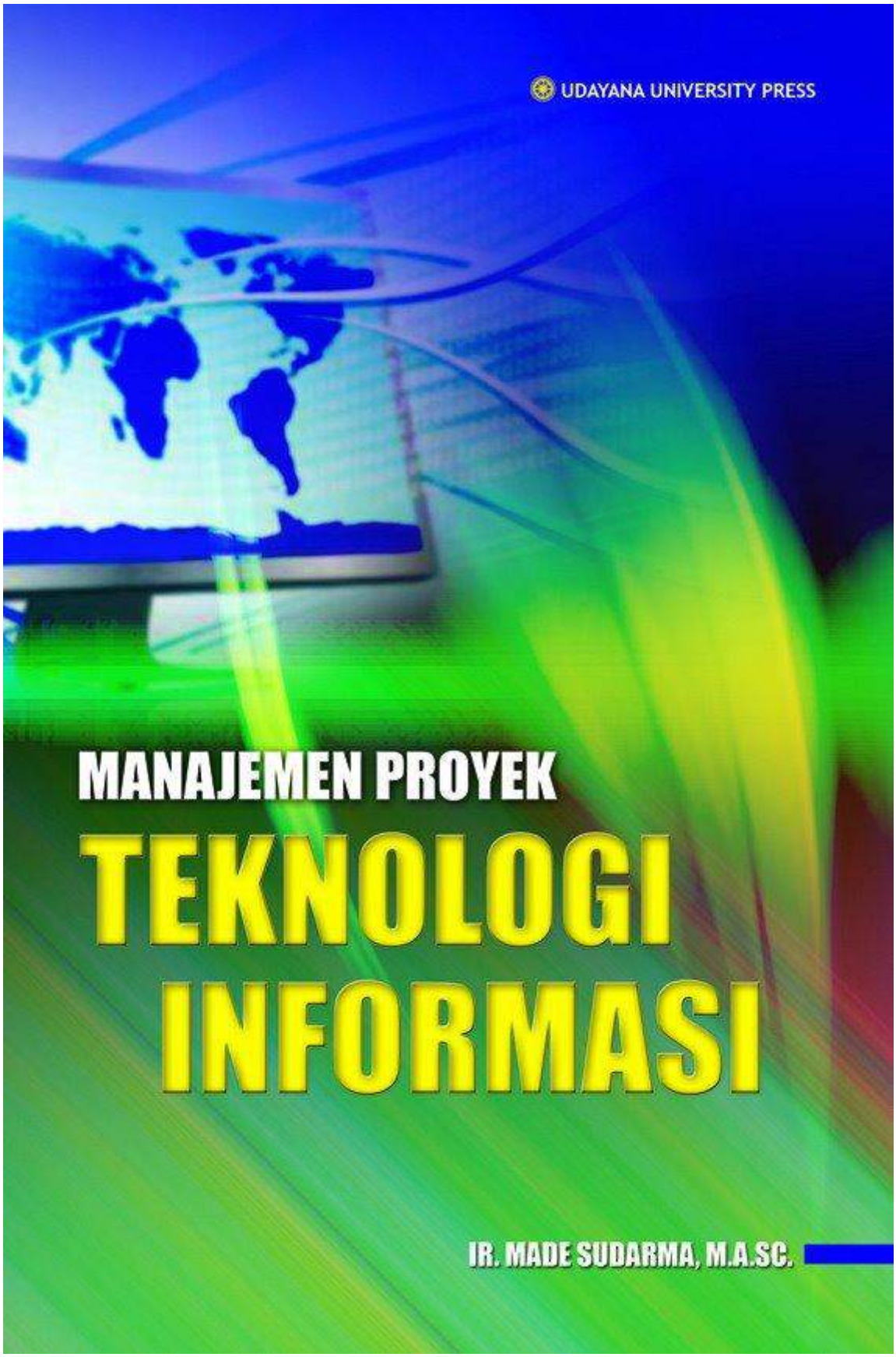

UDAYANA UNIVERSITY PRESS
MANAJEMEN PROYEK
TEKNOLOGI
INFORMASI
IR. MADE SUDARMA, M.A.SC.

MANAJEMEN PROYEK TEKNOLOGI INFORMASI

Undang-Undang Republik Indonesia Nomor 19 Tahun 2002 Tentang Hak Cipta

Lingkup Hak Cipta
Pasal 2

1. Hak Cipta merupakan hak eksklusif bagi Pencipta atau Pemegang Hak Cipta untuk mengumumkan atau memperbanyak Ciptaannya, yang timbul secara otomatis setelah suatu ciptaan dilahirkan tanpa mengurangi pembatasan menurut peraturan perundang-undangan yang berlaku.

Ketentuan Pidana
Pasal 72

1. Barang siapa dengan sengaja melanggar dan tanpa hak melakukan perbuatan sebagaimana dimaksud dalam Pasal 2 Ayat (1) atau Pasal 49 Ayat (1) dan Ayat (2) dipidana dengan penjara masing-masing paling singkat 1 (satu) bulan dan/atau denda paling sedikit Rp 1.000.000,00 (satu juta rupiah), atau pidana penjara paling lama 7 (tujuh) tahun dan/atau denda paling banyak Rp 5.000.000,00 (lima juta rupiah).
2. Barang siapa dengan sengaja menyiarkan, memamerkan, mengedarkan atau menjual kepada umum suatu ciptaan atau barang hasil pelanggaran hak cipta atau hak terbit sebagai dimaksud pada Ayat (1) dipidana dengan pidana penjara paling lama 5 (lima) tahun dan/atau denda paling banyak Rp. 500.000.000,00 (lima ratus juta rupiah).

MANAJEMEN PROYEK TEKNOLOGI INFORMASI

Ir. Made Sudarma, M.A.SC.

UDAYANA UNIVERSITY PRESS
2012

MANAJEMEN PROYEK EKNOLOGI INFORMASI

Penulis:
Ir. Made Sudarma, M.A.SC.

Penyunting:
Jiwa Atmaja

Cover & Ilustrasi:
Repro

Design & Lay Out:
Putu Mertadana

Diterbitkan oleh:
Udayana University Press
Kampus Universitas Udayana Denpasar
Jl. P.B. Sudirman, Denpasar - Bali, Telp. 0361 9173067, 255128
Fax. 0361 255128
Email: unudpress@yahoo.com http://penerbit.unud.ac.id

Cetakan Pertama:
2012, x + 310 hlm, 14 x 21 cm

ISBN: 978-602-9042-53-5

PRAKATA

Teknologi informasi (selanjutnya disebut *IT*) telah menjadi tolak ukur perkembangan peradaban manusia dalam era informasi saat ini. Mereka yang tinggal di kota-kota besar yang telah menikmati kehadiran teknologi modern, di manapun di belahan bumi ini, boleh dikatakan telah menggantungkan sebagian aktivitasnya kepada teknologi, terutama sebagai sarana untuk berkomunikasi, seperti: Internet, telepon, HP, GPS, jaringan komputer, dan lain-lainnya.

Fasilitas dan infrastruktur pendukung IT terus dikembangkan. Proyek-proyek IT bermunculan dimana-mana, antara lain: proyek pembuatan situs Internet, proyek pembenahan jaringan komputer kantor, proyek pembuatan perangkat lunak untuk aplikasi administrasi di suatu kantor dan lain-lainnya lagi.

Di dalam buku ini akan dibahas rangkaian alur kerja untuk mengelola proyek-proyek IT. Buku ajar ini sangat bermanfaat untuk membuka wawasan tentang bagaimana pengelolaan sebuah proyek. Langkah-langkah apa saja yang dibutuhkan, bagaimana pelaksanaannya, bagaimana mengontrol serta bagaimana penyelesaiannya. Sasaran pembaca yang diharapkan adalah para mahasiswa dan khalayak umum yang ingin mengenal dunia manajemen proyek secara global dan *IT* pada khususnya.

Kritik, saran dan koreksi sangat diharapkan untuk perbaikan buku ajar ini pada cetakan yang akan datang. Kesalahan ketik yang mengakibatkan kesalahan algoritmik pada buku ajar ini tak mungkin dikoreksi oleh kompilator.

Denpasar, Oktober 2011
Ir. Made Sudarma, MASc.

DAFTAR ISI

SEKILAS PANDANG

Selamat datang pada mata kuliah Manajemen Proyek *IT*.

Teknologi informasi (selanjutnya di dalam diktat ini disebut dengan: *IT*) telah menjadi tolak ukur perkembangan peradaban manusia dalam era informasi saat ini. Mereka yang tinggal di kota-kota besar (yang telah menikmati kehadiran teknologi modern), di manapun di belahan bumi ini, boleh dikatakan telah menggantungkan sebagian aktivitasnya kepada teknologi, terutama sebagai sarana untuk berkomunikasi, seperti: Internet, telepon, HP, GPS, jaringan komputer, dan lain-lainnya.

Fasilitas dan infrastruktur pendukung *IT* terus dikembangkan. Proyek-proyek *IT* bermunculan dimana-mana, antara lain: proyek pembuatan situs Internet, proyek pendirian menara komunikasi provider mobile phone, proyek pembenahan jaringan komputer kantor, proyek pembuatan perangkat lunak untuk aplikasi administrasi di suatu kantor dan lain-lainnya lagi.

Di dalam diktat kecil ini akan dibahas rangkaian alur kerja untuk mengelola proyek-proyek *IT*. Rangkaian alur kerja tersebut meliputi:

- pengenalan istilah proyek;
- batasan dalam pelaksanaan sebuah proyek;
- pendefinisian tujuan dan arah proyek;

- melakukan riset untuk proyek;
- cara menambah informasi melalui presentasi dan wawancara;
- menyusun kegiatan proyek dalam rangkaian aktivitas dan jaringan kerja;
- estimasi waktu pelaksanaan proyek. Secara khusus ditinjau dari sudut pengembangan perangkat lunak, karena proyek *IT* tidak akan terlepas dari pembuatan kode di dalam komputer;
- pengelolaan biaya, sumberdaya manusia dan non-manusia;
- pengelolaan risiko dan kualitas;
- serta tidak ketinggalan pembahasan mengenai perangkat lunak pendukung manajemen proyek, secara khusus *Microsoft Project.*

Pada bagian akhir diktat akan diberikan daftar pustaka dan referensi sumber-sumber lain yang dapat dipakai sebagai bahan untuk mendalami manajemen proyek.

Pada lampiran akan diberikan materi tambahan, yaitu: tutorial untuk memulai suatu proyek dengan menggunakan *Microsoft Project*, serta suatu gambaran tentang bagaimana pengelolaan sebuah proyek pengembangan situs *Internet*.

Diktat ini sangat bermanfaat untuk membuka wawasan tentang bagaimana pengelolaan sebuah proyek. Langkah-langkah apa saja yang dibutuhkan, bagaimana pelaksanaannya, bagaimana mengontrol serta bagaimana penyelesaiannya. Sasaran pembaca yang diharapkan adalah para mahasiswa dan khalayak umum yang ingin mengenal dunia manajemen proyek secara global dan *IT* pada khususnya.

BAB I
PENDAHULUAN

Sebelumnya, diingatkan kembali mengenai Rekaya Perangkat Lunak. Rekayasa Perangkat Lunak adalah suatu disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal *requirement capturing* (analisis kebutuhan pengguna), *specification* (menentukan spesifikasi dari kebutuhan pengguna), *desain*, *coding*, *testing* sampai pemeliharaan sistem setelah digunakan. Sebelum ini barangkali Anda memahami rekayasa perangkat lunak menggunakan pendekatan terstruktur (*Data Oriented Approach*). Sedangkan pada bahasan-bahasan berikutnya kita akan membahas rekayasa perangkat lunak menggunakan pendekatan objek (*Object Oriented Approach*).

1.1 Perangkat Lunak sebagai Suatu Produk

Perangkat lunak komputer (PL), telah diakui merupakan salah satu penggerak kegiatan industri, bisnis, dan berbagai sektor kehidupan. Perangkat lunak merupakan mesin yang membantu kehidupan manusia dalam pengambilan keputusan. Perangkat lunak menjadi basis pengembangan keilmuan modern dan proses pemecahan masalah.

Perkembangan perangkat lunak yang sangat

pesat ini telah mempengaruhi pemikiran masyarakat. Masyarakat sadar dan melihat perangkat lunak sebagai fakta teknologi yang bermanfaat bagi kehidupan. Manusia mempertaruhkan pekerjaan mereka, kenyamanan, hiburan, keputusan, dan berbagai kepentingan kehidupan mereka kepada perangkat lunak.

Saat ini, perangkat lunak memainkan dua peran. Perangkat lunak merupakan sebuah produk dan pada saat yang bersamaan, PL merupakan sarana atau alat untuk menghasilkan produk. Produk, dapat kita interpretasikan sebagai segala sesuatu yang dapat dihasilkan oleh PL, contohnya layanan.

Sebagai sebuah produk, PL memberikan kemampuan komputasi pada sebuah sistem perangkat keras (komputer). Perangkat lunak juga merupakan agen pengubah informasi (information transformer) – memproduksi, mengatur, mengakuisisi data, memodifikasi, menyampaikan, dan mengirimkan informasi.

Sebagai sebuah sarana untuk menghasilkan sebuah produk, PL berperan sebagai basis kontrol sistem komputer (sistem operasi), komunikasi informasi (networks), dan sebagai penciptaan serta kontrol program-program lain (*software tools* dan *environments*).

1.1.1 Pandangan Industri Mengenai PL

Pada awal komputasi elektronik, sistem berbasis komputer dikembangkan dengan manajemen pengembangan sistem yang berorientasi pada perangkat keras (*hardware*). Manajer proyek saat itu, memfokuskan pengembangan sistem pada penyediaan perangkat keras, karena menurutnya hal inilah yang memakan anggaran proyek paling besar.

Pengontrolan biaya perangkat keras, dilakukan

dengan metode pengontrolan formal dan standar teknis. Mereka melakukan analisis dan desain sebelum membangun sesuatu. Proses diukur untuk dapat menentukan di mana dapat dilakukan peningkatan dan perbaikan kinerja sistem. Mereka pun menekankan pengembangan sistem kepada kualitas (*quality control* dan *quality assurance*). Singkat kata, mereka telah mengaplikasikan kontrol, metode, dan piranti pengembangan yang dikenali sebagai rekayasa perangkat keras (hardware engineering). Pemanfaatan perangkat keras sayangnya jarang atau kurang dipikirkan.

Dahulu, pemrograman dipandang hanya sebagai suatu bentuk seni. Sangat sedikit metode formal yang ada dan masih sedikit orang yang memanfaatkannya. Pada umumnya, mereka melatih kemampuan mereka dan membangun PL dengan *trial and error*. Dunia pengembangan perangkat lunak masih belum disiplin dan banyak praktisi yang menyukainya.

Saat ini, biaya pengembangan sistem berbasis komputer telah berubah secara dramatis. Perangkat Lunak, dibanding dengan perangkat keras, adalah item yang memerlukan alokasi budget yang jauh lebih besar untuk dapat meningkatkan produktivitas industri. Tetapi selama kurang-lebih dua dekade manajer dan praktisi teknis mempertanyakan:

- Mengapa dalam pengembangan perangkat lunak dibutuhkan waktu yang sangat lama?
- Mengapa biayanya sangat besar?
- Mengapa kita menemui kesulitan menemukan kesalahan (*error*) sebelum perangkat lunak tersebut kita berikan kepada pemakai?
- Mengapa begitu sulit menentukan perkembangan perangkat lunak yang sedang dikembangkan?

Pertanyaan-pertanyaan tersebut dan masih banyak pertanyaan-pertanyaan lain yang merupakan pemicu kepedulian kita mengenai perangkat lunak dan cara pengembanganya. Kepedulian tentang pemanfaatan disiplin ilmu rekayasa perangkat lunak.

1.1.2 Perangkat Lunak

Dalam kehidupan sehari-hari sering kita mendengar perangkat lunak atau *software*. Sebenarnya sejauh mana kita mengenal perangkat lunak tersebut? Pernahkan kita bertanya apakah sebenarnya perangkat lunak? Menurut beberapa *textbook* perangkat lunak didefinisikan sebagai beberapa bentuk sebagai berikut:

- Kumpulan atau rangkaian instruksi komputer (program komputer) yang bila kita eksekusi atau jalankan akan menghasilkan performansi dan fungsi yang kita kehendaki.
- Struktur data yang memungkinkan dan mencukupi suatu program untuk dapat memanipulasi informasi.
- Dokumen yang mendeskripsikan teknis pengembangan dan pengoperasian program.

Jadi tiga unsur perangkat lunak adalah program, data, dan dokumen. Perangkat lunak yang lengkap selau memiliki tiga unsur ini. Sebagai agen pembangun dan pengembang perangkat lunak, kita butuh untuk mengenal secara lebih detil segala sesuatu tentang perangkat lunak, lebih dari hanya sekedar definisi.

1.1.3 Karakteristik Perangkat Lunak

Untuk dapat mengembangkan perangkat lunak yang berkualitas, langkah awal yang kita butuhkan adalah

mengetahui karakteristik perangkat lunak, yaitu:

1. Perangkat lunak lebih bersifat sebagai produk logis daripada sebuah elemen fisik sebuah sistem. Oleh sebab itu, pendekatan pengembangan PL berbeda dengan perangkat keras apalagi dengan produksi barang. Perangkat lunak dikatakan sebagai produk logis karena perangkat lunak dibangun dari kumpulan rangkaian logika.
2. Perangkat lunak dikembangkan atau dibangun dengan proses rekayasa (*engineering*), bukan hasil proses manufaktur dalam pengertian produksi klasik.

Meskipun masih terdapat kesamaan antara pengembangan perangkat lunak dan proses manufaktur perangkat keras, kedua aktivitas ini merupakan aktivitas yang benar-benar berbeda. Kedua aktivitas ini membutuhkan proses desain yang baik untuk dapat menghasilkan kualitas yang tinggi, kesalahan yang timbul pada proses manufaktur perangkat keras masih relatif lebih mudah diketahui dan diperbaiki daripada proses pengembangan perangkat lunak.

Kedua aktivitas ini bergantung kepada ketersediaan sumber daya manusia, tetapi hubungan antara sumber daya manusia dengan peningkatan efisiensi dan efektivitas pengembangan produk sangatlah berbeda. Salah satu hal yang dapat kita lihat adalah pada pengembangan perangkat keras, dengan menambah tenaga manusia, maka produksi akan meningkat secara linier. Tetapi tidak begitu halnya dengan penambahan tenaga manusia pada pegembangan perangkat lunak. Berbagai faktor yang mempengaruhi antara lain :

a. Diperlukan usaha untuk mensinkronisasi pembagian

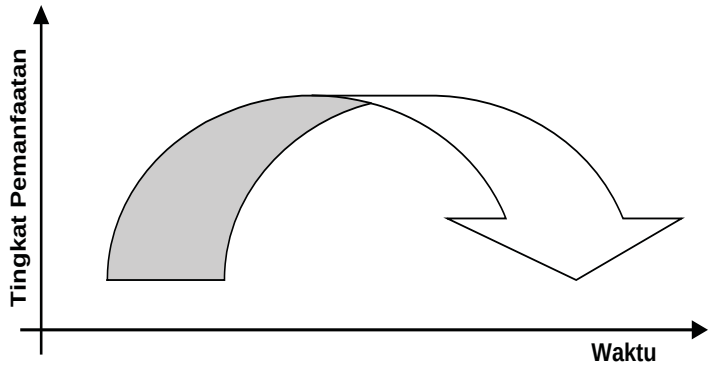
tugas dalam pengembangan perangkat lunak. Semakin banyak team atau pemrogram, ternyata usaha sinkronisasi dan koordinasi semakin rumit.

b. Dengan penambahan tenaga manusia maka diperlukan tambahan waktu untuk penyesuaian atau adaptasi tenaga manusia yang baru untuk memahami perilaku perangkat lunak.

Mengenai hubungan sumber daya manusia dengan pengembangan perangkat lunak dalam sebuah proyek secara lebih detil akan dikaji dalam bagian manajemen perangkat lunak. Kemudian terdapat perbedaan pula pada pendekatan pengembangan sistem.

Biaya atau *budged* pengembangan perangkat lunak terkonsentrasikan pada proses rekayasa (*engineering*), ini berarti pambangaunan dan pengembangan perangkat lunak tidak dapat dikelola seperti halnya proses manufaktur, yang pembiayaannya terkonsentrasi pada biaya seluruh bahan baku dan sumber daya lain.

3. Perangkat lunak tidak dapat kedaluarsa (*'wear out'*)



Kurva pemanfaatan perangkat keras sebagai fungsi waktu

Grafik di atas mengindikasikan bahwa pada awal pembangunan dan pengembangan, perangkat keras selalu diperbaiki dari cacat produksi. Cacat ini diperbaiki dalam selang waktu tertentu hingga pengeleminasian kegagalan yang dimiliki oleh perangkat keras tersebut berjalan dengan sangat lambat. Pada periode ini, pemanfaatan perangkat keras cukup optimal. Akan tetapi dalam beberapa periode waktu pemakaian, pemanfaatan perangkat keras menurun secara drastis. Perangkat keras tersebut telah 'usang' terhadap waktu. Perangkat keras telah usang karena akumulasi debu, eskploitasi penggunaan perangkat keras, temperature, dan berbagai pengaruh buruk lingkungan.

Lain halnya dengan perangkat lunak. PL tidak terpengaruh oleh perubahan dan dampak fisik lingkungan.

Tingkat efisiensi dan efektivitas pemanfaatan perangkat lunak selalu mengalami proses revisi atau perbaikan akibat proses *maintenance*. Sehingga saat fungsionalitas atau pun fitur perangkat lunak sudah mulai 'ketinggalan jaman', perangkat lunak yang baik selalu dapat menyesuaikan fungsionalitasnya dengan kebutuhan yang ada. Artinya, perangkat lunak tersebut mudah dimodifikasi.

4. Meskipun perkembangan industri perangkat lunak bergerak mengarah kepada perakitan komponen-komponen dasar, pengembangan perangkat lunak selalu membutuhkan penyesuaian dengan kebutuhan. Berbeda dengan pembuatan barang (dalam arti fisik: seperti rumah, perangkat keras, dll), selalu ada komponen (contohnya bingkai kaca, jendela, pintu) yang dapat langsung dimanfaatkan atau dirangkai sehingga dapat dihasilkan produk baru (contoh: rumah).

1.2 Kualitas Perangkat Lunak

Kapankah kita dapat menyatakan bahwa sebuah perangkat lunak berkualitas? Apa saja yang dijadikan sebagai parameter kulitas perangkat lunak? Perangkat lunak dapat dikatakan sebagai perangkat lunak yang berkualitas apabila:

1. Perangkat lunak tersebut memenuhi keinginan pemesan atau pihak yang menggunakannya (*user*). Keinginan *user* tersebut meliputi beberapa aspek, antara lain fitur dan antarmuka.
2. Perangkat lunak tersebut berfungsi dan dapat diimplementasikan dalam jangka waktu yang relatif lama.
3. Mudah dimodifikasi untuk memenuhi kebutuhan yang berkembang.
4. Mudah digunakan.
5. Dapat mengubah atau membangun sesuatu dengan lebih baik. Sebagai contoh, bila perangkat lunak dikembangkan untuk menggantikan suatu proses atau fungsi manual, maka perangkat lunak tersebut harus dapat memberikan nilai tambah terhadap proses atau fungsi yang terdahulu. Nilai tambah yang diberikan antara lain kecepatan pemrosesan, kemudahan proses, dan kehandalan data (jaminan bahwa data yang diolah, proses yang dilakukan, dan informasi yang dihasilkan adalah benar).

Dengan pertanyaan sebelumnya, kita dapat mengajukan pertanyaan "Kapan perangkat lunak dikatakan gagal atau tidak berkualitas?". Perangkat lunak dikatakan gagal apabila:

1. *User* tidak puas terhadap performansi perangkat

lunak.

2. Memiliki banyak kesalahan (*error*, kenyataan yang terjadi pengembang tidak mengakui bahwa dia telah melakukan kesalahan, sehingga dia mengatakan bahwa dalam perangkat lunaknya terdapat *bugs* = kutu).
3. Bila perangkat lunak tersebut sulit untuk dimodifikasi untuk kebutuhan yang berkembang.
4. Bila perangkat lunak tersebut sulit untuk dioperasikan.
5. Menghasilkan sesuatu yang tidak dikehendaki. Misalnya, perangkat lunak saat diperasikan, mengakibatkan register sistem menjadi kacau, sehingga sistem operasi yang kita gunakan menjadi tidak stabil.

Kita semua ingin membangun perangkat lunak yang berkualitas. Agar keinginan kita tersebut dapat tercapai, kita membutuhkan suatu disiplin ilmu untuk mendesain dan membangun perangkat lunak. Kita membutuhkan pendekatan rekayasa perangkat lunak.

1.2.1 Pentingnya Proses Rekayasa Perangkat Lunak

Setelah kita mengetahui gambaran umum perangkat lunak yang berkualitas dan kita pun ingin untuk dapat membangun perangkat lunak yang berkualitas. Kemudian mungkin timbul pertanyaan dalam benak kita: Mengapa kita mesti melalui tahap-tahap pembangunan perangkat lunak yang terlihat rumit dan harus melewati tahap-tahap tertentu? Mengapa kita tidak langsung menulis kode program perangkat lunak kita sambil mengira-ngira dan menentukan fitur-fitur apa saja yang bakal dimiliki oleh perangkat lunak kita? Pengembangan perangkat lunak

secara umum melalui beberapa tahap, yaitu:

1. Analisis
2. Desain
3. Pengkodean
4. Testing
5. *Maintenance* atau perawatan

Memang pertanyaan atau ungkapan yang kita pikirkan sebelumnya, benar untuk pembangunan sebuah perangkat lunak yang kecil. Kita tidak perlu melewati tahap-tahap rekayasa perangkat lunak tersebut. Yang kita perlukan adalah sedikit analisis kebutuhan perangkat lunak. Kita hanya perlu mengetahui apa fungsi perangkat lunak yang akan kita bangun, siapa pemakainya, akan dioperasikan dalan lingkungan operasi dan implementasi apa, dan beberapa pertimbangan sederhana lain. Contoh sebuah perangkat lunak kecil yaitu : perangkat lunak untuk menghitung faktorial, konversi suhu *Celcius ke Fahrenheit*, dan sebagainya.

Ukuran perangkat lunak relatif sulit untuk ditentukan. Bila kita membandingkan ukuran pengembangan perangkat lunak dengan pembangunan sebuah gedung, pembangunan sebuah gedung relatif lebih mudah kita tentukan. Misalnya dengan parameter luas lahan, lokasi, dan jenis bahan. Namun perangkat lunak membutuhkan disiplin-disiplin ilmu lain (*software metrics* dan *software measurement*) untuk dapat menentukan ukuran perangkat lunak.

Namun mungkin sebagai pedoman kita, perangkat lunak dapat diukur dengan *Kilo Lines of Codes* (KLOC) atau baris kode program dalam ribuan. Bila perangkat lunak yang kita bangun, memiliki proses yang masih tergolong sederhana dan kira-kira hanya memiliki beberapa puluh

baris instruksi, maka proses rekayasa perangkat lunak kurang perlu kita butuhkan. Kita hanya perlu membuat sedikit dokumentasi teknis perangkat lunaknya saja. Dan sebagai saran, tambahkan komentar-komentar program pada program-sumber kita. Sehingga bila perangkat lunak perlu dikembangkan, kita dapat mengetahui pada proses yang mana kita akan melakukan modifikasi. Namun sebagai seorang analis/desainer perangkat lunak, kita akan sering bergelut dengan pengembangan perangkat lunak yang berukuran sedang hingga sangat besar.

Sehubungan dengan pertanyaan awal tersebut, mengapa kita mesti melalui tahap-tahap pembangunan perangkat lunak yang terlihat rumit dan harus melewati tahap-tahap tertentu? Mengapa kita tidak langsung menulis kode program perangkat lunak kita sambil mengira-ngira dan menentukan fitur-fitur apa saja yang bakal dimiliki oleh perangkat lunak kita? Kita akan dengan mudah memahaminya dengan meninjau mitos-mitos yang ada di masyarakat dan kenyataan yang terjadi.

1.2. 2 Mitos tentang Perangkat Lunak

Dalam pengembangan perangkat lunak, terutama pada awal manusia mengenal perangkat lunak, selalu ada mitos-mitos yang tertanam dalam pikiran dan sikap manusia. Dari tingkat pelaku manajemen, komsumen, hingga ke pelaku teknis (pengembang PL), selalu memiliki anggapan-anggapan dan sikap yang ternyata tidak dapat diterapkan pada pengembangan perangkat lunak:

1. Mitos pada tingkat manajemen

Manajer yang bertanggung jawab terhadap pengembangan perangkat lunak, seperti manajer di bidang-bidang lain, sering berada dalam tekanan dalam

pengelolaan *budged*, pelaksanaan rencana sehingga sesuai jadwal, dan tangung jawab untuk meningkatkan kualitas bisnis. Mereka berpegangan pada mitos perangkat lunak untuk mengurangi tekanan ini.

a. **Mitos:** Orang-orang kami memiliki piranti pengembangan perangkat lunak yang modern dan kami juga memiliki komputer yang memanfaatkan teknologi terbaru. Kami pasti dapat membangun PL yang sangat berkualitas.

Fakta : Pengembangan perangkat lunak membutuhkan lebih dari sekadar komputer atau bahkan *mainframe* terbaru. *Computer Aided Software Engineering* (CASE) *tools* atau piranti bantu pengembangan perangkat lunak, (alat bantu dan metode rekayasa perangkat lunak, contohnya SSADM, Power Analist) lebih dibutuhkan dari sekedar perangkat keras yang berteknologi terbaru.

b. **Mitos:** Jika jadual pengembangan perangkat lunak terlambat dari rencana, kita dapat menambah programmer (sering disebut *Mongolian Horde Concept*).

Fakta : Pengembangan perangkat lunak bukan proses mekanis seperti proses manufaktur. Dalam bukunya "*The Mythical Man-Month* ", F. Brooks mengatakan "*Adding people to a late project makes it later*". Artinya, dengan menambah pemrogram dalam sebuah pengembangan perangkat lunak yang terlambat, akan memperlambat pengembangannya. Karena ketika pemrogram baru ditambahkan, programmer

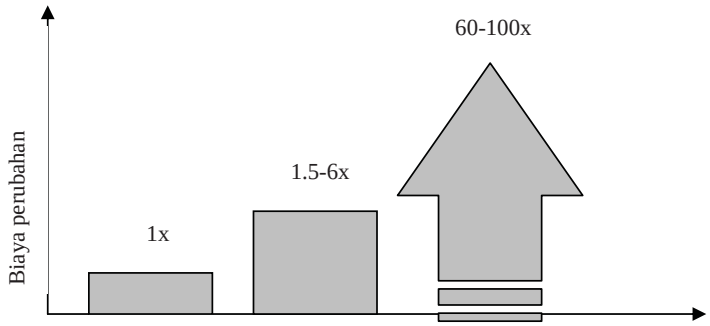
yang telah bekerja sebelumnya, harus menghabiskan waktu untuk mengajari dan menjelaskan sistem yang sedang dibangun, sehingga akan banyak waktu terbuang. Penambahan sumber daya manusia dalam proyek pengembangan perangkat lunak dapat dilakukan dengan rencana awal yang matang.

c. **Mitos:** Pernyataan global mengenai tujuan pemesanan perangkat lunak telah cukup untuk digunakan sebagai dasar memulai penulisan program, detail kemudian.

Fakta : Definisi awal yang kurang jelas merupakan sebab utama kegagalan pengembangan perangkat lunak. Deskripsi yang formal dan detil terhadap domain, fungsi, perilaku, performansi, antarmuka, batasan desain, dan kriteria validasi adalah hal yang vital. Karakteristik-karakteristik ini hanya dapat ditentukan melalui komunikasi antara konsumen (*customer*) dan pengembang.

d. **Mitos:** Kebutuhan perangkat lunak memungkinkan untuk berubah, tapi tenang saja karena perubahan itu akan mudah diakomodasi karena perangkat lunak bersifat fleksibel.

Fakta : Benar bahwa kebutuhan PL dapat berubah, tetapi efeknya bermacam-macam, tergantung kapan perubahan tersebut dilakukan. Semakin dini perubahan kebutuhan tersebut dilakukan, maka biaya (tenaga) yang dibutuhkan semakin kecil.



Akibat perubahan kebutuhan terhadap biaya

Mitos pada pengembang perangkat lunak

e. **Mitos:** Sekali kita telah menulis program yang berhasil dieksekusi, maka tugas kita telah selesai

Fakta: Bahwa semakin cepat kita memulai menulis program (*coding*) sebelum semua analisis dan desain selesai, maka waktu yang kita butuhkan untuk menyelesaikan perangkat lunak akan semakin lama. Hal ini berhubungan dengan proses analisa kebutuhan, pencarian kesalahan, manajemen perubahan perangkat lunak, dan berbagai aspek lain yang semakin sulit dirunut.

Kemudian, sebenarnya tugas yang lebih berat adalah pasca *coding*, yaitu *testing* dan *delivery* ke *customer*. Selalu dibutuhkan penyesuian terhadap sitem customer dan customer memerlukan layanan *support* dan *service*. Bila kita ingin usaha kita di bidang pengembangan perangkat lunak ini terus berjalan, maka kita harus mempertimbangkan beberapa hal tersebut.

f. **Mitos:** Satu-satunya produk (*deliverable*) yang penting untuk kesuksesan suatu produk adalah program yang dapat dieksekusi.

Fakta: Program yang dapat dieksekusi (*working program*) hanyalah salah satu elemen dari produk konfigurasi perangkat lunak. Dokumentasi dan petunjuk *support* perangkat lunak adalah elemen lain yang sangat penting.

g. **Mitos:** Dokumentasi perangkat lunak hanyalah sebagi kumpulan dokumen yang membebani dan tidak berarti, yang dapat memperlambat kerja kita.

Fakta: Rekayasa perangkat lunak bukanlah hal tentang pembuatan dokumen, tetapi merupakan proses pembangunan kualitas. Kualitas yang bagus berdampak kepada pengurangan usaha kerja ulang untuk perbaikan dan perubahan yang memang sering dan selalu terjadi, sehingga pada proses berikutnya kita dapat menghemat waktu.

BAB II
PROSES REKAYASA PERANGKAT LUNAK

2.1 Pengertian Rekayasa Perangkat Lunak

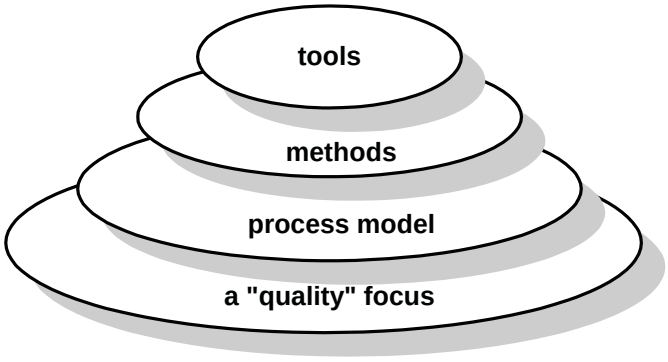
Ketika kita membangun produk atau sistem, kita membutuhkan rangkaian langkah yang dapat diprediksi sehingga dapat menuntun gerak kita. Kita membutuhkan pedoman sehingga kita dapat menciptakan hasil yang berkualitas dan tepat waktu. Pedoman inilah yang kita sebut sebagai proses perangkat lunak (*software process*). Manajer dan pengembang perangkat lunak mengadaptasi proses dan menerapkannya dengan melibatkan *customer* dalam pengembangan perangkat lunak yang dipesannya.

Proses perangkat lunak ini penting, karena proses ini memberikan kestabilan, kontrol, dan pengorganisasian aktivitas pengembangan perangkat lunak. Proses ini menghasilkan integrasi program, data, dan dokumen. IEEE *Computer Society* mendefinisikan rekayasa perangkat lunak (*software engineering*) sebagai:

1. Suatu aplikasi dari pendekatan yang sistematis, disiplin, dan terukur terhadap pengembangan, pengoperasian, dan perawatan perangkat lunak. Atau dengan kata lain, rekayasa perangkat lunak adalah aplikasi rekayasa (*engineering*) terhadap perangkat lunak.
2. Kajian terhadap pendekatan yang sistematis, disiplin,

dan terukur terhadap pengembangan, pengoperasian, dan perawatan perangkat lunak.

2.2 RPL: Proses, Metode, Piranti



Lapisan Rekayasa Perangkat Lunak

Rekayasa perangkat lunak ditujukan untuk peningkatan kualitas produk, fokus pada kualitas. Proses adalah Pondasi rekayasa perangkat lunak. Proses rekayasa perangkat lunak mengintegrasikan teknologi dan memungkinkan proses rasional dan pengembangan perangkat lunak yang tepat waktu. Proses rekayasa perangkat lunak mendefinisikan kerangka kerja yang perlu dijalankan untuk mendapatkan efisiensi pembangunan produk rekayasa perangkat lunak. Proses rekayasa perangkat lunak menjadi dasar manajemen kontrol proyek perangkat lunak dan memberikan pedoman penerapan metode teknis dan jadual pembangunan produk (model, dokumen, data, *reports*, *form,* dsb.), jaminan kualitas, dan manajemen perubahan kebutuhan.

Metode rekayasa perangkat lunak menyediakan cara dan petunjuk membangun perangkat lunak. Metode ini

mencakup kumpulan tugas termasuk analisis kebutuhan, desain, implementasi program (*coding*), *testing,* dan *support*.

Piranti (*tools*) rekayasa perangkat lunak menyediakan dukungan yang otomatis atau semi otomatis terhadap proses dan motode rekayasa perangkat lunak. Ketika suatu piranti diintegrasikan, informasi yang dihasilkan oleh satu *tool* dapat digunakan oleh *tool* yang lain. Sistem ini memberikan dukungan pengembangan perangkat lunak, sehingga disebut *Computer Aided Software Engineering* (CASE). CASE mengintegrasikan perangkat lunak, perangkat keras, dan basisdata rekayasa perangkat lunak (basisdata yang menyimpan seluruh informasi rekayasa perangkat lunak: analisis, desain, *coding*, dan *testing*).

2.3 Fase Proses RPL

Rekayasa adalah analisis, desain, konstruksi, dan proses manajemen suatu entitas (objek kajian). Apapun (entitas) yang dibangun dan dikembangkan, pertanyaan berikut perlu kita jawab:

1. Apa persoalan yang harus dipecahkan?
2. Karakteristik apa dari entitas yang digunakan untuk dapat memecahkan persoalan?
3. Bagaimana suatu entitas dan solusinya dapat direalisasikan?
4. Pendekatan apa yang dapat dan akan digunakan untuk memperbaiki kesalahan yang dilakukan pada tahap desain dan implementasi?
5. Bagaimana kita memberikan *support* setelah menyelesaikan suatu persoalan, ketika perbaikan, adaptasi, peningkatan fitur (*enhancement*), dan perawatan diminta oleh *customer*?

Rekayasa perangkat lunak dapat dikategorikan ke dalam tiga fase utama, tidak peduli area aplikasinya, ukuran, dan kompleksitas proyek. Setiap fase merepresentasikan pertanyaan-pertanyaan sebelumnya yang perlu kita jawab.

Fase tersebut adalah:

1. **Fase definisi (*definition*)**
   Fase ini memfokuskan pada pertanyaan **apa / what**. Selama proses pendefinisian, *software engineer* berusaha untuk mengidentifikasikan:
   a. Informasi apa yang akan diproses?
   b. Fungsi dan performansi apa yang diinginkan?
   c. Perilaku sistem seperti apa yang diharapkan?
   d. Antarmuka apa yang akan dibangun?
   e. Batasan desain apa saja yang ada?
   f. Kriteria validasi apa yang diperlukan untuk dapat mendefiniskan sebuah sistem yang sukses?
2. **Fase pengembangan (*development*)**
   Fase ini memfokuskan pada pertanyaan **bagaimana / how**. Selama proses ini *software engineer* berusaha untuk mengidentifikasikan:
   a. Bagaimana data distrukturkan (struktur data)?
   b. Bagaimana fungsi-fungsi sistem diimplementasikan dalam arsitektur perangkat lunak?
   c. Bagaimana detail prosedur diimplementasikan?
   d. Bagaimana memberikan karakteristik antarmuka?
   e. Bagimana desain akan diimplementasiakan dalam sebuah bahasa pemrograman?
   f. Bagaimana prose pengujiannya?

3. Fase *support*

Fase ini memfokuskan pada manajemen **perubahan / change** saat diminta oleh *customer.* Fase *support* mengaplikasikan ulang langkah-langkah pada fase pendefinisian dan pengembangan, tetapi diimplementasikan pada perangkat lunak yang telah dibangun sebelumnya. Empat tipe perubahan yang mungkin dilakukan adalah:

a. Koreksi (*corrective maintenance*)
   Meskipun dengan mengaplikasikan aktivitas jaminan kualitas, masih terdapat kemungkinan besar ditemukan *error* oleh *customer*, sehingga perlu dilakukan proses koreksi kesalahan.

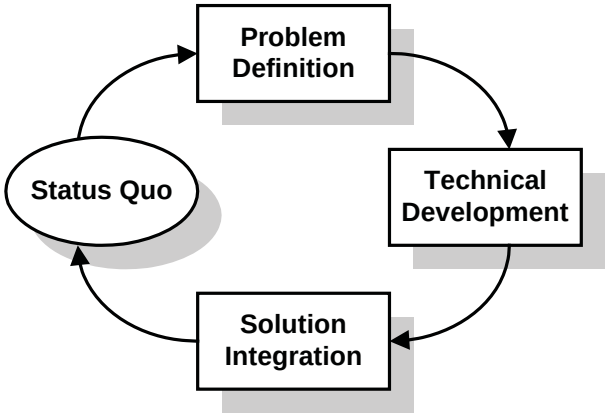
b. Adaptasi (*adaptive maintenance*)
   Dengan perkembangan waktu, lingkungan operasional perangkat lunak (CPU, sistem operasi, *business rules*) oleh *customer* sangat dimungkinkan untuk berubah. Adaptasi perangkat lunak perlu dilakukan untuk mengakomodasi perubahan ini.

c. *Enhancement* (*perfective maintenance*)
   Selama pengoperasian perangkat lunak, *customer* akan membutuhkan tambahan fitur yang dapat meningkatkan efisiensi dan efektivitas bisnisnya.

d. *Prevention (preventive maintenance = software reengineering)*
   Pernah kita membahas efek waktu terhadap kualitas pemanfaatan perangkat lunak. Agar perangkat lunak tersebut tidak 'ketinggalan jaman', maka perlu dilakukan *preventive maintenance* agar kualitas perangkat lunak dapat terus dijaga bahkan ditingkatkan.

2.4 Model Proses RPL/ Paradigma Pengembangan PL

Untuk dapat memecahkan persoalan dalam lingkup industri, seorang *software engineer* atau tim *engineer* harus menerapakan sebuah strategi yang mencakup proses, metode, dan piranti yang digunakan. Strategi ini sering disebut sebagai model proses atau paradigma rekayasa perangkat lunak. Pengembangan perangkat lunak dapat dicirikan sebagai iterasi proses pemecahan persoalan (*problem solving loop*) dengan mencakup empat status:



Iterasi Proses Pemecahan Masalah

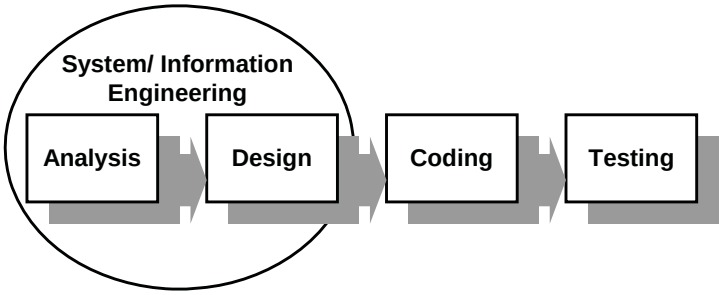
Status quo merepresentasikan status perkerjaan yang sedang dilakukan yang merupakan bagian perkerjaan dari keseluruhan iterasi. Pendefinisian persoalan mendefinisikan persolan yang sedang dihadapi dan akan dicarikan pemecahannya. Pengembangan teknis memecahkan masalah melalui penerapan beberapa teknologi. Integrasi solusi menyampaikan hasil solusi (dokumen, program, data, fungsi bisnis baru, produk baru) ke pemesan.

Model proses rekayasa perangkat lunak cukup

banyak. Seluruh model tersebut membantu kita sebagai pedoman dalam mengontrol dan mengkoordinasikan proyek perangkat lunak. Dalam pembahasan ini, kita akan mengenal dan mencoba memanfaatkan metode yang relatif sederhana, tetapi cukup mendasar dan sering digunakan.

1. Linear Sequential Model

Model proses ini sering disebut sebagai ***Waterfall*** atau ***Classic Life Cycle Model***. Metode *Linear Sequential Model* menyarankan pendekatan yang sistematis dan sekuensial dalam pengembangan perangkat lunak yang dimulai pada level sistem dan bergerak maju mulai tahap analisis, desain, *coding*, *testing*, dan *support*.



The Linear Sequential Model

Model *Linear Sequential* mencakup aktivitas-aktivitas berikut:

1. **Rekayasa dan pemodelan sistem/informasi** (*System/ information engineering*). Dikarenakan perangkat lunak selalu merupakan bagian dari sistem atau bisnis yang lebih besar, kegiatan proses perangkat lunak dimulai dengan melakukan indentifikasi kebutuhan (*requirements*) dari seluruh elemen sistem lalu memetakan bagian dari kebutuhan tersebuat

sebagai kebutuhan perangkat lunak.

Pandangan secara sistem ini sangat dibutuhkan ketika perangkat lunak harus berinterakasi dengan elemen-elemen yang lain seperti perangkat keras, manusia, dan basisdata. Identifikasi dan pengumpulan kebutuhan dilakukan dalam level strategi bisnis dan level manajerial.

2. **Analisis kebutuhan perangkat lunak** (*Software requirements analysis*). Proses identifikasi dan pengumpulan kebutuhan sistem difokuskan pada kebutuhan perangkat lunak. Untuk memahami program-program yang akan dibangun, analis harus memahami domain dan lingkup perangkat lunak yang akan dibangun, termasuk didalamnya fungsi, tingkah laku perangkat lunak, performansi, dan antarmuka. Kebutuhan sistem dan perangkat lunak didokumentasikan dan dikonfirmasikan dengan *customer*.
3. **Desain** (*Design*). Proses desain perangkat lunak fokus kepada sturktur data, arsitektur perangkat lunak, representasi antarmuka, dan algoritma detil proses. Desain merupakan representasi kebutuhan yang akan dijadikan pedoman dalam pengkodean program. Desain didokumentasikan dan menjadi bagian dari manajemen konfigurasi perangkat lunak.
4. **Pengkodean** (*Code generation*). Desain yang telah dibuat, ditranslasikan ke dalam suatu bahasa pemrograman tertentu.
5. **Pengujian** (*Testing*). Setelah kode program dibangun, program diuji untuk memastikan bahwa semua kebutuhan dan persoalan dapat diselesaikan dan benar. Proses pengujian fokus pada logika perangkat lunak. Memastikan bahwa dari proses input,

pemrosesan, hingga *output* benar dan sesuai dengan yang diinginkan.

6. **Support**. Perangkat lunak sangat memungkinkan untuk berubah. Perubahan dapat terjadi karena ditemukannya kesalahan, perangkat lunak harus diadaptasikan kepada sistem yang baru, *customer* menginginkan peningkatan fungsional dan performansi perangkat lunak, dan perawatan perangkat lunak.

   Keunggulan model ini adalah:

   1. Programmer jarang diinterupsi pekerjaanya dengan perubahan kebutuhan, sehingga programmer dapat lebih konsentrasi.
   2. Bila proses yang dilakukan telah jelas dan pasti, model ini akan memberikan efisiensi dan efektivitas yang tinggi.
   3. Dituntut bekerja secara disiplin
   4. Dokumennya lengkap
   5. Maintenance mudah karena memiliki dokumen yang lengkap
   6. Selalu dalam kontrol SQA (Software Quality Assurance)

Kekurangan model ini adalah:

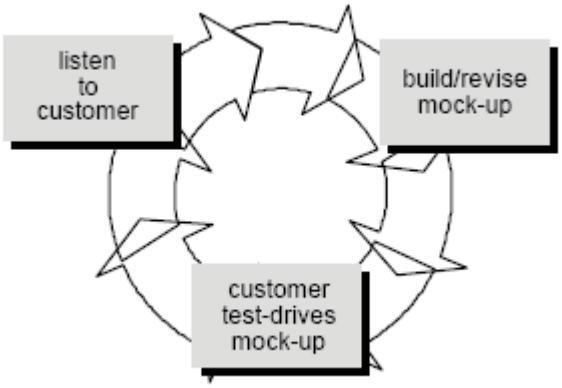
Meskipun model ini merupakan model yang paling banyak/sering digunakan, model ini memiliki kekurangan:

1. Umumnya *customer* kurang bisa menyampaikan seluruh kebutuhan yang diinginkannya dalam tahan pendefinisian, sehingga dalam proses pengembangan perangkat lunak sering *customer* menginginkan perubahan atau tambahan kebutuhan. Model *waterfall* sulit mengakomodasi perubahan ini.

2. Kesalahan yang dibuat akan dibawa ke tahap berikutnya. Pada tahap pengujian (*testing*) bila kesalahan tersebut ditemukan, kesalahan tersebut telah terakumulasi dan berinteraksi dengan proses-proses lain, sehingga biaya (waktu, uang, sumber daya lain) dan usaha harus dikeluarkan dalam jumlah yang besar.
3. Keterlibatan *customer* sangat minim. *Customer* harus sabar menunggu hingga semua program selesai dibangun.
4. Konsumen sulit membaca dokumen menyebabkan komunikasi juga menjadi sulit
5. Alur pengerjaan yang linier menyebabkan proses menjadi lambat,
6. Personil tidak bekerja optimal karena ada waktu tunggu atau jeda setiap tahapan.

II. Prototyping Model

Sering *customer* dalam memesan perangkat lunak tidak dapat mengidentifikasi *input,* proses, dan permintaan output secara detil. Kasus lain, yaitu pengembang tidak yakin terhadap efisiensi algoritma yang digunakan terhadap permasalahan yang dihadapi *customer*, kemampuan adaptasi terhadap sistem operasi, atau bentuk interaksi manusia -perangkat lunak yang akan diterapkan. Pada kasus-kasus ini, paradigma *prototyping* memberikan pendekatan yang lebih baik.



Prototyping Model

Model Prototyping mencakup aktivitas-aktivitas berikut:

1. **Pengumpulan kebutuhan**. Aktivitas dimulai dengan pengumpulan kebutuhan (*requirements*). Pengembang dan *customer* bertemu untuk menentukan tujuan keseluruhan dan global perangkat lunak, mengidentifikasi kebutuhan yang telah diketahui, lalu mendefinisikan area dan lingkup pengembangan.
2. **Desain.** Proses desain dilakukan dengan sangat cepat. Desain difokuskan kepada aspek-aspek desain yang nampak kepada *customer*/*user* (contoh: interface, pendekatan input, format output). Hasil desain inilah yang disebut sebagai prototipe.
3. **Evaluasi Prototipe**. Prototipe yang dihasilkan, direview oleh *customer*. Hasil evaluasi ini dijadikan bahan untuk perubahan dan pengembangan selanjutnya. Iterasi terus dilakukan hingga memenuhi keinginan *customer*, sementara pada saat yang sama, memungkinkan pengembang untuk dapat lebih memahami kebutuhan perangkat lunak.

Tujuan utama model proses prototipe ini adalah untuk memudahkan pengembang memahami kebutuhan *customer*. Brooks, F. dalam bukunya "The Mythical Man-Month", menyarankan kita untuk membuang prototipe ini bila semua kebutuhan *customer* berhasil diungkap. Akan tetapi, dapat juga kita melakukan modifikasi dan memanfaatkan template prototipe sebelumnya.

Keunggulan model ini adalah:

1. Dapat segera memahami kebutuhan *customer*.
2. Dapat segera menemukan kesalahan program atau kesalahan interpretasi kebutuhan perangkat lunak.
3. *Customer* atau *user* dapat segera memahami dan mempunyai gambaran terhadap sistem (PL) yang akan dibangun.

Kelemahan model ini, yaitu:

1. Biasanya *customer* setelah melihat prototipe terakhir yang telah disepakati bersama, *customer* ingin segera memilikinya. Tidak peduli efisiensi algoritma yang digunakan, dokumentasi, dan aspek-aspek rekayasa perangkat lunak lain. Hal ini dapat menyebabkan kualitas perangkat lunak menjadi rendah.
2. Programmer akan sering diinterupsi oleh perubahan. Sedangkan sifat alamiah perubahan adalah mengganggu.

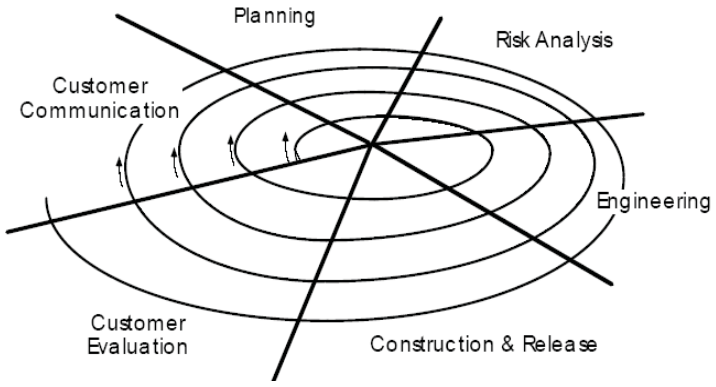
III. Evolutionary Model

Model sequensial linier di atas dirancang untuk pengembangan PL yang memiliki garis lurus, yang berarti bahwa sistem yang lengkap akan disampaikan setelah urutan linier tersebut dilengkapi. Sedangkan model

prototype dirancang untuk mendorong konsumen (atau pengembang) agar memahami kebutuhan (menyampaikan sitem produksi).

Model evolusioner adalah model *iterative* yang tidak termasuk kedalam model klasik, model ini memungkinkan perekayasa PL untuk mengembangkan versi PL yang lebih lengkap sedikit demi sedikit. Model evolusioner terbagi menjadi banyak model, tapi hanya 3 model yang paling digunakan, yaitu:

Spiral Model



Model Spiral, adalah model yang diusulkan oleh Boehm (1988), yaitu model proses perangkat lunak yang evolusioner yang merangkai sifat *iterative* dari model prototype dengan cara kontrol dan aspek sistematis dari model sequential linier. Dengan kata lain model ini menggunakan fitur-fitur yang ada pada Model Sequential dan Prototyping.

Model Spiral mencakup aktivitas-aktivitas berikut:

1. **Customer Communication**, Komunikasi Pelanggan

yaitu tugas-tugas yang dibutuhkan untuk membangun komunikasi yang efektif diantara pengembang dan pelanggan.

2. **Planning**, Perencanaan yaitu tugas-tugas yang dibutuhkan untuk mendefinisikan sumber-sumber daya, ketepatan waktu, dan proyek informasi lain yang berhubungan.
3. **Risk Analysis**, Analisis Resiko yaitu tugas-tugas yang dibutuhkan untuk menaksir resiko-resiko yang mungkin akn dihadapi, baik manajemen maupun teknis.
4. **Engineering**, Perekayasaan yaitu tugas-tugas yang dibutuhkan untuk membangun satu atau lebih representasi dari aplikasi tersebut.
5. **Construction and Release**, Konstruki dan Peluncuran yaitu tugas-tugas yang dibutuhkan untuk mengkonstruksi, menguji, memasang (instal) dan memberikan pelayanan kepada pemakai, contohnya pelatihan dan dokumentasi.
6. **Customer Evaluation**, Evaluasi Pelanggan yaitu tugas-tugas yang dibutuhkan untuk memperoleh umpan balik dari pelanggan dengan didasarkan pada evaluasi representasi perangkat lunak, yang dibuat selama masa perekayasaan, dan diimplementasikan selama masa pemasangan.

