

Modul Ajar

Rekayasa Perangkat Lunak

Dikompilasi Oleh:

Dhany Indra Gunawan, S.T., M.Kom.

DAFTAR ISI

DAFTAR ISI	3
I Pengenalan Rekayasa Perangkat Lunak.....	1
1.1 Latar belakang Disiplin Rekayasa Perangkat Lunak.....	1
1.2 Krisis Perangkat Lunak.....	1
1.3 Rekayasa Perangkat Lunak.....	1
1.4 Mutu Perangkat Lunak	2
1.5 Kategori Perangkat Lunak.....	2
1.6 Karakteristik Perangkat Lunak.....	3
1.7 Proses Perangkat Lunak	3
1.8 Karakteristik Proses Perangkat Lunak.....	3
1.9 Daur Hidup Pembangunan Perangkat Lunak.....	3
1.10 Model Proses Perangkat Lunak	4
1.11 Model Waterfall (Air Terjun)	4
1.12 Biaya Perangkat Lunak	4
1.13 Pemilihan Sebuah Bahasa Pemrograman	5
1.14 Programming-in-the Small Concerns & Programming-in-the Large Concerns	5
2 Model Proses Perangkat Lunak	8
2.1 Pengembangan Perangkat Lunak	8
2.2 Siklus Pengembangan Perangkat Lunak.....	8
2.3 Model Proses Pengembangan Perangkat Lunak	8
3.2 Rekayasa Informasi	12
3.3 Rekayasa Produk.....	17
3.4 Pemodelan Arsitektur Sistem	19
3.5 Spesifikasi Sistem.....	21
4 Analisa Kebutuhan Perangkat Lunak	23
4.1 Kebutuhan.....	23
4.2 Analisis Kebutuhan	24
4.3 Spesifikasi Kebutuhan Perangkat Lunak (SKPL).....	27
4.4 Analisis Terstruktur.....	30
5 Perancangan Perangkat Lunak	41
5.1 Pengertian	41
5.2 Prinsip Perancangan.....	42
5.3 Konsep Perancangan	42
5.4 Transformasi Model Analisa ke Perancangan.....	43
5.5 Tahap Perancangan	44
5.6 Perancangan Data.....	44
6 Implementasi Perangkat Lunak.....	53
6.1 Aktivitas Implementasi	53
6.2 Aktivitas Pemrograman	53
6.3 Modularitas	55
6.4 Abstraksi Data	56
6.5 Analisis Statik	57
7 Pengujian Perangkat Lunak	59
7.1 Dasar-Dasar Pengujian Perangkat Lunak.....	59
7.2 Perancangan Kasus Uji	61
7.3 Pendekatan Strategis untuk Pengujian Perangkat Lunak	65
7.4 Masalah Strategis Pengujian	66
7.5 Strategi Pengujian Perangkat Lunak.....	66

7.6	Debugging	69
8	Pemeliharaan Perangkat Lunak.....	71
8.1	Pengertian Pemeliharaan	71
8.2	Kategori Pemeliharaan Perangkat Lunak.....	71
8.3	Permasalahan Pemeliharaan Perangkat Lunak.....	72
8.4	Model Pemeliharaan Perangkat Lunak	73
8.5	Proses Pemeliharaan Perangkat Lunak	74
8.6	Manajemen Pemeliharaan Perangkat Lunak.....	76
8.7	Perencanaan Pemeliharaan Perangkat Lunak	77
9	Object Oriented Concepts and Principles	79
9.1	Object Oriented.....	79
9.2	Perkembangan metode Object Oriented Analysis and Design (OOAD).....	84
9.3	Konsep OOAD	86
9.4	Object Management Group (OMG)	87
9.5	Tinjauan tentang Unified Modeling Language (UML).....	88
	Daftar Pustaka	90

I Pengenalan Rekayasa Perangkat Lunak

I.1 Latar belakang Disiplin Rekayasa Perangkat Lunak

Faktor-faktor yang melatarbelakangi munculnya RPL :

- Ketidakmampuan organisasi memprediksi waktu, usaha dan biaya u/ membangun perangkat lunak
- Perubahan nisbah/rasio biaya perangkat keras thd harga perangkat lunak
- Kemajuan pesat perangkat keras
- Kemajuan dalam teknik-teknik pembuatan perangkat lunak
- Tuntutan yang lebih tinggi thd jumlah perangkat lunak
- Tuntutan yang lebih tinggi thd mutu perangkat lunak
- Meningkatnya peran pemeliharaan

I.2 Krisis Perangkat Lunak

- Masalah nyata yang sudah mengganggu perkembangan perangkat lunak
- Serangkaian masalah yang terjadi dalam perkembangan perangkat lunak komputer.
- Masalah yang ada tidak hanya terbatas pada perangkat lunak yang tidak berfungsi dengan baik tapi juga pada penderitaan yang melingkupi masalah-masalah yang berhubungan dengan bagaimana mengembangkan perangkat lunak, bagaimana memelihara volume perangkat lunak yang sedang tumbuh dan bagaimana mengejar kebutuhan perangkat lunak lebih banyak lagi
- Penyebab krisis atau penderitaan Perangkat lunak dapat ditelusuri dengan sebuah mitologi yang muncul selama masa sejarah awal perkembangan perangkat lunak.
- Mitos perangkat lunak ini berbicara atas salah informasi dan keraguan. Masa sekarang kebanyakan kaum profesional memiliki banyak pengetahuan mengetahui berbagai mitos di bidang ilmu yang digelutinya (sikap yang salah yang menyebabkan masalah yang serius bagi manajer serta masyarakat teknis). Hal ini terlihat berbagai sisi pandang dari pihak Manajer/Sponsor, pelanggan atau Pengembang/Praktisi

I.3 Rekayasa Perangkat Lunak

Perangkat Lunak Merupakan program-program komputer dan dokumentasi yang berkaitan.

Produk perangkat lunak dibuat untuk pelanggan tertentu ataupun untuk pasar umum terdiri dari:

Generik – dibuat untuk dijual ke suatu kumpulan pengguna yang berbeda

Bespoke (custom) – dibuat untuk suatu pengguna tunggal sesuai dengan spesifikasinya.

Rekayasa perangkat lunak berasal dari 2 kata yaitu Software(Perangkat Lunak) dan Engineering (Rekayasa).

Perangkat Lunak (Software) adalah source code pada suatu program atau sistem. Perangkat lunak tidak hanya dokumentasi terhadap source code tapi juga dokumentasi terhadap sesuatu yang dibutuhkan selama pengembangan, instalasi, penggunaan dan pemeliharaan sebuah sistem.

Engineering atau Rekayasa adalah aplikasi terhadap pendekatan sistematis yang berdasar atas ilmu pengetahuan dan matematis serta aplikasi tentang produksi terhadap struktur, mesin, produk, proses atau sistem.

Rekayasa Perangkat Lunak adalah suatu disiplin rekayasa yang berkonsentrasi terhadap seluruh aspek.

Produk perangkat lunak mengadopsi pendekatan yang sistematis dan terorganisir terhadap pekerjaannya dan menggunakan tool yang sesuai serta teknik yang ditentukan berdasarkan masalah yang akan dipecahkan, kendala pengembangan dan sumber daya yang tersedia

Rekayasa Perangkat Lunak (RPL) juga merupakan pendekatan sistematis dan matematis u/ membangun, memelihara dan mengenyahkan perangkat lunak. Dari cara pandang lain, RPL adalah pendekatan sistematis u/ merekayasa p.l yang handal/bermutu, tepat waktu dan dengan biaya yang optimal.

1.4 Mutu Perangkat Lunak

Terdapat 3 pihak (minimal) yang mempengaruhi mutu p.l yaitu

- **Sponsor**

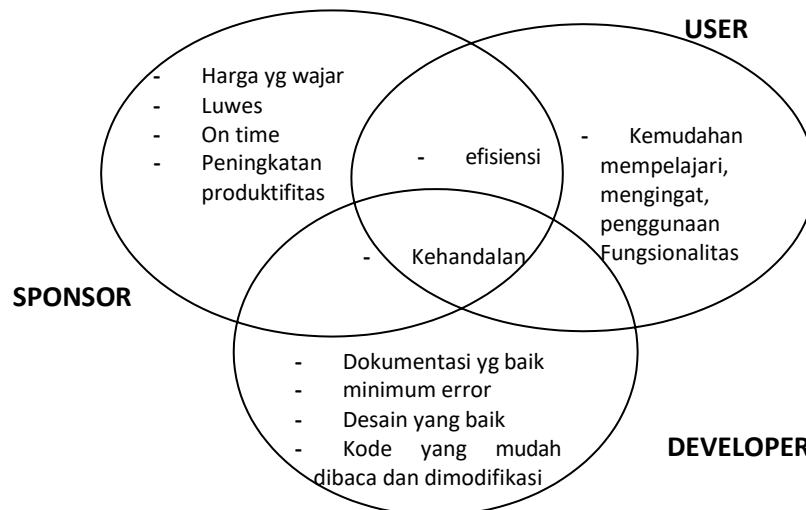
Seseorang atau organisasi yang membiayai/membayar selama pengembangan atau perantaraan sistem software dan biasanya mempunyai respon terhadap pengembangan sistem software itu sendiri dengan melibatkan perhitungan biaya yang optimal.

- **User**

Setiap orang yang secara langsung berinteraksi terhadap eksekusi software, yang secara langsung memberi input ke komputer dan menggunakan/menikmati output dari komputer.

- **Developer**

Seseorang atau organisasi yang memberikan modifikasi dan memelihara terhadap error serta mengembangkan sistem software tersebut.



Gambar Sisi Pandang dari komponen kategori terhadap Mutu Perangkat Lunak

Masing-masing komponen kategori mempunyai sudut pandang tersendiri thd mutu suatu perangkat lunak. Tapi kriteria tersebut tidak saling independen.

1.5 Kategori Perangkat Lunak

Kategori Perangkat lunak secara umum dapat dikelompokkan sebagai berikut:

- **Perangkat Lunak Sistem**, Sekumpulan program yang ditulis untuk melayani program-program yang lain. Seperti kompilator, editor dan utilitas pengatur file.
- **Perangkat Lunak Real-Time**, Program-program yang memonitor/menganalisi/mengontrol kejadian dunia nyata pada saat terjadinya (real-time event)
- **Perangkat Lunak Bisnis**, memroses informasi bisnis spt payroll, inventory dll.
- **Perangkat Lunak Teknik dan Ilmu Pengetahuan**, ditandai dengan penggunaan algoritma *number crunching*.
- **Embedded Software**, produk yang ada dalam *read-only memory* dan dipakai untuk mengontrol hasil dan sistem untuk keperluan konsumen dan pasar industri
- **Perangkat Lunak Komputer Personal**, sesuai kebutuhan personal spt pengolah kata, angka dan manajemen database
- **Perangkat Lunak Kecerdasan Buatan**, menggunakan algoritma non-numeris untuk memecahkan masalah kompleks yang tidak sesuai untuk perhitungan atau analisis secara langsung.

1.6 Karakteristik Perangkat Lunak

Atribut Perangkat Lunak seharusnya memberikan pengguna kebutuhan fungsionalitas dan unjuk kerja yang dapat di rawat, berguna.

Dalam Buku Software Engineering Ian Sommerville, Perangkat Lunak mempunyai Karakteristik sebagai berikut:

1. Maintainability (Dapat Dirawat), Perangkat Lunak harus dapat memenuhi perubahan kebutuhan
2. Dependability, Perangkat Lunak harus dapat dipercaya
3. Efisiensi, Perangkat Lunak harus efisien dalam penggunaan resource
4. Usability, Perangkat Lunak harus dapat digunakan sesuai dengan yang direncanakan

1.7 Proses Perangkat Lunak

Proses Perangkat Lunak merupakan Sekumpulan aktifitas yang memiliki tujuan untuk pengembangan ataupun evolusi perangkat lunak.

Aktifitas umum dalam semua proses perangkat lunak terdiri dari:

1. *Software Specification* – apa yang harus dilakukan oleh perangkat lunak dan batasan/kendala pengembangannya
2. *Software Development* – proses memproduksi sistem perangkat lunak
3. *Software Validation* – pengujian perangkat lunak terhadap keinginan pengguna
4. *Software Evolution* – perubahan perangkat lunak berdasarkan perubahan keinginan.

Suatu proses model adalah suatu representasi abstrak suatu model. Proses model menampilkan suatu deskripsi suatu proses dari beberapa perspektif tertentu,

Proses Perangkat Lunak dapat dikatakan sebagai aktifitas yang saling terkait (koheren) untuk menspesifikasikan, merancang, implementasi dan pengujian sistem perangkat lunak.

1.8 Karakteristik Proses Perangkat Lunak

Karakteristik Proses Perangkat Lunak terdiri dari:

- *Understandability*, membuat proses secara eksplisit didefinisikan dan bagaimana sehingga mudah untuk mengerti definisi proses
- *Visibility*, Aktifitas proses menghasilkan hasil yang jelas sehingga tahapan proses yang dilakukan terlihat
- *Supportability*, Aktifitas Proses dapat didukung atas CASE tools
- *Acceptability*, Penerimaan atas proses yang terdefinisi dan yang digunakan oleh Engineer selama pembangunan Produk Perangkat Lunak.
- *Reliability*, Proses didesain dalam suatu metode untuk dihindarkan dari kesalahan
- *Robustness*, Proses dapat meneruskan dalam masalah yang tidak diharapkan terjadi
- *Maintainability*, Proses yang merefleksikan atas perubahan terhadap permintaan atau perbaikan proses yang diidentifikasi
- *Rapidity*, bagaimana cepat dapat berjalan atas proses pengiriman atau implementasi sebuah sistem dari Spesifikasi yang ada sampai selesai

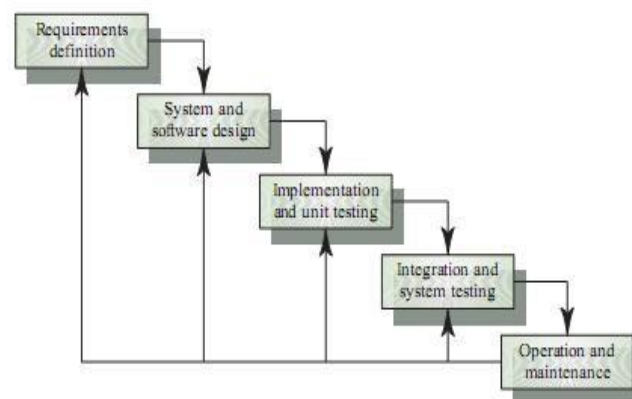
1.9 Daur Hidup Pembangunan Perangkat Lunak

Di dalam pengembangan rekayasa perangkat lunak biasanya dipandu dengan pemodelan dengan Daur Hidup Perangkat Lunak (*Software Development Life Cycle*).

Tak ada standar sehingga bervariasi model proses u/ menggambarkan rekayasa daur hidup p.l. Namun tahap-tahap yang prinsipal terhadap pemetaan model proses kedalam aktifitas pengembangan yang fundamental adalah sbb:

1. *Requirement Analysis and definition*
2. *System and Software Design*
3. *Implementation and unit testing*
4. *Integration and system Testing*
5. *Operation and maintenance*

Model Air Terjun (Water Fall)



Model di atas secara esensial sama dengan model 'waterfall' atau air terjun. Kita dapat menganalogikan daur hidup p. L dengan daur hidup manusia dari benih s/d tua dan meninggal.

1.10 Model Proses Perangkat Lunak

Model Proses Perangkat Lunak merupakan suatu representasi proses perangkat lunak yang disederhanakan, dipresentasikan dari perspektif khusus. Contoh perspektif proses:

- Perspektif Alur-kerja (workflow) - barisan kegiatan
- Perspektif Alur Data (Data flow) – alur informasi
- Perspektif Peran/Aksi – siapa melakukan apa.

Menurut Ian Sommerville, Model proses secara umum terdiri dari:

1. **Pendekatan Model Air terjun (Water fall)**, Menempatkan semua aktifitas sesuai dengan tahapan pada model Waterfall dengan memisahkan dan membedakan antara spesifikasi dan pengembangan
2. **Pengembangan yang berevolusi**, Pendekatan yang melanjutkan Aktifitas satu dan yang lainnya dari Spesifikasi dan pengembangan serta validasi secara cepat
3. **Pengembangan sistem Formal**, Pendekatan aktifitas bersasar suatu model sistem matematika yang ditransformasikan ke implementasi,
4. **Pengembangan Sistem berbasis Re-use (penggunaan ulang) komponen**, sistem dibangun dari komponen yang sudah ada dengan fokus integrasi sistem.

1.11 Model Waterfall (Air Terjun)

Fase Model Air Terjun

1. Analisis Kebutuhan dan pendefinisian
2. Perancangan sistem dan Perangkat Lunak
3. Implementasi dan unit testing
4. Integrasi dan pengujian sistem
5. Pengoperasian dan perawatan

Proses kembali ke state sebelumnya untuk mengantisipasi perubahan Setelah proses menuju ke suatu state di bawahnya adalah sangat sulit.

Masalah pada Model Air Terjun:

- Partisi proyek ke stages yang berbeda tidak fleksibel.
- Hal ini mengakibatkan sulitnya untuk merespon perubahan kebutuhan pengguna
- Oleh sebab itu model ini hanya cocok digunakan apabila kebutuhan pengguna sudah dimengerti dengan baik,

1.12 Biaya Perangkat Lunak

Sekitar 60% untuk biaya pengembangan, 40% biaya pengujian. Untuk perangkat lunak berbasis pengguna (custom), biaya evolusi biasanya melebihi biaya pengembangan.

Biaya beragam tergantung pada tipe sistem yang akan dikembangkan dan kebutuhan sistem seperti unjuk kerja dan kehandalan sistem,

Distribusi biaya bergantung pada model pengembangan yang digunakan.

1.13 Pemilihan Sebuah Bahasa Pemrograman

Pemilihan terhadap penggunaan Bahasa Pemrograman dalam pengembangan sistem merupakan hal yang krusial. Hal ini sangat esensi dalam siklus Hidup Pengembangan Sistem. Pada tahapan Coding dan Testing difasilitasi bahasa yang terpilih dan pada Tahapan Pemeliharaan lebih mudah dikerjakan. Apalagi dihubungkan dengan biaya dalam pengembangan perangkat lunak.

Terdapat 2 faktor yang berhubungan terhadap Pemilihan Bahasa Pemrograman yaitu:

1. Perihal Pragmatik Pemilihan Bahasa
2. Bahasa Pemrograman yang dipilih

Pragmatik dalam Pemilihan Bahasa Pemrograman mengandung perihal berikut:

1. Sponsor Requirement, Permintaan Sponsor
2. Knowledge of coders, Pengetahuan yang mudah dipahami Programmer
3. Languages used in previous and/or concurrent projects, Bahasa Pemrograman yang digunakan proyek sebelumnya atau proyek yang berbarengan terkait dengan Pengetahuan Programmer
4. Availability and quality of language compiler, Ketersediaan dan kualitas Kompiler Bahasa Pemrograman yang sesuai target hardware yang digunakan
5. Availability of supporting software development tools, Ketersediaan Alat Bantu Perangkat lunak Pendukung Editor, Debugger, Linker dan yang lainnya
6. Portability, Sistem yang dikembangkan dapat beroperasi di berbagai mesin komputer dan berjalan pada aneka Sistem Operasi yang berbeda

1.14 Programming-in-the Small Concerns & Programming-in-the Large Concerns

Karakteristik Bahasa Pemrograman secara individu dipertimbangkan ketika memilih sebuah bahasa target yang akan digunakan pada Pengembangan Perangkat Lunak yang efisien, reliable dan pada Pemeliharaan perangkat lunak.

Seorang Software Engineer dapat memperlihatkan fitur-fitur bahasa pemrograman yang mendukung pada pengembangan Perangkat Lunak berasal dari 2 aspek yang berbeda (Rujukan Bell, Morrey & Pugh, 1987) yaitu:

1. Programming-in-the Small, Menguji atau mencoba fitur-fitur yang mendukung dengan Pengkodean program modul-modul tunggal dan program-program kecil oleh kepentingan Programmer secara individu.
2. Programming-in-the Large, Pemrograman ini merujuk pada pengembangan sebuah system yang keseluruhannya dipengaruhi oleh koordinasi atas sekelompok orang (Software Engineer), dimana setiap engineer membuat respon komponen-komponen pada system dengan bagian yang berbeda-beda.

1.14.1 Programming-in-the Small

Dalam Pemrograman kekuatan bahasa adalah kemampuan membawa suatu pekerjaan pembuatan program yang diinginkan secara mudah dan dengan membuat pengaruh kebutuhan pemrograman itu kecil. Kekuatan bahasa itu secara langsung berkaitan dengan fitur-fitur yang ada dan struktur data yang mendukung bahasa tersebut.

Karakteristik Fitur-fitur yang terkait dengan pemrograman dengan skala kecil (Programming-in-the Small) terdiri dari:

- a. Sifat Simplicity, Clarity dan Orthogonality dari bahasanya
- b. Sintaks dari bahasa pemrogramannya
- c. Jumlah dan tipe kontrol struktur (Decision structure, loop control structure dan exception handling)
- d. Abstraksi Data terhadap tipe struktur datanya

Simplicity sebuah bahasa pemrograman merupakan ukuran dari kamus data dari bahasa seperti jumlah operator, operan dan reserved word (seperti if..then dsb) dalam bahasa itu. Simplicity ini perihal yang diinginkan karena sifat ini membuat programmer menjadi familier terhadap keseluruhan isi bahasa yang digunakan.

Clarity merupakan tingkat dari bahasa berkaitan dengan pengertian sintaks natural, pemahaman dan ketidakbingungan programmer dalam menggunakan bahasa pemrograman sampai berjalan.

Orthogonality merupakan tingkat dari programmer bebas mengkombinasikan fitur-fitur yang ada di bahasa pemrograman itu seperti pemanfaatan fungsi yang memiliki nilai balik untuk diolah langsung dengan nilai yang lain dan tipe data tertentu.

1.14.2 Programming-in-the Large

Fitur-fitur pada pemrograman ini terdapat pada pembuatan modul antarmuka, dukungan terhadap fungsional dan abstraksi data dan bagian kompilasi terhadap modul-modul program secara independen oleh linkage editor atau linker.

Programming ini dibutuhkan pada saat sebuah system mengandung banyak baris kode atau mempunyai skala yang besar yang membutuhkan pendekatan pengembangan sistem secara integrasi terhadap sekelompok proyek system.

Konsep penting dalam pemrograman terkait dengan penyelesaian masalah oleh seorang programmer adalah abstraksi prosedural yang berbentuk modul prosedural/fungsi dan abstraksi data. Pada Pemrograman berskala besar (Programming-in-the Large) meminta abstraksi pada level yang lebih tinggi dengan dukungan abstraksi data yang sesuai dengan modul prosedural yang terkait.

Pada Pemrograman ini, bahasa pemrograman seharusnya mempunyai fitur karakteristik sbb:

- a. Mekanisme untuk enkapsulasi atau pembentukan kelompok tingkat tinggi terhadap abstraksi prosedural dan data
- b. Pemisahan yang jelas antara spesifikasi deskripsi sebuah abstraksi dan implementasi abstraksi
- c. Mekanisme untuk memproteksi akses dari luar terhadap informasi yang dienkapsulasi
- d. Metode yang sederhana pada pemberian modul ke bagian modul lain (reusable atau Peggunaan kembali suatu modul ke modul lain)

Unit Enkapsulasi bisa tersiri dari program unit, subprogram, package, tasks dan generic units.

1. Sebutkan faktor faktor yang melatarbelakangi munculnya rekayasa perangkat lunak.
2. Sebutkan karakteristik perangkat lunak
3. Sebutkan hal-hal yang terkandung dalam pemilihan bahasa pemrogramman
4. Jelaskan apa yang dimaksud dengan programming-in-the small
5. Jelaskan apa yang dimaksud dengan programming-in-the large

2 Model Proses Perangkat Lunak

2.1 Pengembangan Perangkat Lunak

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 [4].

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, antarmuka 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).

2.2 Siklus Pengembangan Perangkat Lunak

Siklus pengembangan perangkat lunak atau sering disebut juga dengan siklus hidup perangkat lunak adalah [4]:

- 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.

2.3 Model Proses Pengembangan Perangkat Lunak

Model proses perangkat lunak (atau disebut juga paradigma rekayasa perangkat lunak) adalah suatu strategi pengembangan yang memadukan lapisan proses, metode, dan alat serta tahap-tahap generik. Model proses untuk rekayasa perangkat lunak dipilih berdasarkan sifat proyek dan aplikasi, metode dan alat yang digunakan, serta pengendalian dan hasil yang diinginkan. Berikut adalah beberapa model proses pengembangan perangkat lunak.

2.3.1 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. Pemodelan dan rekayasa sistem/informasi.
Menetapkan kebutuhan untuk seluruh elemen sistem dan kemudian memilah mana yang untuk pengembangan perangkat lunak.
2. Analisis kebutuhan perangkat lunak
3. Perancangan
4. Pembuatan kode
5. Pengujian
6. Pemeliharaan

Beberapa kelemahan linear sequential model:

1. Proyek yang sebenarnya jarang mengikuti alur sekuensial, sehingga perubahan yang terjadi dapat menyebabkan hasil yang sudah didapat tim harus diubah kembali.
2. Linear sequential model mengharuskan semua kebutuhan pemakai sudah dinyatakan secara eksplisit di awal proses, tetapi kadang-kadang hal 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.

2.3.2 Prototyping Model

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

1. Mendefinisikan objektiif 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. Walaupun pemakai melihat berbagai perbaikan dari setiap versi prototype, tetapi pemakai mungkin tidak menyadari bahwa versi tersebut dibuat tanpa memperhatikan kualitas dan pemeliharaan jangka panjang.
2. Pengembang kadang-kadang membuat kompromi implementasi dengan menggunakan sistem operasi yang tidak relevan dan algoritma yang tidak efisien.

2.3.3 RAD (Rapid Application Development) Model

Merupakan model proses pengembangan perangkat lunak secara linear sequential yang menekankan pada siklus pengembangan yang sangat singkat. Pendekatan RAD model mempunyai cakupan:

1. Pemodelan bisnis
2. Pemodelan data
3. Pemodelan proses
4. Pembuatan aplikasi
5. Pengujian dan pergantian

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 berbagai 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.

2.3.4 Incremental Model

Merupakan kombinasi linear sequential model (diaplikasikan secara berulang) dan filosofi pengulangan dari prototyping model. Setiap tahapan linear sequential menghasilkan *deliverable increment* bagi perangkat lunak, dimana *increment* pertamanya merupakan sebuah produk inti yang mewakili kebutuhan dasar sistem. Produk inti ini nantinya dikembangkan menjadi *increment-increment* selanjutnya setelah digunakan dan dievaluasi sampai didapat produk yang lengkap dan memenuhi kebutuhan pemakai.

Kelemahan incremental model:

1. Hanya akan berhasil jika tidak ada *staffing* untuk penerapan secara menyeluruh.
2. Penambahan staf dilakukan jika hasil *incremental* akan dikembangkan lebih lanjut.

2.3.5 Spiral Model

Merupakan model proses perangkat lunak yang memadukan wujud pengulangan dari model prototyping dengan aspek pengendalian dan sistematika dari linear sequential model. Dalam model ini perangkat lunak dikembangkan dalam suatu seri *incremental release*. Spiral model dibagi menjadi 6 aktivitas kerangka kerja sebagai berikut:

1. Komunikasi dengan pemakai
2. Perencanaan
3. Analisis resiko
4. Rekayasa
5. Konstruksi dan pelepasan
6. Evaluasi

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 yang relatif baru.

2.3.6 Component Assembly Model

Menggabungkan berbagai karakteristik dari spiral model. Pembuatan aplikasi dengan pendekatan model ini dibangun dari komponen-komponen perangkat lunak yang sudah dipaketkan sebelumnya dengan cakupan aktivitas sebagai berikut:

1. Mengidentifikasi calon-calon komponen (kelas objek)
2. Melihat komponen-komponen dalam pustaka
3. Mengekstrak komponen jika ada
4. Membangun komponen jika tidak ada
5. Menyimpan komponen baru pada pustaka
6. Mengkonstruksi iterasi ke-n dari sistem

2.3.7 Fourth Generation Techniques (4GT)

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

1. Pengumpulan kebutuhan.
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 dibandingkan dengan bahasa pemrograman, dan juga kode sumber yang dihasilkannya tidak efisien.

Latihan

1. Jelaskan apa yang dimaksud dengan pengembangan perangkat lunak!
2. Jelaskan juga apa yang dimaksud dengan model proses pengembangan perangkat lunak!
3. Sebutkan tahap-tahap pada suatu siklus pengembangan perangkat lunak
4. Ada beberapa model proses yang dapat digunakan untuk mengembangkan perangkat lunak. Jelaskan apa yang dimaksud dengan model-model proses berikut:
 - a. *Waterfall*
 - b. *Prototyping*
 - c. RAD
 - d. Incremental
 - e. Spiral
5. Uraikan secara ringkas tahap-tahap pengembangan perangkat lunak untuk masing-masing model proses tersebut!
6. Pada saat atau kondisi yang bagaimana masing-masing pendekatan tersebut digunakan untuk menyelesaikan pengembangan perangkat lunak? Berilah contoh untuk melengkapi jawaban anda!

3 Rekayasa Sistem

Menurut Pressman [1], cakupan rekayasa sistem meliputi:

- Rekayasa informasi, yaitu rekayasa sistem yang konteks pekerjaannya berfokus pada perusahaan bisnis (business enterprise), meliputi pengumpulan kebutuhan-kebutuhan untuk tingkat bisnis strategis dan tingkat area bisnis.
- Rekayasa produk (sering disebut juga dengan rekayasa sistem), yaitu rekayasa sistem yang merupakan aktivitas penyelesaian masalah. Data, fungsi, dan perilaku produk yang diinginkan dicari, dianalisis, dibuat model kebutuhannya, kemudian dialokasikan ke komponen rekayasa. Selanjutnya komponen-komponen ini disatukan dengan infrastruktur pendukungnya sampai produk tersebut jadi.

Kedua bentuk rekayasa di atas digunakan untuk pengembangan sistem berbasis komputer. Dalam hal ini untuk mengalokasikan peran perangkat lunak komputer serta menentukan kaitan yang menyatukan perangkat lunak dengan elemen sistem berbasis komputer lainnya.

3.1.1 Sistem Berbasis Komputer

Sistem berbasis komputer dapat didefinisikan sebagai [1]:

Kumpulan atau susunan elemen-elemen yang diorganisasi untuk mengerjakan berbagai tujuan (goal) yang sudah didefinisikan sebelumnya dengan cara memproses informasi. Elemen-elemen sistem berbasis komputer [1]:

- Perangkat lunak, yaitu program komputer, struktur data, dan dokumentasi terkait.
- Perangkat keras, yaitu perangkat elektronik yang menyediakan kemampuan komputasi dan perangkat elektromekanik (misalnya: sensor, motor, pompa) yang menyediakan fungsi dunia luar.
- Manusia, yaitu pemakai dan operator perangkat keras dan perangkat lunak.
- Basis data, yaitu kumpulan informasi yang besar dan terorganisasi yang diakses melalui perangkat lunak.
- Dokumentasi, yaitu buku-buku manual, formulir, dan informasi deskriptif lainnya yang menggambarkan penggunaan dan atau operasional sistem.
- Prosedur, yaitu langkah-langkah yang menjelaskan pemakaian spesifik dari setiap elemen sistem.

Beberapa contoh sistem berbasis komputer:

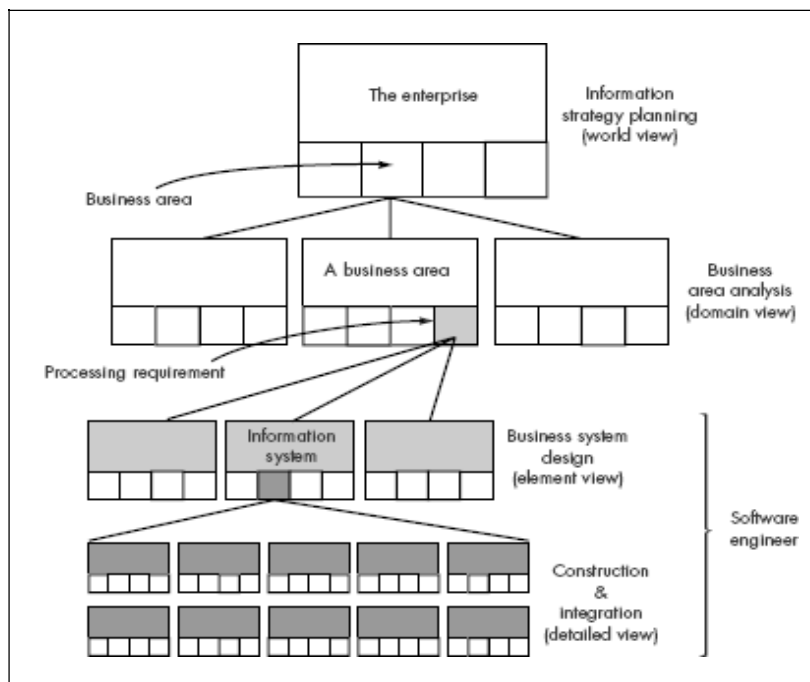
- Sistem informasi, yaitu sistem yang akan mengolah data dari dalam atau luar organisasi menjadi informasi untuk mendukung proses operasional dan manajerial.
- Sistem kendali proses, yaitu sistem yang mengendalikan proses-proses fisis dengan bantuan perangkat elektromekanik dan sensor tertentu.
- Sistem pakar, yaitu sistem yang mengaplikasikan metodologi penalaran pengetahuan untuk ranah tertentu sehingga dapat memberikan saran atau rekomendasi layaknya seorang pakar.

3.2 Rekayasa Informasi

Tujuan dari rekayasa informasi/information engineering (IE) adalah:

- Mendefinisikan suatu arsitektur yang memungkinkan bisnis menggunakan informasi secara efektif.
- Membuat rencana menyeluruh untuk mengimplementasi arsitektur-arsitektur tersebut, yaitu:
 1. arsitektur data: kerangka kerja untuk informasi yang dibutuhkan bisnis/fungsi bisnis. Informasi dari arsitektur ini merupakan objek data yang digunakan oleh suatu basisdata dan ditransformasikan menjadi sebuah informasi yang dibutuhkan oleh bisnis/fungsi bisnis.
 2. arsitektur aplikasi: elemen sistem yang mentransformasi objek pada arsitektur data untuk berbagai keperluan bisnis. Arsitektur aplikasi ini bisa dianggap sebagai sistem program (perangkat lunak) yang melakukan transformasi data tersebut. Namun dalam konteks yang luas arsitektur aplikasi bisa mencakup peranan manusia dan prosedur bisnis yang belum terotomatisasi.
 3. infrastruktur teknologi: fondasi untuk arsitektur data dan aplikasi, mencakup perangkat keras dan perangkat lunak yang digunakan untuk mendukung data dan aplikasi. Hal ini bisa mencakup komputer, jaringan komputer, teknologi penyimpanan, dan arsitektur untuk mengimplementasikan teknologi tersebut.

Untuk memodelkan arsitektur diatas, ditetapkan sebuah hirarki aktivitas rekayasa informasi seperti yang terlihat pada gambar 3.1 dibawah ini.



Gambar 3.1 Hirarki Rekayasa Informasi

Aktifitas dalam hirarki diatas dilakukan melalui *perencanaan strategi informasi (Information Strategy Planning)* yang akan dibahas pada sub bab berikutnya. Perencanaan strategi informasi ini memandang bisnis secara keseluruhan sebagai sebuah entitas dan memisahkan domain bisnis yang penting dalam sebuah enterprise (perusahaan). Selain itu perencanaan strategi informasi mendefinisikan objek data, hubungannya, serta bagaimana data tersebut mengalir dalam domain bisnis yang terlihat pada tingkat enterprise (perusahaan).

Selanjutnya dari pandangan domain pada aktivitas rekayasa informasi disebut sebagai *analisis area bisnis/business area analysis* yang mengidentifikasi data (entitas/objek data) dan fungsi yang dibutuhkan oleh area bisnis (domain) yang telah dipilih.

Keluaran dari aktivitas analisis area bisnis adalah untuk memisahkan area dimana sistem informasi dapat mendukung area bisnis tersebut. Dalam pandangan elemen pada aktivitas rekayasa informasi, kebutuhan dasar dari sistem informasi tersebut akan di modelkan dan diterjemahkan ke dalam arsitektur data, arsitektur aplikasi dan infrastruktur teknologi. Aktivitas ini disebut dengan *desain sistem bisnis/business system design*.

Setelah aktivitas desain sistem bisnis dilakukan, aktivitas konstruksi dan integrasi /construction and Integration yang berfokus pada detail dari implementasi. Kemudian desain arsitektur dan infrastruktur pada aktivitas desain sistem bisnis akan diimplementasikan dengan mengkonstruksikan/membangun basisdata dan struktur data, membangun aplikasi yang menggunakan komponen-komponen program, serta memilih infrastruktur teknologi yang sesuai untuk mendukung desain tersebut. Kemudian akan diintegrasikan untuk membentuk sebuah sistem informasi atau aplikasi yang lengkap.

