BAB 9
FASE PEMROGRAMAN

9.1. PENDAHULUAN

Pemrograman adalah merupakan bagian yang paling mudah – itulah yang kita sangat kenal sebagai tipe-tipe teknik. Pada kenyataannya, sebagai Manajer Proyek anda mungkin menemukan cara sendiri untuk mengendalikan staf anda dari memulai program yang terlalu cepat. Selalu ada tekanan untuk melakukan sesuatu dengan benar, tidak hanya dari tim proyek, tetapi dari manajemen tingkat tinggi.

Aktifitas-aktifitas pada fase ini adalah menulis program. Kejadian pentingnya adalah menguji program, Rencana Tes Sistem, dan paling tidak mulai pada Dokumentasi User.

9.2. DAFTAR PEMERIKSAAN PEMROGRAMAN (*PRE-PROGRAMMING CHECKLIST*)

Sebelum anda memulai pemrograman, jawablah pertanyaan berikut :

- Apakah pemeriksaan disain memerlukan pengerjaan kembali ? Jika ya, jadwalkan waktu mulai dan penundaan pemrograman.

- Apakah sumber daya yang direncanakan dan programmer masih tersedia ? Jangan terlalu optimis dengan proyek orang lain yang selesai tepat waktu. Jika ada pergantian staf, apakah anda akan menguji kembali produktifitas mereka ?

- Apakah orang-orang ini telah dilatih ? Programmer harus mengetahui tentang sistem operasi, bahasa pemrograman, program paket, dan alat-alat pemrograman yang akan digunakan. Mereka juga harus mengenal dengan baik aplikasi user dan masalah bisnis. Pastikan bahwa mereka telah membaca *Requirement Document dan Functional Specification*.

- Apakah lingkungan pemrograman cukup baik ? Anda membutuhkan kemudahan untuk menggunakan software pengembangan dan alat-alat pemrograman. (Lihat bagian 9.4). Komputer pengembangan harus mempunyai respon yang cepat, harus tersedia ketika diperlukan, dan harus dapat diandalkan.

9.3. LANGKAH-LANGKAH PEMROGRAMAN (*THE PROGRAMMING STEPS*)

Langkah 1. Rencana Penggabungan (*Plan The Integration*)

Menurut akal sehat anda tidak akan dapat membuat semua program sekaligus dan kemudian membuang semuanya – ini memerlukan rangkaian langkah demi langkah. Rencanakan urutan dimana anda akan menggabungkannya. Bab 9 ini merinci beberapa metode untuk menggabungkan bagian-bagian tersebut, tetapi anda harus merencanakan urutan penggabungan ini sekarang, karena anda harus menulis program supaya dapat digabungkan. Ini disebut Rencana Tes Sistem (*System Test Plan*).

Langkah 2. Mendisain Modul (*Design The Module*)

Programmer menerima beberapa tingkatan disain dari fase disain. Tugasnya adalah memecah modul secara rinci ke tingkat yang lebih rendah sampi mencapai keadaan programmer siap untuk melakukan pemrograman. Ini disebut disain modul. Level disain modul tingkat menengah dapat dilihat pada gambar 9.1.

Lihat Gambar 9.1. Medium level desin (3rd level)

Programmer menerima penjelasan modul dari perancang seperti berikut ini.

Lihat Gambar penjelasan modul

Programmer pertama-tama menggambarkan diagram struktur dari modul. Terlihat seperti pada gambar 9.2.

Lihat Gambar 9.2. Fourth level module breakout

Disain modul adalah pendekatan *top-down* dimulai dengan kotak paling atas, AMST000 dan dipecah ke dalam sub-komponen yang tepat, seperti pada gambar 9.3.

Lihat Gambar 9.3. Fifth level module breakout

Kemudian setiap sub-komponen dapat dibagi lagi, seperti pada gambar 9.4.

Lihat Gambar 9.4. Sixth level module breakout

Dan kemudian modul tersebut dipecah lagi sampai tercapai sebuah tingkatan dimana mulai dapat diprogram.