Keunggulan model ini adalah:

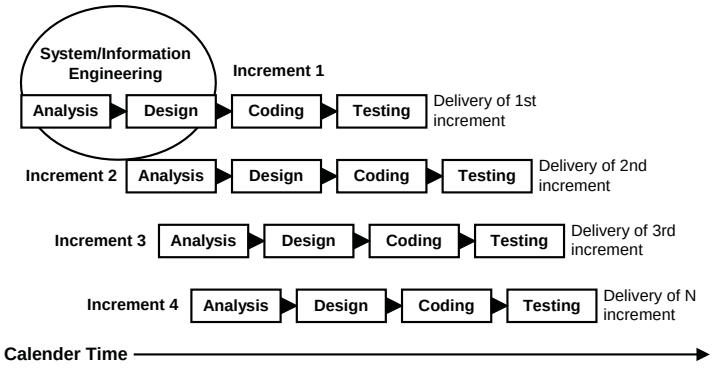
1. Model ini sangat baik digunakan untuk sistem dan perangkat lunak yang besar.
2. Ditekankan pada pencarian alternatif, dan pemaksaan penggunaan kembali software yang telah ada.
3. Adanya analisis resiko pada mekanisme untuk memperkecil resiko

4. Adanya prototype memudahkan komunikasi dengan konsumen

Kelemahan model ini yaitu:

1. Memerlukan waktu yang cukup lama untuk mengembangkan perangkat lunak.
2. Sistem Pengontrolan yang kurang baik
3. Tidak banyak cerita sukses mengenai perancangan menggunakan model ini.
4. Biasanya pihak pengembang dan perusahaan berada pada satu pihak yang sama. Tahapan analisa resiko sewaktu-waktu dapat membatalkan proses rekayasa, jika pihak pengembang adalah pihak diluar perusahaan, maka akan timbul masalah hokum.

5. Incremental Model



Incremental Model, adalah model rekayasa perangkat lunak yang dikerjakan bagian per bagian hingga menghasikan perangkat lunak yang lengkap. Proses pembangunan berhenti bila produk telah mencakup seluruh fungsi yang diharapkan.

Model Spiral mencakup aktivitas-aktivitas berikut:

Pada tahapan sistem *engineering* aktivitas utama yang dilakukan adalah *Requirement*, *Specification* dan *Architecture Design*. Setelah aktivitas utama terpenuhi, tahapan selanjutnya adalah membangun tiap bagian secara berurutan. Setiap bagian yang sudah selesai dilakukan testing, dikirim ke pemakai untuk langsung dapat digunakan.

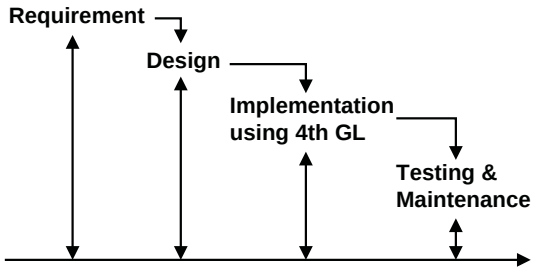
Keunggulan model ini adalah:

1. Personil dapat bekerja secara optimal.
2. Konsumen dapat langsung menggunakan dahulu bagian-bagian yang telah selesai dibangun. Misalnya input data karyawan.
3. Megurangi trauma karena perubahan sistem. Klien dibiasakan perlahan-lahan menggunakan produknya bagian per-bagian.
4. Memaksimalkan pengembalian model investasi konsumen.

Kelemahan model ini, yaitu:

1. Kemungkian tiap bagian tidak dapat diintegrasikan dengan baik.
2. Selalu berubah selama proses rekayasa berlangsung.
3. Harus open architecture.

4th Generation Technicques



Paradigma 4th GT untuk rekayasa perangkat lunak berfokus pada kemampuan spesifikasi perangkat lunak dengan menggunakan bentuk bahasa yang dikhususkan atau sebuah notasi grafik yang menggambarkan masalah yang akan dipecahkan kedalam bentuk yang dapat dipahami oleh pelanggan.

Model 4th GT mencakup aktivitas-aktivitas berikut:

1. *Requirement Gathering*, usaha untuk mengetahui/ mendapatkan kebutuhan atas perangkat lunak yang akan dibangun.
2. *Design Strategy*, menentukan strategy perancangan, ini adalah bagian terpenting.
3. Implementasi, menggunakan 4th GL (*fourth generation language*), karena semua prosedur pemrograman sudah tersedia (tinggal click), dikarenakan sudah tersedia maka implementas menjadi mudah.
4. Testing, maintenance.

Keunggulan model ini adalah:

1. *User* lebih gampang mendesain program
2. Semua orang awam bisa.
3. Megurangi waktu yang dibutuhkan untuk menghasilkan PL aplikasi kecil dan menengah.
4. Mempercepat waktu desain.

Kelemahan model ini yaitu:

Untuk aplikasi yang besar dibutuhkan analisis, desain dan pengujian yang sangat banyak, atau dengan kata lain aktivitas rekayasa perangkat lunak sangat besar.

- **Metode Analisis dan Perancangan PL**

Terlepas dari paradigma pengembangan perangkat lunak yang dijelaskan di atas, ada dua pendekatan yang bisa dilakukan dalam Analisis dan Perancangan Perangkat Lunak, yaitu:

1. Pendekatan Berorientasi Data (*Data Oriented Approach*) / Terstruktur
   a. Data Flow Oriented
   b. Data Structured Oriented
2. Pendekatan Berorientasi Objek (*Object Oriented Approac*h) / Objek
   a. Shlaer/Mellor Method [Shlaer-1988]
   b. Coad/Yourdan Method [Coad-1991]
   c. Booch Method [Booch-1991] / OOAD
   d. OMT Method [Rumbaugh-1991]
   e. Wirfs-Brock Method [Wirfs-Brock-1990]
   f. OOSE Objectory Method [Jacobson-1992]
   g. UML (Unified Modeling Language) [UML-1997].

Bab III
ANALISIS & PERANCANGAN DENGAN PENDEKATAN TERSTRUKTUR

Analisis dan Perancangan perangkat lunak dengan pendekatan terstruktur atau dikenal dengan pendekatan berorientasi data (*Data Oriented Approach*) adalah pendekatan konvensional yang menitik beratkan permasalahan pada aliran Data, yaitu: Arus Data (*Data Flow*) dan Struktur Data (*Data Structure*).

Pendekatan ini sangat dominan untuk digunakan dimasa-masa awal perkembangan rekayasa perangkat lunak. Untuk merancang PL skala kecil dan menengah, perancangan menggunakan pendekatan terstruktur masih layak digunakan, karena lingkup permasalahan masih bisa ditangani dengan melihat kebutuhan data yang akan digunakan. Tapi, jika lingkup permasalahannya cukup besar, misalnya untuk perancangan PL yang besar maka akan mengalami kesulitan dalam menentukan prioritas pengembangan baik pada saat analisis maupun perancangan, yaitu tahapan sebelum tahapan implementasi dan pengujian dilakukan.

3.1 Tahap Analisis

Hal yang utama dalam tahap ini adalah pendefinisian kebutuhan perangkat lunak. Proses rekayasa ini meliputi Pengidentifikasian kebutuhan, Perbaikan identifikasi kebutuhan, Pemodelan, dan Spesifikasi kebutuhan.

Model data yang dibutuhkan dalam tahap analisis adalah Aliran informasi dan kontrol, dan Tingkah laku sistem saat dioperasikan dibangun. Kemudian lebih lanjut lagi, alternatif solusi dapat diajukan kepada *costomer*.

Mengapa proses ini diperlukan? Tentu kita tidak ingin membangun perangkat lunak yang banyak memiliki kesalahan, banyak aspek kebutuhan yang tidak terungkap, banyak faktor lingkungan yang berpengaruh yang tidak dianalisis, membutuhkan waktu pengembangan yang sangat lama dan menghabiskan banyak biaya karena kecerobohan, yang akhirnya juga berakibat kepada ketidakpuasan *customer.* Oleh karena itu, kita membutuhkan analisis kebutuhan perangkat lunak.

Kita perlu mengetahui prinsip-prinsip analisis, yaitu:

1. Domain masalah harus dapat direpresentasikan dan dipahami.
2. Fungsi-fungsi yang harus dimiliki oleh perangkat lunak nantinya harus dapat ditentukan.
3. Tingkah laku perangkat lunak saat dioperasikan sebagai aksi dari input pemakai atau lingkungan harus dapat didefinisikan.
4. Model yang menggambarkan informasi, fungsi, dan tingkah laku perlu di dekomposisi sehingga semua detil informasi, fungsi, dan tingkah laku yang ada dapat diungkap.
5. Proses analisis sebaiknya dimulai dari informasi-informasi yang penting sampai ke detil implementasi.

Kemudian bila programmer dipaksa untuk mengerjak (mengimplementasikan) perangkat lunak yang spesifikasinya tidak lengkap, tidak konsisten, dan

menyesatkan akan mengalami kebingungan dan frustasi dan hasilnya adalah implementasi yang tidak menentu. Spesifikasi kebutuhan perangkat lunak merupakan cara untuk mengarahkan ke pembanguna perangkat lunak yang berhasil dengan baik.

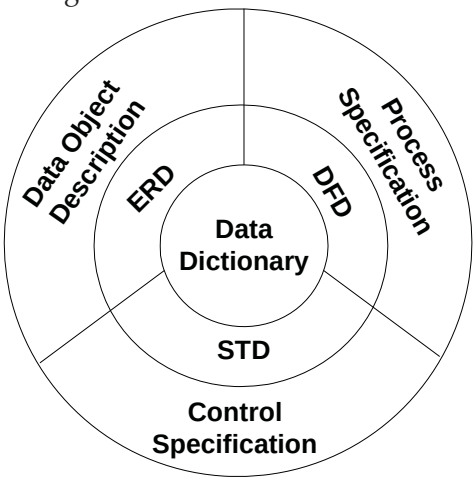
Tujuan tahap analisis adalah:
1. Menjabarkan kebutuhan pemakai.
2. Meletakkan dasar-dasar untuk proses perancangan PL.
3. Mendefinisikan semua kebutuhan pemakai sesuai dengan lingkup kontrak yang disepakati kedua belah pihak.

Aktivitas yang dilakukan:
1. Pendefinisian lingkup perangkat lunak,
2. Identifikasi dan pengumpulan kebutuhan perangkat lunak,
3. Pemodelan data,
4. Pemodelan fungsional,
5. Pemodelan status/kelakuan,

- Pendefinisian Lingkup Perangkat Lunak
  Pendifinisian lingkup perangkat lunak adalah aktivitas penyelidikan awal untuk menentukan rincian perangkat lunak yang akan dibangun, lingkungan luar tempat dimana sistem yang akan dibangun digunakan. Kegiatan pada tahap ini lebih kearah Memanajemen kegiatan pembangunan perangkat lunak. Lebih lanjut akan dipelajari pada matakuliah MPPL (Manajemen Proyek Perangkat Lunak) atau pada matakuliah PSPL (Pengembangan Sistem Perangkat Lunak).

- **Identifikasi dan Pengumpulan Kebutuhan Perangkat Lunak**
  Identifikasi dan pengumpulan kebutuhan perangkat lunak adalah aktifitas untuk mencari, mengidentifikasi, mengumpulkan, dan menentukan kebutuhan dari perangkat lunak yang akan dibangun dengan cara melakukan komunikasi dengan pengguna atau *customer*. Lebih lanjut akan dipelajari pada matakuliah MPPL (Manajemen Proyek Perangkat Lunak) / PSPL (Pengembangan Sistem Perangkat Lunak).

- Pemodelan Data
  Pemodelan data befungsi untuk mendeskripsikan data yang terlibat dalam perangkat lunak. Secara struktural pemodelan data ini dapat digambarkan sebagai berikut:



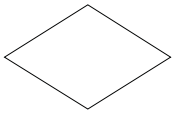
Struktur Pemodelan Analisis

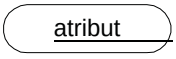
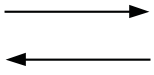
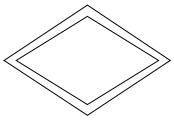
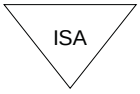
Piranti yang digunakan untuk menjelaskan pemodelan data, yaitu:

1. ERD (*Entity Relationship Diagram*): Merupakan diagram yang menyatakan keterhubungan antar objek data.
2. DOD (*Data Object Description*): Merupakan deskripsi atribut dari setiap objek data.
3. Kamus Data (*Data Dictionary*): Deskripsi semua objek data yang dibutuhkan maupun yang dihasilkan oleh perangkat lunak.

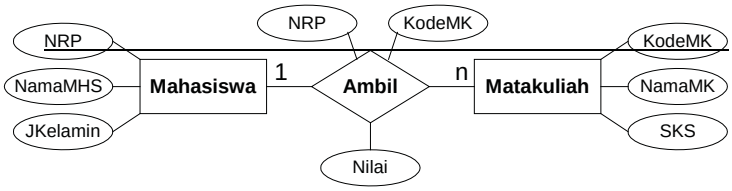
1. ERD (Entity Relationship Diagram)

Diagram Relasi Entitas (ERD-*Entity Relationship Diagram*) adalah suatu diagram yang menggambarkan relasi atau hubungan antar objek. Relasi antar objek dihubungkan dengan garis, ada banyak relasi, diantaranya adalah hubungan satu ke banyak (*one to many relationship*) dan hubungan dari satu ke satu (*one to one relationship*). Diagram tersebut dinyatakan dalam simbol-simbol atau menggunakan notasi-notasi yang dapat dilihat pada tabel berikut.

Notasi	Nama	Arti
	Entity	Entitas eksternal, yaitu entitas luar yang berhubungan dengan sistem.
	Relation	Dalam ER-Diagram notasi ini berarti relasi yang menghubungkan dua atau lebih entitas.

1/n, 1/1	Cardinality	Menunjukan kardinalitas relasi antar entitas, 1/n berarti hubungan *one to many*, 1/1 berarti hubungan *one to one*.
	Attribute	Merupakan atribut dari suatu entitas atau relasi.
	Key Attribute	Merupakan atribut dari suatu entitas atau relasi yang menjadi primary key.
	Connector	Penghubung, untuk mengalirkan data dari satu bagian ke bagian lain sesuai arah panah.
	Low Relation	Relasi lemah, sangat tergantung dan hanya dapat muncul bila terdapat relasi yang menghubungkan ke entitas lemah.
	ISA	"is a", kumpulan entitas yang memuat bagian atau cabang entitas yang berbeda.

Contoh:



Rincian pada gambar di atas terdapat sebagai berikut:

1. Entitas Mahasiswa: Atribut (NRP, NamaMHS, JKelamin)
2. Entitas Matakuliah: Atribut (KodeMK, NamaMK, SKS)
3. Relasi Meminjam: Atribut (NRP, KodeMK, Nilai)
4. Kardinalitas 1/n: Seorang mahasiswa dapat mengambil 1 atau lebih matakuliah.

Penggambaran Atribut pada ER Diagram dapat dihilangkan, dengan memberikan keterangan tambahan berupa paparan atribut pada setiap entitas dan relasi yang muncul.

2. DOD (Data Object Description)

Deskripsi Objek Data (DOD-*Data Object Description*) merupakan bagian dari ERD (*Entity Relational Diagram*) yang telah dirancang. DOD menyimpan keterangan semua atribut entitas dan relasi yang muncul pada tahap perancangan ERD. DOD dapat direpresentasikan dalam bentuk tabel.

Contoh:

Atribut	Tipe	Deskripsi
NRP	Numerik	Nomor identitas mahasiswa yang nilainya unik.
NamaMHS	Karakter	Nama ahasiswa sesuai dengan format nama yang tertulis di ijazah sekolah
JKelamin	Boolean	Jenis Kelamin mahasiswa, 0 wanita, 1 pria
KodeMK	Karakter	Kode nama matakuliah sesuai kurikulum
NamaMK	Karakter	Nama matakuliah sesuai kode matakuliah pada kurikulum
SKS	Numerik	Jumlah kredit matakuliah
Nilai	Karakter	Nilai matakuliah yang diperoleh mahasswa

3. Data Dictionary

Menyimpan semua objek data yang dibutuhkan dan dihasilkan oleh perangkat lunak. Biasanya pembuatan kamus data dilakukan setelah Pemodelan fungsional dan pemodelan status dan kelakuan selesai dibuat.

A. Pemodelan Fungsional

Pemodelan Fungsional adalah Mendeskripsikan seluruh fungsi yang terlibat di dalam perangkat lunak. Piranti yang digunakan pada pemodelan fungsional adalah:

1. *Context Diagram,* Merepresentasikan sistem sebagai sebuah *black box* terhadap lingkungan sekitar yang berhubungan dengan sistem tersebut.

2. DFD (*Data Flow Diagram*), Menggambarkan bagaimana data ditransformasikan dalam perangkat lunak serta menggambarkan fungsi-fungsi yang mentransformasikan data.
3. *Process Specification,* Merupakan deskripsi detil dari fungsi elementer (fungsi yang tidak dapat didekomposisi lagi dalam DFD).

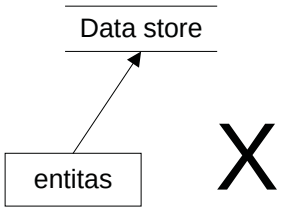
Informasi (data) bergerak mengalir dalam perangkat lunak. Informasi atau data tersebut dapat mengalami transformasi di dalam perangkat lunak. *Data Flow Diagram* adalah representasi grafis yang menggambarkan aliran dan transformasi informasi/data dari *input* ke *output*.

- **Context Diagram**

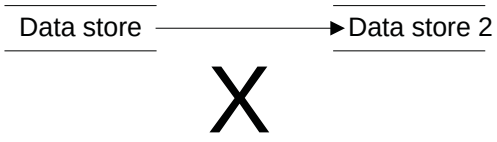
Diagram Konteks digunakan untuk membatasi sistem penyampaian informasi yang akan dirancang dan menunjukan interaksi sistem dengan komponen luar. Diagram tersebut dinyatakan dalam simbol-simbol pada DFD.

Ada beberapa batasan yang harus diperhatikan dalam membuat dan merepresentasikan diagram konteks, yaitu:

1. Entity tidak dapat berhubungan (bertukar data) langsung dengan data store, harus melalui suatu proses.



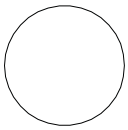
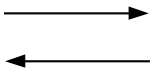
2. Antardata store tidak boleh bertukar data, harus melaui proses.



3. Data store harus ada yang mengisi dan yang memanfaatkan. Tidak boleh hanya diisi saja atau dimanfaatkan saja, harus ada proses yang mengisi dan memanfaatkannya.

- **Data Flow Diagram**

Diagram Alir Data atau DFD (*Data Flow Diagram*) merupakan penjelasan rinci dari Diagram Konteks yang menggambarkan bagaimana proses aliran data terjadi dalam sistem Online Buku Elektronik. Data Flow Diagram menjelaskan tentang aliran data masuk, data keluar dan proses penyuntingan file yang digunakan. Diagram tersebut dinyatakan dalam simbol-simbol atau menggunakan notasi-notasi yang dapat dilihat pada tabel Simbol. Penjelasan DFD terdiri dari level-level, masing-masing level menjelaskan level sebelumnya.

Notasi	Nama	Arti
	Entity	Entitas eksternal, yaitu entitas luar yang berhubungan dengan sistem.
	Frequent Entity	Entitas eksternal, yaitu entitas luar yang sama dan berhubungan dengan system digambarkan secara berulang.
	Data Process	Lingkaran dengan nama proses di dalam lingkaran menyatakan proses-proses yang terdapat didalam sistem.
	Data Store	Simbol yang menyatakan penyimpanan data yang digunakan oleh sistem, nama data terdapat diantara dua garis tersebut.
	Connector	Penghubung, untuk mengalirkan data dari satu bagian ke bagian lain sesuai arah panah.

- **Process Specification**

Spesifikasi Proses atau *Process Specification* termasuk dalam SRS (*Software Requirements Specification*) merupakan deskripsi detil dan lengkap dari fungsi elementer. Fungsi

Elementer adalah fungsi-fungsi atau proses-proses yang tidak dapat didekomposisi lagi dalam Diagram Alir Data atau DFD (*Data Flow Diagram*).

B. Pemodelan Status/ Kelakuan

Pemodelan Status / Kelakuan adalah tahapan analsis untuk mendeskripsikan status sistem yang dapat muncul ketika perangkat lunak digunakan. Contoh mengenai pemodelan status kelakuan dapat dilihat pada subjudul Studi Kasus.

- **Tahap Perancangan / Design**

Tahap desain merupakan tahap rekayasa yang merepresentasikan perangkat lunak yang akan dibangun. Desain dapat digunakan untuk menelusuri dan mengecek kebutuhan *customer* dan sekaligus dapat dijadikan sebagai ukuran untuk menilai kualitas perangkat luak.

Bila kita pernah mendengar cetak biru dari sebuah gedung yang dapat digunakan sebagai dasar pembangunan gedung, pedoman penelusuran sesuatu bila dibutuhkan nanti, dan pedoman untuk melakukan perbaikan dan renovasi, maka begitu pula dengan perangkat lunak. Perangkat lunak lebih komplek dari sebuah rumah besar. Maka dari itu, perangkat lunak juga dan bahkan sangat membutuhkan cetak biru bangunan perangkat lunak, yaitu desain atau perancangan. Fungsi tahap perancangan perangkat lunak adalah:

1. Pengembangan spesifikasi perangkat lunak.
2. Penjabaran *bagaimana* PL dapat berfungsi.
3. Penjabaran *bagaimana* spesifikasi perangkat lunak dapat diimplementasikan

Selama proses perancangan, kualitas perancangan selalu dipantau melalui 'Review Teknis Formal' dan dibahas bersama antara *customer* dan pengembang.

Menurut McGlaughlin, petuntuk untuk evaluasi kualitas perancangan, yaitu sebagai berikut:

1. Perancanan harus mengimplementasikan semua kebutuhan perangkat lunak yang disebut eksplisit dalam SRS (*software requirements specifications*), sekaligus mengakomodasi semua kebutuhan implisit dari SRS.
2. Harus *readabel* (mudah dibaca dan dipahami) oleh *programmer, tester*, dan pelaku perawatan perangkat lunak.
3. Harus menyediakan gambaran lengkap perangkat lunak, meliputi model data, fungsi, dan kelakuan perangkat lunak dari sudut pandang implementasi.

Adapun *prinsip-prinsip perancangan* perangkat lunak adalah:

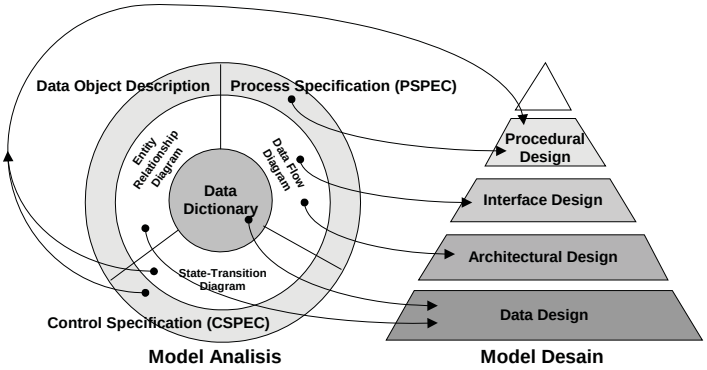
1. Mempertimbangkan beberapa alternatif model solusi.
2. *Traceable* (dapat dicek dan dirunut) terhadap model analisis.
3. Mempertimbangkan dan menghasilkan komponen yang dapat digunakan ulang (*reusable*).
4. Meminimasi kesenjangan antara perangkat lunak dengan kondisi nyata.
5. Memperlihatkan keseragaman perancangan.
6. Memperlihatkan integerasi.
7. Mengakomodasi perubahan.
8. Mengakomodasi kondisi-kondisi insedental yang mingkin muncul.
9. Abstraksi yang lebih detil dari analisis, tetapi lebih

tinggi (general) dari *coding*.
10. Dapat terukur kualitasnya.
11. Harus di-*review* untuk meminimasi kesalahan semantik.

Tahapan perancangan (desain), yaitu:
1. Perancangan Data.
2. Perancangan Arsitektur.
3. Perancangan Antarmuka.
4. Perancangan Prosedur.

Dengan melakukan tahapan-tahapan tersebut, akan dihasilkan dokumen perancangan (*Software Design Document = SDD*), yang berisi:
1. Ruang lingkup
2. Prancangan data
3. Perancangan Arsitektur
4. Perancangan antarmuka
5. Perancangan prosedur
6. Kebutuhan lain
7. Persiapan pengujian
8. Catatan khusus

Hubungan tahap Analisis dengan tahapan Perancangan menggunakan Metode Terstruktur dapat digambarkan sebagai berikut:



1. Perancangan Data

Perancangan data mentransformasikan model domain informasi yang dibangun dalam tahap analisis ke dalam struktur data yang akan dibutuhkan dalam implementasi (*coding*) perangkat lunak. Objek data dan relasi yang didefinisikan dalam ER diagram serta detil data yang dijabarkan di dalam kamus data merupakan basis pengembangan perancangan data ini.

Dalam perancangan data, yang dilakukan adalah:

1. Pemilihan representasi lojik dari objek data yang ditemukan pada proses analisis.
2. Perbaikan (*refinement*) terhadap kamus data menjadi:
   - struktur data tertentu (array, list, dll.)
   - struktur file tertentu.
   - basis data lengkap dengan fieldnya.

Petunjuk teknis perancangan data:

1. Menerapkan prinsip-prinsip analisis sistematis (pada tahap analisis),
2. Mengidentifikasikan semua struktur data dan prosedur yang akan digunakan dalam pengaksesan

data tersebut.
3. Me-*refine* isi kamus data (*data dictionary*).
4. Menunda perancangan data yang "low level" sampai di akhir proses perancangan.
5. Merepresentasikan struktur data sedemikian rupa sehingga hanya modul yang menggunakan data tersebut yang dapat mengaksesnya.
6. Membangun pustaka untuk struktur data dan prosedur yang sering digunakan.

Hasil perancangan data adalah:
1. Struktur data yang siap deprogram,
2. Struktur basis data yang siap dibuat oleh pemrogram,
3. Prosedur atau operasi untuk pengaksesan data yang telah siap diprogram.

2. Perancangan Arsitektur

Perancangan Arsitektur merepresentasikan struktur data dan komponen program yang dibutuhkan untuk membangun sistem yang berbasis komponen.

Ketika kita berbicara tentang cetak biru sebuah bangunan, bangunan juga memiliki sebuah struktur arsitektur. Struktur arsitektur bangunan merepresentasikan cara mengintagrasikan komponen-komponen yang ada sehingga menjadi kesatuan yang kokoh. Struktur bangunan juga mempertimbangkan interaksinya dengan lingkungan sekitar. Struktur ini memudahkan pembangun untuk mengaplikasikan kebutuhan *customer* sehingga memenuhi keinginannya.

Begitu pula dengan perangkat lunak. Representasi struktur arsitektur perangkat lunak ini dapat mempermudah pengembang untuk:

1. Menganalisis efektivitas desain dalam memenuhi kebutuhan perangkat lunak yang telah ditentukan dalam tahap analisis.
2. Mempertimbangkan alternatif perubahan pada desain kebutuhan perangkat lunak. Dengan melihat struktur perangkat lunak dapat diketahui di modul apa yang perlu dilakukan modifikasi, dan modifikasi tersebut akan berpengaruh pada modul apa.
3. Meminimasi resiko pengembangan perangkat lunak.

Tujuan utama perancangan arsitektur adalah:

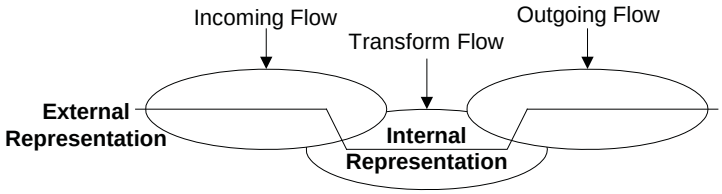
1. Membangun struktur program yang modular.
2. Merepresentasikan hubungan antarmodul (modul adalah kumpulan fungsional kebutuhan perangkat lunak yang relatif independent terhadap kumpulan fungsional kebutuhan perangkat lunak yang lain. Contoh, dalam stugi kasus membuat sistem Sistem Perpustakaan SMK TIKOM IBNU SIENA terdapat modul untuk menangani operasi dan transaksi terhadap buku, terdapat modul untuk menangani operasi dan transaksi terhadap anggota, dsb).
3. Memadukan struktur program dan struktur data.
4. Mendefinisikan antarmuka yang memungkinkan data dapat mengalir pada seluruh program.

Proses yang dilakukan yaitu mengubah aliran informasi (yang direpresentasikan dengan DFD) menjadi struktur perangkat lunak. Langkahnya:

1. Menentukan jenis aliran informasi (dalam sebuah perangkat lunak aliran transformasional dan transaksional dapat digunakan bersama-sama).
2. Menentukan batas aliran informasi.
3. Pemetaan DFD ke struktur program.

Jenis Aliran informasi:

1. Aliran Transformasional



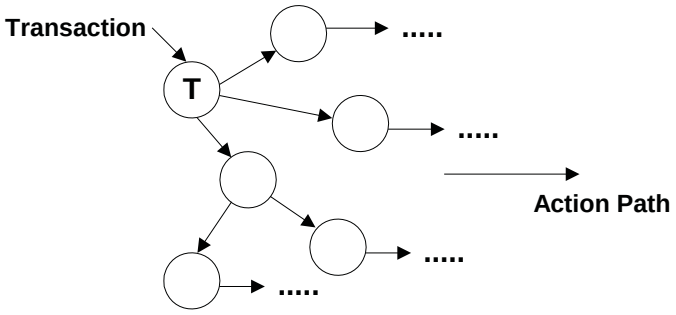
Ciri-ciri jenis aliran Transformasional:

1. Ada input dari entitas eksternal, lalu diproses dalam sistem, kemudian sistem menghasilkan output kembali ke entitas eksternal.
2. Dilakukan secara berurutan (sequensial)

Langkah pemetaan untuk jenis aliran Transformasional:

1. Kaji ulang model sistem dasarnya.
2. Kaji ulang dan perhalus DFD-nya.
3. Tentukan apakah DFD memiliki jenis aliran transaksional.
4. Isolasi pusat transaksi dengan menentukan batas aliran *incoming* dan *outgoing.*
5. Lakukan faktorisasi.
6. Perhalus struktur program.

2. Aliran Transaksional



Ciri-ciri jenis aliran Transaksional:
1. Terdapat input atau transaksi yang dapat memicu jalur aliran data yang lain.
2. Aliran data merupakan transaksi pemilihan jalur aksi informasi.

Langkah pemetaan untuk jenis aliran Transaksional:
1. Kaji ulang model sistem dasarnya
2. Kaji ulang dan perhalus DFD-nya
3. Tentukan pusat transaksi dan jenis aliran di sepanjang setiap jalur aksi.
4. Lakukan faktorisasi
5. Perhalus struktur program

3. Perancangan Antarmuka

Fokus perancangan antarmuka:
1. Antarmuka antar modul-modul perangkat lunak.
2. Antarmuka antara perangkat lunak dengan sumber informasi selain manusia.
3. Antarmuka dengan pengguna (manusia).

Jenis antarmuka yang diperlukan:
1. Antarmuka untuk input parameter proses -> layar.
2. Antarmuka untuk output proses -> layar
3. Antarmuka untuk input data -> layar maupun *parameter passing*
4. Antarmuka untuk output data -> layar maupun *parameter passing*
5. Antarmuka untuk pesan-pesan

Hal-hal yang perlu diperhatikan dalam merancang antarmuka di layar: (MK IMK)
1. Harus konsisten (warna, font, bahasa, layout, dsb).

2. Memberikan umpan balik ke pengguna.
3. Meminta verifikasi untuk semua aksi destruktif penting
4. Memungkinkan aksi reversal (balikan).
5. Mengurangi jumlah informasi yang harus diingat antar aksi.
6. Efisiensi dialog, gerak, dan pikiran pengguna.
7. Mengelompokkan aktivitas berdasarkan fungsi dan mengatur layar sesuai dengan pengelompokan tersebut.
8. Sediakan bantuan (*help*) yang mudah navigasinya dan bila memungkinkan help yang sensitive terhadap konteks pengguna meminta bantuan.
9. Perhatikan representasi data, sesuaikan dengan fungsi pengguna. Contoh pengguna level manajerial lebih membutuhkan diagram/grafik daripada detil data dalam tabel.
10. Pesan kesalahan harus spesifik dan berarti, beri saran juga.
11. Minimalisasi jumlah aksi masukan yang diperlukan.
12. Sesuaikan dengan kebiasaan/kebutuhan user, misalnya:
    - *clerk-keyboard*, manajer-*mouse*
    - *clerk*-input, pembuat keputusan – update/ delete

Perancangan antarmuka dapat dilakukan dengan:

1. Manual pada kertas
2. Dengan memanfaatkan CASE Tools (contoh AppModeller pada PowerDesigner).

Hasil perancangan antarmuka:

1. Definisi antarmuka modul yang siap untuk

diprogram.
2. Definisi/format rancangan layar yang siap diimplementasikan

4. Perancangan Prosedur

Tahapan ini merupakan tahapan terakhir dalam proses perancangan. Pada tahap ini dibangun algoritma (*pseudo-code, program design language*) yang siap diprogram dengan mengacu pada:

1. Struktur data yang dibuat pada perancangan data.
2. Struktur modul dan kendali perangkat lunak yang dibangun pada saat merancang struktur arsitektur perangkat lunak.
3. Struktur dan perancangan menu atau format tampilan layar yang diperoleh pada parancangan antarmuka.

- **Tahap Implementasi**

Tahapan Implementasi adalah tahapan dalam rekayasa perangkat lunak yang dilakukan setelah tahapan Analisis dan Perancangan telah sempurna, dalam artian sudah selesai dan telah siap untuk dibuat dalam sebuah sistem. Dalam tahapan implementasi, bagian terpenting yang perlu diperhatikan adalah: Pengujian (*Testing*), Konversi, Instalasi dan Pelatihan.

1. Pengujian

Testing adalah proses mengeksekusi keseluruhan program atau sistem secara intensif dengan maksud mencari kesalahan-kesalahan. Testing dilakukan tidak hanya untuk mendapatkan program yang benar, namun juga memastikan bahwa program tersebut bebas dari segala kesalahan-kesalahan dalam berbagai kondisi. Tahapan ini akan terpengaruh oleh tahapan Coding atau

pembuatan program, jika pada saat pengkodean dilakukan dengan baik, maka waktu yang digunakan untuk testing juga semakin sedikit.

Kategori pengujian pada perangkat lunak meliputi:

1. *Functional*, pengujian dilakukan untuk memastikan bahwa sistem dapat menjalankan fungsi secara normal. Dilakukan dengan cara memberikan input terhadap sistem dan meneliti output yang dihasilkan.
2. *Recovery*, pengujian dilakukan untuk memastikan bahwa sistem dapat melakukan recovery terhadap berbagai jenis kesalahan atau kegagalan. Dilakukan dengan cara mensimulasikan berbagai kesalahan atau kegagalan, seperti kegagalan listrik, sistem operasi dan lainnya.
3. *Performance*, pengujian dilakukan untuk memastikan bahwa sistem dapat memenuhi persyaratan kinerja yang sesuai.

Menurut Myers, jenis program testing dapat dibedakan menjadi:

1. Module Test, test permodul dalam lingkungan yang tertutup.
2. Integration Test, verifikasi antarmuka modul dan bagian system.
3. External Function Test, verifikasi fungsi-fungsi eksternal.
4. System Test, verifikasi dan valdasi kerja sistem dalam lingkungan yang dibuat secara khusus.
5. Acceptance Test, validasi sistem dan persyaratan pengguna.
6. Instalation Test, untuk mencari kesalahan yang dibuat pada proses instalasi.

7. Simulation Test, mencoba meniru keadaan pada lingkungan tempat sistem tersebut akan dijalankan.
8. Field Test, dilakukan pada lingkungan pemakaian pada kondisi yang sebenarnya.

2. Konversi

Konversi adalah suatu perubahan yang dapat meliputi berbagai hal, misalnya konversi program/sistem dan konversi data. Ada bebeapa hal yang perlu diperhatikan dalam konversi data dan konversi program, yaitu:

1. Konversi Data
    i. Dalam konversi data ada perubahan dari sistem lama ke sistem baru,
    ii. Perlu diingat bahwa data dalam sistem yang lama masih mungkin mengandung kesalahan,
    iii. Harus berhati-hati dengan adanya perubahan domain data pada sistem yang baru, misalnya: perubahan sistem pengkodean, perubahan range.
    iv. Pada umumnya ditempuh dengan cara membuat suatu program atau sistem khusus untuk keperluan konversi, akan tetapi pada banyak kasus tetap harus dibantu justifikasi oleh manusia.
2. Konversi Program
    a. Konversi jarang sekali dilakukan kecuali dalam situasi yang khusus.
    b. Biasanya dengan alasan (biaya dan sistem masih baik) bahwa ada bagian dari sistem lama yang harus diintegrasikan dengan atau ke sistem yang baru. Bagian-bagian yang bersangkutan tidak dirancang kembali.
    c. Diperlukan semacam justifikasi tingkat

kesulitan konversi. Ada bentuk konversi yang agak ringan, misalnya pemindahan jenis sistem operasi.

3. Instalasi

   Instalasi merupakan proses implementasi yang sangat penting karena sistem siap dicoba kedalam lingkungan yang baru. Ada beberapa hal yang perlu diperhatikan, yaitu:

   1. Mempersiapkan Sistem Penunjang, komponen yang terlibat adalah:
      a. Bangunan/gedung/ruangan.
      b. Sistem tenaga listrik.
      c. Sistem telekomunikasi (telepon, radio, satelit).
      d. Sistem kondisi lingkungan (ac).
      e. Sistem keselamatan kerja dan keamanan fisik (petir, banjir, kebakaran)
   2. Mempersiapkan Hardware, hal yang harus diperhatikan adalah:
      a. Disusun suatu prosedur instalasi yang dilengkapi jenis test pada setiap kegiatan.
      b. Harus ada Aceeptance Test yang disetujui oleh semua pihak yang terlibat.
      c. Jika perlu dapat dipergunakan diagnostic software.
   3. Mempersiapkan Software, hal yang harus diperhatikan adalah:
      a. sama dengan point a dan b dalam mempersiapkan hardware
      b. Perlu adanya pengukuran kinerja sistem secara keseluruhan(Benchmarking).

4. Pelatihan

Pelatihan adalah suatu usaha untuk memperkenalkan sistem yang baru ke klien atau suatu kelompok/organisasi. Dalam pelatihan ini perubahan-perubahan yang terjadi dalam sebuah sistem harus diketahui oleh orang yang akan menggunakan sistem tersebut.

BAB IV
STUDI KASUS TERSTRUKTUR: SISTEM PERPUSTAKAAN SMK TIKOM IBNU SIENA

Dengan bekal pemahaman sebelumnya, kita akan membangun sebuah perangkat lunak, yaitu Sistem Perpustakaan SMK TIKOM IBNU SIENA. Perangkat lunak ini berjalan di sebuah *stand alone* komputer. Metode pengembangan menggunakan Metode analisis dan perancangan Terstruktur dengan Model *Waterfall*.

4.1 Tahap Analisis

Aktivitas Analisis yang dilakukan untuk membuat Sistem Perpustakaan adalah:

1. Pendefinisian lingkup perangkat lunak

Lingkup perangkat lunak Sistem Perpustakaan SMK TIKOM IBNU SIENA ini adalah:

1. Perangkat lunak dikembangkan di Perpustakaan SMK TIKOM IBNU SIENA (*fiktif*).
2. Perangkat lunak dinamai SISPUS SMK TIKOM IBNU SIENA atau Sistem Informasi Perpustakaan SMK TIKOM IBNU SIENA.
3. Perangkat lunak dioperasikan pada *stand alone* komputer.

2. Identifikasi dan pengumpulan kebutuhan perangkat lunak

Kebutuhan perangkat lunak Perpustakaan SMK TIKOM IBNU SIENA (SISPUS SMK TIKOM IBNU SIENA = Sistem Informasi Perpustakaan SMK TIKOM IBNU SIENA):

1. Dapat mencatat dan mengolah transaksi peminjaman buku:
    a. Mencatat nomor anggota yang meminjam.
    b. Mencatat nomor buku yang dipinjam.
    c. Mencatat tanggal peminjaman.
    d. Saat peminjaman, ditampilkan keterangan buku yang akan dipinjam: Menampilkan nomor buku, judul, dan pengarangnya.
    e. Saat peminjaman, ditampilkan keterangan peminjam: Menampilkan nomor anggota, nama, dan alamat.
2. Dapat mencatat dan mengolah transaksi pengembalian buku:
    a. Mencatat nomor buku yang dikembalikan.
    b. Mencatat bahwa pengembalian telah dilakukan.
    c. Saat pengembalian, ditampilkan keterangan buku yang dikembalikan: Menampilkan nomor buku, judul, dan pengarangnya.
    d. Saat pengembalian, ditampilkan keterangan peminjam: Menampilkan nomor anggota, nama, dan alamat.
    e. Mengecek keterlambatan pengembalian: Mengecek apakah pengembalian terlambat. Bila terlambat, tampilkan jumlah hari keterlambatan.

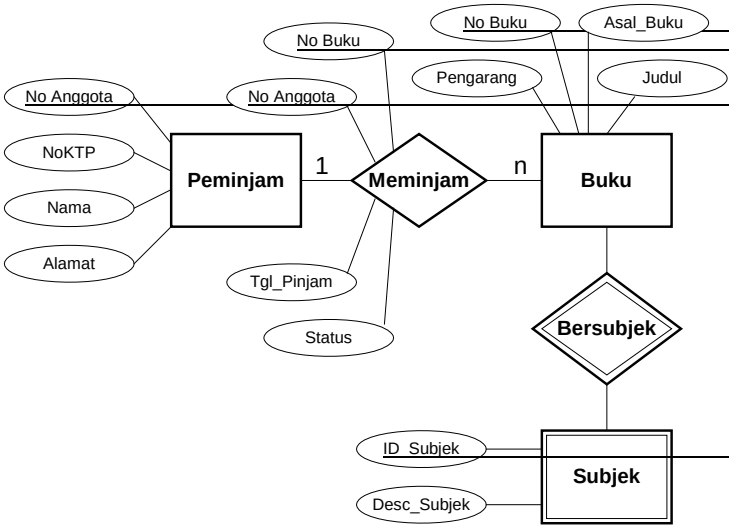
3. Dapat mencatat tambahan buku
    a. Mencatat nomor buku, judul, pengarang, dan asal buku: Bila nomor buku telah ada, tampilkan pesan bahwa nomor tersebut telah ada.
4. Dapat menghapus buku yang pernah ada (hapus data buku)
    a. Dengan memberikan nomor buku, tampilkan data buku tersebut.
    b. Dapat menghapus keberadaan buku dari basis data.
5. Dapat menambah anggota/peminjam
    a. Mencatat nomor anggota, nomor KTP, nama, dan alamat: Bila nomor anggota telah ada, tampilkan pesan bahwa nomor tersebut telah ada.
6. Dapat menghapus anggota/peminjam
    a. Dengan memberikan nomor anggota, tampilkan data anggota: Bila nomor anggota tidak ada, tampilkan pesan bahwa nomor tersebut tidak ada.
    b. Dapat menghapus anggota dari basisdata.
7. Dapat mencari data anggota berdasar nomor.
8. Dapat mencari data anggota berdasar nama.
9. Dapat mencari data buku berdasar nomor.
10. Dapat mencari data buku berdasar judul.
11. Dapat mencari data buku berdasar pengarang.
12. Dapat mencari data buku berdasar subjek.
13. Dapat mencari data buku yang sedang dipinjam beserta peminjamnya (dan keterangan keterlambatan pengembalian).

Asumsi

Data subjek buku pada basisdata sesuai dengan standart penomoran yang digunakan. Tabel subjek (ID_ subjek, Desc_Subjek) telah terisi.

3. Pemodelan data

a. Entity Relationship Diagram:



b. Data Object Description

Atribut	Tipe	Deskripsi
No_anggota	Alpha numerik	Merupakan identitas anggota yang nilainya unik. Format penomoran : (Huruf pertama nama anggota) + (nomor)
No_KTP	Karakter	Sesuai dengan format nomor KTP di Indonesia (gabungan angka dan titik)

Nama	Karakter	Nama anggota
Alamat	Karakter	Alamat tempat tinggal anggota
Tgl_pinjam	Date	Tanggal peminjaman buku
Status	Boolean	Merupakan keterangan apakah buku telah dikembalikan. True = buku telah dikembalikan False = buku masih dipinjam
No_buku	Alpha numerik	Format penomoran menganut format standart penomoran buku UDC
Judul	Karakter	Judul buku
Pengarang	Karakter	***Nama pengarang buku***
Asal_buku	Karakter	Sumber perolehan buku, merupakan keterangan tambahan. Contoh : beli, hadiah dari Bpk A dsb
ID_subjek	Numerik	Merupakan bagian dari penomoran buku (substring) yang memiliki makna penomoran subjek buku
Desc_subjek	Karakter	Keterangan subjek atau nama subjek

c. Data Dictionary

1. Nama : Data Buku

Alias : -

Deskripsi : Merupakan data tentang buku-buku perpustakaan, berisi:

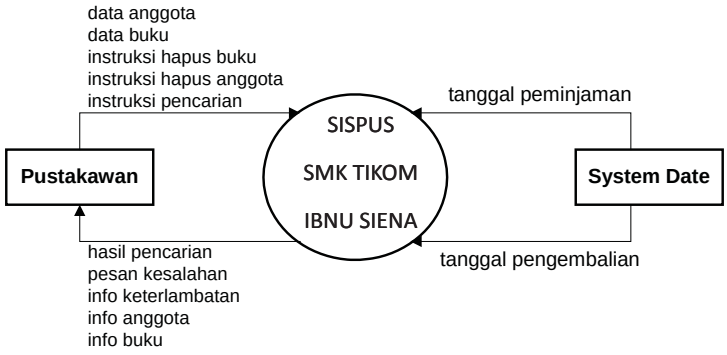
nomor buku = (nomor subjek-nomor subsubjek-nomor urut buku)

nomor subjek	= 1 – 100
nomor sub-subjek	= 1 – 100
nomor urut buku	= 1 - ~
judul	= karakter
pengarang	= karakter
asal buku	= katakter

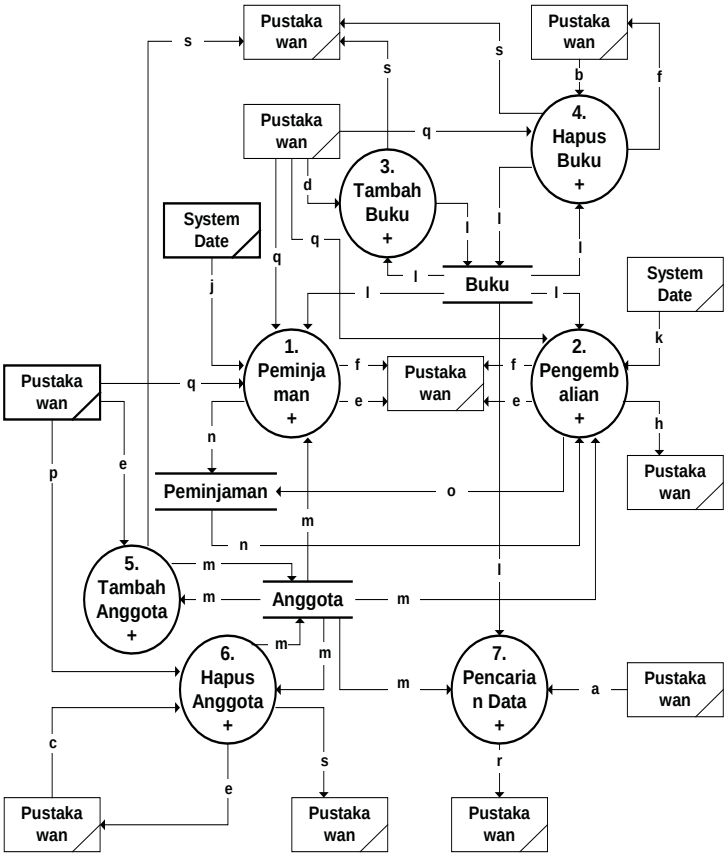
2. dan seterusnya
   Lanjutkan sendiri untuk semua data yang terlibat dalam sistem, berikut field-fieldnya, keterangan mengenai tipe data. Penulisan Kamus Data dapat dilakukan dengan banyak cara, salah satunya adalah point 1 diatas, contoh lainnya akan anda pelajari pada matakuliah Rekayasa Perangkat Lunak.

4. Pemodelan fungsional

a. Diagram Context

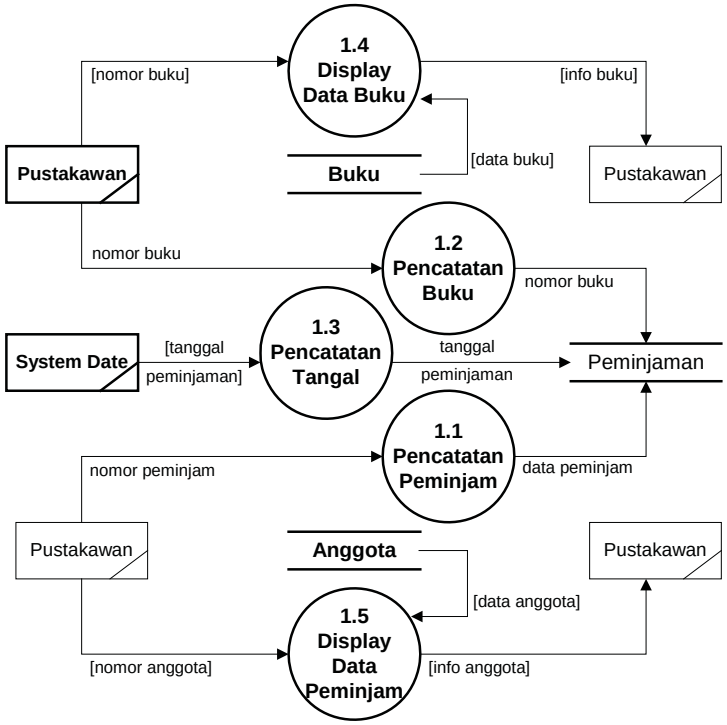


b. Data Flow Diagram:
DFD Level-1



Keterangan:
a : [instruksi pencarian]
b : [instruksi hapus buku]
c : [instruksi hapus anggota]
d : [data buku]
e : [data anggota]
f : [info buku]
g : [info anggota]
h : [info keterlambatan]
i : [info pengembalian]
j : [tanggal pinjam]
k : [tanggal kembali]
l : data buku
m : data anggota
n : data peminjaman
o : data pengembalian
p : nomor anggota
q : nomor buku
r : hasil pencarian
s : pesan kesalahan

DFD Level-2: Dekomposisi Proses 1, Proses Peminjaman



DFD Level-2: Dekomposisi Proses 2, Proses Pengembalian

Silahkan Anda melanjutkan sendiri, Dekomposisi Proses 2 sampai dengan Dekomposisi Proses 7. Caranya sama dengan Dekomposisi yang dilakukan pada tahap Dekomposisi Proses 1 untuk Proses Peminjaman. Proses dekomposisi ini terus dilakukan terhadap semua proses, hingga tidak ada lagi proses yang belum tergantikan. Tata cara atau batasan mengenai pembuatan diagram alir data atau DFD hampir sama dengan batasan pembuatan Diagram Context, karena diagram konteks dapat disebut sebagai DFD Level 0.

c. Process Specification

Proses-1: Peminjaman

1. Proses 1.1, 1.2, 1.3: Input Peminjaman

   Proses 1.1, 1.2, 1.3 disatukan karena dapat diinstruksikan dalam sebuah perintah SQL

   Input:

   Nomor buku, nomor anggota, tanggal peminjaman

   Begin

   {insert nilai ke basis data (tabel peminjaman) dengan nilai nomor buku yang dipinjam, nomor anggota peminjam, tanggal peminjaman yang diperoleh dari sistem, dan sebuah atribut bahwa buku sedang dipinjam (status=false)}

   End

2. Proses 1.4: Display Data Buku

   Input:

   Nomor buku

   Output:

   Hasil seleksi dari basis data yang ditampilkan ke layar

   Begin

   //Masukkan nomor buku yang dipinjam

   //Select (nomor buku, subjek, judul, pengarang) dari basisdata (tabel buku, dan tabel subjek) sesuai dengan nomor buku yang dipinjam. Subjek didapat dari substring nomor buku yang menyatakan subjek

   //Tampilkan hasil select tabel ke layar

   End

3. Proses 1.5: Display Data Peminjam

Input:

Nomor anggota

Output:

Hasil seleksi dari basis data yang ditampilkan ke layar

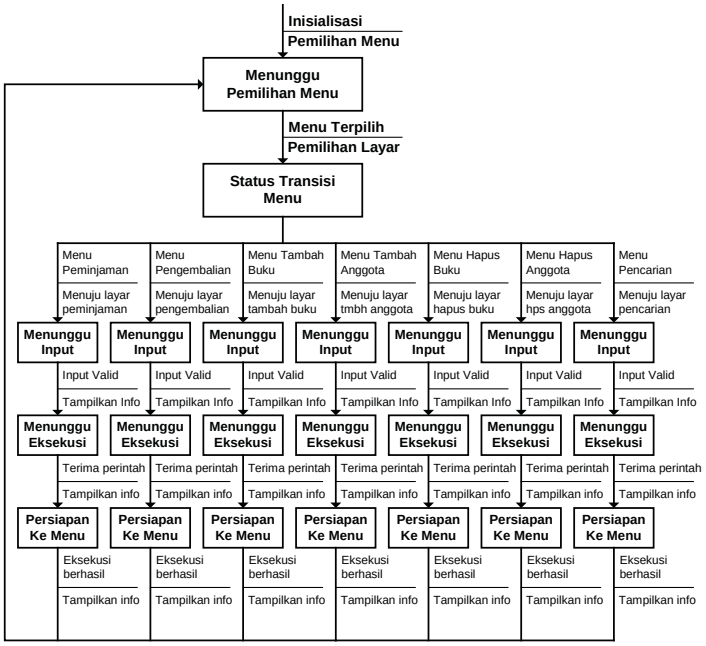
Begin

//Masukkan nomor anggota (peminjam)
//Select (nomor anggota, nama, alamat, nomor KTP) dari basisdata (tabel anggota) sesuai dengan nomor anggota
//Tampilkan hasil select tabel ke layar

End

Proses spesifikasi untuk Pengembalian, Tambah Buku, Hapus Buku, Tambah Anggota, Hapus Anggota, Pencarian dapat dilakukan apabila proses-proses tersebut telah di Dekomposisi hingga akhir (tidak ada lagi proses yang belum tergantikan) pada tahapan pembuatan DFD.

