliran data) dalam DFD fisik menggunakan label/keterangan dari kata benda untuk menunjukan bagaimana sistem mentransmisikan data antara lingkaran-lingkaran tersebut.

Misal :

Aliran Data : Kas, Formulir 66W, Slip Setoran

Proses : Cleck Penjualan, Kasir, Pembukuan, dll.

DFD Logis

Adalah representasi grafik dari sebuah perangkat lunak yang menunjukkan proses-proses dalam sistem tersebut dan aliran-aliran data ke dalam dan ke luar dari proses-proses tersebut. Kita menggunakan DFD logis untuk membuat dokumentasi sebuah perangkat lunak karena DFD logis dapat mewakili logika tersebut, yaitu apa yang dilakukan oleh perangkat lunak tersebut, tanpa perlu menspesifikasi dimana, bagaimana, dan oleh siapa proses-proses dalam sistem tersebut dilakukan.

Keuntungan dari DFD logis dibandingkan dengan DFD fisik adalah dapat memusatkan perhatian pada fungsi-funsi yang dilakukan perangkat lunak.

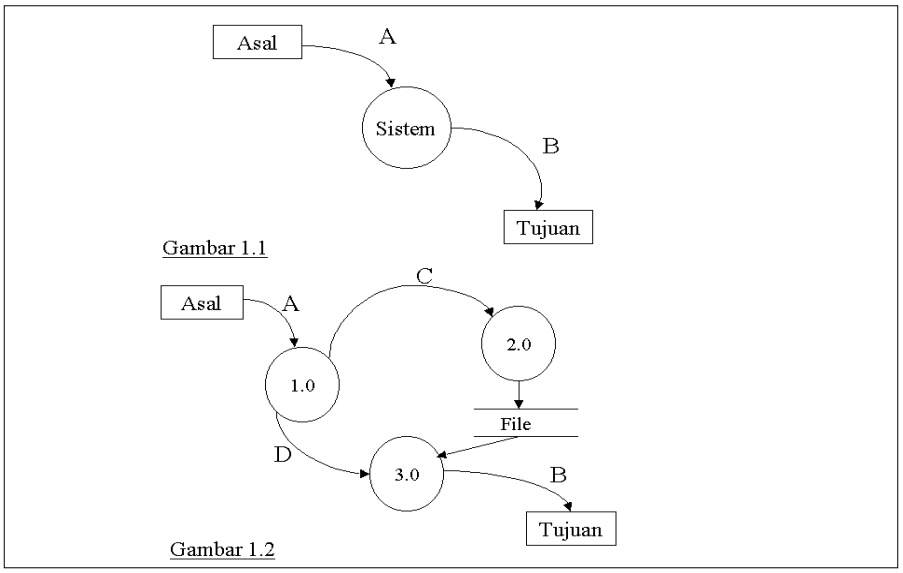
Perlu diperhatikan di dalam pemberian Keterangan/ Label;

- Lingkaran-lingkaran (simbol proses) menjelaskan apa yang dilakukan sistem
  Misal : Menerima Pembayaran, Mencatat Penjualan, Membandingkan

kas dan Daftar Penerimaan, Mempersiapkan Setoran, dll.

