

BAB 1

PENDAHULUAN

1.1. Pengertian Rekayasa Perangkat Lunak

Definisi Rekayasa

Engineering = rekayasa

- Pemakaian ‘science’ untuk menyelesaikan ‘masalah praktis’
- Dari tidak ada menjadi ada

Definisi Perangkat Lunak

Ada beberapa definisi perangkat lunak yang pernah dikemukakan antara lain :

- Software = Perangkat lunak
 - Kumpulan program komputer dengan fungsi tertentu
- Perangkat lunak adalah
 1. Instruksi (program komputer) yang bila dieksekusi dapat menjalankan fungsi tertentu,
 2. Struktur data yang dapat membuat program memanipulasi informasi, dan
 3. Dokumen yang menjelaskan operasi dan penggunaan program (Pressman, 1997).
- Perangkat lunak adalah program komputer, prosedur, aturan, dan dokumentasi yang berkaitan serta data, yang bertalian dengan operasi suatu sistem komputer (IEEE, 1993).

Karakteristik Perangkat Lunak

Perangkat lunak lebih dikenal sebagai elemen logik daripada fisik, oleh karena itu perangkat lunak memiliki karakteristik yang berbeda dari perangkat keras :

1. Perangkat lunak dikembangkan atau direkayasa, jadi tidak diproduksi dalam pengertian klasik.
2. Merupakan produk yang unik (tidak ada seri produksi).
3. Perangkat lunak tidak pernah akan rusak/aus karena selalu diperbaharui
4. Tidak terlihat (*invisible*).
5. Perangkat lunak pada umumnya dibangun sesuai keinginan, jadi tidak dibentuk dari komponen yang sudah ada.
6. Fleksibel, sehingga mudah dimodifikasi.
7. Dihubungkan (*linked*) dengan sistem komputer.

Rekayasa perangkat lunak (*software engineering*) adalah suatu proses rancang bangun. Beberapa definisi tentang rekayasa perangkat lunak :

- Pembentukan dan penggunaan prinsip rekayasa (*engineering*) untuk mendapatkan perangkat lunak secara ekonomis namun andal dan dapat bekerja secara efisien pada komputer (Fritz Bauer, 1968).
- Penerapan pendekatan yang sistematis, disiplin, dan terukur untuk pengembangan, operasi, dan pemeliharaan perangkat lunak (IEEE, 1993).
- Suatu disiplin yang mengintegrasikan proses/prosedur, metode, dan perangkat tools untuk pembangunan perangkat lunak komputer (Pressman, 97).
- Merupakan aplikasi dari prinsip-prinsip sains untuk

- Mengurutkan transformasi masalah menjadi solusi yang dapat bekerja dengan baik
- Urutan pemeliharaan perangkat lunak tersebut sampai tidak dapat digunakan lagi (Alan M. Davis)

Proses RPL dimulai jauh sebelum “*Coding*” dilakukan dan berlanjut terus setelah versi awal dari program selesai dikerjakan.

Tujuan dari RPL adalah

- a. Menghasilkan sebuah perangkat lunak yang berkualitas. Yang dimaksud dengan berkualitas dapat dilihat dari tiga sisi, sisi sponsor (individu atau organisasi yang telah mengeluarkan biaya dalam pembangunan perangkat lunak), sisi pemakai (siapa pun yang menggunakan perangkat lunak tersebut), sisi *maintainer / modifier* (yang memelihara dan memodifikasi perangkat lunak tersebut). Untuk lebih jelasnya lihat gambar 1.1.

Sisi Sponsor :

Tujuan utama sponsor adalah menghasilkan dan atau menghemat uang. Sponsor ingin menggunakan perangkat lunak tersebut untuk meningkatkan produktivitas organisasi. Sponsor mengharapkan untuk dapat menghasilkan sebuah layanan dengan biaya yang rendah tetapi masuk akal. Karena itu sistem yang dibuat harus handal, fleksibel dan efisien. Selain itu biaya dari pemeliharaan, modifikasi dan peningkatan dari sistem tersebut harus serendah mungkin.

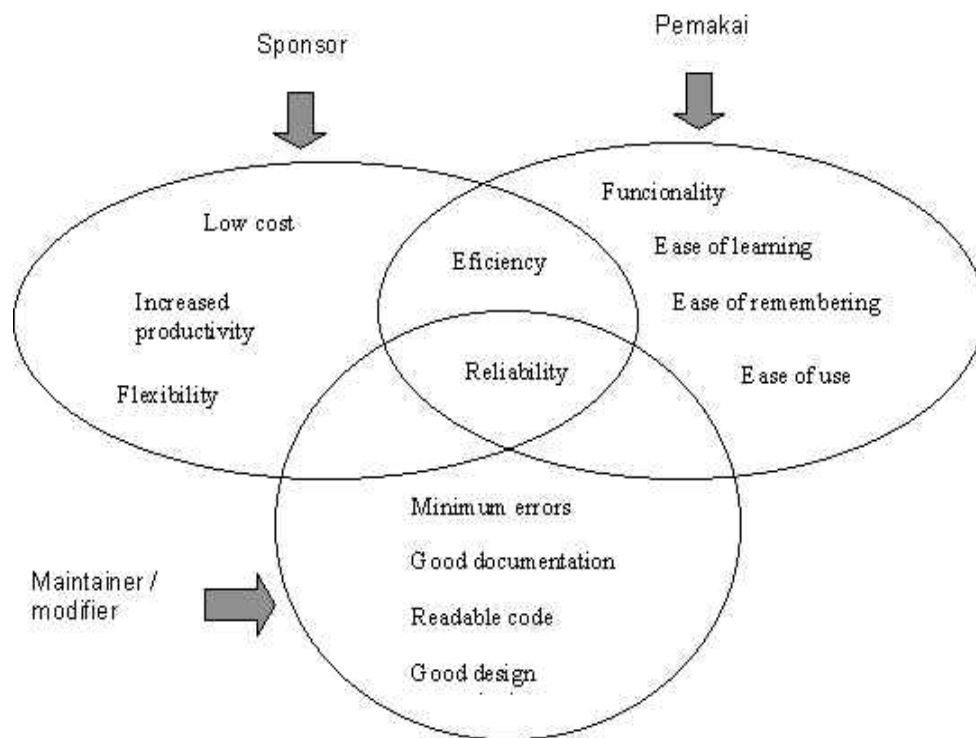
Sisi Pemakai :

Bagi pemakai perangkat lunak adalah alat untuk membantu menyelesaikan tugas-tugasnya. Karena itu perangkat lunak harus menyediakan fungsi-fungsi yang dibutuhkan oleh pemakai. Perangkat lunak juga harus handal dan efisien, perangkat lunak harus dapat menghasilkan *output* yang konsisten. Selain itu pemakai harus merasa perangkat lunak yang dibuat mudah untuk dipelajari, mudah digunakan dan mudah untuk diingat.

Sisi *Maintainer/modifier* :

Yang diinginkan oleh *maintainer/modifier* adalah perangkat lunak tersebut memiliki sangat sedikit *error* pada saat *penginstallan* pertama (catatan : sangat kecil kemungkinannya untuk menghasilkan perangkat lunak yang 100 % bebas dari *bug*). Selain itu perangkat lunak tersebut harus terdokumentasi dengan baik. *Source code* juga harus mudah dibaca, terstruktur dan dirancang dengan baik dan bersifat modular.

- b. Tujuan kedua dari RPL adalah menghasilkan perangkat lunak dengan biaya yang efisien.
- c. Sedangkan tujuan ketiga dari RPL adalah menghasilkan perangkat lunak tepat pada waktunya.



Gambar 1.1 Parameter Perangkat Lunak Yang Berkualitas Berdasarkan Sudut Pandang

Rekayasa perangkat lunak merupakan suatu teknologi berlapis, yaitu proses/prosedur, metode, dan perangkat, dengan fokus kualitas sebagai dasar utamanya.

Mengapa Rekayasa Perangkat Lunak ?

Adanya krisis perangkat lunak (NATO conference, 1968) :

- Perangkat lunak lebih banyak menyebabkan masalah daripada menyelesaikannya.
- Peningkatan ukuran perangkat lunak tanpa pengorganisasian.
- Perbaikan suatu kesalahan menyebabkan timbulnya kesalahan lainnya.
- Tidak ada kendali pemeliharaan.

Masalah-masalah perangkat lunak :

- Perangkat lunak telah diselesaikan dan diserahkan (*delivered*) tetapi tidak pernah digunakan (47%).
- Pemakai (*user*) sudah membayar untuk perangkat lunak tetapi tidak pernah jadi dan diserahkan (29,7%).
- Perangkat lunak digunakan setelah dilakukan modifikasi (3%).
- Perangkat lunak digunakan sebagaimana mestinya (2%).

Selain itu faktor pendukung kehadiran rekayasa perangkat lunak adalah :

- Ketidak mampuan untuk memprediksi waktu, usaha dan biaya pada pengembangan perangkat lunak.
- Kualitas perangkat lunak yang kurang baik.

- Perubahan perbandingan (rasio) harga perangkat keras dan perangkat lunak.
- Kemajuan teknologi perangkat keras.
- Kemajuan teknik perangkat lunak.
- Kebutuhan yang meningkat terhadap perangkat lunak.
- Kebutuhan akan perangkat lunak yang lebih besar dan kompleks.

1.2. Pengertian Perangkat Lunak

Jenis-jenis Perangkat Lunak

Dilihat dari sudut pandang fungsinya, perangkat lunak dapat dikelompokkan menjadi :

1. Perangkat lunak sistem

Perangkat lunak yang kegunaannya lebih banyak ditujukan untuk operasional komputer.

- sistem operasi
- penerjemah bahasa pemrograman (*compiler/interpreter*)

2. Perangkat lunak aplikasi

Perangkat lunak yang kegunaannya lebih banyak ditujukan untuk membantu menyelesaikan masalah-masalah yang dihadapi oleh pemakai.

- program paket yang sudah jadi
- program aplikasi buatan sendiri

Sedangkan dilihat dari aplikasinya, perangkat lunak dibedakan menjadi :

1. Perangkat Lunak Sistem (*Sistem Software*)

Sekumpulan program yang ditulis untuk kepentingan program lain, contoh *editor*, *driver* dan lain-lain

2. Perangkat Lunak Waktu Nyata (*Real Time Software*)

Perangkat lunak yang digunakan untuk mengukur/menganalisis atau mengontrol proses pemasukan data dari lingkungan luar sampai menghasilkan laporan yang diinginkan

3. Perangkat Lunak Bisnis (*Business Software*)

Perangkat lunak yang memberikan fasilitas operasi untuk bisnis atau fasilitas pengambilan keputusan manajemen, contoh sistem akuntansi, *inventory*, *payroll* dan lain-lain

4. Perangkat Lunak Rekayasa dan Sains (*Engineering and Scientific Software*)

Perangkat lunak yang digunakan di dalam bidang aplikasi teknik dan rekayasa
Perangkat lunak jenis ini biasanya berhubungan dengan komputasi data numerik, CAD (*Computer Aided Design*), simulasi sistem, dan lain-lain.

5. *Embedded Software*

Perangkat lunak yang digunakan untuk mengontrol suatu produk dan sistem dimana perangkat lunak tersebut disimpan. Biasanya ditempatkan di ROM, contoh Tombol di *Microwave Oven*

6. Perangkat Lunak Komputer Pribadi (*Personal Computer Software*)

Banyak digunakan pada aplikasi yang bersifat perorangan, contohnya : pengolah kata, *spreadsheet*, *game*, DBMS dan lain-lain.

7. Perangkat Lunak Intelegensia Buatan (*Artificial Intelligent Software*)

Dibuat dengan menggunakan teknik algoritma non-numerik untuk memecahkan masalah yang kompleks, digunakan dalam bidang aplikasi kecerdasan buatan, contohnya : *game, expert sistem, neural network*, Turbo Prolog, dan lain-lain

1.3. Mitos Perangkat Lunak

Berkaitan dengan Manajemen :

Biasanya muncul pada manajer yang bertanggung jawab terhadap perangkat lunak. Mereka biasanya ditekan untuk menjaga budget, jadwal harus selalu terpenuhi dan harus meningkatkan kualitas. Mitos tersebut antara lain :

Mitos : Kita sudah punya buku yang berisi standard dan prosedur yang banyak untuk pengembangan perangkat lunak. Bukankah hal ini sudah cukup untuk mencari semua yang ingin diketahui ?

Kenyataan : Buku-buku itu memang lengkap, tapi apakah digunakan ? Apakah praktisi perangkat lunak sadar dengan keberadaannya? Apakah cocok dengan pengembangan yang modern? Apakah benar-benar lengkap ? Pada banyak kasus jawabannya adalah tidak.

Mitos : Staff Kami mempunyai alat Bantu pengembangan yang canggih, bahkan dibelikan komputer generasi terbaru.

Kenyataan : Masalah pengembangan perangkat lunak yang berkualitas lebih penting dari sekedar komputer yang terbaru. CASE (*Computer Aided Software Engineering*) tools lebih penting daripada perangkat keras untuk mendapatkan kualitas dan produktifitas yang baik, tapi banyak pengembang perangkat lunak yang tidak menyadarinya.

Mitos : Jika Kita dikejar jadwal, tambah programmer untuk mengejanya

Kenyataan : Membuat perangkat lunak bukan proses mekanis seperti industri manufaktur. Jika kita menambah orang pada proyek yang terlambat itu justru akan lebih terlambat

Berkaitan dengan Klien

Konsumen sering mempercayai mitos karena pembuat perangkat lunak kurang berusaha untuk membetulkan misinformasi ini.

Mitos : Sebuah kalimat umum yang menyatakan objektif sudah cukup untuk memulai menulis program. Kita bias perinci lagi nanti.

Kenyataan : Definisi yang tidak jelas, justru akan menggagalkan usaha pengembangan perangkat lunak. Justru diperlukan deskripsi formal dan detil dari domain informasi, fungsi, performansi, antarmuka, batasan desain, dan kriteria validasi. Karakteristik ini hanya bias didapat melalui komunikasi total antara pelanggan dan pengembang.

Mitos : Kebutuhan proyek akan terus berubah, tapi perubahan ini akan dapat ditanggapi dengan mudah karena perangkat lunak itu bersifat fleksibel

Kenyataan : memang betul kebutuhan perangkat lunak akan berubah, namun dampaknya tergantung pada waktu pemunculannya. Jika muncul pada tahap definisi, pengaruhnya tidak banyak, lebih kebelakang dampaknya akan lebih besar.

Berkaitan dengan Pengembang

Mitos : Setelah program selesai ditulis dan dapat dijalankan, maka tugas sudah selesai.

Kenyataan : Ada yang mengatakan bahwa “Lebih cepat program dibuat, maka lebih lama selesainya”. Dari data industri 50-70% dari usaha pada pembuatan program akan bertambah lagi setelah program dilihat untuk pertama kalinya oleh pelanggan.

Mitos : Selama program belum berjalan, sulit untuk mengetahui kualitasnya

Kenyataan : *Software review* adalah cara efektif untuk mencari *Software defect* daripada tahap pengujian

Mitos : Faktor penentu suksesnya proyek adalah program yang dapat berjalan

Kenyataan : Program hanyalah salah satu komponen dari perangkat lunak. Dokumentasi penting sebagai dasar pengembangan yang sukses serta sebagai penunjuk untuk pemeliharaan perangkat lunak

BAB 2

METODOLOGI PENGEMBANGAN PERANGKAT LUNAK

2.1. Latar Belakang

Pada pertengahan tahun 60 sampai 70-an banyak dikembangkan sistem-sistem perangkat lunak yang besar. Sistem-sistem yang dikembangkan ini banyak yang dipandang tidak efisien, kurang berhasil, bahkan banyak yang gagal. Kegagalan ini disebabkan karena tidak tersedianya teknik pengembangan perangkat lunak yang baik. Pada awal tahun 70-an mulai muncul metodologi-metodologi pengembangan perangkat lunak yang cukup baik.

Pengembangan perangkat lunak dapat diartikan sebagai proses membuat suatu perangkat lunak baru untuk menggantikan perangkat lunak lama secara keseluruhan atau memperbaiki perangkat lunak yang telah ada. Agar lebih cepat dan tepat dalam mendeskripsikan solusi dan mengembangkan perangkat lunak, juga hasilnya mudah dikembangkan dan dipelihara, maka pengembangan perangkat lunak memerlukan suatu metodologi khusus. Metodologi pengembangan perangkat lunak adalah suatu proses pengorganisasian kumpulan metode dan konvensi notasi yang telah didefinisikan untuk mengembangkan perangkat lunak. Secara prinsip bertujuan untuk membantu menghasilkan perangkat lunak yang berkualitas. Penggunaan suatu metodologi sesuai dengan persoalan yang akan dipecahkan dan memenuhi kebutuhan pengguna akan menghasilkan suatu produk perakayasaan yang berkualitas dan terpelihara serta dapat menghindari masalah-masalah yang sering terjadi seperti estimasi penjadwalan dan biaya, perangkat lunak yang tidak sesuai dengan keinginan pengguna dan sebagainya.

Metodologi pengembangan perangkat lunak (atau disebut juga model proses atau paradigma rekayasa perangkat lunak) adalah suatu strategi pengembangan yang memadukan proses, metode, dan perangkat (*tools*).

Menurut Pressman (1997) Komponen metodologi pengembangan perangkat lunak dapat dibagi dalam tiga unit, yaitu :

1. Metode, yaitu suatu cara atau teknik pendekatan yang sistematis yang dipergunakan untuk mengembangkan perangkat lunak. Metode ini mencakup : Perencanaan proyek dan perkiraan, analisis keperluan sistem dan perangkat lunak, perancangan struktur data, arsitektur program, prosedur algoritma, Coding, uji coba dan pemeliharaan.
2. Alat bantu (*Tools*), yaitu alat-alat (manual atau otomatis) yang mendukung pengembangan perangkat lunak. Terdapat 2 alat Bantu yang dapat digunakan yaitu : alat Bantu manual dan alat Bantu otomatis.
3. Prosedur, yang dipergunakan untuk mendefinisikan urutan pekerjaan (daur) dari metode dan alat bantu tersebut.

Secara umum daur hidup pengembangan perangkat lunak meliputi tahapan-tahapan atau aktivitas pengembangan yang terdiri dari tahap analisis, tahap perancangan, tahap implementasi serta tahap pengujian dan perawatan perangkat lunak. Tahap analisis dan perancangan merupakan tahapan awal yang penting dalam suatu paradigma pengembangan perangkat lunak, karena sangat mempengaruhi tahapan selanjutnya. Sehingga jika terjadi kesalahan pada tahap analisis dan perancangan, maka akan terdapat juga kesalahan pada tahap implementasi dan tahapan-tahapan selanjutnya. Tahap implementasi perangkat lunak bertujuan untuk menerapkan spesifikasi kebutuhan perangkat lunak ke dalam bahasa pemrograman tertentu. Tahap pengujian perangkat lunak dilakukan untuk menemukan kesalahan (*bug*) yang mungkin terdapat di dalam sebuah perangkat lunak. Sedangkan tahap perawatan perangkat lunak fokusnya adalah pengubahan. Ada tiga pengubahan yaitu : pembetulan, adaptasi (perbaikan terhadap lingkungan) dan perluasan (penambahan karena permintaan pemakai).

2.2. Proses Pengembangan Perangkat Lunak

Proses pengembangan perangkat lunak adalah suatu proses dimana kebutuhan pemakai diterjemahkan menjadi produk perangkat lunak. Proses ini mencakup aktivitas penerjemahan kebutuhan pemakai menjadi kebutuhan perangkat lunak, transformasi kebutuhan perangkat lunak menjadi desain, penerapan desain menjadi kode program, uji coba kode program, dan instalasi serta pemeriksaan kebenaran perangkat lunak untuk operasional (IEEE. 1990). Berdasarkan pengertian tersebut, secara umum dapat dikatakan bahwa proses pengembangan perangkat lunak mengikuti tahap-tahap :

1. Menentukan **APA** yang harus dikerjakan oleh perangkat lunak dalam satu rentang waktu tertentu.
2. Mendefinisikan **BAGAIMANA** perangkat lunak dibuat, mencakup arsitektur perangkat lunaknya, antar muka internal, algoritma, dan sebagainya.
3. Penerapan (penulisan program) dan pengujian unit-unit program.
4. Integrasi dan pengujian modul-modul program.
5. Validasi perangkat lunak secara keseluruhan (pengujian sistem).

2.3. Siklus Pengembangan Perangkat Lunak

Siklus pengembangan perangkat lunak atau sering disebut juga dengan siklus hidup perangkat lunak adalah (IEEE,1987) :

- Periode waktu yang diawali dengan keputusan untuk mengembangkan produk perangkat lunak dan berakhir setelah perangkat lunak diserahkan. Umumnya siklus pengembangan ini terdiri dari tahap analisis kebutuhan, perancangan, penerapan, pengujian, dan instalasi serta pemeriksaan.
- Periode waktu yang diawali dengan keputusan untuk mengembangkan produk perangkat lunak dan berakhir saat produk tidak dapat ditingkatkan lebih jauh lagi oleh pengembang.