Pertanyaan yang bisa diajukan adalah : “Pada tingkatan mana disain sistem berhenti dan disain modul dimulai ?”. Jawabannya adalah “Disain sistem dipecah sampai pada tingkat dimana programmer dapat memulainya”. Tingkatan ini dapat bermacam-macam dari proyek ke proyek dan bahkan dari satu bagian sistem ke bagian lainnya – tergantung pada programmer yang menerima bagian tersebut.

Adapun pertimbangan lainnya adalah :

- Jika pemecahan modul yang dihasilkan adalah sangat penting yang memerlukan prioritas seperti adanya *respon*, *user-friendly* atau konsistensi, perancang bisa melanjutkan ke tingkat yang lebih rendah.

- Tingkat pemecahan dari disain dinyatakan dengan kontrak.

- Jika programmer tidak mengetahui pada waktu disain, pengetahuan programmer tingkat menengah dapat diasumsikan, dan disain dapat diambil alih oleh programmer tingkat menengah yang dapat mengatasinya.

Tetapi perlu diingat bahwa programmer tidak senang menerima disain yang terlalu rinci, yang programnya adalah menerjemahkan bahasa Inggris yang sederhana, seperti pernyataan-pernyataan secara harafiah ke dalam bahasa pemrograman.

Langkah 3. Telusuri Disain Modul (*Walk Through The Module Design*)

Seperti pada tingkat atas dan menengah dari disain, pertukaran harus dibuat sebaiknya pada tingkat yang paling rendah. Telusuri disain dari masing-masing modul sebelum melakukan pengkodean. Penelusuran ini sangat kecil : hanya programmer yang tepat, supervisor dan mungkin programmer lainnya yang perlu diperhatikan. Kegunaan dari penelusuran disain modul adalah untuk memastikan bahwa disain yang terbaik yang telah dilakukan, semua fungsi telah dialamatkan dan semua bagian telah ditangani.

Langkah 4. Rencana Bagaimana Menguji Modul (*Plan How To Test The Module*)

Programmer harus menyiapkan rencana pengujian modul dan data pengujian sebelum dikodekan. Rencana pengujian dilakukan setelah kode ditetapkan. Mereka cenderung hanya menguji bagian kode yang paling ‘sulit’. Pimpinan proyek bisa saja melakukan tuntutan pada penelusuran rencana pengujian sepanjang disain modul sedang dilaksanakan. Kerjakan penelusuran ini bersama-sama.

Langkah 5. Kode Setiap Modul (*Code Each Module*)

Standar pengkodean akan ditetapkan pada saat disain sistem (lihat bagian 7.12). Kita tidak membahas bagaimana membuat program – lihat Referensi 12 (tulisan ini membahas disain sama baiknya dengan pemrograman) dan Referensi 13 untuk lebih jelasnya.

Berikut ini adalah ringkasan dari sebuah program terstruktur, yaitu :

- Jika berukuran kecil. Aturan dasarnya adalah kira-kira 100 baris kode yang dapat dieksekusi dan *listing*nya tidak lebih dari 2 halaman.
- Satu *entry*, satu *exit*.
- Referensi secara keseluruhan sedikit.
- Konstruksi terstruktur yang digunakan : berurutan, IF/THEN/ELSE, CASE, WHILE, UNTIL, CALL (bukan GO TO).

Langkah 6. Menguji Modul (*Test The Module*)

Programmer menguji modul dengan menetapkan lingkungan yang tepat, menyediakan beberapa input, membiarkan modul langsung memproses secara logik dan mendapatkan hasilnya. Beberapa input mungkin tidak sebenarnya, terutama jika modul tersebut tidak menyediakan input yang sebenarnya.

Modul seharusnya diuji dalam dua tahap, yaitu :

- **Tahap Pertama** disebut pengujian "*White Box*". Programmer harus mengetahui isi di dalam modul dan menyediakan data pengujian, sehingga masing-masing *path logical* dalam program dapat dieksekusi.