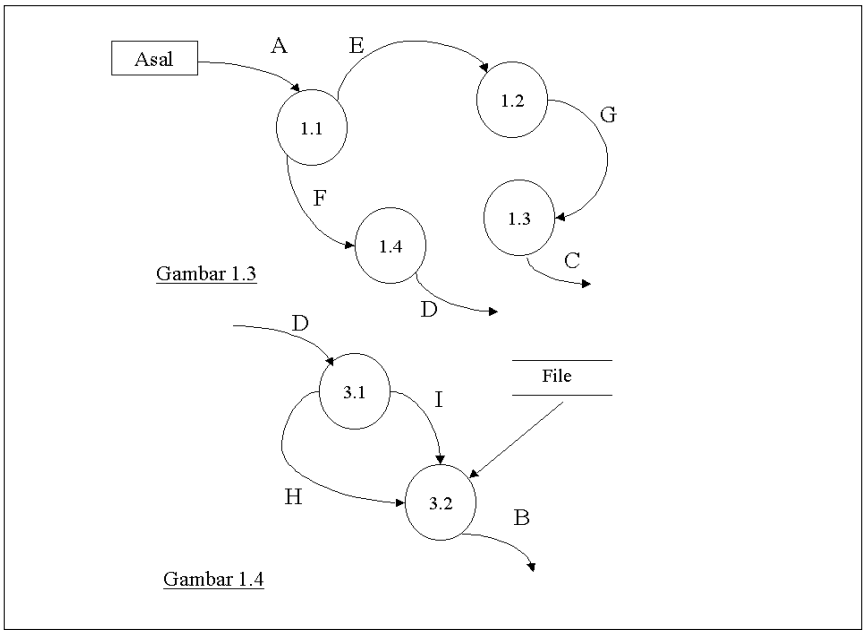
- Aliran-aliran data (simbol aliran data) menggambarkan sifat data.
  Misal : Pembayaran (bukan “Cek”, “Kas”, “ Kartu Kredit”
  Jurnal Penjualan (bukan “Buku Penjualan”), dll

Contoh: Context Diagram dan Diagram Leveled



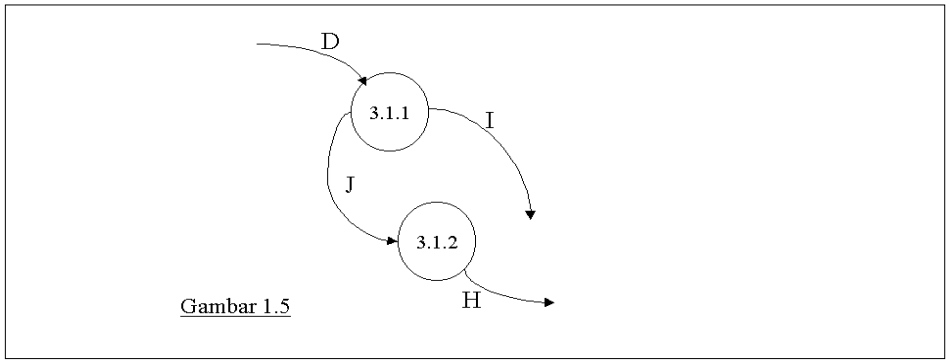
Gambar 1.1

Gambar 1.2



Gambar 1.3

Gambar 1.4

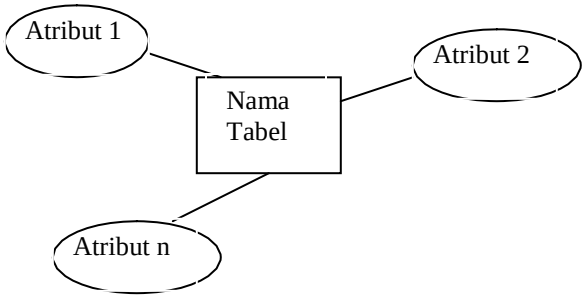


Gambar 1.5

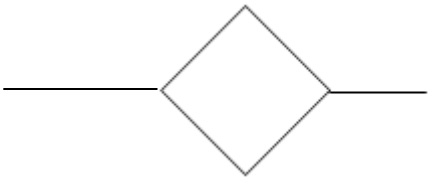
2. Entity Relational Diagram

Dalam melakukan perancangan basisdata relasional, biasa dipergunakan diagram E-R. Komponen-komponen dalam model E-R adalah :

1. Entity Set, merupakan kumpulan dari entity yang memiliki atribut-atribut yang sama. Entity dapat diartikan sebagai sesuatu yang dapat dibedakan dari yang lain. Dalam model E-R, biasa digambarkan sebagai berikut



2. Relationship Set, merupakan himpunan hubungan-hubungan antar entitas-entitas dari dua entity set. Relationship Set dilambangkan sebagai berikut



Bab 5. Unified Modeling Language

5

Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem perangkat lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. (Wahono, R.S, 2003).

Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi perangkat lunak, dimana aplikasi tersebut dapat berjalan pada perangkat keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan perangkat lunak dalam bahasa-bahasa berorientasi obyek seperti C++, Java, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C.

Konsep dasar UML

Abstraksi konsep dasar UML yang terdiri dari *structural classification*, *dynamic behavior*, dan *model management*, bisa kita pahami dengan mudah apabila kita melihat gambar diatas dari *Diagrams*.

Main concepts bisa kita pandang sebagai term yang akan muncul pada saat kita membuat diagram. Dan view adalah kategori dari diagaram tersebut.

Konsepsi dasar UML bisa dirangkumkan dalam gambar 14.

Major Area	*View*	*Diagrams*	*Main Concepts*
structural	static view	class diagram	class, association, generalization, dependency, realization, interface
	use case view	use case diagram	use case, actor, association, extend, include, use case generalization
	implementation view	component diagram	component, interface, dependency, realization
	deployment view	deployment diagram	node, component, dependency, location
dynamic	state machine view	statechart diagram	state, event, transition, action
	activity view	activity diagram	state, activity, completion transition, fork, join
	interaction view	sequence diagram	interaction, object, message, activation
		collaboration diagram	collaboration, interaction, collaboration role, message
model management	model management view	class diagram	package, subsystem, model
extensibility	all	all	constraint, stereotype, tagged values

Gambar 14. Konsepsi Dasar UML

Untuk menguasai UML, sebenarnya cukup dua hal yang harus kita perhatikan:

1. Menguasai pembuatan diagram UMLL
2. Menguasai langkah-langkah dalam analisa dan pengembangan dengan UML.

Langkah-langkah Penggunaan UML :

1. Buatlah daftar *business process* dari level tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul.
2. Petakan *use case* untuk tiap *business process* untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh sistem. Kemudian perhalus use case diagram dan lengkapi dengan *requirement, constraints* dan catatan-catatan lain.
3. Buatlah *deployment diagram* secara kasar untuk mendefinisikan arsitektur fisik sistem.
4. Definisikan *requirement* lain (non-fungsional, *security* dan sebagainya) yang juga harus disediakan oleh sistem.
5. Berdasarkan *use case diagram*, mulailah membuat *activity diagram*.
6. Definisikan obyek-obyek level atas (*package* atau *domain*) dan buatlah *sequence* dan/ atau *collaboration diagram* untuk tiap alir pekerjaan. Jika sebuah *use case* memiliki kemungkinan alir normal dan error, buatlah satu diagram untuk masing-masing alir.
7. Buatlah rancangan *user interface* model yang menyediakan antarmuka bagi pengguna untuk menjalankan skenario *use case*.
8. Berdasarkan model-model yang sudah ada, buatlah *class diagram*. Setiap *package* atau *domain* dipecah menjadi hirarki *class* lengkap dengan atribut dan metodanya. Akan lebih baik jika untuk setiap *class* dibuat *unit test* untuk menguji fungsionalitas *class* dan interaksi dengan *class* lain.
9. Setelah *class diagram* dibuat, kita dapat melihat kemungkinan pengelompokan *class* menjadi komponen-komponen. Karena itu buatlah component diagram pada tahap ini. Juga, definisikan tes integrasi untuk setiap komponen meyakinkan ia berinteraksi dengan baik.
10. Perhalus *deployment diagram* yang sudah dibuat. Detilkan kemampuan dan *requirement* piranti lunak, sistem operasi, jaringan,

dan sebagainya. Petakan komponen ke dalam node.