2.4. Model Proses Pengembangan Perangkat Lunak

A. Linear Sequential Model

Linear sequential model (atau disebut juga “*classic life cycle*” atau “*waterfall model*”) adalah metode pengembangan perangkat lunak dengan pendekatan sekuensial dengan cakupan aktivitas :

1. Rekayasa sistem dan Analisis (*Sistem Engineering and Analysis*)

Karena perangkat lunak adalah bagian dari sistem yang lebih besar, pekerjaan dimulai dari pembentukan kebutuhan-kebutuhan untuk seluruh elemen sistem dan kemudian memilah mana yang untuk pengembangan perangkat lunak. Hal ini penting, ketika perangkat lunak harus berkomunikasi dengan *hardware*, orang dan basis data

2. Analisis kebutuhan perangkat lunak (*Software Requirements Analysis*)

Pengumpulan kebutuhan dengan fokus pada perangkat lunak, yang meliputi :

Domain informasi, fungsi yang dibutuhkan, unjuk kerja/performansi dan antarmuka. Hasilnya harus didokumentasi dan direview ke pelanggan

3. Perancangan (*Design*)

Ada 4 atribut untuk program yaitu : Struktur Data, Arsitektur perangkat lunak, Prosedur detil dan Karakteristik Antarmuka. Proses desain mengubah kebutuhan-kebutuhan menjadi bentuk karakteristik yang dimengerti perangkat lunak sebelum

dimulai penulisan program. Desain ini harus terdokumentasi dengan baik dan menjadi bagian konfigurasi perangkat lunak.

4. Pembuatan kode (*Coding*)

Penterjemahan perancangan ke bentuk yang dapat dimengerti oleh mesin, dengan menggunakan bahasa pemrograman

5. Pengujian (*Testing*)

Setelah kode program selesai testing dapat dilakukan. Testing memfokuskan pada logika internal dari perangkat lunak, fungsi eksternal dan mencari segala kemungkinan kesalahan dan memeriksa apakah sesuai dengan hasil yang diinginkan.

6. Pemeliharaan (*Maintenance*)

Merupakan bagian paling akhir dari siklus pengembangan dan dilakukan setelah perangkat lunak dipergunakan. Kegiatan :

- ***Corrective Maintenance*** : Mengoreksi kesalahan pada perangkat lunak, yang baru terdeteksi pada saat perangkat lunak dipergunakan
- ***Adaptive Maintenance*** : Penyesuaian dengan lingkungan baru, misalnya sistem operasi atau sebagai tuntutan atas perkembangan sistem komputer, misalnya penambahan *printer driver*
- ***Perfektive Maintenance*** : Bila perangkat lunak sukses dipergunakan oleh pemakai. Pemeliharaan ditujukan untuk menambah kemampuannya seperti memberikan fungsi-fungsi tambahan, peningkatan kinerja dan sebagainya.

Kelemahan model *linear sequential*:

1. Proyek yang sebenarnya jarang mengikuti alur sekuensial seperti diusulkan, sehingga perubahan yang terjadi dapat menyebabkan hasil yang sudah didapat tim harus diubah kembali/iterasi sering menyebabkan masalah baru.
2. Linear sequential model mengharuskan semua kebutuhan pemakai sudah dinyatakan secara eksplisit di awal proses, tetapi kadang-kadang ini tidak dapat terlaksana karena kesulitan yang dialami pemakai saat akan mengungkapkan semua kebutuhannya tersebut.
3. Pemakai harus bersabar karena versi dari program tidak akan didapat sampai akhir rentang waktu proyek.
4. Adanya waktu menganggur bagi pengembang, karena harus menunggu anggota tim proyek lainnya menuntaskan pekerjaannya.

B. Prototyping Model

Pendekatan *prototyping* model digunakan jika pemakai hanya mendefinisikan objektif umum dari perangkat lunak tanpa merinci kebutuhan *input*, pemrosesan dan *output*nya, sementara pengembang tidak begitu yakin akan efisiensi algoritma, adaptasi sistem operasi, atau bentuk antarmuka manusia-mesin yang harus diambil. Cakupan aktivitas dari *prototyping* model terdiri dari :

1. Mendefinisikan objektif secara keseluruhan dan mengidentifikasi kebutuhan yang sudah diketahui.
2. Melakukan perancangan secara cepat sebagai dasar untuk membuat prototype.

3. Menguji coba dan mengevaluasi prototype dan kemudian melakukan penambahan dan perbaikan-perbaikan terhadap prototype yang sudah dibuat.

Kelemahan *prototyping* model :

1. Pelanggan yang melihat working version dari model yang dimintanya tidak menyadari, bahwa mungkin saja prototype dibuat terburu-buru dan rancangan tidak tersusun dengan baik
2. Pengembang kadang-kadang membuat implementasi sembarang, karena ingin working version bekerja dengan cepat

C. RAD (Rapid Application Development) Model

Merupakan model proses pengembangan perangkat lunak secara linear sequential yang menekankan pada siklus pengembangan yang sangat singkat. Jika kebutuhan dipahami dengan baik, proses RAD memungkinkan tim pengembangan menciptakan “sistem fungsional yang utuh” dalam periode waktu yang sangat pendek (kira-kira 60-90 hari).

Pendekatan RAD model menekankan cakupan :

1. Pemodelan bisnis (*Business Modelling*)

Aliran informasi diantara fungsi-fungsi bisnis dimodelkan dengan suatu cara untuk menjawab pertanyaan-pertanyaan berikut : Informasi apa yang mengendalikan proses bisnis ? Kemana informasi itu pergi? Siapa yang memprosesnya ?

2. Pemodelan data (*Data Modelling*)

Aliran informasi yang didefinisikan sebagai bagian dari fase pemodelan bisnis disaring ke dalam serangkaian objek data yang dibutuhkan untuk menopang bisnis tersebut. Karakteristik/atribut dari masing-masing objek diidentifikasi dan hubungan antara objek-objek tersebut didefinisikan.

3. Pemodelan proses (*Process Modelling*)

Aliran informasi yang didefinisikan dalam fase pemodelan data ditransformasikan untuk mencapai aliran informasi yang perlu bagi implementasi sebuah fungsi bisnis. Gambaran pemrosesan diciptakan untuk menambah, memodifikasi, menghapus atau mendapatkan kembali sebuah objek data.

4. Pembuatan aplikasi (*Application generation*)

Selain menciptakan perangkat lunak dengan menggunakan bahasa pemrograman generasi ketiga yang konvensional, RAD lebih banyak memproses kerja untuk memakai lagi komponen program yang telah ada atau menciptakan komponen yang bias dipakai lagi. Pada semua kasus, alat-alat Bantu otomatis dipakai untuk memfasilitasi konstruksi perangkat lunak.

5. Pengujian dan pergantian (*Testing and turnover*)

Karena proses RAD menekankan pada pemakaian kembali, banyak komponen yang telah diuji. Hal ini mengurangi keseluruhan waktu pengujian. Tapi komponen baru harus diuji.

Kelemahan RAD model :

1. Untuk proyek dengan skala besar, RAD membutuhkan sumber daya manusia yang cukup untuk membentuk sejumlah tim RAD.
2. RAD membutuhkan pengembang dan pemakai yang mempunyai komitmen untuk melaksanakan aktivitas melengkapi sistem dalam kerangka waktu yang singkat.

3. Akan menimbulkan masalah jika sistem tidak dapat dibuat secara modular.
4. RAD tidak cocok digunakan untuk sistem yang mempunyai resiko teknik yang tinggi.

D. Spiral Model

Merupakan model proses perangkat lunak yang memadukan wujud pengulangan dari model prototyping dengan aspek pengendalian dan sistematika dari linear sequential model, dengan penambahan elemen baru yaitu analisis resiko. Model ini memiliki 4 aktivitas penting, yaitu :

1. Perencanaan (*Planning*), penentuan tujuan, alternatif dan batasan
2. Analisis resiko (*Risk Analysis*), analisis alternatif dan identifikasi/pemecahan resiko
3. Rekayasa (*Engineering*), pengembangan level berikutnya dari produk
4. Evaluasi Pemakai (*Customer Evaluation*) penilaian terhadap hasil rekayasa

Bentuk spiral memberikan gambaran bahwa semakin besar iterasinya, maka menunjukkan makin lengkap versi dari perangkat lunak yang dibuat. Selama awal sirkuit, objektif, alternatif dan batasan didefinisikan serta resiko diidentifikasi dan dianalisa. Jika resiko menunjukkan ada ketidakpastian terhadap kebutuhan, maka prototyping harus dibuat pada kuadran rekayasa. Simulasi dan pemodelan lain dapat digunakan untuk mendefinisikan masalah dan memperbaiki kebutuhan.

Pelanggan mengevaluasi hasil rekayasa (kuadran evaluasi pelanggan) dan membuat usulan untuk perbaikan. Berdasarkan masukan dari pelanggan, fase berikutnya adalah perencanaan dan analisis resiko. Setelah analisis resiko selalu diperiksa apakah proyek diteruskan atau tidak, jika resiko terlalu besar, maka proyek dapat dihentikan.

Model spiral ini adalah pendekatan yang paling realistic untuk sistem skala besar. Metode ini menggunakan pendekatan evolusioner, sehingga pelanggan dan pengembang dapat mengerti dan bereaksi terhadap suatu resiko yang mungkin terjadi

Kelemahan spiral model :

1. Sulit untuk meyakinkan pemakai (saat situasi kontrak) bahwa penggunaan pendekatan ini akan dapat dikendalikan.
2. Memerlukan tenaga ahli untuk memperkirakan resiko, dan harus mengandalkannya supaya sukses.
3. Belum terbukti apakah metode ini cukup efisien karena usianya relatif baru.

E. Fourth Generation Techniques (4GT)

Istilah generasi ke empat, mengarah ke perangkat lunak yang umum yaitu tiap pengembang perangkat lunak menentukan beberapa karakteristik perangkat lunak pada level tinggi.

Saat ini pengembangan perangkat lunak yang mendukung 4GT, berisi tool-tool berikut :

- Bahasa non prosedural untuk query basis data
- *Report generation*
- *Data manipulation*
- Interaksi layar
- Kemampuan grafik level tinggi
- Kemampuan spreadsheet

Tiap *tool* ini ada tapi hanya untuk suatu aplikasi khusus.

Menggunakan perangkat bantu (*tools*) yang akan membuat kode sumber secara otomatis berdasarkan spesifikasi dari pengembang perangkat lunak. Hanya digunakan untuk menggunakan perangkat lunak yang menggunakan bahasa khusus atau notasi grafik yang diselesaikan dengan syarat yang dimengerti pemakai. Cakupan aktivitas 4GT :

1. Pengumpulan kebutuhan, idealnya pelanggan akan menjelaskan kebutuhan yang akan ditranslasikan ke prototype operasional.
2. Translasi kebutuhan menjadi prototype operasional, atau langsung melakukan implementasi secara langsung dengan menggunakan bahasa generasi keempat (4GL) jika aplikasi relatif kecil.
3. Untuk aplikasi yang cukup besar, dibutuhkan strategi perancangan sistem walaupun 4GL akan digunakan.
4. Pengujian.
5. Membuat dokumentasi.
6. Melaksanakan seluruh aktivitas untuk mengintegrasikan solusi-solusi yang membutuhkan paradigma rekayasa perangkat lunak lainnya.

Salah satu keuntungan penggunaan model 4GT adalah pengurangan waktu dan peningkatan produktivitas secara besar, sementara kekurangannya terletak pada kesulitan penggunaan perangkat bantu (*tools*) dibandingkan dengan bahasa pemrograman, dan juga kode sumber yang dihasilkannya tidak efisien.

Untuk aplikasi yang kecil, adalah mungkin untuk langsung berpindah dari pengumpulan kebutuhan ke implementasi dengan menggunakan 4GL. Tapi untuk aplikasi yang besar, dibutuhkan pengembangan strategi desain untuk sistem, walau digunakan 4GL. Penggunaan 4GT tanpa perencanaan yang matang (untuk proyek skala besar) akan menyebabkan kesulitan yang sama (kualitas dan pemeliharaan yang jelek, ketidakpuasan pelanggan) seperti dengan metode konvensional.

BAB 3

MANAJEMEN PROYEK PERANGKAT LUNAK

3.1. Proses-proses Dalam Manajemen Proyek

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

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

Awal Proyek Perangkat Lunak

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

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

Pengukuran dan Satuan Ukuran

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

Estimasi

Dalam aktifitas utama proyek yaitu perencanaan, dilakukan estimasi :

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

Analisis Resiko

Analisis resiko sangat penting dalam manajemen proyek perangkat lunak. Beberapa hal yang harus diperhatikan berkaitan dengan resiko adalah ;

- Masa yang akan datang : resiko apa yang mempengaruhi trend (kecenderungan) proyek perangkat lunak
- Perubahan : Bagaimana perkembangan dunia mempengaruhi keawetan dan kesuksesan perangkat lunak
- Pilihan : metode apa yang dipakai, berapa orang diperlukan, seberapa tinggi kualitas perangkat lunak dan sebagainya

Analisis resiko merupakan serangkaian langkah untuk menyiasati resiko, yaitu :

- **Identifikasi resiko**

Identifikasi resiko melist semua resiko sesuai dengan kategori(secara makro) sebagai berikut :

1. Resiko proyek : masalah pembiayaan, penjadwalan, personil, sumber daya, pelanggan dan kebutuhan dikaitkan dengan akibatnya terhadap pelanggan.

2. Resiko teknis : masalah desain, implementasi, antarmuka, verifikasi dan pemeliharaan.
3. Resiko bisnis : termasuk di dalamnya adalah resiko pasar, resiko manajemen, dan resiko pembiayaan.

Salah satu metode terbaik untuk mengerti tiap resiko adalah dengan sejumlah pertanyaan seperti :

1. Adakah orang-orang yang paling top (*The best*) ?
2. Sesuaikah keahlian orang-orang tersebut?
3. Cukupkah orang-orang yang tersedia?
4. Apakah staf cukup dapat dipercaya untuk keseluruhan proyek?
5. Akan adakah staf yang bekerja paruh waktu?
6. Apakah staf telah memiliki persepsi yang benar tentang pekerjaannya?
7. Sudah cukupkah pelatihan untuk staf?
8. Cukup rendahkah tingkat pelimpahan kerja untuk menjamin kelanjutan proyek?

- **Perkiraan resiko**

Memperhitungkan lebih lanjut estimasi resiko dalam bentuk : $[r_i, l_i, x_i]$ dengan

r_i : resiko

l_i : kemungkinan terjadinya

x_i : akibat dari resiko dengan memprioritaskan resiko dan memulai memikirkan cara mengendalikan dan atau mengurangi resiko yang mungkin terjadi

- **Proyeksi resiko**

Disebut juga estimasi resiko, adalah usaha untuk mengukur setiap resiko dengan 2 cara :

1. Kemungkinan adanya resiko
2. Konsekuensi (masalah yang bisa timbul karena resiko)

Ada 4 aktivitas estimasi resiko :

1. Memastikan skala yang merefleksikan kemungkinan resiko
2. Memperkirakan konsekuensi resiko
3. Estimasi efek dari resiko pada proyek dan produk
4. Menentukan akurasi keseluruhan dari proyeksi resiko

- **Strategi manajemen resiko**

- **Putusan (Resolution) resiko**

- **Dan Pemantauan resiko**

Penjadwalan

Langkah-langkah yang dilakukan dalam penjadwalan :

- Identifikasi sekumpulan tugas
- Pastikan keterkaitan antar tugas
- Estimasi usaha untuk tiap-tiap tugas
- Tentukan pekerja dan sumber-sumber lainnya
- Buat jaringan tugas
- Buat jadwal kerja berdasarkan waktu

Penelusuran dan Pengendalian

Penelusuran dan pengendalian dilakukan setelah ada penjadwalan yang pasti, yaitu memeriksa apakah tugas telah dilaksanakan sesuai dengan jadwal.

3.2. Satuan Ukuran Produktivitas dan Kualitas Perangkat Lunak

Pengukuran perangkat lunak dilakukan untuk :

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

Satuan ukuran perangkat lunak dikategorikan ke dalam :

- Satuan ukuran produktivitas : *Output* dari proses rekayasa
- Satuan ukuran kualitas : indikasi tingkat pemenuhan kebutuhan konsumen
- Satuan ukuran teknik : Karakteristik perangkat lunak

Kategori lain untuk pengukuran :

- **Pengukuran berorientasi besarnya (Ukuran)** : Besarnya perangkat lunak = jumlah baris program

Pengukuran berorientasi ukuran merupakan pengukuran langsung. Pengukuran berorientasi ukuran menggunakan tabel berisi data berorientasi ukuran yang merupakan daftar proyek pengembangan perangkat lunak yang telah diselesaikan dikaitkan dengan data berorientasi ukuran untuk proyek yang bersangkutan

Contoh perhitungan :

- Produktivitas = KLOC (Kilo Line of Code)/Orang-Bulan
 - Kualitas = Cacat (Kesalahan)/ KLOC
 - Biaya = Satuan uang (\$ atau Rp)/KLOC
 - Dokumentasi = Jumlah halaman dokumentasi/KLOC
- **Pengukuran berorientasi fungsi** : fungsi = ruang lingkup informasi dan tingkat kesulitannya
- Merupakan pengukuran tidak langsung, yang menitikberatkan pada fungsionalitas atau utilitas program. Disebut juga metode *Function Point* sesuai dengan informasi-informasi yang didefinisikan sebagai :
- Jumlah masukan dari pemakai
 - Jumlah keluaran dari pemakai
 - Jumlah penyelidikan dari pemakai
 - Jumlah file
 - Jumlah antarmuka eksternal

3.3. Satuan Ukuran Kualitas Perangkat Lunak

Kualitas perangkat lunak dihitung pada saat proses rekayasa perangkat lunak ataupun setelah diserahkan kepada pemakai. Satuan ukuran kualitas perangkat lunak pada saat proses rekayasa :

- Kompleksitas program
- Modularitas yang efektif
- Besarnya program

Definisi pengukuran kualitas menurut Gilb:

- Kebenaran (*Correctness*) : Program harus bekerja dengan benar. Kebenaran merupakan tingkat perangkat lunak bekerja sesuai dengan fungsi yang dibutuhkan. Pengukuran yang umum adalah cacat (*defect*) /KLOC

- Perawatan (*Maintainability*) : Kemudahan perbaikan jika ada kesalahan, penyesuaian terhadap perubahan lingkungan atau peningkatan sesuai permintaan pemakai
- Integritas (*Integrity*) : Pengukuran tingkat ketahanan perangkat lunak terhadap serangan (disengaja/tidak) pada program, data dan dokumen
- Kegunaan (*Usability*) : Berkaitan dengan kemudahan pemakaian yang diukur berdasarkan keahlian yang diperlukan untuk mempelajari sistem, waktu yang dibutuhkan untuk dapat menggunakan sistem, peningkatan produktivitas dengan penggunaan sistem dan perkiraan yang sifatnya subjektif pada kelakuan pemakai

Menurut Basili dan Zelkowitz ada 5(lima) faktor yang mempengaruhi produktivitas perangkat lunak :

- Faktor manusia : jumlah dan tingkat keahlian tim
- Faktor masalah : Tingkat kerumitan masalah yang harus dipecahkan
- Faktor proses : Teknik analisis dan desain, bahasa dan tools
- Faktor produk : keandalan dan performansi sistem berbasis komputer
- Faktor sumber daya : ketersediaan tools, sumber-sumber perangkat keras dan perangkat lunak

BAB 4

ANALISIS KEBUTUHAN PERANGKAT LUNAK

Analisis kebutuhan perangkat lunak (*software requirements analysis*) merupakan aktivitas awal dari siklus hidup pengembangan perangkat lunak. Untuk proyek-proyek perangkat lunak yang besar, analisis kebutuhan dilaksanakan setelah aktivitas *sistem information engineering* dan *software project planning*.

Tahap analisis adalah tahapan pengumpulan kebutuhan-kebutuhan dari semua elemen sistem perangkat lunak yang akan di bangun. Pada tahap ini dibentuk spesifikasi kebutuhan perangkat lunak, fungsi perangkat lunak yang dibutuhkan, performansi (unjuk kerja) sistem perangkat lunak, penjadwalan proyek, identifikasi sumber daya (manusia , perangkat keras dan perangkat lunak yang dibutuhkan) dan taksiran biaya pengembangan perangkat lunak.

Kegunaan analisis adalah untuk memodelkan permasalahan dunia nyata agar dapat dimengerti. Permasalahan dunia nyata harus dimengerti dan dipelajari supaya spesifikasi kebutuhan perangkat lunak dapat diungkapkan. Tujuan aktivitas ini adalah untuk mengetahui ruang lingkup produk (*product space*) dan pemakai yang akan menggunakannya. Analisis yang baik akan mengungkapkan hal-hal yang penting dari permasalahan, dan mengabaikan yang tidak penting.