5. Pemodelan status/kelakuan



4.2 Tahap Perancangan

Aktivitas Perancangan yang dilakukan untuk membuat Sistem Perpustakaan adalah:

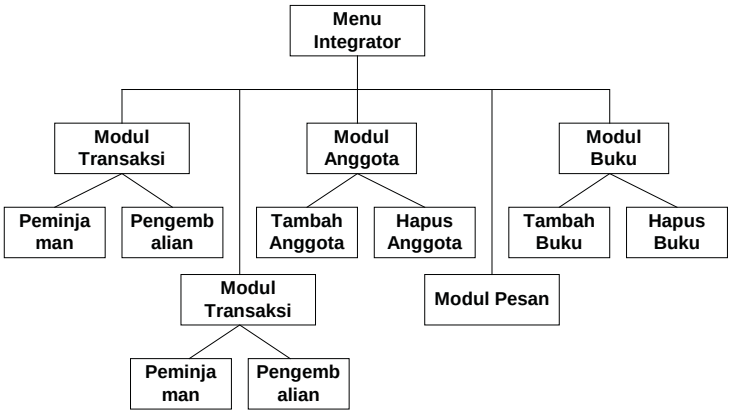
1. Perancangan Data

Nama Tabel	: Buku
Kegunaan	: Menyimpan nama tabel-tabel dan halaman query
Media Penyimpanan	: Harddisk
Field Kunci	: id

No.	Field Name	Type	Size	Note
1	Id	integer	-	NOT NULL auto_ increment
2	No_buku	integer	10	NOT NULL

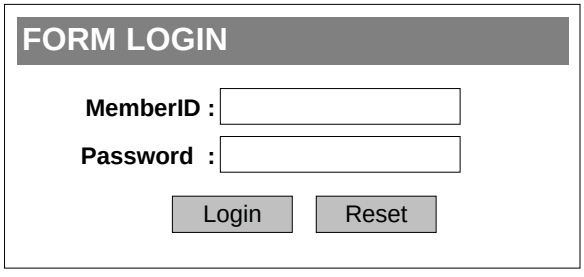
3	Judul	varchar	50	NOT NULL
4	Pengarang	varchar	30	NOT NULL
5	Asal buku	varchar	10	NOT NULL
6	Status	integer	6	NOT NULL

2. Perancangan Arsitektur



3. Perancangan Antarmuka

Perancangan antarmuka adalah bagian terpenting dalam pengembangan perangkat lunak, karena bagian ini sebagai tempat sistem berkomunikasi dengan pengguna. Tata cara perancangan antarmuka yang baik dapat anda pelajari pada matakulah Interaksi Manusia dan Komputer. Contoh dari rancangan antarmuka adalah sebagai berikut:



4. Perancangan Prosedur

Perancangan Prosedur dalam sistem dibuat berdasarkan Struktur Menu dan Struktur Program Sistem Perpustakaan SMK TIKOM IBNU SIENA. Perancangan Prosedur dibuat sedemikian rupa dan akan digunakan pada saat implementasi. Bagian ini dapat anda kembangkan sendiri.

Tugas:

Dengan cara yang sama, coba Anda buat ERD, Context Diagram dan DFD untuk kasus pengajuan KRS di STMIK DCI dari awal (ketika mengambil formulir) sampai akhir (ketika lembaran KRS di berikan ke dosen wali).

BAB V
ANALISIS & PERANCANGAN DENGAN PENDEKATAN BERORIENTASI OBJEK

Sebelum dimulai, perlu diingatkan kembali mengenai Rekaya Perangkat Lunak. Rekayasa Perangkat Lunak adalah Suatu disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal *requirement capturing* (analisa kebutuhan pengguna), *specification* (menentukan spesifikasi dari kebutuhan pengguna), *desain*, *coding*, *testing* sampai pemeliharaan sistem setelah digunakan. Pada bab sebelumnya kita mencoba memahami rekayasa perangkat lunak menggunakan pendekatan terstruktur (*Data Oriented Approach*). Sedangkan pada bab ini rekayasa perangkat lunak menggunakan pendekatan objek (*Object Oriented Approach*).

Pada pendekatan terstruktur ditemui banyak kelemahan dan kesulitan dalam merekayasa perangkat lunak dan tidak menggambarkan 'dunia nyata' dengan baik karena fungsi yang ada berorientasi pada fungsi saja dan tidak berhubungan langsung dengan permasalahan sehingga titik berat perancangan lebih kearah apa yang dibayangkan pengembang bukan apa yang diinginkan pengguna (*user's need expectation*). Karakteristik pendekatan terstruktur adalah sebagai berikut:

1. Penekanan pada sesuatu yang harus dikerjakan (algoritma pemacahan masalah), dimulai dari

menerima *input*, melakukan proses, menghasilkan *output*.

2. Program berukuran besar dipecah menjadi program-program yang lebih kecil.
3. Kebanyakan fungsi/prosedur berbagi data global.
4. Data bergerak secara bebas dalam sistem, dari satu fungsi ke fungsi lain yang terkait.
5. Fungsi-fungsi mentransformasi data dari satu bentuk ke bentuk yang lain.
6. Pendekatan yang digunakan, yaitu pendekatan atas ke bawah (*top down approach*)

Analisis dan Perancangan perangkat lunak dengan pendekatan objek atau dikenal dengan pendekatan berorientasi objek (*Object Oriented Approach*), yaitu *pendekatan untuk pengembangan perangkat lunak yang menitik* beratkan permasalahan pada **abstraksi** objek-objek yang ada di dunia nyata. Abstraksi adalah menemukan serta memodelkan fakta-fakta dari suatu objek yang penting. Pendekatan ini digunakan oleh banyak pengembang sekitar awal tahun 1990-an.

Karakteristik pendekatan objek adalah sebagai berikut:

1. Pendekatan lebih pada data dan bukanya pada prosedur/fungsi.
2. Program besar dibagi pada sesuatu yang disebut objek-objek.
3. Struktur data dirancang dan menjadi karakteristik dari objek-objek.
4. Fungsi-fungsi yang mengoperasikan data tergabung dalam suatu objek yang sama.
5. Data tersembunyi dan terlindung dari fungsi/ prosedur yang ada di luar.

6. Objek-objek dapat saling berkomunikasi dengan saling mengirim message (pesan) satu sama lain.
7. Pendekatan yang digunakan, yaitu pendekatan bawah ke atas (*bottom up approach*)

Secara umum, pendekatan terstruktur lebih sesuai untuk pengembangan aplikasi-aplikasi ilmiah (*scientific application*) karena memiliki fungsi-fungsi yang relatif stabil (ditentukan oleh hukum atau formula yang nilainya tidak berubah, seperti grafitasi). Pendekatan ini kurang memuaskan jika diterapkan pada aplikasi bisnis (*business application*) karena fungsi-fungsi didefinisikan oleh manusia yang bersifat subjektif dan berubah-ubah setiap waktu.

Solusi untuk permasalahan diatas adalah menggunakan pengembangan berorientasi objek dimana fungsi-fungsi disetarakan dan disatukan menjadi sebuah objek. Sebuah Objek adalah entitas yang menggabungkan data serta fungsi pada dirinya sendiri. Dengan cara ini pengembang dapat menghasilkan PL yang fleksibel, modifikatif, dan mudah dipelihara.

5.1 Konsep Berorientasi Objek

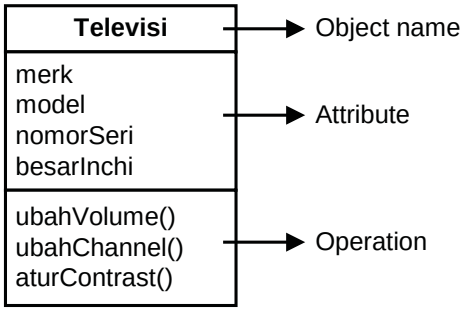
Untuk memahami titik pandang dan maksud dari 'berorientasi objek', kita dapat mempelajarinya dari alam secara luas. Objek ada di sekeliling kita, baik yang konkret atau konseptual. Dalam sudut pandang Eksekutif perusahaan: Karyawan, Absensi, Gaji, Profit dapat disebut sebagai Objek. Seorang Arsitek melihat Gedung, Biaya dan tenaga kerja sebagai objek. Konsep-konsep dasar dalam memahami Objek dapat dilihat pada subjudul berikut:

1. Object / Objek

Objek adalah orang, tempat, benda, kejadian atau konsep-konsep yang ada di dunia nyata dan penting bagi suatu aplikasi. Sebuah objek adalah Entitas yang memiliki Identitas, State (keadaan sesaat) dan Behavior (perilaku).

State sebuah objek adalah kondisi objek tersebut yang dinyatakan dalam **Atribut** atau property. Behavior sebuah objek mendefinisikan bagaimana sebuah objek bertindak/bereaksi yang dinyatakan dalam **Operation**. Satu object dapat diturunkan menjadi object dalam bentuk lain, kemudian saling mengkait menyusun sesuatu yang lebih rumit.

Langkah pertama yang harus dilakukan dalam pengembangan PL berorientasi objek adalah melakukan Abstraksi, yaitu kegiatan atau suatu usaha untuk mengenali objek-objek dan mengelompokkannya kedalam suatu kelas. Misalkan objek Hewan: Unggas, Reptil, maka Unggas dan Reptil adalah kelas-kelas dalam objek Hewan. Tata cara atau notasi pembuatan entitas objek digambarkan sebagai berikut:



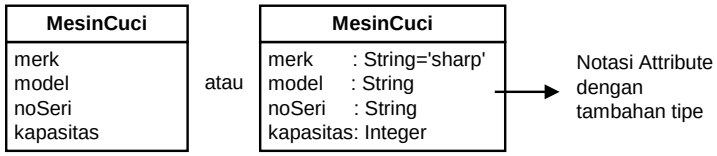
2. Class / Kelas

Class adalah kumpulan atau himpunan objek-objek yang sejenis, memiliki kesamaan atribut/property,

perilaku, serta relasi dengan objek lain yang mirip. Notasi kelas digambarkan dengan kotak, dengan nama kelas didalamnya ditulis menggunakan huruf besar di awal kata. Bila sebuah kelas memiliki 2 suku kata atau lebih, maka penulisannya disatukan tanpa spasi dengan huruf awal tiap suku menggunakan huruf besar.Contohnya adalah Barang Elektronik dapat dikatakan sebagai sebuah Kelas apabila memiliki kesamaan dengan objek yang ada padanya misalnya Mesin Cuci, Televisi, Radio, Kulkas adalah objek-objek yang dapat dikelompokkan kedalam satu kelas, yaitu Barang Elektronik rumah tangga.

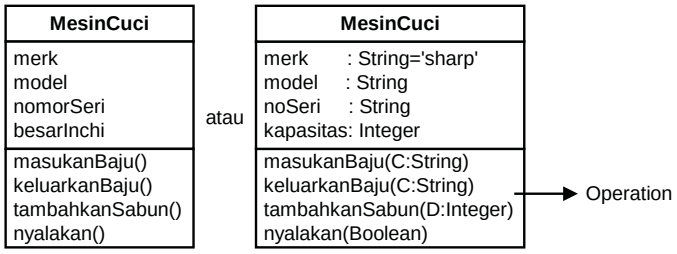
3. Attribute / Atribut

Attribute adalah data yang dimiliki suatu objek atau *property* dari sebuah *Class* yang menggambarkan batas nilai yang mungkin ada pada objek dari kelas. Sebuah bisa memiliki nol atau lebih atribut. Notasi atribut digambarkan dengan kotak dibawah kotak class, dengan nama atribut didalamnya ditulis menggunakan huruf kecil. Jika sebuah atribut memiliki 2 atau lebih suku kata, maka semua suku kata ditulis disatukan tanpa spasi, awal suku kata pertama dengan huruf kecil dan awal suku kata berikutnya dengan huruf besar. Notasi atribut dapat ditambahkan informasi dengan tipe-tipe atribut tersebut. Penulisan tipe pada atribut dipisahkan dengan tanda titik dua (:), tipe yang ditambahkan berupa String, Floating-Point number, Integer dan Boolean.



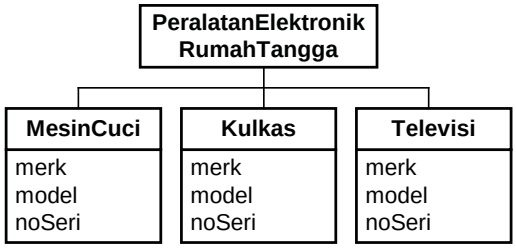
4. *Operation* / Operasi

Operation adalah sesuatu yang bisa dilakukan oleh sebuah *class*. Notasi penulisannya sama dengan atribut. Bagian operation ini juga bisa diberikan tambahan informasi, yaitu dengan menambahkan parameter yang akan dilakukan operation dalam tanda kurung. Contoh parameternya adalah *function*.



5. *Inheritance* / Pewarisan

Inheritance atau pewarisan memungkinkan dibuat class yang menyerupai class lain yang telah ada sebelumnya, tetapi masih memiliki beberapa sifat induknya. Misalkan dari sebuah mobil biasa, Anda dapat membuat mobil balap serta mobil angkutan umum. Prosesnya adalah dengan mengubah sifat dari mobil biasa tersebut.



6. *Polymorphisme* / Kebanyakrupaan

Polymorphism adalah *object* yang memiliki fungsi sama dengan *object* dasar tetapi memiliki satu atau lebih

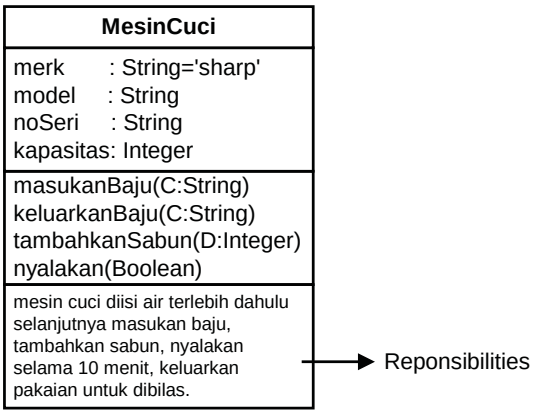
sifat berbeda atau dengan kata lain Polymorphisme adalah pemisahan secara jelas diantara subsistem yang berbeda. Sebagai contoh misalkan sebuah kelas memiliki operasi 'OPEN', operasi open ini bisa dipakai untuk membuka pintu, membuka buku, membuka baju dan lainnya. Meskipun 'OPEN' memiliki tujuan yang sama, tetapi apa yang dilakukannya berbeda.

7. *Encapsulation* / Pembungkusan

Encapsulation sering disebut dengan penyembunyian informasi (*Hidding*), suatu konsep berdasarkan fakta di dunia nyata yang menyatakan bahwa segala sesuatu tidak perlu diperlihatkan. Misalnya kita tidak perlu tahu apa yang dilakukan sistem ketika kita menekan remote untuk menghidupkan televisi.

8. *Responsibilities* / Tanggung Jawab

Responsibilities adalah model tambahan yang digambarkan pada bagian bawah suatu kelas setelah bagian operasi digunakan untuk menjelaskan pernyataan-pernyataan mengenai apa-apa yang bisa dilakukan oleh kelas tersebut.



5.2 Tahap Analisis

Apabila membangun suatu sistem baru, apa pun pendekatan yang digunakan (terstruktur/objek) harus melewati proses analisis. Tahapan analisis menggunakan pendekatan berorientasi objek dikenal dengan OOA (*Object-Oriented Analysis*). OOA adalah aktivitas teknik yang pertama kali dilakukan sebagai bagian dari rekayasa perangkat lunak berorientasi objek.

Ada 5 prinsip dasar OOA untuk membangun model analisis, yaitu:

1. Domain informasi dimodelkan.
2. Fungsi modul digambarkan.
3. Tingkah laku model direpresentasikan.
4. Model di partisi untuk mengekspos detail yang lebih besar.
5. Model awal merepresentasikan inti masalah, sedangkan model selanjutnya. memberikan detail implementasi.

Tujuan OOA adalah menentukan semua kelas (dan hubungan serta tingkah laku yang berkaitan dengannya) yang relevan dengan masalah yang akan dipecahkan.

Agar tujuan dari OOA ini terpenuhi, serangkaian tugas harus dilakukan, yaitu:

1. Persyaratan pemakai dasar harus dikomunikasikan antara *customer* dengan *enginer*.
2. Kelas-kelas harus didefinisikan (misalnya, atribut dan metode yang ditentukan).
3. Hierarki kelas harus dispesifikasikan.
4. Hubungan Objek-Ke-Objek (koneksi objek) harus direpresentasikan.
5. Tingkah laku objek dimodelkan.
6. Tugas 1 sampai 5 diaplikasikan lagi secara iterative sampai model selesai.

Masalah diuji dengan menggunakan model input-proses-output klasik (aliran data, sama seperti menggunakan metode terstruktur) atau dengan menggunakan model yang ditarik secara eksklusif dari struktur informasi hierarkis. Sampai bagian ini Konsep Analisis menggunakan pendekatan berorientasi objek (OOA) menjadi sulit dipahami karena TIDAK ADA kesepakatan Universal mengenai "Konsep" yang berfungsi sebagai dasar dari OOA. Hal ini dikarenakan terlalu banyak metode (konsep) yang bisa digunakan, yaitu:

1. Shlaer/Mellor Method [Shlaer-1988]
2. Booch Method [Booch-1991] / OOAD
3. Coad/Yourdan Method [Coad-1991]
4. OMT Method [Rumbaugh-1991]
5. Wirfs-Brock Method [Wirfs-Brock-1990]
6. OOSE Objectory Method [Jacobson-1992]
7. UML (Unified Modeling Language) [UML-1997]

Meskipun tidak ada kesepakatan mengenai konsep-nya, sasaran yang hendak dicapainya tetap sama. Sasaran OOA adalah mengembangkan sederetan model yang menggambarkan perangkat lunak komputer pada saat perangkat lunak tersebut berkerja untuk memenuhi serangkaian persyaratan yang ditentukan oleh pelanggan.

1. Metode Booch

Metode Booch terfokus pada proses pengembangan mikro dan proses pengembangan Makro. Tingkat mikro menentukan serangkaian tugas analisis yang diaplikasikan lagi untuk masing-masing langkah pada proses makro. Dengan demikian, pendekatan Evolusioner dijaga. Metode Booch didukung oleh berbagai piranti otomatis. *Outline* singkat dari proses pengembangan mikro OOA Booch

adalah sebagai berikut:

a. Identifikasi kelas dan objek
   III. Usulkan objek calon.
   IV. Lakukan analsis tingkah laku.
   V. Identifikasi scenario yang relevan.
   VI. Tentukan atribut dan operasi untuk amsing-masing kelas.

b. Identifikasi semantik dari kelas dan objek
   III. Pilih scenario dan analsis.
   IV. Tentukan tanggungjawab untuk mencapai tingkah laku yang diinginkan.
   V. Bagikan tanggungjawab untuk menyeimbangkan tingkah laku.
   VI. Tentukan objek dan sebutkan tugas dan tanggungjawabnya satu per satu.
   VII. Tentukan operasi untuk memenuhi tanggung jawab.
   VIII. Carilah kolaborasi diantara objek.

c. Identifikasi hubungan diantara kelas dan objek
   III. Tentukan ketergantungan yang ada di antara objek.
   IV. Deskripsikan peran masing-masing objek yang berpartisipasi.
   V. Validasi melalui scenario.

d. Lakukan sederetan penyaringan
   III. Hasilkan diagram yang sesuai untuk kerja yang dilakukan di atas.
   IV. Tentukan hierarki kelas yang sesuai.
   V. Lakukan pengikatan berdasarkan kelaziman data.

e. Implementasikan kelas dan objek (dalam konteks OOA, hal ini mengimplikasikan pelengkapan metoe analisis).

2. Metode Coad-Yourdan

   Metode Coad-Youdan adalah metoda OOA yang paling mudah dipelajari, karena Notasi pemodelannya relatif sederhana dan pedoman untuk mengembangkan model analisis tersebut jelas. Outline singkat mengenai proses OOA Coad-Yourdan adalah sebagai berikut:

   a. Identifikasi objek dengan menggunakan kriteria "apa yang dicari?"
   b. Tentukan struktur generalisasi-spesifikasi.
   c. Tentukan struktur keseluruhan bagian.
   d. Identifikasi subjek (representasi dari komponen subsistem).
   e. Tentukan atribut.
   f. Tentukan pelayanan.

3. Metode Jacobson

Metode Jacobson ada dua. yaitu versi lama disebut Objectory dan versi selanjutnya disebut OOSE (*Object Oriented Software Engineering*). OOSE adalah penyederhanaan dari metode Objectory. Metode OOSE difokuskan pada Use-Case, yaitu deskripsi atau scenario yang menggambarkan bagaimana pemakai berinteraksi dengan produk atau sistem. *Outline* singkat mengenai proses OOA Jacobson adalah sebagai berikut:

a. Identifikasi pemakai sistem dan seluruh tanggung jawab mereka.
b. Bangun model persyaratan
   III. Tentukan aktor dan tanggung jawab mereka.
   IV. Identifikasi *use-case* untuk setiap tingkah laku.
   V. Persiapkan pandangan awal mengenai objek dan hubungan system.
   VI. Kaji model dengan menggunakan use-case

sebagai scenario untuk menentukan validitas.

c. Bangun model analisis
   VII. Identifikasi objek *interface* dengan menggunakan informasi interaksi aktor.
   VIII. Ciptakan pandangan structural mengenai objek interface
   IX. Representasikan tingkah laku objek.
   X. Isolasi subsistem dan model untuk masing-masing.
   XI. Kaji model dengan menggunakan *use-case* seperti scenario untuk menentukan validitas.

4. Metode Rumbaugh

Rumbaugh dan rekan-rekan mengembangkan OMT (*Object Modelling Technique*) menggunakan desain sistem dan desain tingkat objek. Aktivitas analisis menciptakan tiga model:

1. Model Objek, representasi objek, kelas, hirarki dan hubungan.
2. Model Dinamis, representasi dari objek dan tingkah laku sistem
3. Model Fungsional, representasi aliran informasi seperti DFD tingkat tinggi melalui sistem tersebut.

Outline singkat mengenai proses OOA Rumbaugh adalah sebagai berikut:

a. Kembangkan pernyataan ruang lingkup masalah.
b. Bangun model objek
   XII. Identifikasi kelas yng relevan untuk masalah tersebut
   XIII. Tentukan atribut dan asosiasi
   XIV. Tentukan link objek
   XV. Organisasikan kelas objek dengan pewarisan

c. Kembangkan model dinamis
   XVI. Siapkan *scenario*
   XVII. Tentukan *event* dan kembangkan penelusuran event untuk masing-masing *scenario*
   XVIII. Buatlah diagram aliran *event*
   XIX. Kembangkan diagram keadaan
   XX. Kaji tingkah laku untuk konsistensi dan kelengkapannya
d. Buat model fungsional untuk sistem tersebut
   XXI. Identifikasi input dan *output*
   XXII. Gunakan diagram aliran data untuk merepresentasikan transformasi aliran
   XXIII. Kembangkan PSPEC (proses spesifikasi: ERD, DFD, Relationship) untuk masing-masing fungsi.
   XXIV. Tentukan batasan dan kriteria optimasi.

5. Metode Wirfs-Brock

Metode Wirfs-Brock tidak membuat perbedaan yang jelas antara analsis dan tugas desain. Metode ini mengusulkan proses kontinu mulai dengan penilaian terhadap spesifikasi pelanggan dan berakhir dengan desain. Outline singkat mengenai tugas yang berhubungan dengan analisis Wirfs-Brock adalah sbagai berikut:

a. Evaluasi spesifikasi pelanggan.
b. Berikan uraian gramatikal untuk mengekstrak kelas calon dari spesifikasi pelanggan.
c. Kelompokkan kelas dengan tujuan untuk mengidentifikasi superkelas.
d. Tentukan tanggung jawab untuk masing-masing kelas.
e. Identifikasi hubungan antar kelas.
f. Tentukan kolaborasi diantara kelas berdasarkan

tanggung jawab.

g. Bangun representasi hirarki kelas untuk memperlihatkan hubungan pewarisan.
h. Bangun grafik kolaborasi untuk system.

Masing-masing metode OOA di atas memiliki terminology dan langkah proses yang berbeda. Secara keseluruhan proses OOA-nya mirip. Untuk melakukan OOA, perekayasa perangkat lunak harus melewati langkah-langkah generik sebagai berikut:

1. Cari persyaratan pelanggan untuk sistem OOA.
2. Pilih kelas dan objek dengan menggunakan persyaratan dasar sebagai panduan.
3. Identifikasi atribut dan operasi untuk masing-masing objek system.
4. Tentukan struktur dan hierarki yang mengorganisasi kelas.
5. Bangun suatu model hubungan objek.
6. Bangun suatu model tingkah laku objek.
7. Kaji model analsis OO terhadap use-case scenario.

5.3 Tahap Perancangan

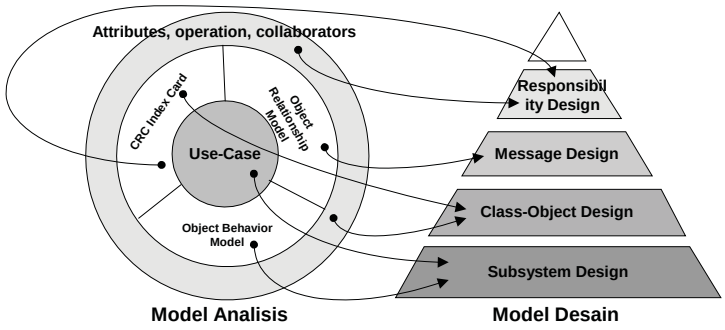
Tahapan perancangan menggunakan pendekatan berorientasi objek dikenal dengan OOD (*Object-Oriented Design*). OOD mentransformasikan model analsis yang dibuat menggunakan OOA ke dalam suatu model desin yang berfungsi sebagai cetak biru bangunan perangkat lunak. OOD menghasilkan desain yang mencapai sejumlah tingkatan yang berbeda dari modularitasnya. Komponen Mayor dikumpulkan (dienkasuplasi) didalam modul tingkat sistem yang disebut subsistem. Subsistem tersebut berisi Data, Operasi untuk memanipulasi data, Atribut,

Detail procedural operasi individu, dan Algoritma. Subsistem tersebut terangkum dalam OO, yang akhirnya menjadi sifat unik dari OOD.

Empat konsep desain perangkat lunak yang penting adalah:

1. Abstraksi.
2. Penyembunyian Informasi.
3. Independensi Fungsional.
4. Modularitas.

Hubungan tahap Analisis dengan tahapan Perancangan menggunakan Metode Berorientasi Object dapat digambarkan sebagai berikut:



1. Metode Booch

Metode Booch meliputi pendekatan proses pengembangan mikro dan proses pengembangan makro, seperti yang dijelaskan pada halaman 50. *Outline* singkat mengenai proses pendekatan mikro OOD Booch adalah sebagai berikut:

a. Perencanaan Arsitektur

XXV. Klusterkan/ satukan objek yang mirip didalam partisi arsitektur yang serupa.

XXVI. Lapiskan objek dengan tingkat abstraksi.
XXVII. Identifikasi scenario yang relevan.
XXVIII. Validasi prototype desain dengan mengaplikasikannya ke scenario kegunaan.

b. Desain Taktis
XXIX. Tentukan aturan domain independent (aturan yang mengatur penggunaan operasi dan atribut).
XXX. Tentukan aturan domain spesifik bagi pengaturan manajemen, penanganan kesalahan, dan fungsi infrastruktur yang lain.
XXXI. Kembangkan scenario yang menggambarkan semantik dari aturan
XXXII. Ciptakan *prototype* untuk masing-masing aturan.
XXXIII. Saringlah instrument dan prototype tersebut.
XXXIV. Kaji masing-masing aturan untuk memastikan bahwa aturan itu menyiarkan visi arsitekturnya.

c. Perencanaan Rilis
XXXV. Kumpulkan scenario yang dikembangkan selama OOA sesuai prioritas.
XXXVI. Alokasikan rilis arsitektur yang bersesuaian dengan scenario.
XXXVII. Rancang dan bangunlah masing-masing rilis arsitektur secara incremental.
XXXVIII. Sesuaikan tujuan dan jadwal rilis incremental sesuai kebutuhan.

2. Metode Coad-Yourdan

Metode Coad-Yourdan untuk OOD dikembangkan dengan mempelajari bagaimana "desainer OO yang efektif" melakukan kerja desain mereka. Pendekatan desain tersebut tidak hanya menyinggung aplikasi tetapi juga infrastruktur untuk aplikasi. Outline singkat proses OOD Coad-Yourdan aalah sebagai berikut:

a. Komponen Domain Masalah
   - XXXIX. Kumpulkan semua kelas spesifik domain.
   - XL. Rancang hierarki kelas yang sesuai untuk kelas aplikasi.
   - XLI. Bekerjalah untuk menyederhanakan pewarisan bila perlu
   - XLII. Saringlah desain untuk meningkatkan kinerja.
   - XLIII. Kembangkan suatu interface dengan komponen manajemen data.
   - XLIV. Saring dan tambahkan objek tingkat data yang diperlukan.
   - XLV. Kaji desain dan kemungkinan beberapa tambahan ke model analisis.

b. Komponen Interaksi Manusia
   - XLVI. Tentukan aktor manusia.
   - XLVII. Kembangkan scenario tugas.
   - XLVIII. Desain sebuah hirarki perintah pemakai.
   - XLIX. Saringlah urutan interaksi pemakai.
   - L. Rancanglah kelas-kelas yang relevan dan hirarki kelas.
   - LI. Integrasikan kelas-kelas GUI yang sesuai.

c. Komponen Manajemen Tugas
   - LII. Identifikasi tipe-tipe tugas (misalnya event yang dikendalikan, jam yang dikendalikan).
   - LIII. Buat prioritas.
   - LIV. Identifikasi suatu tugas untuk berfungsi sebagai koordinator bagi masing-masing tugas.
   - LV. Desain objek yang sesuai untuk masing-masing tugas.

d. Komponen Manajemen Data
   - LVI. Desain struktur data dan *layout*.
   - LVII. Desain pelayanan yang diperlukan untuk mengatur struktur data.
   - LVIII. Identifikasi piranti yang dapat membantu mengimplementasikan manajemen data.
   - LIX. Desain kelas-kelas yang sesuai hirarkinya.

3. Metode Jacobson

Aktivitas desain untuk OOSE merupakan versi sederhana dari metode Objectory. Model desain tersebut menekankan kemampuan penelusuran ke model analisis OOSE. *Outline* singkat mengenai proses OOD Jacobson adalah sebagai berikut:

a. Perhatikan penyesuaian untuk membuat model analsis yang diidealkan dapat memenuhi lingkungan dunia nyata.
b. Buat blok (abstraksi desain yang memperbolehkan representasi objek komposit) sebagai objek desain primer.
   - LX. Tentukan blok untuk mengimplementasikan objek analisis yang sesuai.
   - LXI. Identifikasi blok interface, blok entitas dan

blok control.

LXII. Gambarkan bagaimana blok berkomunikasi selama eksekusi.

LXIII. Identifikasi stimulus yang dilewatkan diantara blok-blok dan urutan komunikasi mereka.

c. Buat diagram interaksi yang memperlihatkan bagaimana stimulus di lewatkan diantara blok-blok.

d. Kumpulkan blok-blok kedalam subsistem

e. Kaji kerja desain

4. Metode Rumbaugh

Object Modelling Tecnique (OMT) meliputi aktivitas desain yang mendorong desain untuk dilakukan paa dua tingkat abstraksi yang berbeda. Desain sistem berfokus pada layout komponen yang diperlukan untuk membangun produk atau sistem yang lengkap. Desain objek menekankan layout detail dari suatu objek individu. Outline singkat mengenai proses OO Rumbaugh adalah sebagai berikut:

a. Lakukan desain sistem

LXIV. Partisi model analisis kedalam subsistem.

LXV. Identifikasi konkurensi yang ditentukan oleh masalah.

LXVI. Alokasikan subsistem ke prosesor dan tugas.

LXVII. Pilih strategi dasar bagi pengimplementasian manajemen data.

LXVIII. Identifikasi sumber daya global dan mekanisme kontrol yang diperlukan untuk mengakses mereka.

LXIX. Rancang mekanisme kontrol yang sesuai untuk sistem tersebut.

LXX. Perhatikan bagaimana kondisi batas harus ditangani.
LXXI. Kajilah dan perhatikan *trade-offs*

b. Lakukan desain objek
LXXII. Pilih operasi dari model analsis.
LXXIII. Tentukan algoritma untuk masing-masing operasi.
LXXIV. Pilih struktur data yang sesuai untuk algoritma.
LXXV. Tentukan setiap kelas internal.
LXXVI. Kajilah organisasi kelas untuk mengoptimalkan akses data dan tingkatkan efisiensi komputasi.
LXXVII. Rancanglah atribut kelas.

c. Implementasi mekanisime kontrol yang ditentukan di dalam desain sistem.
d. Sesuaikan struktur kelas untuk memperkuat pewarisan.
e. Rancanglah pemesanan untuk mengimplementasi hubungan objek.
f. Kemas kelas-kelas dan asosiasi kedalam modul.

5. Metode Wirfs-Brock

Wirfs-Brock mendefinisikan kontinum tugas0tugas teknis di mana analisis membawa kepada desain. Pokok-pokok singkat mengenai tugas-tugas yang berhubungan dengan desain Wirfs-Brock adalah sebagai berikut:

a. Konstruksikan protokol untuk masing-masing kelas
LXXVIII. Saring kontrak diantara objek-objek kedalam protokol tersaring
LXXIX. Rancang masing-masing operasi (tanggung jawab)
LXXX. Rancang masing-masing protokol (desain

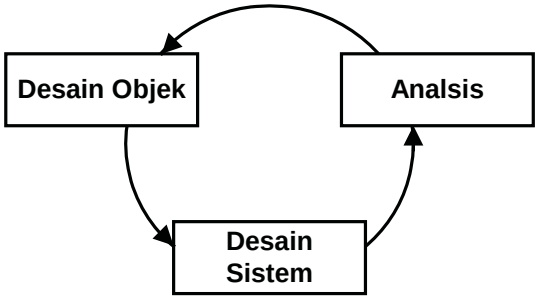
interface)

b. Buatlah spesifikasi desain untuk masing-masing kelas
   LXXXI. Gambarkan masing-masing kontrak secara detail
   LXXXII. Tentukan tanggung jawab privat
   LXXXIII. Spesifikasikan algoritma bagi masing-masing operasi
   LXXXIV. Catatlah pertimbangan dan batasan-batasan khusus
c. Buatlah spesifikasi desain untuk masing-masing subsistem
   LXXXV. Identifikasi semua kelas yang dienkapsulasi
   LXXXVI. Gambarkan kontrak secara detail dimana subsistem merupakan suatu server
   LXXXVII. Catatlah pertimbangan dan batasan-batasan khusus

Masing-masing metode OOD di atas memiliki terminologi dan langkah proses yang berbeda. Secara keseluruhan proses OOD-nya sangat konsisten. Untuk melakukan OOD, perekayasa perangkat lunak harus melewati langkah-langkah generic sebagai berikut:

1. Gambarkan masing-masing subsistem dengan cara yang dapat diimplementasikan.
   LXXXVIII. Alokasikan subsistem ke bagian pemrosesan dan tugas-tugas.
   LXXXIX. Pilih strategi untuk mengimplementasi manajemen data, dukungan interface, dan manajemen tugas
   XC. Rancang mekanisme kontrol yang sesuai untuk sistem tersebut
   XCI. Kajilah dan perhatikan *Trade-offs*

2. Desain objek:
   XCII. Desain masing-masing operasi pada suatu tingkat protokol
   XCIII. Tentukan setiap kelas internal
   XCIV. Desain struktur dan internal bagi atribut kelas
3. Desain pesan:
   Dengan menggunakan kolaborasi diantara objek dan hubungan objek, rancang model pemesanan.
4. Kajilah model desain dan iterasi sesuai kebutuhan.
   Aliran proses untuk OOD atau angkah-langkah desain yang dibahas diatas adalah intraktif; yaitu bahwa langkah-langkah tersebut dapat dieksekusi secara incremental sepanjang aktifitas OOD tambahan samai duatu desain yang lengkap dihasilkan. Adapun aliran proses untuk OOD digambarkan sebagai berikut:



5.4 Simpulan

Pada tahapan analisis, masing-masing metode OOA (*Object Oriented Analysis*) memiliki terminology dan langkah proses yang berbeda. Secara keseluruhan proses OOA-nya mirip.

Pada tahapan desain, masing-masing metode OOD (*Object Oriented Design*) memiliki terminology dan langkah

proses yang berbeda. Secara keseluruhan proses OOD-nya sangat konsisten.

Selain metode-metode OOAD yang telah dijelaskan di atas ada satu metode yang tidak bisa disebut sebagai "metode", yaitu UML (*Unified Modeling Language*). UML merupakan gabungan dari metode Booch, Rumbaugh (OMT) dan Jacobson. Tetapi UML ini akan mencakup lebih luas daripada OOAD. Pada pertengahan pengembangan UML dilakukan standarisasi proses dengan OMG (*Object Management Group*) dengan harapan UML akan menjadi bahasa standar pemodelan pada masa yang akan datang. UML disebut sebagai bahasa pemodelan bukan metode. Kebanyakan metode terdiri paling sedikit prinsip, bahasa pemodelan dan proses. Bahasa pemodelan (sebagian besar grafik) merupakan notasi dari metode yang digunakan untuk mendesain secara cepat. Ini yang akan kita pelajari pada bab selanjutnya.

BAB VI
UML (UNIFIED MODELING LANGUAGE)

- **Pengenalan UML**

UML (*Unified Modeling Lan*guage) merupakan pengganti dari metode analisis berorientasi object dan design berorientasi object (OOA & OOD) yang dimunculkan sekitar akhir tahun 80-an dan awal tahun 90-an.

UML merupakan gabungan dari metode Grady Booch (Booch Method), James Rumbaugh (OMT) dan Ivar Jacobson (OOSE). Tetapi UML ini akan mencakup lebih luas daripada OOA&D. Pada pertengahan pengembangan UML dilakukan standarisasi proses dengan OMG (*Object Management Group*) dengan harapan UML akan menjadi bahasa standar pemodelan pada masa yang akan datang.

UML disebut sebagai bahasa pemodelan bukan metode. Kebanyakan metode terdiri paling sedikit prinsip, bahasa pemodelan dan proses. Bahasa pemodelan (sebagian besar grafik) merupakan notasi dari metode yang digunakan untuk mendesain secara cepat. Bahasa pemodelan merupakan bagian terpenting dari metode. Ini merupakan bagian kunci tertentu untuk komunikasi. Jika anda ingin berdiskusi tentang desain dengan seseorang, maka Anda hanya membutuhkan bahasa pemodelan bukan proses yang digunakan untuk mendapatkan desain.

UML merupakan bahasa standar untuk penulisan

Blueprint Software yang digunakan untuk Visualisasi (*Visualize*), Spesifikasi (*Specify*), Pembentukan (*Construct*) dan Pendokumentasian (*Documentation*) alat-alat dari sistem perangkat lunak.

- **Sejarah UML**

UML dimulai secara resmi pada oktober 1994, ketika Rumbaugh bergabung dengan Booch pada Relational Software Corporation. Proyek ini memfokuskan pada penyatuan metode Booch dan OMT. UML versi 0.8 merupakan metode penyatuan yang dirilis pada bulan Oktober 1995. Dalam waktu yang sama, Jacobson bergabung dengan Relational dan cakupan dari UML semakin luas sampai di luar perusahaan OOSE. Dokumentasi UML versi 0.9 akhirnya dirilis pada bulan Juni 1996. Meskipun pada tahun 1996 ini melihat dan menerima feedback dari komunitas Software Engineering . Dalam waktu tersebut, menjadi lebih jelas bahwa beberapa organisasi perangkat lunak melihat UML sebagai strategi dari bisnisnya. Kemudian dibangunlah UML Consortium dengan beberapa organisasi yang akan menyumbangkan sumber dayanya untuk bekerja, mengembangkan, dan melengkapi UML.

Di sini beberapa partner yang berkontribusi pada UML 1.0, diantaranya Digital Equipment Corporation, Hewlett-Packard, I-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Relational, Texas Instruments dan Unisys. Dari kolaborasi ini dihasilkan UML 1.0 yang merupakan bahasa pemodelan yang ditetapkan secara baik, expressive, kuat, dan cocok untuk lingkungan masalah yang luas. UML 1.0 ditawarkan menjadi standarisasi dari Object Management Group (OMG). Dan pada Januari 1997 dijadikan sebagai standar

bahasa pemodelan.

Antara Januari–Juli 1997 gabungan *group* tersebut memperluas kontribusinya sebagai hasil respon dari OMG dengan memasukkan Adersen Consulting, Ericsson, ObjectTimeLimeted, Platinum Technology, Ptech, Reich Technologies, Softeam, Sterling Software dan Taskon. Revisi dari versi UML (versi 1.1) ditawarkan kepada OMG sebagai standarisasi pada bulan Juli 1997. Dan pada bulan September 1997, versi ini dierima oleh OMG Analysis dan Design Task Force (ADTF) dan OMG ArchitectureBoard. Dan Akhirnya pada Juli 1997 UML versi 1.1 menjadi standarisasi.

Pemeliharaan UML terus dipegang oleh OMG Revision Task Force (RTF) yang dipimpin oleh Cris Kobryn. RTP merilis editorial dari UML 1.2 pada Juni 1998. Dan pada tahun 1998 RTF juga merilis UML 1.3 disertai dengan user guide dan memberikan technical cleanup.

- **Pengertian UML**

UML adalah bahasa untuk menspesifikasi, memvisualisasi, membangun dan mendokumentasikan artifacts (bagian dari informasi yang digunakan atau dihasilkan oleh proses pembuatan perangkat lunak, *artifact* tersebut dapat berupa model, deskripsi atau perangkat lunak) dari sistem perangkat lunak, seperti pada pemodelan bisnis dan sistem non perangkat lunak lainnya [HAN98]. Selain itu UML adalah bahasa pemodelan yang menggunakan konsep orientasi object. UML dibuat oleh Grady Booch, James Rumbaugh, dan Ivar Jacobson di bawah bendera Rational Software Corp [HAN98]. UML menyediakan notasi-notasi yang membantu memodelkan sistem dari berbagai perspektif. UML tidak hanya digunakan

dalam pemodelan perangkat lunak, namun hampir dalam semua bidang yang membutuhkan pemodelan..

- **Gambaran Umum UML**

Gambaran umum mengenai UML dapat dijelaskan berdasarkan kegunaan dari UML itu sendiri, yaitu:

1. **Modeling Language, UML sebagai bahasa untuk pemodelan sistem**

UML merupakan bahasa pemodelan yang memiliki pembendaharaan kata dan cara untuk mempresentasikan secara fokus pada konseptual dan fisik dari suatu sistem. Contoh untuk sistem *software* yang intensive membutuhkan bahasa yang menunjukkan pandangan yang berbeda dari arsitektur sistem, ini sama seperti menyusun/ mengembangkan software development life cycle. Dengan UML akan memberitahukan kita bagaimana untuk membuat dan membaca bentuk model yang baik, tetapi UML tidak dapat memberitahukan model apa yang akan dibangun dan kapan akan membangun model tersebut. Ini merupakan aturan dalam software development process.

2. **Visualizing, UML sebagai bahasa untuk menggambarkan sistem**

UML tidak hanya merupakan rangkaian simbol grafikal, cukup dengan tiap simbol pada notasi UML merupakan penetapan semantik yang baik. Dengan cara ini, satu pengembang dapat menulis model UML dan pengembang lain atau perangkat yang sama lainnya dapat mengartikan bahwa model tersebut tidak ambigu. Hal ini akan mengurangi error yang terjadi karena perbedaan bahasa dalam komunikasi model konseptual dengan

model lainnya.

UML menggambarkan model yang dapat dimengerti dan dipresentasikan ke dalam model tekstual bahasa pemograman. Contohnya kita dapat menduga suatu model dari sistem yang berbasis web tetapi tidak secara langsung dipegang dengan mempelajari code dari sistem. Dengan model UML maka kita dapat memodelkan suatu sistem web tersebut dan direpresentasikan ke bahasa pemrograman.

UML merupakan suatu model eksplisit yang menggambarkan komunikasi informasi pada sistem. Sehingga kita tidak kehilangan informasi code implementasi yang hilang dikarenakan developer memotong coding dari implementasi.

3. Specifying, UML sebagai bahasa untuk menspesifikasikan sistem

Maksudnya membangun model yang sesuai, tidak ambigu dan lengkap. Pada faktanya UML menunjukan semua spesifikasi keputusan analisis, desain dan implementasi yang penting yang harus dibuat pada saat pengembangan dan penyebaran dari sistem software intensif.

4. Constructing, UML sebagai bahasa untuk membangun sistem

UML bukan bahasa pemograman visual, tetapi model UML dapat dikoneksikan secara langsung pada bahasa pemograman visual. Maksudnya membangun model yang dapat dimapping ke bahasa pemograman seperti java, C++, VB atau tabel pada database relational atau penyimpanan tetap pada database berorientasi *object*.

5. Documenting, UML sebagai bahasa untuk pendokumentasian sistem

Maksudnya UML menunjukan dokumentasi dari arsitektur sistem dan detail dari semuanya.UML hanya memberikan bahasa untuk memperlihatkan permintaan dan untuk tes. UML menyediakan bahasa untuk memodelkan aktifitas dari perencanaan project dan manajemen pelepasan (release management).

- **Area dan Tujuan Penggunaan UML**

UML (Unified Modeling Language) digunakan paling efektif pada domain seperti:

a. Sistem Informasi Perusahaan
b. Sistem Perbankan dan Perekonomian
c. Bidang Telekomunikasi
d. Bidang Transportasi
e. Bidang Penerbangan
f. Bidang Perdagangan
g. Bidang Pelayanan Elekronik
h. Bidang Pengetahuan
i. Bidang Pelayanan Berbasis Web Terdistribusi

UML tidak terbatas untuk pemodelan software saja. Pada faktanya UML banyak digunakan untuk memodelkan sistem non-software seperti:

a. Aliran kerja pada sistem perundangan.
b. Struktur dan kelakuan dari Sistem Kepedulian Kesehatan Pasien
c. Desain hardware dll.

Tujuan penggunaan UML adalah, sebagai berikut:

1. Memodelkan suatu sistem (bukan hanya perangkat lunak) yang menggunakan konsep berorientasi *object*.
2. Menciptakan suatu bahasa pemodelan yang dapat digunakan baik oleh manusia maupun mesin.

Keunggulan menggunakan UML dibandingkan menggunakan metodologi terstruktur:

1. Uniformity
   Pengembang cukup menggunakan 1 metodologi dari tahap analsis hingga perancangan. Memungkinkan merancang komponen antarmuka secara terintegrasi bersama perancangan PL dan perancangan struktur data
2. Understandability
   Kode yang dihasilkan dapat diorganisasi kedalam kelas-kelas yangberhubungan dengan masalah sesungguhnya sehingga lebih mudah untuk dipahami.
3. Stability
   Kode program yang dihasilkan relatif stabil sepanjang waktu, karena mendekati permasalahan yang sesungguhnya.
4. Reusability
   Dengan metodologi berorientasi objek, dimungkinkan penggunaan ulang kode, sehingga pada akhirnya akan sangat mempercepat waktu pengembangan perangkat lunak (atau sistem informasi)

- **Bangunan Dasar UML**

Untuk memahami UML diperlukan pemahaman konseptual. Metodologi UML menggunakan 3 bangunan dasar untuk mendeskripsikan system perangkat lunak yang akan dikembangkan, yaitu:

1. *Things* (sesuatu)
2. *Relationship* (hubungan)
3. *Diagrams* (diagram)

Setiap bangunan dasar tersebut dapat diterapkan sepanjang tahap pengembangan sistem, dan masing-masingnya dapat digunakan saling melengkapi satu sama lainnya. Penjelasan dari ketiga bangunan dasar UML yang disebutkan diatas adalah sebagai berikut:

***1.* *Things* (sesuatu)**

Ada 4 macam *Things* dalam UML yaitu:

a. Structural Things

Merupakan bagian yang relatif statis dalam model UML. Bagian yang relatif statis dapat berupa elemen-elemen yang besifat fisik maupun konseptual. Secara keseluruhan ada 7 macam *Structural Things*:

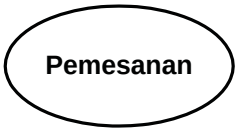
1) Classes / Kelas adalah himpunan dari objek-objek yang berbagi atribut serta operasi yang sama. Kelas mengimplementasikan satu atau lebih antarmuka. Secara grafis, kelas digambarkan

dengan empat persegi panjang yang memuat nama, atribut serta operasi yang dimilikinya.

2) *Interfaces* / Antarmuka adalah kumpulan dari operasi-operasi yang menspesifikasikan layanan/servis suatu kelas atau komponen/ objek. Antarmuka ini mendeskripsikan perilaku yang tampak dari luar suatu elemen. Secara grafis, antarmuka digambarkan dengan lingkaran kecil dan nama yang didahului dengan garis tegak.

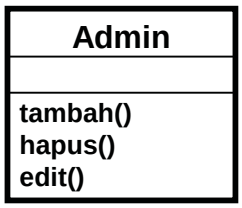
3) *Collaborations* / Kolaborasi adalah sesuatu yang mendefinisikan interaksi aturan-aturan dan elemen lain yang bekerja sama untuk menyediakan perilaku yang lebih besar dan sinergis. Secara grafis, kolaborasi digambarkan elips bergaris putus-putus yang memuat nama kolaborasi itu.

4) *Use-Cases* adalah deskripsi dari urutan aksi-aksi yang ditampilkan sistem dengan hasil yang terukur bagi suatu Aktor. *Use-Case* digunakan untuk menstrukturkan perilaku pada suatu model. Secara grafis use case digambarkan dengan elips tegas yang berisi namanya.



5) *Active Class* / Kelas Aktif adalah kelas (biasa) dimana objek-objek yang dimilikinya memiliki 1 atau lebih proses dan lebih jauh menginisialisasi

suatuu aktifitas kendali. Secara grafis, kelas aktif digambarkan sebagai kelas biasa tapi memiliki batas yang lebih tebal, yang memuat nama, atribut serta operasi yang dimilikinya.



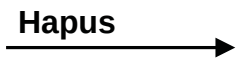
6) *Components* / Komponen adalah bagian fisik dan bagian yang dapat digantikan dalam suatu sistem, misalnya berkas ActiveX, COM, DLL yang keberadaanya dibutuhkan oleh sistem yangakandikembangkan. Komponen ini merepresentasikan konsep-konsep reusable component. Secara grafis, komponen digambarkan dengan empat persegi panjang seperti kelas tetapi ditambahkan *Tab*.

7) *Nodes* / Simpul adalah elemen fisik yang eksis saat aplikasi dijalankan dan mencerminkan suatu sumber daya komputasi; secara umum menggunakan kapasitas memori dan kemampuan pemrosesan. Secara grafis, simpul digambarkan sebagai kubus yang berisi namanya.

b. Behavioral Things
Merupakan bagian yang dinamis pada model UML,

biasanya merupakan kata kerja dari model UML yang mencerminkan perilaku sepanjang ruang dan waktu. Ada 2 macam Behavior Things, yaitu:

1) *Interactions* / interaksi adalah suatu perilaku yang mencakup himpunan pesan-pesan (*message*) atau kumpulan objek & operasinya yang diperlukan untuk menyelesaikan suatu fungsi tertentu. Sebuah interaksi terdiri dari beberapa unsur, yaitu: Pesan, Perilaku dan Link, Secara grafis, pesan digambarkan dengan tanda panah tegas dan memuat nama operasinya.



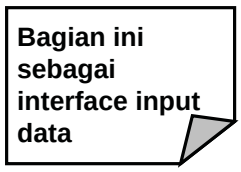
2) State Machine, adalah perilaku yang menspesifikasikan urutan kedudukan suatu objek atau interaksi-interaksi sepanjang waktu untuk menanggapi event-event yang terjadi. Penggambaran state memiliki beberapa unsur, yaitu: State itu sendiri, Transisi (perubahan dari suatu state ke state lain), Event (suatu keadaan yang memicu sebuah transisi), Aktifitas (tanggapan terhadap transisi). Secara grafis State digambarkan sebagai empat persegi panjang dengan sudut tumpul, yang memuat namanya serta subsistem didalamnya jika ada.

c. Grouping Things
Merupakan bagian pengorganisasian (*Packages*) dalam UML. Digunakan untuk menggambarkan paket-

paket untuk menyederhanakan model-model UML yang rumit. Paket-paket ini kemudian dapat di dekomposisi. Paket berguna untuk pengelompokkan sesuatu misalnya model-model serta subsistem-subsistem.

d. Annotational Things

Merupakan bagian yang memperjelas model UML (*Notes*), seperti komentar-komentar yang menjelaskan fungsi secara rinci serta ciri-ciri tiap elemen dalam model UML.



2. *Relationship* (hubungan)

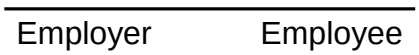
Relationship adalah hubungan-hubungan yang terjadi antara elemen dalam UML. Hubungan-hubungan ini Penting sekali dalam UML. Model-model UML harus dibuat menggunakan relationship. Ada 4 macam relationship dalam UML yang dapat digunakan untuk menggambarkan model-model UML yang representatif:

a. *Dependency* (kebergantungan)

Dependency adalah hubungan dimana perubahan yang terjadi dalam suatu eleman independent (mandiri) akan mempengaruhi elemen yang bergantung padanya. Secara grafis, dependency digambarkan dengan tanda panah terputus-putus.

b. *Association* (asosiasi)

Asosiasi adalah sesuatu yang menghubungkan antara suatu objek dengan objek lainnya yang menggambarkan bagaimana hubungan antarobjek tersebut dalam bentuk agregasi. Secara grafis, asosiasi digambarkan dengan garis tegas tanpa tanda panah.



c. *Generalizations* (generalisasi)

Generalisasi adalah hubungan di mana objek anak (*descendent*) berbagi perilaku dan struktur data dari objek yang ada di atasnya, yaitu objek induk (*ancestor*). Arah dari atas ke bawah atau dari ancestor ke descendent dinamakan *Spesialisasi*. Arah sebaliknya yaitu bawah ke atas dinamakan *Generalisasi*. Secara grafis Generalisasi digambarkan dengan garis dan panah yang berkepala sigitiga kosong dan mengarah ke objek induk.

d. Realizations (realisasi)

Realisasi adalah operasi yang benar-benar dilakukan oleh suatu objek. Secara grafis, realisasi digambarkan dengan tanda panah bergaris putus-putus dengan kepala panah kosong.

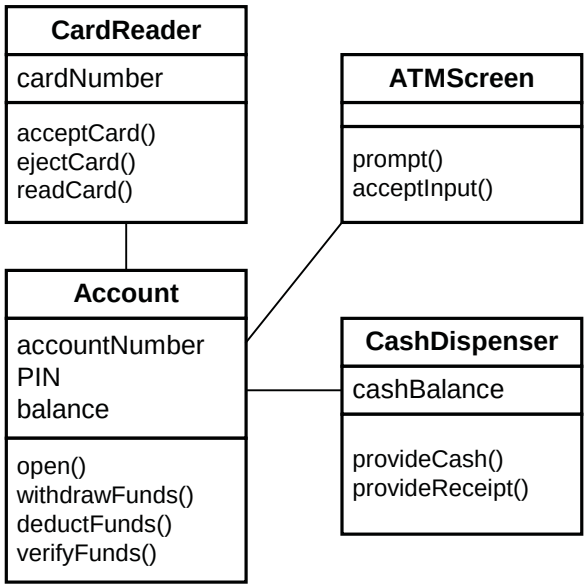
***3. Diagrams* (diagram)**

Suatu sistem yang kompleks harus dapat dipandang dari sudut yang berbeda-beda sehingga didapat pemahaman secara menyeluruh. Untuk keperluan tersebut UML menyediakan 9 jenis diagrams yang dapat dikelompokkan berdasarkan sifatnya; Statis atau

dinamis. Kesembilan diagram ini tidak mutlak digunakan atau dibuat sesuai kebutuhan. Pada pemodelan UML dimungkinkan menggunakan diagram lain sejauh diagram tersebut dapat membantu pemahaman mendalam tentang suatu sistem atau perangkat lunak. Ke-9 diagram tersebut adalah sebagai berikut:

a. Class Diagrams (Statis)

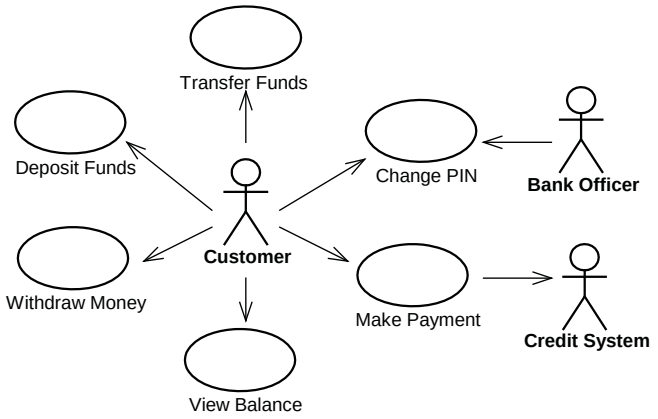
Diagram ini memperlihatkan himpunan kelas-kelas, antarmuka-antarmuka, kolaborasi-kolaborasi, dan relasi-relasi. Diagram ini umum ditemui pada pemodelan sistem berorientasi objek. Meski sifatnya statis, sering pula memuat kelas-kelas aktif.



