BAB 7
FASE DISAIN

7.1. PENDAHULUAN

Aktivitas utama dalam Fase Disain adalah membuat *top* dan *medium level* dari disain sistem dan mendokumentasikannya dalam Spesifikasi Disain. Aktivitas kedua dimulai dengan melakukan Rencana Test Penerimaan (*Acceptance Test Plan* / ATP).

ATP adalah sebuah dokumen tes yang akan digunakan untuk mendemonstrasikan seluruh fungsi sistem kepada user pada fase penerimaan.

Terdapat dua langkah dalam mendisain sistem software, yaitu :
- Pertama, bagilah sistem menjadi beberapa komponen secara fungsional.
- Kedua, hubungkanlah komponen-komponen tersebut.

7.2. DISAIN YANG TERSTRUKTUR (*STRUCTURED DESIGN*)

Tujuan utama dari disain yang terstruktur adalah memecah sistem menjadi bagian yang lebih kecil, teratur dan mudah untuk dibangun.

Disain Top Down (*Top Down Design*)

Disain *Top Down* dimulai dengan *Top Level Design* (TLD).

Lihat Gambar 7.1. Top Level Design

Masing-masing komponen utama atau kotak dalam TLD dipecah menjadi sub-bagian dimulai dengan level teratas, kemudian turun ke level berikutnya, dst.

Dalam kasus ini, dimulai dengan MENU dan mendisainnya sebelum turun ke INQUIRY, UPDATE, dan REPORT GENERATION, yang akan diikuti dengan tingkat selanjutnya, jika ada.

Disain Bottom Up (*Bottom Up Design*)

Pada kasus tertentu mungkin akan lebih mudah mendisain dengan menggunakan pendekatan dari level bawah / rendah ke level atas.

Hal ini sering ditemui pada kasus sistem pengontrolan proses dimana perlatan pengontrolan hardware pada level terbawah menentukan bagaimana sistem tersebut disatukan (integrasi sistem).

Contoh :
Kita akan mendisain sebuah sistem pengujian mesin kendaraan. Kita harus mulai dengan menentukan hardware dasar atau komponen dasar yang terlibat – sensor mesin.

Lihat Gambar 7.2A. Bottom Up Design

Sensor umumnya dipasang pada alat digital atau analog, yang terpasang pada modul software pengendali (*drivers*) alat yang unik.

Lihat Gambar 7.2B. Bottom Up Design (lanjutan)

Software yang digunakan untuk mengontrol alat pengendali / *drivers* kemudian didisain di atas pengendali-pengendali tersebut.

Lihat Gambar 7.2C. Bottom Up Design (lanjutan)

Demikianlah sistem software didisain dari level bawah ke atas. Disain *Bottom Up* juga sangat cocok digunakan pada kasus dimana komponen software yang ada digabungkan dan disatukan dengan modul baru untuk membangun sebuah sistem.

7.3. PERTUKARAN DISAIN TINGKAT ATAS *(TOP LEVEL DESIGN TRADE – OFFS)*

Umumnya banyak disain tingkat atas yang dapat mencapai atau memperoleh hasil yang sama dalam sebuah sistem software.

Contohnya, disain tingkat atas (*top level*) pada gambar 7.1. hanya salah satu cara untuk memecahkan sistem ABC kedalam komponen-komponen utama.

Keputusan untuk membangun sendiri atau membeli merupakan keputusan yang khusus. Ada keuntungan dan kerugian pada setiap kombinasi dari item yang dibangun maupun yang dibeli.

Semakin banyak paket program yang anda beli, semakin berkurang pemrograman yang harus anda lakukan. Keputusan untuk membeli paket program lebih mudah dibandingkan harus membuat sendiri, akan tetapi lebih mahal, dan umumnya kurang efisien dibandingkan dengan program tertulis biasa yang sama.

Disain tingkat atas yang lain ada juga yang cocok. Salah satu masukkan mungkin adalah menghilangkan *INQUIRY, UPDATE* dan *REPORT GENERATION* dan menggunakan rutin *FILE HANDLER* yang umum untuk melakukan semua kegiatan akses file. TLD akses tersebut seperti pada gambar 7.3.

Lihat Gambar 7.3. (Another) Top Level Design

Disini ada lima program yang harus dibuat dan sedikit penurunan kinerja akan terlihat oleh karena pemanggilan yang sering *pada FILE HANDLER*, tetapi sistem akan menjadi lebih kecil. Setiap pilihan TLD memilki keuntungan dan kerugian dan melibatkan pertukaran dan kompromi.