Setiap metode analisis mempunyai pandangan yang berbeda. Tetapi pada dasarnya semua metode analisis memiliki prinsip analisis yang sama, yaitu :

1. Menggambarkan domain informasi masalah
2. Mendefinisikan fungsi perangkat lunak
3. Menghasilkan model yang menggambarkan informasi, fungsi dan kelakuan yang dibagi secara rinci pada sebuah model lapisan (hirarki)
4. Informasi pokok pada tahap analisis memudahkan tahap implementasi yang lebih rinci.

Tujuan tahap analisis adalah :

1. Menjabarkan kebutuhan pemakai
2. Meletakkan dasar-dasar untuk tahap perancangan perangkat lunak
3. Mendefinisikan semua kebutuhan pemakai sesuai dengan lingkup kontrak yang disepakati kedua belah pihak (pengembang dan pengguna).

4.1. Apa yang Disebut Kebutuhan (Requirement)

Pengertian Kebutuhan

Menurut arti kamus, kebutuhan adalah sesuatu yang diminta, sesuatu yang dibutuhkan. Sedangkan menurut IEEE (*The Institute of Electrical and Electronics Engineers*) kebutuhan adalah :

- Kondisi atau kemampuan yang diperlukan pemakai untuk menyelesaikan suatu persoalan, atau untuk mencapai sebuah objek.
- Kondisi atau kemampuan yang harus dipenuhi oleh sistem, dalam arti memenuhi kontrak, standar, spesifikasi atau dokumen formal lain yang diinginkan.

Tahap kebutuhan akan perangkat lunak dimulai dengan :

1. Dikenalnya adanya sebuah permasalahan yang membutuhkan sebuah penyelesaian. Identifikasi sebuah permasalahan mungkin dapat dilakukan dengan berorientasi pada aplikasi, berorientasi pada bisnis, atau berorientasi pada kenaikan produktivitas (*product improvement oriented*).
2. Munculnya ide untuk membuat sebuah perangkat lunak baru (sebagai sebuah kemajuan).

Ada dua jenis kebutuhan :

1. Behavioral

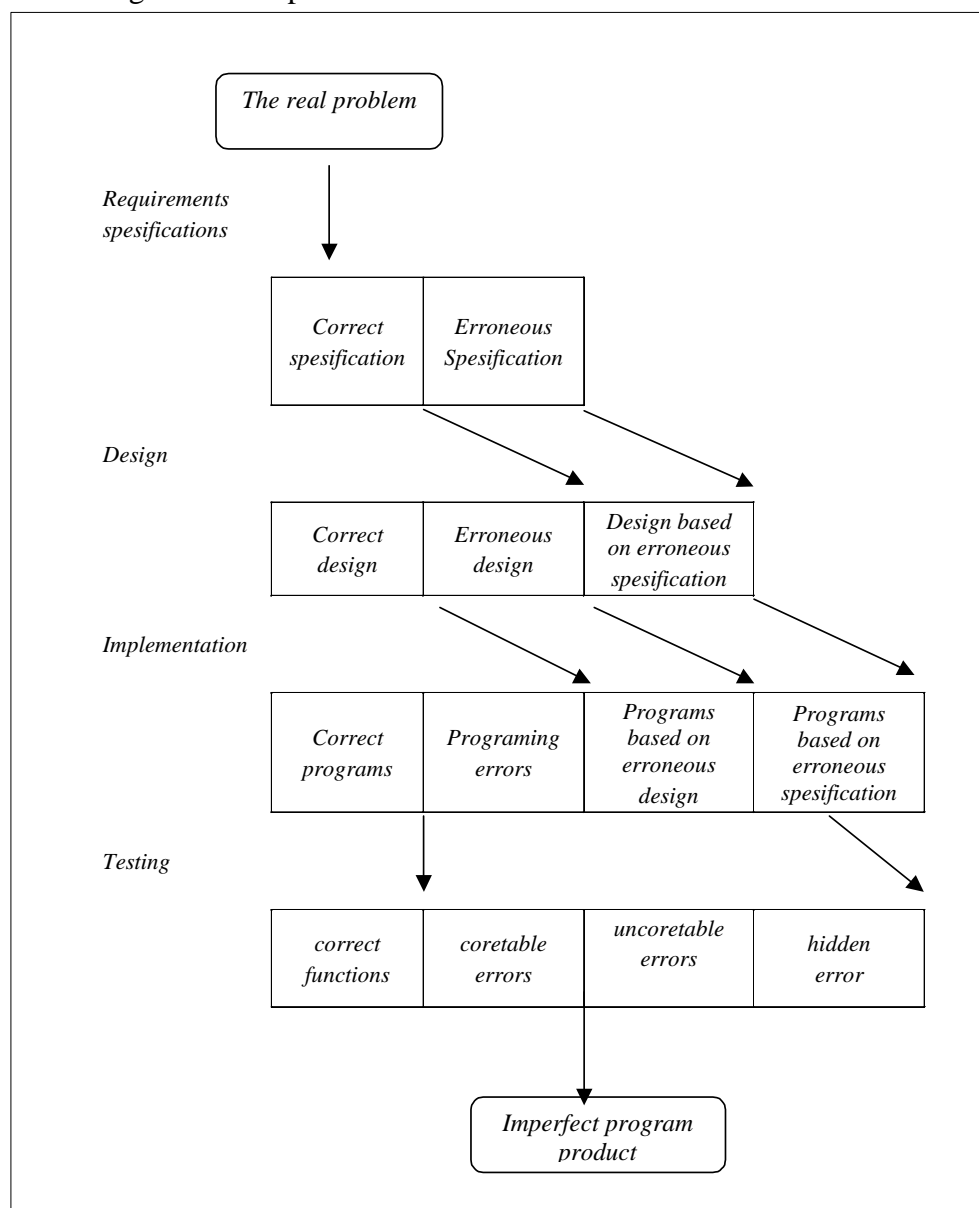
- apa yang dilakukan oleh sistem (input dan output dari dan ke sistem).
- hubungan informasi antara input dan output sehingga menghasilkan sebuah fungsi transformasi.

2. Non-behavioral

Mendefinisikan atribut sistem yang terkait untuk membentuk pekerjaan tersebut. Termasuk deskripsi lengkap tentang efisiensi, keamanan (*security*), *rehability* *maintenability* (bagaimana perawatan untuk sistem), dan *portability* (bisa dipindahkan dari satu perangkat keras ke perangkat keras lainnya).

Mengapa Kebutuhan Penting ?

Perhatikan gambar dampak kumulatif berikut ini :



Gambar 4.1. Dampak Kesalahan Kumulatif

Mencari kesalahan diakhir siklus hidup pengembangan perangkat lunak ternyata akan banyak mengeluarkan uang.

- Jika dapat dideteksi, dilakukan perbaikan pada setiap tahap proses.
- Jika tidak dapat dideteksi, kesalahan baru kelihatan setelah produk selesai dibuat.

4.2. Tahap Analisis Kebutuhan Perangkat Lunak

Tahap pekerjaan analisis kebutuhan perangkat lunak pada dasarnya terdiri dari urutan aktivitas :

1. Menentukan kebutuhan (*requirement*)
Lebih banyak berhubungan dengan pemakai. Hasil belum terstruktur.
 - Data atau informasi apa yang akan diproses
 - Fungsi apa yang diinginkan
 - Kelakuan sistem apa yang diharapkan
 - Antarmuka apa yang tersedia (*user interfaces, hardware interfaces, software interface, dan communications interfaces*)
2. Sintesis
Mengubah kebutuhan yang belum terstruktur menjadi model atau gambar dengan memanfaatkan teknik dan metode analisis tertentu.
3. Membuat dokumen *Software Requirements Specification* (SRS).
Sudah merupakan analisis yang lebih rinci, sebagai tahap awal perancangan.

4.3. Metode Analisis

Metode atau teknik untuk melakukan analisis kebutuhan perangkat lunak dikelompokkan berdasarkan pendekatan yang diambil pada saat melakukan aktivitas tersebut.

1. Berorientasi Aliran Data (*Data Flow Oriented* atau *Functional Oriented*)
Sudut pandang analisis pada pendekatan ini difokuskan pada aspek fungsional dan *behavioral* (perilaku laku) sistem. Pengembang harus mengetahui fungsi-fungsi atau proses-proses apa saja yang ada dalam sistem, data apa yang menjadi masukannya, dimana data tersebut disimpan, transformasi apa yang akan dilakukan terhadap data tersebut, dan apa yang menjadi hasil transformasinya. Selain itu pengembang harus mengetahui keadaan (*state*), perubahan (*transition*), kondisi (*condition*), dan aksi (*action*) dari sistem.
Salah satu metode yang paling populer untuk pendekatan ini adalah Analisis Terstruktur (*Structured Analysis*) yang dikembangkan oleh Tom DeMarco, Chris Gane dan Trish Sarson, dan Edward Yourdon . Pada metode ini, hasil analisis dan perancangan dimodelkan dengan menggunakan beberapa perangkat permodelan seperti :
 - *Data Flow Diagram* (DFD) dan Kamus Data (*data dictionary*) untuk menggambarkan fungsi-fungsi dari sistem.
 - *Entity-Relationship Diagram* (ERD) untuk menggambarkan data yang disimpan (*data storage*).
 - *State Transition Diagram* (STD) untuk menggambarkan perilaku sistem.
 - *Structure Chart* untuk menggambarkan struktur program
2. Berorientasi Struktur Data
Analisis pendekatan ini difokuskan pada struktur data, dimana struktur tersebut dapat dinyatakan secara hirarki dengan menggunakan konstruksi *sequence, selection* dan *repetition*. Beberapa metode berorientasi struktur data ini diantaranya adalah :

- Data Structured System Development (DSSD)
Diperkenalkan pertama kali oleh J.D. Warnier [1974] dan kemudian oleh Ken Orr [1977], sehingga sering disebut juga metode Warnier-Orr. Metode ini menggunakan perangkat *entity diagram*, *assembly line diagram* dan *Warnier-Orr diagram* untuk memodelkan hasil analisis dan rancangannya.
- Jackson Sistem Development (JSD)
Dikembangkan oleh M.A. Jackson [1975] dengan menggunakan perangkat permodelan yang disebut *struktural diagram* dan *sistem spesifikasi diagram*.

3. Berorientasi objek

Berbeda dengan pendekatan-pendekatan sebelumnya, pendekatan berorientasi objek memandang sistem yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata. Pada pendekatan ini, informasi dan proses yang dimiliki oleh suatu objek “dientkapsulasi” (dibungkus) dalam satu kesatuan. Beberapa metode pengembangan sistem yang berorientasi objek ini diantaranya adalah :

- Object Oriented Analysis (OOA) dan Object Oriented Design (OOD) dari Peter Coad dan Edward Yourdon [1990].
- Object Modelling Technique (OMT) dari James Rumbaugh [1987].
- Object Oriented Software Engineering (OOSE)

4.4. Analisis Berorientasi Aliran Data

Pendekatan dari sisi bisnis (DeMarco, Yourdon dan Senn). Analisis aliran data adalah analisis yang dilakukan untuk mempelajari pemanfaatan data pada setiap aktifitas. Menampilkan hasil pengamatan dalam apa yang disebut *Data Flow Diagram* (DFD) atau Diagram Alir Data (DAD).

4.4.1. Diagram Aliran Data (Data Flow Diagram)

Pengertian

- Suatu tampilan grafis yang memunculkan relasi/hubungan antara proses dan data beserta kamus data yang menjelaskan rincian data yang dipergunakan
- Diagram untuk menggambarkan aliran data dalam sistem, sumber dan tujuan data, proses yang mengolah data tersebut, dan tempat penyimpanan datanya.
- Representasi jaringan dari sistem yang menggambarkan sistem berdasarkan komponen-komponennya dengan semua antar muka diantara komponen-komponen tersebut.
- Perangkat permodelan yang dapat menggambarkan sistem sebagai sebuah jaringan proses-proses fungsional yang satu dengan yang lainnya dihubungkan oleh “pipa saluran” data.
- Diagram yang merepresentasikan bagaimana informasi keluar masuk dari ke sistem, proses apa yang mengubah informasi tersebut dan dimana informasi disimpan.
- Diperkenalkan oleh Tom DeMarco serta Chris Gane dan Trish Sarson berdasarkan notasi SADT (*Structure Analysis dan Design Technique*).
- Merupakan salah satu teknik yang cukup penting dalam menganalisa sistem karena :
 - Dapat mendefinisikan batasan sistem.
 - Membantu memeriksa kebenaran dan kelengkapan aliran informasi.

- Merupakan dasar perancangan dengan memunculkan proses-proses pengolahan data.
- Dapat digunakan untuk menggambarkan aktivitas proses secara paralel (beberapa aliran data dapat terjadi secara simultan). Bandingkan dengan flowmap yang hanya dapat menggambarkan aliran data (dokumen) secara serial.

Elemen-elemen DFD

Ada empat elemen yang membentuk suatu *Data Flow Diagram*, yaitu :

1. Aliran Data (*Data Flow*)

- Pipa saluran dimana paket informasi yang diketahui komposisinya mengalir.
- Penghubung antar proses yang merepresentasikan informasi yang dibutuhkan proses sebagai masukan atau informasi yang dihasilkan proses sebagai keluaran.
- Aliran paket informasi dari satu bagian sistem ke bagian sistem lainnya.
- Umumnya mengalir antar proses, tetapi dapat juga mengalir keluar masuk dari ke file (*data store*) atau dari ke sumber tujuan data.
- Data yang dinyatakan dengan aliran data boleh datang dari beberapa dokumen, jadi tidak perlu dirinci menjadi dokumen-dokumen tersebut.
- Diberi nama sesuai dengan substansi isi dari paket informasi (bukan nama dokumen) yang mengalir.
- Jumlah aliran data yang masuk dan keluar proses harus sama

2. Proses

- Transformasi aliran data yang datang menjadi aliran data yang keluar.
- Transformasi bagaimana satu atau beberapa masukan diubah menjadi keluaran.
- Menjelaskan proses-proses transformasi data apa saja yang ada dalam sistem atau yang harus dikerjakan oleh sistem. Komponen-komponen fisik tidak dapat diidentifikasi sebagai proses.
- Diberi nama dan nomor yang akan dipergunakan untuk keperluan identifikasi. Nama yang diberikan harus dapat menjelaskan apa yang dilakukan oleh proses. Nama proses biasanya ditulis dalam kata kerja.

3. Penyimpanan Data (*Data Store*)

- Tempat penyimpanan data atau tempat data yang dirujuk oleh proses.
- Kumpulan paket data yang harus diingat oleh sistem dalam periode waktu tertentu.
- Pada akhir pembangunan sistem, data store biasanya diimplementasi sebagai file atau basis data.

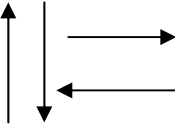
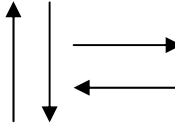
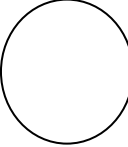
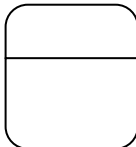


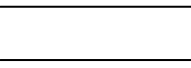
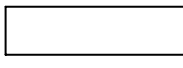
4. Entitas Eksternal/*Terminator/ Source atau Sink*

- Menggambarkan entitas yang berinteraksi dengan sistem yang berada diluar ruang lingkup sistem (bukan yang menjalankan sistem tersebut) atau entitas yang berfungsi sebagai *producer/consumer* dari sistem (sumber atau tujuan data).
- Dapat berupa orang, unit organisasi, komputer eksternal, organisasi eksternal atau sistem lain. Operator yang memasukkan data dalam sistem termasuk entitas internal, karena ia bukan consumer/producer sistem (kecuali untuk ruang lingkup perangkat lunak tertentu).
- Antara terminator tidak boleh berkomunikasi langsung

- Jumlah entitas/terminator yang terkait pada satu level akan muncul dalam jumlah yang sama untuk level lainnya

Berikut adalah tabel yang menunjukkan notasi yang digunakan dalam DFD.

Tabel 4.1. Simbol Data Flow Diagram

DEMARCO/YOURLDON	GALE/SARSON	KETERANGAN
		Aliran Data
		Proses
		Entitas Eksternal/ Terminator
		Data Store

Penggambaran DFD

Ada dua pendekatan penggambaran/pembuatan DFD yaitu pendekatan fisik dan logika.

Pendekatan Fisik

- Mengerjakan apa atau siapa yang mengerjakan proses-proses dalam sistem.
- Biasanya penggambaran DFD fisik dilakukan untuk alasan :
 - Kemudahan tahap awal dalam menguraikan interaksi antar komputer fisik suatu sistem.
 - Memberi kemudahan bagi pihak pemakai untuk memahami sistem dilihat dari sudut pandangnya.
 - Merupakan salah satu cara yang mudah untuk mendapatkan pengesahan dan verifikasi dari pemakai.
- Cukup efektif dalam mengkomunikasikan sistem pada pihak pemakai.

Pendekatan Logika

- Menggambarkan proses atau fungsi transformasi data yang ada dalam sistem (bukan apa atau siapa yang mengerjakannya).

- Dapat dibuat dari DFD fisik dengan cara mentranslasikannya menjadi deskripsi logika yang difokuskan pada data dan proses (jangan melihat siapa yang melakukan pekerjaan tersebut).
- Aturan dasar untuk menggambarkan diagram logic aliran data :
 - Setiap aliran data yang meninggalkan proses harus berdasarkan pada data yang masuk ke dalam proses tersebut.
 - Semua aliran data diberi nama dimana pemberian nama merefleksikan data yang mengalir tersebut antara proses, penyimpanan data dan sumber lainnya.
 - Hanya data yang akan dipergunakan dalam proses yang digambarkan sebagai masukan pada satu proses.
 - Satu proses tidak perlu mengetahui proses lainnya dalam sistem, jadi hanya tergantung pada masukan dan keluarannya saja.
 - Proses selalu berjalan dalam arti tidak ada awal atau akhir. Jadi selalu siap menjalankan fungsinya atau melakukan pekerjaan tertentu.
- Beberapa hal yang harus diperhatikan dalam menggambarkan DFD logika :
 - Perhatikan data aktual, bukan dokumen, yang berhubungan dengan proses.
 - Hilangkan aliran informasi melalui orang/unit organisasi/kantor, munculkan prosedur atau prosedurnya saja.
 - Hilangkan proses yang tidak penting, yang tidak mengubah data/aliran data, misalnya proses menyalin (copy) data.
 - Hilangkan fungsi alat bantu atau peralatan-peralatan lainnya.
 - Konsolidasikan kerangkapan penyimpanan data.
- Dibuat hanya untuk menggambarkan proses yang akan dikerjakan oleh komputer, bukan proses yang sifatnya fisik atau manual.

Diagram Konteks

Menggambarkan secara umum konteks yang terjadi dalam sistem antara dunia internal dan dunia eksternal yang berbatasan. Merupakan lapisan teratas terhadap sistem yang akan di bahas.

DFD Level 1 ... s.d. Level n

Merupakan gambaran rinci dari diagram konteks, makin tinggi levelnya maka akan makin dalam penjabaran rincian prosesnya.

Evaluasi ketelitian DFD

Sangat penting untuk mengevaluasi DFD yang sudah dibuat. Beberapa pertanyaan dapat muncul untuk evaluasi tersebut :

- Apakah ada komponen dalam DFD yang belum diberi nama?
- Apakah ada data yang disimpan yang tidak direfer sebagai masukan/keluaran dari suatu proses?
- Apakah ada proses yang tidak menerima masukan sama sekali?
- Apakah ada proses yang tidak memproduksi keluaran sama sekali?
- Apakah masih ada proses yang melayani beberapa tujuan proses?
- Apakah ada data yang disimpan dan tidak pernah direfer?
- Apakah masukan data sesuai/releven untuk dijalankan pada proses?
- Apakah ada item data yang simpan berlebihan (lebih dari yang dibutuhkan)?

4.4.2. Kamus Data (*Data Dictionary*)

Pengertian

- Merupakan alat Bantu untuk menjelaskan karakteristik logik data yang disimpan dalam sistem yang “*current*” termasuk nama, deskripsi, alias,, isis dan organisasinya.
- Merupakan suatu tempat penyimpanan (gudang) dari data dan informasi yang dibutuhkan oleh suatu sistem informasi.
- Digunakan untuk mendeskripsikan rincian dari aliran data atau informasi yang mengalir dalam sistem, elemen-elemen data, file maupun basis data.
- Ada auran (konvensi) penulisannya dengan menggunakan notasi atau simbol tertentu.

