

SERI DIKTAT KULIAH

PENGANTAR
**IMPLEMENTASI DAN
IMPLEMENTASI
SISTEM INFORMASI**

D. SURYADI H.S.
BUNAWAN

UNIVERSITAS GUNADARMA

DAFTAR ISI

BAB 1 PENGEMBANGAN PERANGKAT LUNAK	1
Apa yang Akan Anda Pelajari dalam Bab Ini?	1
Pendahuluan	2
Sumber Perangkat Lunak Aplikasi	4
Mengorganisasi Proyek Pengembangan Perangkat Lunak	12
Mengukur Produktivitas dalam Pengembangan Perangkat Lunak	20
Memproduksi Perangkat Lunak Berkualitas Tinggi	28
Merencanakan Proyek Siklus Hidup Pengembangan Perangkat Lunak	31
Tinjauan Sasaran Belajar untuk Bab Ini	36
Daftar Periksa Pengembangan Perangkat Lunak	40
Pertanyaan Tinjauan	41
Soal Spesifik Bab Ini	43
Soal Umum	46
Bacaan yang Disarankan	57
BAB 2 PERANCANGAN PERANGKAT LUNAK	58
Apa yang Akan Anda Pelajari dalam Bab Ini?	58
Pendahuluan	59
Alasan Melaksanakan Tahap Rancangan perangkat Lunak	63
Menggunakan Rancangan Perangkat Lunak Terstruktur	66
Apa yang Dimaksud Rancangan Perangkat Lunak Berorientasi-obyek?	85
Menggunakan Pendekatan Rancangan Berorientasi-obyek untuk Merancang Model Perangkat Lunak	89

Melaksanakan Penelusuran Rancangan Perangkat Lunak	97
Tinjauan Sasaran Belajar untuk Bab Ini	99
Daftar Periksa Rancangan Perangkat Lunak	102
Pertanyaan Tinjauan	104
Soal Spesifik Bab Ini	105
Soal Umum	113
Bacaan yang Disarankan	114
BAB 3 PENGKODEAN PERANGKAT LUNAK	120
Apa yang Akan Anda Pelajari dalam Bab Ini?	120
Pendahuluan	121
Apa Perbedaan Antara Bahasa Generasi-keempat dan Bahasa Generasi-ketiga?	124
Bahasa Pemrograman Berorientasi-Obyek	130
Perangkat Bahasa Penggunaan-Khusus	138
Memilih Bahasa yang Tepat	152
Dokumentasi Perangkat Lunak	156
Dokumentasi Operasi	163
Dokumentasi Pemakai	166
Tinjauan Sasaran Belajar untuk Bab Ini	174
Daftar Periksa Pengkodean Perangkat Lunak	179
Pertanyaan Tinjauan	179
Soal Spesifik Bab Ini	180
Soal Umum	183
Bacaan yang Disarankan	185
BAB 4 PENGUJIAN PERANGKAT LUNAK	192
Apa yang Akan Anda Pelajari dalam Bab Ini?	192
Pendahuluan	193
Apa Pengujian Perangkat Lunak Itu?	194
Merancang Test Case	199
Mengembangkan Strategi Pengujian Perangkat Lunak	208
Tinjauan Sasaran Belajar untuk Bab Ini	222
Daftar Periksa Pengujian Perangkat Lunak	225
Pertanyaan Tinjauan	226
Soal Spesifik Bab Ini	227

Soal Umum	228
Bacaan yang Disarankan	230
BAB 5 PENGIMPLEMENTASIAN SISTEM	236
Apa yang Akan Anda Pelajari dalam Bab Ini?	236
Pendahuluan	237
Menciptakan Rencana Implementasi Sistem	237
Menyiapkan Tempat	240
Pelatihan Personel	241
Menyiapkan Dokumentasi	249
Mengkonversi Sistem Baru	253
Metode untuk Mengkonversi File Data yang Ada	257
Mengevaluasi Sistem Baru Setelah Implementasi	260
Tinjauan Sasaran Belajar untuk Bab Ini	275
Daftar Periksa Implementasi Sistem	279
Soal Tinjauan	279
Soal Spesifik Bab Ini	280
Soal Umum	282
Bacaan yang Disarankan	287
BAB 6 PEMELIHARAAN SISTEM	295
Apa yang Akan Anda Pelajari dalam Bab Ini?	295
Pendahuluan	296
Pemeliharaan Sistem	296
Prosedur untuk Pemeliharaan Sistem	302
Menggunakan Case Tools untuk Memelihara Sistem	305
Mengelola Pemeliharaan Sistem	309
Mengembangkan Sistem Manajemen Berubah	316
Tinjauan Sasaran Belajar untuk Bab Ini	323
Daftar Periksa Pemeliharaan Sistem	326
Soal Tinjauan	327
Soal Spesifik Bab Ini	328
Soal Umum	331
Bacaan yang Disarankan	333

1

PENGEMBANGAN PERANGKAT LUNAK

APA YANG AKAN ANDA PELAJARI DALAM BAB INI?

Setelah mempelajari bab ini, anda diharapkan dapat :

- Membedakan sumber-sumber perangkat lunak aplikasi dan membahas bagaimana mengevaluasi dan menyeleksi paket-paket perangkat lunak.
- Mendifinisikan atau menetapkan siklus hidup pengembangan perangkat lunak (SWDLC) dan secara singkat membahas tahap-tahapnya.
- Membahas pengorganisasian proyek pengembangan perangkat lunak; menetapkan tim pengembangan program, tim programer kepala, dan tim pemrograman bersama; dan menganalisis manfaat dan kerugiannya.
- Menetapkan produktivitas perangkat lunak menghadirkan dua cara untuk mengukur produktivitas ini. Dan juga menjelaskan dampak manajemen terhadap produktivitas perangkat lunak.
- Membahas kualitas perangkat lunak dan menjelaskan jaminan kualitas dan pengendalian kualitas.
- Mandeskripsikan bagaimana teknik peninjauan dan evaluasi program (PEV) digunakan sebagai teknik manajemen proyek dan game plan SWDLC.

PENDAHULUAN

Bab ini membahas pengembangan perangkat lunak, salah satu tugas pengimplementasi terbesar dan paling menantang (lihat Gambar 1.1). Pengembangan perangkat lunak berjangkauan antara dua sisi ekstrim, dari sindrom “spreadsheet untuk setiap aplikasi” sampai sindrom “reinventing the wheel”. Sindrom pertama terjadi karena beberapa orang percaya bahwa untuk setiap aplikasi terdapat spreadsheet yang *ready-made* (siap buat) atau terdapat beberapa paket perangkat lunak komersial yang akan menjalankan aplikasi tersebut. Yang harus kita laksanakan adalah membelinya dan menginstalnya, dan masalah perangkat lunak aplikasi terpecahkan. Di sisi lain, orang-orang sistem mengembangkan program komputer baru dari *scratch* (pembuatan sendiri dari awal) untuk setiap aplikasi sistem tanpa mempedulikan apa yang telah dikembangkan secara *in-house* (swa-kelola) atau apa yang tersedia dari vendor (penjual) perangkat lunak.

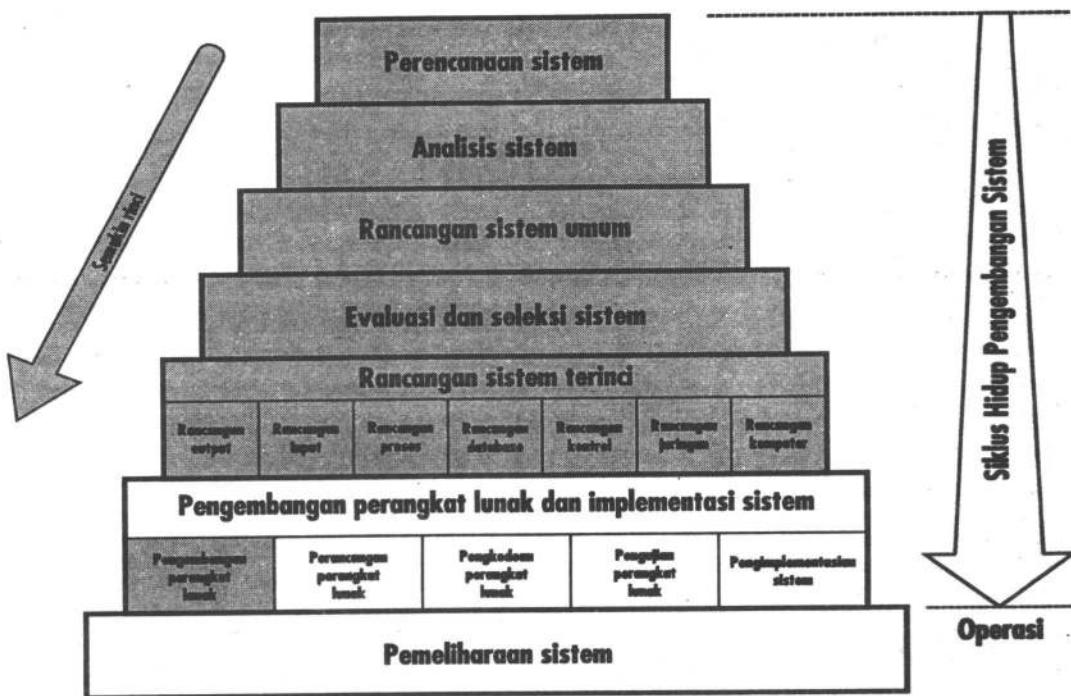
Dalam banyak kasus, setelah atau hampir selesainya tahap rancangan sistem rinci dari SDLC, tim proyek sistem mungkin mulai mencari paket perangkat lunak komersial yang sesuai atau mendukung spesifikasi rancangan sistem dan berjalan pada arsitektur komputernya. Paket perangkat lunak komersial secara luas tersedia untuk aplikasi fungsi spesifik dan aplikasi bisnis yang telah ditetapkan secara baku.

Namun demikian, dalam rancangan sistem yang berurusan dengan kebutuhan pemakai khusus atau unik, paket perangkat lunak tertulis mungkin tidak akan sesuai atau mendukung kebutuhan pemakai ini secara langsung. Oleh karena itu, perangkat lunak yang diharapkan untuk mendukung rancangan sistem seperti itu harus ditulis (dibuat) dari *scratch* agar memenuhi keperluan pemakai dan spesifikasi rancangan sistem.

Setelah anda mengenal sumber perangkat lunak, kita akan mengasumsikan bahwa rancangan sistem *terinci* memerlukan perangkat lunak yang dikembangkan secara *in-house*. Dengan demikian, bagian selanjutnya dari bab ini akan membahas pengorganisasian pengembangan perangkat lunak, pengukuran produktivitas pengembangan perangkat lunak, pemastian kualitas perangkat lunak, dan persiapan

Pengembangan Perangkat lunak di Tranquil Industries

Tranquil Industries adalah pabrikan kecil yang memproduksi perangkat peredam suara sesuai pesanan untuk mobil, pesawat udara, dan peralatan lain. Pada awalnya, Jim



Gambar 1.1 Tahap SDLC dan bab yang terkait dalam buku ini. Bab 1 berkenan dengan sumber-sumber perangkat lunak, diikuti dengan berbagai aspek pengembangan perangkat lunak.

Thompson, pimpinan proyek sistem, dan para anggota timnya ditugasi untuk mengembangkan sistem akunting dan sistem informasi eksekutif (EIS) untuk manajer pabrik dan para stafnya.

Tim proyek sistem Jim ini baru saja menyelesaikan Laporan Rancangan Sistem Terinci mereka, yang berisi lebih dari 300 dokumen, yang ditujukan dan didistribusikan ke manajemen dan end user yang tepat. Bagian laporan yang menyangkut atau mengenai end user tertentu atau manajer tertentu disorot (diberi tekanan) sehingga end user bisa melihat bagian yang menjadi kepentingannya saja. Sebagai contoh, pengontrol terutama berkepentingan dalam pengakuntansian rancangan output, input, proses dan rancangan kendali. Manajer pabrik hanya berkepentingan dalam rancangan output grafis. Administrator database berkepentingan dalam rancangan database. CIO hanya perlu tahu cara pengkonfigurasian platform teknologi dan hasil-hasil dari permintaan akan proposal dan *benchmark*.

Pada saat ini, semua pihak yang berkepentingan diberitahu secara rinci mengenai rancangan sistem baru. Sekarang, Jim telah siap untuk memulai tugas-tugas pengembangan perangkat lunak, akuisisi (pemerolehannya), dan tugas-tugas implementasi sistem EIS, misalnya, harus didukung oleh program perangkat lunak customized yang dikembangkan oleh tim pemrograman in-house (swa-kelola). Beberapa pekerjaan awal dengan beberapa vendor perangkat lunak menunjukkan bahwa sistem akunting akan didukung oleh salah satu dari tiga paket perangkat lunak akunting modular, yang akan dievaluasi lebih jauh untuk menentukan yang terbaik. Setiap paket perangkat lunak akunting berisi buku besar, piutang dagang, hutang dagang, inventari, pesanan penjualan, penggajian, dan modul-modul biaya pesanan pekerjaan yang perlu dimasukkan dalam Laporan Rancangan Sistem Terinci.

Bab ini dan bab-bab berikutnya memerinci apa-apa yang harus dikerjakan Jim dan para profesional sistem anak buahnya untuk mengkonversi Laporan Rancangan Sistem Terinci, yang merupakan “blueprint” dari sistem baru, ke sistem terkonstruksi dan terimplementasi yang siap beroperasi. Materi dalam bab ini dan bab-bab berikutnya tidak khusus ditujukan untuk *Tranquil Industries* saja, namun juga untuk semua organisasi yang telah mengembangkan Laporan Rancangan Sistem Terinci yang berisi semua materi yang secara penuh dan secara fungsional menjabarkan output, input, proses, database, kendali, dan platform teknologi dari sistem baru itu. Pada kenyataannya, beberapa konsep dan teknik yang dikemukakan tidak akan berkaitan dengan Laporan Rancangan Sistem Terinci dari *Tranquil* tersebut, sebab laporan ini tidak memerlukan konsep dan teknik seperti itu. Sebagai contoh, kita akan membahas rancangan berorientasi obyek dalam Bab 2, namun disini kita juga akan membahas *Tranquil* dalam mengembangkan EIS dengan menggunakan rancangan terstruktur.

rencana pekerjaan untuk pengembangan perangkat lunak. Akan disertakan suatu kasus pelengkap yang memberi anda wawasan mengenai bagaimana tim proyek sistem bergerak dari tahap rancangan sistem terinci ke tahap implementasi sistem, yakni dari rancangan ke konstruksi.

SUMBER PERANGKAT LUNAK APLIKASI

Ada dua sumber pokok perangkat lunak aplikasi:

- ◆ Perangkat lunak komersial dari Vendor
- ◆ Perangkat lunak terkustomisasi (eustomized software) yang dikembangkan secara in-house atau oleh kontraktor pemrograman independen.

Dari salah satu sumber tersebut, perangkat lunak harus memenuhi spesifikasi rancangan yang telah ditetapkan dalam Laporan Rancangan Sistem Teinci, yang dikembangkan berdasarkan kebutuhan pemakai. Dengan demikian, SDLC bisa diterapkan untuk pengembangan segala sistem, apakah perangkat lunak yang diperoleh merupakan *paket lengkap* ataupun dikembangkan dari *scratch*.

Paket Perangkat Lunak Komersial

Paket perangkat lunak komersial yang tersedia bisa diterapkan dalam berbagai kebutuhan bisnis. Beberapa paket bersifat generik dan multifungsional, yang memungkinkan para pemakai memprogram perangkat lunak tersebut untuk kebutuhannya sendiri. Contohnya adalah spreadsheet dan DBMS. Paket-paket lain merupakan aplikasi yang bersifat spesifik. Contoh jenis paket ini adalah perencanaan kebutuhan bahan, buku besar, piutang dagang, hutang dagang, penggajian, biaya pesanan pekerjaan, dan pengolah kata. Paket-paket ini mengotomatisasi fungsi-fungsi bisnis dasar yang umumnya tidak terlalu bervariasi dari satu organisasi dengan organisasi yang lain.

Keuntungan/Kelebihan

Keuntungan membeli dan mengimplementasikan paket perangkat lunak komersial biasanya meliputi berikut ini:

Implementasi yang Cepat. Keuntungan/kelebihan pokok dari perangkat lunak yang dibeli adalah pengimplementasiannya yang cepat. Perangkat lunak tersebut bersifat siap, teruji, dan terdokumentasi. Tergantung pada ukuran aplikasinya, suatu organisasi biasanya bisa mengimplementasikan paket yang dibeli secara jauh lebih cepat dari pada jika ia mengembangkan program yang sama secara *in-house* atau dari pada jika ia menyuruh kontraktor independen untuk mengembangkannya. Keuntungan ini secara potensial membantu memecahkan backlog (masalah penimbunan pekerjaan yang belum selesai) yang mengganggu sejumlah organisasi.

Penghematan Biaya. Biaya total suatu paket perangkat lunak komersial akan lebih murah daripada program terkustomisasi yang sama, karena satu paket bisa dijual kepada banyak organisasi, sehingga biaya pengembangan ditanggung oleh banyak pemakai. Kita ragu bahwa program pesanan (*customized program*) bisa dikembangkan dengan biaya yang lebih murah dari pada biaya atau harga paket komersial *off-the-shelf*.

Estimasi Biaya dan Waktu. Biaya atau harga paket komersial telah diketahui. Juga, tanggal pengimplementasinya mudah diestimasi. Pengembangan terkustomisasi, sebaliknya, biasanya cenderung melampaui estimasi waktu dan biaya.

Reliabilitas. Paket perangkat lunak komersial pasti telah diuji secara teliti sebelum ia diterbitkan ke pasaran umum. Melalui penggunaan yang ekstensif oleh sejumlah organisasi, segala kesalahan yang dijumpai telah dideteksi dan dikoreksi. Meskipun tak ada program yang dinyatakan bebas dari kesalahan sepenuhnya, peluang paket komersial akan mempunyai lebih sedikit kesalahan dari pada program dengan ukuran yang sama dan kompleksitas yang lebih besar.

Kerugian/kelemahan

Kerugian paket perangkat lunak komersial mencakup berikut ini:

Kesesuaian Rancangan Sistem yang Tidak baik. Walaupun kita umumnya yakin bahwa paket perangkat lunak komersial lebih cepat dan lebih mudah diimplementasikan dari pada perangkat lunak yang dikembangkan secara *in-house*, namun hal ini tidak mesti demikian. Sebab paket perangkat lunak komersial ditulis atau dibuat untuk berbagai organisasi, tidak untuk organisasi tertentu, maka paket ini mungkin akan mempunyai beberapa fungsi yang tidak diperlukan atau mungkin tidak mempunyai fungsi-fungsi yang diperlukan. Dalam beberapa kasus, paket itu sendiri harus dimodifikasi. Jika vendor tidak membuat kode sumber (*source code*) yang bisa digunakan untuk penyesuaian dan tidak menyediakan layanan penyesuaian, maka rancangan sistem mungkin harus diubah agar sesuai atau cocok dengan paket tersebut. Apabila salah satu hal tersebut terjadi, sebaiknya kita mengembangkan program secara *in-house* agar program ini bisa memenuhi spesifikasi rancangan sistem secara tepat.

Ketergantungan pada Vendor. Organisasi akan tergantung pada vendor dalam perolehan dukungan. Organisasi tersebut akan kesulitan mencari dukungan jika vendor itu telah tiada, sedangkan organisasi memerlukan perubahan paketnya.

Biaya Tidak Langsung dari Kerusakan SDLC. Dalam beberapa kasus, manajemen mungkin ingin melewati tahap SDLC dan secara langsung menuju ke paket perangkat lunak komersial. Prioritas melaju dengan strategi quick-fix (pemasangan yang cepat) adalah dengan menjadikan sistem teknis terbangun dan bisa

dijalankan dan memenuhi kebutuhan pemakai yang spesifik, pelatihan, dan mengatasi masalah sistem dan organisasional kemudian. Strategi seperti ini biasanya terlalu mendadak. Seringkali paket perangkat lunak komersial tidak berjalan sesuai yang diharapkan, dan masalah sistem dan organisasional yang terjadi sebelum implementasi paket tersebut tetap muncul. Sebenarnya salah satu alasan pokok menerapkan SDLC di tempat pertama adalah untuk mengatasi masalah sistem dan organisasional dan meningkatkan kinerja sistem. Seringkali, apa yang dicapai dengan tidak melaksanakan SDLC menyeluruh akan menimbulkan kesulitan atau harus dibayar kemudian, yaitu adanya peningkatan biaya implementasi, operasi, dan pemeliharaan.

Menyiapkan Permohonan untuk Proposal (Request For Proposal-RFP) Berorientasi-Kinerja

Dalam kasus dimana perangkat lunak komersial bisa diterapkan, keuntungan yang diperoleh dari perangkat lunak komersial jauh lebih besar dari pada kerugiannya. Dengan demikian, sekarang kita akan mengalihkan perhatian pada persoalan-persoalan yang berkaitan dengan akuisisi (pemerolehan) perangkat lunak komersial.

Langkah pertama dalam menyeleksi vendor yang tepat dari paket perangkat lunak komersial adalah dengan menyiapkan atau membuat RFP berorientasi-kinerja yang sama dengan RFP untuk rancangan arsitektur komputer. Faktor-faktor evaluasi utama mencakup pemenuhan spesifikasi rancangan terinci untuk output, input, proses, dan database yang ditetapkan dalam Laporan Rancangan Sistem Terinci. RFP harus berisi kendali yang telah ditetapkan selama tahap rancangan terinci. Lebih dari itu, ia harus memenuhi (cocok dengan) batasan waktu dan biayanya. Penggunaan benchmark yang mensimulasi kebutuhan sistem baru harus diterapkan pada setiap paket dari vendor.

Penilaian Paket

Setiap paket harus dinilai. Sebagian dari penilaian mungkin dihasilkan dari benchmark. Penilaian lain bisa diperoleh dari sejumlah publikasi yang didasarkan pada survei dari sejumlah besar pembaca yang menggunakan paket tersebut.

Kinerja Pengoperasian (Operating performance). Penilaian kinerja ini adalah hasil dari benchmark yang dijalankan pada paket yang digunakan untuk mengukur hal-hal seperti transaksi per detik (Transactions Per Second-TPS) dan waktu respon (Response Time-RT).

Dokumentasi. Penilaian dokumentasi mencerminkan kuantitas dan kualitas prosedur tertulis maupun prosedur online, termasuk pedoman quick-start, online tutorial, dan fasilitas help. Pengorganisasian yang tidak baik, informasi yang tidak lengkap, indeks yang tidak lengkap, dan ambiguitas akan membuat rendah penilaian.

Kemudahan Pembelajaran. Penilaian untuk faktor kinerja ini tergantung pada interface pemakai dan rancangan intuitif dari paket tersebut. Kualitas dokumentasi juga mempengaruhi penilaian ini. Paket harus bisa dipelajari oleh rata-rata pemakai.

Kemudahan Penggunaan. Faktor kinerja ini berada dalam bagian terbesar dari fungsi rancangan paket. Ia menentukan seberapa mudahnya pemakai rata-rata menggunakan paket tersebut apabila pemakai telah memahami dasar-dasarnya. Menu yang mudah diikuti dan perintah yang jelas membantu kemudahan penggunaan.

- ◆ **Pengendalian dan Penanganan Kesalahan.** Kita telah mengetahui jenis-jenis kendali, terutama kendali input yang harus dilekatkan dalam logika program. Pengendalian ini mengecek input untuk rentang (range), kelayakan, nilai-nilai lain dari input itu; mengakses dan membandingkan nilai-nilai dalam tabel identifikasi (atau validitas); dan mengkomputasi ulang digit pengecekan. Semua kesalahan yang terdeteksi harus membangkitkan pesan kesalahan yang jelas yang menerangkan sumber masalah tersebut. Paket tersebut juga harus menuliskan kesalahan ke file kesalahan.
- ◆ **Dukungan (Support).** Faktor kinerja ini umumnya dibagi ke dalam dua kategori: kebijakan dan teknis. Kebijakan dukungan mencakup jalur-jalur toll-free, garansi, jaminan uang-kembali, dan pelatihan. Dukungan teknis didapat dari teknisi yang disediakan dan yang berpengalaman.

Menyeleksi Paket

Sasaran akhir adalah untuk menentukan paket perangkat lunak dari vendor mana yang menawarkan manfaat terbesar dengan biaya atau harga termurah. Untuk mencapai tujuan ini, kita tentukan angka penilaian total dan biaya totalnya. Metode untuk menentukan angka penilaian total ditunjukkan dalam Gambar 1.2. Bobot relatif ditentukan ke setiap faktor kinerja umum yang didasarkan pada kepentingan relatifnya. *Base atau nilai* dasarnya adalah 100.. Selanjutnya, didasarkan pada statistik kinerja yang dikumpulkan dari proposal atau usulan vendor, benchmark, dan

Faktor Kinerja Umum	Bobot	Vendor A		Vendor B	
		Pernilaian	Skor	Pernilaian	Skor
Penilaian vendor	10	6	60	8	80
Kinerja pengoperasian	20	7	140	8	160
Dokumentasi	10	8	80	9	90
Kemudahan belajar	20	7	140	6	120
Kemudahan pemakaian	10	5	50	6	60
Kendali dan penanganan kesalahan	20	4	80	6	120
Dukungan	10	7	70	8	80
Total		100		620	
				710	

Gambar 1.2 Penilaian kinerja umum.

penilaian terpublikasi, kita berikan penilaian kepada setiap faktor kinerja (1 = jelek dan 10 = sangat bagus). Bobot ini dikalikan dengan penilaian. Setiap skor yang dihasilkan dijumlahkan guna memberi angka penilaian total untuk setiap vendor. Dalam contoh kita, kita akan mengevaluasi Vendor A dan Vendor B.

Biaya atau harga untuk paket Vendor A adalah \$22.700 dan untuk paket Vendor B adalah \$27.690. Paket mana yang harus dipilih? Jawabannya ditentukan dengan cara membagi angka penilaian total dengan biaya total guna mengkalkulasi biaya per angka penilaian. Komputasi ini direkapitulasi dalam Gambar 1.3. Meskipun paket Vendor A mempunyai penilaian lebih rendah, namun biaya per angka penilaiannya yang sebesar \$37 membuatnya menjadi pilihan biaya/manfaat yang lebih baik dari pada biaya/manfaat Vendor B.

	Biaya Total	Angka Penilaian Total	Biaya per Angka Penilaian
Vendor A	\$22.700	620	\$37
Vendor B	27.690	710	39

Gambar 1.3 Menentukan biaya per angka penilaian.

Program Perangkat Lunak Pesanan

Jika sistem yang sedang dikembangkan tidak bisa didukung oleh perangkat lunak off-the-self (paket), kita harus memesan atau membangun sendiri perangkat lunak agar sesuai atau cocok dengan rancangan sistemnya. Program perangkat lunak terkustomisasi atau original dikembangkan oleh orang-orang yang bekerja pada perusahaan jasa atau kontraktor. Penekanan pokok kita sepanjang buku ini adalah pada masalah pengembangan perangkat lunak secara swa-kelola (in-house), dan inilah yang akan kita kemukakan dalam bab ini dan tiga bab berikutnya.

Apa yang Dimaksud Dengan Siklus Hidup Pengembangan Perangkat Lunak?

Pembangunan program mengikuti tiga-tahap SIKLUS HIDUP PENGEMBANGAN PERANGKAT LUNAK (SOFTWARE DEVELOPMENT LIFE CYCLE—SWDLC):

- ◆ Rancangan (Design)
- ◆ Kode (Code)
- ◆ Uji (Test)

Tiga bab berikutnya akan membahas setiap tahap secara urut dan memberikan materi yang cukup untuk menyiapkan laporan terdokumentasi SWDLC yang siap diserahkan.

Kita akan menyorot SWDLC dan menjadikannya komponen siklus hidup dari SDLC untuk beberapa alasan. Pertama, SDLC mencakup pengembangan sistem keseluruhan, yang memerlukan komponen-komponen lain disamping perangkat lunak. Kedua, dalam sistem yang benar-benar memerlukan pengembangan perangkat lunak yang didasarkan pada rancangan sistem yang diciptakan oleh SDLC, SWDLC akan diinisiasi. Ketiga, apabila SWDLC menjadi berperan, ia, seperti halnya SDLC yang berbasis lebih luas, akan memberikan kumpulan acuan tahap-tahap yang diperlukan untuk mengembangkan perangkat lunak tersebut. SWDLC berisi atau menjabarkan tugas-tugas dan prosedur-prosedur yang harus dijalankan dalam setiap tahap; hasil yang diciptakan oleh setiap tahap; dan metriks untuk menyusun jadwal, mengestimasi biaya, dan mengukur produktivitas. Keempat, SWDLC memberikan atau menyediakan kerangka dan prosedur pengoperasian standart yang mendukung cara pengembangan perangkat lunak yang terstruktur dan terancang baik.

Satu distribusi usaha yang terekomendasi sepanjang tahap-tahap SWDLC kadang-kadang disebut aturan 40-20-40. Aturan atau rule ini menekankan rancangan

front-end yang kuat dan pengujian back-end, sedangkan tugas-tugas pemrograman (pengkodean) mekanik kurang diberi tekanan. Aturan 40-20-40 hanya digunakan sebagai pedoman. Jika rancangannya bebas-kesalahan, pemrograman dan pengujian akan berlangsung hampir tanpa kesulitan. Juga, semakin penting dan besar program, maka akan semakin banyak usaha yang diperlukan dalam tahap perancangan dan tahap pengujian.

Rancangan. Bagian dari rancangan sistem terinci yang akan dikonversi ke program aplikasi dirancang pada suatu tingkatan yang dapat digunakan oleh programmer untuk menuliskan kode (yakni, menulis program dalam bahasa seperti C atau CO BOL). Beberapa paket CASE akan membangkitkan kode dari beberapa rancangan terinci, sehingga menghapus adanya kebutuhan akan pengkode manusia (*human coders*). Dalam keadaan apapun, perangkat rancangan perangkat lunak pokok adalah:

- ◆ Bagan struktur
- ◆ Bahasa Inggris terstruktur
- ◆ Tabel keputusan
- ◆ Pohon keputusan
- ◆ Persamaan
- ◆ Kamus data
- ◆ Diagram Warnier-Orr
- ◆ Diagram Jackson

Kode. Tahap pengkodean (yakni, menulis statement dalam bahasa pemrograman, yang kita asumsikan dijalankan oleh programmer dan tidak secara otomatis dibangkitkan oleh paket CASE, memetakan rancangan ke dalam prosedur program (yang juga disebut statement atau instruksi)). Untuk menulis kode, programmer harus mempunyai keterampilan yang baik dalam menggunakan bahasa pemrograman seperti C atau COBOL.

Uji. Semua modul kode, baik terpisah atau tergabung, harus diuji guna mendeteksi dan menghapus kesalahannya. Setelah pengujian yang teliti, program tersebut dikonversi ke operasi. Walaupun banyak pengujian dilakukan setelah modul-modul tersebut dikode, beberapa bentuk pengujian sebenarnya dilakukan sepanjang SWDLC tersebut.

MENGORGANISASI PROYEK PENGEMBANGAN PERANGKAT LUNAK

Sistem kecil yang berbasis lokal bisa dikembangkan dari perencanaan sampai analisis dan rancangan umum ke rancangan terinci ke pemrograman ke konversi oleh seorang yang disebut programmer/analis. Jika diperlukan program terkustomisasi, biasanya ia merupakan program yang sangat kecil. Pengorganisasian untuk proyek sistem semacam ini sangat mudah. Namun, apabila kita menuju ke proyek yang lebih besar, yang memerlukan berbagai keterampilan khusus, pengorganisasian keterampilan seperti itu ke dalam tim proyek yang bisa berjalan akan merupakan tugas manajemen yang penuh tantangan.

Pentingnya Komunikasi, Integrasi dan Koordinasi yang Baik

Penyerahan tanggung jawab pengembangan sistem kepada terlalu banyak orang dapat mengakibatkan pencapaian hasil sistem akhir jauh berbeda dari apa yang diharapkan pada awalnya. Kesinambungan bisa jadi akan hilang atau terputus.

Mengembangkan sistem dengan cara serampangan tanpa dokumentasi yang jelas, kendali proyek, dan integrasi keterampilan, maka ini hanya seperti permainan dalam pesta dimana kita membisikkan pesan kepada orang di sebelah kita, lalu orang ini membisikkan lagi pesan yang sama ke orang sebelahnya. Kemungkinan besar, apabila pesan tersebut terus disampaikan, maka ia semakin terdistorsi, dimana pesan yang diterima oleh orang paling akhir mungkin akan jauh berbeda dari pesan aslinya. Di pesta, permainan ini akan menghasilkan suasana riang. Dalam pengembangan sistem, ini akan mengakibatkan gagalnya proyek sistem.

Namun demikian, jika rancangan sistem memerlukan agar program terkustomisasi yang akan dikembangkan secara *in-house* lebih besar daripada program yang sangat kecil, maka masih terdapat banyak pengorganisasian dan pekerjaan yang harus dilakukan sebelum kita bisa mencapai hasil akhir yang baik.

Orang-orang yang telah menjalankan sebagian besar pekerjaan pada saat ini mungkin yakin bahwa apabila mereka telah menyelesaikan Laporan Rancangan Sistem Terinci, pekerjaan mereka telah rampung. Mereka menyampaikan laporan terdokumentasi yang siap diserahkan ini ke tim pemrograman guna menuliskan program. Karena para perancang dan analis sistem sama dengan arsitek, kita percaya bahwa mereka pasti terlibat dalam pengembangan perangkat lunak dan tugas-tugas pengimplementasian lainnya persis seperti arsitek terlibat dalam pengonstruksian dan penyelesaian rancangannya.

Seperti halnya arsitek, para profesional sistem harus memfokuskan pada hasil akhir. Walaupun arsitek tidak memasang batu, memancangkan baja, atau menginstalasi bangunan, namun ia tahu bagaimana setiap pekerjaan ini dan juga pekerjaan konstruksi lainnya dilakukan. Dengan cara yang sama, perancang dan analis sistem walaupun tidak mengkode program perangkat lunak, ia harus mengetahui bagaimana program ini dikode dan bagaimana hasil akhirnya.

Pendekatan Organisasional

Selain menggunakan pendekatan tim SWAT, ada berbagai cara untuk mengorganisasikan tim pemrograman. Disini akan dikemukakan tiga diantaranya:

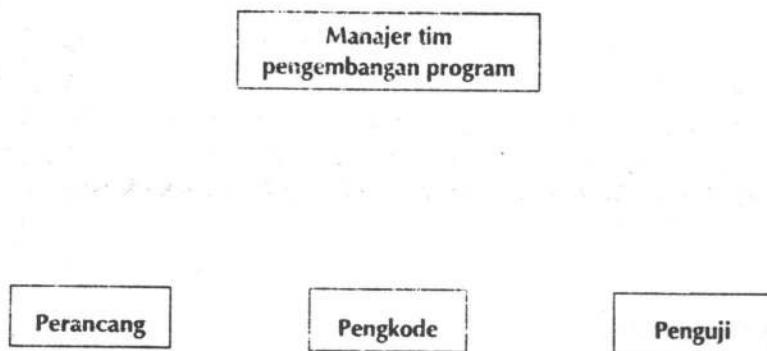
- ◆ Tim pengembangan program (*program development team*)
- ◆ Tim programmer kepala (*chief programmer team*)
- ◆ Tim pemrograman bersama (*egoless programming team*)

Kita akan mendeskripsikan ketiga pendekatan organisasional ini lebih dulu. Kemudian kita akan mengemukakan pabrik perangkat lunak Jepang, suatu konsep organisasional yang menarik dan efektif.

Tim Pengembangan Program

Tim pengembangan program, yang ditunjukkan dalam Gambar 1.4, dikelola oleh manajer (atau pimpinan) tim. Sebaiknya, orang ini adalah orang yang mengelola proyek sistem keseluruhan atau seseorang dari tim proyek sistem yang terlibat dalam SDLC dari awal. Tim pengembangan program didukung oleh pustakawan, editor, klerk program untuk menjalankan tugas-tugas klerikal, administratif, dan dokumentatif.

Tingkatan keterampilan pada tim tersebut bisa menjangkau dari para petata sampai para individu yang berketrampilan tinggi. Jika suatu perusahaan menganut aturan pengembangan program 40-20-40, maka orang-orang yang lebih tinggi keterampilannya harus ditugasi untuk perancangan dan pengujian, seperti yang dikemukakan dalam Gambar 1.5. Teori di balik aturan 40-20-40 ini adalah: Jika rancangan lengkap, jelas, dan akurat, maka fungsi pengkodean akan menjadi proses mekanik yang sederhana yang dapat dijalankan oleh setiap orang yang telah kenal dengan sintaks bahasa pemrograman. (Biasanya, petatar setidaknya juga pernah diberi pelatihan masalah sintaks). Konsep atau gagasan yang sama ini mendukung



Gambar 1.4 Tim pengembangan program.

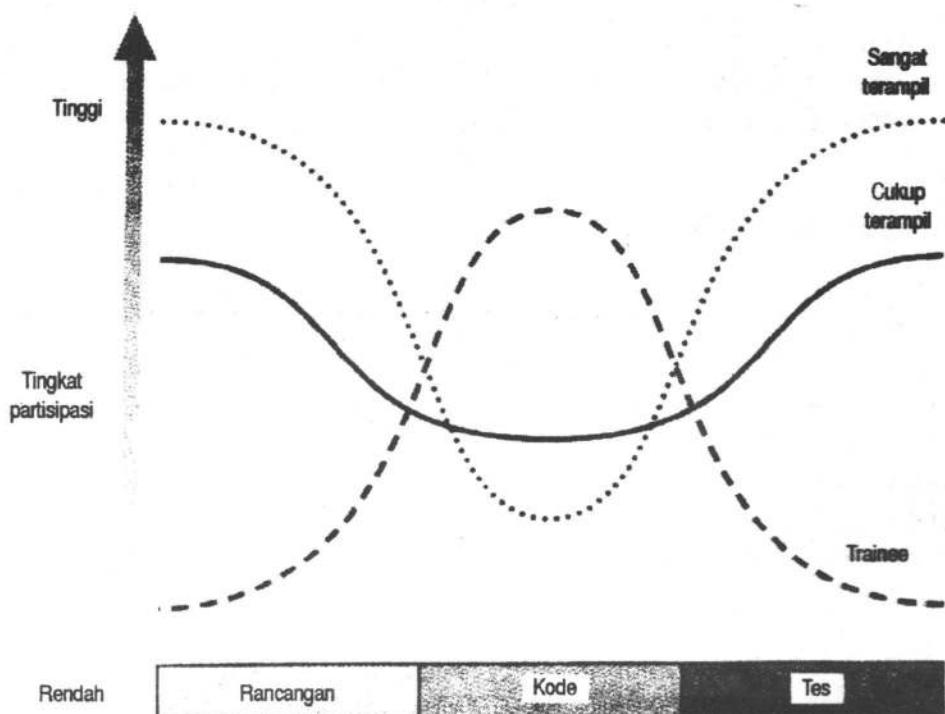
terciptanya teknologi CASE, yang dapat membangkitkan/menghasilkan kode yang didasarkan pada rancangan presisi; sehingga hal ini akan menghilangkan kebutuhan pengkodean dengan tangan atau manual.

Tim Programmer Kepala

Chief Programmer Team dibentuk dari programmer kepala atau senior yang mempunyai banyak pengalaman dan pengetahuan pemrograman. Orang semacam ini dapat berkomunikasi secara efektif dengan para perancang dan analis sistem, pemakai, dan berbagai teknisi. Ia adalah manajer yang baik. Tim programmer kepala seringkali sebanding atau sama dengan tim ahli bedah, dimana tanggung jawab akhir terletak pada ahli bedah kepala. Orang tersebut didukung oleh para staf khusus yang terampil, yang meliputi ahli bedah cadangan, ahli bedah pengamat atau pemeriksa, ahli anestetis, perawat kepala, perawat pembantu, dan teknisi lain.

Programmer kepala didukung oleh asisten kepala, yang merupakan orang cadangan utama pada proyek itu dan yang berkomunikasi dengan setiap orang lain pada tim itu, dimana ia bertindak sebagai penyalur suara dari gagasan programmer kepala. Tergantung pada ukuran proyeknya, kedua orang ini didukung oleh beberapa dari atau semua personel berikut ini:

Programmer Pendukung. Programmer yunior diperlukan untuk proyek besar yang tidak dapat ditangani sendiri oleh programmer kepala dan asisten utamanya. Para programmer pendukung biasanya mengkode modul-modul tingkat rendah.



Gambar 1.5 Partisipasi tingkat-keterampilan dalam tahap SWDLC.

Pustakawan. Orang ini memelihara perpustakaan produksi program, mengindeks file kompilasi dan pengujian, dan menjaga perpustakaan kode sumber dan kode obyek agar selalu up-to-date.

Administrator. Orang ini menangani semua rincian dukungan non-teknis, seperti anggaran, masalah personel, dan alokasi ruang, dan ia berinteraksi dengan seluruh birokrasi organisasi.

Editor. Editor bertanggung jawab untuk menghasilkan dokumentasi, mencari referensi, dan mengamati semua tahap pembuatan ulang dan pendistribusian dokumentasi. Dalam proyek yang lebih kecil, editor bisa juga menjalankan tugas-tugas pustakawan.

Klerk Program. Petugas atau pekerja ini memelihara semua record atau catatan teknis untuk tim pemrograman dan melakukan tugas sekretarial apa saja yang diperlukan oleh tim pemrograman.

Tim Rekan Pemrograman

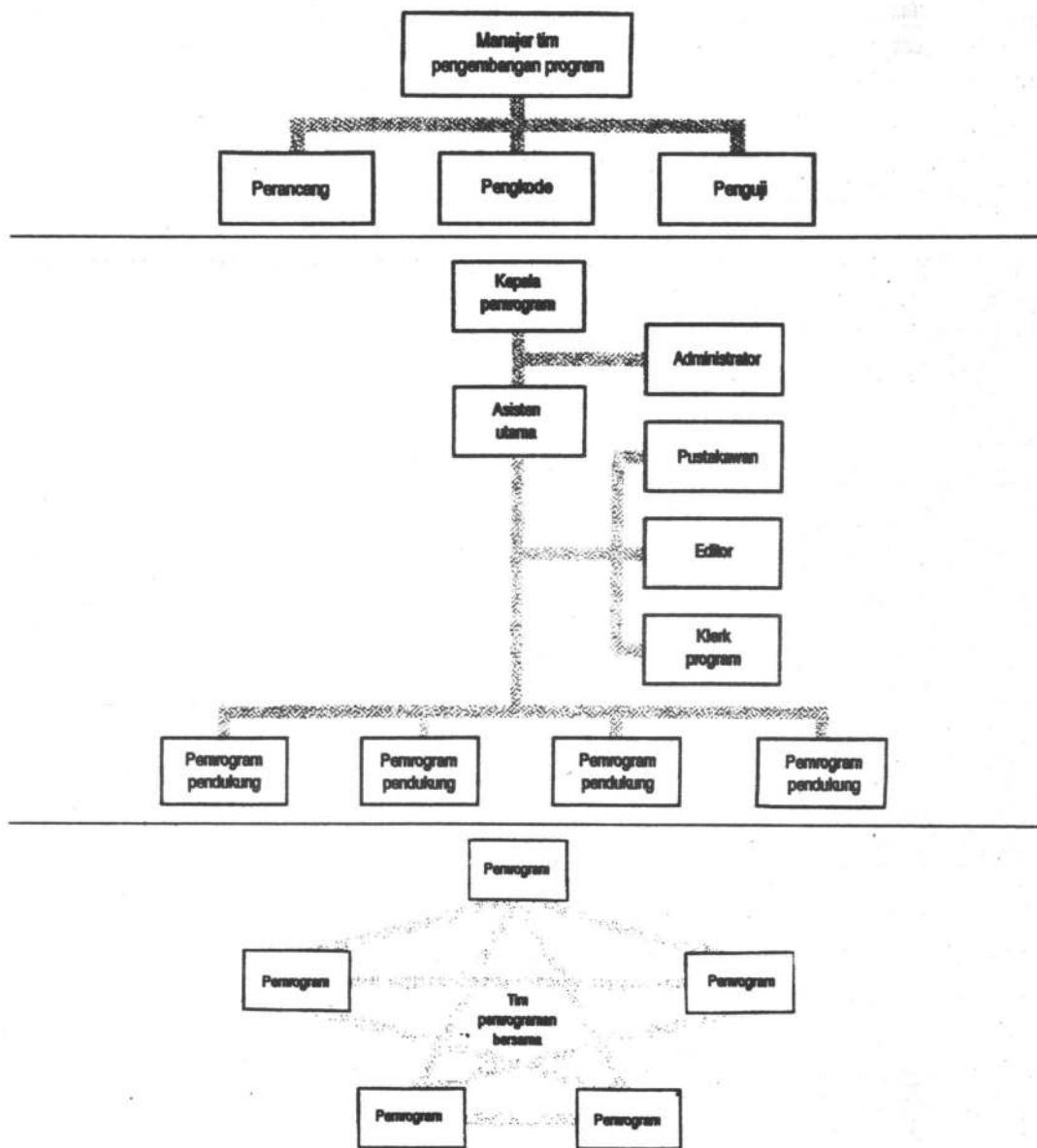
EGOLESS PROGRAMMING TEAM (Tim Pemrograman Bersama) adalah tim tanpa bos atau pimpinan. Tim bersama terbentuk dari seluruh rekan yang bersama-sama bertanggung jawab atas pengembangan perangkat lunak. Tim ini mengelola persoalan atau permasalahan hariannya tanpa supervisi langsung. Para anggota tim membuat keputusan secara demokratis untuk menghindari konflik personal. Setiap anggota tim meninjau dan mengkritik setiap pekerjaan dari para anggota lainnya. Tim pemrograman bersama bisa didukung oleh pustakawan, editor, klerk program untuk menjalankan tugas-tugas klerikal, administratif, dan dokumentatif.

Apa Perbedaan Antara Pendekatan-Pendekatan Tersebut?

Perbedaan sepintas antara pendekatan tim pengembangan program dan pendekatan tim programmer kepala dan tim pemrograman bersama adalah bahwa tim pengembangan program menerapkan (berkaca pada) aturan pengembangan program 40-20-40, sedangkan tim programmer kepala dan tim pemrograman bersama menekankan pada fungsi pengkodean. Karena pendekatan tim pengembangan program menekankan pada perancangan dan pengujian, maka sebagian besar dari pembahasan kita dalam buku ini diasumsikan untuk membahas jenis pengorganisasian ini.

Jumlah dan ukuran lintasan komunikasi diantara para anggota tim berbeda untuk setiap pendekatan tersebut. Sebagian besar kesalahan perancangan dan pengkodean dalam pengembangan program terjadi pada interface; yakni, Perancang A berkomunikasi dengan Perancang B, atau Programmer A berkomunikasi dengan Programmer B.

Interface dan lintasan komunikasi dari setiap pendekatan organisasional ini dikemukakan dalam Gambar 1.6. Tim pengembangan program tersusun atas dua perancang, satu pengkode, dan dua penguji. Interface dan lintasan komunikasi utama berada antara perancang dan pengkode, dan kemudian antara pengkode dan penguji. Interface dan lintasan komunikasi juga berada antara perancang dan penguji. Karena manajer tim tidak terlibat secara langsung dalam pekerjaan yang sebenarnya, interface dan lintasan komunikasi ke orang ini memberikan rekapitulasi dan informasi



Gambar 1.6 Interface dan jalur komunikasi dari ketiga pendekatan tim.

kinerja. Maka interface dan lintasan komunikasi total untuk pekerjaan yang sedang dilakukan ada lima, dengan satu interface manajemen.

Dalam tim programmer kepala, tim yang terdiri dari lima programmer pendukung mempunyai lima interface dan lintasan komunikasi utama yang dikoordinasikan dan dihubungkan oleh asisten utama dan pustakawan. Apabila dibandingkan dengan tim pemrograman bersama, pengaturan ini lebih mengurangi peluang terjadinya kesalahan dan meningkatkan produktivitas. Sebenarnya, sejumlah kewenangan mengisyaratkan bahwa suatu tim programmer kepala memiliki produktivitas dua kali lebih besar dari pendekatan atau cara tim pemrograman bersama. Oleh karena itu, tim programmer kepala lebih mungkin memenuhi deadline yang ketat dari pada tim pemrograman bersama.

Sejumlah interface dan lintasan komunikasi antara n orang dalam suatu tim pemrograman bersama adalah $n(n - 1)/2$. Tim pemrograman bersama dalam gambar tersebut terkomposisi atas lima programmer. Oleh karenanya, terdapat $5(5 - 1)/2$ atau sepuluh interface dan lintasan komunikasi.

Biasanya, komunikasi membutuhkan waktu dan mengurangi produktivitas. Sifat dari segala jenis pekerjaan pengembangan adalah bahwa waktu restart (mulai lagi) bersarnya 30 menit atau lebih setelah setiap interupsi. Jika programmer tertentu, misalnya, hanya memberikan delapan periode yang masing-masing lamanya 10-menit selama sehari untuk tugas pemrograman, maka peluang pemrograman yang bisa dilakukan hanyalah sedikit. Walaupun ada berbagai sudut pandang dan sangat sedikit data empiris untuk mendukung sudut pandang tertentu, nampaknya mayoritas kewenangan percaya bahwa masalah yang terjadi dalam tim pemrograman bersama adalah berupa terlalu banyak bicara namun sedikit tindakan dan produktivitas kualitas yang tinggi. Oleh karena itu, umumnya, apabila terdapat lebih dari tiga programmer yang terlibat, maka sebaiknya ditetapkan seorang supervisor atau pimpinan. Namun validitas pernyataan sebelumnya tergantung pada para anggota tim tersebut. Jika semua anggota bekerja bersama secara baik seperti tim sepak bola dan tidak diperlukan banyak komunikasi, maka tim pemrograman bersama bisa sangat produktif. Namun demikian, situasi seperti ini jarang terjadi. Oleh karena itulah tim sepak bola yang baik atau tim yang lain memerlukan pelatih atau manajer.

Apa yang Dimaksud Dengan Konsep Pabrik Perangkat lunak?

Di Jepang, pabrik-pabrik perangkat lunak menerapkan pengembangan perangkat lunak dengan prinsip pengendalian kualitas dan manajemen proyek yang sama sehingga memungkinkan Jepang mendominasi sektor manufakturing. Para peran-

cang sistem, pengkode, dan penguji diberi workstation-workstation yang diinterkoneksi melalui jaringan. Orang-orang ini diorganisasi dengan cara yang sama seperti pengorganisasian pekerja pabrik. Walaupun di luar cakupan buku ini, studi mengenai manufakturing just-in-time (JIT), kanban, pengendalian kualitas total (TQC), dan reduksi biaya non-nilai tambah, akan memberikan pengetahuan mengenai model metode manufakturing orang Jepang yang saat ini diimplementasikan di pabrik-pabrik perangkat lunak Jepang.

Walaupun pendekatan tim pengembangan proyek diorganisasi secara berbeda dari pengorganisasian pekerja pabrik, tujuan-tujuan pabrik perangkat lunak Jepang sama dengan tujuan-tujuan yang akan dicapai dalam buku ini, yaitu:

- ◆ Pengaplikasian atau penerapan pendekatan/cara termekanisasi (*terekayasa*) untuk pengembangan sistem dan perangkat lunak.
- ◆ Penggunaan perangkat pemodelan dan teknologi CASE.
- ◆ Penginstalasian teknik manajemen proyek.
- ◆ Penekanan pada daya atau kemampuan pemeliharaan, penggunaan, penggunaan ulang, reliabilitas, dan perluasan faktor-faktor rancangan (MURRE).
- ◆ Pencapaian produktivitas pengembangan perangkat lunak dan sistem yang optimum.

Untuk memenuhi permintaan perangkat lunak terkustomisasi, personel pabrik perangkat lunak akan menghabiskan banyak usaha dan waktu untuk mengembangkan kode yang bisa digunakan ulang. Seringkali, apabila diperlukan aplikasi terkustomisasi baru, maka kode baru, jika diperlukan, akan dikombinasikan dengan kode lama, dengan demikian ini akan mengurangi waktu pengembangan perangkat lunak total.

Penginstalasian pabrik perangkat lunak di Jepang telah menjadi sarana untuk menggandakan dan melipatkan tiga kali produktivitas pengembangan perangkat lunak dari produktivitas ketika tanpa pabrik perangkat lunak beberapa tahun sebelumnya. Proyek terakhir/terbaru telah dikurangi dari sebesar 80 persen dengan tahapan pengurangan sebesar 5 persen untuk kebutuhan pengkodeannya. Rata-rata pengurangannya adalah 10 persen. Rata-rata jalur kode yang bisa dieksekusi (LOEC) yang bisa dihadirkan pada akhir 1970-an dan awal 1980-an adalah sekitar 1200 LOEC pemrograman assembly per programmer per bulan. Konversi umum dari assembly ke FORTRAN adalah sekitar sepertiganya atau kurang lebih 400 (1200 / 3) FORTRAN LEOC. Sekarang, rata-ratanya adalah 3600 LEOC pemrograman assembly (atau 1200 FORTRAN LOEC) yang dihadirkan per programmer per bulan,

termasuk kode yang bisa digunakan ulang, yang keseluruhannya sekitar 45 sampai 50 persen dari LOEC yang dihadirkan. Maka penginstalasian konsep pabrik perangkat lunak bersama dengan penekanan pada penggunaan ulang telah meningkatkan produktivitas pengembangan perangkat lunak secara dramatis.

Kualitas perangkat lunak sekaligus juga meningkat dari sekitar 20 kesalahan per seribu LOEC. Kesalahan residu (yakni, kesalahan yang ditemukan setelah perangkat lunak dikonversi ke operasi) setelah pengujian yang teliti (pokok bahasan Bab 18) adalah sekitar 0,2 kesalahan per seribu LOEC. Beberapa perusahaan melaporkan kesalahan residu sebesar 0,01 per seribu LOEC.

Toshiba menggunakan pabrik perangkat lunak untuk mengembangkan perangkat lunak yang sangat kompleks dengan produktivitas dan reliabilitas yang tinggi. Toshiba telah menjadi pemimpin dunia dalam memproduksi perangkat lunak kendali proses otomat real-time yang digunakan untuk pembangkit tenaga nuklir dan listrik. Lebih dari itu, perangkat lunak yang dikembangkan oleh pabrik perangkat lunak Toshiba terbukti memerlukan sedikit pemeliharaan, dan pemeliharaan yang dibutuhkan ini sangat mudah dilakukan. Hal yang sama terjadi pula di pabrik-pabrik perangkat lunak Jepang lainnya.

MENGUKUR PRODUKTIVITAS DALAM PENGEMBANGAN PERANGKAT LUNAK

Produktivitas dapat ditentukan atau ditetapkan dengan formula atau rumus berikut ini:

$$\text{Produktivitas} = \frac{\text{output yang dihasilkan}}{\text{input yang dikonsumsi}}$$

Dengan demikian produktivitas pengembangan perangkat lunak dapat ditingkatkan dengan menaikkan output, menurunkan input, atau keduanya. Input relatif mudah diukur, misalnya tenaga kerja, workstation (misalnya, sistem CASE), pasokan, dan berbagai fasilitas pendukung. Namun, masalahnya terletak pada pengukuran output.

Sejumlah teknik metrik alternatif telah dibuat oleh berbagai pihak yang berwenang (dalam masalah pengukuran output). Mereka berusaha untuk mengukur output pengembangan perangkat lunak. Dua metrik yang nampak paling berperan adalah:

- ◆ Lines of executable code (LOEC) atau jalur kode yang bisa dieksekusi.

- ◆ Function Point.

Dengan menggunakan metrik, kita bisa mengelola proses pengembangan perangkat lunak. Kita juga bisa mengukur dampak perubahan, seperti perubahan ke teknologi CASE atau dari satu generasi bahasa komputer ke generasi lain. Lebih dari itu, dengan menggunakan metrik, akan terjadi persepsi bahwa pengembangan perangkat lunak lebih bersifat ilmiah (yakni, produk yang dimekanisasi) dari pada bersifat seni. Awalnya, banyak programmer menyatakan bahwa mengukur hasil pekerjaan mereka sama mustahilnya dengan mengukur karya William Faulkner atau Pablo Picasso. Namun dari sudut pandang pengembangan perangkat lunak yang menganut atau mengikuti SWDLC, yang sifatnya berorientasi proyek dan berorientasi proses, metrik tidak hanya bisa digunakan, namun perlu digunakan. Metrik apapun yang digunakan, fasilitas penting dari suatu sistem metrik adalah kewajarnya dan kekonsistensianya.

Mencacah Jalur Kode yang Bisa Dieksekusi

Beberapa profesional sistem menggunakan jalur kode sumber (SLOC) sebagai metrik mereka. Jalur kode sumber adalah segala jalur program yang bukan penjelasan atau merupakan jalur kosong, tanpa mempedulikan jumlah statemen atau fragmen statemen pada jalur itu, termasuk statemen yang bisa dieksekusi dan yang tak bisa dieksekusi. SLOC yang lebih dihaluskan lagi bisa untuk mengukur jalur kode yang bisa dieksekusi (LOEC) saja. Kita gunakan LOEC METRIC dalam pembahasan berikut.

Beberapa pihak yang berwenang percaya bahwa LOEC adalah metrik yang paling baik, termasuk titik-titik fungsi (*function point*), karena ia mudah dan banyak digunakan. Pihak-pihak ini menyatakan bahwa tidak ada metrik alternatif yang lebih unggul dari pada LOEC. Memang, banyak keuntungan yang diperoleh organisasi-organisasi dengan terus menggunakan metrik LOEC sebagai ukuran produktivitas output perangkat lunak utamanya, seperti:

- ◆ **Mudah Ditetapkan dan Dibahas Secara Jelas.** End user, manajer, dan profesional sistem biasanya memahami apa yang dimaksud dengan jalur kode yang bisa dieksekusi.
- ◆ **Diakui Secara Luas.** LOEC (atau kadang-kadang hanya disebut jalur kode) adalah metrik yang seringkali diakui digunakan oleh vendor alat pengembangan perangkat lunak ketika berbicara masalah produktivitas perangkat mereka

- ◆ **Mudah Diukur.** Jalur kode yang bisa dieksekusi (LOEC) tinggal dicacah atau dihitung untuk menentukan ukuran program.
- ◆ **Mudah Digunakan untuk Estimasi.** Ukuran perkiraan suatu program ditentukan berdasarkan dokumentasi rancangan sistem terinci. Kemudian angka ini digunakan untuk mengestimasi waktu dan biaya proyek pengembangan perangkat lunak.

Sebagai contoh, program yang diusulkan akan berisi 100 KLOEC, dan jika 2 KLOEC dapat dihasilkan oleh satu orang per bulan, maka diperlukan 50 orang per bulan untuk menyelesaikan proyek tersebut. Jika input yang diperlukan untuk mendukung satu orang per bulan sebesar \$9000, maka proyek pengembangan perangkat lunak tersebut akan membutuhkan biaya \$450.000.

Apa Kelemahan dari Satu Orang Per Bulan (Person-Month) dan Pencacahan LOEC?

Seperti yang dikemukakan Frederick P. Brooks, Jr. dalam bukunya yang terkenal *The Mythical Man-Month*, the person-month (atau man-month) sebagai unit pengukuran ukuran suatu pekerjaan adalah merupakan suatu mitos. Ini mengisyaratkan bahwa *orang dan bulan* bisa saling ditukarkan (saling mengganti). Ia lebih jauh menyatakan bahwa orang dan bulan hanya bisa saling ditukar/diganti apabila tugas dapat dipartisi/dibagikan kepada beberapa pekerja tanpa komunikasi diantara mereka. Namun sebagaimana kita ketahui dari pembahasan sebelumnya mengenai interface dan lintasan komunikasi, ketiadaan komunikasi seperti itu mustahil bisa kita terapkan. Oleh karena itu, penggandaan ukuran tim (sebagaimana dikemukakan Brooks) tidak akan menggandakan produktivitas, dikarenakan adanya peningkatan eksponensial interface dan lintasan komunikasi. Pada kenyataannya, produktivitas tidak meningkat sama sekali, namun menurun. Oleh karena inilah, kita perlu mengorganisasi dan meneglola tim pengembangan secara tepat dan meminimisasi jumlah point atau titik interface dan lintasan komunikasi.

Penggunaan person-month sebagai benchmark atau parameter estimasi tidaklah sempurna. Namun jika digunakan secara wajar dan konsisten, ia dapat memberikan cara untuk membantu estimasi waktu dan biaya proyek, dan ia bisa digunakan sebagai alat bantu dalam pengukuran dan pengevaluasian produktivitas. Para pendukung metrik LOEC per person-month menyatakan secara tegas bahwa metrik ini adalah metrik produktivitas perangkat lunak paling praktis dari pada alternatif lain yang ada saat ini.

Selain kelemahan asli dalam person-month, terdapat juga beberapa kelemahan pada metrik LOEC itu sendiri. LOEC dapat lebih mudah menyebabkan produktivitas yang salah tempat/atur. Sebagai contoh, programmer assembly mungkin memerlukan empat minggu dan 1500 LOEC untuk menulis suatu program. Programmer COBOL bisa membutuhkan waktu dua minggu dan 500 LOEC untuk menulis aplikasi yang sama, dan programmer C mungkin hanya memerlukan waktu satu minggu dan 300 LOEC. Dengan dasar metrik LOEC, programmer assembly adalah yang paling produktif. Namun demikian, secara riil programmer yang paling produktif adalah programmer C yang bisa selesai dalam satu minggu. Selain itu, jika kita mendasarkan pada unsur tahap perancangan dan pengujian, terutama pengujian, proyek pengembangan perangkat lunak bisa dijalankan secara lebih efektif dan lebih efisien dengan bahasa pemrograman C dan COBOL dari pada dengan bahasa pemrograman assembly. Lebih dari itu, metrik LOEC mengukur tahap pengkodean sendiri, yang ini tentunya merupakan kelemahannya yang lain, sebab sebagaimana yang dikemukakan sebelumnya bahwa dalam program yang dikembangkan secara benar, pengkodeannya cuma harus memerlukan 20 persen dari usaha pengembangan perangkat lunak keseluruhan/total. Terakhir, seorang programmer secara mudah bisa menurunkan produktivitas dengan memecah instruksi ke dalam (menjadi jalur-jalur kode yang lebih banyak dari pada yang diperlukan).

Bagaimana dengan kualitas program? Pencacahan LOEC (Counting LOEC) dapat menetapkan basis untuk membandingkan kecepatan seorang programmer dengan kecepatan programmer lain. Namun demikian, ukuran ini tidak menunjukkan kualitas pekerjaan yang sedang dilakukan.

Menggunakan Metrik Titik Fungsi (Point Function)

Metrik titik fungsi dirancang untuk mengatasi beberapa kelemahan metrik LOEC. Ada lima fungsi yang dianalisis:

- ◆ Jumlah input, seperti form dan layar.
- ◆ Jumlah output, seperti laporan dan layar.
- ◆ Jumlah query yang diminta oleh end user.
- ◆ Jumlah file logik yang diakses dan digunakan.
- ◆ Jumlah interface ke aplikasi lain.

Lima jenis fungsi ini adalah fungsi-fungsi yang diukur atau dicacah oleh profesional sistem ketika menggunakan metrik titik fungsi.

Titik fungsi mengukur apa yang akan dihadirkan atau diberikan oleh tim pengembangan perangkat lunak kepada end user. Oleh karenanya, metrik titik fungsi mencakup perancangan, pengkodean, dan pengujian, sementara metrik LOEC hanya menekankan pada pengkodean. Lebih dari itu, metrik titik fungsi mengukur efisiensi maupun efektivitas.

Metrik titik fungsi memberikan keuntungan, yaitu menggunakan cara yang seragam untuk mengukur produktivitas perangkat lunak tanpa memandang bahasa pemrograman yang digunakan. Ia juga telah diterima oleh para ahli. Selain itu, ia sedang dikembangkan atau ditingkatkan guna memberikan aturan pencacahan dan memperluas aplikasinya. Dukungan terhadap metrik titik fungsi ini adalah dari International Function Point User Group (IFPUG).

Untuk menghitung titik fungsi untuk suatu sistem baru, staf pengembangan perangkat lunak mengkonversi rancangan sistem terinci ke dalam lima kategori titik fungsi:

- ◆ Input
- ◆ Output
- ◆ Inquiry
- ◆ File
- ◆ Interface

Sebagai contoh, para staf akan menghitung input. Input bisa saja bervariasi ukuran dan kompleksitasnya. Input yang berisi 60 field data tentunya lebih kompleks dari pada input yang berisi lima field. Demikian pula halnya, input yang memerlukan

Titik Fungsi	Tingkat Kompleksitas			Total
	Rendah	Sedang	Tinggi	
Input	$12 \times 2 = 24$	$3 \times 5 = 15$	$12 \times 8 = 96$	135
Output	$10 \times 3 = 30$	$15 \times 5 = 75$	$14 \times 9 = 126$	231
Inquiry	$10 \times 3 = 30$	$16 \times 6 = 96$	$17 \times 8 = 136$	262
File	$9 \times 4 = 36$	$20 \times 7 = 140$	$10 \times 10 = 100$	276
Interface	$12 \times 4 = 48$	$20 \times 6 = 120$	$20 \times 10 = 200$	<u>368</u>
Total titik fungsi				<u>1272</u>

Gambar 1.7 Analisis titik fungsi untuk proyek pengembangan perangkat lunak.

akses ke sepuluh file (atau tabel) akan lebih kompleks dari pada input yang memerlukan akses ke satu file (atau tabel).

Salah satu cara untuk menjalankan analisis titik fungsi ditunjukkan pada Gambar 1.7. Derajad kompleksitasnya berjangkauan dari 1 pada low end sampai 10 sebagai yang paling kompleks. Gambar tersebut mencakup analisis dan titik fungsi total yang diperlukan untuk mengembangkan proyek perangkat lunak tertentu. Jumlah fungsi dicacah/dihitung dan ditimbang/dibobot dengan faktor kompleksitas. Sebagai contoh, dua belas input mempunyai kompleksitas yang cukup rendah (2), tiga input mempunyai kompleksitas rata-rata (5), dan dua belas input mempunyai kompleksitas cukup tinggi (8). Semua titik fungsi lain dicacah, ditimbang, dan dikalkulasi dengan cara yang sama. Jumlah total titik fungsi adalah 1272 untuk proyek pengembangan perangkat lunak tertentu.

Tingkat produktivitas pengembangan, yang kadang-kadang disebut sebagai tingkat penyerahan (*delivery rate*), dikomputasi/dihitung sebagai berikut:

$$\text{Tingkat produktivitas pengembangan} = \frac{\text{jumlah titik fungsi yang dihadirkan}}{\text{jumlah person-month}}$$

Tingkat produktivitas pengembangan rata-rata untuk perangkat lunak aplikasi umum berada antara 5 dan 10. Yaitu, satu orang bisa menyerahkan atau menghasilkan sekitar lima sampai sepuluh titik fungsi per bulan. Jika proyek pengembangan perangkat lunak baru memerlukan 1272 titik fungsi dan tingkat penyerahan rata-rata dari tim yang akan mengembangkannya adalah delapan titik fungsi per person-month, maka proyek tersebut akan memerlukan sekitar 159 person-month. Jika satu person-month mengkonsumsi \$10.000 untuk sumber daya (yakni, input), maka proyek tersebut akan berbiaya kurang lebih \$1.590.000.

Jika organisasi tersebut menginstal sistem CASE, angka penghasilan bisa melompat ke 15 atau 20 titik fungsi per person-month. Dan jika lingkungan CASE dikombinasikan dengan prosedur pengembangan perangkat lunak yang menerapkan penggunaan ulang kode dan organisasi tersebut bisa mencapai 50 persen penggunaan ulang kode, maka ia akan bisa mencapai tingkat produktivitas pengembangan lebih dari 70 titik fungsi per person-month. Maka, dengan pengadopsian teknologi CASE, pendekatan rancangan, dan penggunaan ulang kode, dan biaya satu person-month bisa meningkat pada awalnya sebesar, katakanlah, 20 persen, menuju ke tingkat tertentu, dan akhirnya menurun. Sebagai contoh, kita asumsikan bahwa biaya per person-month dalam contoh kita tersebut melompat ke \$12.000, namun personel pengembangan dapat meningkatkan tingkat penyerahannya ke 62 titik fungsi

per person-month dengan menggunakan teknik yang lebih maju. Jumlah person-month yang akan menghasilkan 1272 titik fungsi dikurangi menjadi sekitar 20,5 person-month ($1272/62$) dengan biaya sebesar \$246.000 atau penghematan biaya sebesar \$1.344.000 (\$1.590.000 - \$246.000).

Waktu yang diestimasikan sebelumnya dapat diinterpretasikan sebagai bulan yang sudah lewat, yang diukur menurut jumlah orang yang ditugasi. Sebagai contoh, jika lima orang terampil dan bermotivasi tinggi ditugasi untuk proyek tersebut, maka waktu total yang dilewati untuk menyelesaikan proyek itu akan sebesar kurang lebih empat bulan ($20,5/5$).

Estimasi ini mengabaikan konsep man-month mistis dari Brooks, walaupun analisisnya cermat, terkemukakan secara baik, dan dalam banyak kasus, benar. Namun dalam analisis final, validitas penggunaan person-month untuk mengestimasi waktu dan biaya, dan evaluasi produktivitas, tergantung pada orang-orang yang melakukan pekerjaan tersebut. Sebagai contoh, dalam beberapa kasus, lima orang yang bekerja bersama mungkin benar-benar menghasilkan tingkat penyerahan yang jauh lebih besar dari pada lima kali kerja seseorang yang bekerja sendiri. Mungkin sinergisme berperan disini dan keseluruhannya lebih besar dari pada jumlah bagian-bagiannya. Dalam event apapun, formula untuk mengukur tingkat penyerahan relatif dari berbagai tim yang ukurannya berbeda di luar cakupan buku ini. Meskipun demikian, kita menyadari bahwa, dalam beberapa kasus, penggandaan ukuran tim jangan untuk tujuan menggandakan delivery rate tim itu, meskipun kita berpendapat demikian.

Pengaruh Manajemen Terhadap Produktivitas

Manajemen proyek pengembangan perangkat lunak bisa menjadi pengaruh positif atau negatif. Manajemen yang tidak baik dapat menurunkan produktivitas perangkat lunak secara lebih cepat dari pada faktor lain. Berikut adalah beberapa contoh aktivitas manajemen yang seringkali menyebabkan berkurangnya/hilangnya produktivitas.

Metrik Produktivitas yang Tidak Baik atau Tidak Tersedia. Tanpa metrik produktivitas yang wajar dan konsisten, manajer tidak akan bisa mengestimasikan waktu dan biaya pengembangan perangkat lunak, atau tidak bisa mengukur tingkat produktivitas atau tingkat penyerahan. Selain berfungsi sebagai alat estimasi, metrik yang baik dapat digunakan untuk mengidentifikasi peningkatan atau penerusan produktivitas tim. Untuk mengidentifikasi peningkatan, manajer harus menge-

tahui berapa tingkat produktivitas tim saat itu dan apabila telah mengetahuinya, ia harus membandingkannya dengan *baseline* (angka dasar) yang telah ditetapkan.

Metrik produktivitas harus mengukur efisiensi maupun efektivitas. Efisiensi berkaitan dengan sumber-sumber yang dikonsumsi dalam pengembangan suatu aplikasi tertentu secara tepat waktu. Efektivitas berhubungan dengan kualitas program jadi dan kemampuannya dalam memenuhi kebutuhan pemakai.

Perencanaan dan Pengontrolan yang Tidak Baik. Manajer proyek membiarkan proyek berkembang (berevolusi) semaunya tanpa mentarget tanggal penyelesaian dan penyerahannya. Pengaplikasian teknik manajemen proyek seperti PERT akan membantu mengurangi atau menghilangkan perencanaan dan pengontrolan yang jelek.

Pencampuran Keterampilan yang Tidak Baik. Untuk menimbulkan/memberi bobot proyek pengembangan perangkat lunak, tim dengan semua pengkodeanya cenderung kurang memberi tekanan pada fungsi rancangan dan pengujian dan lebih menekankan pada pengkodean. Campuran keterampilan yang ideal adalah 40 persen perancang, 20 persen pengkode, dan 40 persen penguji. Para penguji harus diorganisasi dan dikelola sebagai kelompok independen yang terpisah.

Pengkodean Prematur. Beberapa falsafah para manajer proyek bisa diringkas dalam statemen ini: "Sebaiknya kita selesaikan secara cepat dan mulai melakukan pengkodean, sebab kita akan melakukan banyak debugging." Banyak contoh bisa dikutip dari organisasi-organisasi yang gagal melakukan pekerjaan awalnya karena mereka mengabaikan tahap rancangan, sehingga waktu dan biaya yang mereka keluarkan jauh tidak sebanding dengan manfaat atau keuntungan yang mereka peroleh.

Imbalan yang Tidak Adil. Para pelaksana tim yang menonjol mungkin menerima 5 persen kenaikan upah atau gaji, dan para pelaksana yang tidak menonjol dan sedang-sedang saja bisa menerima $4\frac{1}{2}$ kenaikan upah. Akhirnya pelaksana yang menonjol ini menjadi frustasi dan keluar. Banyak cara yang bisa dilakukan untuk memberikan imbalan atau hadiah ini, seperti pemberian bonus kinerja khusus, pemberian bonus perjalanan, dan pemberian kesempatan untuk mengikuti seminar khusus.

Dalam beberapa kasus, penggunaan anggota-anggota tim yang berketerampilan dan bermotivasi tinggi akan bisa mengurangi waktu yang diperlukan sebesar tiga sampai lima kali dari waktu yang diperlukan apabila kita menggunakan anggota-anggota tim sedang-sedang saja dan kurang inspirasi. Lebih dari itu, pencapaian produktivitas tambahan dapat dicapai dengan memberi para pelaksana puncak dengan perangkat yang lebih canggih seperti CASE.

MEMPRODUKSI PERANGKAT LUNAK BERKUALITAS TINGGI

Sasaran akhir pengembangan perangkat lunak adalah untuk menghasilkan perangkat lunak berkualitas tinggi pada tingkat produktivitas yang tinggi yang memberi nilai tambah kepada perusahaan. Kita telah mengetahui mengenai tingkat produktivitas atau penyerahan, namun apa yang dimaksud dengan kualitas perangkat lunak dan bagaimana kita memperolehnya? Jawaban atas pertanyaan ini mempunyai tiga dimensi.

Pertama, dari sudut pandang end user, kualitas perangkat lunak diukur berdasarkan faktor-faktor kinerja, seperti:

- ◆ Kinerja pengoperasian keseluruhan.
- ◆ Kemudahan pembelajaran.
- ◆ Pengontrolan dan penanganan kesalahan.
- ◆ Dukungan dari pembuat dan pemelihara.

Faktor-faktor rancangan MURRE dan faktor-faktor strategik PDM, memberikan dua dimensi lain — dimensi kualitas rancangan perangkat lunak. Faktor-faktor MURRE mencakup:

- ◆ Kemungkinan pemeliharaan (*Maintainability*)
- ◆ Kemungkinan penggunaan (*Usability*)
- ◆ Kemungkinan penggunaan ulang (*Reusability*)
- ◆ Reliabilitas (*Keandalan*)
- ◆ Kemungkinan perluasan (*Extendability*)

Perangkat lunak berkualitas tinggi mendukung faktor-faktor strategik PDM dengan membantu:

- ◆ Meningkatkan produktivitas
- ◆ Menambah keragaman produk dan pelayanan
- ◆ Meningkatkan fungsi manajemen

Sekarang, bagaimana kita memperoleh kualitas perangkat lunak? Kita mencapainya dengan cara mengimplementasikan jaminan kualitas yang kuat dan teknik pengendalian kualitas yang ketat.

Apa yang Dimaksud Jaminan Kualitas?

Quality Assurance (QA)/jaminan kualitas memastikan bahwa SWDLC yang digunakan dalam pengembangan produk perangkat lunak berkualitas tunduk kepada (sesuai dengan) standart-standart yang telah ditetapkan bagi produk itu. Pada skala yang lebih luas, jaminan kualitas mencakup pemonitoran terus menerus terhadap semua tahap-tahap pengembangan sistem dan perangkat lunak, dari perencanaan sistem sampai pengimplementasiannya. Selain itu, ia mencakup pengoreksian terhadap proses pengembangan, sehingga kualitas sistem dan perangkat lunak yang dihasilkan selanjutnya akan meningkat.

Ada suatu klise dalam sistem yang menyatakan: "Tak pernah ada waktu yang cukup untuk menyempurnakannya, namun banyak waktu untuk menyelesaikannya." Realitasnya adalah bahwa jika pekerjaan dilakukan secara benar pada awalnya, dimana ini merupakan esensi dari jaminan kualitas, maka ia tidak akan berisi kesalahan dan tidak akan perlu dikerjakan ulang. Kadang-kadang, kendala yang ketat terhadap proses pengembangan akan menyebabkan lebih lambatnya tanggal penyerahan, namun tanggal penyerahan dari suatu proyek yang terkelola dengan baik juga akan merupakan tanggal penyerahan akhir suatu program lengkap yang siap dioperasikan. Dalam kasus dimana proyek tidak dikelola secara baik dan dikerjakan secara tergesa-gesa, maka apa yang disebut penyerahan lengkap sebenarnya masih sedang dikembangkan, dan penyerahan yang sebenarnya akan dilakukan lama sesudahnya.

Apa yang Dimaksud Pengendalian Kualitas?

Sementara jaminan kualitas memfokuskan pada proses pengembangan sistem dan perangkat lunak, Quality Control/pengendalian kualitas memfokuskan pada produknya; yaitu, apa yang dihasilkan. Meskipun mungkin prosesnya dikatakan berhasil, namun produknya (yakni, sistem dan perangkat lunak) bisa saja gagal. Karena pengendalian kualitas mengevaluasi sistem dan perangkat lunak setelah mereka dikembangkan, tidak ada aktivitas pengendalian kualitas yang bisa meningkatkan kualitas. Kualitas harus dibangun/dirancang ke dalam sistem dan perangkat lunak ketika mereka sedang dibuat, tidak setelah pembuatannya selesai. Maka, kalau jaminan kualitas adalah teknik pencegahan kesalahan, pengendalian kualitas merupakan teknik penghapusan kesalahan. Jaminan kualitas tidak pernah bisa menggantikan pengendalian kualitas, atau sebaliknya.

Menciptakan Kelompok QA

Beberapa organisasi membentuk kelompok jaminan kualitas (QA) independen. Kelompok seperti ini bisa terbentuk atas wakil-wakil end-user, analis sistem, perancang sistem, dan programmer terampil, yang semuanya tidak tergantung pada developer (pembuat). Tugas-tugas kelompok QA mencakup:

- ◆ Menetapkan standar untuk pengembangan sistem dan perangkat lunak, misalnya menganut SDLC dan SWDLC.
- ◆ Mengevaluasi laporan terdokumentasi yang siap diserahkan.
- ◆ Menjalankan tahapan pemeriksaan rancangan sistem dan perangkat lunak.
- ◆ Melakukan tahapan pemeriksaan pengkodean.
- ◆ Menjalankan pengujian.

Dalam apa yang disebut oleh para ahli sebagai “cleanroom approach”, para programmer pengembangan tidak hanya tidak boleh menguji kode yang mereka tulis, namun juga tidak boleh menyusunnya. Semua pengujian dilakukan oleh kelompok QA.

Dalam PENELUSURAN RANCANGAN PERANGKAT LUNAK (*software design walkthrough*), kelompok QA memeriksa atau meninjau secara seksama spesifikasi rancangan guna mendeteksi ketidakkonsistenan dan kesalahan sebelum pengkodean dimulai. Dalam PENELUSURAN KODE (*code walkthrough*), kelompok QA memeriksa kode sebenarnya untuk mengetahui apakah ia cocok atau sesuai dengan spesifikasi

rancangan, dan kelompok ini juga mensimulasi bagaimana kode seperti itu akan diproses oleh komputer agar kelompok ini bisa menemukan kesalahan pengkodean yang ada. Semua ini dilakukan sebelum dimulainya pengujian berbasis komputer.

Beberapa ahli menyatakan bahwa penelusuran (*walkthrough*) dapat menemukan 75 sampai 90 persen kelemahan rancangan dan kesalahan pengkodean sebelum pengujian. Beberapa ahli percaya bahwa cara tersebut merupakan satu-satunya teknik peningkatan kualitas paling penting untuk perangkat lunak. Selain itu mereka menyatakan bahwa organisasi-organisasi yang melakukan penelusuran independen telah meningkatkan tingkat produktivitas dari 14 ke 25 persen.

Pada saat ini, anda bisa bertanya pada diri anda sendiri: Dimana pengendalian kualitas menjadi berperan? Aktivitas pengendalian kualitas biasanya dilakukan setelah hampir semua prosedur pengujian lain telah diterapkan dan apabila kelompok QA dan developer (pengembangan sistem) cukup yakin bahwa perangkat lunak tersebut siap diterbitkan (dikeluarkan) untuk diimplementasikan dan dikonversi ke operasi. Dengan demikian aktivitas pengendalian kualitas menjadi berperan selama pengujian sistem dan pengujian penerimaan — terutama pengujian penerimaan. Kedua prosedur pengujian ini akan dibahas dalam Bab 4. Sekali lagi, dengan menggunakan manufakturing sebagai analogi, bisa dikatakan bahwa cara penggarapan pengendalian kualitas ini sama dengan pengendalian kualitas dalam manufakturing, dimana produk dievaluasi pada akhir lini perakitan apabila produksi ini selesai dan produk siap ditransfer ke barang jadi yang siap dikirimkan ke pelanggan.

MERENCANAKAN PROYEK SIKLUS HIDUP PENGEMBANGAN PERANGKAT LUNAK

Tujuan game plan SWDL adalah untuk memungkinkan manajer proyek menjadwal dan memonitor semua tugas yang diperlukan untuk menyelesaikan SWDLC. Perangkat populer yang digunakan untuk memformulasikan game plan pengembangan perangkat lunak adalah teknik tinjauan dan evaluasi program (PERT). Gambar 1.8 menunjukkan jaringan PERT yang berisi tugas-tugas yang diperlukan untuk mengembangkan program lengkap.

Sasaran PERT adalah untuk menentukan rangkaian atau urutan pelaksanaan tugas pengembangan perangkat lunak dan untuk mengestimasikan lamanya waktu yang diperlukan dari awal sampai selesainya pelaksanaan tugas. Lamanya proyek yang terdiri atas serangkaian tugas yang sangat panjang harus dijalankan secara urut,

dimana lamanya pelaksanaan rangkaian tugas ini merupakan jalur kritis (*critical path*) dari proyek tersebut.

Empat langkah yang harus diambil untuk menyusun jaringan PERT pengembangan perangkat lunak adalah:

1. Mengidentifikasi semua tugas pengembangan perangkat lunak yang harus dijalankan.
2. Mengestimasi waktu yang diperlukan untuk menjalankan setiap tugas.
3. Menentukan atau menetapkan rangkaian tugas.
4. Menentukan jalur kritis yang akan menunjukkan waktu pengembangan perangkat lunak keseluruhan.

Hasil dari langkah pertama adalah daftar tugas pengembangan perangkat lunak seperti terlihat pada Gambar 1.9. Setiap tugas diidentifikasi dengan nomor dan deskripsi (penjabaran).

Dalam langkah kedua, waktu yang diharapkan guna menyelesaikan tugas diestimasi oleh manajer, dan para staf yang berpengalaman dan dengan menggunakan metrik produktivitas (misalnya, metrik titik fungsi). Tujuan dalam mendapatkan tiga estimasi ini adalah untuk digunakan mengkalkulasi rata-rata bobot tunggal untuk setiap tugas. Rata-rata bobot ini adalah waktu yang diharapkan, yang ditunjukkan dengan formula berikut ini:

$$TE = \frac{O + 4M + P}{6}$$

dimana TE = waktu yang diharapkan

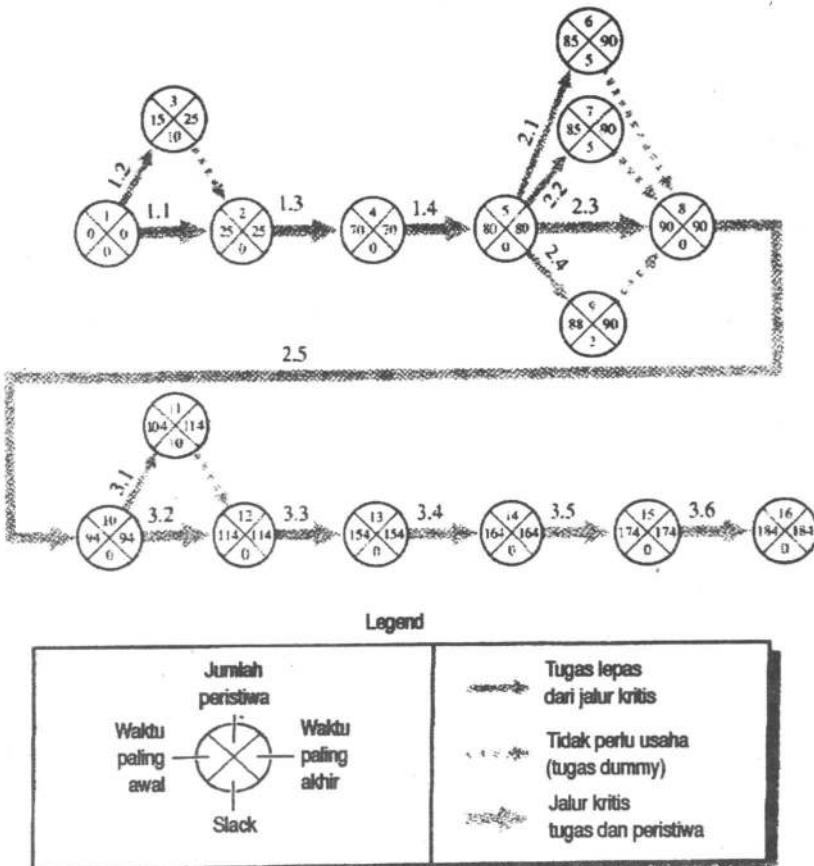
O = estimasi waktu optimistik

M = estimasi waktu yang paling mungkin

P = estimasi waktu pesimistik

Formula ini didasarkan pada asumsi bahwa estimasi waktu mendekati distribusi beta.

Langkah ketiga dalam penciptaan jaringan PERT pengembangan perangkat lunak adalah mengurutkan tugas. Urutan tugas ditentukan berdasarkan ketergantungan (dependencies). Tugas-tugas tertentu harus diselesaikan sebelum tugas-tugas lain dimulai. Dalam beberapa hal, tugas-tugas dapat dijalankan secara serentak. Dalam jaringan PERT pengembangan perangkat lunak kita, kita melihat bahwa Tugas 1.1 dan 1.2 dapat dijalankan secara serentak; yakni, mentransformasikan diagram arus



Gambar 1.8 Jaringan PERT untuk proyek pengembangan perangkat lunak

data ke bagan struktur dan menyiapkan kamus data dari output, input, dan penyimpanan data dapat dikerjakan pada waktu yang sama. Namun Tugas 1.3, penulisan bahasa Inggris terstruktur untuk modul-modul, tidak dapat dilakukan sampai bagan struktur modul selesai. Setelah tiga tugas pokok dalam tahap rancangan perangkat lunak ini selesai, penelusuran rancangan perangkat lunak dapat dilakukan guna melihat apakah rancangan perangkat lunak ini memenuhi spesifikasi dari rancangan sistem terinci.

Tugas	Deskripsi	Waktu yang diharapkan (da- lam Person-day)
1.1	Mentransformasi diagram arus data ke bagan struktur	25
1.2	Mentransformasi output, input, dan penyimpanan data ke kamus data	15
1.3	Menyiapkan Bahasa Inggris terstruktur untuk setiap modul	45
1.4	Melaksanakan penelusuran rancangan	10
2.1	Memprogram modul output	5
2.2	Memprogram modul input	5
2.3	Memprogram modul keputusan	10
2.4	Memprogram modul komputasi	8
2.5	Melaksanakan penelusuran kode	4
3.1	Merancang kasus pengujian	10
3.2	Penyiapan <i>driver</i> dan <i>stub</i>	20
3.3	Menjalankan pengujian modul	40
3.4	Menjalankan pengujian integrasi	10
3.5	Menjalankan pengujian sistem	10
3.6	Menjalankan pengujian penerimaan	10

Gambar 1.9 Tugas-tugas untuk pengembangan perangkat lunak terstruktur dan lamanya waktu yang diharapkan

Setelah pengkodean Tugas 2.1, 2.2, 2.3, dan 2.4 selesai, dilakukanlah penelusuran kode. Jika modul-modul bisa diterima oleh tim penelusuran, pengujian perangkat lunak dapat dimulai. Dalam tahap ini, Tugas 3.1, Perancangan kasus pengujian, dan Tugas 3.2, Penyiapan *driver* dan *stub*, dapat dilakukan bersamaan. Setiap langkah pengujian - 3.3, Pengujian modul; 3.4, Pengujian integrasi; 3.5, Pengujian sistem; dan 3.6, Pengujian penerimaan — harus dilakukan secara urut. Dengan mengasumsikan bahwa perangkat lunak tersebut lulus dari semua pengujian, proyek pengembangan perangkat lunak dianggap selesai dan program itu sendiri siap untuk diimplementasikan ke dalam sistem baru keseluruhan dan dikonversi dari status pengembangan menjadi status yang bisa dioperasikan secara full-time.

Langkah terakhir adalah menganalisis jaringan PERT yang telah diselesaikan sejauh ini dan mengkomputasi jalur kritis. Manajer proyek perlu mengetahui

seberapa dininya suatu event bisa dimulai dan seberapa lambatnya event tersebut bisa berakhir, tanpa mempengaruhi jadwal proyek keseluruhan. Waktu paling dini, yang ditunjukkan sebagai ET, adalah 0 untuk Event 1. Untuk event-event lain, ET adalah jumlah tertinggi dari lamanya tugas dan ET dari event yang tepat mendahuluinya. Sebagai contoh, ada empat tugas yang mendahului Event 5. ET dari Event 3 adalah 15 person-days ($0 + 15$); ET dari Event 2 adalah 25 person-days ($0 + 25$); ET dari Event 4 adalah 70 person-days (ia memerlukan 25 person-days untuk menyelesaikan Event 2 dan 45 person-days untuk menyelesaikan Event 4); ET dari Event 5 adalah 80 ($70 + 10$). ET lainnya untuk setiap event dihitung dengan cara yang sama.

Waktu event paling akhir, yang ditunjukkan dengan LT, adalah waktu paling akhir bisa dimulainya event tanpa menunda proyek. Untuk mengkalkulasi LT, kita akar menghitungnya secara mundur sepanjang jaringan PERT, yang dimulai dari kanan atau event terakhir pada jaringan itu. LT adalah selisih terkecil antara LT event dikurangi waktu lamanya tugas. Sebagai contoh, LT untuk Event-event 6, 7, 8, dan 9 adalah masing-masing 90 person-days. Untuk menghitung LT untuk Event 5 masing-masing waktu yang diharapkan (atau lamanya), seperti terlihat pada Gambar 1.9. Tugas 2.1 adalah 5 person-days, 2.2 adalah 5, 2.3 adalah 10, dan 2.4 adalah 8. Dengan mengurangkan lamanya waktu dari 90, kita akan mendapatkan, berturut-turut, $90 - 5 = 85$, $90 - 5 = 85$, $90 - 10 = 80$, dan $90 - 8 = 82$. Dengan demikian waktu paling lambat (akhir) untuk menyelesaikan Event 5 adalah 80 person-days. Sebagai contoh, jika kita menyelesaikannya dalam 85 person-days, kita tidak akan bisa menyelesaikan Event 8 tepat waktu, sebab kita hanya mempunyai kelonggaran 5 person-days berdasarkan jadwal.

Jalur kritis (*critical path*) dalam contoh kita tersebut ditunjukkan dengan garis garis lebar. Ia ditentukan dengan cara menghubungkan semua event yang berisi slack time (kelonggaran waktu) nol; yakni semua event yang ET dan LT-nya sama. Jalur kritis ini merepresentasikan penetapan tugas yang harus dimonitor oleh manajer proyek secara cermat. Ia mengidentifikasi event-event yang harus dimulai dan diselesaikan tepat waktu dan yang tidak memerlukan apa-apa lagi selain lamanya waktu yang diestimasikan. Suatu kelambatan/penundaan tugas pada jalur kritis akan menghasilkan terhadap proyek keseluruhan. Tugas-tugas lain yang mempunyai slack waktu (kelonggaran waktu) memungkinkan manajer proyek mengalokasikan sumber daya dari tugas-tugas ini ke tugas-tugas yang berada di jalur kritis jika jadwal proyek mulai dikerjakan.

Karena PERT bisa digunakan untuk komputer personal, manajer proyek sebaiknya menggunakan untuk meningkatkan pengendalian proyek dan untuk mencapa-

tingkat produktivitas yang tinggi. Pengestimasian, penjadwalan, dan pengendalian tugas-tugas proyek interdependen akan membantu dalam penetapan atau penentuan jadwal dan anggaran, jika mereka dijalankan secara baik.

TINJAUAN SASARAN BELAJAR UNTUK BAB INI

Tujuan pokok bab ini adalah untuk memungkinkan setiap siswa mencapai tujuh tujuan pembelajaran yang penting. Sekarang kita akan meringkas tanggapan terhadap sasaran belajar ini.

Sasaran belajar 1:

Membedakan berbagai sumber perangkat lunak aplikasi dan menjelaskan cara mengevaluasi dan menyeleksi paket perangkat lunak komersial.

Perangkat lunak aplikasi bisa dibeli dari vendor perangkat lunak komersial sebagai produk siap pakai ataupun dikembangkan oleh para staf sistem informasi perusahaan atau oleh kontraktor independen. Faktor-faktor kinerja perangkat lunak umum adalah:

- ◆ Kinerja pengoperasian
- ◆ Dokumentasi
- ◆ Kemudahan pembelajaran
- ◆ Kemudahan penggunaan
- ◆ Pengendalian dan penanganan kesalahan
- ◆ Dukungan

Setiap faktor ditimbang (diberi bobot) dan dinilai untuk menghasilkan angka penilaian total bagi paket komersial yang sedang dievaluasi. Angka ini dibagi menjadi biaya setiap paket guna mengkalkulasi biaya per angka penilaian. Paket perangkat lunak dengan biaya terendah per angka penilaian kita pilih untuk penginstalasian.

Sasaran belajar 2:

Mendefinisikan atau menetapkan siklus hidup pengembangan perangkat lunak (SWDLC) dan menjelaskan tahapannya secara singkat.