- *Class Diagram* menggambarkan interaksi antar kelas dalam sistem tersebut.
- Pembuatan *Class* sama dengan pembuatan Objek-objek

b. *Object Diagrams* (Statis)
Diagram ini memperlihatkan objek-objek serta relasi-relasi antar objek. Diagram objek memperlihatkan instantiasi statis dari segala sesuatu yang dijumpai dari pada diagram kelas.

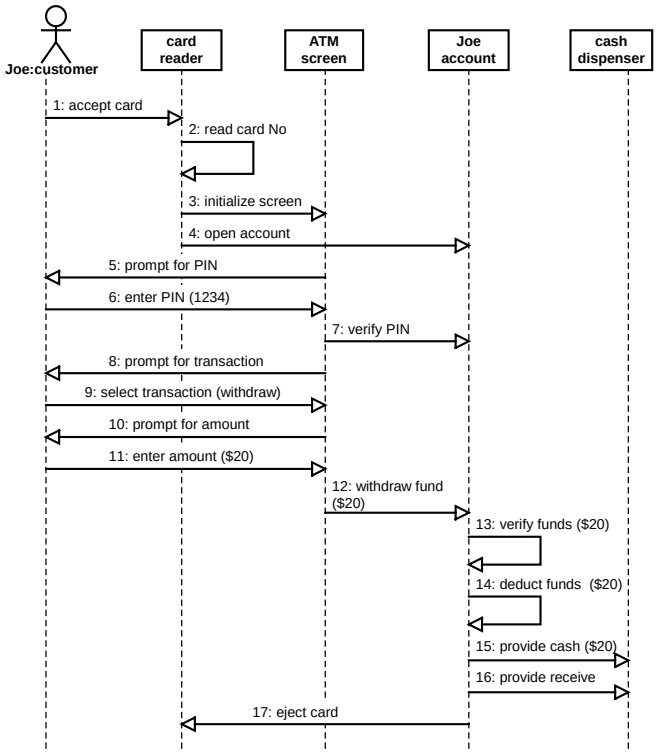
c. *Use-Case* Diagrams (Statis)
Dagram ini memperlihatkan himpunan *Use-Case* dan *Actor-Actor* (jenis khusus dari kelas). Diagram ini penting untuk mengorganisasi dan memodelkan perilaku dari suatu sistem yang dibutuhkan serta diharapkan pengguna.



- *Use-Case* diagram menggambarkan hubungan use-case dengan actor.
- Use-case merepresentasikan fungsi, kebutuhan dari perpektif *user.*
- *Actor* adalah orang atau sistem yang menerima atau memberikan informasi dari sistem ini.

d. *Sequence Diagrams* (Dinamis)
Diagram *sequence* (urutan) adalah diagram interaksi yang menekankan pada pengiriman pesan (*message*)

dalam suatu waktu tertentu.

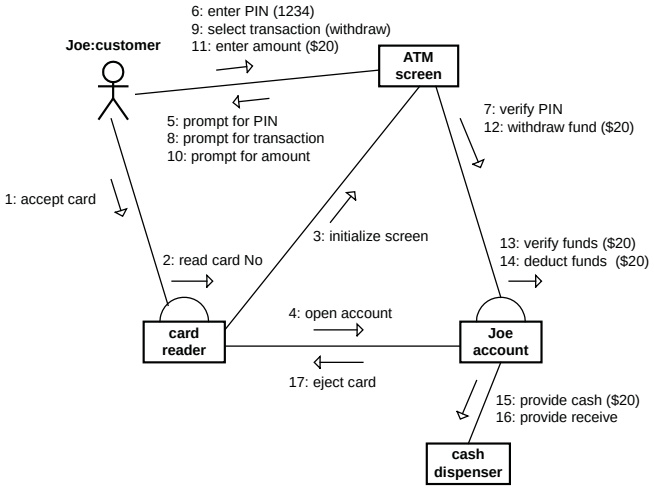


- *Sequence* Diagram mengambarkan alur kerja dari fungsi-fungsi dalam sistem dengan use-case dimana didalamnya terdapat *actor.*
- Actor adalah orang atau sistem yang menerima atau memberikan informasi dari sistem ini. Dalam gambar diatas Actor-nya adalah Joe.
- Diagram ini sangat memperhatikan waktu/ terurut berdasarkan kejadian (*sequence*).

e. *Collaboration* Diagrams (Dinamis)

Diagram kolaborasi adalah diagram interaksi yang

menekankan organisasi structural dari objek-objek yang menerima serta mengirim pesan (*message*).



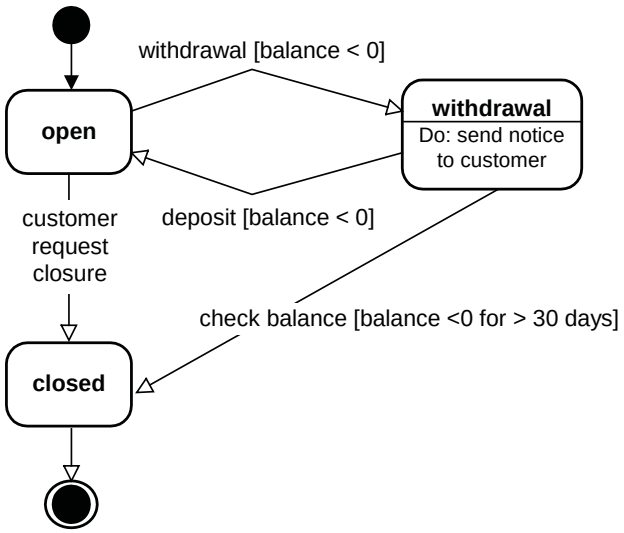
- Informasi yang disampaikan sama dengan *sequencial* diagram namun beda dalam pengambaran dan kegunaan saja.
- Dalam diagram ini digambarkan hubungan antar objek dan *actor* dengan tidak memperhatikan waktu/urutan

f. State Chart Diagram (Dinamis)

Diagram ini memperlihatkan state-state pada sistem; memuat state, transisi, event, serta aktifitas. Diagram ini terutama penting untuk memperlihatkan sifat dinamis dari antarmuka, kelas, kolaborasi, dan terutama penting pada pemodelan sistem-sistem reaktif.

- State Chart Diagram memberikan berbagai cara/ jalan kepada model untuk berbagai kejadian yang mungkin terjadi dalam sebuah objek.

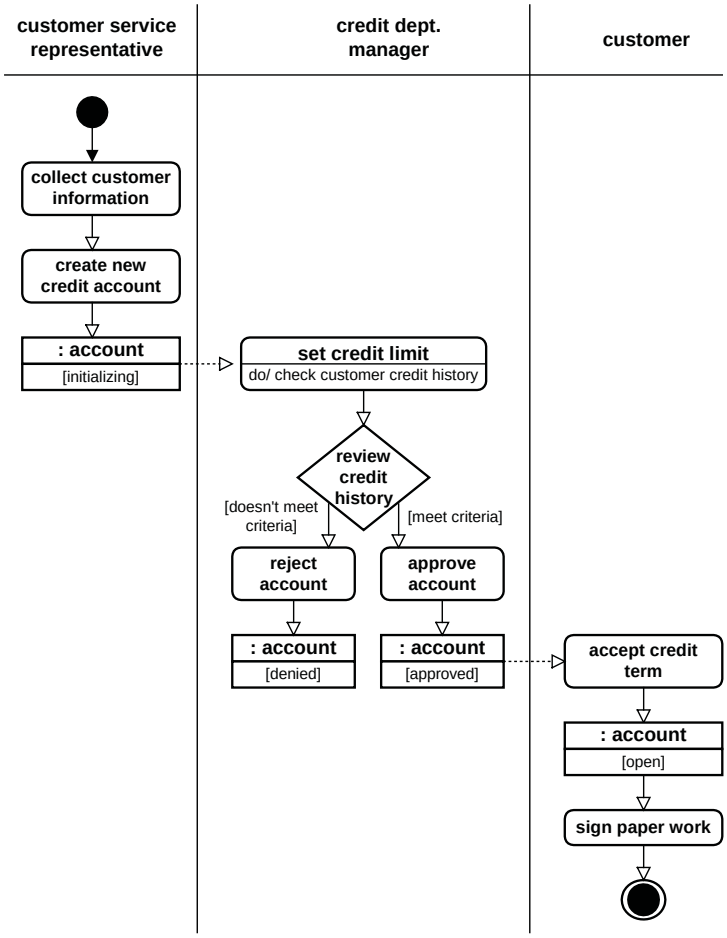
- Diagram ini digunakan untuk menggambarkan berbagai prilaku objek yang sifatnya dinamis dalam sebuah system.



g. *Activity Diagrams* (Dinamis)

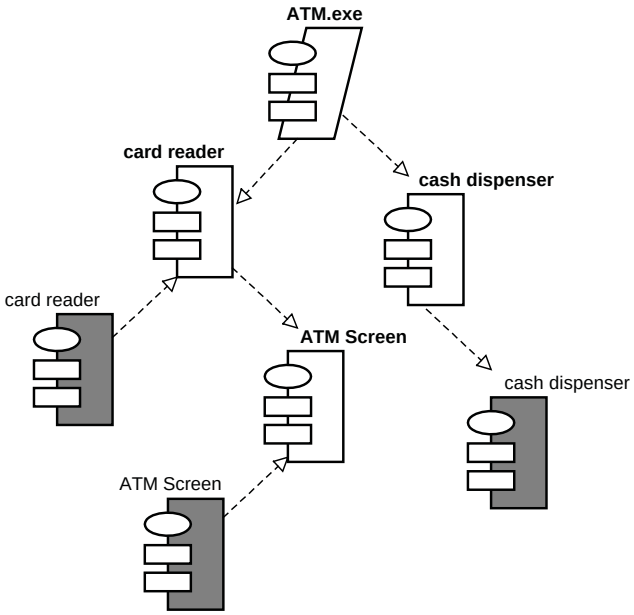
Diagram ini adalah tipe khusus dari diagram state yang memperlihatkan aliran dari suatu aktifitas ke aktifitas lainnya dalam suatu sistem. Diagram ini terutama penting dalam pemodelan fungsi-fungsi dalam suatu sistem dan memberi tekanan pada aliran kendali antar objek.

- Memberikan gambaran ilustrasi alur dari setiap fungsi yang ada dalam sistem.
- Aliran dimulai dari suatu titik hingga ke titik akhir yang disepakati.



h. *Component Diagr*ams (Statis)

Diagram ini memperlihatkan organisasi serta kebergantunngan pada komponen-komponen yang telah ada sebelumnya. Diagram ini berhubungan dengan diagram kelas dimana komponen secara tipikal dipetakan kedalam satu atau lebih kelas-kelas, antarmuka-antarmuka, serta kolaborasi-kolaborasi.

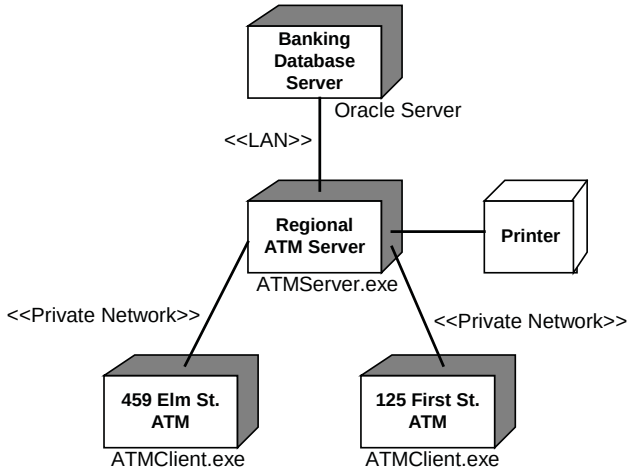


- Menggambarkan model secara fisik sebagai sebuah software komponen yang ada dalam sebuah sistem.
- Komponen-komponen tersebut nantinya diarahkan pada suatu bahasa pemrograman tertentu,

i. *Deployment Diagrams* (Statis)

Diagram ini memperlihatkan konfigurasi saat aplikasi dijalankan (*run-time*), memuat simpul-simpul atau node beserta komponen-komponen yang ada didalamnya. Deployment Diagrams berhubungan erat dengan diagram komponen dimana Deployment Diagrams memuat satu atau lebih komponen-komponen. Diagram ini sangat berguna saat aplikasi

kita berlaku sebagai aplikasi yang dijalankan pada banyak mesin (*distributed computing*).



- Diagram *Deployment* menggambarkan bentuk layout secara fisik bentuk jaringan dan posisi komponen-komponen dari sistem.
- Pendekatan yang digunakan dalah pendekatan terhadap hasil implementasi/ program,

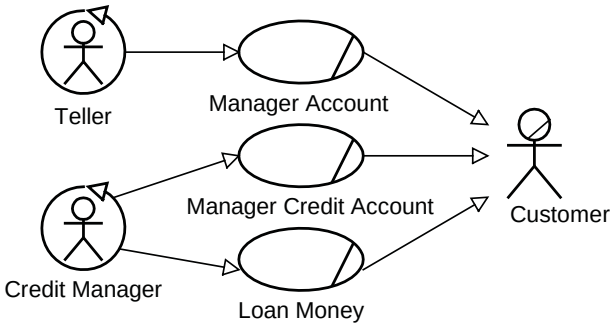
• **Langkah-langkah Penggunaan UML**

Menurut Sri Dharwiyanti (ilmukomputer.com), langkah-langkah pengembangan perangkat lunak menggunakan pendekatan berorientasi Objek dengan bantuan pemodelan visual UML melalui tahapan-tahapan berikut:

1. Buatlah daftar *business process* dari level tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul.

   Dari permasalahan diatas dapat digambarkan

bussines process sebagai berikut:



2. Petakan *use case* untuk tiap *business process* untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh sistem. Kemudian perhalus *use-case* diagram dan lengkapi dengan *requirement*, *constraints* dan catatan-catatan lain.
3. Buatlah *deployment diagram* secara kasar untuk mendefinisikan arsitektur fisik sistem.
4. Definisikan *requirement* lain (non-fungsional, security dan sebagainya) yang juga harus disediakan oleh sistem.
5. Berdasarkan *use-case diagram*, mulailah membuat *activity diagram*.
6. Definisikan objek-objek level atas (package atau domain) dan buatlah *sequence* dan/atau *collaboration diagram* untuk tiap alir pekerjaan. Jika sebuah *use-case* memiliki kemungkinan alir normal dan error, buatlah satu diagram untuk masing-masing alir.
7. Buarlah rancangan user interface model yang menyediakan antarmuka bagi pengguna untuk menjalankan skenario *use-case*.
8. Berdasarkan model-model yang sudah ada, buatlah *class diagram*. Setiap package atau domain dipecah

menjadi hirarki class lengkap dengan atribut dan metodanya. Akan lebih baik jika untuk setiap *class* dibuat unit test untuk menguji fungsionalitas class dan interaksi dengan class lain.

9. Setelah *class diagram* dibuat, kita dapat melihat kemungkinan pengelompokan *class* menjadi komponen-komponen. Karena itu buatlah component diagram pada tahap ini. Juga, definisikan tes integrasi untuk setiap komponen meyakinkan ia berinteraksi dengan baik.
10. Perhalus *deployment diagram* yang sudah dibuat. Detilkan kemampuan dan *requirement* piranti lunak, sistem operasi, jaringan, dan sebagainya. Petakan komponen ke dalam node.
11. Mulailah membangun sistem. Ada dua pendekatan yang dapat digunakan:
    a. Pendekatan use case, dengan meng-assign setiap use case kepada tim pengembang tertentu untuk mengembangkan unit code yang lengkap dengan tes.
    b. Pendekatan komponen, yaitu meng-*assign* setiap komponen kepada tim pengembang tertentu.
12. Lakukan uji modul dan uji integrasi serta perbaiki model berserta code-nya. Model harus selalu sesuai dengan code yang aktual.
13. Piranti lunak siap dirilis.

- **Tools Pendukung UML**

Saat ini banyak sekali tool pendesainan yang mendukung UML, baik itu tool komersial maupun opensource. Beberapa di antaranya adalah:

1. Rational Rose (www.rational.com)

2. Together (www.togethersoft.com)
3. Object Domain (www.objectdomain.com)
4. Jvision (www.object-insight.com)
5. Objecteering (www.objecteering.com)
6. MagicDraw (www.nomagic.com/magicdrawuml)
7. Visual Object Modeller (www.visualobject.com)

Data seluruh tool yang mendukung UML, lengkap beserta harganya (dalam US dolar) bisa anda pelajari di situs
http://www.objectsbydesign.com/tools/umltools_byCompany.html

Disamping itu, daftar tool UML berikut fungsi dan perbandingan kemampuannya juga dapat dilihat di http://www.jeckle.de/umltools.htm. Pointer Penting UML sebagai referensi dalam mempelajari dan menggunakan UML, situs-situs yang merupakan pointer penting adalah:

- http://www.cetus-links.org/oo_uml.html
- http://www.omg.org
- http://www.omg.org/technology/uml/
- http://www.rational.com/uml
- http://www.uml.org/

BAB VII
PENGENALAN PROYEK DAN MANAJEMEN TI

Pengelolaan sebuah proyek IT sangat berbeda dengan pengelolaan proyek-proyek pada umumnya. Hal ini dikarenakan IT memiliki hubungan yang sangat luas dan terikat dengan kemajuan teknologi yang sangat cepat dewasa ini. Dari satu sisi masih banyak ditemukan pengguna akhir (*end user*) yang masih sulit menggunakan alat bantu seperti komputer dalam menjalankan tugasnya. Dari sisi lain peran dari pihak eksekutif (sponsor) yang terlalu berharap bahwa komputer akan memberikan banyak nilai lebih bagi perusahaan, meskipun pada kenyataannya sangat sulit memperoleh keuntungan pada awal-awal penggunaan alat-alat bantu IT − dikarenakan banyak dibutuhkannya training bagi pengguna akhir dengan investasi besar.

Di dalam dunia IT para pekerja, seperti: *implementator / programmer / system analyst / designer dan administrator*, menghadapi berbagai kendala dalam menjalankan fungsinya. Hal ini dikarenakan antara lain: perubahan strategi bisnis perusahaan, kompatibilitas perangkat keras, pilihan perangkat lunak yang beraneka ragam, masalah pengamanan data, *bandwith* jaringan komputer, tingkah laku para pengguna akhir dan pekerja lainnya serta kebijakan-kebijakan dari eksekutif perusahaan.

Semua permasalahan di atas membuat pengelolaan

proyek IT menjadi sangat kompleks dan rentan terhadap kegagalan. Di dalam baguan ini akan dibahas bagaimana mengelola suatu proyek IT mulai dari awal hingga akhir, serta dapat menggunakan konsep-konsep manajemen proyek yang umum dalam proyek IT. Tidak ketinggalan akan diberikan pula suatu panduan untuk menggunakan MS Project 2002, sebagai salah satu tools yang sering digunakan dalam pengelolaan proyek di perusahaan-perusahaan secara luas. Diktat ini tidak akan membuat Anda menjadi manajer proyek IT dalam waktu singkat, tetapi akan membuka wawasan untuk memahami apa dan bagaimana proses pengelolaan proyek itu, dan apa peran yang dapat anda lakukan bila terlibat dalam suatu proyek IT.

Secara umum, manajemen proyek dibutuhkan karena:

- Adanya kompresi dari siklus hidup produk (adalah pendorong utama perlunya manajemen proyek).
- Adanya kompetisi global (persaingan ketat).
- Ledakan pengetahuan (menjadi kompleks);
- Downsizing (desentralisasi) manajemen perusahaan.
- Peningkatan fokus pada konsumen.
- Perkembangan yang sangat cepat di dunia ketiga (sebutan US untuk Asia) dan ekonomi tertutup (seperti negara China yang komunis).
- Proyek kecil yang banyak biasanya menimbulkan banyak masalah.

7.1 Mengapa manajemen proyek IT ?

Di atas telah dibahas bahwa setiap proyek IT sangat rentan terhadap perubahan dan kegagalan. Hal ini sebenarnya sangat bertolak belakang pada kenyataan

bahwa kecepatan prosesor komputer (sebagai media utama pengimplementasian proyek-proyek IT) meningkat dua kali lipat setiap 18 bulan – dikenal sebagai *Moore's law*, bahkan sekarang peningkatan tersebut lebih cepat lagi. Namun pada kenyataannya:

- 31% proyek IT ditangguhkan sebelum penyelesaian;
- 88% melampaui waktu atau pendanaan (atau keduanya) yang ditentukan;
- Dari 100 proyek IT yang dimulai 94 harus diulang;
- Rata-rata ekstra pengeluaran dalam pendanaan adalah 189%
- Rata-rata ekstra waktu penyelesaian yang dibutuhkan adalah 222%

Kenyataan yang tidak dapat disangkal ini, sebagian besar terjadi karena visi dan misi suatu proyek yang melenceng pada saat pelaksanaan. Dalam pelaksanaan inilah peran manajemen proyek sangat diperlukan sebagai suatu alat untuk mengontrol pencapaian visi dan misi, serta pengelolaan sumberdaya pendukung proyek tersebut. Orang yang memikul tanggung jawab besar dalam suatu pelaksanaan proyek adalah manajer proyek, namun tentu saja peran para pekerja dalam tim tidaklah kecil untuk membuat proyek menjadi berhasil.

7.1.1 Pengertian IT, kerja operasional dan proyek, serta proyek IT

Teknologi Informasi (*Information Techology*) adalah:

Suatu **teknologi** (cara, metode, konsep, teknik) yang berhubungan erat dengan **pengolahan data** menjadi **informasi** dan proses **penyaluran** data/informasi tersebut dalam batas-batas **ruang** dan **waktu**. Contoh **produk-**

produk IT: modem, router, oracle, SAP, printer, multimedia, situs internet, e-business, cabling system, satelit, dsb.

Kerja secara *operasional* adalah:

- Pekerjaan rutin secara terus-menerus di suatu lingkungan kerja.

Contoh kerja operasional:

- surat-menyurat di kantor, kegiatan administrasi T.U. , kegiatan belajar-mengajar di kampus.

Proyek adalah:

- Usaha pada satu waktu tertentu yang kompleks, non-rutin, melibatkan SDM yang memiliki dedikasi untuk terlibat dan **dibatasi oleh waktu**, anggaran, sumber daya dan spesifikasi-spesifikasi yang didesain sesuai kebutuhan konsumen (**unik**).

Contoh proyek:

- Pembangunan gedung, pengembangan produk, pendirian usaha, setup network computer, renovasi rumah, pelaksanaan pesta pernikahan, dsb.
- Dalam dunia IT: Pengembangan perangkat lunak administrasi nilai mahasiswa, upgrade network perusahaan, dsb.

Persamaan antara dua kegiatan ini adalah:

- Dilakukan oleh SDM;
- dibatasi oleh (pra)sarana/sumberdaya;
- direncanakan;
- dilaksanakan; dan
- terkontrol.

Proyek IT:

Suatu proyek yang membutuhkan **sumberdaya** dan/ataupun **produk teknologi informasi** dalam penyelesaiannya.

Contoh **proyek IT**:

- Pembangunan jaringan komputer di fakultas teknik UKM.
- Pembuatan software untuk administrasi T.U.
- Pembuatan perangkat *e-commerce* di suatu perusahaan eksport-import.
- Pembuatan infrastruktur kartu ATM (*online banking*).

7.1.2 Mencari visi dan misi proyek IT

Visi adalah suatu tujuan (goal) yang dituju di masa depan. Visi memberi arahan kemana perjalanan suatu proyek harus dituju. Apa yang diharapkan dari suatu proyek, sumberdaya apa saja yang dibutuhkan, apa saja yang harus dihasilkan, kapan proyek harus selesai, siapa saja yang berkepentingan dalam proyek ini, siapa pengguna akhir proyek. Ini adalah sebagian kecil dari berbagai macam pertanyaan yang harus terjawab dalam pelaksaan suatu proyek.

Sebuah proyek tanpa tujuan yang jelas akan menghabiskan banyak waktu, biaya, dan talenta tanpa menghasilkan sesuatu yang bermutu. Sebelum suatu proyek dimulai harus dinyatakan dengan jelas apa hasil yang harus dicapai sehingga suatu proyek dapat dianggap selesai. Sebuah proyek dapat dinyatakan dimulai (dalam konsep) pada saat telah diketahui secara pasti dan jelas apa yang akan dihasilkan dari sebuah pelaksaan proyek.

Setelah proyek terdefinisi perlu dinyatakan kapan

proyek tersebut harus dimulai dan kapan selesaiya. Peran seorang manajer proyek adalah temporer, mengingat rentang waktu proyek yang terbatas. Seorang manajer proyek bertanggung jawab atas tercapainya visi dan misi proyek, pengembangan aktivitas proyek dan kepemimpinan serta pengelolaan sumberdaya selama berlangsungnya proyek.

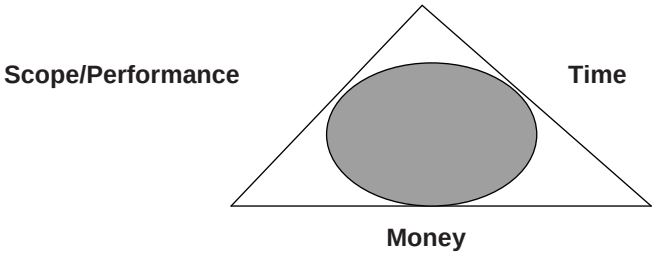
Bagaimana caranya mencari visi yang jelas pada proyek IT? Bertanyalah dan adakan wawancara pada pihak pemberi order (seperti eksekutif pada perusahaan, dan sponsor proyek) dapat memberi jawaban akan hal ini. Selain itu diperlukan pula timbal balik dan masukan dari pihak penggunan akhir, distributor dan vendor perlengkapan IT, kebijaksanaan perusahaan serta data-data dari proyek sejenis sebelumnya.

Setelah visi dapat didefinisikan, maka *misi* – yaitu aktivitas-aktivitas yang harus dijalankan untuk mencapai visi – dalam proyek dapat pula didefinisikan. Apabila misi telah terdefinisi maka seorang manajer proyek harus mencari keseimbangan antara sumberdaya dan teknologi yang dipakai dalam pelaksaan proyek tersebut.

Dengan kata lain *manajemen proyek IT* dapat didefinisikan sebagai berikut:

- Suatu penerapan **pengetahuan**, **keahlian**, **perangkat** dan **teknik** dalam bidang **Teknologi Informasi** di dalam aktivitas-aktivitas proyek dengan tujuan mencapai target (prestasi) tertentu, spt: keinginan dari client, *stakeholders* ataupun dari tim Management kantor (atasan);
- Pencarian **titik temu** dari: ruang lingkup, jadwal, biaya, kualitas, *requirements* dari pihak yang memberi proyek.

- Titik temu ini dikenal pula dengan sebutan *project triangle* (segitiga proyek), yaitu faktor-faktor yang sangat berperan dalam pelaksaan proyek. Dapat digambarkan sebagai berikut:



  - o Waktu (**time**) adalah: jadual dan pembagian tugas.
  - o Pendanaan (**money**) adalah: sumberdaya utk menjalankan tugas.
  - o Ruang lingkup (**scope**) adalah: tujuan (goal/ visi) dan produk (deliverables) dari proyek dan menuju pada penilaian **performance** dari sebuah proyek secara keseluruhan.

Ketiganya saling berkaitan; tergantung pada permasalahan yang ada, namun selalu ada satu yang paling berpengaruh dalam setiap proyek. Dengan demikian, jelaslah bahwa di dalam manajemen proyek termuat titik acuan sebagai berikut:

- o Proyek memiliki tujuan yang jelas;
- o Ada suatu tenggang waktu yang terdefinisi dengan waktu mulai dan akhir proyek;
- o Biasanya melibatkan beberapa department dan profesional;
- o Biasanya, melakukan sesuatu yang belum pernah atau jarang dikerjakan sebelumnya;

- Harus memenuhi suatu syarat waktu, biaya dan kinerja (unik).

7.1.3 Refleksi manajemen

Seorang manajer proyek harus mampu menciptakan keseimbangan antara ketiga faktor pembentuk proyek. Lebih lanjut seorang manajer proyek merupakan refleksi dari aktivitas manajemen secara luas, yaitu:

- Manajemen adalah proses **menentukan** dan **mengimplementasikan** cara-cara untuk memanfaatkan sumberdaya manusia dan non manusia secara efektif dan efisien untuk mencapai **tujuan** yang telah ditentukan.
- Seorang manajer proyek sepertinya tidak berbeda dengan manajer lainnya, yaitu **merencanakan**, **menjadwalkan**, **memotivasi** dan **mengendalikan**.
- Namun, seorang manajer proyek adalah **unik**, karena manajer ini hanya mengatur **aktivitas-aktivitas yang temporer**, tidak repetitif, dan seringkali bertindak secara **independen** di dalam suatu organisasi formal.

7.1.4 Karakteristik Proyek IT

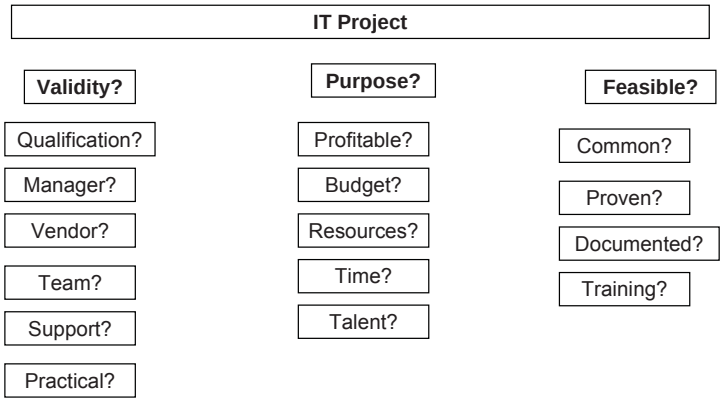
Secara khusus dalam proyek-proyek IT, seorang manajer proyek IT harus mampu melihat tingkat kesulitan dan kompleksitas proyek IT yang memerlukan perlakuan khusus, yaitu:

- **Invisibility (kekasatan)**
  - Bentuk fisik dari proyek-proyek IT terkadang tidak terlihat, sehingga sulit untuk dilihat kemajuannya.
  - Contoh: pengembangan program, upgrade PC, dsb.

- **Complexity (kompleksitas)**
  - Setiap sumberdaya dan pendanaan yang dikonsumsi dalam sebuah produk IT melibatkan kompleksitas yang lebih tinggi dibanding proyek-proyek rekayasa lainnya.
  - Contoh: apabila dalam pengembangan software pada stadium yang hampir selesai, mengalami perubahan yang signifikan pada spesifikasinya, maka perubahan tsb akan sangat sulit untuk diimplementasi (bahkan perubahan terkadang menyebabkan proyek dimulai dari awal lagi).

- **Flexibility (fleksibilitas)**
  - Meskipun perubahan pada spesifikasi pada saat pengimplementasian sangat sulit untuk dilakukan, pada dasarnya proyek-proyek IT sangat fleksibel.
  - Hal ini mengacu pada kenyataan bahwa proyek IT dan keberadaan proyek IT adalah sebagai sarana pendukung bagi komponen lain dalam suatu lingkungan kerja.
  - Dengan demikian proyek IT dapat dikatakan memiliki derajat perubahan yang tinggi (high degree of change).
  - Contoh: pembangunan jaringan komputer di suatu kantor tidak menyebabkan aktivitas di kantor tersebut menjadi mati.

Dengan hadirnya karakteristik khusus proyek-proyek IT ini, maka dapat disimpulkan seorang manajer proyek IT dalam melaksanakan tugasnya wajib menjawab pertanyaan-pertanyaan di seputar hal-hal berikut ini:



7.2 Kegiatan utama sebuah proyek

Di dalam bagian ini akan dibahas kegiatan utama sebuah proyek yang dapat diterapkan dalam segala jenis proyek, namun secara khusus akan dibahas aktivitas-aktivitas dalam proyek IT, sebagai berikut:

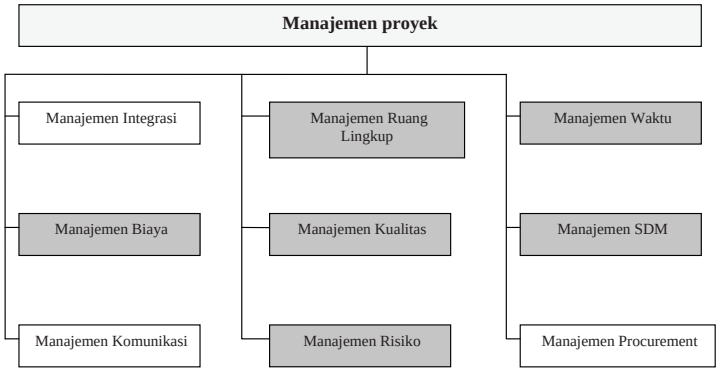
- Pembuatan rencana proyek
  - Feasibility plan dan spesifikasi proyek
  - Riset
  - Rencana (perkiraan) jadwal dan biaya
- Pengamatan dan pengaturan proyek
  - Penjadwalan: Gantt chart, jaringan kerja (AON)
  - Kegiatan proyek: WBS
  - Organisasi proyek: OBS
  - *Project life cycles*
  - Eksekusi dan implementasi
  - Manajemen risiko
- Penutupan proyek
  - Evaluasi,verifikasi, optimasi dan penilaian
  - *Client acceptance.*

BAB VIII
GENESIS PROYEK

8.1 Konsep kerja manajemen proyek

Di dalam manajemen proyek terdapat bagian-bagian penyusun konsep kerja (*frameworks*) yang digunakan untuk memahami konsep manajemen proyek secara keseluruhan sebagai berikut:

- Manajemen integrasi (bagian-bagian) dalam proyek (integration management);
- Manajemen ruang lingkup proyek (scope management);
- Manajemen waktu proyek (time management);
- Manajemen biaya proyek (financial management);
- Manajemen kualitas proyek (quality management;
- Manajemen SDM proyek (human resource management;
- Manajemen komunikasi dalam proyek (communication management);
- Manajemen risiko dalam proyek (risk management);
- Manajemen sumberdaya dari luar organisasi yang menunjang pelaksanaan proyek (procurement management), dikenal juga dengan sebutan *outsourcing* atau detasering.



Di dalam diktat ini tidak akan dibahas keseluruhan, hanya bagian yang digambarkan dengan kotak berarsir saja yang akan dijabarkan. Konsep kerja ini dapat diterjemahkan ke dalam aktivitas-aktivitas sebagaimana telah dituliskan dalam bab 1 terdahulu (lihat sub-bab 1.7).

Dengan dibaginya konsep kerja ke dalam sub-sub bagian diharapkan akan tercipta suatu interaksi antara bagian yang satu dengan bagian lainnya. Namun sebelum kita beranjak lebih jauh akan dibahas terlebih dahulu faktor-faktor penentu keberhasilan suatu proyek IT serta hambatan-hambatan yang sering ditemui dalam suatu proyek IT.

Menurut penelitian dari *Standish Group* (suatu badan independen yang melakukan penelitian terhadap perkembangan industri IT); dalam artikel *"Collaborating on project success"*; Johnson, J. et al., 2001; dipublikasikan dalam Software Magazine & Wiesner Publishing Feb/March 2001. Dituliskan bahwa keberhasilan suatu proyek IT ditentukan oleh berbagai faktor yang dapat dilihat pada tabel berikut ini:

Faktor keberhasilan proyek	Tingkat keyakinan
Dukungan eksekutif	18%
Keterlibatan pengguna (*end-user*)	16%
Pengalaman manager proyek	14%
Sasaran usaha yang jelas	12%
Lingkup yang diminimalkan	10%
Infrastruktur SW standard	8%
Requirement dasar yang kuat	6%
Metodologi formal	6%
Kehandalan estimasi (waktu, biaya)	5%
Lain-lainnya	5%

Terlihat dengan jelas bahwa peran dari seorang proyek manajer sangat menentukan (di peringkat ketiga dengan nilai 14%). Peringkat pertama dan kedua adalah: dukungan dari eksekutif (upper management dan sponsor), dan peran serta pengguna (end user). Meskipun tidak mutlak, pengalaman dan keterlibatan aktif seorang manajer proyek di dalam sebuah proyek IT sangat menentukan keberhasilan proyek.

1. **Dukungan eksekutif**: jelas bahwa kurangnya dukungan dari eksekutif atau manajemen atas dapat menggagalkan proyek.
2. **Keterlibatan pengguna**: bahkan jika diselesaikan tepat waktu dan tepat biaya, suatu proyek masih dapat gagal memenuhi harapan penggunanya.
3. **Pengalaman manajer proyek**: sembilan puluh tujuh

persen proyek yang berhasil dipimpin oleh manajer proyek yang berpengalaman.

4. **Sasaran usaha yang jelas**: selain pengalaman manajer proyek, penentuan *scope* (lingkup) dari proyek sangat menentukan keberhasilan proyek.
5. **Lingkup yang diminimalkan**: lingkup berkaitan dengan waktu penyelesaian, dan karena waktu terbatas oleh jadwal, maka membatasi lingkup akan sangat membantu.
6. **Infrastruktur software standar**: dengan menggunakan infrastruktur yang standar, tim pengembang dapat lebih berfokus pada aspek usaha daripada teknologi, demikian pula integrasi antar aplikasi akan dipermudah (khususnya untuk proyek pengembangan *software*).
7. **Requirement dasar yang kuat**: dengan mendefinisikan keperluan minimum dari hasil pelaksanaan proyek, upaya dapat difokuskan untuk mencapainya.
8. **Metodologi formal**: survei menunjukkan bahwa 46% proyek yang berhasil menggunakan metode manajemen proyek formal, antara lain karena tim proyek dapat segera saling sepakat mengenai prosedur,cara menghadapi suatu masalah, dsb.
9. **Kehandalan estimasi**: estimasi yang baik akan membantu memberi gambaran keseluruhan terhadap proyek, seperti banyaknya aktivitas proyek, deadline dan jumlah uang serta tenaga yang dibutuhkan. Hal ini banyak juga ditentukan oleh pengalaman si estimator dari proyek-proyek sebelumnya.
10. **Kriteria lain**: mencakup berbagai faktor seperti *milestones* yang terperinci, perencanaan yang memadai, staf yang kompeten, komitmen antar anggota tim, dsb.

Yang agak mengherankan adalah bahwa proses perencanaan proyek tidak menduduki posisi atas. Hal ini dapat dijawab dengan kenyataan di lapangan bahwa dalam sebuah proyek, tidak harus selalu mengacu pada rencana awal. Perubahan adalah proses yang biasa, seorang manajer proyek harus mampu berfikir fleksibel agar mampu menyelesaikan proyek agak melenceng dari rencana awal, namun tetap dalam kerangka waktu yang telah diberikan dan dalam batasan dana yang tersedia.

Mengacu pada faktor penentu keberhasilan di atas, maka hambatan-hambatan yang seringkali mengacaukan jalannya proyek IT adalah:

- Estimasi yang terburu-buru;
- Rencana yang tidak matang;
- Kurangnya bimbingan untuk membuat keputusan secara organisasi;
- Kurangnya kemampuan untuk mengukur kemajuan proyek;
- Kurang tepatnya pembagian kerja antara anggota tim;
- Kriteria kesuksesan yang tidak tepat.

Selain itu berdasarkan hasil penelitian dari H.J Thamhain dan D.L. Wilemon yang diterbitkan di *Project Management Journal* Juni 1986 dengan judul *"Criteria for controlling software according to plan"*, dalam proyek pengembangan *software*, para manajer proyek dibebani tugas dan tantangan sebagai berikut:

- Mengatasi *deadlines* (85%);
- Mengatasi keterbatasan sumberdaya (83%);
- Efektif komunikasi antartim kerja (80%);
- Mengatasi komitmen dari tiap anggota tim kerja (74%);

- Pencapaian milestones yang terukur (70%);
- Mengatasi perubahan-perubahan yang terjadi (60%);
- Pengerjaan rencana proyek yang sesuai dengan pembagian tugas (57%);
- Mendapatkan komitmen dari manajemen atas (45%);
- Mengatasi konflik yang terjadi dalam proyek (42%);
- Pengaturan vendor dan sub-sub kontraktor (38%);

Hasil persentase diperoleh melalui survey dari manajer-manajer proyek perusahaan software terkemuka dan setiap manajer diperkenankan menuliskan tugas-tugas mereka dalam proyek lebih dari satu.

8.2 Proses kelahiran proyek

Sebelum seorang manajer proyek dapat melangkah lebih jauh dalam konsep kerja manajemen proyek, syarat utama yang harus dijalankan adalah menetapkan keberadaan proyek sesuai dengan proses kelahiran atau kejadian proyek tersebut. Sebuah proyek dapat terjadi karena dorongan berbagai hal, antara lain:

- Tugas (order) dari atasan (tim *management*, *stakeholders*, sponsors, dll);
- *Stakeholders* adalah suatu istilah untuk pihak-pihak yang menunjukkan perhatiaannya pada kelangsungan sebuah proyek.
- Permohonan dari *client;*
- Inisiatif pelaksana (co.: penelitian);
- Kepentingan bisnis (business need, legal requirement);
- Tuntutan pasar.

Seringkali apa yang yang ditugaskan:
- o Tidak jelas (tujuan, sarana, waktu, biaya)-nya;
- o Mengikuti **trend** teknologi.

Pihak pemberi order acap kali hanya mengikuti trend teknologi yang ada tanpa mengerti arti sebenarnya dari teknologi tersebut. Mereka mengharapkan keuntungan (secara kualitas dan kuantitas) dari kemajuan teknologi secepat dan sebesar mungkin. Dan inilah tugas terberat dari seorang manajer proyek, yaitu harus mampu menyelesaikan proyek pada waktunya dan sesuai dengan keinginan dari si pemberi order. Untuk setiap order dalam sebuah proyek, dibutuhkan penyesuaian ruang lingkup (scope) sehingga apa yang harus dihasilkan pada suatu batasan waktu tertentu dapat menjadi jelas.

Caranya adalah dengan:
- o melalui pendefinisian *requirements* dan *deliverables* lewat jalan riset dan feasibility plan(akan dibahas di bab 3),
- o wawancara dengan pemberi order dan (calon) pemakai, dan
- o informasi dari proyek-proyek sejenis sebelumnya.

8.2.1 Project Charter

Pada tahap awal terbentuknya proyek, kita memerlukan apa yang dikenal sebagai *project charter*. Ini adalah suatu landasan serta definisi formal bagi sebuah proyek. Project charter berisi elemen-elemen yang unik yang hanya berlaku dalam sebuah proyek.

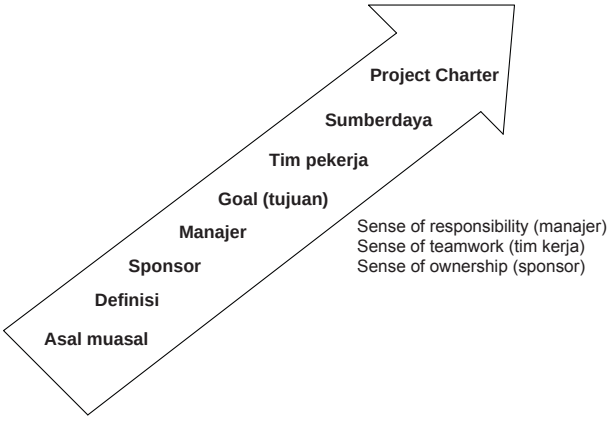
Ada pun elemen-elemen itu adalah:

- Nama proyek resmi;
- Sponsor buat proyek dan kontak informasi;
- Manager proyek dan kontak informasi;
- *Goal* (tujuan) proyek;
- Penjelasan asal-muasal proyek;
- Hasil akhir *Deliverables* dari fase-fase dalam proyek;
- Strategi global dalam pelaksanaan proyek;
- Perhitungan waktu kasar;
- Sarana dan prasarana serta sumberdaya proyek, biaya (kasar), staff, *vendors* / *stakeholders*.

8.2.2 Guna *project charter*

Project charter ini berguna untuk:

- Pendefinisian awal proyek secara jelas;
- Mengenali atribut-atribut suatu proyek;
- Identifikasi autoritas suatu proyek (sponsor, manajer, anggota utama tim kerja);
- Peran kerja orang-orang utama yang terlibat dan kontak informasinya;
- Pondasi yang menopang jalannya proyek (batasan awal dari visi dan misi proyek).



- Sebuah proyek charter akan menumbuhkan:
  - Sense of responsibility/tanggung jawab (manajer)
  - Sense of teamwork/kerja sama (tim kerja)
  - Sense of ownership/kepemikikan (sponsor)
- Setelah project charter terbentuk, akan dilanjutkan dengan feasibility plan dan riset terhadap proyek. Melalui riset ini akan diestimasikan apakah sebuah proyek dapat dijalanankan sesuai pendanaan dan waktu yang ditetapkan.

8.3 Proyek fase

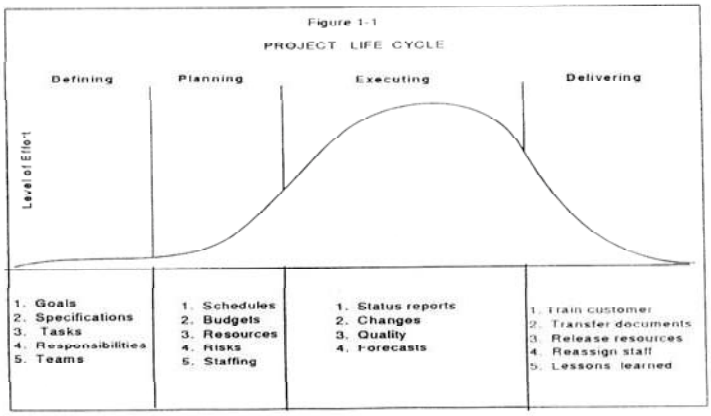
Sebuah proyek dibagi ke dalam fase-fase dan setiap fase menghasilkan suatu bentuk **hasil nyata** tertentu yang dapat digunakan pada fase-fase berikutnya. Setiap fase ditandai dengan selesainya satu atau lebih *deliverables*. Sebuah *deliverable*: dapat dilihat dan dinilai serta diverifikasikan, contoh: hasil studi kelayakan, desain sistem informasi, ataupun software prototip yang dapat digunakan. Proyek fase ini penting untukmenilai performa proyek sampai secara keseluruhan dan tahap penentuan untuk kelanjutan ke fase berikutnya.

8.3.1 Siklus hidup proyek (*Project life cycles*)

Siklus hidup proyek menggambarkan fase-fase global dalam sebuah proyek. Siklus hidup proyek digunakan untuk:

- Menentukan awal dan akhir dari sebuah proyek.
- Menentukan kapan studi kelayakan dilakukan.
- Menentukan tindakan-tindakan transisi.
- Menentukan pekerjaan teknis apa yang harus dilakukan pada setiap fasenya.

Contoh gambaran siklus hidup proyek:



Pada siklus ini proyek terbagi atas empat fase utama (bandingkan juga dengan aktivitas-aktivitas proyek pada bab 1), yaitu:

- Defining (genesis dan pendefinisian proyek);
- Planning (perencanaan proyek);
- Executing (pengimplementasian proyek);
- Delivering (penyerahan hasil proyek kepada yang berhak).

Sebuah siklus hidup proyek, memiliki sifat-sifat umum seperti di bawah ini:

- **Biaya** dan **pengalokasian SDM** rendah pada awal proyek, tinggi pada saat eksekusi dan turun perlahan hingga akhir proyek.
- **Kemungkinan menyelesaikan proyek** terendah (risiko dan ketidakpastian terbesar) pada awal proyek dan kemungkinan sukses semakin besar pada tahap-tahap selanjutnya.
- **Penanam modal** (pemberi order) sangat ber**pengaruh**

pada awal proyek dalam hal menentukan *scope,* biaya dan *deliverables*. Disebabkan: seiring perjalanan proyek banyak **hal-hal tak terduga**, **perubahan-perubahan**, dan **perbaikan**.

8.3.2 Studi Kasus Project Charter

Di bawah ini diberikan sebuah kasus tentang pengembangan dan pembangunan sebuah jaringan komputer. Dari kasus tersebut akan dibuat sebuah *project charter* sebagai langkah awal pelaksanaan proyek.

KASUS: *Upgrade* Jaringan Komputer

- Jaringan komputer sebuah perusahaan (contoh: Universitas Kristen Maranatha) terdiri dari 380 PC memakai OS win/95; 11 server win/NT; dan 5 server Novell NetWare;
- Manajemen perusahaan memutuskan untuk meng-*upgrade* OS semua PC menjadi Win XP, dan semua server, termasuk server NetWare, menggunakan Win 2000 Server.
- Nama proyek: upgrade sistem operasi menuju Win XP dan Win server 2000 dalam lingkungan UKM.
- Sponsor proyek: rektor UKM, CISCO Networking.
- Manajer proyek: kepala NOC.
- Tim kerja proyek: network operation center.
- Tujuan proyek: Semua OS PC akan di-*upgrade* ke Win XP pada akhir tahun (20 Des 2003) ini. Semua server akan di-*upgrade* ke Win 2000 server pada akhir tahun depan (20 Des 2004).

- Kasus bisnis:
  - win 95 telah digunakan selama 5 tahun terakhir ini dalam mengelola bisnis perusahaan.

- Namun saat ini telah ada produk baru yang memiliki kemampuan jauh lebih baik: Win XP.
- Pekerjaan akan lebih produktif, terkendali, aman, dan lebih *user-friendly.*
- Berorientasi teknologi baru spt: jaringan infrared, dan teknologi *web-based* dalam menunjang informasi di perusahaan.
- Menggantikan server yang ada dengan *multi-processors* server yang ditunjang oleh teknologi win 2000 adv. server.
- Win 2000 Adv. Server membantu user dalam menemukan sumberdaya dalam jaringan komputer perusahaan, meningkatkan kinerja jaringan, dan pengamanan yang memadai.

o Hasil proyek yang akan dicapai:
- Instalasi Win XP pada setiap PC
- Instalasi Win 2000 pada setiap server yang ada.
- Seluruh instalasi akan selesai pada 20 Des 2004.

o Penjadualan kasar:
- **Jun**:
- start test metode pengembangan, menginventarisari applikasi setiap pemakai PC, menulis *scripts* untuk proses pemindahan applikasi nantinya.
- **Agust**:
- memulai penggantian (100 user). Mencoba, mendokumentasikan problem dan pemecahannya. Mulai desain Win 2000 Server.
- **Okt**:

- Mulai pelatihan Win XP bagi calon user.
- Sementara itu Win XP mulai diinstalasi pada PC mereka.
- Mengecek masalah-masalah yang mungkin ada, dan helpdesk (support) bagi user.
- Pengetesan tiga server dengan Win 2000.
- **Des**:
- Penyelesaian instalasi Win XP.
- Mulai menginstalasi Win 2000 Server dan membangun infrastrukturnya (utk server-server yang baru) dan inventarisasi problem serta penyelesaian.
- Instalasi untuk server lainnya dimulai.
- Pembangunan infrastruktur diperkirakan memakan waktu satu tahun. 20 Des 2004 keseluruhan proyek selesai.

o Sumberdaya proyek:

- **Perkiraan biaya**: Rp. 800 juta (termasuk biaya software baru, XP, win 2000, lisensi, konsultan, pelatihan).
- Laboratorium pengetesan akan dibooking penuh selama 5 bulan.
- Konsultansi dari CISCO Networking Consultancy.

BAB IX
FEASIBILITAS RENCANA PROYEK

Pada bab terdahulu telah dijelaskan bahwa pada awal proyek, seorang manajer proyek wajib membuat project charter sebagai basis dari proyek keseluruhan. Namun project charter ini tidak cukup karena di dalamnya hanya terdapat anggapan-anggapan (terutama mengenai waktu dan biaya proyek) secara umum, tanpa adanya penelitian apakah memang anggapan itu dapat dijalankan. Untuk membantu realisasi dari proyek, maka pada awal proyek (setelah project charter terbentuk) dibutuhkan adanya riset yang hasilnya dituangkan dalam sebuah feasibility plan, yaitu sebuah rencana proyek yang masuk akal dan paling mungkin untuk dijalankan.

Dalam *project charter* sebuah order terkadang masih terdefinisi secara abstrak (tidak jelas) sehingga hasil akhir proyek yang diharapkan juga masih terkesan abstrak. Misalnya, si pemberi order hanya menyebutkan: jaringan komputer dirasakan terasa sangat lambat, coba gantikan dengan sesuatu yang baru dan cepat. Kadangkala sebenarnya kita tidak perlu mengganti seluruh jaringan komputer yang ada, tapi misalnya cukup dengan mengganti kapasitas bandwith kabelnya saja atau mengganti *switch* yang mulai rusak. Solusi yang ada tidak harus mencari atau membeli yang baru tapi misalnya cukup memperbaiki yang sudah ada.

Untuk menilai apakah tugas dari pemberi order harus dibuat keseluruhan dari mulai nol atau hanya membuat sebagian atau bahkan hanya memperbaiki apa yang sudah ada, diperlukan suatu penelitian (riset) terhadap *order* tersebut.

9.1 Riset proyek

Langkah-langkah apa saja yang dibutuhkan untuk mengadakan riset terhadap proyek?

- Memulai dengan riset terhadap proyek itu sendiri (*purpose statement*).
- Langkah-langkah:
  - Membuat daftar pertanyaan untuk memperjelas keinginan pemberi order (tujuan riset).
  - Menyusun *informasi a*pa saja yang dibutuhkan (*resources*), spt:
    - melalui informasi dari proyek sejenis sebelumnya,
    - buku, internet, brosur-brosur dari *IT vendors*,
    - wawancara dengan client/pemberi order.
  - Mendelegasikan tugas-tugas penelitian (seseorang tidak mungkin melakukan semua secara sendiri saja). Pertolongan tim kerja sangat dibutuhkan. Seorang manajer proyek betapun hebatnya memiliki keterbatasan. Buat pembagian riset yang akan dilakukan kemudian bagikan kepada anggota tim kerja.
  - Mulai bekerja. Mulai dengan membaca, mengevaluasi dan mencatat penemuan-penemuan dalam riset. Dokumentasi situs Internet dapat dilakukan dengan cara mem-

bookmarks halaman Internet yang menjadi sumber informasi. Catatlah buku-buku dan majalah serta artikel yang dapat memberikan informasi dan inspirasi dan jangan lupa untuk mencatat juga halaman yang diacu.

- Merangkumkan hasil riset dan informasi yang diperoleh dalam catatan yang terstruktur. Ini adalah basis dari *feasibility plan* yang akan disusun.
- Mengorganisasikan dokumen-dokumen hasil riset.
- Mengadakan penelaahan ulang (*review*) dengan pemberi order untuk mendapat masukan (*feed-back*) tentang apa yang dimaksud apakah sudah sesuai dengan keinginan. Dan bila masih kekurangan informasi lakukan lagi riset sesuai takaran.

o Sebagai catatan: Jangan menghabiskan waktu hanya dengan riset. Apa yang diperoleh dalam riset harus menjadi titik awal kemajuan proyek dan harus dicoba untuk dimanfaatkan.

9.2 Project Scope Management

Apa yang diperoleh dari *project charter* dan riset merupakan langkah untuk menentukan lingkup (scope) dari proyek yang akan dijalankan. Apa saja yang kita butuhkan untuk menentukan scope dari sebuah proyek?

o Project **objectives**, **vision, mission** dan **goal** (tujuan proyek);
o **Deliverables** (apa yang akan diberikan pada user pada akhir proyek);
o **Milestones** (penanda pencapaian dalam proyek);

- **Technical Requirements** (kebutuhan teknis yang menjamin pemenuhan kinerja yang diminta oleh konsumen);
- **Limits, exclusions** dan **contraints** (batasan dan pernyataan apa yang tidak termasuk);
- **Reviews with customers** (untuk mencek apakah sudah sesuai dengan yang diminta oleh konsumen);
- Secara keseluruhan dituliskan dalam sebuah: **Feasibility Plan.**

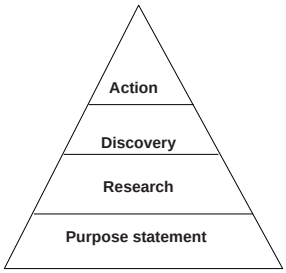
9.2.1 Mendefinisikan ruang lingkup proyek

Project scope statement merupakan definisi hasil akhir atau misi proyek yang dijalani dan bisa merupakan produk atau servis untuk konsumen / klien. Tujuannya adalah agar apa yang akan diberikan sebagai hasil dari proyek pada pengguna akhir menjadi jelas dan agar pelaksanaan proyek dapat lebih fokus kepada tujuan.