= sama dengan atau terdiri dari atau terbentuk dari

+ dan

[] pilih salah satu

{ } iterasi atau pengulangan

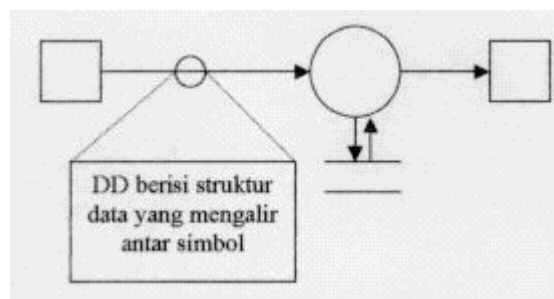
() pilihan (option)

* komentar

(pemisah

- Saat ini ada banyak variasi penulisan kamus data, yang secara umum dibedakan menjadi bentuk lengkap (long form) dan bentuk ringkas (*short form*).

Gambar berikut menunjukkan hubungan antara DFD dengan DD.



Gambar 4.2. Hubungan antara Data Flow Diagram (DFD) dengan Data Dictionary (DD).

Contoh

Id. Barang = Kode_Brg + Nama_Brg + Satuan + Hrg_Beli + Hrg_Jual + Banyak
Kode_Brg = 1 {character} 6
Nama_Brg = 1 {character} 20
Satuan = 1 {character} 3
Hrg_Beli = 3 {numeric} 10
Hrg_Jual = 3 {numeric} 10
Banyak = 1 {numeric} 6
character = [A-Z(a-z(0-9(- ([)
numeric = [0-9]

Spesifikasi Proses (*Process Specification*)

Pengertian

- Digunakan untuk menggambarkan deskripsi dan spesifikasi dari setiap proses yang paling rendah (proses atomik) yang ada pada sistem.
- Menggunakan notasi yang disebut Structured English atau pseudo-code.
- Penulisannya cukup sederhana sehingga dapat digunakan sebagai media untuk mengkomunikasikan proses yang dilakukan sistem kepada pemakai.
- Tersusun dari tiga struktur dasar, yaitu struktur sekuensi, pemilihan dan pengulangan.

Contoh

Nomor : 3.0

Nama Proses : Buat laporan penjualan

Jenis : Pembuatan laporan

Masukan : File Barang, file Jual dan periode transaksi

Keluaran : Laporan penjualan

Deskripsi

Begin

Buka file BARANG dan file JUAL

Baca data periode tanggal transaksi

Saring (filter) data pada file JUAL sesuai periode tanggal transaksi

Cetak Laporan Penjualan

Tutup file BARANG dan file JUAL

End

atau secara lebih ringkas :

Proses 3.0 Buat Laporan Penjualan

Begin

Buka file BARANG dan file JUAL

Baca data periode tanggal transaksi

Saring (filter) data pada file JUAL sesuai periode tanggal transaksi

Cetak Laporan Penjualan

Tutup file BARANG dan file JUAL

End

4.5. Analisis Berorientasi Struktur Data

Metode analisis yang berorientasi ke struktur data mempunyai focus utama pada struktur data dan bukan pada aliran datanya. Metode yang dapat digunakan untuk melakukan analisis cukup beragam, tetapi masing-masing memiliki karakteristik umum yaitu :

- Masing-masing mempunyai tujuan untuk membantu analis dalam mengidentifikasi objek informasi (item/entitas) yang penting dan operasinya.
- Struktur informasinya berbentuk hirarki

- Masing-masing membutuhkan penggambaran struktur data dalam bentuk urutan (*sequence*), pemilihan (*selection*) dan pengulangan (*repetition*).
- Menyediakan sekumpulan langkah untuk pemetaan struktur data hirarki ke struktur program

Pada setiap permasalahan, kemungkinan metode analisis ini dapat mencakup arsitektur dan perancangan perangkat lunak.

Metode analisis yang berorientasi struktur data adalah

A. Data Structured Systems Development (DSSD).

DSSD disebut juga metodologi Warnier-Orr, yang diperkenalkan oleh J.D. Warnier dan dikembangkan oleh Ken Orr. Notasi yang digunakan dinamakan notasi pada diagram Warnier/Orr.

Pada pendekatan ini tahapan yang dilakukan adalah :

1. Menentukan konteks aplikasi sistem

Yaitu menentukan bagaimana data berpindah dari produser (penghasil) informasi kepada konsumen (pemakai) informasi. Pembuatan konteks aplikasi dari suatu permasalahan dapat dimulai dengan menentukan :

- Bagian-bagian informasi yang akan diproses
- Produser dan konsumen informasi
- Pandangan produser dan konsumen tentang konteks aplikasi dari sudut pandang masing-masing

DSSD menggunakan diagram entitas untuk menentukan ketiga bagian diatas. Notasi diagram entitas menggunakan bentuk bulatan sebagai produser atau konsumen informasi (dapat berupa orang, mesin atau sistem). Kemudian semua entitas yang ada digabungkan untuk memperoleh semua produser dan konsumen yang terlibat dan membentuk satu lingkungan terbatas dari sistem yang didefinisikan.

2. Menentukan fungsi aplikasi

Menentukan fungsi aplikasi berarti mengamati aliran informasi yang ada. Penggambarannya menggunakan notasi seperti diagram Warnier yang disebut *Assembly Line Diagram (ALD)* atau diagram penyusunan baris. Dengan ALD, DSSD menyediakan mekanisme untuk merangkai informasi dan proses. Pembuatan diagram penyusunan baris dimulai dengan aliran informasi yang terakhir dan dikerjakan mundur sampai aliran informasi yang pertama.

3. Hasil aplikasi

Sistem pengembangan berorientasi struktur data memerlukan analisis untuk membuat prototype laporan (*paper prototype*) tentang keluaran yang diinginkan oleh system. Identifikasi prototype yang utama adalah keluaran dari system dan operasi dari informasi tiap bagian (item) yang menyusun keluaran tersebut. Setelah prototype selesai, hirarki informasi dapat dimodelkan dengan diagram Warnier Orr.

B. Jackson System Development (JSD)

Dikembangkan oleh Michael Jackson pada tahun 1975-1983, menyerupai pendekatan DSSD. Ciri khas pendekatan ini adalah focus diarahkan pada dunia nyata domain informasi. Pendekatan yang dilakukan membangun suatu model dari dunia nyata (*real world*) untuk mendekati system dengan segala subyek permasalahannya. Pendekatan ini menghasilkan pendefinisian semua fungsi terlibat yang kemudian ditambahkan ke dalam model dunia nyata. Metode ini lebih tepat diterapkan untuk system yang sifatnya dinamik.

Secara umum JSD memiliki karakteristik sebagai berikut :

- Pengidentifikasian objek informasi (entitas, item) dan operasi (aksi, proses)
- Mengasumsikan struktur informasi bersifat hirarki
- Merepresentasikan struktur data menggunakan konstruksi *sekuens, selection, dan iteration*.
- Langkah-langkah yang dilakukan adalah sebagai berikut :
 1. *Entity Action Step* (Langkah aksi entitas)
Dimana dalam tahap ini dilakukan pendefinisian yang memunculkan entitas (orang, objek ataupun unit organisasi) yang ambil bagian dalam system dan aksi-aksi yang ada diantaranya.
 2. *Entity Structure step* (Langkah tatanan entitas)
Mengurutkan aksi-aksi berdasarkan urutan waktu
 3. *Initial Model* (Langkah model awal)
Pemodelan awal dimana entitas-entitas dan aksi-aksi direpresentasikan sebagai model proses.
 4. *Function Step* (Langkah fungsi)
Merancang fungsi dari system dalam bentuk suatu model
 5. *System Timing Step* (Langkah pewaktuan system)
Mempertimbangkan penundaan waktu, hambatan-hambatan waktu
 6. *Implementation Step* (Langkah Pengejawantahan)
Mempertimbangkan Perangkat keras dan perangkat lunak yang tersedia untuk menjalankan system. Jadi fokusnya penjadwalan proses

4.6. Analisis Antarmuka Pemakai

Beberapa objektif dalam analisis antarmuka antara perangkat lunak dan pemakai adalah :

- Memperhatikan kebutuhan dan kemampuan pemakai, dalam arti membuat antarmuka yang familiar/bersahabat.
- Konsistensi antar modus antarmuka yang dipergunakan pada system yang ditinjau, terutama konsistensi anatar subsistem dengan subsistem lainnya
- Mempunyai fasilitas Help (Error message dan recoverynya)
- Minimal surprise, dalam arti pemakai tidak perlu dikejutkan oleh behaviour system.

Jenis interaksi antarmuka :

1. Antarmuka Perintah

- Masukan dengan memberikan perintah yang diketik dan akan membangkitkan aksi-aksi (query, inisiasi, panggil perintah lain, proses dan lain-lain)
- Menggunakan masukan sekilas dengan mesin ketik (*screen 24 X 80 Lines*)

Keuntungan :

- Antarmuka sederhana
- Tampilan alpha numeric – murah
- Banyak teknik-teknik pemrosesan bahasa (Kompiler)
- Penggabungan perintah-perintah sederhana dan mudah dikembangkan
- Usaha *typing command* efeknya pasti

Kerugian :

- Pemakai harus belajar *command language*
- Kemungkinan *incorrect input* lebih besar (salah ketik)
- Selalu perlu *keyboard*

2. Antarmuka Metapor

Grafik (gambar) yang merepresentasikan entitas system sedemikian hingga dapat disamakan dengan pemakai system secara familiar. Contohnya Control panel dalam perancangan punya entitas button.

3. Antarmuka Menu

- Pemakai memilih salah satu dari sejumlah menu yang tersedia untuk menjalankan perintah pada komputer. Pemilihan dilakukan dengan menggunakan mouse atau peralatan penunjuk lainnya.

Keuntungan :

- Pemakai tidak perlu tahu nama perintah
- Usaha pengetikan menjadi minimal
- Beberapa dari kondisi kesalahan pemakai dapat dihindari (kesalahan sintaks perintah jarang terjadi)

4. Antarmuka WIMP (Window, Icon, Menu, dan Pointing)

- Mempunyai kemudahan fasilitas pemakaian yang bervariasi, sehingga memberikan gambaran interaksi manusia komputer yang benar-benar bersahabat

BAB 5

SPESIFIKASI KEBUTUHAN PERANGKAT LUNAK

Spesifikasi kebutuhan perangkat lunak atau *Software Requirements Specification* (SRS) adalah sebuah dokumen yang berisi pernyataan lengkap dari apa yang dapat dilakukan oleh perangkat lunak, tanpa menjelaskan bagaimana hal tersebut dikerjakan oleh perangkat lunak. Suatu SRS harus mencantumkan tentang deskripsi dengan lingkungannya. Mencakup antarmuka untuk perangkat keras, perangkat lunak, komunikasi dan pemakai.

SRS bisa terdiri dari banyak dokumentasi yang saling melengkapi. Suatu SRS harus dapat :

1. Menguraikan definisi masalah
2. Menguraikan masalah dengan tepat dengan cara yang tepat pula

Objektif SRS

1. Persetujuan kerja dengan pelanggan
2. Daftar kebutuhan teknis yang harus dipenuhi oleh perangkat lunak

Syarat Pembentukan SRS

1. Mudah diidentifikasi
2. Diuraikan dengan jelas, *simple*, sederhana dan *concise* (Jelas, tidak *ambiguous*)
3. Bisa divalidasi dan bisa dites (*test reliable, test accessible*).
4. Mampu untuk ditelusuri kembali (*traceability*)

Hindari hal-hal berikut saat pembentukan SRS

1. *Over specification* (penjelasan berlebih dan berulang-ulang sehingga menjadi tidak jelas)
2. Tindakan *unconcistency*
3. *Ambiguity* dalam kata atau kalimat
4. Menuliskan “mimpi-mimpi”, yaitu hal-hal yang tidak bisa dilakukan

Dalam Suatu SRS ada 2 aspek yang harus bisa dilihat :

1. Fungsi
Menjelaskan fungsi dari perangkat lunak (digunakan untuk apa keperluan apa), sifat lunak dan datanya.
2. Non-Fungsi
 - a. *Dependability*
 - *reliability*
 - *maintainability*
 - *security*
 - *integrity*
 - b. *Ergonomic*
 - c. *Performance*
 - d. *Constraint*

Atribut Suatu SRS

1. Benar (*correct*)
Jika salah (*incorrect*), artinya spesifikasi yang ditulis adalah bukan yang diinginkan.
2. Tepat (*precise*)
Berpengaruh pada hasil perancangan dan pembuatan *software requirements design* (SRD).
3. *Unambiguouity*
Setiap permintaan harus punya satu interpretasi, atau hanya ada satu arti dalam satu kalimat.
4. Lengkap (*complete*)
Lengkap jika dilihat dari dua sudut pandang :
 - Dokumen membuat tabel isi, nomor halaman, nomor gambar, nomor tabel, dan sebagainya.
 - Tidak ada bagian yang hilang (*to be define*) yaitu tulisan yang akan didefinisikan kemudian
5. Bisa diverifikasi (*verifiable*)
Bisa diperiksa dan dicek kebenarannya. Setiap kebutuhan selalu dimulai dengan dokumen yang bisa diperiksa.
6. Konsisten
Nilai-nilai kebutuhan harus tetap sama baik dalam karakteristik maupun spesifik misalnya diminta A tetap ditulis A.
7. *Understandable*
Dapat dimengerti oleh pemrograman, analisis sistem atau *sistem engineer*
8. Bisa dimodifikasi (*modifiedable*)
Bisa diubah-ubah dan pengubahannya sangat sederhana tetapi tetap konsisten dan lengkap.
9. Dapat ditelusuri (*traceable*)
Jika ditelusuri, harus tahu mana bagian yang diubah
10. Harus dapat dibedakan bagian *what* (bagian spesifikasi) dan *how* (bagian yang menjelaskan bagaimana menjelaskan *what* tadi)
11. Dapat mencakup dan melingkupi seluruh sistem
12. Dapat melingkupi semua lingkungan operasional, misalnya interaksi fisik dan operasional.
13. Bisa menggambarkan sistem seperti yang dilihat oleh pemakai.
14. Harus toleran (bisa menerima) terhadap ketidaklengkapan, ketidakpastian (*ambiguous*) dan tidak konsisten.
15. Harus bisa dilokalisasi dengan sebuah *coupling*, yaitu hubungan ketergantungan antara dua model yang tidak terlalu erat.

Ada 9 macam orang yang terlibat dalam pembuatan SRS :

1. Pemakai (*user*)
Yang mengoperasikan / menggunakan produk final dari perangkat lunak yang dibuat.
2. Client
Orang atau perusahaan yang mau membuat sistem (yang menentukan).
3. Sistem analyst (*sistem engineer*)
Yang biasa melakukan kontak teknik pertama dengan client. Bertugas menganalisis persoalan, menerima *requirement* dan menulis *requirement*.
4. Software engineer

Yang bekerja setelah kebutuhan perangkat lunak dibuat (bekerja sama dengan sistem engineer berdasarkan SRS)

5. Programmer
Menerima spesifikasi perancangan perangkat lunak, membuat kode dalam bentuk modul, menguji dan memeriksa (tes) modul.
6. Test integration group
Kumpulan orang yang melakukan tes dan mengintegrasikan modul.
7. Maintenance group
Memantau dan merawat performansi sistem perangkat lunak yang dibuat selama pelaksanaan dan pada saat modifikasi muncul (80% dari pekerjaan).
8. Technical Support
Orang-orang yang mengelola (*manage*) pengembang perangkat lunak, termasuk konsultan atau orang yang mempunyai kepandaian lebih tinggi.
9. Staff dan Clerical Work
Bertugas mengetik, memasukkan data dan membuat dokumen.

Keberhasilan pengembangan perangkat lunak bisa dilihat dari 10 aspek atau titik pandang, yaitu :

1. Ketelitian dari pembuatnya
2. Kualitas dari spesifikasi perangkat lunak yang dihasilkan (Baik, jika ada sedikit kesalahan).
3. Integritas
4. Ketelitian
5. Proses Pembuatan yang mantap
6. Mudah dikembangkan
7. Jumlah versi yang tidak banyak
8. Ketelitian dari model pengembangan yang digunakan untuk meramal atribut perangkat lunak
9. Efektivitas rencana tes dan integrasi
10. Tingkat persiapan untuk sistem perawatan (mempersiapkan pencarian *bugs*)

Contoh Layout Dokumen SRS

1. PENDAHULUAN
 - 1.1. Tujuan
 - 1.2. Ruang Lingkup
 - 1.3. Definisi
 - 1.4. Referensi
 - 1.5. Sistematika
2. DESKRIPSI UMUM
 - 2.1. Perspektif
 - 2.2. Kegunaan
 - 2.3. Karakteristik Pengguna
 - 2.4. Batasan-batasan
 - 2.5. Asumsi dan Ketergantungan

- 3. SPESISIKASI KEBUTUHAN
 - 3.1. Kebutuhan Fungsional
 - 3.1.1. Pendahuluan
 - 3.1.2. Input
 - 3.1.3. Proses
 - 3.1.4. Output
 - 3.2. Kebutuhan Antarmuka Eksternal
 - 3.2.1. Antarmuka Pengguna
 - 3.2.2. Antarmuka Perangkat Keras
 - 3.2.3. Antarmuka Perangkat Lunak
 - 3.2.4. Antarmuka Komunikasi
 - 3.3. Kebutuhan Performasi
 - 3.4. Kendala Desain
 - 3.4.1. Standard Compliance
 - 3.4.2. Perangkat Keras
 - 3.5. Atribut
 - 3.5.1. Keamanan Sistem
 - 3.5.2. Pemeliharaan
 - 3.6. Kebutuhan Lain
 - 3.6.1. Database
 - 3.6.2. Pengoperasian
 - 3.6.3. Penyesuaian Tempat

BAB 6

PERANCANGAN PERANGKAT LUNAK

Perancangan adalah langkah awal pada tahap pengembangan suatu produk atau sistem. Perancangan dapat didefinisikan sebagai proses untuk mengaplikasikan berbagai macam teknik dan prinsip untuk tujuan pendefinisian secara rinci suatu perangkat, proses atau sistem agar dapat direalisasikan dalam suatu bentuk fisik. Tujuan perancangan adalah menghasilkan suatu model atau penggambaran dari suatu entity yang akan dibangun kemudian.

6.1. Perancangan Perangkat Lunak

Pengertian :

Perancangan perangkat lunak adalah suatu proses bertahap dimana semua kebutuhan atau persyaratan yang ada pada dokumen SRS diterjemahkan menjadi suatu cetak biru (*blue print*) yang akan digunakan untuk membangun perangkat lunak. Pada tahap awal, cetak biru melukiskan suatu gambaran umum dari perangkat lunak (merupakan penjelasan tingkat tinggi). Pada tahap selanjutnya, penjelasan rinci dilakukan hingga pada tingkat penjelasan paling rendah.

Perancangan perangkat lunak dilakukan dengan anggapan spesifikasi kebutuhan perangkat lunak sudah terdefiniskan dalam model-model analisis. Model-model yang dihasilkan selama perancangan menggambarkan “bagaimana” permasalahan diselesaikan dalam bentuk spesifikasi perangkat lunak yang siap diimplementasikan.

Perancangan dapat juga dipandang sebagai proses penerapan berbagai teknik dan prinsip dengan tujuan untuk mendefinisikan spesifikasi rinci perangkat lunak sehingga mudah diimplementasikan. Dengan suatu metode merancang spesifikasi kebutuhan perangkat lunak yang diwujudkan dalam domain informasi, keperluan fungsional dan performansi dirancang menjadi spesifikasi perangkat lunak yang diwujudkan dalam rancangan arsitektur perangkat lunak, struktur data dan prosedur dari perangkat lunak.

Solusi perangkat lunak dapat berbentuk prosedur-prosedur ataupun objek (paket data dan operasi-operasi terhadap data tersebut). Struktur program menggambarkan organisasi dari komponen-komponen program (modul/objek). Struktur program tidak menggambarkan aspek prosedural perangkat lunak, seperti urutan proses, percabangan atau perulangan.

Struktur data menggambarkan hubungan (organisasi) logika antara elemen-elemen data. Struktur data berpengaruh pada prosedur-prosedur proses yang dipilih dan metode-metode aksesnya.

Prosedur perangkat lunak memfokuskan pada proses secara rinci dari masing-masing modul atau objek. Prosedur ini menerangkan dengan tepat algoritma proses-proses dan struktur data yang digunakannya. Prinsip-prinsip dalam penyusunan modul atau objek adalah sebagai berikut :

1. Modularitas

Prosedur perangkat lunak dibagi atas beberapa modul. Sebuah modul dapat dibagi atas beberapa sub modul. Modul memiliki nama unik. Sebuah modul dapat memanggil (mengirim pesan) modul lainnya.

2. Penyembunyian informasi

Merupakan prinsip dasar dalam pembentukan modul, yaitu struktur data dan logik program pada suatu modul bersifat terselubung. Modul dipandang sebagai kotak hitam, artinya dengan masukan tertentu akan menghasilkan keluaran yang diharapkan tanpa perlu mengetahui proses yang terjadi di dalamnya.