- **Tahap Kedua** atau pengujian "*Black Box*" dapat dilakukan. Dalam pengujian ini, programmer mengabaikan bagian dalam dari modul – data disediakan secara berurut dan dianggap seperti pemakaian sebenarnya.

Langkah 7. Menguji Level Terendah dari Integrasi (*Test The Lowest Levels Of Integration*)

Jika modul utama memanggil sub-modul, programmer harus menggabungkan dan menguji semua modul secara bersama-sama. Bahkan jika programmer tidak bertanggung jawab untuk menulis sub-modul, programmer harus menguji perintah CALL dan RETURN dari seluruh modul.

Metode terbaik untuk melakukan hal ini adalah membuat sebuah "*program stub*" (potongan program) sebagai pengganti sub-modul. Potongan program ini dapat terdiri dari empat baris program yang menunjukkan bahwa kontrol sudah diterima dengan baik, tampilkan parameter penerima, jika perlu lakukan pengontrolan kembali dengan beberapa parameter yang tidak sebenarnya.

Langkah 8. Menyimpan Semua Hasil Pengujian; Penggabungan Modul-modul Yang Telah Diuji (*Save The Results Of All Tests; Submit Finished Modules To Integration*)

Hasil pengujian digunakan untuk menyusun statistik yang menunjukkan penyebab, cara perbaikan serta biaya-biaya yang dibutuhkan untuk memperbaiki kesalahan-kesalahan program. Pimpinan proyek biasanya menguasai/mengepalai penggabungan ini pada sistem berukuran kecil sampai sedang.

Software seperti CMS (*Code Management System*) sangat berguna untuk menajemen konfigurasi – menjamin program tetap berjalan sesuai versinya dan mengubah ke *source code* (lihat bagian 9.4).

Langkah 9. Memulai Dokumentasi User (*Get Started On The User Documentation*)

Apakah programmer bertanggung jawab pada dokumentasi user atau tidak, tahapan ini adalah waktu terbaik untuk menjawabnya.

Dokumen-dokumen berikut mungkin harus ditulis :

Tuntunan Pemakai (*User's Guide*)

Dokumen ini dapat ditulis oleh programmer, penulis teknis atau bahkan user sendiri. Tampilkan kembali FS yang mempunyai bagian rinci mengenai menu, layar, *form*, dan *user interface* lainnya.

USER'S GUIDE yang baik adalah terbagi dalam bagian-bagian yang menunjukkan tingkatan user yang berbeda-beda. Sebagai contoh, dalam USER'S GUIDE sistem ABC, harus ada bagian yang disebut "*Registrar's Functions*" atau "*Warehouse Functions*" atau lainnya. Materinya harus disesuaikan agar user dapat menggunakan secara normal. Hal ini membuat USER'S GUIDE berguna untuk mempelajari sistem.

Urutan popular lainnya untuk USER'S GUIDE adalah menelusuri menu-menu perintah secara logika. Pada akhir dari USER'S GUIDE ini disediakan referensi dari setiap perintah, menu, *form* dan pesan yang ditampilkan secara alphabet.

Tuntunan Pemeliharaan (*Maintenance Guide*)

Bagaimana anda menemukan programmer untuk merinci dokumen dari program mereka untuk pemeliharaan berikutnya ? Kebanyakan Manajer proyek mengalami kesulitan dalam hal berikut : programmer enggan untuk melakukan dokumentasi sebelum program ditulis; dan beruntunglah menemukannya setelah semuanya selesai dikerjakan. Programmer berpikir bahwa pemeliharaan memerlukan penjelasan secara rinci dari logika pemrograman. Sangat membosankan untuk menulisnya dan sebenarnya tidak perlu.

Berikut ini adalah solusi sederhana tentang hal tersebut : lebih baik merinci spesifikasi disain tingkat modul secara struktur, mendokumentasikan sendiri kode, dirasa cukup untuk pemeliharaan sistem.