Prioritas Disain (*Design Priorites*)

Pilihan TLD anda akan mempengaruhi hal-hal berikut ini :

- Biaya Sistem (*System Cost*)
- Waktu yang diperlukan untuk membangun sistem (*Time to Build The System*)
- Sifat mudah dipakai (*User Friendliness*)
- Kinerja (*Performance*)
- Ukuran Sistem (*System Size*)
- Kehandalan (*Reliability*)
- Kemampuan modifikasi (*Modifiability*)

Item-item ini harus menjadi prioritas, bersama dengan user pada waktu perencanaan sistem, pada saat pendefinisian dan analisis. Ini akan membuat pilihan TLD jauh lebih mudah.

7.5. DISAIN TINGKAT MENENGAH (*MEDIUM LEVEL DESIGN*)

Setelah TLD terpilih, kita harus membagi masing-masing fungsi atau komponen utama menjadi beberapa sub fungsi atau komponen. Kita akan lihat bagaimana hal tersebut dilakukan untuk menggabungkan sistem perusahaan Basketweaving. Diawali dengan memberi nomor setiap komponen utama pada TLD.

Lihat Gambar 7.4. Numbering System for the TLD

Disain *top down* ini dimulai dengan kotak menu. Diasumsikan bahwa komponen ini dipanggil ketika seluruh sistem dimulai dan menampilkan menu utama ke bagian register.

Lihat Tampilan Menu Utama

Kemudian program menunggu user untuk memindahkan *mouse*.

Sub fungsi utama komponen MENU adalah :
1. Memulai sistem dan menampilkan main menu
2. Menangani perpindahan *mouse*
3. Menangani tombol pada *mouse*
4. Pindah ke Menu *INQUIRY*, *UPDATE, WAREHOUSE* atau *REPORT* ketika dipilih
5. Menangani kesalahan-kesalahan seperti pada *on line help messages* untuk seluruh sistem
6. Mematikan sistem jika *QUIT* dipilih

Struktur diagram tingkat selanjutnya atau diagram rinci untuk komponen MENU akan tampak seperti berikut ini.

Lihat Gambar 7.5. Rincian Level Kedua

Level terendah dari suatu menu menggambarkan modul. Sebuah modul adalah bagian terkecil yang dapat di*test* dan di*compile*.

Aturan Penamaan (*Naming Conventions*)
Modul diberi nama untuk menunjukkan sistem, fungsi atau subfungsi yang diperlukan.

Aturan Penomoran (*Numbering Conventions*)
Nomor pada setiap kotak disusun dengan aturan sebagai berikut : Pada tiap-tiap tingkat terendah tambahkan sebuah titik dan angka bulat untuk nomor yang terletak di atas kotak. Angka bulat tersebut diurutkan dari kiri ke kanan.

7.6. KAMUS DISAIN (*DESIGN DICTIONARIES*)

Modul Kamus (*Module Dictionaries*)

Dictionary 1
Berdasarkan urutan angka sesuai dengan nomor komponen, berikan nama yang tetap, dan penjelasan singkat untuk setiap modul.

Contoh :

0.0	A0000000	*Amalgamated Basketweaving System*
1.0	AM000000	*Menu System*
1.1	AMST0000	*Startup, disp first menu, shutdown, etc.*

Dictionary 2
Berdasarkan urutan alphabet dengan nama komponen, berikan nomor yang tetap, dan penjelasan singkat untuk setiap modul.

Contoh :

A0000000	0.0	*Amalgamated Basketweaving System*
AM000000	1.0	*Menu System*
AMST0000	1.1	*Startup, disp first menu, shutdown*

Dictionary 3
Berdasarakan urutan alphabet dengan penjelasan singkat, berikan nomor komponen dan nama yang tetap.

Contoh :

Amalgamated Basketweaving System	0.0	A0000000
Menu System	1.0	AM000000
Startup, disp first menu, shutdown	1.1	AMST0000

Kamus Data Umum (*The Common Data Dictionary / CDD*)

Daftar alphabet menyusun semua parameter yang ditunjukkan pada tanda panah aliran data. Untuk setiap item menjelaskan tipe, panjang, batasan, dan modul yang digunakan. CDD ini kemudian akan berisi semua parameter lainnya yang didefinisikan pada level terendah dari pemrograman dan disain, sebagaimana field didefinisikan dalam sebuah file. CDD menjamin bahwa parameter akan konsisten berlaku dlam seluruh sistem.