3. Abstraksi

Perancangan secara modular, memungkinkan beberapa tingkatan abstraksi dapat diperoleh, sehingga perancang dapat berkonsentrasi pada setiap tingkatan abstraksi, tanpa memperdulikan tingkatan abstraksi yang lebih rinci.

4. Kopling
Adalah derajat ketergantungan antara dua modul. Modul yang baik harus memiliki derajat ketergantungan/kopling yang lemah.
5. Kohesi
Adalah ukuran kekuatan hubungan antar elemen-elemen yang membentuk modul. Modul yang baik mempunyai kohesi yang kuat.
6. Integritas
Setiap modul harus bisa menjaga integritasnya masing-masing.
7. Ekstensibilitas
Mampu beradaptasi terhadap perubahan spesifikasi.

Tujuan dari tahap perancangan adalah :

1. Merealisasikan hasil tahap analisis ke dalam bentuk rancangan sistem yang lebih rinci
2. Mendefinisikan bentuk antar muka pemakai pada bagian masukan dan keluaran
3. Mendefinisikan proses pengolahan data atau informasi secara detail
4. Membentuk struktur data atau basis data secara logik (*logical database*)

Perancangan yang baik :

- Melaksanakan seluruh kebutuhan/persyaratan yang tercantum pada dokumen SRS.
- Merupakan acuan yang dapat dibaca, dimengerti oleh pembuat program dan penguji perangkat lunak.
- Menyediakan gambaran yang lengkap tentang perangkat lunak mencakup data, fungsi dan tanggapan, dalam perspektif pelaksanaan pembuatan perangkat lunak.
- Menghasilkan model atau representasi dari perangkat lunak, untuk digunakan dalam proses implementasi atau *coding*.

Proses Perancangan :

Merupakan proses kreatif dalam pembangunan perangkat lunak untuk memecahkan suatu persoalan. Model dari proses perancangan secara garis besar terdiri dari empat tahap proses, yaitu :

1. Mengemukakan suatu solusi
2. Membangun model dari solusi tersebut
3. Evaluasi model terhadap spesifikasi kebutuhan yang telah ada
4. Menjabarkan rincian spesifikasi dari solusi tersebut

Proses perancangan mempunyai masukan, fungsi dan keluaran

- Masukan proses perancangan
Model informasi, model fungsional dan model *behavioral*, dan spesifikasi kebutuhan lain yang diperoleh setelah proses analisis kebutuhan.
- Fungsi proses perancangan
Ada dua fungsi yang dimiliki oleh proses perancangan, yaitu translasi/pengembangan dari spesifikasi perangkat lunak, dan penjabaran bagaimana perangkat lunak menjadi berfungsi dan bagaimana spesifikasi perangkat lunak dapat diimplementasikan.
- Keluaran proses perancangan
Perancangan data, perancangan arsitektural, perancangan prosedural dan perancangan antarmuka pemakai (*User Interface*).

Tahapan Perancangan :

Dari sudut pandang manajemen proyek, perancangan terdiri dari dua bagian, yaitu :

1. Perancangan awal (*preliminary design*)
Menentukan arsitektur perangkat lunak secara keseluruhan :
 - Bagaimanakah lingkungan programnya ?
 - Bagaimanakah bentuk penyimpanan datanya ?
 - Bagaimana bentuk interface-nya ?
2. Perancangan rinci (*detailed design*)
Menentukan modul program (prosedural) yang harus dibuat.

Adapun dari sudut pandang teknis, kegiatan perancangan terdiri atas aktivitas sebagai berikut :

1. Perancangan data
2. Perancangan arsitektural
3. Perancangan prosedural
4. Perancangan antarmuka pemakai

Tahap perancangan mempunyai peran yang cukup penting, karena akan digunakan sebagai basis dari implementasi dan pengembangan perangkat lunak tahap selanjutnya. Sebagai basis implementasi, diperlukan penjabaran aspek perangkat lunak dari berbagai sudut pandang. Semakin kompleks sistem perangkat lunak, semakin banyak sudut pandang perancangan yang dihasilkan sehingga seluruh aspek perangkat lunak tercakup penjabarannya. Secara umum, ada empat sudut pandang pemodelan perancangan perangkat lunak, yaitu :

1. Perilaku (*behaviour*)
2. Fungsional
3. Pemodelan data
4. Struktural

Hasil perancangan didokumentasi dalam SDD (*Software Design Descriptions*) yang berisi model atau representasi perangkat lunak untuk digunakan sebagai dasar proses implementasi (coding).

6.2. Teknik Perancangan Perangkat Lunak

Ada beberapa teknik dan pendekatan yang dapat digunakan pada saat merancang perangkat lunak. Salah satu teknik tersebut adalah teknik Perancangan Terstruktur (*Structured Design*) yang dilaksanakan berdasarkan pendekatan aliran data.

Perancangan Terstruktur

Ada beberapa pengertian tentang Perancangan Terstruktur (*Structured Design*), dan diantaranya adalah :

- Pendekatan disiplin perancangan perangkat lunak yang menganut pada sekumpulan aturan tertentu berdasarkan prinsip-prinsip seperti *top-down design*, *stepwise refinement*, dan analisis aliran data (IEEE, 1983).
- Gabungan teknik, strategi dan metode untuk merancang sistem perangkat lunak dan program melalui tahap demi tahap prosedur perancangan, baik perancangan sistem maupun perancangan rinci, yang didukung oleh sekumpulan strategi perancangan, petunjuk dan teknik-teknik dokumentasi (MAR, 1985).

- Sekumpulan petunjuk dan teknik-teknik untuk membantu perancang membedakan mana perancangan yang baik dan jelek pada tingkat modular (Yourdan, 1979).

Produk dari Perancangan Terstruktur adalah *structure chart* yang memperlihatkan komponen-komponen prosedural program, pengaturan hirarkinya dan data yang menghubungkan komponen-komponen tersebut.

Tahap Perancangan Terstruktur

Ada empat tahap proses yang harus dilakukan pada saat melakukan perancangan dengan pendekatan Perancangan Terstruktur yaitu :


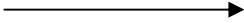
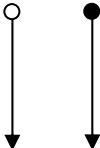
- Nyatakan hasil perancangan sebagai aliran data yang melalui sekumpulan proses-proses dan gambarkan Data Flow Diagramnya.
- Nyatakan hasil perancangan sebagai hirarki dari fungsi (atau komponen-komponen prosedural) dengan *structure chart* berdasarkan DFD yang didapat.
- Evaluasi dan perbaiki kembali hasil perancangan
- Siapkan hasil perancangan untuk tahap penerapan.

Perangkat Pemodelan Perancangan Terstruktur

Ada beberapa perangkat pemodelan yang dapat digunakan untuk menggambarkan hasil dari Perancangan Terstruktur. Dari beberapa perangkat pemodelan tersebut, yang paling umum digunakan diantaranya adalah *DFD*, *structure chart* dan *pseudo-code*. Berikut hanya akan dibahas pemodelan *structure chart*.

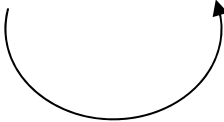
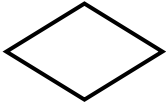
Structure chart adalah suatu teknis grafis untuk menggambarkan arsitektur sebuah program atau sistem yang besar secara keseluruhan tanpa memperlihatkan proses pemilihan dan pengulangannya secara rinci. Teknik ini akan menggambarkan arsitektur program atau sistem seperti diagram organisasi sebuah perusahaan. *Structure chart* mempunyai tiga simbol dasar sebagai komponen-komponen pembentuknya, yaitu :

Tabel 6.1. Simbol *Structure Chart*

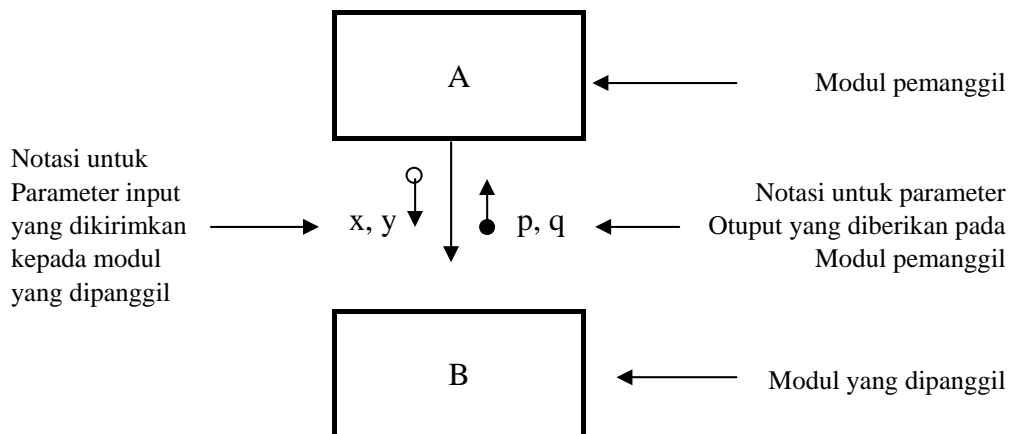
SIMBOL	KETERANGAN
	<u>Module</u> Simbol ini menunjukkan suatu modul
	<u>Connection</u> Digunakan untuk menghubungkan modul, atau simbol untuk menyatakan pemanggilan modul.
	<u>Couple</u> Menunjukkan data atau elemen kontrol yang dikirimkan atau diterima dari satu modul. Panah dengan lingkaran kosong menunjukkan data, sedangkan panah dengan lingkaran yang diblok menunjukkan elemen kontrol.

Dalam perkembangannya, ada dua simbol yang ditambahkan pada *structure chart*, yaitu :

Tabel 6.2. Simbol Tambahan *Structure Chart*

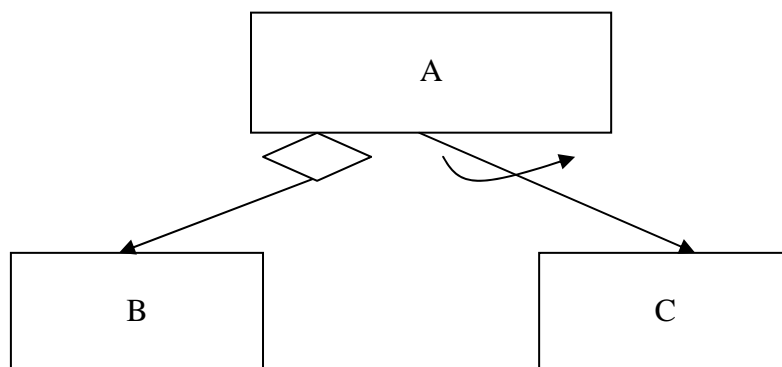
SIMBOL	KETERANGAN
	<u>Loop</u> Menunjukkan suatu pengulangan di dalam modul
	<u>Decision</u> Menunjukkan suatu penyeleksian kondisi di dalam modul, atau menunjukkan transform center

Gambar 6.1. berikut memperlihatkan contoh penggunaan simbol-simbol dasar untuk membentuk suatu *structure chart*. Sedangkan Gambar 6.2. memperlihatkan contoh lain dari penggunaan simbol-simbol tambahan untuk *structure chart*.



Gambar 6.1. Bentuk *Structure Chart* dengan Simbol-simbol Dasar

Structure chart di atas menyatakan bahwa modul A memanggil modul B dengan mengirimkan data x dan y sebagai parameternya. Setelah dieksekusi, modul B mengirimkan data p dan q *return value* ke modul A dan kendali proses kembali ke modul A.



Gambar 6.2. Bentuk *Structure Chart* dengan Simbol-simbol Tambahan

structure chart di atas memberikan arti bahwa modul A akan memanggil modul B jika kondisi dalam modul A dipenuhi, dan jika modul B selesai dieksekusi modul A kemudian akan memanggil modul C berulang-ulang.

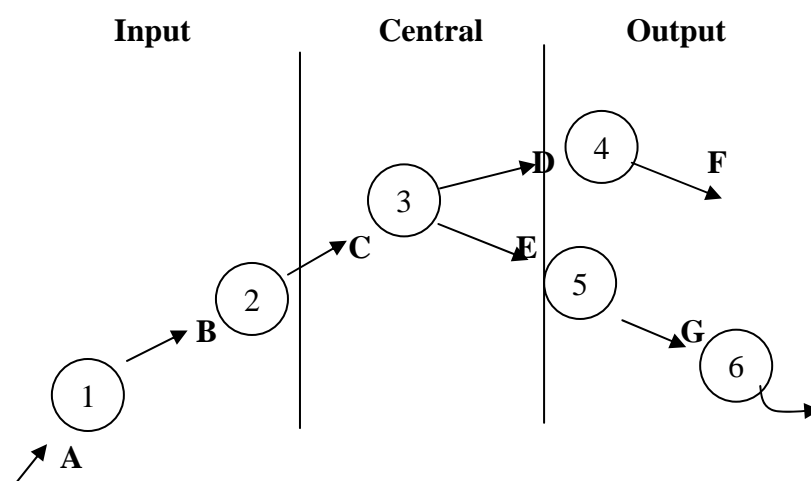
Transformasi DFD Menjadi Structure Chart

Untuk masalah-masalah yang sederhana, pembuatan *structure chart* untuk menggambarkan rancangan modul program dapat dilakukan tanpa harus memodelkan hasil analisisnya terlebih dahulu. Tetapi untuk masalah yang sudah cukup besar dan kompleks, pembuatan rancangan modul program tersebut harus dilakukan setelah hasil analisisnya selesai dimodelkan. Sekarang masalahnya adalah bagaimana membuat *structure chart* dari DFD hasil analisis dan perancangan awal tersebut. Secara praktis, pembuatan *structure chart* dari DFD dapat dilakukan dengan tahapan sebagai berikut :

1. Ubah diagram konteks, menjadi modul utama (*top module* atau *executive module*) dari *structure chart*.
2. Ubah DFD level-0 menjadi modul-modul yang dipanggil oleh modul utama. Jika pemanggilan modul untuk proses-proses yang ada pada DFD level-0 membutuhkan data atau *event* tertentu. Misalnya data atau *event* tersebut.
3. Ubah DFD level-1, 2, 3, dst menjadi modul-modul lainnya sesuai dengan fungsinya dengan pendekatan *Transform Analysis* dan atau *Transaction analysis*.
4. Evaluasi dan perbaiki *structure chart* yang didapat dengan memperhatikan *coupling*, *cohesion*, *fan in*, dan *program shape*

Transform Analysis

Transform analysis atau analisis transformasi adalah model aliran informasi yang digunakan untuk merancang program dengan mengenali komponen-komponen fungsional utama serta masukan dan keluarannya. Dalam DFD, sebuah transformasi direpresentasikan oleh suatu jaringan yang berbentuk linear (*linear network*). Gambar 5.3 menunjukkan sebuah DFD yang berbentuk transformasi.

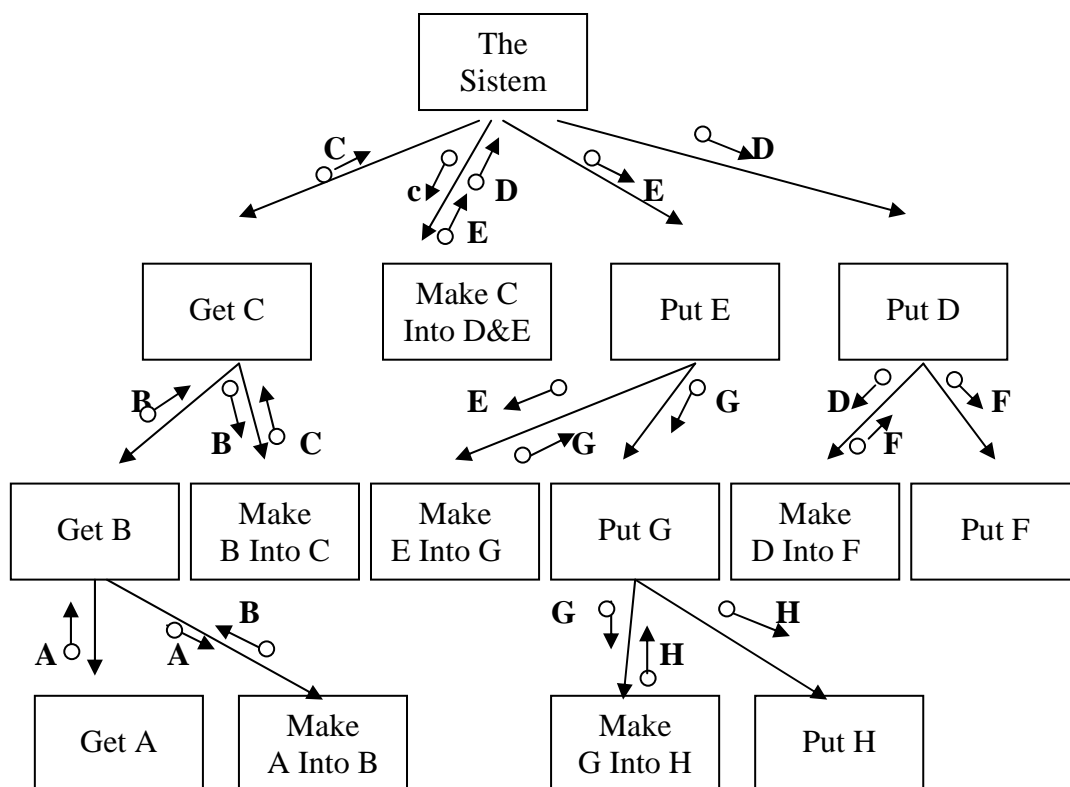


Gambar 6.3 DFD dengan Bentuk Transformasi

Untuk mengubah DFD berbentuk transformasi di atas caranya adalah :

1. Bagi DFD menjadi tiga kelompok bagian yaitu bagian input, pusat transformasi (*central tranform*), dan keluaran (lihat Gambar 3,3).
2. Gambarkan bagian pusat transpormasi, input dan output dari DFD masing-masing sebagai sebuah komponen fungsional (modul). Tempatkan pusat tranformasi atau komponen fungsional yang baru sebagai modul pemanggil di level atas dari structure chart, dan tempatkan yang lainnya di level berikutnya sebagai modul yang akan dipanggil.
3. Tambahkan sub fungsi-fungsi yang dibutuhkan untuk setiap level untuk melengkapi penggambaran structure chart-nya.

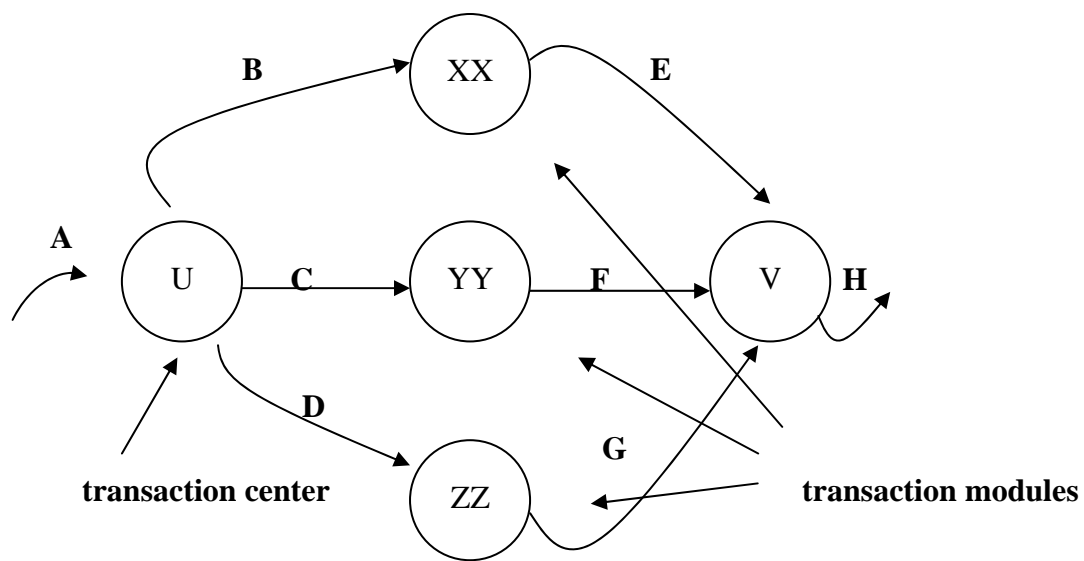
Structure chart hasil perubahan DFD yang berbentuk transformasi di atas ditujukan oleh Gambar 6.4 berikut ini .



Gambar 6.4 Structure Chart dari DFD Berbentuk Transformasi

Transaction Analysis

Transaction Analysis atau analisis transaksi merupakan strategi perancangan alternatif yang digunakan untuk merancang program-program yang memproses transaksi, yaitu elemen data yang memicu sebuah aksi. Gambar 5.5 berikut memperlihatkan sebuah DFD yang berbentuk transaksi.