MAINTENANCE GUIDE akan berisi spesifikasi disain, *listing* program dan penjelasan bagaimana semuanya disesuaikan, bagaimana mengubah pendekatan, dan bagaimana menghubungkan dan menguji semuanya.

Tuntunan Operator / Tuntunan Manajer Sistem (*Operator's Guide / System Manager's Guide*)

Sama seperti USER'S GUIDE untuk orang-orang yang menghidupkan sistem di pagi hari, mematikannya, melakukan *backup*, menangani permasalahan utama, melakukan perhitungan, dsb. Dokumentasi yang disediakan oleh perusahaan hardware dan sistem operasi mungkin cukup – hanya prosedur untuk software tertentu yang harus ditulis ulang.

Dokumentasi Pelatihan (*Training Documentation*)

Jika anda akan memberikan kursus bagaimana menggunakan sistem, rencanakan apakah materi pelatihan akan diperlukan. USER'S GUIDE yang baik harusnya menambahkan hal ini. Anda mungkin harus membuat bantuan pelatihan, seperti transaparansi, buku latihan, pengujian, dsb.

9.4. PERALATAN PROGRAM (*PROGRAMMING CASE TOOLS*)

Berikut ini adalah produk software yang membantu programmer untuk melakukan pekerjaanya dengan lebih baik. Software ini disebut CASE (*Computer Aided Software Engineering*), karena membantu proses pemrograman secara otomatis. Lihar Referensi 2.1. untuk produk tersebut.

Bahasa Pemrograman (*The Programming Language*)

Bahasa pemrograman dan *compiler* adalah alat yang sangat penting. Jika bahasa pemrograman itu sederhana dan sesuai dengan aplikasinya programmer akan dapat mempelajarinya dengan cepat, gunakan jenis yang diperlukan secara tetap, dan lakukan

pemrograman tanpa canggung. *Compiler* harus cepat dan jelas dalam menuliskan pesan kesalahan.

Language Sensitive Editor (LSE)

LSE menyediakan *template* untuk setiap pernyataan dalam bahasa pemrograman. Sebagai contoh, dalam bahasa PASCAL, user dapat mengetikkan 'FOR' dan LSE menghasilkan :

```
FOR % {ctrl-var}% := %{exp}% %{TO | DOWNTO}% %{exp}% DO
        %{statemnets}%
END;
```

Programmer mengisikan variabelnya dan LSE memastikan sintaksnya benar. LSE juga dapat memanggil *compiler*. Jika ada kesalahan yang ditemukan oleh *compiler,* LSE dapat mengontrol kembali, dan programmer dapat kembali ke posisi *edit* – pada pesan dan baris kesalahan tersebut. LSE dapat membuat *program header* dari *template*. LSE membantu dalam pemeriksaan sintaks, kompilasi program dan memastikan *source format* yang konsisten pada sistem.

Pendeteksi (*Debugger*)

Debugger membantu memeriksa dan memperbaiki kesalahan program. *Debugger* dapat memberhentikan program, menelusuri kesalahan dan memeriksa kesalahan berikutnya. *Debugger* yang baik dapat menyesuaikan dan menampilkan variabel pada semua titik, seperti pada pengeksekusian bagian spesifik dari program.

Code Management System (CMS)

Seringkali disebut manajer konfigurasi, CMS tidak tersedia untuk semua bahasa pemrograman. CMS adalah 'perpustakaan' yang memiliki sendiri semua sumbernya. CMS menertibkan orang–orang yang melakukan *update* dan memastikan tidak terjadi konflik, jika dua orang meng-*update* modul yang sama pada saat yang bersamaan. CMS menyimpan segala perubahan yang terjadi pada modul, sehingga segala perubahan pada modul dapat mudah dilihat.

CMS menunjukkan adanya perbaikan atau penambahan yang mudah dengan versi-versi sebelumnya.