7.7. MODUL TERSTRUKTUR, ATAU SEJAUH MANA ANDA DAPAT MERINCINYA ? (*Structured Modules, Or How Far Do You Break It Up* ?)

Sebuah modul terstruktur memiliki ciri-ciri sebagai berikut :

1. Berfungsi sepenuhnya sebagai fungsi tunggal.
   Misalnya dapat diterima, diedit, diformat ulang dan melewati parameter tunggal.

2. Ukurannya kecil.
   Ukuran yang ditetapkan berkisar antara 50 – 100 baris yang dapat dieksekusi atau paling banyak 2 halaman.

3. Dapat diprediksi.
   Semua ciri dapat terlihat dengan membaca kode program. Hal ini tidak dipengaruhi oleh kode tersembunyi dalam modul lain atau dalam sistem operasi.

4. Tidak tergantung (*Independent*)
   Perubahan dalam modul atau parameter tidak mempengaruhi sistem.

5. Meskipun hal ini tidak didefinisikan secara jelas dalam modul terstruktur, lihatlah kegunaannya kembali – suatu modul yang cukup lengkap dan umum mengakibatkan anda dapat menggunakannya pada aplikasi lain dengan memodifikasi sedikit mungkin.

7.8. DISAIN FILE (*FILE DESIGN*)

Dapatkan kinerja yang sesungguhnya (*Getting Real Performance*)

Anda mulai mendisain file dengan melihat hasil dari fase analisis, kebutuhan-kebutuhan dan disain level atas yang sejauh ini dihasilkan. Hasilnya akan tampak seperti item-item kotak pada gambar 7.6.

Lihat Gambar 7.6. Data Types

Sebagai contoh, persyaratan “Pendaftaran seorang siswa pada kursus tertentu” akan menghasilkan penambahan dalam daftar field disamping masing-masing kotak pada gambar 7.6.

Jika STUDENT INFO dan COURSE INFO adalah file yang terpisah, mereka akan dihubungkan dengan sebuah **key**. Tambahkan key STUD_NO dan CRS_NO dan logika akses dapat digambarkan dengan tanda panah, seperti pada gambar 7.7.

Lihat Gambar 7.7. Data types, keys dan access

Sekarang kita dapat menangani pendaftaran siswa sebaik-baiknya, seperti : “Memberikan nama siswa, mencari kursus apa saja yang ia ikuti” (akses file STUDENT FILE dengan nama, kemudian CRS_NO, lihat COURSE FILE dengan CRS_NO tersebut). Diagram dilanjutkan sampai semua pernyataan terpenuhi. Hasilnya dapat digambarkan pada gambar 7.8.

Lihat Gambar 7.8. Data types, key and access

Mengoptimalkan File (*Optimizing Files*)

Langkah selanjutnya adalah mengoptimalkan penyimpanan disk dengan mengurangi kerangkapan field-field dan file-file.

Pada STUDENT FILE, jika banyak siswa mempunyai alamat yang sama, seperti perusahaan yang sama, field alamat akan terulang. File ADDRESS dengan satu record setiap perusahaannya dan COMPANY_NO di record siswa menunjukkan hal itu. File ini juga dapat berisi alamat faktur yang dibutuhkan oleh FINANCE FILE.

Pada COURSE FILE item seperti DESCRIPTION, INSTRUCTOR, dan MATERIAL NO akan selalu terulang setiap mereka kursus di tempat yang sama. Kemudian file ini dipecah menjadi file baru yang bernama SCHEDULE FILE yang mempunyai item unik untuk setiap kursus yang dimulai, dan membiarkan COURSE FILE hanya sebagi informasi.

FINANCE FILE hanya dapat ditandai dengan STUD_NO atau COURSE_NO. Sudah ada beberapa file yang menggunakan key tersebut, yang biasanya menunjukkan field tersebut dalam file ini dapat digabungkan ke dalam file lain, jika informasi pembayaran dan tagihan akan digabungkan dengan siswa. FINANCE FILE tidak diperlukan. Hasil disain file dapat dilihat pada gambar 7.9.