3.2.1 Perencanaan Strategi Informasi

Perencanaan Strategi Informasi merupakan langkah pertama dalam aktivitas rekayasa informasi. Tujuan dari aktivitas ini adalah :

1. Menentukan sasaran dan tujuan dari bisnis.

Sasaran yang dimaksud disini merupakan sebuah pernyataan umum dari arah yang ingin dicapai. Sebagai contoh sasaran bisnis untuk perusahaan perakitan mobil adalah mengurangi biaya pembuatan produk. Sedangkan Tujuan bisnis disini merupakan bentuk kegiatan kuantitatif dari sasaran. Sebagai contoh untuk mencapai sasaran supaya perusahaan perakitan mobil dapat mengurangi biaya pembuatan produk, perusahaan mempunyai tujuan sebagai berikut:

- mengotomatisasi pemasangan komponen yang dilakukan secara manual.
- mengimplementasikan sistem kontrol produksi secara real-time.
- mendapatkan konsensi harga 10% dari pemasok,
- dan lainnya.

Jadi dapat disimpulkan bahwa sasaran lebih cenderung ke strategi sedangkan tujuan bersifat taktis.

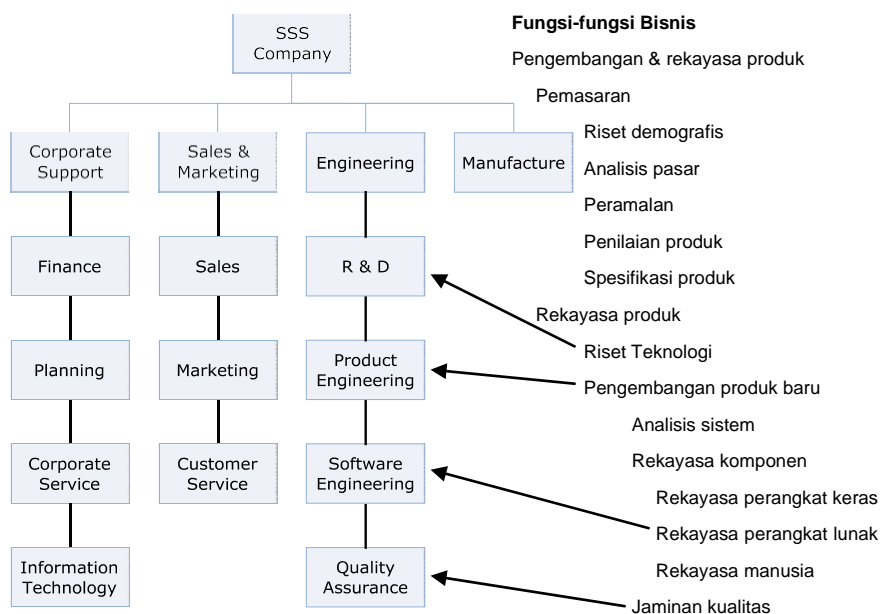
2. Mengisolasi faktor-faktor sukses dan kritis yang memungkinkan tujuan dan sasaran dari bisnis tercapai.
Faktor sukses dan kritis ini harus ada, jika sasaran dan tujuan bisnis akan dicapai, sehingga perencanaan manajemen harus mengakomodasinya. Beberapa faktor sukses kritis untuk sasaran perusahaan perakitan mobil seperti pada contoh diatas berupa:
 - mesin dengan realibilitas yang tinggi.
 - motivasi dan pelatihan pekerja.
 - rencana penjualan untuk meyakinkan pemasok menurunkan harga,
 - dan lainnya.
3. Menganalisa pengaruh teknologi dan otomasi terhadap tujuan dan sasaran dari bisnis.
Tujuan analisa pengaruh teknologi adalah menguji sasaran dan tujuan serta memberikan indikasi mengenai teknologi-teknologi yang akan berpengaruh langsung dan tidak langsung terhadap sasaran dan tujuan dari bisnis. Menganalisa informasi yang ada untuk menentukan perannya dalam pencapaian tujuan dan sasaran bisnis.

Pemodelan Enterprise

Tujuan dari pemodelan enterprise untuk mengetahui pandangan terhadap sebuah bisnis, yaitu

- Menentukan struktur dan fungsi organisasional dalam area bisnis yang digambarkan oleh struktur organisasi.
- Mendekomposisi fungsi bisnis
- Menghubungkan sasaran, tujuan dan faktor sukses kritis dengan organisasi dan fungsinya.

Pemodelan enterprise juga akan menciptakan model data tingkat bisnis yang menentukan objek data dan hubungannya dengan elemen model perusahaan lain yang akan dibahas pada sub bab 3.2.1.2. Gambar 3.3 merupakan pemodelan enterprise beserta area bisnis dan fungsi-fungsi bisnisnya.



Gambar 3.3 Diagram organisasi dan pemetaan fungsi bisnis ke area bisnis

Masing-masing kotak dalam diagram organisasi diatas menunjukkan area bisnis enterprise. Kemudian fungsi-fungsi bisnis dan proses bisnis akan diidentifikasi, kemudian fungsi bisnis tersebut akan dihubungkan dengan area bisnis yang bertanggung jawab terhadap fungsi tersebut. Fungsi bisnis ini merupakan aktivitas yang dimiliki oleh sebuah area bisnis yang harus diselesaikan untuk mendukung keseluruhan bisnis enterprise. Sedangkan proses bisnis sebuah proses yang menerima input dan mengeluarkan output yang merupakan transformasi dari fungsi bisnis pada area bisnis tertentu.

Untuk memahami sebuah fungsi bisnis ditransformasikan ke dalam serangkaian proses bisnis, perhatikan fungsi analisis pasar pada gambar 3.3.

Proses bisnis pada fungsi bisnis analisis pasar berupa :

- Mengumpulkan semua data penjualan
- Menganalisa data penjualan
- Mengembangkan profil pembeli
- Mempelajari trend pembelian
- dan lainnya.

3.2.1.1 Pemodelan Data Tingkat Bisnis

Pemodelan data tingkat bisnis menurut schaum adalah sebuah aktivitas pemodelan enterprise yang berfokus pada objek data (entitas) yang dibutuhkan untuk mencapai sasaran dan fungsi bisnis dari sebuah enterprise (sub bab 3.2.1.1).

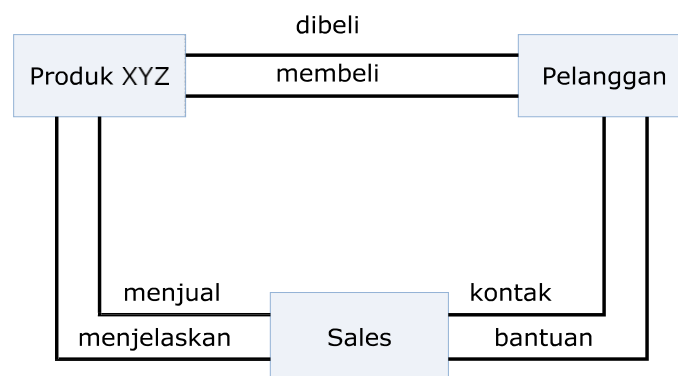
Objek data yang akan dimodelkan pada tingkat bisnis pada umumnya merupakan data yang berhubungan dengan informasi konsumen dan produsen, seperti pelanggan, barang, penjualan, data pegawai dan lainnya. Sebagai contoh perekayasa informasi akan menentukan objek data pelanggan sebagai objek yang berperan dalam pemodelan enterprise, untuk menggambarkan objek pelanggan lebih jelas ditentukan atribut-atribut dari objek tersebut yaitu :

Objek : **Pelanggan**

Atribut :

Nama
Nama Perusahaan
Pekerjaan
Alamat Bisnis
Produk yang diminati
Pembelian
Tanggal kontak terakhir
Status kontak

Setelah menentukan semua objek data yang ada dalam pemodelan enterprise, kemudian akan dilakukan keterhubungan antar objek data tersebut. Sebagai contoh gambar 3.4 menunjukkan hubungan antar objek data pelanggan dengan sebuah produk XYZ dan seorang sales. Pada gambar 3.4 menunjukkan keterhubungan dua arah antar objek data, yaitu seorang pelanggan membeli sebuah produk XYZ dan produk XYZ dibeli oleh seorang pelanggan. Namun secara detail informasi tambahan dari masing-masing objek data ini akan dijelaskan pada bab 6.



Gambar 3.4 Hubungan antar objek data

3.2.2 Analisa Area Bisnis

Analisa area bisnis merupakan tahapan aktifitas rekayasa informasi dari sisi pandangan domain. Dalam aktivitas ini perekayasa informasi akan menganalisa dan menggambarkan bagaimana objek data digunakan dan ditransformasikan pada masing-masing area bisnis, serta bagaimana fungsi dan proses bisnis pada area bisnis mentransformasikan objek data tersebut. Untuk memodelkan objek data dan transformasinya terhadap area bisnis dapat menggunakan sejumlah model yang berbeda yaitu

- Model data
- Model aliran proses
- Diagram dekomposisi proses
- Berbagai matriks lintas referensi

Objek data yang telah disaring pada tahap perencanaan strategi informasi akan digunakan pada masing-masing area bisnis. Sebagai contoh pada sub bab 3.2.1.2 objek data pelanggan akan digunakan oleh bagian penjualan. Setelah dilakukan analisis dan evaluasi kebutuhan bagian penjualan (domain penjualan), selanjutnya objek data pelanggan akan disaring kembali disesuaikan dengan kebutuhan yang diinginkan oleh penjualan.

Objek : **Pelanggan**

Atribut :

Nama

Nama Perusahaan → **Objek : Perusahaan**

Pekerjaan

Alamat Bisnis

Produk yang diminati

Pembelian

Tanggal kontak terakhir → rekaman kontak

Status kontak → status kontak terakhir

→ tanggal kontak selanjutnya

→ sifat kontak yang disepakati

Nama perusahaan merupakan atribut yang dimodifikasi untuk objek data yang lain yaitu Perusahaan. Objek perusahaan ini tidak hanya berisi atribut nama perusahaan saja, namun ada atribut tambahan yang lainnya seperti alamat perusahaan, kebutuhan pembelian, besar perusahaan dan lainnya. Selain itu ada beberapa atribut yang dimodifikasi dan ditambahkan yang berguna untuk domain penjualan seperti rekaman kontak, status kontak terakhir, tanggal kontak selanjutnya dan sifat kontak yang disepakati.

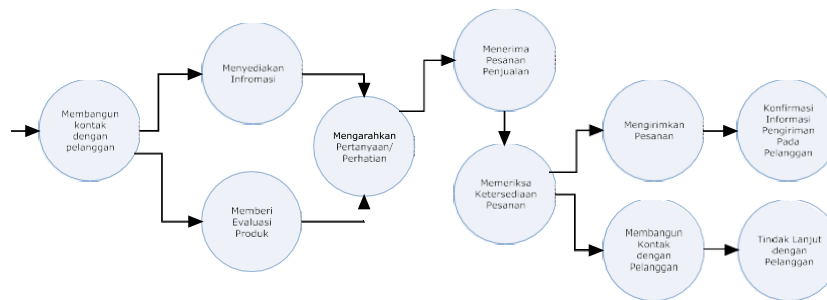
Pemodelan Proses

Aktivitas yang dilakukan pada area bisnis mencakup serangkaian fungsi yang disaring ke dalam beberapa proses bisnis. Untuk menggambarkan urutan proses bisnis ini dapat dimodelkan dengan menggunakan diagram aliran proses. Sebagai contoh pada sub bab 3.2.1.2 dari fungsi penjualan akan disederhanakan menjadi beberapa proses bisnis.

Fungsi penjualan :

- membangun kontak pelanggan
- menyediakan informasi dan literatur yang sesuai
- Mengarahkan pertanyaan dan perhatian
- Memberi evaluasi terhadap produk
- Menerima pesanan penjualan
- Memeriksa ketersediaan pesanan
- Menyiapkan pengiriman pesanan
- Mengkonfirmasi informasi pengiriman seperti penetapan harga, tanggal pengiriman pada pelanggan.
- Mengirim pesan pengiriman ke bagian pengiriman
- Tindak lanjut dengan pelanggan

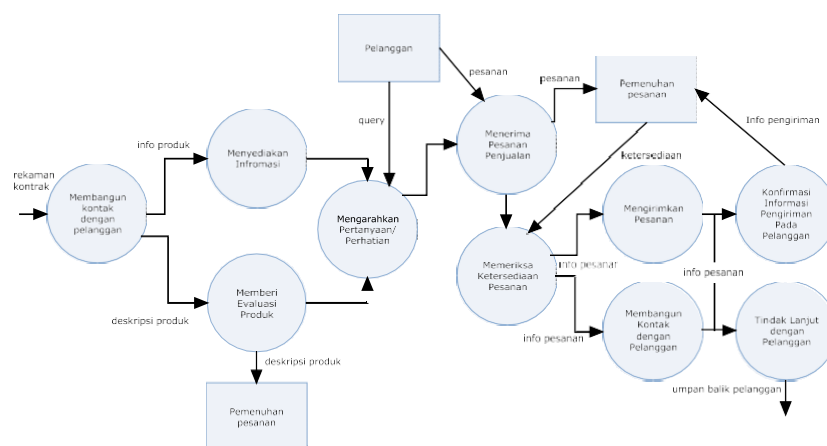
Dari proses bisnis pada fungsi penjualan diatas dapat dimodelkan dengan menggunakan diagram aliran proses seperti pada gambar 3.5.



Gambar 3.5 Diagram aliran proses untuk fungsi penjualan

3.2.2.1 Pemodelan Aliran Informasi

Model aliran proses pada sub bab 3.2.2.1 diintegrasikan dengan model data akan mengidentifikasi bagaimana informasi yang mengalir dalam suatu area bisnis. Objek data masukan dan data keluaran akan diperlihatkan pada masing-masing proses yang menunjukkan transformasi informasi. Informasi ini digunakan untuk menyelesaikan fungsi bisnis. Gambar 3.6 menggambarkan pemodelan aliran informasi dari diagram aliran proses pada fungsi penjualan.



Gambar 3.6 Aliran informasi pada diagram aliran proses untuk fungsi penjualan

Setelah aliran proses dan informasi dimodelkan, maka perekayasa informasi akan menguji apakah proses yang ada akan direkayasa ulang dan apakah sistem informasi atau aplikasi yang ada akan dimodifikasi atau diganti dengan teknologi informasi atau aplikasi yang lebih efisien. Model proses yang telah direvisi akan menjadi dasar untuk spesifikasi perangkat lunak yang baru atau yang telah direvisi untuk mendukung fungsi bisnis.

3.3 Rekayasa Produk

Rekayasa produk disebut juga dengan rekayasa sistem yang merupakan aktivitas pemecahan masalah. Data, fungsi, dan perilaku produk yang diinginkan dicari, dianalisis, dibuat model kebutuhannya, kemudian dialokasikan ke komponen rekayasa. Selanjutnya komponen-komponen ini disatukan dengan infrastruktur pendukungnya sampai produk tersebut jadi. Komponen rekayasa disini seperti perangkat lunak, perangkat keras, data (basisdata) dan manusia. Sedangkan infrastruktur pendukung berupa teknologi yang dibutuhkan untuk menyatukan komponen dan informasi.

Sebagian besar produk dan sistem yang baru masih samar akan fungsi yang dibutuhkan. Oleh karena itu, perekayasa sistem harus membatasi kebutuhan produk dengan mengidentifikasi ruang lingkup fungsi dan kinerja yang diinginkan dari sistem atau produk tersebut.

Dalam rekayasa produk ada beberapa aktivitas yang akan dilakukan untuk mengetahui data, fungsi dan perilaku produk yang diinginkan sebelum pengembangan produk dilakukan diantaranya adalah:

3.3.1 Analisa Sistem

Tujuan dilakukan analisa sistem adalah

- Mengidentifikasi kebutuhan pelanggan.
- Mengevaluasi kelayakan sistem.
- Melakukan analisis teknis dan ekonomis,
- Mengalokasikan fungsi-fungsi untuk perangkat lunak, perangkat keras, basisdata, manusia, dan elemen sistem yang lain.
- Membuat batasan biaya dan jadwal.
- Menentukan definisi sistem yang menjadi dasar kerja bagi komponen sistem baik perangkat lunak, perangkat keras, basisdata dan manusia.

3.3.2 Identifikasi Kebutuhan

Langkah pertama dari aktivitas analisa sistem adalah analisa kebutuhan dengan mengidentifikasi kebutuhan dari pelanggan. Mengidentifikasi kebutuhan ini dilakukan dengan melakukan pertemuan antara seorang analis dengan pelanggan. Seperti halnya rekayasa informasi, tujuan dari pertemuan ini untuk memahami sasaran produk dan menentukan tujuan dibangunnya sebuah produk supaya sasaran tersebut tercapai.

Setelah tujuan ditentukan, analis akan melanjutkan aktivitas evaluasi informasi. Berikut ada beberapa pertanyaan yang dapat digunakan untuk membantu mengevaluasi informasi dari sistem atau produk yang akan dibangun.

- Adakah teknologi untuk membangun sistem?
- Batasan apa saja yang akan dialokasikan terhadap jadwal dan biaya?
- Pengembangan dan sumber daya apa saja yang dibutuhkan?

Jika sistem atau produk yang akan dibangun berupa produk yang akan dijual ke pelanggan, ada beberapa pertanyaan yang bisa diajukan yaitu

- Bagaimana produk tersebut dapat bersaing dengan produk yang telah ada?
- Pasar apa saja yang potensial bagi produk yang akan dibangun?

Setelah semua informasi dikumpulkan pada aktivitas identifikasi kebutuhan. Informasi tersebut akan dispesifikasikan dalam sebuah dokumen konsep sistem.

3.3.3 Studi Kelayakan

Pengembangan sistem atau produk berbasis komputer lebih banyak terganggu dengan kurangnya sumber daya dan waktu penyelesaian dan penyampaian produk. Oleh karena itu, perlu dilakukan lebih awal evaluasi terhadap kelayakan sebuah proyek pengembangan sistem atau produk tersebut. Berikut ada empat studi kelayakan yang dapat dievaluasi.

1. Kelayakan Ekonomis
Studi mengenai evaluasi biaya pengembangan dengan keuntungan yang diperoleh dari sistem atau produk yang dikembangkan.
2. Kelayakan Teknis
Studi mengenai fungsi, sasaran dan kinerja yang perlu dipertimbangkan yang dapat mempengaruhi kemampuan sistem yang akan dikembangkan. Pertimbangan yang dihubungkan dengan kelayakan teknis meliputi
 - Resiko pengembangan
 - Keberadaan sumber daya
 - Teknologi
3. Kelayakan Legal
Studi mengenai pertimbangan yang perlu dilakukan mengenai kontrak, pelanggaran atau liabilitas yang akan dihasilkan dari sistem yang akan dikembangkan.
4. Alternatif
Studi mengenai evaluasi pendekatan alternatif pada pengembangan sistem atau produk.

Hasil dari studi kelayakan akan menentukan proyek dilanjutkan atau dihentikan. Hasil studi kelayakan akan didokumentasikan terpisah dan dilampirkan pada dokumen spesifikasi sistem.

- I. Pendahuluan
 - A. Pernyataan Masalah
 - B. Lingkungan Implementasi
 - C. Larangan-larangan

- II. Ringkasan dan Rekomendasi Manajemen
 - A. Penemuan-penemuan Penting
 - B. Komentar
 - C. Rekomendasi
 - D. Pengaruh
- III. Alternatif-alternatif
 - A. Konfigurasi Sistem Alternatif
 - B. Kriteria yang digunakan dalam pemilihan pendekatan akhir
- IV. Deskripsi Sistem
 - A. Ruang Lingkup Sistem
 - B. Kelayakan Elemen Sistem
- V. Analisis Biaya dan Keuntungan
- VI. Evaluasi Resiko Teknis
- VII. Percabangan Legal
- VIII. Topik-topik Proyek Khusus Lainnya

Gambar 3.7 Outline Dokumen Studi Kelayakan

3.3.4 Analisis Ekonomis

Analisis biaya dan keuntungan merupakan salah satu informasi analisa ekonomis yang paling penting yang diisikan dalam studi kelayakan. Analisis biaya dan keuntungan menggambarkan biaya pengembangan proyek dan membandingkannya dengan keuntungan yang akan diperoleh dari pengembangan sistem.

Analisis biaya dan keuntungan berbeda antara sistem yang satu dengan yang lainnya tergantung dari karakteristik dari sistem yang akan dikembangkan dan ukuran proyek.

Keuntungan yang diperoleh dari pengembangan sistem tidak hanya berupa hal yang bisa diukur. Keuntungan yang tidak bisa diperoleh dari pengembangan proyek seperti kepuasan pelanggan, keputusan bisnis yang lebih baik, kualitas desain yang lebih baik dan lainnya.

Hasil dari analisa ekonomi didokumentasikan menjadi satu dalam dokumen studi kelayakan seperti terlihat pada gambar 3.7

3.3.5 Analisis Teknis

Pada aktivitas analisis teknis, seorang analis melakukan evaluasi secara teknis terhadap sistem serta mengumpulkan informasi mengenai reliabilitas, kinerja, pemeliharaan dan produktifitas dari sistem yang akan dikembangkan.

Analisis meliputi penilaian viabilitas teknis dari sistem seperti teknologi apa yang akan dibutuhkan untuk membangun sistem, materi, metode, algoritma atau proses baru apa yang diperlukan oleh sistem, bagaimana masalah teknologi mempengaruhi sistem dan bagaimana resiko pengembangan sistem.

Pemodelan matematis atau teknik optimasi pada saat analisis teknis dapat digunakan untuk mempermudah analisis dalam menggambarkan sistem yang akan dikembangkan. Menurut Blanchard dan Fabrycky ada beberapa kriteria penggunaan model selama analisis teknis pada sistem, yaitu

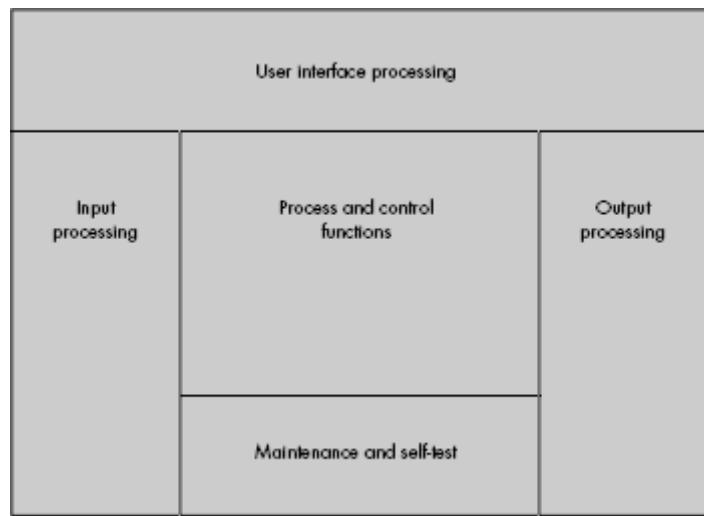
- Model harus mewakili sistem yang sedang dievaluasi dengan cara yang sederhana dan mudah dipahami.
- Model harus menggambarkan faktor-faktor yang relevan dengan masalah yang ada dan hindari faktor yang tidak penting.
- Model harus dibuat komprehensif dengan memasukkan semua faktor yang relevan dan sistem harus mampu memberikan hasil yang sama.
- Desain model harus sederhana supaya memungkinkan pengimplementasian yang tepat waktu dalam pemecahan masalah.
- Desain model harus dapat mengantisipasi faktor-faktor yang memungkinkan adanya modifikasi dan atau perluasan terjadi pada sistem.

Hasil dari analisis teknis merupakan dasar apakah pengembangan sebuah sistem akan diteruskan atau dihentikan.

3.4 Pemodelan Arsitektur Sistem

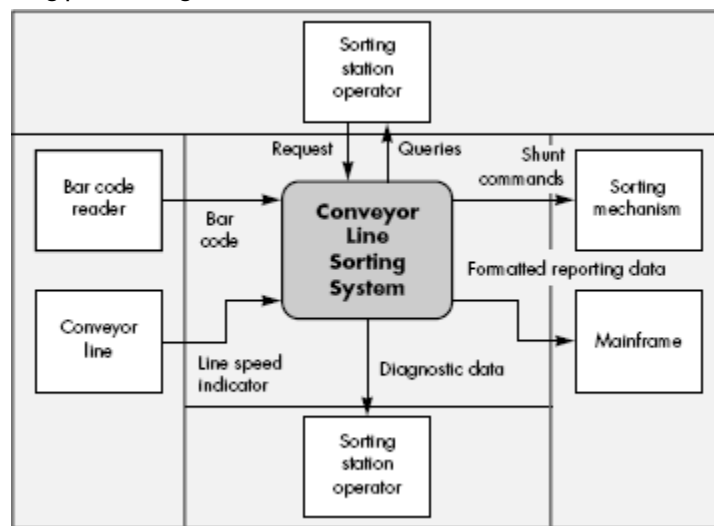
Setiap sistem berbasis komputer dapat dimodelkan sebagai sebuah pemindahan informasi dengan menggunakan arsitektur input-pemrosesan-output. Hartley dan Pirbhai [2] memperluas arsitektur ini dengan menambahkan 2 fitur tambahan yaitu pemrosesan antarmuka pemakai dan pemrosesan self-test. Meskipun dua fitur tambahan ini tidak selalu dipakai tetapi fitur tambahan ini umum dipakai pada pemodelan sistem berbasis komputer yang membuat model sistem menjadi lebih baik. Model sistem ini menjadi dasar bagi analisis kebutuhan dan langkah desain selanjutnya.

Untuk memodelkan sistem maka digunakan model template arsitektur[2] yang mengalokasikan elemen sistem menjadi 5 bagian pemrosesan yaitu 1. antarmuka pemakai, 2.input, 3. fungsi dan kontrol sistem, 4.output dan 5. pemeliharaan dan self-test seperti yang terlihat pada gambar 3.8



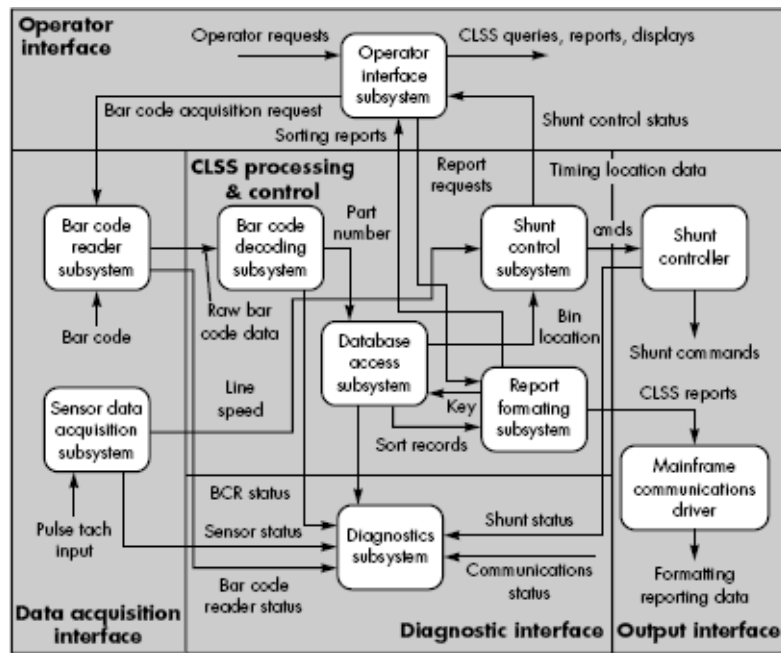
Gambar 3.8 Template arsitektur

Seperti halnya pemodelan pada rekayasa sistem dan perangkat lunak, template arsitektur memungkinkan analisis membuat hierarki sistem secara detail. Diagram konteks arsitektur menggambarkan sistem pada level paling atas. Diagram konteks ini membangun batas informasi diantara sistem yang akan diimplementasikan dengan lingkungan dimana sistem akan dioperasikan [2]. Sebagai contoh pada gambar 3.9 diagram konteks arsitektur untuk sistem pengurutan pembawa barang pada bidang manufaktur.



Gambar 3.9 Diagram konteks arsitektur untuk sistem pengurutan barang pada barang manufaktur

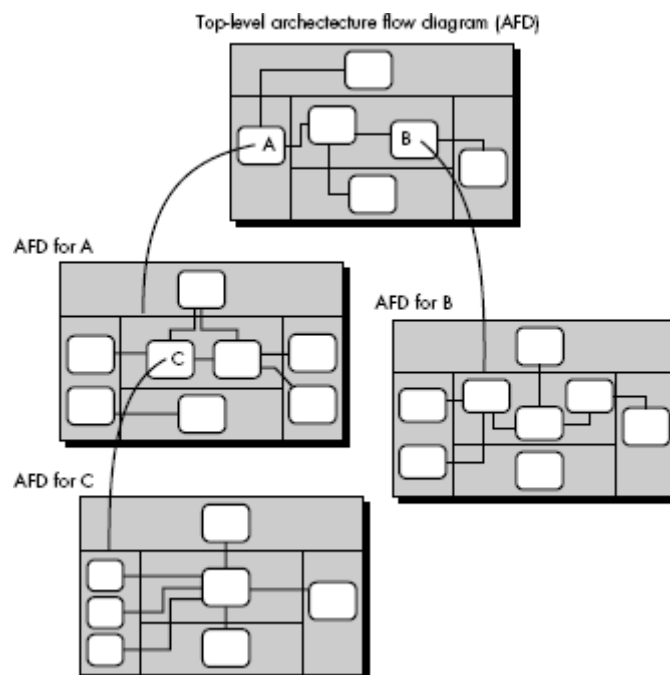
Dari diagram konteks diatas dapat saring kembali menjadi subsistem-subsistem atau modul-modul sistem dengan aliran informasi yang penting antar modul tersebut dengan menggunakan model diagram aliran arsitektur (architecture flow diagram). Diagram aliran arsitektur ini masih membagi pemrosesan sub sistem ke dalam lima elemen pemrosesan seperti diagram konteks arsitektur. Pada tingkat ini, masing-masing dari subsistem dapat berisi satu elemen sistem atau lebih (perangkat keras, perangkat lunak, manusia). Gambar 3.10 Diagram aliran arsitektur untuk sistem pengurutan barang pada gambar 3.9.



Gambar 3.10 Diagram aliran arsitektur untuk sistem pengurutan barang pada barang manufaktur

Diagram aliran arsitektur yang awal menjadi puncak dari hirarki diagram aliran arsitektur. Masing-masing bujursangkar pada diagram aliran arsitektur dapat diperluas atau didetailkan lagi ke dalam template arsitektur yang lain. Proses ini dapat digambarkan seperti pada gambar 3.11.

Deskripsi naratif dari masing-masing subsistem dan definisi semua data yang mengalir pada diagram aliran arsitektur menjadi elemen penting dalam membuat spesifikasi sistem.



Gambar 3.11 Hirarki Diagram Aliran Arsitektur

3.5 Spesifikasi Sistem

Spesifikasi sistem merupakan dokumen yang berfungsi menggambarkan fungsi dan kinerja sistem berbasis komputer yang akan dikembangkan, membatasi elemen-elemen sistem yang telah dialokasikan, serta memberikan indikasi mengenai perangkat lunak dan konteks sistem keseluruhan dan informasi data dan kontrol yang dimasukkan dan dikeluarkan oleh sistem yang telah digambarkan dalam diagram aliran arsitektur. Berikut salah satu format dokumen dari spesifikasi sistem yang bisa anda gunakan.

- I. Pendahuluan
 - A. Lingkup dan Tujuan Dokumen
 - B. Tinjauan
 - 1. Sasaran
 - 2. Batasan
- II. Fungsional dan Deskripsi Data
 - A. Arsitektur Sistem
 - 1. Diagram Konteks Arsitektur
 - 2. Deskripsi Diagram Konteks Arsitektur
- III. Deskripsi Subsistem
 - A. Spesifikasi Diagram Arsitektur untuk n Subsistem
 - 1. Diagram Aliran Arsitektur
 - 2. Penjelasan Modul Sistem
 - 3. Isu-isu Kinerja
 - 4. Batasan-batasan Desain
 - 5. Alokasi Komponen Sistem
 - B. Kamus Arsitektur
 - C. Diagram dan Deskripsi Interlasi Arsitektur
- IV. Hasil Pemodelan dan Simulasi Sistem
 - A. Model Sistem yang Digunakan untuk Simulasi
 - B. Hasil Simulasi
 - C. Isu-isu Kinerja Khusus
- V. Isu-isu Proyek
 - A. Biaya Pengembangan
 - B. Jadwal
- VI. Lampiran

Latihan

- 1. Jelaskan apa yang disebut dengan rekayasa system
- 2. Apa saja yang harus dilakukan sebelum memulai pengembangan produk?
- 3. Jelaskan apa dimaksud dengan rekayasa produk.
- 4. Jelaskan apa yang dimaksud dengan spesifikasi system.
- 5. Jelaskan apa yang dimaksud pemodelan arsitektur system.

4 Analisa Kebutuhan Perangkat Lunak

4.1 Kebutuhan

4.1.1 Apa yang Disebut Kebutuhan?

Menurut Kamus Webster seperti dikutip oleh Davis [DAV93], kebutuhan adalah sesuatu yang disyaratkan; sesuatu yang diinginkan atau diperlukan. Sedangkan menurut IEEE [IEE93] kebutuhan adalah:

- Kondisi atau kemampuan yang diperlukan pemakai untuk menyelesaikan suatu persoalan, atau untuk mencapai tujuan.
- Kondisi atau kemampuan yang harus dimiliki atau dimiliki oleh sistem atau komponen sistem untuk memenuhi kontrak, standar, spesifikasi, atau dokumen formal lainnya.

Dengan mengadopsi pengertian-pengertian di atas, dapat disimpulkan bahwa kebutuhan perangkat lunak adalah kondisi, kriteria, syarat atau kemampuan yang harus dimiliki oleh perangkat lunak untuk memenuhi apa yang disyaratkan atau diinginkan pemakai.

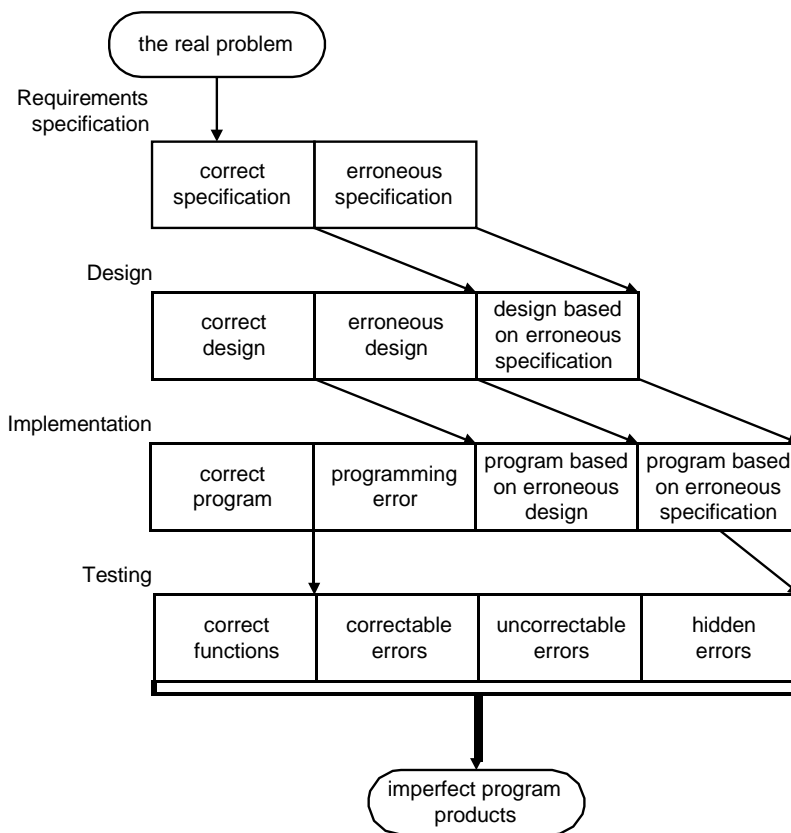
Secara kategoris, ada tiga buah jenis kebutuhan perangkat lunak [IEE93] :

- 1) Kebutuhan fungsional (functional requirement)
Disebut juga kebutuhan operasional, yaitu kebutuhan yang berkaitan dengan fungsi atau proses transformasi yang harus mampu dikerjakan oleh perangkat lunak.
Sebagai contoh:
 - a) Perangkat lunak harus dapat menyimpan semua rincian data pesanan pelanggan.
 - b) Perangkat lunak harus dapat membuat laporan penjualan sesuai dengan periode waktu tertentu.
 - c) Perangkat lunak harus mampu menyajikan informasi jalur pengiriman barang terpendek.
- 2) Kebutuhan antarmuka (interface requirement)
Kebutuhan antarmuka yang menghubungkan perangkat lunak dengan elemen perangkat keras, perangkat lunak, atau basis data.
Sebagai contoh:
 - a) Perangkat untuk memasukkan data dapat berupa keyboard, mouse atau scanner.
 - b) Akses ke basisdata menggunakan ODBC (Open Database Connectivity).
- 3) Kebutuhan unjuk kerja (performance requirement)
Kebutuhan yang menetapkan karakteristik unjuk kerja yang harus dimiliki oleh perangkat lunak, misalnya: kecepatan, ketepatan, frekuensi.
Sebagai contoh:
 - a) Perangkat lunak harus bisa mengolah data sampai 1 juta record untuk tiap transaksi.
 - b) Perangkat lunak harus dapat digunakan oleh multiuser sesuai dengan otoritas yang diberikan pada user.
 - c) Waktu tanggap penyajian informasi maksimal selama satu menit.