Siklus hidup pengembangan perangkat lunak (SWWDL) adalah metodologi yang dianut untuk mengembangkan proyek perangkat lunak; yakni,, suatu program aplikasi yang menjalankan sebagian dari atau semua rancangan sistem. SWDLC terdiri dari tiga tahap:

- ◆ Rancangan
- ◆ Kode
- ◆ Uji

Tahap rancangan mengkonversi rancangan sistem ke rancangan modul program — misalnya, konversi diagram arus data ke bagan struktur. Tahap kode mencakup penulisan sebenarnya bahasa pemrograman ke dalam modul-modul program yang merepresentasikan rancangan. Tujuan tahap uji adalah untuk mendeteksi dan menghapus kesalahan rancangan dan kesalahan pengkodean. Persentase relatif dari usaha yang dicurahkan untuk tahap-tahap ini adalah, berturut-turut, 40-20-40.

Sasaran Belajar 3:

Menejelaskan pengorganisasian proyek pengembangan perangkat lunak; menetapkan tim pengembangan program, tim programmer kepala, dan tim pemrograman bersama; dan menganalisis keuntungan dan kerugiannya.

Pengorganisasian orang-orang ke dalam tim terfokus dan termotivasi tinggi yang bekerja bersama dengan cara yang terpadu dan terkoordinasi merupakan hal terpenting demi tercapainya pengembangan proyek perangkat lunak yang berhasil. Tim pengembangan program adalah campuran dari perancang, pengkode, dan penguji yang merefleksikan aturan SWDLC 40-20-40 dalam hal jumlahnya. Tim programmer kepala terbentuk dari programmer kepala, asisten utama, programmer pendukung, pustakawan, administrator, editor, dan klerk program. Tim pemrograman

bersama terbentuk dari sejumlah programmer yang bekerja bersama dengan cara demokratis sepenuhnya.

Tim pengembangan program disesuaikan dengan SWDLC, sebab merupakan campuran dari perancang, pengkode, dan penguji. Tim programmer kepala diorganisasi dan dikoordinasi secara ketat. Oleh karenanya, sangat cocok untuk menjalankan pengkodean, namun tidak memberi penekanan pada tahap rancangan dan pengujian. Tim pemrograman bersama bekerja tanpa supervisi. Seperti halnya tim-tim yang tidak mempunyai kekuatan koordinasi (tidak ada supervisornya), tim ini cenderung menyimpang dan tidak mencapai tujuannya, atau mencapai tujuannya dengan target waktu yang menyimpang jauh. Tim ini juga lebih menekankan pada pengkodean dari pada perancangan dan pengujian.

Sasaran belajar 4:

Menetapkan produktivitas perangkat lunak dan mengemukakan dua cara untuk mengukur produktivitas ini. Juga menjelaskan dampak manajemen terhadap produktivitas perangkat lunak.

Definisi umum produktivitas adalah:

$$\text{Produktivitas} = \frac{\text{output yang dihasilkan}}{\text{input yang dikonsumsi}}$$

Penetapan produktivitas perangkat lunak dengan menggunakan metrik LOEC adalah LOEC per person-month. Penetapan produktivitas perangkat lunak dengan menggunakan metrik titik fungsi adalah:

$$\text{Tingkat produktivitas pengembangan} = \frac{\text{jumlah titik fungsi yang dihasilkan}}{\text{jumlah person-month}}$$

Titik fungsi terbentuk atas sejumlah input, output, inquiry, file, dan interface, bersama dengan derajad kompleksitas mereka.

Manajemen yang tidak baik akan menurunkan produktivitas perangkat lunak. Manajemen yang tidak baik ini akibat dari:

- ◆ Tidak mempunyai atau tidak menggunakan metrik produktivitas
- ◆ Perencanaan dan pengendalian yang tidak baik
- ◆ Pengorganisasian yang tidak baik
- ◆ Pemberian tekanan yang lebih banyak pada pengkodean dari pada perancangan (*designing*) dan pengujian.
- ◆ Pemberian penghargaan kepada pelaksana terbaik dengan nilai yang sama (hampir sama) seperti pelaksana yang sedang-sedang saja.

Sasaran belajar 5:

Menerangkan kualitas perangkat lunak dan menjelaskan jaminan kualitas dari pengendalian kualitas.

Kualitas perangkat lunak terbentuk dari tiga dimensi:

- ◆ Dimensi end-user, yang memberikan kinerja pengoperasian, kemudahan pembelajaran (mudah dipelajari), pengendalian dan penanganan kesalahan, dan dukungan.
- ◆ Dimensi rancangan, yang mencapai faktor rancangan kemungkinan pemeliharaan, penggunaan, penggunaan ulang, reliabilitas, perluasan (MURRE).
- ◆ Dimensi nilai tambah, yang membantu mendukung faktor-faktor strategik produktivitas, diferensiasi, dan manajemen (PDM) perusahaan.

Jaminan kualitas (QA) adalah proses merancang kualitas ke dalam sistem. Pengendalian kualitas adalah proses yang memastikan bahwa kualitas sistem telah dipenuhi/ tercapai. Kelompok QA memastikan bahwa baik proses jaminan kualitas maupun pengendalian kualitas diterapkan.

Sasaran belajar 6:

Menjabarkan bagaimana teknik tinjauan dan evaluasi program (PERT) digunakan sebagai teknik game plan SWDLC dan manajemen proyek.

Teknik PERT membantu manajer proyek:

- ◆ Mengidentifikasi semua tugas yang harus dijalankan dan menetapkan urutan logik pelaksanaan tugas.
- ◆ Mengestimasi waktu yang diperlukan untuk menjalankan setiap tugas.
- ◆ Menentukan jalur kritis pada proyek tersebut.

Ini akan menghasilkan suatu game plan untuk merencanakan, menjadwal, dan mengendalikan proyek pengembangan perangkat lunak.

DAFTAR PERIKSA PENGEMBANGAN PERANGKAT LUNAK

Berikut ini adalah daftar periksa tentang pelaksanaan pengembangan perangkat lunak. Tujuannya adalah untuk mengingatkan anda tentang persoalan-persoalan pokok yang terlibat dalam pengembangan perangkat lunak. Tahap-tahap spesifik pengembangan perangkat lunak dibahas dalam Bab 2, 3, dan 4.

1. Selesaikan Laporan Rancangan Sistem Terinci, dan pastikan bahwa semua pihak yang berkepentingan menyepakatinya.
2. Gunakan Laporan Rancangan Sistem Terinci tersebut sebagai blueprint untuk pengkonstruksian sistem, seperti halnya pembuat bangunan menggunakan blueprint-nya arsitek untuk mengkonstruksi bangunan.
3. Tentukan aplikasi-aplikasi mana yang memerlukan dukungan paket perangkat lunak komersial atau dukungan perangkat lunak sesuai pesanan.
4. Jika kita memilih paket perangkat lunak komersial, siapkan atau buatlah permintaan proposal (RFP) untuk disampaikan kepada para vendor perangkat lunak komersial.
5. Evaluasilah dan nilailah paket dari vendor perangkat lunak komersial dan pilihlah yang paling tepat.
6. Jika kita memilih program perangkat lunak sesuai pesanan, buatlah organisasi tim pemrograman, dan pastikan bahwa mereka mengikuti atau menerapkan siklus hidup pengembangan perangkat lunak (SWDLC).
7. Kembangkan metrik yang bisa bekerja untuk mengukur produktivitas pengembangan perangkat lunak. Dua metrik yang bisa bekerja adalah metrik jalur kode yang bisa dieksekusi (LOEC) dan metrik titik fungsi.
8. Susunlah (tetapkan) sistem jaminan dan pengendalian kualitas total untuk menjamin kualitas perangkat lunak tingkat tinggi.

9. Gunakan PERT untuk menjadwal, memonitor, dan mengendalikan SWDLC.

PERTANYAAN TINJAUAN

- 1.1 Sebutkan dua sumber utama perangkat lunak aplikasi!
- 1.2 Dalam kondisi bagaimana kita lebih tepat menggunakan paket perangkat lunak komersial? Dalam kondisi bagaimana perangkat lunak terkustomisasi tepat digunakan?
- 1.3 Sebutkan dan jelaskan secara singkat keuntungan dan kerugian paket-paket perangkat lunak komersial!
- 1.4 Mengapa RFP berorientasi-kinerja harus dibuat untuk disampaikan kepada vendor perangkat lunak komersial? Harus berisi apa saja RFP ini?
- 1.5 Sebutkan dan definisikan faktor-faktor kinerja umum yang digunakan untuk mengevaluasi perangkat lunak komersial!
- 1.6 Definisikan/tentukan siklus hidup pengembangan perangkat lunak (SWDLC). Jabarkan secara singkat setiap tahapnya. Apa yang dimaksud dengan aturan 40-20-40?
- 1.7 Jelaskan bagaimana proyek pengembangan perangkat lunak yang besar dapat menimbulkan masalah dalam pengkoordinasian rancangan dan komunikasi.
- 1.8 Jelaskan bagaimana jalur komunikasi dan interface yang berlebihan antara para anggota tim dapat menyebabkan menurunnya produktivitas. Berikan contohnya!
- 1.9 Jelaskan mengapa perancang dan analis sistem harus dilibatkan dalam pengembangan perangkat lunak.
- 1.10 Sebutkan tiga jenis tim yang digunakan untuk mengembangkan perangkat lunak. Mana yang disesuaikan dengan (diadaptasi dari) aturan 40-20-40?
- 1.11 Jelaskan mengapa orang-orang yang mempunyai keterampilan tinggi harus ditugaskan untuk menangani perancangan dan pengujian?
- 1.12 Tim pemrograman bersama terbentuk dari 20 orang. Berapa banyak lintasan komunikasi dan interface yang harus ada diantara para anggota tim ini?
- 1.13 Jelaskan mengapa tim programmer kepala bisa memenuhi deadline pengkodean secara lebih baik dari pada tim pemrograman bersama?

- 1.14 Definisikan/tentukan produktivitas untuk segala usaha. Bagaimana formula atau rumus untuk produktivitas?
- 1.15 Sebutkan dua metrik populer yang digunakan untuk mengukur produktivitas pengembangan perangkat lunak.
- 1.16 Sebutkan keuntungan dan kerugian metrik LOEC.
- 1.17 Jelaskan mengapa Frederick P. Brroks, Jr., mengatakan bahwa person-month (menurut istilahnya, man-month) yang digunakan sebagai parameter estimasi dan parameter kalkulasi produktivitas merupakan mitos yang mempedaya?
- 1.18 Berikan suatu contoh dimana lima orang yang bekerja bersama pada suatu proyek akan lebih produktif dibandingkan dengan lima orang yang bekerja sendiri-sendiri. Definisikan sinergisme dan jelaskan mengapa, dalam beberapa hal, keseluruhan akan lebih besar dari pada jumlah bagian-bagiannya.
- 1.19 Sebutkan dan berilah contoh singkat lima titik fungsi.
- 1.20 Jelaskan bagaimana metrik titik fungsi mencakup perancangan, pengkodean, dan pengujian perangkat lunak.
- 1.21 Apa keuntungan atau keunggulan dari metrik titik fungsi? Apa nama organisasi yang mendukung metrik titik fungsi?
- 1.22 Apa yang dimaksud dengan tingkat produktivitas pengembangan (yang juga disebut tingkat penyerahan)? Bagaimana menghitungnya?
- 1.23 Jelaskan pengaruh manajemen terhadap tingkat produktivitas suatu tim proyek. Sebutkan aktivitas manajemen yang seringkali dapat menghilangkan atau menurunkan produktivitas.
- 1.24 Definisikan kualitas perangkat lunak dari sudut pandang rancangan, dari sudut pandang nilai tambah, dari sudut pandang end-user.
- 1.25 Apa perbedaan pokok antara jaminan kualitas dan pengendalian kualitas?
- 1.26 Jelaskan bagaimana tim proyek mencapai jaminan kualitas yang baik.
- 1.27 Apa yang difokuskan oleh pengendalian kualitas?
- 1.28 Haruskah kelompok QA menjadi bagian dari tim pengembangan? Jelaskan mengapa ya atau mengapa tidak?
- 1.29 Jelaskan statemen ini: Jaminan kualitas tidak pernah bisa mengganti pengendalian kualitas, atau sebaliknya.
- 1.30 Apa yang dimaksud “cleanroom approach?”
- 1.31 Sebutkan tugas-tugas pokok yang dijalankan oleh kelompok QA!

- 1.32 Jelaskan bagaimana PERT digunakan untuk mengestimasi, menjadwal, dan mengelola proyek pengembangan perangkat lunak.

SOAL SPESIFIK BAB INI

Soal-soal ini membutuhkan jawaban eksak yang langsung didasarkan pada konsep dan teknik yang disampaikan dalam buku ini.

- 1.33 Anda telah mempersempit jangkauan pencarian anda terhadap perangkat lunak manajemen-inventori komersial menjadi dua paket: Fifo dan Eoq. Dengan patokan 100, anda menilai (memberi angka) faktor-faktor kinerja umum sebagai berikut: penilaian vendor (20), kinerja pengoperasian (15), dokumentasi (10), kemudahan pembelajaran (10), kemudahan penggunaan (10), pengendalian dan penanganan kesalahan (20), dan dukungan (15). Setelah melakukan penelitian ulang dan melakukan benchmarking terhadap Fifo dan Eoq, anda menilai setiap faktor kinerja sebagai berikut:

Fifo	Eoq
5	.6
7	9
9	5
7	4
6	7
4	9
8	9

Fifo berbiaya \$37.290 dan Eoq berbiaya \$36.980.

Ditanyakan: Hitunglah skor faktor-faktor kinerja umum untuk setiap paket komersial tersebut. Berdasarkan biaya/manfaat, paket mana yang anda pilih?

- 1.34 Diestimasikan bahwa proyek perangkat lunak akan memerlukan 300 KLOEC. 1,5 KLOEC dapat dihasilkan oleh per person-month. Input yang diperlukan untuk mendukung satu person-month sebesar \$8000.

Ditanyakan: Berapa banyak person-month diperlukan untuk menyelesaikan proyek itu, dan berapa biaya yang diestimasikan untuk proyek tersebut? Jika 20 orang dipekerjakan untuk proyek itu, berapa bulan yang diperlukan untuk menyelesaikan proyek itu?

- 1.35 Biaya pengembangan perangkat lunak untuk Space Shuttle adalah \$1,2 miliar. Mereka memerlukan 25.600.000 jalur kode, dan dalam hal ini diperlukan 22.096 person-years untuk mengembangkannya (membuatnya). Biaya pengembangan perangkat lunak untuk perangkat lunak mesin teller Citibank adalah \$13,2 juta. Pengembangan ini memerlukan 780.000 jalur kode, yang dalam hal ini diperlukan 150 person-years.

Ditanyakan: Hitunglah jumlah person-month yang dibutuhkan untuk mengembangkan (membuat) satu KLOEC dan biaya per satu KLOEC untuk kedua proyek perangkat lunak tersebut. Sistem yang mana yang lebih banyak memerlukan usaha dalam hal waktu dan biaya per satu KLOEC? Sistem yang mana yang lebih banyak memerlukan pengujian? Mengapa? Apakah teknik-teknik pengestimasian, penjadwalan, dan pemonitoran bisa diterapkan untuk jenis-jenis proyek pengembangan perangkat lunak ini? Jelaskan!

- 1.36 Proyek perangkat lunak yang diusulkan akan mencakup 60 input, 20 output, 70 inquiry, 10 file, dan 100 interface, yang semuanya mempunyai derajad kompleksitas 5. Tingkat produktivitas pengembangan (tingkat penyerahan)-nya adalah 10 titik fungsi per person-month. Dibutuhkan biaya \$5000 untuk mendukung satu person-month.

Ditanyakan: Hitunglah jumlah titik fungsi dan biaya estimasi proyek tersebut. Jika 10 orang ditugaskan/dipekerjakan pada proyek ini, berapa bulan yang akan diperlukan untuk menyelesaikan proyek ini? Jika perusahaan beralih ke pendekatan workstation CASE pengembangan aplikasi bersama (JAD), diestimasikan bahwa delivery rate akan meningkat menjadi 40 titik fungsi per person-month dengan biaya \$8000 per person-month. Jika perusahaan beralih ke JAD dan CASE, berapa besar biaya proyek itu? Jika 10 orang dipekerjakan pada proyek itu, berapa bulan yang dibutuhkan untuk menyelesaikan proyek tersebut?

- 1.37 Tabel tugas dan lamanya waktu yang direncanakan disebutkan di bawah ini.

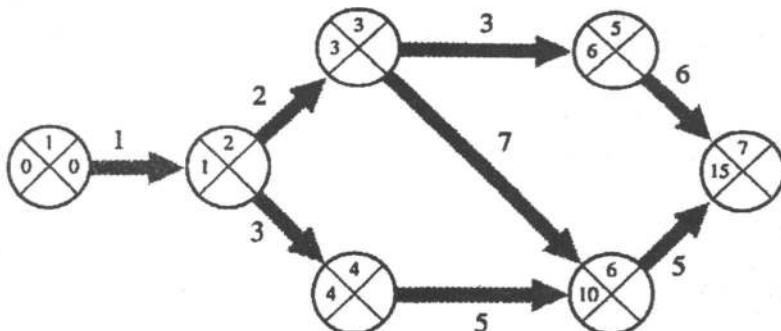
Tahap	Tugas	Waktu yang Diestimasikan (dalam minggu)
1	1.1	4,2
	1.2	4,0
	1.3	3,2
	1.4	3,0
	1.5	3,8

2	2.1	2,0
	2.2	1,2
	2.3	2,0
	2.4	3,8
3	3.1	2,0
	3.2	4,2
	3.3	3,8

Tugas-tugas dalam setiap tahap harus dijalankan secara urut, namun ketiga tahap tersebut bisa dijalankan secara berurutan. Terdapat 11 event, dan ketiga tahap tersebut bertemu pada event 11, yang merepresentasikan selesainya proyek tersebut.

Ditanyakan: Hitunglah waktu paling awal (ET), dan waktu paling lambat (LT), dan waktu slack. Gambarlah jaringan PERT yang menunjukkan nomor event, tugas, calculated times, dan jalur kritis (critical path).

- 1.38 Jaringan PERT berikut mencakup nomor event dan ET dari setiap tugas. TE dari setiap tugas ditempatkan pada jalur-jalur yang merepresentasikan tugas tertentu tersebut.



Ditanyakan: Hitunglah LT dan waktu slack pada setiap event dan sebutkan event-event secara urut sehingga akan merupakan jalur kritis. Waktu slack = LT - ET; LT = waktu paling lambat yang diperbolehkan untuk terjadinya suatu event; ET = waktu paling awal yang diharapkan untuk terjadinya suatu event.

SOAL UMUM

Soal-soal ini lebih memerlukan suatu pendekatan yang masuk akal (*feasible*) daripada suatu penyelesaian yang tepat. Walaupun soal-soal tersebut didasarkan pada bahan dalam bab, bacaan ekstra dan kreativitas mungkin diperlukan untuk mengembangkan penyelesaian yang dapat dikerjakan.

- 1.39 Public utility (dinas pekerjaan umum) memulai proyek pengembangan perangkat lunak pada tahun 1991. Sistem baru ini digunakan untuk mengkombinasikan (menggabungkan) tagihan biaya layanan air dan layanan listrik pada satu statemen bulanan, sebagai pengganti cara pengiriman tagihan untuk masing-masing secara terpisah. Jadwal aslinya menunjukkan bahwa sistem tersebut akan diimplementasikan dalam waktu satu setengah tahun. Sejauh ini para staf pemrograman telah menghabiskan waktu 40 person-year dan \$2 juta pada proyek tersebut. Manajer proyek berfikir bahwa akan diperlukan usaha separuh lagi dan dibutuhkan dana \$1,5 juta sebelum sistem tersebut bisa dioperasikan.

Ditanyakan: Analisis situasi di atas dan bahaslah atau jelaskan apa yang anda yakini sebagai kesalahan dalam proyek itu dan bagaimana jenis kelambatan/penundaan tersebut bisa dicegah. Jelaskan relevansi SDLC, SWDLC, metrik, tim pengembangan perangkat lunak, produktivitas, pengendalian kualitas dan jaminan kualitas, dan diagram PERT atau Gantt dengan masalah ini.

- 1.40 Dorothy Hamilton adalah manajer bagian peralatan berat pada Monarch Construction Company. Tanggung jawab utamanya adalah merencanakan dan mengkoordinasikan pentransferan/pemindahan peralatan berat, seperti bulldoser, derek, pompa air, mesin pemancang tiang/mesin lantak, dan mesin pengaspal, dari satu tempat proyek konstruksi ke tempat proyek lain dengan cara yang seefisien mungkin. Sebagai contoh, manajer proyek bandara mengestimasikan bahwa proyeknya tidak akan memerlukan tiga bulldoser setelah tanggal 1 Maret; manajer proyek jalan raya dapat melepaskan satu bulldoser pada 1 Maret. Manajer proyek pusat perbelanjaan akan memerlukan empat bulldoser pada 15 Maret. Dorothy menggunakan informasi semacam itu untuk memastikan bahwa manajer proyek menerima peralatan yang diperlukan.

Dorothy juga memerlukan informasi pemeliharaan peralatan online yang real-time. Sebagai contoh, oli mesin bulldoser harus diganti segera

setelah 50 jam operasi. Dorothy telah membakukan standart pemeliharaan yang sama untuk semua peralatan. Baik Dorothy maupun para mekanik di lapangan akan menerima informasi pemeliharaan yang sama. Segera setelah tugas-tugas pemeliharaan selesai, data yang menjabarkan tugas-tugas seperti itu akan dimasukkan ke dalam sistem untuk menjaga atau menyimpan status pemeliharaan setiap bagian peralatan pada saat itu. Secara random, Dorothy juga menggunakan informasi ini untuk mengecek peralatan guna memastikan bahwa para mekanik menuhi standart pemeliharaannya.

Sekitar satu tahun yang lalu, Dorothy menyampaikan (mengusulkan) System Service Request ke kelompok sistem informasi (IS) di Monarch. Clarence Moody, analis/programmer sistem, menginterview Dorothy dengan memintanya untuk menentukan keperluannya secara lebih rinci. Dia tidak pernah mendengar kabar dari Clarence lagi sampai sekarang. Selama waktu ini, Dorothy tidak puas dengan kurangnya layanan dari IS. Setelah berdebat dengan Clarence di coffee shop perusahaan, ia ingin bicara lagi dengannya di kantornya. Setelah keduanya berada di dalam kantor dan pintu ditutup, Dorothy berbicara kepada Clarence seperti berikut ini:

"Hampir setahun lewat, kita pernah bertemu di kantor ini untuk mulai merencanakan pengembangan sistem manajer peralatan berbasis komputer (CBEM). Saya jabarkan keperluan dan anda mengatakan memahaminya. Anda juga mengatakan bahwa anda bisa membangun sistem tersebut dalam waktu paling lambat sekitar tiga atau empat bulan — 'tak masalah' menurut anda. Namun, sekarang ada masalah, masalah yang buruk. Bulan-bulan telah berlalu. Saya telah kehilangan para asisten saya yang mengerjakan tugas-tugas manual, yang tugas-tugas ini akan bisa dikerjakan oleh sistem CBEM yang kita rencanakan. Saya terlambat berminggu-minggu dalam meng-update sistem manual. Setiap manajer proyek berada di pundak saya. Mereka tidak berfikir apa yang saya lakukan.

"Saya telah mencoba menghubungi anda lusinan kali, dan yang saya dapatkan hanyalah pesan cengeng dari mesin penjawab anda. Saya juga kadang-kadang telah menyempatkan ke kantor anda pada. Yang saya lihat hanyalah sekelompok orang yang duduk di depan PC bermain game tolol. Area IS kelihatannya seperti toko mainan yang berantakan. Orang-orang anda mungkin tahu caranya untuk 'memencet tombol' dan mengatasi keruwetan DOS, namun nampaknya anda tidak tahu bagaimana mengembangkan sistem untuk bisnis.

"Monarch telah melakukan bisnis ini selama tiga tahun. Sepengetahuan saya IS telah mengimplementasikan paket perangkat lunak akunting komersial segera setelah kita mulai bisnis ini, dan sistem tersebut berjalan secara cukup

baik. Namun demikian, saya juga mengetahui banyak permintaan sistem dari para pemakai di Monarch ini untuk digunakan sebagai aplikasi terkustomisasi, namun tak ada satupun yang telah dipenuhi pengimplementasianya, termasuk permintaan saya. Nampaknya tak ada yang tahu apa yang sedang terjadi. Saya telah mencoba menghubungi Bill Mallory (Bill adalah CIO-nya) dan dia tidak pernah berada di kantornya. Saya berjumpa dengannya di coffee shop sekitar dua bulan yang lalu dan menanyakan padanya mengenai sistem yang saya usulkan. Ia mengatakan pada saya bahwa ia telah mengeceknya. Saya tidak pernah mendengar kabar darinya sejak itu.

Saya bosan dan telah lelah dengan cara-cara yang dilakukan disini. Besok pagi saya akan bertemu dengan Martin (Martin Handelman adalah CEO Monarch). Saya pikir adil memberitahu anda bahwa saya akan mencari akar masalahnya dan ingin mengetahui mengapa orang-orang anda tidak melakukan tugasnya. Segala sesuatunya pasti berubah secara dramatis disini, dengan cara apapun. Jangan disangsikan lagi."

Ditanyakan: Anda bekerja untuk Sirius Systems Consultants, suatu firma yang terdiri atas orang-orang sistem nomor satu. Martin Handelman, CEO pada Monarch, telah memanggil anda dan berkata, "Kelompok/bagian IS kami telah berantakan atau mengalami kekacauan, dan kami memerlukan beberapa bantuan luar yang independen untuk memperbaikinya. Dapatkah anda membantu kami?" Dua bulan telah berlalu sejak anda menandatangani persetujuan bantuan kepada Monarch. Walaupun penjelasan atau komentar yang dibuat oleh Dorothy sangat tajam, dan beberapa komentarnya bahkan tidak beralasan, namun penilaianya terhadap kelompok IS cukup akurat.

Buatlah rekapitulasi eksekutif yang menyebutkan rekomendasi-rekomendasi pokok yang anda buat untuk memperbaiki kinerja kelompok IS dan meningkatkan kredibilitasnya. (*Petunjuk:* Walaupun riset luar mungkin membantu pemerolehan pendekatan atau cara yang layak untuk mengatasi masalah di Monarch, cukuplah bagi kita untuk mengulangi membaca bab-bab sebelumnya untuk mendapatkan cara tersebut. Laporan rekapitulasi eksekutif harus ringkas, namun merupakan set rekomendasi yang komprehensif. Biasanya, para eksekutif hanya akan melihat hal-hal/masalah-masalah pokoknya saja, bukan rinciannya).

- 1.41 "Nampaknya tidak akan pernah ada waktu cukup untuk mengerjakan tugas secara benar pada mulanya, namun selalu ada waktu untuk mengerjakannya kembali."

Ditanyakan: Jelaskan bagaimana SWDLC mengatasi kecenderungan tingkah laku manusia yang tersirat dalam pernyataan di atas. Jenis kesalahan apa yang dilakukan jika orang-orang tergesa-gesa memasuki/mulai menjalankan tahap pengkodean dari suatu proyek perangkat lunak? Bandingkan usaha relatif yang diperlukan untuk mencegah kesalahan dengan usaha relatif yang diperlukan untuk mendeteksi dan mengoreksinya. Bacalah satu atau dua artikel mengenai pabrik perangkat lunak Jepang, dan cobalah kaitkan apa yang anda pelajari dari artikel tersebut dengan pernyataan di atas.

KASUS JOCS: Memanajemen Proyek Pengembangan Perangkat Lunak

Jake Jacoby, manajer proyek tim SWAT untuk sistem pembiayaan job-order (JOCS), selama tahap analisis dan tahap rancangan sistem, mulai memutuskan bahwa JOCS akan memerlukan pemecahan perangkat lunak custom (buatan sendiri). Ia mulai membuat keputusan itu ketika salah satu calon end user JOCS, seorang akuntan pembiayaan, dalam suatu meeting menyatakan:

Saya tidak begitu yakin laporan-laporan apa yang akan kita perlukan (yang dihasilkan) dari sistem ini enam bulan dari sekarang. Kita sedang bekerja dalam suatu lingkungan yang sangat kompetitif, dan sistem ini dirancang untuk membantu kita mengontrol pembiayaan dan memonitor produktivitas. Saya dapat menjabarkan laporan yang akan membantu saya sekarang juga, namun saya tidak bisa memprediksi apa yang akan membantu saya di waktu kemudian. Dari pada hanya merancang laporan, rancangkan saya suatu sistem yang dapat saya gunakan. Saya tahu bagaimana menggunakan komputer; buatlah sistem ini sesuatu yang bisa saya ubah ketika saya memerlukannya berubah. Biarkan saya membuat laporan yang ketinggalan jaman.

Ini bukan satu-satunya end-user yang menginginkan perubahan output JOCS di masa mendatang. Setelah tim SWAT memulai rancangan sistem terinci, banyak end user mendukung dan memberi dorongan untuk diciptakan rancangan sistem yang fleksibel. Supervisor pada departemen manufakturing mengatakan bahwa ia menginginkan

suatu cara untuk membuat estimasi bagi setiap pekerjaan individual. Setiap pengangkat (lifter) hidronautika yang kita buat adalah berlainan/khas. Saya tidak bisa menetapkan set standart bahan dan buruh dan menerapkan standart itu pada setiap lifter. Saya perlu sistem

yang memungkinkan saya mengubah keperluan/persyaratan untuk setiap mesin yang berbeda. Bagaimana jika manajemen memutuskan untuk membuat produk baru di masa mendatang? Berikan saya sistem yang fleksibel. Saya mau belajar cara menggunakan salah satu PC-things jika saya bisa mendapatkan yang saya inginkan darinya. Namun saya tidak ingin menjadi programmer atau semacamnya, maka belikan saya sesuatu yang dapat saya pelajari dalam beberapa minggu.

Jake mengetahui bahwa perangkat lunak tertulis tidak mendukung keperluan pemakai ini. Paket perangkat lunak komersial berfungsi lebih baik/paling cocok diterapkan apabila terdapat penentuan input/output baku yang ditetapkan untuk suatu sistem. JOCS adalah sistem pembiayaan, yang merupakan sistem akunting yang sangat umum, namun end user di Peerless, Inc. menginginkan sistem unik atau yang khas; yaitu sistem yang mendukung beberapa input/output yang telah ditetapkan, dan juga akan mendukung end user yang menentukan data dan informasinya sendiri di waktu yang akan datang. Maka Jake memutuskan untuk mengembangkan perangkat lunak untuk JOCS di dalam organisasi, dari pada membeli aplikasi dari vendor (penjual) perangkat lunak.

Keputusan ini membuatnya khawatir. Jake telah banyak mengembangkan aplikasi terkomputerisasi selama sepuluh tahun terakhir, dan ia menyadari masalah yang mungkin muncul ketika mengembangkan perangkat lunak. Ia mengetahui bahwa biaya dan waktu yang dibutuhkan menjadi masalah yang umum; ia mengetahui bahwa perangkat lunak seringkali tidak bisa hidup dalam jangka waktu sangat lama sesuai harapan pemakai yang tidak wajar; dan ia mengetahui bahwa kerusakan dapat terjadi terhadap paket perangkat lunak yang bahkan telah teruji secara sangat baik. Ia juga mengetahui bahwa beberapa orang yang melakukan riset mengenai perekayasaan perangkat lunak mempunyai gagasan untuk menciptakan paket perangkat lunak awal (pendahuluan), dan perangkat lunak awal ini hanya akan dibuang begitu saja dan mulai menciptakan lagi sampai paket yang benar-benar tepat bisa dihasilkan. Jake memasuki tahap pengembangan sistem ini dengan kesadaran penuh akan kelemahan atau masalah-masalah potensial tersebut.

Jake berpikir bahwa penggunaan keterampilan manajerial yang baik selama pengembangan perangkat lunak akan membantunya menghindari sebagian besar masalah-masalah umum tersebut. "Mungkin koordinasi dan manajemen yang baik akan menghilangkan kebutuhan akan penciptaan JOCS yang 'sia-sia' (yang tidak sempurna lalu dibuang begitu saja)," pikir Jake. "Kita telah mempunyai tim SWAT yang tangguh. Saya pikir kita telah melakukan pekerjaan rancangan dan analisis sistem dengan baik, dan kita mempunyai kesempatan menentukan lingkungan pengembangan sendiri." Sebelum ia menyampaikan tugas kepada tim SWAT atau memulai merancang algoritma atau kode meskipun satu baris saja pada paket itu, Jake lebih dulu mengembangkan atau membuat pendekatan manajerial untuk mengendalikan proyek tersebut.

Manajemen biasanya terdiri atas empat fungsi: perencanaan, pengontrolan, pengorganisasian, dan kepemimpinan. Jake menghasilkan pendekatan manajerial yang terdiri atas segi-segi masalah berikut ini, yang digabungkan ke setiap fungsi manajerial itu:

1. Perencanaan Jake membuat anggaran estimasi proyek. Jake menghasilkan estimasi awal, baik waktu dan biayanya, untuk setiap bagian proyek. Anggaran ini akan digunakan untuk menilai kemajuan dari setiap fase proyek itu.
2. Pengorganisasian Jake menghasilkan diagram Gantt awal untuk mengkonsolidasikan komponen-komponen proyek dan untuk memotret atau mengabadikan tonggak yang penting. Jake juga menetapkan struktur organisasional yang akan digunakan untuk para pekerja yang ditugaskan pada proyek itu.
3. Pengendalian Jake memilih suatu metode untuk mengukur kemajuan proyek secara akurat dan berkelanjutan, termasuk cara untuk mengevaluasi produktivitas peserta/pekerja proyek individual. Jake juga memilih set periode waktu yang akan digunakan untuk membandingkan hasil pengembangan perangkat lunak yang sebenarnya dengan yang dianggarkan guna memastikan bahwa proyek berjalan sesuai dengan rencana awal.
4. Kepemimpinan Jake memilih jenis komunikasi untuk para peserta/pekerja proyek. Jake menciptakan rencana yang digunakan untuk menginformasikan keputusan penting kepada para personel, mengembangkan atau membuat cara guna memperbaiki komunikasi antarpersonal, dan menjadwal pelatihan yang perlu.

Setelah menetapkan pendekatan manajerialnya, langkah Jake berikutnya adalah memastikan bahwa tim SWAT yang ada saat itu memadai untuk menangani pengembangan perangkat lunak. Mengembangkan perangkat lunak merupakan pekerjaan padat karya. Orang-orang yang bekerja pada proyek pengembangan perangkat lunak merupakan elemen paling penting dari proyek itu. Untungnya, tim SWAT yang ada saat itu telah mampu berkomunikasi dengan sangat baik selama fase-fase sebelum siklus hidup pengembangan sistem. Dari empat orang yang bekerja pada proyek tersebut, Christine dan Tom memiliki latar belakang yang kuat dalam hal rancangan dan pengembangan perangkat lunak. Cory baru saja lulus dari sekolah tinggi, namun ia mempunyai pengetahuan kerja yang baik dalam hal sistem manajemen database. Ia pernah membuat/mengembangkan program ketika masih kuliah. Namun, Carla tidak mempunyai pengalaman apapun sebelumnya dalam mengembangkan perangkat lunak. Dia dipilih untuk masuk tim SWAT karena ia mengetahui atau memahami keperluan sistem dari sudut pandang end-user. Karena Carla hanya "pinjaman" dari departemen (bagian) keuangan kepada proyek JOCS, maka ini waktu yang tepat bagiinya untuk kembali ke tugas-tugas biasanya di departemennya. Namun Jake bersik

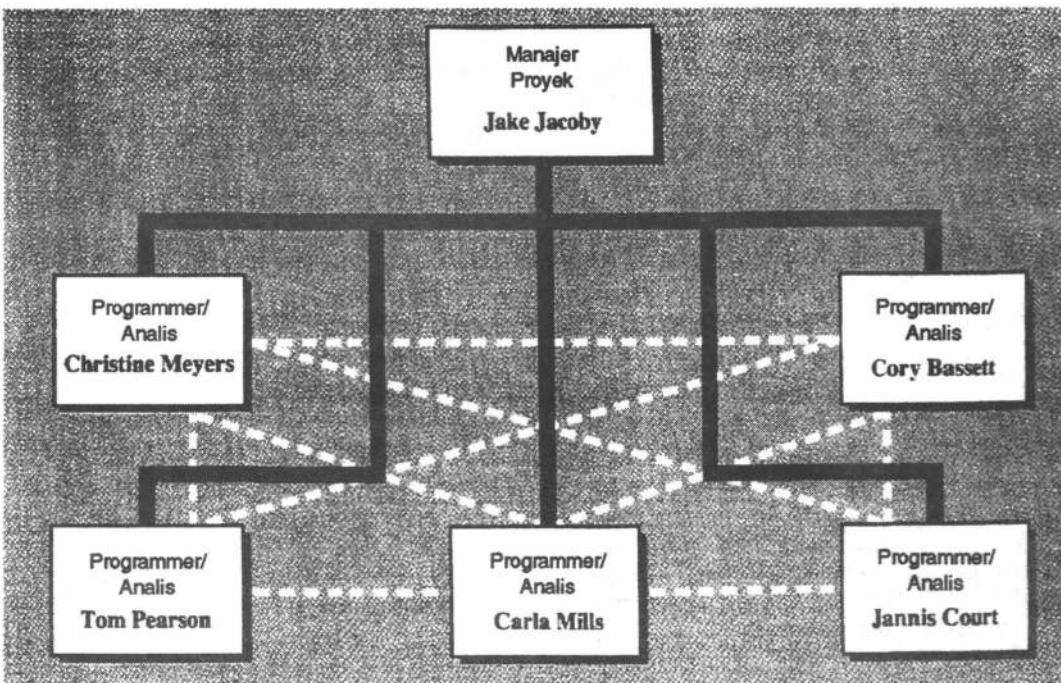
keras mempertahankannya. Carla paham apa yang diinginkan end user dari sistem. Dia bisa menilai apakah program benar-benar user friendly ataukah tidak. Programmer cenderung punya pertimbangan atau opini mengenai daya guna perangkat lunak yang berbeda dari para pemakai, dan kadang-kadang programmer membuat keputusan yang tidak tepat mengenai struktur interface pemakai. Jake berfikir bahwa Carla bisa membantu tim merancang perangkat lunak yang baik. Jake menginginkannya tetap bergabung dengan tim SWAT dan membantu rancangan dan pengujian perangkat lunak.

Untuk bisa menyelesaikan proyek tepat waktu, Jake menambah lagi satu orang ke tim SWAT. Jannis Court, seorang programmer dengan pengalaman tiga tahun, ditambahkan ke tim itu untuk membantu mengembangkan perangkat lunak JOCS. Jake hanya menambah satu orang baru ke tim SWAT, karena riset dari proyek aplikasi lain telah menunjukkan bahwa jumlah orang yang banyak seringkali memperlamban proses pengembangan dikarenakan sulitnya komunikasi.

Jake telah menetapkan struktur organisasional untuk aplikasi JOCS yang menggabungkan elemen-elemen terbaik dari pendekatan tim pengembangan program dan pendekatan tim pemrograman bersama. Gambar 1.10 memberikan tinjauan tentang penempatan organisasional terhadap para personel dalam proyek tersebut. Garis-garis hitam tebal pada gambar tersebut merepresentasikan pelaporan langsung dari tanggung jawab manajemen. Garis-garis terputus-putus menunjukkan potensi komunikasi diantara para anggota tim. Jake memanaj proyek tersebut, dan setiap programmer/analisis melaporkan secara langsung kepadanya. Namun demikian, salah satu anggota tim tidak perlu berhubungan dulu dengan manajer proyek untuk berkomunikasi dengan anggota tim lainnya.

Jake berpendapat bahwa komunikasi adalah komponen paling penting dari proyek pengembangan perangkat lunak. Ia juga berfikir bahwa penting bagi semua peserta/pekerja proyek untuk saling berkomunikasi secara bebas. Namun, penggunaan komunikasi oral (lesan) kadang-kadang mengganggu tugas mereka dan mempengaruhi pekerjaan mereka. Untuk menghindari masalah ini, Jake telah menginstal sistem surat elektronik pada jaringan yang digunakan untuk pengembangan perangkat lunak JOCS.

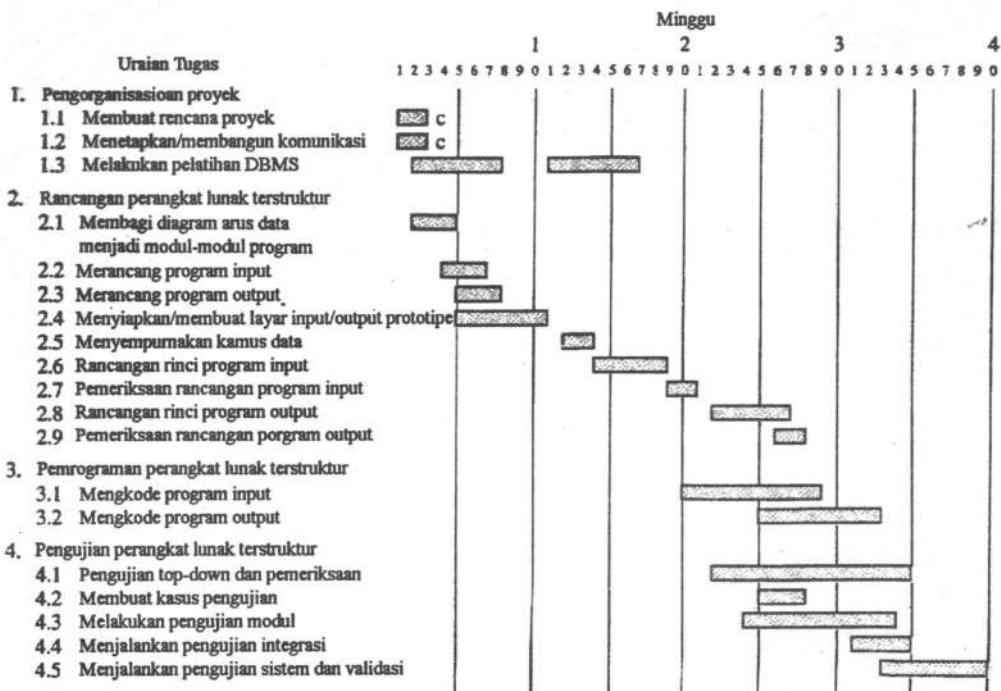
Jake membentuk/membuat rekening umum untuk para peserta proyek JOCS guna memberikan metode penyemaian/pemberian informasi mengenai usaha pengembangan. Setiap peserta dapat secara mudah mengirimkan pesan ke anggota tim lain atau ke semua anggota. Jake juga menggunakan fungsi pengkalenderan dari sistem surat elektronik untuk mengirimkan peristiwa/tanggal penting bulanan dan harian. Dengan menggunakan sistem ini, setiap peserta selalu menyadari atau mengetahui deadline saat itu; setiap peserta mempunyai pemahaman terhadap keseluruhan sistem, bukan hanya hanya satu program spesifik dalam sistem tersebut. Selain komunikasi elektronik, Jake menetapkan meeting dua mingguan, dengan lama waktu yang ditetapkan sebelumnya, guna memberikan kesempatan berkomunikasi antarkelompok.



Gambar 1.10 Struktur organisasional tim SWAT JOCS.

Jake ingin memberikan/menciptakan lingkungan manajerial yang mendorong setiap peserta untuk mengkomunikasikan gagasan dan kebutuhannya, sekaligus memberikan struktur dan kendali untuk proyek itu. Untuk mencapai tujuan ini, ia bekerja dengan timnya untuk menyelesaikan langkah berikutnya dalam pengelolaan proyek ini: yaitu mengestimasi jumlah waktu yang dibutuhkan untuk menyelesaikan proyek JOCS. Estimasi ini dipecah/dibagi dan digambarkan secara grafis dalam diagram Ganit, seperti yang terlihat pada Gambar 1.11. Dalam estimasi itu disertakan pula waktu yang

diperlukan untuk melatih para peserta proyek dalam masalah teknologi sistem manajemen database. Beberapa tugas, seperti pembuatan rencana, dirampungkan selagi diagram dipersiapkan. Tugas-tugas itu dirancang secara lengkap oleh "C" pada diagram Gantt.

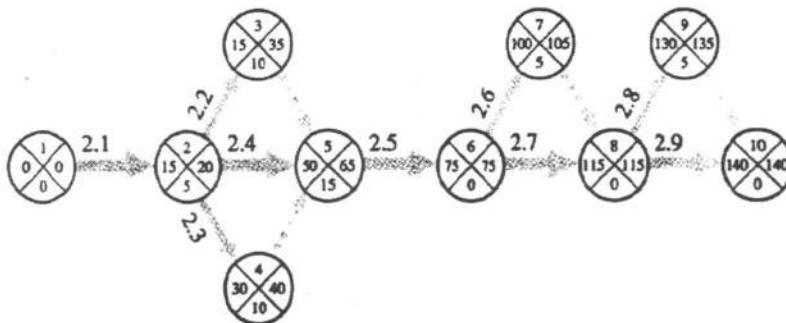


Gambar 1.11 Diagram Gantt untuk pengembangan JOCS.

Diagram Gantt digunakan untuk menandai keseluruhan kemajuan proyek JOCS tersebut. Untuk menelusuri/mencari tugas individual, Jake menggunakan jaringan PERT. Gambar 1.12 adalah contoh jaringan PERT untuk tugas-tugas yang terlibat dalam rancangan perangkat lunak terstruktur. Setiap tugas didaftarkan pada diagram Gantt, seperti terlihat pada Gambar 1.11, dan kemudian digambarkan dalam format lintasan penting, seperti terlihat pada Tugas 2.2 (Merancang program input) dan 2.3 (Merancang program output) ditunjukkan pada jaringan PERT sebagai tugas-tugas di luar lintasan penting yang dapat dilakukan secara paralel. Tujuan tugas-tugas ini adalah Tugas 2.4 Meyiapkan prototipe layar input/output).

Perlu dicatat bahwa Tugas 2.2 dijadwalkan selasai pada waktu paling awal pada Day 25 (hari ke-25) dari set tugas-tugas tersebut, dan waktu paling lambat harus pada hari ke-35. Maka Tugas 2.4 perlu diselesaikan pada waktu paling awal, Day 50m atau waktu paling lambat, Day 65. Tim SWAT menggunakan jaringan PERT baik untuk mengorganisasi maupun untuk mengontrol kemajuan proyek.

Sebagai tugas manajerialnya yang berkelanjutan, Jake harus mengontrol produktivitas timnya dan mengukur kualitas output mereka. Untuk melakukan tugas ini, setiap anggota tim mengetahui bahwa ia bertanggung jawab atas waktu dan kinerja, dan manajemen juga bertanggung jawab atas estimasi dan kualitas sistem. Para anggota tim SWAT merasa puas dengan estimasi tersebut, karena deadline telah ditetapkan oleh orang-orang yang memahami kompleksitas proyek itu. Orang-orang ini adalah mereka yang merancang keperluan sistem tersebut.



Gambar 1.12 Jaringan PERT untuk rancangan perangkat lunak terstruktur JOCS.

Tim SWAT telah memilih untuk menggunakan analisis titik fungsi untuk meng-evaluasi produktivitas setiap anggota proyek. Disini, setiap anggota harus mengetahui waktu yang dibutuhkannya pada setiap segi usaha pengembangan perangkat lunak. Gambar 1.13 adalah kopi dari time sheet entry-screen (form jadwal waktu) yang akan diisi setiap hari oleh setiap peserta proyek. Dalam komunikasi antartingkat terdapat sistem penelusuran waktu untuk para staf pemrogramannya.

Sekarang Jake telah siap dengan pendekatan manajemennya. Ia menyusun tim orang-orang tangguh untuk mengembangkan perangkat lunak; ia telah menciptakan rencana awal waktu yang diperlukan untuk menyelesaikan proyek tersebut; ia telah mengorganisir dan mempublikasikan rencana dengan menggunakan diagram Grantt dan jaringan PERT; ia telah memutuskan suatu metode untuk mengukur produktivitas dan kendali kualitas; dan terakhir, ia telah menetapkan atau membangun suatu lingkungan melalui meeting, surat elektronik, dan pelatihan guna memotivasi dan mengarahkan tim. Ia tinggal memberikan perintah, mengharapkan hasil terbaik, dan mengembangkan aplikasi JOCS.

Programmer name _____		Week starting _____
Date _____ / _____	Activity code _____	Number of hours _____
Date _____ / _____	Activity code _____	Number of hours _____
Date _____ / _____	Activity code _____	Number of hours _____
Date _____ / _____	Activity code _____	Number of hours _____
Date _____ / _____	Activity code _____	Number of hours _____
Date _____ / _____	Activity code _____	Number of hours _____
Date _____ / _____	Activity code _____	Number of hours _____
Date _____ / _____	Activity code _____	Number of hours _____
Activity codes: 01 Training, 02 Design, 03 Coding, 04 Testing, 05 Typing, 06 Meeting, 07 Walkthrough		

Gambar 1.13 Layar entri data pada sistem penelusuran waktu.

BACAAN YANG DISARANKAN

- Abdel-Hamid, Tarek K., and Stuart E. Madnick. "On the Portability of Quantitative Estimation Models." *Information & Management*, 1987.
- Arthur, L. J, *Measuring Programmer Productivity and Software Quality*. New York: Wiley-Interscience, 1985.
- Benbasat. Izak, and Iris Vessey. "Programmer and Analyst Time/Cost Estimation." *MIS Quarterly*. June 1980
- Best, Laurence J. "Building Software Skyscrapers." *Datamation*, March 15, 1990.
- Boehm, B. W. *Software Engineering Economics*, Englewood Cliffs. N.J.: Prentice-Hall, 1981.
- Brooks, Frederick P., Jr. *The Mythical Man-Month*. Reading, Mass.: Addison-Wesley, 1975.
- Carlyle, Ralph Emmett. "Fighting Corporate Amnesia." *Datamation*, February I, 1989.
- Fairley, Richard E. *Software Engineering Concepts*, New York: McGraw-Hill, 1985.
- Fourteenth NASA Goddard Software Engineering Workshop.
- Gray, Paul, William R, King, Ephraim R. McLean, and Hugh J. Watson. *Management of Information Systems*. Chicago: Dryden Press, 1989.
- Gross, Neil. "Now Software Isn't Safe from Japan." *Business Week*, February 11, 1991.
- Inman, W. H. *Information Engineering for the Practitioner: Putting Theory into Practice*. Englewood Cliffs, N.J.: Yourdon Press, A Prentice-Hall Company, 1988.
- Jeffery, D. R. "A Software Development Productivity Model for MIS Environments." *Journal of Systems and Software*, 1981.
- Jones, T. Capers, *Programming Productivity*, New York: McCraw-Hill, 1986.
- Kaner, Cem. *Testing Computer Software*. Blue Ridge Summit, Pa.: TAB Professional and Reference Books. 1988.
- Smith. L. Murphy. "Using the Microcomputer for Project Management." *Journal of Accounting and EDP*, Summer 1989.

2

PERANCANGAN PERANGKAT LUNAK

APA YANG AKAN ANDA PELAJARI DALAM BAB INI?

Setelah mempelajari bab ini, anda diharapkan dapat:

- Menjelaskan alasan pelaksanaan tahap rancangan perangkat lunak.
- Menjabarkan pendekatan rancangan perangkat lunak terstruktur.
- Menjabarkan pendekatan rancangan perangkat lunak berorientasi-objek.
- Menerangkan cara melaksanakan penelusuran rancangan perangkat lunak.

PENDAHULUAN

Bab ini mengemukakan rancangan perangkat lunak (lihat Gambar 2.1). Rancangan perangkat lunak dilakukan untuk memastikan bahwa kita mencapai transisi (peralihan) dari rancangan sistem ke rancangan perangkat lunak. Untuk memastikan bahwa transisi ini telah dicapai, dilakukanlah penelusuran rancangan perangkat lunak.

Bab-bab sebelumnya pada buku ini terutama diarahkan untuk membahas rancangan sistem, termasuk platform teknologinya. Output dan input dirancang secara rinci. Proses-proses telah dirancang dengan menggunakan DFD, STD, dan mungkin, perangkat permodelan yang lain. Jika diperlukan, model-model dan persamaan-persamaan prosedural untuk pemrosesan dikembangkan. Telah diciptakan pula kamus data dan model data logik. Kendali yang harus dilekatkan dalam perangkat lunak telah ditetapkan. Dan, jika diperlukan, teknologi informasi dievaluasi dan didapatkan guna mendukung rancangan sistem. Dari spesifikasi rancangan sistem, khususnya yang direpresentasikan oleh DFD dan STD, para perancang perangkat lunak akan mendapatkan visi yang jelas mengenai apa yang harus mereka kerjakan untuk mengkonversi spesifikasi ini ke rancangan perangkat lunak yang mematuhi spesifikasi itu.

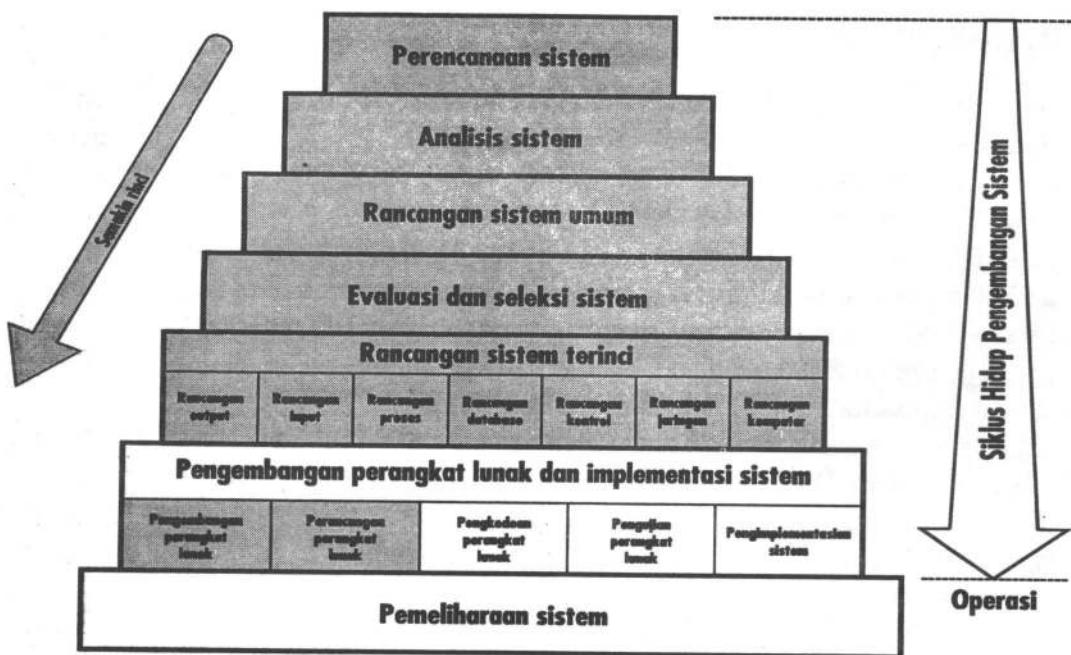
Bab ini mengemukakan dua pendekatan rancangan perangkat lunak yang banyak digunakan:

- ◆ Rancangan perangkat lunak terstruktur
- ◆ Rancangan perangkat lunak berorientasi-obyek

Mereka tidak dibatasi oleh bahasa pemrograman komputer tertentu. Kedua pendekatan tersebut menjanjikan pengendalian kualitas, kemudahan pemeliharaan, kemampuan penggunaan, kemampuan penggunaan kembali, reliabilitas, dan kemampuan perluasan/pengembangan. Sekarang, banyak eksekutif mencoba memutuskan pendekatan mana yang akan digunakan, atau apakah mengimplementasikan kedua pendekatan tersebut. Kasus berikut menjelaskan situasi yang demikian ini.

Penentuan Pendekatan Rancangan Perangkat lunak di Tranquil Industries

Sylvia Porter adalah CIO pada Tranquil Industries. Ia minta melakukan meeting dengan Jim Thompson, pimpinan proyek sistem. Sylvia memulai meeting dengan bertanya, "Permainan apa lagi dengan yang bernama pemrograman dan rancangan berorientasi-obyek ini? Apakah ini berarti kita akan mengubah usaha pengembangan swakelola dari



Gambar 2.1 Tahap SDLC dan bab-bab yang terkait dalam buku ini. Dalam Bab 2 kita membahas rancangan perangkat lunak dari dua pendekatan pokok: rancangan terstruktur dan rancangan berorientasi-obyek.

pendekatan terstruktur konvensional kita yang telah berfungsi dengan baik ke pendekatan berorientasi-obyek?"

"Jawaban atas pertanyaan anda yang pertama adalah bahwa ia bukanlah main-main; pendekatan berorientasi-obyek adalah riil," jawab Jim.

"Sejauh saya bisa menanggapi pertanyaan anda yang kedua, saya belum siap membuang semua peralatan berorientasi-nonobyek kita," kata Jim, "namun saya siap memberikan pendekatan berorientasi-obyek sebagai uji coba, mungkin pada sistem informasi eksekutif yang baru."

"Coba terangkan lagi mengenai perbedaan pokok antara pendekatan terstruktur dan pendekatan berorientasi-obyek", kata Sylvia.

"Baiklah, ada beberapa perbedaan teknis yang mendasar, seperti kelas, peringkasan, pewarisan, dan sebagainya. Dan, tentunya, keuntungan yang menjanjikan adalah kemampuan penggunaan kembali dan kemampuan perluasan/pengembangan. Namun anda harus punya perpustakaan kelas untuk memanfaatkan daya guna penggunaan kembali (reuseability), dari ..."

"Coba sebentar Jim," Sylvia menginterupsi, "saya tidak tahu dengan sebagian istilah-istilah itu. Yang ingin saya tahu adalah perbedaan konseptual antara kedua pendekatan itu."

"Perbedaan konseptualnya adalah cara orang-orang sistem, yakni para programer dan perancang sistem — masalah pendekatannya. Pendekatan terstruktur tradisional kita, seperti yang anda ketahui dengan baik, kurang lebih dibagi ke dalam dua kam: orang-orang berorientasi-proses (juga disebut berorientasi-prosedur dan berorientasi algoritma) dan orang-orang (lama) yang berorientasi-data. Kam (kelompok) pertama berfikir mengenai proses untuk mencapai tugas dan kemudian membangun struktur data untuk proses-proses yang akan digunakan. Kelompok kedua berfikir tentang bagaimana cara menstruktur (atau memodel) data dan membangun proses menurut struktur itu: Maka kita mempunyai sentra proses dan sentra data.

Dengan pendekatan berorientasi-obyek, struktur data dan prosedur (yang juga disebut operasi) dipak bersama sebagai obyek. Obyek-obyek tersebut memproses daftar metode; yakni, hal-hal yang dapat mereka lakukan."

"Itu baru bisa saya mengerti", kata Sylvia. "Saya pernah membaca beberapa artikel mengenai pendekatan berorientasi-obyek, namun saya tak punya waktu untuk memahami rinciannya lebih dalam lagi."

"Anda tahu bahasa prosedural konvensional, seperti COBOL atau FORTRAN, katakanlah, paling tidak istilah umumnya, 'Tulislah nilai bilangan ini pada printer.' Programmer berorientasi-obyek akan mengatakan, 'Tulislah sendiri'. Dengan cara konseptual, programmer berorientasi obyek bekerja dengan hewan terlatih; yakni obyek sebagai hewan terlatih. Pemakai mengirimkan suatu pesan ke suatu obyek untuk menyampaikan metodenya, seperti 'cetaklah statemen pendapatannya.'"

"Bagaimana tentang pembagian para pekerja?" tanya Sylvia. "Dapatkah kita menerapkan konsep modularitas ke pendekatan berorientasi obyek dan meminta para programmer bekerja pada obyek-obyek yang berlainan secara serentak?"

"Oh, itu pasti," jawab Jim. "Modularitas berlaku untuk pendekatan berorientasi obyek sebagaimana ia bisa diterapkan pada pendekatan terstruktur."

"Sebelumnya anda menyebutkan pewarisan (inheritance). Apa arti istilah ini?"

"Inheritance memberi cara kepada programmer untuk memulai dengan obyek yang telah ada yang mendekati (hampir sama dengan) apa yang diperlukan dan untuk menggunakan atribut data dan operasi dalam aplikasi lain. Juga, jika diperlukan, kode dan atribut data bisa ditambah lagi. Pewarisan adalah karakteristik object-oriented (berorientasi obyek) kunci yang memungkinkan dicapainya reuseability (penggunaan kembali). Namun demikian, tidak semua bahasa pemrograman object-oriented mempunyai fasilitas pewarisan."

"Saya suka gagasan reuseability ini," kata Sylvia. Saya selalu berfikir bahwa para programmer menghabiskan banyak waktu untuk menulis dan menulis lagi jenis program yang sama, sebab mereka tidak bisa menggunakan kembali pekerjaan mereka sebelumnya atau memanfaatkan pekerjaan programmer lain."

"Itu benar," kata Jim, "namun rancangan berorientasi obyek tidak menjamin selalu dicapainya reuseability (kemampuan guna ulang), dan pendekatan terstruktur tidak juga pasti tidak punya peluang untuk itu. Kita bisa merancang obyek-obyek yang sulit digunakan ulang. Dimungkinkan juga merancang perangkat lunak dengan menggunakan pendekatan terstruktur yang memproses modul-modul yang bisa digunakan ulang."

"Jadi, dari apa yang anda katakan, saya tentunya tidak percaya pendekatan berorientasi obyek adalah hanya suatu iseng atau main-main. Bagi saya, dunia sistem nampaknya benar-benar sedang mengarah untuk penerapan pendekatan ini. Bagaimana menurut anda, apakah kita juga akan menerapkannya?"

"Saya pikir kita harus mencobanya pada rancangan EIS baru kita," kata Jim. "Pendekatan berorientasi obyek harus bekerja baik disini dan memberi kita dengan lingkungan uji yang baik. Jika ia bekerja seperti yang kita kehendaki, saya juga akan mengusulkan untuk dicoba pada sistem entri-pesanan berbasis LAN yang sedang kita rencanakan saat ini."

"Saya setuju hal itu," kata Sylvia secara antusias. "Mari kita kerjakan. Saya akan memberimu semua dukungan yang anda perlukan."

"Ini akan memerlukan beberapa riset, sebab saya tidak paham betul pendekatan berorientasi obyek untuk saat ini. Juga, saya ingin memilih alat (peralatan) modeling yang bisa bekerja untuk tahap rancangan perangkat lunak dan juga memilih bahasa pemrograman berorientasi obyek. Menurut penglihatan saya, Julie dan Dana, dua programmer senior kita, telah berpengalaman dengan banyak bahasa pemrograman berorientasi obyek (OOP). Maka sebenarnya kita telah mempunyai permulaan yang baik."

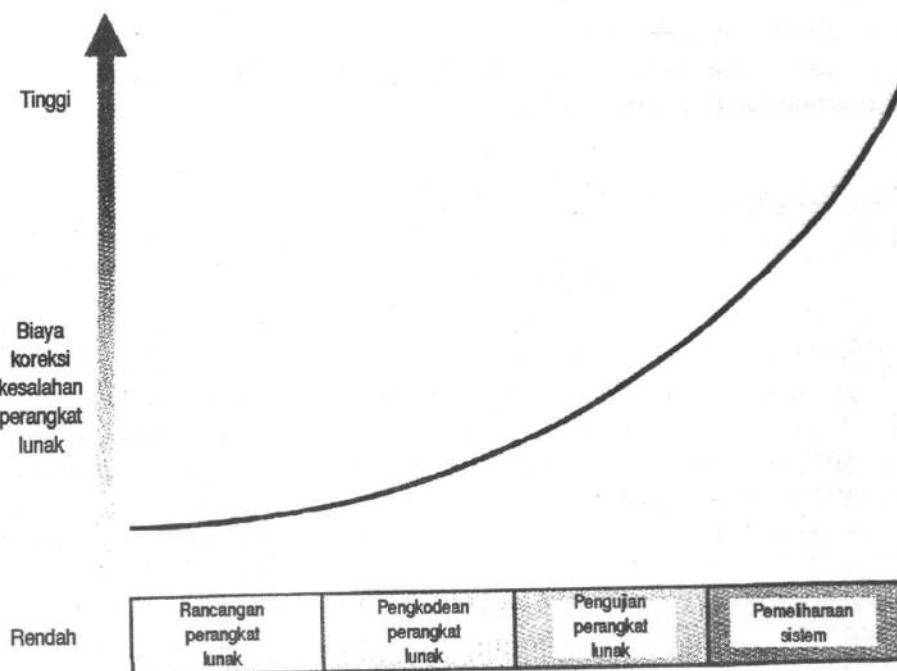
"Bagus!" seru Sylvia. "Kirimi saya terus mengenai kemajuan anda."

Jim segera mulai mempelajari pendekatan berorientasi obyek secara lebih mendalam lagi dan menyiapkan aplikasi (usulan)-nya. Ia sangat percaya bahwa EIS baru akan memberinya sistem yang rapi dan bagus, sehingga ia bisa memfokuskan pada dan memberikan pendekatan berorientasi obyek sebagai uji coba yang baik.

ALASAN MELAKSANAKAN TAHAP RANCANGAN PERANGKAT LUNAK

Orang-orang yang tidak pengalaman dengan sistem informasi seringkali menganggap pengembangan perangkat lunak hanyalah sebagai penulisan kode dalam beberapa bahasa seperti C, COBOL, FORTRAN, RPG, Pascal, atau C++. Namun semakin banyak waktu dan usaha dicurahkan untuk rancangan perangkat lunak, maka akan semakin baik kualitas perangkat lunak itu, semakin sedikit masalah yang akan terjadi di tahap-tahap kemudian, dan akan semakin sedikit sumber daya yang diperlukan untuk siklus hidup total sistem tersebut.

Beberapa orang mungkin mengatakan bahwa terlalu banyak waktu yang dihabiskan untuk rancangan perangkat lunak. Mereka bertanya: "Mengapa kita tidak saja langsung memulai pengkodean dan mengapa khawatir dalam pengoreksian kesalahan nantinya?" Tujuannya bukanlah hanya membuat program perangkat lunak bekerja, namun mem-



Gambar 2.2 Biaya relatif pengoreksian kesalahan perangkat lunak selama pengembangan dan pemeliharaan perangkat lunak.

buatnya bekerja secara benar. Studi-studi menunjukkan bahwa biaya pengoreksian kesalahan jauh lebih tinggi setelah program tersebut diimplementasikan dari pada pengoreksian selama tahap rancangan dan pengkodean. Gambar 2.2 menunjukkan bagaimana biaya pengoreksian kesalahan atau kelemahan rancangan meningkat secara geometris selama pengembangan perangkat lunak. Estimasi dari beberapa ahli, termasuk Departemen Pertahanan, menunjukkan bahwa dibutuhkan biaya antara \$30 dan \$50 untuk menulis satu baris kode yang bisa dijalankan (LOEC) dan sekitar \$4000 per LOEC untuk memeliharanya. Para profesional sistem informasi harus menyadari bahwa perusahaan tidak dapat terus beroperasi secara kompetitif dengan adanya beban biaya pemrograman dan pemeliharaan. Lebih dari itu, biaya hilangnya kepercayaan pemakai hampir tidak bisa diukur, namun mereka yakin bahwa berbagai bentuk kesalahan pasti muncul dan perlu pengoreksian selama pengoperasian sistem baru.

Mengapa kurva biaya pada Gambar 2.2 berujud seperti itu; hal itu karena adanya tahap sebelumnya yang berkelanjutan yang dikerjakan sebelumnya. Jelasnya, pencarian kesalahan selama pemeliharaan selalu menyebabkan proses pengoreksian lebih lama dan berbiaya mahal. Jika, misalnya, kesalahan keperluan pemakai ditemukan selama pemeliharaan, maka keseluruhan proses SWDLC mungkin harus diulang; yakni, melakukan rancangan ulang, pengkodean ulang, pengujian ulang untuk memperbaiki sistem tersebut.

Kisah Code King

Code King telah melakukan pengkodean bertahun-tahun. Di meja Code King terdapat tumpukan pesan pengembalian dari para pemakai. ("Mengapa susah-susah meminta mereka mengembalikannya?" ia menggerutu. "Mereka tidak pernah tahu apa yang mereka inginkan. Lagi pula, saya lebih mengetahui bisnis dari pada mereka!" Code King mempunyai stiker bumper pada mobilnya yang berbunyi, "Kita suka mengkode dan ini kelihaian dengan jelas." Ia pikir bahwa analisis adalah paralisis, dan bahwa rancangan adalah menyisihkan tempat sebelum kode. Para pemakai harus dilepaskan dan dokumentasi harus dicabut.

Apa yang kita buat selama bertahun-tahun adalah menciptakan hubungan simbiosis yang sempurna untuk Code King. Kita menghargai pemadaman dan kemampuan untuk membuat perbaikan/pemasangan demi perbaikan secara cepat. Kita telah menjadi kurang sensitif terhadap kenyataan bahwa sistem tidak terdokumentasi, dan bahwa kode kita tidak terstruktur. Tak apa bila pengujian terdiri dari pengujian-pengujian unit asalkan ada waktu cukup. Jika tak ada waktu untuk itu, "muatkan dan mulailah"; disamping itu, kita bisa memperbaikinya kemudian.

Namun apa yang akan terjadi apabila alat CASE ditempatkan pada meja Code King? Dapatkah kita mengharapkan terjadinya transformasi radikal dari kecintaan permainan ke kecintaan akan sistem yang efektif? Sekarang saya memutuskan untuk pergi ke Techtown, dimana Code King tinggal, untuk menemukan jawaban atas pertanyaan-pertanyaan tersebut.

Ketika saya tiba, Code King menjulurkan kakinya pada keyboard terminal dumb. Saya bertanya padanya, "Bagaimana perasaan anda dalam mengerjakan rancangan dengan menggunakan alat CASE yang baru-baru ini dibeli oleh manajemen?"

Code King cepat-cepat menurunkan kakinya dari meja dan mengusap mukanya yang nampak pucat. Untungnya, kesunyian di ruang itu ternormalisasi dengan adanya dering telepon. Saya menduga telepon itu adalah dari pemakai yang jengkel yang mungkin meminta perbaikan cepat. Selain itu, mungkin telepon itu dari temannya, programmer sistem, yang mengolok-olok Code King bahwa sejarahnya sudah tamat.

"Sebentar lagi, anda akan menjadi artis pesolek yang sebenarnya, yang menggantikan bar semua gambar-gambar cantik itu" saya mendengarnya secara samar-samar. King mengernyitkan dahi. Kawannya mengatakan bahwa tak ada alat CASE-nya yang bererasi (Tuhan tidak menghendaki). Saya selalu berpikir bahwa akan terjadi ikatan antara hubungan khusus diantara para programmer, namun ternyata mereka saling berantem. Apakah tidak ada lagi kesucian?

Code King menutup telepon itu, dan kami kembali bercakap-cakap. Ia mengatakan bahwa "ia" akan tiba Senin yang akan datang bersama PC XT (monokrom). Ia akan dijalankan pada PC XT ini. Terminal dumb akan dibongkar sehingga pada akhir pekan tak ada orang yang menghubunginya.

"Bagaimana mengenai pelatiannya?" tanya saya. Saya mengetahui bahwa standart industri untuk pelatihan alat CASE minimum 20 hari. Code King secara keras mengatakan bahwa tak ada pelatihan untuk saat ini. Manajemen memutuskan untuk menunda pelatihan sampai setiap orang mempunyai waktu untuk memperbaiki-nya".

Pendekatan drop-it-in untuk CASE adalah paradigma populer sekarang ini; ini juga sangat dipengaruhi oleh tenaga penjual alat CASE dalam masalah komisi.

Saya bertanya kepada Code King bagaimana ia akan menggunakan alat analisis dan rancangan ini dalam pekerjaannya sehari-hari. Ia menjawab, "Perkiraan/gagasan anda sejalan dengan gagasan saya. Namun yang benar-benar menyenangkan saya adalah pemberitaan dalam *PC Daily* mengenai PC dengan 586 chip. Dapatkah anda bayangkan, C-quad-plus akan menyusun program 3-Meg dalam sekitar 1,05 nanosecond?" "Bagus," pikir saya, "tunggu sampai Grace Hopper merasa enak dan tidak marah."

King melanjutkan perkataannya, "Ini akan sangat hebat. Oh, akan tetapi saya taruhan anda pasti mau membahas diagram lingkaran dan diagram kotak itu. Mari kita lihat." Ia mengambil buku ungu dari lacinya. (Tom DeMarco mungkin sakit hati karena

bukunya penuh debu; Saya kira demikian. Bagi saya, buku "ungu" selalu merupakan buku penuntun analisis). Seekor kecoa jatuh dari buku itu. Saya bayangkan lamanya buku itu tidak dibuka sejak Code King mempelajari buku tentang rancangan terstruktur.

King melanjutkan perkataannya, "Ini akan sangat hebat. Oh, akan tetapi saya seharusnya anda pasti mau membahas diagram lingkaran dan diagram kotak itu. Mari kita lihat." Ia mengambil buku ungu dari lacinya. (Tom DeMarco mungkin sakit hati karena bukunya penuh debu; Saya kira demikian. Bagi saya, buku "ungu" selalu merupakan buku penuntun analisis). Seekor kecoa jatuh dari buku itu. Saya bayangkan lamanya buku itu tidak dibuka sejak Code King mempelajari buku tentang rancangan terstruktur.

MENGGUNAKAN RANCANGAN PERANGKAT LUNAK TERSTRUKTUR

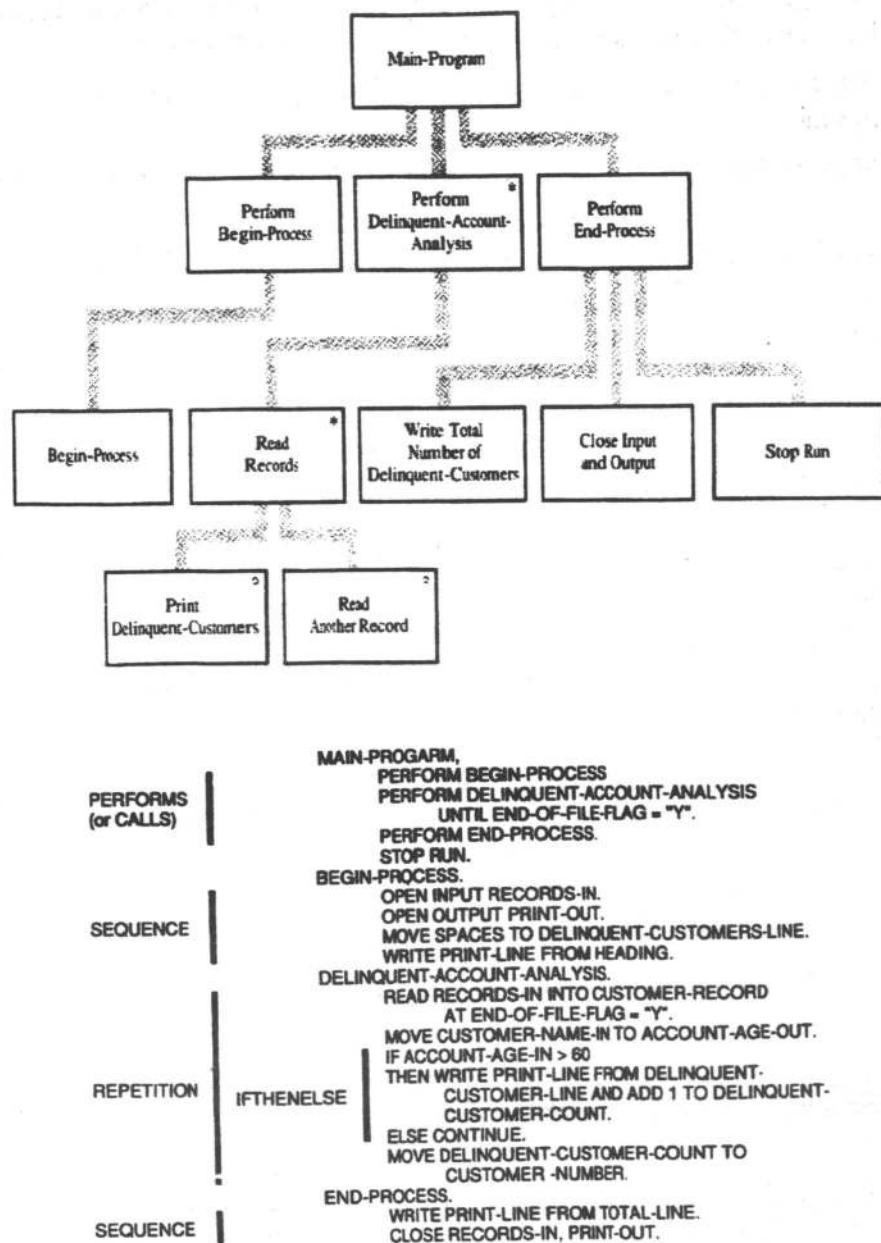
Pada waktu proyek sistem siap untuk Rancangan Perangkat lunak Terstruktur, kamus data dan model data logik dikembangkan. Diagram entitas relasi (ERD) menjelaskan hubungan antara penyimpanan data dari DFD. Jika diperlukan, STD akan memodel cara pelaksanaan berdasarkan waktu. STD ini menjelaskan pengaturan waktu eksekusi dan akses data yang dipicu oleh event.

Berbagai perangkat pemodelan, seperti DFD, digunakan untuk mendeskripsikan fungsi sistem. Dalam tahap rancangan perangkat lunak terstruktur, DFD tingkatan paling bawah dikonversi ke bagan struktur yang mempunyai deskripsi kode bahasa pemrograman. Untuk mencapai titik ini, DFD terus menerus dibagi menjadi sub diagram sampai tinggal proses-proses kecil yang relatif mudah diimplementasikan. Spesifikasi fungsional yang akan diubah (konversi) ke dalam kode bahasa pemrograman bisa diekspresikan (dinyatakan) dengan tabel keputusan, pohon keputusan, Bahasa Inggris terstruktur (atau pseudocode), persamaan, atau teknik lain (misalnya, narasi terstruktur Jackson).

Karakteristik Rancangan

Program terstruktur mempunyai karakteristik rancangan berikut ini (lihat Gambar 2.3):

- ◆ Modul-modul disusun secara hirarkis. Bagan struktur, diagram Jackson, dan diagram Warnier-Orr menyusun modul-modul secara hirarkis.
- ◆ Menggunakan logika CALL-based atau PERFORM-based.



Gambar 2.3 Rancangan program terstruktur.

- ◆ Menggunakan alur kendali (control flow) dan rancangan top-to-bottom (atas ke bawah) dan pengkodean top-to-bottom atau bottom-to-top.
- ◆ Merancang repetisi atau loop yang ketat, yang hanya menjangkau satu modul.
- ◆ Menerapkan konsepsi kendali standar untuk urutan, seleksi, dan repetisi.

Diagram Jackson (bagan struktur yang hanya mudah digunakan) untuk menggambarkan rancangan hirarkis modul-modul yang akan dikode dalam divisi prosedur program COBOL. Identifikasi, lingkungan, dan divisi data tidak ditunjukkan. Modul eksekutifnya adalah MAIN-PROGRAM, yang memanggil modul-modul lain dari divisi prosedur. BEGIN-PROCESS mencetak heading. DELINQUENT-ACCOUNT membaca data pelanggan, menentukan apakah jumlah yang dihutang kedaluwarsa dari 60 hari, dan mencetak nama-nama pelanggan yang lalai membayar rekeningnya. Repetisi (atau loop) tersebut dikendalikan oleh flag end-of-file (EOF). END-PROCESS mencetak jumlah pelanggan yang lalai membayar. STOP RUN dalam MAIN-PROGRAM mengakhiri program tersebut.

Membagi Rancangan ke dalam Modul

MODULARITAS adalah pembagian rancangan perangkat lunak ke dalam komponen-komponen individual yang disebut modul atau obyek, untuk mengurangi kompleksitasnya. Tujuan pendekomposisian rancangan perangkat lunak ke dalam modul-modul adalah untuk:

- ◆ Merancang, mengkode, dan menguji modul-modul secara independen.
- ◆ Merevisi dan memelihara modul-modul secara mudah setelah mereka dikonversi ke operasi.

Sasaran teknis modularitas adalah untuk:

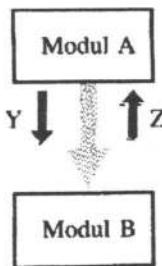
- ◆ Menyampaikan sedikit mungkin data antara modul-modul.
- ◆ Meminimisasi penggunaan data kendali
- ◆ Mengefektifkan independensi modul
- ◆ Memfokuskan modul pada fungsi tunggal

Pencapaian tujuan ini tergantung pada tingkat perangkai (coupling) antara modul-modul dan derajad kohesi di dalam modul-modul.

Coupling

Coupling mengukur derajad independensi dan interaksi antara modul-modul. Jika suatu modul bersifat independen, modul tersebut tidak dimodifikasi oleh modul lain. Interaksi berarti pertukaran dan pemodifikasian variabel antara modul-modul. Jika terjadi sedikit interaksi antara dua modul, modul tersebut bersifat sangat independen dan dianggap *loosely coupled*. Jika terjadi banyak interaksi antara modul-modul, maka modul ini dianggap sebagai saling tergantung (interdependent) dan *tightly coupled*. Jika modul merupakan *tightly coupled*, maka hampir mustahil untuk memelihara satu modul tanpa melakukan perubahan terhadap modul yang lainnya. Sebaliknya, harus ada koneksi antara modul-modul dalam program atau di tempat lain yang tidak menjadi bagian dari program itu. Namun demikian, rancangan berkualitas tinggi berarti bahwa modul adalah *loosely coupled* dan oleh karenanya mudah dipelihara dan digunakan ulang untuk aplikasi lain.

Gambar 2.4 menunjukkan karakteristik coupling antara Modul A dan Modul B. Panah dari Modul A ke Modul B menunjukkan bahwa Modul berisi satu atau beberapa CALL ke Modul B. Istilah "CALL" (nama paragraf **PERFORM** dalam COBOL atau nama modul dalam pernyataan subroutine FORTRAN) mengacu pada mekanisme yang digunakan untuk meminta suatu modul. Statement atau pernyataan CALL atau **PERFORM** adalah mekanisme hubungan paling sederhana, paling fleksibel dan kelihatan dalam kebanyakan bahasa tingkat tinggi. Panah-panah samping menandakan parameter yang disampaikan diantara modul-modul. Panah-panah ini

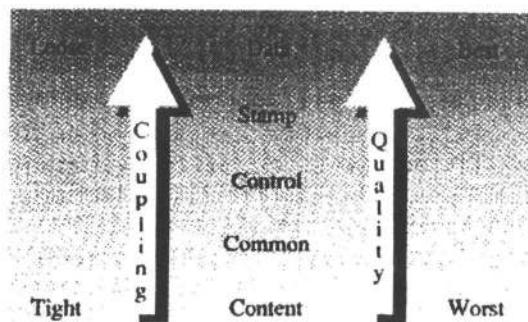


Gambar 2.4 Modul A memanggil (call) Modul B.

juga menunjukkan arah data disampaikan. Modul A menyampaikan item Data Y ke Modul B. Modul B menyampaikan item Data Z kembali ke Modul A. Coupling yang baik adalah berupa pasangan modul-modul yang berkomunikasi pada dua bagian data dari satu modul ke modul lainnya, seperti terlihat pada Gambar 2.4. Coupling yang kurang baik adalah berupa pasangan modul-modul yang mengkomunikasikan berbagai bagian data ke sana kemari.

Definisi coupling yang lebih presisi dilukiskan pada Gambar 2.5. Karena coupling yang baik merupakan fasilitas pokok pada rancangan terstruktur, kita akan melihatnya lebih dekat lagi kaitannya dengan coupling data, stamp, control, common, dan content. Coupling data, stamp, dan control menyajikan coupling normal. Coupling common dan content menyajikan coupling abnormal; mereka ini harus dihindari. Pengujian praktis terhadap kualitas coupling diukur menurut seberapa mudahnya setiap modul dikode oleh programmer yang berbeda selagi programmer memelihara tingkat independensi yang tinggi dari setiap modul itu. Jadi, loose coupling akan memudahkan pembagian pekerjaan diantara para programmer.

Beberapa ahli menyatakan bahwa coupling yang tingkatnya lebih tinggi lagi bisa dilakukan, yaitu apabila Modul A di-coupled ke Modul B hanya dengan cara A memanggil nama B dan tidak ada data yang disampaikan diantara mereka. Yakni, mereka di-coupled namanya saja. Namun, jenis coupling yang kita bahas disini adalah mereka yang disebutkan dalam Gambar 2.5.



Gambar 2.5 Jenis dan tingkat coupling.

Coupling Data. Dua modul akan mempunyai coupling data jika data yang diperlukan dikomunikasikan diantara mereka. Modul A memanggil Modul B, B mengembalikan ke A, dan data disampaikan diantara mereka dengan menggunakan parameter yang disajikan dengan CALL (atau PERFORM) sendiri. Sebagai contoh, Modul B adalah suatu algoritma yang menghitung ulang digit pengecekan nomor rekening untuk menentukan validitasnya. Pesan yang valid atau tidak valid disampaikan ke Modul A yang membaca transaksi pelanggan.

Coupling Stamp. Dua modul dikatakan *stamp coupled* jika mereka mengkomunikasikan sekelompok item data terkait seperti record yang terdiri atas berbagai field. Suatu perubahan format ataupun struktur suatu record atau field dalam record tersebut akan berpengaruh terhadap modul-modul yang menggunakan record itu. Oleh karena itu, coupling stamp cenderung membuka modul ke item data yang lebih banyak dari pada yang ia perlukan. Aturannya disini adalah: Jangan menyampaikan record yang berisi banyak field ke modul yang hanya memerlukan satu atau dua dari field-field tersebut. Sebagai contoh, Modul memesan kembali inventori berdasarkan data dari Modul A yang mengontrol akses ke record inventori. Dari pada menyampaikan deskripsi inventori, harga, biaya, lokasi gudang, dan data tak berguna lain, Modul A hanya menyampaikan nomor inventori, kuantitas yang ada saat itu, dan kode vendor, yang mereka ini adalah item-item data yang hanya diperlukan Modul B untuk menjalankan fungsinya.

Coupling Control. Dua modul dikatakan *control coupled* jika salah satu modul mengkomunikasikan data yang mengontrol logika internal dari modul yang lain. Sebagai contoh, Modul B mengkomunikasikan (menyampaikan) pesan end-of-file (EOF) ke Modul A. Modul A kemudian menghentikan fungsinya dan mentransfer kendali ke modul yang lain.

Coupling Common. Dua modul atau lebih dikatakan *common coupled* jika mereka berbagi pakai data yang disimpan dalam area biasa (umum). Contoh coupling common terjadi ketika seorang programmer menggunakan klausa REDEFINES dari COBOL. Klausa REDEFINES memungkinkan penyimpanan berbagai item data dalam lokasi penyimpanan fisik yang sama. Ia adalah klausa paling kompleks dalam COBOL. Ia dapat menyebabkan pemakai melaporkan kesalahan dan masalah pemeliharaan. Pada mulanya, alasan penggunaannya adalah untuk menghemat penyimpanan memori, namun sekarang ini hal itu tidak berlaku, karena saat ini telah ada

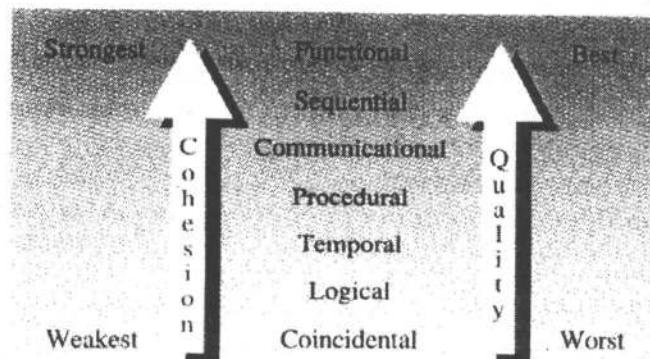
penyimpanan berkapasitas megabyte. Jadi, klausa REDEFINES tidak boleh digunakan.

Coupling Content. Dua modul atau lebih dikatakan *content coupled* jika salah satu modul mengacu pada atau mengubah isi yang lain dengan cara apapun. Ini adalah jenis coupling yang paling jelek, yang kadang-kadang disebut pathological atau sick coupling. Sebagai contoh, ia mengubah statement dalam modul lain, atau ia masuk ke dalam modul lain. Statement ALTER yang digunakan dalam versi COBOL pertama merupakan contoh yang tepat untuk coupling content ini. Ia ditiadakan dalam versi COBOL berikutnya.

Kohesi

Ukuran kedua dan sebagai ukuran pelengkap terhadap independensi modul adalah kohesi, yang kadang-kadang disebut binding. Kohesi megukur kekuatan hubungan antara elemen-elemen kode di dalam suatu modul. Ia merupakan cara untuk menjabarkan derajad suatu modul dalam menjalankan fungsi tunggal yang telah tertetapkan dengan baik. Istilah itu dipinjam dari sosiologi, dimana ia berarti keterkaitan manusia di dalam suatu kelompok.

Gambar 2.6 menggambarkan jenis dan tingkatan kohesi. Suatu sistem yang modul-modulnya sangat kohesif biasanya akan mempunyai low or loose coupling, keduanya merupakan sasaran rancangan dari rancangan perangkat lunak terstruktur.



Gambar 2.6 Jenis dan tingkatan kohesi.

Jika para perancang mengembangkan modul-modul yang melampaui kohesi komunikasional, maka mereka telah menyimpang dari batasan modul-modul yang tertetapkan dengan baik dan bisa dikelola dengan mudah ke modul-modul yang terancang dengan tidak baik dan kurang bisa dikelola.

Kohesi Fungsional. (Functional Cohesion) Modul yang kohesif secara fungsional berisi prosedur-prosedur yang semuanya mengkontribusi pengeksekusian satu atau hanya satu tugas yang telah tertetapkan dengan baik. Tujuan rancangan perangkat lunak adalah agar semua modul secara fungsional kohesif. Modul-modul yang membaca record, menghitung pembayaran bersih, mengkalkulasi poin reorder, atau mencetak laporan adalah contoh-contoh modul yang menjalankan satu tugas sampai selesai dan oleh karenanya mereka secara fungsional kohesif.

Kohesi Berurutan. (Sequential Cohesion) Suatu modul akan mempunyai kohesi berurutan jika prosedurnya terdiri atas urutan aktivitas, misalnya output dari satu aktivitas merupakan input bagi aktivitas berikutnya dari modul itu. Contohnya adalah prosedur yang membaca dan mengedit data, membuat dan menyimpan record dalam suatu file, dan mengakumulasi data dan mencetaknya.

Kohesi Komunikasional. (Communicational Cohesion) Suatu modul dikatakan mempunyai kohesi komunikasional jika prosedur-prosedurnya mengkontribusi aktivitas-aktivitas yang menggunakan data yang sama untuk tujuan yang berbeda. Sebagai contoh, prosedur mungkin menghasilkan banyak output dari aliran data yang sama. Transaksi merupakan input, dan ia digunakan untuk mengupdate file master dan untuk mencetak laporan transaksi, atau prosedur dapat mengupdate atau menghapus suatu record.

Kohesi Prosedural. (Procedural Cohesion) Modul yang secara prosedural kohesif adalah modul yang prosedur-prosedurnya terdiri atas aktivitas-aktivitas yang berbeda dan mungkin tidak berkaitan, dimana di dalamnya kendali mengalir dari setiap aktivitas ke aktivitas berikutnya. Suatu modul, misalnya, bisa membaca record pekerja, menghitung pembayaran bersih, menampilkan angka ketidakhadiran, mencetak laporan evaluasi, mengalokasikan waktu liburan, dan mencetak cek pembayaran. Prosedur-prosedur ini dihubungkan oleh urutan eksekusi, bukannya oleh suatu fungsi tugas khusus.

Kohesi Temporer. (Temporal Cohesion) Tingkat kohesi ini adalah berkaitan dengan waktu. Disini, hubungan terkuat antara prosedur-prosedur adalah bahwa mereka semua dieksekusi pada waktu yang bersamaan. Modul yang mengedit semua field dalam suatu record secara serentak dikatakan kohesif secara temporer. Modul inisialisasi yang memutar balik pita, menyetel pencacah pita, membuka file, memindahkan ruang ke output, dan menyetel operasi switch adalah contoh modul yang kohesif secara temporer.

Kohesi Logik. (Coincidental Cohesion) Modul akan mempunyai kohesi logis jika elemen-elemennya tidak dihubungkan oleh arus data atau arus kendali, namun hanya berhubungan dengan tugas-tugas dari kelas fungsi umum yang sama. Seringkali, kode digunakan bersama (dibagi) oleh fungsi-fungsi yang ada dalam modul yang sama. Modul yang secara logis kohesif bisa mengedit record, melakukan update, dan menghapus atau melakukan penambahan. Setiap fungsi ini harus dibagi ke dalam beberapa modul yang lebih khusus fungsinya.

Kohesi Kebetulan. Dalam modul yang kohesif secara kebetulan, tak ada hubungan yang berarti antara prosedur-prosedur dari modul yang ada; mereka hanya kebetulan bersama. Kohesi kebetulan adalah suatu kebalikan dari apa yang sedang kita coba untuk mencapai rancangan terstruktur; oleh karenanya, ia harus dihindari.

Pedoman

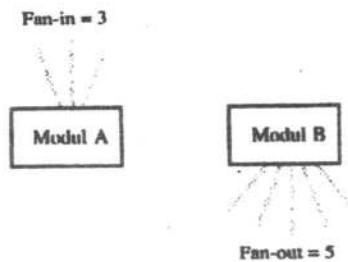
Rancangan modular yang mendasari adalah suatu set pedoman yang sangat saling terkait yang digunakan untuk membantu pengembangan set modul yang terancang dengan baik. Pedoman-pedoman ini akan dikemukakan nanti.

Pemfaktoran dan Ukuran Modul. Pemfaktoran, yang juga disebut penyamaraan (*leveling*), adalah pendekomposision atau pembagian satu modul ke dalam beberapa modul. Tujuan dalam pemfaktoran adalah untuk mengurangi ukuran modul dan untuk meningkatkan sifat coupling dan kohesi dari modul. Jika muncul pertanyaan apakah membagi modul tertentu ataukah tidak, maka bagilah. Lebih baik mengembangkan terlalu banyak modul dari pada terlalu sedikit. Namun modul yang mempunyai kurang dari lima atau enam jalur kode yang bisa dieksekusi harus diperiksa guna melihat apakah mereka harus digabungkan ke dalam caller mereka.