Lihat Gambar 7.9. Data types, key and access

Mengoptimalkan Sejumlah Variabel Item-item (*Optimizing a Variable Number of Items*)

Dalam STUDENT FILE terdapat 2 field, informasi CRS_NO dan PYMNT, yang diulang untuk masing-masing kursus dari siswa yang mendaftar. Dengan cara yang sama akan diulang field-field dalam file SCHEDULE dan COURSE. Hal ini dapat diatasi dengan membuat program yang menggunakan file yang mempunyai ukuran yang berubah-ubah. Panjang dari sebuah record berubah sesuai dengan penambahan maupun penghapusan sebuah item. Metode ini dapat menghemat ruang penyimpanan.

Dalam hal lain, jika jumlah maksimum dari suatu variabel diketahui, maka dapat digunakan record yang memiliki panjang yang tetap. Sebagai contoh, jika suatu bidang kursus tidak akan pernah diambil oleh lebih dari 30 siswa, maka jumlah record di file SCHEDULE disediakan untuk 30 siswa. Metode ini menggunakan ruang penyimpanan yang lebih besar dari metode pertama, namun metode ini membutuhkan waktu proses yang lebih singkat. Selain itu record dengan ukuran tetap lebih mudah untuk didisain, dimengerti, dan dalam hal pemeliharaannya dibandingkan dengan record yang memiliki panjang yang berubah-ubah. Karena harga media penyimpanan yang semakin murah saat ini, maka disarankan untuk menggunakan record yang memiliki ukuran tetap jika memungkinkan.

Sebuah masalah dapat muncul jika batasan tidak dapat ditentukan. Sebagai contoh, Mengapa jumlah siswa di setiap bidang kursus tidak dapat dibatasi dengan jumlah yang sama ? Untuk mengatasi hal ini dapat dibuat suatu file lain yang hanya digunakan untuk menyimpan informasi yang tidak tetap. File ENROLLMENT dapat dibuat untuk setiap bidang kursus, dan setiap file akan berisi satu record per satu siswa yang mendaftar per kursus, seperti pada gambar 7.10. Hal ini mungkin membutuhkan biaya yang besar, baik dalam hal media penyimpanan maupun pemeliharaan file.

Lihat Gambar 7.10. Handling variable information

Cara sederhana untuk menangani masalah tersebut adalah dengan membuat sebuah file yang disebut ENROLLMENT yang akan berisi record untuk setiap siswa yang mendaftar per kursus. Field-field yang digunakan hanya STUDENT_NO dan COURSE_NO.

File Histori (*History Files*)

Apa yang kita lakukan tentang data pada siswa-siswa yang telah mengambil sebuah kursus ? Pemecahan masalah ini dengan mendefinisikan sebuah file STUDENT_HISTORY dan setelah seorang siswa mengambil sebuah kursus, recordnya dipindahkan dari file STUDENT ke File Histori.

Pengujian Disain File (*Testing The File Design*)
Pada disain ini, setiap permintaan kebutuhan yang melibatkan pengaksesan data harus "diproses" dengan disain file. Hal ini menandai perkembangan selanjutnya.

Contoh :
"Tampilkan semua peristiwa pada tempat pelatihan XYZ, lokasi dan biayanya". Mari kita mengikuti logika pengaksesannya. Register mengubah nama bidang kursus menjadi CRS_NO. Record-record pada file SCHEDULE diakses oleh CRS_NO untuk mendapatkan biaya. Dalam permintaan yang umum seperti ini, mungkin nama bidang kursus dapat dijadikan "key" pada file SCHEDULE. Mungkin harga dapat ditambahkan ke file SCHEDULE untuk mengurangi pengaksesan file COURSE setiap waktu. Untuk menghemat ruang penyimpanan, kode biaya dapat digunakan.

7.9. SISTEM MANAJEMEN DATA BASE RELATIONAL (*RELATIONAL DATA BASE MANAGEMENT SYSTEM / RDBMS*)

Pada bagian 7.8. kita mengasumsikan bahwa anda mendapatkan sebuah record dari sebuah file yang mengandung sebuah kunci. Dalam kenyataannya, harus terdapat DBMS untuk memenuhi hal ini.

Gambar 7.11 adalah contoh dari suatu relasi (tabel) yang dapat didefinisikan pada sistem ABC.