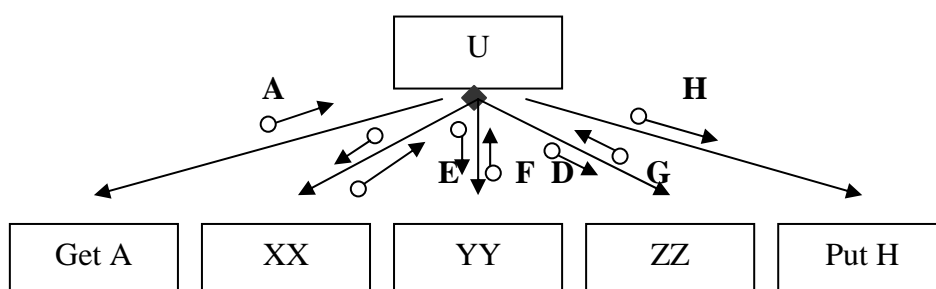


Gambar 6.5. DFD dengan bentuk Transaksi

Untuk mengubah DFD dengan bentuk transaksi menjadi *structure chart*, tahap proses yang harus dilakukan adalah :

1. Tentukan pusat transaksi (*transaction center*) dengan menelaah DFD
2. Kenali modul-modul transaksi dari DFD berdasarkan spesifikasi masalahnya.
3. Gambarkan pusat transaksi sebagai modul pemanggil untuk setiap modul transaksi dan lengkapi dengan sub fungsi yang dibutuhkan untuk setiap modul.

Gambar 6.6 berikut menunjukkan *structure chart* yang dihasilkan dari DFD yang berbentuk transaksi.



Keterangan :

◆ Menyatakan transaction center

Gambar 6.6 Structure Chart dari DFD Berbentuk Transaksi

BAB 7

IMPLEMENTASI DAN PEMELIHARAAN PERANGKAT LUNAK

7.1. Implementasi Perangkat Lunak

Tahap pengkodean merupakan suatu proses translasi. Rancangan detail ditranslasikan ke dalam suatu bahasa pemrograman, proses translasi dilanjutkan bila suatu kompiler menerima *source code* sebagai masukan dan menghasilkan *object code* yang akan diterjemahkan menjadi *machine code*. Bahasa pemrograman adalah alat yang digunakan untuk komunikasi antara manusia dan komputer.

Pemilihan Bahasa Pemrograman

Pemilihan bahasa pemrograman didasarkan atas :

- Lingkup aplikasi
- Algoritma dan kompleksitas
- Lingkungan pemrograman
- Performansi
- Struktur data
- Pengetahuan pemrogram
- Ketersediaan komputer

Macam bahasa pemrograman :

- Bahasa C sering digunakan untuk pengembangan
- Bahasa ADA, Modula2 untuk pemrograman waktu nyata
- Bahasa COBOL digunakan untuk aplikasi bisnis, tapi tempatnya mulai digantikan oleh 4 GL
- Bahasa FORTRAN dominan dalam bidang rekayasa dan sains
- Bahasa BASIC paling sering digunakan pada komputer pribadi
- Bahasa LISP dan Prolog sering digunakan pada aplikasi Kecerdasan Buatan
- Bahasa yang berkembang untuk Object Oriented adalah Smalltalk, C++, dan Eiffel

7.2. Pemeliharaan Perangkat Lunak

Karakteristik perawatan perangkat lunak :

- Aktivitas yang dilakukan untuk menyelesaikan kesalahan yang tidak terdeteksi sebelumnya dan muncul pada saat penggunaan perangkat lunak tersebut.
- Aktivitas yang membantu memecahkan perubahan teknologi komputer yang sangat cepat
- Aktivitas yang membantu mengatasi kebutuhan rekomendasi kapabilitas perangkat lunak yang baru
- Aktivitas yang membantu mengantisipasi perubahan bentuk peningkatan reliabilitas masa yang akan datang.

BAB 8

PENGUJIAN PERANGKAT LUNAK

Pengujian perangkat lunak merupakan suatu tahapan penting dalam pembangunan perangkat lunak. Pengujian dilakukan dengan cara mengevaluasi konfigurasi perangkat lunak yang terdiri dari spesifikasi kebutuhan, deskripsi perancangan, dan program yang dihasilkan. Hasil evaluasi kemudian dibandingkan dengan hasil uji yang diharapkan. Jika ditemukan kesalahan, maka perbaikan perangkat lunak harus dilakukan untuk kemudian diuji kembali.

8.1. Pengertian Pengujian

Pengujian perangkat lunak adalah proses menjalankan dan mengevaluasi sebuah perangkat lunak secara manual maupun otomatis untuk menguji apakah perangkat lunak sudah memenuhi persyaratan atau belum, atau untuk menentukan perbedaan antara hasil yang diharapkan dengan hasil sebenarnya. Pelaksanaan pengujian perangkat lunak biasanya disesuaikan dengan metodologi pembangunan perangkat lunak yang digunakan.

Pada umumnya, pengujian dilakukan sesudah tahap pemrograman. Namun demikian perencanaan pengujian dilakukan mulai tahap-tahap analisis. Untuk pendekatan *waterfall model*, tahap-tahap pengujian meliputi pengujian tahap analisis, perancangan, implementasi, instalasi dan pemeliharaan. Beberapa prinsip pengujian yang harus diperhatikan.

1. Dapat dilacak hingga ke persyaratan atau dokumen SRS
2. Pengujian harus direncanakan sebelum pelaksanaan pengujian
3. Pengujian harus dimulai dari hal yang kecil, diteruskan ke hal-hal yang besar.
4. Pengujian yang berlebihan tidak akan mungkin dapat dilaksanakan
5. Pengujian sebaiknya dilakukan oleh pihak ketiga.

8.2. Tujuan Pengujian

Tujuan yang diinginkan dari pelaksanaan pengujian perangkat lunak adalah :

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

Untuk melihat hasil pengujian yang telah dilakukan, dibuat suatu tabel yang berisi kriteria pengujian dan penelitian yang diberikan terhadap kriteria pengujian tersebut.

8.3. Tahap-tahap Pengujian

Untuk setiap tahap pengembangan, pelaksanaan pengujian perangkat lunak secara umum mengikuti tahap-tahap sebagai berikut :

1. Tentukan apa yang akan diukur melalui pengujian
2. Bagaimana pengujian akan dilaksanakan
3. Membangun suatu kasus uji (test case), yaitu sekumpulan data atau situasi yang akan digunakan dalam pengujian.
4. Tentukan hasil yang diharapkan atau hasil sebenarnya

5. Jalankan kasus pengujian
6. bandingkan hasil pengujian dan hasil yang diharapkan.

8.3.1. Pengujian Tahap Analisis

Pengujian pada tahap analisis ditekankan pada validasi terhadap kebutuhan, untuk menjamin bahwa kebutuhan telah dispesifikasikan dengan benar. Tujuan pengujian pada tahap ini adalah untuk mendapatkan kebutuhan yang layak dan untuk memastikan apakah kebutuhan tersebut sudah dirumuskan dengan baik. Faktor-faktor pengujian yang dilakukan pada tahap analisis ini meliputi :

1. Kebutuhan yang berkaitan dengan metodologi
2. Pendefinisian spesifikasi fungsional
3. Penentuan spesifikasi kegunaan
4. Penentuan kebutuhan portabilitas
5. Pendefinisian antar muka sistem.

8.3.2. Pengujian Tahap Perancangan

Pengujian tahap perancangan bertujuan untuk menguji struktur perangkat lunak yang diturunkan dari kebutuhan. Kebutuhan yang bersifat umum dirinci menjadi bentuk yang lebih spesifik .

Faktor-faktor pengujian yang dilakukan pada tahap perancangan meliputi :

1. Perancangan yang berkaitan dengan kebutuhan
2. Kesesuaian perancangan dengan metodologi dan teori.
3. Portabilitas rancangan
4. Perancangan yang dirawat
5. Kebenaran rancangan berkaitan dengan fungsi dan aliran data.
6. Kelengkapan perancangan antar muka.

8.3.3. Pengujian Tahap Implementasi

Pengujian pada tahap ini merupakan pengujian unit-unit yang dibuat sebelum diintegrasikan menjadi aplikasi keseluruhan. Faktor-faktor pengujian yang dilakukan pada tahap implementasi meliputi :

1. Kendali integritas data
2. Kebenaran program
3. kemudahan pemakaian
4. Sifat coupling
5. Pengembangan prosedur operasi.

8.3.4. Pengujian Tahap Pengujian

Tujuan pengujian pada tahap ini adalah untuk menilai apakah spesifikasi program telah ditulis menjadi instruksi-instruksi yang dapat dijalankan pada mesin. Selain itu, juga untuk menilai apakah instruksi yang ditulis tersebut telah sesuai dengan spesifikasi program. Faktor-faktor pengujian yang dilakukan pada tahap ini meliputi :

1. Pengujian fungsional
2. Dukungan manual
3. Kemudahan operasi.

8.3.5. Pengujian dengan Kasus Uji

Pengujian yang dilakukan meliputi pengujian unit (berupa prosedur atau fungsi) dan pengujian sistem. Dalam pengujian unit, unit-unit yang diuji meliputi unit-unit yang ada dalam sistem. Sedangkan pengujian sistem dilakukan terhadap sistem secara keseluruhan. Setiap pengujian dilakukan dengan menggunakan berbagai data masukan, baik data yang valid maupun tidak.

8.4. Teknik Pengujian

Ada dua teknik pengujian yang dapat digunakan untuk menguji perangkat lunak, yaitu *teknik black box dan white box testing*.

8.4.1. Pengujian Black Box

Pengujian black box digunakan untuk menguji fungsi-fungsi khusus dari perangkat lunak yang dirancang. Pada teknik ini, kebenaran perangkat lunak yang diuji hanya dilihat berdasarkan keluaran yang dihasilkan dari data atau kondisi masukan yang diberikan untuk fungsi yang ada tanpa melihat bagaimana proses untuk mendapatkan keluaran tersebut. Dari keluaran yang dihasilkan, kemampuan program dalam memenuhi kebutuhan pemakai dapat diukur sekaligus dapat diketahui kesalahan-kesalahannya. Beberapa jenis kesalahan yang dapat diidentifikasi :

- Fungsi tidak benar atau hilang
- Kesalahan antar muka
- Kesalahan pada struktur data (pengaksesan basis data)
- Kesalahan inisialisasi dan akhir program
- Kesalahan performansi.

Walaupun sulit untuk menelusuri kesalahan yang mungkin didapat, teknik pengujian black box lebih sering dipilih untuk menguji perangkat lunak karena kemudahan dalam pelaksanaannya.

8.4.2. Pengujian White Box

Berbeda dengan teknik black box teknik ini digunakan untuk mengetahui cara kerja suatu perangkat lunak secara internal. Pengujian dilakukan untuk menjamin operasi-operasi internal sesuai dengan spesifikasi yang telah ditetapkan dengan menggunakan struktur kendali dari prosedur yang dirancang. Pelaksanaan pengujian white box :

- Menjamin seluruh independent path dieksekusi paling sedikit satu kali. Independent path adalah jalur dalam program yang menunjukkan paling sedikit satu kumpulan proses ataupun kondisi baru.
- Menjalani logical decision pada sisi dan false
- Mengeksekusi pengulangan (looping) dalam batas-batas yang ditentukan
- Menguji struktur data internal.

8.5. Strategi Pengujian

Digunakan untuk mengintegrasikan metode-metode perancangan kasus pengujian perangkat lunak menjadi suatu langkah-langkah terencana dengan tujuan mendapatkan perangkat lunak yang sukses. Setiap strategi pengujian perangkat lunak harus meliputi perencanaan pengujian, perancangan kasus-kasus uji, eksekusi pengujian, pengumpulan data, serta evaluasi.

1. Pengujian unit program

Pengujian difokuskan pada unit terkecil dari suatu modul program. Dilaksanakan dengan menggunakan driver dan stub. Driver adalah suatu program utama yang berfungsi mengirim atau menerima data kasus uji dan mencetak hasil dari modul yang diuji. Stub adalah modul yang menggantikan modul sub-ordinat dari modul yang diuji.

2. Pengujian integrasi

Pengujian terhadap unit-unit program yang saling berhubungan (terintegrasi) dengan fokus pada masalah interfacing. Dapat dilaksanakan secara top-down integration atau bottom-up integration.

3. Pengujian validasi

Pengujian ini dimulai jika pada tahap integrasi tidak ditemukan kesalahan. Suatu validasi dikatakan sukses jika perangkat lunak berfungsi pada suatu cara yang diharapkan oleh pemakai.

4. Pengujian sistem

Pengujian yang dilakukan sepenuhnya pada sistem berbasis komputer.

- *Recovery testing*

Pengujian dilakukan dimana sistem diusahakan untuk gagal, kemudian diuji normalisasinya.

- *Security testing*

Dilakukan untuk menguji mekanisme proteksi

- *Stress testing*

Pengujian yang dirancang untuk menghadapkan suatu perangkat lunak kepada situasi yang tidak normal.

BAB 9

ANALISIS DAN PERANCANGAN BERORIENTASI OBJEK

9.1. Pendahuluan

A. Konsep Dasar Pendekatan Objek

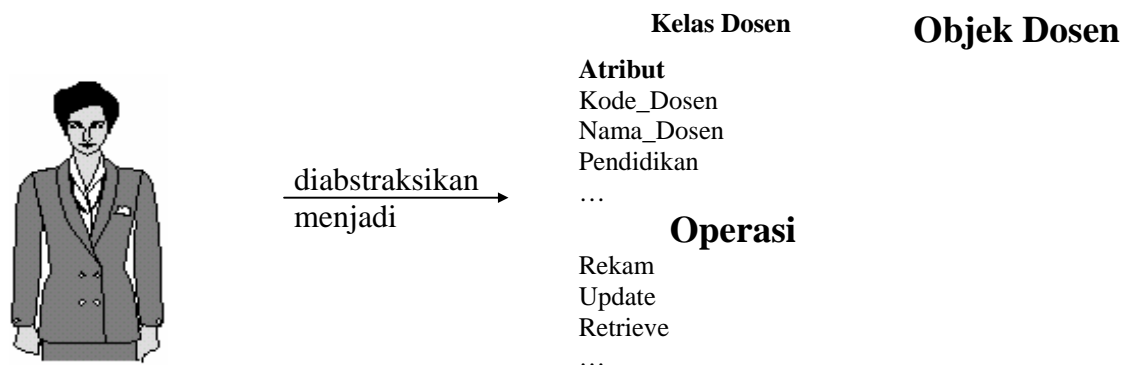
- Suatu teknik atau cara pendekatan baru dalam melihat permasalahan dari sistem (sistem perangkat lunak, sistem informasi, atau sistem lainnya).
- Pendekatan berorientasi objek akan memandang sistem yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata.
- Ada banyak cara untuk mengabstraksikan dan memodelkan objek-objek tersebut, mulai dari abstraksi objek, kelas, hubungan antar kelas sampai abstraksi sistem.
- Saat mengabstraksikan dan memodelkan objek ini, data dan proses-proses yang dimiliki oleh objek akan dienkapsulasi (dibungkus) menjadi satu kesatuan.

Contoh:

Tinjau aktivitas kuliah pada suatu sistem akademik sebagai berikut:



Dari aktivitas kuliah tersebut, secara eksplisit ada 3 objek yang langsung dapat dikenali yaitu **Dosen** yang memberikan kuliah, **Mahasiswa** yang mengikuti kuliah, dan **Materi Kuliah** yang disampaikan. Secara implisit, ada 2 objek lain yang bisa dikenali lagi yaitu **Jadwal** kapan kuliah diadakan dan **Nilai** yang didapat mahasiswa dari kuliah yang sudah diikutinya. Abstraksi dan pemodelan untuk salah satu dari kelima objek tersebut, misalnya untuk objek Dosen adalah:



- Dalam rekayasa perangkat lunak, konsep pendekatan berorientasi objek dapat diterapkan pada tahap analisis, perancangan, pemrograman, dan pengujian perangkat lunak.
- Ada berbagai teknik yang dapat digunakan pada masing-masing tahap tersebut, dengan aturan dan alat bantu pemodelan tertentu.

B. Objek dan Kelas

Apakah yang disebut objek? Apakah yang disebut kelas? Adalah sangat penting untuk membedakan antar objek dengan kelas

Objek

- Objek adalah abstraksi dari sesuatu yang mewakili dunia nyata seperti benda, manusia, satuan organisasi, tempat, kejadian, struktur, status atau hal-hal lain yang bersifat abstrak.
- Suatu entitas yang mampu menyimpan informasi (status) dan mempunyai operasi (kelakuan) yang dapat diterapkan atau dapat berpengaruh pada status objeknya.
- Dalam konteks OOP, objek adalah instansiasi (yang dibentuk secara seketika) dari kelas pada saat eksekusi (seperti halnya deklarasi variabel pada pemrograman prosedural). Jadi semua objek adalah instan dari kelas.
- Objek mempunyai siklus hidup: diciptakan, dimanipulasi, dan dihancurkan.

Kelas

- Kelas adalah kumpulan dari objek-objek dengan karakteristik yang sama.
- Kelas adalah definisi statik dari himpunan objek yang sama yang mungkin lahir atau diciptakan dari kelas tersebut.
- Sebuah kelas akan mempunyai sifat (atribut), kelakuan (operasi), hubungan (*relationship*) dan arti.
- Suatu kelas dapat diturunkan dari kelas yang lain, dimana atribut dari kelas semula dapat diwariskan ke kelas yang baru.

Kesimpulan:

- Objek adalah model eksekusi, sementara kelas adalah deskripsi statik dari objek yang mungkin lahir pada saat eksekusi.
- Pada saat eksekusi yang kita punyai adalah objek, sementara dalam pemodelan (analisis dan perancangan) dan teks program yang kita lihat adalah kelas.

Property Objek

Sebuah objek pada dasarnya mempunyai property sebagai berikut:

Atribut

- Nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek.
- Merupakan ciri dari sebuah objek
- Dipunyai secara individual oleh sebuah objek.
- Contoh: berat, warna, jenis, nama, dan sebagainya.

Layanan (*Service*)

- Metode atau operasi yang berfungsi untuk memanipulasi objek itu sendiri.
- Fungsi atau transformasi yang dapat dilakukan terhadap objek atau dilakukan oleh objek.
- Dapat berasal dari:
 - model objek
 - *event*
 - aktivitas atau aksi keadaan
 - fungsi

- kelakuan dunia nyata
- Contoh: *Read, Write, Move, Copy* dan sebagainya.

Klasifikasi Objek

Menurut [BOO95] objek dapat menjadi:

- ADT (*Abstract Data Type*)
Definisi dari kelas dimana komponen *type* menjadi atribut dan fungsi primitif menjadi operasi/metode/layanan.
- Mesin
Objek pasif yang punyai status yang akan diaktifkan oleh objek lain. Fungsi primitif pada mesin merupakan mekanisme transisi yang mengubah suatu status ke status lain.
- Proses
Objek aktif yang mempunyai “urutan kendali” (*thread of control*)

Sistem Berorientasi Objek

Definisi

- Sebuah sistem yang dibangun dengan berdasarkan metode berorientasi objek adalah sebuah sistem yang komponennya dibungkus (dienkapsulasi) menjadi kelompok data dan fungsi.
- Setiap komponen dalam sistem tersebut dapat mewarisi atribut dan sifat dari komponen lainnya, dan dapat berinteraksi satu sama lainnya.

Karakteristik Sistem Berorientasi Objek

Karakteristik atau sifat-sifat yang dipunyai sebuah sistem berorientasi objek adalah:

- Abstraksi
Prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.
- Enkapsulasi
Pembungkusan atribut data dan layanan (operasi-operasi) yang dipunyai objek, untuk menyembunyikan implementasi dari objek sehingga objek lain tidak mengetahui cara kerjanya.
- Pewarisan (*inheritance*)
Mekanisme yang memungkinkan satu objek (baca: kelas) mewarisi sebagian atau seluruh definisi dari objek lain sebagai bagian dari dirinya.
- *Reusability*
Pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan pada permasalahan lainnya yang melibatkan objek tersebut.
- Generalisasi dan Spesialisasi
Menunjukkan hubungan antara kelas dan objek yang umum dengan kelas dan objek yang khusus.
- Komunikasi antar Objek
Komunikasi antar objek dilakukan lewat pesan (*message*) yang dikirim dari satu objek ke objek lainnya.
- *Polymorphism*
Kemampuan suatu objek untuk digunakan di banyak tujuan yang berbeda dengan nama yang sama sehingga menghemat baris program.

9.2. Metodologi Berorientasi Objek

A. Beberapa Prinsip Dasar

Pengertian Metodologi

- Cara kerja yang sistematis untuk memudahkan pelaksanaan pembuatan perangkat lunak guna mencapai tujuan tertentu.
- Proses untuk menghasilkan perangkat lunak yang terorganisir dengan menggunakan sejumlah teknik dan konvensi notasi yang terdefinisi.