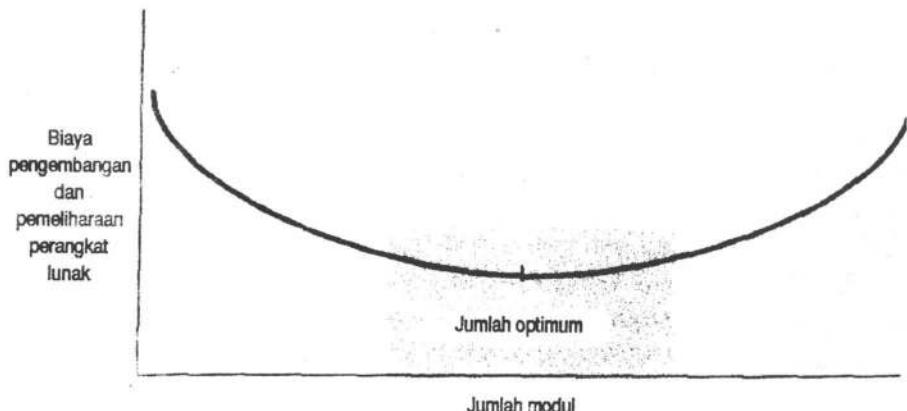
Umumnya, suatu modul harus terdiri tidak lebih dari 50 LOEC. Biasanya, modul satu-fungsi yang tertetapkan dengan baik tidak akan lebih dari 15 sampai 25 LOEC dan 5 sampai 11 jalur penjelasan atau naratif, yang tidak bisa dieksekusi.

Pemisahan Keputusan. (Desicion Splitting) Pemisahan keputusan terjadi jika dua bagian dari suatu keputusan dipisahkan dan ditempatkan ke dalam dua modul yang berbeda. Pemisahaan keputusan harus dihindari jika mungkin: misalnya, keputusan "Jika Reorder-Point sama dengan atau kurang dari Reorder-Point Number, maka Jalankan Reorder-Routine." Keputusan ini masih utuh; yakni, bagian If dan Then dari keputusan itu tidak dipisahkan.

Fan-in/Fan-out. Fan-in dari suatu modul adalah jumlah modul yang memanggilnya. Fan-out, atau jangkauan kendali, adalah jumlah subordinat langsung ke suatu modul. Baik fan-in dan fan-out ditunjukkan dalam Gambar 2.7. Umumnya, modul-modul fan-in harus setinggi mungkin agar mereka bisa melayani fungsi umum, seperti routine pencetakan atau beberapa jenis utilitas. Jenis modul ini mempunyai kemampuan tinggi untuk bisa digunakan kembali. Kecuali untuk modul-modul yang bisa digunakan kembali dan utilitas, jika suatu modul dipanggil oleh lebih dari tiga modul, ia bisa berisi dua fungsi atau lebih dengan modul pemanggilan yang berbeda yang menggunakan fungsi yang berbeda. Jika situasi ini terjadi, bagilah modul yang dipanggil (diminta) ke dalam modul-modul yang terpisah, yang masing-masing berisi fungsi spesifik. Umumnya, modul superior tidak boleh memanggil (call) lebih dari lima modul-modul subordinat.



Gambar 2.7 Modul fan-in dan modul fan-out.



Gambar 2.8 Jumlah optimum modul yang dikaitkan dengan biaya pengembangan dan pemeliharaan perangkat lunak.

Jumlah Modul. Ada batasan maksimum jumlah modul, seperti terlihat pada Gambar 2.8. Untuk memahami rancangan perangkat lunak terstruktur, sebaiknya kita teliti mengapa modularitas cenderung mengurangi biaya pengembangan dan pemeliharaan perangkat lunak.

Pedoman pengukurannya adalah dengan cara membagi total LOEC yang diestimasikan dengan 50 (atau kurang dari itu). Sebagai contoh, jika suatu proyek pengembangan perangkat lunak diestimasikan memerlukan 200 KLOEC, maka 4000 ($200.000 \text{ LOEC} / 50 \text{ LOEC per modul}$) modul adalah angka yang mendekati jumlah optimum modul untuk proyek tersebut. Para ahli lain bisa menaikkan atau mengurangi pembagi sebesar lima LOEC atau lebih, namun 50 adalah jumlah yang cukup umum. Inilah pedomannya; ia tidak presisi, namun memberikan estimasi yang cukup baik.

Rancangan Top-to-Bottom dan Pengkodean Top-to-Bottom atau Bottom-to-Top

Walau pun pendekatan terstruktur untuk analisis dan rancangan sistem adalah top-down, pengkodean terstruktur bisa dengan cara top-down atau bottom-up. Hasil dari pendekatan atau cara yang mana saja adalah program moduler yang terstruktur secara hirarkis. Dalam beberapa situasi, campuran pengkodean dan pengujian secara top-down dan bottom-up lebih dikehendaki. Sebagai contoh, kita akan lebih untung

mengembangkan modul-modul umum atau utiliti secara bottom-up, sebab mereka akan dipanggil oleh modul-modul yang tingkatnya lebih tinggi.

Menggunakan Konsepsi Kendali Standar untuk Membangun Program Terstruktur

Teori di balik rancangan perangkat lunak terstruktur adalah bahwa segala logika program dapat dikonstruksi dari kombinasi tiga konstruksi kendali:

- ◆ Urutan (Sequence)
- ◆ Seleksi (Selection)
- ◆ Repetisi (Repetition)

Ketiga konsepsi ini dilukiskan dengan alat modeling terstruktur dalam Gambar 2.9. Yang pokok atau mendasar untuk ketiga konsepsi ini adalah bahwa masing-masing mempunyai poin entri dan exit tunggal.

Urutan

Dengan konsepssi kendali urutan, statemen program dieksekusi satu persatu dengan urutan yang sama seperti ketika mereka berada dalam kode sumber. Perintah-perintah untuk urutan selalu berupa kata kerja, seperti MOVE, READ, WRITE, SUBTRACT, ADD, DISPLAY, COMPUTE, dan semacamnya. Lihat contoh berikut ini:

```
MOVE ZEROS TO AMOUNT-OUT  
COMPUTE GROSS-PAY = RATE TIMES HOURS-WORKED  
SUBTRACT DEDUCTIONS FROM GROSS-PAY GIVING NET-PAY
```

Seleksi

Suatu kondisi menyebabkan sistem melakukan transisi dan melakukan tindakan spesifik yang didasarkan pada nilai variabel kondisi tersebut. Suatu perangkat statemen akan dieksekusi hanya jika kondisi yang dinyatakan berlaku (bisa diterapkan), menurut kondisi IF-THEN-ELSE. Misalnya:

```
IF  
    HOURS WORKED GREATER THAN 40
```

THEN

```
    COMPUTE GROSS-PAY WITH OVERTIME RATE  
ELSE  
    COMPUTE GROSS-PAY WITH REGULER PAY  
ENDIF
```

Repetisi

Konsepsi repetisi (yang juga disebut looping atau iterasi) memungkinkan program mengeksekusi satu rangkaian/seri langkah atau lebih. Ada dua jenis iterasi: DO UNTIL dan DO WHILE, atau statemen-statemen yang ekuivalen.

Dengan DO UNTIL, set instruksi dieksekusi. Kemudian kondisi terminasi loop diuji. Jika kondisinya true (benar), loop (putaran) akan diterminasi atau dihentikan, dan eksekusi berlanjut dengan instruksi nomor urut berikutnya. Jika kondisinya false (salah), set instruksi itu dieksekusi lagi. Perlu dicatat bahwa loop atau putaran DO UNTIL selalu dieksekusi sedikitnya sekali. Sebagai contoh, jika kita ingin mencetak nama 50 pelanggan, perintah berikut ini akan menjalankan tugas ini:

```
PRINT HEADINGS  
INITIALIZE COUNTER TO 1  
DO UNTIL COUNTER EQUAL TO 51  
    PRINT CUST-NAME  
    INCREMENT ROW COUNTER BY 1  
ENDDO  
STOP
```

Dengan DO WHILE, kondisi terminasi diuji. Jika kondisinya false, maka loop dihentikan dan eksekusi berlanjut dengan instruksi nomor urut berikutnya. Jika kondisinya true, set instruksi dieksekusi, dan kondisinya diuji lagi. Jika kondisinya pada awalnya false, maka loop tersebut tidak akan dieksekusi sama sekali. Contoh berikut menunjukkan suatu loop DO WHILE untuk membaca file record pelanggan dan mencetak alamat pelanggan:

```
PRINT HEADING  
READ CUST-RECORDS  
DO WHILE CUST-RECORDS REMAIN TO BE PROCESSED  
    PRINT CUST-ADDRESS  
    READ CUST-RECORDS  
ENDDO  
STOP
```

Diagram a (lihat halaman 619)

Structured tools Constructs	Structured program flowchart	Jackson diagram	Structure chart	Warren-Orr diagram
Sequence	A — B — C	A — B — C	A — B — C	\overline{A} — B — C
Selection IF-THEN-ELSE	A — B — C	A — B — C	A — ? — C	$\overline{B} \oplus C$
Repetition DO UNTIL or DO WHILE	<p>DO WHILE</p> <pre> graph TD A[A] --> B[B] B --> C{?} C -- T --> A C -- F --> D(()) D --> B </pre>	<p>DO UNTIL</p> <pre> graph TD A[A] --> B[B] B --> C{?} C -- F --> D(()) D --> B </pre>	<p>DO WHILE</p> <pre> graph TD A[A] --> B[B] B --> C{?} C -- T --> A C -- F --> D(()) D --> B </pre>	<p>DO WHILE</p> <pre> graph TD A[A] —> B[B] B —> C{?} C — T —> A C — F —> D(()) D —> B </pre>

Loop (putaran) DO WHILE tersebut meminta agar iterasi terus berlanjut sepanjang (sementara) tes (pengujian) kondisi true. Dalam kasus ini, kondisi yang sedang diuji itu adalah record yang baru saja dibaca ataukah bukan, ia merupakan record terakhir dalam file tersebut (yakni, end-of-file atau EOF). Apabila pengujian kondisinya false, maka operasi loop berakhiri.

Dua Perangkat Pemodelan yang Digunakan untuk Merancang Perangkat Lunak Terstruktur

Dua perangkat pemodelan populer yang digunakan dalam rancangan perangkat lunak terstruktur adalah:

- ◆ Bagan struktur
- ◆ Bahasa Inggris terstruktur

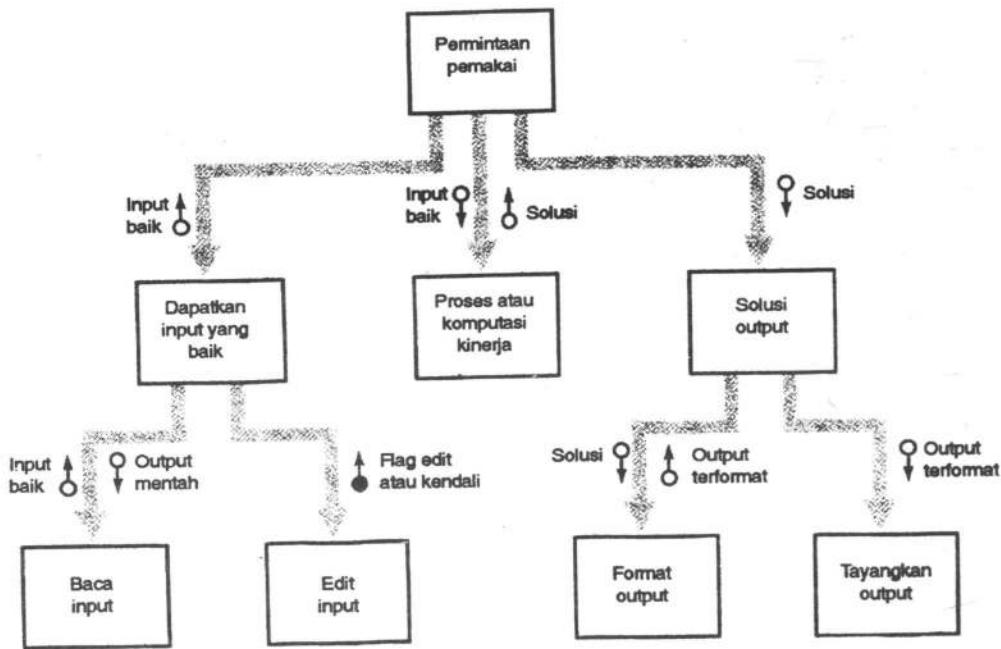
Bagan struktur dan Bahasa Inggris terstruktur biasanya digunakan untuk memodel rancangan perangkat lunak yang diabstraksi dari rancangan sistem yang dimodel dengan DFD dan STD. Dalam beberapa kuartal, alat modeling Jackson dan Warnier-Orr juga populer. Setelah membahas bagan struktur dan Bahasa Inggris terstruktur, kita akan mengemukakan bagaimana mereka digunakan bersama untuk mengkonversi (mengubah) rancangan sistem rinci menjadi rancangan perangkat lunak terstruktur, yang siap untuk tahap pengkodean.

Bagan Struktur

Bagan Struktur adalah diagram hirarkis yang menetapkan keseluruhan arsitektur rancangan perangkat lunak dengan menampilkan modul-modul program dan keterkaitan mereka. Gambar 2.10 menunjukkan bagan struktur generalisasi. Semua bagan struktur menganut bentuk umum ini, tak peduli banyaknya modul yang ada di dalamnya. Contoh payroll (daftar gaji) sederhana, yang ditunjukkan pada Gambar 2.11, digunakan untuk menunjukkan aplikasi bagan struktur tergeneralisasi.

Bahasa Inggris Terstruktur

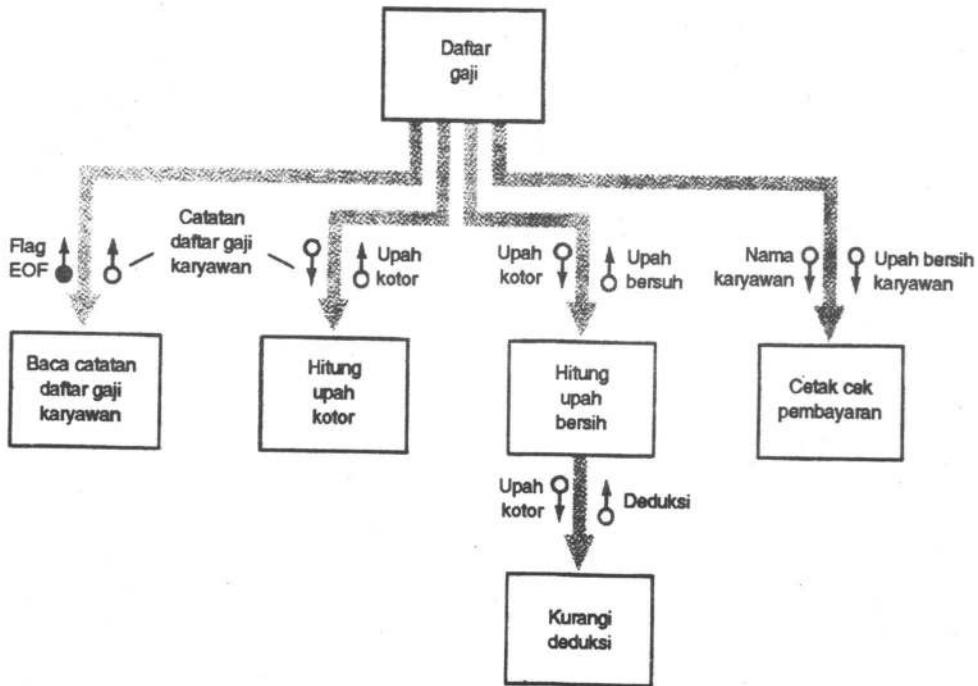
BAHASA INGGRIS TERSTRUKTUR adalah subset dari bahasa Inggris. Ia berfungsi sebagai alat spesifikasi yang menjabarkan secara rinci input, proses, dan output dari bagan struktur.



Gambar 2.10 Bagan struktur tergeneralisasi.

Bahasa Inggris terstruktur adalah suatu bahasa seperti bahasa pemrograman (yakni, pseudocode) yang digunakan untuk berkomunikasi dengan programmer dan juga non-programmer. Kosa katanya terkomposisi atas kata kerja, istilah-istilah dari kamus data, dan kata-kata cadangan yang menunjukkan proses logis. Tergantung dari sifat dan kompleksitas aplikasinya, tabel keputusan, pohon keputusan, dan persamaan bisa lebih tepat diterapkan dari pada Bahasa Inggris terstruktur untuk beberapa aplikasi, atau mereka bisa digunakan untuk melengkapi Bahasa Inggris terstruktur.

Aturan untuk Menulis Bahasa Inggris Terstruktur. Bahasa Inggris terstruktur menganut konsepsi pemrograman dasar, yaitu urutan, seleksi dan repetisi.



Gambar 2.11 Bagan struktur yang digunakan untuk merancang modul-modul untuk suatu program daftar gaji.

Statemen-statemen dirancang untuk menunjukkan hierarki logik. Beberapa keyword Bahasa Inggris terstruktur adalah FOR, GET, READ, SET, ENTER, COPY, WRITE, IF, NOT, EQUAL, THEN, PERFORM, MOVE, ELSE, COMPUTE, ENDIF, REPEAT WHILE atau DO WHILE, REPEAT UNTIL atau DO UNTIL, ENDREPEAT atau ENDDO, dan EXIT. Keyword yang digunakan untuk logika adalah AND, OR, GREATER THAN, dan LESS THAN.

Contoh Bahasa Inggris Terstruktur. Suatu contoh mengenai bagaimana Bahasa Inggris terstruktur digunakan untuk menetapkan pemrosesan suatu pesanan pelanggan ditunjukkan dalam Gambar 2.12. Perlu dicatat bahwa kumpulan instruksi ini tidak hanya mengkomunikasikan proses secara jelas, namun juga dapat dikonversi secara mudah ke bahasa program seperti C atau COBOL.

```

ORDER_ENTRY:
    FOR each customer PURCH_ORD
        GET CUSTOMER record
        IF CUS_NUM is valid
            SET INVOICE_HEADER record
            ENTER CUS_NAME, CUS_ADDR, DATE_ORD,
                  and PO_NUM in INVOICE_RECORD
            WRITE INVOICE_HEADER record
            GET DISCOUNT from table
        ELSE
            Display "Invalid Customer Number"
            QUIT ORDER_ENTRY
        ENDIF
        FOR each line item
            COPY ITEM_NUM and QTY_ORD on
                INVOICE record
            GET ITEM_PRICE from price table
            SET ITEM_SUBTOTAL to ITEM_PRICE ×
                QTY_ORD × (100-DISCOUNT)
            SET INVOICE_TOTAL to sum of ITEM_SUBTOTAL
            Write INVOICE_RECORD
        ENDFOR
        Prepare bill of lading
    ENDFOR
    EXIT ORDER_ENTRY.

```

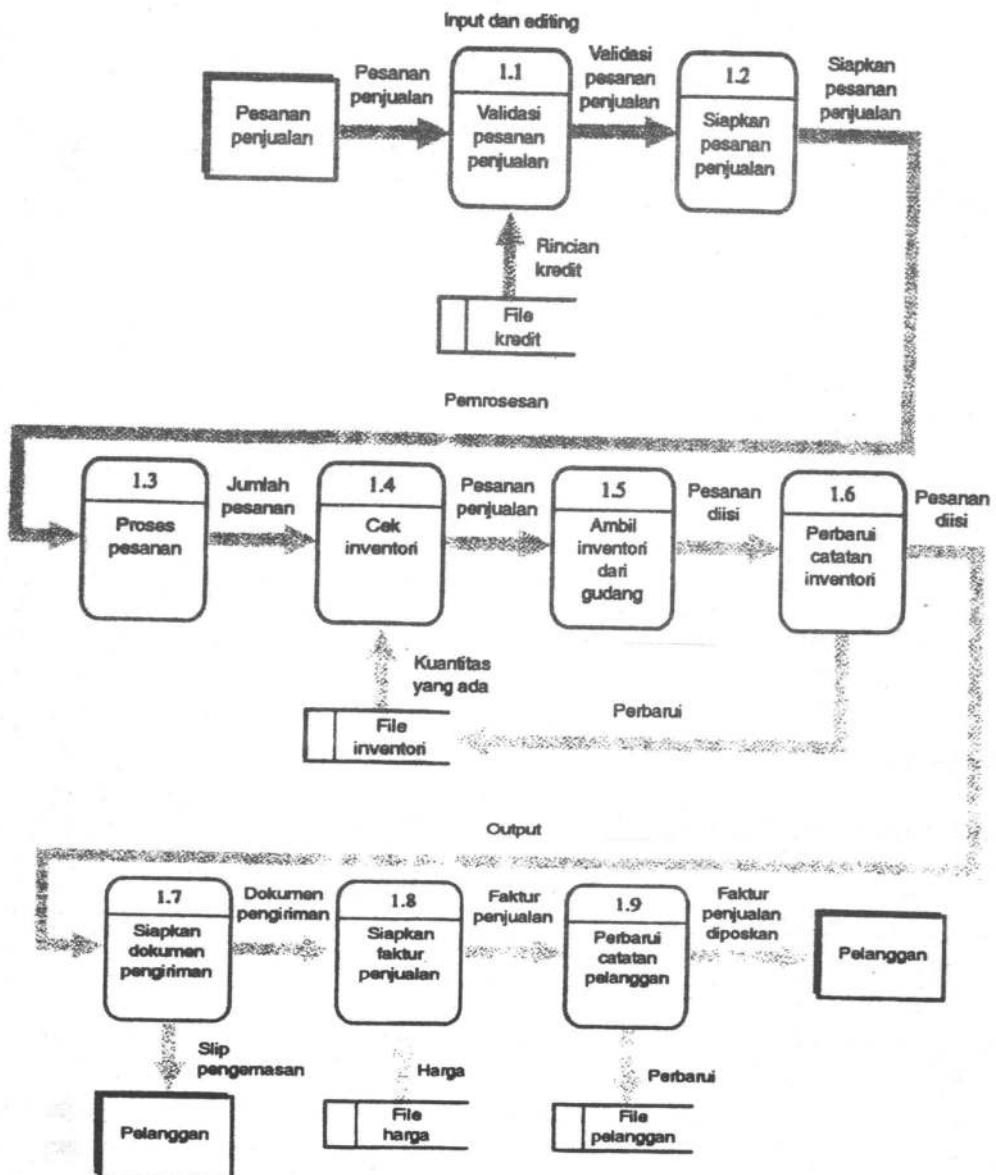
Gambar 2.12 Bahasa Inggris terstruktur yang digunakan untuk menetapkan entri pesanan.

Mentransformasi Diagram Arus Data Ke dalam Bagan Struktur

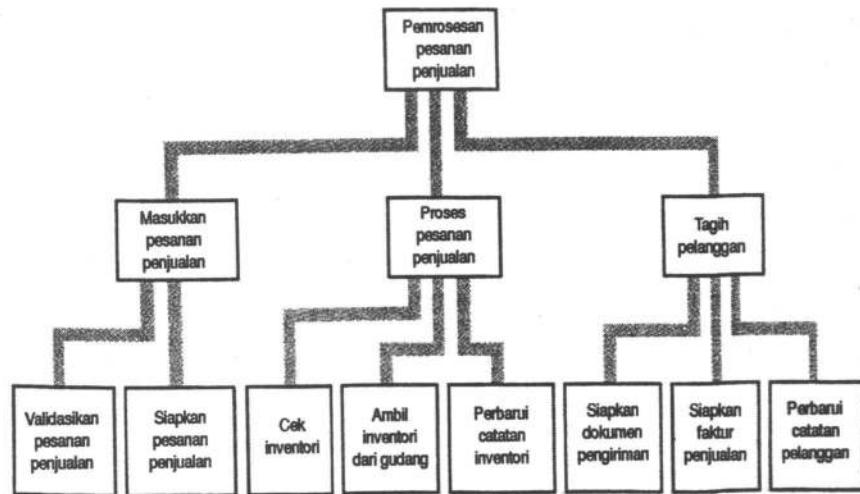
DFD yang dilukiskan pada Gambar 2.13 merepresentasikan spesifikasi rancangan sistem untuk memproses transaksi penjualan. DFD ini telah didekomposisi secara fungsional ke suatu tingkat yang rinci, dan dengan demikian ia menjadi dasar untuk rancangan perangkat lunak terstruktur yang menggunakan bagan struktur.

Pentransformasian DFD seperti yang ditampilkan pada Gambar 2.13 tersebut ke dalam bagan struktur akan melibatkan Analisis Transformasi, yang meminta (mengharuskan) pemeriksaan DFD membagi proses-prosesnya ke dalam proses-proses yang:

- ◆ Melakukan input dan pengeditan, seperti validasi pesanan penjualan.
- ◆ Melakukan pemrosesan, seperti pengecekan inventori.
- ◆ Menggenerasi output, seperti penyiapan dokumen pengiriman.



Gambar 2.13 Rancangan DFD dari suatu sistem pemrosesan pesanan penjualan.



Gambar 2.14 Bagan struktur untuk pemrosesan pesanan penjualan yang dihasilkan dari diagram arus data.

Bagan struktur yang dihasilkan dari pembagian DFD ke dalam proses-proses input, pemrosesan, dan output ditunjukkan pada Gambar 2.14. Bagan struktur ini merepresentasikan modul-modul yang akan dikode.

Hampir semua sistem CASE memudahkan analisis transformasi. Mereka terutama berguna untuk menggambar bagan struktur dan menghasilkan Bahasa Inggris terstruktur. Beberapa sistem CASE juga bisa menghasilkan bagan struktur secara langsung dari DFD.

APA YANG DIMAKSUD RANCANGAN PERANGKAT LUNAK BERORIENTASI-OBYEK?

Dalam bab ini, kita akan membahas rancangan perangkat lunak berorientasi obyek secara lebih rinci, khususnya dalam kaitannya dengan penyiapan/pembuatan RANCANGAN PERANGKAT LUNAK BERORIENTASI-OBYEK.

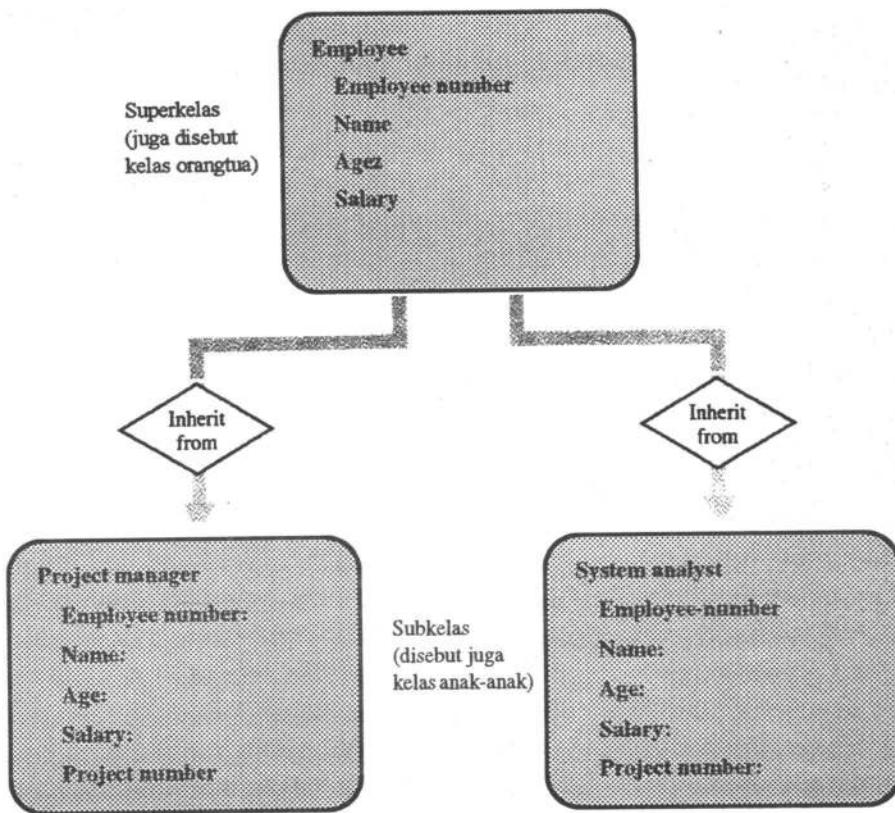
Lebih Jauh tentang Obyek

Obyek memberi identitas kepada orang atau benda. Obyek merepresentasikan entitas kongkrit dari bidang aplikasi yang sedang dirancang. Beberapa contoh obyek adalah entitas seperti siswa, pelanggan, analis sistem, inventori, rekening, dan entri pesanan. Obyek-obyek adalah contoh dari satu kelas atau lebih. Obyek meringkas (encapsulate) atribut data (yang juga disebut struktur data atau disebut atribut saja) dan operasi (yang juga disebut prosedur). Operasi berisi metode (yakni, kode program) yang beroperasi pada atribut itu. Beberapa pakar mendefinisikan obyek sebagai data dan kode teringkas, atau data dan metode. Alasan kita menggunakan istilah “atribut” dan “operasi” adalah bahwa istilah-istilah ini memberikan cara yang bisa kita gunakan untuk mengembangkan rancangan perangkat lunak berorientasi obyek (yakni, kerangka kerja semantik logis) tanpa perlu memikirkan masalah format data dan sintaks kode. Dengan kata lain, model rancangan harus bersifat programming-language-independent (tidak tergantung bahasa pemrograman tertentu). Apabila rancangan perangkat lunak telah tercipta, format data akan dijabarkan secara rinci, dan kode program (metode) akan ditulis/dibuat untuk mengimplementasikan rancangan tersebut.

Lebih Jauh Tentang Kelas

Kelas obyek mendeskripsikan kelompok obyek yang mempunyai atribut sama, tingkah laku umum, dan hubungan umum (biasa) dengan obyek lain. (Tingkah laku adalah sesuatu/apa yang dilakukan obyek ketika ia dikirim pesan, seperti apa yang dilakukan harimau ketika ia diberi instruksi oleh pelatihnya). Istilah kelas (class) lebih sering digunakan daripada kelas obyek.

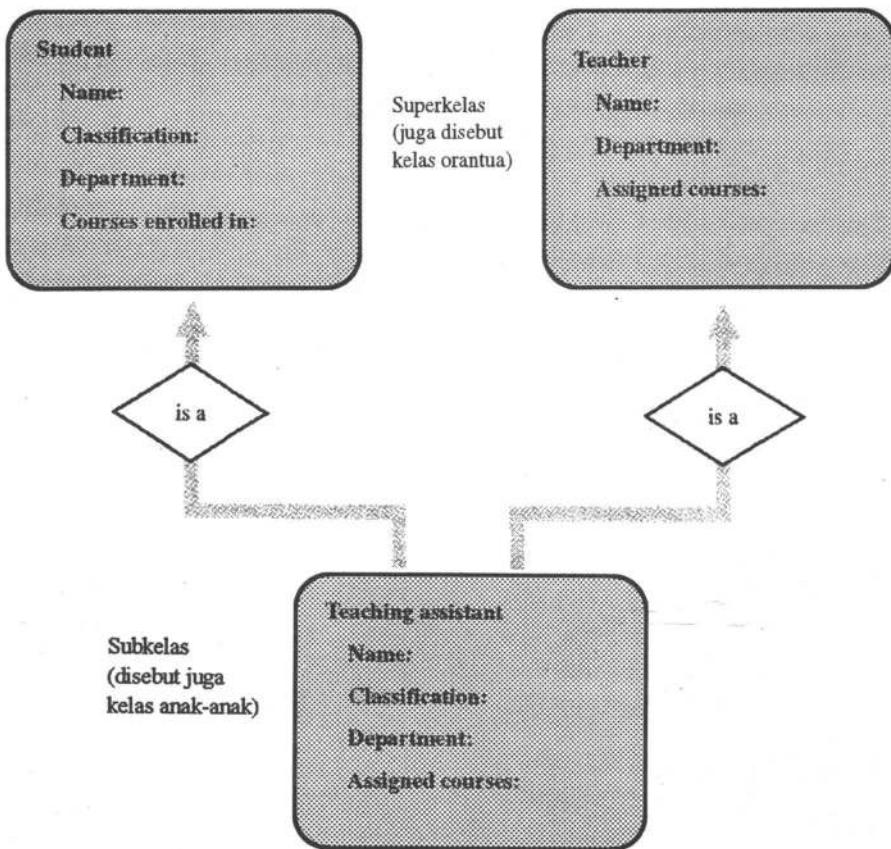
Kelas bisa dinyatakan/dianggap sebagai subkelas dari kelas lain yang dikenal dengan nama superkelas. Dalam hal ini, semua obyek yang termasuk dalam subkelas tersebut juga termasuk dalam superkelas. Pewarisan (inheritance) adalah kemampuan untuk mendefinisikan subkelas obyek dari suatu kelas obyek. Apabila kita lihat menurut ilmu genetika, pewarisan adalah kemampuan suatu keturunan untuk menerima sifat atau karakteristik dari orang tuanya. Sebagai contoh, dalam Gambar 2.15, PROJECT MANAGER dan SYSTEMS ANALYST adalah subkelas (turunan) dari superkelas EMPLOYEE (orang tua) dan mereka menuruni/mewarisi semua sifatnya. Dengan demikian Employee-Number, Name, Age, dan Salary, dan kode (tidak terlihat) yang beroperasi pada atribut data ini tidak perlu didefinisikan ulang



Gambar 2.15 Subkelas yang mewarisi sifat-sifat dari super-kelas.

dalam subkelas. Namun atribut-atribut lain bisa ditambahkan ke obyek-obyek ini, seperti Project-Number.

Subkelas mungkin mempunyai banyak superkelas, sehingga ia mempunyai pewarisan banyak. Atau, kita bisa mengatakan bahwa turunan atau anak mungkin bisa mempunyai beberapa orang tua. Seperti halnya seorang anak bisa mewarisi sifat tertentu dari bapak atau ibunya, suatu obyek juga dapat mewarisi sifat/karakteristik dari satu penurunnya atau lebih. Sebagai contoh, TEACHING ASSISTANT bisa menjadi subkelas dari superkelas STUDENT maupun superkelas TEACHER, seperti dilukiskan pada Gambar 2.16.



Gambar 2.16 Subkelas dengan beberapa superkelas.

Obyek bisa mendapatkan dan melepaskan kelas secara dinamis. Obyek yang merepresentasikan seseorang bisa diciptakan sebagai contoh kelas STUDENT. Apabila ini terjadi, semua atribut dan operasi yang didefinisikan dalam EMPLOYEE akan menjadi berlaku pada obyek itu, dan atribut dan operasi dalam STUDENT akan menjadi tidak berlaku.

Lebih Jauh Tentang Relasi

Relasi mendeskripsikan asosiasi antara kelas dan obyek. Relasi kelas menunjukkan beberapa macam pembagian (penggunaan bersama) atau koneksi bermakna antara

kelas-kelas. Relasi kelas memungkinkan pewarisan. Relasi bisa bersifat satu-satu, satu-banyak, atau banyak-banyak. Dalam rancangan berorientasi obyek, aspek hubungan ini disebut *multiplicity*, yang menetapkan atau menentukan banyaknya kejadian dari satu kelas yang mungkin berhubungan dengan kejadian tunggal dari kelas yang terkait. Umumnya, perancang perangkat lunak tidak akan khawatir dengan adanya multiplicity pada awal pengembangan perangkat lunak. Hal pertama yang harus dikerjakan adalah menentukan obyek, kelas, dan relasi. Kemudian baru memutuskan multiplicity-nya.

MENGGUNAKAN PENDEKATAN RANCANGAN BERORIENTASI-OBYEK UNTUK MERANCANG MODEL PERANGKAT LUNAK

Rancangan perangkat lunak berorientasi obyek menghasilkan suatu model yang mendeskripsikan obyek-obyek, kelas-kelas, dan relasi mereka antara satu dengan lainnya. Pengidentifikasi kelas dan obyek merupakan tujuan fundamental dari rancangan perangkat lunak berorientasi obyek. Model rancangan sistem, seperti DFD, kamus data, dan model data logik menentukan bidang sistem. Rancangan perangkat lunak berorientasi obyek mengidentifikasi kelas dan obyek yang akan mengimplementasikan bidang ini. Umumnya, cara terbaik untuk menciptakan model rancangan perangkat lunak berorientasi obyek adalah dengan membangun hirarki kelas, dimana dalam hirarki ini subkelas akan mewarisi atribut dan operasi dari kelas atau superkelas yang sifatnya lebih umum.

Pemodelan Berorientasi-Obyek

Kita perlu cara formal untuk mengekspresikan rancangan berorientasi obyek secara logik dan benar. Walaupun beberapa perangkat pemrograman menyediakan cara untuk melakukan hal ini, kita ditawari menggunakan ERD, dengan beberapa modifikasi, sebagai pendekatan yang bisa bekerja. (Pada kenyataannya, tak ada ERD standart). Walaupun versi ERD dari Chen biasa digunakan, khususnya untuk pemodelan data, kita akan menggunakan variasi dari versi ini untuk memodel rancangan berorientasi obyek. Variasi ini memberikan notasi grafis formal untuk memodel obyek, kelas, dan hubungannya satu sama lain. Namun demikian, rancangan perangkat lunak yang baik akan dihasilkan oleh perancang yang baik, bukan berkat perangkat pemodelan. Perangkat pemodelan (modeling tool) berorientasi obyek, seperti ERD

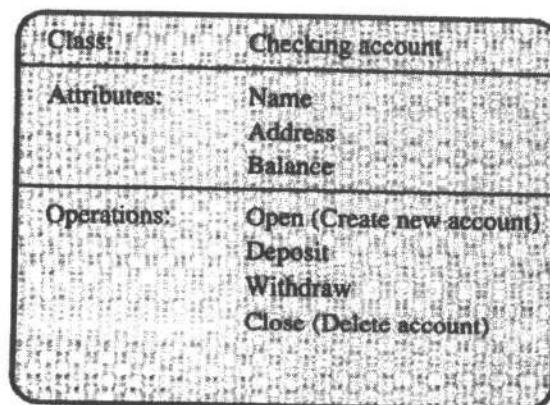
termodifikasi, hanya membantu perancang untuk berkonsentrasi pada aspek kreatif rancangan tanpa peduli banyak mengenai mekanismenya. Seperti rancangan perangkat lunak terstruktur, rancangan berorientasi obyek seringkali diabstraksi dari suatu DFD. Pada kenyataannya, setiap sumber atau sink, proses, atau penyimpanan data pada suatu DFD yang terdekomposisi secara penuh bisa menunjukkan calon obyek, atau, mungkin, beberapa obyek. Kelas-kelas calon bisa dihasilkan atau diperoleh dari arus data DFD. Kamus data dan model data logik yang disiapkan selama rancangan sistem terinci akan menjadi atribut data dalam suatu obyek, atau mungkin, beberapa obyek. Model ini juga menunjukkan relasi kelas dan obyek yang mungkin terjadi. STD menunjukkan tingkah laku dinamik yang berkaitan dengan kelas tertentu.

Mendeskripsikan Obyek dengan Menggunakan Notasi ERD Termodifikasi

Simbol entitas termodifikasi dari ERD konvensional (yakni, ERD yang digunakan untuk pemodelan skala perusahaan dan pemodelan data) adalah persegi dengan pojok-pojok yang bulat, seperti terlihat pada Gambar 2.17. Simbol ini sekarang disebut kotak kelas (class box).

Apa yang Dimaksud Kotak Kelas?

KOTAK KELAS mempunyai tiga wilayah. Wilayahnya, dari atas ke bawah, berisi nama kelas, daftar atribut, dan daftar operasi. Pada model rancangan berorientasi obyek



Gambar 2.17 Kotak kelas.

yang pertama, atribut dan operasi bisa atau mungkin tidak didaftar. Ini tergantung pada tingkat kerincian yang dikehendaki. Biasanya, rancangan pertama akan mencakup nama kelas. Kemudian, semakin diperlukan definisi, atribut dan operasi ditambahkan ke rancangan kedua dan seterusnya sampai proses rancangan selesai. Pendekatan untuk pemodelan ini sama dengan pendekomposisian fungsional dari pendekatan rancangan terstruktur, dimana rincian terus ditambahkan apabila fakt-fakta tambahan diperoleh.

Mendaftar Atribut

Ketika atribut-atribut ditambahkan, mereka merepresentasikan properti atau sifat obyek-obyek individual, seperti Name, Weight, Color, Amount, Model, atau Style. Sebagai contoh, CAR adalah suatu obyek; Color adalah atribut dari obyek CAR tersebut. Name, Address, dan Balance, dalam contoh perbankan, adalah atribut dari obyek CHECKING ACCOUNT. Setiap atribut mempunyai nilai untuk setiap kejadian obyeknya. Sebagai contoh, atribut Name mempunyai nilai Mary Doakes, nilai Address-nya adalah 24 Morningstar Lane, dan nilai Balance-nya adalah 1500.00. Kelas tersebut adalah CHECKING ACCOUNT; kejadian obyeknya (yakni, obyek individual) adalah Mary Doakes. Kejadian obyek yang berbeda mungkin mempunyai nilai yang sama atau berbeda untuk atribut tertentu. Sebagai contoh, John Doakes adalah kejadian obyek yang lain dari subkelas CHECKING ACCOUNT tersebut, yang mempunyai nilai Address 24 Morningstar Lane, yakni nilai Address yang sama untuk kejadian obyek Mary Doakes, dan nilai Balance-nya 500.00. Terry Parson adalah kejadian obyek yang lain dari subkelas CHECKING ACCOUNT; ia mempunyai nilai Address 214 Culver Street, dan sebagainya.

Mendaftar Operasi dan Metode

Operasi-operasi didaftar dalam wilayah terendah ketiga pada kotak kelas. Operasi adalah fungsi atau transformasi yang bisa diterapkan pada atau oleh obyek dalam suatu kelas. Open, Deposit, Withdraw, dan Close adalah operasi-operasi pada subkelas CHECKING ACCOUNT. Semua kejadian obyek dalam kelas membagi (menggunakan bersama) operasi yang sama.

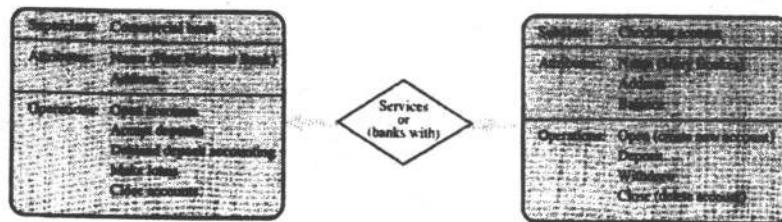
Operasi yang sama bisa diterapkan pada kelas-kelas yang berbeda, yang ini merupakan karakteristik *polimorfis* (polymorphic) dari pendekatan berorientasi obyek. Karakteristik polimorfis ini berarti bahwa operasi yang sama mengambil bentuk yang berbeda dalam kelas yang berbeda.

Metode (yakni, kode program) adalah pengimplementasian suatu operasi untuk suatu kelas. Sebagai contoh, subkelas CHECKING ACCOUNT mempunyai suatu operasi, Deposit. Metode yang berbeda diimplementasikan pada Deposit oleh teller atau oleh transfer dana elektronis. Kedua metode tersebut secara logis sama-sama menjalankan tugas yang sama, yakni mendepositkan dana ke checking account. Namun demikian, setiap metode diimplementasikan oleh set kode pemrograman berorientasi obyek yang berbeda.

Memodel Relasi Diantara Obyek dan Kelas

Relasi mendeskripsikan asosiasi antara kelas dan obyek. Sebagai contoh, Mary Doakes menyimpan uang pada (bank with) First National Bank, seperti terlihat pada Gambar 2.18. Relasi pada umumnya bersifat bidireksional (dua arah). Nama (atau kata kerja) dari relasi biasanya membaca dengan arah tertentu, namun relasi tersebut dapat dilintaskan dengan arah satunya. Sebagai contoh, "bank with" mengkoneksikan First National Bank dengan subkelas CHECKING ACCOUNT-nya. Kedua arah tersebut sama-sama bermakna: mereka mengacu pada relasi pokok yang sama. Hanya nama relasi yang akan membentuk atau menentukan arah tersebut. Walaupun relasi dimodel secara dua arah, mereka tidak harus diimplementasikan dalam dua arah.

Relasi antara obyek berarti bahwa obyek dapat mengirimkan pesan dari satu ke lainnya. Biasanya, pesan juga bersifat dua arah. Sebagai contoh, suatu pesan bisa berupa GetMaryDoakesAccountBalance. Ia menampilkan jumlah uang yang saat itu ada dalam rekening cek (checking account) Mary Doakes.



Gambar 2.18 Relasi kelas obyek.

Memodel Pewarisan

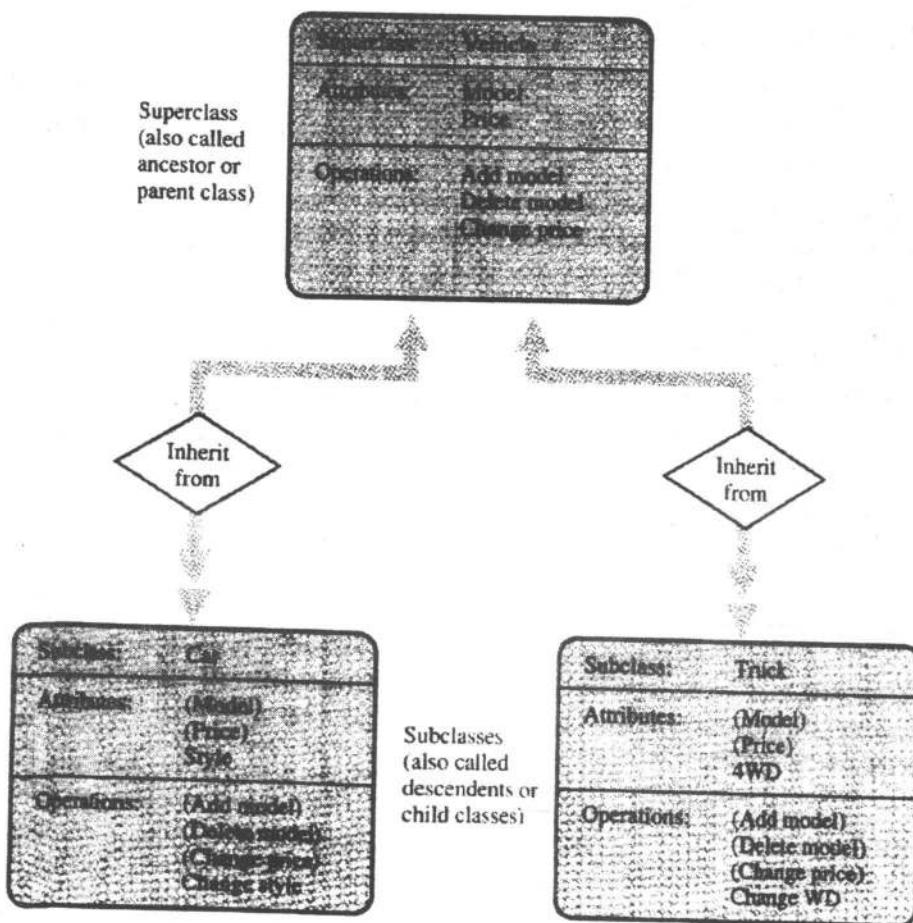
Segala operasi pada superkelas dapat diterapkan pada segala kejadian subkelas. Setiap subkelas tidak hanya dapat mewarisi semua fasilitas dari superkelasnya, namun juga dapat menambahi sendiri atribut dan operasi. Ini adalah suatu fasilitas pewarisan yang disebut ekstensi (perluasan). Sebagai contoh, Gambar 2.19 menunjukkan superkelas VEHICLE. Atribut dan operasi yang diwarisi oleh kelas CAR dan TRUCK dari VEHICLE ditunjukkan sambil lalu atau dalam sisipan.

Disamping menyederhanakan rancangan, keuntungan pokok pewarisan adalah bahwa ia memberi kode yang bisa digunakan kembali. Sebagai contoh, kode yang mengimplementasikan operasi Add Model dituliskan sekali dan diwariskan kapan saja diperlukan. Beberapa bahasa OOP memberikan dukungan yang kuat untuk pewarisan. Pada kenyataannya, pewarisan adalah sinonim dengan penggunaan kembali kode, sepanjang komunitas OOP dikaitkan.

Tugas umum pemodelan berorientasi obyek adalah untuk mengelompokkan bersama kelas-kelas yang sama dan menggunakan kembali kode yang umum. Ada dua macam penggunaan kembali kode: membagi (menggunakan bersama) kode yang baru ditulis di dalam suatu proyek dan menggunakan kembali kode yang dituliskan sebelumnya pada proyek baru. Rancangan berorientasi obyek yang dimodel secara lengkap menunjukkan dimana kita bisa menggunakan kembali kode, guna menghindari pengulangan usaha pemrograman. Dalam beberapa hal, kode juga bisa tersedia dalam perpustakaan kelas dari implementasi berorientasi obyek yang lalu, yang mana ia bisa digunakan kembali atau sedikit dimodifikasi untuk mencapai tingkah laku yang dikehendaki. Beberapa bahasa OOP menawarkan perpustakaan kelas ekstensif.

Menggabungkan (Memasukkan) Faktor Rancangan MURRE dalam Rancangan Perangkat lunak Berorientasi Obyek

Faktor-faktor rancangan MURRE — kemampuan pemeliharaan, kemampuan penggunaan, kemampuan penggunaan kembali, reliabilitas, dan kemampuan perluasan (yang kadang-kadang juga disebut extensibility) — bisa diterapkan pada rancangan perangkat lunak berorientasi obyek seperti halnya mereka diterapkan pada rancangan perangkat lunak terstruktur. Modularitas, yang mendukung faktor rancangan MURRE, bisa diterapkan pada rancangan berorientasi obyek. Sebagai contoh, kelas yang sangat kohesif akan lebih dikehendaki. Bentuk kohesi yang paling kurang dikehendaki adalah kohesi kebetulan, dimana obyek-obyek yang sama sekali tidak



Gambar 2.19 Model yang menunjukkan relasi dan pewarisan.

berkaitan dimasukkan bersama ke dalam kelas yang sama. Seperti halnya dengan rancangan perangkat lunak terstruktur, bentuk kohesi yang paling dikehendaki untuk rancangan perangkat lunak berorientasi obyek adalah kohesi fungsional, dimana semua obyek dalam suatu kelas bekerja bersama untuk memberikan beberapa tindakan (tingkah laku) yang terpadu secara baik. Kumpulan obyek spesifik fungsi seperti itu tidak hanya mudah dipelihara, namun reliabilitasnya dapat diuji secara

mudah, kemampuan penggunaan kembalinya bisa ditingkatkan, dan perluasannya, jika diperlukan, akan lebih efisien.

Modul-modul untuk rancangan perangkat lunak terstruktur harus kecil dan bisa dipahami. Begitu pula halnya kumpulan metode. Metode harus menggunakan nama variabel yang bermakna dan standar, menghindari penyingkatan, dan menggunakan kondisi dan tipe data standar. Metode harus bisa dipahami oleh orang lain selain programmer metode itu, dan selang beberapa waktu kemudian, harus juga bisa dipahami oleh programmer. Seperti halnya kode terstruktur, kode berorientasi obyek harus didokumentasi. Dokumentasi metode akan mendeskripsikan tujuan, fungsi, input, dan outputnya, dan juga merupakan deskripsi obyek. Penjelasan internal di dalam metode tersebut harus mendeskripsikan langkah-langkah utamanya.

Encapsulation/penyembunyian (yang juga disebut penyembunyian informasi) melibatkan pembagian aspek-aspek internal suatu obyek yang bisa diakses oleh obyek lain dari atribut internal dan operasi obyek tersebut. Fasilitas ini mencegah terjadinya content coupling. Encapsulation mencegah objek menjadi berhubungan terlalu erat (coupled) sehingga perubahan kecil dalam suatu obyek tidak akan berpengaruh terhadap rancangan berorientasi obyek secara keseluruhan. Rancangan moduler yang baik (loose coupling antara obyek dan obyek yang kohesif secara ketat) mendukung terjadinya encapsulation (penyembunyian).

Namun, ada beberapa ketegangan antara encapsulation dan pewarisan. Seperti yang baru saja kita lihat, encapsulation memerlukan coupling yang sangat longgar, namun pewarisan memerlukan coupling yang lebih rapat. Di satu pihak, lebih disukai kelas-kelas yang loosely coupled. Di pihak lain, pewarisan lebih menghendaki tightly coupled antara superkelas dan subkelasnya, untuk memanfaatkan kesamaan dan penggunaan kembali.

Penyembunyian/encapsulation terlanggar jika kode yang dihubungkan dengan satu kelas secara langsung mengakses atribut dan operasi dari kelas lain. Banyak bahasa OOP menganggap atribut dan operasi bersifat publik (*umum*) dan private (*pribadi*). Atribut *umum* bisa dibaca dan operasi *umum* bisa dieksekusi. Atribut dan operasi *pribadi* tidak bisa dibaca dan dieksekusi. Mereka hanya digunakan secara internal oleh metode-metode lain dari kelas yang sama dan mereka disembunyikan dari kelas-kelas lain, sehingga kelas-kelas lain ini tidak bisa menggunakan mereka.

Pedoman di atas, selain untuk meningkatkan faktor-faktor rancangan MURRE, juga meningkatkan pewarisan kode. Cara umum untuk meningkatkan peluang me-wariskan kode terbagi adalah dengan memfaktor kode umum ke dalam metode tunggal sebagai suatu subroutine yang dipanggil oleh metode lain.

Apa Perbedaan Antara Rancangan Perangkat lunak Terstruktur dan Rancangan Perangkat lunak Berorientasi-Obyek?

Dalam pemodelan rancangan perangkat lunak terstruktur dan rancangan perangkat lunak berorientasi obyek terdapat beberapa kesamaan umum. Keduanya menggunakan alat pemodelan yang sama. Namun demikian, perbedaan pokoknya adalah bahwa pendekatan terstruktur merancang sistem berdasarkan proses dan data, sedangkan pendekatan berorientasi-obyek merancang sistem berdasarkan obyek.

Dalam pendekatan rancangan perangkat lunak terstruktur, bagan struktur menunjukkan proses (atau fungsionalitas). Dalam pendekatan rancangan perangkat lunak berorientasi obyek, model berorientasi obyek menunjukkan kelas-kelas obyek dan hubungan antara obyek dan kelas. Perubahan keperluan pemakai dengan pendekatan berbasis proses terstruktur menyebabkan perubahan proses, bukannya perubahan obyek. Maka, perubahan dalam rancangan berbasis proses relatif lebih sulit dibandingkan perubahan dalam rancangan berbasis obyek, karena perubahan dalam suatu obyek dilakukan dengan menambahkan atau mengubah operasi dan metode pendukungnya, dengan membiarkan struktur obyek itu sendiri tidak berubah.

Rancangan perangkat lunak terstruktur biasanya didasarkan pada cakupan atau batasan sistem, yang mungkin membuatnya sulit untuk memperluas rancangan ke cakupan atau batasan yang baru. Rancangan berorientasi obyek jauh lebih mudah diperluas dengan hanya menambahkan obyek-obyek.

Para pendukung (penganut) rancangan perangkat lunak berorientasi obyek yakin bahwa pendekatan rancangan ini lebih fleksibel untuk diubah dan lebih bisa diperluas. Mereka juga mengatakan bahwa obyek-obyek lebih mudah dipahami oleh para pemakai, karena obyek-obyek merepresentasikan atau mewakili konsep dan sesuatu hal di dunia nyata. Oleh karena itu, para pemakai tidak perlu memahami atribut dan operasi; karena pada kenyataannya, mereka ini selalu tersembunyi dari end-user. Yang harus dilakukan pemakai adalah memahami tingkah laku obyek dan mengetahui pesan apa yang harus dikirim untuk memanggil/meminta tingkah laku yang dikehendaki. Sebagai contoh, tingkah laku (behavior) dipanggil (dibuat terjadi) oleh pesan-pesan, seperti ChangeModel, GetStudent, AssignGrade, ChangePay, DeleteCustomer, dan GetBirthdate. Dengan kata lain, pesan akan memberitahu obyek apa yang akan dilakukan. Bagaimana obyek mengimplementasikan pesan bukanlah menjadi kepentingan pemakai sepanjang hasilnya benar.

Ditengah-tengah pilihan antara pendekatan rancangan perangkat lunak terstruktur dan pendekatan rancangan perangkat lunak berorientasi obyek adalah kemampuan penggunaan kembali (reuseability). Tujuan ideal penggunaan kembali adalah

menempatkan semua obyek yang bisa digunakan untuk mendukung aplikasi rancangan di dalam perpustakaan kelas. Perpustakaan kelas ini didokumentasi dan diterbitkan dengan tujuan agar yang lain tahu bahwa ia ada. Namun karena pendekatan rancangan perangkat lunak berorientasi obyek ini baru, maka masih jauh bagi kita untuk menerapkan perpustakaan besar yang berisi obyek-obyek yang bisa digunakan kembali, khususnya untuk aplikasi bisnis. Juga, perlu dicatat bahwa modul-modul terancang dengan baik yang mengikuti (menganut) pendekatan rancangan perangkat lunak terstruktur juga memberikan peluang untuk penggunaan kembali modul-modul seperti itu.

MELAKSANAKAN PENELUSURAN RANCANGAN PERANGKAT LUNAK

Rancangan perangkat lunak yang didasarkan pada pendekatan terstruktur atau pendekatan berorientasi obyek harus melaksanakan PENELUSURAN RANCANGAN PERANGKAT LUNAK (*software design walkthrough*). Rancangan perangkat lunak yang berisi kelemahan atau kesalahan rancangan, ketidakkonsistenan, dan ambiguitas akan mengalami penyimpangan fungsi dan akan lebih parah dari pada tidak adanya rancangan sama sekali. Oleh karena itu, setelah rancangan perangkat lunak selesai atau lengkap, kita perlu melakukan penelusuran rancangan perangkat lunak untuk memastikan bahwa rancangan tersebut benar dan bahwa ia memenuhi spesifikasi Laporan Rancangan Sistem Terinci.

Apa yang Dimaksud Penelusuran Rancangan Perangkat lunak?

Ada dua variabel utama dalam suatu jenis penelusuran. Yang pertama adalah derajad formalitas atau struktur dari penelusuran tersebut. Jika ia sangat formal dengan suatu kelompok jaminan kualitas yang independen atau kelompok peer yang menjalankan penelusuran, maka pengorganisasian semacam ini kadang-kadang disebut penelusuran terstruktur. Variabel utama kedua adalah pengaturan waktu. Kita dapat melaksanakan penelusuran terstruktur kapan saja selama SDLC atau SWDLC. Sebagai contoh, penelusuran umumnya terjadi setelah tahap analisis sistem, rancangan, dan evaluasi. Beberapa perusahaan juga melaksanakan penelusuran terstruktur baik sebelum dan setelah semua tahap SDLC dan SWDLC. Dalam bab-bab mengenai pengembangan perangkat lunak dalam buku ini, kita memfokuskan pada penelusuran terstruktur setelah perancangan dan pengkodean perangkat lunak. Penelusuran

terstruktur terhadap kode program perangkat lunak seringkali disebut sebagai pengujian kotak putih; ini akan dibahas pada Bab 4.

Penelusuran rancangan perangkat lunak yang dilakukan oleh kelompok jaminan kualitas atau kelompok peer independen mensimulasi cara perangkat lunak tersebut akan bekerja. Simulasi tersebut menunjukkan bagaimana bagian-bagian yang berbeda dari rancangan perangkat lunak berinteraksi. Ia dapat memperlihatkan adanya ambiguitas, redundansi, atau spesifikasi rancangan sistem yang kurang.

Penelusuran rancangan perangkat lunak tidak dilakukan untuk “menangkap (kesalahan) orangnya”, namun untuk menyingkap hal-hal yang tidak diharapkan. Baik titik dimana kesalahan ditimbulkan dan titik dimana kesalahan itu ditemukan adalah titik yang sangat penting dalam pengembangan perangkat lunak. Dua pertiga kesalahan perangkat lunak berasal dari rancangan perangkat lunak, bukannya dari pengkodean perangkat lunak.

Keuntungan yang jelas dari suatu penelusuran rancangan perangkat lunak adalah bahwa kita bisa menemukan secara dini kesalahan rancangan perangkat lunak, sehingga setiap kesalahan bisa dikoreksi sebelum kita menjalankan langkah berikutnya dalam proses pengembangan perangkat lunak.

Bagaimana Penelusuran Rancangan Perangkat Lunak Dilaksanakan?

Penelusuran rancangan perangkat lunak yang ideal diurus oleh manajer meeting atau moderator. Dokumen rancangan perangkat lunak (misalnya, bagan struktur, Bahasa Inggris terstruktur, model berorientasi obyek seperti ERD termodifikasi, tabel keputusan, dan sebagainya) diperiksa secara cermat oleh para pemeriksa (reviewer). Mereka ini bisa jadi para insinyur (yakni, para programmer yang mengetahui prosedur perancangan dan pengujian maupun pengkodean), wakil pemakai, dan analis sistem atau perancang.

Para pemeriksa harus mempunyai prosedur dan instruksi yang singkat dan jelas. Mereka harus telah membaca dokumentasi rancangan sebelum pertemuan pelaksanaan pemeriksaan dan menanyakan tentang dokumentasi itu saat meeting. Pada waktu meeting, moderator maupun pencatat tidak boleh berkomentar mengenai rancangan perangkat lunak. Moderator hanyalah menjalankan meeting itu. Untuk bisa melakukan hal ini, ia harus menjalankan beberapa tugas, yang meliputi penetapan peraturan meeting, penyusunan waktu dan tempat meeting, mengenali para pemeriksa dan para perancang perangkat lunak, menghentikan interupsi, mencatat jalannya pemeriksaan, dan menyiapkan laporan rekapitulasi. Pencatat menulis semua komentar penting pada lembaran diagram besar sehingga setiap orang dalam

meeting itu dapat melihatnya secara mudah. Komentar-komentar ini digabungkan dengan hasil penelusuran rancangan perangkat lunak. Dan, hasil penelusuran rancangan perangkat lunak dituangkan dalam Laporan Penelusuran Rancangan Perangkat lunak, seperti terlihat pada Gambar 2.20.

Pada akhir penelusuran rancangan perangkat lunak, tim pemeriksa harus memutuskan apakah:

- ◆ Menerima rancangan tanpa modifikasi lagi.
- ◆ Menolak rancangan karena adanya kesalahan utama.
- ◆ Menerima rancangan tersebut dengan syarat.

Dengan kata lain, suatu keputusan dibuat untuk melepaskan rancangan tersebut ke tahap pengkodean atau mengirimkannya kembali ke tahap rancangan ulang. Apabila keputusan telah dibuat, semua anggota tim penelusuran rancangan perangkat lunak harus tanda tangan, yang hal ini menunjukkan bahwa mereka telah berpartisipasi dalam penelusuran tersebut dan bahwa mereka telah setuju dengan penemuan-penemuan tim penelusuran. Setelah beberapa waktu kemudian, tim penelusuran tersebut akan melakukan penelusuran follow-up (tindak lanjut) guna memastikan bahwa semua kesalahan telah dikoreksi dan bahwa tak ada satupun yang telah diabaikan.

Rancangan perangkat lunak tidak boleh dianggap selesai sampai ia telah disetujui oleh tim penelusuran rancangan perangkat lunak. Beberapa pertemuan penelusuran bisa diselenggarakan sebelum persetujuan tersebut selesai dibuat.

TINJAUAN SASARAN BELAJAR UNTUK BAB INI

Tujuan utama bab ini adalah untuk memungkinkan setiap siswa mencapai empat sasaran belajar penting. Sekarang kita akan meringkas tanggapan terhadap sasaran belajar ini.

Sasaran belajar 1:

Menjelaskan alasan pelaksanaan tahap rancangan perangkat lunak.

Semakin banyak waktu dan usaha yang dicurahkan untuk rancangan perangkat lunak, maka akan semakin baik kualitas perangkat lunak tersebut. Jika rancangan perangkat lunaknya benar, maka peluang untuk mengembangkan program perangkat

LAPORAN PENELUSURAN RANCANGAN PERANGKAT LUNAK

Identifikasi Penelusuran

Proyek:	EIS	Penelusuran nomer:	W104
Tanggal:	MMDDYY	Lokasi:	Ruang 302
		Waktu:	9:00

Identifikasi Perangkat lunak

Materi yang Diperiksa: Rancangan rinci Modul untuk EIS
 Perancang Perangkat lunak: Herb Miller

Materi yang Diperiksa

1. Rancangan terinci Modul A, B dan C.
2. Bahasa Inggris terstruktur untuk modul-modul

Tim penelusuran

Nama	Tanda tangan
1. Anne Graham, Moderator	1. _____
2. Jim Kincaid, Recorder	2. _____
3. Julie Byars, Software Engineer	3. _____
4. Dana Sellers, Software Engineer	4. _____
5. Felix Garcia, User Representative	5. _____
6. Patsy Matusz, Systems Analis	6. _____

Penilaian Rancangan Perangkat lunak:

Diterima: sebagai _____ dengan modifikasi minor _____
 Tidak diterima: revisi mayor _____ revisi minor

Daftar Kesalahan

1. Rumus untuk kuantitas pesanan ekonomis ditunjukkan sebagai $Q = \sqrt{4yspc}$;
 ia seharusnya $Q = \sqrt{2yspc}$
2. Statemen Bahasa Inggris terstruktur:
 "Jika penjualan divisi 2 lebih besar dari pada 50.000 unit, cetaklah laporan kekecualian."
 Ia seharusnya:
 "Jika penjualan divisi 2 kurang dari 30.000 unit atau lebih besar dari pada 50.000 unit, maka cetaklah laporan kekecualian."

Saran-saran:

1. Obyek B bisa ditingkatkan dengan membaginya ke dalam dua modul. Satu modul akan menghitung ramalan penjualan; modul satunya akan menghitung kuantitas pesanan ekonomis.
2. Obyek A, yang menginput dan mengedit data inventori, tidak diperlukan.
 Obyek Z berada dalam perpustakaan kelas bahasa OOP yang sedang kita periksa.

Gambar 2.20 Laporan Penelusuran Rancangan Perangkat lunak.

lunak berkualitas tinggi akan meningkat. Jika rancangan perangkat lunak belum disiapkan, kesalahan belum bisa diketahui sampai program tersebut dikonversi ke operasi. Pengoreksian kesalahan selama jauh lebih mahal dari pada menciptakan rancangan perangkat lunak yang benar pada waktu pertama kali.

Sasaran belajar 2:

Menjabarkan pendekatan rancangan perangkat lunak terstruktur.

Elemen-elemen pokok pendekatan terstruktur adalah:

- ◆ Modularitas
- ◆ Kendali top-to-bottom
- ◆ Konsepsi kendali urutan, seleksi, dan repetisi.

Modul harus sangat independen dan bersifat spesifik-fungsi. Modul seperti ini dikatakan terpasangkan secara renggang dan sangat kohesif. Modul-modul superior memanggil modul-modul subordinat melalui mekanisme seperti CALL atau PERFORM.

Modul tidak boleh berisi lebih dari 50 LOEC. Pemecahan fungsi keputusan antara modul-modul harus dihindari. Biasanya, modul subordinat tidak boleh dipanggil oleh lebih dari tiga modul superior, dan modul superior tidak boleh memanggil atau meminta lebih dari lima modul subordinat. Jika pedoman rancangan ini dipatuhi, maka rancangan yang dihasilkan umumnya akan terbentuk dari jumlah modul yang optimum.

Semua alat yang dibahas dalam buku ini dapat digunakan untuk membantu para perancang perangkat lunak terstruktur. Dua alat permodelan populer adalah bagan struktur dan Bahasa Inggris terstruktur. Bagan struktur ditransformasi dari diagram arus data. Ia menunjukkan setiap modul yang membentuk rancangan perangkat lunak. Bahasa Inggris terstruktur memberikan spesifikasi yang akan diikuti atau dipatuhi oleh para pengkode. Dalam beberapa contoh, tabel keputusan, pohon keputusan, dan persamaan untuk algoritma khusus digunakan untuk melengkapi Bahasa Inggris terstruktur.

Sasaran belajar 3:

Menjabarkan pendekatan rancangan perangkat lunak berorientasi-obyek.

Model rancangan perangkat lunak berorientasi obyek dapat diabstraksi dari DFD, STD, kamus data, dan model data logis. Rancangan perangkat lunak berorientasi obyek yang dihasilkan terkomposisi dari obyek, kelas, dan hubungan-hubungan.

Variasi dari ERD berfungsi sebagai alat pemodelan berorientasi obyek. Simbol utamanya adalah kotak kelas yang berisi tiga wilayah:

- ◆ Nama kelas
- ◆ Daftar atribut
- ◆ Daftar operasi.

ERD menunjukkan relasi kotak-kotak kelas tersebut.

Sasaran belajar 4:

Menjabarkan cara melaksanakan penelusuran rancangan perangkat lunak.

Biasanya, tim penelusuran rancangan perangkat lunak terkomposisi atas anggota-anggota dari kelompok jaminan kualitas. Idealnya, pemeriksaan rancangan perangkat lunak harus diorganisasi oleh seorang moderator dan dicatat oleh pencatat (recorder), yang keduanya tidak boleh ikut memeriksa dokumentasi rancangan. Dokumentasi rancangan diperiksa oleh para anggota tim pemeriksa lain sebelum meeting pemeriksaan rancangan perangkat lunak. Para perancang diberi kesempatan untuk merespon atau menanggapi dan memperjelas dokumentasi rancangan mereka. Setelah meeting, para peserta diberi laporan. Semua kesalahan dikoreksi oleh para perancang dan diverifikasi oleh para pemeriksa pada meeting berikutnya. Pengkodean rancangan perangkat lunak tidak boleh dilakukan sampai rancangan perangkat lunak disetujui oleh tim pemeriksa rancangan perangkat lunak.

DAFTAR PERIKSA RANCANGAN PERANGKAT LUNAK

Berikut ini adalah urutan cara melakukan rancangan perangkat lunak. Tujuannya adalah untuk mengingatkan anda tentang aspek-aspek pokok rancangan perangkat lunak.

1. Kenali pentingnya rancangan perangkat lunak dan membuat komitmen atau ketetapan pokok untuk tahap SWDLC.
2. Pahami keperluan pemakai dan rancangan sistem secara penuh seperti yang ditetapkan dalam Laporan Rancangan Sistem Rinci dan yang dimodel oleh alat pemodelan seperti DFD, STD, dan semacamnya. Sebagai contoh, DFD rinci menunjukkan apa yang harus dilakukan. STD menjelaskan kapan hal tersebut dilakukan. Bagan struktur, Bahasa Inggris terstruktur, pohon keputusan, tabel keputusan, persamaan, dan model ERD termodifikasi, pada tingkat rancangan perangkat lunak, menunjukkan bagaimana hal tersebut dilakukan.
3. Abstraksilah rancangan perangkat lunak dari rancangan sistem dengan menggunakan satu atau kombinasi alat pemodelan, seperti bagan struktur, Bahasa Inggris terstruktur, diagram Jackson, diagram Warnier-Orr, ERD termodifikasi, tabel keputusan, pohon keputusan, dan persamaan.
4. Jangan mencoba merancang segala sesuatunya secara presisi pada pertama kali. Seringkali, model anda akan memerlukan revisi-revisi (iterasi). Setiap iterasi mungkin akan memerlukan kerincian yang lebih. Baik rancangan terstruktur maupun rancangan berorientasi obyek akan melibatkan proses yang iteratif dan inkremental (berkembang). Apabila anda berfikir bahwa rancangan tersebut lengkap atau selesai pada satu tingkat abstraksi, tambahkan kerincian lagi dan haluskan lagi sehingga mencapai tingkat kerincian yang lebih kecil. Anda mungkin menjumpai modul-modul baru dan atribut dan operasi baru yang harus ditambahkan ke kelas-kelas. Mungkin kelas baru akan diidentifikasi. Mungkin modul-modul baru akan diperlukan. Sebaiknya kita revisi perangkaian (coupling) antara modul-modul. Pengiterasian sebanyak tiga, empat kali, atau lebih dari itu tidak biasa dalam rancangan perangkat lunak, baik untuk pendekatan terstruktur atau pendekatan berorientasi obyek. Tentunya, karena iterasi berbiaya mahal dan banyak makan waktu, maka selalu diharapkan agar anda bisa menyempurnakannya sesegera atau sedini mungkin.
5. Identifikasilah semua modul atau obyek yang akan berperan dalam aplikasi sistem baru.
6. Spesifikasilah fungsi atau operasi yang dijalankan oleh setiap modul atau obyek dan atribut data yang ada padanya.
7. Tunjukkan hubungan antara modul atau obyek yang mematuhi tujuan modularitas, yakni pemasangan yang renggang dan k Kohesi yang ketat.
8. Lakukan pemeriksaan rancangan perangkat lunak terstruktur terhadap model-model rancangan lengkap guna memastikan bahwa rancangan perangkat lunak tersebut memenuhi spesifikasi rancangan sistem dan dapat digunakan secara benar oleh para programmer.

PERTANYAAN TINJAUAN

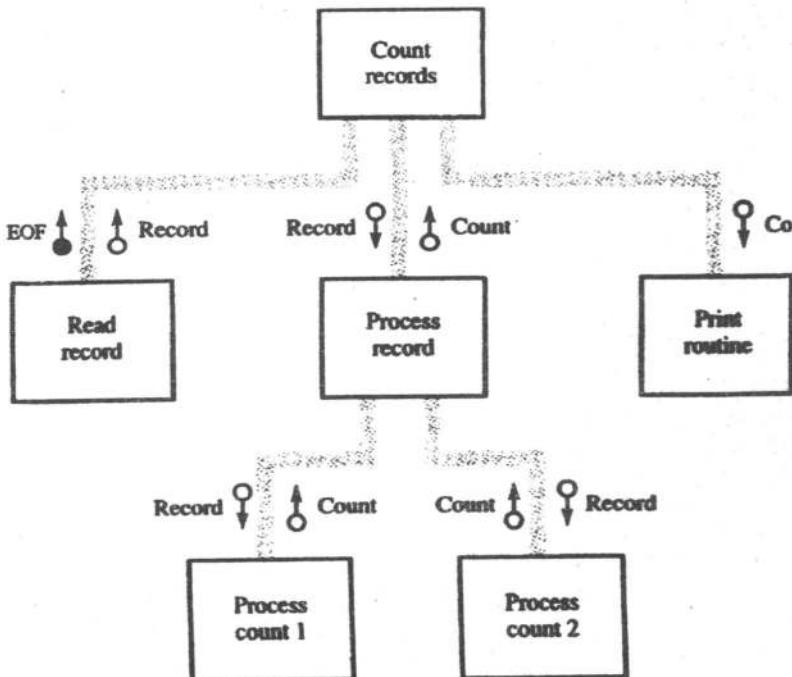
- 2.1 Jelaskan keuntungan mencurahkan sedikitnya 40 persen usaha pengembangan perangkat lunak untuk tahap rancangan.
- 2.2 Sebutkan dua faktor yang mengukur derajad modularitas.
- 2.3 Definisikanlah perangkaian (coupling) dan kohesi.
- 2.4 Jelaskan mengapa modul-modul perangkat lunak yang tightly coupled sulit dikode, diuji, dan dipelihara?
- 2.5 Sebutkan dua perintah umum yang digunakan untuk memanggil atau meminta modul.
- 2.6 Sebutkan dan deskripsikan pedoman-pedoman rancangan moduler.
- 2.7 Tujuan utama rancangan terstruktur adalah modularitas. Dapatkah rancangan perangkat lunak mempunyai terlalu banyak modul? Jelaskan.
- 2.8 Jelaskan mengapa program-program seperti spaghetti yang besar sulit dikode, diuji, dan dipelihara?
- 2.9 Apa peranan bagan struktur dalam rancangan perangkat lunak terstruktur? Apa peranan Bahasa Inggris terstruktur? Mengapa tabel keputusan, pohon keputusan, dan persamaan kadang-kadang diperlukan sebagai alat rancangan pelengkap?
- 2.10 Sebutkan aturan-aturan untuk menulis Bahasa Inggris terstruktur.
- 2.11 Jika diagram arus data digunakan sebagai alat rancangan sistem, mengapa ia ditransformasi ke dalam bagan struktur?
- 2.12 Definisikanlah obyek, kelas, dan hubungan.
- 2.13 Jelaskan bagaimana pewarisan mendukung kemampuan penggunaan kembali.
- 2.14 Deskripsikan isi suatu kotak kelas yang ditetapkan secara penuh.
- 2.15 Jelaskan cara untuk mencapai faktor-faktor rancangan MURRE dengan menggunakan pendekatan berorientasi obyek.
- 2.16 Jelaskan penyembunyian (encapsulation) dan hubungannya dengan fasilitas pribadi dan publik dari rancangan berorientasi obyek.
- 2.17 Jelaskan kesamaan antara pendekatan terstruktur dan pendekatan berorientasi obyek. Jelaskan perbedaan kedua pendekatan tersebut.
- 2.18 Jelaskan mengapa memperbaiki kesalahan selama tahap pemeliharaan akan lebih sulit dan berbiaya mahal dari pada selama tahap rancangan?
- 2.19 Apa tujuan pemeriksaan rancangan perangkat lunak? Jelaskan bagaimana pemeriksaan seharusnya dilakukan.

- 2.20 Dari mana mayoritas kesalahan perangkat lunak berasal?
- 2.21 Jelaskan fungsi moderator, pencatat, dan pemeriksa.
- 2.22 Deskripsikan elemen-elemen yang membentuk Laporan Pemeriksaan Rancangan Perangkat lunak.
- 2.23 Kapan suatu rancangan perangkat lunak dinyatakan lengkap dan siap untuk dikode?

SOAL SPESIFIK BAB INI

Soal-soal berikut ini memerlukan jawaban pasti yang didasarkan secara langsung pada konsep dan teknik yang dikemukakan dalam teks ini.

- 2.24 Bagian dari suatu program mengeksekusi suatu loop (putaran). Kondisi EOF diuji, dan jika kondisinya false (salah), maka loop ini akan dihentikan. Di dalam loop tersebut, jika SALES-CODE sama dengan TRANSCODE, maka harga penjualan akan dihitung, INV-UPDATE akan dijalankan, dan record lain akan dibaca. Jika SALES-CODE tidak sama



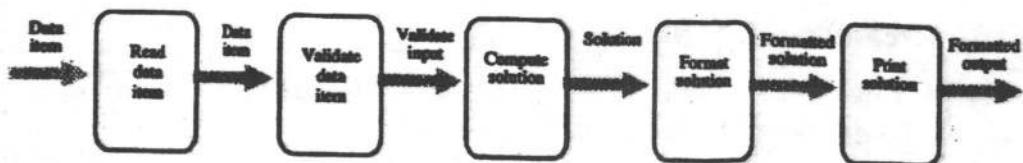
dengan TRANS-CODE, akan ditampilkan "No-Sale", dan record lain akan dibaca.

Ditanyakan: Tulislah Bahasa Inggris terstruktur untuk proses ini.

- 2.25 Lihatlah bagan struktur pada halaman 105.

Ditanyakan: Tulislah Bahasa Inggris terstruktur (atau pseudocode) yang menentukan bagan struktur ini secara lebih rinci.

- 2.26 Pelajarilah diagram arus data berikut ini:



Ditanyakan: Diagram arus data tersebut akan terdiri dari sembilan modul apabila ia ditransformasikan ke bagan struktur. Modul puncak (paling atas) adalah nama transformasi sentral. Tiga modul antara (menghitung) membaca data, memproses solusi, dan menulis. Modul-modul bawah membaca dari file input, memvalidasi data, dan menghasilkan flag EOF, memformat solusi, dan menulis ke file output. Gambarlah bagan struktur modul-modul ini, dimana arus data diantara mereka telah terancang secara tepat.

- 2.27 Periksalah kemampuan terbaca dan hirarki instruksional instruksi berikut ini.

```

ORDER-ENTRY
FOR ALL ORDERS
ENDIF'
IF CUSTOMER-NUMBER IS VALID
GET CUSTOMER-RECORD
ENDFOR
ELSE DISPLAY '' INVALID-RECORD
CALCULATE ORDER-DETAILS
EXIT ORDER-ENTRY
WRITE ORDER-RECORD
    
```

Ditanyakan: Dengan menggunakan pengindenan, tingkatkan kemampuan terbaca dan perjelaslah hirarki instruksional untuk set instruksi ini.

- 2.28 Aplikasi perangkat lunak daftar gaji pada tingkat 0.0 memanggil modul-modul untuk membaca data daftar gaji (1.0), memproses record daftar gaji (2.0), dan mencetak cek daftar gaji (3.0). Proses-proses lain meliputi:

membaca time ticket (1.2), membaca file daftar gaji (1.3), menghitung pembayaran kotor (2.1), menghitung upah straight time (2.2.1), menghitung lembur (2.2.2), mencetak register daftar gaji (3.1), dan mencetak cek pembayaran (3.2).

Ditanyakan: Gambarlah bagan struktur untuk sistem yang dideskripsikan di atas.

2.29 Tabel berikut ini berisi daftar modul dan prosedur mereka.

Modul	Prosedur	Jenis Coupling
A	Modul ini mentransfer record dengan dua belas field ke modul yang memerlukan tiga dari field-field ini.	
B	Modul ini mengubah statemen-statemen dalam modul lain	
C	Modul ini berisi tiga klausa REDEFINES	
D	Modul ini mengirimkan INTEREST RATE yang tepat ke modul kalkulasi pinjaman	
E	Modul ini mengirimkan pesan EOF ke modul lain	

Ditanyakan: Lengkapilah tabel tersebut dengan memasukkan jenis pemasangan yang sesuai.

2.30 Tabel berikut ini berisi daftar modul dan prosedur mereka.

Modul	Prosedur	Jenis Kohesi
A	Menerima transaksi, mengedit, dan mengembalikannya	
B	Dengan menggunakan nomor pekerja, ia akan menentukan nama pekerja dan data lain pekerja, dan mengembalikan item-item data ini	
C	Menerima data, menuju ke proses, mencocokkan kode, menjalankan utiliti, mengupdate master, dan memproses routine audit.	
D	Mengupdate jadwal penagihan mingguan atau faktur mingguan, tergantung pada switch-nya	
E	Mengakses transaksi, mengedit, menerima record master, mencocokkan kode	

F Menginisialisasi, membaca, mengedit, menulis, dan menutup file dalam pesanan itu

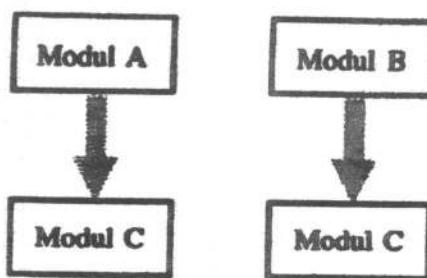
G Menhitung pembayaran kotor

Ditanyakan: Lengkapilah tabel ini dengan memasukkan jenis kohesi yang sesuai.

- 2.31 Modul A berisi dua fungsi. Ia dipanggil oleh lima modul.

Ditanyakan: Buatlah bagan struktur yang menunjukkan pengaturan calling (pemanggilan) yang lebih baik.

- 2.32 Modul A memanggil modul C, dan Modul B memanggil Modul C, seperti ditunjukkan di bawah ini.



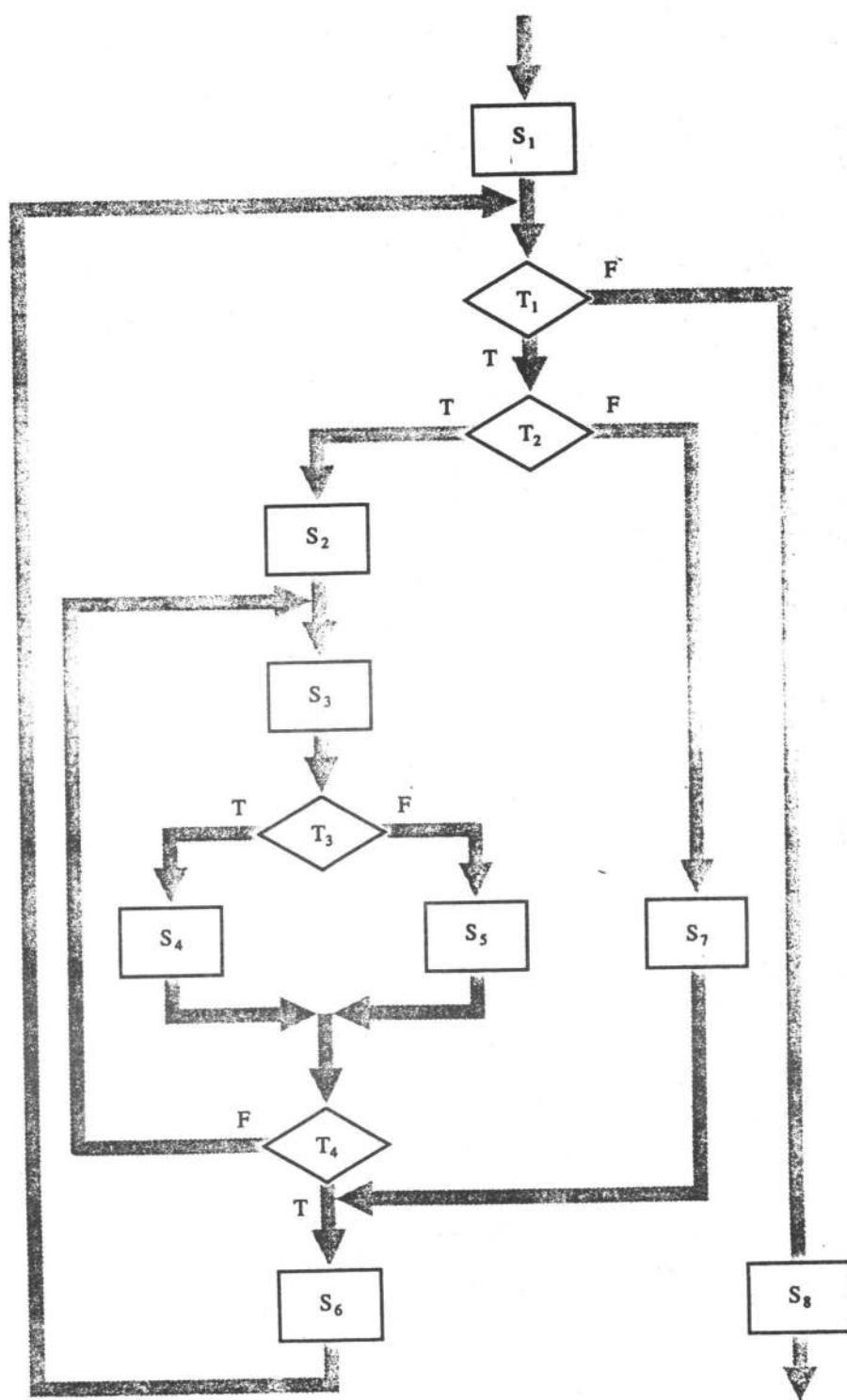
Ditanyakan: Gambarlah bagan struktur yang menunjukkan pengaturan pemanggilan yang lebih baik.

- 2.33 Anda sedang merancang aplikasi perangkat lunak untuk memverifikasi pesanan pelanggan. Nomor pelanggan akan mengakses record kredit. Pelanggan bisa mempunyai atau tidak mempunyai record. Data ini dikembalikan ke modul verify order. Kemudian dilakukan inquiry ke modul authority yang mengkalkulasi digit pengecekan sendiri, yang mengecek batasan kredit, dan yang menggenerasi nomor otorisasi kredit. Respon dari modul authority dikembalikan ke modul verify order. Modul verify order mentransmisi respon ke modul respon yang ada, yang memformat respon tersebut dan mentransmisi respon itu.

Ditanyakan: Rancanglah bagan struktur untuk aplikasi ini, yang memberikan label pada setiap modul dan menyampaikan data diantara modul-modul itu. Rancanglah bagan struktur menurut prinsip pemasangan dan kohesi yang baik.

- 2.34 Pelajarilah flowchart terstruktur pada halaman berikut:

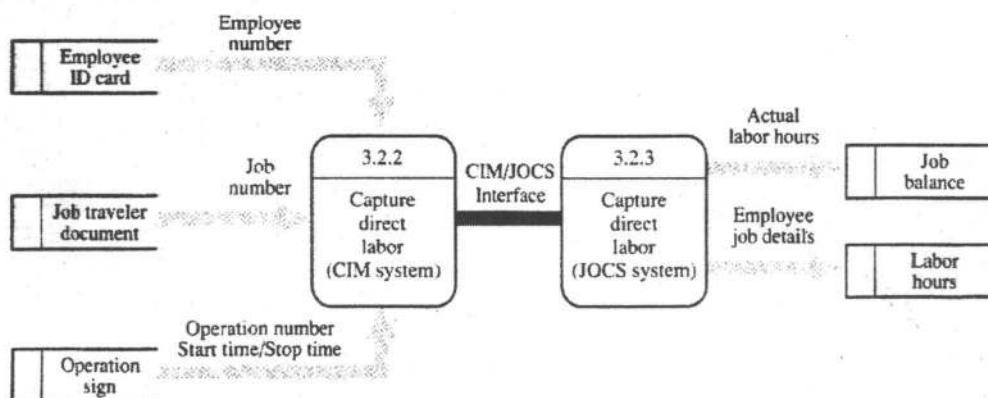
Ditanyakan: Tulislah ekuivalen Bahasa Inggris terstruktur.



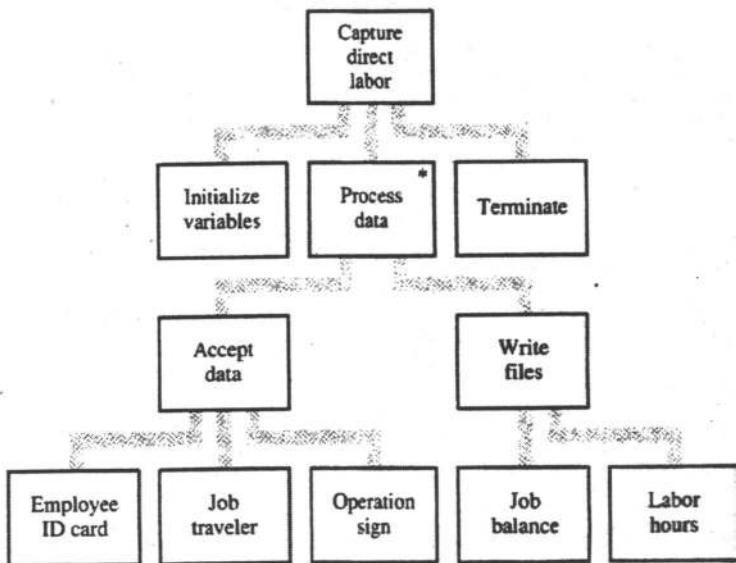
Apabila telah selesai, pekerja tersebut menyampaikan lagi tongkat kode bar melalui ID card-nya, dokumen perjalanan tugasnya, dan tanda nomor operasinya, untuk menangkap nomor ID pekerja, nomor tugas, nomor operasi, dan waktu selesainya mengerjakan operasi tertentu. Diagram arus data yang dirancang selama analisis sistem untuk komponen JOCS ini ditunjukkan pada Gambar 2.21.

Jannis menggunakan diagram arus data untuk menciptakan rancangan sementara yang digambarkan dalam diagram Jackson seperti terlihat pada Gambar 2.22. Selagi mengembangkan diagram tersebut, Jannis menyadari bahwa beberapa persoalan harus dibahas atau dipecahkan sebelum perancangan bisa dilanjutkan. Persoalan pertama berkaitan dengan luas (jangkauan) interface online untuk penangkapan data.

"Haruskah interface tersebut hanya mengumpulkan data dari sistem CIM dan menyimpannya pada perangkat penyimpanan sekunder?" tanya Jannis dalam hati. "atau dari pada demikian, haruskah interface tersebut memberikan umpan balik ke sistem CIM sehingga pekerja yang menggunakan tongkat kode bar dapat diketahui tentang adanya kekeliruan?" Ia terus berfikir mengenai rancangan tersebut dan munculkan kekeliruan pekerja yang mungkin terjadi. "Saya khawatir apa yang terjadi jika pekerja mencoba menjalankan operasi untuk suatu tugas (job) ketika job ini tidak memerlukan operasi itu. Sistem tersebut bisa memberikan umpan balik dengan segera ke pekerja yang menyatakan bahwa operasi yang tidak benar sedang dijalankan. Saya bisa menggenerasi pesan yang memberitahu pekerja bahwa ia telah memasuki operasi yang salah (invalid)." Jalur pemikiran ini bisa menyingkap berbagai kesalahan yang bisa dideteksi sebelum data dimasukkan ke dalam sistem tersebut.



Gambar 2.21 Diagram arus data untuk penangkapan data pekerja sebenarnya JOCS.



Gambar 2.22 Diagram Jackson — penangkapan data pekerja sebenarnya JOCS.

Jannis berfikir mengenai kesalahan-kesalahan lain: "Mungkin pekerja akan mencoba menjalankan operasi untuk suatu job yang telah dilaporkan selesai. Maksud saya, bagaimana jika supervisor membuat kekeliruan dan secara tak sengaja menugaskan suatu operasi yang telah dikerjakan/terselesaikan? Atau, mungkin operasi tersebut belum benar-benar dikerjakan/terselesaikan, namun ia telah memakan banyak waktu dan sistem mengira bahwa operasi tersebut telah selesai. Saya pastikan supervisor ingin mengetahui mengenai situasi ini sebelum pekerja mulai mengerjakan operasi".