9.2.2 Menciptakan feasibility plan

Feasibility plan adalah suatu dokumen yang dihasilkan dari riset terhadap proyek dan sebagai acuan untuk langkah implementasi proyek. Dengan *feasibility* plan validitas dan ruang lingkup proyek, secara keseluruhan atau sebagian, dapat dinilai. Penilaian proyek ini akan dipresentasikan kepada pengguna, pemberi order dan/atau tim manajemen atas. Perlu dicatat juga bahwa setiap proyek IT tidak hanya mengandalkan teknologi yang digunakan tetapi harus mampu memberikan nilai lebih kepada perusahaan, pengguna ataupun pemberi order. Pendek kata harus menghasilkan keuntungan (dinilai dengan uang pada setiap instansi atau perusahaan).

Proses terbentuknya *feasibility plan* dapat digambarkan sebagai berikut:



Dimulai dari pernyataan apa yang akan diriset, kemudian melakukan riset, riset akan menghasilkan penemuan dan ide-ide baru, kemudian ditransfer ke dalam aksi dan implementasi proyek. Sebuah *feasibility plan* terdiri atas beberapa bagian sebagai berikut:

- *Executive summary* (rangkuman *project objectives* and *goal*);
    - Pada bagian awal dari *feasibility plan* harus dituliskan *executive summary*, yang memiliki kegunaan untuk mempresentasikan kepada pembaca mengenai hasil penemuan riset dan untuk mengidentifikasikan rencana-rencana selanjutnya dalam proyek.

- Produk yang akan digunakan (teknologi yang disarankan);
    - Dalam bagian ini dituliskan keuntungan dan keunggulan dari teknologi yang telah dipilih serta alasannya untuk diimplementasikan dalam proyek. Apabila bagian ini dituliskan secara singkat dan jelas maka semua pihak, termasuk yang tidak mengerti teknologipun akan mengerti apa yang menjadi sasaran dari proyek dengan penggunaan teknologi yang telah dipilih.

- Contoh: Perbedaan kualitas produk yang dipilih dengan kompetitor lainnya; support apa saja yang ditawarkan produk terpilih; keberhasilan dari perusahaan lain yang telah mengimplementasikan produk terpilih dsb.

- Akibat yang akan dirasakan oleh pemakai akhir;
  - Hal-hal yang mempengaruhi jalannya perusahaan dan pengguna akhir harus juga dituliskan, sehingga semua pihak yang terpengaruh pada jalannya proyek dapat memposisikan diri dan mengambil langkah-langkah yang dibutuhkan.
  - Contohnya: berapa lama training untuk software yang baru; berapa lama seorang pekerja kantor harus meninggalkan kantor pada saat jaringan baru diimplementasikan; setelah berapa lama *software* ini harus di-*upgrade* kembali; dsb.

- Biaya yang dibutuhkan untuk teknologi yang disarankan;
  - Bagian ini menerangkan tentang perkiraan jumlah biaya yang dibutuhkan untuk pelaksanaan proyek secara keseluruhan, dilihat dari teknologi yang telah dipilih.
  - Contohnya: harga pembelian software;licensing; biaya training; support dari vendor atau distributor; biaya subkontraktor; biaya bulanan dalam operasional kelak; dsb.
  - Dapat juga disertakan pembahasan tentang ROI (*return on investment*), yaitu perhitungan setelah berapa lama si pemberi order atau perusahaan akan memperoleh balik modal.

- Rencana kerja (*action*) yang harus diambil.
  - Dalam bagian ini diterangkan langkah-langkah yang harus ditempuh dalam mengimplementasikan teknologi terpilih dalam proyek.
  - Dapat pula dijelaskan bagaimana teknologi tersebut akan diimplementasikan; sumberdaya apa saja yang dibutuhkan dan mungkin saja rencana untuk menggunakan teknologi lain di masa mendatang yang lebih canggih dapat pula diikutsertakan pada bagian ini.

Secara global dalam *project objective* perlu dituliskan requirements, yaitu tuntutan terhadap produk yang akan dihasilkan. Ada beberapa bagian requirements (terutama dalam sebuah produk perangkat lunak, dikenal dengan istilah *software scope requirements*) yang perlu dianalisa dan dilaporkan dalam rencana kelayakan proyek ini, yaitu:

- Functional requirements, yaitu apa kegunaan (input yang dibutuhkan, proses yang dibutuhkan dan output yang dihasilkan) dari produk yang akan dihasilkan. Untuk mendapatkan requirements yang jelas perlu digunakan metode-metode *system analysis and design*, seperti SDM (*Software Design and Methodology*) atau RAD (*Rapid Application Development*).
- Quality requirements, yaitu hal-hal yang menyangkut kualitas dari suatu produk, misalnya dalam sebuah pengembangan software, perlu dianalisa waktu responsi yang dibutuhkan dalam operasional sebuah fungsionalitas.

- Resource requirements, yaitu penjabaran tentang waktu, biaya dan sumberdaya yang diperlukan untuk

pelaksanaan proyek. Misalnya: berapa modal yang berani dikeluarkan pihak sponsor dalam melakukan penelitian di awal proyek.

Sehingga pada akhir dari riset, melalui feasibility plan, akan terjawab secara tuntas:

Scope dari proyek:

- Tujuan proyek (*objectives* dan *goal)*.
- hasil yang akan dicapai (*deliverables*);
- batasan (*limitations / constraints*: biaya, jadwal, kualitas);
- *Technical requirements*: spesifikasi produk yang harus dicapai, peralatan yang diperlukan, cara pengimplementasian, dsb;
- *Milestones*, rencana kerja global, organisasi kerja;

9.2.3 Aspek pengukuran (measurement)

Secara khusus dalam penyusunan rencana sebuah proyek perangkat lunak (software product), setelah requirements serta objectivitas dari proyek tuntas terdefinisi, perlu dipertimbangkan pula adanya waktu yang dibutuhkan untuk membuat software tersebut secara efektif sampai dapat digunakan di lingkungan pengimplementasian.

Sebuah produk *software* sebagian besar terdiri atas aspek-aspek yang tidak kasat mata sehingga tidak mudah untuk mengukur kegunaannya secara kuantitatif (dalam hitungan angka). Secara global proses pengukuran software terbagi dalam:

o **Predictive measures** (pengukuran dengan perkiraan). Seorang manajer proyek yang berpengalaman (dibantu juga oleh tim kerjanya) akan mampu memperkirakan bagaimana seharusnya sebuah produk nantinya akan

berfungsi. Dalam proses pengembangan produk perkiraan ini akan menjadi pertimbangan juga.
Contoh pengukuran dengan perkiraan antara lain *modularity*, yaitu bagaimana kode dalam software dibagi dalam modul-modul sehingga akan mempermudah proses perbaikan bila ada perubahan nantinya. Berdasarkan pengalaman juga akan diketahui seberapa cepat seorang operator komputer dapat berinteraksi dengan program yang telah dibuat, sehingga dapat diperkirakan bagaimana rancangan sistem interaksi manusia dengan komputer harus dibuat.

- **Performance measures** (pengukuran kinerja). Yang diukur di sini adalah karakteristik serta fungsionalitas software dalam lingkungan pengimplementasiannya. Kinerja diukur setelah software digunakan oleh pemakai. Kita mengenal adanya istilah *prototype*, yang digunakan untuk mengukur kinerja produk yang dihasilkan. Dari feed-back yang didapat selama masa uji coba ini, produk dapat disempurnakan hingga akhirnya diserahkan sepenuhnya kepada pemberi order pada akhir proyek.

Contoh pengukuran kinerja antara lain: waktu responsi bila terjadi error (reliability) dan waktu untuk menggunakan produk bagi pemakai (usability).

9.3 Prioritas proyek

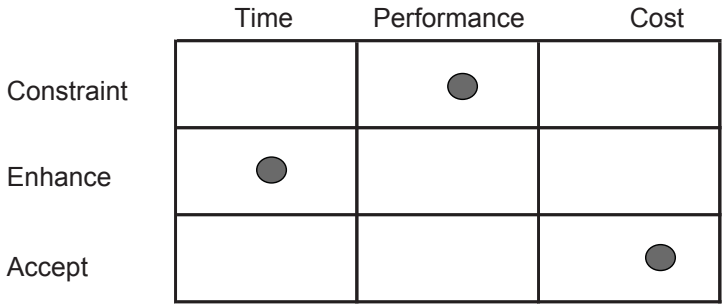
Seperti telah dituliskan pada bab terdahulu bahwa seorang manajer proyek harus mampu mengatur trade-offs (untung rugi) dari waktu, biaya dan kinerja sumberdaya

proyek. Dalam setiap proyek ada bagian prioritas yang diutamakan, entah itu kinerjanya, waktu ataupun biayanya.

Prioritas ini dapat dituangkan dalam sebuah matriks prioritas:

- Constraint / batasan: harus dan pasti;
- Enhance / tingkatkan: perlu dioptimalisasikan;
- Accept / terima: boleh diubah, masih bisa diterima.

Contoh:



Di sini digambarkan bahwa:

- Performance (kinerja) dari hasil proyek tidak dapat diganggu gugat, harus mengikuti requirements yang sudah ada;
- Waktu pelaksanaan proyek masih perlu dioptimalisasikan lagi, dan dapat diatur pada saat pelaksanaan proyek;
- Biaya boleh diubah (fleksibel) asalkan kinerja proyek optimal.

Setelah *feasibility plan* ini selesai, manajer proyek biasanya mempresentasikan kepada tim manajemen atas atau client atau pemberi order untuk persetujuan serta sebagai titik awal (secara waktu dan jadwal) dalam

pelaksanaan proyek, dikenal juga dengan sebutan *project kick-off*. Tahap- tahap selanjutnya yang harus dijalankan adalah:

- Mulai menyusun rencana kerja yang pasti dan terperinci dalam *Work Breakdown Structure* (WBS) dan *Milestone List* serta rencana jaringan kerja (network planning);
- Membentuk organisasi kerja dan tim kerja (human resource management);
- Menyusun dan menjalankan jadwal rencana kerja (time management);
- Review hasil kerja (risk management);
- Penyempurnaan (quality management) dan menyerahkan hasil kerja.

BAB X
MANAJEMEN RESIKO

Setelah hasil dari *feasibility plan* dipresentasikan, proyek dapat dilanjutkan sampai dengan tahap penyelesaiannya. Yang dibutuhkan setelah presentasi *feasibility plan* adalah menambah *input* (masukan) terhadap proyek. Hasil riset yang telah dilakukan mungkin saja harus ditambahkan dengan masukan-masukan baru, sehingga hasil akhir yang diharapkan dapat dicapai. Tambahan masukan untuk proyek ini dapat dilakukan antara lain dengan cara:

- Dari hasil presentasi dengan tim manajemen (*feed-back input*).
- Lewat informasi proyek-proyek sejenis sebelumnya, melalui perpustakaan, Internet, database IT vendors, laporan ilmiah, jurnal ilmiah, dsb.
- Lewat wawancara dengan pemakai akhir dan/atau personal yang pernah menggunakan produk sejenis, sponsor, dsb.

Contoh penambahan informasi untuk kelangsungan proyek ini dapat berupa antara lain:

- Pertanyaan dari pihak managemen mengapa tidak menggunakan teknologi lain yang lebih murah.
- Harapan dari pihak pengguna bahwa program

software yang akan dibuat mudah untuk dimengerti juga oleh mereka yang tidak memiliki basis IT yang kuat.

- Adanya data dari *database* perusahaan bahwa proyek sebelumnya teknologi terpilih ternyata memiliki kelemahan mendasar, seperti ketidakstabilan suatu program yang ditulis dalam Java di dalam lingkungan windows.

Perlu diingat informasi tambahan ini hanya sebagai masukan dan harus dicari solusi pemecahan bila memang menghambat jalannya proyek. Tidak diharapkan bahwa informasi tambahan justru akan membuat proyek menjadi tersendat. Yang tetap menjadi acuan harus tetap *feasibility plan* yang semula. Karena dari *feasibility plan*, diharapkan:

- Memenuhi keinginan pemberi order.
- Dapat menggunakan teknologi yang sepadan dengan criteria.
- Dapat menyusun biaya dan rencana kerja lebih detail (dan mungkin lebih rendah dari perkiraan semula).
- Sebagai bahan untuk presentasi pada pihak manajemen dan pengguna (*report* dan *speech work*) serta dapat dijadikan suatu kekuatan untuk *negotiating position.*

10.1 Presentasi tentang Proyek

Dengan berbekal *feasibility plan*, seorang proyek manajer harus mampu **mempresentasikan** hasil temuannya dan meyakinkan semua pihak untuk menggunakan idenya. Dalam berpresentasi dibutuhkan:

Seorang presentator harus memiliki pengetahuan topik yang mendalam sehingga mampu meyakinkan pihak lain. Presentator harus juga menempatkan dirinya sebagai pemirsa sehingga tahu bagaimana dan apa yang layak dipresentasikan. Bayangkan jika berpresentasi di hadapan tim manajemen perusahaan yang tidak memiliki latar belakang IT, tentu saja akan membosankan bila mereka harus mendengar tentang penjelasan teori jaringan komputer. Bagi mereka yang terpenting adalah penghematan berapa persen yang akan diperoleh perusahaan jika menggunkan jaringan komputer yang diyakini canggih tersebut.

Presentator juga harus bisa memberi pembukaan presentasi yang menarik untuk menggugah perhatian pemirsa. Presentasi juga harus mampu meyakinkan pemirsa bahwa apa yang dipresentasikan itu akan memberi nilai plus bagi proyek, dalam hal ini presentator harus memiliki bukti-bukti kuat yang meyakinkan, baik dari segi teknis maupun praktis. Untuk menjamin bahwa presentasi sukses, seorang presentator harus mampu mambawakan dirinya di hadapan pemirsa. Pembawaan harus tenang, bicara tidak terburu-buru, dan pengucapan harus jelas sehingga pemirsa dapat mengikuti presentasi dengan baik dan sesuai dengan target, yaitu: ide proyek dan teknologi

dapat diterima oleh semua pihak. Dengan demikian Inti *presentasi* sebenarnya adalah: **Menjual produk agar proposal dapat diterima.**

10.2 Wawancara dengan Pihak Terkait

Selain melalui presentasi, informasi juga dapat diperoleh lewat wawancara langsung dengan pihak-pihak terkait. Dalam suatu wawancara harus diperhatikan hal-hal sebagai berikut:

- Tujuan wawancara harus jelas (*Purposes*);
- buat daftar hal-hal yang ingin ditanyakan dan berhubungan langsung dengan proyek (*Enumerating activities*);
- harapan dari pemakai akhir (*Work methods and Interconnections among users*);
- harapan dari pemberi order (*Performance issues*);
- punya pandangan yang lebih jauh melebihi yang diwawancara.

Sebuah wawancara harus memiliki tujuan. Apa yang hendak dicapai harus jelas. Setelah tahu apa yang hendak dicapai, pewawancara harus membuat daftar pertanyaan sebagai arahan pada saat mewawancarai. Tidak boleh dilupakan dalam wawancara, harus juga dilibatkan bagaimana sebenarnya produk dari proyek akan dihasilkan. Apa implikasinya terhadap pengguna dan perusahaan, dan bagaimana akan diperoleh hasil terbaik. Lebih jauh dari ini semua, seorang pewawancara harus memiliki pandangan yang jauh, sehingga tidak kehilangan arah pertanyaan dan didikte oleh lawan bicaranya pada saat wawancara.

10.3 Manajer Proyek yang Efektif

Seorang manajer proyek, seperti telah disinggung dalam bab terdahulu, memiliki tugas untuk mencari keseimbangan antara teknologi, konsep, biaya, dan waktu dalam penyelesaian proyek. Peran ini dapat dipenuhi secara efektif apabila dia mampu untuk:

- Berperan sebagai manajer yang berpengertian (ikuti langkah-langkah positif manajer sebelumnya);
- mendelegasikan tugas bila diperlukan;
- komunikasi antara atasan dan bawahan;
- mengingat peran serta *client* atau pemakai akhir;
- memfokuskan diri pada hasil akhir sesuai tujuan.

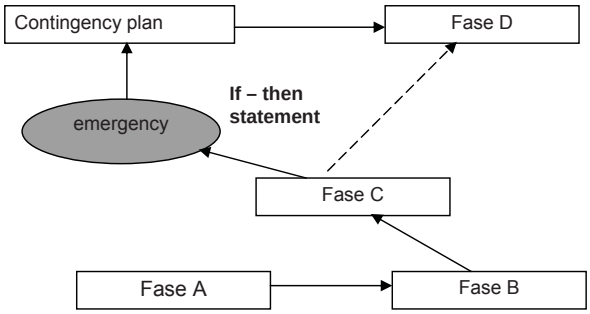
Apabila hal ini dapat dipenuhi, maka proyek akan berjalan sesuai dengan rencana awal dan berhasil dengan baik.

10.4 Contingency Plan

Kadangkala dalam proses penyelesaian proyek dibutuhkan juga suatu *backup-plan* (*contingency plan* atau rencana darurat) apabila ternyata terjadi hal-hal di luar dugaan. Dengan rencana darurat ini proyek tetap dapat diselesaikan pada waktunya dan dalam batasan biaya yang telah direncanakan. Rencana darurat ini dapat juga dikatakan sebagai alternatif pemecahan masalah dan juga harus mendapat persetujuan dari pihak pemberi order. *Contingency plan* dapat disusun bersamaan dengan riset yang dilakukan atau dapat dikatakan pula, *contingency plan* merupakan rencana alternatif yang diperoleh dari hasil riset.

Ada baiknya pula rencana darurat disusun sebelum

bertemu dengan tim manajemen atau pemberi order. Hal ini dikarenakan, mereka sering bertanya hal-hal mendasar karena takut investasinya merugi. *Contingency plan* ini dapat disusun dengan konsep *if-then rule*.



Dengan *contigency plan* ini kita dapat menganalisis konsekuensi terhadap proyek, antara lain apa yang akan terjadi apabila:

- Peningkatan biaya?
- Peningkatan jumlah pekerja?
- Peningkatan waktu kerja?

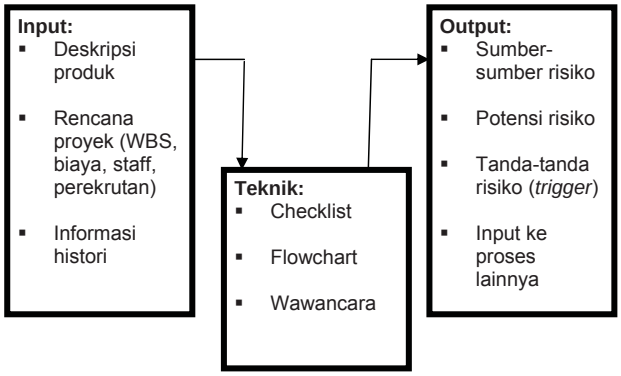
Cara yang dipakai adalah dengan menggunakan teknik-teknik dalam manajemen risiko (***risk management***).

10.5 Manajemen Resiko

Ada pun dalam manajemen resiko tujuan yang hendak dicapai adalah:

- Identifikasi terhadap resiko;
- Evaluasi (analisa) resiko dan (estimasi) pengaruhnya terhadap proyek;
- Mengembangkan responsi terhadap resiko;
- Mengontrol responsi resiko.

1.5.1 Identifikasi Resiko



Identifikasi resiko terdiri atas pengawasan dan penentuan resiko apa saja yang dapat mempengaruhi proyek serta mendokumentasikan setiap dari risiko tersebut. Identifikasi tidak hanya dilakukan sekali, namun harus dilakukan sepanjang perjalanan proyek dari awal sampai akhir.

Faktor internal [di dalam] dan eksternal [di luar] proyek harus diidentifikasi. Faktor internal antara lain penugasan anggota tim kerja, perhitungan biaya dan waktu, serta *support* dan pengaruh dari tim manajemen. Faktor eksternal antara lain melibatkan kebijaksanaan pemerintah, bencana alam, dan hal-hal lain di luar kontrol atau pengaruh tim proyek. Identifikasi terhadap risiko harus melibatkan pengaruh baik maupun pengaruh buruk dari pengaruh faktor-faktor penentu risiko.

Dari gambar di atas dapat dilihat **input** bagi identifikasi risiko adalah:

o Deskripsi produk
  - Produk yang berbasis pada teknologi yang telah dibuktikan kebenarannya memiliki resiko yang lebih kecil dibandingkan dengan produk

yang menuntut inovasi dan penemuan.

- o Rencana proyek
  - *Work breakdown structure:* pendekatan pada *deliverables* setiap unit kerja secara detail. Dengan cara ini identifikasi terhadap risiko bisa sampai ke level yang sangat detail;
  - Estimasi biaya dan waktu: estimasi yang terlalu kasar dan terburu-buru dapat meningkatkan risiko proyek.
  - Penempatan SDM: setiap pekerjaan yang spesifik dan hanya dapat dilakukan oleh orang tertentu meningkatkan risiko proyek, apabila orang tersebut berhalangan untuk hadir;
  - Perekrutan dan sub-kontraktor: pengaruh ekonomi dan kebijakan politik di sekitar proyek dapat menyebabkan fluktuasi nilai kontrak proyek.

- o Informasi historis. hal-hal yang pernah terjadi di masa lalu, dan berkaitan dengan proyek dapat dilihat dari:
  - File-file proyek sejenis dari perusahaan;
  - *database* komersial, contohnya: Internet knowledge-bases;
  - ilmu dan pengalaman dari tim kerja, dikenal juga dengan sebutan: *tacit knowledge*.

Teknik yang digunakan dalam proses identifikasi risiko adalah:

- o *Checklist*: dari informasi (riset, dll) yang diperoleh dapat dibuat *checklist* yang mendata sumber-sumber risiko;
- o *Flowcharting*: dapat digunakan untuk menggambarkan

penyebab dan efek dari risiko yang ada;

- Wawancara: data-data yang tersimpan dari hasil wawancara proyek-proyek terdahulu dapat digunakan sebagai referensi, dan juga masukan dari stakeholders merupakan sumber informasi yang berpengaruh untuk mengidentifikasi risiko.

Ada pun hasil *output* dari pengidentifikasian risiko adalah:

- *Daftar sumber-sumber risiko*
  - Yang seringkali menjadi sumber risiko proyek antara lain: perubahan *requirements*, kesalahan design, pendefinisian peran kerja yang lemah, kesalahan estimasi, dan tim kerja yang kurang mapan.
  - Pada umumnya penjelasan mengenai sumber-sumber risiko ini disertai pula dengan: perhitungan kemungkinan terjadinya risiko tersebut, kemungkinan akibat dari risiko tersebut, kemungkinan kapan terjadinya, pengantisipasian tindakan terhadap risiko tersebut.

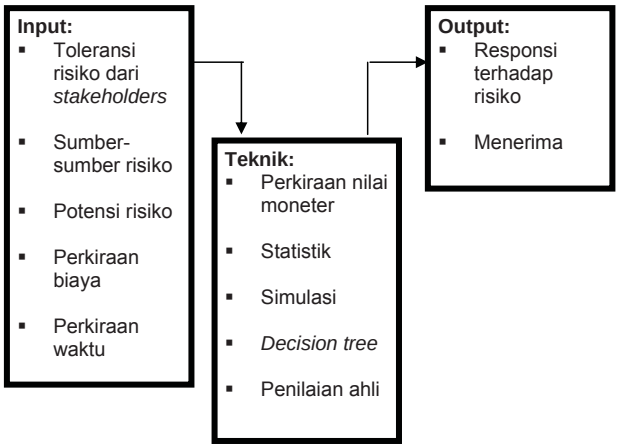
- *Kejadian yang berpotensi menjadi risiko*: biasanya merupakan kejadian-kejadian luar biasa yang jarang terjadi.
  - Contohnya bencana alam, perkembangan teknologi baru yang tiba-tiba.

- *Tanda-tanda datangnya risiko* (*risk symptoms*), sering juga disebut *triggers*, sebab-sebab yang mengakibatkan munculnya bencana pada saat ini.
  - Contohnya biaya yang mengembang pada awal

proyek disebabkan oleh estimasi yang terburu-buru dan tidak akurat.

o *Input pada proses-proses lainnya*: identifikasi risiko mungkin saja menyebabkan diperlukannya pelaksanaan suatu aktivitas di area lain. Contohnya: bila identifikasi risiko memperkirakan bahwa harga barang kebutuhan utama proyek akan naik, maka ada baiknya pada penjadwalan, pembelian barang utama tersebut dilakukan di awal proyek.

10.5.2 Kuantifikasi dan Evaluasi Resiko



Kuantifikasi resiko meliputi pengevaluasian, dan interaksi antara risiko, serta akibatnya.

Input:

o *Toleransi dari stakeholders dan sponsor*: setiap organisasi memiliki toleransi yang berbeda-beda terhadap resiko. Ada yang hanya 10% dari modal, tetapi ada juga yang berani hingga 40% dari modal proyek, asalkan proyek selesai tepat waktu.

- *Sumber resiko* (dibahas di atas);
- *Kejadian yang berpotensi menjadi resiko* (dibahas di atas);
- *Estimasi waktu dan biaya* (akan dibahas pada Manajemen waktu dan biaya);

Teknik:

- *Perkiraan nilai moneter*: bagaimana efek sebuah resiko yang telah dievaluasi nilainya? Mungkin ada yang resiko yang kemungkinannya kecil, tetapi nilai resikonya dapat membuat proyek berhenti. Ada pula resiko yang kemungkinannya besar, tetapi efeknya kecil terhadap jalannya proyek.
- *Perhitungan statistik*: menghitung jangkauan (range) perhitungan minimum dan maksimum untuk biaya dan penjadwalan kerja proyek.
- *Simulasi model*: dengan bantuan model yang disimulasikan dapat diketahui estimasi yang lebih tepat, contoh: penggunaan model statistik Monte Carlo untuk menghitung estimasi durasi proyek.
- *Decision trees*: diagram yang memberikan alur kemungkinan dan interaksi antara keputusan serta akibatnya.
- *Penilaian ahli*: penilaian ahli dapat digunakan sebagai masukan tambahan setelah penggunaan teknik-teknik di atas.

Output:

Setelah dianalisis, manajer proyek harus mampu memutuskan berbuat apa terhadap risiko yang mungkin ada. Menerimanya, membuat rencana lanjutan atau mencari alternatif lain yang tidak terpengaruh resiko.

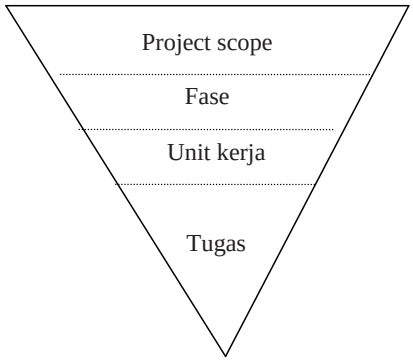
BAB XI
WORK BREAKDOWM STRUCTURE (WBS) DAN PROJECT TIME MANAGEMENT(PTM)

Sebuah proyek adalah suatu himpunan (dikenal juga dengan istilah ruang lingkup / scope) yang terdiri atas aktivitas-aktivitas dalam suatu organisasi kerja yang menghasilkan bentuk nyata (*deliverable*) tertentu dalam suatu batasan waktu tertentu. Sudah barang tentu sebuah proyek memiliki *deadline* (batas waktu penyerahan hasil).

Aktivitas-aktivitas dikelompokkan ke dalam *fase* kerja, di mana setiap fase harus selesai sebelum fase berikutnya dapat dilaksanakan (*sequential*) atau dapat pula dijalankan secara *paralel*. Di dalam setiap fase terdapat *unit-unit kerja*. Unit kerja ini adalah kumpulan tugas-tugas yang membentuk satu kesatuan. Unit kerja ini dapat dibagi lagi menjadi tugas/aktivitas (meskipun tidak selalu), yang didefinisikan sebagai suatu aktivitas tunggal yang membentuk unit kerja. Setiap tugas memberikan suatu hasil bagi unit kerja, yang berkelanjutan pada akhirnya membentuk fase, dapat juga dituliskan bahwa **aktivitas = proses + deliverable**. WBS dapat diartikan sebagai suatu kesatuan logis aktivitas dalam proyek dan *deliverables oriented* (berorientasi pada hasil kerja nyata). WBS ini dapat dibangun segera setelah feasibility plan (scope) selesai terdefinisi dan disetujui oleh pemberi order atau

pihak manajemen atas. Seorang manajer proyek harus membentuk WBS sehingga pada akhirnya dapat menilai apakah proyek berjalan sesuai rencana atau tidak.

Bagian-bagian dalam WBS dapat dilihat sebagai berikut:



11.1 Kegunaaan WBS

WBS dapat dijadikan sebagai alat atau metode untuk:

- Mendefinisikan aktivitas dan rencana keseluruhan yang dibutuhkan dalam proyek. Dengan aktivitas yang terstruktur dari global hingga mendetail, WBS memberikan gambaran global tentang keseluruhan aktivitas proyek, sehingga sesuatu yang tertinggal atau tertinggal untuk didefinisikan dapat terlihat dengan jelas.
- Memberikan gambaran tentang *deadline* dan urgensi dalam proyek.

Dengan memecahkan aktivitas hingga ke bagian detail yang kemudian dilaksanakan oleh tim kerja, WBS dapat menjadi acuan hasil dari proyek. Kapan tim kerja harus menyelesaikan suatu tugas, dan dapat

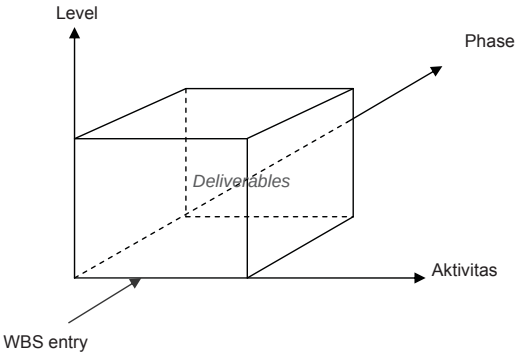
diperhitungkan apabila suatu saat terjadi urgensi dalam suatu aktivitas tertentu.

- Mencegah berkurangnya *scope* pekerjaan.
  Dengan rencana yang mendetail, WBS memberi gambaran total tentang ruang lingkup kerja proyek. Penambahan atau penghapusan unit kerja dari WBS akan memperlihatkan apakah ruang lingkup proyek secara keseluruhan masih dalam batas-batas yang telah disetujui sebelumnya.
- Alat kontrol, komunikasi dan koordinasi.
  Status pekerjaan (selesai, tertunda ataupun dibatalkan) akan terlihat dengan jelas melalui WBS. Seorang manajer proyek dapat menyesuaikan jadwal, berkonsultasi dengan tim kerja atas dasar status aktivitas yang tertera dalam WBS.

11.2 Pembagian level dalam WBS

WBS bersifat hierarkis, dalam pengertian, dimulai dari *scope* proyek hingga detail dalam tugas dalam unit kerja. Sifat WBS ini sering pula disebut *WBS entry*, yaitu term umum pada setiap level dalam WBS, yang selalu menyatakan suatu deliverable.

Penentuan berapa banyak level yang dibutuhkan dalam setiap proyek dapat berbeda-beda, hanya perlu disadari bahwa pembagian level harus sesuai dengan besar proyek. Yang utama adalah bahwa pembagian itu harus mencakup tingkatan dari scope proyek hingga tugas/ aktivitas tim kerja. Abstraksi antara WBS entry, deliverables, WBS level , aktivitas dan fase dapat digambarkan sebagai berikut:

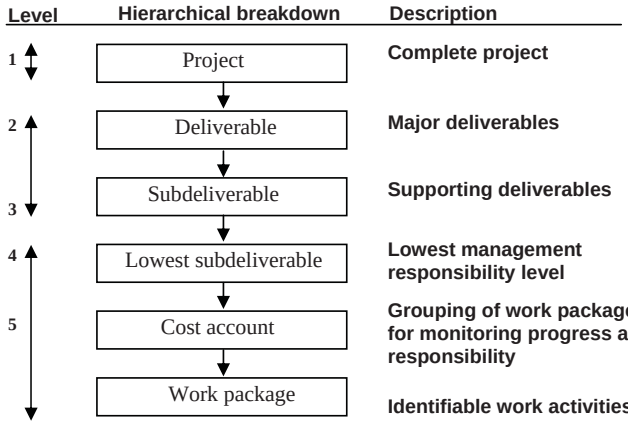


Kubus yang terbentuk adalah scope dari proyek keseluruhan. Scope ini dibagi ke dalam level-level, fase dan aktivitas (tugas/unit kerja). Setiap *entry* pada masing-masing bagian ini disebut WBS-*entry*.

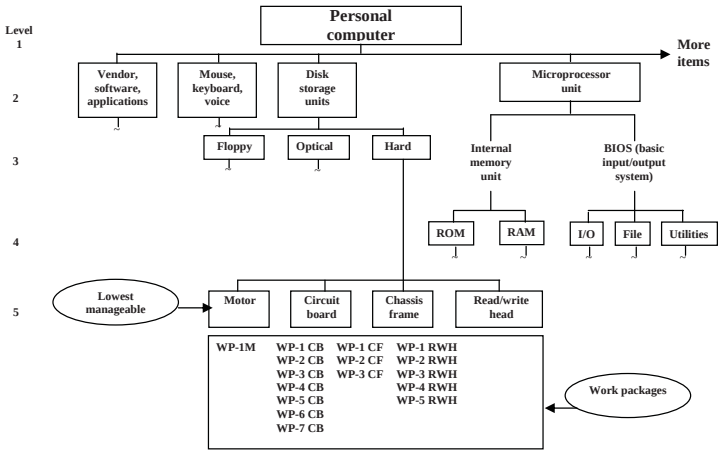
Kesimpulan WBS:

- WBS: pengidentifikasian proyek menjadi elemen kerja yang semakin kecil yang dilakukan secara **hierarkikal**;
- WBS bisa juga disebut **outline** di mana pada tingkat yang berbeda, terdapat tingkat detail yang berbeda, sehingga WBS berguna **untuk mengevaluasi** perkembangan dari segi biaya, waktu dan kinerja teknis sepanjang umur proyek;
- WBS juga **mengintegrasikan** serta m**engkoordinasikan** proyek dan organisasi (seringkali prosesnya disebut **OBS**/*Organization Breakdown Structure*);

Perhatikan contoh berikut, yaitu sebuah WBS yang disederhanakan untuk proyek pengembangan PC yang baru: Pertama kita lihat bagaimana pembagian levelnya sebagai berikut



Kemudian di bawah ini adalah WBS *entry* pada masing-masing level tersebut di atas:



Sebuah proyek pengembangan PC di sini, memiliki *scope* untuk menghasilkan sebuah spesifikasi PC dengan kemampuan menjalankan aplikasi-aplikasi multimedia modern (level 1); terdiri atas: aplikasi, input devices, unit penyimpanan, dan unit microprosesor (level 2). Ambil jalur unit penyimpanan yang terdiri atas: floppy disk,

optical dan hard disk (level 3). Pada level 4 dan 5 dalam jalur *hard disk*, terlihat bagian terkecil dari proyek yang dapat dilihat, yaitu: motor, *circuit board*, *chasis* dan *head*. Setiap dari bagian terkecil ini memiliki unit kerja lagi yang tidak terlihat namun berguna untuk mengaktifkan sebuah *hard disk.* Misalnya putaran pada head untuk dapat menuju kepada cilinder yang tepat pada saat pengambilan data. Bila sudah terdefinisi sampai level detail, maka dengan mudah ditentukan spesifikasi masing-masing komponen (mulai dari unit kerja terbawah), untuk menghasilkan spesifikasi umum multimedia PC. Seperti misalnya memory apa yang harus dipakai, berapa besar hard disknya, *hard disk* dengan interface ATA jenis apa yang diperlukan, dsb.

Level 1 cocok untuk dinilai oleh tim manajemen atas, level 2,3 dan 4 cocok untuk dinilai dan dikelola oleh manajemen menengah, level 5 cocok untuk dikelola oleh manajer yang langsung berhadapan dengan tim kerja lapangan (first line managers). Dengan pembagian yang terperinci seperti ini, pengelolaan proyek terlihat lebih kompleks, namun memberikan suatu gambaran yang jelas dari tingkat umum sampai ke detailnya.

11.3 Proses penyusunan WBS

Untuk membuat suatu WBS, langkah pertama adalah menanyakan hal-hal sebagai berikut:

- Adakah pembagian aktivitas secara logis (terstruktur) di dalam proyek?
- Adakah hasil nyata dalam *Milestones* yang dapat dimasukkan ke dalam setiap fase?
- Adakah hal-hal yang mempengaruhi bisnis secara keseluruhan kepada *client* / organisasi pemberi order?

- Adakah kewajiban-kewajiban finansial yang mempengaruhi jalannya proyek?
- Faktor-faktor apakah dari organisasi secara keseluruhan yang bisa mempengaruhi proyek?
- Adakah proses-proses lainnya yang bukan bagian dari proyek (yang tengah berjalan) sehingga dapat mempengaruhi proyek?

Ini adalah gambaran global tentang scope proyek. Kemudian untuk masing-masing WBS *entry* hingga pada level terendah (unit kerja ataupun aktivitas dan tugasnya), dapat menanyakan hal-hal berikut ini:

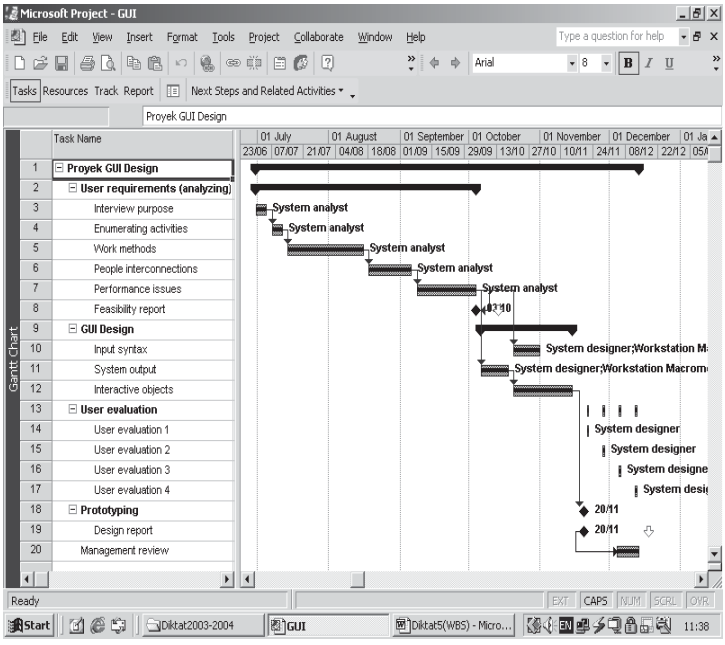
- Mendefinisikan kerja (**apa**).
- Mengidentifikasikan waktu untuk menyelesaikan sebuah paket kerja (**berapa lama**), start-end date.
- Mengidentifikasikan anggaran berjangka waktu untuk menyelesaikan sebuah paket kerja (**biaya**).
- Mengidentifikasikan sumberdaya yang dibutuhkan menyelesaikan sebuah paket kerja (**berapa banyak**).
- Mengidentifikasikan seseorang yang bertanggungjawab atas unit kerja (**siapa**).
- Mengidentifikasikan titik monitoring untuk mengukur perkembangan (**bagaimana**).

Di dalam level WBS (lihat juga contoh di atas), terdapat *sub-deliverables* (*supporting deliverables*, yaitu level 3,4, dan 5), level ini dibuat dengan tujuan sebagai perantara antara unit kerja dengan *deliverables*. Di dalam support *deliverables* ini dituliskan apa-apa saja yang dibutuhkan untuk membentuk deliverables secara total bagi proyek, yang terdiri dari beberapa *work package* yang berasal dari beberapa departemen (unit kerja). *Subdeliverable* tidak memiliki waktu mulai dan selesai yang pasti, tidak

mengkonsumsi sumberdaya, atau mewakili biaya secara langsung.

11.4 WBS dan Gantt Chart

Setelah aktivitas-aktivitas (sampai level terkecil) selesai didefinisikan, maka aktivitas-aktivitas tersebut dapat digambarkan pada sebuah Gantt Chart (diagram kotak / bar chart). Di dalam Gantt Chart ini waktu mulai dan akhirnya sebuah aktivitas dituliskan secara terperinci. Masing-masing kotak/bar merepresentasikan lama sebuah aktivitas berlangsung. Penggunaan project management tools akan sangat membantu proses pembuatan Gantt chart ini. Salah satu contohnya adalah dengan **MS Project**. Contoh WBS yang digambarkan sebagai Gantt chart:



Dapat dilihat di dalam Gantt chart bahwa proyek terdiri dari fase dan setiap fase terdiri dari unit kerja atau aktivitas dengan masing-masing terlihat timeline waktunya (waktu mulai dan akhir).

11.5 Manajemen Waktu Proyek (Project Time Management)

Di dalam *feasibility plan*, estimasi global terhadap waktu proyek sudah didefinisikan. Namun permasalahannya adalah terkadang, dibutuhkan estimasi yang lebih tepat untuk menjamin kelancaran proyek dari awal sampai dengan akhirnya.

Estimasi waktu ini masuk ke dalam bagian Project Time Management. Dalam estimasi waktu secara global, dasar pemikiran yang digunakan adalah:

durasi = banyaknya pekerjaan / sumberdaya yang tersedia;

Lebih jauh dari estimasi global tersebut, ada beberapa pendekatan yang sering dilakukan untuk melakukan estimasi waktu secara lebih terperinci, yaitu:

- **Top-down approach**
  - o Deduktif: mulai dari hal umum menuju spesifik.
  - o Logis dan terstruktur.
  - o Ideal untuk penyusunan estimasi **awal** (seperti pada Feasibility plan) .
  - o Perhitungan global dan tidak terperinci.

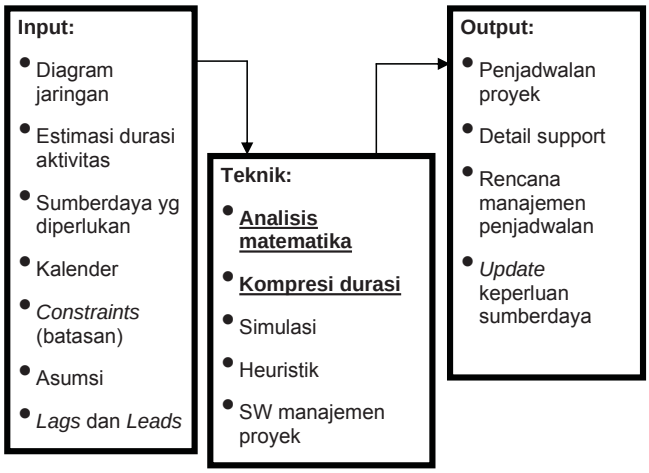
- **Bottom-up approach**
  - o Induktif: mulai dari hal yang spesifik menuju hal yang umum.

- Ideal untuk *brainstorming* (tukar pikiran).
- Perkiraan terinci (langsung ke aktivitas tunggal).

- **Kombinasi** kedua pendekatan di atas untuk estimasi dalam **WBS**.

11.6 Proses Pengelolaan Waktu Kerja

Dalam pengelolaan waktu kerja, termasuk di dalamnya estimasi dan kontrol, dapat digambarkan sebagai proses seperti di bawah ini:



Dengan menggunakan satu atau lebih media input, setelah diproses dengan teknik yang tersedia, akan diperoleh hasil estimasi yang dituangkan sebagai *output* proses estimasi, biasanya dalam bentuk penjadwalan proyek. Di dalam diktat ini yang akan dibahas secara khusus adalah penggunaan teknik secara analisis matematika (CPM dan Pert) dan kompresi durasi (crashing).

11.7 Analisis matematika

Teknik-teknik yang biasa digunakan adalah:

- **CPM** (*Critical Path Method*): mengkalkulasikan langkah-langkah aktivitas proyek secara logis (deterministis) dalam suatu jaringan kerja. Melalui jalur kritis dapat diketahui melalui jalur yang mana proyek dapat dilaksanakan secara optimal;
- GERT (*Graphical Evaluation and Review Technique*): mengevaluasi langkah kerja secara probabilistis dalam suatu jaringan kerja, dengan memperhitungkan bagaimana suatu aktivitas harus dilaksanakan (total, sebagian atau tidak sama sekali) sebelum suatu aktivitas lanjutan dapat dijalankan;
- **PERT** (Program Evaluation and Review Technique): menggunakan urutan logis dalam jaringan kerja ditambahkan dengan perhitungan probalistik pada durasi setiap aktivitas. Pada PERT biasa digunakan perhitungan distribusi rata-rata (*mean distribution*) untuk menghitung durasi setiap aktivitas.

GERT dan PERT jarang digunakan dewasa ini, tetapi estimasi dengan teknik yang menyerupai PERT dapat digunakan untuk CPM, yaitu untuk menghitung durasi rata-rata setiap aktivitas.

11.7.1 CPM (*Network Planning*)

Karakteristik sebuah diagram CPM adalah sebagai berikut:

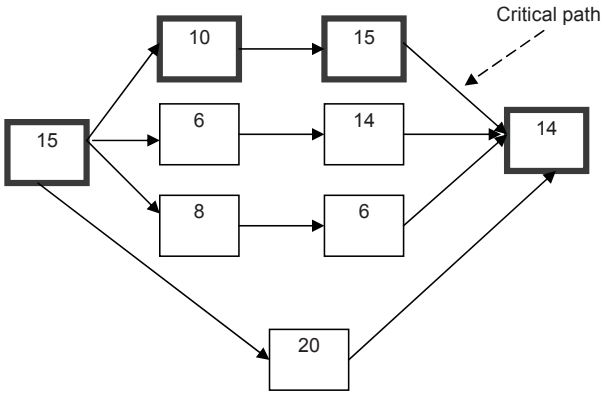
- Adanya sebuah **critical path** dalam sebuah jaringan kerja yang menggambarkan aktivitas berangkai serta menyatakan waktu tersingkat untuk menyelesaikan keseluruhan proyek. Atau dengan kata lain jumlah

aktivitas dengan waktu terpanjang pada suatu jaringan.

- Apabila satu aktivitas pada jalur kritis ini mengalami penundaan maka waktu penyelesaian proyek keseluruhan juga akan mengalami penundaan.
- Analisis aspek waktu secara menyeluruh dengan suatu **logika** sequensial (**deterministis**).
- Fokus pada perhatian pada kemungkinan munculnya masalah dan indikasi utk **mereduksi** biaya dan *delay* atau disebut juga *lag* dalam CPM.
- Elemen dalam CPM: aktivitas, durasi, dan kaitan logis (*relationship*).

Ada dua buah cara pembuatan CPM, yaitu:

1. **Activity on Arrow (AOA):** rangkaian aktivitas dituliskan pada panah, simpul (node) menunjukkan suatu peristiwa (*event)* tercapainya hasil akhir suatu aktivitas;
   Contoh:



Angka pada setiap simpul menunjukkan durasi yang dibutuhkan untuk menyelesaikan suatu aktivitas.

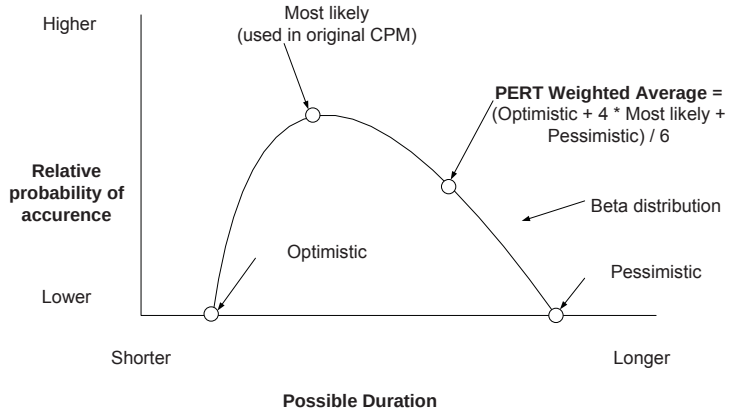
2. **Activity on Node (AON):** rangkaian aktivitas ditunjukkan pada simpul (node), panah menggambarkan arah dari aktivitas (dari aktivitas yang satu menuju aktivitas yang lain).

Yang lebih sering digunakan dewasa ini adalah AON (akan dibahas pada bab selanjutnya).

11.7.2 PERT

PERT merupakan teknik estimasi yang menggunakan metode statistik. Teknik ini berbasis pada peristiwa (*event oriented*) untuk setiap aktivitas. Untuk setiap aktivitas dievaluasi waktu penyelesaian yang paling cepat (optimistis), paling lama (pesimistis) dan yang paling realistisnya. Dari data-data ini, kemudian dihitung distribusi rata-ratanya, dan dianggap sebagai nilai akhir yang paling memungkinkan. Dengan menggunakan teknik PERT maka estimasi akan lebih realistis karena mendasarkan perhitungan pada teori peluang dan variasinya.

Apabila distribusi ini digambarkan, maka pada setiap event akan menghasilkan grafik sebagai berikut:



11.7.3 Kompresi durasi

Kompresi durasi adalah suatu bentuk khusus dari metode analisis matematis. Lewat kompresi durasi akan diupayakan suatu cara untuk memperpendek durasi proyek tanpa mengurangi ruang lingkup proyek. Teknik-teknik yang digunakan untuk melakukan kompresi durasi ini, antara lain:

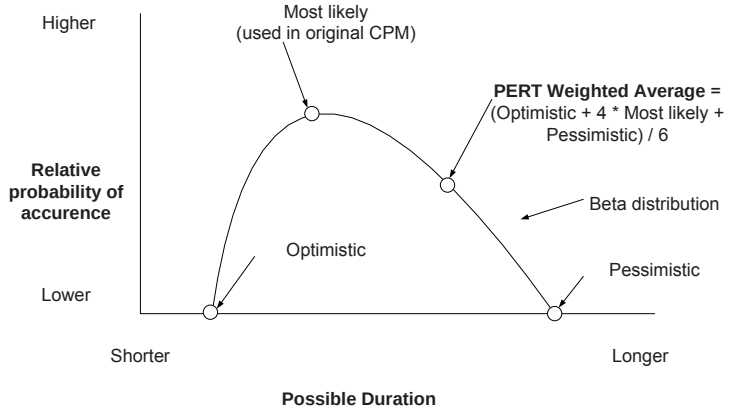
- **Crashing:** di mana perbedaan biaya dan waktu dalam setiap durasi dianalisis untuk menentukan perpendekan durasi yang bagaimana yang optimal (perpendekan waktu terbesar, dengan biaya terendah). Biasanya crashing ini tidak menghasilkan suatu alternatif yang menguntungkan bagi proyek dan hampir selalu terjadi peningkatan biaya proyek. Teknik ini akan dibahas lebih lanjut pada bab berikutnya, bersamaan dengan proses pembuatan jaringan kerja AON.
- **Fast tracking:** melakukan aktivitas secara paralel yang biasanya dilakukan secara berurutan. Teknik ini hampir selalu memperbesar risiko proyek secara keseluruhan. Contohnya penulisan kode pada pembuatan *software*, sebelum rancangan selesai.

BAB XII
PROJECT NETWORK

Jaringan proyek adalah alat yang digunakan untuk **perencanaan**, **penjadwalan** dan **pengawasan** perkembangan suatu proyek. Jaringan ini dikembangkan dari informasi yang dikumpulkan untuk WBS dan merupakan **grafik diagram alir** untuk rencana pekerjaan proyek. Jaringan ini menampilkan aktivitas proyek yang harus diselesaikan, **urutan logisnya**, **ketergantungan** satu aktivitas dengan yang lainnya, dan juga waktu penyelesaian suatu aktivitas dengan waktu start dan finishnya, serta jalur yang terpanjang di dalam suatu network – disebut juga **critical path** (lihat bab 5). Dengan terbentuknya jaringan ini, seorang manajer proyek dapat membuat keputusan yang menyangkut masalah penjadwalan, biaya dan kinerja proyek.

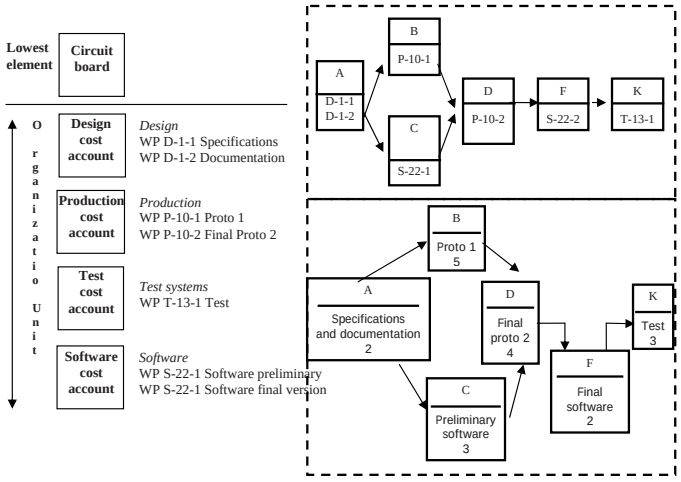
Aktivitas-aktivitas yang tertera pada jaringan kerja proyek, merupakan penggunaan hasil yang telah diperoleh dari proses WBS. Setiap aktivitas dalam WBS, dapat diterjemahkan langsung sebagai suatu aktivitas dalam jaringan kerja. Dan lama waktu penyelesaian untuk aktivitas tersebut dapat dilihat pada Gantt chart-nya.

Proses penyusunan diagram jaringan kerja ini mulai dari WBS-entry (level 1 sampai level-level berikutnya) dapat digambarkan sebagai berikut:



Di sini terlihat untuk aktivitas D pada level 1, dapat diuraikan menjadi beberapa sub-aktivitas pada level 2, yang juga terangkai secara logis. Demikian pula aktivitas di level 3, merupakan rangkaian sub-aktivitas dari level 2.

Bila ditinjau sekarang dari level terbawah, yang langsung mendefinisikan aktivitas yang memberikan hasil nyata, maka penguraian ke dalam jaringan kerja, dapat dilakukan sebagai berikut:



Lintasan atau jalur dari A-B-D-F-K merupakan jalur kritis dengan durasi total 11 satuan waktu.

12.1 Istilah-istilah dalam Jaringan Kerja

Di dalam konteks jaringan kerja, istilah-istilah berikut ini memegang peranan yang sangat penting untuk dapat memahami, membuat dan mengevaluasi suatu jaringan kerja (*project network*).

- **Activity** - aktivitas, yaitu elemen yang memerlukan waktu.
- **Merge activity** – aktivitas gabungan, yaitu aktivitas yang memiliki lebih dari satu aktivitas yang mendahuluinya.
- **Parallel activity** – aktivitas paralel, yaitu aktivitas-aktivitas yang dapat terjadi pada waktu bersamaan, jika diinginkan.
- **Burst activity** – aktivitas yang memiliki beberapa aktivitas yang perlu dilakukan sesudah aktivitas ini selesai.
- **Path** – jalur, suatu urutan dari aktivitas-aktivitas yang tergantung satu sama lain.
- **Critical path** – jalur kritis, jalur dengan waktu (durasi) terpanjang yang terdapat di suatu jaringan kerja. Jika satu atau lebih aktivitas yang ada di jalur kritis tertunda, maka waktu penyelesaian seluruh proyek akan tertunda sebanyak waktu penundaan yang terjadi. Bisa saja terjadi dalam suatu jaringan kerja akan terbentuk lebih dari satu jalur kritis, namun hal ini jarang terjadi.
- **Event** – kejadian, adalah satu titik waktu di mana suatu aktivitas dimulai atau diselesaikan. Tidak membutuhkan waktu.