Student Relations :

Stud-No	Stud-Name	Company-No
1	John Blake	999
2	Jane Smith	999

Run of a Course Relations :

Course-No	*Course-Date*	*Location*	*Instructor*	*Cost*
123	1/1/90	Ottawa	Rakos	1000
123	1/2/90	New York	Rakos	1500

Enrollment Relations :

Course-No	*Stud-No*	*Pymnt*
123	1	0

Course Relations :

Course-No	*Crs-Name*	*Desc*	*Mat-No*
123	Weaving	Intro	001

Company Relations :

Comp-No	*Addr*	*Ship-To*	*Bill-To*	*Tot-Owing*
999	First ST.	X Y	A B	10000

Material Relations :

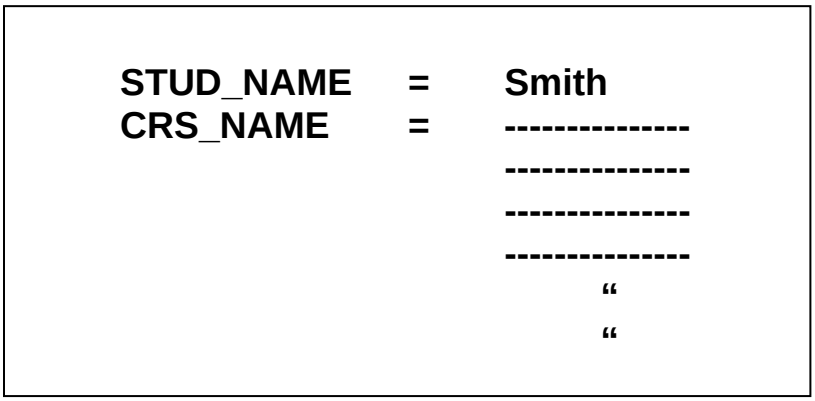
Mat-No	*Desc*	*Whse*	*Source*	*Cost*
001	Straw	1-1	X Co	1.00

Gambar 7.11 Relasi pada Data Base Relational ABC

Bentuk untuk jenis *database query* khusus ini sudah distandarisasi, dan ini disebut ***Structured Query Language (SQL)***. Sebagai contoh, berikut ini adalah instruksi SQL untuk menampilkan kursus apa saja yang diikuti oleh “Smith”.

```
SELECT CRS-NAME FROM COURSE
WHERE COURSE-NO IN
      SELECT COURSE-NO FROM ENROLLMENT
      WHERE STUD-NO IN
            SELECT STUD-NO FROM STUDENT
            WHERE STUD-NAME = ‘Smith”
```

Sayangnya SQL yang dibuat oleh IBM memiliki banyak kekurangan sehingga produk-produk baru mempunyai perluasan bahasa untuk menyediakan keistimewaan tambahan. Anda dapat melihat banyak bahasa 4GL yang baru (yang mendukung *Query By Example* [QBE]) untuk mengisi formulir di bawah ini :



Kelebihan utama dari RDBMS adalah fleksibilitasnya.
Sebagai contoh : jika kita ingin mengakses data yang sejenis secara berlainan dari berbagai aplikasi, maka sistem ini akan dapat menampungnya.

Kekurangan RDBMS hanya pada *performance*. Membutuhkan banyak memori dan waktu untuk menyimpan, menjalankan dan merinci seluruh bagian tabel.

Penghematan waktu yang diakibatkan oleh pemakaian yang mudah dan fleksibilitas dari sistem, membuat RDBMS yang terdapat pada sebuah komputer (CPU) yang baik sebagai investasi yang berharga.

7.10. KEUNTUNGAN DARI ANALISIS & DISAIN YANG TERSTRUKTUR (*BENEFITS OF STRUCTURED ANALYSIS AND DESIGN*)

Mengurangi Jumlah Kesalahan (*Reducing the Number Of Initial Errors*)

Tabel statistik berikut ini diambil dari hasil survei oleh TRW untuk proyek besar, dan DEC’s Customer Services Systems Engineering (yaitu departemen yang bertanggung jawab untuk memastikan bahwa produk-produk DEC baik software maupun hardwarenya benar-benar bebas dari virus).