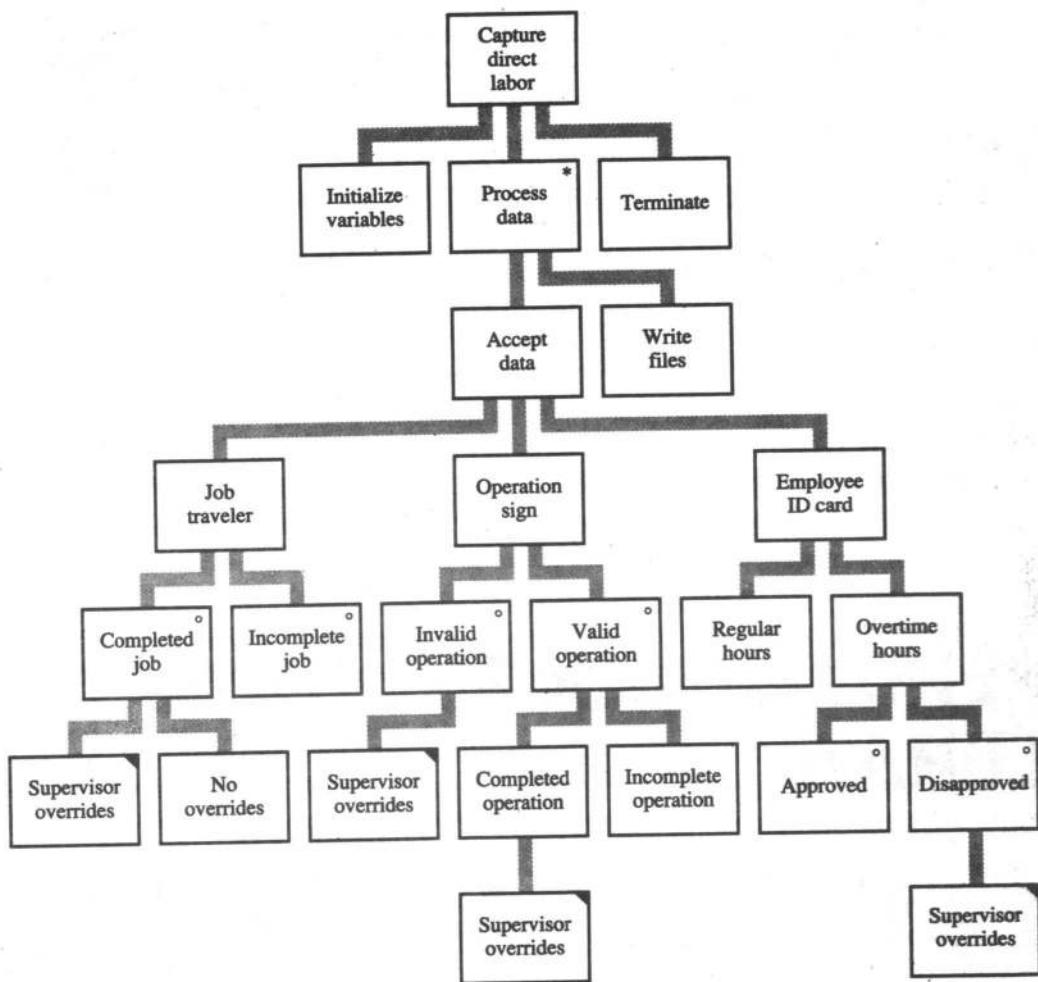
Jenis pengecekan kesalahan online ini akan memerlukan interface lebih luas antara sistem CIM dan aplikasi JOCS. Jika dikehendaki/diperlukan interface yang lebih luas, Jannis pasti akan dihadapkan pada persoalan lain lagi yang harus ia pecahkan. Persoalan ini mengenai metode untuk mengatasi kelemahan potensial. Jannis memikirkan cara untuk membiarkan (memberi ijin) pekerja mengerjakan operasi tersebut atau tidak mengerjakan operasi tersebut: "Jika kesalahan benar-benar terjadi, haruskah saya membiarkan pekerja mengesampingkan sistem tersebut dan menjalankan operasinya? Mungkin saya harus meminta supervisor untuk memasukkan kode supervisor khusus untuk menyetujui setiap operasi yang dikerjakan oleh setiap pekerja." Jannis ragu

dengan gagasannya ini. Ia berpikir, "Persoalan ini sangat penting. Maka sebaiknya saya membicarakannya dengan supervisor manufakturing. Saya tidak boleh memutuskan hal ini begitu saja lalu memberitahu orang-orang manufakturing bahwa ini adalah cara yang akan kita lakukan. Karena kenyataannya saya harus mengetahui lebih banyak mengenai cara sistem CIM berinterface dengan JOCS untuk memastikan bahwa saya dapat melakukan semua pengecekan kesalahan ini.

Ia mengirim Jake pesan E-mail (surat elektronik) yang menyebutkan masalah-masalahnya. Jake membalasnya dengan pesan yang memberi tahu Jannis untuk mengontak Tom Pearson guna meminta penjelasan tentang interface CIM/JOCS yang akan dipakai jika segala pengecekan kesalahan dapat dilakukan antara dua sistem tersebut. Jannis kemudian mengirim Tom pesan E-mail. Tom membalas surat Jannis dengan mengatakan bahwa kecepatan antara dua sistem tersebut memungkinkannya untuk melakukan pengecekan kesalahan data online, dan Tom memberinya informasi rinci mengenai kecepatan transfer data. Jannis mengirim Jake pesan E-mail yang mengatakan bahwa ia perlu berbicara dengan orang-orang manufakturing untuk memutuskan rancangan untuk komponen sistem ini. Jake setuju, pesan E-mail-nya, bahwa ini adalah persoalan yang harus dibahas dengan para pemakai sistem tersebut sebelum rancangan komponen ini bisa diselesaikan.

Jake menetapkan waktu meeting untuk Jannis dengan supervisor manufakturing untuk membahas persoalan tersebut. Meeting antara Jannis dan personel manufakturing ini membantu memperjelas fungsi-fungsi sistem tersebut selama pengembangan kepada setiap orang yang terlibat dalam proyek itu. Meeting tersebut juga membantu membangun hubungan kerja yang erat antara Jannis dan orang-orang yang akan menggunakan perangkat lunak-nya. Jake menginginkan agar para perancang perangkat lunak dan para pemakai bekerja sama untuk menetapkan keperluan perangkat lunak yang pasti.

Setelah meeting dengan personel dari departemen manufakturing, Jannis bisa menyelesaikan rancangan untuk komponen penangkapan online data pekerja sebenarnya. Gambar 2.23 adalah diagram Jackson komponen aplikasi JOCSS ini.



Gambar 2.23 Diagram Jackson — penangkapan data pekerja sebenarnya JOCS.

3

PENGKODEAN PERANGKAT LUNAK

APA YANG AKAN ANDA PELAJARI DALAM BAB INI?

Sebelum mempelajari bab ini, anda diharapkan dapat:

- Membandingkan bahasa generasi-keempat (4GL) dengan bahasa generasi-ketiga (3GL).
- Mendeskripsikan bahasa pemrograman berorientasi objek (OOP), menyebutkan lima bahasa OOP yang penting, dan secara singkat menjelaskan perangkat OOP.
- Menjelaskan bagaimana perangkat bahasa penggunaan-khusus digunakan untuk memungkinkan para pemakai akhir berkomunikasi dengan sistem komputer.
- Mencocokkan/memilih bahasa yang tepat dengan aplikasi rancangan perangkat lunak.
- Menjelaskan pemilihan bahasa yang berkaitan dengan masalah seperti tingkat penggunaan dalam dunia bisnis, expressiveness, kemudahan, portabilitas, kemampuan pemeliharaan, dan kemampuan perluasan.
- Mendeskripsikan dokumentasi perangkat lunak, dokumentasi operasi, dan dokumentasi pemakai.

PENDAHULUAN

Tujuan pengkodean perangkat lunak adalah untuk mengkonversi aplikasi rancangan perangkat lunak ke program perangkat lunak yang terdokumentasi dengan baik (lihat Gambar 3.1). Pilihan perangkat dan bahasa pemrograman komputer yang akan digunakan untuk mengkode aplikasi rancangan perangkat lunak merupakan keputusan yang sangat penting untuk para profesional sistem. Pilihan yang tidak baik dapat menyebabkan timbulnya masalah, sedangkan pilihan yang tepat secara dramatis dapat meningkatkan produktivitas pengembangan, pengujian, dan pemeliharaan.

Apabila keputusan ini telah dibuat, aspek sintaktis (yakni, gramatiskal) bahasa itu dengan sendirinya akan menjadi proses mekanis, dan dalam beberapa kasus akan menjadi proses otomatis jika digunakan generator kode otomatis CASE. Namun demikian, tujuan bab ini tidak untuk membahas sintaks atau pengkodean sebenarnya instruksi bahasa, namun untuk memberikan materi yang akan membantu dalam pembuatan keputusan penting di atas.

Keputusan Pengkodean di Tranquil Industries

"Lingkungan window benar-benar bekerja dengan baik," kata Julie. "Toolkits pengembangan berbasis-window melakukan tugas yang baik dalam mendukung pengembangan aplikasi yang cepat."

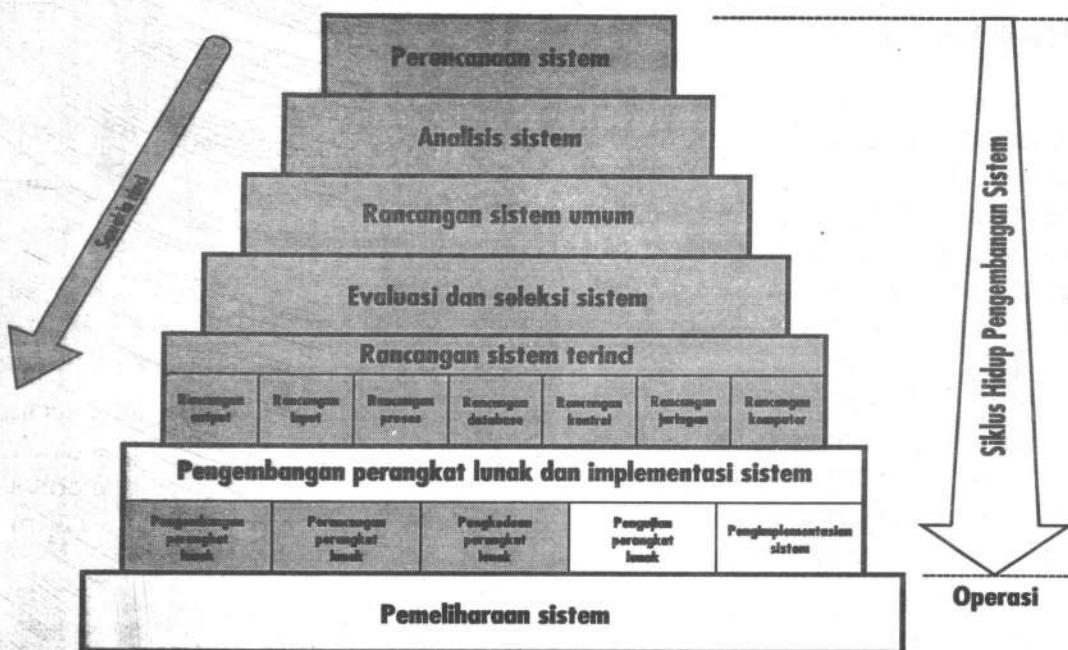
"Saya tidak bisa menyetujuinya lagi," kata Dana. "Kita bisa mengembangkan sistem informasi eksekutif dalam waktu sekitar lima hari. Sebelum menggunakan perangkat pengembangan ini, saya yakin hal itu akan memakan waktu beberapa minggu, mungkin bulanan."

"Set perangkat berorientasi-obyek memungkinkan para eksekutif 'melihat' apa yang mereka inginkan. Mereka mungkin tidak memahami seluk beluk bahasa, namun mereka pasti memahami obyek dan cara meminta obyek tersebut bertindak," tambah Julie.

"Bagaimana tentang perpustakaan kelas bahasa pemrograman berorientasi obyek yang baru?" tanya Jim. "Sudahkan ia memenuhi harapan anda?"

"Ya begitulah," jawab Julie. "Saya terutama puas dengan obyek-obyek, seperti kotak dialog, icon (gambar), dan menu, yang membantu kita mengembangkan secara cepat interface pemakai sehingga para eksekutif bisa memahaminya secara intuitif. Mereka secara mudah bisa mengakses informasi tanpa dibantu."

"Juga, kita membangun beberapa obyek milik kita sendiri dan menyertakan mereka dalam perpustakaan kelas untuk aplikasi rancangan sistem selanjutnya," tambah Dana.



Gambar 3.1 Tahap SDLC dan bab yang terkait dalam buku ini. Bab 3, mempunyai tujuan, untuk memeriksa bahasa perangkat lunak dan mendeskripsikan dokumentasi.

"Bagaimana sistem order-entry berbasis-LAN bisa diadakan?" tanya Jim.

"Kita menyelesaikan rancangan perangkat lunak minggu yang lalu. Semua modul kita sepenuhnya ditetapkan dengan bagan struktur, Bahasa Inggris terstruktur, tabel keputusan, dan persamaan," jawab Julie. "Dan pemeriksaan rancangan perangkat lunak terstruktur kita berhasil dilakukan. Kita berencana untuk mengkodenya dalam COBOL."

"Saya kira anda mau mengkodenya dalam bahasa OOP baru," kata Jim agak terkejut.

"Kita satu-satunya yang ada di Tranquil yang dapat mengkode dalam bahasa OOP, dan kita saat ini mengerjakan sistem truck-dispatching untuk Fred dalam masalah transportasi," jawab Julie.

"Para programmer lain merupakan pengkode-pengkode COBOL yang punya pengalaman tahunan, dan mereka enggan untuk berubah ke bahasa lain. Mereka sangat antisosial untuk mencoba workbench C++ dari kita untuk COBOL," kata Dana.

"Bagaimana bila menggunakan COBOL berorientasi-obyek?" tanya Jim.

"Kita tidak yakin bahwa COBOL berorientasi-obyek cukup matang untuk digunakan saat ini, namun kita sedang mempertimbangkannya," kata Julie.

"Okelah, anda semua adalah ahli pemrograman, namun napaknya kita belum mempertimbangkan bahasa generasi-keempat untuk sistem order-entry berbasis LAN baru kita," kata Jim.

"Kita berencana untuk menggunakan 4GL untuk sistem truck-dispatching," kata Julie.

"Harapan kita adalah bahwa toolkit pengembangan CASE COBOL kita akan bisa meningkatkan produktivitas para programmer COBOL kita," kata Dana.

"Saya pernah membaca artikel menarik tulisan Capers Jones, 'Menggunakan Poin Fungsi untuk Mengevaluasi Alat CASE, Metodologi, Pengalaman Staf, dan Bahasa' pada terbitan Januari/Pebruari 1991 dari *CASE Trends*," kata Jim. "Ia (penulis) tersebut mengatakan bahwa programmer yang belum pengalaman yang menggunakan metode tak-terstruktur, pena dan pensil biasa, dan bahasa tingkat rendah, dapat menghasilkan lima poin fungsi sebulan. Namun demikian, programmer yang berpengalaman yang menggunakan metode terstruktur, perangkat CASE, dan bahasa tingkat tinggi, dapat menghasilkan 100 poin fungsi per bulan. Jika kita dapat melakukan peningkatan yang dramatis tersebut dalam produktivitas pengembangan di Tranquil, maka saya yakin toolkit pengembangan CASE kita akan sukses."

"Akankah COBOL memberi anda peningkatan produktivitas yang demikian?" tanya Jim.

"Pada saat ini, saya belum tahu sebenarnya," kata Dana. "Namun, kita akan mengujinya dengan menggunakan toolkit CASE."

"Tidakkah perangkat CASE secara otomatis menggenerasi kode?" tanya Tom Flanders, asisten Jim Thompson.

"Banyak orang beranggapan bahwa CASE artinya 'perekayasaan perangkat lunak otomatis komputer', bukannya 'perekayasaan perangkat lunak computer-aided (dengan alat bantu komputer)'," balas Dana. Ia bukan perangkat sulap. Set perangkat (tool) tersebut menggenerasi interface kode dan struktur data, termasuk call prosedur dan penyampaian parameter, namun kita masih harus menulis kode prosedural atau pseudocode, seperti Bahasa Inggris terstruktur."

"Akankah sistem order-entry berbasis LAN yang baru tersebut memberi (menyediakan) para pemakai kemampuan inquiry online?" tanya Jim.

"Ya, kita akan menggabungkan beberapa kode bahasa query terstruktur ke dalam program COBOL untuk memudahkan pemrosesan transaksi online dan mendukung query dari sistem manajemen database relasional," kata Julie.

"Jadi hal ini hanyalah masalah penggunaan perangkat bahasa terbaik untuk job yang kita lakukan kan?" tanya Jim.

"Benar," kata Julie, "dan oleh karena itulah toolkit rumah tangga kita terdiri dari borong, palu, obeng, tang, dan kunci Inggris, bukannya hanya satu perangkat. Apabila satu-satunya perangkat anda adalah palu, maka setiap masalah akan nampak seperti paku."

"Oke, bagi saya nampaknya anda telah merencanakan semuanya dengan sangat matang sampai masalah pengkodean rancangan perangkat lunak," kata Jim. "Kabarilah saya terus."

APA PERBEDAAN ANTARA BAHASA GENERASI-KEEMPAT DAN BAHASA GENERASI-KETIGA?

Subbagian berikut ini akan membandingkan Bahasa Generasi-Keempat (4GL) dengan Bahasa Generasi-Ketiga (3GL). Walaupun beberapa 3GL dapat digunakan dalam pemrograman berorientasi-obyek (OOP), kita akan membahasnya lagi ketika kita berbincang mengenai bahasa OOP di bagian lain.

Keunggulan 4GL

Daya tarik 4GL terletak pada adanya kenyataan tentang dirinya bahwa ia menggunakan jalur kode yang lebih sedikit untuk mencapai apa yang diharapkan oleh 3GL yang mempunyai banyak jalur. 4GL juga dirancang untuk memberikan potensi dilakukannya pemrograman oleh end user untuk aplikasi mereka sendiri. Keunggulan-keunggulan lebih spesifik yang ditawarkan 4GL dibahas dalam subbagian berikut ini.

Metodologi Pengembangan

Metodologi pengembangan 4GL tidak selalu menyerupai metodologi pengembangan 3GL. Pengembangan aplikasi 3GL terkait erat (lebih dekat) dengan siklus hidup pengembangan perangkat lunak (SWDLC) dan analis dan rancangan top down yang terstruktur. Beberapa hal yang perlu dipertimbangkan dalam metodologi pengembangan 3GL adalah:

- ◆ "Kerjakan dulu paling awal."

- ◆ “Jangan memulai pengkodean sampai keperluan pemakai ditetapkan sepenuhnya dan sebelum aplikasi dirancang secara detail.”
- ◆ “Semakin lambat anda melakukan (membuat) perubahan dalam pengkodean, maka akan semakin mahal biayanya.”

Sebaliknya, para pengembang/perekayasa 4GL memulai dengan asumsi bahwa para user (pemakai) tidak mengetahui secara pasti keperluan mereka dan tidak mengetahui secara tepat apa yang mereka inginkan. Maka pengembang 4GL memulai dengan prototipe yang beraneka ragam, meskipun prototipe ini sebagian besar didasarkan pada kerja rekaan. Gagasan tersebut adalah untuk memberi pemakai sesuatu, dengan tidak mempermasalahkan perbedaannya dari aplikasi akhirnya. Pendekatan atau cara ini memberi pemakai sesuatu yang nyata yang dapat ia gunakan bekerja dan dapat ia evaluasi. Untuk mengembangkan sistem yang berhasil, harus kita mulai dengan analisis dan proses pemikiran.

Kemudian, pengembang 4GL mulai mengembangkan prototipe yang semakin bisa bekerja mendekati keperluan pemakai. Dengan cara ini, rancangan akan dihasilkan terakhir dalam proses pengembangan. Untuk mengerjakan aplikasi 3GL, pendekatannya adalah dengan menggunakan spesifikasi rancangan sebelum pengkodean dimulai, namun dalam mengerjakan 4GL, programmer 4GL mulai menulis kode untuk menentukan rancangan.

Produktivitas yang Meningkat

Salah instruksi dalam 4GL ekuivalen dengan page atau lebih dari kode 3GL. Lebih dari itu, banyak 4GL berisi fungsi-fungsi spesifik, seperti model statistikal dan matematikal. Kemampuan prototyping dan penggambaran layar membantu menetapkan keperluan pemakai secara cepat dan dengan demikian mempersingkat SWDLC. Kenyataannya, 4GL lebih bermakna atau lebih bisa disebut alat bantu rancangan generasi-keempat atau alat perangkat lunak, bukannya disebut bahasa.

Layanan yang Meningkat

Banyak pemakai tidak puas karena lamanya waktu yang dibutuhkan untuk memenuhi request layanan sistem yang mereka inginkan. Sebagian besar request ini adalah untuk sistem berbasis lokal dan aplikasi ad hoc, yang mereka ini dapat dapat secara mudah dipenuhi oleh 4GL. Aplikasi-aplikasi seperti penjadwalan kasus pengadilan

atau analisis portofolio pinjaman atau sistem penelusuran keluhan serikat buruh bisa didukung oleh 4GL secara mudah.

Partisipasi Pemakai

Umumnya, partisipasi end-user akan meningkatkan aplikasi akhir yang diimplementasikan. Walaupun keterampilan komputer diantara organisasi-organisasi mungkin berbeda, banyak end user cukup terampil dengan komputer sehingga mudah diajari 4GL. Para pemakai seperti ini umumnya senang dan mempunyai motivasi untuk mengetahui apa yang mereka kembangkan dan memelihara sendiri aplikasi mereka atau dengan hanya sedikit bantuan dari profesional sistem. Walaupun 4GL bisa membebaskan end user dari ketergantungannya dari programmer profesional, namun ia tidak bisa membebaskannya dari standart-standart pemrogramann dan pendokumentasian profesional.

Seringkali request ad hoc perlu ditangani dengan segera mungkin agar nampak bernilai bagi para pemakai. Apabila request seperti ini ditempatkan (dijalankan) pada antrian pengembangan 3GL, maka ini akan memperlambat pengembangan dan penginstalasian jauh dari apa yang dikehendaki pemakai, kecuali apabila programmer/analis 3GL didukung oleh set perangkat CASE. Jika demikian, waktu pengembangan 3GL akan sangat dipersingkat.

Keunggulan 3GL

Sekarang kita akan membahas keunggulan bahasa generasi ketiga (3GL) dengan mengacu pada karakteristik berikut ini:

- ◆ Kepadatan
- ◆ Efisiensi mesin
- ◆ Fungsionalitas
- ◆ Kompatibilitas
- ◆ Produktivitas pengkodean
- ◆ Pengujian dan pemeliharaan

Kita akan menggunakan COBOL sebagai model 3GL kita sebab ia sudah meresap dalam dunia bisnis.

Kepadatan

Kebanyakan aplikasi bisnis berisi input dan output yang sangat banyak, dan sebagian besar hal yang panjang lebar dari COBOL ini akibat dari pendeskripsian dan pendokumentasian semua input dan output itu. COBOL menghendaki agar input dan output dideskripsikan secara rinci dan cermat. (Apa yang tidak diberitahukan kepada pembaca mengenai 4GL adalah bahwa jumlah rincian yang cermat mengenai data input dan output harus — lebih baik — didefinisikan atau ditetapkan oleh seseorang di tempat lain. Di tempat lain ini biasanya adalah kamus data, yang bisa diakses oleh 4GL untuk deskripsi datanya).

Efisiensi Mesin

Biasanya, 4GL mengkonsumsi sumber daya yang lebih banyak dan menghasilkan waktu respon yang lebih lamban dari pada program 3GL ekuivalennya. Tambahan run-time yang meningkat terutama disebabkan oleh sifat interpretif dari 4GL. Bahasa ini menerjemahkan kode sumber ke dalam bahasa mesin pada waktu eksekusi. Karena program tertentu akan dieksekusi beberapa kali, hal ini berarti bahwa overhead atau tambahan interpretasinya akan berlipat ganda. Ini juga berarti bahwa akan terjadi waktu respon yang lamban untuk end user dan keluaran.

Fungsionalitas

Bahasa generasi-keempat tidak mengandung ketahanan dan kemampuan yang penuh dari 3GL. Segala aplikasi bisnis dapat dikode dalam 3 GL. Sebaliknya, 4GL biasanya tidak cocok untuk aplikasi yang kompleks dan bervolume tinggi.

Pada kenyataannya, sering terjadi kegagalan yang diakibatkan karena 4GL salah digunakan dalam pengembangan sistem yang kompleks dan bervolume tinggi. Kegagalan klasik 4GL adalah sistem registrasi kendaraan New Jersey. Sistem ini tidak bisa menangani volume transaksi dan gagal, sehingga menghaburkan dana negara bagian puluhan juta dolar. Sejumlah ahli percaya bahwa kasus New Jersey ini adalah contoh berskala besar dari pengalaman-pengalaman yang sama yang sangat sering menimpa orang-orang sistem yang salah menggunakan 4GL. Dalam banyak kasus, pengoreksian masalah seperti itu akan melibatkan pengkodean ulang aplikasi yang gagal tersebut dalam COBOL.

Kompatibilitas

Ketika vendor (pembuat yang menjual) arsitektur komputer melakukan upgrade, hal ini bisa menyebabkan aplikasi 4GL tidak kompatibel dengan lingkungan pengoperasian komputer. Situasi ini memaksa para pemakai menunggu penyesuaian atau pencocokan upgrade dari vendor 4GL untuk mencapai kembali kompatibilitas. Dalam beberapa kasus, upgrade ini mungkin tidak akan terlaksana.

Produktivitas Pengkodean

Pengkodean dengan tangan terhadap 3GL seperti COBOL pasti merupakan proses yang membosankan, namun ada generator kode otomatis CASE yang bisa digunakan untuk menggenerasi secara otomatis kode COBOL untuk menjalankan aplikasi yang terancang dengan baik. Bagan struktur, Bahasa Inggris terstruktur, dan tabel keputusan dikonversi secara langsung ke kode COBOL tanpa perlu pengkodean dengan tangan.

Vendor-vendor perangkat lunak besar telah mengetahui ketahanan COBOL dan oleh karenanya terus menawarkan perangkat pengembangan CASE COBOL. Walaupun para vendor seperti itu terlalu banyak disebutkan disini, namun sebagai contoh vendor perangkat lunak raksasa adalah Microsoft Corporation. Sistem Pengembangan Profesional COBOL berbasis-PC dari Microsoft memberikan atau menyediakan workbench programmer yang terpadu secara penuh, termasuk perangkat pengembangan perangkat lunak seperti compiler, editor, routine diagnostik, pemformat kode sumber (untuk readibilitas), dan interface pemakai grafis (GUI). Alat pemrograman dan pengembangan COBOL ini menghasilkan program yang berjalan dalam berbagai lingkungan, seperti DOS, OS/2, UNIX, MVS, dan VMS. Workbench CASE COBOL juga menghasilkan rancangan layar yang membantu memudahkan pelaksanaan prototyping dan pengembangan aplikasi bersama (JAD). Workbench ini juga memberikan atau menyediakan bahasa campuran yang dapat dipadukan dengan COBOL, seperti bahasa query terstruktur (SQL). Generator kode otomatis CASE sepenuhnya meniadakan manfaat efisiensi pengkodean dari 4GL.

Lebih dari itu, tempat penyimpanan (repository) sentral CASE digunakan untuk menyimpan perpustakaan modul yang memberikan nama, deskripsi, dan prosedur akses untuk modul-modul yang bisa digunakan kembali yang telah dikode dan diuji. Modul-modul ini mungkin bersifat standart diantara pembuat-pembuatnya, seperti routine statistikal tertentu, atau mungkin mereka berlaku untuk fungsi tertentu di

dalam perusahaan, seperti pemrosesan order-entry. Apabila mereka diperlukan sebagai bagian dari program baru, maka modul-modul itu tinggal dipanggil dari perpustakaan modul menurut nama mereka, seperti CALL ORDER-ENTRY.

Pengujian dan Pemeliharaan

Konsep logika tersembunyi — yaitu, instruksi-instruksi 4GL makro yang menggenerasi ratusan atau ribuan instruksi mesin — membuat pengujian dan pemeliharaan sulit dan membosankan. Dengan melakukan sedikit perubahan dalam satu area program, maka akan menyebabkan malfungsi di area lain, dan usaha selanjutnya untuk mengoreksi kesalahan dapat mengakibatkan kesalahan atau kebingungan yang lebih parah.

Kode 3GL nampak jelas bagi penguji dan pemelihara. Jika rancangan terstruktur dianut dan kode didokumentasi dengan benar, program C atau COBOL dapat mudah dibaca dan dipahami. COBOL digembar-gemborkan sebagai bahasa self-documenting.

Jika suatu organisasi memilih bahasa yang tak dikenal dan tak populer, baik 3GL atau 4GL, ia akan kesulitan menemukan programmer untuk mengembangkan, menguji, dan memelihara program. Penggunaan bahasa yang standart dan universal, seperti C atau COBOL, membuat tugas pemeliharaan jauh lebih mudah. Banyak orang yang mengetahui cara mengkode perangkat lunak dalam C atau COBOL. Memang, C dan COBOL dianggap sebagai bahasa standart.

Ringkasan Penjelasan Mengenai 3GL dan 4GL

Karena 3GL dan 4GL saat ini merupakan generasi bahasa dominan yang digunakan untuk perangkat lunak aplikasi, kita akan membahas perbandingannya secara singkat. Gambar 3.2 merekapitulasi beberapa fasilitas pokok 3GL dan 4GL. COBOL 3GL mungkin akan tetap menjadi bahasa yang dominan untuk aplikasi bisnis transaction-intensive, setidaknya untuk dekade yang akan datang meskipun mungkin tidak sampai memasuki abad ke-dua puluh satu. Ada, C, dan Pascal mempunyai banyak pendukung (pengguna), dan bisa kita ramalkan bahwa bahasa ini akan semakin banyak digunakan. Sebagai contoh, Ada dimandatkan oleh Departemen Pertahanan dan Keamanan untuk digunakan sebagai perangkat pengembangan perangkat lunak baru.

BAHASA PEMROGRAMAN BERORIENTASI-OBYEK

“Panacea”

Marilyn Ageloff, CIO, sedang mencoba mengkoordinasikan 16 proyek sistem, yang semuanya di luar jadwal. Sepanjang pagi ia marah-marah dengan para pemakai dan anak buahnya. Ia baru saja kembali dari meeting dengan komite pengarah. Dalam meeting tersebut, komite ini mencoba menjelaskan mengapa investasi \$1,5 juta untuk membangun jaringan baru dan arsitektur komputer belum meningkatkan produktivitas seperti yang diestimasikan.

Don Melanson, salah satu programmer terbaik Marilyn, mengajaknya masuk kantor dan mulai memberitahu dia bahwa ia telah menemukan solusi sempurna atas masalah tersebut.

“Yang perlu kita lakukan adalah beralih untuk menggunakan perangkat bahasa berorientasi-obyek,” kata Don. “Alat ini akan memberi kita obyek, encapsulation, inheritance, polymorphism, kemampuan perluasan, dan ia akan ...”

“Tentu Don, namun pagi ini saya telah marah-marah. Maukah anda menjelaskan mengapa cepat saja manfaat riil dari apa yang akan anda kemukakan? Misalnya, bagaimana teknologi berorientasi-obyek tersebut membantu kita mengatasi backlog kita?”

“Kita dapat melancarkan backlog kita dalam sekejap. Kita dapat menulis program itu kali lebih cepat. Lihat, anda dapat membangun program dengan obyek-obyek yang telah ada, bukannya dengan cara mengkodenya dari scratch (goresan gambar). Jika anda memerlukan obyek baru, anda bisa men-subkelasnya dari obyek yang telah ada. Sekarang, anda tidak membangun rumah dan membuat jendela sendiri. Anda tinggal membeli rumah dan jendela yang sudah jadi. Dan ...”

“Jadi, segala sesuatu yang telah kita pelajari selama tahunan mengenai pendekatan berorientasi-obyek telah bisa kita praktekkann?”

“Ya,” jawab Don. Itu merupakan pergeseran paradigma, suatu perpindahan dari metode pemrograman terstruktur sekarang ke cara yang lebih natural dalam menangani komputer. Suatu perputaran baru secara keseluruhan.”

“Bagi saya masih kedengaran sulit digapai, namun saya akan memberikan peluang untuk diterapkannya teknologi obyek. Bagaimana anda membuat pergeseran paradigma dan menggunakan pendekatan berorientasi-obyek pada proyek analisis portofolionya Jim? Kita tentu saja tidak akan bergeser atau beralih ke OOP dalam waktu singkat. Karena telah ada banyak pengalaman dan keterampilan dalam paradigma terstruktur.”

“Memang saya juga berpendapat demikian,” kata Don. “Saya akan selalu memberitahu anda tentang kemajuan yang saya capai.”

"Baiklah," jawab Marilyn. "Maafkan ketegangan gesekan saya, sayang. Saya akan meeting dengan para auditor untuk menginformasikan rencana perbaikan yang baru."

Beberapa orang yang selama ini telah memahami dan berpengalaman dengan pendekatan berorientasi struktur dalam pengembangan sistem dan perangkat lunak menerima pesan dari para pendukung pendekatan berorientasi-obyek: "Segala sesuatu yang telah anda pelajari dan praktekkan adalah salah." Pesan ini bukan saja terlalu menyakitkan namun juga tidak benar. Yang benar adalah bahwa pendekatan

Ciri	Generasi Bahasa	
	3GL	4GL
Keringkasan	Tidak	Menyesatkan
Efisiensi mesin	Ya	Tidak
Kompilasi	Ya	Sebagian quasi-kompilasi
Fungsionalitas	Lengkap	Terbatas
Kompatibilitas	Ya	Dapat jadi masalah
Portabilitas	Ya	Sebagian
Batch dan real-time	Ya	Sebagian
Bahasa standar	Ya	Tidak
Kebebasan penjual	Ya	Tidak
Pengkodean end-user	Umumnya tidak	Untuk beberapa aplikasi ad hoc
Efisiensi pengkodean	Mungkin superior	ya
Prototipe	Ya	ya
Kendali terpasang	Ya	Sulit
Mendukung SQL	Sebagian	Sebagian
Mudah mendokumentasi	Ya	Tidak
Ketersedian pemrogram	Ya	Sebagian
Kemudahan pemeliharaan	Ya jika terstruktur Tidak jika tak terstruktur	Umumnya tidak

Gambar 3.2 Rekapitulasi perbandingan antara 3GL dan 4GL.

berorientasi-obyek berbeda dalam beberapa hal dari pendekatan berorientasi-struktur, namun, jika keduanya diikuti secara benar, dipastikan akan memberikan hasil pengembangan sistem yang berhasil.

Memang pendekatan berorientasi-obyek tidak memberikan sesuatu yang benar-benar baru. Ia hanya menawarkan set perangkat bahasa OOP yang baik untuk mencapai modularisasi dan reuseability yang baik. Maka seharusnya pesan di atas berbunyi: "Segala sesuatu yang anda ketahui adalah benar dan bermanfaat. Namun ini ada beberapa perangkat bahasa yang bisa membantu anda melakukan segala sesuatunya dengan lebih baik dalam situasi-situasi tertentu."

Bahasa OOP (Object-Oriented Programming) banyak mendukung konsep rancangan berorientasi obyek yang dideskripsikan dalam buku ini. Bahasa OOP dengan perpustakaan kelas yang ekstensif secara dramatis bisa menurunkan atau mengurangi waktu dan biaya SWDLC. Namun demikian, kebanyakan bahasa OOP selalu berubah. Sebagai contoh, C++ secara berkala muncul dengan revisi baru.

Kelompok Manajemen Obyek (OMG) adalah badan penentu standart untuk rancangan dan pemrograman perangkat lunak berorientasi-obyek. Object Management Architecture Guide-nya menetapkan kerangka yang bisa digunakan oleh para vendor untuk mengembangkan lingkungan berorientasi-obyek. Lingkungan ini berisi bahasa OOP, seperti C++, interface pemakai grafis (GUI), dan database berorientasi-obyek. Tujuan OMG adalah untuk menetapkan spesifikasi yang nantinya dapat digunakan oleh para pengembang untuk membangun aplikasi rancangan perangkat lunak. Spesifikasi ini akan bisa berjalan (berlaku) di bawah lingkungan berorientasi-obyek dan dalam berbagai platform komputer.

Sementara 4GL dan khususnya 3GL mengimplementasikan rancangan perangkat lunak berorientasi-struktur, bahasa OOP mengimplementasikan berbagai fasilitas rancangan perangkat lunak berorientasi-obyek yang telah dibahas dalam Bab 2. Larik luas bahasa OOP berfungsi sebagai implemter. Namun demikian, bahasa OOP spesifik bervariasi dalam mendukung konsep rancangan perangkat lunak berorientasi-obyek. Beberapa bahasa, seperti C++ dan Eiffel, mendukung pewarisan yang banyak. Sedangkan sebagian yang lain tidak.

Jenis Bahasa OOP

Ada dua jenis bahasa OOP:

- ◆ Murni
- ◆ Hybrid

Bahasa OOP Murni

Bahasa OOP murni membangun obyek dari scratch. Ia berisi kemampuan berorientasi-obyek yang lebih banyak dari pada yang dipunyai hybrid. Mungkin contoh paling tepat bahasa OOP murni adalah Smalltalk yang dikembangkan oleh Alan Kay pada Xerox Corporation. Karena segala sesuatu yang berkaitan dengan bahasa OOP murni seperti Smalltalk adalah obyek, maka ia memberikan studi yang bermanfaat dalam pengimplementasian OOP lengkap. Namun, kurva pembelajaran Smalltalk cocok bagi programmer yang telah memahami benar pemrograman terstruktur, dan kurva pembelajaran tersebut secara literal akan mengatur programmer yang sedang menulis program non-terstruktur dan yang belum mengenal konsep rancangan berorientasi-obyek.

Bahasa OOP Hybrid

Mungkin aplikasi pendekatan OOP yang lebih umum adalah melalui Bahasa OOP Hybrid. Bahasa OOP hybrid mencangkokkan kemampuan-kemampuan berorientasi-obyek ke 3GL yang terkenal seperti C, COBOL, atau Pascal. Bahasa OOP hybrid yang paling populer, dan umumnya diakui sebagai standart de facto, adalah C++, yang didasarkan pada C.

Keunggulan utama bahasa OOP hybrid adalah bahwa ia memungkinkan programmer yang telah paham dengan base 3GL seperti Ada, C, COBOL, atau Pascal, untuk pindah ke bahasa OOP hybrid sedikit demi sedikit. Sementara Smalltalk dan Eiffel lebih disukai oleh para pendukung bahasa OOP murni berorientasi-obyek, sedangkan C++, Object Pascal, dan Object COBOL lebih disukai oleh para tradisionalis. Agar bisa menguasai pendekatan OOP, para perancang dan programmer harus mampu memahami karakteristik rancangan OOP yang pokok, yaitu:

- ◆ Kemampuan penggunaan kembali
- ◆ Kemampuan perluasan
- ◆ Pesan
- ◆ Obyek dan kelas obyek
- ◆ Encapsulation
- ◆ Pewarisan
- ◆ Polimorfism

Apabila karakteristik rancangan ini telah dikuasai, sintaks bahasa tertentu akan lebih mudah dikuasai. Pada point ini, para profesionalis sistem dapat mempertimbangkan sendiri untuk berusaha memahami pendekatan OOP.

Smalltalk

Smalltalk adalah bahasa OOP murni populer yang pertama. Keberhasilan awalnya menyebabkan terjadinya pengembangan bahasa-bahasa OOP lain. Smalltalk menghasilkan banyak konsep berorientasi-obyek dan terminologi sekarang ini.

Smalltalk terbangun dari dua sua konsep sederhana:

- ◆ Segala sesuatunya diberlakukan sebagai obyek.
- ◆ Obyek-obyek berkomunikasi dengan cara menyampaikan pesan.

Smalltalk sangat bagus untuk pencapaian penggunaan kembali (reuseability) dan kemampuan perluasan (extendability).

Smalltalk adalah bahasa interpreted-based (berbasis interpretasi) yang dikoneksikan ke lingkungan yang berbasis grafis yang secara penuh memadukan window, editing, debugging, compiling, dan graphic. Oleh karena itu, ia memberi perancang/programmer suatu lingkungan pengembangan yang sangat interaktif, yang ini menghindarkan perancang dari delay/keterlambatan siklus edit-compile-link yang terjadi dalam bahasa berbasis-compiler tradisional. Ia mempunyai perpustakaan kelas yang ekstensif, yang dapat mengkontribusi pelaksanaan prototyping suatu aplikasi secara cepat.

Eiffel

Eiffel adalah bahasa OOP murni yang dikembangkan oleh Bertrand Meyer. Program-programnya terdiri dari kumpulan deklarasi kelas yang mencakup metode-metode. Pewarisan yang banyak bisa diperoleh darinya, dan disediakan pula perpustakaan kelas kecil. Eiffel didukung oleh perangkat pengembangan yang sangat bagus. Compiler Eiffel menerjemahkan program sumber ke dalam C. Ia mempunyai fasilitas yang baik untuk encapsulation, dan merupakan bahasa OOP berorientasi-bisnis yang sangat baik.

C++

C++ adalah bahasa OOP hybrid yang dikembangkan oleh peneliti AT&T Bell Lab, Bjarne Stroustrup, sebagai varian berorientasi-obyek dari C. Beberapa orang menamakan C++ sebagai "C with classes". Umumnya, kelemahan utama C++ adalah kompleksitas relatifnya. Yakni, C biasanya merupakan bahasa pilihan para perekayasa perangkat lunak profesional, dan C++ memiliki keruwetan dan kekuatan yang sama.

Dengan C++, beberapa entiti diberlakukan sebagai obyek, dan beberapa yang lain tidak. C++ dikembangkan tidak hanya menambah kemampuan berorientasi-obyek, namun juga untuk mengoreksi beberapa kelemahan C. Karena ia berasal dari C, C++ lebih menjamin kompatibilitas dengan base program C yang telah ada.

C++ tidak menawarkan atau memberikan perpustakaan standart, meskipun beberapa perpustakaan kelas telah dikembangkan oleh berbagai vendor C++. Perpustakaan-perpustakaan kelas yang berbeda ini mungkin tidak kompatibel. Namun demikian, C++ memberikan metode yang baik untuk menentukan akses ke atribut dan operasi dari suatu kelas. Pewarisan (inheritance) memungkinkan para pengembang menggunakan kembali obyek-obyek yang telah teruji dan telah ada untuk menciptakan obyek-obyek terkustomisasi (atau spesifik aplikasi) mereka sendiri. Lebih dari itu, pewarisan yang banyak memungkinkan pengembang menurunkan (mewariskan) sifat-sifat yang lebih dari obyek itu.

C++ adalah bahasa non-proprietary yang didukung oleh para vendor yang besar C++ saat ini merupakan bahasa OOP yang dominan untuk penggunaan umum.

Object Pascal

Object Pascal adalah bahasa OOP hybrid yang diciptakan oleh para pengembang dari Apple Computer bekerja sama dengan Niklaus Wirth, perancang Pascal. Karena Object Pascal lebih bersifat seperti Bahasa Inggris (more English-like) dan menambah beberapa perintah baru dalam pengimplementasian OOP-nya, ia biasanya dianggap sebagai bahasa yang lebih mudah untuk mempelajari OOP dari pada C++. Seperti halnya Smalltalk, Object Pascal mempunyai perpustakaan kelas (class library) yang lebih ekstensif dan lebih kaya. Para perancang/programmer biasanya mulai dengan cara menggunakan kelas yang paling mudah/jelas dalam suatu perpustakaan. Apabila mereka telah semakin terbiasa menggunakan kelas yang mudah ini, mereka sedikit demi sedikit menggunakan kelas yang lebih canggih/rumit. Akhirnya mereka bisa menambah kelas sendiri.

Object Pascal memungkinkan polimorfism tunggal dan pewarisan tunggal. Tidak seperti Smalltalk dan beberapa bahasa OOP yang lain, semua atribut dalam Object Pascal tidak disembunyikan (unencapsulated). Dan tidak seperti halnya C++, semua metode dalam Object Pascal bersifat publik.

Sebagaimana C++, Object Pascal memberikan kompatibilitas dengan program-program Pascal yang telah ada. Program-program ini bisa mempunyai konsepsi berorientasi-obyek yang ditambahkan atau dimasukkan kepadanya. Fasilitas ini menciptakan kurva pembelajaran yang lebih singkat bagi para penganut konsepsi obyek yang telah mengetahui Pascal.

Object COBOL

Sebagian besar sistem bisnis umum sekarang ini menggunakan COBOL tried-and-true. Tak diragukan lagi bahwa milyaran jalur kode COBOL akan dibuang/dihapus. Inilah alasan utama mengapa Object COBOL muncul.

Object COBOL adalah bahasa OOP hybrid yang akan memberikan suatu pintu gerbang ke laris layanan yang saat ini belum siap/ belum bisa diakses dari COBOL tradisional, seperti kemampuan untuk merakit obyek-obyek yang telah dikode dan diuji. Dengan kemampuannya merakit (assemble) obyek-obyek yang telah dikode dan diuji sebelumnya, bukannya mengkode semua fungsionalitas dari scratch, maka Object COBOL akan menyingkat waktu perancangan, pengkodean, dan pengujian dalam persentasi yang besar. Lebih dari itu, pengadopsian Object COBOL akan lebih mudah bagi para programmer COBOL tradisional, dan juga bagi mereka yang merasa lebih suka dengan perintah-perintah seperti Bahasa Inggris.

Perangkat OOP

Vendor perangkat pengembangan dan bahasa memberikan laris toolkit pengembangan berorientasi-obyek yang luas. Sebagai contoh, Smalltalk diadaptasikan untuk bekerja dengan window dan standart windowing. Turbo C++-nya Borland menawarkan lingkungan pengembangan C++ yang sangat baik.

Dengan kelompok seperti Object Management Group (OMG) yang menghasilkan dan mempublikasikan standart-standart untuk perangkat OOP, sistem pengembangan OOP berbasis-window menjadi semakin terstandardisasi. Interface pemakai grafis (GUI) mouse-driven point-and-click mencakup perpustakaan kelas seperti berikut ini:

- ◆ Kotak dialog
- ◆ Menu

- ◆ Tombol
- ◆ Kotak daftar (plus browser/perangkat penglihat obyek dan kelas)
- ◆ File dan editor pemrosesan
- ◆ Compiler
- ◆ Pengujian dan debugger interaktif
- ◆ Generator kode interaktif

Microsoft's Application Factory memberikan sistem pengembangan terpadu yang mencakup:

- ◆ Alat untuk menciptakan dan memanipulasi obyek, termasuk perangkat untuk melihat perpustakaan-obyek dan fasilitas perakitan-obyek.
- ◆ Lingkungan pemrograman visual untuk mengkonstruksi obyek-obyek interface dan aplikasi.
- ◆ Shell untuk mendukung pemrograman berorientasi-obyek maupun pemrograman berorientasi-struktur.

Workbench CASE ini berisi repository (perangkat penyimpanan) sentral yang menyimpan semua rincian yang berkaitan dengan proyek perangkat lunak yang sedang berlangsung.

Objectwork, suatu lingkungan CASE untuk C++, memberikan fasilitas berikut ini:

- ◆ Kompilasi inkremental dan perakitan obyek
- ◆ Akses ke kode menurut/dengan kelas
- ◆ Representasi grafis pewarisan tunggal dan banyak
- ◆ Inspeksi atau pemeriksaan obyek yang memberikan pengawasan dan perubahan obyek-obyek pada saat run time
- ◆ Alat pengujian dan debugging interaktif.

Banyak bahasa OOP membantu mempercepat waktu pengembangan dengan memberikan set perangkat yang memungkinkan perancang/programmer lari dari (escape) siklus edit-link-compile. Lebih dari itu, beberapa vendor perangkat pengembangan memadukan ke paket mereka fasilitas-fasilitas sistem pakar dan hypertext dan hypermedia (atau multimedia).

Dengan diperlengkapi dengan toolkit pengembangan GUI mouse-driven point-and-click, programmer telah menyelesaikan sebagian pekerjaannya dalam bentuk

obyek-obyek yang telah ditetapkan dan diuji sebelumnya. Yang terutama bermanfaat atau membantu adalah obyek-obyek yang digunakan untuk membangun interface pemakai, seperti kotak dialog, menu, icon, tombol, pointer, scroll bar, dan sebagainya. Jika obyek-obyek ini telah siap tersedia, programmer dapat menyelesaikan proyeknya lebih cepat, sebab ia tidak perlu menulis kode yang terlalu banyak. Juga, dengan semakin sedikitnya jalur kode yang ditulis, maka akan semakin sedikit kesalahan (bug) yang akan muncul, sehingga program yang reliabel atau handal akan semakin mudah dibuat.

PERANGKAT BAHASA PENGGUNAAN-KHUSUS

Bahasa-bahasa pemrograman yang dideskripsikan di atas adalah bahasa umum (*generalized*) dan multipurpose yang biasanya digunakan untuk obyek-obyek besar. Walaupun mereka dapat juga digunakan untuk aplikasi penggunaan-khusus, dalam bagian ini kita akan membicarakan sekelompok perangkat bahasa (beberapa orang menganggap mereka sebagai perangkat pengembangan atau alat bantu rancangan) yang dirancang untuk tujuan/penggunaan tertentu (yakni, *ad hoc*). Ada tiga contoh perangkat bahasa penggunaan/tujuan-khusus, yaitu:

- ◆ Alat bahasa berorientasi-pemakai interaktif
- ◆ Bahasa query sistem manajemen database (DBMS)
- ◆ Alat bahasa hypertext dan multimedia (atau hypermedia)

Kasus yang menyertai tersebut memberikan setting real-world (dunia nyata) dimana perangkat bahasa penggunaan khusus ini diterapkan atau diberlakukan.

Merancang Sistem Pendukung Keputusan di Delta Manufacturing Company

Delta Manufacturing Company adalah pabrikan ukuran sedang yang memproduksi hand tool dan peralatan leding. Derek Wright adalah direktur pengembangan sistem pada Delta tersebut. Sistem informasi penggantian yang baru telah diimplementasikan empat bulan yang lalu. Sistem ini sangat berhasil. Ia telah memenuhi keperluan atau kebutuhan pemrosesan informasi dan transaksi bisnis dari para sebagian besar pemakai. Namun demikian, Derek akan mengadakan meeting dengan tiga eksekutif puncak pada Delta tersebut untuk membahas pengembangan sistem pendukung keputusan (DSS) untuk mereka.

Clyde William, plant manager, memberi ucapan selamat kepada Derek dan para stafnya atas keberhasilan mereka mengembangkan sistem informasi baru di Delta. "Derek", sapa Clyde, "Kami sangat senang dengan apa yang telah anda kerjakan. Tujuan meeting kali ini adalah untuk meneliti kelayakan guna mengembangkan sistem pendukung keputusan bagi kami, yang mana sistem itu bisa kami gunakan tanpa perlu pergi ke kantor anda atau bertanya kepada staf anda setiap kali kami memerlukan suatu laporan. Dengan kata lain, sesuatu yang bisa kami lakukan sendiri."

"Yang saya idam-idamkan," kata Margaret Benfeld, marketing manager, "adalah workstation yang ada di depan saya, dimana saya dapat mengakses database untuk informasi penjualan dan informasi mengenai trend dan persaingan. Saya juga ingin membuat beberapa analisis what-if. Namun saya ingin melakukannya sendiri."

"Itu sama dengan yang saya inginkan," kata Jeff Trainor, kepala bagian keranjang. "Saya ingin mengakses sistem manajemen database baru dari waktu ke waktu dari workstation saya sendiri."

"Derek, apakah kami menginginkan sesuatu yang tidak layak atau tidak mungkin?" tanya Clyde.

"Tidak, tidak sama sekali," jawab Derek. "Saya dan staf saya dapat mengembangkan sistem pendukung keputusan untuk anda semua. Kami perlu melakukan sejumlah interview untuk memperjelas apa yang anda semua perlukan. Juga, saya telanjur dalam hal: Anda harus mau menjalani pelatihan, sebab dengan adanya kemajuan dalam teknologi informasi, bahasa pemrograman, dan perangkat pengembangan, maka para pemakai masih harus mencurahkan banyak waktu dan tenaga mereka untuk mempelajari bagaimana bekerja dengan sistem tersebut."

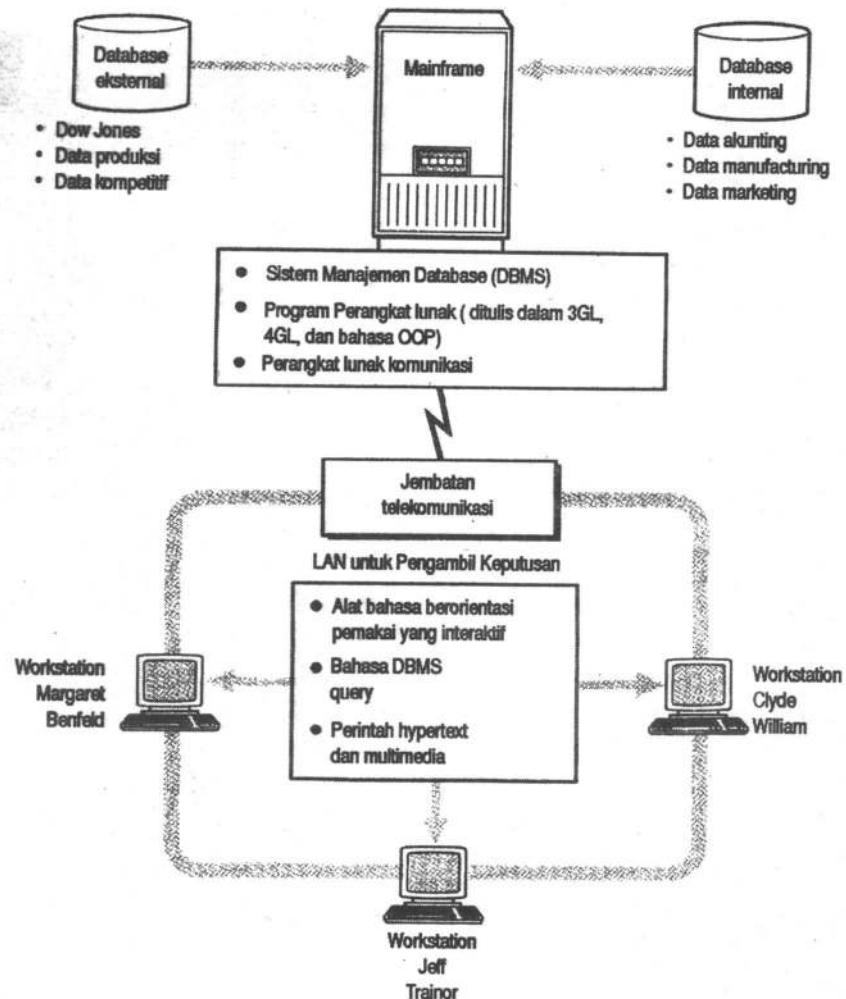
"Kami siap meluangkan waktu dan tenaga untuk mempelajari menggunakan sistem tersebut," kata Clyde. Jeff dan Margaret juga menyatakan setuju dengan mengangguk.

"Bagus," kata Derek. "Namun lebih dulu kita perlu menyusun pelaksanaan interview dengan anda semua."

"Bagaimana dengan sesion JAD seperti yang kita lakukan untuk sistem baru tersebut?" tanya Margaret. "Sesion-sesyon JAD awal benar-benar membantu menyusun pondasi yang baik dan memberi kita permulaan yang sangat bagus."

"Kedengarannya bagus itu," Derek menjawab dengan antusias (Derek dikenal sebagai tokoh atau pakar pengembangan aplikasi bersama di Delta). Bagaimana kalau kita menyusun (set up) sesion JAD mini pada hari Rabu yang akan datang? Apakah anda semua bisa meluangkan waktu? Semua orang setuju dan punya waktu untuk bertemu pada hari Rabu yang akan datang.

Diperlukan waktu tiga bulan untuk melakukan rancangan dan rancangan ulang sebelum sistem pendukung keputusan tersebut bisa diimplementasikan di Delta. Rancangan arsitektural umumnya dilukiskan pada Gambar 2.3. Bagian paling atas dari gambar tersebut meliputi komponen-komponen yang telah tersedia.



Gambar 3.3 Sistem pendukung keputusan (DSS).

Alat tersebut merepresentasikan DSS baru dan koneksinya ke sistem informasi Delta. Koneksi eksternal telah ditambahkan guna memberikan data mengenai lingkungan operasi dan pelanggan. Selain itu, dia juga akan menjalankan tugas pemrosesan transaksi dan mendukung analisis. Dalam hal ini, dia berfungsi sebagai pelayan database dan pengolah data yang bersifat intuitif. Ini adalah bentuk yang mendeskripsikan

Perangkat Bahasa Berorientasi-Pemakai Interaktif

Interactive User-Oriented Language Tools (perangkat bahasa berorientasi pemakai interaktif) memungkinkan para pemakai berinteraksi dengan sistem dalam bentuk:

- ◆ Perangkat bahasa dialog interaktif
- ◆ Perintah spreadsheet untuk analisis what-if

Menggunakan Perangkat Bahasa Dialog Interaktif

Interactive Dialogue Language Tools (perangkat bahasa dialog interaktif) berikut ini memungkinkan para eksekutif Delta melakukan atau menyelenggarakan dialog dengan sistem tersebut.

Perintah. Bentuk dialog dengan sistem ini menggunakan satu dari dua jenis bahasa:

- ◆ Sintaktis
- ◆ Natural

Untuk perintah sintaktis, para eksekutif Delta mempelajari kosa kata khusus untuk menjalankan DSS tersebut. Perintah-perintah seperti COPY, RETRIEVE, PRINT, EDIT, dan MOVE diikuti oleh satu argumen atau lebih, seperti FILEA, FILEB, dan FILEC. Sistem command-driven (yang dikendalikan perintah) sintaktis menawarkan fleksibilitas interaktif yang tinggi, namun sistem ini memerlukan pembelajaran yang banyak dan paling mudah terlupakan. Untuk mengatasi masalah ini, perintah-perintah sintaktis disusun (compiled) di dalam perpustakaan perintah (command dictionary) untuk para eksekutif Delta.

Tujuan atau maksud interface bahasa natural adalah untuk mengurangi jumlah sintaks khusus yang harus dipelajari para eksekutif. Perintah bahasa natural, misalnya, adalah: "Display division B's sales for the first quarter" (Tampilkan penjualan divisi B untuk kuartal pertama).

Menu. DSS menu-driven memungkinkan para eksekutif memilih suatu laris item. Lebih luas lagi, menu memberi pedoman interaksi.

Gambar 3.4 menunjukkan interface menu DSS biasa, yang dalam hal ini menggunakan teknologi layar sentuh. Perlu dicatat bahwa ia memberikan cara langsung kepada eksekutif yang sibuk dan yang mempunyai keterampilan komputer yang sedikit untuk memanggil informasi khusus perusahaan maupun informasi mengenai

lingkungan kompetitif perusahaan. Utiliti surat elektronik juga tersedia, maka eksekutif tersebut bisa me-request (meminta/memanggil) informasi tambahan dari personel kunci.

Icon. Ini melibatkan pemberian atau penyediaan representasi gambar tentang prosedur-prosedur yang berbeda. Kabinet file menunjukkan suatu prosedur untuk menyimpan file, atau gambar suatu suatu grafik/diagram balok yang merepresentasikan prosedur untuk menampilkan informasi pada diagram balok.

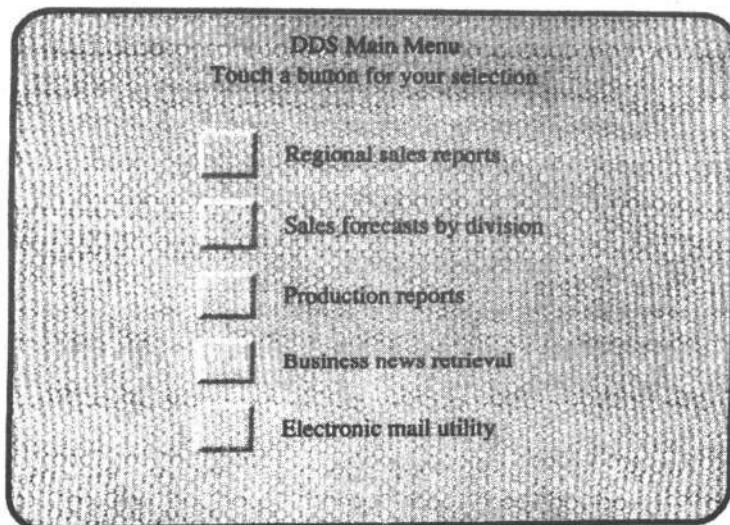
Pertanyaan dan Jawaban. Dengan pendekatan atau cara ini, DSS menanyakan serangkaian pertanyaan, eksekutif memberikan jawaban, dan pada akhir sesion Q/A (tanya/jawab), sistem membuat rekomendasi. Contoh sederhananya adalah sebagai berikut:

DSS: Apa tujuan anda?

USER: Untuk memperoleh pertumbuhan perusahaan.

DSS: Nama-nama perusahaan apa yang sedang anda selidiki?

USER: Inputlah simbol-simbol ketikan dari perusahaan-perusahaan calon.



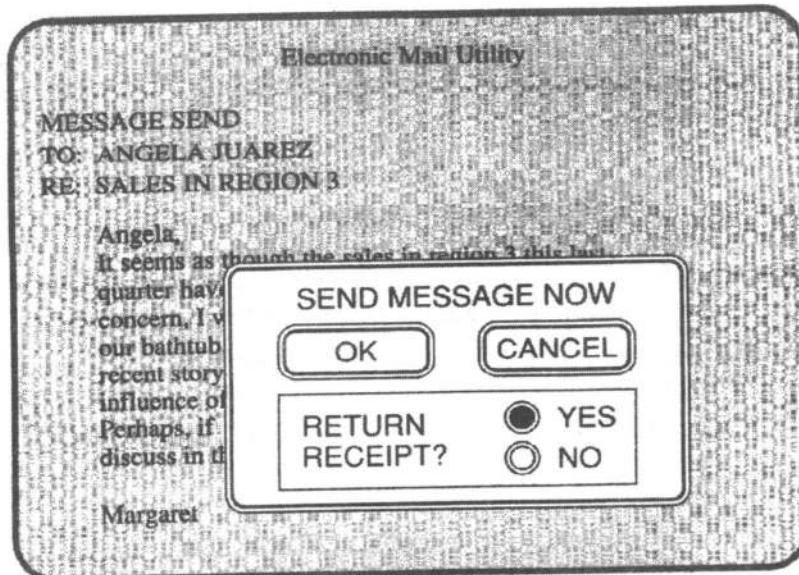
Gambar 3.4 Interface DSS menu-driven.

- DSS: Berapa penilaian kreditnya, harga stok high-low pada 1992 dan 1993, rasio harga/pendapatan, pendapatan per saham sejak 1991, aset saat itu, liabilitas saat itu, liabilitas jangka-panjang, dan modalnya?
- USER: Inputlah variabel-variabel ini melalui bentuk elektronik entri-data.

DSS mengkombinasikan variabel-variabel ini dengan data eksekutif yang disimpan dalam database eksternal dan memproses mereka melalui model akuisisi investasi. Eksekutif tersebut kemudian diberi skenario akuisisi dan rekomendasi.

Kotak Dialog. Kotak ini, yang ditampilkan pada layar, berisi opsi yang menanyakan/minta informasi yang berbeda. Setelah informasi yang diminta diberikan, pemakai memilih suatu tombol dan menyentuhnya untuk menjalankan perintah atau memilih opsi tersebut.

Utiliti surat elektronik yang kita bahas sebelumnya merupakan aplikasi yang ideal untuk kotak dialog. Gambar 3.5 menunjukkan bagaimana layar surat elektronik kelihatan/nampak. Eksekutif baru saja mengetikkan pesan ke salah satu manajer marketingnya mengenai laporan penjualan. Kotak dialog pop-up muncul pada layar,



Gambar 3.5 Kotak dialog pop-up sebagai bagian dari sistem pendukung keputusan.

	Period 1	Period 2	Period 3	Period 4	Period 5
Sales	\$100,000	\$125,000	\$150,000	\$175,000	\$200,000
Variable costs	\$75,000	\$90,000	\$105,000	\$120,000	\$135,000
Contribution margin	\$25,000	\$35,000	\$45,000	\$55,000	\$65,000
Profit volume margin	0.25	0.28	0.30	0.31	0.33

Sales = 100. Previous sales * 1.25
 Variable costs = 75. 100, 125, 150, 175
 Contribution margin = Sales - Variable costs
 Profit volume ratio = Contribution margin / Sales

Gambar 3.6 Layar what-if awal (dalam ribuan).

dimana ia memungkinkan eksekutif tersebut mengirimkan pesan atau membatalkannya, dan menawarkan opsi suatu return receipt ketika pesan tersebut dibaca. Opsi-opsi pada kotak dialog mudah dipilih dengan pointer mouse atau dengan cara menyentuh layar sentuh sensitif.

Menggunakan Perintah Spreadsheet untuk Analisis What-If

Salah satu fasilitas rancangan DSS di Delta yang paling berdaya guna adalah kemampuannya untuk memberikan analisis what-if untuk para eksekutif. Kita asumsikan, misalnya, mereka ingin memproyeksikan data biaya-volume-keuntungan (CVP) untuk lima periode. Departemen marketing telah mengestimasikan bahwa penjualan untuk periode pertama untuk produk tertentu akan sebesar \$100.000, dan mereka ini akan meningkat sebesar 25 persen setiap periode. Departemen akuntansi biaya memproyeksikan biaya variabel sebesar \$75.000 untuk periode pertama, \$100.000 untuk periode kedua, \$125.000 untuk periode ketiga, \$150.000 untuk periode keempat, dan \$175.000 untuk periode kelima. Dengan data ini, layar what-if diset-up untuk mengungkap hasil perubahan kondisi. Layar what-if awal dan hasilnya ditampilkan pada Gambar 3.6.

Pada bagian bawah layar what-if awal terdapat model dan data yang digunakan untuk menggenerasi nilai-nilai pada layar bagian atas. Layar awal ini akan menjadi base case (kasus dasar). Apabila diberikan definisi baru kepada suatu variabel, maka definisi ini akan mengganti definisi yang ada dalam base case. Namun demikian, base case ini masih disimpan dalam memori dan dapat dipanggil kembali kapan saja.

Dalam banyak analisis what-if, eksekutif mungkin ingin mengubah variabel dalam satu atau beberapa periode dari nilainya dalam base case. Sebagai contoh, statemen what-if:

`Sales = Prior * 1.10`

memberitahukan atau meminta perangkat lunak untuk menggunakan definisi dalam base case dan menaikkannya sebesar 10 persen.

Fasilitas berdaya guna lain dari fasilitas what-if adalah goal-seek (pencarian tujuan). Melalui goal-seek, eksekutif bisa mencari nilai yang harus dippunyai variabel tertentu guna mencapai tingkat kinerja yang dikehendaki. Sebagai contoh, eksekutif mungkin mengetikkan tujuan sebagai berikut:

`Goal : Contribution Margin = 25. Previous + 20`

	1	2	3	4	5
Sales	100.00	125.00	156.25	195.31	244.14
Variable costs	75.00	80.00	91.25	110.31	139.14
Contribution margin	25.00	45.00	65.00	85.00	105.00
Profit volume margin	0.25	0.36	0.41	0.44	0.43

Sales = 100, Previous sales * 1.25
 Variable costs = 75, 100, 125, 150, 175
 Contribution margin = Sales - Variable costs
 Profit volume ratio = Contribution margin / Sales
 Goal-seek:
 Goal: Contribution margin = 25, Previous + 20
 Adjust: Variable costs

Gambar 3.7 Layar goal-seek (dalam ribuan).

Goal atau tujuan ini menyatakan bahwa margin kontribusinya mulai pada \$25.000 dan meningkat \$20.000 setiap periode.

Apabila tujuan tersebut telah ditetapkan, sistem akan meminta variabel untuk disesuaikan atau disetarakan. Untuk mengetahui bagaimana eksekutif bisa mencapai tujuan dengan cara mengubah biaya variabel, ia mengetikkan:

Adjust : Variable Cost

Segera setelah eksekutif menekan tombol Return, DSS akan menyelesaikan model dan menampilkan hasil seperti yang terlihat pada Gambar 3.7.

Hasil dari layar goal-seek menunjukkan pada eksekutif bahwa ia harus mengurangi biaya variabel sebesar jumlah yang ditunjukkan untuk setiap periode jika penjualan sebenarnya sama dengan yang diproyeksikan dalam base case, dan jika Delta menyadari atau mengetahui margin kontribusinya naik sebesar \$20.000 setiap periode, yang diawali dengan margin kontribusi sebesar \$25.000 dalam periode 1.

Menggunakan Bahasa Sistem Manajemen Database Relasional

Dua bahasa query DBMS (atau RDBMS) yang terutama dipilih untuk DSS di Delta adalah:

- ◆ Bahasa query terstruktur (SQL)
- ◆ Query-by-example (QBE)

Bahasa Query Terstruktur

Bahasa query terstruktur (SQL) adalah perintah yang memungkinkan eksekutif Delta menetapkan, memanipulasi, dan mengontrol data dalam DBMS relasionalnya Delta.

SQL dapat digunakan untuk operasi matematika dan subquery. Sebagai contoh, Clyde ingin mencari semua pekerja yang gajinya lebih besar dari rata-rata gaji pekerja di perusahaan tersebut. Perintah SQL yang ditulis Clyde adalah

```
SELECT Employee_Name  
FROM Employee_Table  
WHERE Salary > SELECT AVG (Salary)  
FROM Employee_Table)
```

SQL dapat juga digunakan untuk memasukkan/menyisipkan, mengupdate, dan menghapus data dalam database. Sebagai contoh, untuk memasukkan pelanggan baru yang bernama Tricor, perintah SQL yang diperlukan mungkin akan nampak seperti berikut ini:

```
INSERT INTO Customer_Table (Customer_Number,
    Customer_Name, Street, City,
    Amount_Owed).
VALUES      (19978, Tricor, 49 Comstock,
    Reno, 0.0);
```

Eksekutif bukannya satu-satunya orang yang bisa mengambil manfaat dari SQL seperti yang ditunjukkan contoh sebelumnya, para pemakai lain juga memperoleh akses ke DBMS relasional untuk melakukan update terhadap DBMS ini dan membuat atau melakukan query ad hoc tertentu. Oleh karena itu, para staf Jeff menggabungkan SQL ke dalam COBOL, bahasa pemrosesan transaksi utama dari sistem informasi tersebut. SQL yang digabungkan ini bisa mengurangi banyak waktu dan tenaga yang diperlukan untuk merancang dan mengkode program 3GL jalur utama. Ia juga meningkatkan mutu fasilitas query-and-control dari sistem informasi tersebut.

Query-By-Example

Query-by-example (QBE) adalah salah satu interface visual ke DBMS relasional yang paling populer. QBE menetapkan contoh query atau update yang diperlukan. Konsep querying (permintaan/pemanggilan) dengan contoh adalah bahwa eksekutif tidak menjabarkan perintah yang harus diikuti dalam memperoleh informasi yang dikehendaki, namun ia memberikan contoh tentang apa yang diperlukan.

Gambar 3.8 melukiskan suatu aplikasi QBE. Nilai Atribut P. mengidentifikasi atribut-atribut yang akan dicetak pada laporan atau yang akan ditampilkan pada suatu layar. Sebagai contoh, kita asumsikan query-nya: : I want to find all valves ordered alphabetically by description in wharehouse A with quantity on hand greater than 30 and price of more than \$12,00." (Saya ingin mencari semua pesanan valve/katup yang dipesan secara alfabetis menurut deskripsi di gudang A yang kuantitasnya saat itu lebih besar dari pada 30 dan harganya lebih tinggi dari pada \$12.00). Query ini terlihat pada gambar sebelah kiri; hasil dari query QBE-nya terlihat pada gambar sebelah kanan.

Seperti halnya SQL, QBE dapat juga menjalankan prosedur seperti pengupdatean, operasi aritmetika, penggabungan relasi, penyisipan, dan penghapusan.

QBE Query						QBE Response		
Inventory	Inventory Item	Description	Quantity on hand	Price	Warehouse	Description	Quantity on hand	Price
B	Valves	Order by	>30	>12.00	A	2" Ball	61	14.50

Gambar 3.8 Query QBE dan responnya.

Sebagai contoh, misalnya Margaret ingin memasukkan/menyisipkan item inventarisasi baru yang mencakup empat belas pilinan kotak 24-inci berharga \$9,00 dan yang disimpan dalam gudang B. Juga, gergaji tarik yang ditempatkan dalam gudang C merepresentasikan suatu item inventarisasi yang akan dihapus dari inventarisasi itu. Terakhir, katup Ball 2-inci yang ditempatkan dalam gudang A akan dinaikkan

QBE Query					
Inventory	Inventory Item	Description	Quantity on hand	Price	Warehouse
B	Wrenches	24" Box	#14	\$9.00	B
D	Steel Valves	Grooving 2" Ball	-	-	C
E					

Gambar 3.9 Prosedur penyisipan, penghapusan, dan pengup-datean QBE.

sebesar 20 persen. Harga katup 2-inci saat itu adalah \$14,50. Tabel yang menangani prosedur QBE ini ditampilkan pada Gambar 3.9.

Perangkat Hypertext dan Multimedia

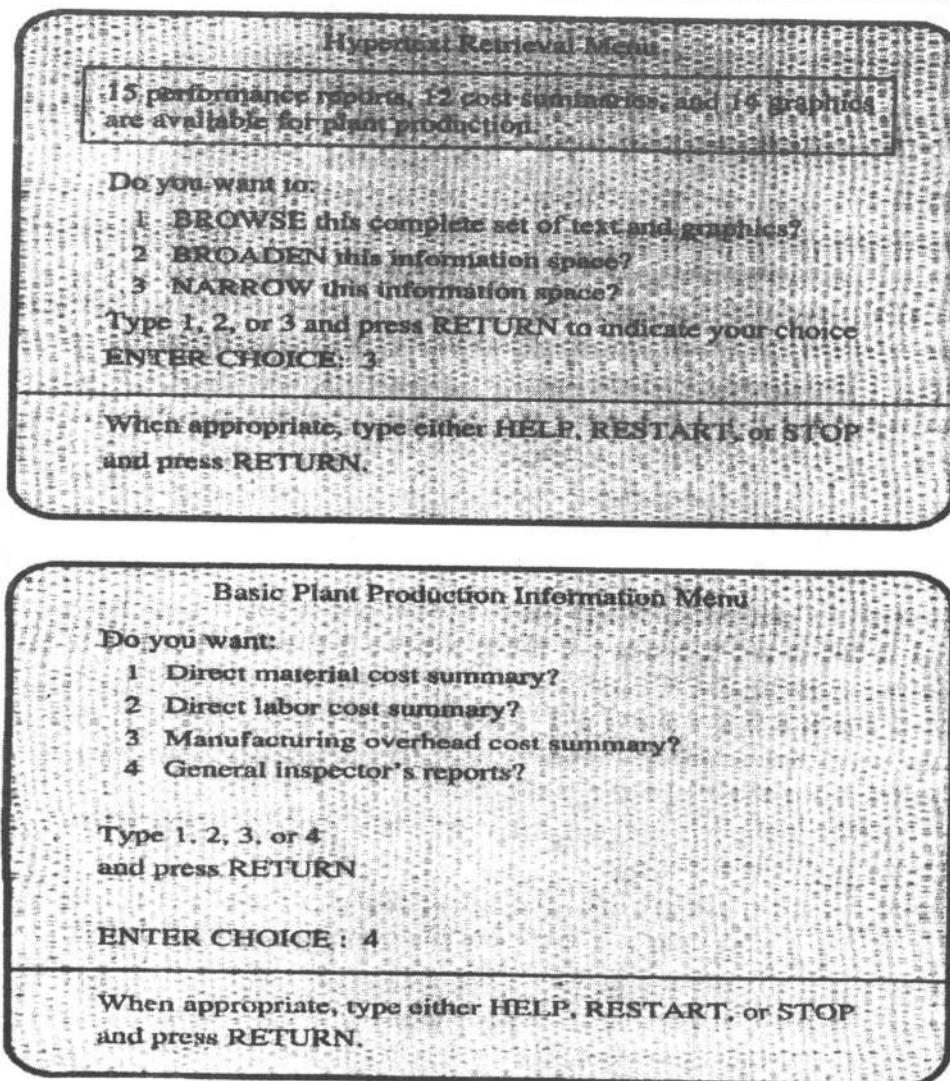
HYPertext adalah metode pengorganisasian dan pengasosiasian informasi tekstual dengan suatu cara hirarkis tak-berangkai (tak urut). Apabila konsep hypertext diperluas untuk mencakup penggabungan bersama bukan saja informasi tekstual namun juga grafik, icon, video, suara, program, dan bentuk informasi penting lain, maka konsep tersebut diperluas menjadi **MULTIMEDIA**, yang juga disebut sebagai **HYPER-MEDIA**. Alat ini digunakan untuk membantu mengelola arus/muatan informasi yang semakin meningkat di Delta, dan untuk membantu eksekutif mengasimilasi informasi penting tanpa ditenggelamkan olehnya.

Aplikasi biasa dari perangkat hypertext dan multimedia ini adalah pengembangan/pembuatan suatu laporan kinerja produksi untuk Clyde Williams. Ia melihat-lihat potongan-potongan informasi, dan kemudian memilih, menggabungkan bersama, dan memetakan potongan ini ke dalam laporan multimedia.

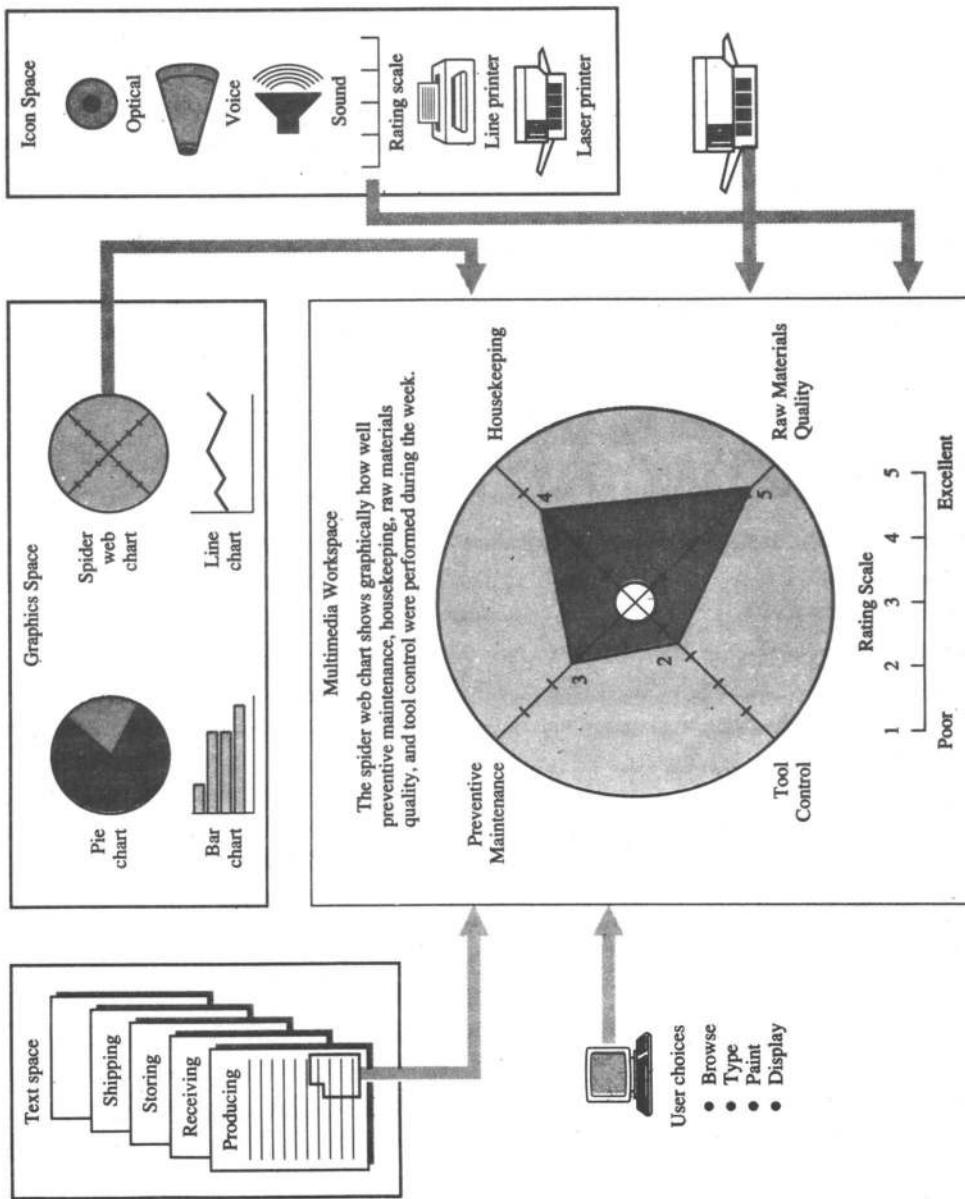
Layar pertama yang dipresentasikan kepada Clyde adalah menu pemanggilan hypertext, seperti terlihat pada bagian atas Gambar 3.10. Pada bagian bawah gambar tersebut terdapat hasil dari pilihan 3, yang ini merupakan menu lain yang memberikan laporan produksi pabrik dasar: yaitu rekapitulasi biaya bahan langsung, rekapitulasi biaya buruh langsung, rekapitulasi biaya tambahan manufakturing, laporan pengawas umum, dan pilihan untuk menuju ke topik lain. Pilihan Clyde adalah 4, yaitu laporan pengawas umum.

Hasil dari pilihan akhir Clyde adalah suatu layar, seperti terlihat pada Gambar 3.11, yang digunakan untuk menghubungkan potongan-potongan informasi dalam suatu ruang kerja (workspace) multimedia (atau hypermedia). Bahan tekstual tersebut berisi sejumlah laporan mengenai evaluasi pengawas umum terhadap operasi-operasi produksi, yang meliputi produksi, penerimaan, penyimpanan, dan pengiriman.

Pertama kali Clyde ingin melihat-lihat beberapa laporan ini. Hal ini seperti membuka-buka secara elektronis tumpukan kertas. Ia memilih BROWSE dan meng-klik tombol mouse guna memungkinkannya melakukan hal itu. Setelah menemukan potongan atau fragmen tertentu dari teks yang ia inginkan, ia menggerakkan atau memindahkan kursor ke TYPE dan meng-klik mouse. Fragmen/potongan teks yang ia pilih akan terkopi secara otomatis dalam workspace multimedia. Kemudian ia melihat-lihat pada ruang grafik dan memilih diagram/grafik jaring laba-laba untuk menunjukkan empat elemen evaluasi-kinerja manufakturing (yakni, ia bekerja di shop

**Gambar 3.10** Menu hypertext.

floor): pemeliharaan preventif, perawatan, kualitas bahan mentah, dan pengontrolan perangkat. (Elemen-elemen ini dinilai oleh pengawas umum). Skala penilaian juga dimasukkan dari ruang icon untuk memperjelas arti atau maksud dari apa yang digambarkan oleh grafik jaring laba-laba.



Gambar 3.11 Penggunaan hypertext dan multi-media untuk mengasosiasikan (menggabungkan) potongan-potongan informasi.

Baik grafik jaring laba-laba maupun skala penilaian secara otomatis dituliskan pada layar workspace multimedia dengan memilih grafik dan icon yang tepat, memindahkan kursor ke PAINT, dan meng-klik mouse. Dan terakhir, Clyde menginginkan workspace multimedianya yang berisi fragmen teks, grafik jaring laba-laba, dan skala penilaian dicetak pada printer laser. Ini dilakukan dengan cara memilih icon laser printer, memindahkan kursor ke DISPLAY, dan meng-klik mouse. Hard copy dari workspace multimedia tersebut akan dicetak secara otomatis melalui printer laser.

Alat pengembangan multimedia GUI (interface pemakai grafis) memungkinkan pengembang menggabungkan (menyatukan) audio, grafik, teks, dan animasi dalam aplikasinya. Manfaat atau keuntungan nyata dari perangkat semacam itu adalah bahwa ia memungkinkan non-programmer menciptakan aplikasi secara mudah. Para pabrikan hardware telah me-release PC multimedia yang mendukung/menyediakan semua fasilitas multimedia. Platform hardware ini berisi drive CD-ROM maupun dukungan audio dan animasi.

MEMILIH BAHASA YANG TEPAT

Mungkin satu-satunya bahasa dan perangkat yang sempurna untuk para pemakai adalah bahasa yang disampaikan secara natural (alami); yakni, pemakai tinggal meminta komputer untuk melakukan apa yang ia inginkan dan komputer merespon dengan segera dan secara presisi. Ada sejumlah riset yang sedang dilakukan yang bertujuan untuk mencapai atau mendapatkan interface pemakai/sistem akhir ini. Tercapainya cita-cita ini didasarkan pada sejumlah spekulasi. Sementara ini, kita harus memilih bahasa yang tepat yang tersedia saat ini untuk perancangan perangkat lunak.

Mencocokkan Bahasa dengan Aplikasi Rancangan Perangkat lunak

Lihat Gambar 3.12 untuk mengetahui pedoman ringkas yang bisa membantu anda dalam memilih bahasa atau campuran bahasa yang tepat dan perangkat bahasa penggunaan-khusus. Ketika kita memilih bahasa dan perangkat tertentu untuk mengkode aplikasi rancangan perangkat lunak tertentu, ada beberapa hal yang harus dipertimbangkan:

- ◆ Apakah sistem komputer yang ada saat itu mempunyai assembler, compiler, atau translator untuk bahasa tertentu?
- ◆ Apakah dalam organisasi/perusahaan tersebut terdapat programmer yang mahir atau berpengetahuan baik terhadap bahasa itu?

Aplikasi Rancangan Perangkat lunak	Bahasa			
	3GL	4GL	Bahasa OOP	Perangkat Bahasa Kegunaan Khusus
Aplikasi jika efisiensi mesin sangat penting	Bagus sekali	Sedang sampai buruk tergantung pada aplikasi	Bagus	TB
I/O bound	Bagus sekali	Buruk	Bagus sekali	TB
Proses-bound	Bervariasi, FORTRAN: Bagus sekali COBOL: Sedang sampai buruk	Sedang	Sedang sampai bagus sekali	TB
Sistem pendukung keputusan (DSS)	Sedang sampai bagus sekali	Bagus sekali	Bagus sekali	Bagus sekali
Sistem informasi eksekutif (EIS)	Sedang sampai bagus sekali	Bagus sekali	Bagus sekali	Bagus sekali
Prototipe	Sedang sampai bagus sekali	Bagus sekali	Bagus sekali	TB
Ad hoc	Bagus sekali jika SQL digunakan	Bagus sekali	Bagus sekali	Bagus sekali
Interaksi suara	TB	TB	TB	Bagus dan berkembang
Mendukung rancangan terstruktur	Ya	Tidak begitu bagus	TB	TB
Mendukung rancangan berorientasi objek	Ada, C, COBOL dan Pascal dapat dimodifikasi untuk mengimplementasi beberapa rancangan konsep OO.	Tidak	Smalltalk, Eiffel, C++, Object Pascal, dan Object COBOL dirancang secara khusus untuk konsep rancangan OO.	TB

TB = tidak berlaku.

Gambar 3.12 Rekapitulasi aplikasi rancangan perangkat lunak dan bahasa yang mendukungnya.

- ◆ Apakah ada banyak programmer di komunitas bisnis tersebut yang terampil terhadap bahasa itu?
- ◆ Jika program tersebut akan dijalankan pada arsitektur komputer yang berbeda, apakah bahasa yang dikodenya bersifat portabel (bisa dijalankan pada mesin lain)?
- ◆ Ada banyakkah vendor CASE yang mendukung bahasa itu?
- ◆ Jika perangkat bahasa penggunaan-khusus akan digunakan, apakah para pemakai mau mencurahkan cukup waktu dan tenaga untuk diberi latihan cara menggunakannya?

Namun, hal/pertanyaan yang terpenting adalah: Bagaimana sifat dari aplikasi rancangannya? Jawaban atas pertanyaan ini adalah sebagai berikut.

- ◆ Jika aplikasi adalah process-bound (bersifat rangkaian proses), yang memerlukan sejumlah besar kalkulasi kompleks, maka disarankan menggunakan bahasa seperti Ada, APL, C, FORTRAN, Pascal, atau PL/1.
- ◆ Jika aplikasi merupakan input/output-bound (rangkaian input/ output), yang memerlukan pemanipulasi file yang besar (ekstensif), seperti memproses daftar gaji yang besar atau file account receivable, maka pilihan bahasa yang disarankan adalah COBOL atau RPG.
- ◆ Jika aplikasinya melibatkan pemanipulasi data untuk analisis manajemen, laporan cepat, atau prototyping, maka sebaiknya dipilih 4GL.
- ◆ Untuk keperluan pemrosesan informasi yang sangat khusus, kita perlu mempertimbangkan pilihan pada perangkat bahasa berorientasi-pemakai interaktif, bahasa query DBMS, dan perangkat hypertext dan multimedia (hypermedia).
- ◆ Untuk mengimplementasikan pendekatan rancangan modular terstruktur, disarankan menggunakan host 3GL seperti Ada, C, COBOL, dan Pascal.
- ◆ Untuk mengimplementasikan rancangan perangkat lunak berorientasi-obyek, banyak tersedia bahasa OOP untuk itu, seperti Smalltalk, Eiffel, C++, Object Pascal, dan Object COBOL.

C++ cocok digunakan oleh komunitas ilmiah dan mekanis karena ia mempunyai warisan/turunan dari C. Smalltalk, Object Pascal, dan terutama Object COBOL cocok untuk komunitas bisnis. Memang, para anggota komite COBOL ANSI (American Standard Institute telah bergabung dengan komite COBOL CODASYL (Confe-

rence on Data Systems Languages) dan Kelompok Kerja COBOL Berorientasi Obyek (OOCTG). Tujuan mereka adalah untuk mendukung, mengembangkan, dan menstandardisasi Object COBOL.

Persoalan Lain dalam Memilih Bahasa

Persoalan nyata lain dalam memilih bahasa pemrograman meliputi hal-hal berikut ini:

Tingkat Penggunaan dalam Dunia Bisnis. Jika suatu bahasa digunakan secara luas dalam dunia bisnis, maka tentunya ia mendapatkan dukungan pengembangan perangkat lunak yang berkelanjutan dari para vendor CASE besar dan dari sebagian besar orang yang berkepentingan dengannya. Bahasa tersebut kemungkinan besar juga akan portabel dari satu platform teknologi ke platform yang lainnya, sebab sebagian besar, bila tidak bisa dikatakan semuanya, vendor komputer memberinya dukungan. Kita asumsikan situasi berikut ini: Seluruh staf sistem pada ABC Company, yang meliputi analis sistem, perancang sistem, dan programmer, membeli kupon lotere berhadiah \$100 juta dan memenangkannya. Jika mereka tidak kembali bekerja, akankah para profesional sistem lain bisa melihat kemajuan pekerjaan, memahami bahasa (bahasa-bahasa) yang sedang digunakan ketika itu, dan melanjutkan pekerjaan dengan kehilangan waktu yang minimal? Jika jawabannya tidak, maka waktunyalah untuk beralih ke bahasa dan metode pengembangan sistem yang sifatnya standart dalam dunia bisnis dan yang lebih ekspresif.

Ke-ekspresif-an (kejelasan). Kemampuan bahasa untuk mrepresentasikan rangangan perangkat lunak secara penuh sehingga baik programmer maupun non-programmer dapat memahaminya disebut expressiveness (keekspressifan). Keekspressifan COBOL adalah salah satu alasan mengapa 3GL ini sangat populer dalam komunitas bisnis.

Kemudahan (convenience). Tingkat kemudahan atau kenyamanan didasarkan pada fasilitas-fasilitas seperti:

- ◆ Kemudahan penggunaan
- ◆ Kemudahan pembelajaran
- ◆ Produktivitas pengembangan

3GL dengan perangkat pengembangan CASE ekstensif, 4GL, bahasa OOP dengan perpustakaan kelas yang kaya, dan perangkat bahasa penggunaan-khusus memberikan kenyamanan.

Portabilitas. Portabilitas adalah kemampuan untuk memindahkan bahasa ke lingkungan yang berbeda. Portabilitas ini didasarkan pada definisi bahasa yang diterima, konsistensi dan kelengkapan fasilitas bahasa, dan set fungsi perpustakaan yang standart.

Kemampuan Pemeliharaan (maintainability). Setelah program perangkat lunak dikode dalam bahasa tertentu, diuji, dan dikonversi ke operasi, maka kemampuan peliharanya menjadi persoalan yang sangat penting. Dengan demikian, kemudahannya untuk berubah guna menyesuaikan diri dengan dinamika bisnis yang ia layani akan sangat penting. Namun demikian, kemampuan pelihara (maintainability) lebih ke fungsi rancangan dan dokumentasi dari pada fungsi bahasa jika bahasa tersebut dikenal dan programmer pemeliharaan ada. Dalam kasus seperti itu, maintainability akan menjadi suatu fungsi bahasa. Jika diterapkan prinsip rancangan terstruktur yang baik, dan rancangannya dikode dalam COBOL, maka program perangkat lunak yang dihasilkan akan relatif mudah dipelihara. Sebaliknya, jika rancangan perangkat lunak yang sama dikode dalam bahasa yang tak terkenal, pemeliharaannya akan menjadi masalah, karena sebagian besar orang tidak kenal dengan bahasa tersebut.

Kemampuan Perluasan (extendability). Kemampuan bahasa untuk meningkatkan cakupan rancangan perangkat lunak aslinya tanpa mengkode kembali program disebut extendability. Bahasa-bahasa OOP tidak tercatat mempunyai kemampuan diperluas.. Program-program 3GL, jika dikode menurut pendekatan rancangan perangkat lunak berorientasi-struktur, juga akan mudah diperluas.

DOKUMENTASI PERANGKAT LUNAK

Dokumentasi perangkat lunak menjabarkan program-program yang dirancang dan dikode untuk mendukung sistem tersebut. Dokumentasi perangkat lunak terkomposisi dari:

- ◆ Dokumentasi internal
- ◆ Dokumentasi eksternal

Dokumentasi Perangkat lunak Internal

Dokumentasi perangkat lunak internal dibubuhkan (digabungkan) dalam kode program. Karena adanya fasilitas ini, sejumlah bahasa dianggap sebagai bahasa "self-documenting". Sebagai contoh, COBOL dianggap bersifat self-documenting karena program yang dikode dalam COBOL terbaca seperti Bahasa Inggris sederhana. Namun agar suatu program terbaca seperti Bahasa Inggris sederhana (biasa), analis sistem harus menetapkan nama-nama yang standart dan bermakna. Sebagai contoh, field dalam suatu record yang berisi berbagai nomor/bilangan (yang berlainan) yang mengidentifikasi para pelanggan bisa disebut CUSTOMER-NUMBER. Nama ini bermakna; ia juga standart, sebab segala program yang menggunakan akan mengejanya dengan cara yang sama. Sebagai contoh lain, lihat segmen program COBOL berikut ini:

```

P1.      IF HOURS > 40 PERFORM P2
        ELSE PERFORM P3
P2.      COMPUTE RP = R * 40
        COMPUTE OT = 1.5 * R * (H - 40)
        COMPUTE GP = RP + OT
P3.      COMPUTE RPGP = R * H

```

Segmen yang mendahului akan berjalan sebab ia ditulis secara benar sejauh sintaks COBOL diberikan. Namun demikian, secara semantik, ia sebenarnya tak bermakna bagi non-programmer atau programmer lain yang bertugas memeliharanya. Akan tetapi, dengan penerapan nama-nama yang bermakna dan standart, segmen tersebut dapat dikonversi ke segmen yang dapat dipahami oleh programmer dan juga non-programmer, yaitu seperti berikut ini:

```

COMPUTE-GROSSPAY.
IF HOURS-WORKED > 40
    PERFORM GROSSPAY-WITH-OVERTIME
ELSE
    PERFORM GROSSPAY-WITHOUT-OVERTIME.
GROSSPAY-WITH-OVERTIME.
    COMPUTE REGULAR-PAY = HOURLY-RATE * 40.
    COMPUTE OVERTIME-PAY = 1.5 * HOURLY-RATE *
        (HOURS-WORKED - 40).
    COMPUTE GROSSPAY = REGULAR-PAY + OVERTIME-PAY.
GROSSPAY-WITHOUT-OVERTIME.
    COMPUTE REGULAR-PAY-GROSSPAY = HOURLY-RATE *
        HOURS-WORKED.