4.1.2 Mengapa Kebutuhan Penting?

Pendefinisian kebutuhan merupakan aktivitas yang sangat penting, karena sangat mempengaruhi sukses atau gagalnya pelaksanaan pengembangan perangkat lunak. Menurut hasil survey DeMarco, 56% kegagalan proyek pengembangan perangkat lunak dikarenakan ketidaklengkapan pendefinisian kebutuhan dari perangkat lunak tersebut. Perhatikan gambar dampak kesalahan kumulatif akibat kesalahan dalam pendefinisian kebutuhan pada Gambar 4.1.

Dari gambar terlihat bahwa produk perangkat lunak yang tidak sempurna akan dihasilkan karena kesalahan pada saat menentukan spesifikasi kebutuhan. Jika kesalahan tersebut diketahui di akhir siklus hidup pengembangan, usaha untuk memperbaikinya akan sangat mahal.



Gambar 5.1. Dampak Kesalahan Kumulatif

Selain itu, kesalahan penentuan kebutuhan akan memberikan dampak [DAV93]:

- 1) Perangkat lunak yang dihasilkan tidak akan memenuhi kebutuhan pemakai yang sebenarnya.
- 2) Interpretasi kebutuhan yang berbeda-beda sehingga dapat menyebabkan ketidaksepakatan antara pelanggan dan pengembang, menyia-nyiakan waktu dan biaya, dan mungkin akan menghasilkan perkara hukum.
- 3) Pengujian kesesuaian perangkat lunak dengan kebutuhan yang dimaksud tidak akan mungkin dilaksanakan dengan sesungguhnya.
- 4) Waktu dan biaya akan terbuang percuma untuk membangun sistem yang salah.

4.2 Analisis Kebutuhan

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 tahap rekayasa sistem/informasi dan software project planning.

4.2.1 Pengertian

Dengan mengadopsi pengertian tentang kebutuhan pada sub bab sebelumnya, analisis kebutuhan dapat diartikan sebagai berikut :

- 1) Proses mempelajari kebutuhan pemakai untuk mendapatkan definisi kebutuhan sistem atau perangkat lunak [IEE93].
- 2) Proses untuk menetapkan fungsi dan unjuk kerja perangkat lunak, menyatakan antarmuka perangkat lunak dengan elemen-elemen sistem lain, dan menentukan kendala yang harus dihadapi perangkat lunak [PRE01].

Tujuan pelaksanaan analisis kebutuhan adalah

- 1) Memahami masalah secara menyeluruh (komprehensif) yang ada pada perangkat lunak yang akan dikembangkan seperti ruang lingkup produk perangkat lunak (product space) dan pemakai yang akan menggunakannya.
- 2) Mendefinisikan apa yang harus dikerjakan oleh perangkat lunak untuk memenuhi keinginan pelanggan.

4.2.2 Tahapan Analisis Kebutuhan

Secara teknis pelaksanaan pekerjaan analisis kebutuhan perangkat lunak pada dasarnya terdiri dari urutan aktivitas:

- 1) Mempelajari dan memahami persoalan

Pada tahap ini, seorang analis mempelajari masalah yang ada pada perangkat lunak yang dikembangkan, sehingga dapat ditentukan

- a) siapa pemakai yang menggunakan perangkat lunak.
- b) dimana perangkat lunak akan digunakan .
- c) pekerjaan apa saja dari pemakai yang akan dibantu oleh perangkat lunak.
- d) apa saja cakupan dari pekerjaan tersebut, dan bagaimana mekanisme pelaksanaannya.
- e) apa yang menjadi kendala dilihat dari sisi teknologi yang digunakan atau dari sisi hukum dan standar.

Cara yang digunakan oleh pengembang khususnya analis dalam memahami masalah perangkat lunak biasanya dilakukan

- a) wawancara dengan pemakai
- b) observasi atau pengamatan lapangan
- c) kuesioner
- d) mempelajari referensi atau dokumen-dokumen yang digunakan, seperti dokumen hasil analisa dan perancangan perangkat lunak.

Hasil dari pemahaman masalah tersebut dapat digambarkan dengan model-model tertentu sesuai dengan jenis permasalahannya. Sebagai contoh jika masalah bisnis dapat digambarkan dengan flowmap atau bussiness use case untuk analisa berorientasi objek. Sedangkan untuk masalah matematika dapat digambarkan dengan graf.

- 2) Mengidentifikasi kebutuhan pemakai

Pada tahap identifikasi kebutuhan pemakai (user requirement) in pada prakteknya menjadi satu pelaksanaannya dengan pemahaman masalah. Hanya saja substansi yang ditanyakan ada sedikit perbedaan, yaitu

- a) fungsi apa yang diinginkan pada perangkat lunak.
- b) data atau informasi apa saja yang akan diproses.
- c) kelakuan sistem apa yang diharapkan.
- d) antarmuka apa yang tersedia (software interfaces, hardware interfaces, user interfaces, dan communication interfaces)

Untuk menangkap kebutuhan dari pemakai dengan baik, terutama kesamaan persepsi. seorang analis membutuhkan

- a) komunikasi dan brainstorming yang intensif dengan pelanggan.
- b) pembuatan prototype perangkat lunak atau screenshoot.
- c) Data atau dokumen yang lengkap.

- 3) Mendefinisikan kebutuhan perangkat lunak

Saat melakukan pengidentifikasian kebutuhan pemakai, informasi yang diperoleh masih belum terstruktur. Biasanya pemakai akan mengungkapkan apa yang diinginkan dengan bahasa sehari-hari yang biasa mereka gunakan. Sebagai contoh, ungkapan kebutuhan pemakai dibagian akutansi.

- a) saya ingin data yang dimasukkan oleh bagian penjualan bisa langsung di jurnal.
- b) Informasi neraca keuangan bisa saya lihat kapan saja.

Kemudian pada tahap ini, kebutuhan pemakai yang belum terstruktur tersebut akan akan dianalisis, diklasifikasikan, dan diterjemahkan menjadi kebutuhan fungsional, antarmuka dan unjuk kerja perangkat lunak. Sebagai contoh, kebutuhan “data yang dimasukkan oleh bagian penjualan bisa langsung di jurnal” setelah dianalisis, diklasifikasikan dan diterjemahkan, mungki akan menghasilkan pendefinisian kebutuhan sebagai berikut.

- a) Kebutuhan fungsional
 - Entri dan rekam data transaksi penjualan.
 - Retrieve data transaksi penjualan untuk periode tertentu (periode sesuai dengan inputan periode yang diinputkan pada keyboard).
 - Rekam data akumulasi transaksi penjualan periode tertentu ke jurnal umum berikut account pasangannya (kas).

- b) Kebutuhan antarmuka
 - Antarmuka pemakai untuk memasukkan dan merekam data penjualan.
 - Antarmuka pemakai untuk menyajikan dan menjurnal informasi transaksi penjualan pada periode tertentu.
 - Antarmuka untuk jaringan lokal yang menghubungkan perangkat lunak aplikasi dibagian penjualan dengan perangkat lunak aplikasi dibagian akuntansi.
- c) Kebutuhan unjuk kerja
 - proses jurnal hanya bisa dilakukan sekali setelah data transaksi penjualan direkam.
 - Adanya otoritas pemakaian perangkat lunak dan akses data sesuai dengan bagian pekerjaan masing-masing.

Kemudian kebutuhan tersebut akan dimodelkan atau digambarkan dengan teknik analisis dan alat bantu tertentu. Sebagai contoh kebutuhan fungsional dapat dimodelkan dengan menggunakan

- Data flow diagram, kamus data, dan spesifikasi proses jika menggunakan analisis terstruktur
- Use case diagram dan skenario sistem jika menggunakan analisis berorientasi objek.

Metode analisis terstruktur tersebut akan dibahas secara detail pada bab ini dan bab.16 untuk metode analisis berorientasi objek.

- 4) Membuat dokumen spesifikasi kebutuhan perangkat lunak (SKPL)
Semua kebutuhan yang telah didefinisikan selanjutnya dibuat dokumentasinya yaitu Spesifikasi Kebutuhan Perangkat Lunak (SKPL) atau Software Requirement Specification (SRS). Dokumen ini dibuat untuk menyatakan secara lengkap apa yang dapat dilakukan oleh perangkat lunak, termasuk deskripsi lengkap semua antarmuka yang akan digunakan. Penjelasan rinci tentang SKPL ini akan dibahas pada sub bab 5.3.
- 5) Mengkaji ulang (review) kebutuhan
Proses untuk mengkaji ulang (validasi) kebutuhan apakah SKPL sudah konsisten, lengkap, dan sesuai dengan yang diinginkan oleh pemakai. Proses ini bisa dilakukan lebih dari satu kali. Dan sering kali akan muncul kebutuhan-kebutuhan baru dari pemakai. Oleh karena itu, diperlukannya negosiasi antara pengembang dengan pelanggan sesuai dengan prinsip “win win solution” sampai kebutuhan tersebut disetujui oleh kedua belah pihak.

Sedangkan menurut Pressman [PRE01], analisis kebutuhan perangkat lunak dapat dibagi menjadi lima area pekerjaan, yaitu:

- a) Pengenalan masalah
- b) Evaluasi dan sintesis
- c) Pemodelan
- d) Spesifikasi
- e) Tinjau ulang (review)

4.2.3 Metode Analisis

Metode atau teknik untuk melakukan analisis kebutuhan perangkat lunak dapat 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) 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 [DEM76], Chris Gane dan Trish Sarson, dan Edward Yourdon [YOU89]. Pada metode ini, hasil analisis dan perancangan dimodelkan dengan menggunakan beberapa perangkat pemodelan seperti:

- Data Flow Diagram (DFD) dan Kamus Data (data dictionary) untuk menggambarkan fungsi-fungsi dari sistem (system functions).
- Entity-Relationship Diagram (ERD) untuk menggambarkan data yang disimpan (data stored).
- State Transition Diagram (STD) untuk menggambarkan perilaku sistem.
- Structure Chart untuk menggambarkan struktur program.

2) Berorientasi Struktur Data (Data Structured Oriented)

Analisis dengan 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:

a) 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 perancangannya.

b) Jackson System Development (JSD)

Dikembangkan oleh M.A. Jackson [1975] dengan menggunakan perangkat pemodelan yang disebut structure diagram dan system specification diagram.

3) Berorientasi Objek (Object Oriented)

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 “dinkapsulasi” (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 Modeling Technique (OMT) dari James Rumbaugh (1987).
- Object Oriented Software Engineering (OOSE).

4.3 Spesifikasi Kebutuhan Perangkat Lunak (SKPL)

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 SKPL harus mencantumkan tentang deskripsi lengkap dari semua antarmuka yang ada dalam sistem yang dapat menghubungkan sistem dengan lingkungannya, mencakup antarmuka untuk perangkat keras, perangkat lunak, komunikasi dan pemakai.

SKPL bisa terdiri dari banyak dokumentasi yang saling melengkapi. Suatu SKPL harus dapat menguraikan definisi masalah, dan menguraikan masalah dengan tepat dengan cara yang tepat pula.

4.3.1 Tujuan Pembuatan SKPL

Ada beberapa tujuan pembuatan SKPL, dan itu tergantung kepada siapa yang menulisnya. Pertama, SKPL dapat ditulis oleh pemakai potensial (pelanggan) dari sistem, dan kedua oleh pengembang sistem.

Untuk kasus pertama, tujuan penulisan SKPL adalah untuk mendefinisikan keinginan yang biasanya dinyatakan dalam bentuk penjelasan umum. Untuk yang kedua, tujuan pembuatan SKPL adalah:

- Sarana komunikasi antara pelanggan, pemakai, analis, dan perancang perangkat lunak.
- Dasar untuk merencanakan dan melaksanakan aktivitas pengujian sistem.
- Acuan untuk melakukan perbaikan dan perubahan perangkat lunak.

Sedangkan manfaat dan kegunaan SKPL menurut Witarto[WIT04] dari IEEE, adalah

- Memastikan kesamaan antara kebutuhan untuk pengembangan dengan kebutuhan yang ditulis didalam dokumen.
- Mendefinisikan kerangka kerja bersama untuk proses-proses pengembangan perangkat lunak.
- Memperjelas peran dan antarmuka bagi para pihak yang terlibat dalam proses pengembangan perangkat lunak.
- Memperjelas jenis dan isi dokumen.
- Mengenali tugas, tahapan, baseline, aktivitas kaji ulang, dan dokumentasinya.
- Belajar pendekatan praktis yang diterapkan di dunia industri.
- Menghilangkan persoalan-persoalan seperti yang pernah dialami masa lalu.

4.3.2 Syarat Pembentukan SKPL

Ada empat syarat yang harus diperhatikan saat pembentukan SKPL, yaitu:

- 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 SKPL:

- 1) Over specification (penjelasan berlebih dan berulang-ulang sehingga menjadi tidak jelas)
- 2) Tindakan unconsistency (seperti menggunakan istilah yang tidak konsisten)
- 3) Ambiguity dalam kata atau kalimat seperti menyatakan keterukuran kebutuhan secara tidak jelas misalkan menggunakan kata-kata :minimal, maksimal, optimal, cepat, user friendly, efisien, fleksible dan lainnya.
- 4) Menuliskan “mimpi-mimpi”, yaitu hal-hal yang tidak bisa dilakukan

Dalam suatu SKPL ada 2 aspek yang harus bisa dilihat:

- 1) Fungsi
Menjelaskan fungsi dari perangkat lunak (digunakan untuk apa keperluan apa), sifat perangkat lunak, dan datanya.
- 2) Non-fungsi
 - Dependability
 - Ergonomic
 - Performance
 - Constraint

4.3.3 Atribut Penulisan SKPL yang Baik

Dokumen SKPL yang baik (sempurna) akan ditulis secara:

- 1) Benar (correct)
Suatu dokumen SKPL disebut benar jika dan hanya jika setiap kebutuhan yang dinyatakan dalam dokumen merepresentasikan sesuatu yang disyaratkan dari sistem yang akan diangun.
- 2) Tepat (precise)
Berpengaruh pada hasil perancangan dan pembuatan software requirements design (SRD).
- 3) Unambiguouity
Setiap permintaan harus punya satu intepretasi, atau hanya ada satu arti dalam satu kalimat.
- 4) Lengkap (complete)
Lengkap jika dilihat dari dua sudut pandang:
 - 1) dokumen memuat tabel isi, nomor halaman, nomor gambar, nomor tabel, dan sebagainya.
 - 2) 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 spesifikasi, misalnya diminta A tetap ditulis A.
- 7) Understandable
Dapat dimengerti oleh pemrogram, analis sistem atau system 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 menyelesaikan 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 ketidakkonsistenan.
- 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 SKPL:

- 1) Pemakai (user)
Kelompok orang yang mengoperasikan/menggunakan produk final dari perangkat lunak yang dibuat.
- 2) Client
Orang atau perusahaan yang mau membuat sistem (yang menentukan).
- 3) System analyst (system engineer)
Kelompok orang yang biasa melakukan kontak teknik pertama dengan client. Bertugas menganalisis persoalan, menerima requirement dan menulis requirement.
- 4) Software engineer
Kelompok orang yang bekerja setelah kebutuhan perangkat lunak dibuat (bekerja sama dengan system engineer saat mendefinisikan kebutuhan perangkat lunak dan membuat deskripsi perancangannya).
- 5) Programmer
Kelompok orang yang menerima spesifikasi perancangan perangkat lunak, membuat kode dalam bentuk modul, menguji dan memeriksa (tes) modul.
- 6) Test integration group
Kelompok orang yang melakukan tes dan mengintegrasikan modul.
- 7) Maintenance group
Kelompok orang yang 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
Kelompok orang yang bertugas mengetik, memasukkan data, membuat dokumen.

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

- 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 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)

4.3.4 Tata Letak Dokumen SKPL

Tata letak atau format dokumen SKPL baku menurut ANSI/IEEE std 830 I 984 adalah:

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. SPESIFIKASI 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 Performansi
 - 3.4 Kendala Disain
 - 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

4.4 Analisis Terstruktur

Pada sub bab sebelumnya telah dibahas tentang analisis kebutuhan yang merupakan tahap dari pengembangan perangkat lunak untuk mendefinisikan kebutuhan perangkat lunak. Untuk dapat mengerjakan tahap ini, dibutuhkan serangkaian metode teknis yang dilaksanakan menurut sudut pandang atau pendekatan tertentu. Salah satu metode teknis untuk melaksanakan analisis kebutuhan tersebut adalah Analisis Terstruktur.

4.4.1 Pengertian

Analisis Terstruktur (Structured Analysis) merupakan salah satu teknik analisis yang menggunakan pendekatan berorientasi fungsi. Teknik ini mempunyai sekumpulan petunjuk dan perangkat komunikasi grafis yang memungkinkan analis sistem mendefinisikan spesifikasi fungsional perangkat lunak secara terstruktur. Pada metode ini, semua fungsi sistem direpresentasikan sebagai sebuah proses transformasi informasi, dan disusun secara hirarkis sesuai tingkat abstraksinya (sistem maupun perangkat lunak) yang hasilnya ditujukan untuk entitas-entitas eksternal.

Analisis Terstruktur pertama kali diperkenalkan oleh Tom DeMarco sekitar tahun 1978 [DEM79]. Prinsip dari teknik ini adalah dekomposisi fungsi dari sistem berdasarkan aliran data dan proses-prosesnya untuk mendapatkan produk analisis yang dapat diubah dan diperbaiki secara mudah (highly maintainable). Dalam bukunya itu, DeMarco mendefinisikan Analisis Terstruktur sebagai teknik untuk mendeskripsikan spesifikasi sistem baru melalui Data Flow Diagrams, Data Dictionary, Structured English, dan Data Structure Diagrams. Spesifikasi sistem tersebut dinyatakan dalam suatu dokumen yang disebut Spesifikasi Terstruktur (Structured Specification).

Dalam perkembangannya, teknik Analisis Terstruktur mengalami perubahan, penambahan, dan penyempurnaan, baik untuk perangkat pemodelannya maupun mekanisme atau cara pelaksanaannya. Salah satunya oleh Edward Yourdon [YOU89] yang memperkenalkan pendekatan baru dari Analisis Terstruktur, yaitu Analisis Terstruktur Modern (Modern Structures Analysis).

4.4.2 Perangkat Pemodelan Analisis Terstruktur

Perangkat Pemodelan Analisis Terstruktur adalah alat bantu pemodelan yang digunakan untuk menggambarkan hasil pelaksanaan Analisis Terstruktur. Perangkat Analisis Terstruktur yang disampaikan oleh DeMarco [DEM78] adalah:

- Diagram Aliran Data atau Data Flow Diagram (DFD)
- Kamus Data atau Data Dictionary
- Structured English
- Tabel Keputusan atau Decision Table
- Pohon Keputusan atau Decision Tree

Kelima perangkat tersebut oleh Yourdon [YOU89] dilengkapi dengan:

- Diagram Entitas-Relasi atau Entity-Relationship Diagram (ERD)
- Diagram Transisi Keadaan atau State Transition Diagram (STD)

Dan sebagai pengembangan untuk menggambarkan sistem waktu nyata, disertakan Diagram Aliran Kendali atau Control Flow Diagram (CFD). Berikut adalah penjelasan rinci untuk masing-masing perangkat, khususnya untuk DFD, Kamus Data, dan Structured English.

4.4.2.1 Diagram Aliran Data (Data Flow Diagram)

Pengertian

- 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 antarmuka diantara komponen-komponen tersebut.
- Perangkat pemodelan yang dapat menggambarkan sistem sebagai sebuah jaringan proses-proses fungsional yang satu dengan 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.
- Sistem yang dimaksud disini adalah sistem perangkat lunak, sistem informasi, sistem perangkat keras, atau sistem berbasis komputer lainnya. (Dalam buku ini, sistem tersebut akan diartikan sebagai sistem perangkat lunak).
- Diperkenalkan oleh Tom DeMarco (1978) dan Chris Gane dan Trish Sarson (1977) berdasarkan notasi SADT (Structure Analysis and Design Technique).
- Merupakan salah satu teknik yang cukup penting dalam menganalisis sistem karena:
 - o Dapat mendefinisikan batasan sistem dan proses-proses pengolahan data yang ada didalamnya.
 - o Membantu memeriksa kebenaran dan kelengkapan aliran informasi.
 - o Merupakan dasar perancangan dengan memunculkan proses-proses pengolahan data.
 - o Dapat digunakan untuk menggambarkan aktivitas proses secara paralel (beberapa aliran data dapat terjadi secara simultan).

Elemen-elemen DFD

Ada empat elemen yang membentuk suatu Data Flow Diagram, yaitu aliran data, proses, penyimpanan data dan sumber/tujuan data.

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.

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.

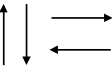
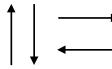
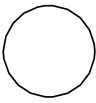
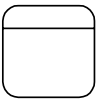


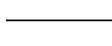
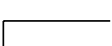
3) Penyimpanan Data (Data Store)

- 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) Sumber/Tujuan Data

- Menggambarkan entitas yang berinteraksi dengan sistem yang berada di luar 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).
- Disebut juga dengan nama entitas eksternal, terminator, source atau sink.

Berikut adalah tabel yang menunjukkan notasi yang digunakan dalam DFD.

DEMARCO/YOURDON	GALE & SARSON
	
aliran data	aliran data
	
proses	proses
	
entitas eksternal	entitas eksternal
	
data store	data store

Tabel 5.1 Notasi Data Flow Diagram

Penggambaran DFD

Ada dua pendekatan penggambaran atau pembuatan DFD yaitu pendekatan fisis dan logis.

1) Pendekatan Fisis

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

2) Pendekatan Logis

- Menggambarkan proses atau fungsi transformasi data yang ada dalam sistem (bukan apa atau siapa yang mengerjakannya).
- Dapat dibuat dari DFD fisis dengan cara mentranslasikannya menjadi deskripsi logis yang difokuskan pada data dan proses (jangan melihat siapa yang melakukan pekerjaan tersebut).
- Beberapa hal yang harus diperhatikan saat menggambarkan DFD logika:
 - i) Perhatikan data aktual, bukan dokumen, yang berhubungan dengan proses.
 - ii) Hilangkan aliran informasi melalui orang/unit organisasi/kantor, munculkan prosedur atau prosesnya saja.
 - iii) Hilangkan proses yang tidak penting, yang tidak mengubah data/aliran data, misalnya proses menyalin (copy) data.
 - iv) Hilangkan fungsi alat bantu atau peralatan-peralatan lainnya.
 - v) Konsolidasikan kerangkapan penyimpanan data.
- Dibuat hanya untuk menggambarkan proses yang akan dikerjakan oleh komputer, bukan proses yang sifatnya fisik atau manual.

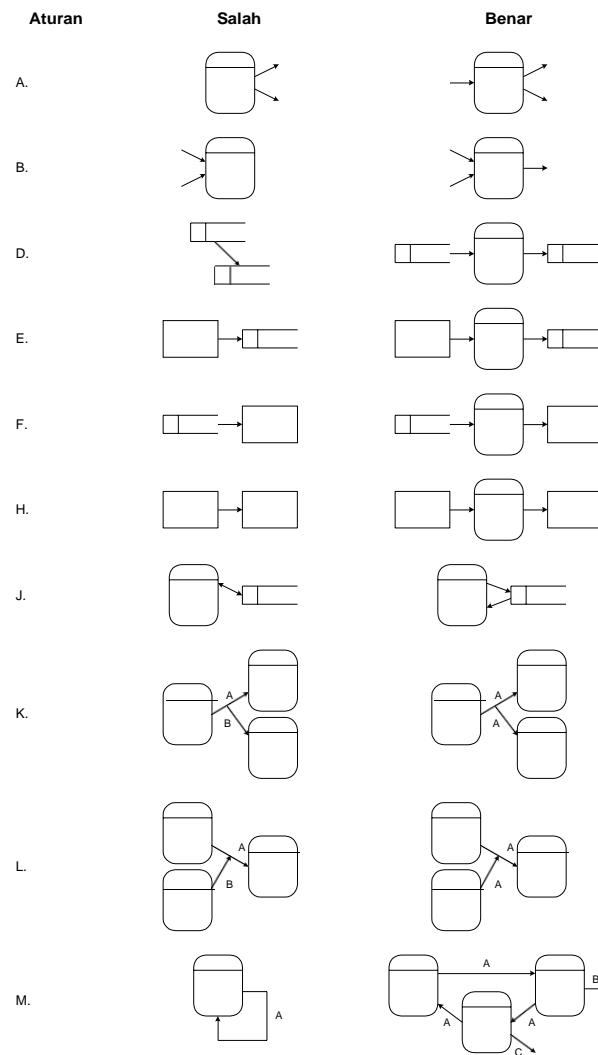
Aturan Penggambaran DFD

Beberapa ketentuan saat menggambar DFD ditunjukkan oleh tabel dan gambar berikut ini.

<p>Proses :</p> <p>A. Tidak ada proses yang hanya mempunyai data keluaran saja. Jika objek hanya mempunyai data keluaran, maka objek tersebut adalah sumber data.</p> <p>B. Tidak boleh hanya mempunyai data masukan saja (black hole). Jika sebuah objek hanya mempunyai data masukan saja, maka objek tersebut adalah tujuan data.</p> <p>C. Nama proses harus menggunakan kata kerja (misal cetak laporan) atau nama yang dibendakan (misal pencetakan laporan).</p> <p>Data Store:</p> <p>D. Data tidak dapat mengalir langsung dari satu data store ke data store yang lain. Data harus berpindah melalui proses terlebih dahulu.</p> <p>E. Data tidak dapat mengalir langsung dari sumber data ke data store. Data harus berpindah melalui proses terlebih dahulu.</p> <p>F. Data tidak dapat mengalir langsung ke data store ke tujuan data, data harus melalui proses terlebih dahulu.</p> <p>G. Nama data store harus menggunakan kata benda (misal barang).</p> <p>Sumber / Tujuan Data (Entitas Eksternal)</p> <p>H. Data tidak dapat mengalir secara langsung dari sumber data ke tujuan data. Data harus melalui proses terlebih dahulu. Jikapun ada, hal tersebut tidak digambarkan dalam DFD.</p> <p>I. Nama sumber/tujuan data harus menggunakan kata benda (misal operator)</p>	<p>Aliran Data :</p> <p>J. Aliran data hanya boleh memiliki satu arah aliran data antara simbol yang satu dengan yang lainnya. Aliran dua arah bisa dimiliki antara proses dengan data store yang menunjukkan pembacaan data sebelum data diupdate, yang diidentifikasi dengan dua arah yang terpisah yang terjadi pada waktu yang berbeda atau penggabungan anak panahnya tidak boleh ganda.</p> <p>K. Aliran data yang sama yang menuju beberapa proses, data store atau sumber/tujuan data berbeda boleh digambarkan bercabang.</p> <p>L. Aliran data yang sama dari beberapa proses, data store atau sumber/tujuan data yang menuju satu proses tertentu boleh digambarkan bercabang.</p> <p>M. Aliran data tidak boleh secara langsung mengalir ke dirinya sendiri (sirkuler). Aliran data tersebut harus diproses minimal satu atau lebih proses yang akan menghasilkan beberapa aliran data yang lain dan kembali ke aliran data yang asli ke proses yang awal.</p> <p>N. Aliran data ke data store maksudnya mengupdate data baik berupa penghapusan data maupun perubahan data</p> <p>O. Aliran data dari data store maksudnya proses mengambil atau membacadata dalam data store tersebut.</p> <p>P. Nama aliran data menggunakan kata benda. Beberapa aliran data dapat digunakan untuk satu anak panah asalkan kesemua nama data tersebut merupakan satu kesatuan paket data.</p>
---	--

Tabel 5.2 Aturan Penggambaran DFD

Untuk lebih jelasnya tentang aturan-aturan penggambaran DFD diatas, dapat dilihat detail simbol-simbol penggambaran DFD sesuai dengan tabel aturan penggambaran DFD diatas.



Gambar 5.2 Penggambaran DFD yang Benar dan Salah

4.4.2.2 Kamus Data (Data Dictionary)

Merupakan suatu tempat penyimpanan (gudang) dari data dan informasi yang dibutuhkan oleh suatu sistem informasi. Kamus data digunakan untuk mendeskripsikan rincian dari aliran data atau informasi yang mengalir dalam sistem, elemen-elemen data, file maupun basis data (tempat penyimpanan) dalam DFD.

Ada aturan (konvensi) penulisannya dengan menggunakan notasi atau simbol tertentu sebagai berikut:

=	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). Sebagai contoh dibawah ini bentuk kamus data yang lengkap (long form):

Id. Barang	=	Kode_Brg + Nama_Brg + Satuan + Hrg_Beli + Hrg_Jual + Banyak
Kode_Brg	=	I{character}6
Nama_Brg	=	I{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]

```

Artinya:

- Identitas Barang tersusun dari atribut Kode_Brg dan Nama_Brg dan Satuan dan Hrg_Beli dan Hrg_Jual dan Banyak.
- Kode_Brg tersusun dari minimal 4 karakter dan maksimal 6 karakter.
- Nama_Brg tersusun dari minimal 8 karakter dan maksimal 20 karakter.
- Satuan tersusun dari 3 karakter.
- Hrg_Jual tersusun dari minimal 3 digit numerik dan maksimal 10 digit numerik
- Jml_Stok tersusun dari 1 digit numerik dan maksimal 6 digit numerik.
- Character terdiri dari huruf besar A sampai Z, atau huruf kecil a sampai z atau angka 0 sampai 9, atau karakter -, atau karakter spasi.
- Numeric terdiri dari angka 0 sampai 9.

Sedangkan contoh bentuk ringkas (short form) dari kamus adalah

Identitas Barang = Kode_Brg + Nama_Brg + Satuan + Hrg_Jual + Jml_Stok

4.4.2.3 Spesifikasi Proses (Process Specification)

Digunakan untuk menggambarkan deskripsi dan spesifikasi dari setiap proses yang paling rendah (proses atomik) yang ada pada sistem dengan 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.

Seperti halnya notasi-notasi yang lain, ada cukup banyak variasi penulisan spesifikasi proses dengan Structured English ini. Pada buku ini akan digunakan notasi penulisan yang menggunakan kata-kata bahasa Indonesia, kecuali untuk kata-kata yang sering digunakan dalam penulisan program, misalnya Read, Write, If, While, atau Repeat.

Ada tiga struktur dasar yang dapat digunakan untuk menyusun spesifikasi proses, yaitu struktur sekuensi, pemilihan dan pengulangan. Berikut adalah contoh penulisan spesifikasi proses untuk proses pembuatan laporan penjualan.