Menggunakan metode tidak terstruktur :

	Analysis and Design	Remaining Phase	After Operation
Effort Spent :	10 %	23 %	67 %
Problem Introduced :	64 %	36 %	
Problems Found :	19 %	27 %	54 %
Dollars Spent (AVG) (Total = 250 K)	25 K	57.5K	167.5 K

Menggunakan metode terstruktur :

	Analysis and Design	Remaining Phase	After Operation
Effort Spent :	20 %	50 %	30 %
Problem Introduced :	32 %	68 %	
Problems Found :	30 %	33 %	37 %
Dollars Spent (AVG) (Total = 190 K)	40 K	50 K	100 K

(1) Bagan di atas adalah setengah dari biaya sebelumnya

Gambar 7.12. Causes and costs of problems

Gambar 7.12 menunjukkan bahwa meskipun biaya dimuka mengalami kenaikan, metode terstruktur tetap mengurangi biaya sistem secara keseluruhan.

7.11. PROSES DISAIN (*THE DESIGN PROCESS*)

Tim Disain (*The Design Team*)

Pilihlah orang-orang terbaik untuk tim disain. Tim disain yang baik tidak perlu orang yang menguasai bahasa pemrograman. Mereka haruslah orang yang dapat mengkonsep semuanya. Hindari orang-

orang yang selalu menginginkan kesempurnaan (*perfectionis*) dalam tim disain.

Pertemuan Disain (*The Design Meeting*)

Merancang sesuatu mirip dengan urun remuk (*brainstorming*) : beberapa orang berkumpul dalam suatu ruangan yang tenang dan tidak terganggu. Setiap orang diharapkan untuk mengeluarkan semua ide mereka agar semua elemen yang berfungsi dapat digunakan dan juga memikirkan bagaimana cara menguasainya. Tulis semua ide yang ada, dan kemudian akhirnya ide-ide yang ada disusun ke dalam modul-modul yang unik.

7.12. DOKUMENTASI TEKNIK (*TECHNICAL DOCUMENTATION*)

Pertimbangkan hal-hal berikut ketika menulis dokumentasi teknik :

1. Gunakan bahasa yang formal dan tepat.
2. Gunakan gambar, diagram yang terstruktur dan yang sejenisnya.
3. Buatlah agar maksud dari disain menjadi jelas pada beberapa halaman pertama. Kemudian uraikan.
4. Cobalah untuk konsisten pada grafik-grafik dan struktur kalimat.

7.13. STANDAR KETENTUAN PADA WAKTU DISAIN (*STANDARDS 'DICTATED' AT DESIGN TIME*)

Buatlah aturan yang standar seperti berikut :

1. Beberapa ketentuan disain.
   Format struktur diagram, modul dan kaidah penamaan variabel ini digunakan untuk semua tingkatan yang rendah.

2. Parameter yang mendahului.
   Rincian perintah, panjang, format / tipe.

3. Penanganan kesalahan.
   Setiap modul melewati keadaan dimana kesalahan muncul dan nomor kesalahan untuk ditangani.

4. Standar pemrograman.
   Standar pemrograman terstruktur seperti pemunculan kode (*white space*, memasukkan, komentar-komentar), konsep yang diperbolehkan, organisasi, ukuran modul, dan ketergantungan secara rinci. Membuat *'template'* yang berisi komentar seperti :
   - judul
   - parameter-parameter (penerima, pengirim)
   - masukkan (hanya satu)
   - variabel yang digunakan
   - memanggil *subroutine*
   - penanganan kesalahan
   - *exit* (hanya satu)

5. Programmer memulai dengan *template* ini dan mengisikan kode proses. Lihat bagian 9.4 untuk alat pemrograman yang membantu format program secara tetap.

7.14. GARIS BESAR SPESIFIKASI DISAIN (*OUTLINE OF THE DESIGN SPECIFICATION*)

Spesifikasi disain terdiri atas :
1. Judul halaman dan daftar isi
2. Gambaran umum (Overview)
3. Daftar hardware / software yang akan dipakai
4. Daftar urutan prioritas disain
5. Diagram disain dan beberapa modul dictionary yang umum
6. Beberapa kebiasaan penamaan modul yang umum
7. Parameter yang dipakai dan Data Dictionaries
8. Penanganan kesalahan. Jelaskan bagaimana kesalahan akan ditangani
9. Standar pemrograman terstruktur
10. Alat pemrograman terstruktur
11. Top Level Design. Termasuk struktur diagram TLD
12. Medium Level Design. Termasuk struktur diagram MLD
13. Modul dan kamus data
14. File dan Tabel

7.15. MENGUJI DISAIN (*TESTING THE DESIGN*)

Ketika disain telah diselesaikan, semuanya harus berjalan dengan benar. Maksudnya adalah untuk menjamin hal-hal berikut ini :

1. Semua keperluan Spesifikasi Fungsi sudah ketemu.
2. Disain mudah diprogram dan dipelihara.
3. Dapat diimplementasikan sesuai dengan waktu dan anggaran.