11. Mulailah membangun sistem. Ada dua pendekatan yang dapat digunakan :
    - Pendekatan *use case*, dengan meng-*assign* setiap *use case* kepada tim pengembang tertentu untuk mengembangkan *unit code* yang lengkap dengan tes.
    - Pendekatan komponen, yaitu meng-*assign* setiap komponen kepada tim pengembang tertentu.
12. Lakukan uji modul dan uji integrasi serta perbaiki model berserta *code*nya. Model harus selalu sesuai dengan *code* yang aktual.
13. Piranti lunak siap dirilis.

Use Case Diagram

Use case merupakan *design* yang berfokus pada user dan tugas- tugas *user* untuk memenuhi keinginan *user*. (Thimoty C, 2002)

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng- *create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. (Wahono, R.S,2003).

Use case diagram dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.

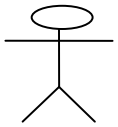
Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal.

Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*.

Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

Use case ini terdiri dari :

- *Actor* : pemakai sistem/ sesuatu yang berinteraksi dengan sistem merepresentasikan pesan, bukan pemakai individual.



- *Use case* : cara spesifik penggunaan sistem oleh aktor.

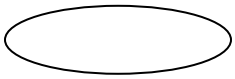


Diagram use case :

- Merupakan salah satu diagram untuk memodelkan aspek perilaku sistem.
- Masing-masing menunjukkan sekumpulan use case, aktor, dan hubungannya.
- Untuk memvisualisasikan dan mendokumentasikan kebutuhan perilaku sistem.
- Melibatkan : sistem, aktor, use case, dan relasi.

Tujuan utama memodelkan use case :

- Memutuskan dan mendeskripsikan kebutuhan fungsional sistem.
- Memberikan deskripsi jelas dan konsisten dari apa yang harus dilakukan.
- Menyediakan basis untuk melakukan pengujian sistem yang memverifikasi sistem.

- Menyediakan kemampuan melacak kebutuhan fungsi analistis menjadi class-class, operasi-operasi, dan aktual sistem.

Ciri-ciri use case :

- Pola perilaku yang harus dipenuhi oleh sistem
- Sekuen transaksi terhubung yang dilakukan aktor dan sistem
- Memberikan sesuatu yang berharga (informasi) bagi *user*

Kegunaan use case :

- Menangkap kebutuhan sistem
- Berkomunikasi dengan pemakai akhir dan pakar domain masalah
- Pengkajian sistem

Activity Diagram

Diagram activity seperti diagram state, merupakan diagram yang dapat digunakan untuk memahami alur kerja dari obyek/ komponen yang dilakukan. Diagram activity dapat digunakan untuk memvisualisasikan interelasi dan interaksi antara use case yang berbeda, serta sering dipakai untuk mengasosiasikan dengan *class* yang berbeda. Kekuatan diagram activity adalah mempresentasikan *concurrent activity.* (Thimoty C, 2002).

Diagram Activity menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. Diagram Activity juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. (Wahono, R.S, 2003)

Diagram Activity merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu Diagram Activity tidak menggambarkan behaviour internal sebuah sistem(dan interaksi antar

subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal.

Diagram Activity dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan obyek mana yang bertanggung jawab untuk aktivitas tertentu.

Class Diagram

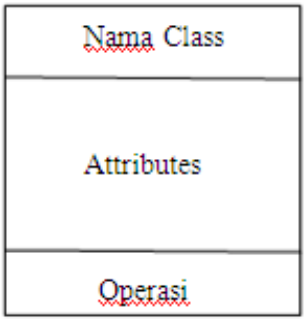
Diagram class merupakan diagram yang terdiri dari sekumpulan obyek yang memiliki atribut-atribut dan method. Obyek adalah sebuah benda/ unit/ sifat kerja yang memiliki atribut-atribut. Operasi adalah prosedural *abstraction* yang menspesifikasi tipe dari perilaku yang terdiri dari fungsi. *Super class* adalah *class* induk yang nantinya mempunyai *class-class* yang terdiri dari *class* dan *subclass*. (Thimoty C, 2002).