```

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

```

4.4.2.4 Tabel Keputusan (Decision Table)

Tabel keputusan atau decision table adalah tabel yang memuat alternatif tindakan-tindakan apa saja yang dapat diambil berdasarkan kondisi atau variabel tertentu sebagai masukannya. Sebagai contoh, misal akan ditentukan status kelulusan untuk peserta matakuliah tertentu berdasarkan nilai teori dan praktikumnya:

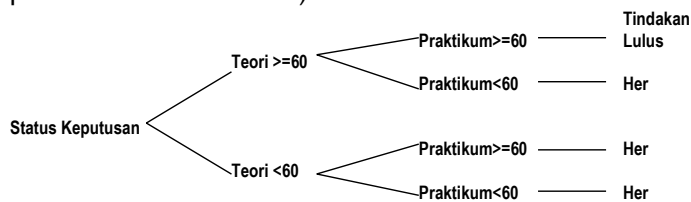
Kondisi	Aturan			
	1	2	3	4
Nilai teori > 60	Y	Y	T	T
Nilai praktikum > 60	Y	T	Y	T

Tabel diatas dibaca sebagai berikut :

- Jika nilai teori dan nilai praktikum > 60 (aturan 1), maka lulus (tindakan 1).
- Jika salah satu nilai teori atau nilai praktikum > 60 (aturan 2 dan 3), maka her (tindakan 2)
- Jika nilai teori dan nilai praktikum < 60 (aturan 4), maka gagal (tindakan 4)

4.4.2.5 Pohon Keputusan (Decision Tree)

Pohon keputusan atau decision tree adalah representasi grafis dari tabel keputusan. Sebagai contoh (untuk masalah penentuan status kelulusan):



Gambar 5.3 Contoh Pohon Keputusan Penentuan Status Kelulusan

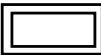

4.4.2.6 Diagram Entitas-Relasi (Entity Relationship Diagram)

Diagram Entitas-Relasi atau Entity Relationship Diagram (ERD) adalah diagram yang menggambarkan keterhubungan antar data secara konseptual. Penggambaran keterhubungan antar data ini didasarkan pada anggapan bahwa dunia nyata terdiri dari kumpulan objek yang disebut entitas (entity), dan hubungan yang terjadi diantaranya yang disebut relasi (relationship).






Ada dua versi penggambaran ERD, yaitu versi Peter P. Chen (berikut variasi dan pengembangan-pengembangannya) dan versi James Martin. Notasi ERD yang digunakan oleh Peter P. Chen maupun James Martin ditunjukkan oleh Gambar 5.4 dan Gambar 5.5.

Simbol	Arti
	Entitas
	Garis Penghubung
	Relasi
	Atribut

Gambar 5.4 (a) Notasi ERD versi Peter P. Chen

Simbol	Arti
	Entitas Lemah
	Generalisasi



Gambar 5.4 (b) Pengembangan Notasi ERD versi Peter P. Chen

Simbol	Arti
	Entitas
	Asosiasi ke satu
	Asosiasi ke banyak
	Asosiasi ke nol atau satu
	Asosiasi ke nol atau banyak

Gambar 5.5 Notasi ERD versi James Martin

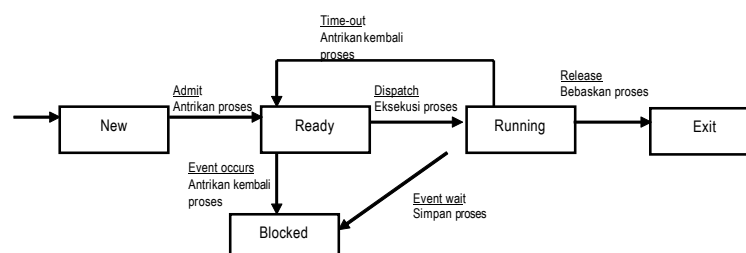
4.4.2.7 Diagram Transisi Keadaan

Diagram transaksi keadaan atau state transition diagram (STD) adalah diagram yang digunakan untuk menggambarkan keadaan keadaan yang menjadi perilaku sistem berikut perubahan atau transisinya. Notasi STD yang umum digunakan dapat dilihat pada Gambar 5.6.

Simbol	Arti
	Keadaan
	Transisi/Perubahan keadaan

Gambar 5.6 Notasi STD







STD biasanya digunakan untuk menggambarkan keadaan dari perangkat lunak sistem waktu nyata, atau perangkat lunak non-bisnis lainnya. Gambar 5.7 berikut memperlihatkan contoh penggambaran STD untuk keadaan proses pada suatu sistem operasi.



Gambar 5.7 Contoh Penggambaran STD untuk Keadaan Proses

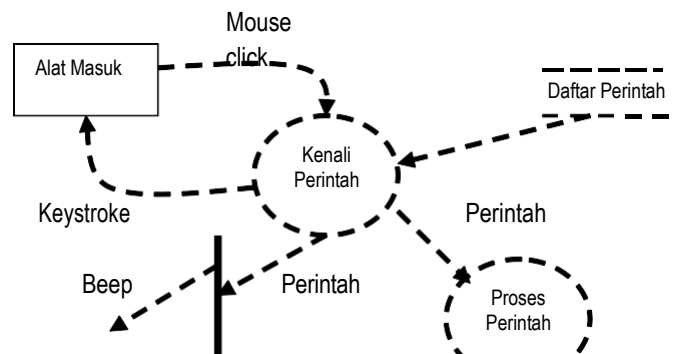
4.4.2.8 Diagram Aliran Kendali

Diagram aliran kendali atau control flow diagram (CFD) adalah diagram untuk menggambarkan perilaku sistem yang berhubungan dengan pemrosesan kendali atau event. Notasi CFD diperkenalkan oleh Ward and Mellor dan dilengkapi oleh Hateley and Pirbhai [PRE01] seperti ditunjukkan oleh Gambar 5.8.

Simbol	Arti
	Proses
	Proses kendali
	Event atau item kendali
	Data kontinyu
	Tempat penyimpanan item kendali
	Referensi bagi spesifikasi kendali

Gambar 5.8 Notasi CFD

Gambar 5.9 berikut memperlihatkan contoh penggunaan CFD untuk pemrosesan perintah dalam bentuk penekanan tombol keyboard (keystroke) atau mouse click.



Gambar 5.9 Contoh Penggambaran CFD untuk Pemrosesan Perintah

4.4.3 Tahap Analisis Terstruktur

Seperti telah disampaikan sebelumnya, ada dua versi analisis terstruktur yaitu Analisis Terstruktur Klasik (Classical Structured Analysis) dan Analisis Terstruktur Modern (Modern Structured Analysis). Walaupun secara historis kedua versi ini berangkat dari pendekatan yang sama, tetapi keduanya mempunyai perbedaan yang cukup mendasar saat penggunaannya.

4.4.3.1 Pendekatan Lama (Classical Structured Analysis)

Analisis Terstruktur Klasik dibuat untuk memperbaiki penyajian spesifikasi fungsional sebagai hasil analisis melalui penggunaan perangkat pemodelan secara grafis. Saat itu, spesifikasi fungsional banyak ditulis secara naratif, sehingga memungkinkan terjadinya kesalahan interpretasi.

Untuk itu, pengerjaan setiap tahap Analisis Terstruktur Klasik dilakukan dengan menggunakan konsep DFD. Adapun urutan pelaksanaannya adalah sebagai berikut: [DEM78]

- 1) Mempelajari lingkungan fisis dari sistem yang sedang berjalan dan menggambarkan DFD fisis sistem yang lama (Current Physical Data Flow Diagram)
- 2) Gambarkan DFD logis sistem lama (Current Logical Data Flow Diagram) yang ekuivalen dengan DFD fisis sistem lama.
- 3) Buat DFD logis sistem baru (New Logical Data Flow Diagram) berdasarkan DFD logis sistem lama dan kebutuhan pemakainya berikut dokumen pendukung.
- 4) Gambarkan sekumpulan DFD fisis sistem baru (New Physical Data Flow Diagram) yang ekuivalen dengan DFD logis sistem baru yang bersifat sementara.
- 5) Mengkuantifikasi biaya dan jadwal untuk masing-masing DFD fisis sistem yang baru dihasilkan.
- 6) Memilih DFD fisis sistem baru yang memungkinkan bagi pemakai berdasarkan nomor 5.
- 7) Buat dokumen spesifikasi terstruktur.

Pada penjelasan diatas kata sistem yang digunakan disini mengartikan sistem sebagai perangkat lunak.

4.4.3.2 Pendekatan Baru (Modern Structured Analysis)

Merupakan pendekatan yang diusulkan oleh Edward Yourdon [YOU89] untuk memperbaiki pelaksanaan analisis dengan pendekatan klasik. Beberapa alasan yang melatarbelakangi lahirnya Analisis Terstruktur Modern adalah:

- Alokasi waktu yang digunakan untuk membuat model fisis dan logis sistem lama akan memberikan dampak pada aktivitas lain dari proyek, padahal sistem tersebut akan digantikan oleh yang baru.
- Kaburnya pengertian tentang model fisis dan model logis. Model fisis adalah model implementasi (model bagaimana sistem akan diimplementasi), sementara model logis adalah model yang tidak tergantung pada implementasi.
- Selain untuk sistem informasi, Analisis Terstruktur juga digunakan untuk sistem waktu nyata atau non-sistem informasi lainnya. Bagaimana penggambaran model fisis dan logis sistem lama jika akan dibuat perangkat lunak word processor?
- Analisis Terstruktur Klasik hanya fokus pada pemodelan fungsional, belum mencakup pemodelan data dan perilaku sistem.

Secara umum tahap Analisis Terstruktur dengan pendekatan baru ini adalah:

- 1) Bangun model sistem esensial (essential system model)
Model esensial adalah model yang menggambarkan apa yang harus dilakukan oleh sistem berdasarkan apa yang diinginkan oleh pemakai. Model esensial terdiri dari
 - Model lingkungan (environmental model)
 - i) Buat pernyataan kegunaan sistem (statement of purpose).
 - ii) Buat Diagram Konteks dari sistem
 - iii) Daftar semua kejadian dari konteks
 - Model kelakuan (behavioral model)
 - i) Buat DFD logis sistem baru berdasarkan daftar kejadian dari konteks dan kebutuhan pemakai.
 - ii) Buat Kamus Data (Data Dictionary)
 - iii) Buat Spesifikasi Proses (Process Specification)
- 2) Bangun model implementasi (implementation model)
Model implementasi adalah model yang menggambarkan bagaimana kira-kira bentuk implementasi sistem bagi pemakai. Pembuatan model implementasi mencakup :
 - Menentukan batas sistem yang akan diotomasi
 - Menentukan antarmuka manusia-mesin:
 - i) perangkat masukan keluaran
 - ii) format untuk semua masukan yang mengalir dari terminator ke sistem
 - iii) format untuk semua keluaran yang mengalir dari sistem kembali ke terminator
 - iv) urutan dan waktu dari masukan dan keluaran untuk on-line system
 - Mengidentifikasi aktivitas-aktivitas dukungan manual tambahan
 - Menentukan kendala-kendala operasional

4.4.4 Mekanisme Analisis Terstruktur

Tahap Analisis Terstruktur dimulai dengan membuat Diagram Konteks (disebut juga model sistem dasar atau model konteks) yang menggambarkan fungsi sistem sebagai sebuah proses transformasi tunggal. Proses dan aliran informasi pada Diagram Konteks kemudian dipecah dan dibagi-bagi menjadi tingkat-tingkat DFD selanjutnya untuk pemerincian (tingkat 1, 2, 3, dan seterusnya) sampai didapat proses atomik (proses-proses yang akan dikerjakan oleh komputer). Pemerincian dilakukan untuk menghindari kompleksitas penggambaran DFD.

Untuk mendeskripsikan data yang mengalir dalam sistem digunakan Kamus Data, sementara Spesifikasi Proses ditulis untuk menerangkan prosesnya. Hubungan antar tempat penyimpanan data dapat dimodelkan dengan menggunakan Diagram Entity-Relationship (Diagram E-R).

Beberapa arahan dalam penurunan (pemerincian) DFD:

- Diagram Konteks harus menunjukkan sistem sebagai suatu proses tunggal.
- Semua anak panah dan proses harus diberi label dengan nama-nama yang mempunyai makna (arti).
- Masukan dan keluaran sistem harus secara hati-hati ditentukan.
- Konsistensi aliran informasi harus dijaga dari tingkat ke tingkat berikutnya.
- Hanya satu proses yang diperbaiki pada suatu saat.
- Perbaikan harus dimulai dengan mengisolasi calon-calon proses, item-item data dan penyimpanan yang direpresentasikan pada tingkat selanjutnya.

Beberapa perbaikan untuk arahan di atas diberikan oleh Edward Yourdon [YOU89] yang dapat dilaksanakan setelah Diagram Konteks didapat:

- Daftar kejadian-kejadian yang terjadi pada sistem atau yang harus ditangani oleh sistem.
- Buat satu proses untuk masing-masing kejadian yang menunjukkan aliran data yang terjadi pada kejadian tersebut.
- Setelah masing-masing kejadian dinyatakan sebagai proses, maka akan terlihat informasi/aliran data yang terlibat. Berdasarkan hasil tersebut perbaiki diagram konteks dengan mencantumkan seluruh aliran data yang terlibat dalam sistem.
- Gabungkan kejadian-kejadian yang satu kelompok untuk mendapatkan satu proses yang merupakan turunan dari diagram konteks.
- Setelah itu proses-proses dapat diuraikan lagi menjadi tingkat-tingkat selanjutnya sampai menjadi proses-proses atomik.

Latihan

1. Sebutkan tahap tahap dalam analisa kebutuhan.
2. Sebutkan pendekatan dalam melakukan metoda analisis
3. Sebutkan tujuan dari pembuatan SKPL
4. Jelaskan syarat-syarat pembentukan SKPL

5 Perancangan Perangkat Lunak

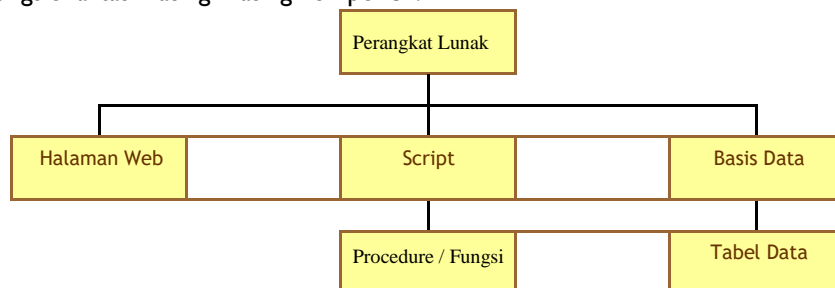
5.1 Pengertian

Perancangan perangkat lunak dapat didefinisikan sebagai

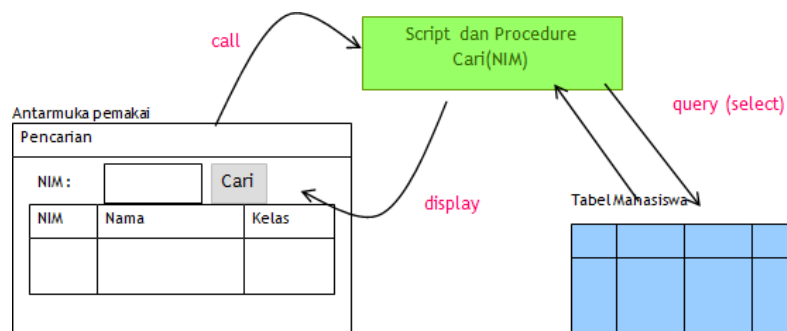
- Proses untuk mendefinisikan suatu model atau rancangan perangkat lunak dengan menggunakan teknik dan prinsip tertentu sedemikian hingga model atau rancangan tersebut dapat diwujudkan menjadi perangkat lunak.
- Proses mendefinisikan arsitektur perangkat lunak, komponen, modul, antarmuka, pendekatan pengujian, serta data untuk memenuhi kebutuhan yang sudah ditentukan sebelumnya. [IEE98]
- Proses bertahap dimana semua kebutuhan yang ada diterjemahkan menjadi suatu cetak biru yang akan digunakan untuk mengkonstruksi perangkat lunak. [PRE01]

Tujuan dilakukannya perancangan oleh seorang designer system (software engineer) adalah

- Mendekomposisi sistem (perangkat lunak) menjadi komponen-komponennya (data, antarmuka, prosedur, arsitektur). Sebagai gambaran, pada gambar 5.1 menunjukkan dekomposisi perangkat lunak menjadi halaman web (antarmuka), script (prosedur) dan basisdata/tabel data (desain data).
 - Menentukan relasi antar komponen.
 - Menentukan mekanisme komunikasi antar komponen.
- Sebagai gambaran, pada gambar 5.2 yang menunjukkan mekanisme dan relasi antar komponen perangkat lunak yaitu relasi antarmuka pemakai ke prosedur/script untuk meminta sebuah data yang diinginkan pengguna serta bagaimana sebuah prosedur mengakses tabel data agar dapat ditampilkan sesuai dengan permintaan pemakai pada antarmuka pemakai.
- Menentukan antarmuka komponen.
 - Menjelaskan fungsionalitas masing-masing komponen.



Gambar 5.1 Dekomposisi perangkat lunak menjadi komponen-komponennya.



Gambar 5.2 Mekanisme dan relasi antar komponen perangkat lunak.

5.2 Prinsip Perancangan

Perancangan perangkat lunak merupakan model dan proses. Proses perancangan merupakan serangkaian langkah yang memungkinkan seorang desainer menggambarkan semua aspek perangkat lunak yang dibangun, sedangkan model perancangan hampir sama dengan rencana arsitek untuk sebuah rumah yaitu memulai dengan menyajikan totalitas hal yang akan dibangun (misal pandangan 3 dimensi dari rumah yang akan dibangun, setelah itu akan disaring hal-hal yang memberikan panduan bagi pembangunan setiap detail dari rumah, seperti layout ruangan, layout pipa dan lainnya). Sama halnya dengan model perancangan yang dibuat untuk perangkat lunak memberikan berbagai pandangan yang berbeda terhadap program komputer.

Ada beberapa prinsip yang dikemukakan oleh Davis [DAV95] yang perlu diketahui oleh desainer untuk dapat mengendalikan proses perancangan, yaitu

- Perancangan harus dapat ditelusuri sampai ke model analisis.
- Perancangan tidak boleh berulang, maksudnya dapat menggunakan kembali rancangan yang sudah ada sebelumnya (reusable component).
- Perancangan dapat diperbaiki atau diubah tanpa merusak keseluruhan sistem.
- Perancangan harus dinilai kualitasnya pada saat perancangan, bukan setelah sistem jadi dengan kata lain siap diimplementasikan.
- Perancangan harus mempunyai beberapa pendekatan alternatif rancangan.
- Perancangan harus mengungkap keseragaman dan integrasi
- Perancangan harus meminimalkan kesenjangan intelektual antara perangkat lunak dan masalah yang ada di dunia nyata. Maksudnya perancangan perangkat lunak harus mencerminkan struktur domain permasalahan.
- Perancangan bukanlah pengkodean dan pengkodean bukanlah perancangan.
- Perancangan harus dikaji untuk meminimalkan kesalahan-kesalahan konseptual. Desainer harus menekankan pada hal-hal yang penting seperti elemen-elemen konseptual (ambiguitas, inkonsisten).

Jika prinsip perancangan diatas diaplikasikan dengan baik, maka desainer telah mampu menciptakan sebuah perancangan yang mengungkapkan faktor-faktor kualitas eksternal dan internal[MEY88]. Faktor-faktor eksternal adalah sifat-sifat perangkat lunak yang dapat diamati oleh pemakai (misal kecepatan, reliabilitas, ketepatan, usabilitas). Sedangkan faktor internal lebih membawa pada perancangan berkualitas tinggi dan perspektif teknis dari perangkat lunak yang sangat penting bagi para perekayasa perangkat lunak. Untuk mencapai kualitas faktor internal, seorang desainer harus memahami konsep-konsep dari perancangan perangkat lunak.

5.3 Konsep Perancangan

Pada dasarnya konsep perancangan memberikan kerangka kerja atau pedoman untuk mendapatkan perangkat lunak yang bisa berjalan dengan baik. Ada beberapa konsep perancangan yang dikemukakan oleh Pressman [PRE01] dan perlu dipahami oleh seorang desainer agar mendapatkan perancangan yang berkualitas tinggi yaitu

1. Abstraksi

Abstraksi merupakan cara untuk mengatur kompleksitas sistem dengan menekankan karakteristik yang penting dan menyembunyikan detail dari implementasi. Tiga mekanisme dasar dari abstraksi yaitu :

- a. Abstraksi Prosedural, urutan instruksi yang mempunyai sebuah nama yang menggambarkan fungsi tertentu
- b. Abstraksi Data, kumpulan data yang mempunyai nama yang menggambarkan objek data.
- c. Abstraksi Control, mengimplikasikan sebuah mekanisme kontrol dari program.

2. Dekomposisi

Dekomposisi merupakan mekanisme untuk merepresentasikan detail-detail dari fungsionalitas. Dengan adanya dekomposisi membantu para desainer mengungkapkan detail tingkat rendah ketika perancangan sedang berjalan. Jadi dekomposisi membagi perancangan secara top-down/menyaring tingkat detail dari prosedural.

3. Modularitas

Mekanisme membagi perangkat lunak ke dalam elemen-elemen kecil dan dapat dipanggil secara terpisah, biasanya elemen ini sering disebut dengan modul.

Modularitas merupakan karakteristik penting dalam perancangan yang baik karena

- a. Menyediakan pemisahan fungsionalitas yang ada pada perangkat lunak.
- b. Memungkinkan pengembang mengurangi kompleksitas dari sistem.
- c. Meningkatkan skalabilitas, sehingga perangkat lunak dapat dikembangkan oleh banyak personal.

Modularitas perangkat lunak ditentukan oleh coupling dan cohesion:

- a. Coupling: derajat ketergantungan antar modul yang berinteraksi.
- b. Cohesion: derajat kekuatan fungsional dalam suatu modul

Modul yang baik harus mempunyai chesion yang tinggi dan coupling yang rendah.

Faktor-faktor yang mempengaruhi coupling:

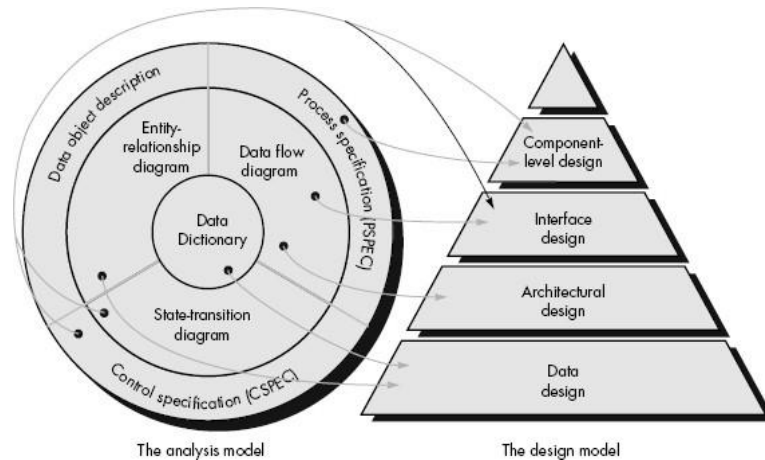
- a. Banyaknya data yang dilewatkan antar modul (passing parameter)
 - b. Banyaknya kontrol data yang dilewatkan antar modul.
 - c. Banyaknya data global yang digunakan bersama oleh beberapa modul.
4. **Arsitektur Perangkat Lunak**
Arsitektur perangkat lunak merupakan struktur hirarki dari komponen program (modul), cara bagaimana komponen tersebut berinteraksi dan struktur data yang digunakan oleh komponen.
5. **Hirarki Kontrol**
Hirarki kontrol disebut juga dengan struktur program, yang merepresentasikan oraganisasi (hirarki) komponen program (modul) serta mengimplikasikan suatu hirarki kontrol. Hirarki kontrol tidak mengimplikasikan aspek prosedural dari perangkat lunak, seperti urutan proses, kejadian/urutan keputusan, atau pengulangan operasi.
Hirarki kontrol juga merepresentasikan dua karakteristik yang berbeda dari arsitektur perangkat lunak yaitu visibilitas dan konektivitas. Visibilitas menunjukkan serangkaian komponen program yang dapat diminta dan dipakai sebagai data oleh komponen yang diberikan dan dilakukan secara tidak langsung. Sedangkan konektivitas mengindikasikan serangkaian komponen program yang diminta secara tidak langsung atau digunakan data oleh sebuah modul yang ditetapkan.
6. **Partisi Struktural**
Struktur program harus dipartisi secara horisontal maupun struktural. Partisi ini membagi cabang-cabang yang terpisah dari hirarki modul untuk menjadi sebuah fungsi program. Ada beberapa keuntungan yang didapat mempartisi arsitektur secara horisontal, yaitu
- a. menghasilkan perangkat lunak yang mudah diuji.
 - b. menghasilkan penyebaran efek samping yang sedikit.
 - c. menghasilkan perangkat lunak yang lebih mudah diperluas.
 - d. menghasilkan perangkat lunak yang lebih mudah dipelihara.
- Selain struktur program bisa dipartisi secara horisontal bisa juga dipartisi secara vertikal, dimana kontrol dan kerja dari arsitektur program didistribusikan secara top-down.
7. **Struktur Data**
Struktur data merepresentasikan hubungan logis antara elemen-elemen data. Selain itu struktur data juga menentukan organisasi, metode akses, tingkat assosiativitas dan alternatif pemrosesan untuk informasi.
8. **Prosedur Perangkat Lunak**
Prosedur perangkat lunak lebih berfokus pada detail-detail pemrosesan dari masing-masing modul. Prosedur harus memberikan spesifikasi yang teliti terhadap pemrosesan, mencakup event, keputusan, operasi, dan struktur data.
9. **Penyembunyian Informasi**
Sebuah mekanisme perancangan modul sehingga informasi yang terkandung dalam modul tidak dapat diakses oleh modul lain yang tidak berkepentingan dengan informasi tersebut. Informasi yang disembunyikan terdiri dari
- a. Representasi data
 - b. Algoritma seperti teknik pengurutan dan pencarian
 - c. Format masukan dan keluaran
 - d. Perbedaan mekanisme/kebijakan
 - e. Antarmuka modul tingkat rendah

Ada beberapa alasan kenapa konsep perancangan ini perlu dipahami oleh desainer yaitu

1. Mengatur sistem perangkat lunak yang kompleks.
2. Meningkatkan kualitas faktor dari perangkat lunak.
3. Memudahkan penggunaan kembali simantik sistem atau perangkat lunak.
4. Memecahkan permasalahan-permasalahan perancangan yang ada pada umumnya.

5.4 Transformasi Model Analisa ke Perancangan

Masing-masing elemen pada model analisis (bab 4) akan memberikan informasi yang diperlukan untuk menciptakan model perancangan. Pada gambar 5.3 menunjukkan bagaimana menterjemahkan model analisis menjadi empat model perancangan.



Gambar 5.3 Transformasi model analisis ke model perancangan perangkat lunak

Pada tahap perancangan ini akan dihasilkan empat model/objek perancangan, yaitu

- Perancangan data, yang berupa tabel-tabel basis data / file data konvensional Dan struktur data internal (jika diperlukan).
- Perancangan arsitektur yang berupa Structure chart dan struktur menu program (sebagai pelengkap)
- Perancangan antarmuka (interface)
- Perancangan level komponen/prosedural yang berupa spesifikasi program (algoritma)

5.5 Tahap Perancangan

Berikut tahapan-tahapan dalam perancangan perangkat lunak.

1. Menentukan bagaimana (how) solusi untuk memenuhi kebutuhan (what) yang sudah didefinisikan.
2. Memvalidasi solusi:
 - a. Apakah sudah benar-benar memenuhi kebutuhan
 - b. Apakah sudah menjamin kualitas yang diinginkan
 - c. Apakah dapat diimplementasikan di lingkungan yang sudah ditetapkan
 - d. Apakah sudah memperhatikan kendala-kendala perancangan
3. Mendekomposisi dan memodelkan solusi:
 - a. Rancangan data
 - b. Rancangan arsitektur perangkat lunak
 - c. Rancangan antarmuka pemakai
 - d. Rancangan prosedural (spesifikasi program)
4. Mendokumentasikan hasil rancangan pada dokumentasi deskripsi perancangan perangkat lunak atau Software Design Descriptions (SDD)

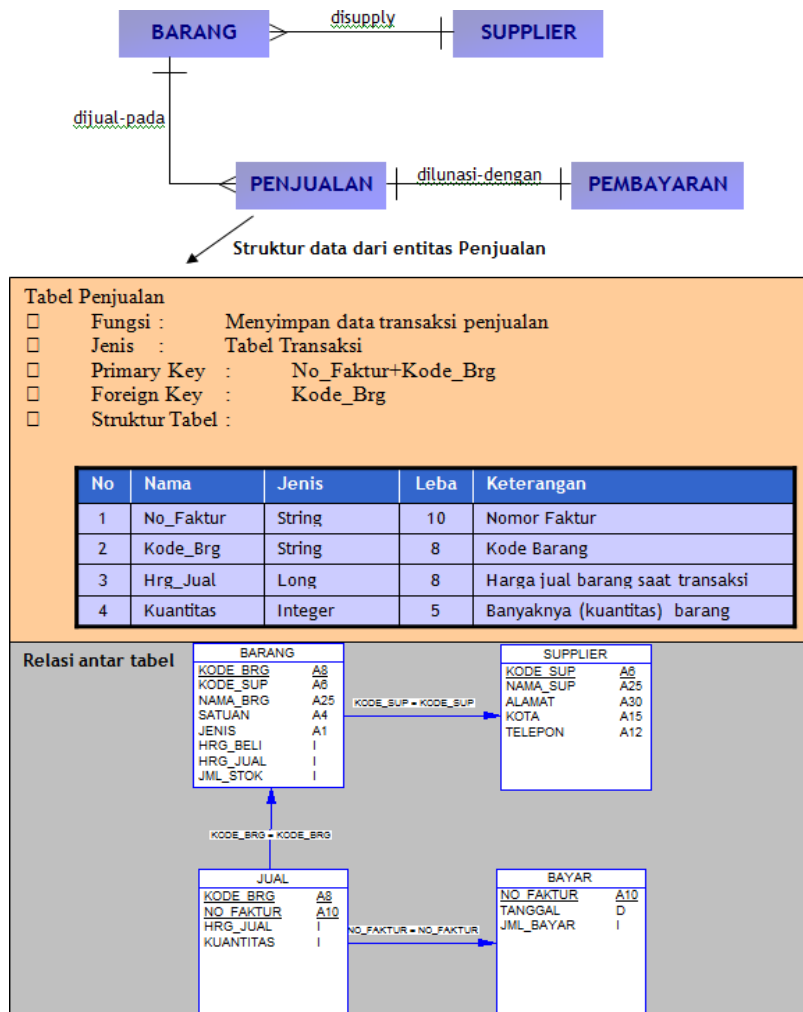
5.6 Perancangan Data

Perancangan data adalah aktivitas pertama dan beberapa sering mengatakan yang terpenting dari empat aktivitas perancangan yang dilakukan selama rekayasa perangkat lunak. Pengaruh struktur data pada struktur program dan kompleksitas prosedural menyebabkan perancangan data berpengaruh penting terhadap kualitas perangkat lunak. Konsep penyembunyian informasi dan abstraksi data memberikan dasar bagi pendekatan terhadap perancangan data. Tanpa melihat teknik perancangan yang digunakan, data yang didesain dengan baik dapat membawa kepada struktur program dan modularitas yang lebih baik, serta mengurangi kompleksitas prosedural.

Berikut tahapan untuk mentransformasikan model data pada tahap analisis yaitu E-R

- Transformasi Diagram E-R (conceptual data model, CDM) menjadi model relasi (skema relasi, tabel relasi).
- Penentuan atribut relasi sesuai dengan kamus data yang telah dibuat.
- Normalisasi.
- Pendefinisian struktur tabel.
- Pembuatan relasi antar tabel (physical data model, PDM)

Sebagai gambaran, perhatikan gambar 5.4 Transformasi E-R Diagram ke struktur data dan relasi antar tabel.



Gambar 5.4 Transformasi ER-diagram ke struktur data dan relasi antar tabel

5.5.1 Perancangan Arsitektur Perangkat Lunak

Gambaran bagaimana elemen/komponen fungsional perangkat lunak disusun, diorganisasi dan distrukturkan sehingga:





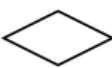
- **Hubungan** antar elemen/komponen dapat dijelaskan.
- **Interface** yang menghubungkan elemen/komponen dapat didefinisikan.
- **Wujud** dan **penempatan** elemen/komponen dalam tempat penyimpanan sekunder **secara fisik** dapat ditetapkan.

Sasaran utama perancangan arsitektur adalah untuk mengembangkan struktur program modular dan merepresentasikan hubungan kontrol antar modul. Perancangan arsitektur juga membentuk struktur program dan struktur data dengan menentukan antarmuka yang memungkinkan data mengalir melalui program. Alat pemodelan untuk merancang arsitektur perangkat lunak menggunakan structure chart.

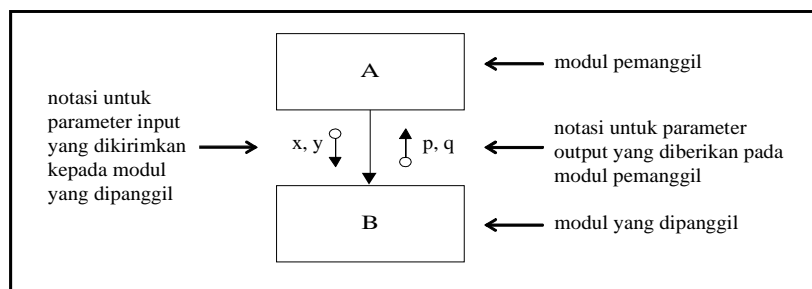
Structure Chart

- Diagram yang menggambarkan hirarki modul serta hubungan antar modul tersebut (khususnya hubungan dan coupling antar modul) [DEM78].
- Diagram untuk menggambarkan arsitektur perangkat lunak secara keseluruhan tanpa memperlihatkan proses pemilihan dan pengulangannya secara rinci. Teknik ini menggambarkan arsitektur perangkat lunak seperti diagram organisasi sebuah perusahaan.[MAR85]

Berikut daftar simbol-simbol yang digunakan untuk membangun structure chart.

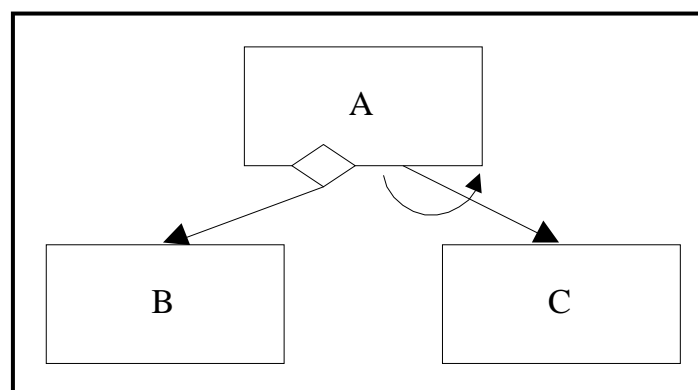
Simbol	Arti
	• Modul
	• Pemanggilan modul
	• Data atau elemen kontrol yang dikirimkan atau diterima dari satu modul
	• Pengulangan di dalam modul
	• Penyeleksian kondisi di dalam modul

Sebagai contoh, pada gambar 5.5 dibawah menunjukkan pemakaian diagram structure chart



Gambar 5.5 a. Pemakaian diagram structure chart

Pada gambar 5.5 a menunjukkan modul A memanggil modul B dengan data x dan y sebagai parameternya dan modul B mengirimkan data p dan q sebagai return value ke modul A.



Gambar 5.5 b. Pemakaian diagram structure chart

Sedangkan pada gambar 5.5 b menunjukkan modul A akan memanggil modul B jika kondisi dalam modul A dipenuhi dan modul A akan memanggil modul C secara berulang.

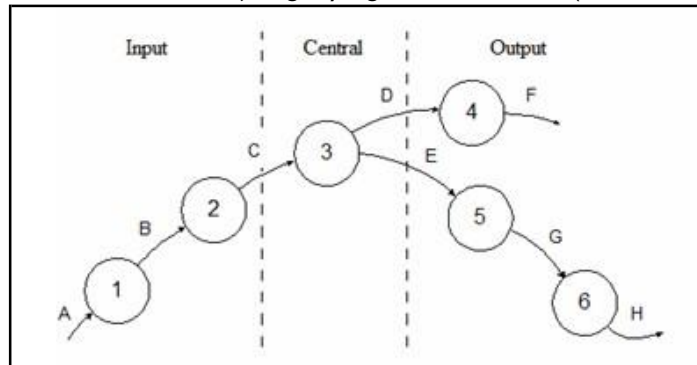
Tahapan pelaksanaan mentransformasikan DFD ke Structure Chart

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

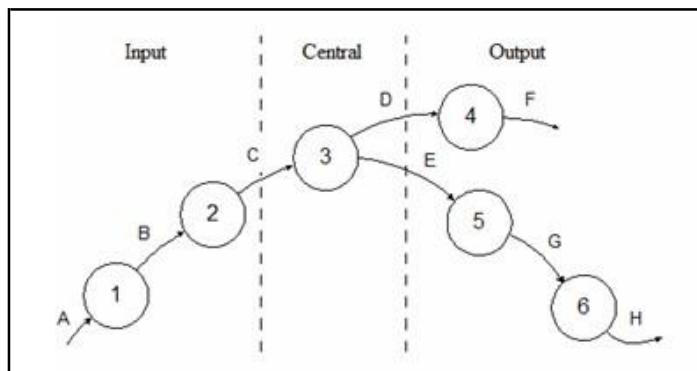
Ada 2 teknik pendekatan dalam memetakan DFD ke structure chart yaitu

1. Analisis transformasi

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 5.6 Aliran Informasi berbentuk transformasi

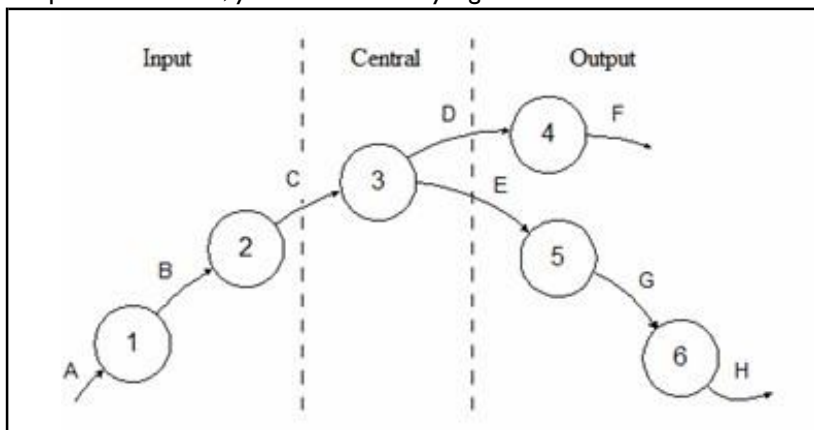


Gambar 5.7 Transformasi DFD gambar 6.6 ke structure chart

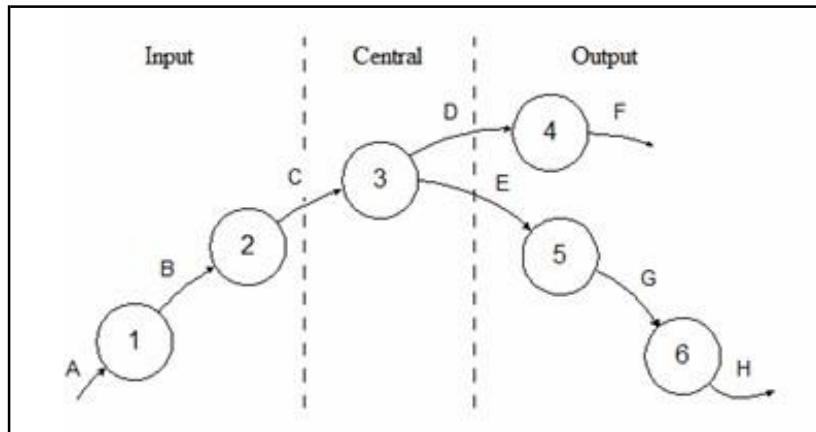
Gambar 5.7 menunjukkan hasil transformasi DFD ke structure chart

2. Analisis transaksi

Analisis transaksi merupakan strategi perancangan alternatif yang digunakan untuk merancang program-program yang memproses transaksi, yaitu elemen data yang memicu sebuah aksi.



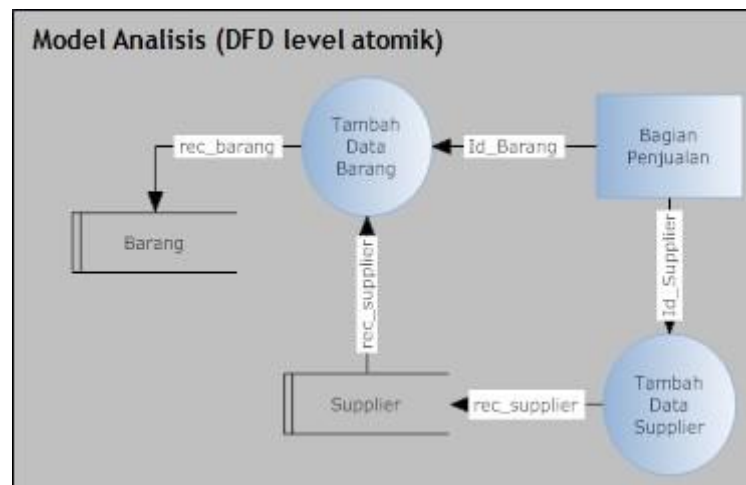
Gambar 5.8 Aliran Informasi berbentuk transaksi