12.2 Pendekatan Jaringan Kerja dan Konsepnya

Di bawah ini akan dijelaskan bagaimana konsep suatu jaringan kerja dan asumsi-asumsi apa saja yang ada di dalamnya:

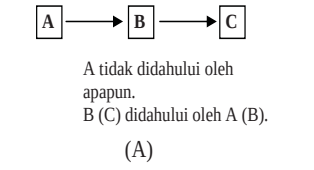
- **Activity-on-node** (AON) – aktivitas digambarkan di dalam suatu node (simpul).
- **Activity-on-arrow** (AOA) – aktivitas digambarkan pada panah.
- Pada kenyataannya, lebih banyak yang menggunakan **AON**, dan yang akan. **Aturan-aturan dasar AON:**
    - o Jaringan biasanya dari kiri ke kanan;
    - o Satu aktivitas tidak dapat mulai sampai semua aktivitas pendahulunya selesai;
    - o Panah-panah di dalam jaringan mengidentifikasikan pendahulu dan alurnya;
    - o Panah dapat bersilangan;
    - o Dua aktivitas (node) yang saling berhubungan namun tidak berpengaruh pada jadwal keseluruhan proyek, dihubungkan dengan panah pelengkap (*dummy*), biasanya digunakan pada AOA;
    - o Setiap aktivitas harus memiliki nomor identifikasi unik;
    - o Sebuah nomor identifikasi aktivitas harus lebih besar dari aktivitas yang mendahuluinya;
    - o *Looping* (pemutaran balik) tidak diperbolehkan, jadi panah loop tidak boleh ada;
    - o Pernyataan kondisi tidak diperbolehkan (contoh: jika aktivitas a sukses, maka…. ⊚ tidak boleh);
    - o Pengalaman menyarankan jika ada beberapa point untuk memulai, satu node awal dapat

digunakan untuk mengidentifikasikan kapan proyek dimulai;

- Hal ini juga berlaku untuk mengidentifikasi akhir yang jelas.

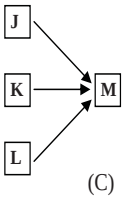
12.3 Activity on Node (AON)

Di dalam pengaplikasian konsep kerja AON, ada beberapa dasar yang harus diketahui. Dasar ini akan mempengaruhi cara pandang terhadap proyek dan aktivitasnya.



A tidak didahului oleh apapun.
B (C) didahului oleh A (B).

(A)

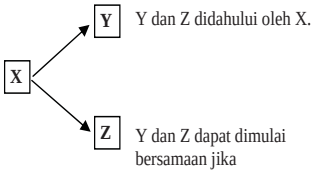


J,K, dan L dapat dimulai bersamaan (pada dasarnya merupakan aktivitas paralel)

tetapi

J,K, dan L harus selesai sebelum M dimulai.

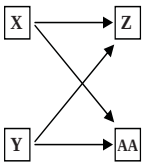
(C)



Y dan Z didahului oleh X.

Y dan Z dapat dimulai bersamaan jika dikehendaki.

(B)

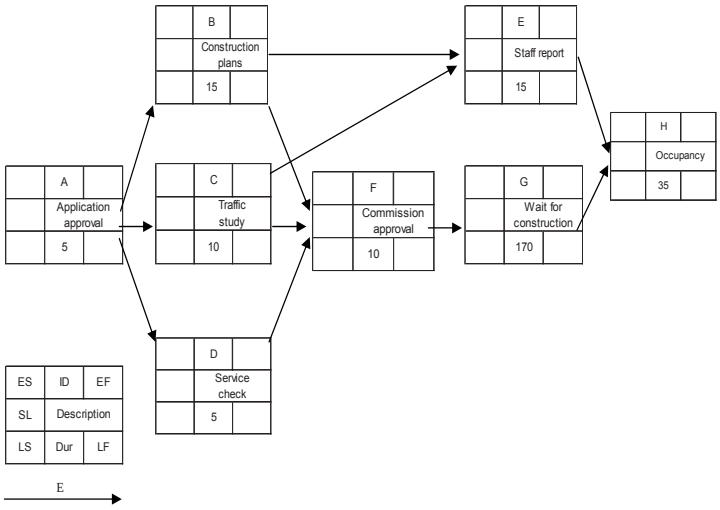


Z didahului oleh X dan Y.

AA didahului oleh X dan Y.

(D)

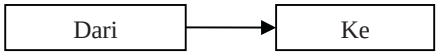
Contoh AON lengkap dengan durasinya adalah sebagai berikut:



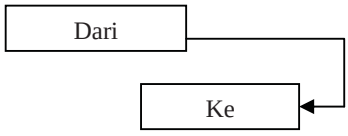
12.4 Precedence Diagramming Method (PDM)

Konsep kerja AON di atas juga mengimplementasikan apa yang dikenal sebagai Precedence Diagramming Method (Metode Diagram Pendahuluan). Di dalam PDM ini dikenal istilah-istilah sebagai berikut

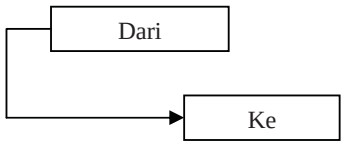
- **Finish-to-start (FS)**: aktivitas "dari" harus selesai sebelum aktivitas "ke" boleh dimulai;



- **Finish-to-finish (FF)**: aktivitas "dari" harus selesai sebelum aktivitas "ke" boleh selesai;



- **Start-to-start (SS)**: aktivitas "dari" harus dimulai sebelum aktivitas "ke" boleh dimulai (dengan kata lain aktivitas "dari" dan "ke" boleh mulai bersamaan);



- **Start-to-finish (SF)**: aktivitas "ke" tidak boleh selesai sampai aktivitas "dari" dimulai;

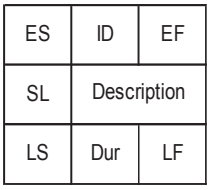
- **Lead**: perubahan pada logika aktivitas yang mengijinkan percepatan "ke" aktivitas.
- **Lag**: perubahan pada logika aktivitas yang menyebabkan perlambatan / penundaan (delay) pada "ke" aktivitas karena harus menunggu "dari" aktivitas selesai.
- **Hammock**: rangkuman dari aktivitas. Aktivitas-aktivitas yang berhubungan diperlihatkan sbg satu kesatuan.
- **Slack** (**Float**): Jumlah waktu yang diijinkan dalam perlambatan suatu proyek dari waktu dimulainya tanpa memperlambat waktu akhir penyelesaian proyek keseluruhan.

12.5 Network Forward dan Backward Pass

Terlebih dahulu diperkenalkan istilah-istilah sebagai berikut:

- **Forward pass** – Earliest Times (dasar perhitungan adalah waktu tercepat)
  - Seberapa awal sebuah aktivitas dapat dimulai? (early start – **ES**)
  - Seberapa awal sebuah aktivitas dapat diselesaikan? (early finish – **EF**)
  - Seberapa awal sebuah proyek dapat diselesaikan? (time expected – **TE**)
- **Backward pass** – Latest Times (dasar perhitungan adalah waktu terpanjang dalam proyek), dapat digunakan untuk menghitung *float*.
  - Seberapa terlambat sebuah aktivitas dapat dimulai? (late start – **LS**)
  - Seberapa terlambat sebuah aktivitas dapat diselesaikan? (late finish – **LF**)
  - Berapa lama sebuah aktivitas dapat ditunda? (slack or float – **SL**)
- Aktivitas-aktivitas mana yang merepresentasikan jalur kritis/critical path (**CP**)?

Pada setiap aktivitas digunakan node (simpul) dengan model sebagai berikut:



Untuk arti masing-masing kotak lihat keterangan di atas.

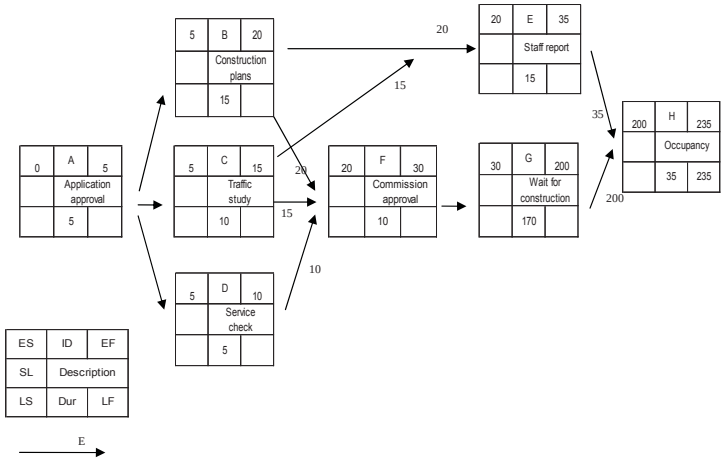
Tambahan untuk: **ID** menunjukkan aktivitas; **Description** memberi keterangan kepada aktivitas; **Dur** menunjukkan durasi aktivitas tersebut.

12.5.1 Forward pass

Berarti pembuatan jaringan dimulai dari aktivitas awal hingga aktivitas terakhir.

Syarat penggunaan:

- Setiap simpul memiliki EF yang dihitung dengan cara **EF = ES + Dur**;
- EF pada aktivitas "dari" dapat dibawa menuju ES pada aktivitas "ke", kecuali;
- Aktivitas "ke" diawali beberapa aktivitas lainnya (merge activity), dalam hal ini harus dipilih nilai EF terbesar pada aktivitas yang mengawalinya.
- Contoh:

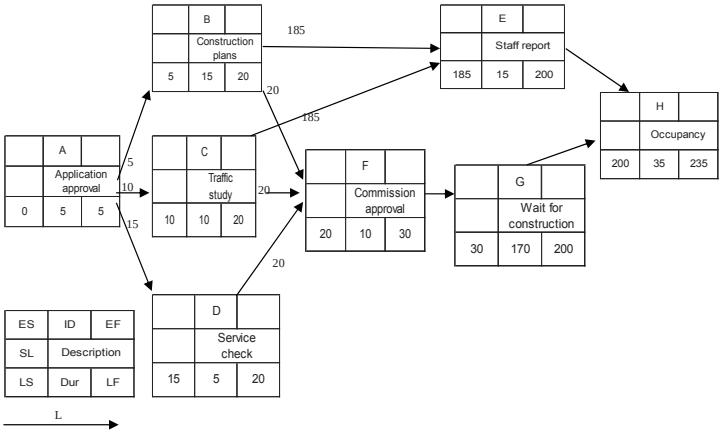


12.5.2 Backward pass

Berarti pembuatan jaringan dimulai dari aktivitas terakhir terus dihitung mundur hingga aktivitas awal.

Syarat penggunaan:

- Setiap simpul memiliki LS yang dihitung dengan cara **LS = LF - Dur**;
- LS pada aktivitas "ke" dapat dibawa menuju LF pada aktivitas "dari", kecuali;
- Aktivitas "dari" mengawali beberapa aktivitas lainnya (burst activity), dalam hal ini harus dipilih nilai LS terkecil yang berasal dari aktivitas "ke".
- Contoh:



12.5.3 Menghitung SLACK

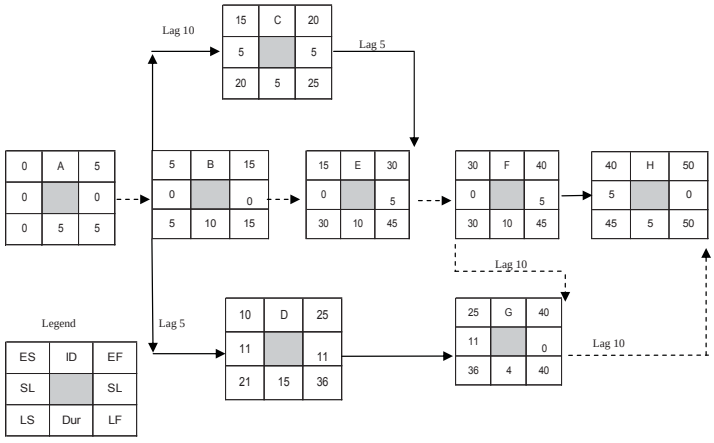
Setelah *Forward* dan *Backward pass* terkonstruksi, maka dapat dihitung nilai slack-nya, yaitu mengevaluasi aktivitas mana saja yang dapat ditangguhkan pelaksanaannya dan berapa lama dapat ditangguhkan. Caranya adalah dengan mencari selisih antara LS dan ES atau **SL = LS – ES**.

Setelah slack dari masing-masing aktivitas dikalkulasi, dapat terlihat dengan jelas aktivitas mana saja yang membentuk critical path / jalur kritis. Ini ditandai dengan mencari aktivitas-aktivitas yang nilai slack-nya nol.

12.5.4 Menggunakan LAG

Lag dapat diartikan sebagai waktu penundaan untuk memulai aktivitas karena menunggu aktivitas pendahulu selesai. *Lag* biasanya digunakan pada aktivitas-aktivitas yang saling bergantung. Sebagai contoh:

Aktivitas C dan D di bawah ini tergantung pada aktivitas B (Start on Start). Dimulainya aktivitas C relatif terhadap aktivitas B adalah 10 satuan waktu (C dapat dimulai apabila B telah dimulai, ditambah lagi dengan 10 satuan waktu). Kemudian penyelesaian aktivitas H, harus menunggu 10 satuan waktu setelah aktivitas G selesai.

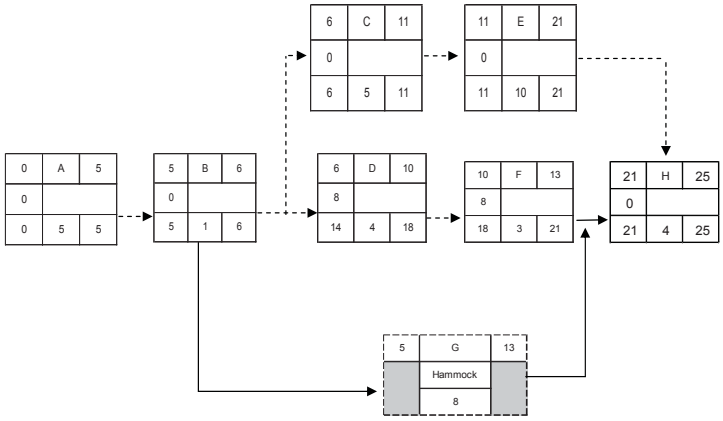


1.5.5 Aktivitas Hammock

Aktivitas Hammock, berarti penggabungan beberapa aktivitas sebagai satu kesatuan, misalnya untuk memudahkan pengaturan penggunaan sumberdaya pada aktivitas-aktivitas yang disatukan tersebut. Durasi aktivitas Hammock adalah jumlah total durasi aktivitas-aktivitas yang digabungkan. Contoh:

Pada contoh di bawah ini, aktivitas B,D dan F digabungkan menjadi satu aktivitas. ES dari aktivitas

Hammock ini adalah: 5 (dari aktivitas B), EF adalah 13 (dari aktivitas F), dan durasinya adalah 1 + 4 + 3 = 8 (B,D,F).



12.6 Kompresi Durasi (Crashing)

Ada beberapa alasan mengapa pada saat pelaksanaan proyek dibutuhkan percepatan, antara lain:

o Adanya pembatasan *deadline* di luar rencana, misalnya: perbaikan jaringan komputer pada suatu instansi harus dipercepat karena gedungnya akan diperbaiki;

- o Sebagai kompensasi penundaan pelaksaan aktivitas-aktivitas proyek pada jalur kritisnya;
- o Untuk mengurangi risiko pada hal-hal yang mendadak, misalnya: ada isu bahwa harga perangkat komputer akan naik tajam pada masa akhir proyek;
- o Mengurangi biaya-biaya langsung pada jalur kritis;
- o Untuk mengalihkan sumberdaya pada proyek atau aktivitas lainnya;
- o Adanya bonus dari pihak pemberi order jika proyek dapat diselesaikan sebelum *deadline*.

Dengan berdasarkan pada salah satu dari alasan ini, seorang manajer proyek dapat mengambil tindakan untuk mempercepat laju proyek. Hal ini dikenal dengan istilah *gcrashin*.

Crash time merupakan tindakan untuk mengurangi durasi keseluruhan proyek setelah menganalisa alternatif-alternatif yang ada (dapat dilihat dari jaringan kerja) untuk mengoptimalisasikan waktu kerja dengan biaya terendah. Seringkali dalam crashing terjadi "**trade-off**", yaitu pertukaran waktu dengan biaya. Biaya yang dikeluarkan untuk melakukan crashing disebut: **crash cost**.

Waktu untuk crashing:

- Jika melakukan di awal proyek mungkin dapat berakhir menghabiskan biaya dengan percuma. Dalam beberapa kasus, biaya besar yang dihabiskan di awal proyek akan hilang begitu saja dan kurang bermanfaat.
- Akan tetapi bisa saja perpendekan di awal bisa berguna jika kita sudah memperkirakan bahwa kemungkinan tertundanya aktivitas-aktivitas kritis di belakang cukup tinggi. Maka perlu dipertimbangkan dulu kapan yang paling baik.

Meskipun crashing dianggap sesuatu yang positif, namun harus diperhitungkan kelemahan dari *crashing* antara lain:

- o Mengurangi kualitas proyek;
- o Mengsubkontrakkan (outsourcing) sebagian aktivitas;
- o Menambah sumberdaya;
- o Menyusun kembali logika jaringan kerja;
- o Mengurangi cakupan proyek;

- Bertambahnya biaya produksi langsung (*trade-off* dengan biaya).

Dalam melakukan implementasi *crashing* ada beberapa asumsi yang harus diperhatikan:

- **Asumsi linearitas**: belum tentu hubungan antara waktu dan biaya adalah linear. Untuk mengatasi hal ini bisa saja digunakan teknik *present value* agar lebih akurat (akan dibahas lebih lanjut).
- **Solusi komputer**: hati-hati, jangan terlalu mengandalkan hasil perhitungan crashing komputer, karena komputer tidak memperhitungkan resiko dan ketidakpastian.
- Bagaimana kita tahu apakah crashing cukup berharga? Jawabannya adalah *tergantung*. Resiko harus dipertimbangkan. Untuk ini, dapat dilakukan *analisa sensitivitas* untuk proyek.

12.6.1 Prosedur crashing

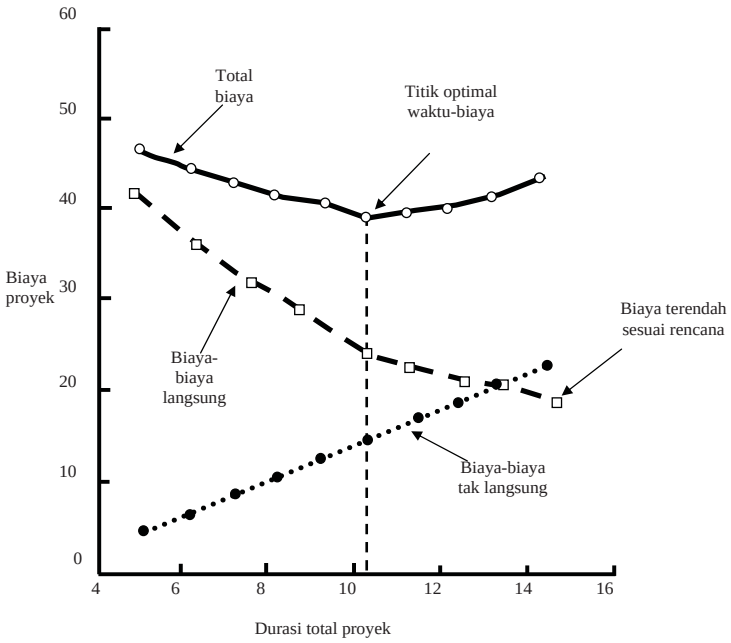
Untuk melakukan *crashing,* diperlukan pengamatan yang cermat dan analisis terhadap kemungkinan-kemungkin yang ada. Dalam melakukan analisis untuk crashing, harus dilakukan konstruksi waktu dan biaya.

Tiga langkah diperlukan untuk mengkonstruksikan *grafik waktu-biaya*:

- Cari *total biaya langsung*, contoh: biaya pegawai dan peralatan yang dipakai dalam proyek, untuk lama **proyek yang telah dipilih.**
- **Cari** *total biaya tidak langsung*, contoh: biaya konsultansi dengan consultan dan biaya administrasi, untuk lama proyek yang telah dipilih.
- **Totalkan** biaya langsung dan tidak langsung untuk lama proyek yang telah dipilih tersebut.

Grafik waktu-biaya ini digunakan untuk membandingkan alternatif tambahan biaya dan menilai manfaatnya bagi proyek secara keseluruhan. Tantangan tersulit yang dihadapi dalam mengkonstruksikan grafik biaya-waktu ini adalah mencari total biaya langsung untuk lama proyek tertentu dalam jangka waktu yang relevan. Misalkan saja pengurangan waktu selama 3 hari, mungkin akan lebih mahal dibandingkan dengan pengurangan waktu selama 5 hari (ditinjau dari biaya tidak langsung dan sumberdaya), sedangkan usaha yang dibutuhkan untuk melakukan pengurangan selama 5 hari jauh lebih besar. Maka akan dipilih pengurangan waktu selama 3 hari, karena lebih optimal ditinjau dari perbandingan biaya dan waktunya.

Contoh grafik waktu-biaya:



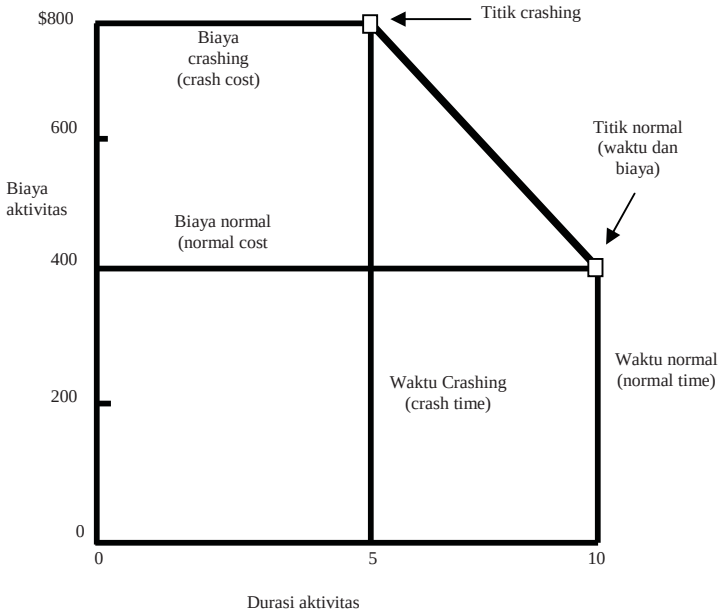
Dari contoh grafik ini terlihat bahwa proyek dengan durasi selama 10 satuan waktu adalah yang paling optimal.

Pertimbangan utama pada saat *crashing* adalah menentukan **aktivitas mana yang perlu dikurangi** dan sebagaimana jauh untuk melaksanakan pengurangan proses. Manajer perlu mencari *aktivitas kritis* yang dapat diperpendek dengan tambahan biaya per unit waktu yang terkecil. Rasional untuk memilih aktivitas ini tergantung dari pengidentifikasian aktivitas normal dan waktu perpendekan serta biaya yang berhubungan dengannya. Yang perlu menjadi masukan juga adalah normal time (waktu normal), yaitu penyelesaian aktivitas dalam kondisi normal yang telah direncanakan sebelumnya. Jika perbedaan waktunya tidak signifikan, maka crashing tidak memberikan nilai lebih bagi proyek, malah hanya akan memperbesar risiko proyek secara keseluruhan.

Informasi mengenai normal time maupun crash time serta crash cost didapatkan dari orang yang paling familiar dengan penyelesaian aktivitas, seperti dari: pekerja dalam tim, konsultan ataupun pihak sponsor. Setelah informasi ini didapat kemudian dibuatlah grafik aktivitas.

12.6.2 Grafik aktivitas

Dalam melakukan analisa untuk crashing, sebelum mengevaluasi grafik biaya-waktu, harus diamati grafik per aktivitas (untuk menentukan **biaya langsung**) dalam proyek. Analisa ini berlaku untuk semua aktivitas di dalam proyek, tidak hanya yang berada di jalur kritis karena dengan mengubah durasi pada salah satu aktivitas, jalur kritis proyek bisa berubah.



Asumsi pada grafik aktivitas dalam kondisi **ideal** adalah:

1. Hubungan **waktu-biaya** adalah linear.
2. **Normal time** adalah penyelesaian dengan biaya rendah dan metode yg efisien.
3. **Crash time** adalah limitasi – waktu perpendekan terbesar yg memungkinkan.
4. **Slope** mewakili biaya per unit waktu (konstan).
5. Semua **percepatan** harus terjadi dalam normal time dan crash time.

Bila harga dari slope sudah diketahui maka seorang manajer proyek dapat membandingkan *aktivitas pada jalur kritis* yang mana yang dapat dipilih untuk diperpendek.

Rumus untuk slope adalah:

Slope = (biaya crash – biaya normal) / (waktu normal – waktu crash);

Contoh:

Activity ID	Slope	Maximum crash time	Direct costs			
			Normal		Crash	
			Time	Cost	Time	Cost
A	20	1	3	$50	2	$70
B	40	2	6	80	4	160
C	30	1	10	60	9	90
D	25	4	11	50	7	150
E	30	2	8	100	6	160
F	30	1	5	40	4	70
G	0	0	6	70	6	70

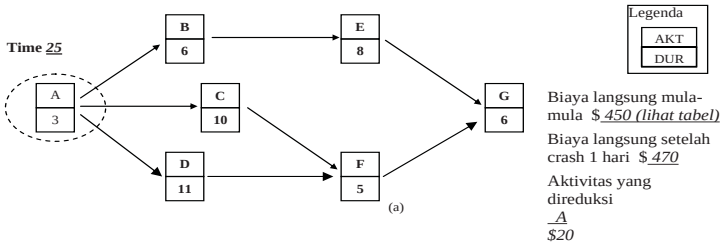
Total biaya $450

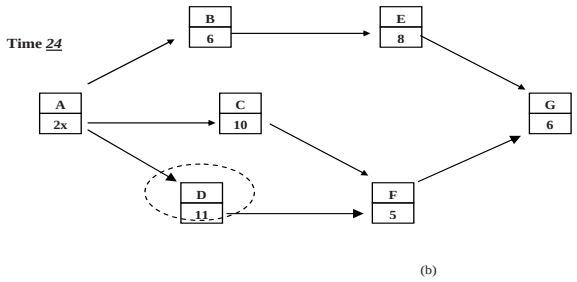
- Setelah menghitung *slope* dan waktu crash maksimum pada tabel di atas, mulailah proses crashing dengan melihat *nilai slope terkecil pada jalur kritis_(critical path)* dalam jaringan kerja sebagai titik awal proses *crashing*.
- Setelah terbentuknya jaringan kerja yang baru, lihat kembali pada *critical* pathnya, kemudian pilih kembali aktivitas dengan nilai slope terkecil untuk melanjutkan proses crashing. Beberapa pengecualian:
  - Jika terbentuk beberapa jalur kritis, maka pilihlah aktivitas yang merupakan "aktivitas gabungan (*merge activity*)" dari jalur-jalur kritis yang terbentuk.
  - Jika *keseluruhan jalur* dalam jaringan menjadi kritis, karena "aktivitas gabungan" tidak dapat direduksi lagi, maka pilihlah slope terkecil dari aktivitas yang masih dapat direduksi pada jalur-jalur kritis tersebut.
  - Aktivitas yang sudah berada dalam kondisi

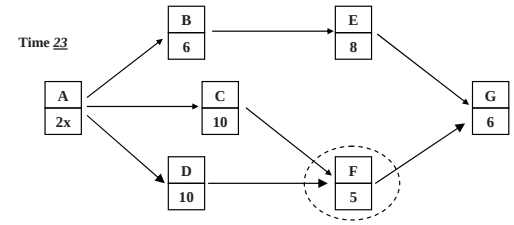
"waktu maksimum crashing", tidak dapat dipilih kembali. (Lihat prosedur selengkapnya dalam lampiran C).

Proses *crashing* dapat dilangsungkan sejauh yang kita inginkan, namun perlu diingat bahwa yang ingin dicapai adalah biaya yang optimal dengan waktu penyelesaian yang tersingkat. Sisi lain yang harus diperhatikan adalah waktu crashing maksimum (*maximum crash time*) pada setiap aktivitas. Bila aktivitas-aktivitas pada jalur kritis yang ada sudah tidak dapat direduksi lagi, maka proses crashing harus dihentikan. Sebagai contoh, lihat proses *crashing* dalam gambar-gambar jaringan kerja sebagai berikut.

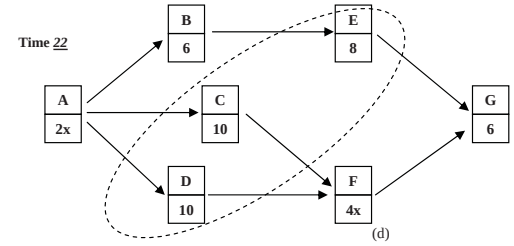
Time_25 adalah kondisi awal jaringan dan kita berupaya untuk mengoptimalkannya.



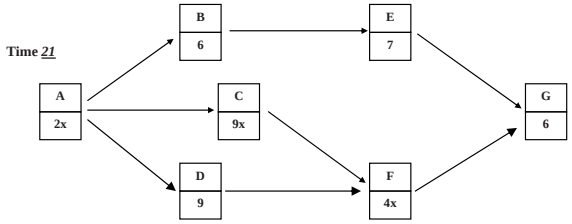




(c)



(d)



(e)

Untuk biaya langsungnya harus dihitung slope-nya pada setiap aktivitas (lihat tabel hal. 63). Dalam tabel terlihat bahwa aktivitas dengan perpendekan satu hari harus dimulai dari aktivitas dengan nilai slope terendah di jalur kritisnya (dalam contoh ini aktivitas A). Lihat gambar (a) ke (b). Biaya perpendekan untuk 1 hari yang terjadi

adalah: $ 450 + $ 20 + $ 350 = $ 820. "x" pada durasi aktivitas A menyatakan, bahwa aktivitas itu tidak bisa diperpendek lagi.

Langkah berikutnya adalah mereduksi waktu aktivitas D (slope terkecil dari sisa aktivitas pada jalur kritisnya). Lihat dari gambar (b) ke (c) untuk durasi 23 hari. Total biaya langsung saat ini menjadi $ 450 + $ 20 + $ 25 = $ 495. Saat ini terdapat dua jalur kritis A,C,F,G dan A,D,F,G.

Untuk mereduksi menjadi 22 hari, aktivitas F menjadi pilihan (karena berada pada kedua jalur kritis, pilih "merge activity"-nya). Total biaya langsung untuk ini adalah $ 495 + $ 30 = $ 525. Aktivitas F tidak dapat direduksi lagi. Lihat gambar (c) ke (d)

Setelah direduksi menjadi 22 hari, semua jalur menjadi kritis. Yang harus dilakukan sekarang adalah mereduksi sejumlah 1 hari semua aktivitas pada jalur kritis tersebut dengan biaya yang terendah, yaitu aktivitas C,D dan E dengan biaya masing-masing $ 30, $ 25 dan $ 30. Total biaya langsung saat ini menjadi $ 525 + $ 85 = $ 610. Lihat gambar (d) ke (e) di atas.

Dalam contoh ini dianggap bahwa untuk setiap unit waktu perpendekan (per hari), **biaya tak langsung**nya mengalami penurunan $50. Nilai biaya tak langsung mula-mula (25 hari) adalah $400.

Dengan membuat grafik biaya-waktu (diberikan sebagai latihan bagi pembaca) dapat dilihat bahwa reduksi proyek dari 25 hari ke 22 hari adalah yang paling optimal. Sehingga pilihan jatuh untuk crashing proyek selama 3 hari (dari 25 menjadi 22 hari). Lihat hasil perbandingan durasi, biaya langsung dan tak langsung, serta totalnya dalam tabel di bawah ini.

Durasi proyek	Biaya langsung	Biaya tak langsung	Biaya total
25	450	400	850
24	470	350	820
23	495	300	795
22	***525***	***250***	***775***
21	610	200	810

Dengan demikian selesailah proses crashing untuk contoh sederhana ini, dengan terpilihnya durasi 22 unit waktu sebagai titik optimum dalam crashing pada jaringan kerja ini.

BAB XIII
ESTIMASI PENGEMBANGAN PERANGKAT LUNAK

Salah satu kriteria penilaian proyek sehingga bisa dikatakan sukses adalah bahwa produknya selesai tepat waktu dan sesuai dengan rencana biayanya. Setelah *objectives*, *goal* dan *requirements fas*e selesai, kemudian fase rencana aktivitas proyek juga telah selesai. Maka langkah krusial berikutnya adalah memperkirakan waktu (dan juga berkaitan erat dengan biaya produksi) penyelesaian proyek. Di dalam bab ini akan dijabarkan secara khusus mengenai cara pengestimasian durasi rekayasa perangkat lunak. Suatu proyek IT, hampir tidak akan pernah terlepas dari pengembangan perangkat lunak (*software coding*) . Pada satu atau lebih aktivitas dalam suatu proyek IT, pasti didapatkan tugas untuk menulis kode komputer, entah itu bersifat sederhana, seperti: *scripting*; ataupun kompleks (contohnya: penulisan kode untuk suatu aplikasi lengkap).

Perangkat lunak sebagai suatu produk yang kompleks dan intangible (tidak kasat mata), memerlukan perlakuan khusus dalam pengestimasiannya karena dalam proses pengembangannya perangkat lunak tidak dapat dinilai secara mekanis ataupun kuantitas. Secara khusus ada bidang ilmu dalam dunia informatika yang mempelajari teknik pengukuran perangkat lunak, dikenal

dengan sebutan *Software Metrics and Quality*. Dalam bidang ilmu ini selain pengukuran perkembangan pada saat perangkat lunak dibuat, juga diukur kinerjanya pada saat telah dipakai dalam lingkungan aplikasi, dikenal dengan istilah *software maturity model*, contohnya adalah CMM (Capability and Maturity Model) dan Bootstrap Model.

Dalam hubungannya dengan manajemen proyek IT, khususnya dalam pengembangan proyek perangkat lunak, di dalam diktat ini akan dibahas secara khusus cara pengestimasian pengembangan perangkat lunak lewat teknik ***function point analysis*** (analisis titik fungsi), lewat metodologi parametris dengan membahas secara tuntas penggunaan COCOMO (*Constructive Cost Model*).

13.1 Kesulitan Estimasi Perangkat Lunak

Selain karena perangkat lunak merupakan produk yang tidak dapat dinilai secara kasat mata dan karena kompleksitasnya, ada beberapa faktor lain yang membuat pengembangan suatu produk perangkat lunak sulit diestimasi baik secara biaya maupun waktunya.

- Keunikan produk perangkat lunak.
  Perangkat lunak atau program biasanya unik dikembangkan untuk suatu permasalahan tertentu. Lain dengan produk rekayasa lainnya, rekayasa perangkat lunak tidak dapat begitu saja menggunakan pengalaman dari masa silam untuk menentukan estimasi biaya ataupun waktu dalam pengimplementasiannya di masa kini.
- Perubahan teknologi.
  Perkembangan dunia informasi sangat cepat, teknologi yang digunakan lima tahun lalu, sudah dianggap usang pada masa kini. Perangkat lunak

yang dibuat dalam bahasa C, dianggap sudah tidak memenuhi syarat untuk turut serta dalam pertukaran informasi melalui Internet saat ini, program misalnya harus diubah mengikuti perkembangan program berorientasi obyek saat ini, dengan menggunakan bahasa Java.

- Perbedaan pengalaman pekerja dalam proyek
  Programmer sebagai unit pekerja yang mengimplementasikan requirements sebuah produk perangkat lunak, memiliki pengalaman yang berbeda-beda dalam menggunakan teknik pemrograman dan teknologi seputar IT. Ini berkaitan erat juga dengan *tacit knowledge,* yaitu pengetahuan intern yang dimiliki pekerja dalam suatu organisasi.

13.2 Pengenalan Rekayasa Perangkat Lunak

Rekayasa Perangkat Lunak (*software engineering*) atau dikenal juga dengan sebutan metodologi pengembangan perangkat lunak (*software development methods*) merupakan suatu cabang ilmu dalam dunia IT yang mempelajari secara khusus teknik-teknik dalam upaya pengembangan perangkat lunak dari mulai fase awal (requirements) hingga fase akhir (evaluasi).

Suatu metodologi menggambarkan fase-fase dan keterkaitan antar fase tersebut dalam proses pengembangan produk, dalam hal ini produk perangkat lunak. Untuk memudahkan melihat keterkaitan antar fase diciptakanlah life-cycles, yaitu gambaran siklus hidup dalam pengembangan produk. Dengan pertolongan life-cycles ini, maka estimasi waktu dan biaya dapat dipecah-pecah dalam setiap fase (ingat juga metode yang serupa digunakan dalam WBS).

13.2.1 Sofware Life-cycles

Secara umum kegunaan dari life-cycles ini adalah:

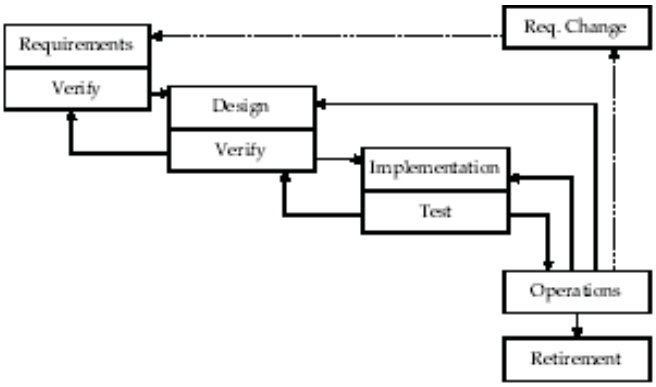
- Membagi proyek dalam fase-fase secara jelas dan menyeluruh;
- Membantu pengontrolan dan pengarahan proyek sesuai dengan *objectives* dan *requirements*;
- Hasil dari setiap fase dalam life cycles dapat digunakan sebagai Milestones dan sebagai input untuk fase selanjutnya.

Macam-macam life-cycles yang sering digunakan adalah:

- **Spiral model;**
- **Waterfall model;**
- Throw-away prototyping model (metode cepat dan kotor, biasa digunakan dalam pemrograman individu);
- Evolutionary prototyping model (digunakan pada proyek-proyek berisiko rendah);
- Incremental / iterative development (Rapid Application Development, iterasi proyek, *client-minded*);
- Automated software synthesis (requirements and specifications tools, masih dalam pengembangan).

Dalam diktat ini akan dibahas *spiral model* dan *waterfall model*.

13.2.2 Waterfall Model (model air terjun)



Karakteristik:

- o *Life cycles* ini masih digunakan secara luas sampai sekarang;
- o Fase dalam proyek terukur dan perkembangan setiap fase dapat diikuti.
- o Pengalaman dari proyek (fase) sebelumnya dapat digunakan sebagai feed-back dalam estimasi.
- o Hasil (bagian) proyek dapat digunakan sbg acuan di proyek mendatang.
- o Setiap fase harus tuntas sebelum dapat melanjutkan proyek ke fase berikutnya atau feed-back ke fase sebelumnya.
- o Perkembangan proyek mudah diikuti pada setiap fasenya.

13.2.3 Spiral model

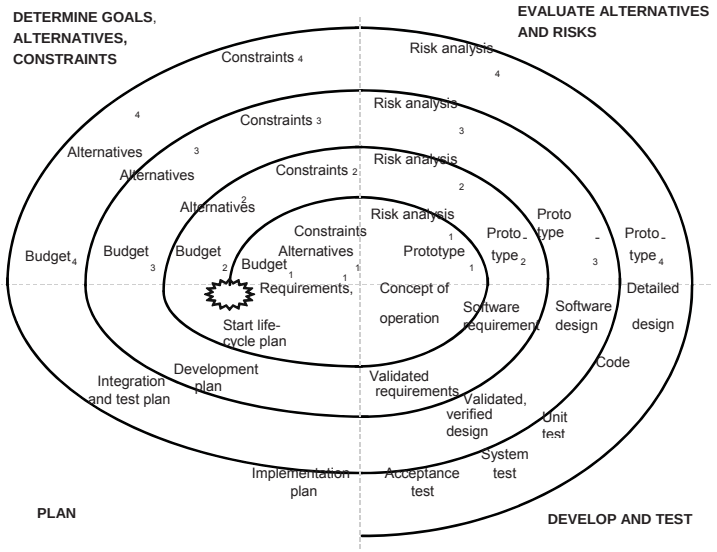
Karakteristik:

- o Menggunakan prinsip iterasi, namun dalam setiap kali iterasi diperhitungkan dengan "manajemen risiko"-nya.
- o Pada setiap siklus (iterasi) dinilai bagaimana status

proyek saat ini, apakah sesuai dengan tujuan (objectives) semula;

- o Mempertimbangkan risiko-risiko apa yang dapat muncul bila diadakan perubahan pada suatu iterasi dan melihat alternatif apa saja yang tersedia dan menilai dampaknya bagi proyek;
- o Tahap iterasi berikutnya harus menitik-beratkan pada penanggulangan risiko-risiko;
- o Setiap iterasi ditutup dengan pengeksekusian rencana.

Singkatnya: desain (rencana); identifikasi tujuan; evaluasi alternatif dan risiko; dan pengembangan (implementasi) dilakukan pada setiap fase. Produk berkembang pada setiap fase. Setiap fase menghasilkan suatu prototype sebagai input bagi fase berikutnya dengan tujuan pada fase terakhir produk menjadi lengkap.



13.3 Analisis domain

Sebuah proyek perangkat lunak beroperasi pada salah satu dari produk domain berikut ini:

- *Data oriented design* (information engineering);
  - Contoh: pengembangan database untuk data pegawai;
- *Function oriented design* (structured analysis);
  - Contoh: pengembangan sistem transaksi online e-commerce;
- *Object Oriented* (OO) methods (gabungan antara data orientasi dengan fungsi orientasi);
  - Contoh: pengembangan pengembangan perangkat lunak untuk sistem keamanan rumah.
- *Formal methods* (evaluasi): untuk menggambarkan atau membuktikan kebenaran jawaban dari suatu problem dalam teori tentang informasi;
  - contoh: penggunaan penggunaan bukti-bukti matematika diskrit (otomata, graphs, perhitungan kombinatorial, dll) untuk membuktikan nilai kebenaran suatu program.

Berdasarkan pada pembagian domain ini, riset terhadap proyek serta estimasi terhadap biaya dan waktu dapat lebih fokus.

13.4 Biaya-biaya Proyek Perangkat Lunak

Dalam Bab VI, telah disinggung sedikit mengenai pengertian biaya langsung dan tak langsung dalam suatu proyek. Dalam bab ini akan dibahas secara khusus, biaya-biaya apa saja yang berperan dalam pengembangan sebuah produk perangkat lunak. Pengenalan akan pos pengeluaran

ini termasuk bagian dari manajemen biaya proyek (*Project Cost Management*).

Biaya-biaya ini meliputi:

- Biaya langsung:
  - Berhubungan **langsung** dengan jalannya proyek.
  - Harga barang-barang baik perangkat keras maupun lunak.
  - Lisensi perangkat lunak.
  - Jam kerja organisasi dan/atau outsourcing.
- Biaya tak langsung:
  - Biaya yang **mendukung** jalannya proyek.
  - Biaya rapat.
  - Material kerja (kertas-kertas, printer, alat tulis, disket, dsb).
  - Biaya tak terduga, misalnya untuk waktu kerja yang melebihi perkiraan.

Seperti telah disinggung dalam bab-bab terdahulu, informasi mengenai biaya dapat dilakukan melalui riset pada awal terjadinya proyek. Hal ini meliputi:

- Kontak dengan IT vendor terpercaya (atau dengan **expert**);
- Informasi proyek sejenis (misalnya dari **database perusahaan** atau dari **IT-Vendors** terpercaya);
- Menggunakan teknik-teknik perhitungan finansial (**parametric model**), disusun dari WBS yang telah dibuat:
  - **Total budgeted costs** (alokasi dana untuk implementasi);
  - **Cumulative actual costs** (biaya yang sampai saat ini telah dikerluarkan);

- **Cost variance** (selisih total rencana biaya dgn biaya aktual);

13.5 Manajemen Biaya Proyek Perangkat Lunak

Secara global *project cost management* meliputi aktivitas di bawah ini:

- Perencanaan sumberdaya;
- Estimasi biaya;
- Pembuatan anggaran;
- Kontrol anggaran.

Perencanaan sumberdaya sebagian besar telah terbahas pada bagian *WBS* dan *network planning*. Dalam bab ini akan dibahas kelanjutan dari permasalahan biaya, yaitu estimasi biaya. Untuk pembuatan anggaran tidak akan dibahas dalam kuliah ini, masalah ini dibahas khusus dalam bidang ilmu *financial project management*. Sedangkan kontrol anggaran akan dibahas sebagian kecil saja, yaitu mengenai *earned value, cost performance index* dan *schedule performance index*.

13.5.1 Teknik Umum Estimasi Biaya dan Usaha (*effort*)

1. *Top-down estimating* (analogous estimating): menggunakan informasi dari proyek-proyek sejenis sebelumnya sebagai dasar perhitungan. Penggunaan teknik ini adalah yang paling cepat dan sederhana, namun berisiko tinggi karena kurang akurat. Disebabkan karena perhitungannya berkaitan erat dengan keunikan setiap proyek yang berbeda dan kemampuan estimasi dari pelaksananya (*expert* atau manajer proyek) yang juga berbeda-beda. Salah satu contoh penggunaan teknik ini adalah dengan

penggunaan *case-based reasoning*.

2. *Paremetric modelling*: menggunakan data-data langsung dari proyek sebagai input untuk diolah dengan menggunakan model matematis. Keakuratan teknik ini bergantung pada proses pembuatan model matematis yang digunakan, keakuratan data proyek dan model dapat digunakan secara general untuk proyek kompleks maupun sederhana. Contoh model matematis: estimasi jadwal dengan metode PERT, estimasi jumlah *function point* dengan COCOMO model.
3. *Bottom-up estimating*: menggunakan perancangan aktivitas (WBS) yang sudah dibuat. Setiap aktivitas dinilai sendiri-sendiri kemudian ditotalkan untuk keseluruhan proyek. Keakuratan teknik ini bergantung pada besarnya aktivitas proyek yang diestimasi. Semakin kecil aktivitasnya akan meningkatkan keakuratan, tapi akan memperbesar estimasi biayanya.
4. *Tools* komputer: menggunakan misalnya software manajemen proyek, seperti MS-Project dalam melakukan estimasi.

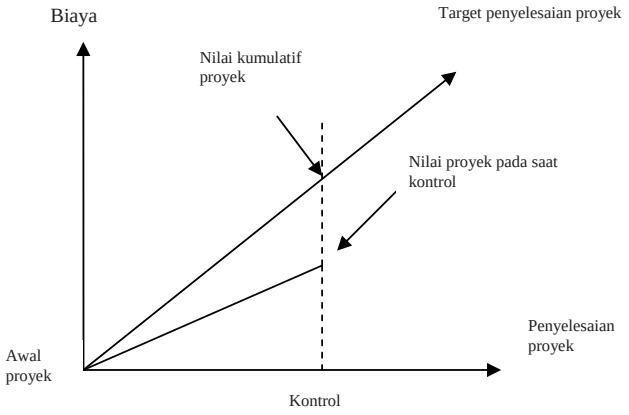
13.5.2 Kontrol Biaya

Setelah proyek berjalan, semua *cash flow* (pengeluaran dan pemasukan uang) harus dikontrol sehingga tetap berada pada batasan yang direncanakan melalui estimasi biaya. Ada pun data-data yang diperoleh dari kontrol biaya dapat pula dijadikan masukan bila ada perubahan rencana pada aktivitas kerja proyek (sebagai informasi historis).

Untuk mengadakan pengontrolan terhadap pemasukan dan pengeluaran ini ada beberapa istilah yang wajib untuk dipahami:

- **Total budgeted costs**, yaitu jumlah biaya total yang dianggarkan, terutama untuk fase implementasi;
- **Cumulative actual costs**, yaitu jumlah biaya yang dikeluarkan dalam proyek pada saat kontrol dilakukan;
- **Cost variance**, yaitu perbedaan jumlah pengeluaran yang terjadi pada saat kontrol dengan yang dianggarkan;
- **Earned value**, yaitu jumlah uang yang dihitung dari nilai pekerjaan sampai pada saat kontrol dilakukan. Persentase pekerjaan yang terselesaikan pada saat kontrol akan membantu manajer proyek untuk menghitung nilai pekerjaan pada suatu aktivitas ataupun proyek keseluruhan.
- **Qualitative value**, yaitu hal-hal yang tidak dapat dinilai dengan uang atau angka, seperti kepuasan pelanggan, perbaikan proses dalam proyek.

Bila digambarkan dalam sebuah grafik antara hubungan waktu proyek dan biaya, maka mekanisme pengontrolan biaya proyek dapat dijelaskan sebagai berikut:



Apabila nilai dari pekerjaan yang telah dijalani (***actual cost earned value***), yaitu biaya sesungguhnya dari suatu aktivitas pada saat kontrol dilakukan, telah dihitung; maka dengan menggunakan informasi sejenis dari aktivitas-aktivitas lainnya, kinerja keseluruhan proyek dapat pula dihitung.

Besaran untuk menghitung kinerja aktivitas ini dikenal dengan istilah Indeks Biaya dan Kinerja (***Cost Performance Index***), yaitu perbandingan nilai pekerjaan yang diperoleh dengan total pengeluaran yang terjadi pada saat suatu kontrol dalam proyek diadakan.

Besaran lainnya yang dapat dihitung dan berperan dalam menentukan apakah kinerja proyek harus ditingkatkan atau sudah memadai adalah Indeks Penyelesaian Kinerja (***To-complete Performance Index***).

Adapun rumus yang digunakan dalam perhitungan-perhitungan di atas adalah:

Earned value = actual % compl. work * budgeted cost of work to date;

Cost Performance Index = earned value / total cost of work to date;

To-Complete Performance Index = (Budget – Budgeted Cost Work Performed) / (Estimated Cost at Completion – Actual Cost Work Performed);

Contoh penggunaan perhitungan:

VARIABEL	ATRIBUT	KETERANGAN
Biaya kerja per jam	$ 130	Biaya yang dialokasikan untuk setiap jam kerja.
Aktivitas	Upgrade server	Jenis pekerjaan atau aktivitas terkait.
Alokasi jam	50	Alokasi jam kerja, sesuai rencana pada WBS dan/atau AON.
Anggaran biaya total	$ 6500 (= 50 * $ 130)	Jumlah anggaran yang dialokasikan untuk aktivitas terkait.
Jam kerja pada saat kontrol	10	Jumlah jam kerja sampai pada saat kontrol, seperti yang dilaporkan tim kerja.
Target persentase pekerjaan	20% (= 10 / 50 * 100%)	Target persentase pekerjaan yang seharusnya selesai.
Persentase aktual	13%	Persentase pekerjaan yang selesai seperti dilaporkan tim kerja.
Variansi	7%	Selisih antara target dengan kenyataan.
Target nilai pekerjaan yang seharusnya tercapai	$ 1300 (= 20% * $ 6500)	Biaya yang dikeluarkan untuk aktivitas pada saat kontrol.
Nilai pekerjaan aktual	$ 845 (= 13% * $ 6500)	Nilai pekerjaan pada kenyataannya.

Sebagai usaha tambahan untuk mengetahui kemajuan proyek secara keseluruhan, maka perhitungan-perhitungan sejenis seperti di atas untuk setiap aktivitas dalam proyek harus dilakukan.

VARIABEL	JUMLAH	KETERANGAN
Anggaran untuk penyelesaian	$ 209,300	Anggaran proyek keseluruhan sampai penyelesaian.
Biaya kumulatif	$ 20,875	Biaya yang dikeluarkan sampai pada saat kontrol.
Target nilai	$ 20,875	Nilai pekerjaan yang diharapkan pada saat kontrol. Sebanding besarnya dengan biaya kumulatif.
Nilai kumulatif aktual	$ 18,887	Dihitung dengan cara menjumlahkan keseluruhan nilai pekerjaan aktual, dari setiap aktivitas dalam proyek.
Variansi	- $ 1988	Selisih antara target nilai dengan kenyataan. Semakin dekat variansi dengan nilai 0, maka proyek semakin berjalan sesuai rencana.

Dari tabel di atas dapat langsung dihitung bahwa ***Cost Performance Index*** (menunjukkan berapa dekat pelaksanaan proyek mendekati total biaya yang telah dikeluarkan) pada saat kontrol diadakan, adalah:

CPI = 18887 / 20875 = 0.9… = 90%

Dengan perhitungan CPI dapat dilihat bahwa proyek mengalami sedikit perlambatan kinerja (perbandingan lebih kecil dari 100%), dan harus diupayakan perbaikan kinerja dan efektivitas tim kerja.

Selain CPI, dapat pula dihitung nilai SPI-nya (***Scheduled Performance Index***), yaitu rasio antara keadaan di lapangan (nilai pekerjaan yang diperoleh) dengan rencana kerja anggaran mula-mula sampai dengan saat kontrol. Caranya adalah dengan menghitung:

SPI = BCWP / BCWS

BCWP adalah ***Budgeted Cost Work Performed*** (Nilai biaya yang dikeluarkan sampai dengan saat kontrol);

BCWS adalah ***Budgeted Cost Work Scheduled*** (Anggaran biaya total yang direncanakan sampai dengan saat kontrol)

Terakhir adalah menghitung besaran ***To-Complete Performance Index*** (menunjukkan prediksi berapa besar usaha yang diperlukan supaya proyek tetap berjalan sesuai rencana). Apabila nilai besaran ini lebih besar dari 1 maka kinerja tim harus ditingkatkan. Bila nilai besaran lebih kecil dari 1 atau sama dengan 1, berarti kinerja tim sesuai dengan target.

Ambil contoh: anggaran keseluruhan proyek adalah $ 75,000; anggaran awal dalam rencana $ 5,000; biaya yang diperkirakan akan dihabiskan pada akhir proyek adalah $ 75,000; dan biaya yang telah dikeluarkan pada saat kontrol $ 7,500.

Maka:

TCPI = (Budget – BCWP) / (EAC – ACWP), di mana

Budget = Anggaran awal untuk proyek;
BCWP = Budgeted Cost Work Performed (anggaran biaya saat kontrol);
EAC = Espected cost At Completion (estimasi biaya pada saat selesai)
Besarnya EAC mungkin saja berbeda dengan anggaran awal setelah adanya penyesuaian dalam perhitungan;
ACWP = Actual Cost Work Performed (biaya aktual pada saat kontrol);

TCPI = (75000 – 5000) / (75000 – 7500) = 1,037.

Dengan demikian kinerja kerja harus ditingkatkan 0,037 kali.

13.6 Teknik Estimasi Usaha Pengembangan Perangkat Lunak

Pada sub-bab 7.6.1 telah dijelaskan teknik umum untuk menghitung estimasi biaya dan usaha. Secara khusus dalam pengembangan suatu produk (tidak hanya untuk produk-produk IT), perlu diambil asumsi-asumsi sebagai berikut:

- Apabila target produk terlalu rendah maka kinerja tim akan menurun. Ini dikenal sebagai ***Parkinson's law***, yang menuliskan bahwa ***"Work expands to fill the time available"***.
- Usaha yang diperlukan untuk mengimplementasikan sebuah proyek akan semakin besar seiring dengan bertambahnya anggota dalam tim kerja. Hal ini terkait dengan usaha yang diperlukan untuk koordinasi dan komunikasi. Asumsi seperti ini dikenal sebagai ***Brooks' law***, yang menuliskan bahwa ***"Putting more people on a late job makes it later"***.

Pengambilan asumsi ini untuk mendasari keputusan dalam menentukan seberapa jauh kita dapat mengestimasi jumlah pekerja ataupun biaya dalam pelaksanaan proyek. Penggunaan estimasi jumlah FP dalam model perhitungan perangkat lunak, dimulai pada akhir tahun 70-an dengan ide awal dari A.J. Albrecht, yang berkeinginan untuk menilai berapa besar suatu perangkat lunak harus dikembangkan dalam suatu proyek software dan bagaimana produktivitas para programmer yang bekerja di dalam proyek tersebut.