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). Jadi, *Class diagram* menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. (Wahono, R.S, 2003)

Diagram class terdiri dari :

a. Nama *class*
b. Attributes

c.Operasi



Gambar 15. Diagram Class

Atribut dan metoda dapat memiliki salah satu sifat berikut :

- *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan
- *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya
- *Public*, dapat dipanggil oleh siapa saja

Elemen penting dalam diagram class adalah :

- *Class*
- Antar muka
- Kolaborasi
- Hubungan (*relationship*)

Hubungan Antar Class

1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah *query* antar *class*.
2. Agregasi, yaitu hubungan yang menyatakan bagian (“terdiri atas..”).
3. Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan

metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.

4. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-*passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram* yang akan dijelaskan kemudian.

Kegunaan diagram class :

- Memodelkan kosakata sistem
- Memodelkan distribusi tanggung jawab dari sistem
- Memodelkan tipe
- Memodelkan entitas bukan perangkat lunak
- Memodelkan kolaborasi
- Memodelkan sistem basis data logic

Kriteria diagram class yang baik :

- Fokus
- Essensial
- Konsisten
- Tidak hilang

Pentingnya UML untuk diagram class :

- *Class*
- *Association* (relasi antar *class*)
- Atribut
- *Operation*
- *Generalization*

Sequence Diagram

Diagram sequence adalah diagram yang menunjukkan urutan dari pertukaran pesan antar obyek dan tugas yang dilakukan, dan merupakan diagram yang menjelaskan urutan kejadian dari sistem (urutan lacak kejadian). (Thimoty C, 2002).

Diagram sequence menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. Diagram sequence terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). (Wahono, R.S, 2003).

Diagram sequence biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan.

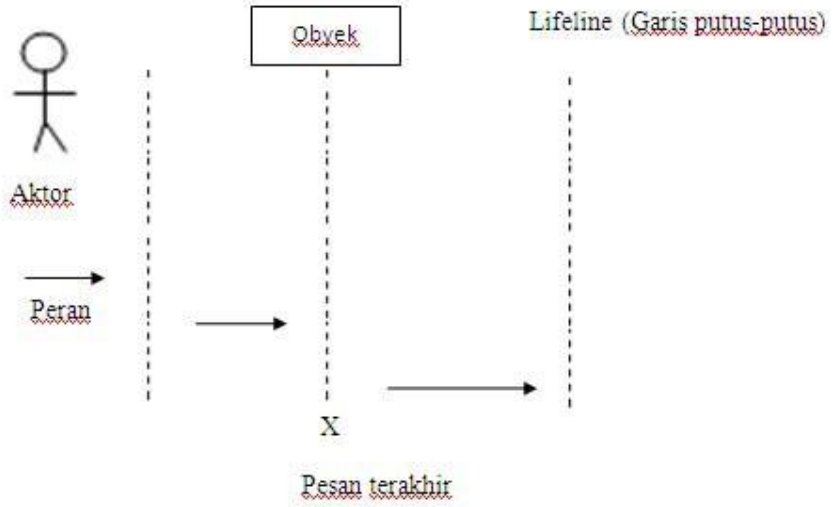
Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah message.

Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk objek *boundary, controller* dan *persistent entity*.

Diagram sequence digunakan untuk :

- *Overview* perilaku sistem
- Menunjukkan obyek-obyek yang diperlukan
- Mendokumentasikan skenario dari diagram use case
- Memberikan jalur-jalur pengaksesan

Notasi diagram sequence :



Gambar 16. Diagram Sequence

Daftar Pustaka

Raymond McLeod, Jr., *"Management Information System A Study of Computer Based Information Systems"*, Prentice Hall, Inc, 1995.

Roger S. Pressman, "Software Engineering, a Practitioner's Approach" Fourth Edition, McGraw Hill, 1997.

Roger S. Pressman, "Software Engineering, A Beginner's Guide", McGraw Hill, 1998.

Barbee Teasley Mynatt, *"Software Engineering with Student Project Guidance"*, Prentice Hall, Inc, 1990.