```

Seorang programmer dapat juga meningkatkan dokumentasi internal dari program tersebut dengan memasukkan atau menyisipkan komentar dan penjelasan yang dibatasi dengan tanda bintang ke dalam program tersebut. Sebagai contoh, penjelasan berikut ini bisa disisipkan persis di depan (sebelum) segmen program yang mengkalkulasi kuantitas pesanan ekonomis untuk item inventarisasi.

```
*****  
* THE FOLLOWING MODULE *  
* CALCULATES ECONOMIC *  
* ORDER QUANTITIES (EOQ) FOR *  
* INVENTORY ITEMS. *  
*****
```

Seperti yang telah kita lihat dalam Bahasa Inggris terstruktur, kita lebih baik memasukkan kode dengan cara yang sama, dan jangan menulis lebih dari satu kalimat (yakni, statemen program) pada baris yang sama. Aturan ini meningkatkan readibilitas (daya baca atau pemahaman) kode tersebut. Sebagai contoh, perhatikan segmen kode berikut ini:

```
IF EMPLOYEE-NO = RETIREMENT-NO PERFORM RETIREMENT-UPDATE ELSE  
DISPLAY "NO UPDATE." PERFORM READ-ROUTINE.
```

Untuk memperoleh readibilitas yang lebih baik, ubahlah segmen di atas sebagai berikut ini:

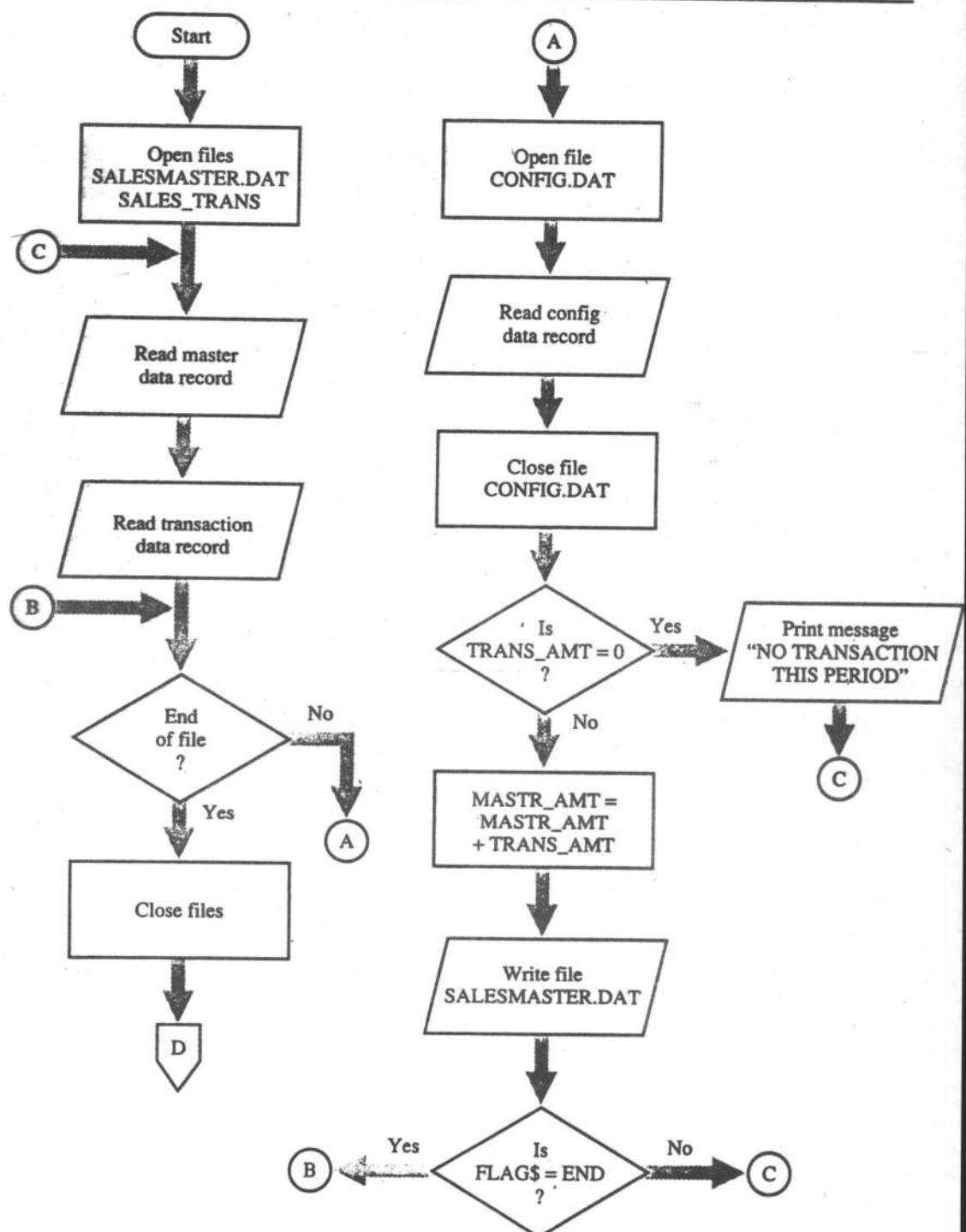
```
IF EMPLOYEE-NO = RETIREMENT-NO  
    PERFORM RETIREMENT-UPDATE  
ELSE  
    DISPLAY "NO UPDATE."  
PERFORM READ-ROUTINE.
```

Dokumentasi Perangkat lunak Eksternal Berbasis (menggunakan)-Kertas

Ada sejumlah form yang dapat digunakan untuk mendokumentasi program perangkat lunak. Setiap perusahaan biasanya menetapkan standart Dokumentasi Eksternal Berbasis-Kertas-nya sendiri dengan disesuaikan dengan item-item yang membentuk

Judul Halaman	
Nama Program:	CREATE_SALE
Nomer Program:	P7212
Tujuan Program	Program ini merekonsiliasi (menyesuaikan) record transaksi penjualan dengan record master penjualan, dan menggenerasi laporan.
Programmer:	Gerald Dwayne Orland
Departemen yang memakai:	Sales 415
Tanggal Penulisan:	17 September 1992
Manajer Proyek:	Helen T. Berg
Kendali:	Nomor serial transaksi harus sesuai dengan record yang berada dalam file CONFIG.DAT pada akhir run.
Input	(1) MASTER _FILE Label File: SALESMASTR.DAT Data Record: TRANS_REC (2) TRANSACTION_FILE Sumber: SALE_TRANS Data Record: TRANS_REC
Output:	(1) SALES REPORT (2) EXCEPTION REPORT
Output Distribusi:	Salinan 1 untuk Penjualan Salinan 2 untuk Manajer Warehouse Laporan Exception untuk Manajer Pemasaran

Gambar 3.13 Halaman judul untuk manual program.



Gambar 3.14 Flowchart terstruktur yang mendokumentasi logika program.

		RULES	
		1	2
IF (Conditions)	TRANS_AMT = 0	Y	N
	TRANS_AMT ≠ 0	N	Y
THEN (Actions)	PRINT MESSAGE: "NO TRANSACTION THIS PERIOD"	X	
	MASTR_AMT = MASTR_AMT + TRANS_AMT		X

Gambar 3.15 Tabel keputusan yang digunakan bersama dengan flowchart program terstruktur untuk mendokumentasi logika program.

RECORD LAYOUT WORKSHEET

Program Name: CREATE_SALE Program Number: P7212
 Record Name: TRANS_REC File Name: SALE_TRANS

Field		Size	Char.	Field Name	Remarks
From	To				
1	6	6	N	CUSTOMER_NUMBER	9(6)
7	21	15	A	LAST_NAME	
22	31	10	A	FIRST_NAME	
32	33	1	A	MIDDLE_INITIAL	
34	36	2	N	DEPT_CODE	
37	38	1	N	SALES_CODE	10=SHOES; 20=SHIRTS; 30=OTHER 1=WHOLESALE; 2=RETAIL
39	58	20	AN	ITEM_DESCRIPTION	
59	67	9	AN	ITEM_IDENTIFIER	99AAA99VA
68	73	6	N	UNIT_PRICE	
74	88	15	A	SALES_PERSON	9(4)V99

CHAR:
 A—Alpha N—Numeric S—Signed Numeric AN—Alphanumeric

New

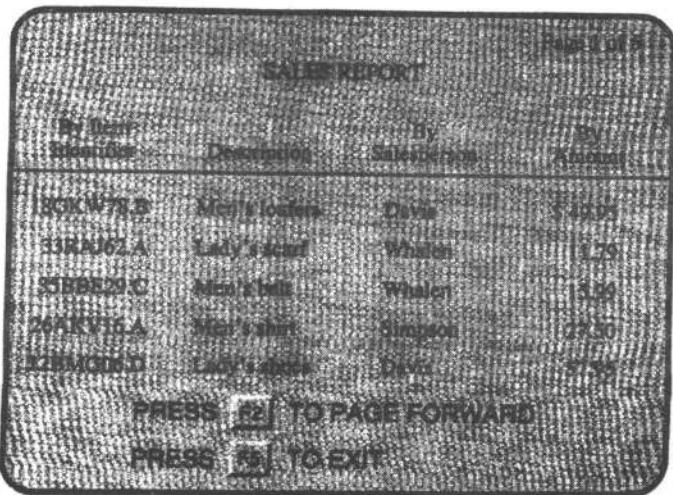
Replaces Pages

Approved: _____ Date: 7/15/92

Revision

_____ thru _____

Gambar 3.16 Worksheet tata letak record.



Gambar 3.17 Output sederhana dari program tersebut.

dokumentasi ini. Item-item ini disusun menjadi Manual Program. Beberapa item lebih penting yang dijumpai dalam manual program ini adalah:

- ◆ Halaman judul
- ◆ Berbagai perangkat modeling yang menjabarkan rancangan perangkat lunak, seperti flowchart program terstruktur, bagan struktur, diagram Jackson, diagram Warnier-Orr, tabel keputusan dan pohon keputusan, Bahasa Inggris terstruktur, dan persamaan.
- ◆ Deskripsi input
- ◆ Deskripsi output
- ◆ Kopi dari kode sumber
- ◆ Lembaran perubahan program

Setelah program perangkat lunak diuji, pokok bahasan bab berikutnya dan kasus uji yang digunakan untuk menguji program tersebut juga dimasukkan atau disertakan.

Gambar 3.13 menunjukkan suatu contoh halaman judul. Ia mengidentifikasi program dan memberikan informasi lain mengenai program tersebut agar kita memperoleh referensi yang mudah.

Flowchart program terstruktur (perangkat modeling lain bisa juga digunakan), yang dilukiskan pada Gambar 3.14, merepresentasikan rancangan logis dari program itu. Ia menunjukkan urutan langkah yang harus diambil atau dilakukan oleh komputer untuk mengkonversi input ke output. Gambar 3.15 adalah tabel keputusan yang digunakan dengan flowchart program terstruktur. Tabel keputusan ini mendeskripsikan logika yang ada dalam keputusan “Is Trans_Amt = 0?” dalam flowchart program terstruktur tersebut.

Gambar 3.16 menggambarkan lembaran tata letak record yang digunakan untuk mendokumentasirecord transaksi yang ada dalam file transaksi. Ia meliputi informasi seperti jenis (type) karakter yang ada di dalam field record. Tata letak (layout) yang sama dapat digunakan untuk mendokumentasi record master dalam file master.

Laporan yang terlihat pada Gambar 3.17 adalah contoh output yang dihasilkan oleh program perangkat lunak. Tata worksheet letak output yang digunakan dalam rancangan aslinya tersebut digambarkan pada Gambar 3.18.

Lembaran Perubahan Program

Program Change Sheet (lembaran perubahan program) digunakan untuk mencatat perubahan program yang dibuat setelah ia dikonversi ke operasi. Ia ditunjukkan pada Gambar 3.19. Biasanya, suatu perusahaan akan menggunakan Maintenance Work Order (WO) Form (Form Urutan Kerja Pemeliharaan untuk menginisiasi (memulai) dan memberikan wewenang dilakukannya perubahan terhadap program tersebut setelah ia dikonversi ke operasi. Form WO (urutan pekerjaan pemeliharaan dideskripsikan pada Bab 6.

DOKUMENTASI OPERASI

Tugas pemrosesan yang besar biasanya ditangani oleh kelompok orang khusus yang disebut operator komputer, yang bekerja dengan mainframe. Para operator berbeda dari end user yang biasanya duduk di depan terminal dan workstation yang dikoneksikan ke jaringan untuk melakukan tugas-tugas mereka.

Dokumentasi Operasi umumnya berada dalam bentuk Run Manual (Manual Berjalan) yang disimpan atau diletakkan dekat console komputer. Manual berjalan meliputi informasi berikut ini:

- ◆ Identifikasi tugas (aplikasi) dan waktu kapan ia dijalankan
- ◆ Identifikasi media input (misalnya, disk, pita)

Gambar 3.18 Worksheet tata letak output.

PROGRAM CHANGE SHEET		
To: _____	Date: _____	
Description of Program Change Requested:		
Date Desired: _____		
Requested By: Name: _____ Title: _____ Phone: _____ Department: _____		
SPACE BELOW FOR PROCESSING USE ONLY		
Program Name: _____		Program Number: _____
Change Approved By: _____		Date: _____
Assigned To: _____		
Change Reviewed By: _____		Approved By: _____
Date: _____		Date: _____
NEW DOCUMENTATION CHECKLIST:		
<input type="checkbox"/> Change title page, if necessary <input type="checkbox"/> Change software design, if necessary <input type="checkbox"/> Change description of input, if necessary <input type="checkbox"/> Change description of output, if necessary <input type="checkbox"/> Create new source code listing <input type="checkbox"/> Change test cases, if necessary <input type="checkbox"/> Change operations documentation, if necessary <input type="checkbox"/> Change user documentation, if necessary		

Gambar 3.19 Lembaran perubahan program.

- ◆ Nomor-nomor form untuk form output khusus atau nomor-nomor bagian (komponen) yang akan digunakan
- ◆ Instruksi yang digunakan untuk mensejajarkan (menyesuaikan) form pada printer
- ◆ Perangkat hardware yang diperlukan (misalnya, disk, printer, tape drive)
- ◆ Waktu pemrosesan yang diharapkan atau diramalkan
- ◆ Instruksi khusus yang harus dihentikan secara abnormal oleh program
- ◆ Pesan program dan tindakan operator yang diperlukan
- ◆ Pengontrolan
- ◆ Pendistribusian output

Dalam sistem berbasis mainframe tersentral yang besar, manual berjalan bahkan bisa mencakup nomor telepon rumah dan kantor dari para programmer pemeliharaan dan analis yang sewaktu-waktu diperlukan untuk mengatasi masalah selama 24 jam sehari. Run Sheet biasa yang ada (diisikan) dalam manual berjalan ditunjukkan pada Gambar 3.20. Umumnya, para operator tidak akan bisa menjalankan suatu tugas tanpa berkonsultasi dengan run sheet. Sistem yang besar mungkin mempunyai ribuan tugas dan ratusan diantaranya dihapus dan ditambahkan.

DOKUMENTASI PEMAKAI

Para pemakai (users) biasanya berinteraksi dengan sistem melalui layar dan keyboard, mouse, atau stylus pen. Untuk merancang User Documentation (Dokumentasi Pemakai), sebaiknya kita klasifikasikan para pemakai ini lebih dulu.

Mengklasifikasikan Pemakai

Beberapa profesional sistem mengklasifikasikan pemakai berdasarkan skala yang rumit, seperti:

- ◆ Parrot (peniru)
- ◆ Novice (baru)
- ◆ Intermediate (menengah)
- ◆ Expert (ahli)
- ◆ Master (yang sudah menguasai)

Nama Program: CREATE_SALE

Nama Pekerjaan: CSREV2

Frekuensi: Harian

Tipe: Transaksi

Input:

1. File master penjualan
2. File transaksi penjualan

Output:

1. Laporan penjualan yang dicetak pada form dua-bagian
2. Laporan kekecualian yang dicetak pada kertas stok

Instruksi Khusus:

1. Mengakses modul generator laporan penjualan dari pustakawan program dengan CSREV2 dan password
2. Apabila terjadi penggagal tugas, prosedur restart dituliskan pada halaman lampiran

Pesan Terprogram:

1. Menempatkan laporan penjualan pada printer A, mensejajarkan form-form, dan memasukkan S1 pada console
2. Menempatkan paper stock satu-bagian pada printer B dan memasukkan R1 pada console

Kendali

Nomor-nomor serial transaksi permulaan dan akhir yang diisikan (ditempatkan) dalam file CONFIG.DAT pada akhir pelaksanaan (run) tugas harus sesuai (cocok) dengan form pengiriman.

Distribusi Output:

1. Mengirimkan dan memilah-milahkan laporan penjualan. Memberikan kopi 1 ke bagian penjualan dan kopi 2 ke manajer gudang.
2. Memberikan laporan kekecualian ke manajer marketing
3. Mengembalikan kopi form pengiriman ke petugas (clerk) pengontrolan akunting.

Gambar 3.20 Run sheet biasa untuk memberikan instruksi kepada para operator komputer.

Ben Schneiderman telah mengklasifikasikan pemakai, berikut ini:

- ◆ Novice
- ◆ Occasional user (pemakai tak tetap/kadang-kadang)
- ◆ Frequent light user (pemakai bukan ahli yang sering)
- ◆ Frequent power user (pemakai ahli yang sering)

Klasifikasi pemakai yang digunakan dalam buku ini didasarkan pada kategori Horton, yaitu:

- ◆ **Pemakai Baru (Novice User).** Para pemakai ini tidak mempunyai pengetahuan sintaktis mengenai komputer dan perangkat lunak. Juga, novice users biasanya hanya sedikit mengetahui tentang tugas-tugas yang dibebankan pada mereka. Umumnya, mereka sangat tegang apabila menjumpai atau menghadapi kesalahan. Mereka kesulitan dalam membedakan apa yang penting dan apa yang sepele. Pemakai baru biasanya segan untuk meminta bantuan, karena mereka miskin kosa kata untuk mengekspresikan pertanyaan atau permintaan mereka.
- ◆ **Pemakai kadang-kadang (Occasional User).** Para pemakai ini mengetahui cara bekerja dengan sistem pada satu saat, namun karena jarang menggunakan, para pemakai kadang-kadang ini lupa perintah dan prosedur yang esensial. Mereka hanya menerima pelatihan awal yang terbatas dan tidak suka dengan dokumentasi perangkat lunak berbasis-kertas (misalnya, berbagai manual prosedur). Mereka tidak hati-hati dalam menggunakan istilah-istilah komputer yang formal dan presisi.
- ◆ **Pemakai Transfer (Transfer User).** Mereka ini telah mengetahui cara bekerja dengan sistem; para transfer user tinggal mencoba mentransfer apa yang telah mereka ketahui ke suatu sistem yang baru.
- ◆ **Pemakai Ahli (Expert User).** Orang-orang ini memahami cara bekerja dengan sistem yang baru maupun dengan sebagian besar sistem-sistem yang lain. Expert user tidak suka dengan segala prosedur dan instruksi yang menghabiskan waktu mereka. Tuntutan utama mereka adalah waktu respon yang cepat. Mereka suka shortcut (jalan pintas), macro command, dan singkatan.

Merancang Dokumentasi Online untuk Pemakai

Umumnya para pemakai memerlukan kombinasi dari:

- ◆ Tutorial
- ◆ Pesan
- ◆ Menu
- ◆ Icon
- ◆ Fasilitas Help
- ◆ Shortcut
- ◆ Manual referensi online

DOKUMENTASI ONLINE dapat berisi semua fasilitas ini guna memberi pedoman dan memberi instruksi kepada semua pemakai secara interaktif.

Tutorial

Banyak paket perangkat lunak yang tersedia secara komersial memberikan *tutorial*, atau pelajaran online, untuk memberi pedoman pemakai baru dan atau pemakai kadang-kadang dalam menggunakan perangkat lunak. Walaupun terpisah dari program perangkat lunak yang bekerja itu sendiri, tutorial ini dalam berbagai hal selalu menyerupai paket yang sebenarnya.

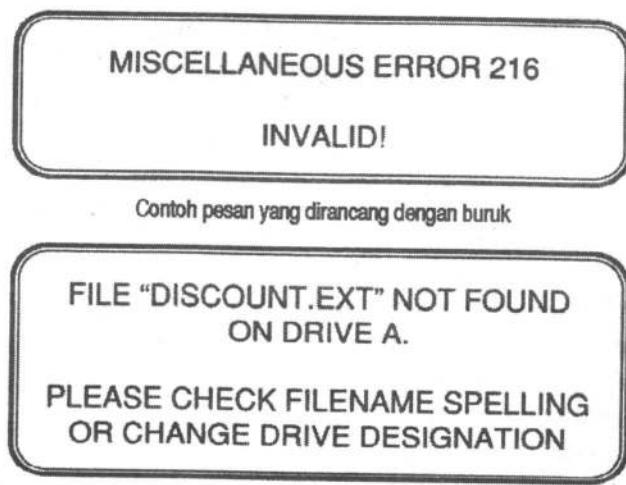
Keberhasilan tutorial online sebagai perangkat pelatihan sangat tergantung pada jumlah realisasi yang terbangun dalam pelajaran atau lesson tersebut. Ketika merancang tutorial, sebaiknya para pemakai sebanyak mungkin diberi akses ke fasilitas program. Tutorial yang dirancang secara tidak baik akan menghambat pemakai dalam melihat hasil suatu kesalahan. Tutorial ini akan menampilkan suatu pesan seperti: "Anda menekan tombol yang salah. Maka sebagai gantinya tekan yang ini." Tutorial yang terancang dengan baik memungkinkan pemakai melakukan kesalahan, melihat hasil dari kesalahan itu, dan akhirnya mengoreksi kesalahan tersebut dengan bantuan pesan dan prompt dari sistem itu.

Pesan

Cara atau perangkat memberitahu pemakai terhadap informasi mengenai sistem adalah dengan melalui **PESAN** pada layar. Pesan yang spesifik, padat, dan jelas akan sangat efektif.

Ketika merancang pesan, harus diingat bahwa pesan pada layar (onscreen message) tidaklah permanen. Ia biasanya hilang dari layar sebelum pemakai menindaklanjutinya. Agar pesan efektif, ia harus cukup singkat sehingga pemakai dapat mengingat semua poin-pokoknya sampai pemakai ini bisa menindaklanjutinya.

Gaya pesan harus konsisten sepanjang semua dokumentasi online. Gambar 3.21 melukiskan pesan yang terancang secara tidak baik, yang tidak nyaman untuk pemakai, dan yang tidak memberikan informasi yang membantu. Yang juga ditunjukkan adalah pesan kesaianan, yang menentukan secara pasti apa yang salah, dan secara bijak memberikan nasehat kepada pemakai tentang bagaimana mengoreksi masalah tersebut.



Gambar 3.21 Dua rancangan alternatif untuk pesan kesalahan.

Menu

Seringkali para pemakai dapat belajar banyak mengenai apa yang dilakukan sistem dan bagaimana ia beroperasi dengan hanya mempelajari menu perintah dari sistem tersebut. Menu memungkinkan pemakai baru atau pemakai kadang-kadang bisa mengoperasikan sistem yang rumit (canggih) secara baik dengan cara memanfaatkan kekuatan pikir manusia dalam hal pengenalan, bukannya kelemahan pikir dalam hal recall.

Dalam merancang menu, cobalah tetap memudahkan tugas pemakai dengan cara membuat setiap opsi menu bersifat self-explanatory (bisa menjelaskan sendiri). Salah satu cara melakukan hal ini pada menu yang lebih besar adalah dengan memberikan deskripsi setiap pilihan menu di sebelah kanan menu itu sendiri. Untuk menu yang lebih kecil, yang ruangnya terbatas, menu harus dirancang sedemikian rupa sehingga pemakai bisa melompat secara langsung ke tampilan help.

Tujuan keseluruhan suatu menu adalah untuk menghilangkan kompleksitas untuk menghafal dan memasukkan sejumlah perintah keyboard, dan menu yang sederhana (mudah) akan memungkinkan pemakai lebih produktif dengan hanya sedikit melakukan kesalahan. Sebaiknya pilihan menu difrasakan sebagai kata kerja tindakan, seperti "Realign Text", bukannya "Text Justification."

Terakhir, para perancang harus menghindarkan pemakai dari pilihan menu yang tidak tersedia bagi mereka atau pilihan menu yang tidak sesuai dengan tugas pemakai. Misalnya, banyak sistem yang besar memberikan berbagai tingkatan privilege (khusus) kepada pemakai untuk alasan keamanan. Karena dalam hal ini pemakai yang high-privilege-level (yang tingkat kekhususannya/keragaman tinggi) akan mempunyai akses ke perintah sistem yang lebih banyak, bila tidak bisa dikatakan semua-nya. Sedangkan para pemakai lain hanya dibatasi untuk bisa mengakses serangkaian perintah tertentu. Sebaiknya pemakai yang low-privilege-level tidak diberi pilihan menu yang tidak tersedia dalam menu tersebut, atau tidak memperlihatkan secara terbuka pilihan menu tersebut dengan cara menampilkannya secara agak tidak kentara.

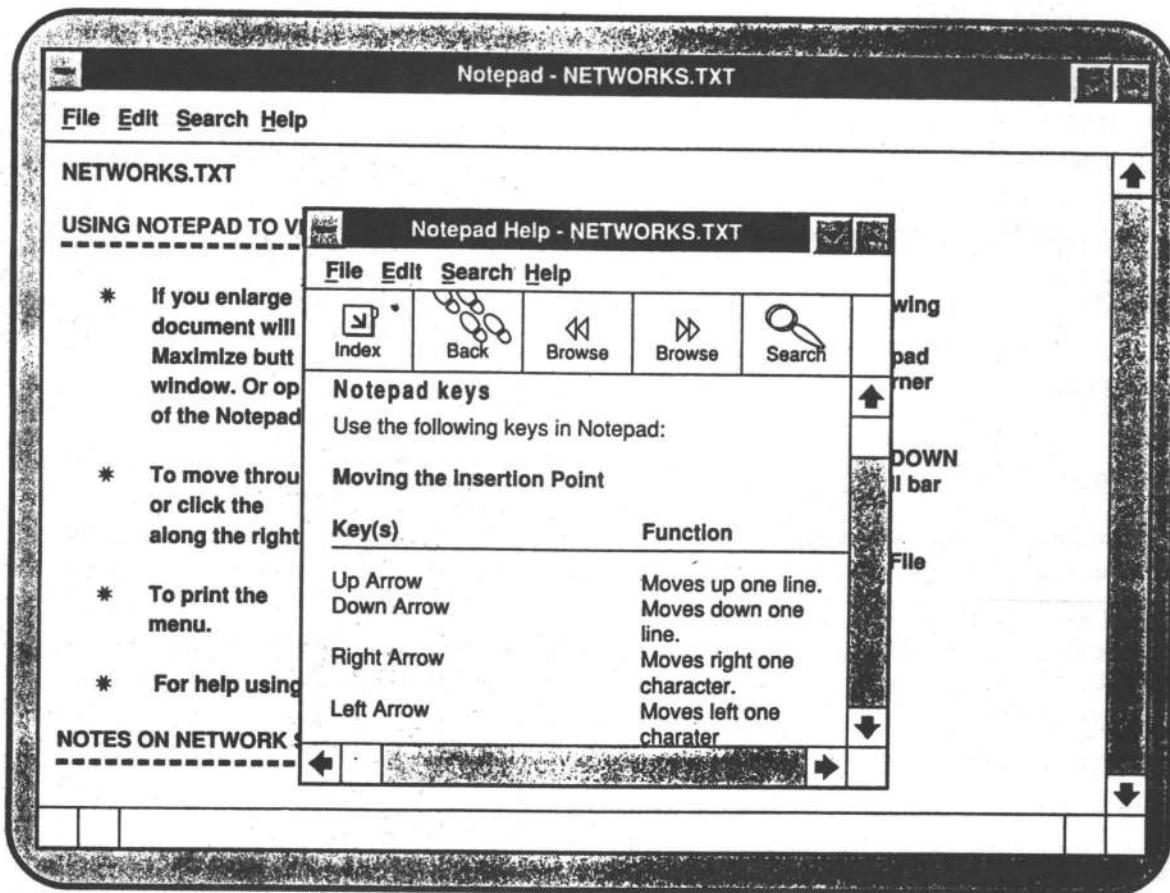
Icon

Banyak pemakai, terutama pemakai baru (novice), memanfaatkan icon grafis dari pada kata-kata dalam dokumentasi online. Beberapa sistem bahkan memungkinkan para pemakai merancang sendiri icon-nya, sehingga mereka bisa memaksimisasi kemampuan mereka untuk mengenali pilihan pada suatu menu secara cepat. Akal atau pikiran manusia dapat mengenali gambar kecil seperti icon secara sangat cepat, sedangkan perintah teks harus dibaca lebih dahulu. Oleh karenanya waktu yang diperlukan untuk memilih opsiion program akan terkurangi.

Fasilitas Help

Para pemakai di semua tingkatan dapat diklasifikasikan bersifat ingin cepat. Para pemakai baru ingin produktif pada sistem baru secepat mungkin, dan mereka lebih senang belajar secara trial dan error (mencoba-coba). Para pemakai ahli akan cepat frustasi bila mereka dipaksa untuk menghentikan pekerjaan mereka dan diminta melihat atau mencari informasi di dalam manual. Fasilitas Help online akan memenuhi kebutuhan para pemakai di semua tingkatan pengalaman secara lebih baik.

Fasilitas help bisa berupa layar rekapitulasi perintah sederhana, yang secara serentak menjabarkan semua perintah program, sampai berupa help yang bersifat context-specific dan diagnostik, yang didasarkan pada dimana pemakai saat itu berada dalam sistem dan pada apa yang sedang ia lakukan saat help itu diminta. Sebagai contoh, jika anda sedang mencoba mengeksekusi perintah tertentu dan anda meminta help, maka fasilitas help context-specific akan memberikan bantuan pada perintah itu, dan mungkin pada perintah-perintah yang terkait, saja. Gambar 3.22



Gambar 3.22 Fasilitas help context-specific yang dijumpai dalam Microsoft Windows 3.0.

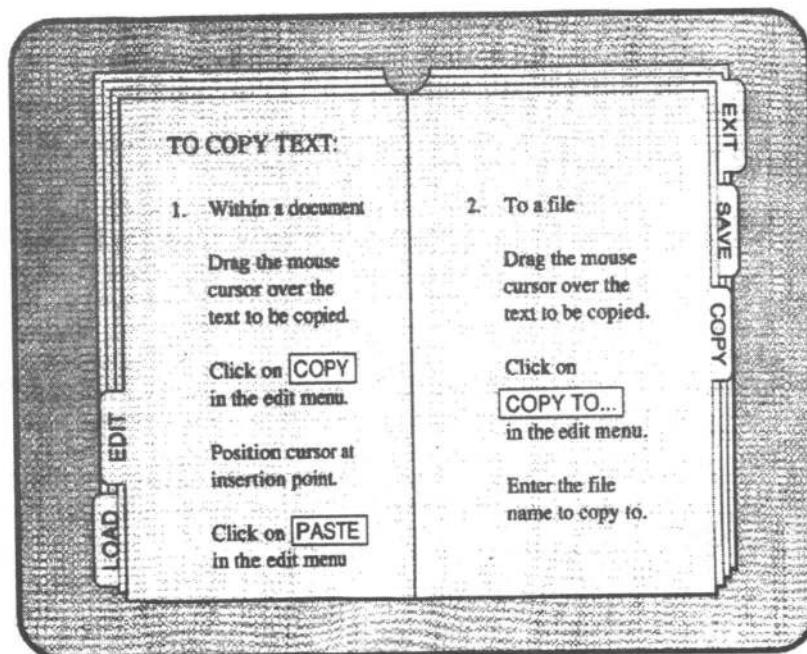
menunjukkan fasilitas help context-specific yang dijumpai dalam Microsoft Windows 3.0.

Shortcut

Pemakai ahli telah mempelajari sistem tersebut. Pemakai ini tidak memerlukan prompting yang disediakan oleh menu, dan ia menganggap pemilihan perintah pada menu hanya akan memperlamban dan menjadikan tidak efisien. Para pemakai seperti ini sebaiknya diberi shortcut yang memungkinkan mereka mengakses perintah sambil melewati sebagian besar atau semua interface menu.

Shortcut yang ideal adalah shortcut yang bisa disediakan oleh tombol-tombol pada keyboard untuk memilih menu yang ada. Untuk mencetak laporan penjualan mungkin pemakai harus menyebarangi (melewati) tiga tingkat menu sebelum perintah untuk mencetak dapat dihadirkan oleh pemakai tersebut. Orang yang berpengalaman akan bisa melewati tiga tahapan itu dengan hanya menekan rangkaian tombol "SHIFT P," misalnya. Banyak waktu yang bisa dihemat apabila perintah dapat dihadirkan (issued) dari keyboard. Ini memungkinkan tangan tetap berada pada posisinya dan tidak harus meraih perangkat pointing berbasis-mouse.

Terakhir, kita mungkinkan agar pemakai ahli bisa mengaitkan serangkaian perintah pada satu baris perintah dari pada memasukkan setiap perintah secara individual. Ekstensi atau perluasan dari ini adalah macro, yang kadang-kadang disebut script file. Script file ini memungkinkan pemakai merekam serangkaian keystroke (pemencetan/penekanan tombol) untuk operasi yang repetitif. Fasilitas seperti ini mempunyai manfaat, karena ia bisa mengurangi penginputan keyboard dari ratusan penekanan sampai hanya dua atau tiga penombolan saja.



Gambar 3.23 Contoh manual referensi online yang dibuat kelihatan/nampak seperti buku.

Manual Referensi Online

Manual Referensi Online biasanya akan memberikan semua informasi mengenai manual kertasnya (manual sejenis yang dituliskan pada kertas), namun banyak dijumpai bahwa para pemakai hanya akan menggunakan manual seperti itu untuk referensi cepat, tidak untuk dibaca dan dipelajari secara lama. Oleh karena itu, meskipun manual online harus menduplikasi susunan logis manual kertas, ia memerlukan fasilitas pemanggilan informasi tambahan.

Ketika menciptakan manual referensi seperti itu, kita harus membuat manual yang berversi online seperti manual kertasnya, yakni dengan disertai tabel isi, indeks, daftar istilah, dan heading. Gambar 3.23 menunjukkan ssuatu pengimplementasian manual referensi online, yang dibuat kelihatan seperti buku (yang ditempatkan) pada layar. Selain itu, para perancang harus menyertakan menu untuk mengakses perintah pencarian informasi dan memberikan hubungan antara topik-topik yang berkaitan sehingga memungkinkan pemakai melompat secara langsung ke topik-topik tersebut tanpa melakukan pencarian lagi. Agar mendapatkan hasil yang baik, para perancang harus selalu menggunakan tugas-tugas pokok yang dijalankan oleh program tersebut sebagai heading atau nama topik untuk manual itu.

TINJAUAN SASARAN BELAJAR UNTUK BAB INI

Tujuan utama bab ini adalah untuk memungkinkan setiap siswa memahami enam sasaran belajar yang penting. Rekapitulasi atau ringkasan dari sasaran belajar bab ini adalah sebagai berikut.

Sasaran belajar 1:

Membandingkan bahasa generasi-keempat (4GL) dengan bahasa generasi-kelima (5GL).

Bahasa generasi-keempat membantu pemakai mendefinisikan atau menetapkan keperluan dengan cara memberitahu atau menyediakan apa yang akan dikerjakan dan kemudian memungkinkan pemakai bertindak. Bahasa generasi-keempat berisi fungsi-fungsi canggih yang menjalankan fungsi host, seperti analisis statistikal atau finansial. Seringkali para pemakai dapat mengkode aplikasi ad hoc mereka sendiri.

Bahasa generasi-ketiga lebih unggul untuk aplikasi rancangan perangkat lunak I/O-bound yang transaction-intensive. Untuk beberapa aplikasi, 3GL bersifat sebaik (sepadat) 4GL. Karena 3GL terkompilasi, ia memberikan efisiensi kepada mesin yang lebih besar/ unggul. Bahasa generasi-ketiga seperti COBOL bersifat portabel antara dan kompatibel dengan platform-platform aplikasi. Dengan munculnya generator kode CASE untuk 3GL, khususnya COBOL, produktivitas pengkodeannya bisa lebih unggul dari pada 4GL. Apabila 3GL telah dikode, ia biasanya lebih mudah diuji dan dipelihara, jika dirancang dan didokumentasi secara baik, dari pada 4GL.

Sasaran belajar 2:

Mendeskripsikan bahasa pemrograman berorientasi-obyek (OOP), menyebutkan lima bahasa OOP yang penting, dan secara singkat menjelaskan perangkat OOP.

Pengimplementasian metode (yakni, kode yang mengeksekusi operasi-operasi obyek) dilakukan dengan menggunakan bahasa OOP. Bahasa OOP bervariasi dalam mendukung konsep rancangan berorientasi-obyek. Tak ada satu bahasa yang mendukung setiap konsep rancangan berorientasi-obyek. Oleh karena itu, pemilihan bahasa OOP didasarkan pada konsep yang sama seperti pemilihan bahasa lain. Bahasa tersebut harus sesuai dengan aplikasi rancangan perangkat lunak. Sebagai contoh, jika rancangan perangkat lunak memerlukan kemampuan pewarisan yang banyak, maka bahasa OOP yang dipilih harus mendukung pewarisan yang banyak. Dengan memilih bahasa OOP yang mempunyai perpustakaan kelas yang ekstensif dan matang, maka produktivitas pengembangan akan bisa ditingkatkan, seperti peningkatan suatu aplikasi set perangkat OOP berbasis-window point-and-click menu-driven. Lima bahasa OOP yang perlu kita ingat adalah:

- ◆ Smalltalk
- ◆ Eiffel
- ◆ C++
- ◆ Object Pascal
- ◆ Object COBOL

Perangkat bahasa penggunaan-khusus (special-purpose) memungkinkan para pemakai akhir (end user) yang hanya memiliki sedikit pengetahuan mengenai komputer dan pemrograman bisa menggunakan sistem pada aplikasi ad hoc lokal mereka sendiri. Bahasa-bahasa penggunaan-khusus adalah:

- ◆ Perangkat bahasa berorientasi-pemakai interaktif
 - ◆ Bahasa query DBMS
 - ◆ Hypertext dan multimedia (atau hypermedia)
-

Untuk aplikasi rancangan perangkat lunak dikembangkan dengan menggunakan pendekatan berorientasi-struktur, maka bahasa yang dipilih harus mendukung pendekatan ini. Contoh-contoh bahasa untuk itu adalah Ada, C, COBOL, dan Pascal. Jika aplikasi rancangan perangkat lunak dikembangkan dengan menggunakan pendekatan berorientasi-obyek, maka kita harus memilih bahasa OOP (pemrograman berorientasi-obyek). Contohnya bahasa OOP ini adalah Smalltalk, C++, Object Pascal, dan Object COBOL.

Untuk aplikasi pemrosesan transaksi yang besar, biasanya lebih disarankan untuk menggunakan 3GL. Untuk aplikasi ilmiah dan mekanik yang memerlukan sejumlah besar pemrosesan internal, disarankan untuk menggunakan bahasa berorientasi-matematis seperti FORTRAN. Untuk sistem pendukung keputusan (DSS), sistem informasi eksekutif (EIS), dan aplikasi ad hoc, umumnya akan lebih tepat jika kita menggunakan kombinasi antara 4GL, bahasa OOP, dan bahasa penggunaan-khusus. Namun demikian, akan tepat pula jika kita gunakan 3GL, misalnya COBOL yang didukung oleh perangkat CASE COBOL.

Sasaran Belajar 5:

Menjelaskan pemilihan bahasa yang berkaitan dengan masalah seperti tingkat penggunaan dalam dunia bisnis, kemudian mengekspresikan, kemudahan, portabilitas, kemampuan bisa dipelihara, dan kemampuan perluasan.

Tingkat penggunaan bahasa dalam komunitas bisnis berarti bahwa bahasa tersebut akan didukung secara luas oleh:

- ◆ Para pabrikan hardware komputer, vendor CASE, dan sejumlah besar programmer yang terampil dalam aplikasinya.
- ◆ Kelompok-kelompok yang akan menciptakan dan menyebarluaskan standart untuknya dan merevisinya untuk memenuhi kebutuhan saat itu.

Bahasa yang dikode dalam bentuk suatu program adalah pengejawantahan (perwujudan) dari rancangan perangkat lunak. Oleh karena itu, ia harus mengekspresikan rancangan ini sedemikian rupa sehingga non-programmer pun dapat memahaminya.

Jelasnya, dengan bahasa yang lebih mudah dan jelas, kita akan bekerja dengannya secara lebih baik. Sebagai contoh, dengan adanya perpustakaan kelas yang ekstensif dan kaya berarti banyak obyek yang tidak perlu dikembangkan/dibuat oleh programmer. Alat pengembangan perangkat lunak yang tersedia guna melihat-lihat perpustakaan kelas, mengedit kode sumber, melakukan compiling, debugging, dan sebagainya, dapat meningkatkan kemudahan dan meningkatkan produktivitas.

Sejumlah proyek sistem mungkin secara ideal cocok untuk bahasa tertentu yang mempunyai banyak dialek berdasarkan sudut pandang fungsionalitasnya. Namun demikian, sebaiknya kita tidak memilih bahasa seperti ini, karena ia tidak mempunyai standart, dan ini akan memperburuk masalah portabilitas.

Setelah program perangkat lunak dikonversi ke operasi, orang-orang yang merancang, mengkode, dan menguji program tersebut tidak boleh dipekerjakan oleh perusahaan yang mereka layani itu. Orang-orang akan masuk dan keluar. Oleh karena itu, bahasa dimana program tersebut dikode di dalamnya harus sepenuhnya bersih dari mereka yang akan memelihara kehidupan program itu. Dalam bisnis, program umumnya tidak dalam keadaan operasi selama 10 sampai 20 tahun, mungkin lebih. Agar tidak tergantung sepenuhnya dari dukungan vendor dan programmer atas kehidupan program tersebut, kita bisa membebankan tanggung jawab pemeliharaan pada perusahaan.

Biasanya, kebanyakan perangkat lunak akan memerlukan perluasan. Oleh karenanya, bahasa menyediakan teknik modularitas. Modularitas mendukung kemampuan perluasan maupun kemampuan penggunaan kembali.

Sasaran belajar 6:

Mendeskripsikan dokumentasi perangkat lunak, dokumentasi operasi, dan dokumentasi pemakai.

Dokumentasi perangkat lunak meliputi dokumentasi internal dan eksternal. Dokumentasi internal merupakan bagian terpadu dari kode sumber program itu. Ia mencakup pemformatan dan pemasukan kode untuk readability, penggunaan nama data yang standart dan bermakna, dan penggunaan penjelasan/komentar yang bebas. Dokumentasi perangkat lunak eksternal selalu berbasis-kertas. Ia mencakup halaman judul, berbagai perangkat modeling yang merepresentasikan logika dan rancangan perangkat lunak, deskripsi input dan output, lembaran perubahan program, dan kasus uji.

Dokumentasi operasi biasanya berada dalam bentuk manual berjalan (run manual), yang berisi lembaran run (run sheet). run sheet mencakup identifikasi program dan waktu kapan ia dijalankan, media input, form, hardware yang digunakan, berbagai instruksi, pengendalian, dan pendistribusian output.

Dokumentasi pemakai memberikan dokumentasi online bagi para end user seperti pemakai baru (novice), pemakai kadang-kadang, pemakai transfer, dan pemakai ahli. Dokumentasi online untuk ketiga pemakai pertama tersebut selalu berisi campuran dari:

- ◆ Tutorial
- ◆ Pesan
- ◆ Menu
- ◆ Icon
- ◆ Fasilitas Help

Untuk pemakai ahli, dokumentasinya biasanya meliputi:

- ◆ Shortcut
- ◆ Manual referensi online

DAFTAR PERIKSA PENGODEAN PERANGKAT LUNAK

Tahap pengkodean perangkat lunak mengkonversi rancangan perangkat lunak ke dalam program perangkat lunak dengan menggunakan bahasa pemrograman komputer atau kombinasi dari bahasa dan perangkat. Program perangkat lunak akhir didokumentasi secara penuh. Berikut ini adalah daftar (urut-urutan) tentang bagaimana memastikan bahwa bahasa yang tepat telah kita pilih, bahwa tahap pengkodean perangkat lunak telah dijalankan secara benar, dan bahwa program perangkat lunak akhir telah didokumentasi secara lengkap.

1. Jangan memulai pengkodean perangkat lunak secara prematur (terlalu dini). Kita harus menyelesaikan aplikasi rancangan perangkat lunak lebih dulu dan ia harus diperiksa melalui tahap pemeriksaan rancangan perangkat lunak terstruktur.
2. Pilihlah suatu bahasa (atau campuran bahasa) yang tepat atau sesuai untuk aplikasi rancangan perangkat lunak. Bahasa seperti ini harus merupakan bahasa yang banyak digunakan di komunitas bisnis. Ia juga harus ekspresif, mudah (jelas), portabel, bisa dipelihara (maintainable), dan bisa diperluas. Bahasa yang dipilih tersebut harus merupakan bahasa yang banyak didukung oleh vendor dan komunitas bisnis dalam jangka yang panjang.
3. Untuk dokumentasi perangkat lunak, siapkan halaman judul; sertakan model-model untuk program perangkat lunak yang didasarkan pada perangkat modeling, deskripsi input dan output, dan lembaran perubahan program.
4. Untuk dokumentasi operasi, siapkan/buatlah lembaran run untuk manual berjalan.
5. Untuk dokumentasi pemakai, pertama kali klasifikasikan para pemakai dan tentukan kebutuhan dokumentasi mereka. Kemudian ciptakan suatu campuran tutorial, pesan, menu, icon, dan fasilitas help untuk memenuhi kebutuhan ini. Untuk pemakai ahli, sediakan shortcut perintah dan manual referensi online rinci.

PERTANYAAN TINJAUAN

- 3.1 Jelaskan mengapa 4GL bisa tidak lebih padat/ringkas dari pada 3GL.
- 3.2 Jelaskan mengapa 3GL lebih bersifat machine-efficient dari pada 4GL.
- 3.3 Jelaskan mengapa 3GL lebih tahan dan fungsional dari pada 4GL.

- 3.4 Jelaskan mengapa 3GL, khususnya COBOL, bersifat portabel untuk ditempatkan dengan berbagai arsitektur komputer. Jelaskan mengapa 4GL tidak demikian.
- 3.5 Jelaskan bagaimana 3GL bisa dikode secara lebih cepat dari pada 4GL.
- 3.6 Jelaskan bagaimana sistem CASE dapat meningkatkan produktivitas pengembangan 3GL.
- 3.7 Apa yang dimaksud bahasa OOP murni? Sebutkan satu. Apakah bahasa OOP hybrid itu? Sebutkan satu. Mereka itu cocok untuk apa?
- 3.8 Dapatkah 3GL digunakan untuk mengkode rancangan berorientasi-obyek? Jelaskan jawaban anda.
- 3.9 Adakah bahasa OOP yang mendukung dan mengimplementasikan semua konsep rancangan berorientasi-obyek? Jelaskan jawaban anda.
- 3.10 Apa komponen pokok dari suatu set perangkat pengembangan bahasa OOP yang meningkatkan produktivitas dan kemampuan penggunaan kembali?
- 3.11 Sebutkan tiga perangkat bahasa penggunaan-khusus. Jelaskan tujuan (penggunaan) mereka dan jabarkan bagaimana mereka digunakan.
- 3.12 Disamping daya dan fungsionalitas bahasa yang berkaitan dengan aplikasi rancangan perangkat lunak, sebutkan dan jabarkan secara singkat enam persoalan lain dalam pemilihan bahasa.
- 3.13 Deskripsikan dokumentasi perangkat lunak dan elemen-elemen esensialnya.
- 3.14 Sebutkan isi pokok dari suatu manual berjalan. Untuk siapa manual berjalan dibuat?
- 3.15 Sebutkan dan jabarkan secara singkat empat kategori end user. Jenis dokumentasi apa yang harus dibuat atau disiapkan untuk setiap kategori itu?
- 3.16 Jabarkan karakteristik pesan on-screen yang terancang dengan baik.
- 3.17 Sebutkan beberapa alasan mengapa perancang suatu menu mungkin ingin mengurangi penekanan terhadap pilihan menu tertentu.
- 3.18 Apa yang dimaksud macro, dan apa keuntungan yang didapat oleh pemakai ahli darinya?

SOAL SPESIFIK BABINI

Soal-soal ini membutuhkan jawaban pasti yang didasarkan secara langsung pada konsep dan teknologi yang dikemukakan dalam buku atau bab ini.

- 3.19 Isikan F untuk pernyataan yang salah atau T untuk pernyataan yang benar di sebelah setiap frase yang menjelaskan 3GL berikut ini.

Dikembangkan secara algoritmis tahap demi tahap.

Sangat padat/ringkas/berisi.

Mudah dipelajari.

Sulit didokumentasi.

Bersifat machine-efficient.

Bersifat proprietary.

Partisipasi pemakainya besar.

Sulit dipelihara.

- 3.20 Tunjukkan apakah frase-frase berikut ini T (benar) atau F (salah), dalam kaitannya dengan 4GL.

Siklus pengembangannya lama.

Sangat bertele-tele (tidak ringkas).

Bersifat prosedural.

Sulit dipelajari.

Sangat bagus untuk prototyping.

Partisipasi pemakainya sedikit.

Mudah dipelihara.

- 3.21 Ceklah istilah-istilah yang berlaku (bisa diterapkan) untuk bahasa OOP.

Hybrid

Murni

GO TO

REDEFINES

Kemampuan penggunaan kembali (reusability)

ISDN

Perpustakaan kelas

Pewarisan

C++

FORTRAN

Smalltalk

Modem

Metode

_____ Kemampuan perluasan (extendability)
_____ VSAT

- 3.22 Relasi STUDENT dikomposisi (tersusun) dari atribut-atribut berikut ini: S_Name, SSN, S_Address, Major, dan GPA.

Ditanyakan: Rancanglah perintah-perintah SQL yang:

1. Memberi nama dan alamat semua siswa yang major (mata kuliah pokok)-nya adalah sistem.
2. Memberi nama dengan urutan alfabetis kepada semua siswa yang mata kuliah pokoknya akunting, yang perempuan, dan yang mempunyai grade point rata-rata (GPA) lebih besar dari pada 3,0.

- 3.23 Anda sedang merancang interface query-by-example (QBE) untuk petugas pengontrolan inventarisasi. Nama tabel anda adalah INVENTORY. Tabel INVENTORY ini mencakup tiga atribut:

- ◆ Deskripsi
- ◆ Kuantitas yang ada saat itu
- ◆ Harga penjualan

Ditanyakan: Gambarlah tabel tersebut dan masukkan tiga entri QBE berikut ini: (1) Cetak semua sekop yang harga penjualannya lebih dari \$30,00, (2) Hapuslah sekop salju, dan (3) Naikkan harga penjualan sekop ladang sebesar 15 persen.

- 3.24 Anda sedang duduk di suatu workstation yang mempunyai akses ke fasilitas rancangan multimedia dan hypertext. Anda telah melihat-lihat sepanjang laporan penjualan yang bertumpuk dan menjumpai paragraf pokok yang menjabarkan kinerja penjualan departemen B. Anda ingin memanggil fragmen teks ini dan mengkopinya dalam workspace multimedia anda. Untuk memberikan atau menunjukkan presentasi grafis kinerja penjualan departemen B itu, anda memilih diagram kue dari space grafik, dimana space grafik ini juga menyediakan diagram balok dan diagram baris. Anda memilih printer laser untuk menggenerasi hard copy workspace multimedia lengkap anda, yang berisi fragmen teks dan diagram kue yang telah anda pilih.

Ditanyakan: Gambarlah skematisasi dari aplikasi multimedia dan hypertext tersebut dan jelaskan bagaimana ia bekerja.

- 3.25 Anda adalah seorang programmer yang sedang menciptakan interface menu untuk sistem informasi yang digunakan oleh biro pengacara. Sistem ini berbasis mainframe, dan sejumlah tingkat privilege yang

berbeda disediakan untuk berbagai tingkatan pemakainya. Tingkatan ini adalah dari akses sangat terbatas yang disediakan untuk sekretaris sampai akse penuh yang disediakan untuk operator sistem tersebut.

Ditanyakan: Sebutkan dan deskripsikan beberapa teknik untuk mengurangi penekanan pilihan tertentu pada menu anda, misalnya setiap pemakainya (yang mempunyai tingkatan berbeda) hanya akan mempunyai akses ke perintah tertentu menurut tingkat privilege mereka.

SOAL UMUM

Soal-soal ini membutuhkan jawaban berupa pendekatan/cara yang layak, bukannya pemecahan yang presisi. Walaupun soal-soal ini didasarkan pada bahan pada bab ini, kita memerlukan bacaan tambahan dan memerlukan kreativitas untuk mengembangkan pemecahan atau solusi yang bisa bekerja/bisa diterapkan.

- 3.26 Di masa lampau, COBOL banyak mendapat kritikan yang negatif. Beberapa ilmuwan komputer dan perekayasa perangkat lunak mendapatkan cercaan yang menyatakan bahwa mereka membuat "verbose dinosaur of a language" (bahasa tidak ringkas yang bertele-tele). Banyak "ahli" meramalkan kematian bahasa itu tidak lama lagi. Para penganjur 4GL dan penganjur selain 3GL (misalnya, penganjur PL/1) telah mencoba mengganti COBOL tanpa merusaknya. Sekarang, para penganjur COBOL mengatakan bahwa ia lebih kuat dari pada sebelumnya. Memang, kenyataan masih menyatakan bahwa COBOL adalah bahasa komputer bisnis yang banyak dikenal dan banyak digunakan. Seperti yang dikatakan oleh seorang ahli. Bahasa lain ini, khususnya 4GL, tidak memenuhi permintaan kekuatan industrial karena kesulitannya dalam menghadapi dunia nyata. Apakah anda ingin menggunakan kode sumber 4GL ataukah 3GL tak terkenal yang dikembangkan oleh orang lain dan yang aslinya ditulis untuk platform komputer yang berbeda, yang menggunakan perpustakaan fungsi yang berbeda pula? Saya harap anda menyukai cerita detektif. Sebaliknya, lihat COBOL atau Ada, suatu bahasa cast-iron yang mengatur apa yang dikode oleh pengkode dan yang mengatur bagaimana pengkode ini mengkodenya. Kenyataannya, saya baru saja melihat laporan riset dan setelah saya menjalani hal-hal yang ada di dalamnya, seperti memprogram perangkat pengembangan dan memelihara program yang dikode oleh seseorang, saya sadar bahwa saya menemukan kembali COBOL. Semakin segala sesuatunya berubah, semakin saja mereka tetap sama.

Ditanyakan: Jelaskan mengapa para ahli telah meramalkan kematian COBOL, sedangkan COBOL semakin kuat sekarang dari pada sebelumnya. Jelaskan mengapa COBOL dianggap sebagai "bahasa kaku/sukar yang dapat memenuhi permintaan kekuatan industrial."

- 3.27 Perusahaan tempat anda bekerja dulunya melakukan pengkodean perangkat lunak dengan bahasa non-standart yang tak terkenal dan gagal mengembangkan dokumentasi. CIO yang baru kini menyatakan: "Kita akan mengembangkan sistem baru yang benar, standart, dan bisa dipelihara. Kelemahan dari segala sistem adalah pada masalah dokumentasinya. Tanpa dokumentasi yang jelas, baru, dan benar, kita bahkan tidak akan mendapatkan suatu sistem."

Ditanyakan: Anda telah dipilih oleh CIO tersebut untuk menyelidiki dan melaporkan kepadanya bagaimana perangkat lunak aplikasi standart yang terdokumentasi dengan baik dapat dikembangkan di masa yang akan datang. Buatlah laporan ini. Lihat Bab 1 dan 2 untuk membuat atau menyiapkan laporan anda tersebut.

- 3.28 Suatu compiler menerjemahkan semua kode sumber dalam suatu program ke instruksi bahasa mesin sebelum instruksi-instruksi yang dikode dieksekusi. Pengeksekusian program umumnya akan terjadi beberapa kali tanpa perlu menyusun (compile)-nya kembali. Sebaliknya, interpreter (atau translator) menerima instruksi kode sumber satu persatu dan dengan segera menggenerasi dan mengeksekusi bahasa mesin yang bersesuaian dengannya, tanpa harus melihat dulu bagaimana bentuk instruksi yang dikode berikutnya. Compiler sama dengan translation (penerjemahan) edisi bahasa Inggris dari *War and Peace*. Interpreter agak sama dengan penerjemahan kalimat demi kalimat secara serentak yang dilakukan di United Nations (PBB).

Ditanyakan: Metode yang mana, kompilasi ataukah penerjemahan, yang lebih bersifat machine-efficient? Metode mana yang anda sarankan untuk aplikasi pemrosesan data bisnis? Metode mana yang akan akan bekerja dengan baik untuk aplikasi random, interaktif, dan real-time. Jelaskan saran atau usulan anda tersebut.

- 3.29 Berikut ini adalah beberapa aplikasi yang telah dirancang dan sekarang memerlukan pengkodean:

Sistem akunting besar yang mempunyai file account payable dan account receivable yang besar.

Aplikasi simulasi dengan (yang mempunyai) banyak persamaan yang kode sumbernya harus memberikan efisiensi kepada mesin.

Aplikasi ilmiah yang mensimulasi tubukan planet-planet. Aplikasi ini melibatkan ribuan komputasi matematis yang kompleks. Efisiensi pengkodean lebih dipentingkan dari pada efisiensi mesin.

Pemakai harus mengelola volume informasi yang besar dan mengkonversinya ke beberapa presentasi.

Pemakai perlu mengakses DBMS relasional pada basis ad hoc.

Pemakai perlu melakukan dialog dengan sistem itu. Sistem penelusuran marketing untuk manajer marketing.

Aplikasi yang akan berisi obyek-obyek reusable dalam jumlah besar.

Ditanyakan: Pilihlah bahasa yang anda yakini paling tepat untuk aplikasi-aplikasi di atas. Jelaskan alasan pilihan anda itu.

- 3.30 Anda adalah salah satu anggota tim pemrograman yang terdiri dari lima orang. Tim ini baru saja menyelesaikan kode, yang dituliskan dalam C++, untuk modul program utama sebagai bagian dari proyek sistem yang besar.

Ditanyakan: Buatlah gambaran garis besar paket dokumentasi program eksternal lengkap untuk diserahkan kepada CIO perusahaan anda. Kerangka/gambaran ini harus menyebutkan semua dokumen operasi dan perangkat lunak berbasis-kertas untuk dimasukkan dalam paket dokumentasi tersebut.

- 3.31 Deskripsikan lima pesan yang terancang secara tidak baik yang anda lihat selagi anda menggunakan program-program perangkat lunak mikrokomputer, minikomputer, atau mainframe. Jabarkan secara singkat konteks program tempat munculnya setiap pesan, dan jelaskan mengapa anda merasa bahwa setiap pesan tersebut terancang secara tidak baik. Bagaimana pesan-pesan ini diperbaiki?

BACAAN YANG DISARANKAN

Alderson, Rusty. "Standards: Good, Bad, or Ugly?" *Inferact*, May 1991.

Beckman, Michael. and Dimtry Lenkov, "The C++ Language." *Interact*, May 1991.

Booch, Grady. *Object Oriented Design*. Redwood City. Calif.: Benjamin/Cummings, 1991.

- Chare, Tim. "Everything You Wanted to Know About Object Oriented Programming But Were Afraid to Ask." *Interact*, February 1991.
- Coffee, Peter. "COBOL Evolves as Mission-Critical Tool." *PC Week*, January 21, 1991.
- Coffee, Peter. "Heavy-Duty Applications Need Cast-Iron Tools." *PC Week*, May 28, 1990.
- Cox, Brad J. *Object Oriented Programming*. Reading, Mass.: Addison-Wesley, 1986.
- Crabb, Don. "OOP Tools Ease Windows Developers' Pains." *Infoworld*, February 25, 1991.
- Dern, Daniel. "Wanted: Productivity Improvements." *Digital News*, January 21, 1991.
- Dyckman, Roger J. "COOL/3000: The Object is the Subject." *Interact*, February 1991.
- Eskow, Dennis. "Programmers Find Smooth Road to PCs." *PC Week*, October 22, 1990.
- Gray, Paul. *Guide to IFPS*. 2nd ed. New York: McGraw-Hill, 1987.
- Hill, Tom. "Object Oriented Programming Using COBOL." *Interact*, April 1990.
- Horton, William K. *Designing and Writing Online Documentation: Help Files to Hypertext*. New York: John Wiley; 1990.
- Kill, D. "Anatomy of a 4GL Failure." *Computer Decisions*, February 11, 1986.
- Lee, Al. "The Problem with 4GLs." *Computer Programming Management*, Boston: Auerbach Publishers, 1989.
- McIntyre, Scott C., and Lexis F. Higgins. "Object-Oriented Systems Analysis and Design Methodology and Application." *Journal of Management Information Systems*, Summer 1988.
- Meyer, Bertrand. *Object-Oriented Software Construction*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.
- Parsaye, Kamran, IMark Chignell, Setrag Khoshafian, and Harry Wong. *Intelligent Databases: Object-Oriented, Deductive Hypermedia Technologies*. New York: John Wiley, 1989.
- Perreault, Bob, and Katie Rotzell. "Object Database Management Systems." *Interact*, April 1990.
- Pinson, Lewis J., and Richard S. Wiener. *Applications of Object-Oriented Programming*. Reading, Mass.: Addison-Wesley, 1990.
- Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design*. Englewood Cliffs, N.J.: Prentice-Hall, 1991.
- Stroustrup, Bjarne. *The C++ Programming Language*. Reading, Mass.: Addison-Wesley, 1987.
- Taylor, David. "How to Explain This Stuff to Your Boss." *Object Magazine*. July/August 1991.
- Vesely, Eric Garrigue. *A Guide to Structured, Portable, Maintainable, and Efficient Program Design*. Englewood Cliffs, N.J.: Prentice-Hall, 1989.
- Winblad, Ann L., Samuel D. Edwards, and David R. King. *Object-Oriented Software*. Reading, Mass.: Addison-Wesley, 1990.

KASUS JOCS: Pengkodean Perangkat lunak

"Saya hanya punya waktu beberapa menit untuk membahas persoalan ini," Kata Kyle Bartwell, presiden pada Peerless, Inc., yang berbicara kepada Mary Stockland, kepala

bagian informasi peerless. "Saya senang anda mau melakukan meeting singkat." Kyle mempersilakan Mary masuk kekantornya dan menutup pintu. "Anda mengerti bahwa kita mempunyai beberapa masalah dimasa lalu dengan sistem komputer kita, dan saya ingin yakin kita tepat membicarakannya saat ini."

"Apakah ada soal tertentu yang akan anda bahas, Kyle?" tanya Mary. ia (Mary) memcarci-cari alasan yang tepat untuk meeting itu dengan mengatakan, "Kami saat ini sedang mengembangkan satu sistem untuk menangani akuntansi dan estimasi biaya pekerjaan kita, seperti saya kemukakan garis besarnya dalam komite pengarah MIS, dan kita sedang mulai menganalisis kelayakan untuk mengembangkan sistem pendukung keputusan keuangan."

"Ya, saya tau tentang JOCS dan FIDS, Orang-orang akuntasi biaya dan orang-orang manufakturing benar-benar ingin mendapatkan informasi uang up-to-date. Kyle menatap gambar di atas kepala Mary, dan mengatakan lebih lanjut, "Saya cuma khawatir mengenai pengalokasian orang-orang kita untuk proyek ini?"

Dengan masih mengingat alasan meeting ini, Mary Berkata: "Apakah anda mempunyai kepentingan dengan seseorang secara khusus? Apakah anda khawatir mengenai jumlah orang yang bekerja pada proyek ini?"

"Nah, sekarang anda mengingatkannya, Biaya personel di departemen anda telah meningkat 400% selama enam bulan terakhir ini. Departemen keuangan sekarang masih menambah anggaran untuk setiap orang yang dipekerjakan di departemen anda."

Mary diam sejenak, dan kemudian berkata dengan suara yang sangat hati-hati, "Saya memulai proyek ini dengan dukungan yang kuat dari komite pengarah MIS. Saya tahu bahwa anda ingin menyediakan sumber-sumber yang diperlukan untuk mengembangkan sistem ini. Saya perlu tahu apakah anggapan saya ini telah berubah."

Kyle, dengan sedikit sinis dengan jawaban tersebut, mengatakan, "Saya memang menyediakan sumber-sumber. Saya cuma ingin memastikan bahwa kita bisa menggunakan sumber-sumber tersebut dengan cara sebaik mungkin."

"Saya dapat yakinkan anda bahwa saya percaya pada staf saya dan keputusan mengontrol usaha pengembangan kami," Suara Mary semakin terasa formal tatkala meeting itu semakin memanas. Ia terbebani oleh pemikiran-pemikirannya saat ini dan mencoba memutuskan cara untuk melanjutkan meeting tersebut. Dia berfikir, "Saya menemukan jawabanya sekarang. Manajemen tidak memberikan sumber-sumber yang diperlukan untuk menyelesaikan proyek itu. ini terjadi pada pekerjaan terakhir saya. Anda, manajemen, hanya mendapat kemajuan riilnya dan anda terlalu banyak mendapat bagian keberhasilan itu." Ia menatap langsung Kyle dan bertanya, "Apakah ada masalah dengan sumber yang saya gunakan untuk proyek ini?"

"... yang dibuat segera diefleksikan dalam dokumentasi yang disimpan dalam tempat penyimpanan sentral itu."

Mary selanjutnya menjelaskan teknologi databasenya. "Kami menggunakan sistem manajemen database yang berpadu dengan produk CASE tersebut. Setelah analis mengidentifikasi keperluan data baru, item-item itu segera ditambahkan atau dimasukkan ke kamus data."

"Rancangan layar, seperti layar untuk melihat keragaman materi untuk tugas atau pelajaran spesifik, dan rancangan laporan, seperti laporan untuk mendata keragaman materi untuk semua tugas, diselesaikan melalui produk CASE. Produk CASE kita menciptakan perpustakaan COBOL untuk layar dan laporan, yang kemudian oleh para programmer dicadangkan ke dalam program mereka dengan perintah COBOL yang disebut Copy."

"Apakah hal ini berarti layar dan laporan tersebut dapat diubah tanpa memodifikasi program COBOL?" tanya Kyle.

"Ya, jika di masa mendatang, personel kita di bagian akuntansi biaya menginginkan suatu laporan yang tersusun dengan cara yang berbeda, analis kita tinggal mengubahan rancangan laporan tersebut dan kemudian menyusun kembali program yang akan bisa mereka terapkan."

"Jadi lima kali peran COBOL?" desak Kyle.

Kami sedang menggunakan COBOL untuk menulis bagian pemrosesan program-program kita. Input, seperti data layar dan data database, berasal dari generator CASE COBOL kita. Output, seperti rancangan layar dan format laporan, juga berasal dari generator CASE COBOL. Para programmer kami memanfaatkan atau mengambil kode yang dibuatkan dari produk CASE tersebut dan memadukannya ke dalam program yang bekerja yang memtransformasi input menjadi output."

"Sebelum anda melanjutkan pertanyaan, biarkan dulu saya menjelaskan mengapa kami memilih COBOL," sahut Mary sebelum Kyle mengajukan pertanyaan lebih lanjut. "Kami memerlukan suatu bahasa yang memungkinkan kami fleksibel dalam menciptakan program-program yang kompleks. Kami harus menggabungkan (mengkonsolidir) sistem computer-integrated dalam manufakturing dengan sistem pembiayaan kerja besar kami. Ini memerlukan program tersendiri yang tidak dapat diciptakan tanpa menggunakan bahasa prosedural. COBOL adalah satu-satunya bahasa yang benar-benar BISUMIN-oriented, yang menangani file dan record data dengan mudah, namun tetap memungkinkan programmer bebas menciptakan aplikasi-aplikasi yang tidak biasa."

"Aku setuju, saya tak ingin orang-orang saya di bagian akuntansi biaya dan manufakturing harus menyediakan waktu untuk mempelajari bahasa ringi seperti COBOL. Saya seringster di sana punya tinggi terlalu lama bagi saya," kata Kyle. Namun, dia tetap bersikeras tentang hal yang berharganya untuk menulis laporan komputer mereka.

"Saya setuju. Saya juga tak ingin para end user kita menjadi programmer. Namun saya sangat ingin mereka bisa menciptakan aplikasi ad hoc mereka sendiri ketika diperlukan. Oleh karena itulah, Jake Jacoby menyusun rencana kelas pelatihan untuk para end user. Ia akan menyusun pelatihan yang berkefungsian untuk mengajari para end user menggunakan bahasa database dan generator laporan."

"Saya masih ada meeting yang telah dimulai lima menit yang lalu," kata Kyle. Kedengarannya proyek pengembangan kita berjalan baik. Namun kalau ada vendor datang menawari saya produk baru, atau apabila saya membaca artikel yang membahas teknologi yang asing, saya akan membicarakannya dengan anda. Mungkin anda dapat mengajari saya cara menggunakan paket surat elektronik sehingga anda dan saya dapat bekerja secara sedikit lebih efisien juga."

4

PENGUJIAN PERANGKAT LUNAK

APA YANG AKAN ANDA PELAJARI DALAM BAB INI?

Setelah mempelajari bab ini, anda akan mampu:

- mendeskripsikan tentang pengujian perangkat lunak dan menentukan tujuannya.
- menuliskan dan menjelaskan untuk menangani kasus pengujian (test case).
- mendekskripsi apa yang harus dilakukan sebelum melaksanakan uji coba.
- mengidentifikasi sifat-sifat teknologi manusia dalam uji coba.
- menentukan cara mendekati secara simbolik tentang teknologi manusia dalam pengujian perangkat lunak.
- mendekskripsi teknologi manusia dalam pengujian perangkat lunak.

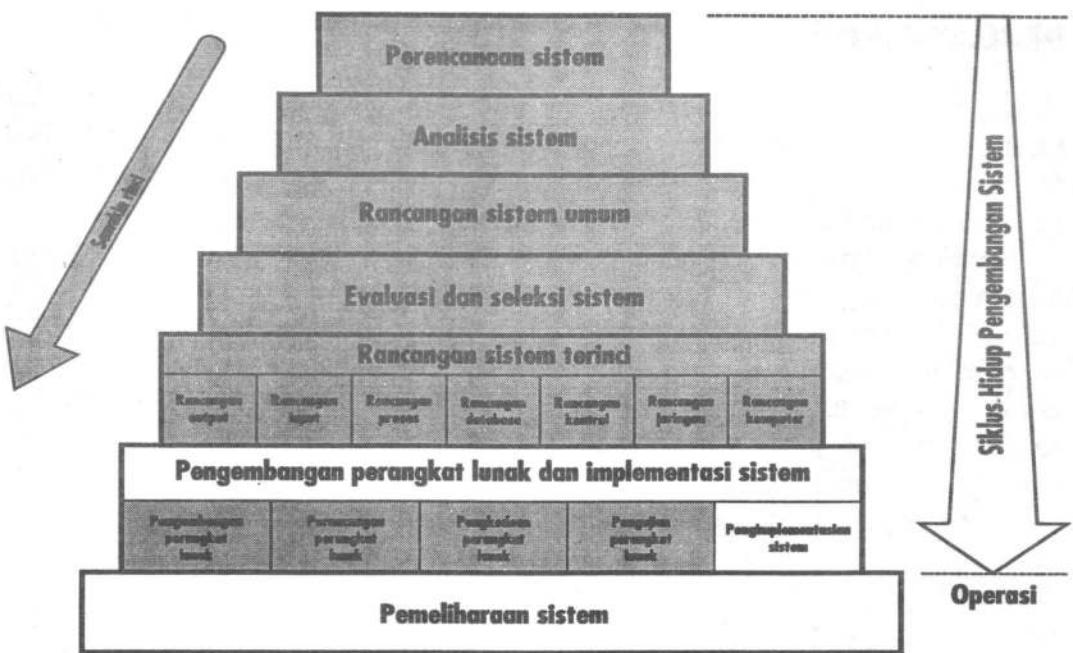
PENDAHULUAN

Tujuan bab ini adalah untuk mengemukakan cara-cara untuk menguji perangkat lunak, yang merupakan tahap terakhir dari pengembangan perangkat lunak (lihat Gambar 4.1). Banyak sistem CASE memberikan utiliti uji dan perangkat debugging yang mendukung teknik-teknik yang dikemukakan dalam bab ini.

Tujuan pengujian perangkat lunak adalah reliabilitas, yang ini memerlukan pende-tekstian dan penghapusan kesalahan. Pengujian perangkat lunak tidak dapat mem-balikkan program perangkat lunak yang terancang secara tidak baik menjadi program yang baik; namun ia dapat membantu menentukan tingkat reliabilitas sebelum perangkat lunak itu di-release untuk digunakan. Reliabilitas umumnya akan mening-kat sebanding dengan jumlah pengujian; ia sangat terkait dengan jadwal proyek dan biaya. Karena manajer proyek harus mengontrol jadwal dan biaya, maka akhirnya reliabilitas akan terkait atau didasarkan pada seberapa jauh atau seberapa bagus proyek sistem keseluruhan tersebut dikelola. Jika tahap rancangan perangkat lunak dilakukan secara terburu-buru dan secara tidak benar, maka perubahan rancangan selama tahap pengujian cenderung akan menurunkan reliabilitas, sebagai akibat dari pengubahan ini. Jika nampak diperlukan perubahan perubahan rancangan dalam jumlah besar, maka sebaiknya kita ubah jadwalnya dan kembali ke tahap rancangan SWDLC dan memulai proses design-code-test (rancangan-pengkodean-uji) lagi.

Selalu ada trade-off (timbal balik/perbandingan terbalik) antara waktu dan biaya menyelesaikan program dan reliabilitasnya. Program yang tak reliabel (handal) akan menyebabkan tim proyek mendapatkan reputasi yang jelek dan kehilangan kredibili-tas. Di sisi lain dari skala tersebut adalah masalah keterlambatan. Keterlambatan dari tanggal penyelesaian dapat juga menyebabkan hasil negatif yang sama. Yang lebih sering lagi, manajer proyek biasanya berada di pihak tengah dalam perang antara kepentingan-kepentingan yang bertentangan ini. Manajer proyek, yang dibantu oleh para anggota tim sistem dan pemakai, seringkali harus merevisi jadwal dan anggaran dan tujuan reliabilitas guna mencapai keseimbangan yang baik antara kepentingan-kepentingan yang bertentangan tersebut.

Seorang ahli mengatakan, "Seseorang yang yakin bahwa programnya akan berjalan secara benar pada kali pertama adalah seorang tolol, seorang optimis, atau mungkin programmer baru." Tanpa memandang apakah perangkat lunak tersebut dikembangkan secara swa-kelola (in-house) atau diperoleh dari vendor, pengujian perangkat lunak akan memberikan basis terdokumentasi untuk memastikan bahwa program tersebut akan berjalan sesuai harapan. Materi berikut ini membahas bagai-mana kita menjalankan proses sangat penting ini.



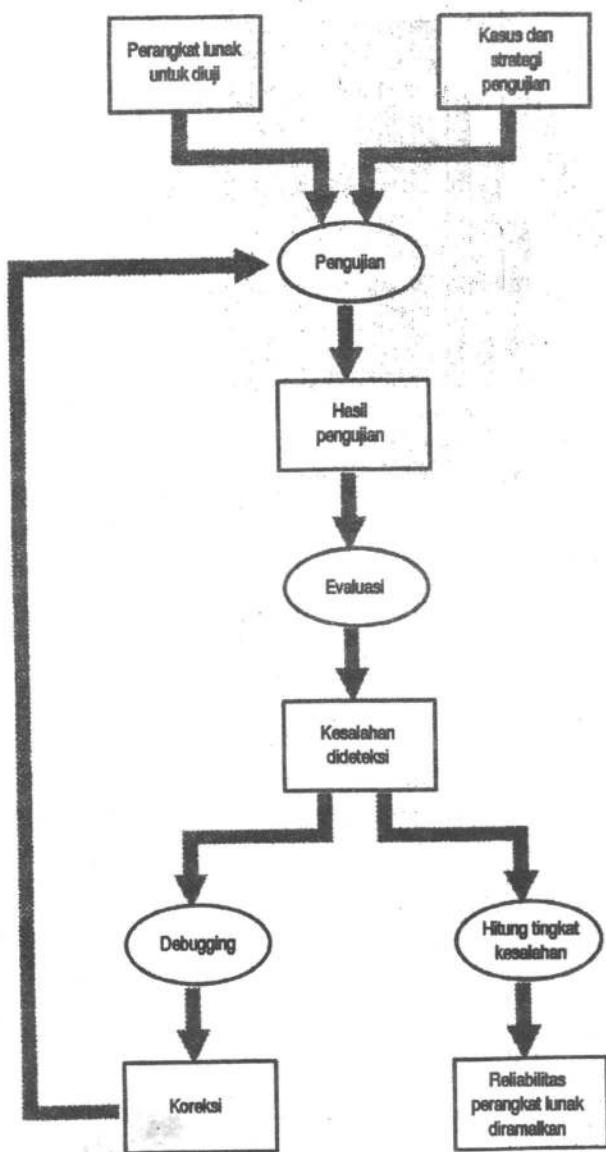
Gambar 4.1 Tahap SDLC dan bab yang terkait dalam buku ini. Dalam Bab 4, kita membahas pengujian perangkat lunak dan penanganan pendekripsi dan pengoreksian kesalahan.

APA PENGUJIAN PERANGKAT LUNAK ITU?

Pengujian perangkat lunak adalah proses yang harus mengikuti suatu pola dan rencana yang telah ditetapkan secara baik. Proses pengujian ini seringkali dijalankan oleh kelompok penjaminan kualitas yang independen, yang juga disebut tim uji (team test), untuk membantu memastikan diperolehnya paket perangkat lunak yang berkualitas tinggi dan mempunyai kehandalan (reliabilitas) yang tinggi pula dengan cara mencari dan mengoreksi kesalahan.

Menyiapkan Test Case

Pengujian dilakukan dengan cara seperti yang ditunjukkan dalam Gambar 4.2. Perangkat lunak target (yang diharapkan) perlu dikenai test case dan test strategy, yang menggenerasi hasil uji. Test Case adalah prosedur yang memeriksa perangkat lunak dan memberikan hasil yang akan menentukan penerimaan, permodifikasi,



Gambar 4.2 Skenario pengujian perangkat lunak terstruktur.

atau penolakan terhadap perangkat lunak tersebut. Test case harus dicatat dan hasilnya harus didokumentasi secara baik. Kualitas dan efisiensi pengujian sangat tergantung pada pengembangan dan aplikasi (penerapan) test case tersebut. Berikut ini adalah daftar area dimana test case biasanya dibuat:

- ◆ **Field** Ujilah atribut dari setiap field.
- ◆ **Record** Ujilah entri (pemasukan), penyimpanan, pemanggilan, dan pemrosesan record.
- ◆ **File** Ujilah pembukaan, pemanggilan, penggunaan, dan penutupan file.
- ◆ **Entri Data** Ujilah untuk mengetahui adanya ketidakakuratan, ketidaklengkapan, atau kekunoan data. Validasilah fungsi-fungsi yang diperlukan untuk memasukkan data yang benar sesuai atau menurut spesifikasi rancangan.
- ◆ **Kendali** Ceklah semua kendali tergabung yang dibahas pada Bab 12 untuk memastikan keakuratan, kelengkapan, dan kewenangan pemrosesan transaksi.
- ◆ **Arus Program** Pastikan bahwa perangkat lunak tersebut menjalankan konsepsi rangkaian, seleksi, dan repetisi secara benar.

Hasil-hasil tes ini dievaluasi. Kesalahan-kesalahan yang terdeteksi harus di-debug (ditangkap lalu dikeluarkan), dan harus dilakukan pengoreksian terhadap perangkat lunak tersebut. Jika pengoreksian tidak dilakukan, maka perangkat lunak tersebut akan ditolak.

Apa yang Dimaksud Kesalahan Perangkat Lunak dan Apa Hubungannya dengan Reliabilitas Perangkat Lunak?

Angka kesalahan dihitung untuk memprediksi reliabilitas perangkat lunak. Secara kuantitatif, **RELIABILITAS PERANGKAT LUNAK** dapat dinyatakan dalam jumlah kesalahan per KLOC yang diantarkan. Berikut adalah jenis kesalahan yang kemungkinan dihadapi oleh penguji.

KESALAHAN FATAL ada tiga jenis:

1. Crash, dimana program berhenti secara abnormal.
2. Logika, dimana program tidak menjalankan fungsi secara benar; misalnya, mengeksekusi branch yang salah atau membuka file yang salah.

-
3. Hang, dimana program atau sebagian dari program itu nampak berputar (yakni, mengulang-ulang) secara tak terbatas.

KESALAHAN SERIUS menghasilkan output yang tidak benar. **KESALAHAN MINOR** menyebabkan pemakai tidak puas dengan hasil program tersebut, seperti terjadinya kolom yang tidak sejajar pada gambar; kesalahan ini tidak serius, namun mengganggu.

Penyebab kesalahan program perangkat lunak adalah sesuatu yang disebut **BUG**. Bug adalah kerusakan, kelemahan, kesalahan, atau ketidaksempurnaan yang tak diharapkan. Jika kesalahan fatal atau kesalahan serius ditemukan, maka reliabilitas perangkat lunak dipertanyakan, dan mungkin perlu dilakukan rancang ulang dan pengkodean ulang, atau mungkin program tersebut akan ditolak. Sebaliknya, jika kesalahan minor mudah diatasi, dan modul perangkat lunak nampak berfungsi dengan baik, maka bisa kita tarik satu dari dua kesimpulan.

1. Reliabilitas perangkat lunak bisa diterima.
2. Test case tidak memadai untuk mendeteksi kesalahan fatal dan kesalahan serius.

Dengan menggunakan model error-seeding (penyemaian kesalahan), yang akan dibahas kemudian dalam bab ini, hal itu bisa membantu memastikan atau mengetahui dengan pasti efektivitas test case.

Apa yang Dimaksud Debugging?

DEBUGGING adalah pengeluaran bug. Ia bisa dilakukan karena adanya pengujian yang berhasil. Test case mendeteksi kesalahan; debugging mengeluarkan bug dan mengoreksi kesalahan. Dalam banyak kasus, suatu kesalahan (error) adalah gejala dari penyebab dasar yang tidak dapat dibuktikan atau diketahui dalam kesalahan itu. Proses debugging mencoba mencocokkan gejala dengan penyebab, yang dengan demikian mengarah pada pengoreksian kesalahan total. Agar bisa melakukan debugging secara lengkap, kita sebaiknya mencari akar penyebab suatu kesalahan.

Dalam rancangan dan pemeriksaan perangkat lunak, pertanyaan, "Apakah kita membangun perangkat lunak yang benar?" terjawab. Dalam pengujian perangkat lunak, pertanyaan, "Apakah kita membangun perangkat lunak secara benar?" terjawab. Debugging akan mengidentifikasi prosedur pengkodean dan mengontrol dan

mengungkap kode yang tidak benar, tidak sesuai, tidak efektif, tidak efisien, dan tidak standart.

Apa Tujuan Pengujian?

Tujuan pengujian adalah bisa diperolehnya perangkat lunak yang reliabel. Namun demikian, pengujian tidak pernah bisa menunjukkan bahwa perangkat lunak tersebut reliabel sepenuhnya; ia hanya menunjukkan keberadaan kesalahan, tidak membuktikan keberadaannya. Selalu dimungkinkan akan terjadi kesalahan lagi setelah dilakukan pengujian yang sangat komprehensif dan sangat teliti.

Semua profesional sistem bekerja sampai tahap pengujian ini selesai. Ketika pengujian dimulai, ada usaha untuk menghancurkan sistem tersebut. Setidaknya bagi pengkode, pengujian nampak bersifat destruktif. Namun demikian, kesalahan tidak ada, dan kesalahan ini harus ditemukan sebelum pemakai menemukannya selama operasi, dimana ini merupakan waktu yang paling tidak menyenangkan untuk mengungkap kesalahan.

Pengujian Perangkat lunak di Capital Industries

"Saya tidak suka gagasan pengujian," Kata Maurice Singh, programmer baru pada Capital Industries. "Ita membosankan, tak pernah berakhir, dan tidak menyenangkan."

"Baik atau burukkah hila kita mencari bug?" tanya Jane Cappola, pimpinan proyek.

"Saya tak tahu," jawab Maurice. "Sya pikir itu tergantung pada siapa yang mencarinya."

"Tepat," kata Jane, "Bagi saya, mencari bug adalah hal yang baik. Dalam beberapa kelompok, pencarian bug merupakan, seolah-olah programmer yang baik tidak pernah membuat kesalahan dan tidak potensial menghasilkan bug. Sebenarnya, pencarian bug berarti anda menghemat biaya dan mencegah anda malu terhadap pemakai karena bug telah dikeluarlkan. Memang, pencarian bug selama pengembangan perangkat lunak jauh lebih murah dan lebih tidak memalukan."

"Saya tidak pernah berpikir yang demikian, kata Maurice. "Saya bisa memahami apa yang anda ucapkan. Apakah kita mempunyai metodologi pengujian perangkat lunak yang bisa kita pakai?"

"Ya," jawab Jane. "Kita mempunyai pengujian white box, dan beberapa orang menyebutnya pengujian glass box. Kemudian kita mempunyai pengujian black box."

"Saya minta maaf," kata Maurice dengan muka menyesal. "Kuliah kami di universitas mengatakan mata kuliah pengujian. Apa yang dimaksud dengan pengujian white box dan pengujian black box?"

"Dalam bahasa sederhana, pengujian white box dilakukan dengan melihat kode di dalam modul – yakni, 'box-box' – untuk memastikan bahwa kode tersebut ditempatkan atau diletakan bersama secara benar. Sebaliknya pengujian black box memastikan bahwa modul-modul tersebut menghasilkan soutput yang benar. beberapa orang menyebut pengujian black box sebagai pengujian berbasis-spesifikasi atau pengujian berbasis-fungsi."

"Jadi, anda menguji kode tersebut untuk mengkonfirmasikan strukturnya dan untuk melihat apakah terdapat pengkodean yang menyimpang atau lintasan yang belum sempurna dengan white box. Kemudian, dengan pengujian black box, anda menetukan apakah program tersebut melakukan apa yang diharapkan untuk ia Kerjakan," kata Maurice.

"Ya, begitulah," jawab Jane. "Juga karena kita mempercayai pengujian di Capital ini, kita sebenarnya berpikir banyak tentang pengujian selama tahap rancangan dan oleh karena itu kita menjalankan 'pemrograman defensif'. Kita yakin bahwa perencanaan pengujian selama rancangan adalah sesuatu yang sangat sehat untuk dilakukan."

"Kedengarannya ada banyak pekerjaan buat saya," kata Maurice.

"Memang," kata Jane. "Namun kita hanya melakukan pengujian sedikit pada satu saat. Kita memberikan modul-modul ke beberapa programmer yang berbeda. Misalnya, mungkin anda bertanggung jawab untuk empat atau lima modul. Modul-modul ini tidak akan memerlukan usah pengkodean lebih dari satu atau dua hari. Kemudian, usaha pengkodean itu diikuti dengan pengujian selama satu atau dua hari. Kita padukan semua modul dari para programmer, dan menguji mereka lagi. Apabila kita puas dengan hasilnya, kita berikan modul lagi, kemudian mengkodenya dan mengujinya. Selanjutnya, modul-modul itu kita padukan dengan modul-modul sebelumnya. Kita teruskan proses ini sampai semua modul rancangan perangkat lunak tersebut dikode, diuji, dan dipadukan. Dengan langkah-langkah seperti ini, pengujian tidak akan membosankan. Moto kita di Capital ini adalah: 'Untuk melakukan perjalanan jauh, kita menggunakan langkah-langkah kecil yang banyak.'"

MERANCANG TEST CASE

Segala produk perekayasaan, termasuk perangkat lunak, dapat diuji dengan dua cara:

1. Mengetahui kerja internal suatu produk dan mengujinya untuk melihat apakah ia bisa dipraktekkan secara baik (pengujian white box).

2. Mengetahui fungsi-fungsi yang diharapkan akan dilakukan produk tersebut dan menguji produk itu untuk melihat apakah ia menjalankan fungsi-fungsi tersebut secara benar (pengujian black box).

Pengujian White Box

WHITE BOX TESTING (pengujian white box) didasarkan pada pemeriksaan langsung terhadap struktur logis internal dari perangkat lunak tersebut. Ia menggunakan pengetahuan/pemahaman struktur program tersebut untuk mengembangkan pengujian fungsionalitas program secara efisien dan efektif. Lintasan logis sepanjang perangkat lunak diuji dengan memberikan test case yang mempraktekkan set spesifik dari konsepsi-konsepsi SEQUENCE, IF_THEN_ELSE, DO WHILE, dan DO UNTIL.

Kita bisa mengasumsikan bahwa pengujian white box dapat menghasilkan program yang sepenuhnya benar. Namun lintasan logis sepanjang suatu program dapat menjadi membingungkan meskipun dalam program yang cukup sederhana. Oleh karena itu, pengujian white box dalam jumlah besar sifatnya tidak praktis. Sejumlah lintasan penting dan berisiko tinggi dapat dipilih dan dipraktekkan secara secara cermat, dengan menggunakan metode white box.

Perintah-perintah spesifik dan percabangan pengujinya adalah:

- ◆ **SELECT.** Perintah ini mengaitkan atau menghubungkan file ke perangkat input/output. Ia dapat dicek untuk melihat apakah program tersebut memproses file yang benar dan tepat.
- ◆ **OPEN/CLOSE.** Jenis perintah ini membuat file bisa (tersedia) untuk diproses. Perintah OPEN/CLOSE dalam suatu program berarti suatu file sedang disediakan (bisa) untuk diproses secara sesuai.
- ◆ **COPY REPLACING.** Perintah ini mengubah definisi item data yang dikopian ke dalam program dari perpustakaan sumber. Ia harus diperiksa untuk melihat bahwa perubahan tersebut memberikan hasil yang benar.
- ◆ **IF.** Ini bisa menjadi statemen kondisional mayor yang digunakan dalam suatu program. Ia bisa digunakan untuk mengeksekusi bagian kode yang tidak tepat dan mengandung kesalahan apabila kondisi tertentu benar atau salah.
- ◆ **PERFORM UNTIL and PERFORM WHILE.** Statemen UNTIL dan WHILE memungkinkan loop (putaran) dieksekusi. Loop tersebut harus diperiksa untuk memastikan bahwa ia diaktifkan dalam jumlah waktu yang tepat.

- ◆ **CALL.** Perintah ini digunakan untuk memanggil subprogram atau modul. Modul yang dipanggil mungkin saja bisa berupa modul yang salah atau modul yang tak tepat.

Pengujian Black Box

PENGUJIAN BLACK BOX menunjukkan bahwa fungsi-fungsi perangkat lunak operasional, bahwa output dihasilkan secara benar dari input, dan bahwa database diakses dan diupdate secara benar. Untuk melakukan test seperti itu, pemakai perlu mempunyai pengetahuan seperti yang disyaratkan dalam keperluan pemakai. Dengan demikian, pengujian black box tidak secara langsung memeriksa sintaks dan struktur logis internal dari perangkat lunak tersebut, dan oleh karenanya ia tidak menjadi alternatif dari pengujian white box.

Test case black box terdiri atas set kondisi input, baik yang secara sengaja valid ataupun tak valid, yang secara penuh mempraktekkan semua keperluan atau persyaratan fungsional dari suatu program. Umumnya, test ini akan mengungkap berbagai kelas kesalahan dari metode white box. Baik pengujian white box dan black box difokuskan pada pengungkapan kesalahan yang terjadi selama pengkodean, namun hanya pengujian black box yang difokuskan pada pengungkapan kesalahan yang terjadi dalam pengimplementasian persyaratan/keperluan pemakai dan spesifikasi rancangan sistem.

Tidak seperti halnya pengujian white box, yang dilakukan pada awal proses pengujian, pengujian black box cenderung dilakukan selama tahap-tahap akhir pengujian. Test case black box oleh karenanya lebih tepat dilakukan pada tingkat-tingkat pengujian integrasi, sistem, dan penerimaan dan biasanya tidak digunakan pada tingkat modul.

Kelas Ekuivalensi Pengujian

KELAS EKUIVALENSI (equivalence class) pengujian adalah bagian pokok dari pengujian black box. Dua nilai input akan berada dalam kelas ekuivalensi yang sama jika mereka ditangani oleh program dengan cara yang sama. Sebagai contoh, misalkan input yang bisa diterima untuk suatu field data adalah bilangan antara 1 dan 50. Maka akan membuang-buang usaha jika kita menguji 45, 38, 12, dan 6, misalnya, karena bilangan-bilangan ini berada dalam kelas ekuivalensi yang sama. Jika 38 bekerja, misalnya, maka yang lainnya mempunyai probabilitas tinggi untuk bekerja. Akan lebih efisien mendaftar atau menyebutkan satu saja bilangan antara 1 dan 50,

menguji poin akhir 1 dan 50 (pengujian bound), kemudian mengeluarkan atau memindahkan kelas ekuivalensi yang lain seperti 0, bilangan negatif, atau bilangan yang lebih besar dari pada 50.

Ada dua jenis kelas ekuivalensi: valid dan invalid. Pada contoh tersebut, bilangan 1 sampai 50 merepresentasikan kelas ekuivalensi valid. Segala bilangan di luar jangkauan ini merupakan kelas ekuivalensi invalid, karena merupakan karakter non-numerik. Pengujian yang lengkap memerlukan pengaplikasian test case valid maupun invalid.