Pengertian Metodologi Berorientasi Objek

- Suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya.
- Suatu cara bagaimana sistem perangkat lunak dibangun melalui pendekatan objek secara sistematis.
- Metode berorientasi objek didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas.
- Metode berorientasi objek meliputi rangkaian aktivitas **analisis berorientasi objek, perancangan berorientasi objek, pemrograman berorientasi objek, dan pengujian berorientasi objek.**
- Ada teknik yang digunakan, produk yang dihasilkan, prosedur verifikasi, dan kriteria untuk setiap aktivitas yang dikerjakan.
- Ada alat bantu untuk memodelkan (mendokumentasikan) hasil dari setiap aktivitas.

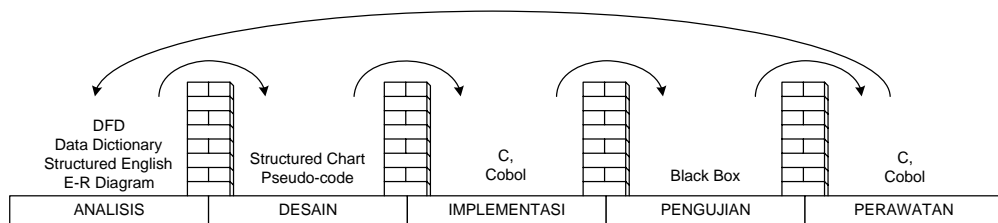
Metodologi Berorientasi Objek vs Fungsi

- Strategi utama untuk menangani kompleksitas pembangunan perangkat lunak adalah dekomposisi permasalahan menjadi bagian-bagian kecil yang dapat dikelola.
- Pada metode berorientasi fungsi atau aliran data (DFD), dekomposisi permasalahan dilakukan berdasarkan fungsi atau proses secara hirarki, mulai dari konteks sampai proses-proses yang paling kecil.
- Sementara pada metode berorientasi objek, dekomposisi permasalahan dilakukan berdasarkan objek-objek yang ada dalam sistem.

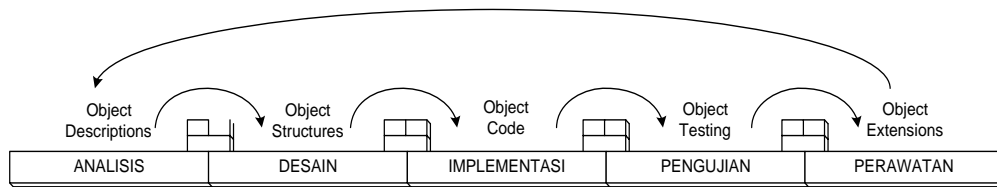
Mengapa Metodologi Berorientasi Objek ?

Metode berorientasi objek banyak dipilih karena :

- Metodologi lama banyak menimbulkan masalah
Adanya kesulitan pada saat mentransformasi hasil dari suatu tahap pengembangan ke tahap berikutnya, misalnya pada metode *Structured Analysis and Design*



Bandingkan dengan pendekatan objek sebagai berikut :



- Jenis aplikasi yang dikembangkan saat ini berbeda dengan masa lalu. Aplikasi yang dikembangkan pada saat ini sangat beragam (aplikasi bisnis, real-time, utility dan sebagainya) dengan platform yang berbeda-beda, sehingga menimbulkan tuntutan kebutuhan metodologi pengembangan yang dapat mengakomodasi ke semua jenis aplikasi tersebut.

Keuntungan Metodologi Berorientasi Objek

Pengembangan sistem dengan metode berorientasi objek dapat memberikan keuntungan-keuntungan sebagai berikut, walaupun beberapa buku (referensi) menunjukkan “pitfall” dari metodologi ini.

- Meningkatkan produktivitas
Karena kelas dan objek yang ditemukan dalam suatu masalah masih dapat dipakai ulang untuk masalah lainnya yang melibatkan objek tersebut (*reusable*).
- Kecepatan pengembangan
Karena sistem yang dibangun dengan baik dan benar pada saat analisis dan perancangan akan menyebabkan berkurangnya kesalahan pada saat pengkodean.
- Kemudahan pemeliharaan
Karena dengan objek, pola-pola yang cenderung tetap dan stabil dapat dipisahkan dari pola-pola yang mungkin sering berubah.
- Adanya konsistensi
Karena sifat pewarisan dan penggunaan notasi yang sama pada saat analisis, perancangan, maupun pengkodean.
- Meningkatkan kualitas perangkat lunak
Karena pendekatan pengembangan lebih dekat dengan dunia nyata dan adanya konsistensi pada saat pengembangannya, perangkat lunak yang dihasilkan akan mampu memenuhi kebutuhan pemakai serta mempunyai sedikit kesalahan.

B. Beberapa Metode Berorientasi Objek

Ada beberapa metode pengembangan perangkat lunak berorientasi objek yang sudah dikenal dan diantaranya adalah :

- *Object Oriented Analysis* (OOA) dan *Object Oriented Design* (OOD) dari Peter Coad dan Edward Yourdan [1990].
- *Object Modeling Technique* (OMT) dari James Rumbaugh, Michael Blaha, William Premerlan, Frederick Eddy dan William Lorensen [1991].
- *Object Oriented Software Engineering* (OOSE) dari Ivar Jacobson [1992].
- *Booch Method* dari Grady Booch [1994].
- *Syntropy* dari Steve Cook dan John Daniels [1994].
- *UML (Unified Modeling Language)* dari James Rumbaugh, Grady Booch dan Ivar Jacobson [1997].

9.3. ANALISIS BERORIENTASI OBJEK

A. Pengertian Dasar

Analisis

- Penguraian suatu pokok atas berbagai bagiannya dan penelaahan bagian itu sendiri serta hubungan antar bagian untuk memperoleh pengertian yang tepat dan pemahaman arti keseluruhan.
- Studi dari suatu permasalahan dengan cara memilah-milah permasalahan tersebut sehingga dapat dipahami dan dievaluasi, sebelum diambil tindakan-tindakan tertentu.

Analisis Berorientasi Objek

- Investigasi masalah untuk menemukan (mengidentifikasi) dan mendefinisikan objek-objek atau konsep-konsep yang ada di ruang masalah.
- Proses untuk menentukan objek-objek potensial yang ada dalam sistem dan mendeskripsikan karakteristik dan hubungannya dalam sebuah notasi formal.
- Aplikasi konsep berorientasi objek untuk memodelkan permasalahan dan sistem, baik untuk lingkup perangkat lunak maupun non-perangkat lunak.

Tujuan Analisis

- Memahami permasalahan secara menyeluruh.
- Mengungkapkan apa yang harus dikerjakan oleh sistem untuk memenuhi kebutuhan pemakai.
- Mengetahui ruang lingkup produk (*product space*) dan pemakai yang akan menggunakan produk tersebut.

Tahap Analisis

- Mempelajari permasalahan
- Menentukan kebutuhan pemakai
- Mengubah kebutuhan yang belum terstruktur menjadi model-model atau gambar-gambar dengan memanfaatkan metode dan teknik analisis tertentu.
- Mendokumentasikan hasil analisis, misalnya *Software Requirement Specification* (SRS).

B. Metode Analisis Berorientasi Objek

Pengertian

- Cara kerja yang sistematis untuk mengerjakan tahap analisis berdasarkan pendekatan objek.
- Ada kumpulan aturan-aturan tertentu yang harus diikuti untuk menyelesaikan pekerjaan analisis tersebut.
- Mempunyai urutan-urutan aktivitas, teknik, dan alat bantu (*tools*) tertentu untuk memodelkan (mendokumentasikan) hasil dari setiap aktivitas.
- Ada beberapa metode yang dapat digunakan untuk melakukan analisis berorientasi objek, dan diantaranya adalah sebagai berikut.

Metode Coad & Yourdan

- Diperkenalkan oleh Peter Coad dan Edward Yourdan pada tahun 1990.
- Disebut juga dengan nama *Object Oriented Analysis* (OOA), dan dipandang sebagai salah satu teknik yang mudah untuk dipelajari.
- Notasi model relatif sederhana karena didasarkan pada struktur fisik dunia nyata, dan petunjuk untuk melakukan analisis cukup jelas.
- Tahap atau skema pelaksanaan:
 - Identifikasi kelas dan objek
 - Identifikasi struktur

- Struktur "*generalization-specification*"
- Struktur "*whole-part*" atau "*a-part-of*"
- Identifikasi subjek
- Definisikan atribut
 - Atribut implisi objek
 - Koneksi instan (*instance connection*)
- Definisikan layanan
 - Layanan implisit objek
 - Layanan yang berasosiasi dengan atribut
 - Layanan yang berasosiasi dengan "*message-connection*"

Metode Rumbaugh

- Diperkenalkan oleh James Rumbaugh, Michael Blaha, William Premerlan, Frederick Eddy dan William Lorensen pada tahun 1991.
- Lebih dikenal dengan *Object Modeling Technique* (OMT) yang dapat digunakan baik untuk analisis maupun desain.
- Selain model-model fisik dari objek, pendekatan analisis dilakukan juga untuk model-model dinamik dan model fungsional.
- Tahap atau skema pelaksanaan:
 - Tentukan ruang lingkup masalah
 - Buat model objek
 - Identifikasi kelas yang relevan dengan permasalahan
 - Definisikan atribut dan asosiasi
 - Definisikan keterkaitan (*link*) antar kelas dan objek
 - Organisasikan kelas objek dengan menggunakan pewarisan
 - Buat model dinamik
 - Siapkan skenario
 - Definisikan kejadian (event) dan buat penelusurannya untuk setiap skenario
 - Bangun diagram aliran kejadian (event flow diagram)
 - Buat diagram keadaan (*state diagram*)
 - Buat model fungsional sistem
 - Identifikasikan masukan dan keluaran
 - Gunakan diagram aliran data untuk merepresentasikan aliran transformasi
 - Buat spesifikasi proses untuk setiap fungsi

Metode Jacobson

- Diperkenalkan oleh Ivar Jacobson dengan nama *Object Oriented Software Engineering* (OOSE) pada tahun 1992.
- Merupakan versi yang juga sederhana dari metode berorientasi objek.
- Sudut pandang atau fokus analisis ditekankan pada "*use case*", yaitu deskripsi atau skenario yang menggambarkan bagaimana pemakai berinteraksi dengan produk atau sistem yang akan dikembangkan.
- Tahap atau skema pelaksanaan:
 - Identifikasi pemakai sistem dan semua tanggung jawabnya
 - Buat model kebutuhan
 - Definisikan aktor dan tanggung jawabnya
 - Identifikasi use-case untuk setiap aktor
 - Inisialisasi gambaran sistem objek dan hubungannya
 - Buat model analisis

- Identifikasi antarmuka objek
- Buat gambaran struktural dari antarmuka objek
- Representasikan perilaku objek
- Isolasi sub-sistem dan buat masing-masing modelnya

Metode Booch

- Diperkenalkan oleh Grady Booch pada tahun 1994.
- Meliputi proses pengembangan makro dan mikro, dengan anggapan bahwa analisis dan desain merupakan rangkaian kesatuan aktivitas yang tidak dipisahkan.
- Tahap atau skema pelaksanaan:
 - Identifikasi kelas dan objek
 - Identifikasi kandidat objek
 - Identifikasi skenario yang relevan
 - Definisikan atribut dan layanan untuk setiap kelas
 - Identifikasi Semantik dari kelas dan objek
 - Pilih skenario kemudian analisis
 - Pilih objek dan daftar peran serta tanggung jawabnya
 - Cari kolaborasi diantara objek-objek
 - Identifikasi hubungan diantara kelas dan objek
 - Definisikan ketergantungan yang ada diantara objek
 - Jelaskan peran dari setiap objek
 - Validasi berdasarkan skenario
- Buat diagram yang berhubungan dengan langkah-langkah di atas
- Implementasikan kelas dan objek

C. Metode Analisis Secara Umum

- Pada prinsipnya semua metode analisis berorientasi objek adalah sama, perbedaan hanya terletak pada sudut pandang dan teknis pelaksanaannya.
- Secara umum, metode analisis berorientasi objek mencakup representasi kelas dan hirarki kelas, model hubungan objek, dan model perilaku objek.
- Tahap atau skema pelaksanaan analisis berorientasi objek :
 - Tentukan kebutuhan pemakai untuk sistem berorientasi objek
 - Identifikasi kelas dan objek
 - Identifikasi atribut dan layanan untuk setiap objek
 - Definisikan struktur dan hirarki
 - Buat model hubungan objek
 - Buat model perilaku objek

Menentukan Kebutuhan Pemakai untuk Sistem Berorientasi Objek

- Mengidentifikasi proses-proses bisnis dan kebutuhan pemakai dan mengekspresikan dengan ‘*use-case*’.
- Sebenarnya bukan merupakan aktivitas analisis berorientasi objek, karena tidak membicarakan pembahasan tentang objek.
- Diperlukan karena dapat menjelaskan aktivitas-aktivitas apa saja yang harus dikerjakan oleh sistem, dan menjelaskan juga perilaku dari komponen-komponen sistem.
- Ada diagram tertentu yang dapat merepresentasikan model kebutuhan dari ‘*use-case*’ yang diperoleh.

Identifikasi Kelas dan Objek

- Mengidentifikasi kelas-kelas dan objek-objek yang ada dalam lingkup aplikasi:
 - eksplisit pada pernyataan masalah
 - implisit pada lingkup aplikasi atau pengetahuan atas lingkup aplikasi
- Kelas dan objek dapat diidentifikasi dari:
 - **entitas** eksternal yang memproduksi dan memakai informasi yang akan digunakan oleh sistem berbasis komputer
 - sesuatu yang merupakan bagian dari wilayah informasi dari permasalahan
 - kejadian, misalnya prosedur operasional, yang muncul dalam lingkup operasional sistem
 - peran yang dimainkan oleh orang-orang yang berinteraksi dengan sistem
 - unit organisasi yang relevan dengan aplikasi
 - tempat yang menentukan ruang lingkup masalah dan seluruh fungsi dari sistem
 - struktur yang mendefinisikan kelas dari objek atau yang menghubungkan kelas-kelas objek.
- Abaikan kelas dan objek yang tidak tepat karena:
 - redunden
 - tidak relevan
 - lebih tepat berupa atribut
 - lebih tepat berupa operasi
 - lebih tepat berupa peran
 - lebih merupakan konstruksi implementasi.

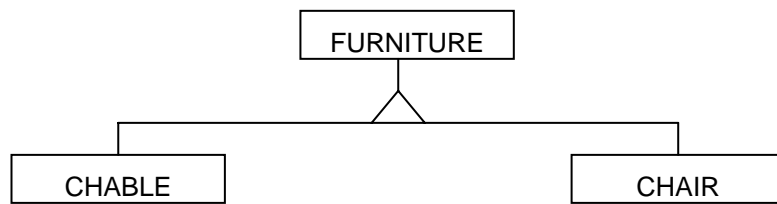
Identifikasi Atribut dan Layanan

- Mengidentifikasi atribut dan layanan yang terkait untuk setiap atribut tersebut.
- Atribut diidentifikasi dari elemen-elemen data yang dapat menggambarkan (mencirikan) sebuah objek secara utuh.
- Layanan diidentifikasi dari perilaku spesifik yang dapat menunjukkan peran dan tanggung jawab suatu objek.
- Abaikan atribut yang tidak tepat karena:
 - berupa objek
 - berupa qualifier
 - berupa nama
 - berupa identifier pada implementasi
 - menyatakan status internal objek
 - merupakan atribut yang sangat kecil (minor)
 - bertentangan dengan atribut lain

Definisi Struktur dan Hirarki

- Mendefinisikan struktur dan hirarki dari objek yang akan mengorganisasikan kelas objek.
- Mengatur dan menyederhanakan objek-objek menjadi kelas-kelas objek melalui konsep agregasi dan pewarisan.
- Struktur dan hirarki yang mungkin didefinisikan:
 - Struktur “*generalization-specification*”

Contoh:



- Struktur “*whole-part*” atau “*a-part-of*”

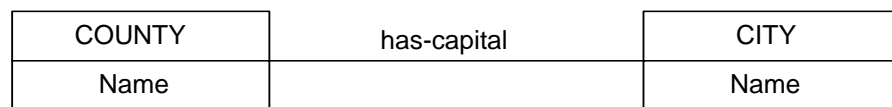
Contoh:



Buat Model Hubungan Objek

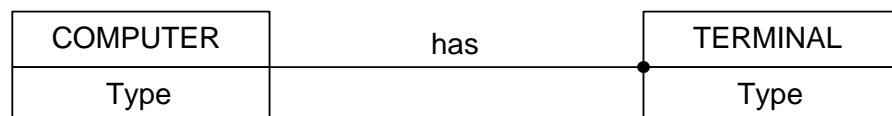
- Mendefinisikan hubungan (asosiasi atau koneksi) antar kelas, yaitu ketergantungan antar satu kelas atau lebih dengan kelas lainnya.
- Asosiasi dapat berbentuk:
 - lokasi fisik atau penempatan (next, to, part, of contained in)
 - aksi terarah (drive)
 - komunikasi (transmit to, acquires from)
 - kepemilikan (incorporated by, is composed of)
 - pemenuhan kondisi (*manages, coordinates, controls*)
- Jenis-jenis asosiasi:
 - Asosiasi 1 – 1 (*one-to-one association*)

Contoh:



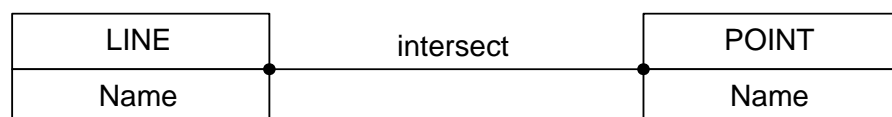
- Asosiasi 1 – n (*one-to-many association*)

Contoh:



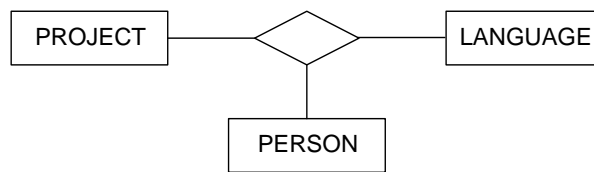
- Asosiasi n – n (*many-to-many association*)

Contoh:



- *Ternary Association*

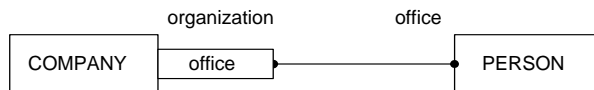
Contoh:



- *Kualifikasi*

Hubungan asosiatif berkualifikasi antara 2 kelas objek.

Contoh:



- *Ordering*

Hubungan berdasarkan urutan kejadian.