Estimasi FP menggunakan entitas fungsional dan entitas logikal dalam suatu sistem perangkat lunak. Entitas ini meliputi input, output, inquiries, serta keterkaitan suatu program dengan program lainnya. Dengan kata lain FP menunjukkan hubungan dan keterkaitan antar prosedur, fungsi dan lingkungan pendukung dalam suatu perangkat lunak. Contoh model parametrus untuk estimasi FP, antara lain: metode Albrecht (sekarang dikenal dengan metode IFPUG), metode Mark II dan COCOMO model.

Di dalam diktat ini akan dibahas secara khusus penghitungan estimasi biaya dan usaha pengembangan perangkat lunak dengan **COCOMO** (Constructive Cost Model = Model Konstruksi Biaya), yang berbasis pada model matematis dengan menghitung estimasi **FP** (**Function Points / titik fungsi**) dan besarnya jumlah kode. FP dapat diartikan sebagai sebuah unit pengukuran dalam pengembangan, pemakaian ataupun perawatan perangkat lunak, dengan cara menggunakan keterkaitan pengukuran domain informasi perangkat lunak serta perkiraan kompleksitasnya.

COCOMO model, yaitu suatu model parametris pengestimasian yang menghitung jumlah FP dalam perencanaan serta pengembangan perangkat lunak, mengenal tiga macam pengimplementasian dalam evolusinya sejak dari awal kejadiannya hingga kini, yaitu:

- **Basic** (COCOMO I 1981)
  - Menghitung dari estimasi jumlah LOC (Lines of Code);
- **Intermediate** (COCOMO II 1999)
  - Menghitung dari besarnya program dan "*cost drivers*" (faktor-faktor yang berpengaruh langsung kepada proyek), seperti: perangkat

keras, personal, dan atribut-atribut proyek lainnya;
    - Mempergunakan data-data historis dari proyek-proyek yang pernah menggunakan COCOMO I, dan terdaftar pengelolaan proyeknya dalam COCOMO database.
- **Advanced**
    - Memperhitungkan semua karakteristik dari "*intermediate*" di atas dan "*cost drivers*" dari setiap fase (analisis, desain, implementasi, dsb) dalam siklus hidup pengembangan perangkat lunak.

13.6.1 Basic COCOMO (COCOMO 81)

Pengenalan Cocomo ini diawali tahun 70-an akhir. Sang pelopor Boehm, melakukan riset dengan mengambil kasus dari 63 proyek perangkat lunak untuk membuat model matematisnya. Model dasar dari model ini adalah persamaan:

effort = C * size^M , di mana

- *effort* adalah usaha yang dibutuhkan selama proyek, diukur dalam person-months;
- *c* dan *M* adalah konstanta-konstanta yang dihasilkan dalam riset Boehm dan tergantung pada penggolongan besarnya proyek perangkat lunak;
- *size* adalah estimasi jumlah baris kode yang dibutuhkan untuk implementasi, dalam satuan KLOC (kilo lines of code);

13.6.2 Konstanta COCOMO

Penggolongan suatu proyek perangkat lunak didasarkan pada sistem aplikasi dimana perangkat lunak tersebut dikembangkan dan lingkungan pendukungnya.

Penggolongan ini terbagi atas:

- ***Organic mode***: digunakan pada proyek-proyek kecil dengan sedikit pekerja dan dikembangkan pada lingkungan yang tidak memerlukan program antar-muka (interface) yang kompleks, contoh: pembuatan situs mandiri untuk perusahaan;
- ***Semi-detached mode***: dalam mode ini produk dikembangkan dalam sistem yang memiliki banyak batasan atau syarat tertentu untuk pemrosesan dalam perangkat keras dan lunak tertentu. Apabila terjadi perubahan pada sistem maka akan menyebabkan biaya produksi akan bertambah tinggi, contoh: transaksi sistem pada database sebuah bank;
- ***Embedded mode***: mode ini merupakan kombinasi antara dua mode di atas dan memiliki karekteristik gabungan antara keduanya. Proyek mode ini dikembangkan ke dalam serangkaian perangkat keras, lunak dan batasan operasional yang ketat, contoh: aplikasi pengontrolan penerbangan pada pesawat terbang.

Ada pun konstanta yang dibutuhkan pada masing-masing mode tersebut adalah (didapatkan dari hasil riset Boehm):

TIPE SISTEM	C*A*	M*A*	C*B*	M*B*
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Dengan demikian rumusan dasar di atas, dapat digunakan untuk perhitungan-perhitungan sebagai berikut:

- (**E**) *effort* = C*A* x (size)^ **M*A***
  (satuan: **ManMonth** dalam COCOMO I (Person Month dalam COCOMO II) = 152 jam kerja);

- (**D**) *duration* = C*B* x E^**M*B***
  (satuan: **Month**);

- ***Productivity*** = size / E (satuan: **KLOC/Man Month**);

- ***Average staffing*** = E / D (satuan: **FTE** = Full Time Employees, yaitu jumlah orang yang bekerja penuh dalam 1 hari kerja ~ 8 jam)

13.6.3 Penggunaan COCOMO

Ada pun langkah-langkah untuk menghitung estimasi dengan menggunakan Basic COCOMO adalah:

1. Menghitung estimasi informasi **nilai total domain,** yaitu informasi mengenai karakter spesifik perangkat lunak yang akan dihasilkan;
2. Menyesuaikan kompleksitas proyek berdasarkan faktor pemberat dan "*cost drivers*" (∑ **F*ij***); kemudian menghitung estimasi jumlah titik fungsi (**Function Points**);
   FP = nilai total domain * [0.65 + 0.01 * ∑ F*ij*];
3. Menghitung estimasi **LOC** (Line of Code). Tekniknya dasarnya sama dengan yang digunakan dalam perhitunga PERT (*three points estimation*), dengan cara;

- o **EV = (Sopt + 4 Sm + Spess) / 6,** dimana: EV berarti *Estimated Value;*
- o **Atau** menghitung **KLOC/ FP** (dari *tabel hasil riset*) berdasar pada bahasa pemrograman yang digunakan dalam implementasi proyek perangkat lunak;

4. Memilih kompleksitas proyek (menentukan **C** dan **M**), dari organic, embedded atau semi-detached model.
5. Menghitung **E** = effort dan **D** = duration, dengan demikian akan menghasilkan estimasi usaha, biaya dan waktu.

13.6.4 Menghitung Nilai Domain

Untuk menghitung karakter spesifik produk perangkat lunak yang akan dihasilkan, digunakan analisis domain sebagai berikut:

Informasi nilai domain	(Simple	Average	Complex)	Jumlah		
Jumlah input pemakai	3	4	6	*	=	
Jumlah output pemakai	4	5	7	*	=	
Jumlah inquiry pemakai	3	4	6	*	=	
Jumlah file	7	10	15	*	=	
Jumlah eksternal interface	5	7	10	*	=	+
					Total nilai domain	

Keterangan:

- o **Input pemakai:** setiap input data dari user yang dipakai untuk menjalankan aplikasi.
- o **Output pemakai:** setiap hasil output dari proses yang ditampilkan kepada user.
- o **Inquiry pemakai:** setiap *on-line* input yang menghasilkan responsi software secara langsung.

- **Jumlah file:** setiap master file yang menjadi bagian dari aplikasi.
- **Eksternal interface:** setiap interface (sarana) eksternal yang menyalurkan informasi dari sistem satu ke sistem lainnya.

13.6.5 Menghitung Faktor Pemberat ("cost drivers")

Setelah total analisis domain selesai dihitung, langkah berikutnya adalah menghitung faktor pemberat, sebagai berikut:

Ada 14 faktor pemberat yang diperhitungkan, dengan masing-masing **berbobot dari 0 sampai dengan 5**, bergantung pada kebutuhan sebuah produk perangkat lunak terhadap masing-masing faktor tersebut, dengan urutan:

(No Influence = tidak berpengaruh**, Incidental** = terkadang dibutuhkan**, Moderate =** dibutuhkan kurang dari rata-rata**, Average =** rata-rata dibutuhkan**, Significant =** dibutuhkan namun tidak harus**, Essential =** harus terimplementasi**)**.

Ke-14 faktor pemberat yang dimaksudkan adalah:

0	1	2	3	4	5

1. Backup dan recovery
2. Komunikasi data
3. Proses terdistribusi
4. Kepentingan performa
5. Keberadaan lingkungan operasi
6. Online data entry
7. Input melalui beberapa tampilan / operasi
8. Peng-*update*-an file master secara online
9. Kompleksitas nilai 'domain' (tahap1) diatas
10. Kompleksitas proses internal aplikasi
11. Perulangan (*reuse*) penggunaan code
12. Ketersediaan rancangan untuk konversi dan instalasi
13. Rancangan untuk pengulangan instalasi di lingkungan yang berbeda
14. Fleksibilitas bagi pemakai

Total F*ij* Σ F*ij* =Σ [(F*i* = 0..5) * (ΣF*j* = 1..14)]

Dari sini dapat dihitung estimasi titik fungsi (**function points**), sebagai berikut:

FP = nilai total domain * [0.65 + 0.01 * $\sum$ F*ij*];

13.6.6 Estimasi LOC

Langkah berikutnya dalam proses estimasi COCOMO adalah menghitung jumlah *Line of Code* (LOC).

Diawali oleh penelitian Boehm, muncul kemudian estimasi jumlah LOC untuk berbagai bahasa pemrograman yang digunakan dalam implementasi proyek perangkat lunak. Estimasi ini tidak bersifat mutlak, karena perhitungannya didapatkan dari data-data historis berbagai proyek perangkat lunak, dan diambil nilai rata-ratanya dengan menggunakan teknik PERT. Data-data ini

bersifat statistis dengan mengandalkan kekuatan distribusi rata-rata (*mean distribution*).

Tabel LOC/FP untuk berbagai jenis bahasa pemrograman dapat dilihat di bawah ini (data dari http://www.engin.umd.umich.edu/CIS/course.des/cis525/js/f00/gamel/cocomo.html, bandingkan juga dengan tabel dari [ALB83, JON91,ROG97]):

Programming Language	LOC/FP (rata-rata)
Bahasa Assembly	320
C	128
COBOL	105
Fortran	105
Pascal	90
Ada	70
Bahasa Berorientasi Obyek	30
Bahasa Generasi Keempat (4GLs), yaitu bahasa yang digunakan spesifik untuk suatu tools, biasa untuk aplikasi database, contoh: PL/SQL dalam Oracle.	20
Generator Kode	15
Spreadsheets	6
Desain Grafis (icons)	4

Jumlah LOC/FP ini harus diubah ke KLOC/FP (Kilo LOC) dalam perhitungan, dengan membaginya dengan 1000 (sesuai dengan satuan dari hasil riset Boehm). Pada akhir perhitungan tahap ini akan dihasilkan *size*, yaitu jumlah baris kode dalam satuan KLOC.

Tahap selanjutnya adalah menggunakan KLOC yang dihasilkan disini untuk mengestimasikan: usaha (effort), waktu (*duration*), dan jumlah pekerja yang dibutuhkan dalam proyek (FTE).

13.6.7 Revisi COCOMO II

Mengingat penggunaan COCOMO I tidak selalu memenuhi syarat karena melihat estimasi dengan data-data statistik. Maka model COCOMO I diperbarui menjadi COCOMO II. Database COCOMO selalu di-update secara berkala untuk memberi informasi kepada pengguna model ini mengenai nilai parameter yang digunakan, seperti LOC/FP, cost drivers, scale factor, konstanta-konstanta dsb.

- Penggunaan persamaan baru untuk perhitungan LOC dan person-months, yaitu dengan menggunakan faktor skala (*scale factor* (*sf*)). *Scale factor* ini akan menggambarkan kemampuan rata-rata anggota tim kerja;
- Membagi proyek dalam tingkatan, seusai dengan pembagian rencana kerja yang ada. Di dalam masing-masing tingkatan ini akan diadakan penilaian yang berbeda, sehingga tingkat kedewasaan perangkat lunak (*software maturity*) dapat terukur. Software maturity akan menilai bagaimana sebuah produk perangkat lunak memenuhi perannya dalam lingkungan aplikasi. Masing-masing tingkatan dihitung dengan sf yang berbeda.

Tingkatan yang ada dalam COCOMO II adalah:

- *Early design* (perancangan awal): pada tingkatan ini dasar-dasar perancangan dari sebuah produk perangkat lunak dibentuk. Contohnya: jenis transaksi apa yang dibutuhkan, berapa cepat responsi transaksi harus terjadi, dsb;
- *Application composition* (komposisi aplikasi): pada tingkatan ini dibuat perancangan kemampuan perangkat lunak (*features*) seperti yang akan terjadi dalam lingkungan aplikasi (*use-case design*). Hasil dari

tingkatan ini adalah sebuah prototipe dari produk atau pada produk-produk sederhana pengembangannya dapat dihentikan pada tingkatan ini ;

- *Post architecture* (arsitektur lanjutan): dalam tingkatan ini sebuah produk software akan mengalami revisi dan terus diperbaiki kinerjanya sehingga dapat memenuhi perannya dalam lingkungan aplikasi.

13.7 Tips Estimasi Biaya, Waktu dan Penjadualan dalam Proyek

Sebagai rangkuman dari pembahasan tentang estimasi waktu, biaya, rencana kerja (AON) dan penjadwalan, perlu diperhatikan *point-point* di bawah ini:

- Tidak terburu-buru.
- Gunakan dokumentasi proyek-proyek sebelumnya.
- Gunakan estimasi langsung dari orang yang akan mengerjakan (tim kerja);
- Gunakan *software tools* untuk estimasi (sbg pembanding).
- Jangan hanya menggunakan estimasi dari satu orang saja, untuk akurasinya, gunakan *second opinion*.
- Gunakan prosedur estimasi standar (parameter matematis, statistis).
- Evaluasi hasil estimasi (rencana proyek) dan kenyataan yang terjadi di lapangan pada setiap fase dalam proyek.

BAB XIV
ORGANISASI TIM KERJA PROYEK

Seorang manajer proyek memikul tanggung jawab yang besar atas jalannya proyek. Selain memegang peranan yang besar mulai dari tahap pendefinisian proyek sampai dengan perencanaan dan estimasi waktu serta biaya, peranan besar lainnya – bahkan boleh dikatakan terbesar, karena berhubungan langsung dengan faktor manusia – adalah bagaimana mencari, menyusun, mengalokasikan. mengkoordinir serta mengontrol perkembangan anggota tim kerjanya. Di dalam hal inilah sebenarnya peran serta manajer proyek akan sangat terasa.

Dalam perkembangan dan jalannya proyek, manajer proyek seringkali berhadapan dengan berbagai macam problema, antara lain: motivasi, komunikasi, relasi dan tanggung jawab dari masing-masing anggota tim. Jika manajer proyek memiliki keahlian dan mampu mangatasi problem-problem organisasi, maka tingkat keberhasilan proyek akan menjadi lebih besar. Dukungan peralatan serta teknologi modern, pendanaan yang besar ataupun keterlibatan konsultan terpercaya; tidak akan terasa dampak langsungnya bagi pelaksanaan proyek apabila tidak dilaksanakan oleh organisasi kerja yang mapan.

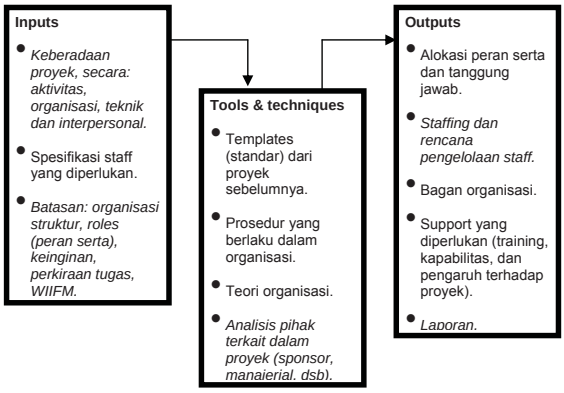
Konsep kerja yang dibutuhkan untuk menunjang peran serta aktif manajer proyek dalam kaitannya dengan keberlangsungan proyek dikenal dengan istilah *Project Human Resource Management*.

Secara global konsep kerja dalam manajemen sumber daya manusia (SDM) ini, terbagi atas: rencana organisasi; pencarian dan penerimaan anggota tim kerja (*staffing*); dan pada akhirnya pengembangan efisiensi dan efektivitas (kinerja) organisasi.

14.1 Alur dan Konsep Kerja Manajemen SDM

Dalam manajemen SDM, seorang manajer proyek perlu memikirkan langkah dan strategi yang tepat untuk menjalankan konsep kerja manajemen SDM. Konsep kerja sebagaimana tersebut pada bagian pendahuluan, sangat berkaitan satu dengan lainnya, bahkan juga terkait dengan proses lain dalam manajemen proyek. Contohnya adalah: dalam menentukan jumlah serta kualifikasi anggota tim kerja proyek sangat bergantung pada dana dan waktu yang tersedia bagi keseluruhan proyek.

Bagaimana konsep kerja manajemen SDM dapat diterapkan di dalam proyek dapat dilihat dalam rangkaian alur kerja di bawah ini:



Melalui gambaran ini terlihat bahwa dengan dimulainya keberadaan sebuah proyek, sudah

direncanakan pula bagaimana dan siapa-siapa saja yang layak diikutsertakan dalam proyek, dikenal dengan istilah rencana organisasi (*organizational planning*). Perlu menjadi pertimbangan pula bahwa jika proyeknya besar, rencana awal tentang organisasi harus disusun secara cermat dengan terlebih dahulu menyusun WBS untuk aktivitas proyek secara keseluruhan, sehingga lebih memungkinkan untuk membentuk bagan organisasi yang seimbang. Dari rencana ini akan terbentuk suatu organisasi kerja yang secara bersama-sama wajib berperan dalam proyek untuk menjamin keberhasilan proyek.

Dalam pembentukan organisasi proyek, diperlukan: pengidentifikasian, pendokumentasi-an serta pengalokasian peran (*rol*) SDM dalam proyek, tanggung jawab, dan saling keterkaitan peran peserta organisasi tersebut. Peserta organisasi proyek dapat diambil dari internal dalam perusahaan ataupun eksternal seperti dari sub-kontraktor dan tenaga bantuan. Adakalanya peserta internal langsung diasosiakan dengan fungsi-fungsi tertentu yang dibutuhkan dalam proyek, seperti: teknis (*engineering*), marketing ataupun akunting.

Dalam kebanyakan proyek, perencanaan global organisasi selesai pada tahap awal proyek (dalam *project charter*). Meskipun demikian untuk meningkatkan kinerja tim atau organisasi proyek secara keseluruhan, rencana organisasi ini perlu ditelaah ulang pada setiap fase, untuk menilai perbaikan apa yang dibutuhkan.

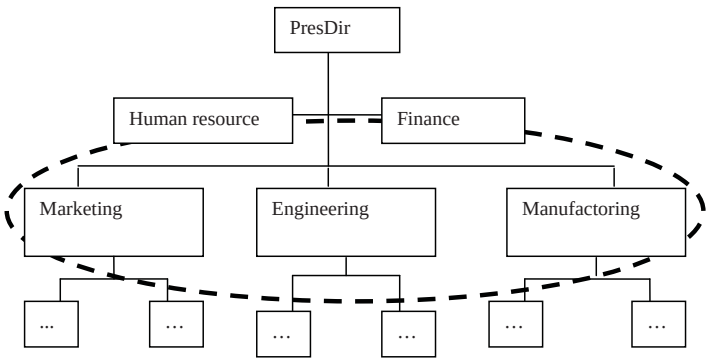
14.1.1 Hubungan antar-muka Aktor proyek

Di dalam sebuah proyek akan terjadi komunikasi antaraktor. Secara umum aktor dan peran sertanya dibagi ke dalam empat bagian besar, yaitu:

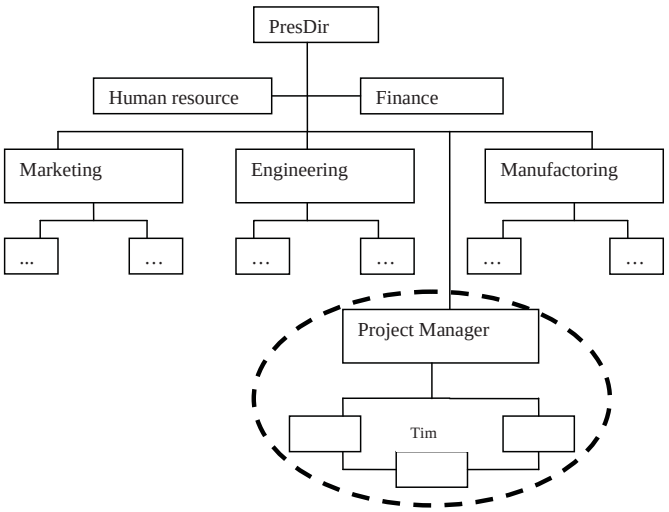
- Hubungan antar **aktivitas**. Dengan terbentuknya suatu proyek, tujuan dari proyek perlu diterjemahkan

dalam suatu rangkaian aktivitas. Perencanaan yang matang untuk aktivitas-aktivitas ini akan sangat membantu terbentuknya komunikasi yang efektif selama masa durasi proyek.

- Hubungan **organisasi internal** instansi terkait dimana proyek akan dilaksanakan. Di sini, akan terjadi komunikasi dan peran serta (formal dan informal) antar berbagai unit kerja yang berbeda. Hubungan ini dapat bersifat kompleks ataupun sederhana. Sebagai contoh dalam sebuah proyek telekomunikasi yang kompleks, dibutuhkan hubungan antara berbagai sub-kontraktor, sedangkan di dalam sebuah pengembangan web-site untuk sebuah perusahaan dibutuhkan hubungan antara tim pengembang, pemakai dan pihak penilai. Biasanya hubungan organisasi internal ini diperjelas dengan bagan organisasi suatu instansi, dan tim kerja proyek internal dimasukkan sebagai bagian dari bagan organisasi tersebut.;
  - Secara umum ada tiga jenis pembagian hubungan organisasi internal, yaitu:
    - Sebagai bagian dari organisasi fungsional;



- Sebagai tim kerja proyek secara khusus (biasanya hal ini terjadi dalam pelaksanaan kerja berbasis kontrak / *outsourcing*);



- Pengaturan secara matrix, yaitu alokasi tim kerja dalam garis fungsional dengan sentralisasi pengaturan proyek-proyek yang ada. Matrix yang terbentuk dapat digolongkan ke dalam tiga bagian, yaitu:
  - Weak matrix;
  - Balanced matrix;
  - Strong matrix.

- Hubungan **teknis**. Pada hubungan ini ditekankan hubungan antara berbagai disiplin teknik yang terkait dalam proyek Sebagai contoh: dalam upgrade sebuah server, perlu diperhatikan masalah teknis dalam rangkaian perangkat kerasnya (kabel, server, rack, jenis server, dsb) dengan pengembangan perangkat lunaknya (sistem operasi jaringan, firewall, dsb).

- Hubungan **interpersonal**. Dalam hal ini, hubungan terjadi antaranggota tim (baik secara individu ataupun unit kerja) dalam proyek.

14.1.2 Batasan Lingkup Kerja

Selain hubungan antarkomponen di atas, batasan lingkup kerja merupakan faktor yang penting sebelum organisasi proyek dapat disusun. Batasan lingkup kerja dapat menjadi hambatan dalam mengatur hubungan dan komunikasi dalam lingkungan proyek. Ada suatu slogan yang berbunyi ***"What's in it for me (WIIFM)"*** , yang secara tegas menggambarkan bahwa setiap proyek memiliki batasan, terutama apabila dilihat secara organisasi. Tidak bisa disangkal bahwa dalam proyek, setiap individu yang terlibat memiliki kepentingan, baik secara terang-terangan ataupun terselubung.

Secara umum, faktor-faktor yang dapat membatasi komunikasi dan hubungan kerja adalah:

o Struktur organisasi dalam instansi di proyek akan dilaksanakan;

Di sini, perlu diperhatikan bagaimana peran seorang manajer proyek sebenarnya. Apakah hanya sebagai pelaksana tanpa dukungan untuk mengambil keputusan atau seorang manajer proyek berhak juga mengambil keputusan demi kelancaran proyek.

o Persetujuan tawar-menawar dalam tim kerja;

Di sini, tersirat bagaimana posisi seorang anggota tim. Setiap anggota tim memiliki tugas-tugas yang harus dijalankan, dan ini harus dinyatakan secara jelas, sehingga tidak terjadi kesalahpahaman ketika proyek berjalan. Pada dasarnya setiap anggota tim adalah bagian dari stakeholders, yaitu mereka yang memiliki kepentingan, tanggung jawab dan ambil

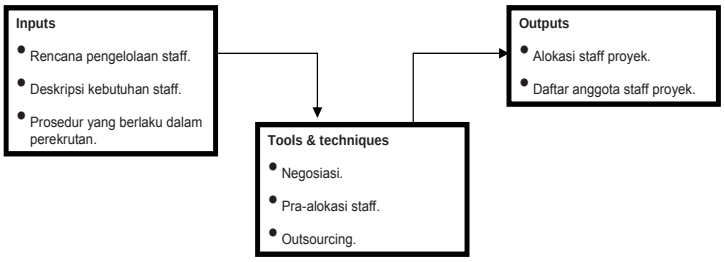
bagian secara aktif di dalam proyek .

o Keinginan dari tim manajemen atas;
Tim manajemen atas sebagai penanggung jawab utama dalam sebuah proyek, tidak akan mengambil risiko dengan tindakan coba-coba. Keberhasilan suatu proyek di masa lalu, akan menjadi tolak ukur bagi mereka. Hal ini akan menurunkan fleksibilitas dalam pengelolaan tim kerja, karena manajemen atas secara tegas menyatakan bahwa organisasi proyek harus disusun berdasarkan keinginan mereka.

o Keahlian dan alokasi SDM;
Bagaimana sebuah proyek diorganisasikan sangat bergantung kepada keahlian dan kapasitas individu yang menjadi bagian tim kerja.

14.2 Rekrutmen SDM (staffing)

Melalui proses perekrutan, tenaga kerja (secara individu maupun tim) yang dibutuhkan di dalam proyek akan ditentukan. Di dalam kebanyakan proyek sangat sukar untuk membentuk satu tim yang terbaik (*the dream team*). Tanggung jawab harus dipikul bersama oleh mereka yang terlibat di dalam proyek untuk meyakinkan bahwa sumberdaya yang tersedia mampu memenuhi persyaratan proyek.



Gambar di atas menunjukkan apa-apa saja yang dibutuhkan dalam proses penerimaan SDM.

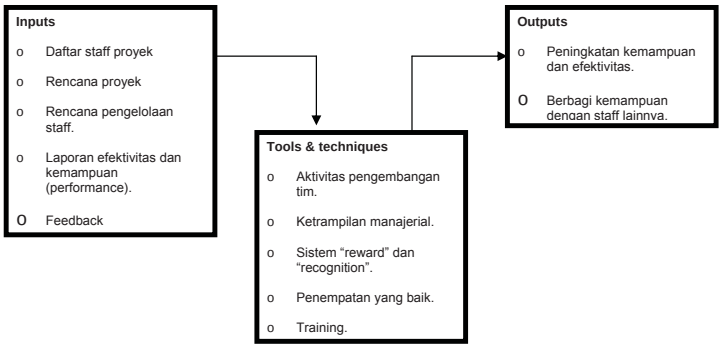
Setiap organisasi memiliki prosedur dalam proses rekrutmen. Selain prosedur yang berlaku, harus diperhatikan juga spesifikasi akan kebutuhan terhadap sumberdaya yang dimaksud. Apakah seseorang kompeten untuk menempati suatu pos dalam organisasi kerja atau tidak adalah dasar pertanyaan yang harus dijawab. Seorang manajer proyek , kadangkala dibantu juga oleh tim manajemen atas dari instansi dimana proyek akan dilaksanakan, harus mampu menjawab pertanyaan yang mendasar ini. Selain itu perlu dilakukan penilaian terhadap masing-masing calon staf dengan mengacu kepada: pengalaman tenaga kerja yang bersangkutan, niat dan ketertarikan akan proyek (*personal interest*), karakter tenaga tersebut cocok atau tidak dengan organisasi dan kesediaan serta kemampuan untuk berperan aktif dalam proyek.

Setelah latar belakang dari calon ditelaah, akan dilakukan suatu negosiasi dengan calon yang dinilai cocok. Pada tahap ini harus jelas, apa-apa saja yang diharapkan dari si tenaga kerja, dan apa yang akan didapatkannya. Apabila calon yang dibutuhkan tidak ada dalam lingkungan internal organisasi, maka dapat dilakukan *outsourcing*, yaitu proses pencarian tenaga atau unit kerja atau sub-kontraktor dari luar organisasi. Bila keseluruhan pos dalam organisasi telah terisi maka dapat dibentuk suatu bagan organisasi yang lengkap dengan sumberdaya manusia, peran serta, alokasi waktu dan tanggung jawabnya.

14.3 Pengembangan Tim Kerja

Untuk meningkatkan kinerja organisasi proyek dan hasil kerjanya, terutama untuk proyek-proyek yang

besar, durasi lama dan kompleks, dibutuhkan suatu sistem pengembangan tim kerja (*team development*).



Bagan di atas menunjukkan proses yang dibutuhkan dalam usaha meningkatkan kinerja tim. Kesempatan untuk pengembangan diri bagi setiap individu dalam tim kerja, tidak hanya dalam bidang teknis, tetapi juga mencakup kemampuan manajerial yang sangat dibutuhkan, mengingat dalam suatu tim, sangat diperlukan komunikasi antar anggotanya.

Pelaksanaan pengembangan tim dapat dilakukan secara berkala, misalnya dilakukan pada setiap fase dalam proyek. Pada setiap akhir fase, dilakukan penilaian terhadap kinerja per individu. Dengan berdasar pada penilaian ini, dapat dilakukan:

- Sistem *reward* (penghargaan), dapat berupa materi maupun non-materi;
- Training untuk menambah ketrampilan (manajerial maupun teknis);
- Rotasi alokasi tenaga kerja, untuk mengoptimalkan kinerja, misalnya: pada awal proyek seorang programmer dengan keahlian utama visual basic, ditempatkan pada pengembangan *database Oracle*, maka hasilnya tidak akan baik. Sedangkan pada tim

kerja lain ada seorang dengan keahlian utama SQL, oleh karena itu dapat dilakukan tukar posisi antara kedua tim tersebut.

- Untuk meningkatkan kerja sama serta kekerabatan antara anggota tim, dapat juga dilakukan kegiatan-kegiatan bersama di luar proyek yang menunjang kerja sama, seperti misalnya pertandingan biljart di luar jam kerja proyek antar tim programmer dengan tim administrasi ataupun acara liburan bersama.

14.4 Organisasi Kerja IT

Dalam konteks proyek-proyek IT, pembagian kerja biasanya dilakukan dengan penggolongan sebagai berikut:

14.4.1 Fungsi Struktural

Pembagian fungsi secara struktural biasanya meliputi:

- Manajer proyek.
- *System Analyst*.
- *System Designer*.
- *System Developer* (Programmer).
- Network Administrator.

Ada kalanya seorang pekerja dapat ditempatkan pada satu atau lebih fungsi, contohnya: seorang manajer proyek juga merangkap sebagai system analyst. Namun tentu saja dengan perhitungan bahwa rangkapnya tugas tidak akan mempengaruhi kinerja tim dan proyek secara keseleruhan.

14.4.2 Fungsi Keahlian Teknis

Pembagian fungsi menurut keahlian biasanya meliputi:

- *General tools*: spreadsheets, editor, MS-Project, dsb;
- *Programming language*: java, C/C++, VB, HTML, dsb.
- *Application*: statistical programs, macromedia, dsb.
- *Operating system*: Windows (9x / 2000/ NT / XP), Linux, Unix, Mac.

14.4.3 Fungsi Manajerial

Ditinjau dari sudut manajerial, dalam proyek IT secara umum terdapat pembagian sebagai berikut:

- Dokumentasi historis produk (*documentation*);
- Pusat informasi (*information centre*);
- Laporan dan presentasi (*rapport and presentation*).

14.5 Matriks Alokasi

Setelah masing-masing aktivitas atau fase mendapat alokasi sumberdayanya, akhirnya dapat disusun suatu matriks yang menunjukkan hubungan antara aktivitas dengan sumberdaya. Matriks ini dikenal dengan sebutan matriks alokasi.

Contoh:

Fase proyek / Anggota Tim	A	B	C	D	...
Requirements	P	R	A	P	
Functional	A	P	A	I	
Design	A	A	P	I	
Development	I	R	S	A	
Testing	S	I	P	P	

Legenda:

P = Participant; **A** = Advices; **R** = Review required; **I** = Input required; **S** = Sign-off required.

Dari contoh di atas pada fase *requirements*: si A dan D aktif berperan serta dalam penyusunannya, si B bertindak sebagai seorang reviewer (penilai) dan si C sebagai penasihat umum.

14.6 Komunikasi Tim Proyek

Dalam sub-sub bab terdahulu telah disinggung tentang pentingya komunikasi antar anggota tim kerja, individu , stakeholders, dan tim manajer atas dalam proyek. Hal ini dimaksudkan untuk penyebaran informasi mengenai proyek. Dengan penyebaran informasi ini diharapkan kinerja proyek akan meningkat karena mendapat masukan-masukan baru dari berbagai pihak yang terlibat dalam proyek.

Dalam komunikasi dibutuhkan:

- Kemampuan berkomunikasi (secara lisan dan tulisan, internal, eksternal, formal, informal, vertikal dan horizontal);
- Penyediaan informasi (melalui manual dokumentasi / paper works, elektronik files, software proyek manajemen);
- Sarana distribusi (melalui jaringan komputer, fax, email, database, rapat).

Melalui informasi dan komunikasi ini dapat ditelaah bagaimana status proyek sebenarnya serta perbaikan-perbaikan apa saja yang diperlukan. Informasi yang diperoleh ini dapat digunakan sebagai informasi historis pada proyek-proyek lainnya ataupun sebagai informasi historis untuk fase-fase selanjutnya dalam proyek.

14.7 Laporan

Laporan adalah suatu sarana dokumentasi untuk menginformasikan status suatu pekerjaan. Sejauh mana perkembangan proyek sampai suatu masa dalam proyek dapat dituangkan dalam bentuk laporan. Laporan ada yang bersifat formal ataupun non-formal. Selain melalui laporan, pengontrolan status dapat juga dilakukan melalui rapat atau tatap muka secara berkala.

Melihat fungsinya yang sangat penting ini, maka dalam sebuah proyek, kehadiran laporan yang berkualitas (formal, lengkap, mudah dimengerti) sangat penting. Bahkan ada kalanya suatu laporan dapat dianggap sebagai suatu milestone dalam proyek, contohnya laporan tentang *feasibility plan*. Seperti halnya pertemuan tatap muka, sangat baik apabila dipertahankan dokumen laporan fokus kepada masalah dan pemecahannya serta status proyek. Tidak perlu bertele-tele, namun harus membentuk satu kesatuan cerita yang masuk akal serta dapat diterima oleh semua bagian yang turut serta dalam proyek.

14.8 Alokasi Sumberdaya nonmanusia

Selain berhubungan dengan manusia, sebuah proyek juga berhubungan dengan pemakaian sumberdaya non manusia, seperti: peralatan komputer, ruang kerja, laboratorium, sarana komunikasi, dan lain sebagainya.

Pengalokasian sumberdaya ini harus diperhitungkan secara cermat supaya tidak bentrok antara aktivitas yang satu dengan lainnya atau pemakaian sumberdaya antara beberapa proyek yang berjalan paralel. Seringkali jadwal yang telah direncanakan menjadi tertunda karena adanya bentrok sumberdaya.

Suatu cara untuk mengelola sumberdaya dalam suatu proyek adalah menyesuaikan pemakaian sumberdaya di dalam aktivitas-aktivitas yang telah direncanakan sebelumnya melalui diagram jaringan kerja proyek. Metode yang sering digunakan adalah dengan upaya mencegah perlambatan dalam proyek melalui cara *heuristics*. Lewat *heuristics* akan disusun skala prioritas aktivitas mana yang boleh menggunakan suatu sumberdaya bila ada pemakaian yang bentrok.

Secara berurutan skala prioritas dalam pemilihan optimalisasi aktivitas-sumberdaya adalah, ini dikenal juga dengan sebutan "*priority rules on resources*":

- Waktu renggang (*slack*) terkecil;
- Durasi tercepat;
- Aktivitas dengan nomor identifikasi terkecil (fase awal proyek).

Aturan ini digunakan apabila dalam suatu waktu tertentu ada sebuah sumberdaya yang diperlukan melebihi kapasitas dan jumlahnya di dalam aktivitas-aktivitas proyek.

Setelah alokasi sumberdaya optimal, dapat dilakukan pemetaan sumberdaya dalam sebuah matriks seperti dalam pengelolaan SDM pada sub-bab terdahulu. Contoh matriks alokasi sumberdaya ini adalah:

Resources / Fase proyek	Computer	Printer	Program	Support	…
Requirements	V	V	V	V	
Functional	V		V		
Design	V		V		
Development	V	V	V	V	
Testing	V		V	V	

Pada setiap fase terlihat jenis sumberdaya apa saja yang dibutuhkan. Dari matriks alokasi sumberdaya ini akan dapat dilakukan juga kontrol apabila suatu saat terjadi perlambatan dalam pelaksanaan proyek. Apakah perlambatan akan menyebabkan aktivitas-aktivitas selanjutnya juga terlambat (karena pemakaian sumberdaya tertentu), atau keterlambatan itu tidak berdampak buruk bagi proyek, karena masih dalam batas yang wajar (ada di dalam rentang waktu *slack*nya).

BAB XV
SOFWARE TOOLS PROJECT MANAGEMENT

Perangkat lunak pendukung pengelolaan proyek (selanjutnya disebut PM *software* dalam buku ini), dewasa ini tersedia dengan berbagai harga, fasilitas dan menawarkan berbagai macam fungsi. Melihat kapasitas dan kegunaannya di dalam dunia nyata, PM *software* lebih jarang digunakan dibanding perangkat lunak lainnya, seperti: teks editor, spreadsheets, database ataupun presentasi software. Hal ini dikarenakan antara lain karena PM *software* cenderung lebih mahal daripada jenis-jenis perangkat lunak lainnya dan lebih sedikit orang dalam lingkungan perusahaan atau instansi yang mengetahui bagaimana menggunakan PM *software* secara efektif.

Di dalam diktat ini akan diberikan pengantar tentang kegunaan PM *software*, fungsi-fungsi pentingnya dan pedoman pemilihannya.

15.1 Kegunaan PM Software

Secara umum sebuah PM *software* memiliki kegunaan yang tersebut di bawah ini:

- Mempermudah pembuatan ***rangkaian kerja*** yang sebelumnya disusun secara *manual* (lewat Feasibiliy plan dan rencana global di atas kertas);

- Mempermudah kontrol terhadap jalannya proyek (lewat PND = ***Project Network Diagram***) dan **constraints** terhadap: schedule (jadwal), resource (sumberdaya), serta scope (ruang lingkup);
- Mempermudah ***penjadwalan***, penghitungan ***waktu*** kerja dan ***biaya***;
- Mempermudah ***koordinasi*** dan ***komunikasi*** antara peserta proyek (misalnya dalam suatu multi proyek).

Dari sini terlihat bahwa kegunaan suatu PM *software* terpaku pada pelaksanaan proyek. Tidak ada fungsionalitas lain yang memberi nilai tambah kepada perusahaan, sehingga hanya sebagian kecil perusahaan yang bersedia menyediakan fasilitas PM *software* dalam jaringan komputernya.

Lebih jauh lagi, penilaian terhadap sebuah PM *software* lebih bersifat subyektif dan tidak ada standar dalam penggunaannya. Seorang manajer proyek memiliki kebebasan untuk memilih antara menggunakannya atau tidak. Dan apabila menggunakan, pemilihannya diserahkan kepada si manajer proyek, kecuali ada standar yang berlaku di dalam proyek yang dipimpinnya.

Hambatan lain dalam pengunaan sebuah PM software adalah, sangat sedikit orang yang mengerti ataupun bersedia mempelajarinya. Seorang manajer proyek merupakan pemakai utama dari PM software ini. Keputusan untuk menggunakan sebuah PM software harus jelas. Semuanya tergantung pada besar pengelolaan proyek dan dinamika organisasi yang dipimpin. Bisa dibayangkan jika seorang manajer proyek lebih banyak kehilangan waktunya untuk mempelajari suatu jenis PM software daripada tugas utamanya sebagai pemimpin proyek. Untuk itu semuanya dikembalikan kepada si manajer proyek dalam pemilihan *software* yang cocok.

15.2 Pemilihan PM software

Sulit untuk menentukan secara umum jenis software apa yang diperlukan oleh berbagai tingkat pengguna. Pada diktat ini akan diberikan beberapa kriteria yang layak untuk dipertimbangkan:

- o Lihat besarnya proyek, sebuah proyek sederhana tidak perlu menggunakan PM software yang canggih.
- o Mudah dalam penggunaan.
- o Lihat *features* yang ditawarkan oleh software tertentu.

  Secara umum yang ditawarkan:
  - Format dan fasilitas Gantt chart dalam laporan dan penjadwalan;
  - Analisa PERT dalam pembuatan laporan dan penjadwalan;
  - Network process planning (utk. critical path, AON);
  - Pembuatan kalender kerja;
  - Fungsi-fungsi khusus dalam membuat laporan: *cash flow*, jadwal, persentase penyelesaian, alokasi sumberdaya, dll.
  - Interfacing dengan fasilitas online (email, web, ataupun software lainnya).
- o Pertimbangan harga (*lisensiper user*, lisensi per komputer, *open source*).
- o Kemungkinan adanya sentralisasi database, apabila terjadi pengelolaan multiproyek.
- o Adanya support (dokumentasi, pelatihan, dsb) dari vendor PM software tersebut.
- o Adanya *open source* dan *support* di Internet secara bebas, sebagai salah satu tolak ukur pertimbangan harga:

- Contoh dapat dilihat: http://www.phprojekt.com; software ini menggunakan konfigurasi php, apache server dan MySQL database dalam penggunaannya.

Seorang manajer proyek jangan sampai tenggelam dalam penggunaan perangkat lunak yang digunakan sehingga kehilangan kontak dengan tim proyek dan pelaksanaan proyek yang sedang dalam penyelesaian. Memilih software yang tepat sangat penting untuk mendukung kelancaran proyek keseluruhan. Dan ingat penggunaan software ini tidak mutlak, hanya sebagai pendukung yang menunjang koordinasi, informasi dan komunikasi antar pekerja dalam tim proyek.

15.3 Beberapa contoh PM software

Di bawah ini adalah beberapa contoh PM *software* yang saat ini banyak digunakan dalam berbagai proyek (diambil dari sebuah survei *"Tools of the Trade: a Survey of Project Management Tools"*; Project Management Journal September 1998).

- *Microsoft Project*.
- Primavera: Project Plan, Sure Track.
- Microsoft Excel (dengan pemanfaatan kelebihan penggunaan formula untuk perhitungan-perhitungan parameter, seperti dalam analisa PERT; dan tersedianya berbagai macam bagan serta grafik).
- Project Workbench.
- Time Line.
- Project Scheduler.
- Artemis.
- CA-SuperProject;

- Fas Tracs (banyak digunakan oleh para manajer proyek dalam proyek-proyek kecil, karena kemudahan dan kelengkapannya).

Program-program ini dinilai dalam hal ketepatan, format data dan tampilan bagan, serta kemudahan penggunaan.

15.4 Pembagian Proyek dan PM Software

Besar kecilnya proyek juga seringkali mempengaruhi pemilihan PM *software*, dan akan dibahas secara khusus di sini.

15.4.1 Proyek kecil

- Proyek kecil dalam satu area fungsional, PM *software* harus memenuhi fungsionalitas:
  - *Plan*
  - *Schedule*
  - *Durations (Gantt and PERT Charts)*
- Contoh:
  - TurboProject, Milestone Simplicity, Project Vision, QuickGantt, Primavera Sure Trak, dan Fas Tracs.
- Harga secara umum < $ 100.

15.4.2 Proyek Besar dengan Koordinasi

- Proyek besar dengan koordinasi (lebih dari satu unit fungsional) antara manajemen atas, sponsor, client, dan manajer proyek.
- Diperlukan:
  - Planning dan penjadwalan.
  - Simulasi proyek.

  - o Rescheduling dan optimalisasi bila ada perubahan.
  - o Mengestimasian biaya dan waktu.
  - o Diagram jaringan kerja.
- Contoh: Microsoft Project dan Primavera Sure Trak.
- Harga antara $ 300 - $ 500.

15.4.3 Multiproyek

- Koordinasi berbagai proyek atau sub-proyek yang menggunakan sumberdaya yang sama.
  - o Pengelolaan proyek secara keseluruhan.
  - o Protokol dalam *planning* dan *tracking*.
  - o Konsistensi dalam *roll-up* biaya.
  - o Dapat mengidentifikasi konflik-konflik sumberdaya (skala prioritas).
  - o Perhitungan biaya secara detail.
  - o Dapat memanajemen risiko.
- Contoh: MS-Project (dengan Microsoft Project Central), Primavera Project Planner, Open Plan, Cobra, Enterprise PM, Micro Planner X-Pert.
- Harga antara $ 400 - $ 20.000.

15.5 Microsoft Project

Secara khusus di dalam kuliah ini akan dibahas mengenai MS-Project. MS-Project memiliki beberapa fungsionalitas khusus yang dianggap melebihi PM software lainnya, yaitu:

- Adanya MS-Project Database, dengan kemampuan:
  - o Detail proyek di dalam databasenya;
  - o Informasi untuk menghitung dan memelihara jadwal, biaya, sumberdaya;
  - o Menciptakan suatu rencana proyek (beseline);

  - o Perbandingan antara baseline dan perubahan yang ada (dapat dijadikan alat simulasi);
- Hasil perhitungan secara langsung dapat dilihat.
- Estimasi biaya dan jadwal dengan menggunakan analisa PERT, catatan: jika menggunakan features MS Project Server (biaya ekstra pada saat pembelian software).

Perlu diperhatikan lebih jauh lagi bahwa features dan perbaikan dalam PM software berubah begitu cepat sehingga terkadang informasi atau kursus mengenai suatu software tertinggal dengan perkembangan software itu sendiri. Akibatnya pengetahuan proyek manajer tentang software ini seringkali terlambat dan terkesan usang.

Ditekankan sekali lagi, bahwa penggunaan PM software ini penting apabila diperlukan koordinasi antar unit kerja dalam proyek, koordinasi sumberdaya dalam proyek dan apabila proyek bersifat dinamis, misalnya jika si pemberi order seringkali melakukan perubahan requirements atau jika diperlukan iterasi berulang-ulang (penyesuaian prototip) untuk hasil kerja proyek.

BAB XVI
MENILAI KUALITAS PROYEK PL

Pengertian mengenai kualitas selalu mudah untuk dikatakan, tetapi sulit untuk dipahami. Semua orang selalu berbicara mengenai kualitas. Namun, apa sebenarnya kualitas tersebut? Secara umum kualitas digambarkan sebagai suatu titik dimana pemakai produk menjadi puas. Tentu saja ini dapat meliputi banyak hal. Tingkat kepuasan setiap orang pasti berbeda. Jika pengertian ini yang digunakan dalam menilai hasil suatu produk, maka tidak akan ada produk yang selesai di dunia ini. Seorang manajer proyek, dalam bidang apapun, perlu mempersempit pengertian kualitas ini.

Bagi seorang manajer proyek, pengertian kualitas dapat dipersempit sebagai berikut:

1. Kualitas dalam penyelesaian produk yang dihasilkan (deliverable); dan
2. Kualitas proses yang diperlukan dalam penyelesaian produk.

Yang pertama erat kaitannya dengan faktor-faktor dari luar (eksternal), yaitu keinginan pengguna dan pemberi order yang dituangkan dalam *requirements* serta bagaimana mencegah terjadinya kesalahan yang berakibat buruk bagi pengguna. Yang kedua erat dengan faktor-faktor internal dalam organisasi proyek, yaitu: bagaimana

mengendalikan proses dalam proyek sehingga dapat memenuhi suatu target sesuai jadwal dan biaya, serta menguraikan bagaimana tanggung jawab dari masing-masing anggota dalam tim kerja proyek.

Dalam manajemen proyek, proses pemenuhan kualitas ini dikenal dengan istilah: *Project Quality Management*. Kegiatan di dalam manajemen kualitas proyek ini meliputi: perencanaan, jaminan kualitas dan kontrol kualitas. Pendekatan dari proses-proses dalam pengelolaan kualitas ini diharapkan juga dapat memenuhi pendekatan dari *International Standard Organization* (ISO), di dalam seri ISO 9000 dan 10000, dan *Total Quality Management* (TQM).

15.1 Pembagian Fase Manajement Kualitas

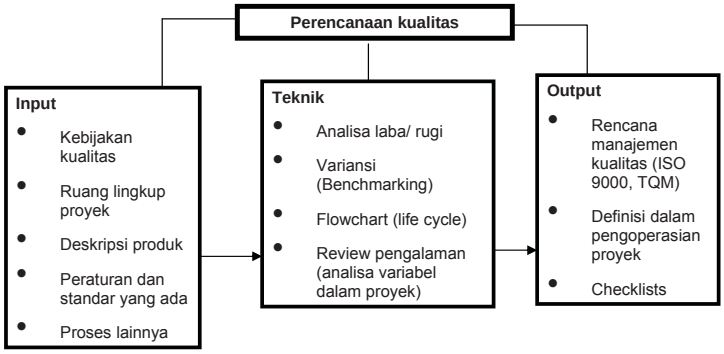
Selain menggunakan fase proyek sebagaimana telah disusun dalam WBS ataupun jaringan kerja, pembagian fase untuk pengelolaan kualitas dapat pula dilakukan melalui pembagian fase-fase seperti berikut ini:

- *Project genesis* (awal);
- *Project planning* (perencanaan);
- *Project execution* (pengimplementasian);
- *Project control* (iterasi dari pengimplementasian);
- *Project closure* (penutupan proyek).

Dalam masing-masing fase di atas, dapat dilakukan *quality cycles*, yaitu melakukan perencanaan, penjaminan dan pengontrolan kualitas produk untuk setiap deliverables yang dihasilkan.

15.2 Perencanaan Kualitas

Dalam merencanakan suatu kualitas (*Quality Planning*), seorang manajer proyek dapat menggunakan pendekatan dengan alur kerja sebagai berikut:



Sebuah pendefinisian kualitas bagi produk dari sebuah proyek memerlukan penduan atau arah yang telah didefinisikan dari pihak manajemen atas. Tim kerja proyek, di bawah koordinasi manajer proyek perlu memperhatikan panduan ini dalam membatasi kriteria produk yang akan dihasilkan melalui pelaksanaan proyek. Dalam ruang lingkup proyek (*project charter*) sebenarnya telah dituliskan secara global, seperti apa produk yang harus dihasilkan dan elemen-elemen apa saja yang harus diikutsertakan untuk membentuk produk pelaksanaan proyek yang handal, seperti: teknologinya, teknik pengerjaannya ataupun deskripsi produk itu sendiri. Dalam perencanaan kualitas perlu juga diperhatikan peraturan dan standar yang berlaku, yang sekiranya dapat mempengaruhi hasil kerja dalam proyek, misalnya: penggunaan format data baku (seperti XML) untuk pertukaran informasi proyek.

Dalam membuat rencana kualitas, tim kerja proyek,

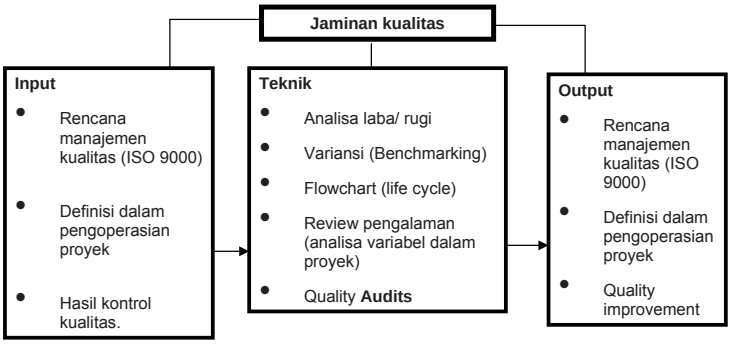
harus melakukan analisa yang cermat. Setiap perubahan dalam biaya, waktu dan risiko harus diamati dan dicari kemungkinan terbaiknya. Teknik-teknik dalam *flowcharting* (analisa risiko), *benchmarking* (menghitung perbedaan biaya antara rencana dan kenyataan serta *trade-offs*-nya), ataupun analisa laba / rugi akan sangat menolong dalam melakukan analisa untuk jaminan kualitas.

Hasil dari perencanaan kualitas ini, dapat berupa suatu *checklist* yang harus dipenuhi untuk dapat menjamin bahwa produk yang dihasilkan melalui proyek memang memenuhi kriteria kualitas yang handal.

15.3 Penjaminan Kualitas

Jaminan kualitas (*Quality Assurance*) merupakan tindakan yang dilaksanakan untuk memenuhi kriteria kualitas seperti yang telah direncanakan. Pelaksanaan jaminan kualitas harus terjadi pada setiap fase dalam proyek, serta hasilnya harus transparan baik bagi tim kerja, pihak manajemen, sponsor dan juga bagi pengguna akhir.

Alur kerja dalam proses penjaminan kualitas dapat dilihat di bawah ini:



15.3.1 Laporan Kemajuan Proyek

Pada setiap fase diperlukan adanya laporan yang menjelaskan bagaimana pelaksanaan proyek sampai dengan fase tersebut selesai. Laporan kemajuan proyek ini merupakan rangkuman untuk peninjauan kualitas, penjadwalan dan pembiayaan. Hal-hal yang perlu dilaporkan, antara lain:

- Status proyek saat ini (dari *review* sebelumnya)
- Laporan kumulatif
  - Pekerjaan yang selesai;
  - Pekerjaan yang terlambat (*lagging/delay*) dan rencana untuk mengatasi keterlambatan tersebut;
  - Pekerjaan yang signifikan bila dilihat dari tujuan proyek;
  - Variansi / benchmarking biaya dan waktu;
  - Informasi biaya.
- *Management summary* (ringkasan dari laporan kumulatif)
- Laporan variansi yang terperinci
  - Keadaan biaya;
  - Keadaan implementasi di lapangan;
  - Persentase proyek yang selesai;
  - Perubahan dalam tim kerja;
  - Perubahan rencana kerja;
  - Posisi saat ini dalam critical path;
  - Pekerjaan (*tasks*) yang tertunda;
  - Deadline yang tertunda;
  - Rencana selanjutnya.

Hasil dari laporan kemajuan proyek pada setiap fase dapat digunakan sebagai basis dalam perbaikan kualitas (*quality improvement*).

15.3.2 Audit kualitas

Selain laporan kemajuan di atas, untuk setiap fase perlu dilakukan audit (review) untuk mengidentifikasi hal-hal apa yang dapat diperbaiki dan sebagai acuan untuk menyelaraskan rencana selanjutnya dalam pelaksanaan proyek.

Audit kualitas, biasanya dilakukan oleh pihak ketiga, seperti: pihak sponsor ataupun tenaga ahli (konsultan) yang disewa pihak manajemen untuk mendukung penilaian proyek. Dengan pengadaan review pihak ketiga ini diharapkan akan:

- Menjamin kualitas dan akurasi;
- Objektivitas terjaga;
- Tim kerja dapat menilai performance serta produktivitas masing-masing;
- Mengangkat mutu pekerjaan dari masing-masing pekerja;
- Manajer proyek dapat menilai jalannya proyek keseluruhan secara tepat;
- Manajer proyek dapat mengambil tindakan penyesuaian bila dibutuhkan.