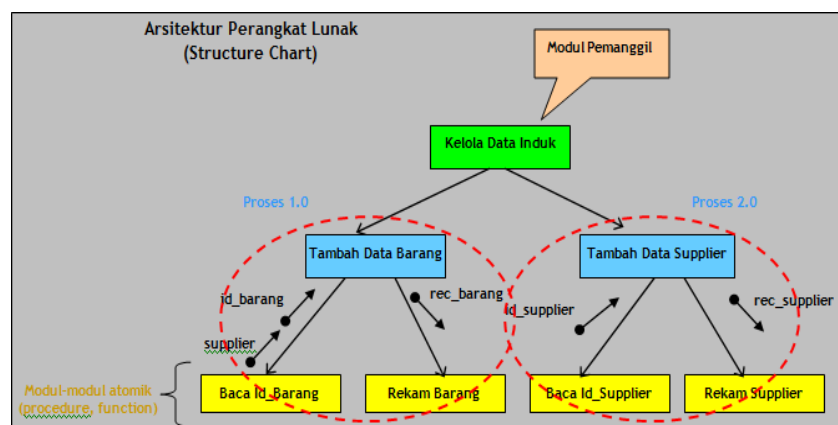
Gambar 5.9 Transformasi DFD gambar 6.8 ke structure chart

Gambar 5.9 menunjukkan hasil transformasi DFD ke structure chart

Sebagai contoh untuk memberikan gambaran bagaimana mentransformasikan DFD ke struktur chart. Perhatikan gambar 5.10 dibawah ini.



Gambar 5.10 DFD level atomik pada model analisis



Gambar 5.11 Hasil traansformasi DFD level atomik pada model analisis ke structure chart

5.5.2 Perancangan Antarmuka (Interface)

Secara fisik antarmuka pengguna yang dirancang adalah tampilan layar (form, halaman web). Jenisnya dapat berupa:

- Menu pilihan
- Form isian (entry)
- Penyajian informasi (report, query)
- Kotak dialog, jika diperlukan
- Fasilitas bantuan (Help), jika diperlukan

Perancangan antarmuka menfokuskan pada tiga area yaitu rancangan antarmuka antara modul-modul perangkat lunak, rancangan antarmuka antara perangkat lunak dengan entitas eksternal dan rancangan antarmuka antara perangkat lunak dengan pengguna perangkat lunak (manusia dengan komputer)

5.5.2.1 Perancangan antarmuka internal dan eksternal

Perancangan antarmuka internal, kadang disebut dengan interface intermodular yang dikendalikan oleh data yang mengalir diantara modul-modul dan karakteristik bahasa pemrograman yang akan diimplementasikan pada perangkat lunak.

Model analisis berisi banyak informasi yang dibutuhkan bagi perancangan interface intermodular. Diagram alir data (Bab 5) menggambarkan bagaimana objek data ditransformasikan ketika data bergerak melalui suatu sistem. Transformasi DFD dipetakan kedalam modul pada struktur program (Sub bab 6.5.2) sehingga anak panah (objek data) yang mengalir masuk dan keluar dari masing-masing transformasi DFD harus dipetakan ke dalam suatu perancangan untuk antarmuka modul yang sesuai dengan transformasi DFD tersebut.

Perancangan antarmuka eksternal dimulai dengan mengevaluasi masing-masing entitas eksternal yang direpresentasikan pada DFD model analisis. Kebutuhan data dan kontrol dari entitas eksternal akan ditentukan kemudian akan dirancang antarmuka yang sesuai.

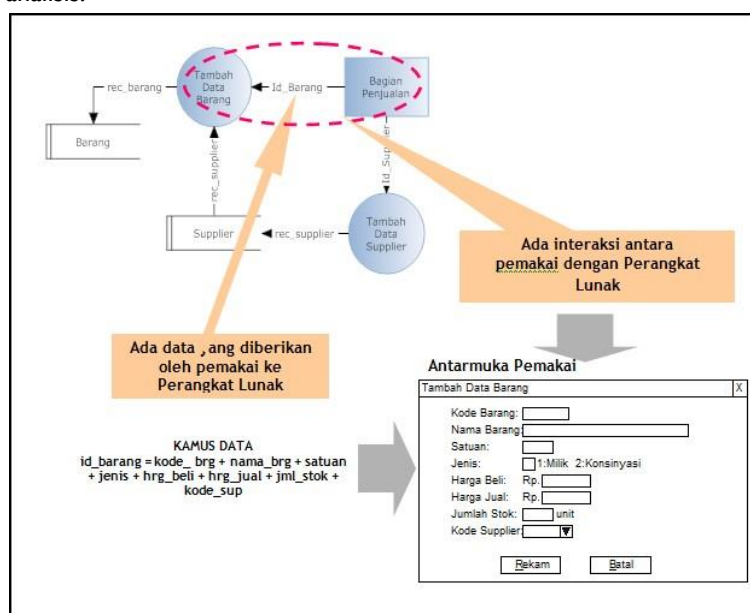
Baik perancangan antarmuka internal dan eksternal harus dirangkai dengan validasi data dan algoritma penanganan kesalahan dalam sebuah modul. Karena akan terjadi efek samping yang menyebar melalui antarmuka program, oleh karena itu penting untuk mengecek semua aliran data dari modul ke modul untuk memastikan bahwa data sudah sesuai dengan batas yang telah ditentukan pada saat analisis kebutuhan.

Perancangan antarmuka pengguna

Perancangan antarmuka pengguna berkaitan dengan studi terhadap manusia yaitu pengguna dari perangkat lunak yang dibangun. Selain terhadap pengguna berkaitan juga dengan isu-isu teknologi yang berhubungan dengan perangkat lunak. Berikut beberapa pertanyaan yang bisa mendukung dalam penentuan perancangan antarmuka pengguna.

- Siapakah para pengguna perangkat lunak?
- Bagaimana pengguna belajar berinteraksi dengan sistem komputer yang baru?
- Bagaimana pengguna menginterpretasikan informasi yang dihasilkan oleh sistem?
- Apakah yang diharapkan oleh sistem tersebut?

Sebagai gambaran, perhatikan gambar 5.12 yang menunjukkan bagaimana mengidentifikasi antarmuka pengguna dari DFD pada model analisis.



Gambar 5.12 Transformasi antarmuka pengguna dari DFD pada model analisis

5.5.3 Perancangan Prosedural (Spesifikasi Program)

Perancangan prosedural terjadi setelah data, perancangan arsitektur dan antarmuka dibangun. Perancangan prosedural merupakan penjelasan lebih rinci dan teknis dari spesifikasi proses. Deskripsi prosedural (algoritma) untuk semua modul-modul program yang menjadi elemen-elemen struktural dari arsitektur perangkat lunak dapat berupa:

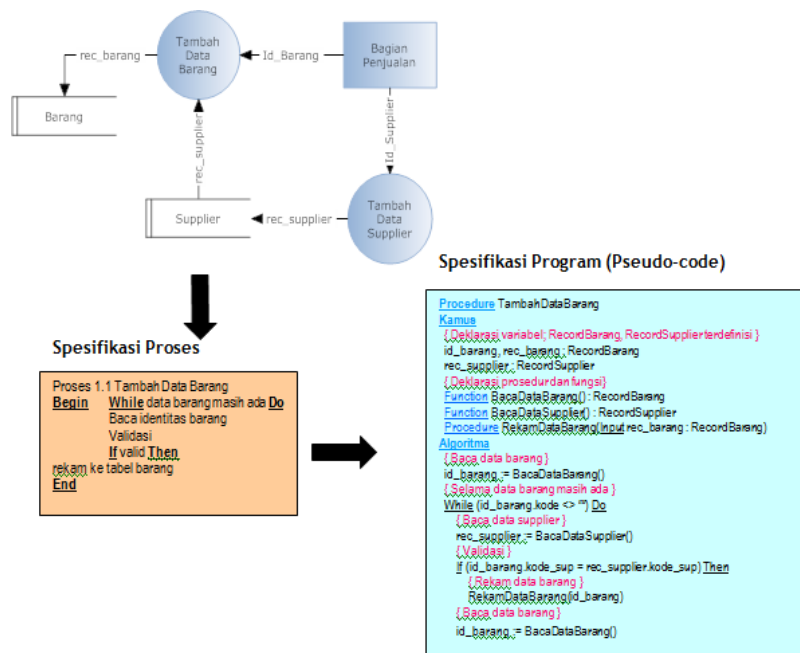
- Prosedur
- Fungsi

Spesifikasi prosedural diperlukan untuk menetapkan detail algoritma yang akan dinyatakan dengan menggunakan notasi pseudo-code, atau notasi yang mirip dengan bahasa pemrograman yang digunakan.

Pseudo-code sering juga dikenal dengan PDL (Program Design Language) notasi bahasa yang menggunakan kosakata dari satu bahasa (misal bahasa Inggris) and keseluruhan sintaks dari yang lain (misal bahasa pemrograman terstruktur).

Perbedaan antara PDL dengan pemrograman modern terletak pada penggunaan teks narasi (seperti bahasa Inggris) yang dilekatkan langsung pada struktur sintaksis pada statemen PDL. Sehingga PDL tidak dapat dicompile seperti bahasa pemrograman modern. Namun, sekarang sudah ada processor PDL yang bisa menterjemahkan PDL ke dalam representasi grafis (misal sebuah diagram alir) dari desain dan informasi lainnya.

Sebagai gambaran, penggunaan pseudo-code dalam merepresentasikan spesifikasi proses perhatikan pada gambar 5.13 dibawah ini.



Gambar 5.13 Transformasi Spesifikasi Proses ke pseudo-code

5.5.4 5.6 Dokumentasi Perancangan

Dokumentasi perancangan disebut juga dengan Deskripsi Perancangan Perangkat Lunak (DPPL) atau Software Design Description (SDD), merupakan sebuah dokumen yang berisi spesifikasi/deskripsi perancangan model perangkat lunak yang akan dibangun. Pada gambar 5.14 dapat digunakan sebagai outline dokumen SDD.

I.	Ruang Lingkup
A.	Sasaran Sistem
B.	Kebutuhan Utama Perangkat Lunak
C.	Batasan Perancangan
II.	Perancangan Data
A.	Objek Data dan Struktur Data
B.	Struktur File dan Basisdata
1.	Struktur File Eksternal
a.	Struktur Logis
b.	Deskripsi Record Logis
c.	Metode Akses
2.	Data Global
3.	File & Referensi Lintas Data
III.	Perancangan Arsitektur
A.	Kajian Data dan Aliran Kontrol
B.	Struktur Program
IV.	Perancangan Antarmuka
A.	Spesifikasi Antarmuka Pemakai
B.	Aturan Perancangan Antarmuka Pemakai
C.	Perancangan Antarmuka Eksternal
1.	Antarmuka untuk Data Eksternal
2.	Antarmuka untuk Peralatan Eksternal
V.	Perancangan Prosedural
	<i>Untuk masing-masing modul</i>
A.	Naratif Pemrosesan
B.	Deskripsi Antarmuka
C.	Deskripsi Bahasa Perancangan
D.	Modul-modul yang digunakan
E.	Struktur Data Internal
F.	Keterangan/Batasan
VI.	Kebutuhan Lintas –Referensi
VII.	Ketentuan Pengujian
A.	Panduan Pengujian
B.	Strategi Integrasi
C.	Pertimbangan Khusus
VIII.	Catatan Khusus
IX.	Lampiran

Gambar 5.14 Outline Dokumen Deskripsi Perancangan Perangkat Lunak (DPPL)

Perancangan dan Kualitas Perangkat Lunak

McGlaughin [McG91] mengusulkan 3 karakteristik yang berfungsi sebagai pedoman bagi evaluasi sebuah perancangan yang baik :

1. Perancangan harus mengimplementasikan keseluruhan kebutuhan eksplisit yang dibebankan dalam model analisis, dan harus mengakomodasi semua kebutuhan implisit yang diinginkan oleh pelanggan.
2. Perancangan harus menjadi panduan yang dapat dibaca, mudah dipahami bagi mereka yang akan membuat kode dan yang pengujian serta memelihara perangkat lunak.
3. Perancangan harus memberikan sebuah gambaran lengkap mengenai perangkat lunak, yang menekankan data, dan domin perilaku dari perangkat lunak.

Sedangkan pressman [PRE01] untuk mengevaluasi kualitas dan representasi perancangan yang baik mengusulkan beberapa kriteria teknis sebagai berikut :

1. Perancangan harus memperlihatkan suatu organisasi hirarki yang baik dengan menggunakan kontrol diantara elemen-elemen perangkat lunak.
2. Perancangan harus modular, yaitu perangkat lunak harus dipartisi secara logika ke dalam elemen-elemen yang melakukan fungsi dan subfungsi khusus.
3. Perancangan harus berisi data dan abstraksi prosedural.
4. Perancangan harus membawa ke arah modul (misal subrutin atau prosedur) yang memperlihatkan karakteristik fungsional.
5. Perancangan harus mengarah kepada antarmuka untuk mengurangi kompleksitas hubungan antar modul-modul dan lingkungan eksternal.
6. Perancangan harus dapat menggunakan metode berulang yang dikendalikan oleh informasi yang diperoleh selama analisis kebutuhan perangkat lunak.

Beikut beberapa faktior kualitas pada perancangan perangkat lunak:

- Corectness: Tingkatan sebuah perangkat lunak dalam memenuhi spesifikasi dan sasar n yang diinginkan oleh pelanggan..
- Testability: Usaha yang diperlukan untuk menguji sebuah perangkat lunak untuk memastikan apakah perangkat lunak sudah melakukan fungsi-fungsi yang dimaksud..
- Integrity: Tingkat kendali perangkat lunak terhadap penggunaan oleh orang-orang yang tidak berkepentingan.
- Reusability: tingkat dimana sebuah program (bagian dari suatu program) dapat digunakan kembali ke dalam aplikasi yang lain – yang berhubungan dengan ruang lingkup dan fungsi yang dilakukan oleh program.
- Portability, usaha yang diperlukan untuk memindahkan program dari satu perangkat keras dan atau lingkungan sistem perangkat lunak ke yang lainnya.

- Reliability, tingkat dimana sebuah program dapat melakukan fungsi yang diharapkan dengan ketelitian yang diminta.
 - Maintability, usaha yang diperlukan untuk mencari dan membetulkan kesalahan pada sebuah program.
 - Interoperability, usaha yang diperlukan untuk merangkai sistem yang satu dengan yang lainnya.
 - Effiecient, jumlah sumber daya perhitungan dan kode yang diperlukan oleh program untuk melakukan fungsinya.
- Usability, usaha yang dibutuhkan untuk mempelajari, mengoperasikan, menyiapkan input, dan menginterpretasikan output sebuah program.

Latihan

1. Jelaskan tahap tahap dalam perancangan perangkat lunak
2. Jelaskan pegertian dari perancangan perangkat lunak
3. Jelaskan prinsip dari perancangan perangkat lunak
4. Jelaskan apa yang dimaksud dengan DPPL atau SDD

6 Implementasi Perangkat Lunak

6.1 Aktivitas Implementasi

Merupakan sekumpulan aktivitas di mana rancangan perangkat lunak yang telah dibuat pada tahap perancangan kemudian dikodekan ke dalam bentuk kode program dengan menggunakan bahasa pemrograman tertentu agar dapat dijalankan pada komputer.

Fondasi dari aktivitas ini adalah pemrograman. Tools untuk membuat program disebut bahasa pemrograman. Programmer membuat program dengan panduan dokumentasi rancangan perangkat lunak, namun pada umumnya programmer juga memeriksa semua dokumen dari tahapan-tahapan sebelumnya (semisal SKPL) untuk memeriksa konsistensi dari dokumentasi-dokumentasi yang ada.

6.2 Aktivitas Pemrograman

Program adalah serangkaian ekspresi yang disusun menjadi kesatuan prosedur berupa urutan langkah untuk menyelesaikan suatu permasalahan dan diimplementasikan dalam bentuk bahasa pemrograman sehingga dapat dijalankan pada komputer. Adapun bahasa pemrograman merupakan tatacara penulisan program. Pada bahasa pemrograman terdapat dua faktor penting, yakni:

- Sintaks, yaitu aturan-aturan gramatikal yang mengatur tatacara penulisan ekspresi/statemen
- Semantik, yaitu aturan-aturan untuk menyatakan suatu arti

6.2.1 Standar Program yang Baik

Standar pemrograman dibutuhkan untuk menciptakan suatu program dengan portabilitas yang tinggi sehingga memudahkan dalam merancang dan merawat program serta meningkatkan efektivitas penggunaan peralatan komputer. Beberapa standar dasar penilaian untuk sebuah program dikatakan baik antara lain:

1. Teknik pemecahan masalah
2. Penyusunan program
3. Perawatan program
4. Standar prosedur

6.2.1.1 Standar Teknik Pemecahan Masalah

Setelah masalah dipahami dengan baik, seorang pemrograman membutuhkan suatu teknik untuk memecahkan masalah tersebut. Ada dua pendekatan yang umum digunakan, yakni:

- Teknik Top-Down merupakan teknik pemecahan masalah di mana suatu masalah yang kompleks dibagi-bagi menjadi beberapa struktur hingga unit yang paling kecil, setelah itu kemudian disusun langkah-langkah untuk menyelesaikan masalah secara rinci. Teknik semacam ini digunakan pada metode pemrograman terstruktur
- Teknik Bottom-Up merupakan teknik pemecahan masalah yang berkebalikan dengan teknik Top-Down di mana penyelesaian masalah dimulai dari hal-hal yang bersifat khusus, kemudian naik ke bagian yang bersifat umum. Teknik semacam ini digunakan pada metode pemrograman berorientasi objek

Setelah memilih teknik pemecahan masalah, pemrogram mulai menyusun langkah-langkah untuk memecahkan masalah, yang disebut dengan algoritma. Algoritma yang baik memiliki ciri-ciri sebagai berikut:

- Tepat, benar, sederhana, standar, dan efektif
- Logis, terstruktur, dan sistematis
- Semua operasi terdefinisi
- Semua proses harus berakhir setelah sejumlah langkah dilakukan
- Menggunakan bahasa standar sehingga tidak ambigu

6.2.1.2 Standar Penyusunan Program

Beberapa faktor yang menjadi standar dalam penyusunan program antara lain:

- **Kebenaran logika dan penulisan**
Program yang disusun harus memiliki kebenaran logika dalam pemecahan masalah maupun penulisan kode program. Program harus tepat dan teliti dalam perhitungan sehingga hasilnya dapat dipercaya
- **Waktu minimum untuk penulisan program**
Penulisan program harus memiliki waktu minimum, artinya waktu minimal yang harus tersedia untuk menuliskan kode program dari awal hingga siap untuk dieksekusi
- **Kecepatan maksimum eksekusi program**
Agar program memiliki kecepatan eksekusi maksimum, perlu diperhatikan beberapa hal antara lain bahasa pemrograman yang digunakan, algoritma yang disusun, teknik pemrograman yang dipakai, dan perangkat keras yang digunakan. Kecepatan maksimum juga dapat ditingkatkan dengan memperbaiki struktur program misalkan:
 - Menghindari proses pengujian yang berulang-ulang secara percuma
 - Meletakkan syarat pengujian yang akan menolak data dengan jumlah terbanyak sebagai syarat pengujian pertama, syarat pengujian dengan jumlah terbanyak kedua sebagai syarat pengujian kedua, dan seterusnya
 - Memperbaiki susunan baris program guna meningkatkan kecepatan eksekusi
- **Ekspresi penggunaan memori**
Semakin sedikit penggunaan memori, semakin cepat program dieksekusi. Untuk meminimumkan penggunaan memori, maka perlu diperhatikan:
 - Menggunakan tipe data yang cocok untuk kebutuhan pemrograman
 - Menghindari penggunaan variabel berindeks secara berulang kali
- **Kemudahan merawat dan mengembangkan program**
Program yang memiliki struktur yang baik, struktur data jelas, dan dokumentasi yang lengkap dan mudah dipahami, akan mudah untuk dirawat dan dikembangkan
- **User friendly**
Program yang baik harus memiliki layanan untuk mempermudah pemakai untuk menggunakannya, misalkan layanan online help
- **Portabilitas**
Program yang baik harus dapat dijalankan pada kondisi platform yang berbeda-beda, baik itu sistem operasi maupun perangkat keras
- **Modular**
Pada pendekatan pemrograman, masalah dibagi-bagi menjadi unit terkecil, yang disebut modul untuk menyederhanakan pengimplementasian langkah-langkah pemecahan masalah dalam bentuk program

6.2.1.3 Standar Perawatan Program

Beberapa standar yang harus dipenuhi agar memudahkan pemrogram dalam merawat dan mengembangkan program antara lain:

1. **Dokumentasi**
Dokumentasi merupakan catatan dari setiap langkah pekerjaan membuat program dari awal hingga akhir. Dokumentasi ini penting untuk memudahkan menelusuri adanya kesalahan maupun untuk pengembangannya. Dokumentasi yang baik akan memberikan informasi yang memadai sehingga orang lain dapat mengerti dan memahami alur logika program
2. **Penulisan Instruksi**
Untuk memudahkan perawatan program, sebaiknya penulisan program dilakukan sebagai berikut:
 - Menuliskan satu instruksi pada satu baris program
 - Memisahkan modul-modul dengan memberikan spasi beberapa baris untuk mempermudah pembacaan
 - Membedakan bentuk huruf dalam penulisan program
 - Memberikan tabulasi yang berbeda untuk penulisan instruksi-instruksi yang berada pada loop atau struktur kondisional
 - Menghindari penggunaan konstanta dalam penulisan rumus, jika konstanta tersebut mungkin berubah
 - Melakukan pembatasan jumlah baris instruksi per modul

6.2.1.4 Standar Prosedur

Penggunaan prosedur standar akan memudahkan bagi pengembang program untuk mengembangkan program tersebut

6.3 Modularitas

Modularitas merupakan sebuah konsep untuk memecah program menjadi modul-modul kecil di mana masing-masing modul berinteraksi melalui antarmuka modul. Dengan adanya modularitas, kesalahan di satu bagian program dapat dikoreksi tanpa perlu mempertimbangkan bagian-bagian lainnya, program menjadi lebih sederhana sehingga lebih mudah dipahami.

6.3.1 Kriteria Modularitas

Terdapat lima kriteria modularitas, yakni:

1. **Decomposability**
Kemampuan untuk mendekomposisi masalah menjadi submasalah yang lebih sederhana dan dihubungkan dengan struktur yang sederhana
2. **Composability**
Kemampuan membangun modul-modul program yang kemudian dapat diintegrasikan menjadi program pada lingkungan yang mungkin berbeda dengan saat modul tersebut dibangun
3. **Understandability**
Kemampuan menghasilkan program di mana programmer dapat memahami masing-masing modul tanpa perlu mengetahui detailnya
4. **Continuity**
Kemampuan meredam propagasi perubahan, yaitu suatu kondisi di mana perubahan kecil pada satu modul memicu perubahan hanya pada satu modul atau sedikit modul yang terkait
5. **Protection**
Kemampuan meredam kondisi abnormal hanya pada satu modul

6.3.2 Aturan Modularitas

Terdapat pula lima aturan modularitas, antara lain:

1. **Direct mapping**
Struktur model yang ada pada masing-masing tahap pengembangan perangkat lunak semestinya kontinyu, dalam artian modul yang terdapat pada analisis masih merupakan modul pada tahap perancangan dan tetap menjadi modul pada saat pemrograman
2. **Few interfaces**
Setiap modul seharusnya berinteraksi dengan sesedikit mungkin dengan modul lain sebab jika terjadi banyak interaksi antar modul akan meningkatkan propagasi perubahan
3. **Small interfaces (weak coupling)**
Untuk modul-modul yang berkomunikasi, diusahakan informasi yang dipertukarkan pada saat komunikasi adalah sesedikit mungkin sehingga mengurangi ketergantungan antar modul
4. **Explicit interface**
Kapan saja modul X dan Y berkomunikasi maka komunikasi ini harus dari teks X atau Y atau keduanya
5. **Information hiding**
Pemrogram harus merancang modul dengan sekelompok fitur pada suatu modul tampak pada modul lain, sedangkan fitur lainnya diusahakan tersembunyi dari modul lain. Modul lain hanya berhubungan dengan modul lewat deskripsi pada fitur yang terlihat tersebut

6.3.3 Prinsip Modularitas

Terdapat juga lima prinsip modularitas, yakni:

1. **The Linguistic Modular Units principle**
Modul harus merupakan unit sintaks pada bahasa pemrograman yang digunakan. Prinsip ini umumnya dilanggar karena itu pengembang terpaksa harus melakukan translasi atau restrukturisasi terhadap model rancangan yang diperolehnya

2. The Self-Documentation Principle

Perancang modul harus membuat semua informasi mengenai modul yang berkaitan terdapat pada modul tersebut. Dokumentasi internal ini sangat penting untuk proses pengembangan dan pemeliharaan perangkat lunak

3. The Uniform Access Principle

Semua layanan modul seharusnya tersedia melalui notasi yang seragam tanpa memperhatikan pengimplementasian layanan tersebut apakah untuk keperluan penyimpanan atau komputasi

4. The Open-Closed Principle

Modul harus bersifat terbuka dalam artian terbuka untuk dikembangkan serta bersifat tertutup dalam artian komunikasi antar modul hanya melalui antarmuka yang telah ditetapkan mekanismenya

5. The Single Choice Principle

Kapan saja program harus mendukung beberapa alternatif, satu dan hanya satu modul pada program yang mengetahui daftar lengkap dari yang dimilikinya

6.3.4 Kriteria Modul yang Baik

Beberapa kriteria dari modul yang baik antara lain:

1. Kohesif

Modul dikatakan kohesif jika fungsionalitasnya terdefinisi dan terfokus dengan baik. Kohesi mengacu pada derajat elemen-elemen modul yang saling berhubungan. Modul kohesif melakukan satu tugas tunggal pada suatu prosedur program yang memerlukan sedikit interaksi dengan prosedur yang sedang dilakukan pada bagian lain program.

Modul yang melakukan serangkaian tugas yang saling berhubungan secara lepas disebut sebagai kohesif koincidental. Modul yang melakukan tugas yang berhubungan secara logis disebut kohesif secara logis. Bila modul berisi tugas-tugas yang dieksekusi dalam jangka waktu sama, maka modul-modul tersebut disebut kohesif temporal.

Bila elemen pemrosesan dari suatu modul dihubungkan dan dieksekusi dalam suatu urutan yang spesifik, maka akan muncul kohesi prosedural. Dan bila semua elemen pemrosesan berkonsentrasi pada satu area pada suatu struktur data, maka terjadi kohesi komunikasional

2. Loosely coupled

Coupling mengacu kepada derajat modul-modul saling berkomunikasi. Modul-modul harus seminimal mungkin berkomunikasi dengan modul-modul lain. Maka dari itu nilai derajat coupling harus sekecil mungkin

3. Enkapsulasi

Modul harus memenuhi persyaratan information hiding. Atribut dari modul seharusnya tidak secara langsung tersedia untuk modul-modul lain. Atribut-atribut modul hanya tersedia ke modul-modul lain melalui antarmuka yang telah ditetapkan. Enkapsulasi mengimplikasikan pemahaman implementasi modul tertentu tidak dibutuhkan bagi pemakai modul sehingga tidak perlu mengetahui detail dan keseluruhan isi modul

4. Reuseability

Merupakan sasaran strategis rekayasa perangkat lunak dan dapat meningkatkan produktivitas pengembangan perangkat lunak. Implikasi dari reuseability adalah fungsionalitas modul harus segeneral dan seluas mungkin sehingga dapat digunakan oleh modul lain dan dapat mengurangi waktu dan biaya yang dikeluarkan

6.4 Abstraksi Data

Abstraksi data merupakan suatu cara untuk menggambarkan data dengan memisahkannya dari implementasinya. Salah satu jenis abstraksi data adalah tipe data dan juga ADT (Abstract Data Type).

Dengan abstraksi, seorang pemrogram tidak memperdulikan bagaimana data itu diimplementasikan, contohnya tipe data int merupakan abstraksi dari sekumpulan bit di memori sebagai bilangan bulat.

Tipe data merupakan sekumpulan nilai dan operasi yang diasosiasikan pada nilai-nilai itu. Sedangkan ADT mendeklarasikan sekumpulan nilai, operasi pada nilai, dan aksioma-aksioma yang senantiasa dipenuhi oleh operasi-operasi tersebut. ADT tidak mendefinisikan cara nilai tersebut diimplementasikan sehingga mungkin terdapat beberapa implementasi berbeda untuk ADT yang sama.

Ciri-ciri dari ADT adalah:

- Berisi struktur data dan operasi-operasi terhadap struktur data tersebut
- Menyediakan pengkapsulan
- Menyediakan information hiding
- Menyediakan abstraksi
- Tidak menspesifikasikan implementasi struktur data
- Menspesifikasikan perilaku dari ADT

Kegunaan ADT antara lain:

- ADT menyediakan dasar untuk modularitas perangkat lunak
- Mengidentifikasi setiap modul dengan implementasi ADT, yaitu deskripsi sekumpulan objek dengan antarmuka bersama
- Antarmuka didefinisikan oleh sekumpulan operasi yang dibatasi oleh properti-properti yang abstrak
- Masing-masing operasi diimplementasikan menggunakan satu representasi dari ADT

Terdapat tiga komponen dalam implementasi ADT, yakni:

- Spesifikasi ADT berisi fungsi-fungsi, aksioma-aksioma, dan prakondisi-prakondisi
- Pemilihan representasi bagi ADT
- Sekumpulan subprogram, masing-masing mengimplementasikan salah satu fungsi pada spesifikasi ADT yang beroperasi pada representasi yang telah dipilih

6.5 Analisis Statik

Analisis statik merupakan proses menganalisis kode program yang dilakukan tanpa mengeksekusi kode program tersebut, berbeda dengan analisis dinamis di mana kode program dianalisis dengan mengeksekusi kode programnya. Terdapat beberapa alasan melakukan analisis statik antara lain:

- Analisis statik dapat dilakukan pada tahap awal pengkodean, tidak perlu menunggu sampai kode selesai dibuat
- Beberapa jenis kesalahan sulit untuk ditemukan melalui proses pengujian, misalkan kesalahan yang berkaitan dengan timing
- Proses pengujian dan analisis pada dasarnya bersifat komplemen, saling melengkapi satu sama lain

Metode formal digunakan untuk melakukan analisis statik dengan menggunakan serangkaian metode matematis. Teknik matematika yang digunakan antara lain semantik detonasional, semantik aksiomatik, semantik operasional, dan interpretasi abstrak. Salah satu implementasi dari teknik analisis statik adalah Data Flow Analysis

6.5.1 Data Flow Analysis

Data flow analysis merupakan sebuah teknik untuk memperoleh informasi tentang sekumpulan nilai yang mungkin dihitung pada bagian tertentu pada program. Sebuah program Control Flow Graph (CFG) digunakan untuk menentukan bagian dari program di mana ketika nilai tertentu diberikan kepada variabel, maka kemungkinan terjadi propagasi. Informasi yang diperoleh seringkali digunakan oleh compiler untuk proses optimasi program.

Cara yang mudah untuk melakukan Data Flow Analysis adalah dengan menyediakan persamaan data flow untuk setiap node pada CFG dan menyelesaikannya dengan cara menghitung output dari input secara berulang pada setiap node secara lokal hingga keseluruhan sistem stabil.

Persamaan data flow berbentuk:

$$out_b = trans_b(in_b)$$

$$in_b = join_{p \in pred_b}(out_p)$$

di mana $trans_b$ merupakan fungsi transfer pada blok b . Operasi join menggabungkan exit state dari predesesor $p \in pred_b$, dan menjadi entri state untuk b .

Setelah menyelesaikan persamaan ini, entri/exit state dari blok dapat digunakan untuk menjalankan properti program pada batasan blok. Persamaan ini kemudian diselesaikan dengan cara iteratif hingga keseluruhan blok terselesaikan persamaannya.

Ada dua pendekatan dari Data Flow Analysis, yakni:

- Forward Analysis
 - Disebut juga sebagai Available Variable Analysis
 - Sebuah definisi disebut berguna jika dapat mempengaruhi komputasi pada lokasi yang bersangkutan
 - Variabel yang tidak diinisialisasi mengindikasikan kesalahan
 - Contoh penggunaannya:


```

1: if b==4 then
2:   a = 5;
3: else
4:   a = 3;
5: endif
6:
7: if a < 4 then
```

Definisi variabel a pada line 7 adalah sekumpulan nilai $a=5$ pada line 2 dan $a=3$ pada line 4

- Backward Analysis
 - Disebut juga sebagai Like Variable Analysis
 - Sebuah variabel disebut hidup jika nilainya saat ini digunakan untuk proses berikutnya
 - Dead assignment mungkin mengindikasikan kesalahan
 - Contoh penggunaannya:

```
// out: {}  
b1: a = 3;  
    b = 5;  
    d = 4;  
    if a > b then  
// in: {a,b,d}  
  
// out: {a,b}  
b2: c = a + b;  
    d = 2;  
// in: {b,d}  
// out: {b,d}  
b3: endif  
    c = 4;  
    return b * d + c;  
// in: {}
```

Latihan

1. Jelaskan aktivitas-aktivitas yang dilakukan pada tahapan implementasi!
2. Jelaskan perbedaan pendekatan penyelesaian masalah secara Top-Down dan Bottom-Up!
3. Jelaskan perbedaan kohesi dan coupling!
4. Berikan contoh penggunaan dari Data Flow Analysis untuk studi kasus algoritma pencarian nilai rata-rata dari tiga buah bilangan!

7 Pengujian Perangkat Lunak

7.1 Dasar-Dasar Pengujian Perangkat Lunak

Pada proses RPL, pelaku RPL mula-mula berusaha untuk membangun perangkat lunak mulai dari konsep abstrak sampai kepada tahap implementasi yang dapat dilihat, baru kemudian dilakukan pengujian.

Pada pengujian perangkat lunak, pelaku RPL menciptakan sekumpulan kasus uji untuk diujikan kepada perangkat lunak. Proses ini lebih terkesan berusaha untuk “membongkar” perangkat lunak yang sudah dibangun. Proses pengujian merupakan tahapan dalam RPL di mana secara fisik terlihat lebih banyak sisi deskriptifnya dibandingkan sisi konstruktifnya karena tujuannya adalah untuk menemukan kesalahan pada perangkat lunak.

7.1.1 Sasaran Pengujian Perangkat Lunak

Sasaran pengujian perangkat lunak antara lain:

- Pengujian adalah proses mengeksekusi program dengan tujuan khusus untuk menemukan kerusakan
- Kasus uji yang baik adalah yang memiliki tingkat kemungkinan tinggi untuk menemukan kerusakan yang belum ditemukan
- Pengujian dikatakan berhasil jika berhasil menemukan kerusakan yang belum ditemukan

Sasaran di atas sekaligus mengimplikasikan adanya perubahan cara pandang di mana sebelumnya dikatakan bahwa pengujian yang berhasil adalah pengujian yang tidak menemukan kesalahan.

Jika pengujian sukses dilakukan, maka akan ditemukan kesalahan dalam perangkat lunak. Sebagai keuntungan tambahan, pengujian menunjukkan bahwa fungsi perangkat lunak bekerja sesuai spesifikasi dan bahwa persyaratan kinerja telah dipenuhi.

Data yang dikumpulkan pada saat pengujian dilakukan memberikan indikasi yang baik mengenai realibilitas perangkat lunak dan beberapa indikasi dari kualitas keseluruhan perangkat lunak. Akan tetapi, pengujian tidak dapat memperlihatkan bahwa perangkat lunak yang diuji tidak memiliki cacat.

7.1.2 Prinsip Pengujian Perangkat Lunak

Serangkaian prinsip pengujian perangkat lunak antara lain:

- **Semua pengujian harus dapat ditelusuri sampai ke kebutuhan pelanggan.** Sasaran pengujian perangkat lunak adalah untuk menemukan kesalahan. Hal ini memenuhi kriteria bahwa cacat yang paling fatal dilihat dari sisi pandang pelanggan adalah cacat yang mengakibatkan program gagal memenuhi persyaratannya
- **Pengujian harus direncanakan lama sebelum pengujian dimulai.** Perencanaan pengujian dapat dimulai segera setelah model kebutuhan dilengkapi. Definisi detail kasus uji dapat dibuat segera setelah model perancangan dibuat. Dengan demikian, semua pengujian dapat direncanakan dan dirancang sebelum semua kode dibangkitkan
- **Prinsip Pareto berlaku untuk pengujian perangkat lunak.** Prinsip Pareto mengimplikasikan bahwa 80% dari semua kesalahan yang ditemukan selama pengujian akan dapat ditelusuri sampai 20% dari modul program. Permasalahannya adalah bagaimana mengisolasi modul yang dicurigai dan mengujinya dengan teliti
- **Pengujian harus mulai “dari yang kecil” dan berkembang ke pengujian “yang besar”.** Pengujian pertama kali dilakukan fokus pada modul individual perangkat lunak, kemudian pengujian mengubah fokus menjadi menemukan kesalahan pada cluster modul yang terintegrasi, dan akhirnya pada sistem secara keseluruhan
- **Tidak mungkin melakukan pengujian yang mendalam.** Jumlah jalur permutasi untuk program yang berukuran menengah sangat besar. Karena itulah, tidak mungkin untuk mengeksekusi setiap kombinasi jalur skema pengujian. Tetapi dimungkinkan untuk secara memadai mencakup logika program dan memastikan bahwa semua kondisi dalam rancangan prosedural telah diuji
- **Agar memperoleh pengujian yang paling efektif, pengujian harus dilakukan oleh pihak ketiga yang independen.** Maksudnya, agar pengujian memiliki tingkat kemungkinan yang tinggi untuk menemukan kesalahan, maka pelaku RPL yang membuat sistem tersebut bukanlah orang yang paling tepat untuk melakukan semua pengujian bagi perangkat lunak

7.1.3 Testabilitas

Testabilitas perangkat lunak adalah seberapa mudah sebuah perangkat lunak dapat diuji. Karena pengujian merupakan proses yang sangat sulit, perlu diketahui apa saja yang dapat dilakukan untuk membuatnya menjadi mudah. Salah satu caranya adalah dengan menyediakan checklist mengenai masalah-masalah desain yang mungkin, fitur, dan lain sebagainya yang dapat membantu untuk bernegosiasi dengan pemrogram. Checklist berikut memberikan serangkaian karakteristik sebuah perangkat lunak dapat diuji:

1. **Operabilitas.** “Semakin baik dia bekerja, semakin efisien dia dapat diuji.”
 - Sistem memiliki sedikit bug (bug menambah analisis dan biaya pelaporan ke proses pengujian)
 - Tidak ada bug yang memblokir eksekusi program
 - Produk berubah pada tahapan fungsional (memungkinkan pengembangan dan pengujian secara simultan)
2. **Observabilitas.** “Apa yang Anda lihat adalah apa yang Anda uji.”
 - Output yang berbeda dihasilkan oleh masing-masing input
 - Status dan variabel sistem dapat diamati selama eksekusi
 - Status dan variabel sistem di masa lalu dapat diamati (mis. transaction log)
 - Semua faktor yang mempengaruhi output dapat diamati
 - Output yang salah dapat diidentifikasi dengan mudah
 - Kesalahan internal dapat dideteksi secara otomatis melalui mekanisme self-testing
 - Kesalahan internal dilaporkan secara otomatis
 - Source code dapat diakses
3. **Kontrolabilitas.** “Semakin baik perangkat lunak dapat dikontrol, semakin banyak pengujian yang dapat diotomatisasi dan dioptimalkan”
 - Semua output yang mungkin dapat dimunculkan melalui beberapa kombinasi input
 - Semua kode dapat dieksekusi melalui berbagai kombinasi input
 - Keadaan dan variabel perangkat lunak dan perangkat keras dapat dikontrol secara langsung oleh perekayasa pengujian
 - Format input dan output konsisten dan terstruktur
 - Pengujian dapat dispesifikasi, dioptimasi, dan diproduksi ulang dengan baik
4. **Dekomposabilitas.** “Dengan mengontrol ruang lingkup pengujian, kita dapat lebih cepat mengisolasi masalah dan melakukan pengujian kembali dengan lebih baik”
 - Sistem perangkat lunak dibangun dari modul-modul yang independen
 - Modul-modul perangkat lunak dapat diuji secara independen
5. **Kesederhanaan.** “Semakin sedikit yang perlu diuji, semakin cepat pengujiannya”
 - Kesederhanaan pengujian (contohnya kumpulan fitur adalah kebutuhan minimum untuk memenuhi persyaratan)
 - Kesederhanaan struktural (contohnya arsitektur dimodularisasi untuk membatasi propagasi kesalahan)
 - Kesederhanaan kode (contohnya sebuah standar pengkodean diadopsi untuk kemudahan inspeksi dan pemeliharaan)
6. **Stabilitas.** “Semakin sedikit perubahan, semakin sedikit gangguan dalam pengujian.”
 - Perubahan pada perangkat lunak jarang
 - Perubahan pada perangkat lunak dapat dikontrol
 - Perubahan pada perangkat lunak tidak membuat pengujian yang telah ada menjadi tidak valid
 - Perangkat lunak dapat pulih dengan baik dari kerusakan
7. **Kemampuan untuk dapat dipahami.** “Semakin banyak informasi yang dimiliki, semakin baik pengujiannya”
 - Rancangan dapat dipahami dengan baik
 - Ketergantungan antara komponen internal, eksternal, dan yang dipakai bersama dapat dipahami dengan baik
 - Perubahan terhadap rancangan dikomunikasikan
 - Dokumen teknis dapat diakses secara cepat
 - Dokumen teknis diorganisasikan dengan baik
 - Dokumen teknis bersifat spesifik dan detail
 - Dokumen teknis bersifat akurat

Sementara itu, pengujian yang “baik” dapat dilihat dari atribut-atribut berikut:

1. **Pengujian yang baik memiliki probabilitas yang tinggi untuk menemukan kesalahan.** Untuk mencapai hal ini, penguji harus memahami perangkat lunak dan berusaha mengembangkan gambaran mengenai bagaimana perangkat lunak dapat gagal. Kemudian kegagalan-kegagalan tersebut diselidiki
2. **Pengujian yang baik tidak redundan.** Waktu dan sumber daya yang tersedia untuk pengujian terbatas. Tidak ada gunanya melakukan pengujian dengan tujuan yang sama dengan pengujian yang telah dilakukan sebelumnya. Setiap pengujian harus memiliki tujuan yang berbeda

3. **Pengujian yang baik seharusnya “jenis terbaik”.** Untuk pengujian-pengujian yang memiliki tujuan serupa, batasan waktu dan sumber daya dapat menghalangi eksekusi kelompok pengujian tersebut. Pada kasus semacam ini, maka pengujian yang memiliki kemungkinan paling besar untuk mengungkap seluruh kesalahan yang harus digunakan
4. **Pengujian yang baik tidak boleh terlalu sederhana atau terlalu kompleks.** Meskipun kadang-kadang mungkin untuk menggabungkan serangkaian pengujian ke dalam satu kasus uji, namun secara umum masing-masing kasus uji harus dieksekusi secara terpisah

7.2 Perancangan Kasus Uji

Saat ini sudah banyak berkembang berbagai metode untuk pengujian perangkat lunak. Metode-metode tersebut memberikan pendekatan yang sistematis untuk pengujian perangkat lunak kepada pengembang. Selain itu, metode-metode tersebut memberikan mekanisme yang dapat membantu memastikan kelengkapan pengujian dan memberikan kemungkinan tertinggi untuk mengungkap kesalahan pada perangkat lunak.

Semua produk yang direkayasa dapat diuji dengan satu atau dua cara, yaitu:

1. Dengan mengetahui fungsi yang ditentukan untuk dilakukan oleh suatu produk, pengujian dapat dilakukan untuk memperlihatkan bahwa masing-masing fungsi beroperasi sepenuhnya dan pada waktu yang sama mencari kesalahan pada setiap fungsi
2. Dengan mengetahui kerja internal suatu produk, maka pengujian dapat dilakukan untuk memastikan bahwa seluruh operasi internal bekerja sesuai spesifikasi dan semua komponen internal telah diamati dengan memadai

Pendekatan pengujian pertama disebut sebagai pengujian black-box dan pengujian kedua disebut sebagai pengujian white-box.

Pengujian black-box berkaitan dengan pengujian yang dilakukan pada antarmuka perangkat lunak. Meskipun dirancang untuk mengungkap kesalahan, pengujian black-box digunakan untuk memperlihatkan bahwa fungsi-fungsi perangkat lunak dapat beroperasi, bahwa input diterima dengan baik dan output dihasilkan dengan tepat, dan integritas informasi eksternal (seperti file data) dipelihara. Pengujian black-box menguji beberapa aspek dasar suatu sistem dengan memperhatikan sedikit struktur logika internal perangkat lunak tersebut.

Pengujian white-box didasarkan pada pengamatan yang teliti terhadap detail prosedural. Jalur-jalur logika yang melewati perangkat lunak diuji dengan memberikan kasus uji yang menguji serangkaian kondisi dan atau loop tertentu. Status program tersebut dapat diuji pada berbagai titik untuk menentukan apakah status yang diharapkan sesuai dengan status yang sebenarnya.

Sekilas terlihat bahwa pengujian white-box yang sangat teliti akan membawa kepada program yang benar 100%. Yang diperlukan adalah menentukan semua jalur logika, mengembangkan kasus uji untuk mengujinya, dan mengevaluasi hasil, yaitu memunculkan kasus uji untuk menguji logika program secara mendalam. Namun sesuai dengan prinsip pengujian, pengujian secara mendalam akan menimbulkan masalah logistik. Bahkan untuk program yang kecil, dapat dibayangkan jumlah jalur logika yang besar.

Tetapi pengujian white-box tidak boleh dianggap tidak praktis. Sejumlah jalur logika yang penting dapat dipilih dan digunakan. Struktur-struktur data yang penting dapat diperiksa validitasnya. Atribut pengujian black-box dan white-box dapat digabungkan untuk memberikan pendekatan yang memvalidasi antarmuka perangkat lunak, dan secara selektif menjamin bahwa kerja internal perangkat lunak itu benar.

7.2.1 White-Box Testing

White-box testing (kadang-kadang disebut sebagai glass-box testing) adalah metode desain kasus uji yang menggunakan struktur kontrol desain prosedural untuk memperoleh kasus uji. Dengan menggunakan metode pengujian white-box, perancang sistem dapat melakukan kasus uji yang:

1. Memberikan jaminan bahwa semua jalur independen pada suatu modul telah digunakan paling tidak satu kali
2. Menggunakan semua keputusan logis pada sisi true dan false
3. Mengeksekusi semua loop pada batasan mereka dan pada batas operasional mereka
4. Menggunakan struktur data internal untuk menjamin validitasnya

Muncul pertanyaan tentang mengapa menghabiskan waktu dan sumber daya untuk menguji logika jika dapat memastikan bahwa persyaratan program telah dapat dipenuhi dengan lebih baik. Jawaban dari pertanyaan ini ada pada sifat cacat perangkat lunak seperti:

- **Kesalahan logis dan asumsi yang tidak benar berbanding terbalik dengan probabilitas jalur program yang akan dieksekusi.** Kesalahan cenderung muncul pada saat perancangan dan implementasi fungsi, kondisi, atau kontrol yang berada di luar mainstream
- **Kepercayaan bahwa jalur logika mungkin tidak akan dieksekusi bila pada kenyataannya mungkin dieksekusi pada basis reguler.** Aliran logika dari program kadang bersifat konterintuitif, artinya asumsi yang tidak disadari mengenai aliran dan data kontrol dapat menyebabkan kesalahan perancangan yang akan terungkap setelah pengujian jalur dimulai

- **Kesalahan tipografi sifatnya acak.** Bila sebuah program diterjemahkan ke dalam source code bahasa pemrograman, maka dimungkinkan akan terjadi banyak kesalahan pengetikan. Beberapa kesalahan dapat ditemukan melalui mekanisme pengecekan sintaks, namun yang lainnya tidak akan terdeteksi sampai dilakukan pengujian

Sifat cacat tersebut sangat mungkin ditemukan dengan menggunakan pengujian white-box sedangkan pengujian black-box tidak dapat menemukannya seberapa cermat pun pengujian black-box dilakukan. Alasan inilah yang mendasari mengapa pengujian white-box dilakukan.

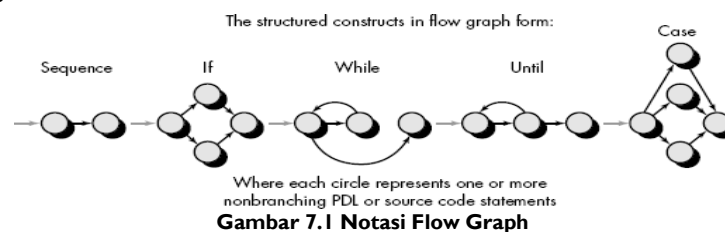
Jenis pengujian White-Box testing antara lain:

- Basis path testing
- Control Structure Testing, yang terdiri dari:
 - Condition Testing
 - Data Flow Testing
 - Loop Testing

7.2.1.1 Basis Path Testing

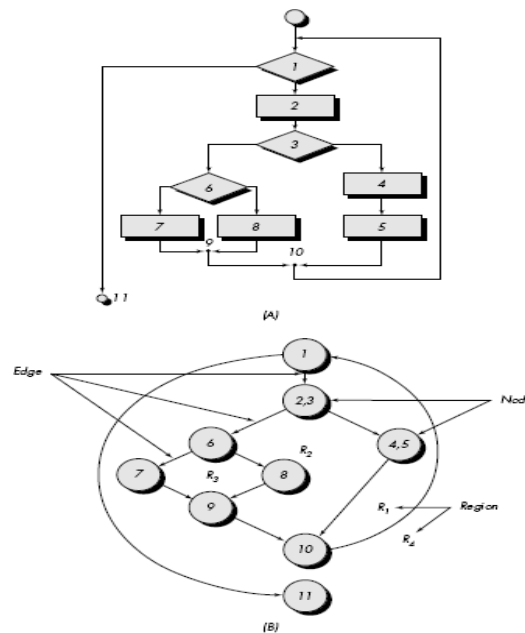
Teknik pengujian ini pertama kali diperkenalkan oleh Tom McCabe. Metode ini memungkinkan perancang kasus uji untuk mengukur kompleksitas logis dari rancangan prosedural dan menggunakannya sebagai pedoman untuk menetapkan sekumpulan jalur eksekusi dasar (basis set). Kasus uji yang dilakukan untuk menggunakan basis set tersebut dijamin untuk menggunakan setiap statement dalam program paling tidak sekali selama pengujian.

Metode ini menggunakan notasi flow graph yang menggambarkan aliran kontrol logika yang menggunakan notasi yang ditunjukkan pada gambar di bawah ini:



Gambar 7.1 Notasi Flow Graph

Contoh penggunaannya dapat dilihat pada gambar 7.2a yang merepresentasikan desain prosedural menggunakan flowchart, dan gambar 7.2b merupakan hasil pemetaan desain prosedural tersebut ke dalam notasi flow graph (dengan mengasumsikan bahwa tidak ada kondisi yang kompleks dalam notasi percabangan pada flowchart). Setiap lingkaran pada flow graph, disebut sebagai flow graph node merepresentasikan satu atau lebih statemen prosedural. Urutan kotak proses dan notasi percabangan dapat dipetakan menjadi satu node. Node yang berisi sebuah kondisi disebut sebagai node predikat dan ditandai dengan dua atau lebih edge yang berasal darinya. Anak panah pada flow graph disebut edges atau links, yang merepresentasikan aliran kontrol dan sesuai dengan anak panah pada flowchart. Edge harus berhenti pada suatu node, meskipun node tersebut tidak merepresentasikan statemen prosedural. Area yang dibatasi oleh edge dan node disebut sebagai region. Pada saat menghitung region, kita memasukkan area di luar graph sebagai sebuah region juga.



Gambar 7.2 (A) flowchart, (B) flow graph

Untuk menentukan jalur independen dapat digunakan nilai Kompleksitas Siklomatis. Kompleksitas siklomatis adalah metrik perangkat lunak yang merupakan ukuran kuantitatif terhadap kompleksitas logika suatu program. Bila metrik ini digunakan untuk Basis Path Testing, maka nilai yang terhitung untuk kompleksitas siklomatis menentukan jumlah jalur independen dalam basis set suatu program dan memberikan batas atas bagi jumlah pengujian yang harus dilakukan untuk memastikan bahwa semua statemen telah dieksekusi sedikitnya satu kali.

Jalur independen adalah jalur yang melalui program yang menghasilkan sedikitnya satu rangkaian statemen proses baru atau suatu kondisi baru. Bila dinyatakan dengan terminologi flow graph, jalur independen bergerak sepanjang paling tidak satu edge yang tidak dilewatkan sebelum jalur tersebut ditentukan. Sebagai contoh, serangkaian jalur independen untuk flow graph pada gambar 7.2b adalah:

Jalur 1 : 1-11

Jalur 2 : 1-2-3-4-5-10-11

Jalur 3 : 1-2-3-6-8-9-10-11

Jalur 4 : 1-2-3-6-7-9-10-11

Perhatikan bahwa masing-masing jalur baru memperkenalkan sebuah edge baru.

Jalur 1-2-3-4-5-10-11-2-3-6-8-9-10-11

tidak dianggap jalur independen karena merupakan gabungan dari jalur-jalur yang telah ditentukan dan tidak melewati edge baru.

Jalur 1, 2, 3, dan 4 yang ditentukan di atas terdiri dari sebuah basis set untuk flow graph pada gambar 11.2b. Bila pengujian dapat dieksekusi pada jalur-jalur tersebut, maka setiap statemen pada program tersebut akan dieksekusi minimal sekali dan setiap kondisi telah dieksekusi pada sisi true dan false-nya.

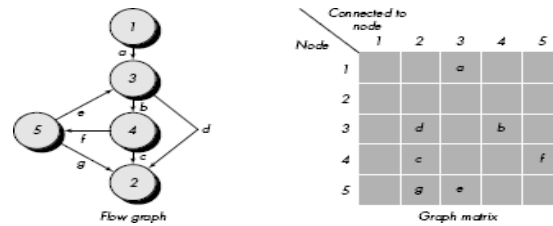
Kompleksitas siklomatis dihitung menggunakan salah satu dari ketiga cara berikut ini:

1. Jumlah region flow graph
2. Kompleksitas siklomatis $V(G)$ untuk flow graph G dihitung sebagai $V(G) = E - N + 2$ di mana E adalah jumlah edge dan N adalah jumlah node
3. $V(G) = P + 1$ di mana P adalah jumlah node predikat pada flow graph G

Untuk mengembangkan piranti perangkat lunak yang membantu Basis Path Testing, dapat digunakan struktur data berbentuk matriks yang disebut sebagai matriks graph.

Matriks graph merupakan matriks bujur sangkar yang ukurannya adalah sesuai dengan jumlah node pada flow graph. Masing-masing baris dan kolom sesuai dengan node masing-masing, dan entri matriks sesuai dengan edge di antara simpul.

Contoh sederhananya adalah sebagai berikut:



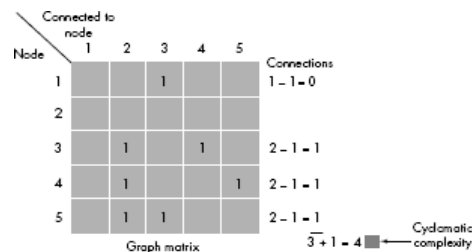
Gambar 7.3 Matriks graph

Pada gambar tersebut, terlihat bahwa masing-masing isi matriks merepresentasikan edge antara kedua node, misalkan dari node 3 ke node 4 dihubungkan oleh edge b.

Jika isi dari matriks diganti dengan link weight, maka akan diperoleh sebuah metode yang ampuh untuk mengevaluasi struktur kontrol program selama pengujian. Link weight dapat berupa:

- Probabilitas sebuah edge akan dieksekusi
- Waktu pemrosesan yang digunakan selama melewati suatu link
- Memori yang diperlukan selama melewati suatu link
- Sumber daya yang diperlukan untuk melewati suatu link

Jika link weight direpresentasikan dengan angka 1 dan 0 di mana angka 1 jika ada link dan 0 jika tidak (matriks dengan bentuk seperti ini sering disebut matriks koneksi), maka akan didapatkan metode lain untuk menghitung kompleksitas siklomatis.



Gambar 7.4 Matriks koneksi

7.2.2 Black-Box Testing

Pengujian ini fokus kepada persyaratan fungsional perangkat lunak. Pengujian ini memungkinkan pelaku RPL mendapatkan serangkaian kondisi input yang memenuhi persyaratan fungsional suatu program.

Pengujian ini berusaha menemukan kesalahan dengan kategori sebagai berikut:

1. Fungsi-fungsi yang salah atau hilang
2. Kesalahan antarmuka
3. Kesalahan struktur data atau akses basisdata eksternal
4. Kesalahan kinerja
5. Kesalahan inisialisasi atau terminasi

Pengujian ini cenderung untuk dilakukan pada tahap akhir pengujian berbeda dengan white-box testing. Penguji dituntut untuk menjawab pertanyaan-pertanyaan berikut:

- Bagaimana validitas fungsional diuji?
- Kelas input apa yang akan membuat kasus uji menjadi baik?
- Apakah sistem sangat sensitif terhadap nilai input tertentu?
- Bagaimana batasan suatu data diisolasi?
- Berapa kecepatan dan volume data yang dapat ditolerir sistem?
- Apa pengaruh kombinasi tertentu dari data terhadap operasi sistem?

Dengan mengaplikasikan teknik pengujian ini, penguji membuat serangkaian kasus uji yang:

- Mengurangi jumlah kasus uji tambahan yang harus dirancang untuk mencapai pengujian yang benar
- Memberi tahu mengenai ada atau tidaknya kesalahan

Contoh pengujian Black-Box testing antara lain:

- Graph Based Testing Method
- Equivalence Partitioning
- Boundary Value Analysis
- Comparison Testing
- Orthogonal Array Testing

7.2.2.1 Boundary Value Analysis

Merupakan teknik pengujian yang membagi domain-domain input dari suatu program ke dalam kelompok-kelompok data, kemudian melakukan pengujian hanya pada batas-batas domain input tersebut. Metode ini merupakan pengembangan dari metode sebelumnya, Equivalence Partitioning, yang hanya membagi domain input, namun melakukan pengujiannya bukan pada nilai batasnya.

Pedoman untuk BVA:

- Bila suatu kondisi input mengkhususkan suatu range dibatasi oleh nilai a dan b, maka kasus uji harus dirancang dengan nilai input a, b, persis di atas a, persis di bawah a, persis di atas b, dan persis di bawah b
- Bila suatu kondisi input mengkhususkan sejumlah nilai, maka kasus uji harus dikembangkan dengan menggunakan jumlah minimum dan maksimum. Nilai tepat di atas dan di bawah minimum dan maksimum juga diuji
- Pedoman 1 dan 2 diaplikasikan ke kondisi output. Kasus uji harus dirancang agar menghasilkan output maksimum dan minimum dari program
- Bila struktur data memiliki batasan, maka kasus uji harus dirancang sesuai batasan tersebut

7.3 Pendekatan Strategis untuk Pengujian Perangkat Lunak

Pengujian merupakan serangkaian aktivitas yang dapat direncanakan sebelumnya dan dilakukan secara sistematis. Banyak ditemukan strategi pengujian perangkat lunak yang memberikan sebuah template untuk pengujian yang memiliki karakteristik sebagai berikut:

- Pengujian dimulai pada level komponen dan dilanjutkan dengan menguji pada level integrasi keseluruhan sistem berbasis komputer.
- Teknik pengujian yang berbeda dibutuhkan pada saat yang berbeda.
- Pengembang perangkat lunak melakukan pengujian dan mungkin dibantu oleh kelompok penguji yang independen untuk keperluan proyek-proyek besar.
- Pengujian (Testing) dan debugging adalah aktivitas yang berbeda.
- Debugging harus dapat dilakukan pada strategi pengujian apapun.

7.3.1 Verifikasi dan Validasi

Verifikasi dan validasi merupakan dua istilah yang sering dikaitkan dengan tahapan pengujian perangkat lunak. Verifikasi mengacu pada serangkaian aktivitas untuk memastikan bahwa perangkat lunak mengimplementasikan fungsi tertentu secara benar, sedangkan validasi mengacu pada serangkaian aktivitas untuk memastikan bahwa perangkat lunak yang telah dibuat sesuai dengan kebutuhan konsumen.

Definisi V&V mencakup serangkaian aktivitas dari penjaminan kualitas perangkat lunak (SQA) yang meliputi kajian teknis formal, audit kualitas dan kontrol, monitoring kinerja, simulasi, studi feasibilitas, kajian dokumentasi, kajian basisdata, analisis algoritma, pengujian pengembangan, pengujian kualifikasi, dan pengujian instalasi.

7.3.2 Pengorganisasian Pengujian Perangkat Lunak

Proses pengujian sebuah perangkat lunak sebaiknya melibatkan pihak yang memang secara khusus bertanggung jawab untuk melakukan proses pengujian secara independen. Untuk itulah diperlukan Independent Test Group (ITG).

Peran dari ITG adalah untuk menghilangkan “conflict of interest” yang terjadi ketika pengembang perangkat lunak berusaha untuk menguji produknya sendiri.

Walaupun seperti itu, sering terjadi beberapa kesalahan pemahaman berkaitan dengan peran ITG, antara lain:

- Pengembang tidak boleh melakukan pengujian sama sekali. Pendapat ini tidak 100% benar, karena dalam banyak kasus, pengembang juga melakukan proses unit testing dan integration test
- Perangkat lunak dilempar begitu saja untuk diuji secara sporadis. Hal tersebut adalah salah karena pengembang dan ITG bekerja sama pada keseluruhan proyek untuk memastikan pengujian akan dilakukan. Sementara pengujian dilakukan, pengembang harus memperbaiki kesalahan yang ditemukan
- Penguji tidak terlibat pada proyek sampai tahap pengujian dimulai. Hal tersebut salah karena ITG merupakan bagian dari tim proyek pengembangan perangkat lunak di mana ia terlibat selama spesifikasi proses dan tetap terlibat pada keseluruhan proyek besar

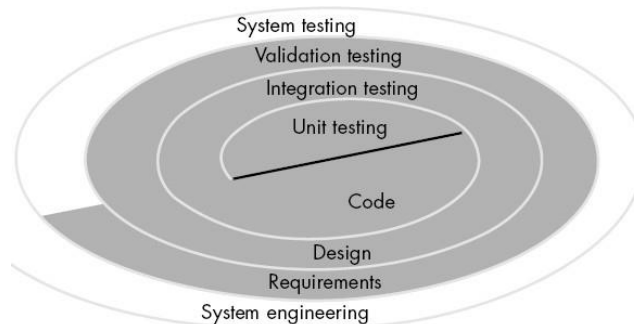
7.4 Masalah Strategis Pengujian

Masalah-masalah berikut harus diselesaikan bila pengujian ingin berlangsung sukses:

- Menspesifikasikan kebutuhan produk pada kelakuan yang terukur sebelum pengujian dimulai. Strategi pengujian yang baik tidak hanya untuk menemukan kesalahan, namun juga untuk menilai kualitas program
- Menspesifikasikan tujuan pengujian secara eksplisit. Sasaran spesifik dari pengujian harus dinyatakan dalam bentuk yang terukur
- Mengidentifikasi kategori user untuk perangkat lunak dan membuat profilnya masing-masing. Beberapa kasus yang menggambarkan skenario interaksi bagi masing-masing kategori dapat mengurangi kerja pengujian dengan memfokuskan pengujian pada penggunaan aktual produk
- Membangun rencana pengujian yang menegaskan rapid cycle testing. Umpan balik yang muncul dari rapid cycle testing dapat digunakan untuk mengontrol kualitas dan strategi pengujian yang sesuai
- Membangun perangkat lunak yang tangguh yang dirancang untuk menguji dirinya sendiri. Perangkat lunak dirancang dengan teknik antirunning, di mana perangkat lunak dapat mendiagnosis jenis-jenis kesalahan tertentu dan mengakomodasi pengujian otomatis dan pengujian regresi
- Menggunakan tinjauan formal yang efektif sebagai filter sebelum pengujian. kajian teknis formal dapat mengungkap kesalahan seefektif pengujian sehingga dapat mengurangi jumlah kerja pengujian
- Mengadakan tinjauan formal teknis untuk menilai strategi dan kasus uji. Kajian teknis formal dapat mengungkap inkonsistensi, penghapusan, dan kesalahan seketika dalam pendekatan pengujian
- Membangun pendekatan yang meningkat secara berkelanjutan untuk proses pengujian. Strategi pengujian harus terukur. Metrik yang terkumpul selama pengujian harus digunakan sebagai bagian dari pendekatan kontrol proses statistik bagi pengujian perangkat lunak

7.5 Strategi Pengujian Perangkat Lunak

Proses pengujian perangkat lunak dapat digambarkan sebagai berikut:

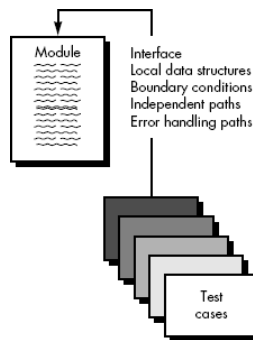


Gambar 7.5 Strategi Pengujian Perangkat Lunak

Spiral di atas menggambarkan bagaimana rekayasa sistem membangun sebuah perangkat lunak dimulai dari penentuan kebutuhan perangkat lunak, kemudian (spiral bergerak ke dalam) proses dilanjutkan ke dalam bentuk rancangan, dan akhirnya ke pengkodean (di pusat spiral). Strategi pengujian serupa dengan hal tersebut, namun bergerak dimulai dari pusat menuju keluar spiral. Dimulai dengan unit testing di pusat spiral di mana masing-masing modul/unit dari perangkat lunak yang diimplementasikan dalam source code menjadi sasaran pengujian. Kemudian bergerak keluar spiral, dilakukan integration testing dengan fokus pengujian adalah desain dan konstruksi arsitektur perangkat lunak. Bergerak lagi keluar, dilakukan validation testing dengan sasaran pengujian adalah kesesuaian dengan kebutuhan perangkat lunak yang telah ditentukan di awal. Terakhir pada lingkaran terluar spiral sampai pada system testing, di mana perangkat lunak dan keseluruhan elemen sistem diuji.

7.5.1 Unit Testing

Berfokus pada verifikasi unit terkecil dari perangkat lunak. Dengan menggunakan gambaran desain prosedural, jalur kontrol yang penting diuji untuk mengungkap kesalahan pada modul tersebut. Pengujian ini biasanya berorientasi white-box, dan dapat dilakukan secara paralel untuk modul bertingkat.



Gambar 7.6 Unit Testing

Interface modul diuji untuk memastikan bahwa informasi secara tepat masuk dan keluar dari program yang diuji. Struktur data lokal diuji untuk memastikan bahwa data yang tersimpan secara temporal tetap terjaga integritasnya selama keseluruhan algoritma dieksekusi. Kondisi batas diuji untuk memastikan bahwa modul beroperasi dengan tepat pada batas yang ditentukan untuk membatasi pemrosesan.

Pengujian terhadap jalur eksekusi merupakan tugas penting selama unit testing. Kasus uji harus dirancang untuk mengungkap kesalahan yang berkaitan dengan kesalahan komputasi, kesalahan komparasi, dan aliran kontrol yang tidak tepat. Basis path testing dan loop testing merupakan teknik yang efektif untuk mengungkap kesalahan tersebut.

Prosedur unit testing biasanya melibatkan adanya driver dan stub. Driver adalah suatu program utama yang menerima data kasus uji, melewati data tersebut ke modul yang akan diuji, kemudian menampilkan hasilnya. Sedangkan stub adalah subprogram dummy yang berfungsi untuk menggantikan modul yang merupakan subordinat dari modul yang akan diuji.

7.5.2 Integration Testing

Merupakan teknik sistematis untuk mengkonstruksi struktur program sambil melakukan pengujian untuk mengungkap kesalahan berkaitan dengan interfacing. Sasarannya adalah modul yang telah diuji dengan unit testing dan konstruksi program dari modul tersebut sesuai rancangan perangkat lunak.

Pengujian ini akan lebih efektif pada integrasi inkremental, karena kesalahan interfacing dapat lebih mudah dideteksi pada modul mana saja yang mengalaminya, berbeda dengan integrasi secara bersamaan langsung di mana modul-modul yang mengalami kesalahan interfacing sulit untuk dideteksi.

Ada dua pendekatan integrasi, yakni:

- Top-down, di mana modul diintegrasikan ke bawah melalui hirarki kontrol dimulai dari modul utama. Subordinat modul selanjutnya digabungkan ke dalam struktur secara depth-first atau breadth-first. Proses integrasi dilakukan dalam lima langkah:
 1. Modul kontrol utama digunakan sebagai test driver dan stub ditambahkan pada semua modul yang secara langsung subordinat terhadap modul kontrol utama
 2. Stub subordinat diganti dengan modul yang sebenarnya
 3. Pengujian dilakukan pada saat masing-masing modul diintegrasikan
 4. Pada saat pengujian hampir berakhir untuk sebuah modul, stub yang lain diganti dengan modul yang sebenarnya
 5. Dilakukan pengujian regresi untuk memastikan bahwa kesalahan baru belum muncul
- Bottom-up, dimulai dari konstruksi dan pengujian modul atomik yang berada pada tingkat paling rendah pada struktur program. Pendekatan ini dapat mengeliminasi peran stub karena pemrosesan untuk modul subordinat selalu tersedia. Langkah-langkahnya sebagai berikut:
 1. Modul tingkat rendah digabungkan ke dalam cluster (build) yang melakukan subfungsi perangkat lunak tertentu
 2. Driver dibuat untuk mengkoordinasi input dan output kasus uji
 3. Cluster diuji
 4. Driver diganti dan cluster digabungkan pada struktur di atasnya pada hirarki program

Setiap kali sebuah modul ditambahkan, maka perangkat lunak akan berubah yang dapat menyebabkan masalah pada fungsi-fungsi yang telah bekerja sebelumnya. Pengujian integrasi merupakan eksekusi ulang dari beberapa subset yang telah dilakukan untuk memastikan bahwa perubahan tidak menimbulkan efek samping yang tidak diinginkan. Pengujian regresi dapat dilakukan secara manual atau dengan menggunakan alat capture playback otomatis.

Pengujian regresi berisi tiga jenis kasus uji yang berbeda, yakni:

- Sampel representatif dari pengujian yang akan menggunakan semua fungsi perangkat lunak
- Pengujian tambahan yang berfokus pada fungsi-fungsi perangkat lunak yang mungkin dipengaruhi oleh perubahan tersebut
- Pengujian yang berfokus pada komponen perangkat lunak yang telah diubah

Ada pendekatan khusus dari integration testing yang sering digunakan untuk perangkat lunak “shrink wrapped” yang disebut dengan Smoke Testing. Pendekatan Smoke Testing mencakup aktivitas berikut ini:

- Komponen perangkat lunak yang telah dikodekan diintegrasikan ke dalam build
- Sekumpulan pengujian dirancang untuk menunjukkan kesalahan yang akan menghalangi build untuk menjalankan fungsinya dengan baik
- Build diintegrasikan dengan build-build lainnya dan keseluruhan produk diuji dengan smoke testing secara rutin per harinya

Pendekatan ini memiliki beberapa keuntungan untuk proyek pengembangan yang kompleks dan bersifat time-critical, yakni:

- Meminimalkan resiko integrasi
- Meningkatkan kualitas produk akhir
- Menyederhanakan diagnosa kesalahan dan koreksi
- Memudahkan penilaian kemajuan pengembangan perangkat lunak

Untuk tiap fase pengujian pada integration testing, digunakan kriteria-kriteria berikut:

- **Integritas antarmuka.** Antarmuka internal dan eksternal diuji pada saat masing-masing modul ditambahkan ke dalam struktur
- **Validitas fungsional.** Pengujian dirancang untuk mengungkap kesalahan fungsional yang dilakukan
- **Isi informasi.** Pengujian dirancang untuk mengungkap kesalahan yang berhubungan dengan struktur data global atau lokal yang digunakan
- **Kinerja.** Pengujian dirancang untuk memeriksa batasan kinerja yang dibuat selama perancangan perangkat lunak

7.5.3 Validation Testing

Validation testing dilakukan setelah perangkat lunak selesai dirangkai sebagai suatu kesatuan dan semua kesalahan interfacing telah ditemukan dan dikoreksi. Validasi dikatakan berhasil jika perangkat lunak berfungsi sesuai dengan kebutuhan konsumen.

Validasi perangkat lunak diperoleh melalui sederetan pengujian Black-Box yang memperlihatkan kesesuaian dengan kebutuhan perangkat lunak. Semua kasus uji dirancang untuk memastikan apakah semua persyaratan fungsional telah dipenuhi, semua persyaratan kinerja tercapai, semua dokumentasi telah benar, dan persyaratan lainnya (transportabilitas, kompatibilitas, kemampuan pulih dari kesalahan, dan maintainabilitas) dipenuhi.

Adalah sangat tidak mungkin bagi pengembang perangkat lunak untuk meramalkan bagaimana konsumen benar-benar menggunakan perangkat lunak. Untuk mengatasi hal ini, dapat dilakukan acceptance testing untuk memungkinkan konsumen memvalidasi semua persyaratan. Dengan adanya penggunaan langsung oleh end-user, acceptance testing dapat mencakup test drive informal sampai deretan pengujian yang dieksekusi secara sistematis dan terencana.

Dua jenis acceptance testing yang biasa dilakukan oleh perusahaan pengembang perangkat lunak, yakni:

1. **Alpha test**, yakni pengujian yang dilakukan pada perangkat lunak oleh end-user dengan adanya supervisi dan kontrol dari pengembang perangkat lunak
2. **Beta test**, yakni pengujian yang dilakukan pada perangkat lunak oleh end-user tanpa adanya supervisi dan kontrol dari pengembang perangkat lunak. Jika ditemukan masalah, maka konsumen pemakai akan melaporkannya kepada pengembang perangkat lunak tersebut

7.5.4 System Testing

Pengujian yang sasaran utamanya adalah pada keseluruhan Sistem Berbasis Komputer, tidak hanya kepada perangkat lunak.

Ada beberapa tipe dari system testing di antaranya:

1. **Recovery testing** merupakan pengujian sistem yang memaksa perangkat lunak untuk gagal dengan berbagai cara dan memeriksa apakah proses perbaikan dilakukan dengan tepat. Bila perbaikannya otomatis, maka inisialisasi ulang, mekanisme checkpoint, perbaikan data, dan restart masing-masing dievaluasi koreksinya. Bila tidak, maka waktu rata-rata perbaikan dievaluasi untuk menentukan apakah masih dapat ditolerir atau tidak
2. **Security testing** berusaha untuk membuktikan apakah mekanisme perlindungan yang dibangun pada sebuah sistem akan benar-benar dapat melindungi dari pengaruh yang salah
3. **Stress testing** dirancang untuk melawan program pada keadaan abnormal. Pengujian ini mengeksekusi sistem dalam kondisi kuantitas sumber daya yang abnormal
4. **Performance testing** dirancang untuk menguji kinerja perangkat lunak yang telah terintegrasi pada sistem pada saat run-time. Pengujian ini sebenarnya terjadi pada setiap tahapan, mulai dari unit testing pada modul, sampai kepada system testing ketika perangkat lunak telah terintegrasi pada sistem. Pengujian ini sering dikolaborasikan dengan stress testing dan menggunakan instrumen perangkat lunak dan perangkat keras untuk mengukur penggunaan sumber daya dengan cara yang tepat sehingga dapat mengungkap situasi yang menyebabkan kegagalan sistem.

7.5.5 Strategi Pengujian Perangkat Lunak Berorientasi Objek

Strategi pengujian untuk perangkat lunak berorientasi objek serupa dengan strategi pengujian yang telah dibahas sebelumnya. Namun terdapat beberapa perbedaan dengan beberapa strategi yang telah dibahas sebelumnya, yakni:

1. Pada unit testing
 - Bagian terkecil yang diuji pada unit testing adalah kelas atau objek, tidak seperti unit testing konvensional yang fokus pada detail algoritmik dari perangkat lunak
 - Tidak menguji operasi yang ada secara terpisah, seperti halnya unit testing konvensional, karena operasi-operasi pada satu kelas diuji bersamaan pada satu kelas
2. Pada integration testing
 - Memiliki dua strategi yakni thread-based testing dan use-based testing
 - Thread-based testing yang mengintegrasikan sekumpulan kelas yang dibutuhkan untuk merespon suatu input atau event pada sistem. Setiap thread diintegrasikan dan diuji secara individual, kemudian dilakukan pengujian regresi untuk memastikan tidak ada efek samping yang muncul
 - Use-based testing yang menguji kelas independen (kelas yang menggunakan sangat sedikit kelas server) kemudian kelas yang menggunakan layanan kelas tersebut, kemudian dilanjutkan sampai keseluruhan perangkat lunak dibangun
 - Tahapan cluster testing di mana sekumpulan kelas yang berkolaborasi diuji untuk menemukan kesalahan pada saat berinteraksi

7.6 Debugging

Debugging bukan merupakan pengujian, namun merupakan konsekuensi dari pengujian yang berhasil. Jika sebuah kasus uji berhasil menemukan kesalahan, maka proses debugging bertujuan untuk menghilangkan kesalahan tersebut.

Debugging merupakan proses yang sulit untuk dilakukan karena adanya beberapa karakteristik bug seperti:

- Gejala dan penyebab dari bug bisa saja sangat jauh, gejala dapat muncul pada bagian tertentu dari program dan penyebabnya bisa saja berada pada bagian lain yang sangat jauh dari tempat munculnya gejala
- Gejala dapat hilang ketika kesalahan yang lain diperbaiki
- Gejala dapat ditimbulkan oleh sesuatu yang tidak salah (mis. pembulatan yang tidak akurat)
- Gejala dapat disebabkan oleh kesalahan manusia yang sulit untuk ditelusuri
- Gejala dapat disebabkan oleh masalah timing
- Kemungkinan sulit untuk memproduksi kondisi input secara akurat
- Gejala dapat terjadi tiba-tiba
- Gejala dapat disebabkan oleh sesuatu yang didistribusikan melewati sejumlah tugas yang bekerja pada prosesor yang berbeda-beda

Terdapat tiga jenis pendekatan debugging antara lain:

1. Brute Force
Merupakan teknik yang paling sering digunakan dan paling tidak efisien dalam mengisolasi penyebab kesalahan. Dengan prinsip “biarkan komputer menemukan kesalahan”, maka seluruh sumber daya komputer digunakan dengan tujuan untuk menemukan penyebab kesalahan
2. Backtracking
Merupakan pendekatan yang dimulai dari penemuan gejala kemudian menelusuri balik hingga ke penyebab
3. Cause Elimination
Dimanifestasikan oleh induksi atau deduksi dan menggunakan konsep partisi biner. Data yang berhubungan dengan kesalahan yang muncul dikumpulkan untuk mengisolasi penyebab. Kemudian dibuat sebuah hipotesis dan data digunakan untuk membuktikan hipotesis tersebut. Daftar serangkaian penyebab yang mungkin dibuat dan dilakukan pengujian untuk mengeliminasi penyebab-penyebab tersebut. Jika pengujian menunjukkan kebenaran hipotesis untuk suatu penyebab, maka data diperbaiki untuk mengisolasi bug

Sekali bug ditemukan, bug harus diperbaiki. Namun, perbaikan pada bug dapat memunculkan kesalahan lain, maka ada beberapa pertimbangan sebelum bug dihilangkan antara lain:

- Apakah penyebab bug ada pada bagian lain dari program?
- Apakah “bug yang lain” mungkin terjadi pada saat perbaikan dilakukan?
- Apakah yang telah dilakukan untuk mencegah bug pada tempat pertama?

Latihan

1. Jelaskan perbedaan antara validasi dan verifikasi!
2. Jelaskan pengaruh dari penjadwalan proyek terhadap integration testing!
3. Siapakah yang sebaiknya melakukan proses ValidationTesting? Jelaskan!
4. Sebutkan dan jelaskan kekurangan dan kelebihan pendekatan Integration Testing top-down dan bottom-up!
5. Jelaskan perbedaan antara Unit Testing pada perangkat lunak terstruktur dan pada perangkat lunak berorientasi objek!
6. Rancanglah sebuah algoritma untuk mencari modus dari sekumpulan data, kemudian lakukan pengujian Basis Path Testing pada algoritma tersebut!
7. Jelaskan tentang metodologi pengujian Boundary Value Analysis pada Black-Box Testing!
8. Sebutkan dan jelaskan faktor-faktor yang memudahkan pengujian perangkat lunak!
9. “Pengujian merupakan anomali dalam aktivitas RPL”. Jelaskan maksud dari pernyataan tersebut!
10. Jelaskan pertimbangan-pertimbangan sebuah bug harus dihilangkan!

8 Pemeliharaan Perangkat Lunak

8.1 Pengertian Pemeliharaan

Pemeliharaan perangkat lunak merupakan proses memodifikasi sistem perangkat lunak atau komponennya setelah penggunaan oleh konsumen untuk memperbaiki kerusakan, meningkatkan kinerja, manfaat, atau kualitas lainnya atau untuk menyesuaikan sistem perangkat lunak dengan lingkungan yang berubah. Definisi ini menegaskan bahwa proses pemeliharaan perangkat lunak merupakan proses yang bersifat post-delivery, artinya dilakukan setelah sistem perangkat lunak digunakan oleh konsumen.

Aktivitas ini dimulai sejak sistem dilepaskan ke pasaran dan digunakan oleh konsumen dan mencakup semua aktivitas yang menjaga operasional sistem dan kesesuaian dengan kebutuhan pengguna.

Sebagian ahli berpendapat tidak demikian. Menurut mereka, pemeliharaan perangkat lunak harus dimulai sebelum operasional sistem berjalan. Schneidewind berpendapat bahwa pandangan tentang pemeliharaan perangkat lunak merupakan aktivitas post-delivery adalah salah satu penyebab mengapa aktivitas pemeliharaan menjadi hal yang sangat sulit dilakukan. Osborne dan Chikofsky berpendapat bahwa penting untuk mengadopsi pendekatan SDLC untuk mengelola dan mengubah sistem perangkat lunak pada tahapan pemeliharaan perangkat lunak.

Pigoski kemudian memberikan definisi baru tentang pemeliharaan, yakni sebuah aktivitas keseluruhan yang dilakukan untuk menyediakan dukungan yang murah dan efektif terhadap sistem perangkat lunak. Aktivitas dapat berupa pre-delivery dan post-delivery. Aktivitas pre-delivery berupa perencanaan untuk operasi post-delivery, suportabilitas, dan penentuan logistik. Aktivitas post-delivery berupa modifikasi perangkat lunak, pelatihan, dan mengoperasikan help desk.

8.2 Kategori Pemeliharaan Perangkat Lunak

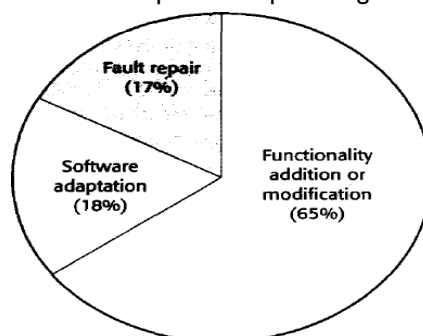
Lientz dan Swanson membagi pemeliharaan perangkat lunak ke dalam tiga komponen, yakni pemeliharaan korektif, adaptif, dan perfektif.

Pemeliharaan korektif mencakup semua perubahan yang dilakukan untuk menghilangkan kerusakan aktual pada perangkat lunak.

Pemeliharaan adaptif mencakup semua perubahan yang dibutuhkan sebagai konsekuensi dari perubahan lingkungan di mana sistem beroperasi, misalkan perubahan perangkat keras, sistem operasi, DBMS, atau jaringan komputer.

Pemeliharaan perfektif mencakup semua perubahan yang berasal dari permintaan pengguna.

Presentase masing-masing kategori pemeliharaan dapat dilihat pada diagram berikut ini:



Gambar 8.1 Presentase Kategori Pemeliharaan

Pigoski menggabungkan pemeliharaan adaptif dan perfektif sebagai enhancement karena kedua tipe ini tidak bersifat korektif, namun merupakan peningkatan kemampuan perangkat lunak.

Namun sebagian organisasi menggunakan istilah pemeliharaan perangkat lunak jika itu berkaitan dengan perubahan kecil pada sistem perangkat lunak, sedangkan untuk perubahan besar pada sistem perangkat lunak disebut dengan pengembangan perangkat lunak.

Idealnya, pemeliharaan tidak boleh mengurangi realibilitas dan struktur dari sistem, sebab akan menyusahakan perubahan di masa datang. Namun, kasus ini tidak berlaku pada dunia nyata di mana usia sistem akan mengakibatkan struktur sistem menjadi lebih kompleks dan sumber daya ekstra harus ditambahkan untuk menyediakan semantik dan menyederhanakan struktur. Karena itu, beberapa ahli menyarankan kategori keempat dari pemeliharaan perangkat lunak, yang disebut dengan pemeliharaan preventif.

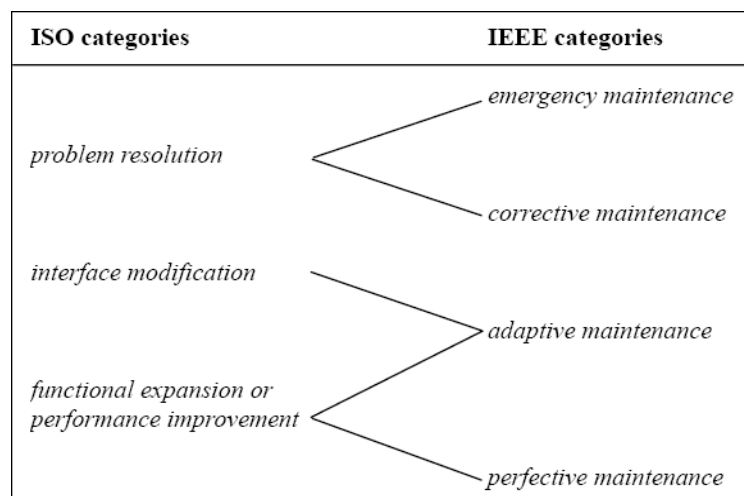
ISO mendefinisikan juga tiga kategori pemeliharaan perangkat lunak, yakni:

- **Resolusi permasalahan** yang mencakup deteksi, analisis, dan koreksi terhadap ketidaksesuaian perangkat lunak yang menyebabkan permasalahan operasional
- **Modifikasi antarmuka** diperlukan ketika perubahan dilakukan kepada sistem perangkat keras yang dikendalikan oleh perangkat lunak
- **Peningkatan kinerja/ekspansi fungsional** yang diperlukan oleh konsumen pada fase pemeliharaan. Sebuah rekomendasi adalah semua perubahan harus dilakukan dengan prosedur yang sama dengan yang digunakan pada pengembangan perangkat lunak

IEEE mengkategorikan pemeliharaan perangkat lunak ke dalam empat kategori, yakni:

- **Pemeliharaan korektif.** Perubahan reaktif pada perangkat lunak yang dilakukan setelah penggunaan perangkat lunak oleh konsumen untuk memperbaiki kerusakan yang ditemukan
- **Pemeliharaan adaptif.** Perubahan pada perangkat lunak yang dilakukan setelah penggunaan perangkat lunak oleh konsumen agar perangkat lunak dapat digunakan pada lingkungan yang berubah
- **Pemeliharaan perfektif.** Perubahan pada perangkat lunak yang dilakukan setelah penggunaan perangkat lunak oleh konsumen untuk meningkatkan kinerja atau maintainabilitas
- **Pemeliharaan emergensi.** Pemeliharaan korektif yang tidak dijadwalkan untuk menjaga operasional sistem

Berikut adalah hubungan antara kategorisasi yang dilakukan oleh ISO dengan yang dilakukan oleh IEEE:



8.3 Permasalahan Pemeliharaan Perangkat Lunak

Pemeliharaan merupakan aktivitas yang sangat menghabiskan biaya. Satu alasannya adalah karena untuk menambahkan fungsionalitas sistem yang sedang beroperasi jauh lebih mahal dibandingkan dengan menambahkan fungsionalitas sistem ketika fase pengembangan.

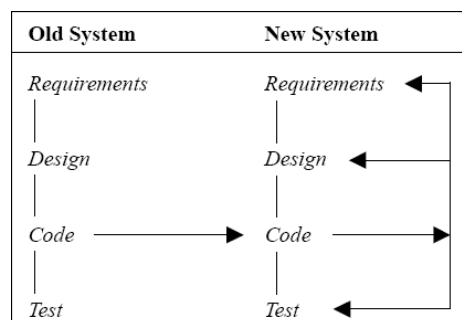
Faktor yang membedakan antara pengembangan dan pemeliharaan yang berakibat pada mahal nya pemeliharaan adalah:

- **Stabilitas sistem.** Setelah sistem dipasarkan, biasanya tim pengembang dibubarkan dan masing-masing anggota bekerja pada proyek yang baru. Tim yang kemudian bertanggung jawab terhadap pemeliharaan perangkat lunak tidak memiliki pemahaman yang lengkap terhadap perangkat lunak yang bersangkutan sehingga diperlukan usaha tambahan untuk memahami perangkat lunak yang bersangkutan
- **Tanggung jawab kontraktual.** Kontrak untuk pemeliharaan perangkat lunak biasanya terpisah dari kontrak untuk pengembangan perangkat lunak
- **Keahlian staf.** Staf pemeliharaan seringkali kurang pengalaman dan tidak terbiasa dengan domain aplikasi. Proses pemeliharaan seringkali dilihat sebagai proses yang membutuhkan skill tidak terlalu tinggi dibandingkan dengan proses pengembangan perangkat lunak, hal ini menyebabkan staf bagian pemeliharaan seringkali adalah staf dengan level junior. Lebih parah lagi, sistem yang dipelihara adalah seringkali sistem yang menggunakan bahasa pemrograman dengan versi lama. Staf bagian pemeliharaan tentu saja kurang familiar dengan bahasa pemrograman model ini sehingga perlu usaha untuk memahami bahasa pemrograman tersebut
- **Usia dan struktur program.** Seiring dengan usia program, struktur dari program juga ikut berubah sehingga semakin sulit untuk dimengerti apalagi diubah. Beberapa bagian sistem tidak dibuat dengan menggunakan teknik RPL modern, sehingga tidak pernah diatur dengan baik. Dokumentasi sistem mungkin saja hilang atau inkonsisten

Tiga permasalahan awal dapat diselesaikan dengan cara merencanakan sebuah proses pengembangan berkelanjutan sepanjang usia dari perangkat lunak sedangkan permasalahan yang terakhir dapat diselesaikan melalui teknik merekayasa ulang perangkat lunak (Software Re-engineering).

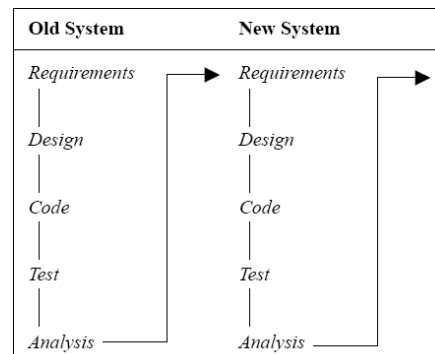
8.4 Model Pemeliharaan Perangkat Lunak

Pendekatan tipikal untuk pemeliharaan perangkat lunak adalah dengan mengubah kode program terlebih dahulu, kemudian membuat perubahan yang diperlukan pada dokumentasi program. Pendekatan ini disebut pendekatan quick-fix model. Idealnya setelah kode diubah, maka dokumentasi terkait kebutuhan, analisis, perancangan, pengujian, dan hal-hal terkait perangkat lunak yang bersangkutan harus diubah juga menyesuaikan dengan perubahan pada kode program. Namun realita di lapangan menunjukkan bahwa perubahan pada kode program kadang tidak didokumentasikan disebabkan oleh tekanan waktu dan biaya sehingga tim pemelihara tidak sempat untuk mengubah dokumentasi program.



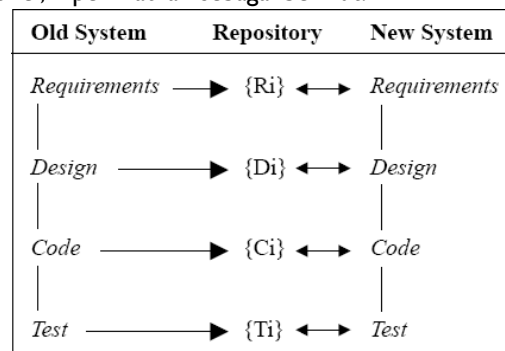
Gambar 8.2 Quick-Fix Model

Model siklus hidup evolutionary menawarkan pendekatan alternatif untuk pemeliharaan perangkat lunak. Model ini menyatakan bahwa kebutuhan sistem tidak dapat dikumpulkan dan dipahami pada tahap awal, sehingga sistem dibangun dengan memperbaiki kebutuhan dari bangunan sistem sebelumnya berdasarkan feedback dari pengguna. Kelebihan dari model ini adalah dokumentasi dari sistem senantiasa berubah seiring dengan perubahan pada kode program.



Gambar 8.3 Iterative-Enhancement Model

Ada lagi pendekatan full-reuse model, diperlihatkan sebagai berikut:



Gambar 8.4 Full-Reuse Model

Model ini memandang pemeliharaan sebagai sebuah kasus dari pengembangan perangkat lunak berorientasi gunaulang. Full-reuse dimulai dengan analisis kebutuhan dan perancangan dari sistem yang baru dan menggunakan ulang kebutuhan, rancangan, kode, dan pengujian dari sistem versi sebelumnya yang telah ada. Ini adalah perbedaan dari model iteratif-enhancement yang dimulai dari analisis terhadap sistem yang telah ada.

Model iterative-enhancement cocok digunakan pada sistem yang memiliki umur yang panjang dan berevolusi seiring dengan waktu. Model ini mendukung evolusi sistem untuk memudahkan modifikasi ke depannya.

Model Full-reuse cocok digunakan pada pengembangan sistem-sistem yang berkaitan. Model ini mengumpulkan komponen-komponen yang reuseable pada level abstraksi yang berbeda-beda dan menjadikan pengembangan sistem ke depannya menjadi lebih hemat.

8.5 Proses Pemeliharaan Perangkat Lunak

Ada beberapa model proses pemeliharaan perangkat lunak. Model-model ini mengorganisasikan pemeliharaan menjadi serangkaian aktivitas terkait dan menentukan urutan dari masing-masing aktivitas. Kadang-kadang juga disertai dengan penentuan hal-hal yang harus disampaikan antara aktivitas satu dengan aktivitas lainnya.

Dua jenis di antara model-model tersebut adalah versi IEEE yang menggunakan standar yang khusus dan ISO yang menggunakan standar sesuai dengan siklus hidup perangkat lunak.

8.5.1 Proses Pemeliharaan Versi IEEE-1219

Standar IEEE mengorganisasikan proses pemeliharaan menjadi tujuh fase. Pada tiap fase, standar IEEE menetapkan input dan output pada tiap fase, mengelompokkan dan menghubungkan aktivitas-aktivitas, mendukung proses-proses, kontrol, dan sekumpulan metrik.

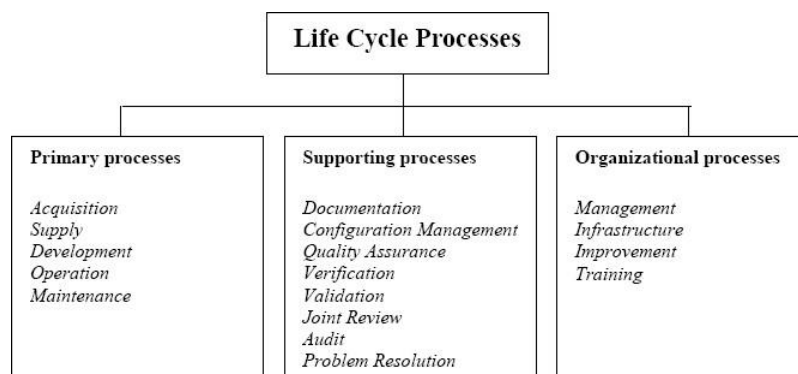
Ketujuh fase tersebut adalah:

1. **Identifikasi, klasifikasi, dan penentuan prioritas modifikasi.** Pada fase ini, permintaan perubahan yang diajukan oleh pengguna, konsumen, programmer, atau manajer ditetapkan sebagai kategori pemeliharaan dan menetapkan prioritas. Fase ini juga mencakup aktivitas untuk menentukan apakah permintaan tersebut disetujui atau tidak dan menetapkan ke dalam jadwal pengimplementasian

2. **Analisis.** Fase ini mencakup perencanaan awal untuk perancangan, implementasi, pengujian, dan pemasaran. Fase ini terdiri dari dua level, analisis feasibilitas yang menentukan solusi alternatif beserta efek dan biaya solusi tersebut dan juga analisis detail yang menentukan kebutuhan untuk modifikasi, strategi pengujian, dan juga membangun rencana pengimplementasian
3. **Perancangan.** Modifikasi sistem dirancang pada fase ini. Kegiatan ini menggunakan keseluruhan dokumentasi sistem dan proyek, basisdata dan perangkat lunak yang ada, dan output dari fase analisis. Aktivitas ini mencakup identifikasi terhadap modul yang terpengaruh, modifikasi dokumentasi modul perangkat lunak, pembuatan kasus uji untuk rancangan yang baru, dan identifikasi pengujian regresi
4. **Implementasi.** Fase ini mencakup aktivitas coding dan unit testing, integrasi modul yang telah dimodifikasi, integration dan regression testing, analisis resiko, dan kajian. Fase ini juga mencakup kajian kesiapan pengujian untuk menetapkan kesiapan untuk pengujian sistem dan regresi
5. **Regression/system testing.** Pada fase ini keseluruhan sistem diuji untuk memastikan kesesuaian dengan kebutuhan awal dan juga modifikasi kebutuhan tersebut. Selain pengujian fungsional dan antarmuka, fase ini juga mencakup pengujian regresi untuk memvalidasi tidak ada kerusakan baru yang muncul
6. **Acceptance testing.** Pengujian ini fokus pada sistem yang telah terintegrasi sepenuhnya dan melibatkan pengguna, konsumen, atau pihak ketiga yang dirancang oleh konsumen. Pengujian ini mencakup pengujian fungsional, interoperabilitas, dan regresi
7. **Delivery.** Pada fase ini, sistem dirilis untuk diinstal dan dioperasikan. Aktivitas ini mencakup pemberitahuan kepada pengguna, melakukan instalasi dan pelatihan, serta menyiapkan backup dari perangkat lunak versi sebelumnya

8.5.2 Proses Pemeliharaan Versi ISO-12207

Standar ISO fokus pada siklus hidup perangkat lunak. Standar ini menetapkan 17 aktivitas yang dikelompokkan ke dalam tiga kelas besar, yakni primary, supporting, dan organizational processes. Berikut pembagiannya:



Gambar 8.5 Proses Siklus Hidup ISO

Pemeliharaan merupakan satu dari kelima proses pada kelompok primary, di mana aktivitas pemeliharaan ini terdiri dari:

1. **Implementasi Proses.** Aktivitas ini mencakup rencana pengembangan dan prosedur pemeliharaan perangkat lunak, menciptakan prosedur penerimaan, pencatatan, dan penelusuran permintaan pemeliharaan, dan membangun antarmuka organisasional dengan proses manajemen konfigurasi. Perencanaan pemeliharaan sebaiknya dipersiapkan paralel dengan perencanaan pengembangan
2. **Analisis Masalah dan Modifikasi.** Aktivitas ini mencakup analisis terhadap permintaan pemeliharaan, apakah merupakan laporan permasalahan atau permintaan perubahan, mengklasifikasikannya, untuk menentukan besar skalanya, biaya, dan waktu yang dibutuhkan. Aktivitas lainnya adalah pengembangan dan pendokumentasian alternatif implementasi modifikasi dan penentuan opsi terpilih sesuai kontrak
3. **Implementasi Modifikasi.** Aktivitas ini mencakup identifikasi item yang perlu dimodifikasi dan pengajuan proses pengembangan untuk merealisasikan perubahan yang direncanakan. Tambahan kebutuhan untuk proses pengembangan adalah prosedur pengujian untuk memastikan bahwa kebutuhan yang telah dimodifikasi telah diimplementasikan dengan benar sepenuhnya dan kebutuhan awal yang tidak dimodifikasi tidak terpengaruh
4. **Penerimaan/Pengkajian Pemeliharaan.** Aktivitas ini mencakup penilaian integritas dari sistem termodifikasi hingga pengembang memperoleh pernyataan kepuasan dari terpenuhinya permintaan perubahan. Beberapa aktivitas lain yang mungkin dilakukan adalah penjaminan kualitas, verifikasi, validasi, dan joint review
5. **Migrasi.** Aktivitas ini terjadi ketika sistem perangkat lunak dipindahkan dari satu ke lingkungan ke lingkungan lainnya. Hal ini mengakibatkan harus dibuat sebuah perencanaan migrasi dan diketahui oleh pengguna sistem, alasan mengapa lingkungan yang lama tidak mendukung, dan sebuah deskripsi dari lingkungan baru dan kapan

bisa dipakai. Aktivitas ini juga fokus kepada proses paralel pada lingkungan lama dan baru serta kajian tentang efek migrasi ke lingkungan baru

6. **Pemberhentian Operasi Perangkat Lunak.** Aktivitas ini mencakup pemberhentian operasi dari sebuah perangkat lunak dan perencanaan pengembangan dari perangkat lunak tersebut serta pemberitahuan kepada pengguna mengenai hal tersebut

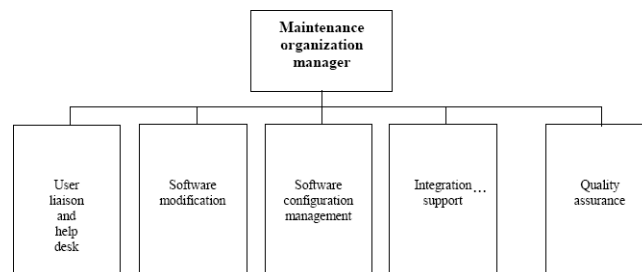
8.6 Manajemen Pemeliharaan Perangkat Lunak

Fungsi manajemen terdiri dari beberapa hal yakni:

1. **Planning.** Terdiri dari penentuan tujuan, misi, dan serangkaian aksi untuk merealisasikannya. Komitmen dari manusia dan sumber daya serta penjadwalan aksi adalah aktivitas yang penting pada fungsi ini
2. **Organizing.** Fungsi manajemen yang membangun pembagian peran manusia pada sebuah organisasi. Termasuk juga membangun hubungan antar manusia dan pemberian tanggung jawab serta hak yang dibutuhkan
3. **Staffing.** Mencakup bagaimana mengisi posisi pada organisasi dengan orang yang terpilih dan terlatih. Aktivitas kunci dari fungsi ini adalah mengevaluasi personal dan menyediakan pembangunan SDM contohnya peningkatan pengetahuan, sopan santun, dan keahlian
4. **Leading.** Menciptakan lingkungan kerja dan atmosfer yang akan membantu dan memotivasi orang agar mereka dapat berkontribusi maksimal untuk mencapai sasaran organisasi
5. **Controlling.** Mengukur kinerja aktual dengan sasaran yang hendak dicapai dan jika terjadi penyimpangan akan melakukan aksi korektif. Aktivitas juga mencakup reward and punish bagi personal

Organisasi pemeliharaan perangkat lunak dapat dirancang dan dibangun dengan menggunakan tiga struktur organisasi yang berbeda, yakni:

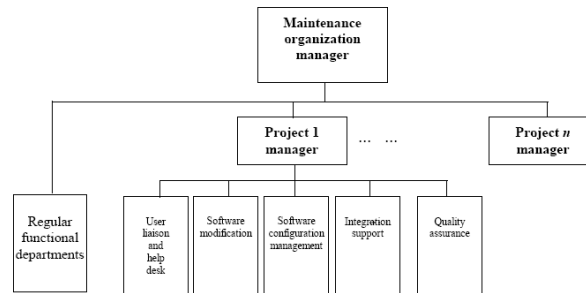
1. Fungsional Organization.



Gambar 8.6 Susunan Organisasi Fungsional

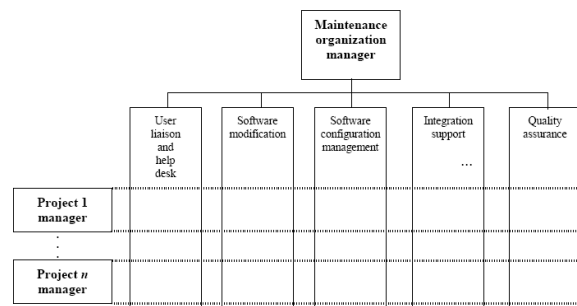
Organisasi dibagi menjadi unit-unit fungsional yang berbeda-beda, seperti modifikasi perangkat lunak, pengujian, dokumentasi, penjaminan kualitas, dsb. Organisasi fungsional menampilkan kelebihan dari organisasi terpusat dari sumber daya yang serupa. Kelemahan utamanya adalah permasalahan antarmuka yang sulit untuk diselesaikan misalkan departemen dilibatkan pada lebih dari satu proyek akan mengakibatkan konflik mengenai prioritas proyek-proyek yang ada karena keterbatasan sumber daya bahkan dengan kurangnya hak dan tanggung jawab pusat terhadap proyek akan mengakibatkan departemen fokus hanya pada spesialisasinya saja dibandingkan dengan sasaran proyeknya

2. **Project Organization.** Merupakan kebalikan dari tipe pertama. Pada tipe ini, seorang manajer diberikan tanggung jawab dan hak penuh untuk mengatur orang, semua sumber daya yang dibutuhkan untuk pengerjaan proyek dipisahkan dari struktur fungsional regulernya dan diorganisasikan pada bagian swantara tertentu. Manajer proyek mungkin saja mendapatkan tambahan sumber daya dari luar organisasi. Kelebihan dari tipe ini adalah kontrol penuh terhadap proyek, pengambilan keputusan yang cepat, dan masing-masing personal mendapatkan motivasi yang tinggi. Kekurangannya adalah adanya waktu yang dibutuhkan untuk membentuk sebuah tim dan kemungkinan inefisiensi sumber daya



Gambar 8.7 Susunan Organisasi Proyek

3. **Matrix Organization.** Gabungan kedua tipe di awal dengan tujuan untuk memaksimalkan kelebihan dan meminimalkan kekurangan kedua tipe di atas. Kelebihan dari tipe ini adalah adanya keseimbangan antara sasaran departemen fungsional dengan sasaran proyek itu sendiri. Masalah utama adalah setiap orang akan berkoordinasi dengan dua orang manajer dan ini bisa menjadi sumber konflik. Solusinya bisa dengan penentuan peran yang jelas, tanggung jawab dan hak dari manajer fungsional dan manajer proyek untuk setiap jenis keputusan

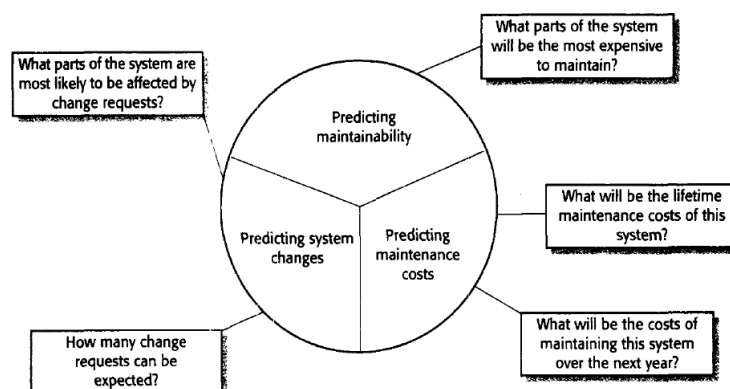


Gambar 8.8 Susunan Organisasi Matriks

8.7 Perencanaan Pemeliharaan Perangkat Lunak

Perencanaan pemeliharaan perangkat lunak sangat erat kaitannya dengan bagaimana memperkirakan perubahan-perubahan sistem yang mungkin terjadi dan bagian-bagian mana dari sistem yang kemungkinan sulit untuk dipelihara. Selain itu, harus memperkirakan biaya pemeliharaan untuk jangka waktu tertentu. Perkiraan-perkiraan berikut sangat terkait satu sama lain:

- Apakah perubahan sistem harus diterima tergantung dari maintainabilitas dari komponen sistem yang dipengaruhi oleh perubahan tersebut
- Mengimplementasikan perubahan sistem akan mendegradasi struktur sistem dan akan mengurangi nilai maintainabilitas dari sistem
- Biaya pemeliharaan tergantung pada jumlah perubahan dan biaya terhadap perubahan sistem bergantung dari maintainabilitas komponen sistem



Gambar 8.9 Memperkirakan Pemeliharaan Perangkat Lunak

Memperkirakan jumlah permintaan perubahan membutuhkan pemahaman tentang hubungan antara sistem dengan lingkungan eksternalnya. Beberapa sistem memiliki hubungan yang sangat kompleks dengan lingkungan eksternalnya dan perubahan pada lingkungan tersebut akan menyebabkan perubahan pada sistem. Untuk menentukan hubungan antara sistem dengan lingkungan eksternalnya, ada beberapa faktor yang perlu dinilai antara lain:

- **Jumlah dan kompleksitas antarmuka.** Semakin besar jumlah antarmuka dan semakin kompleks antarmuka tersebut, akan semakin tinggi permintaan perubahan yang muncul
- **Jumlah kebutuhan sistem yang berubah-ubah.** Kebutuhan yang mencerminkan prosedur atau kebijakan organisasional sangat mudah berubah dibandingkan dengan kebutuhan yang berasal dari domain yang karakteristiknya stabil
- **Proses bisnis dari sistem.** Seiring dengan perubahan proses bisnis akan menghasilkan permintaan-permintaan untuk perubahan sistem. Semakin banyak bisnis proses yang menggunakan sistem, semakin banyak permintaan perubahan terhadap sistem

Untuk memperkirakan maintainabilitas sistem, harus dipahami mengenai jumlah dan tipe dari hubungan antar komponen sistem dan juga kompleksitas dari komponen-komponen tersebut. Pengukuran kompleksitas tersebut sangat berguna untuk menentukan komponen program yang sangat sulit untuk dipelihara.

Selain itu untuk menentukan maintainabilitas sistem, dapat menggunakan beberapa metrik berikut:

- **Jumlah permintaan pemeliharaan korektif.** Peningkatan jumlah laporan kerusakan dapat mengindikasikan semakin banyak kesalahan yang muncul pada program dibandingkan dengan yang diperbaiki selama proses pemeliharaan. Ini dapat menunjukkan penurunan nilai maintainabilitas
- **Rata-rata waktu yang dibutuhkan untuk analisis akibat.** Hal ini mencerminkan jumlah komponen program yang terpengaruh oleh permintaan perubahan. Jika waktu ini meningkat, akan mengakibatkan semakin banyak komponen yang terpengaruh perubahan dan nilai maintainabilitas menurun
- **Rata-rata waktu yang dipakai untuk mengimplementasikan perubahan sistem.** Ini adalah waktu yang dibutuhkan untuk memodifikasi sistem dan dokumentasinya setelah menentukan komponen mana saja yang terpengaruh oleh perubahan. Peningkatan waktu yang dibutuhkan untuk mengimplementasikan perubahan sistem mengindikasikan penurunan nilai maintainabilitas
- **Jumlah permintaan perubahan yang drastis.** Peningkatan nilai ini dapat berakibat kepada penurunan nilai maintainabilitas

Perkiraan tentang permintaan-permintaan perubahan sistem dan maintainabilitas sistem dapat digunakan untuk memprediksi biaya pemeliharaan. Model COCOMO 2 menyatakan bahwa perkiraan besar usaha untuk pemeliharaan dapat dilihat dari besar usaha untuk memahami kode program yang ada pada sistem dan besar usaha untuk mengembangkan kode program yang baru.

Latihan

1. Jelaskan definisi dari pemeliharaan perangkat lunak versi Pigoski!
2. Sebutkan dan jelaskan aktivitas pemeliharaan yang bersifat pre-delivery dan post-delivery!
3. Sebutkan dan jelaskan kategori pemeliharaan perangkat lunak versi IEEE!
4. Sebutkan dan jelaskan faktor-faktor yang menyebabkan proses pemeliharaan perangkat lunak menjadi sangat mahal!
5. Jelaskan persamaan dan perbedaan antara model pemeliharaan Iterative-Enhancement dan Full-Reuse!
6. Jelaskan perbedaan antara model organisasi pemeliharaan fungsional dan proyek!

9 Object Oriented Concepts and Principles

9.1 Object Oriented

Saat ini piranti lunak semakin luas dan besar lingkupnya, sehingga tidak bisa lagi dibuat asal-asalan. Piranti lunak saat ini seharusnya dirancang dengan memperhatikan hal-hal seperti scalability, security, dan eksekusi yang robust walaupun dalam kondisi yang sulit. Selain itu arsitekturnya harus didefinisikan dengan jelas, agar bug mudah ditemukan dan diperbaiki, bahkan oleh orang lain selain programmer aslinya.

Keuntungan lain dari perencanaan arsitektur yang matang adalah dimungkinkannya penggunaan kembali modul atau komponen untuk aplikasi piranti lunak lain yang membutuhkan fungsionalitas yang sama.

Pemodelan (modeling) adalah proses merancang piranti lunak sebelum melakukan pengkodean (coding). Model piranti lunak dapat dianalogikan seperti pembuatan blueprint pada pembangunan gedung. Membuat model dari sebuah sistem yang kompleks sangatlah penting karena kita tidak dapat memahami sistem semacam itu secara menyeluruh. Semakin kompleks sebuah sistem, semakin penting pula penggunaan teknik pemodelan yang baik.

Dengan menggunakan model, diharapkan pengembangan piranti lunak dapat memenuhi semua kebutuhan pengguna dengan lengkap dan tepat, termasuk faktor-faktor seperti scalability, robustness, security, dan sebagainya.

Kesuksesan suatu pemodelan piranti lunak ditentukan oleh tiga unsur, yang kemudian terkenal dengan sebuah segitiga sukses (the triangle for success).

Ketiga unsur tersebut adalah metode pemodelan (notation), proses (process) dan tool yang digunakan.

Memahami notasi pemodelan tanpa mengetahui cara pemakaian yang sebenarnya (proses) akan membuat proyek gagal. Dan pemahaman terhadap metode pemodelan dan proses disempurnakan dengan penggunaan tool yang tepat.

Dalam suatu proses pengembangan software, analisa dan rancangan telah merupakan terminologi yang sangat tua. Pada saat masalah ditelusuri dan spesifikasi didefinisikan, dapat dikatakan kita berada pada tahap rancangan.

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

Sistem Konvensional

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

Sistem Berorientasi Objek

- Fokus pada Domain Objek, tidak pada prosedur;
- Data dan Prosedur disimpan dalam Objek;
- Dekomposisi Data

Tiga tahap dasar dalam pengembangan sistem:

1. Analisis : investigasi / memahami permasalahan (*what*) – *Conceptual Model, System Requirements*.
2. Perancangan : mengorganisasikan atau menstrukturkan permasalahan untuk memenuhi persyaratan (*how*). *System Design, Detailed Design*
3. Pemodelan : memahami struktur dan perilaku.
4. Implementasi : membuat solusi pemecahan masalah dapat dilaksanakan. *Coding – Testing*

Konsep Objek

Obyek dalam 'software analysis & design' adalah sesuatu berupa konsep (concept), benda (thing), dan sesuatu yang embedkannya dengan lingkungannya. Secara sederhana obyek adalah mobil, manusia, alarm dan lainlainnya.

Tapi obyek dapat pula merupakan sesuatu yang abstrak yang hidup didalam system seperti tabel, database, event, system messages.

Objek (object)

Objek adalah benda secara fisik dan konseptual yang ada di sekitar kita. Beberapa contoh objek, misalnya hardware, software, dokumen, manusia, konsep, dan lainnya.

Untuk kepentingan pemodelan, misalnya seorang eksekutif akan melihat karyawan, gedung, divisi, dokumen, keuntungan perusahaan sebagai sebuah objek.

Sedangkan seorang teknisi mobil, akan melihat ban, pintu, mesin, kecepatan tertentu dan banyaknya bahan bakar sebagai sebuah objek. Contoh lainnya adalah seorang software engineer akan memandang tumpukan, antrian intruksi, window, check box sebagai sebuah objek.

Sebuah objek mempunyai keadaan sesaat yang disebut state.

State dari sebuah objek adalah kondisi dari objek itu atau himpunan keadaan yang menggambarkan objek tersebut. Sebagai contoh, state dari rekening tabungan, dapat memuat saldo yang berjalan, state dari sebuah jam adalah catatan saat itu; sedangkan state dari sebuah bohlam lampu adalah suatu keadaan “nyala” atau “mati”.

State dinyatakan dengan nilai dari atribut objeknya.

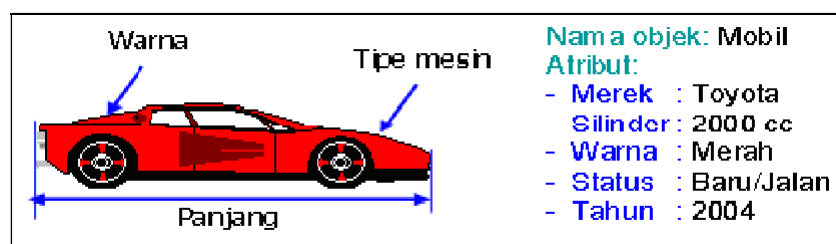


Gambar 9.1 : Object

Obyek dikenali dari keadaannya dan juga operasinya. Sebagai contoh sebuah mobil dikenali dari warnanya, bentuknya, sedangkan manusia dari suaranya. Ciri ciri ini yang akan membedakan obyek tersebut dari obyek lainnya.

Atribut:

adalah nilai internal suatu objek yang mencerminkan antara lain karakteristik objek, kondisi sesaat, koneksi dengan objek lain dan identitas. Perubahan state dicerminkan oleh perilaku (behaviour) objek tersebut.



Gambar 9.2 : Attribute

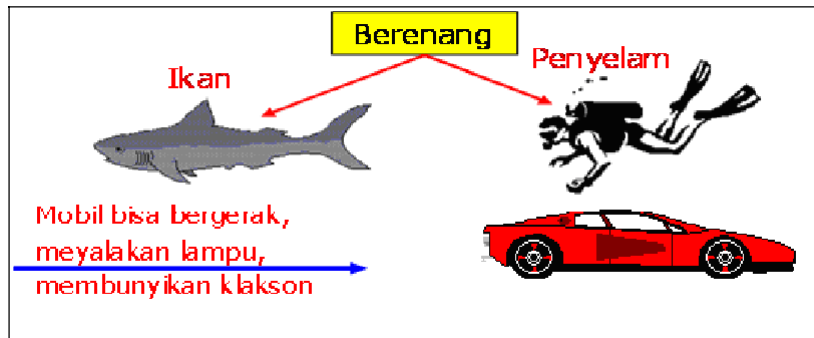
- Behavior mendefinisikan bagaimana suatu objek bertindak dan bereaksi, dan berhubungan dengan fungsi diterapkan pada suatu atribut.
- Behavior objek disebut metoda atau operasi pelayanan (service).

Behaviour atau perilaku sebuah objek mendefinisikan bagaimana sebuah objek bertindak(beraksi) dan memberi reaksi. Behaviour ditentukan oleh himpunan semua atau beberapa operasi yang dapat dilakukan oleh objek itu sendiri. Behaviour dari sebuah objek dicerminkan oleh interface, service dan method dari objek tersebut.

Interface adalah pintu untuk mengakses service dari objek.

Service adalah fungsi yang dapat dikerjakan oleh sebuah objek.

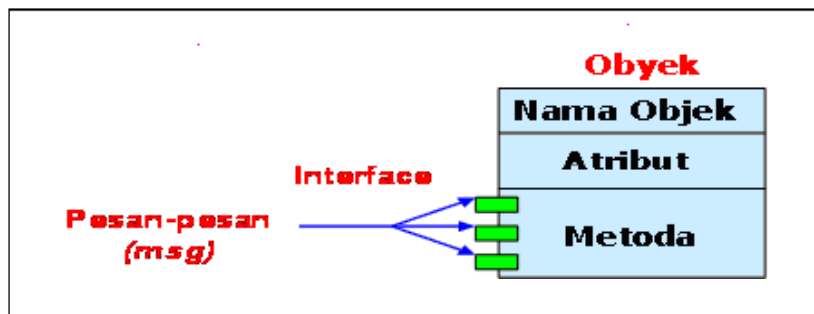
Method adalah mekanisme internal objek yang mencerminkan perilaku(behaviour) objek tersebut.



Gambar 9.3 : Behavior

Anatomi suatu Obyek

- **Obyek** adalah sekumpulan atribut (data) bersama dengan gabungan metoda (fungsi) yang digunakan untuk mengoperasikan atribut tersebut.
Obyek = Atribut + Metoda
- Dunia luar berkomunikasi ke obyek dengan mengirimkan pesan (*message*).



Gambar 9.4 : Kelas

Kelas (Class)

Class adalah definisi umum (pola, template, atau cetak biru) dari himpunan objek yang sejenis. Kelas menetapkan spesifikasi perilaku (behaviour) dan atribut-atribut dari objek tersebut. Class adalah abstraksi dari entitas dalam dunia nyata.

Sedangkan objek adalah contoh ("instances") dari sebuah kelas.

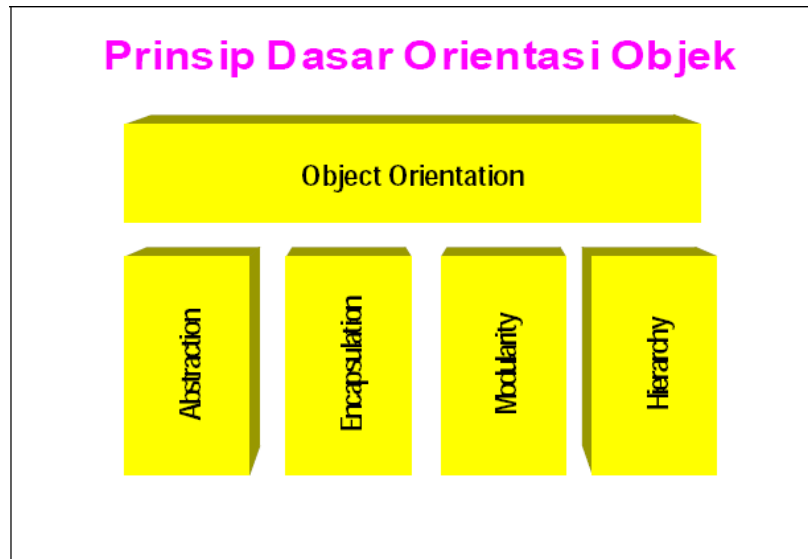
Misalnya, atribut dari kelas binatang adalah berkaki empat dan mempunyai ekor. Perilakunya adalah makan dan tidur. Sedangkan contoh (instance) untuk kelas binatang ini adalah kucing, gajah, dan kuda.

Kotak Hitam (Black Boxes)

Sebuah objek adalah **kotak hitam (black-boxes)**. Konsep ini menjadi dasar untuk implementasi objek.

Dalam operasi OO, hanya para developer

(programmer, desainer, analis) yang dapat memahami detail dari proses-proses yang ada didalam kotak hitam tersebut, sedangkan para pemakai (user) tidak perlu mengetahui apa yang dilakukan, tetapi yang penting mereka dapat menggunakan objek untuk memproses kebutuhan mereka.



Gambar 9.5 :Prinsip dasar Orientasi Objek

Abstraksi adalah menemukan serta memodelkan fakta-fakta dari suatu object yang penting dari suatu aplikasi



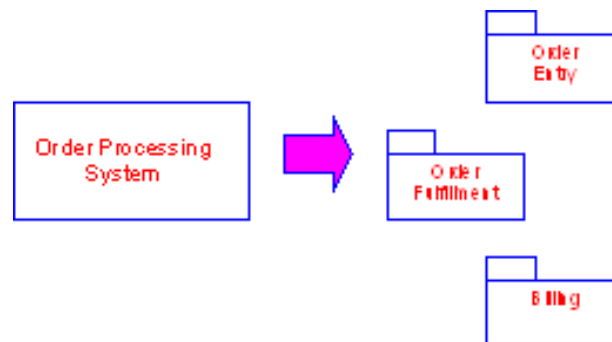
Gambar 9.6 : Abstraksi

Encapsulation (Pengapsulan)

- Pengapsulan berarti mengemas beberapa item bersama-sama menjadi satu unit yang tertutup dalam rangka menyembunyikan struktur internal suatu obyek dari lingkungan/dunia luar.
- Pengapsulan seringkali dianggap sebagai “penyembunyian informasi”.
- Setiap kelas hanya menampilkan interface yang diperlukan untuk berkomunikasi dengan dunia luar melalui message dan menyembunyikan (encapsulating) implementasi aktual didalam kelas.
- Kita hanya membutuhkan pemahaman tentang interface (methode), tidak perlu paham tentang internalnya (implementation).

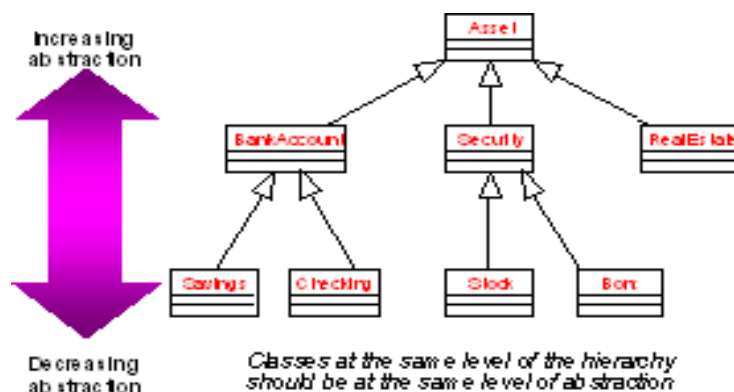
Modularity adalah

Memecah sesuatu yang kompleks menjadi bagian-bagian yang terkelola.



Gambar 9.7 : Modularity

Hierarchy adalah Tingkat-tingkat abstraksi



Gambar 9.8 : Hierarchy

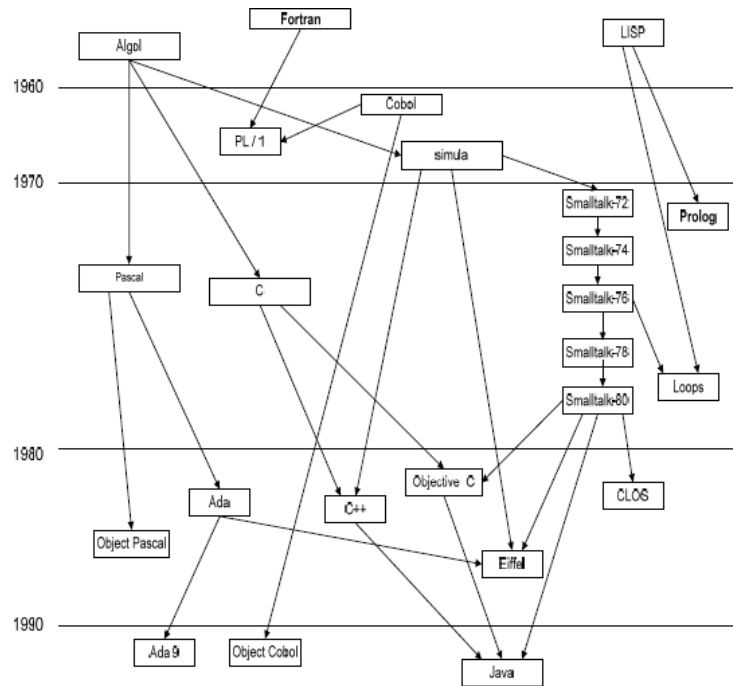
9.1.1 Karakteristik Sistem Berorientasi

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

- **Abstraksi**
Prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi suatu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.
- **Enkapsulasi**
Pembungkusan atribut data dan layanan (operasi-operasi) yang dimiliki 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 sebagian bagian dari dirinya.
- **Reuseability**
Pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan lainnya yang melibatkan objek tersebut.
- **Generalisasi dan Spesialisasi**
Menunjukkan hubungan antara kelas dan objek yang umum dengan kelas dan objek 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 Perkembangan metode Object Oriented Analysis and Design (OOAD)

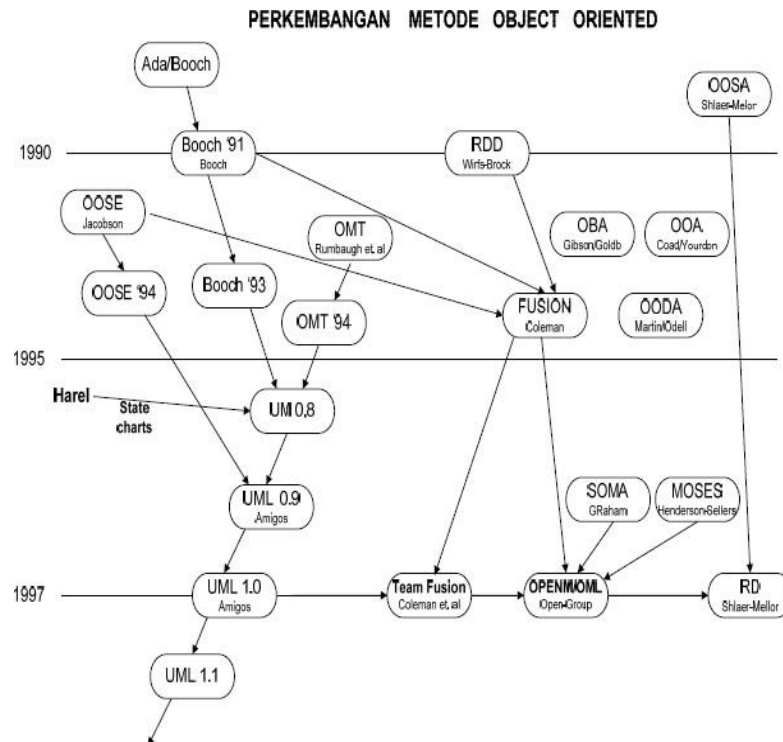
Konsep objek telah dikenal sejak lebih dari tiga puluh tahun yang lalu (Oesterich, p.5). Diawali dengan penggunaan pemrograman berorientasi objek (OOP), lalu berkembang menjadi konsep perancangan berorientasi objek (OOD) dan selanjutnya metode analisis dan perancangan berorientasi objek (OOAD) di tahun 1990.



Gambar 9.6: Perkembangan Bahasa Pemrograman

Di awal tahun 1990, metode OO yang dikenalkan oleh Grady Booch dan James Rumbaugh menjadi amat populer, Rumbaugh menekankan pengembangan berorientasi objek berdasarkan pendekatan terstruktur, sementara Booch menerapkan metode objek pada bidang teknik dan bisnis. Selanjutnya pada tahun 1995 muncul gagasan Booch dan Rumbaugh untuk menggabungkan metode mereka dengan membakukan notasi symbol yang digunakan dalam menggambarkan komponen sebuah aplikasi dan selanjutnya disebut Unified Method (UM).

Kemudian Ivar Jacobson bergabung bersama mereka untuk menyempurnakan metode objek ini, dengan menyusun konsep Use-Case. Munculnya metode yang dibuat oleh Booch, Rumbaugh dan Jacobson (ketiganya sering disebut pendekar metode objek) selanjutnya lebih dikenal sebagai bahasa Unified Modeling Language (UML ver 0.9). Penggunaan UML semakin populer, dan pada tahun 1997 UML ver 1.0 dibawa ke dalam konferensi organisasi pemakai objek (Object Management Group – OMG) yang akhirnya menyepakati untuk dijadikan sebagai standar metode pengembangan sistem dengan munculnya UML versi 1.1



Gambar 9.7: Perkembangan metode Object Oriented Analysis and Design (OOAD)

UML mendefinisikan notasi-notasi tunggal dan semantiknya bersama-sama di dalam sebuah meta-model, yaitu kumpulan berbagai notasi-simbol UML, yang secara bersama digunakan untuk pengembangan suatu aplikasi yang lengkap.

UML adalah bahasa pemodelan yang dapat dikembangkan lebih lanjut ke dalam satu bahasa program dengan menggunakan code-generator, sehingga berpotensi untuk menjadi dasar pengembangan suatu Case tools pengembangan sistem.

Sebagai catatan, sebenarnya di luar bahasa UML yang telah disepakati tersebut, berkembang pula metode lainnya misalnya metode yang dikembangkan oleh Harel(disebut state diagram), selain itu juga metode yang dikembangkan oleh Shlaer&Mellor (OO Analysis/Design) , selain yang dikembangkan oleh Coleman berdasarkan ide-ide Wirfs Brock yaitu Open Modeling Language (OPEN/OML) yang merupakan pesaing OOAD, dengan membuat Konsorsium Open didukung oleh Brian Henderson-Seller, Ian Graham dan Donald Firesmith.

Beberapa kriteria untuk memilih metode berorientasi objek ini adalah :

- Pertama, metode tersebut harus cocok untuk requirement aplikasi termasuk didalamnya tahapan dalam life cycle system, dan cocok dengan bahasa pemrograman yang dipakai.
- Kedua, pengalaman developer (programmer, desainer sistem analis) mempengaruhi seberapa bagus mereka menggunakan metode yang dipilih
- Ketiga, apakah metode mendukung fitur pengembangan sistem lainnya, misalnya tools untuk membuat model
- Keempat, metode harus mudah digunakan dan mudah dimengerti

Sebagai ilustrasi, dapat dijelaskan, metode Rumbaugh OMT berdasarkan analisis terstruktur dan pemodelan entity-relationship. Tahapan utama dalam metodologi ini adalah analisis, desain system dan desain objek serta implementasi. Keunggulan metode ini adalah dalam notasi yang mendukung semua konsep OO

Selanjutnya metode Shlaer&Mellor(OOA/D) menggunakan teknik pemodelan informasi tradisional untuk menjelaskan entitas dalam system, menggunakan state diagram untuk memodelkan keadaan (state) entitas, menggunakan data-flow diagram untuk memodelkan alur data dalam sebuah sistem. Metode ini menghasilkan tiga jenis model, yaitu information model, state model dan proses model. Keunggulan metode ini adalah dalam memandang masalah dari sudut pandang yang berbeda, mudah dikonversi dari model struktural.

Sedangkan metode Booch, dikenal sebagai metode desain OO. Metode ini menjadikan proses analisis dan desain ke dalam empat tahapan tahapan iterative (berulang), yaitu identifikasi kelas-kelas dan objek-objek,

identifikasi semantic dan hubungan objek dan kelas tersebut, rincian interface dan implementasinya. Metode ini sangat detail dan kaya dengan notasi dan elemen gabungan dari metode lain..

Sejak standarisasi metodologi, OMG telah mengeluarkan Request for Proposal (RFP) pada Juni 1996 untuk mendorong perusahaan-perusahaan system informasi, developer software dan para user system computer agar membuat sebuah RFP bersama sebagai respon. Satu perusahaan software, Rational Software telah membentuk konsorsium dengan berbagai organisasi untuk meresmikan pemakaian UML sebagai standar dalam OOAD.

Kontribusi untuk UML telah dihasilkan dari perusahaan-perusahaan besar yang mendukungnya, diantaranya : Digital Equipment Corp (DEC), Hawlet-Packard (HP), i-Logic, IntellCorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, Texas Instrument (TI), Unysis Paltinum Technology, PTech, Taskon&Reich Technologies, dan Softeam.

9.3 Konsep OOAD

OOAD mencakup analisis dan desain sebuah sistem dengan pendekatan objek.

Analisis berorientasi objek (OOA) adalah metode analisis yang memeriksa requirement(syarat/keperluan yang harus dipenuhi sebuah system) dari sudut pandang kelas-kelas dan objek-objek yang ditemui dalam ruang lingkup perusahaan.

Sedangkan desain berorientasi objek (OOD) adalah metode untuk mengarahkan arsitektur software yang didasarkan pada manipulasi objek-objek sistem atau subsistem Terdapat beberapa konsep dasar dalam OOAD, yaitu :

a. Objek (object)

Objek adalah benda secara fisik dan konseptual yang ada di sekitar kita. Beberapa contoh objek, misalnya hardware, software, dokumen, manusia, konsep, dan lainnya.

Untuk kepentingan pemodelan, misalnya seorang eksekutif akan melihat karyawan, gedung, divisi, dokumen, keuntungan perusahaan sebagai sebuah objek. Sedangkan seorang teknisi mobil, akan melihat ban, pintu, mesin, kecepatan tertentu dan banyaknya bahan bakar sebagai sebuah objek. Contoh lainnya adalah seorang software engineer akan memandang tumpukan, antrian intruksi, window, check box sebagai sebuah objek. Sebuah objek mempunyai keadaan sesaat yang disebut state.

State dari sebuah objek adalah kondisi dari objek itu atau himpunan keadaan yang menggambarkan objek tersebut. Sebagai contoh, state dari rekening tabungan, dapat memuat saldo yang berjalan, state dari sebuah jam adalah catatan saat itu; sedangkan state dari sebuah bohlam lampu adalah suatu keadaan “nyala” atau “mati”. State dinyatakan dengan nilai dari atribut objeknya.

Atribut adalah nilai internal suatu objek yang mencerminkan antara lain karakteristik objek, kondisi sesaat, koneksi dengan objek lain dan identitas. Perubahan state dicerminkan oleh perilaku (behaviour) objek tersebut.

Behaviour atau perilaku sebuah objek mendefinisikan bagaimana sebuah objek bertindak(beraksi) dan memberi reaksi. Behaviour ditentukan oleh himpunan semua atau beberapa operasi yang dapat dilakukan oleh objek itu sendiri. Behaviour dari sebuah objek dicerminkan oleh interface, service dan method dari objek tersebut.

Interface adalah pintu untuk mengakses service dari objek.

Service adalah fungsi yang dapat dikerjakan oleh sebuah objek.

Method adalah mekanisme internal objek yang mencerminkan perilaku(behaviour) objek tersebut.

b. Kelas (Class)

Class adalah definisi umum (pola, template, atau cetak biru) dari himpunan objek yang sejenis. Kelas menetapkan spesifikasi perilaku (behaviour) dan atribut-atribut dari objek tersebut. Class adalah abstraksi dari entitas dalam dunia nyata. Sedangkan objek adalah contoh (“instances”) dari sebuah kelas.

Misalnya, atribut dari kelas binatang adalah berkaki empat dan mempunyai ekor. Perilakunya adalah makan dan tidur. Sedangkan contoh (instance) untuk kelas binatang ini adalah kucing, gajah, dan kuda.

c. Kotak Hitam (Black Boxes)

Sebuah objek adalah **kotak hitam (black-boxes)**. Konsep ini menjadi dasar untuk implementasi objek. Dalam operasi OO, hanya para developer (programmer, desainer, analis) yang dapat memahami detail dari proses-proses yang ada didalam kotak hitam tersebut, sedangkan para pemakai (user) tidak perlu mengetahui apa yang dilakukan, tetapi yang penting mereka dapat menggunakan objek untuk memproses kebutuhan mereka.

Encapsulation, proses menyembunyikan detail implementasi sebuah objek. Satu-satunya jalan untuk mengakses data objek tersebut adalah melalui interface.

Interface melindungi internal state sebuah objek dari “campur tangan” pihak luar. Oleh karena itu objek digambarkan sebagai sebuah kotak hitam yang menerima dan mengirim pesan-pesan (messages).

Dalam OOP kotak hitam tersebut berisi kode (instruksi yang dipahami computer) dan data (informasi dimana instruksi tersebut beroperasi dengannya). Dalam OOP kode dan data disatukan dalam sebuah “benda” yang tersembunyi isinya yaitu objek. Pengguna objek tidak perlu mengetahui isi dalam kotak tersebut; untuk berkomunikasi dengan objek, diperlukan

pesan(message).

Message adalah permintaan agar objek menerima (receive) untuk membawa metode yang ditunjukkan oleh perilaku dan mengembalikan result dari aksi tersebut kepada objek pengirim (sender).

Contohnya, satu objek orang mengirim kepada objek bola lampu sebuah pesan message untuk menyalakan melalui saklar.

Objek bola lampu memiliki perilaku yang akan mengubah keadaannya(state) dari padam menjadi menyala. Objek lampu menyalakan dirinya dan menunjukkan kepada objek orang tersebut bahwa state barunya adalah menyala.

d. Asosiasi dan Agregasi

Asosiasi adalah hubungan yang mempunyai makna antara sejumlah objek. Asosiasi digambarkan dengan sebuah garis penghubung di antara objeknya.

Contoh :

Asosiasi antara objek mobil dengan seseorang. Mobil dapat dimiliki oleh satu atau beberapa orang, sedangkan seseorang dapat mempunyai nol, satu atau banyak mobil Asosiasi antara karyawan dengan unit-kerja. Seorang karyawan bekerja di satu unit-kerja.

Sedangkan sebuah unit-kerja dapat memiliki beberapa orang karyawan

Agregasi adalah bentuk khusus sebuah asosiasi yang menggambarkan seluruh bagian pada satu objek merupakan bagian dari objek yang lain.

Contoh :

Kopling dan piston adalah bagian dari mesin. Sedangkan mesin, roda, body adalah merupakan bagian dari sebuah mobil.

Tanggal, bulan dan tahun adalah bagian dari tanggal-lahir. Sedangkan tanggal-lahir, nama, alamat, jenis kelamin adalah bagian dari identitas Seseorang.

9.4 Object Management Group (OMG)

UML adalah bahasa berbasis simbol yang dapat digunakan untuk visualisasi, spesifikasi, membuat dan mendokumentasikan setiap tahap dalam pengembangan sebuah sistem.

UML diterima sebagai standar pengembangan software oleh OMG pada Nopember 1997.

- a. Object Management Group, Inc. (OMG) adalah sebuah organisasi internasional yang dibentuk pada tahun 1989, didukung lebih dari 800 anggota, terdiri dari perusahaan sistem informasi, software developer dan para user sistem komputer. OMG mempromosikan teori dan praktek-praktek object oriented technology dalam rekayasa software.
- b. Organisasi ini salah satunya bertugas membuat spesifikasi “manajemen objek” untuk menetapkan kerangka bersama dalam rekayasa software. Spesifikasi tersebut dibuat dengan tujuan utama untuk menghasilkan reusability, portability dan interoperability software yang berdasarkan OO dalam lingkungan yang heterogen dan dapat dioperasikan dalam semua platform hardware dan sistem operasi
- c. Sasaran OMG adalah membantu mengembangkan teknologi OO dan mengarahkannya dengan mendirikan Object Management Architecture (OMA), yang bertugas menentukan infrastruktur konseptual yang didasarkan pada seluruh spesifikasi yang dikeluarkan oleh OMG.
- d. Selanjutnya OMG mengeluarkan UML dengan maksud dapat mengurangi kekacauan dalam bahasa pemodelan yang saat itu terjadi dalam lingkungan industri.

UML diharapkan dapat menjawab masalah penotasian dan mekanisme tukar menukar model yang terjadi.

9.5 Tinjauan tentang Unified Modeling Language (UML)

UML merupakan penggabungan berbagai konsep terbaik dari pemodelan, yaitu pemodelan data (entity-relationship diagram), pemodelan bisnis (Workflow), pemodelan objek dan komponennya. UML merupakan bahasa standar untuk visualisasi, spesifikasi, konstruksi dan pendokumentasian dari artefak dari sebuah software, dan dapat digunakan untuk semua tahapan dalam proses pengembangan sistem mulai dari analisis, desain, sampai implementasi.

Artefak UML

UML menyediakan beberapa notasi dan artefak standar yang dapat digunakan sebagai alat komunikasi bagi para pelaku dalam proses analisis dan desain sistem. Artefak dalam UML didefinisikan sebagai informasi dalam berbagai bentuk yang digunakan atau dihasilkan dalam proses pengembangan software. Terdapat beberapa artefak utama dalam UML, yaitu :

1. Use Case Diagram

Diagram yang menggambarkan actor, use case dan relasinya

2. Class Diagram

Diagram untuk menggambarkan kelas dan relasi diantara kelas-kelas tersebut

3. Behaviour Diagram, yang terdiri dari :

a. Activity Diagram

Menggambarkan aktifitas-aktifitas, objek, state, transisi state dan event

b. Collaboration Diagram

Menggambarkan objek dan relasinya, termasuk struktur perubahannya yang disebabkan oleh adanya suatu message

c. Sequence Diagram

Menggambarkan objek dan relasinya termasuk kronologi (urutan) perubahan secara logis setelah menerima sebuah message

d. Statechart Diagram

Menggambarkan state, transisi state dan event

4. Implementation Diagram, terdiri dari :

a. Component Diagram

Menggambarkan komponen dan relasi antara komponen tersebut

b. Deployment Diagram

Menggambarkan komponen, titik awal dan relasi antara komponen tersebut

Use case diagram merupakan artefak dari proses analisis, sementara sequence diagram dan class diagram merupakan artefak dari proses desain. Yang perlu diperhatikan, untuk menjaga konsistensi antara artefak selama proses analisis dan desain, maka setiap perubahan yang terjadi pada satu artefak harus juga dilakukan pada artefak sebelumnya. Misalnya ditemukan satu cara yang lebih efisien sewaktu membuat sequence diagram, maka perbaikan itu perlu diverifikasi terhadap use case diagram dan use case specification yang dibuat sebelumnya.

Dibuatnya berbagai jenis diagram tersebut karena :

- setiap sistem yang kompleks selalu paling baik jika didekati melalui himpunan berbagai sudut pandang yang kecil, yang satu sama lain hampir saling bebas (independen). Sudut pandang tunggal senantiasa tidak mencukupi untuk melihat sistem yang besar dan kompleks.
- Diagram yang berbeda-beda tersebut dapat menyatakan tingkatan yang berbeda dalam proses rekayasa.
- Dengan diagram diharapkan dapat membuat model sistem yang semakin mendekati realitas.

Berbagai diagram di atas ditambah dengan kemampuan dokumentasi merupakan artefak utama UML.

Semantik dalam UML

OMG telah menetapkan semantik (makna istilah) semua notasi diagram UML dalam model struktural dan model behavioral. Model struktural, disebut juga sebagai model statis, menekankan struktur objek dalam sebuah sistem, menyangkut kelas-kelas, interface, atribut, dan hubungan antar komponen. Model behavioral, yang juga disebut model dinamis, menekankan perilaku objek dalam sebuah sistem, termasuk metode, interaksi, kolaborasi dan state history.

Notasi dalam UML

UML memiliki notasi untuk menjelaskan secara visual mengenai elemen-elemen pemodelan.

Pada diagram Use-Case terdapat notasi untuk use-case, actor dan System. Sedangkan notasi untuk menggambarkan Class diagram, terdiri dari notasi Class, asosiasi, agregasi, generalisasi dan spesialisasi dan seterusnya.

Beberapa notasi dalam UML :

1. Actor
2. Class
3. Use Case dan use case specification
4. Realization
5. Interaction
6. Dependency
7. Note
8. Interface, dll
9. Association
10. Generalization
11. Use Case Diagram
12. Sequence Diagram
13. Class Diagram
14. Package

Latihan

1. Jelaskan kenapa konsep OO berkembang dimulai dari object oriented programming (OOP) ?
2. Siapakah yang disebut tiga pendekar (amigos) metode objek?
3. Apakah peranan Rational Software dalam perkembangan UML?
4. Apakah bedanya objek dengan kelas ? Asosiasi dengan agregasi?
5. Jelaskan peranan dari Object Management Group (OMG) ?
6. Apakah yang dimaksud dengan artifak dari UML ?
7. Sebutkan beberapa notasi dalam UML
8. Jelaskan konsep-konsep di bawah ini :
 - a. State
 - b. Atribut
 - c. Behavior
 - d. Interface, Service dan Method
 - e. Black box

Daftar Pustaka

1. Roger Pressman, "Software Engineering A Practitioner's Approach", 5th Edition, Mc GrawHill
2. Barbee Teasley Mynatt, " Software Engineering with Student Project Guidance", Prentice Hall 1990
3. Ian Somerville, Software Engineering, 5th Edition, Addison-Wesley
4. Developing Software with UML, Bernd Oestereich (1999), Addison-Wesley
5. Davis, Alan M., "201 Principles of Software Development", McGraw-Hill 1995.
6. IEEE Standart 1016-1998 Software Design Description.
7. Meyer, B. "Object-Oriented Software Construction", Prentice-Hall, 1988.
8. McGlaughin, R., "Some Note on Program Design", Software Engineering Note, vol. 16. No. 4, Oktober 1991, h53-54
9. Davis, Alan M., "Software Requirements: Objects, Functions and States", Prentice-Hall International Editions, Englewood Cliffs, New Jersey, New Jersey, 1993.
10. DeMarco, Tom., "Structured Analysis and System Specifications", Prentice Hall, New York, 1979.
11. The Institute of Electrical and Electronics Engineers, "IEEE Std 610.12-1993 Standard Glossary of SW Engineering Terminology", 1993.
12. The Institute of Electrical and Electronics Engineers, "IEEE Std 830-199 Recommended Practice for SW Requirements Specifications (SRS)", 1998.
13. Witarto, "Memahami Sistem Informasi", Penerbit Informatika, Bandung, 2004
14. Yourdon, Edward, "Modern Structured Analysis", Prentice-Hall International Inc., Englewood Cliffs, New Jersey, 1989.