7.16. MERUBAH PERMINTAAN SESUAI DENGAN DISAIN (*CHANGES TO REQUIREMENTS DUE TO DESIGN*)

Beberapa disain yang rinci akan selalu membawa pada perubahan permintaan. Anda harus kembali kepada user sekarang dan meyakinkan user bahwa dia sebenarnya tidak menginginkan apa yang dia minta sebelumnya.

7.17. PERENCANAAN PENERIMAAN (*PLANNING THE ACCEPTANCE*)

Meskipun penerimaan adalah tahapan pada bab tersendiri nantinya, perencanaan penerimaan dapat dimulai setelah disain level menengah dikerjakan.

7.4. DISAIN SIAP PAKAI (*DESIGN WALK-THROUGHS*)

Ketika hendak mengambil keputusan diantara beberapa pendekatan teknis terhadap suatu masalah, lebih mudah mengambil keputusan dengan meminta pendapat orang lain.

Adakan pertemuan dengan beberapa ahli untuk melakukan TLD siap pakai. Tujuan dari pertemuan itu adalah memilih TLD yang terbaik.

Kisah Tentang Perang Disain

Cerita ini melibatkan dua orang pemuda yang menjadi sahabat karib ketika mereka mengikuti kuliah Ilmu Komputer pada sebuah universitas terkemuka di Ontario, Kanada. Setelah lulus yang satu menjadi menjadi seorang Manajer Pemrosesan Data pada sebuah perusahaan penerbitan terkenal, dan yang satu menjadi seorang perancang sistem pada Famous Minicomputer Manufacturing Company (FMMC). Beberapa tahun kemudian perusahaan penerbit memutuskan untuk memasang komputer baru untuk sistem pengolahan. Manajer Pemrosesan Data tidak mamapu menangani beban pekerjaan itu sendiri, maka ia menyewa temannya dari FMMC untuk mengerjaka desain dan program.

Setelah mereka mengerjakan top level desain bersama-sama, mereka membagi modul utama, yang masing-masing mengerjakan desain medium level dari modul-modul yang spesifik. Perancang dari FMMC mendesain modul yang pertama. Untuk menunjukkan kehebatan desainnya pada temannya, ia membuat rancangan itu dengan sangat “bagus” untuk menghemat sedikit byte dari media penyimpanan atau beberapa nanodetik dari waktu CPU. Ia bahkan menambahkan sedikit “keistimewaan” tersendiri : sesuatu yang tidak seorangpun meminta atau mengerti atau pernah menggunakannya.

Si Manajer Pemrosesan Data menanggapinya dengan mendesain modul yang kedua. Ketika dia melihat rancangan hebat milik temanya, ia mengatakan “saya dapat membuat yang lebih bagus dari itu”. Lalu ia membuat modulnya sedikit lebih bagus dari modul temannya, dan menambahkan sedikit “suara bel dan melodi-melodi” pada modulnya. Ia mengerjakan lebih lama dari jadwal yang ditentukan dan modulnya menjadi lebih besar dari rencana. Si perancang tertantang untuk membuat modul yang ketiga yang lebih “sempurna” dengan menambah lebih banyak lagi suara bel-bel dan melodi sampai modul menjadi dua kali lebih besar dari seharusnya. Dan perlombaan terus berlanjut sampai mendesain sebagus mungkin tahap pemrograman.

Sistem tersebut dibuat untuk 16 orang user. Ketika sistem akhirnya berjalan (50% terlambat) sistem bekerja baik sampai 4 orang user. Ketika 6 orang user bekerja, sistem menjadi sangat lambat, dengan 8 orang user sistem crash – program menjadi sangat besar untuk dapat ditangani oleh sistem.

Komentar : Orang teknis dapat terbawa dengan tantangan teknis. Selalu banyak jalan yang baik untuk menyelesaikan masalah, tapi tantangan harus menghasilkan tujuan, yaitu waktu dan biaya.

Penutup : FMMC memberikan CPU yang besar kepada perusahaan penerbit tanpa biaya.