Menggunakan Kelas Ekuivalensi untuk Membangun Test Case

Dengan menggunakan kelas ekuivalensi, kita dapat mendefinisikan atau menetapkan test case yang mencari kelas-kelas kesalahan, sehingga ini akan mengurangi jumlah total test case yang perlu dirancang. Biasanya, kondisi input merupakan nilai numerik, jangkauan nilai, set nilai yang berkaitan, atau kondisi yes-or-no. Beberapa contoh spesifik test case kelas ekuivalensi adalah sebagai berikut:

- ◆ Ceklah untuk melihat apakah total kendali dipersiapkan secara benar. Misalnya, jika 100 record test diproses, maka jumlah transaksi yang diproses harus menunjukkan 100.
- ◆ Cobalah untuk memproses transaksi sensitif tanpa otorisasi yang benar (misalnya, ubahlah batasan kredit pelanggan) dan lihatlah apakah sistem tersebut menolaknya.
- ◆ Lakukan pengecekan karakter numerik, alfabetis dan spesial. Misalnya, jika semua karakter dalam nomor pelanggan dirancang numerik, inputkan karakter alfabetis dalam field ini. Kendali yang bekerja dengan baik akan mendetect kesalahan ini sebelum pemrosesan dilakukan.
- ◆ Inputlah field dengan tanda negatif untuk melihat apakah ia ditangani sebagai nilai negatif. Dalam beberapa sistem, tanpa kendali yang benar, tanda negatif akan dikonversi ke tanda positif.
- ◆ Bagilah suatu jumlah dengan nol.
- ◆ Lakukan pengecekan validitas pada field data kunci. Misalnya, inputkan kode invalid atau cobalah memproses satu nomor departemen sebagai nomor nomor departemen lain.

- ◆ Lakukan pengecekan jangkauan dan kewajaran. Jika tak ada pekerja yang bisa bekerja lebih dari 60 jam per minggu, proseslah suatu time card dengan lebih dari 60 jam kerja.
- ◆ Lakukan pengecekan untuk mengetahui rangkaian transaksi yang benar. Apabila transaksi-transaksi dirancang untuk berada dalam rangkaian (urutan), kocoklah (supaya tidak urut) urutan beberapa transaksi test sehingga mereka di luar urutan.
- ◆ Masukkan nomor rekening dengan digit pengecekan yang telah ditentukan sebelumnya dan lihatlah apakah ia diproses secara benar.
- ◆ Gunakan unit-unit pengukuran yang berbeda dari yang diijinkan (yang biasa digunakan), misalnya menggunakan feet (kaki) untuk mengganti pound.
- ◆ Inputlah beberapa field dengan data tak lengkap atau data yang tak sempurna.
- ◆ Masukkan karakter-karakter dalam field agar menjadikannya overflow.
- ◆ Cobalah membaca dan menulis dari file yang salah.

Kelompok test harus menemukan cara yang tepat untuk merekam case. Gambar 4.3 menampilkan matriks test case yang merekam dan mendokumentasi tujuan-tujuan test, hasil-hasil yang diharapkan dari test itu, test case yang dilakukan, dan hasil-hasil sebenarnya dari test itu.

Melaporkan dan Memperbaiki Kesalahan

Segera setelah kesalahan ditemukan, laporan kesalahan harus diisi (dibuat). Laporan seperti ini ditunjukkan pada Gambar 4.4. Kita tidak perlu mengartikan saran dari penguji karena adanya kesalahan sebagai sesuatu yang salah, namun pesan ini anda artikan saja sebagai suatu gagasan yang disampaikan oleh penguji tentang cara memperbaiki kesalahan itu. Kesalahan pengkodean tidak disengaja oleh pengkode-nya. Sebagai contoh, pengkode mungkin menulis IF A > B PERFORM SOMETHING, yang sebenarnya maksudnya adalah IF A = B PERFORM SOMETHING. Kesalahan rancangan biasanya merupakan kesalahan interface pemakai, seperti penerjemahan rancangan input atau output yang tidak benar ke dalam kode program. Kesalahan dokumentasi berarti bahwa kode dan dokumentasi tidaklah cocok atau sesuai. Mungkin yang salah adalah dokumentasinya, atau kodennya, atau keduanya. Query artinya program melakukan sesuatu yang tidak bisa dipahami oleh penguji, dan oleh karenanya hal ini memerlukan penjelasan dari pengkode.

Hasil yang Diharapkan				
Tes Obyektif	Ditolak	Pesan kesalahan yang Ditampilkan	Rancangan Tes Kasus	Hasil yang Sebenarnya
Untuk menetukan apakah program mengkomputasi digit pengecekan dan menolak nomor rekening yang (nomor urutnya) diubah.	X	X	Inputkan nomor rekening yang (nomor urutnya) diubah.	Pesan kesalahan yang ditolak dan ditampilkan.
Untuk menetukan apakah nomor-nomor departemen dikenakan validitasnya.	X	X	Inputkan nomor-nomor departemen invalid.	Pesan kesalahan yang ditolak dan ditampilkan.
Untuk memverifikasi keakuratan penghitungan pembayaran jam lembur.		X	Bayarlah pekerja jama-jaman untuk waktu lembur 15 jam.	Pembayaran jam lembur dihitung sebesar 1,5 kali tarif reguler.

Gambar 4.3 Matriks test case

Error Report Number: _____		
Program Name: _____		
Report Type (1-5) _____	Severity (1-3) _____	Attachments (Y/N) _____
1-Suggestion 2-Design error 3-Coding error 4-Documentation error 5-Query	1-Minor 2-Serious 3-Fatal	If yes, describe: _____ _____ _____
Can the error be reproduced? (Y/N) _____		
Error and how to reproduce it: _____ _____ _____		
Suggested fix: _____		
Name of tester: _____ Date _____ / _____ / _____		
To be filled out by coding team		
Assigned to: _____ Date _____ / _____ / _____		
Resolution code (1-6) _____		
1-Fixed 2-Can't reproduce it 3-Can't be fixed	4-Disagree with suggestion 5-Withdrawn by tester 6-Works according to specifications	
Resolution certification Resolved by: _____ Date _____ / _____ / _____ Coder Tester		
Project manager approval: _____ Date _____ / _____ / _____		

Gambar 4.4 Laporan Kesalahan.

Attachment (pembubuhan) mencakup hal-hal seperti disk yang berisi data test, printout, memory dump, atau memo yang mendeskripsikan secara rinci tes-tes apa yang telah dijalankan. Juga, jika pengujian tidak dapat mereproduksi kesalahan, deskripsi tentang apa yang mungkin bisa menunjukkan kesalahan harus disertakan atau dimasukkan. Dalam beberapa contoh, pengujian mungkin telah mengungkap kesalahan, mencoba untuk mereproduksinya, dan gagal. Pengujian ini tidak yakin dengan bagaimana kesalahan tersebut dipicu. Untuk membantu seseorang yang bertanggung jawab memperbaiki kesalahan, pengujian harus menuliskan segala sesuatu mengenai apa yang telah terjadi (dilakukan) sebelum kesalahan tersebut dipicu. Perkiraan yang tepat bisa juga diterapkan. Jika sebagian atau semua item ini bisa membantu orang yang ditugasi memperbaiki kesalahan, maka item-item tersebut harus dibubuhkan ke (dimasukkan ke dalam) Laporan Kesalahan.

Resolusi kode menunjukkan apa yang dilakukan programmer yang ditugasi untuk menangani kesalahan yang dilaporkan tersebut. Resolusi terhadap kesalahan yang dilaporkan meliputi hal-hal berikut ini:

1. Ia diperbaiki sebagaimana yang direkomendasikan.
2. Kesalahan tersebut tidak dapat direproduksi.
3. Kesalahan tersebut tidak dapat diperbaiki.
4. Pengkode tidak setuju dengan saran pengujian.
5. Kesalahan yang dilaporkan telah ditarik oleh pengujian.
6. Kode untuk melaporkan kesalahan bekerja secara bersesuaian dengan spesifikasi rancangan.

Pengujian dan pengkode harus mencapai kesepakatan untuk resolusi akhir. Keduanya harus menandatangi dan memberi tanggal laporan tersebut untuk memverifikasi resolusi. Manajer proyek juga harus menandatanginya untuk mengesahkan resolusi akhir.

Setelah kesalahan program diperbaiki, tes yang mengungkap kesalahan pada tahap pertama ini harus diulangi. Ini adalah tes regresi. Pengujian ulang ini bisa saja menemukan kesalahan program baru yang tertutupi oleh kesalahan yang ditemukan dalam tes pertama. Variasi tambahan pada tes awal, yang digunakan untuk memastikan apakah perbaikan tersebut bekerja, merupakan bagian dari tes regresi.

Kemudian, setelah perangkat lunak tersebut dikonversi ke operasi, ia biasanya harus dipelihara. Untuk mengubah perangkat lunak tersebut, kita harus melakukan pengujian regresi untuk memastikan bahwa perubahan ini tidak mengganggu modul-

modul lain dari program tersebut; gangguan seperti ini disebut ripple effect. Test case yang dipersiapkan selama tahap pengujian SWDLC disimpan dalam perpustakaan test case untuk digunakan melakukan pengujian regresi selama tahap pemeliharaan dalam siklus hidup perangkat lunak tersebut. Dalam suatu sistem yang berbasis CASE, perpustakaan test case disimpan dalam repository (tempat penyimpanan) sentral.

Menguji Test Case

Dalam teorinya, penguji dapat mencari jumlah kesalahan dalam program terkode dengan metode yang disebut error seeding atau error bebugging. Suatu prorgam disemaikan (seeded) secara random dengan jumlah kesalahan artifisial (tiruan) yang diketahui yang merepresentasikan jenis kesalahan yang biasa dijumpai. Kesalahan seperti ini tidak diketahui oleh para penguji. Program tersebut dijalankan dengan test case. Probabilitas penemuan i kesalahan riil dalam populasi total sebesar I kesalahan yang tak diketahui bisa berkaitan atau dihubungkan dengan probabilitas penemuan j kesalahan yang disemaikan dari j kesalahan yang terbubuh (terdapat) dalam kode tersebut.

PENYEMAIAN/seeding (atau prebugging atau bebugging) suatu program akan memotivasi penguji untuk mencari atau menemukan kesalahan. Jika para penguji mengetahui ada kesalahan dalam program yang sedang dimonitor, mereka akan membangun test case yang kuat untuk menemukan kesalahan itu. Untuk melakukan hal ini, mereka juga akan mencari kesalahan riil.

Metode penyemaian kesalahan mengasumsikan bahwa reliabilitas program berkaitan dengan jumlah kesalahan yang dikeluarkan darinya. Setelah kesalahan riil dan kesalahan tersebut dideteksi selama test run, jumlah kesalahan riil yang masih ada dihitung dengan perkiraan dengan menggunakan rumus:

$$\frac{\text{jumlah kesalahan riil yang masih ada}}{\text{jumlah kesalahan tersebut yang masih ada}} = \frac{\text{jumlah kesalahan riil yang terdeteksi}}{\text{jumlah kesalahan tersebut yang terdeteksi}}$$

Proporsi kesalahan yang tidak terdeteksi membantu menentukan kualitas test case dan proses pengujian umum, yang selanjutnya akan membantu mengestimasikan reliabilitas perangkat lunak. Dengan demikian, jika 100 kesalahan disemaikan ke dalam program, dan penguji menemukan 40 diantaranya dan 200 kesalahan riil, maka terjadi kemungkinan bahwa masih ada 300 kesalahan riil yang ada dalam program itu.

$$\frac{\text{jumlah kesalahan riil yang masih ada}}{60} = \frac{200}{40}$$

jumlah kesalahan riil yang masih ada = 300

Agar metode seeding (penyemaian) memberikan aproksimasi jumlah kesalahan riil yang masih ada secara tepat; yakni, menunjukkan tingkat reliabilitas program, maka kesalahan yang disemaikan harus sama dengan kesalahan riil.

MENGEMBANGKAN STRATEGI PENGUJIAN PERANGKAT LUNAK

Strategi pengujian perangkat lunak memadukan pendekatan test case perangkat lunak ke dalam serangkaian tahapan terencana berikut ini:

- ◆ Pengujian modul
- ◆ Pengujian integrasi
- ◆ Pengujian sistem
- ◆ Pengujian penerimaan

Tahapan ini dan kemajuannya ditunjukkan pada Gambar 4.5.

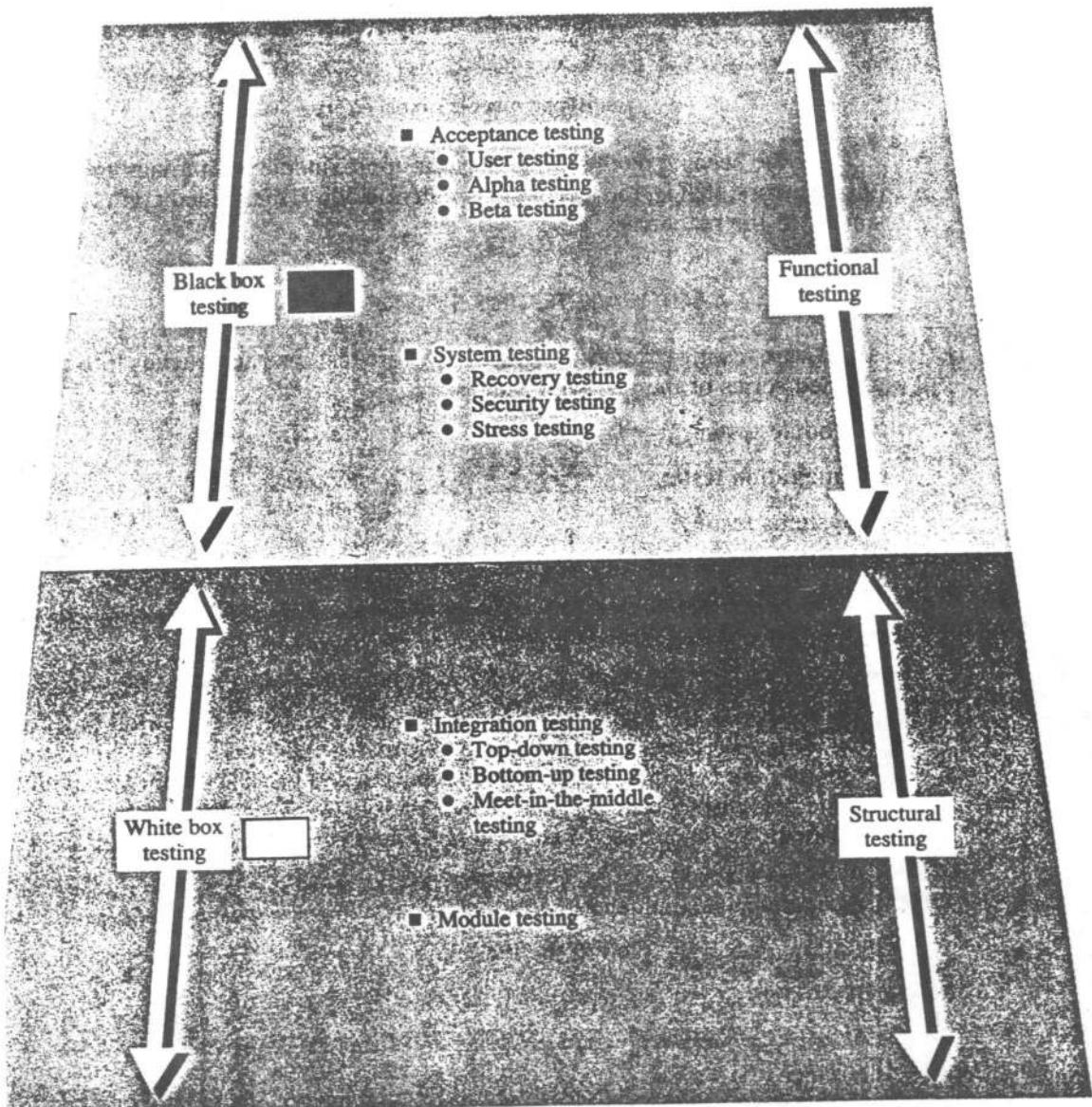
Menguji Modul Perangkat Lunak Individual

Pengujian modul adalah proses pengujian unit-unit terkecil dalam program perangkat lunak keseluruhan sebelum mereka ditempatkan bersama untuk membentuk program keseluruhan. Setelah kode program tingkat sumber ditulis, diperiksa, dan diverifikasi untuk mengetahui kebenaran sintaksnya, dimulailah perancangan test case modul.

Pengujian modul adalah bentuk pengujian white box yang paling murni. Tujuan pengujian modul adalah untuk:

- ◆ Mengeksekusi setiap perintah yang ada di dalam suatu modul.
- ◆ Mengikuti setiap lintasan logika dalam modul tersebut.
- ◆ Mengkomputasi ulang setiap perintah komputasional.
- ◆ Menguji modul dengan setiap set data input yang mungkin.

Namun demikian, kita tidak mungkin mencapai struktur modul dengan cakupan penuh, sebab jumlah test yang diperlukan akan terlalu banyak dan sulit.



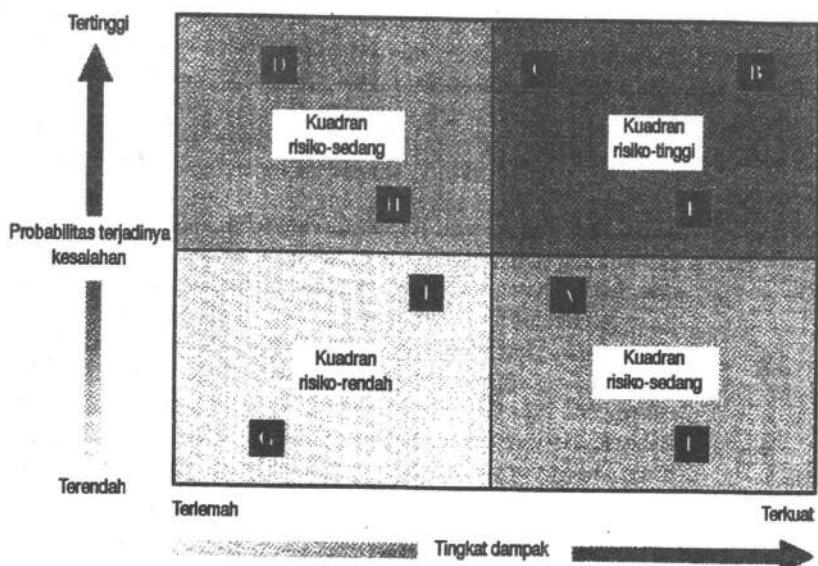
Gambar 4.5 Tahapan pengujian perangkat lunak.

Pengujian modul akan lebih sederhana dan jauh lebih berbiaya murah apabila dirancang modul-modul yang berkohesi tinggi. Apabila satu modul hanya menangani satu fungsi, maka jumlah test case akan berkurang dan kesalahan dapat lebih mudah diungkap dan dikoreksi.

Area-area yang Berisiko Paling Besar

Dalam suatu program dengan 50 branch atau lebih, akan terdapat test case yang lebih banyak dari pada banyaknya pasir di Malibu Beach. Dalam suatu program dengan n branch, maka akan ada 2^n lintasan berlainan sepanjang program tersebut, dan terjadi 2^n power yang menjadikan tambah besar secara sangat cepat. Sementara semua kombinasi branch tidak dapat diuji, setiap branch setidaknya harus diuji sekali.

Karena kita tidak dapat menguji semua kemungkinan, maka sebaiknya kita berkonsentrasi pada area-area yang berisiko terbesar. Modul-modul perangkat lunak harus diplot pada jaringan risiko berdasarkan probabilitas kesalahan mereka dan berdasarkan tingkat dampaknya. Lihat Gambar 4.6.



Gambar 4.6 Jaringan penilaian resiko modul.

Dari jaringan penilaian risiko modul tersebut, kita lihat bahwa Modul B, C, dan E mempunyai risiko tertinggi. Oleh karenanya, modul-modul inilah yang paling perlu mendapatkan pengujian, karena mereka mempunyai probabilitas kesalahan tertinggi dan mempunyai dampak terbesar terhadap program tersebut.

Dampak akan tergantung pada apa yang dilakukan program itu. Dampak terbesar berhubungan dengan kesalahan-kesalahan yang dapat menyebabkan akibat terburuk.

- ◆ Jika suatu modul berjalan (bersekusi) secara tidak benar, apa yang mungkin akan terjadi?
- ◆ Bisakah bill atau tagihan dibayarkan lebih dari sekali?
- ◆ Bisakah cek dicetak dengan nol ekstra dalam field jumlah?
- ◆ Bisakah suatu perintah menyebabkan mesin membora lubang di lokasi yang salah dalam suatu chassis truk.
- ◆ Bisakah laporan biaya berisi gambaran yang salah yang akan menyebabkan manajemen membuat tawaran kontrak yang lebih rendah.
- ◆ Apakah hasil-hasil yang tidak benar mengupdate dan merusak database?

Jawaban-jawaban ya atas pertanyaan-pertanyaan di atas bisa jadi menunjukkan adanya dampak yang besar terhadap sistem itu. Tentunya, suatu kesalahan yang merusak file receivable rekening pelanggan akan lebih buruk dari pada kesalahan yang hanya menunjukkan halaman laporan yang salah.

Probabilitas terjadinya kesalahan adalah estimasi kemungkinan modul mengalami kegagalan, yang ini akan menghasilkan dampak yang tidak diinginkan. Beberapa faktor yang mengkontribusi probabilitas yang tinggi atas terjadinya dampak tersebut mencakup:

- ◆ Ukuran dan kompleksitas suatu modul.
- ◆ Banyaknya modul yang berinteraksi dengannya.
- ◆ Keunikan/kekhasan sistem baru.
- ◆ Tingkat pengalaman pengodenya.

Setelah modul-modul tersebut dinilai menurut risiko relatifnya, tingkatan usaha pengujian dapat dialokasikan secara lebih efisien dan efektif. Penilaian modul-modul dari sudut pandang risikonya juga harus mempengaruhi urutan penulisan modul-modul itu. Umumnya, modul-modul yang berisiko paling tinggi harus dihasilkan atau dibuat lebih dulu. Dari tinjauan praktisnya, jika modul-modul berisiko paling tinggi dian-

tarkan (selesai dibuat dan digunakan) lambat dalam jadwal, mereka akan mempunyai sedikit peluang untuk mendapatkan pengujian yang banyak.

Modul yang Menyebabkan Jumlah Kesalahan Terbesar

Kerapatan kesalahan umumnya berada dalam sebagian besar program. Dua puluh persen modul bisa menyebabkan 80 persen kesalahan atau lebih. Probabilitas keberadaan kesalahan yang lebih besar dalam suatu modul proporsional dengan jumlah kesalahan yang telah dideteksi dalam modul itu. Oleh karena itulah beberapa manajer tidak akan membuang lebih banyak waktunya untuk membuat modul-modul yang error-prone (cenderung salah) menjadi modul yang error-free (bebas kesalahan). Namun mereka akan mengesampingkan mereka dan menulis ulang kodennya. Mereka juga akan memberikan tanggung jawab pengkodean kepada pengkode yang lain. Karena kesalahan awal menjadi pertanda kesalahan selanjutnya, maka strategi yang tepat yang perlu kita lakukan adalah membuang atau mengesampingkan pekerjaan yang sedang kita kerjakan yang terlihat ada kecenderungan mengalami kesalahan.

Menguji Modul Perangkat lunak Terpadu

Pengujian terpadu dilakukan dengan mengkombinasikan modul-modul secara bertahap. Sementara pengujian modul mengkonsentrasi pada modul-modul spesifik, pengujian terpadu dilakukan pada suatu hierarki modul, khususnya pada interface antara modul-modul itu.

Jika modul-modul tersebut bekerja sendiri, mengapa mereka tidak akan bekerja apabila dikombinasikan? Karena pemanfaatan modul-modul bisa menyebabkan kesalahan interface berikut ini:

- ◆ Data bisa hilang di sepanjang interface itu.
- ◆ Fungsi mungkin tidak berjalan sesuai harapan apabila dikombinasi dengan fungsi lain.
- ◆ Satu modul bisa mempunyai pengaruh balik terhadap modul lain.

Pengujian integrasi adalah suatu cara sistematik untuk membangun program selagi menjalankan pengujian untuk mendeteksi kesalahan interfacing.

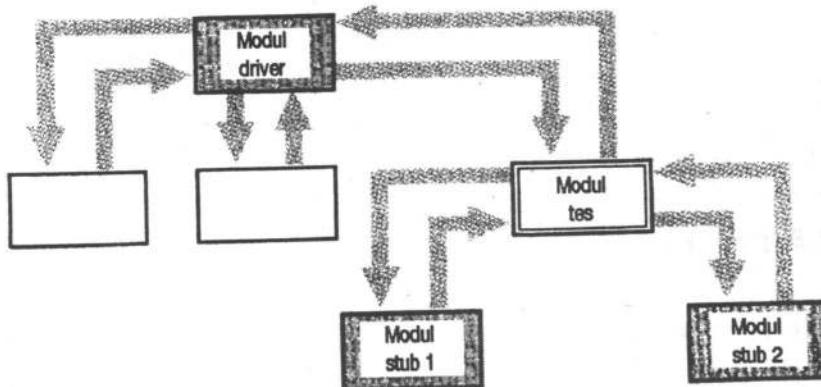
Pengintegrasian atau pemanfaatan modul menganut pendekatan inkremental (meningkat). Di dalam pendekatan ini, pengujian dimulai dengan modul tunggal yang akan mendapatkan test case yang tepat. Apabila pengujian terhadap modul ini telah

menghasilkan hasil yang memuaskan, maka mosul kedua dikedepankan, dan diterapkan lagi test case selanjutnya. Proses ini berlanjut sampai semua modul akhirnya dipadukan ke dalam program yang utuh. Karena pendekatan ini bersifat inkremental, maka dapat diasumsikan bahwa jika kesalahan terjadi ketika modul baru dikedepankan, kesalahan ini disebabkan oleh modul baru tersebut, atau disebabkan oleh beberapa aspek atau cara kita dalam mengedepankannya. Oleh karena itu, sumber kesalahan tersebut harus dilokalisasi, dan dengan demikian pendekripsi dan pengoreksian kesalahan akan menjadi lebih mudah.

Menggunakan Stub dan Driver

Karena suatu modul bukanlah program stand-alone, maka harus dikembangkan atau dibuat modul-modul stub dan driver untuk setiap pengujian modul. Seperti terlihat pada Gambar 4.7, modul pengujian yang memanggil dan mentransmisikan data memerlukan stub module (modul stub) untuk memodel hubungan ini. Stub tersebut mengambil tempat modul yang dipanggil yang belum dikode. Modul driver memanggil modul pada waktu pengujian dan menyampaikan data test kepadanya. Stub dan driver menghubungkan modul-modul agar mereka ini berjalan dalam suatu lingkungan yang mendekati lingkungan riil yang akan datang.

Stub dan driver sering kali terlihat seperti kode throw-away. Walaupun mereka tidak berada dalam versi akhir program tersebut selama pengembangan, mereka dapat digunakan kembali untuk menguji ulang program ini kapan saja ia berubah,



Gambar 4.7 Peranan modul stub dan modul driver dalam pengujian integrasi.

seperti halnya dalam pengujian regresi. Perpustakaan driver dan stub yang komprehensif akan menjadi perangkat pengujian yang sangat berdaya guna.

Cara-cara untuk Menguji Modul-modul Terpadu

Dua cara dasar untuk menjalankan pengujian terpadu adalah:

- ◆ Top-down
- ◆ Bottom-up

Integrasi Top-Down

PENGUJIAN INTEGRASI TOP-DOWN mengandalkan pengantaran inkremental modul-modul berlevel tinggi dan stub sebagai basis untuk pengujian. Modul-modul dipadukan dengan memindahkan ke bawah di sepanjang hirarki kendali, yang dimulai dari modul eksekutif utama, seperti yang ditunjukkan dalam bagan struktur biasa. Setelah modul ditulis pada setiap setiap level, stub-stub diganti dengan modul-modul riil yang bersesuaian dengan mereka sampai modul riil paling bawah ditulis, dan program keseluruhannya dapat diuji sebagai program yang terpadu secara keseluruhan.

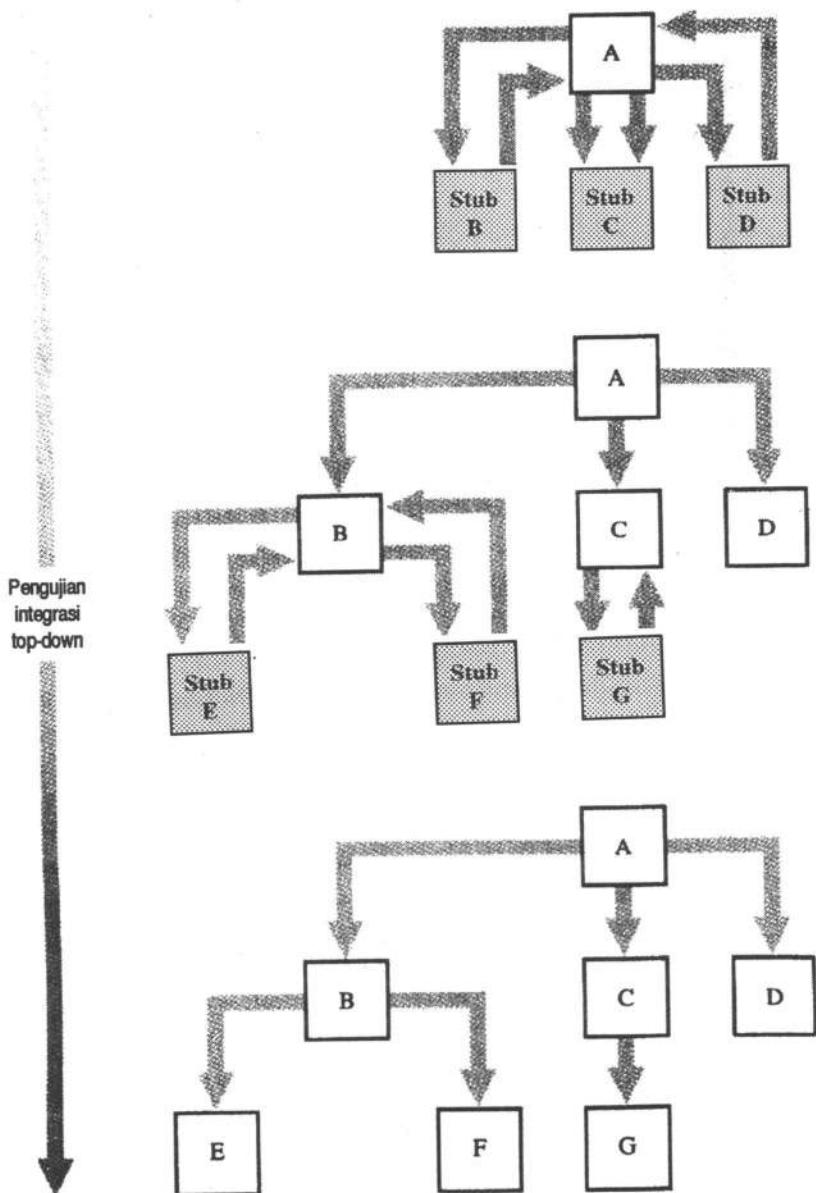
Gambar 4.8 menunjukkan suatu contoh pengujian integrasi top-down, dimana modul kendali, yaitu Modul A, diuji pertama kali. Dalam pengujian ini, stub-stub harus bersesuaian dengan Modul B, C, dan D. Ketika Modul B, C, dan D ditulis, mereka diuji dengan Modul A dan stub-stub untuk Modul E, F, dan G. Terakhir, ketika Modul E, F, dan G ditulis, keseluruhan program diuji sebagai paket terpadu.

Dalam pengujian yang tepat terhadap modul yang diminta (yakni, yang dipanggil), moul stub harus mengecek parameter input dan mengembalikan nilai yang wajar urituk parameter output. Stub tersebut harus mensimulasi tingkatan bawah dari program tersebut secara dekat agar pengujian top-down efektif.

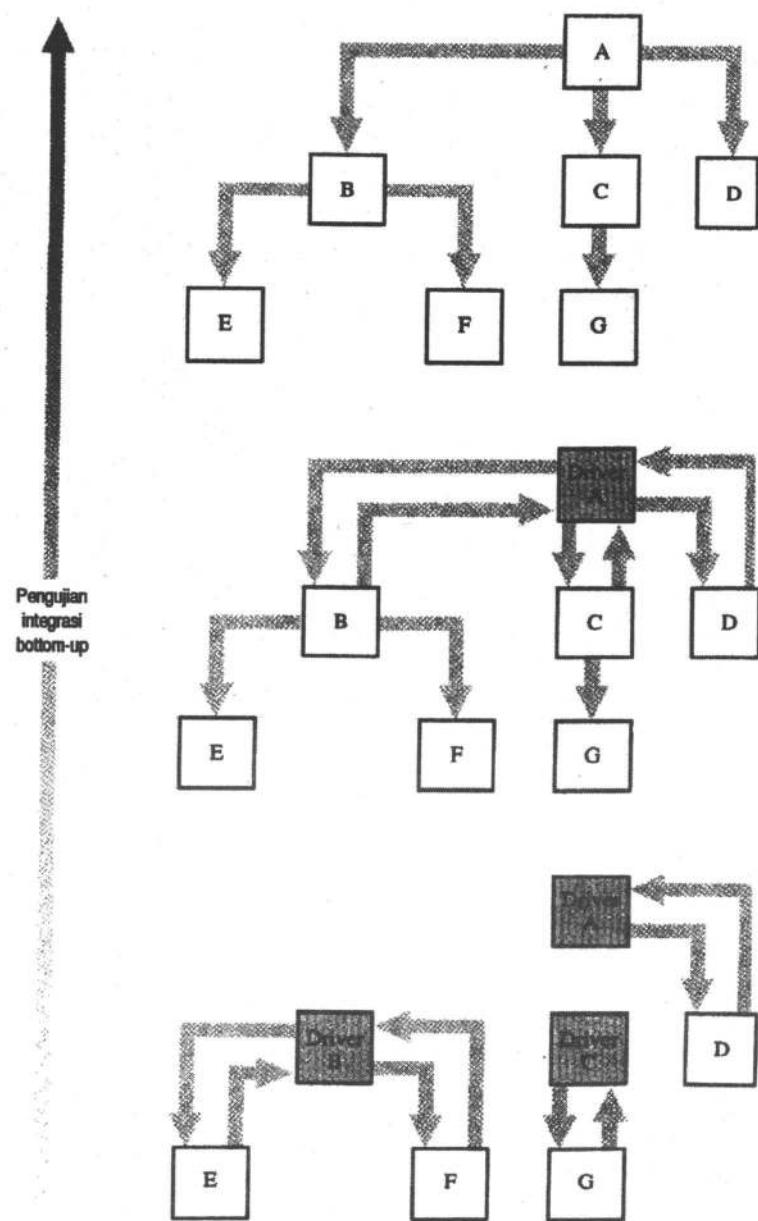
Integrasi Bottom-Up

Gambar 4.9 menunjukkan pengujian integrasi bottom-up, dimana modul-modul tingkatan paling bawah dikode dan diuji lebih dulu dengan driver yang tepat. Bentuk pengujian ini mengandalkan pengantaran modul-modul tingkat bawah dan driver sebagai basis untuk pengujian integrasi.

Dalam contoh kita sebelumnya, Modul E dan F memerlukan Driver B; Modul G memerlukan Driver C; dan Modul D memerlukan Driver A. Pada tingkat berikutnya,



Gambar 4.8 Pengujian integrasi top-down.



Gambar 4.9 Pengujian integrasi bottom-up

Modul B dan C sepenuhnya dikode, dipadukan, dan diuji dengan Driver A. Kemudian, Driver A dikode secara penuh dan modul-modul tersebut dipadukan secara penuh dan diuji secara keseluruhan.

Kelebihan dan Kelemahan Pendekatan Top-Down dan Bottom-Up

Umumnya, kelebihan pendekatan top-down adalah yang menjadi kelemahan pendekatan bottom-up, dan sebaliknya. Kedua pendekatan tersebut mengkode sedikit, kemudian menguji sedikit.

Salah satu faktor utama untuk pertimbangan ketika kita memilih pengujian integrasi top-down dan bottom-up adalah biaya penyiapan atau pembuatan stub dan driver. Pada umumnya, modul driver lebih mudah dikembangkan (dibuat), sehingga membuat pengujian bottom-up lebih murah.

Dengan pendekatan top-down, modul-modul utama atau pokok dikode dan diuji lebih dulu. Akibatnya, kesalahan pokok atau kelemahan rancangan bisa ditemukan lebih awal dalam proses, sehingga hal ini akan menghemat usaha perancangan ulang dan penulisan ulang kode. Lebih dari itu, pengembangan program yang bisa bekerja, meskipun terbatas, lebih awal akan memberikan dorongan psikologis kepada semua orang yang terlibat dalam pengembangan sistem itu. Kelemahan pokok pendekatan top-down adalah diperlukannya stub. Jika aplikasi sistemnya kompleks, maka akan sulit, jika tidak boleh dikatakan mustahil, untuk menulis modul-modul stub yang bisa mensimulasi aplikasi tersebut secara memadai dan menggenerasi output yang realistik.

Pendekatan bottom-up melibatkan pengujian modul pada tingkat-tingkat bawah dalam hirarki terstruktur terlebih dahulu, dan kemudian pelaksanaan pengujian ini merambat ke atas dalam hirarki tersebut sampai modul kendali akhir diuji. Driver memberi modul-modul tingkat lebih bawah dengan input yang tepat, dan umumnya, test case-nya akan lebih mudah dikembangkan (dibuat) dari pada untuk pengujian top-down.

Kelemahan pokok pendekatan bottom-up adalah bahwa tak ada program yang bisa bekerja yang dapat ditunjukkan sampai modul terakhir diuji. Oleh karena itu, kesalahan rancangan sistem yang mungkin ada tidak akan terdeteksi sampai modul terakhir tersebut diuji. Untuk bisa mendeteksi kesalahan rancangan, keseluruhan program harus ditulis ulang dan diuji ulang. Situasi seperti ini mengakibatkan diperlukannya rancangan yang tangguh dan pemeriksaan rancangan yang akurat.

Ringkasnya, jika modul-modul kendali dan modul-modul tingkat atas kompleks dan sangat penting bagi keberhasilan perangkat lunak, maka disarankan untuk

menggunakan pendekatan top-down. Sebaliknya, jika modul-modul tingkat bawah lebih penting, maka lebih baik digunakan pendekatan bottom-up. Umumnya, pendekatan meet-in-the-middle (dengan mencari titik tengah diantara keduanya) yang mengkombinasikan top-down untuk tingkat-tingkat atas struktur perangkat lunak, yang dipasangkan dengan bottom-up untuk tingkat-tingkat subordinat (bawah), akan merupakan pendekatan yang terbaik.

Apa yang Dimaksud Pengujian Sistem?

Pengujian sistem adalah proses pengujian perangkat lunak terpadu dalam konteks sistem keseluruhan yang ia dukung. Tes atau pengujian yang dilakukan pada tingkat ini meliputi berikut ini:

- ◆ **Pengujian Pemulihan.** Pengujian sistem memaksa perangkat lunak gagal dalam berbagai cara dan memverifikasi apakah pemulihan (recovery) penuh telah dilakukan dengan baik.
- ◆ **Pengujian Keamanan.** Test case dilakukan untuk memverifikasi apakah kendali yang tepat telah dirancang dan diinstal dalam sistem tersebut guna melindunginya dari sejumlah risiko. Penguji sistem (kadang-kadang disebut Tiger Team) melakukan segala sesuatu untuk menembus kelemahan sistem itu dan melakukan tindakan yang merusaknya.
- ◆ **Pengujian Stress** (penekanan). Jenis pengujian ini, yang sama dengan konsep pengujian keamanan, mengeksekusi sistem dengan cara meminta sumber-sumber dalam kuantitas, frekuensi, atau volume yang tak normal. Misalnya, penguji boleh menggenerasi 20 interrupt per detik atau menggenerasi ribuan instruksi untuk mencoba merusak atau memberhentikan program itu. Pengujian stress memvalidasi kemampuan program untuk menangani volume transaksi yang besar dengan cara yang tepat waktu.

Apa yang Dimaksud Pengujian Penerimaan?

Pengujian penerimaan mengevaluasi sistem baru guna menentukan apakah ia memenuhi keperluan pemakai dalam kondisi operasional. Apabila perangkat lunak dan dokumentasi terlihat stabil oleh kelompok penguji, maka telah waktunyalah untuk me-release perangkat lunak dan dokumentasi tersebut kepada kelompok penguji pemakai untuk mendapatkan umpan balik dari pemakai ini. Kelompok penguji pemakai harus terkomposisi atas semua atau sebagian (wakil) orang-orang yang akan

bekerja dengan sistem itu selama pengembangan. Semua atau sebagian besar pekerjaan mereka akan tergantung pada bekerjanya sistem ini secara benar. Ada dua prosedur pengujian penerimaan yang dapat digunakan:

- ◆ Pengujian alfa
- ◆ Pengujian beta

Pengujian penerimaan adalah peluang terakhir untuk menguji dan mempertimbangkan kembali segala sesuatunya sebelum perangkat lunak tersebut dikonversi dari pengembangan ke operasi.

Pengujian Alfa

PENGUJIAN ALFA dilakukan dengan setting alami (yakni, dalam lingkungan pengoperasian dunia-nyata) dengan bantuan profesional sistem, yang biasanya sebagai pengamat yang mencatat kesalahan dan masalah penggunaan. Dua teknik yang khususnya berlaku (bisa diterapkan) untuk pengujian alfa adalah sebagai berikut:

Usability Labs (laboratorium pendayagunaan). Rancangan usability labs berlainan, namun tema pusatnya konstan: Dapatkan sampel wakil-wakil orang yang akan akhirnya akan menggunakan perangkat lunak tersebut untuk melakukan pengujian, dan tempatkan mereka dalam lingkungan pengujian yang terstruktur dan terkendali. Ukuran sampel bisa dari hanya satu orang pemakai untuk aplikasi yang kecil dan berbasis-lokal, sampai 10 atau lebih untuk aplikasi yang besar dan berbasis-global.

Usability lab (laboratorium pendayagunaan) mungkin berupa suatu ruang dengan satu PC, atau ruangan luas tersekat-sekat dan diperlengkapi dengan kaca riben, kamera televisi, dan beberapa PC atau workstation. Para pemakai berada di bagian pengujian pada lab tersebut dan pengamat (misalnya, profesional sistem) ditempatkan di balik kaca riben (satu-arah) untuk mengamati para pemakai yang bekerja dengan sistem tersebut. Satu kamera televisi difokuskan pada layar komputer dan keyboard; kamera kedua difokuskan atau diarahkan ke pemakai; dan kamera ketiga, yang diarahkan ke atas kepala, mencatat interaksi pemakai dengan dokumentasi pemakai. Tujuan usability lab ini tidak terlalu untuk menentukan apakah perangkat lunak tersebut bekerja dari sudut pandang teknis, namun untuk menentukan sejauh mana pemakai berpikir bahwa ia bekerja. Sebagai contoh, pengamatan mungkin menunjukkan seorang pemakai yang bingung mencoba mengisi form atau mengikuti prosedur yang tak jelas. Setelah mengamati videotape dari sesion pengujian itu,

pengamat harus mencari jawaban secara presisi mengapa pemakai bingung. Misalnya, alasan atau sebab kebingungan tadi mungkin berupa rancangan form-nya atau prosedur yang tidak jelas.

Usability Factors Checklist (Daftar Periksa Faktor Pendayagunaan). Lihat-lah lagi faktor rancangan MURRE yang telah dijelaskan pada bab sebelumnya. Sampai saat ini, kemampuan pemeliharaan (maintainability) telah dievaluasi oleh pemeriksaan kode dan rancangan. Pendayagunaan (usability) telah diuji selama pengembangan, namun tidak diperlukan selama kondisi kerja. Sebagai contoh, rancangan layar telah dievaluasi, dan transaksi uji spesifik telah dijalankan. Tingkat daya penggunaan kembali (reusability) bisa sangat bagus. Misalnya, sistem yang sedang dikembangkan tersebut mungkin telah memerlukan 100 KLOEC, namun 80 persen kodennya dapat digunakan untuk sistem baru. Maka hanya 20 KLOEC yang perlu ditulis dari scratch. Tingkat reliabilitas mungkin telah dinilai tinggi, karena perangkat lunak tersebut telah lulus dari pengujian modul dan integrasi dan pengujian pemulihan, stress, dan keamanan tanpa cela. Tingkat kemampuan perluasan (extendability) bisa saja tinggi sebab rancangan modularnya sangat baik atau penggunaan metode pemrograman berorientasi-obyeknya juga berhasil. Dengan demikian, sistem baru akan bisa beradaptasi dengan dan tumbuh dengan perubahan keperluan pemakai secara mudah.

Meskipun semua faktor rancangan MURRE dinilai bagus, naun evaluasi faktor pendayagunaan belum selesai. Penilaian yang lengkap dan final terhadap faktor rancangan ini harus dilakukan oleh kelompok uji pemakai dalam kondisi yang mewakili cara pemakai bekerja dengan sistem tersebut setelah sistem ini di-release ke operasi.

Karena pemakai adalah penilai (wasit) terakhir dari kualitas perangkat lunak, maka kelompok uji pemakai harus menerapkan suatu teknik yang bisa membantu mereka mengevaluasi dan merampungkan penilaian terhadap faktor rancangan pendayagunaan, yakni Usability Factors Checklist (Daftar Periksa Faktor Pendayagunaan) (lihat Gambar 4.10). Daftar ini dapat diperluas dan disesuaikan. Ia juga dapat digunakan bersamaan dengan laboratorium pendayagunaan. Daftar faktor pendayagunaan memberi pemakai peluang untuk mengevaluasi kualitas perangkat lunak baru sebelum ia dikonversi dari pengembangan ke operasi.

Selain menyelesaikan daftar faktor pendayagunaan, kelompok uji pemakai juga harus menjelaskan kepada profesional sistem tentang mengapa faktor-faktor tersebut dinilai rendah; apabila penilaianya demikian. Misalnya, mengapa seorang pemakai mendapati kemudahan penggunaan program itu dinilai rendah? Apakah ini karena

USABILITY FACTORS					
	1 = low to 5 = high				
A. Ease of use	1	2	3	4	5
B. User friendliness	1	2	3	4	5
C. Understandability	1	2	3	4	5
D. Level of confidence	1	2	3	4	5
E. Conformance to requirements	1	2	3	4	5
F. Conformance to response time	1	2	3	4	5
G. Comfort level	1	2	3	4	5

Please circle the number applicable to each quality factor.

Gambar 4.10 Worksheet faktor pendayagunaan perangkat lunak.

dokumentasi yang tidak baik dan kurangnya instruksi? Pemakai tersebut harus menjawab pertanyaan seperti ini dan, jika mungkin, memberikan saran tentang bagaimana atau cara meningkatkan sebagian atau semua faktor pendayagunaan tersebut.

Pengujian Beta

Pengujian Beta sama dengan pengujian alfa; perbedaannya adalah bahwa dalam pengujian beta ini tidak ada profesional sistem yang hadir (misalnya, di dalam laboratorium pendayagunaan) selama pengujian penerimaan pemakai. Selain menerapkan daftar faktor pendayagunaan, kelompok uji pemakai mencatat semua masalah, yang riil atau yang dibayangkan, yang dijumpai selama pengujian beta dan melaporkan masalah ini kepada orang-orang sistem secara berkala. Modifikasi dilakukan guna membuat sistem tersebut siap di-release untuk implementasi dan operasi penuh.

Merelease Perangkat lunak

Seberapa banyaknya pengujian yang dianggap cukup? Sebenarnya, kita tak pernah bisa menyelesaikan pengujian. Tahap pengujian itu sendiri dalam SWDLC hanya akan berhenti apabila perangkat lunak lulus uji penerimaan.

Setelah semua pengujian sebelumnya dilakukan, dan perangkat lunak tersebut telah berhasil lulus dari ujian-ujian itu, maka layaklah nampaknya perangkat lunak

tersebut bisa diterima dan di-release dari status pengembangan (menuju) ke operasi. Pe-release-an perangkat lunak merepresentasikan titik akhir dari SWDLC. Perangkat lunak tersebut diterima untuk diimplementasikan bersama dengan komponen-komponen lain dari sistem keseluruhan. Penyelesaian tahap implementasi merepresentasikan titik akhir dari SDLC.

Siapa pun proyek manajernya, ia selalu ingin mengesahkan perangkat lunak tersebut sebagai perangkat lunak yang bebas kesalahan dan bebas masalah sepenuhnya. Namun demikian, pengesahan semacam ini selalu tidak mungkin. Manajer proyek mengetahui apa yang telah diuji, namun tidak tahu sedikit pun apa yang belum diuji.

Jadi, dalam pelepasan (releasing) perangkat lunak baru ke implementasi dan operasi akhir kita harus membuat pertimbangan atau keputusan. Dengan menggunakan prosedur pengujian sebelumnya, kita bisa diperkuat olehnya dan mendapatkan basis yang kuat pula untuk membuat pertimbangan atau keputusan ini. Sebagai contoh, manajer proyek bisa membuat statemen berikut ini: "Dengan pertimbangan yang diambil prosedur pengujian yang dilakukan dan didokumentasi disini, saya yakin Perangkat lunak A 95 sampai 99 bebas kesalahan, dan faktor-faktor rancangan kemampuan pemeliharaan, pendayagunaan, pendayagunaan kembali, reliabilitas, dan kemampuan diperluasnya dinilai bagus. Oleh karenanya saya rekomendasikan agar untuk merelease implementasi dan operasi sistem."

TINJAUAN SASARAN BELAJAR UNTUK BAB INI

Tujuan-tujuan pokok (pokok-pokok pembelajaran) bab ini memungkinkan setiap siswa mencapai enam tujuan pembelajaran yang penting. Kita sekarang akan merekapitulasi jawaban-jawaban atas tujuan-tujuan pembelajaran ini.

Sasaran belajar 1:

Mendeskripsikan skenario pengujian perangkat lunak dan menunjukkan tujuannya.

Skenario pengujian perangkat lunak melibatkan:

- ◆ Pengembangan test case dan strategi pengujian.
- ◆ Pelaksanaan test-test itu.

- ◆ Pelaporan kesalahan yang terdeteksi.
- ◆ Pengoreksian kesalahan.
- ◆ Peramalan tingkat reliabilitas perangkat lunak.

Pengujian merupakan tahap vital dalam pengembangan perangkat lunak. Menguji perangkat lunak dan mengeluarkan kesalahannya sebelum ia menuju (dikonversi) ke operasi akan membantu mengurangi jumlah kesalahan yang mungkin akan berkembang selama operasi — apabila kesalahan terjadi ketika perangkat lunak sudah operasional, maka ini adalah saat terburuk terjadinya kesalahan.

Sasaran belajar 2:

Mendefinisikan dua cara dasar untuk merancang test case.

Dua cara dasar merancang test case adalah:

- ◆ Pengujian white box
- ◆ Pengujian black box

Pengujian white box memfokuskan pada antarkerja kodennya. Sebaliknya, pengujian black box menguji fungsionalitas perangkat lunak itu. Semua test case harus dicatat dalam suatu matriks test case, dimana dalam matriks ini juga dituliskan hasil yang diharapkan dan hasil sebenarnya.

Sasaran belajar 3:

Menjelaskan apa yang harus dilakukan apabila kesalahan terdeteksi.

Jika kesalahan-kesalahan terdeteksi, mereka harus dicatat dalam Laporan Kesalahan dan disampaikan (dikomunikasikan) kepada orang yang ditugasi mengoreksi dan memperbaiki kesalahan-kesalahan itu. Jenis laporan ini adalah:

- ◆ Saran
- ◆ Kesalahan rancangan
- ◆ Kesalahan pengkodean

- ◆ Kesalahan dokumentasi
- ◆ Query

Kesalahan bisa berupa kesalahan minor, serius, atau fatal. Resolusi akhirnya adalah satu dari berikut ini:

- ◆ Kesalahan tersebut diperbaiki
- ◆ Kesalahan tersebut tidak dapat direproduksi
- ◆ Kesalahan tersebut tidak dapat diperbaiki
- ◆ Pengkode tidak sependapat dengan saran penguji
- ◆ Kesalahan atau saran tersebut ditarik
- ◆ Perangkat lunak tersebut bekerja sesuai dengan spesifikasi rancangan

Pengkode yang ditugasi memperbaiki kesalahan, penguji, dan manajer proyek menandatangani Laporan Kesalahan setelah dilakukan resolusi tersebut.

Sasaran belajar 4:

Mengemukakan suatu cara untuk membantu mengukur reliabilitas test case.

Penyemaian kesalahan ke dalam program yang akan diuji bisa membantu mengukur reliabilitas test case. Jika kesalahan yang disemaikan merepresentasikan jenis kesalahan riil, dan test case mengungkap semua kesalahan yang disemaikan, kita dapat mengambil kesimpulan bahwa semua kesalahan riil juga telah terdeteksi.

Sasaran belajar 5:

Menyetujukan dan menjelaskan secara singkat empat tahap pengujian perangkat lunak.

Empat tahap pengujian perangkat lunak adalah:

- ◆ Modul
- ◆ Integrasi
- ◆ Sistem
- ◆ Penerimaan

Pengujian modul adalah proses pengujian komponen-komponen terkecil dalam program keseluruhan sebelum mereka digabungkan bersama untuk membentuk perangkat lunak secara keseluruhan. Pengujian integrasi adalah proses pengujian modul-modul tergabung untuk melihat apakah perangkat lunak tersebut bekerja bersama sebagai keseluruhan. Pengujian sistem adalah proses pengujian perangkat lunak terpadu dalam konteks sistem total (keseluruhan) yang ia dukung. Pengujian penerimaan adalah proses yang memberikan keleluasaan kepada pemakai maupun operator untuk menguji program guna melihat apakah ia memenuhi keperluan mereka sebelum program itu di-release dan dikonversi ke operasi.



Pengujian integrasi top-down dimulai dengan modul yang terkode lebih unggul dan stub untuk menguji tingkat hirarki terstruktur paling puncak lebih dulu. Kemudian modul-modul tingkatan berikutnya dikode dan diuji dengan cara yang sama sampai modul tingkat paling bawah dikode dan diuji dalam program keseluruhan yang terpadu. Untuk pengujian integrasi bottom-up, prosesnya berkebalikan. Modul-modul tingkat paling bawah dikode lebih dulu dan driver dipersiapkan untuk menguji mereka. Kemudian tingkat modul-modul berikutnya (lebih atasnya) dikode dan diuji dengan cara yang sama sampai modul paling atas yang terakhir dikode dan diuji dalam suatu program terpadu.

DAFTAR PERIKSA PENGUJIAN PERANGKAT LUNAK

Berikut ini adalah urut-urutan cara melakukan pengujian perangkat lunak, yang merupakan tahap terakhir dari SWDLC.

1. Tetapkan rencana pengujian perangkat lunak dengan menggunakan skenario pengujian perangkat lunak terstruktur sebagai pedoman dasar anda.
2. Buatlah atau kembangkan test case white box dan black box, dan gunakan matriks test case untuk mencatat test case. Apabila anda rasanya perlu, semaikan kesalahan untuk menguji efektivitas test case tersebut.

3. Siapkan atau buatlah Laporan Kesalahan untuk mendokumentasi semua kesalahan yang ditemukan dan resolusinya.
4. Ciptakan strategi pengujian perangkat lunak yang memecah (membagi) pengujian tersebut ke dalam tahapan, seperti pengujian modul, integrasi, sistem, dan penerimaan.
5. Tentukan tingkat reliabilitasnya. Jika test tersebut menunjukkan reliabilitas yang tinggi, release-lah perangkat lunak tersebut untuk operasi.

PERTANYAAN TINJAUAN

- 4.1 Apakah tahap pengujian perangkat lunak memastikan bahwa suatu program sepenuhnya bebas kesalahan? Jelaskan jawaban anda.
- 4.2 Berilah garis besar skenario pengujian perangkat lunak.
- 4.3 Jelaskan tujuan pengujian white box.
- 4.4 Jelaskan tujuan pengujian black box.
- 4.5 Terangkan secara singkat alasan (tujuan) untuk menguji perintah-perintah berikut:
SELECT, OPEN/CLOSE, COPY REPLACING, IF, PERFORM UNTIL atau PERFORM WHILE, dan CALL.
- 4.6 Definisikan kelas ekuivalensi. Berikan dua contoh pengujian kelas ekuivalensi.
- 4.7 Elemen-elemen apa yang harus disertakan dalam matriks test case?
- 4.8 Elemen-elemen apa yang harus ada dalam Laporan Kesalahan?
- 4.9 Sebutkan jenis laporan kesalahan dan derajad kepelikannya (kekerasananya).
- 4.10 Sebutkan enam cara untuk memperbaiki atau memecahkan jenis laporan kesalahan.
- 4.11 Jika seorang penguji tidak yakin dengan apa yang memicu kesalahan, apa yang seharusnya ia lakukan?
- 4.12 Definisikan pengujian regresi dan jelaskan tujuannya.
- 4.13 Apa tujuan model penyemaian? Jelaskan bagaimana ia bekerja dan hubungannya dalam membantu mengestimasi reliabilitas perangkat lunak.
- 4.14 Definisikan pengujian risk-driven. Apa tujuannya?
- 4.15 Definisikan kerapatan kesalahan.

- 4.16 Bedakan antara pengujian modul dan pengujian integrasi.
- 4.17 Definisikan modul stub dan terangkan bagaimana ia bekerja. Definisikan modul driver dan terangkan bagaimana ia bekerja.
- 4.18 Sebutkan tiga komponen pengujian sistem. Terangkan secara singkat tujuan ketiganya.
- 4.19 Apa tujuan pengujian penerimaan? Sebutkan dua orang yang berperan pokok dalam pengujian penerimaan.
- 4.20 Definisikan pengujian alfa dan pengujian beta.

SOAL SPESIFIK BAB INI

Soal-soal ini memerlukan jawaban pasti yang didasarkan secara langsung pada konsep dan teknologi yang dikemukakan dalam bab atau teks ini.

- 4.21 Seorang pengkode menuliskan instruksi berikut ini:

```

IF
  A = B
THEN
  SET B = 10
  SET B = 20

```

Ditanyakan: Jelaskan apa yang dilakukan instruksi ini. Kodelah ulang instruksi tersebut untuk men-set B sama dengan 10 jika A sama dengan B, dan jika tidak setlah B sama dengan 20.

- 4.22 Seorang pengkode menuliskan instruksi berikut ini:

```
COMPUTE A = B/C
```

Ditanyakan: Buatlah empat test case untuk instruksi ini, dan nyatakan hasil yang diharapkan untuk setiap test case itu.

- 4.23 Lihatlah instruksi berikut ini:

```

IF
  A < B AND C = 3
THEN
  DO SOMETHING

```

Ditanyakan: Buatlah empat test case untuk instruksi ini, dan jabarkan hasil yang diharapkan dari setiap test case itu.

- 4.24 Suatu program dirancang bisa menerima segala bilangan antara 1 dan 99.

Ditanyakan: Rancanglah empat pengujian kelas ekuivalensi, satu diantaranya valid dan tiga yang lain invalid.

SOAL UMUM

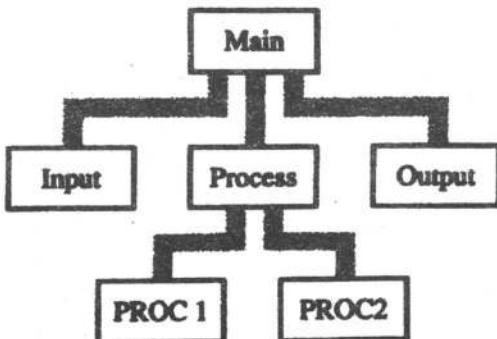
Soal-soal ini meminta jawaban berupa pendekatan atau cara yang layak, bukannya solusi yang presisi. Walaupun soal-soal ini didasarkan pada materi bab ini, kita diminta melihat bacaan ekstra dan diperlukan kreativitas dari kita untuk mengembangkan atau membuat solusi yang tepat dan baik.

- 4.25 Ada sejumlah kendali yang harus dirancang ke dalam program. Beberapa dari kendali ini adalah:

- ◆ Total kendali
- ◆ Laporan kesalahan
- ◆ Log transaksi
- ◆ Daftar urutan atau rangkaian
- ◆ Data yang hilang
- ◆ Pengecekan jangkauan
- ◆ Pengecekan kewajaran
- ◆ Digit pengecekan
- ◆ Label file
- ◆ Validitas (atau tabel identifikasi)

Ditanyakan: Buatlah test case untuk setiap kendali ini untuk memastikan bahwa program tersebut mengeksekusi kendali-kendali tersebut secara benar. Tentukan suatu jumlah atau buatlah suatu asumsi yang anda rasa perlu untuk mengembangkan pendekatan yang layak untuk mengatasi masalah yang kita buat tadi.

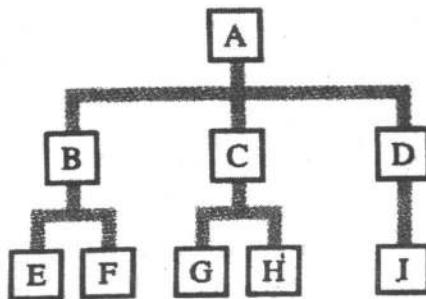
- 4.26 Pelajari bagan struktur berikut ini:



Ditanyakan: Dalam bentuk bahasa Inggris terstruktur, atau menggunakan naratif (keterangan) singkat anda sendiri, tetapkan bagaimana anda menjalankan pengujian integrasi top-down untuk bagan struktur di atas. Lakukan hal yang sama untuk pengujian integrasi bottom-up.

4.27

Asumsikan bahwa set modul berikut ini akan diuji:



Ditanyakan: Tunjukkan bagaimana anda akan menjalankan pengujian integrasi top-down untuk set modul yang lebih dulu. Ada jenis pengujian integrasi ketiga. Disebut apa ia? Tunjukkan bagaimana ia bekerja. Jelaskan masalah yang akan terjadi dengan pengujian bottom-up. Jelaskan bagaimana pengujian top-down menawarkan beberapa keunggulan atau kelebihan atas pengujian bottom-up. Nyatakan masalah tersebut dengan pengujian top-down.

4.28

Di Maltese Drapery Company, program-program perangkat lunak jarang diuji sebelum mereka dikonversi ke operasi. Para pemakai seringkali menjumpai kesalahan dalam program, namun seringkali mengbaikannya. Misalnya, suatu program mungkin mengalami crash. Para pemakai ini lebih senang mencoba metode yang agak berbeda agar pekerjaan mereka saat itu lebih cepat selesai dari pada harus mencari kesalahan dan melaporkannya. Anda adalah seorang konsultan sistem yang disewa untuk Maltese yang tugas anda menilai situasi saat itu dan membuat rekomendasi untuk meningkatkan reliabilitas perangkat lunak. Beberapa komentar yang anda dengar dari para pemakai adalah:

"Dari pengalaman saya, adalah sangat tidak berguna melaporkan kesalahan jika saya tidak dapat menunjukkan penyebab pasti kesalahan tersebut dan mereproduksi kesalahan itu."

"Saya belum melaporkan kesalahan ini, karena pemulihannya selalu lebih cepat dari pada waktu yang diperlukan untuk melaporkan dan meminta programmer mengoreksinya."

"Umumnya saya tidak melaporkan kesalahan, sebab orang-orang sistem kemungkinan tidak akan memperhatikannya, mencatatnya sebagai kejadian tak penting atau sebagai kesalahan pemakai, atau mereka berlama-lama untuk memperbaikinya."

Ditanyakan: Buatlah suatu laporan kepada manajemen puncak yang isinya penilaian anda tentang situasi saat itu kaitannya dengan reliabilitas perangkat lunak di Maltese dan rekomendasi anda untuk meningkatkan reliabilitas perangkat lunak.

BACAAN YANG DISARANKAN

- Beizer, B, *Software Sistem Testing and Quality Assurance*. New York: Van Nostrand Reinhold, 1984.
- Burch, John G, "Basic Program Development and Testing." *Journal of Accounting and EDP*, Summer 1989.
- Crosby, P. B. *Quality Is Free: The Art of Making quality Certain*. New York: McGraw-Hill, 1979.
- Desmond, John, "The Friendly Master." *Software Magazine*, February 1988.
- Durant, Jerry E. "Applying Synematic Testing to Application Development Audits." *Internal Auditor*, February 1991.
- Kaner. Cem, *Testing Computer Software*. Blue Ridge. Pa TAB Professional and Reference Books, 1988.
- Keyes, Jessica. "Gather a Baseline to Assess CASE Impact." *Software Magazine*, August 1990.
- Martin, James, and Carma McClure, *Structured Techniques for Computing*. Englewood Cliffs, N.J.: Prentice-Hall, 1981.
- Meyers, G. *The Art Software Testing*. New York: John Wiley, 1979.
- Miller, Barton P., Lars Fredriksen, and Brian So, "Study of the Reliability of UNIX Utilities." *Communications of the ACM*, Vol. 33. No. 12, December 1990.
- Musa, John D., Anthony Iannino, and Kazuhira Okumoto *Soltware Reliabiliiy*. New York: McGraw-Hill, 1990.
- Perry, William E, *A Standard Testing Applicalion Sofrware*, Boston: Auerbach Publishers, 1990.
- Pressman, Roger S. *Software Engineering: A Practitioner's Approach*, 2nd ed. New York: McGraw-Hill, 1987.
- Rettig, Marc. "Testing Made Palatable." *Communications of the ACM*, Vol. 34. No.5, May 1991.
- Sivula, Chris. "Back to the Lab." *Daramation*, August 15. 1990.
- Stahl, Bob. "The Ins and Outs of Software Testing." *Compuuterworld*, October 24. 1988.

KASUS JOCS: Pengujian Perangkat lunak

Jake Jacoby telah mengimplementasikan strategi perangkat lunak siklis yang berkelanjutan untuk proyek JOCS. Ia memutuskan pengujian harus dilakukan sepanjang atau selama SWDLC, bukannya melakukan pengujian hanya pada akhir siklus. Gambar 4.11 adalah representasi grafik dari strategi pengujian perangkat lunak yang dibuat Jake. Persis sebagaimana programmer menggunakan DFD untuk memahami suatu sistem, Jake mereasa bahwa menciptakan model grafik ini akan membuat strategi pengujian bisa dipahami oleh para anggota tim SWAT yang lain.

Empat lingkaran pada Gambar merepresentasikan empat tahapan yang berlainan dalam pengujian perangkat lunak. Sebagai contoh, lingkaran 1 merepresentasikan proses sirkuler yang terkomposisi dari dua tugas: rancangan penggunaan umum dan pemeriksaan terstruktur yang menyertainya terhadap rancangan itu. Kedua tugas ini adalah bagian dari lingkaran pertama, karena rancangan dan pemeriksaan dilakukan secara berulang-ulang, atau dengan cara sirkuler (berputar), sampai programmer yakin bahwa rancangan yang diharapkan telah di capai.

Tahan kedua dalam proses tersebut, Lingkaran 2, memasukkan tugas-tugas sirkuler rancangan modul, pemeriksaan modul, dan pengujian top-down. Jake berfikir bahwa perangkat lunak paling baik dikembangkan dengan kebijakan inkremental: Merancang sedikit, mengkode sedikit, menguji sedikit. Merancang sedikit lagi, mengkode sedikit lagi, menguji sedikit lagi. Ia lebih suka kebijakan ini dari pada kebijakan dengan cara merancang semua perangkat lunak, kemudian mengkode semua perangkat lunak, kemudian mengandalkan semunya dan menguji semua perangkat lunak. Lingkaran kedua pada Gambar 4.11 tersebut merepresentasikan kebijakan inkremental Jake. Lingkaran 1 dan 2 merepresentasikan tahapan pengujian white box untuk program yang ada. Tugas-tugas yang mengkompisisi lingkaran-lingkaran ini akan diselesaikan oleh programmer yang bertanggung jawab untuk melakukan rancangan, pengodean, dan pengujian awal untuk program itu. Selagi menjalankan tugas-tugas ini, program tersebut mempunyai kesempatan untuk menguji logika program tersebut secara keseluruhan dan rinci. Jake berfikir bahwa programmer harus mempunyai tanggung jawab atas suatu program, dan ini akan menjadi kebanggaannya. Penyelesaian tugas-tugas dari dua lingkaran yan pertama harus memuaskan kebutuhan manusia, sehingga ia merasakan bahwa pekerjaan benar-benar selesai.

Dua tahap berikutnya dilakukan oleh personel selain programmer aslinya. Lingkaran 3 terkomposisi dari tiga tugas, pengujian stress, pengujian pemulihan, dan pengujian keamanan. Jake mempunyai rencana agar tiga tugas ini dikerjakan oleh orang lain di dalam tim SWAT. Orang ini akan bertanggung jawab untuk melakukan pengujian validitas program dengan cara menjalankan program tersebut dengan serangkaian input dan melakukan pengecekan mengenai output apa yang diciptakan.

Ini adalah pengujian black box. Sebagaimana ditunjukkan oleh anak panah pada diagram itu, tugas-tugas dalam Lingkaran 2 selalu memasukkan tugas-tugas dalam Lingkaran 3. Namun demikian, garis terputus-putus menunjukkan bahwa kadang-kadang tugas-tugas yang dikerjakan dalam Lingkaran 3 akan menghendaki program agar dikembalikan ke Lingkaran 2. Misanya, jika pengujian menemukan masalah dengan suatu program selama pengujian Lingkaran 3, maka pengujian ini mungkin akan memperbaiki masalah ini atau mungkin akan mengembalikan program itu ke programmer, dimana programmer ini akan kembali lagi ke tugas-tugas yang ditunjukkan dalam Lingkaran 2.

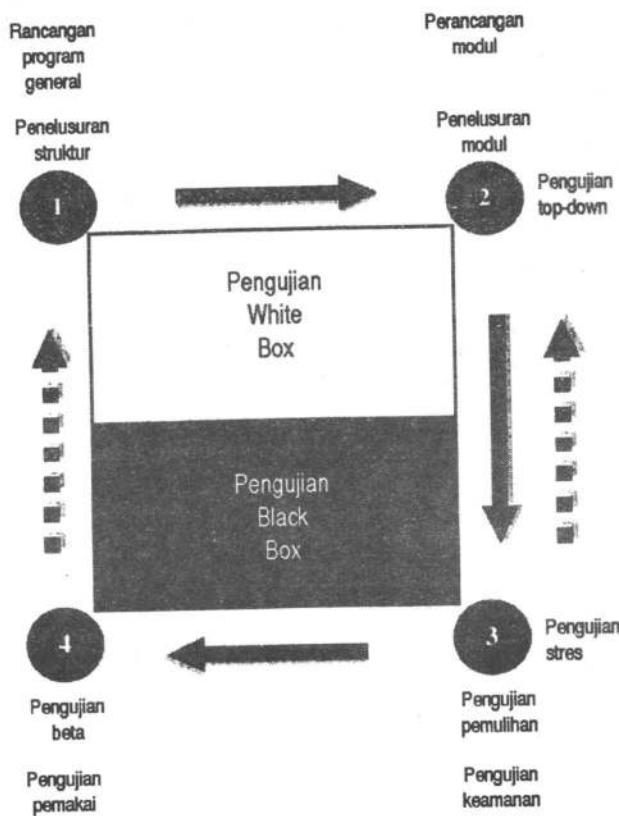
Lingkaran 4 mempresentasikan pengujian yang dilakukan oleh pemakai: pengujian pemakai dan beta. Pengujian pemakai akan dilakukan oleh para pemakai, dimana anggota tim SWAT berdiri di dekatnya sambil mengamati segala masalah pemrograman. Pengujian ini akan dilakukan untuk mensimulasi lingkungan yang hidup, namun tanpa memasukkan data riil. Pengujian beta, sebaliknya, akan dilakukan oleh para pemakai akan mengeksekusi program tersebut dengan data akurat seolah-olah perangkat lunak tersebut berfungsi penuh. Pada Gambar 4.11, ada garis terputus-putus antara Lingkaran 1 dan 4. Jika para pemakai menemukan masalah selama pengujian beta, masalah ini mungkin akan menyebabkan programmer merancang kembali program itu. Para pemakai mungkin pula menemukan keperluan (persyaratan) sistem baru selama pengujian bta. Keperluan baru ini mungkin bisa diatasi saat itu, atau mungkin para pemakai menunda mengatasinya sampai mengimplementasian sistem selesai.

Sebagai contoh strategi pengujian perangkat lunak Jake ini, marilah kita lihat pengembangan program penangkapan data pekerja jam-jaman langsung yang diciptakan oleh Jannis Court.

Lingkaran 1: Jannis menerjemahkan analisis sistem DFD ke dalam diagram Jackson dalam Bab 2. Dengan menggunakan strategi pengujian yang dibuat Jake, Jannis melakukan pemeriksaan terstruktur pada rancangan itu dengan anggota tim yang lain.

Beberapa masalah ditemukan selama pemeriksaan, dan Jannis perlu mengerjakan kembali rancangan program umum. Pemeriksaan rancangan lain menyetujui perubahan yang ia buat.

Lingkaran 2: Jannis memanfaatkan diagram Jackson yang diciptakan pada Bab 2 dan mengkonversinya ke dalam bahasa Inggris terstruktur. Contoh Bahasa Inggris terstruktur untuk programnya ditunjukkan pada Gambar 4.12. Jannis emeriksa Bahasa Inggris terstruktur dengan anggota tim SWAT yang lain dan mendapatkan persetujuan tim atas rancangannya.



Gambar 4.11 Strategi pengujian perangkat lunak JOCS.

Jannis sekarang mengetikkan modul utama program tersebut dengan stub-stub yang merepresentasikan modul-modul rinci dan ia melakukan pengujian top-down untuk mengidentifikasi kesalahan sintaks dan logika dalam modul-modul utama tersebut.

Setelah ia meyakinkan dirinya melalui pengujian online dengan data sampel bahwa logika keseluruhan telah benar, modul-modul rinci selanjutnya dirancang dan dikode dengan teknik pemeriksaan terstruktur yang dideskripsikan di atas. Namun demikian, jika logika keseluruhan teridentifikasi tidak benar selama pemeriksaan terstruktur atau selama pengujian top-down, Jannis akan kembali ke tahap rancangan dari siklus dalam lingkaran ini dan merancang ulang program tersebut. Modul-modul rinci diketikkan (dimasukkan) dan diuji sekali setiap saat untuk mengetahui adanya kesalahan sintaks dan logika.

```

Begin Capture Direct Labor Hours

Clear employee id, job number, operation number
Set data input complete flag = "no"
Do process data module until data input complete flag = "yes"
    Get job number
    If job status = "complete"
        Do completed job module
    Else
        Get operation number
        If valid operation for job status = "no"
            Do invalid operation module
        Else
            If operation status = "complete"
                Do completed operation module
            Else
                Get employee id
                If hours > weekly total hours
                    Do overtime hours module
                Else
                    Set process transaction flag = "yes"
                Endif
            Endif
        Endif
    Endif
    If process transaction flag = "yes"
        Get current date and time
        Do write files module
        Set data input flag complete flag = "yes"
    Endif
Enddo
Do terminate module

End Direct Labor Hours Capture

```

Gambar 4.12 Bahasa Inggris terstruktur keseluruhan: Program penangkapan data pekerja langsung JOCS.

Lingkaran 3: Setelah Jannis menyelesaikan pengujian top-down pada program tersebut, program ini disampaikan dikembalikan dalam kesatuan kepada Tom Pearson untuk dilakukan pengujian black box terhadapnya. Dalam strategi pengujian perangkat lunak Jake, anggota tim yang bertanggung jawab untuk merancang kembali dan mengkode program tidak akan melakukan pengujian data yang terlalu banyak pada program yang sama.

Tom Pearson diberi tugas melakukan pengujian program lengkap untuk proyek itu, karena ia berpengalaman dalam pemrograman dan analisis fungsi, dan ia diberi tanggung jawab untuk mengidentifikasi dan memperbaiki kerusakan atau kesalahan dalam program-program orang lain.

Tom menggunakan tiga set data yang berlainan untuk melakukan pengujian stress terhadap program-program itu: *Data normal*, data yang diharapkan berada dalam batasan atau lingkup program; *data abnormal*, data yang keluar dari jangkauan nilai yang diharapkan; dan *data tidak benar*, data yang berisi nilai-nilai invalid. Sebagai contoh, Tom menguji program Jannis dengan menggunakan nomor pekerjaan yang tak lengkap, nomor pekerjaan yang lengkap, nomor pekerjaan yang tidak berada dalam sistem itu, dan nomor pekerjaan yang berisi nilai alfabetis maupun nilai numerik.

Ia juga menjalankan berbagai macam pengujian untuk mencari tahu apa yang terjadi terhadap program jika ia digagalkan selama pemrosesan, seperti jika catu daya mati di tengah pemrosesan, dan ia mengecek ketentuan keamanan untuk akses program.

Lingkaran 4: Setelah Tom menyelesaikan pengujian program Jannis, ia meminta seorang pemakai untuk menjalankan program tersebut. Tom mengamati segala masalah yang dialami pemakai, seperti kesalahan input data, kesulitan interface, dan ke salahpahaman yang disebabkan oleh pesan kesalahan yang tak jelas.

Tugas pengujian terakhir, pengujian beta, akan ditunda sampai keseluruhan sistem itu siap diuji. Ketika telah siap, program Jannis akan direlease untuk persetujuan pengujian akhir.

Jake ingin memonitor seluruh proses pengembangan. Sebagaimana dibahas dalam Bab 1, ia mengukur jumlah waktu yang diperlukan oleh anggota tim SWAT untuk mengerjakan proyek itu guna mengukur angka produktivitas. Selain untuk mengukur produktivitas, Jake juga mengukur kualitas staf. Untuk melakukan hal ini, Jake menginstruksikan Tom untuk mencatat kesalahan atau kerusakan yang ia temukan dalam program, dan juga menuruhnya untuk mengidentifikasi penyebab kesalahan itu. Tom mungkin bisa benar-benar memperbaiki kerusakan jika waktunya mengijinkan. Informasi Tom ini diumpulkan balik ke programmer, yang mungkin akan memperbaiki kerusakan itu (jika Tom belum memperbaikinya). Para programmer selalu diberi informasi kerusakan yang ditemukan dalam program mereka ini, sehingga mereka dapat menggunakan informasi ini untuk menghasilkan program-program baru tanpa mengandung jenis kerusakan atau kesalahan yang sama.

5

PENGIMPLEMENTASIAN SISTEM

APA YANG AKAN ANDA PELAJARI DALAM BAB INI?

Setelah mempelajari bab ini, anda diharapkan dapat:

- Menyusun rencana implementasi sistem dengan menggunakan PERT.
- Mengelaskan bagaimana menyiapkan tampilan untuk platform teknologi.
- Menjelaskan cara melatih orang-orang untuk bekerja dengan sistem baru.
- Mengelaskan dokumentasi yang harus disiapkan.
- Mengelaskan empat metoda konversi sistem.
- Mengelaskan aruslan posco implementasi.

PENDAHULUAN

Implementasi sistem adalah tahap terakhir dalam siklus pengembangan sistem (SDLC). Untuk mendapatkan gambaran tentang apa yang telah kita capai sejauh ini, lihat Gambar 5.1.

Implementasi sistem melibatkan pengintegrasian semua komponen rancangan sistem, termasuk perangkat lunak, dan pengkonversian sistem total atau keseluruhan ke operasi. Implementasi sistem melibatkan proses dua langkah:

1. Perencanaan
2. Pengeksekusian

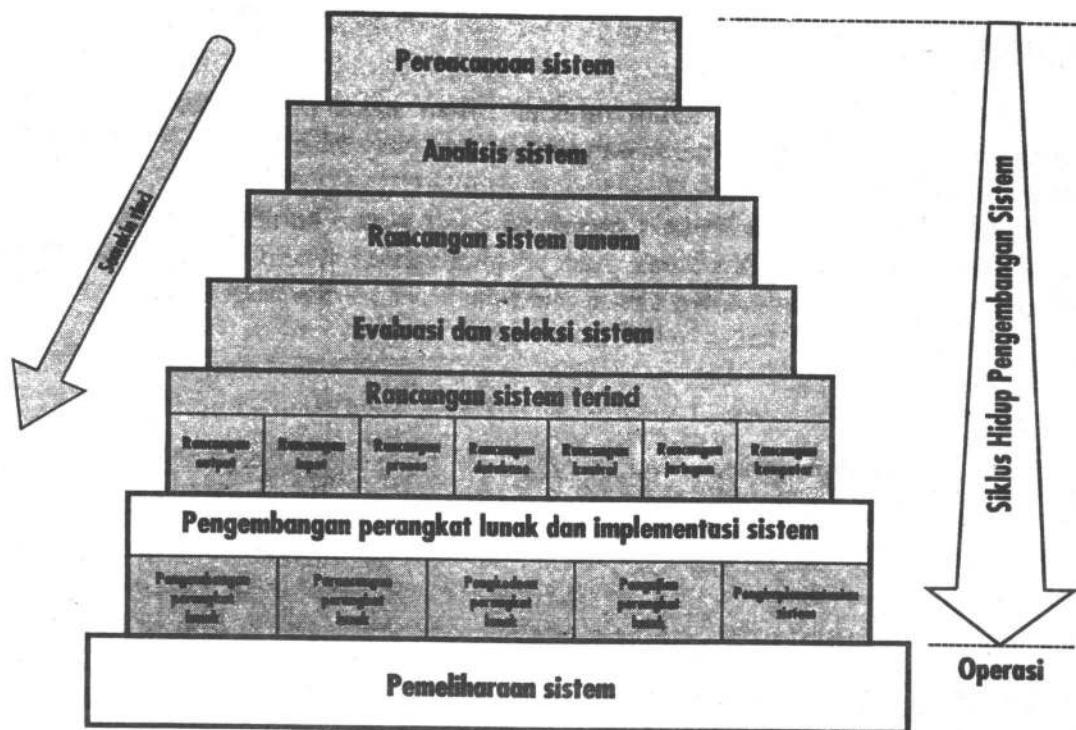
Perencanaan menentukan atau menetapkan tugas-tugas pengimplementasi sistem, apa yang akan dijalankan dan kapan waktunya. Pengeksekusian adalah menjalankan tugas-tugas pengimplementasi sistem sebenarnya.

MENCIPTAKAN RENCANA IMPLEMENTASI SISTEM

Rencana Implementasi Sistem adalah formulasi rinci dan representasi grafik mengenai cara pencapaian implementasi sistem yang akan dilaksanakan. Umumnya, rencana implementasi sistem disiapkan beberapa minggu atau bulan sebelumnya, tergantung pada cakupan dan kompleksitas proyek. Laporan implementasi sistem berisi rencana implementasi dan hasil-hasil tugas yang dilakukan menurut rencana ini. Untuk proyek yang sangat sederhana, seperti penginstalan paket perangkat lunak untuk mikrokomputer, pengimplementasi yang sebenarnya mungkin hanya memerlukan waktu satu jam atau kurang. Dalam kasus ini, laporan implementasi sistem bisa berupa memo singkat yang berisi instruksi kepada pemakai.

Untuk proyek lebih besar yang dibangun dari scratch, waktu pelaksanaan rencana diukur dalam bulanan. Ia berisi pengkoordinasian dan penjadwalan aktivitas dan tugas-tugas yang akan dijalankan oleh tim implementasi yang dibentuk atau terdiri dari berikut ini:

- ◆ Profesional sistem yang merancang sistem tersebut
- ◆ Para manajer dan berbagai staf
- ◆ Perwakilan vendor
- ◆ Pemakai primer

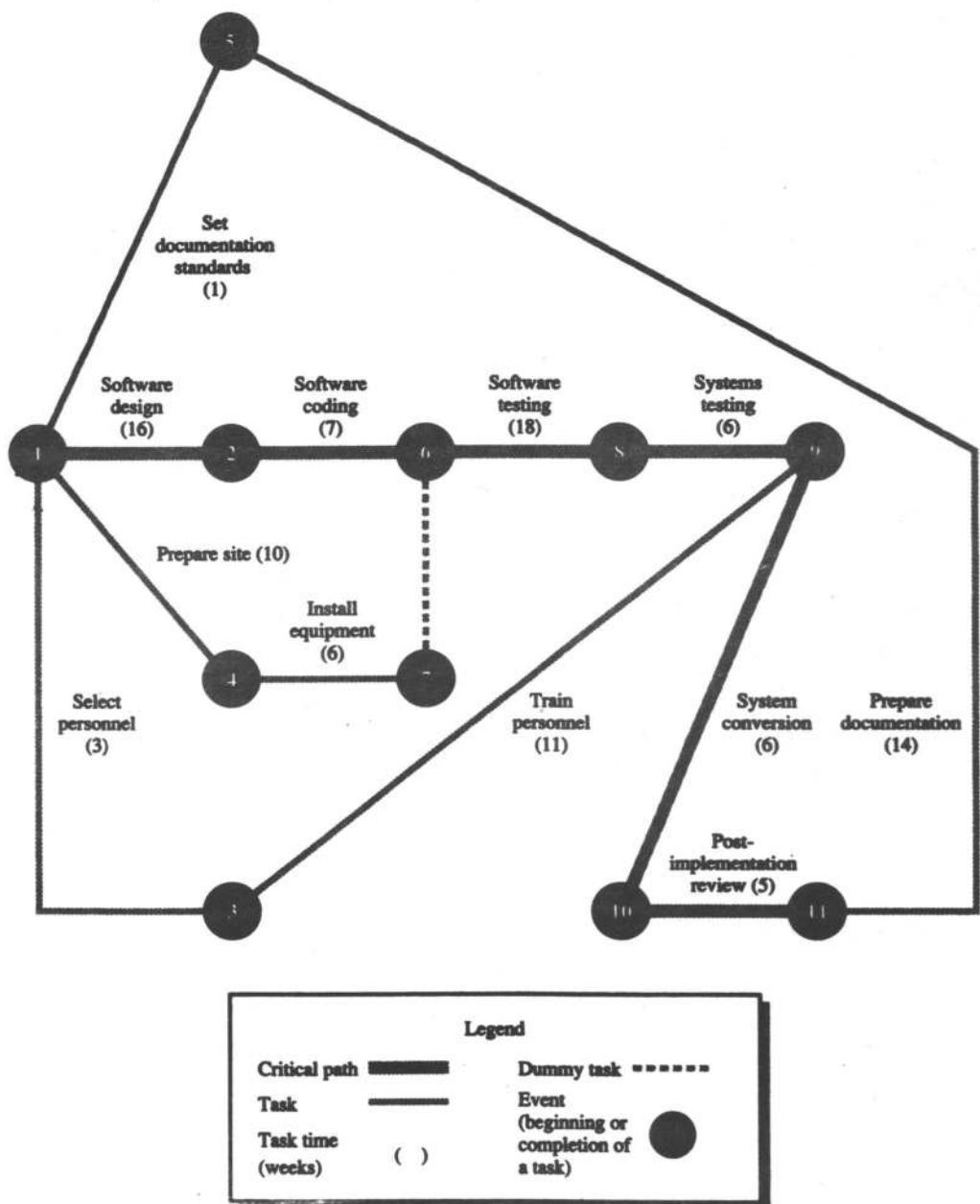


Gambar 5.1 Tahap SDLC dan bab yang terkait dalam buku ini. Dalam Bab 5, penekanan pembahasannya pada implementasi sistem, yang meliputi perencanaan dan tinjauan pasca implementasi.

- ◆ Pengkode
- ◆ Teknisi

Rencana implementasi proyek sistem besar biasa ditunjukkan pada Gambar 5.2. Pada poin ini, teknologi telah dievaluasi dan dipilih. Pesanan pembelian telah dikeluarkan untuk membeli dan memperoleh teknologi terpilih. Persiapan tempat dimulai segera, sehingga ia siap menerima platform teknologi dari vendor.

Tim pemrograman dibentuk, dan pengkode mulai memeriksa spesifikasi rancangan untuk memastikan pemahaman yang jelas sebelum pengkodean dan pengujian dimulai. Pelatihan untuk semua personel dan pemakai dimulai segera. Pelatihan ini berlanjut sampai tamat. Apabila perangkat telah tiba, maka tempatnya telah siap untuk penginstalasian.



Gambar 5.2 Rencana implementasi sistem yang direpresentasikan oleh diagram PERT.

Program-program harus diuji dengan test case untuk memastikan bahwa kita telah memperoleh program yang mempunyai reliabilitas tinggi. Test case juga disiapkan dan diterapkan pada input, output, database, dan kendali untuk memastikan bahwa sistem total bekerja sesuai yang diharapkan.

Jika hasil pengujian berhasil baik, sistem tersebut masuk atau menuju ke proses konversi. Biasanya, peninjauan pasca implementasi dilakukan untuk memastikan sistem tersebut berhasil dikonversi dan untuk memastikan bahwa ia beroperasi seperti yang diharapkan dengan menggunakan data hidup. Fine-tuning akan bisa meningkatkan kinerja sistem baru.

Selama jangka waktu ini, para pemakai atau wakil mereka menjalankan tes penerimaan. Jika segala sesuatunya pada tahap ini memuaskan — berjalan lancar, memberikan hasil yang benar, tercipta dokumentasi yang jelas dan komprehensif, para personelnya telah terlatih dengan baik, dan prosedur pengoperasiannya jelas — maka sistem tersebut bisa dialihkan dari status pengembangan ke status operasional.

Karena pengembangan perangkat lunak biasanya menjadi bagian penting dari implementasi sistem, maka kita curahkan tiga bab sebelumnya untuk membahas pengembangan perangkat lunak ini. Sekarang akan kita alihkan perhatian kita untuk membahas bagian-bagian pokok implementasi sistem:

- ◆ Persiapan tempat
- ◆ Pelatihan personel
- ◆ Persiapan atau pembuatan dokumentasi
- ◆ Konversi file dan sistem
- ◆ Peninjauan pasca implementasi

MENYIAPKAN TEMPAT

Jika platform teknologinya adalah mikrokomputer, kita hanya memerlukan persiapan tempat yang kecil, kecuali apabila kita ingin membangun kendali lingkungan, fisik, dan kendali database, seperti yang dikemukakan sebelumnya. Jika kita akan menginstal mainframe atau jaringan yang besar, kita mungkin perlu membangun fasilitas baru dan memodel ulang fasilitas yang telah ada.

Setiap perlengkapan dan furniture harus disediakan secara memadai. Kita harus menginstal listrik, telepon, sambungan-sambungan lain, ventilasi, dan AC untuk membentuk lingkungan yang bersih dan layak untuk bekerja. Harus disediakan pula keset antidebu dan karpet. Kita sediakan pula rak, penyangga barang-barang, meja,

penyimpan disk dan pita, dan lemari kabinet. Untuk personel, diperlukan tempat-tempat pribadi. Mereka diberi tempat printer sendiri yang kedap suara, dudukan printer, dan perlengkapan furniture serta workstation yang dirancang secara ergonomis.

Tersebut tersebut relatif harus tidak bisa diakses oleh umum dan pekerja yang tidak mempunyai wewenang. Selanjutnya, uji teknologi on-site (di tempat) dapat dimulai dengan angka keberhasilan yang tinggi, sehingga jika terdeteksi kesalahan atau terjadi kegagalan, pengujian ini tidak akan berlangsung terus tanpa terganggu oleh ketidakcukupan tempat.

Sebelum vendor mengantar perangkat, kita perlu mendiagnosis perangkat tersebut di tempat vendor (penjual). Diagnosa dilakukan dengan mensimulasi pelaksanaan (run) operasi yang terus menerus dalam beberapa minggu dengan menggunakan perangkat itu. Pengujian burn-in ini untuk mengecek adanya korsleteng (hubungan singkat) sistem listrik, tombol-tombol, konektor, dan berbagai komponen lainnya.

Sementara pengujian burn-in ini dilakukan, preparasi tempat untuk perangkat harus mendekati penyelesaian. Selain itu, tempat ini harus memenuhi standart lingkungan agar pengujian perangkat lunak dan sistem bisa valid apabila perangkat tersebut telah tiba dari vendor.

PELATIHAN PERSONEL

Tak ada sistem yang dapat bekerja secara memuaskan jika para pemakai dan orang lain yang berinteraksi dengan sistem tersebut tidak dilatih secara benar. Tanpa adanya keterampilan dan pengetahuan yang memadai dan jika personel tidak menerimanya, sistem tersebut bisa jadi gagal. Pelatihan personel tidak hanya meningkatkan keahlian/keterampilan pemakai, namun juga memudahkan penerimaan mereka terhadap sistem baru tersebut — yang ini adalah faktor penting untuk keberhasilan sistem itu.

Umumnya, ada tiga kelompok yang perlu diberi pelatihan:

1. Personel teknis yang akan mengoperasikan dan memelihara sistem tersebut.
2. Berbagai pekerja dan supervisor yang akan berinteraksi langsung dengan sistem itu untuk mengerjakan tugas-tugas mereka dan membuat keputusan.
3. Manajer umum (general manager).

Jika perusahaan telah memperluas sistem informasinya ke luar batas atau tembok korporasi, misalnya dengan menerapkan sistem pertukaran data elektronik, maka

ada lagi kelompok keempat yang juga perlu diberi pelatihan, yaitu mereka yang tidak bekerja untuk perusahaan tersebut namun akan berinteraksi dengan sistemnya, seperti pelanggan dan pemasok. Semua kelompok ini diberi nama “user” (pemakai).

Pelatihan harus dimulai sedini mungkin dengan waktu yang hampir bersamaan dengan operasionalnya sistem itu. Pelatihan membuat para pemakai percaya diri dan mengarahkan mereka secara pasti, dan juga meminimisasi kerusakan atau kesalahan selama tahap awal operasi sistem.

Idealnya, pelatihan harus dilakukan sebelum sistem komputer dan jaringan yang baru diinstal, dengan asumsi bahwa sistem tersebut memerlukan teknologi baru. Walaupun pendekatan atau cara ini tidak selalu layak, namun pelatihan pra-instalasi seharusnya dilakukan sesering mungkin. Apabila tidak demikian, sistem tersebut akan idle (mengalami masa kosong), karena ketika sistem telah bisa operasional, para pemakainya sedang menjalani pelatihan.

Program Pelatihan

Cakupan pelatihan bisa dari berupa tutorial, yang mengajarkan cara menjalankan tugas baru namun sederhana/mudah untuk satu pemakai, sampai pelatihan untuk mengajarkan pokok-pokok sistem baru kepada sebagian pemakai, jika tidak semua, di seluruh organisasi itu. Program pelatihan yang mendukung agenda pelatihan ekstensif meliputi:

- ◆ Pelatihan in-house (swa-kelola/di tempat)
- ◆ Pelatihan yang disediakan oleh vendor
- ◆ Jasa pelatihan luar

Salah satu atau kombinasi dari program-program pelatihan tersebut bisa digunakan.

Pelatihan In-House

Keuntungan pelatihan pemakai on-site (di tempat) adalah bahwa instruksi-instruksinya dapat disesuaikan dengan kebutuhan tertentu sistem itu dan pemakainya. Kelemahannya adalah bahwa para pemakai berada di lingkungan mereka sendiri dan mungkin akan terganggu oleh dering dan panggilan telepon dan terjadinya keadaan darurat, yang ini akan mengganggu sesion pelatihan itu.