Contoh:



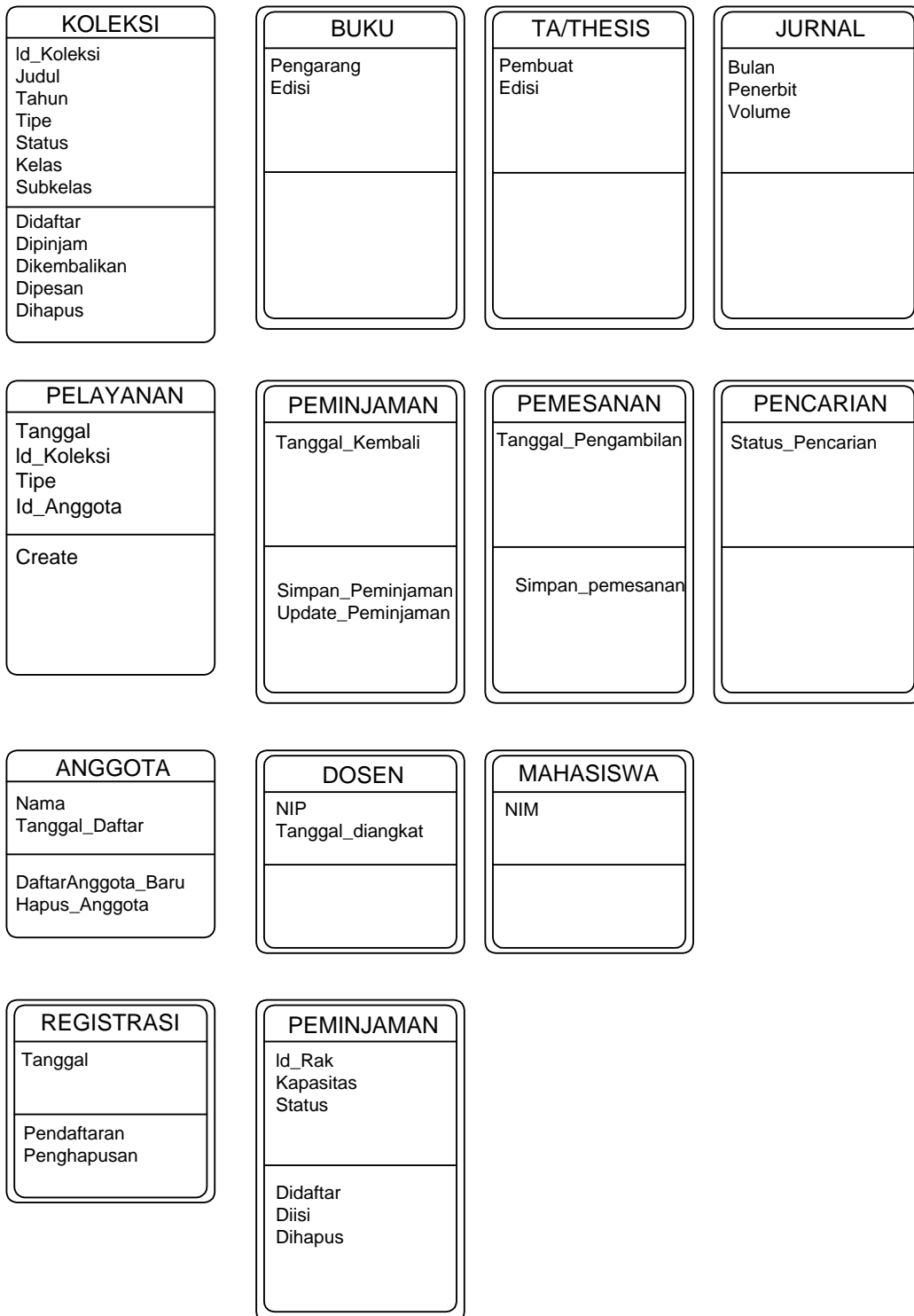
- Nama hubungan dan garis atau anak panah digunakan untuk menyatakan hubungan antar kelas-kelas tersebut.
- Abaikan asosiasi yang tidak tepat karena:
 - asosiasi antara kelas yang diabaikan
 - asosiasi implementatif atau tidak relevan
 - asosiasi yang berupa aksi
 - asosiasi ternary
 - asosiasi turunan

Buat Model Perilaku Objek

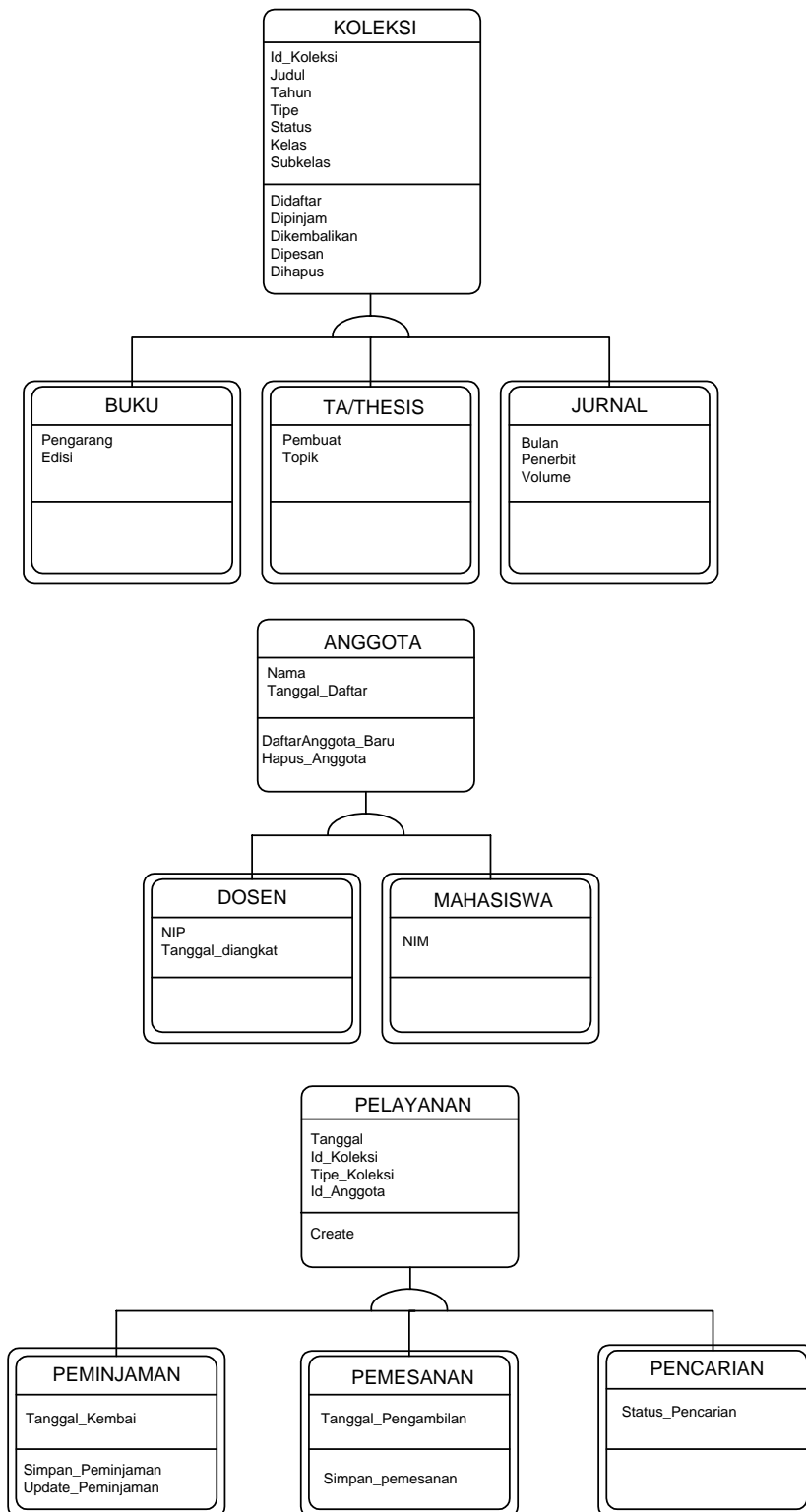
- Menyatakan bagaimana sistem berorientasi objek akan menanggapi kejadian atau stimuli eksternal (memunculkan sifat dinamis objek).
- Tahap-tahap untuk membuat model perilaku objek:
 - evaluasi semua “use-case” untuk memahami urutan interaksi yang ada dalam sistem
 - identifikasi kejadian yang menggerakkan urutan interaksi, dan pahami bagaimana kejadian-kejadian tersebut berhubungan dengan objek tertentu
 - buat penelusuran kejadian untuk setiap “use-case”
 - buat diagram transisi keadaan untuk sistem
 - tinjau ulang model perilaku objek untuk verifikasi keakuratan dan konsistensi

Contoh Hasil Analisis Berorientasi Objek dengan Metode Coad-Yourdon

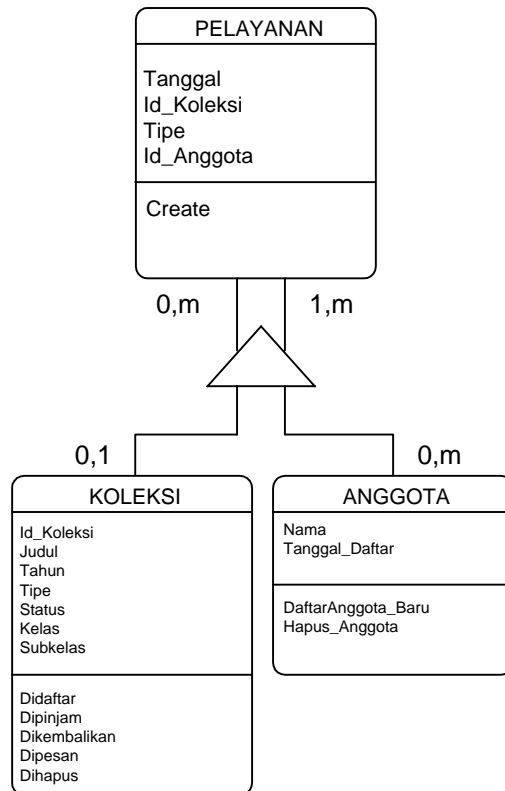
- Identifikasi Objek, Atribut, dan *Service*



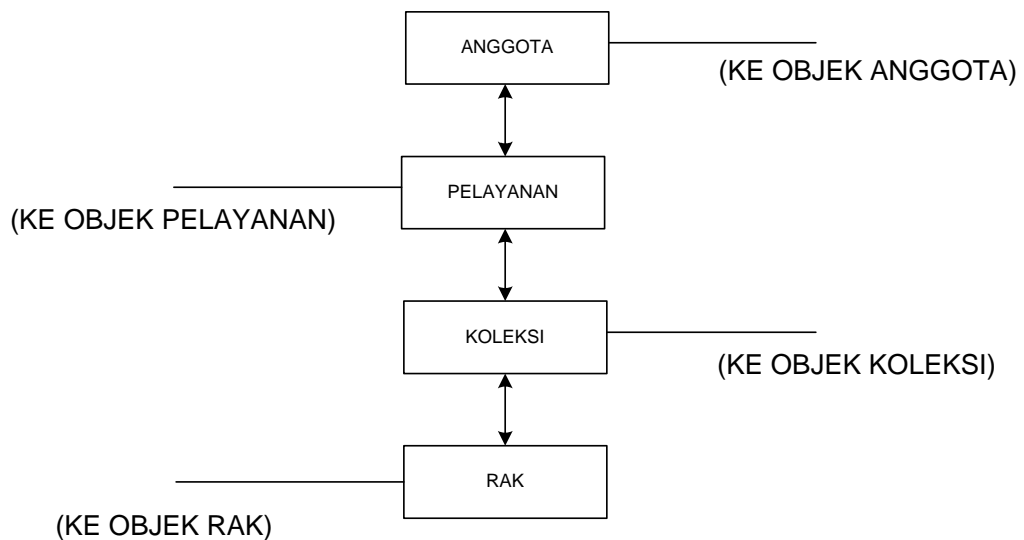
- **Identifikasi Struktur**
Struktur Generalisasi – Spesialisasi



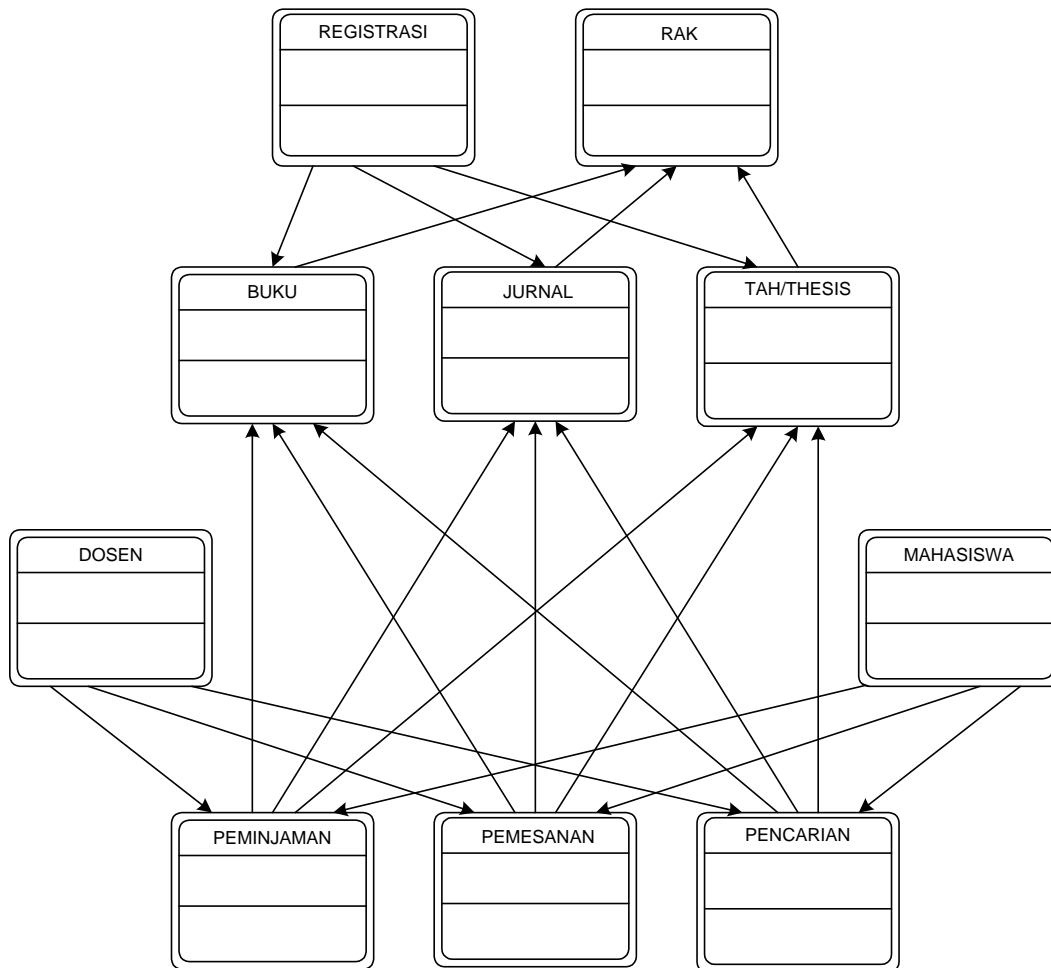
Struktur *Whole – Part*



- Identifikasi Subjek**



- **Pengiriman Message Antar Objek**



9.4. Perancangan Berorientasi Objek

Pengertian Dasar

Perancangan

- Proses untuk merencanakan atau mengatur segala sesuatu menurut tahapan tertentu, sebelum bertindak, mengerjakan, atau melakukan sesuatu tersebut [KBBI].
- Proses untuk mengaplikasikan berbagai macam teknik dan prinsip untuk tujuan pendefinisian secara rinci suatu perangkat, proses atau sistem agar dapat direalisasikan dalam suatu bentuk fisik [PRE97].
- Membuat solusi logika atau caar bagaimana kebutuhan-kebutuhan yang ada dipenuhi (diselesaikan) oleh sistem [LAR98].
- Pendefinisian arsitektur perangkat lunak, komponen, modul, antarmuka, pendekatan pengujian, dan data untuk memenuhi kebutuhan-kebutuhan yang sudah ditentukan sebelumnya [IEEE].

Perancangan Berorientasi Objek

- Proses untuk menerjemahkan model analisis hasil OOA menjadi model implementasi yang spesifik yang dapat direalisasi menjadi sebuah perangkat lunak [PRE97].
- Spesifikasi dari solusi perangkat lunak secara logika dalam kerangka objek-objek perangkat lunak, seperti kelas, atribut, metode dan hubungan antar kelas [LAR98].
- Proses pembangunan arsitektur sistem melalui konsep berorientasi objek [MEY97].

Tujuan Perancangan

- Secara umum, tujuan perancangan adalah menghasilkan suatu model atau penggambaran dari suatu entitas yang akan dibangun kemudian.
- Dalam konteks perancangan berorientasi objek (OOD), tujuan perancangan adalah menurunkan objek-objek dari setiap kelas dan bagaimana mengimplementasikan hubungan, perilaku dan komunikasi antar objek-objek tersebut [PRE97].

Proses Perancangan

Merupakan proses kreatif dalam pembangunan perangkat lunak untuk memecahkan suatu persoalan. Model dari proses perancangan secara garis besar terdiri dari empat tahap proses:

- Mengemukakan suatu solusi
- Membangun model dari solusi tersebut
- Evaluasi model terhadap spesifikasi kebutuhan yang telah ada
- Menjabarkan rincian spesifikasi dari solusi tersebut

Perancangan yang baik:

- Melaksanakan semua kebutuhan dan persyaratan yang tercantum pada dokumen SRS.
- Merupakan acuan yang dapat dibaca, dimengerti oleh pembuat program dan penguji perangkat lunak.
- Menyediakan gambaran lengkap dari perangkat lunak mencakup data, fungsi, dan tanggapan, dalam perspektif pelaksanaan pembuatan perangkat lunak.
- Menghasilkan model atau representasi dari perangkat lunak untuk digunakan dalam proses implementasi atau *coding*.

Tahap Perancangan

Dari sudut pandang manajemen proyek, perancangan terdiri dari dua bagian, yaitu:

- Perancangan awal (*preliminary design*)
Menentukan arsitektur perangkat lunak secara keseluruhan (*preliminary design*).
 - Bagaimanakah lingkungan programnya?
 - Bagaimana bentuk penyimpanan datanya?
 - Bagaimana bentuk antarmukanya?
- Perancangan rinci (*detailed design*)
Menentukan modul program (prosedural) yang harus dibuat

Adapun dari sudut pandang teknis, kegiatan perancangan terdiri dari aktivitas:

- Perancangan arsitektural program
 - arsitektural logika
 - arsitektural fisik
- Perancangan modul program (prosedural)
- Perancangan data
 - struktur data internal

- struktur data fisik
- Perancangan antarmuka
 - Perancangan antarmuka antar modul-modul
 - Perancangan antarmuka antar perangkat lunak dengan non-humanity (external entity)
 - Perancangan antarmuka pemakai

B. Metode Perancangan Berorientasi Objek

Pengertian

- Cara kerja yang sistematis untuk mengerjakan tahap perancangan berdasarkan pendekatan objek.
- Seperti halnya analisis, perancangan berorientasi objek mempunyai urutan-urutan aktivitas, teknik, dan alat bantu (tools) tertentu untuk memodelkan hasil dari setiap aktivitasnya.
- Beberapa metode yang dapat digunakan untuk melakukan perancangan berorientasi objek diantaranya adalah sebagai berikut.

Metode Coad & Yourdon

- *Problem domain component*
- *Human interaction component*
- *Task management component*
- *Data Management component*

Metode Rumbaugh

- *Perform design system*
- *Conduct object design*
- *Implement control mechanisms defined in system design*
- *Adjust class structure to strengthen inheritance*
- *Design messaging to implement the object relationship (associations)*
- *Package classes and associations into modules*

Metode Jacobson

- *Consider adaptations to make the idealized analysis model fit the real world environment*
- *Create blocks as the primary design object*
- *Create an interaction diagram shows how stimuli are passed between blocks*
- *Organize blocks into subsystems*
- *Review the design work*

Metode Booch

- *Architectural planning*
- *Tactical design*
- *Release planning*

C. Metode Perancangan Secara Umum

- Pada prinsipnya semua metode perancangan berorientasi objek adalah sama.
- Tahap pelaksanaan perancangan berorientasi objek secara umum:
 - Perbaiki dan lengkapi model objek hasil analisis
 - Perancangan objek

- Rancang setiap operasi pada level prosedural
- Definisikan kelas-kelas internal
- Rancang struktur data internal untuk setiap atribut kelas
- Rancang model pesan berdasarkan kerjasama (kolaborasi) dan hubungan antar objek
- Rancang antarmuka pemakai
- Kaji ulang model perancangan dan ulangi sesuai kebutuhan.

DAFTAR PUSTAKA

- Coad, Peter and Edward Yourdan. 1991. *Object Oriented Analysis*. New Jersey : Prentice Hall.
- Coad, Peter and Edward Yourdan. 1991. *Object Oriented Design*. New Jersey : Prentice Hall.
- Goodland, Mike. 1995. *SSADM A Practical Approach Version 4*. London : Mc Graw Hill
- Handoko, Mary. 1999. *Bahan Kuliah Manajemen Proyek Perangkat Lunak*. Bandung : Magister Informatika ITB
- Jacobson, I. 1995. *Object Oriented Software Engineering*. Edinburg Gate Harlow : Addison Wesley.
- Laksmiwati, Hira. 1998. *Bahan-bahan Kuliah Rekayasa dan Analisis Perangkat Lunak*. Bandung : Magister Informatika ITB
- Leman. 1998. *Metodologi Pengembangan Sistem Informasi*. Jakarta : Elex Media Komputindo
- Lorent, M and J Kidd. 1994. *Object Oriented Software Metrics*. New Jersey : Prentice Hall.
- Mahyuzir, Tavri D. 1989. *Analisis dan Perancangan Sistem Pengolahan Data*. Jakarta : Elex Media Komputindo
- Mahyuzir, Tavri D. 1991. *Pengantar Analisis dan Perancangan Perangkat Lunak*. Jakarta : Elex Media Komputindo
- Mardiyanto, M. Sukrisno 1998. *Bahan-bahan Kuliah Pembangunan Sistem Perangkat Lunak*. Bandung : Magister Informatika ITB
- Santoso. Oerip S. 1999. *Bahan-bahan Kuliah Uji Kualitas Perangkat Lunak*. Bandung : Magister Informatika ITB
- Sucahyo, Yudho Giri. 1997. *Bahan Kuliah Rekayasa Perangkat Lunak*. Jakarta : Ilmu Komputer UI
- Suharto, Toto. 2001. *Diktat Kuliah Rekayasa Perangkat Lunak*. Bandung : STT Telkom
- Pressman, Roger S. 1997. *Software Engineering*. New York : Mc Graw Hill.
- Rumbaugh, J. 1991. *Object Oriented Modeling and Design*. Edinburg Gate Harlow : Addison Wesley.