CMS dapat menangani semua file ASCII. Oleh karena itu CMS berguna tidak hanya untuk menelusuri file-file sumber, tapi juga untuk menyimpan file dokumentasi, file pengujian, dan file-file untuk membangun sistem.

Module Management System (MMS)

MMS digunakan untuk proses *compile* dan *link* secara otomatis atau membangun sebuah sistem. MMS hanya dapat membangun kembali semua komponen tersebut yang diubah sejak pembangunan yang terakhir. MMS dapat digunakan untuk menjalankan secara otomatis sekumpulan pengujian modul. MMS sangat berguna ketika anda membangun sebuah 'release' sistem : menyatukan sumber-sumber yang benar dan meng-eksekusi *image,* seperti seluruh dokumen yang terdapat dalam satu paket. MMS bekerja *hand-in-hand* dengan CMS dimana semua sumber, file-file dokumen dan file-file perintah yang diproses MMS dapat disimpan.

Test Manager (TM)

TM digunakan untuk menguji sebuah modul secara otomatis. Untuk menggunakan TM, anda harus mendefinisikan serangkaian pengujian terhadap modul. TM akan menjalankan pengujian, dan memberitahukan *programmer* jika hasilnya bebeda dengan yang diharapkan.

9.5. HAK CIPTA (*COPYRIGHTS*)

Subyek dari hak cipta software adalah tetap pada pengadilan, tetapi terdapat peraturan pemerintah yang tidak hanya merupakan bagian dari software yang memiliki hak cipta, tetapi juga berkenaan dengan hal-hal lain yang berhubungan dengan software (apapun tujuan dan artinya). Jika anda ingin melindungi kode anda, tambahkan

pemberitahuan hak cipta pada masing-masing modul dan dokumen asli. “Copyright © 20nn, Company Name” yang biasanya diperlukan.

9.6. KESIMPULAN DARI FASE PEMROGRAMAN

Berikut ini beberapa pemikiran tentang pemrograman :

Pemrograman telah menjadi sebuah karya seni. *Programmer* diperbolehkan untuk “mengerjakan segala sesuatunya sendiri”. Hal tersebut sangat cepat ditemukan dan sangat mahal untuk melaksanakan proses tersebut. Pemrograman haruslah dipertimbangkan sebagai sebuah ilmu pengetahuan.

Pemrograman adalah kesenangan tetapi *debugging* bukanlah kesenangan. Perhatikan pernyataan seperti “Pengkodean telah dilakukan, semua yang *debug* dihilangkan, sehingga 90% telah dikerjakan”. Data statistik menunjukkan bahwa programmer hanya 50% berhasil setelah pengkodean.

Berikut ini beberapa pemikiran tentang programmer :

Programmer selalu meremehkan tugas. Mengajari mereka pesimis merupakan hal yang salah.

Programmer akan menikmati pekerjaan mereka jika anda memotivasi mereka dengan sebuah tantangan. Setiap tugas harusnya lebih sulit atau berbeda dari sebelumnya. Jika anda ingin belajar bagaimana memotivasi programmer, bacalah buku G.Weinberg, “The Psychology of Computer Programming” (Referensi 14).

Programmer sangat mudah tertekan, mereka akan bekerja lembur jika diperlukan. Tetapi hati-hati terhadap lembur yang tetap. Sewaktu-waktu tidak ada lagi produktifitas ekstra yang diperoleh dan programmer akan kehabisan tenaga.

Pada akhir fase pemrograman, lihatlah hal utama berikut ini :

1. Disain modul sudah dijalankan dan diselesaikan.

2. Program-program individual sudah selesai di-kodekan, diuji dan diselesaikan oleh pimpinan proyek.

3. Susunan dari penggabungan telah ditentukan, ditulis dalam Rencana Pengujian Sistem (dan pemrograman telah dijalankan pada saat itu).

4. Tanggung jawab terhadap dokumentasi user telah diberikan, dan jika anda beruntung hal tersebut telah dilakukan.