Beberapa perusahaan menggunakan atau mempekerjakan staf bagian informasi dan staf pelatihan untuk membantu pelatihan pemakai ini. Staf bagian informasi

dapat memberikan kursus pendahuluan atau pengantar dan mendemonstrasikan perangkat atau aplikasi, memberikan pelatihan khusus, membantu pemakai menggunakan paket perangkat lunak tertentu untuk mengerjakan masalah bisnis tertentu pula, dan menyusun rencana workshop. Bagian atau pusat informasi ini bertugas mendorong pemakai untuk mengembangkan dan meneliti lebih jauh manfaat dan kegunaan sistem informasi dan membantu pemakai dalam memecahkan masalahnya sendiri. Dalam beberapa kasus, para pemakai yang bermotivasi tinggi dilatih sebagai super user yang tugas atau fungsi mereka membantu melatih pemakai lain dalam departemen mereka.

Pelatihan yang Disediakan Vendor

Acapkali sumber pelatihan terbaik yang mengajarkan cara menggunakan perangkat komputer, jaringan, atau sistem manajemen database adalah vendor yang memasoknya. Banyak vendor menawarkan program pelatihan ekstensif sebagai bagian dari layanan mereka. Dalam beberapa kasus, vendor akan mengenakan biaya untuk layanan ini; dalam kasus lain, jasa pelatihan ini gratis. Kebanyakan pelatihan yang disediakan vendor bersifat praktis (*hands-on*), sehingga para petatar bisa menggunakan sistem tersebut dengan sebenarnya dengan kehadiran pelatih. Lebih dari itu, sistem yang sedang digunakan untuk latihan petatar memang dirancang khusus untuk tujuan pelatihan. Oleh karena itu, tidak terjadi ketergesa-gesaan untuk menyelesaikan pelatihan ini, sehingga sistem itu bisa dikonversi ke operasi secara sepenuhnya pada waktu pelatihan, dan ini nantinya akan dialami para petatar di perusahaan mereka. Pengiriman pemakai ke kursus singkat di luar perusahaan yang memberikan pelatihan lebih mendalam mungkin lebih disukai dari pada pelatihan *in-house*. Juga, dalam hal ini para pemakai dapat dilatih sebelum hardware dan perangkat lunak diinstal. Apabila telah diinstal, para pemakai akan bisa mulai bekerja dengannya dengan segera.

Jasa Pelatihan Luar

Jika vendor tidak dapat menyediakan pelatihan yang diperlukan, atau perusahaan tidak mempunyai dana dan keperluan yang berkelanjutan untuk mendukung program pelatihan *in-house*, manajemen harus memilih jasa pelatihan luar, seperti universitas, masyarakat profesional, lembaga pelatihan (misalnya, MIS Institute Training), dan pusat-pusat pelatihan.

Sekolah bisnis seringkali memberikan atau menyediakan berbagai macam kursus-kursus bahasa pemrograman, aplikasi perangkat lunak, dan penggunaan komputer. Walaupun kursus ini mungkin membantu dan informatif, banyak pemakai tidak sanggup menunggu sampai habis semester untuk menguasai keterampilan yang diperlukan. Namun demikian, divisi pendidikan yang berkelanjutan dari suatu universitas biasanya menawarkan kursus singkat tentang paket perangkat lunak populer. Ia juga dapat dikontrak untuk memberikan kursus-kursus yang disesuaikan dengan kebutuhan perusahaan.

Baik masyarakat profesional dan lembaga pelatihan menawarkan berbagai macam seminar yang membahas topik-topik spesifik. Seminar-seminar ini cocok untuk para pemakai yang sibuk, sebab mereka hanya memerlukan waktu beberapa jam atau hari dan umumnya memberikan pelatihan yang praktis.

Pusat-pusat pelatihan didirikan untuk melatih para pemakai mengenali jalur produk vendor tertentu. Pusat-pusat pelatihan resmi (yakni, yang diberi wewenang oleh vendor tersebut) memenuhi kebutuhan akan pelatihan yang seragam dan efektif. Banyak pusat pelatihan resmi menawarkan kelas-kelas yang terjadwal secara reguler, kurikulum dan materi kursus yang disesuaikan, dan penilaian kelompok korporasi.

Teknik dan Alat Bantu Pelatihan

Untuk melaksanakan pelatihan yang sebenarnya, berbagai digunakanlah teknik dan alat bantu pelatihan. Mereka ini meliputi:

- ◆ Teleconferencing
- ◆ Perangkat lunak pelatihan interaktif
- ◆ Pelatihan dengan instruktur
- ◆ Pelatihan magang
- ◆ Manual prosedur
- ◆ Buku teks

Teleconferencing

Teleconferencing, yang juga disebut videoconferencing, dapat digunakan untuk beberapa tujuan, termasuk pelatihan. Sebagai alat bantu pelatihan, teleconferencing memungkinkan pemakai dan petatar yang secara geografis tempatnya berjauhan untuk menampilkan dan melihat teks, data, grafik, dan tampilan dan untuk saling berinteraksi secara real-time selagi mereka duduk di workstation mereka atau dalam

stasiun teleconferencing. Subset dari teleconferencing full-blown adalah video satu-satu arah dan audi dua-arah. Tampilan dan suara petatar ditransmisikan ke pemakai, dimana pemakai ini dapat mendengar namun tidak bisa melihatnya, seperti halnya dalam beberapa call-in di TV show. Para pemakai memanggil (call-in) dengan menanyai para petatar.

Setiap orang yang terlibat dalam sesion pelatihan dapat berjumpa sekali dan menjawab pertanyaan tanpa menyita banyak waktu dari pekerjaan mereka. Biasanya, kita tidak mungkin menghadirkan orang dalam jumlah besar pada satu lokasi dan sekali saat. Bagi mereka yang ikut dalam teleconferencing ini, mereka dapat menghemat biaya tiket pesawat terbang, biaya hotel, dan biaya makanan.

Alat bantu pelatihan ini memungkinkan para petatar menjangkau (berhubungan dengan) banyak pemakai sekali saat. Ia terutama akan bermanfaat apabila patatar mengemukakan tinjauan mereka terhadap sistem tersebut. Ia juga berguna dalam organisasi yang besar dimana banyak orang mempunyai tugas yang sama. Namun demikian, teleconferencing tidak akan mengganti pelatihan tutorial atau pelatihan dengan menggunakan instruksi orang ke orang.

Perangkat Lunak Pelatihan Interaktif

Perangkat lunak pelatihan interaktif memungkinkan dilakukannya dialog antara pemakai dan komputer. Perangkat lunak pelatihan seperti ini tersedia dalam berbagai bentuk:

Computer-Based Training (CBT). Software computer-based training (CBT) menggunakan mikrokomputer untuk memberikan pedoman kepada pemakai melalui serangkaian pelajaran yang efektif dan mudah dipelajari. Sebagian besar program CBT memungkinkan pemakai meninjau pelajaran (lesson), namun ia mempunyai fasilitas penangkapan kesalahan yang mencegah pemakai untuk berlanjut ke lesson berikutnya atau ke pertanyaan berikutnya jika mereka tidak diberi jawaban yang benar mengenai tugas yang mereka kerjakan saat itu.

Audio-Based Training. Biasanya perangkat lunak audio-based training disertai buku kerja. Untuk menggunakan jenis program pelatihan seperti ini, pemakai perlu mempunyai akses ke cassette player dan komputer. Seperti halnya program CBT, program audio yang paling efektif tersebut memungkinkan siswa menggunakan aplikasi perangkat lunak yang sebenarnya dalam pelatihannya.

Video-Based Training. Video-based training memerlukan akses ke TV, VCR, dan komputer. Biasanya, disertakan pula workbook (buku kerja). Program pelatihan ini memerlukan ruang fisik yang lebih besar dan perangkat yang lebih banyak. Lebih dari itu, para pemakainya dituntut untuk banyak menggerakkan mata. Ping-Ponging visual (gerakan mata kesana kemari) seperti ini membuat kikuk dalam melihat video dan dalam mencoba menggunakan komputer secara serentak.

Video-Optical Disk. Program pelatihan video-optical disk (yang juga disebut video interaktif), yang menggunakan CD-ROM, mengantarkan materi yang direkam sebelumnya ke monitor yang dihubungkan ke komputer. Pemakai dapat mengamati lesson dan menyelesaikan lesson ini dengan menggunakan layar yang sama. Ini mengurangi jumlah pergerakan mata yang diperlukan apabila kita menggunakan program video-based training. Namun demikian, program video-optical disk training relatif mahal, sebab ia memerlukan monitor khusus untuk mendukung toggling (penyerasan yang berhubungan) antara video dan program.

Semua program pelatihan interaktif di atas akan efektif jika para pemakai memanfaatkan mereka dan menggunakan mereka secara ermat dan sungguh-sungguh. Namun demikian, apabila para pemakai dibiarkan bekerja sendiri di tempat mereka sendiri, maka sulit bagi para petatar untuk memonitor kemajuan para pemakai ini. Jika pemakai mempunyai kebutuhan sendiri, perangkat lunak interaktif tidak dikustomisasi. Oleh karena itu, jika para pemakai tidak menggunakan perangkat lunak pelatihan interaktif, atau keperluan mereka perlu dikustomisasi, maka sebaiknya digunakan teknik pelatihan lain secara sendiri atau bersamaan dengan perangkat lunak pelatihan interaktif. Teknik pelatihan seperti ini adalah pelatihan dengan instruktur (instructor-based).

Pelatihan Dengan Instruktur

Instructor-based training (pelatihan dengan instruktur) memungkinkan instruktur ini memonitor kemajuan pemakai dan mengkustomisasi (menyesuaikan) lesson agar cocok untuk kebutuhan tertentu pemakai. Dengan hadirnya instruktur, ini akan membantu mengurangi intimidasi yang dirasakan oleh beberapa pemakai yang ditinggal sendiri dengan suatu komputer. Biasanya, hanya instruktur manusia yang akan bisa membantu pemakai mengatasi ketakutannya akan komputer dan membantu mengurangi kebingungan dan frustasi.

Seperti yang diisyaratkan namanya, teknik pelatihan ini lebih bersifat personal dan, akibatnya, cukup mahal. Dengan menggunakan teknik ini bersama dengan teknik

pelatihan lain, pelatihan dengan instruktur dapat menghilangkan segala kekosongan yang masih ada yang mengakibatkan pemakai tidak bisa puas memahami sistem tersebut. Dalam sistem yang tugas-tugasnya sangat kompleks atau yang tugas-tugasnya menjadi bagian kunci dari keberhasilan operasi, pelatihan dengan instruktur rupanya satu-satunya teknik yang layak.

Pelatihan Magang

Mungkin pendekatan atau cara yang paling banyak digunakan untuk melatih personel pengoperasian adalah hanya dengan menempatkan personel tersebut ke pekerjaan mereka yang sebenarnya. Setiap orang selalu dibebani tugas-tugas sederhana dan diberi instruksi spesifik tentang apa yang akan dilakukan dan bagaimana tugas ini dilakukan. Apabila tugas-tugas awal ini dikuasai, mereka diberi lagi tugas-tugas selanjutnya. Kurva pembelajaran dalam pendekatan ini bisa begitu panjang, dan, dalam banyak kasus, apabila kita tergesa-gesa dengan apa yang mereka hasilkan, maka hasilnya tidak akan baik. Lebih dari itu, jika operasi tertentu sangat kompleks dan sulit dikuasai, mungkin orang yang dibebani untuk mengerjakan tugas itu bisa saja frustasi dan minta tukar pekerjaan.

Manual Prosedur

Manual prosedur (yang juga disebut manual pemakai) membantu para pemakai yang tidak tahu cara mengoperasikan sistem tersebut. Instruksi-instruksinya akan memberitahu pemakai bagaimana menjalankan aplikasi-aplikasi yang berlainan, bagaimana menyimpan dokumen pada media penyimpanan sekunder, bagaimana mengupdate file, bagaimana menggunakan printer, dan sebagainya. Sebagai contoh, Gambar 5.3 menunjukkan cara penempatan bersama instruksi-instruksi guna menunjukkan cara pengisian layar kontrak pembelian. Layar pada bagian atas halaman menunjukkan rupa layar setelah ia diisi. Prosedur yang ada pada layar bagian bawah memberitahu pemakai secara pasti cara pengisian field entri data. Dengan tutorial yang minimal, para pemakai akan bisa menjalankan tugas entri data ini dengan sedikit kendala dan sedikit pelatihan lanjutan.

Buku Teks

Buku teks tradisional menjadi alat bantu pelatihan yang sangat baik untuk berbagai aplikasi perangkat lunak dan hardware. Namun demikian, apabila kita membaca

Rancangan Layar Kontrak Pembelian

CREATE PURCHASE CONTRACT		DATE: 07/14/93
SELLER CODE	CO	CORB FARMS
CONTRACT DATE	10/12/93	ROUTE/L: BOX 47
COMMODITY	CORN	DES MOINES IA 50318
COMMODITY COMMENT	14% PROTEIN OR MORE	
QUANTITY	300 BUSHELS	
DOCKAGE DEDUCTIBLE	ALL	
PRICE	\$4.50 PER BUSHEL UNDER KANSAS CITY CORN DECEMBER'S 93 OPTION	
SHIP TO	LUBBOCK	
SHIP DATE	12/20/93	
SHIP BY:	RAIL	
PF1=ACCEPT ENTRY; PF4=MODIFY ENTRY; PF16=CANCEL ENTRY		

Prosedur Proses

Get CRPURCNT Program

Do until operator indicates end of program
by pressing PF16.

Edit data field

Seller code:	must be valid code from SELLERS file
Contract date:	must be MM/DD/YY format
Commodity:	must be valide code from COMDTY file
Commodity comment:	no restrictions
Quantity:	must be positive value, no number greater than 999,999 and units must be bushels, pounds, or tons
Dockage Deductible:	must select all, all over 1%, or none
Price:	must be positive value not greater than 999,999.99 per bushel, pound, or ton
Ship to:	must enter a non-blank value
Ship date:	must be MM/DD/YY
Ship by:	must select rail, truck, or water

End-Do

Gambar 5.3 Instruksi-instruksi untuk menunjukkan cara mengisi layar kontrak pembelian.

buku, maka kita akan menghabiskan waktu yang terlalu banyak, dan dengan tidak adanya instruktur untuk menjawab pertanyaan, pemakai mungkin akan sangat cepat frustasi. Oleh karena itu, buku paling cocok digunakan untuk melengkapi teknik-teknik pelatihan yang lain.

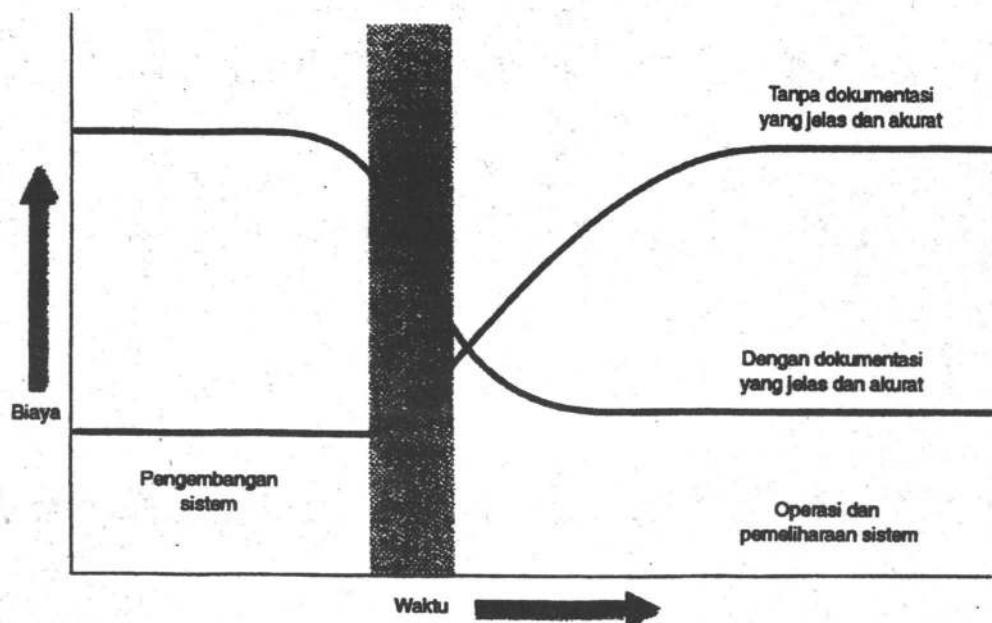
Mungkin pendekatan yang secara umum paling baik adalah dengan menggunakan kombinasi antara teknik-teknik pelatihan di atas. Sebagai contoh, teknik pelatihan interaktif dan buku yang digunakan bersama dengan pelatihan instructor-based (dengan instruktur) di kelas dapat memberikan basis pembelajaran yang kuat. Jika waktunya pendek dan sementara diperlukan pelatihan yang mendalam secara cepat, sebaiknya kita tidak menggunakan teknik pelatihan self-directed yang menggunakan buku atau kaset video. Teknik pelatihan dengan instruktur yang digabung dengan tutorial one-on-one (satu-satu) mungkin merupakan teknik pelatihan yang paling efektif untuk situasi seperti itu.

MENYIAPKAN DOKUMENTASI

Dokumentasi adalah materi tertulis (walaupun sebagian mungkin dalam bentuk video atau audio) yang menjelaskan cara beroperasinya suatu sistem. Deskripsi ini mencakup apa-apa yang dilakukan program dan prosedur yang harus diikuti oleh pemakai. Dokumentasi sistem yang baik adalah pokok bahasan yang perlu dikuasai dan ini sudah menjadi kesepakatan dari para pemakai. Namun, banyak diantaranya lalu mengabaikannya. Keadaan ini ironis, sebab hampir setiap orang memperoleh manfaat dari dokumentasi, termasuk manajer dan supervisor, pemakai, petatar, operator, profesional sistem, programmer pemeliharaan, dan auditor. Dokumentasi digunakan untuk tujuan-tujuan berikut ini:

- ◆ Pelatihan
- ◆ Penginstruksian
- ◆ Pengkomunikasian
- ◆ Penetapan standart kinerja
- ◆ Pemeliharaan sistem
- ◆ Referensi historis

Repository (tempat penyimpanan) sentral sistem CASE menyimpan dokumentasi untuk keperluan referensi dan pengupdetan yang mudah. Perubahan dalam operasi bisnis dan perubahan keperluan pemakai mengharuskan dilakukannya modifikasi



Gambar 5.4 Biaya sistem selama kehidupan sistem.

sistem dan program. Sebelum perubahan dapat dilakukan, orang yang membuat atau melakukan perubahan ini lebih dulu harus memahami apa yang diharapkan akan dilakukan oleh sistem tersebut. Program-program yang telah ditulis beberapa bulan atau tahun sebelumnya harus didokumentasi secara baik, sebab kita mudah lupa rincianya hanya dalam beberapa hari. Lebih dari itu, banyak proyek tidak bisa dipelihara dan dioperasikan selanjutnya, sebab personel yang membangun sistem yang dulu tidak meninggali pekerja lain dalam perusahaan itu dokumentasi yang memadai.

Analisis biaya total untuk sistem baru selama kehidupannya menunjukkan bahwa biaya pengembangan lebih sedikit dari pada biaya pemeliharaan dan operasi yang berkelanjutan, seperti terlihat pada Gambar 5.4. Rancangan sistem yang baik akan membantu mengurangi kebutuhan pemeliharaan dan biayanya. Namun ketika pemeliharaan diperlukan, ia akan bisa dijalankan secara lebih mudah dan lebih cepat jika dokumentasinya tersedia. Ada empat area utama dokumentasi, yaitu:

- ◆ **File Transaksi.** File ini selalu diciptakan dengan memproses suatu sub sistem individual di dalam sistem informasi. Akibatnya, ia harus dicek secara seksama selama pengujian sistem. Namun demikian, file transaksi yang digenerasi dalam area sistem informasi selain subsistem baru harus dikonversi jika file master yang ia update berubah format atau mediumnya.
- ◆ **File Indeks.** File ini berisi kunci atau alamat yang menghubungkan berbagai file master. File indeks baru harus diciptakan kapan saja file master yang berhubungan dengannya mengalami konversi.
- ◆ **File Tabel.** File ini dapat juga diciptakan dan dikonversi selama konversi sistem. File tabel bisa juga diciptakan untuk mendukung pengujian perangkat lunak. Jadi ia siap untuk konversi. Pertimbangan yang sama yang diperlukan master file berlaku disini.
- ◆ **File Backup.** Kegunaan atau tujuan file backup adalah untuk memberikan keamanan bagi database apabila terjadi kesalahan pemrosesan atau kerusakan dalam pusat data. Oleh karenanya, ketika suatu file dikonversi atau diciptakan, file backup harus diciptakan. Prosedur backup untuk file terkonversi hampir sama seperti prosedur yang ada untuk file asli. Namun demikian, media filenya mungkin berbeda. Sebagai contoh, file master disk magnetis bisa dibackup pada pita cartridge.

Konversi File Gradual

Beberapa perusahaan mengkonversi file-file data mereka secara gradual (sedikit dem sedikit). Record-record akan dikonversi hanya ketika mereka menunjukkan beberapa aktivitas transaksi. Record-record lama yang tidak menunjukkan aktivitas tidak pernah dikonversi. Metode ini bekerja dengan cara berikut:

1. Suatu transaksi diterima dan dimasukkan ke dalam sistem.
2. Program mencari file master baru (misalnya file inventarisasi atau file account receivable) untuk record yang tepat yang akan diupdate oleh transaksi itu. Jika record tersebut telah siap dikonversi, berarti pengupdatean record telah selesai.
3. Jika record tersebut tidak ditemukan dalam file master baru, file master lama diakses untuk record yang tepat, dan ia ditambahkan ke file master baru dan diupdate.

4. Jika transaksi tersebut adalah untuk record baru, yakni record yang tidak dijumpai pada file lama maupun file baru (misalnya, pelanggan baru), maka record baru disiapkan dan ditambahkan ke file master baru.

Biasanya pada akhir periode 30-hari, sebagian besar record aktif akan dikonversi ke file master baru. Beberapa perusahaan akan menemukan bahwa porsi file master lama mereka terkomposisi dari record-record tak-aktif. Oleh karena itu, penggunaan konversi file gradual yang dipicu oleh transaksi dapat sangat mengurangi usaha konversi file dan ruang yang akan diperlukan jika metode konversi file total digunakan.

MENGEVALUASI SISTEM BARU SETELAH IMPLEMENTASI

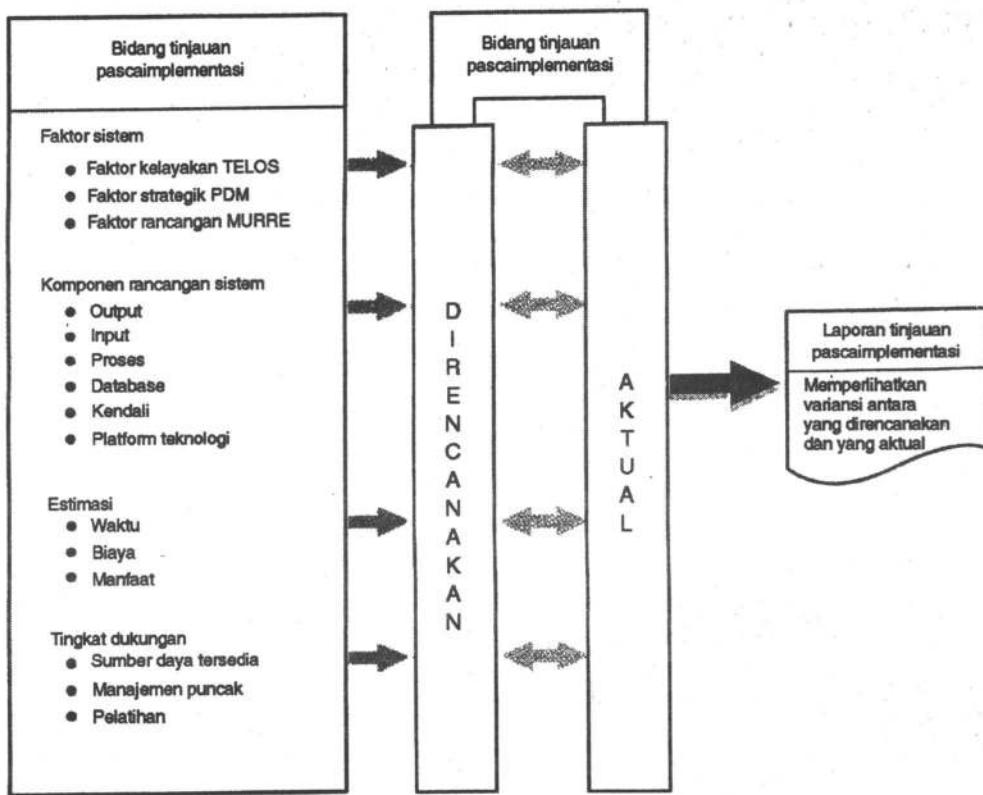
Walaupun tujuan pengembangan sistem baru berkualitas adalah untuk menghasilkan sistem yang tidak melampaui budget, on time, dan memenuhi keperluan pemakai, namun tujuan tersebut tidak selalu kesampaian. Kita memerlukan suatu proses untuk menganalisis apa yang berjalan dengan baik dan apa yang salah dalam proyek yang berhasil maupun yang tidak berhasil. Analisis ini dijalankan oleh peninjauan pascaimplementasi (postimplementation review) (yang juga disebut pengauditan atau evaluasi pascaimplementasi). Peninjauan pascaimplementasi adalah pencarian terorganisir untuk menemukan cara meningkatkan efisiensi dan efektivitas sistem baru, dan untuk memberikan informasi yang akan membantu dalam pengembangan sistem mendatang.

Biasanya, peninjauan pascaimplementasi dilakukan dua sampai enam bulan setelah konversi sistem. Periode ini memungkinkan paling sedikit dua siklus bulanan pengoperasian sistem. Pada point ini, ingatan semua orang yang terlibat masih agak segar dan bisa diandalkan, dan sistem bisa mempunyai waktu yang cukup untuk masuk ke dalam pola pengoperasian yang agak normal.

Peninjauan pascaimplementasi bisa dilakukan oleh tim yang terdiri atas wakil pemakai, auditor internal, dan profesional sistem. Namun, di dalam beberapa perusahaan, disertakan pula konsultan eksternal atau auditor independen untuk melakukan peninjauan guna membantu meningkatkan obyektivitas dan mengurangi kepentingan politik yang terjadi diantara kelompok-kelompok internal.

Berikut ini adalah empat area peninjauan pascaimplementasi yang perlu mendapat perhatian:

- ◆ Faktor-faktor sistem



Gambar 5.7 Proses tinjauan pascaimplementasi.

- ◆ Komponen rancangan sistem
- ◆ Keakuratan estimasi
- ◆ Tingkat dukungan

Peranan mereka dalam proses peninjauan pascaimplementasi ditunjukkan pada Gambar 5.7

Area peninjauan pertama mencakup faktor-faktor sistem, yaitu:

- ◆ Faktor kelayakan teknis, ekonomis, hukum, operasional, dan jadwal (TELOS)
- ◆ Faktor strategis produktivitas, diferensiasi, dan manajemen (PDM)
- ◆ Faktor rancangan kemampuan pemeliharaan, pendayagunaan, pendayagunaan kembali, reliabilitas, dan kemampuan perluasan (MURRE).

Area peninjauan kedua berkaitan dengan:

- ◆ Output
- ◆ Input
- ◆ Proses
- ◆ Database
- ◆ Kendali
- ◆ Platform teknologi

Area peninjauan ketiga menyelidiki:

- ◆ Waktu
- ◆ Biaya
- ◆ Keuntungan

Area peninjauan keempat dan terakhir mengenai:

- ◆ Sumber daya yang tersedia
- ◆ Manajemen puncak
- ◆ Pelatihan

Tinjauan Pascaimplementasi di Vulcan Industries

Vulcan Industries telah mengoperasikan sistem informasi baru selama kurang lebih empat bulan. Saat ini, ia menyewa SystemPro, suatu perusahaan konsultasi sistem independen, untuk menjalankan peninjauan pascaimplementasi terhadap sistem baru tersebut.

Tim peninjauan pascaimplementasi menyertakan Noel Stevenson, pimpinan tim peninjau; Sam Bennison; dan Trudy Carson. Semua anggota mempunyai latar belakang yang baik dalam pengembangan sistem dan konsultasi.

Komponen utama yang digunakan tim peninjauan pascaimplementasi SystemPro untuk mendokumentasi pekerjaannya adalah checklist (daftar urutan kerja). Setiap area peninjauan pascaimplementasi mempunyai daftarnya sendiri. Daftar-daftar (checklists) ini berisi pertanyaan peninjauan inti, skala penilaian, dan skor akhir area peninjauan. Daftar-daftar ini berfungsi sebagai dokumen inti yang dikemukakan kepada panitia pengarah dan pihak-pihak lain yang berkepentingan di Vulcan. Hasil-hasil peninjauan pascaimplementasi terutama membantu para profesional sistem untuk melakukan evaluasi kinerja. Juga, hasil-hasil ini memberikan informasi yang bisa digunakan untuk mengestimasi waktu, biaya, dan area-area masalah pengembangan sistem yang akan datang, dan untuk mengoreksi kesalahan.

Sejumlah besar pertanyaan yang ditujukan kepada pemakai dan profesional sistem merupakan perangkat pokok yang digunakan oleh tim peninjauan pascaimplementasi SystemPro. Disamping menanyai dan menginterview berbagai orang di seluruh Vulcan, tim peninjau juga melakukan sejumlah observasi dan pengujian sendiri. Sebagai contoh, Noel Stevenson menjalankan beberapa transaksi uji untuk menentukan efektivitas program perangkat lunak yang digabungkan kendali. Ia juga mensimulasi perusakan file pelanggan untuk melihat apakah file ini bisa diciptakan kembali dengan menggunakan prosedur backup database saat itu. Jadi, metode peninjauan pascaimplementasi SystemPro terdiri atas penginterviewan orang-orang seluruh organisasi dan menanyai mereka. Kemudian, apabila diperlukan, tim peninjauan pascaimplementasi tersebut melakukan observasi independen dan pengujian untuk memastikan bahwa keadaan sistem tersebut sesuai dengan jawaban-jawaban dari para responden yang diinterview. Metode peninjauan seperti ini memberikan basis yang solid yang digunakan untuk mendasari penilaian terhadap area peninjauan pasca implementasi.

Meninjau Faktor Sistem

Pada tahap perencanaan sistem, penilaian terhadap proyek sistem yang diusulkan ditujukan untuk faktor-faktor kelayakan TELOS dan faktor-faktor strategis PDM. Sekali lagi, selama evaluasi sistem dan tahap pemilihan sistem, faktor-faktor tersebut dinilai lagi berdasarkan faktor-faktor rancangan MURRE untuk rancangan sistem secara keseluruhan. Selama peninjauan pascaimplementasi, akan ditentukan sejauh mana faktor-faktor sistem dari sistem baru yang sekarang beroperasi sebanding

(sesuai) dengan faktor-faktor sistem yang diprediksikan untuknya selagi ia dikembangkan.

Faktor Kelayakan TELOS

Selama perencanaan sistem dan sekali lagi selama evaluasi dan pemilihan sistem, kelayakan proyek sistem dinilai berdasarkan faktor-faktor kelayakan TELOS. Penilaian TELOS yang aslinya (pada awalnya) ditujukan untuk sistem tersebut harus dibandingkan dengan sejauh mana yang dibayar oleh sistem yang diimplementasikan tersebut.

Faktor Strategik PDM

Alasan untuk mengembangkan sistem baru pada mulanya adalah karena adanya peluang untuk meningkatkan faktor-faktor strategis produktivitas, diferensiasi, dan manajemen (PDM). Penilaian PDM aslinya (awal) sekarang dibandingkan dengan penilaian PDM yang ditujukan untuk sistem tersebut setelah implementasinya. Penilaian ini harus memberikan hasil yang mendekati. Khususnya, manajemen ingin mengetahui apakah sistem baru tersebut telah memberi perusahaan keuntungan kompetitif.

Faktor Rancangan MURRE

Sekarang, kita akan relatif mudah menilai sejauh mana kemampuan pemeliharaan sistem baru tersebut. Mungkin programmer pemeliharaan telah melakukan perubahan berdasarkan pada perubahan dalam operasi bisnis. Apakah programmer pemeliharaan memahami program tersebut dan apakah perubahan itu dilakukan dengan mudah? Pendayagunaan dan pendayagunaan kembali harus ditentukan secara baik dari sekarang, sebab sistem tersebut telah digunakan secara sebenarnya. Juga, reliabilitas harus diuji secara baik pada point ini. Derajad kemampuan perluasan/pengembangan mungkin belum bisa ditentukan, sebab sistem tersebut belum beroperasi cukup lama.

Hasil-hasil peninjauan faktor-faktor sistem yang disusun oleh tim peninjauan pascaimplementasi SystemPro terlihat pada Gambar 5.8. Pertanyaan-pertanyaan yang disampaikan oleh tim peninjau untuk mendapatkan penilaian yang wajar disertakan atau dimasukkan di bawah setiap kategori faktor-faktor sistem.

DAFTAR PERIKA PENILAIAN TINJAUAN FAKTOR SISTEM

Faktor kelayakan TELOS:

8,5

- Apakah teknologi yang mendukung rancangan sistem tersebut seperti yang diestimasikan pada awalnya?
- Apakah ada dana yang cukup untuk memperoleh sumber-sumber komputasi?
- Apakah ada dana yang cukup untuk mengoperasikan dan memelihara sistem tersebut?
- Apakah sistem tersebut sesuai dengan (tidak melanggar) hukum dan peraturan?
- Apakah orang-orang mempunyai keterampilan yang diperlukan untuk mengoperasikan, memelihara, dan menggunakan sistem tersebut?
- Apakah jadwal keseluruhan proyek sistem itu diikuti?

Faktor Strategis PDM:

8,0

- Apakah pencapaian produkivitas telah dihasilkan oleh sistem baru tersebut?
- Apakah sistem baru itu telah mengkontribusi pendeferensiasian produk perusahaan, seperti gaya, kualitas, harga, dan kekhasan?
- Apakah sistem tersebut telah mengkontribusi pendeferensiasian layanan perusahaan kepada para pelanggannya dengan memberikan produk dan penelusuran informasi dan meningkatkan kualitas respon terhadap pesanan dan mengurangi waktu pengantaran?
- Apakah sistem tersebut menghasilkan informasi yang bisa meningkatkan kualitas perencanaan, pengontrolan, dan pembuatan keputusan manajemen?

Faktor Rancangan MURRE:

7,5

- Apakah dokumentasinya komprehensif, jelas, dan baru?
- Apakah sistem manajemen perbaikan (CMS) tersedia dan bekerja?
- Apakah keperluan pemakai terpenuhi?
- Apakah modul-modul perangkat lunak bisa digunakan kembali?
- Apakah sistem tersebut bebas-kesalahan?
- Apakah sistem tersebut fleksibel dan adaptif?

Total 24,0

Hasil akhir (24,0/3) 8,0

Skala penilaian:

0 5 10

Buruk Sedang Baik sekali

Gambar 5.8 Hasil-hasil peninjauan faktor sistem di Vulcan Industries.

Meninjau Komponen Rancangan Sistem

Komponen-komponen rancangan sistem ditinjau kembali untuk melihat sejauh mana mereka dirancang dan diimplementasikan. Mereka akan dibahas berikut ini.

Output

Sistem informasi tersebut ditinjau dan dievaluasi dalam hal informasi yang ia berikan. Sistem yang cocok untuk pemakai yang harus menjalankan tugas-tugas dan membuat keputusan bukan hanya sistem yang melulu berfungsi dengan baik. Output yang dihasilkan oleh sistem tersebut harus bisa digunakan kembali, bisa diakses, relevan, dan akurat. Oleh karena itu, komponen rancangan output perlu diperhatikan secara khusus oleh tim peninjauan pascaimplementasi.

Peninjauan output teknis lebih berurusan dengan perbandingan bentuk output dan substansinya. Peninjauan ini meliputi pengecekan heading yang benar, jumlah yang teredit (misalnya, peniadaan nol depan, notasi debet/kredit, tanda dolar), urutan jumlah halaman yang benar, indikator akhir-laporan, dan tanggal yang benar (misalnya, tanggal laporan dibuat dan tanggal saat itu).

Input

Peninjauan komponen rancangan input melibatkan penentuan apakah form-form kertas dan elektronik dan layar entri data memenuhi pedoman dan spesifikasi rancangan. Para pemakai juga diuji untuk menentukan apakah mereka mengisi form-form tersebut secara benar. Disini kita asumsikan bahwa mereka telah diajari cara mengisi form-form tersebut selama pelatihan. Peninjauan pascaimplementasi dilakukan untuk menentukan apakah para pemakai itu dilatih dengan benar.

Jika input ditangani oleh perangkat point-of-sale, kita pilih sampel sederhana secara random dan kita sampaikan kepada pembaca untuk menentukan kebenaran harga dan deskripsinya. Jika produk tertentu tidak berisi kode bar, harus tersedia keyboard untuk memasukkan data tersebut secara manual. Jika input yang dimasukkan dari keyboard ditampilkan pada VDT, maka perlu kita buat layout layar yang sesuai dan baik. Layar yang tidak jelas dan berisi instruksi atau tampilan yang tidak penting dan membingungkan harus diidentifikasi dan dikoreksi.

Proses

Secara lebih luas lagi, peninjauan komponen rancangan lain adalah serupa dengan proses peninjauan. Sebagai contoh, jika outputnya benar, maka perangkat lunak dianggap berjalan secara benar. Perlu diingat bahwa perangkat lunak juga mengalami sejumlah pengujian selama tahap-tahap awal pengimplementasi. Namun perangkat lunak bisa saja menghasilkan output yang benar dan ternyata masih belum beroperasi secara memuaskan menurut pandangan operator. Misalnya, suatu program yang menghasilkan output yang baik secara sempurna bisa saja berhenti suatu saat karena alasan yang tak jelas. Menurut pandangan orang yang mengoperasikan sistem itu, perangkat lunak seperti ini jelas berkualitas tidak baik dan harus dikoreksi.

Aspek-aspek lain dari komponen rancangan proses yang perlu mendapatkan peninjauan adalah prosedur dan dokumentasi. Prosedur harus mudah diikuti dan diselesaikan agar berbagai personel bisa menjalankan tugas-tugas yang dibebankan kepada mereka. Prosedur yang ditulis secara tidak baik akan sulit dipahami dan diikuti, dan dalam beberapa kasus, dapat menyebabkan suatu tugas dikerjakan secara tidak benar. Jika terjadi prosedur seperti ini, maka ia harus ditulis kembali. Segala ketidakteraturan dalam dokumentasi perlu dikoreksi.

Database

Selanjutnya, peninjauan output mengharuskan dilakukannya peninjauan database secara serentak. Namun demikian, peninjauan tambahan harus dilakukan untuk memastikan bahwa database memenuhi semua permintaan atau keperluan yang telah ditetapkan, khususnya dari sudut pandang kendali dan kinerja umum. Peninjauan kendali mencakup tugas-tugas berikut ini:

- ◆ Menciptakan record baru setelah record terakhir
- ◆ Menciptakan suatu record untuk divisi atau departemen yang belum ada
- ◆ Mencoba membaca dari atau menulis ke file dengan label header yang salah
- ◆ Berusaha memproses melalui indikator end-of-file
- ◆ Mencoba menciptakan dan memasukkan suatu record yang tidak lengkap

File-file harus dicek kelengkapannya. Total kendali yang telah ditetapkan sebelumnya harus dibandingkan dengan total yang dihasilkan dari file-file baru. Sebagai contoh, total kendali yang telah ditetapkan sebelumnya bisa dicek berdasarkan jumlah record dalam suatu file atau jumlah total dari field jumlah tertentu, seperti

AMOUNT-OWED dalam file master account receivable. Deskripsi dan layout file harus dibandingkan dengan ENVIRONMENT dan DATA DIVISION dalam program COBOL untuk mencari kesepakatan atau kesesuaianya.

Dalam lingkungan query-intensive, pengujian penggabungan SQL sangat penting. Yang biasa digunakan adalah penggabungan dua-tabel, tiga-tabel, dan dalam beberapa kasus, empat-tabel. Semua penggabungan (join) ini harus diuji dengan data yang merefleksikan transaksi hidup.

Kualitas query optimizer adalah komponen tunggal terpenting dari DBMS berbasis-SQL. Oleh karena itu, ia harus diuji untuk memastikan kemampuannya menangani penggabungan secara efisien. Test case harus dibuat guna menguji sensitivitas optimizer dalam menggabungkan kondisi-kondisi dalam klausa WHERE, seperti

```
SELECT NAME
FROM EMPLOYEE
WHERE EMPLOYEE.EMP-NO = DEPARTMENT.EMP-NO
```

Kemudian ujilah dengan mentransposisi (mengubah urutan) kondisi WHERE sebagai berikut:

```
DEPARTMENT.EMP-NO = EMPLOYEE.EMP-NO
```

Hasilnya mungkin berbeda dari query pertama sebab optimizer kadang-kadang dapat diblokir menggunakan indeks-indeks tabel yang salah.

Kendali

Tujuan peninjauan kendali adalah untuk memastikan bahwa kendali telah terbentuk dan bekerja sesuai yang diharapkan. Ini disebut pengujian compliance (pemenuhan). Ada tiga tahap yang terjadi dalam pengujian pemenuhan, yaitu:

1. Mempelajari dan mengobservasi kendali.
2. Melakukan pengujian pemenuhan yang sesungguhnya.
3. Mengevaluasi sejauh mana efektivitas kendali tersebut memenuhi pengujian pemenuhan ini.

Transaksi uji, misalnya, membantu memastikan bahwa kendali pemrosesan, seperti jangkauan dan kewajaran pengecekan dan pengontrolan akses, telah terbentuk dan bekerja secara benar.

Para peninjau bisa saja puas dengan hanya mengamatinya bahwa kendali tersebut bekerja dengan baik. Sebagai contoh, mereka dapat mengamati tag inspeksi dan sistem supresi-file. Para peninjau atau pengamat bahkan bisa menuji coba teknik-teknik kendali tertentu untuk melihat apakah mereka bekerja (misalnya, berbagai kendali akses). Dalam kasus lain, satu-satunya cara untuk memberitahu bahwa kendali tertentu bekerja adalah dengan menyusun (set up) situasi uji dan melihat apa yang terjadi. Sebagai contoh, simulasi bencana atau kerusakan bisa dijalankan dengan basis kejutan. Untuk menjalankan simulasi kerusakan, para peninjau menyegel file master tertentu dan memberitahu manajer pusat komputer bahwa sistem tersebut rusak. Situasi tersimulasi ini mengharuskan personel pemrosesan data untuk membawa sistem tersebut ke status saat itu dengan menggunakan file backup dan siklis, fasilitas backup yang lain, dan prosedur kebetulan. Jika gagal, penyebab kegagalan ini dipastikan dan perlu diambil tindakan pengoreksian. Juga, dengan persetujuan eksekutif, para peninjau mencabut switch catu daya yang menuju ke pusat komputer untuk menguji prosedur pemulihan. Kadang-kadang, para penembus profesional atau tiger team disewa untuk berusaha memperoleh akses ke pusat komputer dan database. Latihan-latihan mengatasi kebakaran juga dijalankan untuk melihat atau mengetahui apakah prosedur pengoperasian standar telah diikuti.

Platform Teknologi

Platform teknologi untuk sistem baru juga ditinjau, yaitu meliputi peripheral, workstation, prosesor, dan jaringan. Target atau tujuan utamanya adalah untuk membandingkan kinerja saat itu dengan spesifikasi rancangan. Hasil dari peninjauan ini akan menunjukkan perbedaan antara hasil yang diharapkan dengan hasil yang sebenarnya. Ia juga menunjukkan perlunya adanya modifikasi yang harus dilakukan.

Beberapa perangkat peninjauan platform teknologi yang lebih populer adalah:

- ◆ Sistem akunting job
- ◆ Monitor hardware
- ◆ Monitor perangkat lunak

Perangkat ini bertindak sebagai stetoskop atau perangkat pemeriksaan yang mengecek sejauh mana arsitektur komputer beroperasi.

Sistem Akuntasi Pekerjaan. Sistem ini dapat digunakan untuk menguji efisiensi rancangan; membantu perencanaan kapasitas, dan memproyeksikan pola pertumbuhan.

buhannya. Fasilitas manajemen sistem (SMF) IBM adalah contohnya. Ia menunjukkan siapa yang menggunakan sistem tersebut dan berapa lama mereka menggunakan-nya, bersama dengan file data yang diakses. SMF menunjukkan keberadaan ruang pada perangkat penyimpanan direct-access dan memberikan statistik dasar untuk file-file pita magnetis. Vendor-vendor lain, selain IBM, juga menyediakan paket perangkat lunak yang sama.

Monitor Hardware. Monitor hardware terdiri atas server-server yang dikoneksikan ke circuitry prosesor. Ia mengukur wait prosesor aktif, seek disk, transfer data disk, mount disk, active pita, read pita, dan timing dan pemanfaatan memori internal. Sensornya, selanjutnya, diarahkan ke komputer kecil yang merekam dan menampilkan berbagai sinyal.

Dengan statistik pemanfaatan yang cukup, program bisa dievaluasi, anggaran arsitektur komputer total bisa dikurangi membuktikan efisiensi melalui pengubahan prosesor, penambahan channel, atau perekonfigurasian hirarki penyimpanan. Tujuan utama penggunaan monitor ini adalah untuk mencocokkan power arsitektur komputer dengan permintaan sistem informasi. Namun demikian, selain untuk waktu respon terminal untuk setiap terminal dan jalur, monitor hardware tidak diperlengkapi dengan fasilitas untuk mengukur kinerja aplikasi sistem. Area ini diukur oleh monitor perangkat lunak.

Monitor Perangkat lunak. Untuk mengevaluasi kinerja sistem keseluruhan, monitor perangkat lunak dapat digunakan sendiri atau bersama dengan monitor hardware dan sistem akunting job. Monitor perangkat lunak adalah program yang berada dalam arsitektur komputer yang sedang ditinjau. Komponen-komponen perangkat lunak yang biasanya diukur adalah sistem pengoperasian, perangkat lunak pendukung, dan program aplikasi. Pengukuran sistem pengoperasian mengidentifikasi bagian-bagian kode yang tidak efisien. Keberadaan bagian-bagian semacam ini bisa mendorong manajemen untuk meminta vendor melakukan perbaikan, atau jika hal ini mustahil, perlu dicari perangkat lunak komersial lain sebagai penggantinya.

Perangkat lunak pendukung gagal dalam area kelabu yang berada diantara program aplikasi dan sistem pengoperasian. Walaupun ditangani dalam sistem persis seperti perangkat lunak aplikasi, perangkat lunak pendukung ditulis oleh vendor dan bisa terlihat oleh pemakai biasa sebagai bagian dari sistem pengoperasian. Apabila monitor perangkat lunak menunjukkan bahwa perangkat lunak pendukung tidak beroperasi secara optimal, vendor harus mengubahnya.

Perangkat lunak aplikasi diukur guna menentukan pemanfaatan (pendayagunaan) sumber dan efisiensi kode. Pengukuran ini mencatat dan melaporkan jumlah waktu yang diperlukan program untuk menggunakan setiap sumber, seperti memori internal, disk, dan pita. Sebagai contoh, suatu program mungkin meminta delapan tape drive, meskipun secara normal ia hanya membutuhkan empat. Jika kita perlu membatasi tape drive, para peninjau harus meminta programmer pemeliharaan untuk mengubah program itu.

Monitor perangkat lunak dapat mengisolasi area heavy paging dalam sistem penyimpanan virtual. Heavy paging dapat terjadi apabila perangkat lunak aplikasi mengakses routine secara berulang-ulang, sehingga menyebabkan terjadinya kondisi thrashing-of-pages. Dengan analisis dan penulisan kembali program, ketidakefisiensian perangkat lunak ini bisa dikurangi. Juga, dengan caching (menyembunyikan) data dan program yang sering digunakan dalam disk semikonduktor (yang juga disebut solid-state disk), gap I/O antara prosesor dan penyimpanan pendukung (auxiliary) dapat dipersempit, sehingga akan meningkatkan waktu respon dan keluarannya.

Daftar urutan peninjauan komponen-komponen rancangan sistem yang ditunjukkan pada Gambar 5.9 mewakili penemuan dari tim peninjauan pascaimplementasi SystemPro. Daftar tersebut berisi pertanyaan mengenai komponen rancangan sistem, tiap-tiap penilaian komponen, dan skor akhir untuk area peninjauan ini.

Meninjau Estimasi Waktu, Biaya, dan Keuntungan

Seperti yang baru saja kita lihat, peninjauan pascaimplementasi adalah suatu proses formal untuk menentukan sejauh mana sistem bekerja, bagaimana ia diterima, dan apakah diperlukan perbaikan atau perancangan ulang. Alasan penting lain untuk melakukan peninjauan pascaimplementasi adalah untuk membantu meningkatkan kemampuan mengestimasi waktu, biaya, dan keuntungan proyek. Dengan membandingkan waktu, biaya, dan keuntungan yang sebenarnya dengan yang diestimasikan pada awal proyek, kita dapat mempelajari dimana kesalahan-kesalahan yang kita lakukan dan bagaimana menghindari kesalahan-kesalahan ini pada proyek di masa mendatang.

Analisis Waktu

Tanggal target yang didasarkan pada PERT, Gantt, atau teknik-teknik yang sama digunakan untuk mengestimasi waktu penyelesaian berbagai tahap dan tugas. Juga, sejumlah person-month (atau beberapa unit waktu yang lain) yang diperlukan untuk

DAFTAR PERIKSA PENILAIAN TINJAUAN KOMPONEN RANCANGAN SISTEM

Komponen Rancangan Output:	9,0								
<ul style="list-style-type: none"> ■ Apakah output aman dari pemakai yang tak berbakat? ■ Apakah output akurat, tepat waktu, dan relevan? ■ Dapatkah para pemakai mendapatkan akses ke informasi ketika mereka memerlukannya? ■ Apakah output tersebut cocok dengan gaya kognitif para pemakai? Yakni, apakah para pemakai "big picture" diberi (disediakan) grafik, dan apakah para pemakai "detail" diberi angka-angka? ■ Apakah laporan-laporan diedit dan diidentifikasi secara tepat? 									
Komponen Rancangan Input:	9,5								
<ul style="list-style-type: none"> ■ Apakah tersedia kendali input yang cukup untuk memverifikasi dan memvalidasi entri data? ■ Apakah para pemakai mengetahui cara mengisi form-form? ■ Apakah form elektronik dan perangkat direct-entry digunakan sebagai pengganti dokumen sumber? 									
Komponen Rancangan Proses:	9,0								
<ul style="list-style-type: none"> ■ Apakah tersedia prosedur yang memadai dan apakah terdokumentasi? ■ Apakah perangkat lunaknya reliabel atau handal? ■ Apakah dokumentasi perangkat lunak merefleksikan secara akurat rancangan perangkat lunak dan fungsionalitasnya? ■ Apakah digunakan model-model statistik dan akunting yang tepat? ■ Apakah konversi input ke output dijalankan secara tepat waktu? 									
Komponen Rancangan Database:	7,5								
<ul style="list-style-type: none"> ■ Apakah databasenya dibackup dan aman dari risiko hilang? ■ Apakah file-file diberi label secara benar? ■ Apakah file-file ini cocok atau sesuai dengan program yang memproses mereka? ■ Apakah tersedia kamus data? ■ Apakah database dirancang untuk merespon query ad hoc secara cepat dan secara akurat? ■ Apakah database dirancang untuk memproses aliran job yang panjang (misalnya, daftar gaji dan billing) dengan cara yang paling efisien? 									
Komponen Rancangan Kendali:	9,0								
<ul style="list-style-type: none"> ■ Apakah perusahaan mempunyai kebijakan kendali yang kuat atau ketat? ■ Apakah rencana pemulihan dari bencana atau kerusakan telah ditetapkan? ■ Apakah rencana pemulihan dari kerusakan lulus dari pengujian kerusakan simulasi? ■ Apakah sistem tersebut berisi kendali akses yang ketat? ■ Apakah kendali akses ini memenuhi atau lulus dari pengujian pemenuhan tertentu? ■ Apakah sistem tersebut terlindungi dari perangkat lunak destruktif (perusak) seperti virus? 									
Komponen Rancangan Platform Teknologi:	7,0								
<ul style="list-style-type: none"> ■ Apakah konfigurasi komputer tersebut memenuhi keperluan atau persyaratan sistem dan pemrosesan pemakai? ■ Apakah teknologi tersebut dikonfigurasi secara optimal? ■ Apakah respon waktunya bisa diterima? 									
Skala penilaian:	<table style="width: 100px; border-collapse: collapse;"> <tr> <td style="width: 20px; text-align: center;">0</td> <td style="width: 20px; text-align: center;">5</td> <td style="width: 20px; text-align: center;">10</td> </tr> <tr> <td style="text-align: center;">Buruk</td> <td style="text-align: center;">Sedang</td> <td style="text-align: center;">Baik sekali</td> </tr> </table>	0	5	10	Buruk	Sedang	Baik sekali		
0	5	10							
Buruk	Sedang	Baik sekali							
	<table style="width: 100px; border-collapse: collapse;"> <tr> <td style="width: 50px;"></td> <td style="width: 50px; text-align: center;">Total</td> </tr> <tr> <td></td> <td style="text-align: center;"><u>51,0</u></td> </tr> <tr> <td></td> <td style="text-align: center;">Hasil akhir(51,0/6)</td> </tr> <tr> <td></td> <td style="text-align: center;"><u>8,5</u></td> </tr> </table>		Total		<u>51,0</u>		Hasil akhir(51,0/6)		<u>8,5</u>
	Total								
	<u>51,0</u>								
	Hasil akhir(51,0/6)								
	<u>8,5</u>								

Gambar 5.9 Hasil-hasil peninjauan komponen-komponen rancangan sistem di Vulcan Industries.

menghasilkan sejumlah jalur kode yang bisa dieksekusi (LOEC) atau point fungsi digunakan sebagai cara untuk mengestimasi waktu pengembangan perangkat lunak. Waktu sebenarnya yang diperlukan untuk menyelesaikan tugas-tugas dan proyek tersebut dibandingkan dengan waktu yang diestimasikan guna menentukan kesalahan estimasi. Jika kesalahannya besar, kita harus mencari cara baru agar kita bisa mendapatkan estimasi waktu yang akurat.

Analisis Biaya/Manfaat

Ada suatu keputusan penting yang harus dibuat setelah tahap rancangan sistem umum. Keputusan ini adalah mengenai pemilihan rancangan alternatif yang akan dirancang secara rinci dan diimplementasikan. Teknik kuantitatif pokok yang digunakan untuk membuat keputusan ini adalah analisis biaya. Biaya dan manfaat yang ditentukan saat itu dibandingkan dengan biaya dan manfaat yang diestimasikan selama tahap awal SDLC. Kalkulasi saat itu harus cukup mendekati estimasi awal.

Dalam banyak organisasi, departemen pemakai dibebani biaya untuk pengembangan dan biaya berjalan sistem baru. Oleh karena itu, para pemakai seperti ini sangat sensitif dengan keakuratan estimasi biaya. Umumnya, jika biaya sebenarnya berada dalam (tidak melenceng lebih dari) 5 sampai 10 persen dari biaya yang diestimasikan, para pemakai akan puas.

Daftar peninjauan estimasi yang ditampilkan pada Gambar 5.10 dengan pertanyaan-pertanyaan pendukung, merepresentasikan tingkat keakuratan estimasi. Daftar tersebut berisi setiap estimasi yang dibuat pada awal proyek sistem dan skor akhir yang menyatakan seberapa dekatnya estimasi ini dengan yang sebenarnya terjadi. Informasi semacam ini sangat bermanfaat untuk membuat estimasi di masa mendatang.

Meninjau Tingkat Dukungan yang Diberikan

Pada umumnya, keberhasilan suatu proyek sistem sangat berkaitan dengan tingkat dukungan yang diterima para profesional sistem selama pengembangannya.

Sumber Daya yang Tersedia

Pada permulaan proyek sistem, biasanya manajemen menjanjikan atau bertanggung jawab (berkomitmen) atas sumber-sumber daya yang digunakan untuk mengembangkan dan mengimplementasikan sistem baru. Jika rancangan awalnya perlu dimodifikasi karena kurangnya sumber daya, maka kemungkinan besar keperluan

DAFTAR PERIKSA PENILAIAN TINJAUAN ESTIMASI		
Waktu:		6,5
■ Apakah digunakan teknik-teknik perencanaan dan kendali proyek PERT, Gantt, atau yang lain?		
■ Apakah metriks, seperti jalur kode yang bisa dieksekusi (LOEC) atau point fungsi, digunakan untuk mengestimasi waktu dan biaya proyek dan untuk memonitor produktivitas?		
■ Apakah kesalahan estimasi bisa diterima oleh manajemen?		
Analisis Biaya/Manfaat:		8,5
■ Apakah biaya yang sebenarnya sebanding dengan biaya yang diestimasikan?		
■ Apakah para pemakai puas dengan biaya yang dialokasikan ke departemen mereka?		
■ Apakah para pemakai menerima manfaat yang dijanjikan pada awalnya?		
Total		15,0
Hasil akhir		<u>7,5</u>
Skala Tingkatan		
0 5 10		
Buruk Sedang Baik sekali		

Gambar 5.10 Hasil-hasil peninjauan estimasi di Vulcan Industries.

pemakai awal juga harus dimodifikasi. Kecukupan sumber harus dipastikan dan sepenuhnya diperlihatkan kepada semua pihak yang terlibat.

Dukungan dari Manajemen Puncak

Selain komitmen dari manajemen yang menyatakan tercukupinya sumber untuk proyek sistem, manajemen puncak juga harus memberikan dukungan moral dan politis. Jika manajemen puncak terlibat secara aktif dalam suatu proyek dan mendukungnya, maka masalah yang akan terjadi akan lebih sedikit; para peserta proyek akan lebih kooperatif, dan birokrasinya akan lebih mudah.

Pelatihan

Berbagai teknik pelatihan diterapkan untuk melatih semua pemakai dengan di-dasarkan pada kebutuhan tertentu. Semua pemakai harus menerima pelatihan yang

memadai, dan dengan telah menjalani pelatihan ini, para pemakai diharapkan bisa memahami dan menggunakan sistem tersebut secara mudah.

Daftar tingkat-dukungan yang ditunjukkan pada Gambar 5.11 mengungkap sejauh mana proyek sistem tersebut dijalankan sepanjang atau selama SDLC. Daftar tersebut mencakup setiap kategori tingkat-dukungan, penilaianya, dan skor terakhir untuk area peninjauan pascaimplementasi. Pertanyaan-pertanyaan yang dikemukakan oleh tim SystemPro untuk menghasilkan penilaian yang wajar disertakan di bawah setiap kategori.

TINJAUAN SASARAN BELAJAR UNTUK BAB INI

Tujuan pokok bab ini adalah untuk memungkinkan setiap siswa mencapai pemahaman enam sasaran belajar yang penting. Kita sekarang akan meringkas tanggapan dari sasaran belajar ini.

Sasaran belajar 1:

Menyiapkan rencana implementasi sistem dengan menggunakan PERT.

Jika sistem yang akan diimplementasikan adalah sistem berbasis-global, maka diperlukan teknik PERT atau teknik yang sama untuk menyiapkan rencana implementasi. Para profesional sistem harus melakukan lima langkah berikut ini untuk membentuk rencana implementasi berbasis-PERT.

1. Mengidentifikasi semua tugas implementasi yang harus dijalankan.
2. Menentukan urutan tugas.
3. Mengestimasi waktu yang diperlukan untuk menjalankan setiap tugas.
4. Menyiapkan jaringan tugas yang terskala-waktu.
5. Menentukan lintasan penting dalam jaringan tersebut.

Sasaran belajar 2:

Mendeskripsikan bagaimana menyiapkan tempat untuk platform teknologi.

DAFTAR PERIKSA PENINJAUAN TINGKAT-DUKUNGAN		
Sumber-sumber yang Tersedia:		9,0
<ul style="list-style-type: none"> ■ Apakah dana, perangkat, pasokan, dan profesional sistem yang terampil tersedia di sepanjang siklus hidup pengembangan sistem? ■ Apakah proyek sistem mengalami penundaan atau keterlambatan akibat dari tidak cukupnya sumber-sumber seperti itu? ■ Apakah skala proyek sistem harus diperkecil atau dirancang ulang akibat dari tidak cukupnya sumber-sumber tersebut? 		
Dukungan Manajemen Puncak:		7,0
<ul style="list-style-type: none"> ■ Apakah para manajer tingkat-senior berpartisipasi dalam perencanaan sistem? ■ Apakah proyek sistem dan tim proyek menerima dukungan yang besar dari manajemen puncak sepanjang SDLC? 		
Pelatihan:		8,0
<ul style="list-style-type: none"> ■ Apakah para pemakai kompeten (berkepentingan) dalam berinteraksi dengan dan bekerja dengan sistem tersebut? ■ Apakah digunakan kombinasi dari program pelatihan in-house dan program pelatihan yang disediakan oleh vendor? ■ Apakah digunakan teknik pelatihan baru seperti teleconferencing dan perangkat lunak pelatihan interaktif? ■ Apakah digunakan pelatihan dengan instruktur (instructor-based) untuk mengerjakan tugas-tugas kompleks? ■ Apakah dokumentasi dan prosedur pemakai dimanfaatkan selama sesion pelatihan untuk memastikan ketepatan dan kejelasannya? 		
	Total	24,0
	Hasil akhir	8,0
Skala Tingkatan		
0 5 10		
Buruk Baik Baik sekali		

Gambar 5.11 Hasil-hasil peninjauan tingkat-dukungan di Vulcan Industries.

Semua platform teknologi, apakah itu untuk mikrokomputer ataupun untuk arsitektur komputer berbasis-kooperatif, harus ditempatkan dalam suatu tempat yang terkendali dengan baik dan lingkungannya aman. Tempat ini harus dipersiapkan dan siap untuk menerima platform teknologi ketika ia tiba.

Sasaran belajar 3:

Menjabarkan cara melatih orang-orang untuk bekerja dengan sistem baru.

Tugas implementasi pokok adalah untuk mendapatkan tingkat kemahiran yang sesuai yang diperlukan oleh orang-orang untuk bekerja dengan sistem tersebut. Umumnya, orang-orang harus dilatih untuk mencapai kemahiran atau keterampilan ini dan untuk menyesuaikan mereka dengan sistem baru. Program pelatihan meliputi:

- ◆ Pelatihan in-house
- ◆ Pelatihan yang disediakan vendor
- ◆ Jasa pelatihan luar

Teknik-teknik dan alat bantu pelatihan terdiri atas:

- ◆ Teleconferencing
- ◆ Perangkat lunak pelatihan interaktif
- ◆ Pelatihan dengan instruktur
- ◆ Pelatihan magang
- ◆ Manual prosedur
- ◆ Buku teks

Sasaran belajar 4:

Menjabarkan dokumentasi yang harus disiapkan.

Untuk menggunakan, mengoperasikan, dan memelihara sistem, dokumentasi harus dipersiapkan. Prosedur pemakai, yang meliputi materi tertulis, video, dan pesan/tampilan layar, membantu pemakai bekerja dengan sistem tersebut. Dokumentasi sistem membantu dalam konversi dan peninjauan pascaimplementasi. Dokumentasi program diperlukan untuk pemeliharaan perangkat lunak. Dokumentasi operasi memberitahu operator cara bekerja dengan sistem tersebut.

Sasaran belajar 5:

Menjelaskan empat metode konversi sistem.

Konversi sistem dapat dilakukan dengan menggunakan salah satu dari empat metode berikut:

- ◆ Langsung
- ◆ Paralel
- ◆ Phase-in
- ◆ Pilot

Konversi langsung adalah penginstalasian sistem baru secara serentak dan pemutusan sistem lama. Konversi paralel memungkinkan sistem lama dan yang baru beroperasi bersama selama beberapa periode. Konversi phase-in adalah penginstalasian sedikit demi sedikit sistem baru dan penghapusan sistem lama. Konversi pilot menginstal replika atau versi yang bekerja dari sistem keseluruhan dalam beberapa segmen organisasi. Apabila sistem pilot nampak berhasil, semua segmen akan menerima sistem baru total atau keseluruhan.

Sasaran belajar 6:

Menjelaskan tinjauan pasca implementasi.

Tinjauan pascaimplementasi dilakukan beberapa bulan setelah sistem baru beroperasi. Ini untuk menentukan sejauh mana keberhasilannya. Para peninjau pascaimplementasi menentukan sejauh mana sistem tersebut hidup atau bekerja sesuai yang diharapkan. Proses peninjauan pascaimplementasi menyelidiki empat area:

- ◆ Faktor sistem
- ◆ Komponen rancangan sistem
- ◆ Keakuratan estimasi
- ◆ Tingkat dukungan

Apa yang sebelumnya diharapkan atau direncanakan dibandingkan dengan apa yang terjadi sebenarnya. Varian atau perbedaan antara yang direncanakan dan yang sebenarnya diungkap atau diperlihatkan dalam laporan peninjauan pascaimplementasi.

DAFTAR PERIKSA IMPLEMENTASI SISTEM

Berikut ini adalah daftar urutan cara melakukan implementasi sistem. Tujuannya adalah untuk meringkas atau merekapitulasi bab ini dan mengingatkan anda tentang cara pelaksanaan tugas-tugas implementasi sistem.

1. Siapkan rencana implementasi sistem dengan menggunakan PERT atau teknik yang sama. Organisasilah personel implementasi sistem.
2. Siapkan tempat sistem baru sebelumnya.
3. Latihlah personel dengan menggunakan salah satu atau kombinasi dari program pelatihan in-house, yang disediakan vendor, atau jasa pelatihan luar.
4. Terapkan salah satu atau kombinasi dari teleconferencing, perangkat lunak pelatihan interaktif, pelatihan dengan instruktur, pelatihan magang, manual prosedur, dan buku teks sebagai teknik dan alat bantu pelatihan.
5. Kembangkan atau buatlah prosedur pemakai, dokumentasi sistem, dokumentasi perangkat lunak, dan dokumentasi operasi.
6. Lakukan konversi sistem dengan menggunakan salah satu atau kombinasi dari metode berikut ini: langsung, paralel, phase-in, dan pilot.
7. Pekerjakan tim peninjau pascaimplementasi independen untuk menilai efisiensi dan efektivitas sistem baru setelah ia beroperasi selama dua bulan sampai enam bulan. Tinjaulah dan evaluasilah faktor-faktor sistem, komponen-komponen rancangan sistem, keakuratan estimasi, dan tingkat dukungan. Laporkan hasil-hasilnya kepada komite atau panitia pengarah dan pihak-pihak lain yang berkepentingan. Gunakan informasi peninjauan pascaimplementasi ini untuk mengambil tindakan korektif jika diperlukan dan untuk membantu memberi pedoman pengembangan sistem di masa mendatang.

SOAL TINJAUAN

- 5.1 Sebutkan dua bagian utama dari suatu Laporan Implementasi Sistem.

- 5.2 Jelaskan penggunaan PERT atau diagram Gantt dalam proses implementasi.
- 5.3 Terangkan peranan persiapan tempat. Deskripsikan persiapan tempat yang diperlukan untuk arsitektur komputer baru.
- 5.4 Apa yang dimaksud pengujian burn-in? Mengapa ia dilakukan?
- 5.5 Sebutkan tiga kelompok orang yang harus diberi pelatihan untuk sistem baru.
- 5.6 Sebutkan dan jelaskan secara singkat program pelatihan.
- 5.7 Sebutkan dan jelaskan secara singkat teknik dan alat bantu pelatihan.
- 5.8 Sebutkan jenis-jenis dan tujuan (kegunaan) dokumentasi.
- 5.9 Sebutkan empat metode konversi sistem. Jelaskan keunggulan dan kelebihan setiap metode itu.
- 5.10 Terangkan dua metode dasar yang digunakan untuk konversi file.
- 5.11 Sebutkan dan jabarkan secara singkat jenis-jenis file yang dikonversi.
- 5.12 Jelaskan bagaimana konversi file gradual (sedikit demi sedikit) bekerja.
- 5.13 Definisikan peninjauan pascaimplementasi. Mengapa ia dilakukan?
- 5.14 Sebutkan dan terangkan secara singkat area-area peninjauan pascaimplementasi.
- 5.15 Pada hakikatnya, apa yang diungkap oleh laporan peninjauan pascaimplementasi? Apa yang bisa dipelajari dari laporan ini?

SOAL SPESIFIK BABINI

Soal-soal ini memerlukan jawaban pasti yang didasarkan secara langsung pada konsep dan teknik yang dikemukakan dalam bab ini.

- 5.16 Anda baru saja menyelesaikan rancangan sistem rinci untuk Wilmington Sporting Goods. Anda telah menentukan bahwa tugas-tugas berikut ini harus dilakukan untuk menjalankan implementasi sistem:

Tugas	Deskripsi
1,2	Memesan teknologi
1,3	Meninjau atau memeriksa spesifikasi
1,4	Menyiapkan tempat

2,5	Menginstal teknologi
3,6	Menulis program
4,9	Melatih personel
5,7	Menguji teknologi
6,8	Menguji program
7,9	Dummy
8,9	Menguji input, output, database, dan kendali
9,10	Mengkonversi sistem
10,11	Melakukan peninjauan pascaimplementasi

Ditanyakan: Sketlah diagram PERT dari tugas-tugas yang berkelanjutan tersebut. Menyiapkan tempat, melatih personel, mengkonversi sistem, dan menjalankan peninjauan pascaimplementasi adalah tugas-tugas yang berada pada lintasan penting.

5.17 Untuk setiap situasi berikut, pilihlah program pelatihan yang paling tepat:

1. Pelatihan akan dilakukan dalam lingkungan petatar.
2. Perusahaan tidak dapat menyediakan atau memberikan pelatihan yang diperlukan, namun ia dekat dengan universitas yang bisa memberikan program pelatihan yang berkelanjutan dalam bidang sistem.
3. Para petatar akan mempelajari kekhasan arsitektur komputer baru dari vendor.

5.18 Dalam situasi berikut ini, pilihlah teknik dan alat bantu pelatihan yang paling tepat:

1. Petatar perlu melakukan dialog audio dengan teknik pelatihan.
2. Para petatar (yang dilatih) secara geografis terpisah dan tidak dapat meninggalkan pekerjaan mereka, namun semua harus berinteraksi dengan yang lainnya dan perlu dilatih dalam waktu yang bersamaan.
3. Setiap petatar memerlukan pelatihan tersendiri.

5.19 Dalam situasi berikut, pilihlah metode konversi yang paling tepat:

1. Sistem account receivable berbasis-komputer akan dikonversi dari manual ke operasi berbasis-komputer. Tim proyek sistem gagal

melakukan pengujian, dan akibatnya, para pemakai tidak yakin apakah sistem baru tersebut akan menghasilkan hasil-hasil yang akurat.

2. Sistem lotere akan dikonversi dalam suatu state dimana kita belum ada sebelumnya.
3. Sistem deposito permintaan bank baru akhirnya akan melayani 60 cabang.
4. Sistem informasi eksekutif baru telah diuji secara seksama. EIS lama mempunyai manfaat yang minim.
5. Salah satu pabrik Hercules Manufacturing akan berfungsi sebagai tempat untuk sistem pelaporan kinerja just-in-time baru.
6. Kultur korporasi dimana sistem baru sedang diimplementasikan lamban untuk mengubah caranya. Sistem baru dapat disegmentasi.

SOAL UMUM

Soal-soal ini memerlukan jawaban berupa pendekatan atau cara yang layak, bukan hanya solusi yang presisi. Walaupun soal-soal ini didasarkan pada materi bab ini, kita memerlukan bacaan ekstra dan perlu kreativitas agar bisa mengembangkan atau membuat solusi yang layak atau bekerja.

- 5.20 Sistem account receivable manual pada Calico Pet Supply sedang dikonversi ke sistem berbasis-komputer. Sistem account receivable saat itu mempunyai karakteristik-karakteristik berikut ini:

1. Setiap pelanggan dengan (yang mempunyai) neraca account receivable non-nol mempunyai berkas yang berisi kopi dari semua faktur yang belum terbayar dan nota kredit yang diterbitkan.
2. Ketika pembayaran dan remitan yang menyertai diterima dari pelanggan, remitan tersebut dicocokkan dengan faktur yang belum terbayar, dan kedua dokumen tersebut ditempatkan dalam file tertutup saat itu, yang selanjutnya akan dihapus setiap enam bulan.
3. File tertutup permanen, yang terkomposisi dari dokumen-dokumen file tertutup yang dihapus, dipelihara dalam periode selama tujuh tahun.
4. Pada akhir bulan, neraca setiap account pelanggan diklasifikasikan menurut umur penungguan neraca tersebut. Ini dilakukan oleh klerk yang menghitung dan menanggali jumlah berkas.

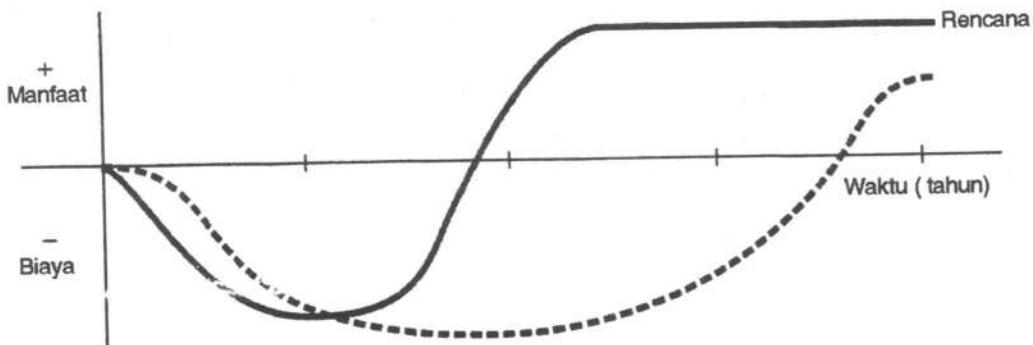
Frekuensi akses ke ketiga file tersebut sangat bervariasi. File berkas diakses secara sering. File tertutup saat itu diakses secara berkala, biasanya ketika diminta oleh manajer kredit atau pelanggan. File tertutup permanen diakses secara jarang. Sistem baru yang akan diimplementasikan akan berisi tiga file:

1. File master pelanggan, yang berisi informasi mengenai setiap pelanggan.
2. File item terbuka, yang bersesuaian dengan file "berkas" dari sistem manual.
3. File item tertutup, yang menyusun kombinasi file tertutup permanen dan file tertutup saat itu

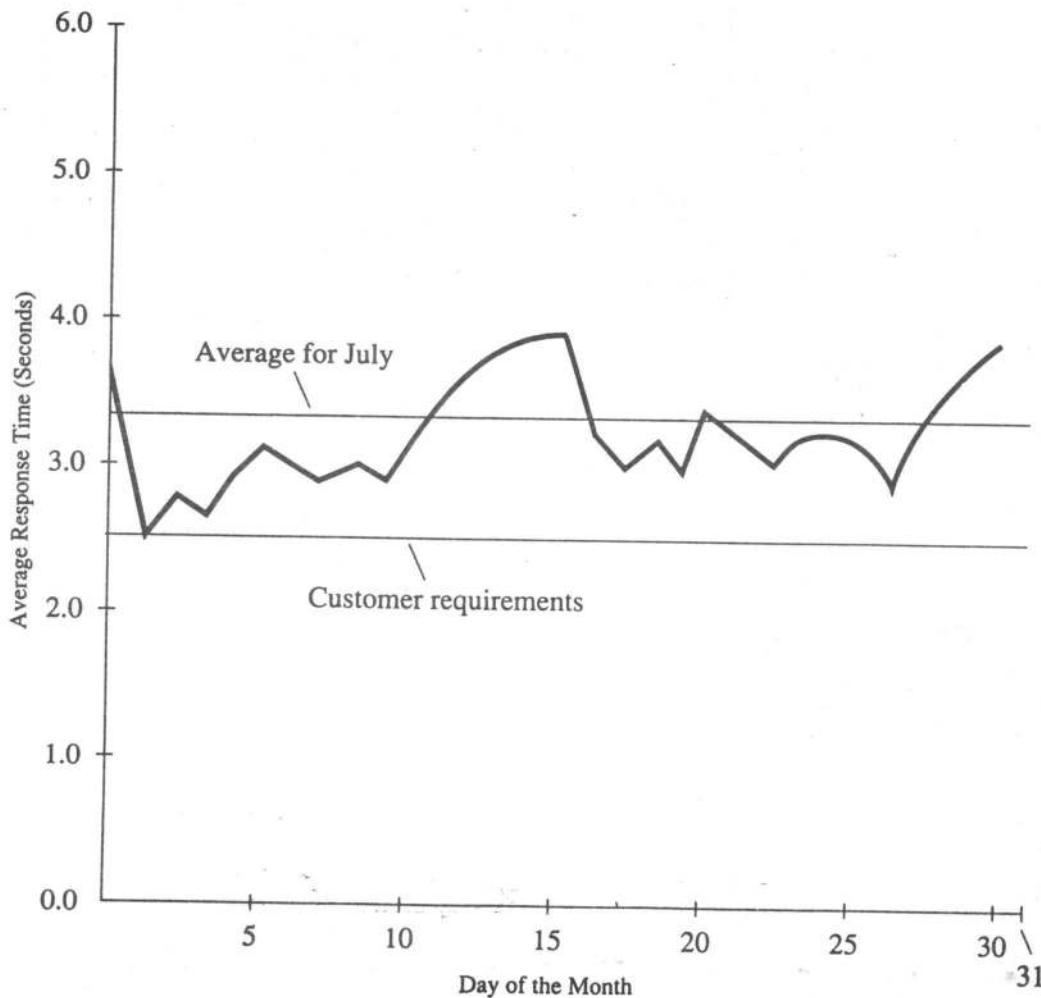
Asumsikan bahwa anda diberi tanggung jawab untuk mengkonversi sistem lama tersebut ke sistem baru. Jawablah pertanyaan-pertanyaan berikut ini:

1. Metode konversi apa yang akan anda rekomendasikan untuk kasus ini? Jelaskan dan buktikan jawaban anda.
2. Prosedur klerikal khusus apa yang perlu anda tetapkan guna memvalidasi kebenaran operasi sistem baru dibandingkan sistem lama?
3. Bagaimana anda dapat menangani konversi file? Jelaskan jawaban anda.

- 5.21 Anda telah mem-plot gambaran biaya dan manfaat yang sebenarnya, yang dibandingkan dengan yang diestimasikan, pada awal proyek sistem, sebagai berikut:



- Ditanyakan:** Apa yang dikatakan grafik ini dalam kaitannya dengan pemenuhan biaya dan manfaat yang diharapkan?
- 5.22 Waktu respon terminal adalah waktu yang harus ditunggu pemakai untuk memulai suatu transaksi setelah pemakai ini menyelesaikan transaksi sebelumnya. Penetapan waktu respon yang baik akan tergantung pada aplikasi dan pemakainya. Biasanya, aplikasi entri data paling banyak diminati, karena diperlukan pemikiran yang sedikit dari pihak pemakai. Oleh karena itu, segala penundaan antara waktu penyelesaian satu transaksi dan permulaan transaksi berikutnya dianggap sebagai gang-



guan yang berarti dalam arus kerja. Para operator entri data mahir akan sangat mengeluh bila waktu respon terminal lebih dari 2,5 detik. Anda tengah menggunakan monitor hardware sebagai bagian dari peninjauan pascaimplementasi anda. Grafik berikut ini menampilkan waktu respon online rata-rata selama satu bulan operasi pemonitoran.

Ditanyakan: Dengan berdasarkan pada informasi yang ditampilkan dalam grafik di atas, buatlah tanggapan yang akan disertakan (dimasukkan) dalam laporan peninjauan pascaimplementasi. Jika sistem yang sedang anda tinjau tersebut dirancang untuk mendukung, terutama, para operator entri data, apakah sistem itu memadai? Perubahan-perubahan layak apa yang mungkin dilakukan agar membuat sistem itu lebih memadai? Buatlah rekomendasi spesifik. Agar lebih mudah memformulasikan tanggapan anda, lihatlah kembali bab-bab yang membahas rancangan rinci jaringan telekomunikasi dan sistem komputer.

- 5.23 Persentase waktu keberadaan-sistem terkait erat dengan waktu respon terminal. Namun demikian, persentasi yang tidak baik dari waktu keberadaan dan waktu respon terminal, bisa menjadi gejala berbagai masalah. Persentase keberadaan sistem dianggap sebagai ukuran kinerja sistem yang penting.

Persentase keberadaan sistem dikalkulasi dengan persamaan berikut:

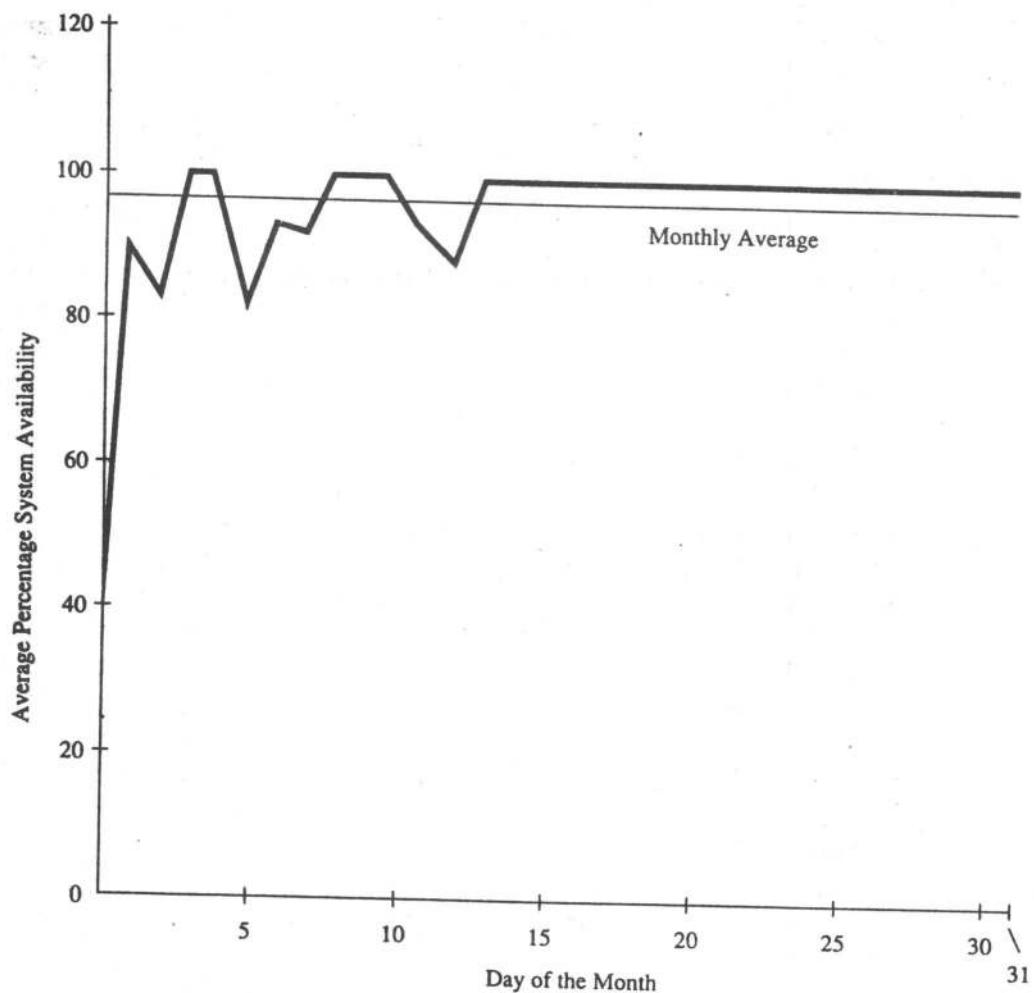
$$\text{Persentase Keberadaan} = \frac{(\text{keberadaan terjadwal total} - \text{downtime}) \times 100}{\text{keberadaan terjadwal total}}$$

Keberadaan terjadwal total (total scheduled availability) adalah jumlah jam total penjadwalan sistem untuk beroperasi dalam periode waktu yang ditentukan. Periode waktunya mungkin bisa berupa hari, minggu, bulan, atau tahun. Downtime adalah jumlah jam selama sistem tersebut tidak bisa digunakan dalam periode waktu yang sama.

Anda telah memonitor suatu sistem selama sebulan, yang kemudian diikuti dengan pengkonversian sistem itu. Hasil monitoring ini ditampilkan dalam grafik pada halaman berikut.

Ditanyakan: Penilaian apa yang bisa anda buat terhadap keberadaan sistem tersebut menurut grafik di atas? Buatlah suatu asumsi yang anda rasa tepat dan jelaskan alasan atau alasan-alasan di balik fluktuasi dalam separuh bulan yang pertama. Apakah anda berencana terus memonitor sistem tersebut bulan berikutnya? Mengapa tidak?

- 5.24 Sistem order-entry (entri-pesanan) baru telah beroperasi selama empat bulan setelah dilakukan konversi langsung. Selama waktu ini, pesanan-pesanan telah dilaporkan terlambat 35 persen dari waktunya, karena



terjadi malfungsi sistem dan ketidakpedulian pemakai. Rerun adalah hal yang biasa dilakukan, dan perangkat lunak order-entry telah mengalami 14 penghentian abnormal. Pada tiga penghentian abnormal pertama, operasi-operasi memerlukan waktu dua hari untuk menjalankan kembali sistem tersebut.

Ditanyakan: Buatlah kritik terhadap implementasi sistem ini. Apa yang seharusnya dilakukan untuk mencegah masalah-masalah ini terjadi? Apakah anda yakin bahwa konversi langsung akan tepat? Metode konversi mana yang anda sarankan? Prosedur pengujian dan pelatihan apa yang tepat? Buktikan semua jawaban anda tersebut.

BACAAN YANG DISARANKAN

- Auerbach Information Management Series. RennsauKen, N.J.: Auerbach Publishers, 1980.
- Burch, John, and Gary Grudnitski. *Information Systems: Theory and Practice*, 5th ed. New York: John Wiley, 1989.
- Carr, Houston H. *Managing End User Computing*, Englewood Cliffs, N.J.: Prentice-Hall, 1988.
- Chandler, John S., and H. Peter Holzer, *Management Information Systems*, New York: Basil Blackwell, 1988.
- Cohen, Isabelle. "Computer Training Programs: What's Available and How to Select Them." *Computers in Accounting*, August 1988.
- Eckols, Steve. *How to Design and Develop Business Systems*, Fresno, Calif: Mike Murach & Associates, 1983.
- Emery, James C. *Management Information Systems: The Critical Strategic Resource*. New York: Oxford University Press, 1987.
- Inmon, W. H. *Information Engineering for the Practitioner*, Englewood Cliffs, N.J.: Yourdon Press, A Prentice-Hall Company, 1988.
- Leeson, Majorie. *Systems Analysis and Design*, 2nd ed. Chicago: Science Research Associates, 1985.
- Long, Larry E. *Design and Strategy for Information Systems: MIS Long-Range Planning*. Englewood Cliffs, N.J.: Pientice-Hall, 1982.
- Martin, Merle P. "The Day-One Systems Changeover Tactic." *Journal of Systems Management*, October 1989.
- Methlie, Leif B. *Information Systems Design: Concepts and Methods*. New York: Columbia University Press, 1988.
- Walsh, Robert. "The Postimplementation Audit." *EDPACS*, August 1989.

KASUS JOCS: Pengimplementasian Sistem

Status meeting (rapat pemantauan) untuk proyek pengembangan perangkat lunak JOCS dijadwalkan pada Rabu ketiga setiap bulan. Selama status meeting, Jake Jacoby, manajer proyek JOCS, dan anggota tim SWAT lainnya bertermu untuk mengevaluasi kemajuan proyek pengembangan perangkat lunak JOCS. Pada meeting status terakhir, Jake mencatat bahwa sebagian besar proyek berjalan (mengalami kemajuan) sesuai yang dijadwalkan pada awalnya. Namun demikian, ada beberapa penyimpangan (pemoloran) waktu dalam perancangan dan pengembangan sistem penangkapan data online. Seperti dibahas dalam Bab 2 dan 4, Carla Mill mengerjakan program aplikasi yang digunakan untuk menangkap data online dari sistem CIM (computer-integrated manufacturing), sedangkan Tom Pearson bertanggung jawab untuk pengadaan interface antara sistem CIM dan jaringan area lokal JOCS. Selama status meeting terakhir,

Carla ingin mengetahui kapan interface antara sistem CIM dan JOCS tersebut diselesaikan, sehingga ia bisa memulai pengujian program. Tom baru saja memulai mengerjakan interface tersebut. Ia berkata bahwa ia akan kembali lagi kepadanya melalui surat elektronik dengan tanggal pengantar interface perusahaan.

Tim SWAT bertemu sejorang mungkin agar setiap anggota tim bisa menyelesaikan pekerjaannya dengan sedikit mungkin gangguan/interupsi. Sebagian besar waktu digunakan oleh para anggota tim untuk berkomunikasi melalui surat elektronik (bukannya melalui telepon atau dengan meeting) ketika ada sesuatu yang penting yang harus dibahas. Status meeting berikutnya akan diadakan dalam beberapa hari lagi. Jake sedang bersiap untuk membahas jadwal implementasi sistem pada meeting itu. Carla masih mengurus tanggal pengantar (tanggal jadi) untuk interface CIM/JOCS. Ia menginginkan beberapa jawaban sebelum meeting berikutnya. Berikut adalah beberapa surat elektronik yang dikirimkan untuk membahas persoalan-persoalan ini:

Date: September 12, 1991 10:13 A.M.

From: Carla Mills

To: Tom Pearson

Copies:

Attach:

Subject: Online Capture Delay

Apa yang sedang terjadi dengan interface antara sistem CIM dan JOCS? Saya ingin menguji bagian pertama dari program penangkapan data online saya minggu ini, namun saya tidak dapat menguji program saya sampai interface tersebut rampung. Penundaan apa yang sedang terjadi? Menurut anda, kapan interface tersebut akan siap?

Date: September 12, 1991 09:30 A.M.

From: Tom Pearson

To: Jake Jacoby

Copies: Carla Mills

Attach:

Subject: Reply to: Online Capture Delay

Saya telah gagal menginstal interface antara sistem CIM dan JOCS karena masalah pengantaran dengan vendor kami. Koneksi hardware antara dua sistem tersebut dikarenakan pengantaran lebih dari sebulan yang lalu. Vendornya, Connect-Plus, berjanji kepada saya akan memberikan bagian hardware yaitu bridge di kemudian hari, dan sekarang ia berhenti membela telepon saya.

Vendor bridge perangkat lunak, Interface International. Itu sebenarnya membuat masalah atas tanggal pengantarnya yang terlambat. Keinginannya saya berbicara dengan manajer produk teknik. Ia mengatakan bahwa produknya masih dalam pengujian. Ia sangat bingung dengan apa yang telah terjadi dan berkata bahwa salesman-nya yang menjual produk itu mengetahui bahwa penginstalasianya adalah pada akhir Oktober, bukannya pada awal September. ia menjamin akan mengirimkan produk yang telah teruji sepenuhnya pada akhir Oktober, namun ia mengatakan bahwa kelompoknya sedang melakukan pengujian beta.

Saya kirimkan juga kopip dari catatan hasil pembicaraan saya dengan kedua vendor tersebut. Saya tak ingin mengatakan hal ini, namun saya perlu bantuan untuk menekan orang-orang tersebut mengantarkan produk mereka.

Carla siap untuk memulai pengujian terhadap program penangkapan data online-nya, namun ia tetap tidak bisa berbuat apa-apa tanpa adanya interface CIM/JOCS.

Date: September 12, 1991 01:02 P.M.

From: Jake Jacoby

To: Tom Pearson

Copies:

Attach: Hardware Vendor Listing

Subject: Vendor Can Be a Pain

Akhirnya saya menghubungi seseorang yang duduk dalam manajemen pada Connect Plus. Orang-orang dalam manajemen itu sulit untuk diajak berpikir dan diajak bicara, namun inilah yang saya temukan. Produk yang kita beli telah tersedia dan siap dikirimkan. Namun demikian, produk yang kita inginkan tersebut masih dalam pengembangan dan tidak akan siap dalam waktu enam bulan. Salesman yang kita hubungi tidak mengerti secara pasti apa yang kita inginkan dan menjual kepada kita produk yang salah. Ia jelas tidak mau mengakui kesalahannya dan ia selama ini menanggung bahwa produk yang kita inginkan akan siap saat kita siap. Namun toh demikian, kesudahannya kita membatalkan pesanan kita dengan Connect-Plus, dan kita harus mencari vendor baru untuk koneksi hardware kita segera mungkin.

Saya telah melampirkan daftar vendor hardware untuk anda hubungi. Inilah proyek prioritas nomor satu sekarang ini, Tom. Carilah bridge baru secepat mungkin. Segera setelah anda mendapatkan tanggal pengantaran, kirimi saya pesan sehingga saya dapat mengupdate jadwal kita.

Omong-omong, saya juga berbicara dengan manajer produk teknik pada Interface International. ia benar-benar ingin tahu apa yang sedang ia kerjakan, dan saya ingin membeli produk mereka. Jika negoisasi kita berhasil, kita akan ditempati untuk pengujian beta untuk perangkat lunak bridge mereka. Saya akan kabari anda besok pagi

apakah segala sesuatunya bekerja dengan Interface International, atau apakah kita perlu mencari vendor perangkat lunak baru.

Date: September 12, 1991 01:08 P.M.

From: Jake Jacoby

To: Carla Mills

Copies: Tom Pearson

Attach:

Subject: Online Capture Delay

Kita harus menunda pengujian terhadap program penangkapan data online anda sampai kita menyelesaikan interface antara sistem CIM dan LAN JOCS. Tom sedang menyelesaikan masalah interface dan akan terus mengabari anda mengenai jadwal waktunya. Ia pasti telah bisa memberi anda jadwal pengantaran baru pada akhir minggu ini.

Karena anda harus berhenti mengerjakan program penangkapan tersebut, silakan hubungi Christine dan bantulah ia mengerjakan program pelaporan varian.

Date: September 13, 1991 02:25 P.M.

From: Jake Jacoby

To: Mary Stockland

Copies:

Attach:

Subject: Telecommunications Support

Kita telah membahas implikasi dari jaringan telekomunikasi kita yang sedang tumbuh berulang kali, namun sekaranglah waktunya menyelesaikan area masalah ini secara serius. Saya pikir kita perlu orang tambahan yang berkualitas untuk mendukung jaringan kita. Tom sedang mengerjakan tugas yang begitu penting, namun kita harus mempunyai cadangan apabila ia tidak hadir pada waktu yang telah ditentukan.

Tom dijadwalkan untuk mengikuti kelas pelatihan dua-minggu yang dimulai pada 23 September. Saya mengirimkannya ke kelas yang disponsori vendor. Vendor tersebut menawarkan harga yang kompetitif dan akan bisa menjelaskan semua rincian teknis penting dari sistemnya selama berlangsung kelas itu. Tom ingin ia jauh kantornya dan dari tanggung jawab pengujinya yang kadang-kadang menjemuhan, agar ia dapat berkonsentrasi dengan materi kursus. Saya meminta Tom untuk menghubungi kita selama kelas tersebut; kemudian ia akan bisa memanggil anggota kelas lain untuk dimintai bantuan selama implementasi kita. Saya sebenarnya ingin mengirimkan dua orang ke kelas (kursus) ini. Tom dan orang telekomunikasi lain akan bisa saling

membantu selama kursus tersebut dan menjalankan instalasi sistem tersebut bersama. Kita akan mendapatkan diskon setengah harga untuk orang tambahan jika mendaftarkannya minggu depan. Bagaimana menurut anda?

Date: September 13, 1991 02:45 P.M.

From: Mary Stockland

To: Jake Jacoby

Copies:

Attach:

Subject: Telecommunications Support

Jake, anda pasti telah bisa membaca pikiran saya. Sejak diskusi terakhir kita, saya telah bernegosiasi dengan VP (vice president) perekayasaan/mekanik untuk membagi posisi dukungan telekomunikasi dengan departemen kita. Kita akan menggunakan bersama (membagi) David Martinez untuk bekerja di bagian perekayasaan dan MIS untuk instalasi dan dukungan telekomunikasi. David telah memiliki pengalaman perekayasaan elektris lima tahun. Ia telah mengenal dengan baik sistem Ethernet yang digunakan dalam departemen perekayasaan. Selain itu, ia bekerja dengan sistem UNIX selagi di sekolah dan dia adalah yang pertama kali mengkonfigurasi minikomputer berbasis-UNIX yang ditempatkan didepartemen perekayasaan. Ia benar-benar telah mahir bekerja dengan sistem komputer, dan ia akan senang dengan posisinya yang baru ini.

Silakan daftarkan David segera untuk kursus jaringan area lokal tersebut. Saya akan menghubunginya dan memberitahukan kepadanya mengenai kursus ini dan juga status meeting (rapat pemantauan) kita minggu yang akan datang.

Date: September 13, 1991 03:24 P.M.

From: Jake Jacoby

To: Mary Stockland

Copies:

Attach:

Subject: JOCS Implementation Ideas

David Martinez sekarang telah didaftarkan pada pelatihan jaringan area lokal yang dimulai 23 September. Saya telah minta sekretaris saya untuk mengurus penerbangan dan penginapannya.

Saya sedang menyusun jadwal implementasi JOCS kita. Saya ingin melontarkan beberapa gagasan kepada anda.

Pelatihan pemakai: Kita harus melatih para pemakai dalam dua area yang berbeda: sintaks dan penggunaan DBMS dan penggunaan rinci JOCS. Kita akan melatih para

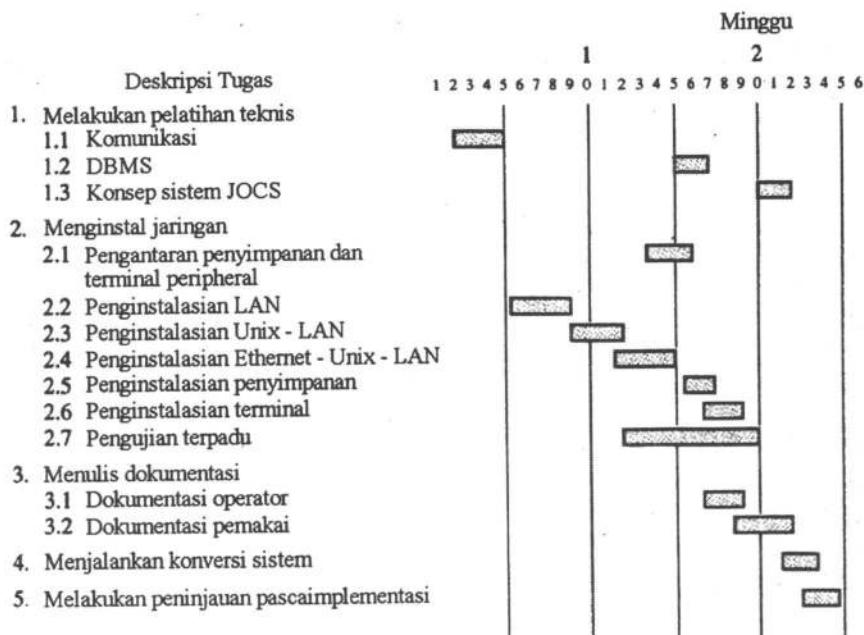
pemakai mengenai JOCS dengan menggunakan staf kita sendiri. Namun demikian, saya lebih suka para pemakai tersebut mendapatkan berbagai macam pelatihan luar mengenai DBMS. Pelatihan vendor sangat mahal. Bagaimana gagasan anda?

Konversi: JOCS adalah paket berbasis-DBMS yang sepenuhnya terintegrasi yang akan mengganti sistem akunting biaya manual. Kenyataan ini menyulitkan kita untuk melakukan konversi secara sedikit demi sedikit ke sistem baru. Hampir tidak mungkin menginstal sistem baru itu bagian per bagian, sebab setiap bagian akan sangat mendukung data dari bagian lain dari sistem itu. Sebagai contoh, kita tidak mungkin menginstal modul penangkapan data online tanpa juga menginstal modul inventarisasi dan modul akunting job. Selain masalah ini, sistem manual tidak memberikan semua informasi yang akan kita gunakan dengan JOCS. Saat ini, keseluruhan job, seperti pekerja dan bahan, tersedia dua sampai empat minggu setelah mereka diaplikasikan ke job tersebut. Total saat ini seringkali tidak benar. Dengan menggunakan data saat itu untuk memvalidasi sistem baru, maka kita akan terlalu memerlukan banyak waktu dan secara potensial akan menyimpang. Akibatnya, sungguh tidak mungkin menjalankan penggantian gradual (sedikit demi sedikit) untuk mengadakan sistem baru, dan sama sekali mustahil untuk mengerjakannya secara paralel. Saya sarankan kita ambil risiko dan menjalankan konversi langsung ke sistem baru. Bagaimana menurut anda?

Date: September 13, 1991 04:10 P.M.
From: Mary Stockland
To: Jake Jacoby
Copies:
Attach:
Subject: Reply to: JOCS Implementation Ideas

Pelatihan Pemakai: Bagaimana jika menghubungi universitas setempat dan/atau community college dan mengecek kelas-kelas (jadwal-jadwal pengadaan kursus) mereka? Kita menggunakan DBMS umum, sehingga mereka mungkin menawarkan atau menyediakan kursus dalam area ini. Ceklah untuk mengetahui apakah mereka menawarkan seminar singkat melalui departemen edukasi lanjutan mereka. Anda mungkin juga harus mengecek sekolah bisnis teknis lokal. Hubungi salah satu distributor komputer lokal dan carilah tahu apakah mereka mau mensponsori kursus bagi kita. Jika saran-saran ini tidak berhasil, saya tahu konsultan yang mau menangani kursus untuk kita. Satu-satunya masalah adalah bahwa kita harus menggunakan fasilitas hardware kita sendiri dan saya lebih suka pelatihan itu dilakukan di luar.

Konversi: Kedengarannya anda benar-benar telah memikirkannya secara cermat. Jika anda berpikir bahwa konversi langsung adalah pilihan kita yang terbaik, mari kita coba.



Gambar 5.12 Diagram Gantt untuk pengimplementasian JOCS (dimulai dari minggu ke-24 dari pengembangan perangkat lunak).

Date: September 15, 1991 03:25 P.M.
 From: Jake Jacoby
 To: Mary Stockland
 Copies: SWAT team group
 Attach: Gantt Chart for JOCS Implementation
 Subject: Implementation Schedule

Kita saat ini sedang memulai minggu ke-24 dari siklus pengembangan JOCS. Pengembangan JOCS kira-kira ketinggalan tiga minggu dari jadwal semula. Kita mengalami beberapa penundaan karena adanya masalah dengan vendor bridge perangkat lunak dan hardware kita dan terjadi lebih dari lima perubahan pemakai yang tidak kita ramalkan terhadap rancangan layar dan laporan. Namun demikian, meskipun terjadi penundaan ini, saya kira kita dapat memulihkan waktunya selama implementasi untuk merampungkan proyek ini agar mendekati tanggal pengantaran yang direncanakan sebelumnya.

Pokok bahasan status meeting (rapat pemantauan) kita Rabu yang akan datang adalah jadwal implementasi untuk JOCs. Saya telah melampirkan diagram Ganit (lihat Gambar 5.12) yang akan kita gunakan sebagai kerangka kerja untuk rencana implementasi kita. Diagram ini dimulai pada minggu ke-24 dari jadwal kita semula (lihat Gambar 1.12).

Tugas-tugas yang ditunjukkan dalam diagram ini akan melengkapi tugas-tugas yang ditunjukkan dalam jadwal awal kita. Agar bisa membantu kita mengimplementasikan jadwal pelengkap ini, saya telah menambahkan orang lain ke tim SWAT. Secara sementara kita meminjam David Martinez dari departemen perkayasaan untuk membantu kita dalam hal pengujian dan penginstalasian telekomunikasi. David dan Tom akan bekerja bersama, setelah mereka kembali dari pelatihan, untuk menyelesaikan instalasi jaringan telekomunikasi kita.

Sampai ketemu di ruang konferensi 308 pada pukul 10.00 A.M. tanggal 18-09-91 untuk rapat pemantauan kita.

6

PEMELIHARAAN SISTEM

APA YANG AKAN ANDA PELAJARI DALAM BAB INI?

Setelah mempelajari bab ini, anda diharapkan dapat:

- Menjelaskan pemeliharaan sistem dan mendefinisikan berbagai jenis pemeliharaan sistem.
- Mengikhtisarkan langkah-langkah proses pemeliharaan sistem langkah-langkah proses pemeliharaan sistem.
- Mendaftarkan prosedur untuk meningkatkan kemampuan pemeliharaan sistem.
- Menamakan dan menjelaskan perangkat CASE yang membantu pemeliharaan sistem.
- Menjelaskan cara pengelolaan pemeliharaan sistem.
- Menjabarkan sistem manajemen perubahan dan menyatakan tujuannya.

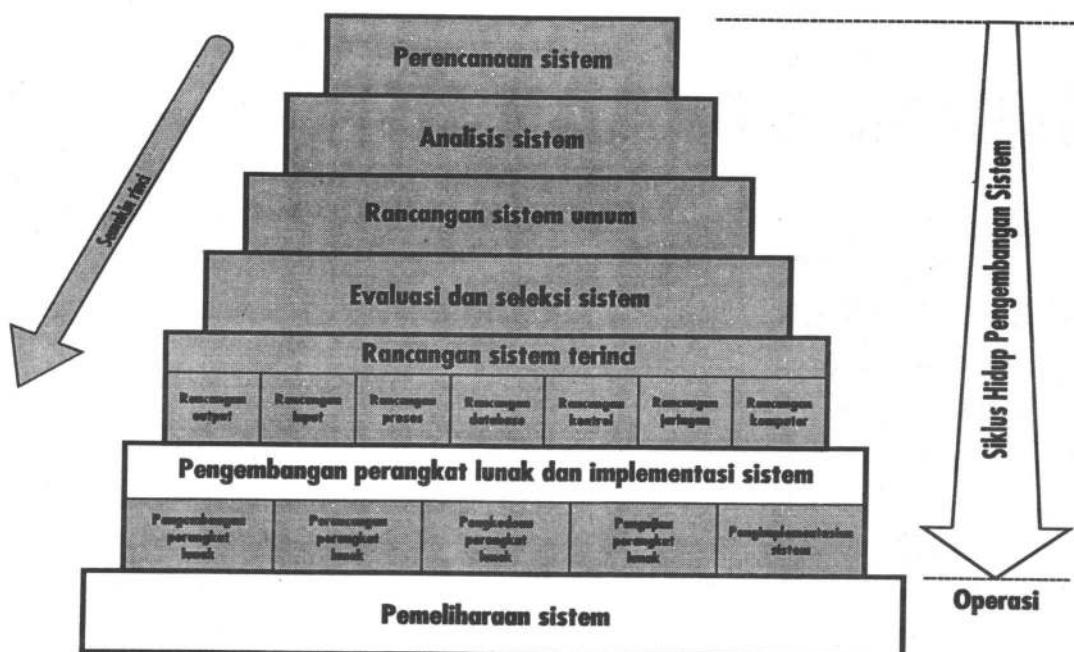
PENDAHULUAN

Pada poin ini sistem baru telah dibangun dan mulai beroperasi (lihat Gambar 6.1). SDLC sudah selesai, dengan sistemnya mulai beroperasi. Kini sistemnya berada di dalam tahap pemeliharaan dari siklus hidupnya. Meskipun hardware maupun perangkat lunak yang menyusun suatu sistem harus dikelola selama operasi, hampir semua bahan dalam bab ini ditujukan ke arah pemeliharaan perangkat lunak.

PEMELIHARAAN SISTEM

Semua sistem informasi sewaktu-waktu berubah. **PEMELIHARAAN SISTEM** adalah kegiatan yang membuat perubahan ini.

Pemeliharaan sistem berasal begitu sistem baru menjadi operasional dan berakhir masa hidupnya, sebagaimana dalam Gambar 6.2. Garis tebal yang menunjukkan pemeliharaan sistem menandakan bahwa tahap pemeliharaan sistem dari siklus



Gambar 6.1. Tahap pemeliharaan relatif pada SDLC.

hidup sistem melebihi waktu, tenaga dan beaya dari semua tahap siklus hidup penyusunan sistem (SDLC) yang digabung.

Umumnya sebagian besar informasi yang dihimpun sewaktu pemeriksaan pasca implementasi digunakan untuk melakukan pemeliharaan pendahuluan. Pemeriksaan periodik, audit dan permintaan para pengguna akan tenus menjadi sumber utama untuk melakukan perawatan sistem di seluruh masa hidup sistem.

Setelah pemeliharaan sistem permulaan dilakukan seusai pemeriksaan pasca implementasi, beaya dan tenaga pemeliharaan sistem seharusnya berkurang. Tetapi setelah beberapa bulan atau tahun semakin banyak permintaan untuk perubahan akan dibuat yang memerlukan peningkatan jumlah biaya maupun tenaga.

Pada suatu hal, sistem ini lebih merepotkan dan memakan biaya untuk dikelola daripada harganya. Bila sebuah sistem menjadi problem nyata bagi para pengguna (pemakai) atau tersedia kesempatan baru, sistem ini akan diganti dengan sistem yang baru.

Biaya pemeliharaan perangkat lunak telah terus menerus naik selama 25 tahun terakhir. Beberapa perusahaan membelanjakan 80% atau lebih dari anggaran sistem mereka pada pemeliharaan perangkat lunak. Ini berarti bahwa timbunan aplikasi baru semakin menumpuk. Persyaratan pengguna baru tertinggal sampai dua tahun atau lebih. Sebenarnya, tidaklah sulit untuk memvisualisasikan perusahaan yang menjadi sedemikian tergantung pada pemeliharaan sehingga semua anggaran sistem digunakan untuk pemeliharaan, dan dengan demikian tidak satupun sistem baru dibangun.

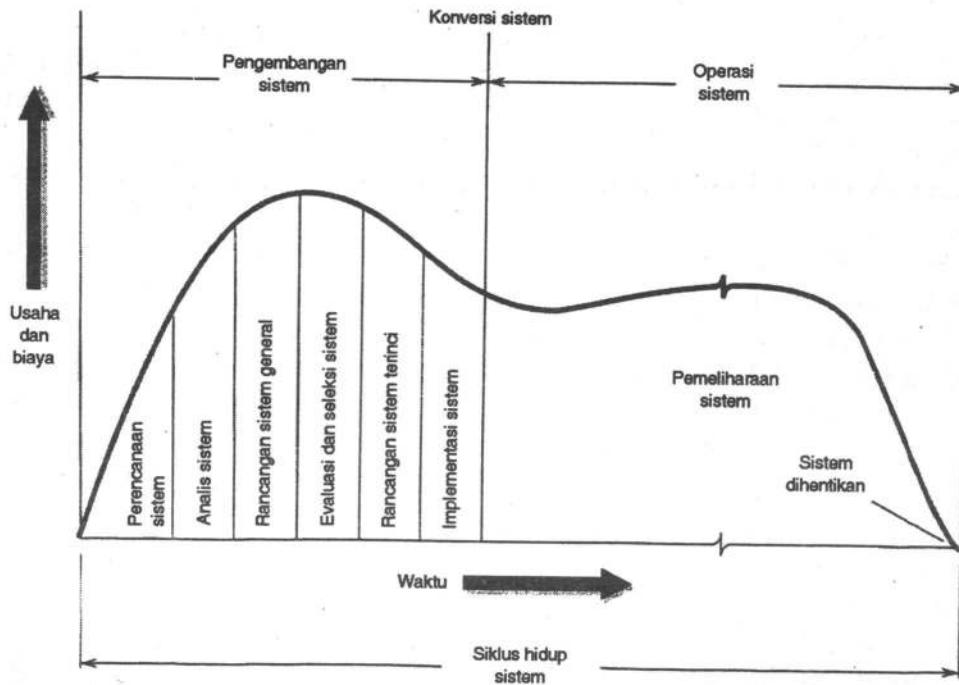
Jenis Pemeliharaan Sistem

Pemeliharaan sistem dapat digolongkan menjadi empat jenis:

- ◆ Pemeliharaan Korektif
- ◆ Pemeliharaan adaptatif
- ◆ Pemeliharaan penyempurnaan
- ◆ Pemeliharaan preventif

Pemeliharaan Korektif

PEMELIHARAAN KOREKTIF adalah bagian pemeliharaan sistem yang tidak begitu tinggi nilainya dan lebih membebani karena pemeliharaan ini mengoreksi kesalahan-kesalahan rancangan, pengkodean, dan implementasi yang seharusnya tidak



Gambar 6.2 Total siklus hidup sistem.

perlu terjadi. Kebutuhan pengelolaan sistem korektif seringkali disebabkan oleh aplikasi SDLC dan SWDLC yang sembarang.

Umumnya pemeliharaan korektif ini mencakup kondisi penting atau bahaya yang memerlukan tindakan segera. Kemampuan untuk mendiagnosa atau memperbaiki kesalahan atau malfungsi dengan cepat sangatlah berharga bagi perusahaan.

Pemeliharaan Adaptatif

PEMELIHARAAN ADAPTATIF dilakukan untuk menyesuaikan perubahan dalam lingkungan data atau pemrosesan dan memenuhi persyaratan pengguna baru. Lingkungan tempat sistem beroperasi adalah dinamik; dengan demikian, sistem harus

terus merespon perubahan persyaratan pengguna. Misalnya, Undang-Undang perpajakan yang baru mungkin memerlukan suatu perubahan dalam kalkulasi pembayaran bersih. Atau diperlukan laporan baru, atau metode depresiasi akuntansi baru harus dipasang sebelum akhir periode fiskal.

Umumnya pemeliharaan adaptatif ini baik dan tidak dapat dielakkan. Namun demikian, terlalu banyak pemeliharaan jenis ini akan berarti bahwa SDLC serta SWDLC-nya tidak menyeluruh dan dilakukan dengan cermat.

Pemeliharaan Penyempurnaan

PEMELIHARAAN PENYEMPURNAAN mempertinggi cara kerja atau maintainabilitas [kemampuan untuk dirawat]. Tindakan ini juga memungkinkan sistem untuk menuhi persyaratan pengguna yang sebelumnya tidak dikenal. Ketika membuat perubahan substansial modul apapun, petugas pemeliharaan juga menggunakan kesempatan untuk mengupgrade kode, mengganti cabang-cabang yang kadaluwarsa, membetulkan kecerobohan, dan mengembangkan dokumentasi. Sebagai contoh, kegiatan pemeliharaan ini mungkin berbentuk perekayasaan ulang atau restrukturisasi perangkat lunak, penulisan ulang dokumentasi, pengubahan format dan isi laporan, penentuan logika pemrosesan yang lebih efisien dan pengembangan efisiensi pengoperasian perangkat.

Pemeliharaan Preventif

PEMELIHARAAN PREVENTIF terdiri atas inspeksi periodik dan pemeriksaan sistem untuk mengungkap dan mengantisipasi permasalahan. Karena personil pemeliharaan bekerja dalam sistem ini, mereka seringkali menemukan cacat-cacat (bukan kesalahan yang sebenarnya) yang menandakan permasalahan potensial. Sementara tidak memerlukan tindakan segera, cacat ini bla tidak dikoreksi di tingkat awal, jelas sekali akan mempengaruhi baik fungsi sistem maupun kemampuan untuk merawatnya di dalam waktu dekat. Motto unuk pemeliharaan preventive mungkin adalah: "Bayar sekarang atau bayar nanti."

Siklus Hidup Pemeliharaan Sistem

Sejumlah tenaga profesional menyarankan agar pemeliharaan perangkat lunak dilakukan dalam siklus hidup **PEMELIHARAAN PERANGKAT LUNAK (SMLC)**. Pada hakekatnya SMLC meliputi tahap-tahap sebagai berikut:

1. **Permintaan Pemeliharaan.** Permohonan pemeliharaan sistem di-dokumenkan dan dikumpulkan oleh seorang pengguna. Permohonan ini digunakan untuk mempersiapkan order pekerjaan (WO,Work Order) pemeliharaan, yang akan dijelaskan belakangan dalam bab ini.
2. **Mengubah Permohonan Pemeliharaan menjadi suatu Perubahan.** Petugas pemeliharaan memiliki deskripsi sistem yang ada disamping sistem yang diinginkan. Mentransformasi permintaan ke perubahan meliputi perbedaan diantara kedua sistem dan mengeliminasi perbedaan yang tepat.
3. **Menspesifikasi Perubahan.** Perubahan dapat mencakup salah satu atau semua kode, data, prosedur, atau pun perangkat keras yang ada. Kode, data, dan prosedur kadangkala dianggap sebagai suatu tambal sulam. Perubahan dalam perangkat keras mungkin meliputi penggantian atau perbaikan suku cadang atau merekonfigurasi perangkat.
4. **Membangun Pengganti.** Perubahan atau tambal sulam dalam program perangkat lunak, kecuali kalau sangat sederhana, dirancang dan dikode dalam suatu cara yang mirip dengan tahap perancangan dan pengkodean SWDLC.
5. **Menguji Pengganti.** Soal-soal pengetesan yang digunakan dalam tahap pengujian SWDLC yang tersimpan di repository sentral sistem CASE dapat digunakan untuk mengetes perangkat lunak setelah diganti. Tujuan pengetesan adalah untuk membantu memvalidasi dan memverifikasi bahwa perubahan yang benar telah dibuat dan dibuat dengan benar.
Pengujian regresi atau revalidasi mencoba mencocokkan bahwa fungsi sistem yang tidak diganti tetap perfungsi sebagaimana mestinya sebelum perubahan yang dikehendaki dilakukan. Jadi tes regresi memfokuskan pada integritas fungsional sistem.
6. **Melatih Pengguna dan Melakukan Tes Penerimaan.** Bila perubahannya relatif sederhana, langkah ini dapat dilewati. Sebaliknya, bila perubahan ini memperkenalkan cara baru tempat para pengguna melakukan pekerjaannya, pengguna ini harus dilatih. Setelah mereka dilatih, tes penerimaan sebaiknya dilakukan dengan menggunakan testing alfa seperti yang dijabarkan di Bab 4.
7. **Pengkonversian dan Penglepasan ke Operasi.** Setelah sistem yang diperbarui lulus tes dengan baik, sistem itu siap dikonversi dan diluncurkan kembali untuk beroperasi.
8. **Mengupdate Dokumentasi.** Semua dokumentasi yang berkenaan dengan kegiatan pemeliharaan harus diupdate untuk mencerminkan perubahan dan sistem baru untuk sistem.
9. **Melakukan Pemeriksaan Pascaimplementasi.** Setelah sistemnya beroperasi selama beberapa minggu (atau mungkin beberapa hari) seusai

dilakukan pemeliharaan, petugas pemeliharaan harus melakukan pemeriksaan pasca implementasi untuk menentukan bahwa perubahan itu terus memenuhi keinginan penggunanya.

Memelihara Perangkat Lunak

Tidak seperti mesin pabrik atau perangkat komputer, perangkat lunak tidak usang karena program perangkat lunak tidak berisi suku cadang yang bergerak. Namun meskipun tidak usang dalam arti fisik, perangkat lunak tentu saja memerlukan perubahan.

Perangkat lunak aplikasi mungkin terstruktur, mungkin pula tidak, atau mungkin didokumentasi mungkin pula tidak. Beberapa perangkat lunak yang tidak terstruktur dan tidak terdokumentasi mungkin hampir tidak dapat di-maintain. Sebenarnya salah satu sebab utama mengapa pemeliharaan sistem memerlukan anggaran sistem yang amat banyak adalah karena kenaikan tenaga yang dibutuhkan untuk mencoba memaintain perangkat lunak yang didokumentasi serta distruktur secara acak-acakan.

Di lain pihak program perangkat lunak yang tidak terstruktur dan tidak ter-dokumen juga tidak dapat di-maintain. Seandainya suatu perubahan dalam operasi memaksa program itu untuk berubah, maka program itu harus disingkirkan dan dikembangkanlah program baru, sehingga menyia-nyiakan semua sumber yang dikeluarkan untuk membangun program asli yang tidak dapat di-maintain tersebut, belum lagi kerugian operasi bisnis bila hari yang ditentukan tiba.

Memelihara Perangkat keras

Meskipun sebagian besar bab ini membahas pemeliharaan perangkat lunak, pemeliharaan perangkat keras juga merupakan bagian penting dalam pemeliharaan sistem. pemeliharaan perangkat lunak terutama pemeliharaan preventif yang memerlukan reparasi, penggantian, atau penambahan suku cadang dan komponen untuk merestorasi atau menjaga agar perangkat keras tetap bekerja dengan baik. Dalam banyak hal perangkat komputer mirip dengan mobil; bilamana tidak dirawat dengan tepat ia akan rusak dan menimbulkan masalah. Mobil diservis secara periodik untuk mencegah problem-problem mekanis utama. Dengan cara yang sama komponen perangkat keras sistem informasi sebaiknya dicek dan diservis secara periodik.

Jelasnya, sumber pemeliharaan perangkat keras baru yang paling lazim adalah penyulur yang menyuplai perangkat tersebut. Khususnya, penyulur mainframe atau

Kegiatan pemeliharaan harian terutama berkaitan dengan administrasi pengguna, misalnya menambah dan menghapus pengguna; memasang tingkat keamanan elemlenter semacam password; dan memperbanyak ruang piringan (disk space).

Pemeliharaan mingguan mencakup pemeliharaan disk, pembackupan, dan konfigurasi penyaji. Disk di format ulang dan file dibuat backupnya serta disimpan di tempat yang terpisah jauh.

Kegiatan pemeliharaan berkala dikerjakan berdasarkan hal khusus. Kegiatan ini meliputi penyesuaian parameter sistem; melakukan pemeliharaan printer; dan membuat tambahan dan upgrade-upgrade.

Upgrade dan penambahan ini lebih sederhana bila aplikasi dan perangkat lunak jaringan hanya bersisa di penyaji. Hanya driver jaringan dan perintah logon yang tetap berada di bengkel kerja. Meskipun ini akan meningkatkan lalu lintas pada jaringan, keuntungan pengendalian, standardisasi, dan sebuah lokasi menilai pemusatan aplikasi dan perangkat lunak jaringan.

Mengawali dan Merekam Kegiatan Pemeliharaan Sistem Tidak Terjadwal

Apa yang memicu pemeliharaan sistem tidak terjadwal? Pemeriksaan pasca implementasi yang dibahas pada Bab 5 umumnya mengawali pemeliharaan sistem pada permulaan masa pengoperasian sistem baru. Semasa pemeriksaan pasca implementasi kesalahan dapat dideteksi dan cara-cara untuk mempertinggi perangkat lunak dapat ditemukan. Lebih daripada itu, para manager seharusnya melakukan audit periodik untuk memastikan bahwa tujuan, kebijaksanaan, dan prosedur yang ada telah dijalankan dengan tepat. Tetapi melangsungkan pemeliharaan sistem memerlukan cara formal untuk mengawali permintaan pemeliharaan. Ini dapat dicapai dengan menggunakan **FORMULIR PERINTAH KERJA (WO) PEMELIHARAAN**.

WO ini, sebagaimana yang digambarkan pada Gambar 6.5, adalah dokumen yang digunakan untuk mengawali dan merekam pekerjaan pemeliharaan sistem. Manajemen khususnya tertarik pada :

- ◆ Pekerjaan yang diperlukan.
- ◆ Pekerjaan yang dilakukan.
- ◆ Waktu yang diperkirakan dibandingkan dengan waktu yang sebenarnya.
- ◆ Kode pemeliharaan.
- ◆ Biaya pemeliharaan.

Deskripsi pekerjaan yang diminta seharusnya sejelas mungkin, yang menjelaskan apa yang perlu diubah. Orang yang membuat permintaan seharusnya mencantumkan nama dan jabatannya. Nomor yang digunakan untuk setiap WO pemeliharaan adalah khusus. Tanggal permintaan juga dimasukkan. Prioritas menunjukkan seberapa cepat pemeliharaan sistemnya harus dimulai. Bila gawat darurat, pekerjaan pemeliharaan segera dimulai. Prioritas gawat darurat umumnya berarti bahwa sistem berhenti beroperasi. Bila sangat mendesak, pekerjaan pemeliharaan untuk prioritas ini menyela jadwal pemeliharaan dan berawal pada jadwal harian berikutnya yang tersedia. Prioritas mendesak biasanya mengimplikasikan bahwa problem timbul yang memiliki suatu kemungkinan dapat menghentikan pengoperasian dalam waktu dekat. Bila permintaan WO adalah permintaan rutin, pekerjaan ini ditempatkan pada jadwal mingguan berikutnya yang tersedia. Prioritas rutin biasanya berarti bahwa kondisi defektif telah teridentifikasi. Kondisi ini pada umumnya tidak menghentikan operasi menimbulkan kerusakan bila dikoreksi selama minggu berikutnya sampai dengan empat minggu berikutnya.

Supervisor pemeliharaan bersama-sama dengan teknisi atau programmer pemeliharaan seharusnya memasukkan estimasi waktu yang diperlukan untuk melakukan pekerjaan pemeliharaan dalam bentuk jam dan menit. Setelah pekerjaan pemeliharaan selesai, waktu sebenarnya yang diperlukan dimasukkan dan dibandingkan dengan waktu perkiraan. Perbandingan ini digunakan untuk maksud kendali dan penjadwalan yang akan datang.

Kode pemeliharaan adalah singkatan deskripsi pekerjaan pemeliharaan. Ini menunjukkan jenis pemeliharaan, seperti:

- ◆ Korektif.
- ◆ Adaptif.
- ◆ Penyempurnaan.
- ◆ Preventif.

Kode ini menunjukkan prioritas permintaan, seperti:

- ◆ Gawat darurat
- ◆ Mendesak
- ◆ Rutin.

Kode tersebut juga mengklasifikasi jenis tugas yang tercakup, seperti:

- ◆ Membongkar (scrap)

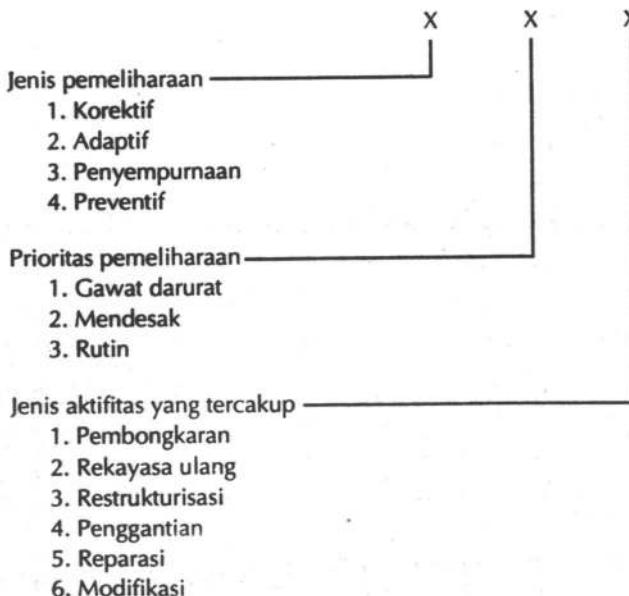
MAINTENANCE WORK ORDER	
Work required:	Add vendor performance code to inventory record.
	Number <input type="text"/>
	Request Date <input type="text"/> MM DD YY
Name:	Tom Barkley Title: Purchasing Agent
Work performed:	Added vendor performance code. Tested and documented change.
Priority	<input type="checkbox"/> Emergency <input type="checkbox"/> Urgent <input type="checkbox"/> Routine
Estimated time	<input type="text"/> HH:MM
Name:	Maria Gomez Title: Maintenance Programmer
Change approved:	Harry Feldman Date: MM/DD/YY
Actual time	<input type="text"/> HH:MM
Maintenance code	<input type="text"/>
Labor cost (actual time * labor rate) \$ <input type="text"/>	
Material or parts cost <input type="text"/>	
Total maintenance cost \$ <input type="text"/>	

Gambar 6.5. Formulir Perintah Kerja Pemeliharaan.

- ◆ Rekayasa ulang (reengineer)
- ◆ Restrukturisasi
- ◆ Mengganti
- ◆ Mereparasi
- ◆ Memodifikasi.

Pembongkaran, rekayasa ulang, dan restrukturisasi telah dijelaskan sebelumnya. Penggantian berarti bahwa suatu modul perangkat lunak dihapus dan diganti dengan modul yang baru. Reparasi berarti suatu elemen sistem direstorasi ke tingkat yang kuat. Modifikasi meliputi perubahan-perubahan kecil. Misalnya, suatu instruksi mungkin disisipkan pada layar masukan data supaya formulir elektronik lebih mudah dipahami. Kode pemeliharaan dapat disusun seperti yang ditunjukkan pada alinea berikut ini. Misalnya, kode 116 menunjukkan pemeliharaan jenis korektif dalam prioritas gawat darurat yang memerlukan aktifitas modifikasi.

Pekerjaan yang dilakukan seharusnya dijelaskan agar memberikan arti tambahan pada kode pemeliharaan. Dalam beberapa hal pekerjaan sebenarnya yang ditampilkan mungkin memerlukan lebih banyak waktu dan tenaga daripada yang sebelumnya



dipikirkan oleh orang yang mengajukan permintaan. Orang yang melakukan pekerjaan pemeliharaan seharusnya memasukkan nama serta jabatannya. Bila pekerjaannya selesai, supervisor harus membubuhkan namanya pada baris yang disetujui dan memasukkan tanggal penandatanganan. Tanda tangan ini akan menunjukkan bahwa sang supervisor telah menginspeksi pekerjaan yang sudah selesai dan bahwa pekerjaannya telah dikerjakan sesuai dengan kebijaksanaan pemeliharaan perusahaan khusus itu serta bahwa kualitasnya tepat.

Tujuan utama formulir WO pemeliharaan selain memicu pekerjaan pemeliharaan adalah untuk :

- ◆ Menyediakan sarana untuk menyaring dan mengesahkan pekerjaan.
- ◆ Menyebarkan data biaya pemeliharaan.
- ◆ Menutup dampak negatif operasi.
- ◆ Berperan sebagai masukan untuk perencanaan perencanaan, penjadwalan, dan pengontrolan pekerjaan pemeliharaan.

Menentukan sebab untuk permintaan pemeliharaan sangatlah penting bagi manajemen. Penyimpanan catatan yang baik dari formulir WO pemeliharaan sepanjang waktu dan analisis mengenai WO-WO ini akan menunjukkan:

- ◆ Kurangnya program pemeliharaan preventif yang memadai.
- ◆ Kesalahan pemeliharaan sebelumnya.
- ◆ Ketidakbenaran evaluasi dan seleksi perangkat.
- ◆ Kekeliruan dalam dalam implementasi.
- ◆ Kesalahan ataupun kelalaian operator atau pengguna.

Mempelajari sebab-sebab pemeliharaan secara rinci merupakan cara yang penting untuk membantu mengembangkan program pemeliharaan sistem suatu perusahaan.

Menggunakan Sistem Perangkat lunak Help-desk

Perangkat lunak untuk sistem help-desk mirip dengan apa yang mungkin digunakan dalam bagian bantuan teknis atau bagian pelayanan pelanggan dari penyalur hardware atau perangkat lunak. Sistem help-desk memungkinkan permintaan pemeliharaan dimasukkan ke daftar prioritas antrian. Sistem menyebarkan formulir WO pemeliharaan. Sistem juga menyediakan informasi mengenai problem pemeliharaan sejenis sebelumnya dan solusinya. Kemampuan ini mengeliminasi problem potensial

personil pemeliharaan yang secara kontinyu menggambarkan bagaimana mengatasi masalah yang berulang-ulang timbul ketika mereka dapat mencari database untuk solusi permasalahan itu. Sistem ini juga membantu saat personil pemeliharaan meninggalkan perusahaan atau berlibur.

Penerapan sistem help-desk menurut langkah-langkah ini :

1. Pengguna terakhir menjumpai suatu problim dan memanggil help-desk.
2. Koordinator help-desk menjawab panggilan ini dan memasukkan masalah ke sistem help-desk.
3. Formulir WO pemeliharaan dicetak dan diberikan kepada teknisi pemeliharaan.
4. Teknisi pemeliharaan mencari database sistem help-desk untuk setiap solusi pada problim yang sama.
5. Setelah memecahkan problimnya, teknisi pemeliharaan memasukkan solusi ini ke database untuk acuan di masa mendatang.
6. Formulir WO pemeliharaan diisi dan dimasukkan ke dalam database.

Mengevaluasi Aktivitas Pemeliharaan Sistem

Untuk bekerja menuju tujuan program optimalisasi pemeliharaan (dibahas dalam bagian yang akan datang), manajer pemeliharaan dapat menggunakan formulir WO pemeliharaan sebagai dokumen utama untuk tiap aktivitas berikut ini :

- ◆ Komputasikan suatu jenis analisis biaya pemeliharaan.
- ◆ Hitunglah jumlah kegagalan per program.
- ◆ Kalkulasikan jam kerja yang digunakan untuk setiap type maintenance.
- ◆ Komputasikan proporsi pemeliharaan gawat darurat, mendesak dan rutin.
- ◆ Ambillah jumlah rata-rata perubahan yang dibuat per program, per bahasa, dan per type pemeliharaan.
- ◆ Kembangkan suatu profil mengenai hal-hal yang paling memerlukan pemeliharaan dan masalah yang biasa dijumpai.
- ◆ Kalkulasikan setiap putaran waktu per WO pemeliharaan. Kalkulasi ini memerlukan MTTR.
- ◆ Susunlah profil MTBF pada semua aplikasi. Dari profil ini tentukan aplikasi-aplikasi yang memerlukan porsi pemeliharaan terbanyak dan sebab-sebabnya. Seringkali ditemukan bahwa 20% dari aplikasi menyebabkan 80% atau lebih problem.

Mengoptimalkan Program Pemeliharaan Sistem

Tujuan program pemeliharaan sistem yang terarah dengan baik adalah untuk mendapatkan biaya terrendah dari jumlah total dua kuantitas:

- ◆ Biaya pemeliharaan, yang meliputi tenaga kerja, material, suku cadang dan waktu komputer.
- ◆ Kerugian operasi yang disebabkan oleh penundaan, inefisiensi, atau produksi hasil-hasil yang tidak benar.

Sebagaimana ditunjukkan oleh Gambar 6.6, biaya gabungan terrendah muncul dimana tingkat pemeliharaan dioptimalkan.

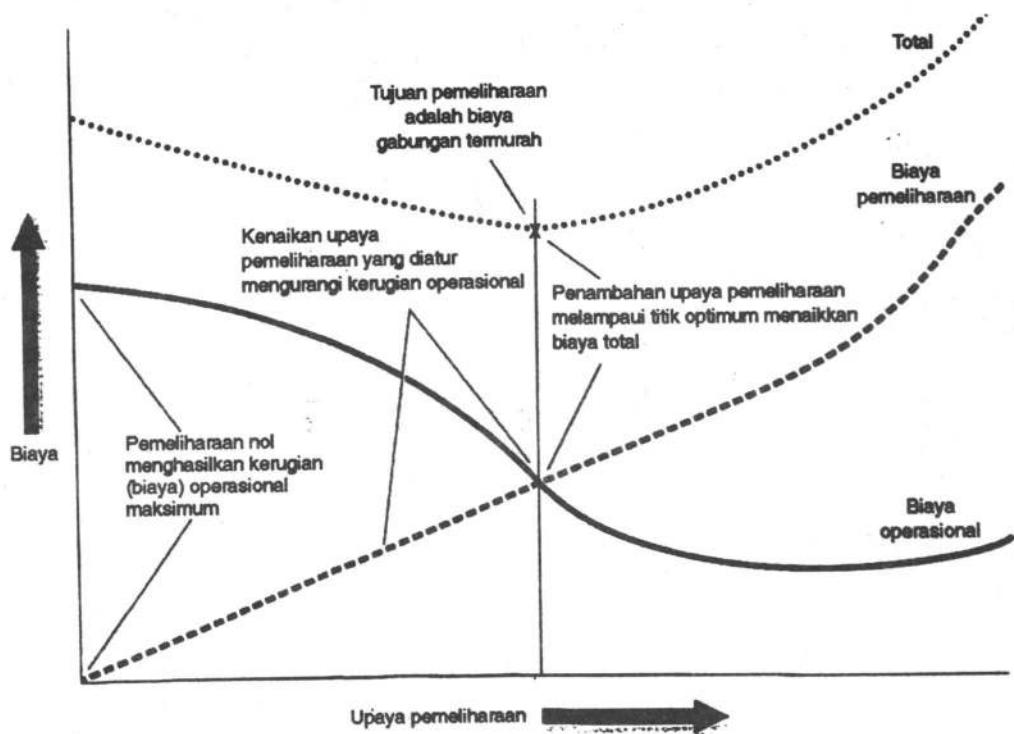
Karena usaha pemeliharaan sistem dengan cerdik bertambah, kerugian operasi berkurang sampai biaya gabungan terendah dicapai. Pada poin ini tujuan program pemeliharaan yang terarah dengan baik tercapai. Usaha pemeliharaan sistem yang diperlukan, bila melampaui point ini menambah biaya total dan mengubah pemeliharaan sistem dari fungsi optimal dan penting menjadi sangat buruk.

Seberapa banyak usaha harus dicurahkan pada pemeliharaan preventif sebagai bagian dari program optimalisasi pemeliharaan? Tugas-tugas pemeliharaan preventif akan mempertinggi biaya pemeliharaan ketika pertama kali dimulai, sampai dampak menguntungkan tugas-tugas ini mempunyai waktu untuk berlaku. Kemudain total biaya harus mulai berkurang.

Untuk beberapa bagian perangkat yang harganya hanya beberapa dollar, mungkin tindakan yang terbaik adalah tidak melakukan suatu pemeliharaan preventif. Daripada melakukan pemeliharaan preventif, adalah lebih efektif bila membiarkan bagian itu terus bekerja sampai hancur dan kemudian mereparasi atau menggantinya. Sebaliknya, perangkat dan perangkat lunak yang harganya relatif lebih mahal, yang harus dioperasikan pada tingkat penurunan yang dapat dicapai terrendah karena kekritisan aplikasi yang mereka bantu, haruslah memiliki program optimalisasi pemeliharaan sistem yang seluruhnya agresif.

MENGEMBANGKAN SISTEM MANAJEMEN BERUBAH

SISTEM MANAJEMEN BERUBAH (CMS) yang komprehensif dapat membantu mengurangi kebingungan dan kerumitan pengembangan sistem baru dan pemaintainan sistem-sistem yang ada. CMS, yang bisa jadi merupakan bagian dari sistem CASE

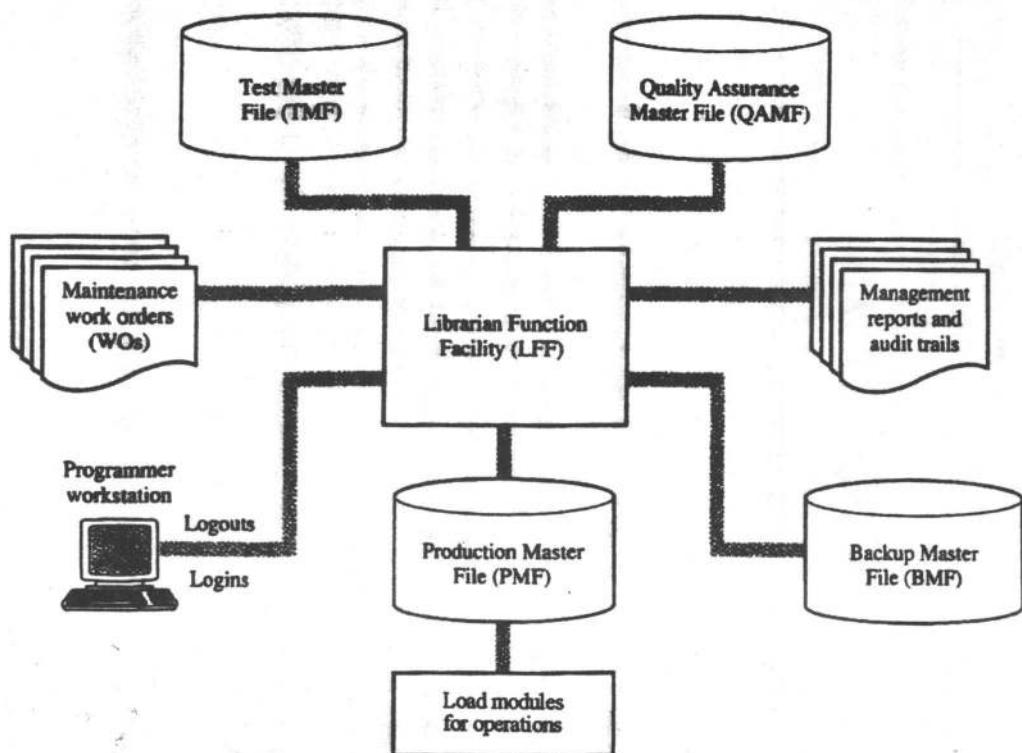


Gambar 6.6 Grafik yang menunjukkan program pemeliharaan sistem yang dioptimalkan.

atau pun sebagai sistem yang berdiri sendiri, dapat pula menyediakan sebagian besar dari prosedur pemeliharaan yang dibahas pada bahan-bahan terdahulu.

Model umum CMS digambar dalam Gambar 6.7. CMS adalah:

- ◆ Membatasi akses kepada sumber produksi dan kode obyek.
- ◆ Mengurangi kesalahan dan mendesain cacat-cacat dari sejak diperkenalkan sampai dengan diproduksi.
- ◆ Mencegah keberadaan lebih dari satu versi program sumber dan kode obyek dalam file master produksi (PMF, production master file).



Gambar 6.7 Sistem Manajemen Berubah (CMS, Change Management System).

- ◆ Mengembangkan kualitas dan kehandalan (reliabilitas) perangkat lunak.
- ◆ Mempertinggi kamanan dan kendali keseluruhan yang lebih baik terhadap pengembangan perangkat lunak dan aktivitas pemeliharaan sistem.
- ◆ Mempertinggi produktivitas perangkat lunak.

Apa Sajakah Komponen CMS?

Komponen CMS meliputi:

- ◆ Fasilitas fungsi perpustakaan (The Librarian Function Facility, LFF).
- ◆ Perintah Kerja pemeliharaan (The pemeliharaan Work Order[WO]).
- ◆ Bengkel Kerja Programmer.
- ◆ File Master Tes (The Test File Master, TMF).
- ◆ File Master Jaminan Kualitas (The Quality Assurance Master File, QAMF).
- ◆ File Master Produksi (The Production Master File, PMF).
- ◆ File Master Cadangan (The Backup Master File, BMF).
- ◆ Laporan Manajemen dan jejak audit.

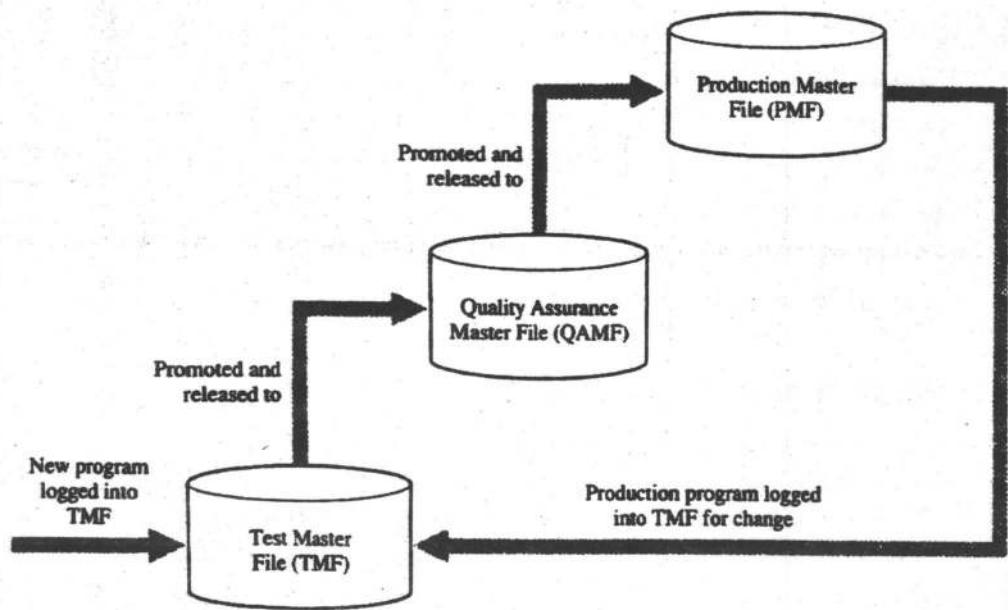
Fasilitas Fungsi Perpustakaan (LFF, Librarian Function Facility)

LFF adalah jantung CMS. LFF bekerja dalam suatu cara yang mirip dengan setiap fungsi perpustakaan. Komponen ini merupakan paket perangkat lunak yang memusatkan, melacak, mengontrol, dan mengotomatisasi perubahan-perubahan pada program diluar WO pemeliharaan yang disetujui. Komponen ini juga mengontrol implementasi program yang baru dikembangkan atau didapatkan. Gambar 6.8 menggambarkan hirarki peluncuran dan promosi program. Bila program yang siap beroperasi harus diubah, program tersebut dimasukkan ke file master tes (TMF). Bila program baru harus dioperasikan, pertama-tama program itu harus memasuki PMF. Tidak boleh ada perubahan di dalam file master produksi (PMF).

LFF mengendalikan hubungan antara kode sumber dan obyek, dan secara otomatis memuat modul yang siap bereksekusi dan dengan demikian mengasumsikan sinkronisasi antara dua kode. Laporan manajemen dan jejak audit komprehensif tersedia melalui layar dan copy tertulis untuk informasi sejarah, status, pelacakan, dan penampilan/cara kerja. Semua file master dibackup untuk mengamankan sistem dari kerusakan. Para programmer memiliki akses online untuk memperbaik produktivitas perubahan. Keleluasan akses programmer dikendalikan oleh sarana kendali password atau biometrik.

Bengkel Kerja Programmer

CMS berperan sebagai titik tunggal pengendalian, dan WO pemeliharaan yang disetujui memulai pengeluaran dan pemasukan yang memungkinkan programmer melakukan pekerjaannya di tempat kerja mereka. Pengeluaran (logout) dikerjakan diluar WO yang disetujui dan ditandatangani. (WO ditandatangani oleh supervisor CMS). Ketika bekerja diluar WO khusus, programmer diperbolehkan melogout



Gambar 6.8 Hirarki promosi dan peluncuran program.

sebanyak mungkin modul yang penting untuk melengkapi tuntutan pekerjaan yang ditugaskan. Modul program yang menyolok bagi WO lain dicatat selama proses logout.

Login (pemasukan) dilakukan untuk mempromosikan dan meluncurkan program yang diubah dan dites ke file master jaminan kualitas (QAMF, Quality Assurance Master File), dan kemudian ke file master produksi (PMF, production master file). Pengetesan yang seksama dan prosedur berjalan dilakukan sebelum setiap program atau modul program dipromosikan dan diluncurkan ke PMF.

Problem produksi yang muncul di tengah malam, umumnya mengharuskan para programmer tetap ada di tempat atau, bila di rumah, harus kembali ke kantor untuk mengkoreksi problem itu. Programmer siap panggil dengan bengkel kerja diinstall di rumahnya dan dihubungkan dengan CMS dapat menambah efisiensi pekerjaan pemeliharaan. Meskipun pengesetan ini tidak mengeliminasi panggilan telepon mengkhawatirkan, hal ini dapat mengurangi keharusan untuk pergi ke kantor.

File Master Uji

Untuk mengubah program produksi, WO harus dibuka dan programnya diturunkan dari PMF ke TMF. Hal ini mengubah program dari status produksi menjadi status tes. Selama status tes ini dijaga programnya dapat diubah.

Ketika suatu program pertama kali memasuki TMF, secara otomatis ia diberi nama dan tingkat 1, atau tingkat 0 pada beberapa sistem. Setiap kali program ini diubah, levelnya ditambah dengan angka 1. Untuk mengubah suatu program, si programmer harus menggunakan nama maupun nomor level yang benar. Nomor level berfungsi untuk :

- ◆ Melindungi (memprotect) dari penggunaan kode sumber lama yang didaftar dalam versi sebelumnya.
- ◆ Melindungi dari pengupdatean duplikat.
- ◆ Menyediakan sejarah perubahan program dan aktivitas programmer.

Program perangkat lunak baru yang memasuki TMF menjadi sasaran testing modul, testing integrasi modul, testing sistem, dan testing penerimaan, yang diungkapkan dalam Bab 4. Bila lulus tes, program ini dipromosikan dan diluncurkan ke QAMF. Program produksi yang diturunkan ke TMF telah menjadi sasaran tes-tes ini, tetapi setelah diubah, ia harus menjalani tes regresi. Ujuan tes regresi adalah untuk memastikan bahwa perubahan itu bekerja sesuai dengan yang dimaksudkan, dan tidak menyebabkan suatu dampak yang merugikan.

File Master Jaminan Kualitas (Quality Assurance Master File, QAMS)

Beberapa CMS mempekerjakan kelompok jaminan kualitas (quality assurance, QA) independen yang memiliki akses pada QAMF. Kelompok QA ini menampilkan walkthrough rancangan perangkat lunak dan berbagai tes lain. Group QA juga memastikan bahwa software baru dikembangkan sesuai dengan standar perusahaan sebelum dipromosikan dan diluncurkan ke PMF.

File Master Produksi (Production Master File, PMF)

Begitu suatu program memasuki PMF, program itu dikunci pada status produksi dan tidak dapat diubah. Dengan pengesahan yang tepat suatu program dapat dicopy dan

dilog kedalam TMF dengan nama baru dan versi copyan ini dapat diubah. Fungsi protektif ini membantu memastikan agar program perangkat lunak produksi tidak dapat diubah tanpa pemberitahuan. Hal ini juga mencegah dari perubahan yang tidak sah atau yang jahat.

Bila manajemen tidak lagi menginginkan suatu program khusus di dalam PMF, statusnya diubah dari dapat diubah menjadi tidak dapat diubah. Perubahan status ini tidak menghapus program dari PMF tetapi menurunkannya untuk penghapusan. Hanya para manajer yang berhak, yang mungkin memerlukan penjagaan dua kali atau tiga kali lipat, yang dapat menghapus program produksi yang tidak bisa diubah tersebut dari PMF. Penampilan ini mendukung kendali program, kontinuitas dan penjagaan program.

File Master Cadangan (Backup Master File, BMF)

Bila terjadi suatu peristiwa yang menyebabkan suatu file master rusak, CMS mengijinkan personil sistem informasi untuk memulihkan setiap file atau modul-modul tertentu, yang mungkin telah hilang, dari BMF. Biasanya sebuah copy BMF dirawat secara lokal, dan copy lain disimpan di tempat yang terpisah jauh di lokasi yang aman.

Laporan Manajemen dan Jejak Audit

Tujuh laporan membantu manajer mengembangkan CMS yang dioptimalkan dan membantu para auditor menguji integritas CMS. Sebagaimana dibahas sebelumnya dalam bab ini, laporan manajemen membantu mengevaluasi dan mengoptimalkan kegiatan pemeliharaan sistem. Jejak audit membantu memastikan integritas sistem.

Resiko-resiko apa saja yang CMS hindarkan?

CMS membantu sistem informasi menghindarkan hal-hal berikut ini :

- ◆ Kekurangan inventaris program perangkat lunak yang akurat dan sumber-sumber sistem informasi lainnya.
- ◆ Ketidaklengkapan sejarah perubahan program.

- ◆ Modul-modul program perangkat lunak terduplicasi.
- ◆ Perubahan program perangkat lunak yang tidak sah.
- ◆ Kekurangan dokumentasi yang jelas, komprehensif dan terbaru.
- ◆ Rendahnya kualitas dan reliabilitas perangkat lunak.

TINJAUAN SASARAN BELAJAR UNTUK BAB INI

Tujuan utama bab ini adalah memungkinkan setiap siswa untuk mencapai enam sasaran belajar penting. Kita sekarang akan meringkas tanggapan terhadap sasaran belajar ini.

Sasaran belajar 1:

Menjelaskan pemeliharaan sistem dan mendefinisikan berbagai jenis pemeliharaan sistem.

Setelah sebuah sistem diimplementasi, sistem ini harus dikelola sampai tidak lagi ekonomis bila dilakukan, pada saat mana sistem yang ada dihilangkan dan sistem baru dikembangkan. pemeliharaan sistem memerlukan perangkat keras maupun perangkat lunak.

Ada empat jenis pemeliharaan sistem :

- ◆ Korektif.
- ◆ Adaptatif.
- ◆ Penyempurnaan (perfektif)
- ◆ Preventif.

Pemeliharaan korektif meralat kesalahan rancangan, pengkodean, atau implementasi. pemeliharaan adaptatif menyesuaikan perubahan kondisi lingkungan. pemeliharaan perfektif cenderung membuat sistem sempurna. pemeliharaan preventif mengantisipasi dan mencegah problem-problem potensial.

Sasaran belajar 2:

Mengikhtisarkan langkah-langkah proses pemeliharaan sistem.

Proses pemeliharaan sistem (dalam hal ini, SMLC) meliputi sembilan langkah berikut ini :

- ◆ Memahami permintaan pemeliharaan.
- ◆ Mentransformasikan permintaan pemeliharaan menjadi pengubahan.
- ◆ Menspesifikasi perubahan.
- ◆ Mengembangkan perubahan.
- ◆ Mengetes perubahan.
- ◆ Melatih para pengguna dan menjalankan tes penerimaan.
- ◆ Mengkonversi dan meluncurkan operasi.
- ◆ Mengupdate dokumentasi.
- ◆ Melakukan pemeriksaan pasca implementasi.

Sasaran belajar 3:

Mendaftar prosedur untuk meningkatkan kemampuan pemeliharaan sistem.

Semua sistem seharusnya dikembangkan dengan mengingat kemampuan pemeliharaannya. Semakin mudah suatu sistem dipelihara, semakin kecil pula waktu serta biaya yang harus dikeluarkan untuk pemeliharaan. Rancangan untuk maintainabilitas sistem meliputi prosedur-prosedur berikut ini :

- ◆ Menerapkan SDLC dan SWDLC.
- ◆ Menspesifikasi definisi data standar.
- ◆ Menggunakan bahasa pemrograman standar.
- ◆ Merancang modul-modul yang terstruktur dengan baik.
- ◆ Mempekerjakan modul yang dapat digunakan kembali.

- ◆ Mempersiapkan dokumentasi yang jelas, terbaru dan komprehensif.
- ◆ Menginstall perangkat lunak, dokumentasi dan soal-soal tes di dalam sentral repositor sistem CASE atau CMS.

Personel pemeliharaan sistem mungkin disusun dengan menggunakan :

- ◆ Pendekatan Pemisahan.
- ◆ Pendekatan Gabungan.
- ◆ Pendekatan Fungsional.

Sasaran belajar 4:

Menamakan dan menjelaskan perangkat CASE yang membantu pemeliharaan sistem.

CASE tools spesial yang membantu pemeliharaan sistem dan mengurangi timbunan pengembangan adalah :

- ◆ Rekayasa maju.
- ◆ Rekayasa mundur.
- ◆ Rekayasa ulang.
- ◆ Restrukturisasi.
- ◆ Sistem pakar pemeliharaan.

Rekayasa maju mengembangkan sistem baru langsung pada saat pertama kali. Rekayasa mundur berawal dengan sistem yang dirancang sembarangan dan mengabstraksi spesifikasi rancangan tingkat tinggi darinya untuk penelitian dan evaluasi. Rekayasa ulang menggunakan kombinasi rekayasa maju dan mundur untuk memperbaiki lagi sistem yang ada dengan mengubah bentuk dan fungsionalitasnya. Restrukturisasi mengubah kode yang seperti benang kusut menjadi kode yang terstruktur, tetapi tidak mengubah fungsionalitas kode. Sistem pakar pemeliharaan membimbing maintainer yunior.

Sasaran belajar 5:

Menjelaskan cara pengelolaan pemeliharaan sistem.

Tujuan utama mengarahkan pemeliharaan sistem adalah menggaji atau melatih maintainer yang sangat berminat dan trampil. Kegiatan pemeliharaan mereka seharusnya dijadwal dan dihitung. Pemeliharaan tak terjadwal seharusnya dipicu oleh WO pemeliharaan. Lebih jauh lagi, formulir perintah kerja pemeliharaan akan menyediakan manajemen dengan data yang cukup untuk membuat sejumlah ukuran guna mengevaluasi sejauh mana kemajuan program pemeliharaan sistem. Tujuan manajemen yang paling utama adalah untuk mendapatkan program pemeliharaan sistem yang optimal.

Sasaran belajar 6:

Menjabarkan sistem manajemen perubahan dan menyatakan tujuannya.

Salah satu kegiatan yang termudah disrang adalah penginstallan sistem baru dan perubahan sistem yang ada (kadangkala disebut sistem warisan). Selama proses pengubahan kendali yang ketat harus dipekerjaan untuk memastikan bahwa perubahan program diminta, disetujui, ditetapkan, dikode, dites, didokumentan, dan diluncurkan untuk berproduksi. Sumber utama yang memungkinkan kendali ini adalah sistem manajemen berubah (CMS, change management system). CMS menyediakan manajemen yang kaya informasi agar memungkinkan manajemen untuk mengevaluasi dan mengoptimalkan aktivitas pemeliharaan sistem.

DAFTAR PERIKSA PEMELIHARAAN SISTEM

Berikut ini adalah daftar mengenai cara melakukan pemeliharaan sistem. Tujuannya adalah untuk mengingatkan anda mengenai bagaimana metode pemeliharaan sistem penting diaplikasikan.

1. Memahami bahwa pemeliharaan sistem akan meluas setelah beberapa tahun diimplementasi dan menjadi tahap yang paling memakan biaya dalam siklus hidup sistem itu.
2. Membandingkan empat jenis pemeliharaan sistem, yaitu: korektif, adaptatif, perfektif, dan preventif.
3. Menetapkan Siklus hidup pemeliharaan sistem (SMLC, System pemeliharaan Life Cyclic).

4. Menginstall metode-metode yang akan memastikan pengembangan sistem yang dapat dimaintain pada tempat pertama.
5. Mengorganisir pemeliharaan sistem, dengan menggunakan salah satu dari tiga pendekatan: pemisahan, gabungan, atau fungsional.
6. Untuk pemeliharaan sistem warisan, mempekerjakan CASE tools spesial.
7. Menginstall sarana yang kuat untuk mengarahkan pemeliharaan sistem. Mengarahkan pemeliharaan sistem seharusnya mencakup suatu cara untuk mengawali permintaan pemeliharaan, semisal formulir WO pemeliharaan yang didukung oleh sistem Help-desk.
8. Mengimplementasi CMS.

SOAL TINJAUAN

- 6.1. Apakah tahap terpanjang dalam siklus hidup sistem? Umumnya, seberapa banyakkah anggaran sistem keseluruhan yang dialokasikan ke pemeliharaan sistem?
- 6.2. Apa sajakah dua sumber utama untuk melakukan pemeliharaan sistem?
- 6.3. Sebutkan dan terangkan dengan singkat empat jenis pemeliharaan.
- 6.4. Jelaskan mengapa pemeliharaan korektif dapat dianggap sebagai pemeliharaan buruk! Apa yang umumnya menyebabkan perlunya pemeliharaan korektif?
- 6.5. Berikan contoh suatu pemeliharaan adaptatif. Apakah pemeliharaan adaptatif penting dan bernilai tinggi? Apakah penyebab munculnya terlalu banyak pemeliharaan adaptatif?
- 6.6. Apakah tujuan pemeliharaan perfektif?
- 6.7. Apakah tujuan pemeliharaan preventif?
- 6.8. Sebutkan dan terangkan dengan singkat kegiatan-kegiatan proses pemeliharaan sistem.
- 6.9. Sebuah perusahaan telepon, setelah pengubahan pemeliharaan, menjumpai kesalahan dalam program tagihan untuk panggilan jarak jauh. Ketika kesalahan itu ditemukan dan dikoreksi, perusahaan tidak dapat menutup kerugian sebanyak \$35 juta karena tagihan macet. Jelaskan bagaimana problem ini seharusnya dapat dihindarkan.
- 6.10. Apakah type utama pemeliharaan sistem yang digunakan dalam pemeliharaan perangkat keras?

- 6.11. Jelaskan mengapa MTTR mengukur efisiensi pemeliharaan, dan MTBF mengukur efektivitas pemeliharaan.
- 6.12. Sebutkan langkah-langkahnya, dan jelaskan dengan singkat bagaimana perancangan untuk mencapai tingkat maintainabilitas tinggi dapat dicapai.
- 6.13. Jabarkan dengan singkat tiga cara untuk mengorganisir proses pemeliharaan sistem.
- 6.14. Bedakan antara rekayasa maju dengan rekayasa mundur.
- 6.15. Jelaskan proses rekayasa ulang. Apakah tujuannya?
- 6.16. Jabarkan proses restrukturisasi. Apakah tujuannya?
- 6.17. Apakah tujuan sistem pakar pemeliharaan? Sebutkan keuntungan-keuntungannya.
- 6.18. Bagaimana pemeliharaan sistem terjadwal biasanya dijalankan?
- 6.19. Siapa yang umumnya mengawali pemeliharaan sistem tidak terjadwal?
- 6.20. Jelaskan mengapa pemeriksaan pasca implementasi biasanya memicu penjalanan beberapa pemeliharaan pertama.
- 6.21. Sebutkan dan jelaskan tujuan dari setiap elemen dalam formulir WO pemeliharaan
- 6.22. Jelaskan bagaimana formulir WO pemeliharaan membantu mengoptimalkan pemeliharaan sistem.
- 6.23. Jabarkan CMS dan komponennya.
- 6.24. Apakah tujuan CMS?

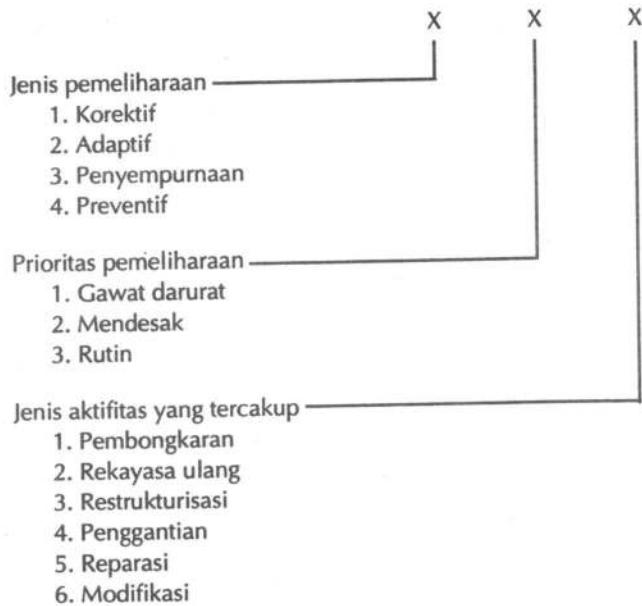
SOAL SPESIFIK BAB INI

Soal-soal ini memerlukan jawaban atau tanggapan pasti yang didasarkan langsung pada konsep-konsep dan teknik yang disajikan dalam bab ini.

- 6.25. Perusahaan anda menggunakan struktur pengkodean untuk pelayanan pemeliharaan tak terjadwal sebagaimana terlihat pada diagram kode di bawah ini:

Tetapkan suatu kode berdasarkan struktur pengkodean pemeliharaan itu untuk situasi pemeliharaan sebagai berikut :

1. Sistem pemasukan order telah beroperasi selama 18 bulan. Selama periode ini garis produk perusahaan telah bertambah luas, dua gudang baru telah didirikan, dan tiga kantor penjualan telah dibuka di Kanada. Selain itu para pengguna telah menemukan bahwa



laporan yang menunjukkan pesanan mundur (back-order) nilainya akan lebih besar bila mencapai mejanya sebelum tengah hari dari-pada bila tiba pada jam 4 sore. Pengguna utama sistem baru akan siap pada hari Senin untuk menganalisis sistem baru.

2. Anda sebagai analis pemeliharaan untuk sistem penggajian baru saja menerima memo dari wakil manajer Humas. Dia telah meminta informasi jumlah gaji terkecil tahun ini. Mmo tersebut lebih jauh menyatakan bahwa mulai sekarang informasi ini akan dibutuhkan berdasarkan triwulanan.
3. Begitu anda meninggalkan lapangan tennis, anda menerima pesan yang meminta anda untuk segera menelepon bagian operasi. Nam-paknya program analisis penjualan, yang telah berjalan lancar se-lama enam bulan, baru saja tiba-tiba berhenti.
4. Staf akuntansi baru saja menerima monitor VGA-nya. Staf akuntansi menginginkan monitor itu diinstall untuk menggantikan monitor monochrome mereka.
5. Suatu aplikasi inventaris dikembangkan 15 tahun yang lalu tanpa menggunakan pendekatan terstruktur. Dokumentasinya brupa sketsa dan kadaluwarsa. Para pengguna puas dengan fungsionalitas

aplikasi tersebut, namun mereka telah meminta agar field deskripsiya diperluas. Supervisor anda memerintahkan kepada anda untuk merestruktur dan sepenuhnya mendokumentkan sistem lama tersebut.

6. Jeff Travis, seorang operator pemasukan data, baru saja menelepon anda bahwa keyboardnya terkunci. Selanjutnya dia mengeluh bahwa perangkat tersebut tidak pernah berfungsi dengan baik, tuts-tuts nya lengket, dan sekarang tidak mau berfungsi samasekali.
7. Capstan pada drive pita telah membeku dan tidak mau berputar, sehingga menyebabkan pembacaan file master penggajian menjadi tidak mungkin. Pada saat itu jam 1.30 siang dan penggajian harus diproses pada jam 4.00 sore.
8. Berdasarkan spesifikasi yang diturunkan dari rekayasa mundur, diputuskan bahwa perangkat lunak yang ada harus dihentikan dan dikembangkanlah perangkat lunak baru.
9. Paket perangkat lunak rekening dapat diterima harus distruktur dan didokumenkan. Beberapa kode juga harus diubah untuk mencapai efisiensidan memodifikasi paket secara fungsional.

6.26. Ada tabel sebagai berikut :

```
01 CITIES
  05 FILLER    PIC X(10)      'BIGCITY'
  05 FILLER    PIC X(10)      'LITTLECITY'
  05 FILLER    PIC X(10)      'DREADVILLE'
  05 FILLER    PIC X(10)      'ROSYVILLE'
01 CITY-TABLE REDEFINES CITIES
  05 CITY OCCURS 4 TIMES    PIC X(10)
```

Nama kota apa yang akan diMOVE ke LINE-OUT (tidak terlihat) oleh pasangan pernyataan berikut ini.

```
MOVE 2 TO CITY-SUBSCRIPT
MOVE CITY(CITY-SUBSCRIPT) TO LINE-OUT
```

Latihan ini berisi suatu porsi program COBOL. Anggaplah anda mengetahui serba sedikit tentang COBOL. Dapatkah anda memahami kode ini? Seandainya anda seorang programmer yunior, dapatkah anda dengan cepat memahami kode itu dan dengan mudahnya membuat beberapa pengubahan penting?

6.27. Ada tabel sebagai berikut :

```

IF C1
  PERFORM P1
IF C2
  NEXT SENTENCE
ELSE
  PERFORM P2
  IF C3
    PERFORM P3
    PERFORM P4
  
```

Rekayasa-mundurkanlah kode itu dalam bentuk diagram aliran program terstruktur. Komentari dengan menggunakan nama-nama semacam C1, C2 dan seterusnya. Bekerakah nama setiap nilai ini bagi anda dalam memahami jenis aplikasi program? Apa yang akan menyebabkan program ini lebih mudah dipahami disamping diagram aliran program terstruktur?

- 6.28. Suatu program perangkat lunak yang ditulis 20 tahun yang lalu adalah tidak terstruktur dan tanpa dokumentasi. Anda telah menelusuri sumber kode tak bermakna dan mewawancarai seorang operator. Anda telah menyimpulkan bahwa program ini memproses file rekening dapat diterima dengan header dan trailer. File ini memuat N record, dan setiap record berisi pembayaran, faktur atau keduanya.

Ditanyakan: Sesuai catatan anda, buatlah rekayasa mundur dari program ini dalam bentuk diagram Warnier-Orr dan Jackson.

SOAL UMUM

Soal-soal ini lebih memerlukan pendekatan atau cara yang mudah dikerjakan dari pada solusi yang presisi. Meskipun soal-soalnya didasarkan pada materi pada bab ini, diperlukan bacaan tambahan dan kreatifitas ekstra untuk mengembangkan solusi yang dapat bekerja atau layak.

- 6.29. Seorang manajer di Nept Company telah membayar anda untuk membantu memimpin pengubahan yang dibuat pada program. Dia mengungkapkan garis besar problem-problem berikut ini:

- ◆ **Kurangnya Inventaris program uang akurat.** Tidak seorang pun pada sistem informasi mengetahui secara persis program-program yang telah disetujui untuk digunakan, program apa

yang benar-benar digunakan, dimana program ditempatkan, dan program mana yang dijadwalkan untuk diubah.

- ◆ **Tidak lengkapnya sejarah pengubahan program.** Tidak seorang pun memiliki sejarah yang lengkap mengenai siapa yang meminta atau mengesahkan pengubahan, program apa yang telah diubah, siap yang mengubah dan mengapa diubah.
- ◆ **Abend Program.** Seorang menghabiskan waktu berjam-jam memeriksa daftar program dan mencoba menemukan apa yang salah, hanya untuk menemukan bahwa kode sumber yang dilihat si programmer bukanlah versi yang aslinya digunakan untuk membuat modul muatan yang sekarang digunakan. Sumber kode sebenarnya berada di file pribadi seseorang, atau rusak, semuanya hilang.
- ◆ **Modul Program terduplikasi.** Sejumlah program dapat menggunakan modul yang sama untuk proses-proses standar. Seringkali modul yang melakukan proses standar ini diduplikasi bukannya dikembangkan sekaligus dan digunakan kembali. Nampaknya, duplikasi ini menyia-nyiakan baik personel maupun sumber-sumber yang mengkomputasi. Demikian pula, modul yang digunakan bersama dapat diubah untuk memenuhi kebutuhan sebuah program hanya untuk mengacaukan operasi program lain.
- ◆ **Pengubahan tidak sah.** Apapun alasannya beberapa orang mungkin membuat pengubahan tidak sah pada program, produksi. pengubahan semacam ini mungkin tidak salah. pengubahan yang lainnya mungkin suatu kecurangan, destruktif, atau pekerjaan "programmer tengah malam" yang bereputasi sangat buruk.
- ◆ **Kurangnya dokumentasi dan Testing.** Penekanan untuk memasang program dapat mengurangi dokumentasi program yang efektif dan akan mendorong penggunaan program sebelum program itu benar-benar diuji.
- ◆ **Ketidakmampuan untuk mengundurkan pengubahan.** Apa yang terjadi bila kode yang direvisi gagal? Akankah si programmer mampu meretrieve versi program yang lebih awal? Merekonstruksi file asli akan mustahil.

Ditanyakan: Jelaskan ke manajer betapa CMS dapat membantu memecahkan setiap problem.

- 6.30. Marsha Colvin, salah satu programmer pemeliharaan anda yang lebih berpengalaman dan efektif, telah meminta untuk bertemu guna membahas kemungkinan transfer ke bidang pengembangan sistem.

"Saya ingin pindah ke bidang pengembangan karena saya letih menertibkan kekacauan yang dibuat oleh orang lain. Dan saya ingin pindah ke pengembangan karena kesempatan untuk berhadapan dengan top manajemen lebih besar sehingga saya akan lebih cepat maju," dia mengawali.

"Kemajuan saya di sini cukup bagus, dan saya tidak mempunyai masalah dengan anda dan orang-orang di bagian ini. Cuma, nampaknya saya akan mampu melakukan suatu hal yang lebih penting, menurut saya, di bidang pengembangan. Maksud saya, adalah dimana ada tindakan, dan dimana mereka semua menciptakan sistem baru yang benar-benar akan menyumbang kepada perusahaan," lanjut Marsha." Disamping itu, pengembangan bekerja dengan perangkat dan bahasa baru. Tidaklah begitu menyenangkan kalau terus terikat di pemeliharaan dan harus bekerja dengan perangkat generasi kedua dan perangkat yang ketinggalan jaman."

Ditanyakan: Bagaimana anda menjawab Marsha untuk meyakinkan dia mengenai nilai fungsi pemeliharaan sistem dalam suatu perusahaan, dan peran penting yang dapat ia mainkan dalam fungsi itu? Anggaplah anda ditugaskan untuk mengepalai sistem informasi. Jenis program pemeliharaan sistem apa yang akan anda usulkan? Berilah alasannya!

BACAAN YANG DISARANKAN

- Carlyle, Ralph Emmett. "Fighting Corporate Amnesia." *Datamation*, February 1, 1989.
- Chapin, Ned. "Changes in Change Control." *Proceedings of the Conference on Software Maintenance 1989*. Washington, D.C.: Computer Society Press of the IEEE, 1989.
- Chapin, Ned. "The Job of Software Maintenance." *Proceedings of the Conference on Software Maintenance 1987*. Washington, D.C.: Computer Society Press of the IEEE, 1987.
- Chapin, Ned. "Software Maintenance Life Cycle." *Proceedings of the Conference on Software Maintenance 1988*. Waihington, D.C.: Computer Society Press of the IEEE, 1988.
- Corder, A. S. *Maintenance Management Techniques*. New York: McCraw-Hill, 1976.
- Dallimonti, Renzo. "Smarter Maintenance with Expert Systems." *Plant Engineering*. June 18, 1987.
- Foster, Gerald D., and Hien Van Tran. "Maintenance and Money." *Information Strategy: The Executive's Journal*, Spring 1990.
- Hanna, Mary Alice. "Defining the 'R' Words for Automated Maintenance." *Software Magazine*, May 1990.

- Heintzelman, J. *Complete Handbook of Maintenance Management*, Englewood Cliffs, N.J.: Prentice-Hall, 1976.
- Herbaly, Frank, *Cost-Effective Maintenance Management*. Park Ridge, N.J.: Noyes Publications, 1983.
- Higgins, David A. "Structured Maintenance: New Tools for Old Problems." *Computerworld*, June 15. 1981.
- Longstreet, David H, (Ed.). *Software Maintenance and Computers*, Los Alimitos, Calif.: Computer Society Press of the IEEE, 1990.
- Marek, Bill. *CA Librarian Change Control Facility: Source Management the 1990's and Beyond*. Phoenix, Ariz.: Computer Associates International, Inc., March 1990.
- Martin, James, "Restructuring Code Is a Sound Investment in the Future." *PC Week*, May 7, 1990.
- Martin, James, "Reverse-Engineering Gives Old Systems New Lease on Life." *PC Week*, April 16, 1990.
- Parikh, Girish (Ed.). *Techniques of Program and System Maintenance*, Boston: Little, Brown, 1982.
- Parikh, Girish, and Nicholas Zvezintzov. *Tutorial on Software Maintenance*. New York: Institute of Electrical and Electronics Engineers, Inc., 1983.
- Pressman, Roger S. *Software Engineering: A Practitioner's Approach*, 2nd ed. New York: McGraw-Hill, 1987.
- Rayl, Eric. "SupportMagic Tops Help-Desk Software." *PC Week*, February 4, 1991.
- Tayntor, Christine B. "Maintenance Magic." *System Builder*, October/November 1990.
- Walters, Roger E. *The Business Systems Development Process: A Management Perspective*, New York: Quorum Books, 1987.
- Weinberg, Gerald M., and Dennis P, Geller. *Computer Information Systems*. Boston: Little. Brown, 1985.

KASUS JOCS: Pemeliharaan Sistem

"Aku tidak tahu kau punya tenaga cadangan apa, tetapi yang jelas aku masih lelah," kata Carla Mills. "Kita sudah dua bulan mengimplementasi JOCS, dan aku masih mencoba mengejar ketinggalan pesanan-pesannya. Tak seorang pun memberitahuku orang-orang sistem bekerja 80 jam per minggu saat mereka menginstall sistem baru. Ini seperti bekerja dalam musim pajak seperti keika aku di akuntan publik."

"Ya, tetapi setelah sistem baru diinstall, orang-orang sistem hanya bekerja 20 jam per minggu sampai situasi kritis berikutnya muncul," Kata Christine Meyers. Dia tertawa. "Para akuntan memang cenderung siap bekerja sepanjang waktu."

"Aku suka pekerjaanku yang seperti roller-coaster ini," kata Cory Bassett. "Salah satu alasan aku memasuki profesi ini adalah rasa lega dan puas yang kita dapatkan ketika suatu pekerjaan selesai dikerjakan. Aku menyukai orientasi proyek kerja sistem."

"Tunggu sebentar," seru Jake Jacoby. "Siapa bilang kalau proyek ini sudah selesai? Kita mengakhiri bagian pengembangan sistem dari siklus ini. Sekarang kita harus

bekerja pada operasi sistem. Aku sedang merencanakan agar paket JOCS hidup sehat dan panjang umur. Menurut rencana strategis perusahaan JOCS harus hidup paling tidak selama 7 tahun dengan baik."

"Weleh-weleh, Jake," keluh Tom Pearson, "kita semua tahu bahwa pekerjaan pemeliharaan tidak memerlukan banyak tenaga. Bagaimana bila kita menugaskan orang baru untuk melakukan pekerjaan pemeliharaan sehingga kita dapat beralih ke pengembangan sistem pendukung keputusan finansial? Aku punya ide-ide untuk sistem baru yang ingin kubicarkan hari ini."

"Itukah yang kalian pikirkan mengenai tujuan rapat ini?" tanya Jake. "Membahas paket FIDS baru?"

JOCS, para anggota tim yang ingin menjadi mantan SWAT saling melempar pandang dan mengangguk. Beberapa komentar samar-samar semacam "JOCS bekerja dengan baik"; sekarang waktunya beralih"; "Kita telah melihat sistem FID baru"; dan "Aku pikir bahwa proyek FID barulah yang menjadi alasan mengapa aku pergi ke kelas sistem pakar" bersilangan diantara anggota kelompok.

Jake menggeleng dan berkata, "JOCS belum selesai dan belum dikerjakan. Bila sistem ini akan beroperasi selama tujuh tahun, kita harus mendukungnya selama tujuh tahun. Tujuan pertemuan ini adalah untuk menyusun rencana pemeliharaan jangka panjang untuk paket ini."

Jake melanjutkan: "Waktu usaha dan perencanaan yang kita taruh pada JOCS menyebabkannya tepat waktu dan didalam estimasi biaya kita. Aku tahu bahwa kita hanya memiliki sedikit masalah dan ini tidak mudah. Setiap orang di tim ini sebenarnya membanting tulang untuk menyelesaikan proyek ini. Maka aku menganggap bahwa kalian semua akan tertarik untuk memastikan agar JOCS terus berfungsi secara efektif pada jangka panjang."

"Tentu saja kita ingin JOCS hidup, Jake," kata Christine dengan manisnya. "Tetapi kita melakukan pekerjaan yang sedemikian baik, sehingga kuharap kita tidak memiliki kesalahan apapun yang perlu dikoreksi di masa mendatang."

"Christine," jawab Jake, "Jangan coba menenteramkan aku. Kau telah bekerja dalam sistem cukup lama untuk mengetahui bahwa pernyataan itu tidak pernah benar. Selalu ada cacat untuk dikoreksi. Tetapi meskipun kita tidak pernah harus melakukan suatu pemeliharaan korektif pada perangkat lunak JOCS, kita masih harus menambah peningkatan dan modifikasi pada paket ini. Tidak satupun perangkat lunak dapat memenuhi persyaratan bisnis selama tujuh tahun tanpa sejumlah substansial pemeliharaan adaptatif dan perfektif."

Cory masuk ke diskusi dengan, "Apa yang kau katakan, Jake, berarti sistem kita harus evolusioner bukannya statis."

"Tepat," Jake menyetujui. "Kita tidak berada dalam bisnis statis. Produk baru, personel baru, peraturan pemerintah baru, manajemen baru, pesaing baru; setiap salah satu faktor itu akan meminta kita untuk membantu JOCS berkembang. Kita semua mengetahui bahwa perangkat lunak tidak berkembang dengan sendirinya. Seseorang harus mengubahnya. Aku merasa bahwa ini bukan masalah populer pada tim saya, tetapi ijin saya mengungkapkan pada kalian apa yang sudah kumerjakan berkaitan dengan pemeliharaan sistem, dan apa yang harus kita bahas hari ini."

Jake mengeluarkan lembar kerja singkat yang mensketsa bidang pemeliharaan yang ingin ia bahas. (Gambar 6.9)

"Kita akan mulai dengan pemeliharaan perangkat keras karena, bagi kita, ini benar-benar merupakan bagian termudah. garansi perangkat keras pertama adalah 90 hari untuk semua komponen. Waktunya berakhir sewaktu tahap implementasi pengembangan sistem. Aku telah menandatangani kontrak dengan dua penyalur perangkat keras untuk menyediakan dukungan perangkat keras terus menerus untuk minikomputer, pelayan file dan jembatan. Dengan pembayaran bulanan, penyalur akan menyediakan semua suku cadang dan tenaga kerja yang perlu untuk mendukung minikomputer,

JOCS ONGOING SYSTEMS MAINTENANCE

- I. Hardware Maintenance
 - A. Vendor-supplied maintenance agreement
 - 1. LAN file servers
 - 2. Minicomputer
 - 3. Bridge hardware
 - B. On-call maintenance agreement
 - 1. Individual microcomputers
 - 2. Terminals
- II. Software Maintenance
 - A. Vendor-supplied maintenance agreement
 - 1. Operating system
 - 2. DBMS
 - 3. Bridge software
 - B. To be identified
 - 1. JOCS application software
 - 2. Data dictionary
- III. Documentation—To Be Identified
- IV. User Training—To Be Identified

Gambar 6.9. Garis besar lembar kerja pemeliharaan JOCS.

minikomputer, pelayan file dan jembatan. Dengan pembayaran bulanan, penyalur akan menyediakan semua suku cadang dan tenaga kerja yang perlu untuk mendukung sistem. Selain itu, penyalur melakukan fungsi pemeliharaan preventif satu pagi setiap bulan. Ini merupakan alternatif yang mahal, tetapi kukira pelayanan ini akan membantu kita menghindarkan problem hardware potensial dan akan menyelamatkan uang kita dalam jangka panjang. AKu mengkalkulasi biaya untuk perjanjian pemeliharaan perangkat keras untuk mikrokomputer dan bengkel kerja kita, dan menggambarkan bahwa kita akan dapat membeli mikrokomputer baru tiap tahun sebagai imbalan perjanjian pemeliharaan. Asuransi perjanjian pemeliharaan tidak usah membayar komponen itu, sehingga kita dapat bekerja dengan dukungan siap panggil untuk mereka."

"Aku juga telah membuat kesepakatan dengan setiap penyalur untuk menyuplai dukungan perangkat lunak terus menerus untuk sistem pengoperasian, DBMS, dan perangkat lunak jembatan kita. Dukungan ini akan meliputi pengupgradean otomatis pada terbitan baru, petunjuk baru, dan dukungan telepon yang tidak terbatas. Kita merupakan posisi tes beta untuk perangkat lunak jembatan, maka kuharap kita akan menerima dukungan yang baik dari penyalur itu.

"Sekarang kita sampai pada bidang yang lebih sulit. Kita harus mengembangkan metode untuk melakukan pemeliharaan JOCS. Ini meliputi koreksi, peningkatan, modifikasi, dan semua hal yang kita putuskan untuk menambahnya di masa depan untuk mencegah problem-problem perangkat lunak. Sekarang, Tom, sebelum kau mengatakan sesuatu, kupikir anggota tim JOCS asli seharusnya bertanggungjawab mengenai tugas-tugas ini. Kita mengetahui perangkat lunaknya, spesifikasinya, dan dokumentasinya. Kita seharusnya mendukung sistem ini."

"Aku tidak akan berkata apa-apa," kata Tom. "Aku hanya tidak sependapat bila dikatakan bahwa pemeliharaan adalah hal yang sulit. Setiap orang seharusnya dapat memahami sistemnya."

"Tom," serghah Jake tajam, "kita memiliki lebih dari 25.000 saluran kode COBOL, sebuah antar muka DBMS, antarmuka antara sistem pembuatan terpadu komputer dengan JOCS, dan masih menggunakan beberapa terminal khusus yang manis. Dapatkan kalian berjalan di pintu perusahaan ini dan memahami jenis sistemnya?"

"Aku punya usul, Jake," kata Christine. "Bagaimana bila kita mengambil giliran memaintain perangkat lunaknya? Kamu dapat menugaskan seorang anggota tim JOCS SWAT pada proyek itu selama enam bulan, dan juga menugaskan orang yang tidak kenal dengan JOCS. Dua orang ini dapat bekerja bersama-sama untuk memaintain sistem. Setelah enam bulan alihkan anggota tim JOCS SWAT ini ke proyek yang lain sehingga dia tetap ditantang dengan bekerja pada sistem yang baru. Kamu dapat memutar orang baru pada jadwal yang berbeda."

"Aku setuju," kata Tom. "Problem terbesar pada pemeliharaan adalah bahwa kebosanan akan melanda setelah beberapa lama. Aku tidak bermaksud meremehkan pemeliharaan. Cuma, ini akan menjadi keterlaluan kotornya sepanjang waktu. Bila kita menggunakan usul Christine, tak seorang pun akan secara permanen terkubur di balik proyek pemeliharaan."

"Kalian tahu," lanjut Tom, "meskipun demikian aku masih belum memahami mengapa harus dua orang yang bekerja pada pemeliharaan. Seorang saja kukira cukuplah itu."

Carla, hampir tidak kuat menahan komentarnya, memandang Tom dan berkata, "Kau pasti bergurau, Tom. Dengan semua giliran yang kita punya di akuntansi, salah seorang akan selalu sibuk hanya untuk memimpin kelas-kelas latihan."

"Ya," kata Cory berusaha menghindari perdebatan, "kupikir kita juga harus membicarakan tentang pelatihan. Kita harus menyusun program pelatihan terus menerus untuk generator laporan DBMS maupun Sintaksis bahasa query, disamping penggunaan JOCS. Mungkin orang-orang yang ditugaskan pada pemeliharaan dapat menggunakan minggu-minggu permulaan untuk membuat dan menyusun program pelatihan yang terus menerus berjalan.

"Kedengarannya itu ide yang baik," kata Jake. "Setelah kita membuat tim pemeliharaan, tim ini dapat membuat prosedur dokumentasi dan pelatihan seperti dua tugas pertamanya."

"Alangkah leganya," seru Jannis Court. "Aku selalu berdiam diri karena aku tidak dapat menggambarkan apa yang kalian pikirkan mengenai tim pemeliharaan selama enam bulan bulan pertama waktu pemeliharaan. Maksudku, sistemnya sudah diinstall dan bekerja, dan aku tidak dapat membayangkan bahwa kita akan mengalami peningkatan pada bulan-bulan pertama ini."

"Kukira kau benar, Jannis," kata Jake. "Kita telah menghabiskan waktu dua bulan terakhir ini untuk meluncurkan sistem JOCS dan mengoreksi cacat-cacat yang nampak. Aku ingin tim pemeliharaan menyediakan metode dokumentasi berjalan, identifikasi dan reparasi masalah, dan pelatihan semasa bulan-bulan pertama yang relatif mudah."

"Naahh," kata Jannis, "Aku mau mempelopori bekerja di proyek itu. Aku punya beberapa ide mengenai cara mengerjakan fungsi-fungsi itu, dan aku sangat suka mengembangkan prosedur awal. Sejak aku keluar dari kelas sistem pakarku, aku telah berangan-angan untuk membuat sistem pakar untuk melakukan fungsi pemeliharaan. Aku ingin mengambil suatu "shell" sistem pakar dan melihat bahwa teoriku terbukti. Kupikir aku dapat merancang sebuah sistem pakar untuk membantu kita mengidentifikasi cacat-cacat program, selain juga menempatkan penempatan program pada peningkatan potensial. Misalnya, salah satu tujuanku adalah dapat memasukkan jenis peningkata."

peningkatan yang ingin kita kerjakan, mengatakan sesuatu seperti menambahkan jenis baru dari transaksi, dan sistem pakar akan memberitahu kita tempat yang tepat untuk dimodifikasi dalam sistem itu."

"Wow, ini benar-benar menarik," kata Cory. "Aku mengetahui tentang hal itu ketika kuliah, tetapi kita tidak pernah memiliki waktu untuk menyelesaikan proyek sistem pakar yang sebenarnya. Jake, bila tidak ada pekerjaan baru untukku pada saat ini, aku ingin bekerja dengan Jannis di dalam proyek ini."

"OK, Cory! Naahh," kata Jake, "nampaknya kita sudah menempatkan giliran pemeliharaan pertama kita. Aku terkesan dengan konsep sistem pakar pemeliharaan, Jannis. Buatlah rincian biayanya, dan sampaikan kepadaku dalam bentuk surat E yang merinci apa yang akan kau butuhkan untuk mengerjakan prototype untuk proyek itu. Kupikir kita akan menemukan dana untuk mendukung pengembangan sistem pemeliharaan yang baik." Dia tertawa dan berkata, "Itu tadi kalimat yang agak janggal. Kukira kita telah sampai satu putaran penuh pada poin ini. Maksudku, inilah kita yang sedang membangun sistem pemeliharaan kita. Mungkin beberapa developer sistem selalu mengembangkan sistem, tidak perduli bila kita menyebut sistem tersebut baru atau pun lama."