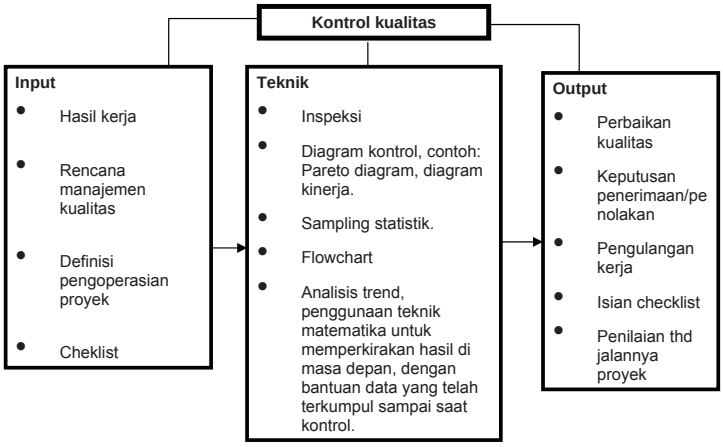
Selain lewat review dari pihak ketiga, penjaminan kualitas dapat juga dilakukan antara sesama anggota tim kerja dalam proyek, dengan melakukan *peer review*. Tujuannya antara lain:

- Meyakinkan bahwa setiap task terkontrol;
- Tim kerja saling mempresentasikan hasil kerjanya;
- Mempelajari bidang lain dalam proyek (totalitas proyek);
- Proyek manajer dapat mengotrol sejauh mana proyek berlangsung;
- Mempererat tanggung jawab sesama tim kerja untuk keseluruhan proyek.

15.4 Kontrol Kualitas

Kontrol kualitas (*Quality Control*) meliputi pemonitoran hasil kerja proyek secara berkala untuk menilai apakah hasil kerja tersebut sesuai dengan standar kualitas yang telah direncanakan dan untuk mengidentifikasikan tindakan yang diperlukan guna memperbaiki kualitas dari hasil proyek yang kurang memuaskan.

Alur kerja kontrol kualitas dapat dilihat di bawah ini:



15.5 Software Quality Management

Secara khusus dalam proyek pengembangan perangkat lunak, standar pengelolaan kualitas diatur secara internasional dalam ISO 9126. Apabila sebuah proyek perangkat lunak dapat memenuhi standar ini maka, kualitasnya sudah sangat baik sekali meskipun nantinya dalam pengimplementasiannya kepada pihak pengguna masih akan terlihat berbagai macam kekurangan.

Banyaknya kekurangan ini harus disadari antara lain

karena dalam penggunaan sebuah produk perangkat lunak, terkadang lebih bersifat subyektif dan kurang objektif. Bisa dibayangkan bahwa pengguna sebuah produk perangkat lunak memiliki tingkat pengertian yang tidak sama dalam pengaplikasiannya.

Dalam melakukan peninjauan kualitas terhadap produk perangkat lunak, **ISO 9126**, melakukan penilaian terhadap enam karateristik utama, yaitu:

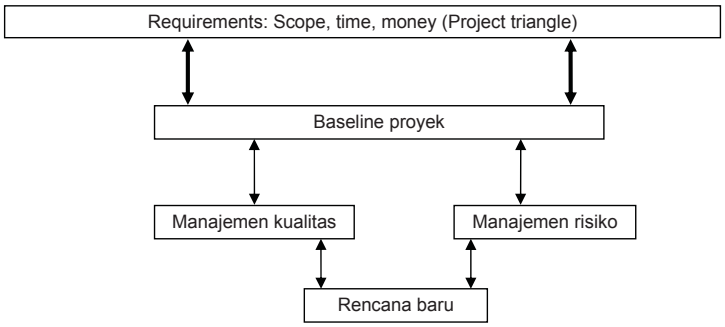
1. *Functionality* (tingkat fungsionalitas):
   - o suitability, accuracy, interoperability, compliance, security;
2. *Reliability* (tingkat kepercayaan):
   - o maturity, fault tolerance, recoverability;
3. *Usability* (tingkat penggunaan):
   - o understandability, learn ability, operability;
4. *Efficiency* (tingkat efisiensi):
   - o time behavior; resource behavior;
5. *Maintainability* (tingkat pemeliharaan):
   - o analyzability, changeability, stability, testability;
6. *Portability* (tingkat efisiensi transfer data):
   - o adaptability, install ability, conformance, replace ability.

15.6 Pengelolaan Perubahan Proyek (Project Change Management)

Dengan adanya pengawasan terhadap kualitas dan juga perencanaan dalam menanggulangi risiko, sebuah proyek dalam perjalanannya dapat mengalami beberapa perubahan. Perubahan ini bisa ditinjau dari berbagai sudut, yaitu: dari pembagian fase, penjadwalan, lingkup pekerjaan (*requirements*), urutan aktivitas, ataupun pembiayaan.

Perubahan pada salah satu sudut di atas menuntut penyesuaian rencana semula (*baseline adjustment*). Dengan penyesuaian ini akan didapatkan suatu baseline baru untuk pelaksanaan proyek selanjutnya. Perlu diperhatikan bahwa dengan mengubah rencana awal, maka risiko bahwa durasi pengerjaan proyek akan menjadi lebih panjang, dan ini juga akan merambat kepada meningkatnya biaya proyek.

Manajemen risiko dan manajemen kualitas bisa berfungsi sebagai alat kontrol yang ampuh dalam pengelolaan proyek. Hubungan antara scope, manajemen kualitas, manajemen risiko dan baseline dalam baseline proyek dapat dilihat pada gambar di bawah ini:



15.7 Peran manajer Proyek

Dalam setiap pengambilan keputusan, peran seorang manajer proyek sangat penting. Boleh dikatakan untuk mengendalikan proyek, manajer proyek wajib mengambil langkah-langkah membuat proyek dapat berjalan sesuai rencana. Sebagai contoh: bila pada proyek peremajaan jaringan komputer, ternyata vendor server-nya tidak dapat mengirim barang tepat waktu, bahkan harus ditunda selama dua minggu karena sesuatu dan lain hal, maka apa tindakan yang harus dilakukan? Apakah harus menunggu selama itu, sedangkan waktu penyelesaian sudah semakin

dekat? Apakah harus berganti merk server atau … ???

Dalam pengambilan keputusan ini, manajer proyek dapat berperan dengan cara:

o ***Directive***:
  - sedikit input dari anggota tim atau bahkan tanpa input dari anggota tim lainnya.

o ***Participative***:
  - pengambilan keputusan lewat diskusi, kompromis, tukar pengalaman dan brainstorming.

o ***Consultative***:
  - kombinasi kedua metode di atas.
  - Tim kerja mengajukan usul perubahan, dan manajer mengambil keputusan untuk melaksanakan atau tidak.
  - Biasanya terjadi pada proyek dengan deadline singkat, kompleks dan biaya terbatas.

BAB X VI
PENYELESAIAN AKHIR PROYEK

Pada bab-bab terdahulu telah dipelajari konsep kerja utama dalam pengelolaan sebuah proyek dengan titik berat pada proyek IT, terutama dalam pengembangan perangkat lunak. Sebagai penyelarasan akhir akan dibahas kapan sebenarnya dapat diambil kesimpulan bahwa sebuah proyek itu dinilai selesai.

Sebuah proyek merupakan gabungan berbagai macam aktivitas yang terbagi dalam fase-fase dengan pelaksanaan oleh sumberdaya manusia yang dipercaya beserta bantuan sumberdaya non-manusia yang diperlukan dan harus selesai dalam batasan suatu waktu serta biaya.

Setiap fase dalam proyek menghasilkan satu atau lebih deliverables, sebagaimana direncanakan melalui WBS dan AON. Pada akhir dari jalannya proyek, keseluruhan deliverables tersebut harus dilengkapi dan dievaluasi apakah telah sesuai dengan permintaan (*requirements*) dari pemberi order. Setiap aktivitas beserta sumberdaya manusia pelaksananya harus dievaluasi kinerjanya, untuk menentukan apakah semua yang telah dihasilkan sesuai dengan permintaan. Setelah evaluasi ini selesai harus diadakan suatu pertemuan khusus dengan si pemberi order guna menjelaskan dan menyerahakn hasil pekerjaan.

Bisa saja terjadi bahwa si pemberi order masih merasa belum puas, padahal semua pekerjaan telah dilakukan sesuai dengan scope dan requirements awal. Bila hal ini

terjadi, seorang manajer proyek harus bisa mencari jalan keluar dan penjelasan yang logis sehingga si pemberi order mengerti bahwa semuanya telah dilakukan dalam batasan yang diberikan. Apabila pemberi order masih belum mau menerima juga, perlu dilakukan suatu *post-project* fase, guna melengkapi requirements yang "tertinggal" ini. Sekali lagi semuanya harus tetap dilakukan dalam batasan waktu dan biaya, sehingga proyek nantinya dapat dikatakan berhasil (tahap *client acceptation*).

16.1 Mengevaluasi Deliverables

Hasil proyek dapat dilihat pada deliverables yang telah disetujui oleh pihak pemberi order dan pihak pelaksana. Dalam mengevaluasi deliverables ini perlu dilakukan evaluasi terhadap jalur kritis yang digambarkan dalam jaringan kerja. Apabila semua jalur kritis telah dilewati dan menghasilkan produk yang sesuai dengan persetujuan, maka kemungkinan proyek dapat dinilai berhasil akan menjadi semakin besar.

Dalam melakukan evaluasi deliverables dalam critical path-nya ini perlu dilakukan:

- Penilaian bahwa semua jalur kritis sudah dilewati dan menghasilkan produk yang sesuai (deliverables yang lengkap);
- *Review* deliverables setiap fase dalam jaringan kerja;
- Menggabungkan produk dari awal hingga akhir (masa-masa kritis), disini peran serta manajer proyek sanagat penting;
- Adanya suatu ***Management reverse***: perekayasaan waktu kerja pada akhir proyek untuk kompensasi delay/lagging, biasanya antara 10% - 15% dari total durasi proyek.

Dalam evaluasi jalur kritis, proyek sebenarnya memasuki masa kritis berikutnya, namun tidak tercantum dalam jaringan kerja, yaitu usaha untuk menggabungkan semua hasil kerja guna membentuk satu produk yang utuh dan lengkap sesuai requirements dari pemberi order. Sebagai contoh dalam suatu proyek pembuatan perangkat lunak, pada babak akhir proyek tim kerja harus menggabungkan semua fungsionalitas (dalam hal ini berbagai jenis GUI, fungsi, prosedur, ataupun sub-routine) sebagai satu *executable* untuk aplikasi tertentu. Proses penggabungan ini tidak mudah dan melelahkan, bahkan biasanya banyak perbaikan (*error correction and modification*) yang harus dilakukan dalam tahap ini.

16.2 Evaluasi tim Kerja

Bersamaan dengan (atau setelah) evaluasi deliverables di atas, harus dilakukan juga suatu evaluasi terhadap tim kerja proyek. Evaluasi sangat berguna untuk menilai kinerja tim dan menjadi landasan dalam pelaksanaan proyek-proyek berikutnya.

Secara umum evaluasi tim kerja akan menghasilkan hal-hal berikut ini:

- *Performance review* (penilaian kinerja), hal ini terdiri atas:
  - Penilaian kontribusi terhadap proyek dari masing-masing anggota tim.
  - Penilaian kemampuan bekerja sama antara anggota.
  - Penilaian komitmen terhadap kualitas proyek.

- Pengalaman untuk proyek berikutnya.
  - Pelajaran, usul, saran dan data-data proyek dapat didokumentasikan dan dapat berfungsi sebagai acuan dalam proyek-proyek selanjutnya.

- Pengaruh terhadap posisi, gaji, status, dsb dalam organisasi atau lingkungan.
  - Dalam suatu instansi atau perusahaan yang besar, keberhasilan pelaksanaan suatu proyek dapat membawa dampak terhadap gaji, jabatan ataupun status bagi pegawai tetapnya.
  - Pengaruh-pengaruh ini dapat menjadi suatu motivasi bagi anggota tim untuk berprestasi lebih baik lagi dalam proyek-proyek selanjutnya.

- Pendapat objektif yang membangun.
  - Perlu diingat bahwa pendapat yang diberikan harus obyektif terhadap kinerja yang dihasilkan. Jika tidak justru malah akan berbahaya karena penilaian yang bersifat subyektif akan menyebabkan perasaan sinis dan mungkin akan mengganggu kerja sama tim yang selama ini telah terbina dengan baik.
  - Pendapat yang subyektif dalam proyek akan sangat berbahaya pada saat proyek berada pada tahap akhir, karena dengan pendapat yang tidak baik mengenai seseorang mungkin akan menyebabkan penyelarasan akhir proyek tidak akan pernah tercapai, dan proyek dinilai gagal.

16.3 Client Acceptation

Jika hasil deliverables selesai digabungkan dan membentuk suatu produk utuh yang sesuai dengan keinginan pemberi order, maka proyek dapat dianggap telah selesai. Tinggal sekarang bagaimana proses penyerahan produk kepada si pemberi order.

Ada dua macam penyerahan produk kepada pemberi order:

1. Penerimaan informal, biasanya dilakukan pada proyek-proyek kecil dan sederhana.
   Penilaian dilakukan terhadap:
   - o Dead-line dan tujuan proyek.
   - o Produk tidak butuh evaluasi lebih jauh, karena dapat langsung diaplikasikan kepada pengguna dan telah sesuai dengan *requirements*.
2. Penerimaan formal
   Dalam suatu acara penerimaan formal akan dilakukan:
   - o ***Final Sign-off***: presentasi dan penjelasan kepada pemberi order / client, tim kerja, ataupun pihak manajerial atas.
   - o ***Project acceptance agreement*** (konfirmasi *deliverables* yang diharapkan).
     - Sebagai catatan: bisa saja terjadi bahwa semua telah sesuai dengan requirement, namun pemberi order masih ingin sesuatu yang lebih, maka mungkin saja dibutuhkan ekstra pekerjaan di dalam proyek. Hal ini dikenal juga dengan istilah *post project activities*.

16.4 Laporan akhir dan pendokumentasian

Dalam suatu proyek besar dan formal, tim kerja di bawah koordinator seorang manajer proyek, perlu memberi suatu laporan akhir yang lengkap sebagai salah bukti pelaksanaan proyek, dan dapat digunakan sebagai historis data dalam proyek-proyek sejenis selanjutnya di masa depan.

Dalam laporan akhir harus tercantum:

- Tercantum dengan jelas visi dan misi dari proyek.
- Proposal proyek awal dan ide-ide teknis implementasi.
- Baseline proyek (rencana awal biaya dan jadual).
- WBS, OBS dan jaringan kerja.
- Kumulatif laporan berkala pada tiap fase.
- Perubahan yang terjadi dan penanggulangan risiko.
- Laporan informal yang berkaitan (memo, email, dsb).
- Biaya total proyek (dalam realitasnya).
- Perjanjian penerimaan deliverables dari pemberi tugas (*client acceptance agreement*).
- Post-project activities and audit (maintenance, succes story, extra business value).
- Suatu nilai tambah bagi proyek, yang mengiringi kesuksesan proyek.

16.5 Indikator Keberhasilan Suatu Proyek IT

Dalam pelaksanaannya, sebuah proyek IT memiliki indikator-indikator yang menjadi tolak ukur penilaian kesuksesan. Indikator-indikator tersebut adalah:

- Visi yang jelas dari proyek dan *deliverables*-nya.

- Ketrampilan serta profesionalisme (komitmen kerja) yang memadai dari manajer proyek dan tim kerja.
- Sumberdaya keuangan yang memadai untuk melengkapi implementasi.
- Estimasi waktu yang tepat.
- Manajemen risiko dan kualitas yang memadai.

16.6 Kegagalan proyek IT

Meskipun segala sesuatunya sudah direncanakan secara matang, tentu saja ada faktor-faktor yang dapat melemahkan rencana tersebut. Dalam suatu proyek IT faktor-faktor pelemah tersebut dapat digolongkan ke dalam hal-hal berikut ini:

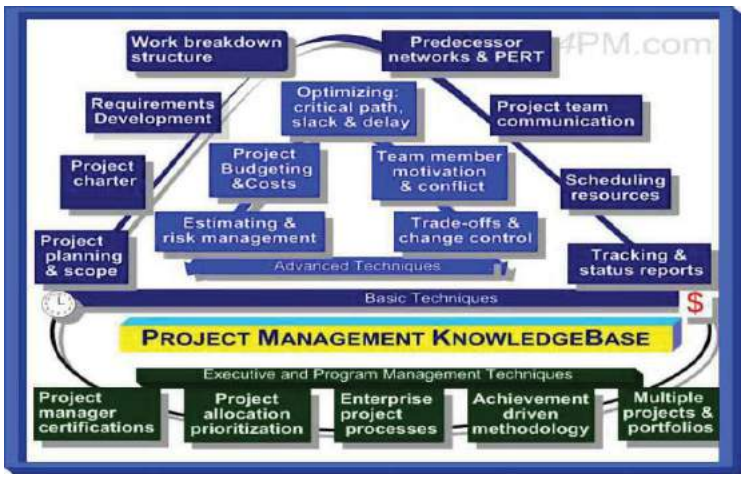
- Adanya teknologi alternatif baru yang lebih murah dan kualitasnya tidak berdeda.
- Adanya solusi yang lebih baik dari tim kerja lain.
- Pemberi order tidak menindaklanjuti deliverables yang disepakati. Dalam hal ini proyek selesai, namun hasilnya tidak digunakan sesuai dengan rencana semula.
- Organisasi (perusahaan) berganti haluan.
- Organisasi (perusahaan / sponsor) mengalami kesulitan dana.
- Organisasi (perusahaan) mengalami restrukturisasi, reorganisasi, misalnya: merger (penggabungan) dengan organisasi lain.

16.7 Kata Akhir

Di sini, aktivitas-aktivitas utama dalam manajemen proyek telah disinggung, dan dianalisis penggunaannya, terutama untuk pengelolaan suatu proyek dalam lingkungan IT.

Secara lengkap dalam manajemen proyek, aktivitas-aktivitasnya digambarkan sebagaimana tercantum pada halaman selanjutnya (bandingkan juga dengan rangkaian konsep kerja manajemen proyek yang dibahas pada bab 2). Aktivitas yang tergambar ini merupakan pemilahan dari konsep kerja pada bab 2. Untuk menjamin keberhasilan suatu proyek, tidak harus semua aktivitas dalam manajemen proyek dilakukan. Yang terpenting adalah bahwa harus terdapat aktivitas berikut ini (bandingkan juga dengan fase pada manajemen kualitas pada bab yang lalu):

- Proyek *genesis,* yaitu proses kelahiran proyek yang dilanjutkan dengan riset untuk membentuk suatu *feasibility plan*.
- Perencanaan global (*Planning on cost and schedule*)
  - o Pembuatan rencana / estimasi jadwal dan biaya.
- *Eksekusi* proyek
  - o Organisasi kerja (CPM, AON, alokasi matriks)
- Kontrol proyek
  - o Manajemen risiko
  - o Manajemen kualitas
  - o Pengembangan kualitas
- Laporan akhir (*Closure*) sebagaimana tertulis pada sub-bab sebelum ini.



Dengan penguasaan teori dan konsep kerja yang ditawarkan dalam manajemen proyek, maka tingkat keberhasilan suatu proyek akan semakin tinggi dan hasilnya akan dapat sesuai dengan rencana atau permintaan dari sang pemberi order. []

DAFTAR PUSTAKA

Philips, Joseph. 2002. ***IT Project Management: On Track from Start to Finish***. Mc Graw Hill Osborne.

Gray, Clifford F. 2000. ***Project Management: The Managerial Process***. Larson; Mc Graw Hill International Editions.

Hughes, Bob & Mike Cotterell. 1999. ***Software Project Management 2/e***. Mc Graw Gill Companies.

Pressmen, Roger S. 1997. ***Rekayasa Perangkat Lunak, Pendekatan Praktisi***; McGraw-Hill Book, Co.

Anonim. 1996. ***A Guide toThe: Project Management Body of Knowledge***, Project Magement Institute.

Davison, Jeff. 2000. ***Manajemen Proyek dalam 10 menit***. (terjemahan oleh Sisnuhadi). Yogyakarta: Penerbit Andi.

Christianto, I Made Wiryana. 2002. ***Pengantar Manajemen Proyek Berbasis Internet***. Jakarta: Elex Media Komputindo.

Martadi, Alif. 1986. ***Perencanaan Projek dengan Metode Jaringan Kerja, Beserta Aplikasi pada Mikro Komputer***. Jakarta: Penerbit PT. Golden Terayon Press.

Anonim. 2000. ***A Buyer's Guide To Selecting Project Management Software***. The Hampton Group, Inc., Jun.

Hewson, Geoff. ***2000. Managing Web Projects: Recipes for Success***. Soffware Productivity Center Inc, Jul 2000.

Abrecht, A.J. & J.E. Gaffney. 1983. ***Software function, source lines of code and development effort production; a software science validation.*** IEEE Trans. Software Engineering; Nov, hal. 639-648.

Jones, C. & McGraw-Hill. 1991. ***Applied software measurement***;

Project Management Institute : http://www.pmi.org

Project Management Training, Tools, Techniques and Textbooks : http://www.4pm.com

Lampiran A

MENGGUNAKAN MICROSOFT PROJECT

Apa yang dimaksud dengan Manajemen Proyek?

Manajemen proyek adalah proses perencanaan, pengorganisasian, dan pengaturan kegiatan-kegiatan dan sumberdaya-sumberdaya untuk mencapai suatu target prestasi tertentu, biasanya dalam batasan waktu, sumberdaya dan biaya. Suatu rencana proyek dapat berupa sesuatu yang sederhana, seperti daftar kegiatan-kegiatan dan waktu mulai dan selesai yang ditulis di dalam buku catatan. Atau dapat juga berupa sesuatu yang kompleks contohnya ribuan kegiatan dan sumberdaya dan anggaran yang mencapai jutaan dollar.

Kebanyakan aktivitas di dalam proyek memiliki kemiripin, di antaranya memecahkan proyek ke dalam kegiatan-kegiatan lebih kecil yang dapat diatur dengan lebih mudah, penjadwalan kegiatan-kegiatan, komunikasi dengan tim, dan memonitor kemajuan dari kegiatan-kegiatan yang dijalankan. Semua proyek pada dasarnya terdiri dari 3 fase utama yaitu:

1. Pembuatan Rencana
2. Pengamatan dan Pengaturan Proyek
3. Penutupan Proyek

Semakin sukses ketiga fase tsb. dilaksanakan, semakin besar kesempatan suatu proyek untuk sukses.

Segitiga Proyek

Kita selalu berharap untuk dapat meramalkan apakah sebuah proyek dapat berjalan dengan baik, atau tidak baik. Untuk itu, kita perlu mengetahui ***ketiga hal berikut, yang sangat penting di dalam manajemen proyek:***

- Waktu: Waktu yang diperlukan untuk menyelesaikan proyek tercermin di dalam jadual proyek.
- Uang: Anggaran proyek, didasarkan pada biaya sumberdaya-sumberdaya: manusianya, perlengkapan, dan materi yang dibutuhkan untuk melaksanakan kegiatan-kegiatan yang ada.
- Cakupan: Tujuan dan kegiatan-kegiatan dari proyek, dan usaha yang diperlukan untuk menyelesaikannya.

Ketiga faktor membentuk suatu segitiga. Menyesuaikan satu dari ketiga elemen ini akan mempengaruhi 2 lainnya. Sementara ketiga elemen ini penting, biasanya hanya ada salah satu dari ketiganya yang akan sangat mempengaruhi proyek yang dikerjakan. Hubungan antara elemen-elemen tsb. berbeda di setiap proyek dan menentukan tipe masalah yang akan ditemui dan pemecahan yang dapat diterapkan. Mengetahui sebagaimana proyek dibatasi atau sebagaimana fleksibel mempermudah perencanaan dan pengaturan dari proyek.

Microsoft Project Database

Sebagai manajer proyek, seringkali banyak sekali yang harus dilakukan. Bagaimana software MS Project ini dapat menolong? Pertama, software ini menyimpan

detail mengenai proyek di dalam databasenya. Kemudian, software ini juga menggunakan informasi tsb. untuk menghitung dan memelihara jadwal, biaya, dan elemen-elemen lain, termasuk juga menciptakan suatu rencana proyek. Semakin banyak informasi yang disediakan, semakin akurat rencana yang dapat dibuat. Seperti spreadsheet, MS Project memperlihatkan hasil perhitungan secara langsung. Tapi, rencana proyek tidak akan selesai sebelum semua informasi kritis mengenai proyek dan kegiatan-kegiatannya dimasukkan. Setelah itu, baru dapat dilihat kapan proyek selesai dan kapan jadwal keseluruhan dari semua aktivitas benar-benar terlihat.

MS Project menyimpan informasi yang dimasukkan oleh pengguna dan menghitung di dalam suatu fields, yang menyimpan informasi spesifik seperti nama kegiatan, atau lamanya. Di dalam MS Project, setiap field dimunculkan di dalam sebuah kolom.

Melihat data yang dibutuhkan

Hari ini, fokus pada deadlines. Besok, pada biaya. Database proyek terdiri dari berbagai informasi, tapi pada waktu tertentu, proyek hanya membutuhkan sebagian saja dari semua itu. Untuk mendapatkan hal-hal tertentu tsb., MS Project memiliki alat-alat berikut ini:

- **Views** memperlihatkan suatu set informasi di dalam format yang mudah diinterpretasikan. Contohnya, Gantt Chart memperlihatkan informasi dasar di dalam kolom dan grafik batang;
- **Tables** mendefinisikan kolom yang ingin diperlihatkan;
- **Filters** berfokus pada kegiatan atau sumberdaya spesifik.

Seperti channel TV, setiap view memperlihatkan informasi yang berbeda. Tabel-tabel dan filters memperjelas informasi. Seperti juga berpindah dari channel yang satu ke yang lainnya, berpindah dari satu view ke yang lain tidak akan menghapus informasi. Sementara filter dapat juga menyembunyikan informasi, tapi tetap tidak menghapuskannya.

Bagaimana MS Project melakukan penjadwalan

Bagaimana MS Project menjadwalkan suatu awal dan akhir suatu kegiatan? Banyak faktor dipertimbangkan, termasuk ketergantungan kegiatan-kegiatan, batasan-batasan, dan interupsi seperti hari libur. Lebih penting lagi, MS Project menjadwalkan setiap kegiatan dengan menggunakan formula:

Duration = work / resource effort , di mana:

- Duration adalah **jumlah waktu** sesungguhnya yang diperlukan untuk menyelesaikan suatu kegiatan.
- Work adalah **usaha** yang diperlukan pada suatu periode waktu untuk menyelesaikan suatu kegiatan.
- Resource effort adalah **jumlah usaha sumberdaya** yang dialokasikan untuk mengerjakan kegiatan tsb.

Contohnya, jika:

- Tiga tukang cat bekerja selama 2 hari untuk suatu kegiatan tertentu, dengan usaha 8 jam per hari, work untuk setiap sumberdaya adalah 16 jam. (2 hari* 8 jam).
- Total effort dari sumberdaya adalah 24 jam per hari (3 tukang* 8 jam).

- Total work untuk kegiatan tsb. adalah 48 jam: (2 hari * 8 jam* 3 tukang).
- Duration adalah 2 hari yang didapatkan dari 48 jam/ (3 tukang* 8 jam).

Pengertian akan formula ini sangat penting untuk mengerti bagaimana perubahan yang dilakukan mempengaruhi waktu penyelesaian proyek.

Menyatukan semuanya

Setelah daftar kegiatan terciptakan dan menyediakan informasi jadwal, rencana dapat disusun. Sekarang dapat terlihat model keseluruhan dari proyek, termasuk tanggal penyelesaian dan tanggal mulai untuk setiap kegiatan. Apa lagi selanjutnya?

- Pada waktunya untuk suatu proyek dari awal sampai akhir. Jika ada kegiatan yang tertunda, ini dapat menyebabkan penundaan penyelesaian proyek.
- Evaluasi rencana sampai optimal. Sebelum proyek dimulai dan secara periodik selama proyek berlangsung, akan diperlukan evaluasi dan penyesuaian rencana. Pertimbangkan cakupan, sumberdaya dan jadwal.
- Update Microsoft Project mengenai kemajuan dari kegiatan-kegiatan. Sebagai gantinya, software ini akan memperlihatkan rencana proyek yang sudah diupdate. Proyek dapat diupdate melalui MS Project Central atau email. Setelah rencana diupdate, review untuk melihat efek dari perubahan. Apakah proyek menjadi over budget? Apakah ada seorang anggota tim yang sekarang harus dijadwalkan untuk lembur? Apakah proyek akan menjadi terlambat?

- Tutup proyek. Evaluasi apa yang sudah diterima dan praktek terbaik apa yang dapat dilakukan untuk proyek-proyek selanjutnya.

Berikut ini adalah petunjuk-petunjuk untuk mempelajari MS Project. Perlu disadari bahwa di dalam handout ini, belumlah semuanya lengkap, karena banyak sekali features dari MS Project ini, dan yang dibahas hanya seperlunya saja. Jika ada pertanyaan lebih, dapat diperiksa melalui Help menu di dalam MS Project.

MANUAL 1:
MEMBUAT SUATU RENCANA PROYEK

Ketika Anda pertama mendefinisikan tujuan proyek dan memikirkan fase-fase di dalam proyek anda, adalah sangat penting untuk menciptakan rencana proyek.

Pertama, masukkan dan organisasikan daftar kegiatan-kegiatan untuk diselesaikan, juga lama penyelesaian setiap kegiatan (duration). Kemudian, tambahkan orang, perlengkapan, dan materi beserta biayanya untuk rencana anda. Kemudian, tempatkan sumberdaya-sumberdaya tsb. untuk kegiatan-kegiatan ybs. Dengan informasi itu, MS Project akan menciptakan suatu jadual. Anda dapat memeriksa dan menyesuaikannya sebagaimana perlu.

1.1 Menciptakan suatu proyek baru

Ketika memulai suatu proyek baru didalam MS Project, anda dapat memasukkan waktu mulai atau finish, tapi tidak keduanya. Direkomendasikan bahwa anda hanya memasukkan waktu mulai proyek dan membiarkan MS Project untuk menghitung waktu penyelesaiannya

setelah anda memasukkan semua kegiatan dan menjadwalkannya.

Jika proyek anda harus diselesaikan pada tanggal tertentu, masukkan finish date saja. Bahkan jika anda pada mulanya menjadwalkan mulai dari finish date, lebih baik untuk menjadwalkan dari start date setelah proyek dimulai.

Langkah-langkah:

1. Klik **File – New**, atau tombol shortcutnya.
2. Pilih menu **Project – Project Information**, masukkan start date atau finish date dari proyek anda, kemudian klik **OK**.
3. Klik **Save**
4. Di dalam File name box, ketik nama proyek anda, kemudian **Save**

1.2 Mengatur Kalender Proyek

Anda dapat mengganti kalender proyek sesuai dengan hari kerja dan jam-jam untuk setiap orang di proyek anda. Kalender dasar yang berlaku adalah Senin ke Jumat, 8 pagi sampai 5 sore, dengan 1 jam istirahat untuk makan siang.

Anda dapat juga menentukan hari-hari di mana orang tsb. libur, seperti akhir minggu, sore, dan juga hari libur khusus misalnya untuk hari-hari libur Nasional. Ini semua dapat dimasukkan dari kalender proyek.

1. Dari menu **View**, klik **Gantt Chart**
2. Dari menu **Tools**, klik **Change Working Time**
3. Pilih satu tanggal di dalam kalender tsb.
    - Untuk mengubah satu hari dalam satu minggu untuk seluruh kalender, misalnya untuk selalu memiliki hari Jumat bekerja hanya dari jam 8.00

sampai jam 12.00, klik singkatan untuk hari tsb. pada kolom heading di kalender tsb. dan ganti waktu kerjanya.

- Untuk mengubah semua hari kerja, misalnya selalu hari kerja mulai dari Selasa sampai Jumat, dapat dilakukan dengan melakukan klik pada singkatan (misalnya T untuk Tuesday atau Selasa) untuk hari kerja pertama dari minggu tsb. Tekan tombol SHIFT, kemudian klik singkatan untuk hari terakhir yaitu Jumat (F untuk Friday), kemudian lakukan perubahan yang diinginkan.

4. Klik **Nonworking time** agar hari yang diinginkan menjadi libur, atau **Nondefault working time** untuk mengganti jam kerja.
5. Jika anda klik Nondefault working time, ketik waktu di dalam kotak yang disediakan untuk waktu mulai di kotak **From,** dan waktu berakhir di kotak **To**.
6. Jika anda telah selesai melakukan perubahan, klik **OK**.

MANUAL 2:
BAGAIMANA MEMASUKKAN DAN MENG-ORGANISASIKAN KEGIATAN-KEGIATAN?

Pertama, buat daftar langkah-langkah yang dibutuhkan untuk mencapai tujuan dari proyek anda, secara manual. Mulai dengan garis besarnya dulu, kemudian dari setiap item tsb. pecahkan menjadi kegiatan-kegiatan kecil yang akan menghasilkan suatu hasil. Tambahkan milestones (penanda pencapaian). Kemudian Akhirnya, cari dan masukkan estimasi lama waktu penyelesaian.

Setelah informasi kegiatan didapat, kemudian

dimasukkan dalam MP-tools, ciptakan suatu kerangka/ outline untuk membantu anda melihat struktur proyek.

2.1 Masukkan kegiatan-kegiatan dan lama waktunya

Sebuah proyek pada umumnya adalah suatu seri kegiatan yang berhubungan satu sama lain. Suatu kegiatan menyajikan banyaknya kerja dengan suatu hasil tertentu (deliverable). Kegiatan-kegiatan sebaiknya dipecah dalam lama waktu antara 1 hari sampai 2 minggu, untuk mempermudah monitor kemajuan yang telah dijalani.

Masukkan kegiatan di dalam urutan kapan mereka akan dikerjakan. Kemudian estimasikan berapa lama waktu yang dibutuhkan untuk menyelesaikan setiap kegiatan, dan masukkan estimasi lamanya tsb. di dalam duration. MS Project menggunakan duration ini untuk menghitung berapa banyak kerja yang perlu dilakukan untuk satu kegiatan.

Catatan: Jangan masukkan tanggal di dalam Start dan Finish field karena MS Project menghitung tanggal start dan finish berdasarkan durasi yang dimasukkan dan juga hubungannya antara satu kegiatan dengan yang lainnya.

1. Dari menu **View**, klik **Gantt Chart**
2. Di dalam field **Task Name**, masukkan nama kegiatan
3. Di dalam field **Duration**, masukkan lama waktu untuk setiap kegiatan. Jangan dihitung mengenai waktu di mana ada **nonworking time**. Singkatan-singkatan berikut ini dapat digunakan:
   a. Bulan = months = mo
   b. Minggu = weeks = w
   c. Hari = days = d
   d. Jam = hours = h

e. Menit = minutes = m
4. Tekan **Enter**

Catatan:

Secara default waktu adalah *1 day?*. Dan ini dapat diubah sesuai rencana kerja yang telah dibuat. Tanda tanya dibelakang menunjukkan ini adalah waktu yang diperkirakan. Bila anda yakin akan durasi suatu aktivitas maka tanda tanya ini dapat dibuang.

Ada pula yang namanya "elapsed"-time. Artinya adalah perhitungan berdasarkan *waktu sebenarnya* (24 jam/hari, 7 hari/minggu), bukan berdasarkan pada waktu kerja.

Tips: Anda juga dapat menulis suatu catatan mengenai suatu kegiatan. Di dalam field **Task Name**, pilih kegiatan yang dipilih, kemudian klik **Task Notes**. Ketik catatan anda, dan klik **OK**.

2.2 Menciptakan milestone (penanda pencapaian)

Sebuah milestone adalah kegiatan yang anda gunakan untuk mengidentifikasi suatu kejadian yang signifikan di dalam jadwal ada, seperti penyelesaian suatu fase yang utama. Ketika anda memasukkan suatu kegiatan dan menuliskan lama waktunya adalah 0, itu dijadikan sebagai milestone. Ada simbol berbentuk wajik hitam pada Gantt Chart yang menandai bahwa itu adalah suatu *milestone*.

1. Setelah menuliskan nama kegiatan, di dalam **duration**, ketik 0.
2. Tekan Enter

Selain dengan membuat suatu kegiatan menjadi sebuah milestone dengan memasukkan durasi 0, dapat

juga kegiatan-kegiatan yang tidak berdurasi 0 menjadi milestone. Caranya:

1. Klik kegiatan yang diinginkan menjadi milestone
2. Dari menu **Project**, klik **Task Information**
3. Klik **Advanced** tab, kemudian pilih checkbox **Mark task as milestone**

2.3 Membuat recurring task atau kegiatan yang dilakukan berulang-ulang

Recurring tasks adalah kegiatan-kegiatan yang berulang secara reguler, misalnya pertemuan mingguan. Recurring task dapat juga timbul harian, mingguan, bulanan, atau bahkan tahunan. Anda dapat menentukan lama setiap kegiatan tsb. dilaksanakan, kapan dilaksanakannya, dan seberapa lama, maupun berapa kali kegiatan tsb. harus dilaksanakan.

1. Di dalam field **Task Name**, klik baris di mana anda ingin recurring task muncul
2. Di dalam menu **Insert**, klik **Recurring Task**
3. Di dalam **Task Name** box, ketik nama kegiatan tsb.
4. Di dalam **Duration** box, ketik atau pilih durasi dari satu kali kegiatan tsb. berlangsung
5. Pilih pola recurrence, misalnya Daily untuk harian, Weekly untuk mingguan, Monthly untuk bulanan, dan Yearly untuk tahunan.
6. Di sebelah kanan dari Daily, Weekly, Monthly, atau Yearly, tentukan frekuensi kegiatan
7. Di dalam Range of recurrence, ketik tanggal start di dalam kotak Start dan kemudian pilih **End after** atau **End by**
    a. Jika End after yang dipilih, ketik jumlah berapa

kali kegiatan ingin dilakukan.

b. Jika End by yang dipilih, ketik tanggal kapan anda ingin recurring task tsb. berakhir.

Tips: untuk melihat semua kegiatan yang berulang tsb., klik tanda plus yang ada di sebelah recurring task.

2.4 Menyusun struktur kegiatan-kegiatan menjadi kerangka logis

Dengan menggunakan outline, anda akan dapat menyusun suatu hirarki kegiatan, sehingga akan lebih mudah mengaturnya. Untuk hal itu, dapat digunakan tombol-tombol outline (bila tidak aktif, dapat diaktifkan dari: **View → Toolbars → Formatting**). Tombol tombol ini adalah yang berupa panah ke kanan (indent), panah ke kiri (outdent), tanda plus (show subtasks), dan tanda minus (hide subtasks).

MANUAL 3:
KAPAN KEGIATAN AKAN DIMULAI DAN DIAKHIRI?

Setelah Anda menciptakan kegiatan dan juga menyusun *outline* dari daftar kegiatan anda, baru anda perlu untuk menghubungkan bagaimana suatu kegiatan berhubungan satu sama lain dan pada tanggal spesifik. Banyak sekali tipe hubungan kegiatan, yang mana disebut task dependencies. MS Project secara otomatis menentukan tanggal start dan finish untuk kegiatan-kegiatan yang berhubungan satu sama lain.

Keuntungan dari hubungan kegiatan-kegiatan ini adalah jika satu kegiatan berubah, kegiatan-kegiatan yang berhubungan akan secara otomatis dijadwal ulang.

Anda dapat menyempurnakan jadwal kegiatan dengan menggunakan batasan, overlap atau kegiatan yang ditunda, dan memecahkan kegiatan-kegiatan ketika pekerjaan yang dilakukan dihentikan untuk sementara.

3.1 Ciptakan hubungan antara kegiatan-kegiatan

Untuk menciptakan hubungan antara kegiatan, gunakan **task dependencies**. Pertama-tama, pilih kegiatan-kegiatan yang berhubungan, hubungkan, dan kemudian ganti dan sesuaikan ketergantungan jika diperlukan. Kegiatan yang waktu start dan finishnya tergantung yang lain merupakan successor, sementara successor adalah bergantung pada predecessornya. Contohnya, jika anda menghubungkan "Pasang jam dinding" dengan "Cat tembok kamar tidur", maka "Pasang jam dinding" adalah successor, sementara "Cat tembok kamar tidur" adalah predecessor.

Setelah semua kegiatan terhubung, perubahan pada tanggal predecessor akan mempengaruhi tanggal successor. MS Project pada dasarnya, by default, menciptakan hubungan **finish-to-start (FS)** Karena ini mungkin tidak selalu berlaku di setiap situasi, anda dapat menggantinya dengan **start-to-start (SS), finish-to-finish (FF)**, atau **start-to-finish (SF)** untuk membuat model proyek anda lebih realistik.

Catatan:

- FS = kegiatan "dari" harus selesai sebelum kegiatan "ke" boleh dimulai.
- FF = kegiatan "dari" harus selesai sebelum kegiatan "ke" boleh selesai (dapat pula selesai berbarengan).
- SS = kegiatan "dari" harus dimulai sebelum kegiatan

"ke" boleh dimulai (boleh mulai bersamaan).

- SF = kegiatan "dari" harus dimulai sebelum kegiatan "ke" boleh selesai, dengan kata lain mulainya kegiatan "dari" harus menunggu kegiatan "ke" selesai.
    1. Pada menu **View**, klik **Gantt Chart**
    2. Di dalam field Task Name, pilih 2 atau lebih kegiatan untuk dihubungkan.
    3. Pada menu **Edit**, Klik **Link Task (atau klik tombol berbentuk seperti rantai)**
    4. Untuk mengganti hubungan antara kegiatan, double click pada garis penghubung antara kegiatan-kegiatan yang ingin diganti.

Catatan: untuk menghilangkan link antara kegiatan, pilih kegiatan-kegiatan yang ingin diputus hubungannya, dan dari menu Edit, pilih Unlink Task.

3.2 Membuat kegiatan yang overlap atau menambahkan lag time (waktu tunggu) di antara kegiatan-kegiatan tsb.

Dengan MS Project, anda dapat membuat kegiatan-kegiatan overlap satu sama lain dengan memasukkan **lag time** (waktu penundaan) atau **lead time** (waktu percepatan). Lag dan lead time dapat juga dimasukkan dalam bentuk persentase.

1. Di dalam field **Task Name**, klik kegiatan yang ingin ditambahkan lead atau lag timenya, kemudian, pilih **Task information**.
2. Klik **Predecessor** tab
3. Di dalam kolom **Lag**, ketik berapa lama waktu penundaan yang diinginkan, sebagai durasi waktu, atau sebagai persentase dari predecessornya.
    a. Ketik **lead time** sebagai angka negatif (misalnya

–2d untuk lead time 2 hari) atau sebagai persentase.

b. Ketik **lag time** sebagai angka positif

4. Klik **OK**.

Tips: Untuk memasukkan lead dan lag time dengan cepat, dapat juga dimasukkan langsung dengan *double-click pada garis penghubung* pada gantt chart.

3.3 Set tanggal start dan finish yang spesifik untuk suatu kegiatan

Anda dapat menjadwalkan kegiatan-kegiatan dengan lebih efektif dengan menciptakan ketergantungan antar kegiatan, dan membiarkan MS Project menghitungnya untuk anda. Akan tetapi, anda juga dapat menentukan sendiri kapan anda ingin kegiatan-kegiatan dimulai atau diakhiri bilamana perlu.

Batasan kegiatan yang membuat ketergantungan kegiatan terhadap suatu tanggal yang spesifik disebut *inflexible constraints*. Yang paling tidak fleksibel adalah yang mana tanggal start atau finishnya anda tentukan. Karena MS Project memperhitungkan hal ini di dalam menghitung waktu penyelesaian proyek , gunakan batasan ini jika ada keterbatasan waktu penyelesaian.

1. Di dalam field **Task Name**, klik kegiatan yang ingin diset tanggal start dan finishnya, kemudian klik **Task Information** dari menu **Project**
2. Klik **Advanced** tab
3. Di dalam box **Constraint Type**, klik tipe batasan.
4. Ketik atau pilih tanggal di dalam constraint date box, kemudian klik **OK**.

Tipe contraints waktu dalam MS-Project:

- ***As Soon As Possible (ASAP)*** – default bila memasukkan aktivitas dengan "start date": memulai suatu pekerjaan secepat mungkin.
- ***As Late As Possible (ALAP)*** – default bila memasukkan aktivitas dengan "end date": memulai pekerjaan selambat mungkin tanpa mengurangi waktu kerja keseluruhan.
- ***Start No Earlier Than (SNET):*** memulai aktivitas pada atau setelah tanggal tertentu.
- ***Start No Later Than (SNLT):*** memulai aktivitas tidak lebih awal dari tanggal tanggal tertentu.
- ***Finish No Earlier Than (FNET):*** menyelesaikan suatu aktivitas pada atau setelah tanggal tertentu.
- ***Finish No Later Than (FNLT):*** menyelesaikan suatu aktivitas pada atau sebelum tanggal tertentu.
- ***Must Start On (MSO):*** harus memulai aktivitas pada tanggal tertentu.
- ***Must Finish On (MFO):*** harus menyelesaikan aktivitas pada tanggal tertentu.

Catatan:

o Jika anda memilih tanggal tertentu di dalam ***start field,*** berarti MS Project akan menentukan o batasan **Start No Earlier Than (SNET)** atau mulai tidak lebih terlambat dari…

o Jika anda menetapkan ***finish date*** , MS Project secara otomatis menentukan **Finish No Earlier Than (FNET)** atau berakhir tidak lebih awal dari …

3.4 Menambah deadline suatu kegiatan

Ketika Anda menentukan suatu deadline untuk suatu kegiatan, MS Project menunjukkan suatu indikator jika suatu kegiatan dijadwalkan untuk selesai setelah deadline. Menentukan suatu deadline tidak mempengaruhi bagaimana kegiatan-kegiatan dijadwalkan. Ini hanya cara MS Project untuk menginformasikan jika suatu kegiatan diselesaikan setelah deadline. Kemudian anda mempunyai kesempatan untuk melakukan penyesuaian agar jadwal dapat mencapai deadline yang ditentukan.

1. Di dalam menu **View**, pilih **Gantt Chart.**
2. Di dalam field **Task Name**, klik kegiatan yang ingin diset deadlinenya.
3. Klik **Task Information** dan **Advanced** Tab.
4. Di dalam Constraint Task, ketik atau pilih tanggal deadline di dalam deadline box, kemudian klik OK.

MANUAL 4:
BAGAIMANA MENENTUKAN SUATU KEGIATAN PADA SUMBERDAYA YANG DIMILIKI?

Kita perlu menentukan sumberdaya yang mana akan mengerjakan suatu kegiatan tertentu ketika anda ingin:

- Melihat kemajuan pekerjaan yang dilakukan oleh orang maupun perlengkapan yang ditentukan untuk suatu kegiatan, atau untuk memonitor materi-materi yang digunakan.
- Memiliki lebih banyak fleksibilitas di dalam menjadwalkan kegiatan
- Memonitor sumberdaya yang mendapat beban terlalu banyak atau terlalu sedikit.
- Mengawasi biaya dari sumberdaya

Jika resource information ini tidak digunakan, MS Project akan menghitung jadwal anda dengan berdasarkan duration dan dependencies saja.

4.1 Membuat daftar sumberdaya

Untuk membuat daftar sumberdaya, anda dapat menggunakan Resource Sheet di dalam MS Project. Sumberdaya dapat bervariasi dari orang, perlengkapan, maupun materi yang dibutuhkan untuk melaksanakan kegiatan.

1. Dari menu **View**, klik **Resource Sheet**
2. Dari menu **View**, klik ke **Table**, dan klik **Entry**
3. Di dalam field **Resource Name**, ketik nama dari sumberdaya
4. Untuk memasukkan sumberdaya-sumberdaya di dalam suatu grup, ketik nama grup di dalam **Group** field
5. Di dalam field **Type**, sebutkan tipe sumberdaya:
   a. Untuk work resource (orang atau perlengkapan), set resource type menjadi Work
   b. Untuk material resource (yang dikonsumsi sepanjang proyek), set resource type menjadi Material
6. Untuk setiap work resource, ketik jumlah unit sumberdaya yang tersedia untuk sumberdaya ini di dalam **Max unit** field, sebagai persentase. Misalnya, ketik 300% untuk mengindikasikan 3 full-time unit dari sumberdaya tertentu.
7. Untuk setiap material resource, ketik di dalam field **Material label**, unit pengukuran untuk unit tsb., misalnya ton.

4.2 Mengganti jadwal kerja sumberdaya

Waktu kerja dan waktu tidak kerja telah didefinisikan sebelumnya, yang mana by default sudah ditentukan. Jika seorang individu bekerja pada berbagai jadwal, atau jika anda membutuhkan perhitungan untuk adanya liburan atau waktu downtime perlengkapan, anda dapat memodifikasi kalender sumberdaya dari setiap individu.

1. Di dalam menu **View**, pilih **Resource Sheet**, kemudian pilih resource yang ingin diganti jadwalnya
2. Dari menu **Project**, klik **Resource Information**, kemudian klike **Working Time** tab
3. Dari kalender, pilih hari yang ingin diganti, kemudian untuk mengganti keseluruhan kalender, seperti sebelumnya, pilih singkatan yang ada di kolom teratas di kalender.
4. Kemudian klik Use default, Nonworking Time, atau Nondefault working time, sesuai yang diinginkan.
5. Ubah sesuai kebutuhan.

Tips: Jika suatu grup sumberdaya atau seseorang memiliki waktu kerja yang spesial yang terus menerus digunakan, anda dapat juga membuat kalender baru (new base calendar) dengan nama tersendiri untuk keperluan tsb. Caranya adalah:

1. Dari menu **Tools**, pilih **Change Working Time**
2. Klik **New**, dan ketik nama untuk kalender yang baru
3. Kemudian klik **Create** new base calendar, yang akan memulai dari kalender standar
4. Kemudian **View** pada **Resource Sheet,** di sana terdapat salah satu field yaitu **Base Calendar**, di mana kalender yang baru diciptakan dapat dipilih dan digunakan.

4.3 Mengatur sumberdaya untuk kegiatan-kegiatan

Anda dapat mengatur sumberdaya untuk suatu kegiatan, dan dapat mengubahnya kemudian dengan mudah sebagaimana diperlukan. Jika ada sumberdaya yang overload, atau melebihi kapasitasnya, MS Project akan memberikan tanda merah sebagai peringatan.

1. Dari menu **View**, pilih **Gantt Chart**
2. Dari field **Task Name**, pilih kegiatan yang ingin diberikan sumberdaya-nya, kemudian klik kanan **Task Information** → tab **Resources** atan Tools → Assign Resources (Alt + F10).
3. Di dalam field **Name**, klik sumberdaya yang ingin ditempatkan di kegiatan tsb.
4. Untuk sumberdaya dialokasikan secara part-time, ketik atau pilih persentase kurang dari 100% di dalam kolom unit untuk menunjukkan persentase waktu kerja yang ingin dialokasikan oleh sumberdaya tsb. untuk kegiatan tsb.
    a. Untuk menempatkan lebih dari satu sumberdaya, tekan tombol CTRL dan klik nama-nama sumberdaya
    b. Untuk menempatkan lebih dari 1 untuk sumberdaya yang sama, ketik atau pilih persentase lebih dari 100% di dalam kolom unit. Jika diperlukan, ketik nama dari sumberdaya.
5. Click Assign, kemudian Close.

4.4 Check dan edit resource assignments

Untuk melakukan pengecekan, dapat digunakan Resource Usage view. Dengan ini anda akan dapat menemukan berapa banyak waktu yang dibuhkan oleh

setiap sumberdaya untuk bekerja pada kegiatan yang spesifik dan melihat apakah sumberdaya tsb sudah overallocated (melebihi kapasitasnya).

1. Dari menu **View**, klik **Resource Usage.** Untuk melihat informasi yang berbeda mengenai resource assignment misalnya dari kerja dan biaya, point pada **Table** di menu **View**, dan kemudian klik tabel mana yang anda ingin lihat.
2. Di dalam kolom Resource Name, review pengalokasian kegiatan pada sumberdaya yang ada.
3. Jika perlu mengatur kembali penugasan dari satu orang ke yang lain, lakukan edit dengan memilih baris dari satu kegiatan untuk dipindahkan kepada sumberdaya yang lain.

Catatan: untuk mengubah *overallocated resources*, sehingga dapat menjadi benar kembali, dapat dilakukan secara otomatis, yaitu melalui Leveling Resources. Konsekuensinya adalah mengubah waktu kerja (penjadwalan dari sumberdaya tsb). MS Project dapat melakukannya secara otomatis dengan cara:

- View → Resource Sheet;
- View → Table: Entry;
- Tools → Resource Leveling (open dialog box);
- Pada field "Leveling Calculation" pilih manual;
- Pada drop-down menu "Look for overallocation" pilih Day-by-day;
- "Clear Leveling Values" check box harus tercentang;
- "Leveling range" pilih Entire Project;
- "Leveling order" pilih Standard;
- "Level Only Within Available Slack" tidak

tercentang, untuk memungkinkan pengunduran *end-date*;
- "Leveling can adjust …" dan "Leveling can create …" harus tercentang;
- Klik pada "Level Now";
- Konfirmasi pada "Entire Pool" dan klik OK".

Untuk melihat perbedaan antara penjadwalan sumberdaya sebelum dan sesudah "leveling", dapat dilakukan dengan cara: View → More Views → Leveling Gantt → Apply, bagian yang (default) hijau adalah sebelum dan yang biru adalah sesudah leveling.

MANUAL 5:
BAGAIMANA CARA MENGHITUNG BIAYA?

Dengan memasukkan tarif untuk pekerjaan dari sumberdaya akan memudahkan anda untuk melihat apakah Anda masih berada di dalam budget yang telah ditetapkan. Anda dapat memilih apakah Anda akan menumpukkan biaya sampai waktu tertentu, memasukkan persekali pakai, menentukan tarif overtime, atau juga merencanakan kenaikan tarif.

5.1 Menentukan biaya suatu sumberdaya

MS Project dapat menghitung tarif baik untuk manusia ataupun materi, sehingga anda dapat mengatur proyek dengan lebih akurat. Anda dapat menggunakan standard rates, overtime rates, atau per-use rates sesuai dengan kebutuhan.

1. Dari menu **View**, pilih **Resource Sheet**
2. Dari menu **View**, pilih **Table**, dan klik **Entry**

3. Di dalam field **Resource Name**, pilih suatu sumberdaya atau ketik nama baru untuk suatu sumberdaya
4. Tentukan dan sesuaikan apakan sumberdaya tsb. merupakan work atau material resources
5. Untuk work resource, dalam Std. Rate, Ovt. Rate, atau Cost/Use fields, ketik tarifnya.

Untuk material resource, di dalam Material Label field, ketik unit pengukuran untuk sumbedaya materi (misalnya ton) dan di dalam Std. Rate atau Cost/Use fields, ketik tarifnya.

5.2 Tentukan kapan biaya akan terkumpul

Di dalam MS Project, biaya sumberdaya dihitung secara **prorated** by default, yaitu sejalan dengan persentase penyelesaian kerja yang dilakukan. Hal ini dapat diganti sebagaimana perlu, dengan menggunakan **accrual** method, sehingga pembayaran/biaya yang dikeluarkan baru berlaku pada **awal** atau **akhir** dari kegiatan tsb.

1. Dari menu View, klik Resource Sheet
2. Dari menu View, pilih Table, kemudian klik Entry
3. Di dalam field Accrue At, pilih metode accrual yang ingin digunakan

5. 3 Lihat biaya dari sumberdaya tertentu

Setelah dipilih suatu tarif, anda mungkin ingin mereview biaya total untuk meyakinkan apakah biaya tsb. sesuai dengan harapan anda. Jika total biaya dari suatu sumberdaya tidak sesuai dengan anggaran Anda, anda mungkin ingin memeriksanya dan mencari di bagian mana biaya bisa dikurangi.

1. Untuk melihat biaya suatu kegiatan, dari menu **View**, klik **More Views**, kemudian klik **Task Sheet**. Untuk melihat biaya suatu sumberdaya, dari menu **View**, klik **Resource Sheet**
2. Dari menu **View**, pilih **Table**, dan klik **Cost**.

5.4 Melihat biaya keseluruhan proyek

Anda dapat melihat biaya *baseline*, aktual dan remaining untuk melihat apakah anda masih berada di dalam *budget* yang telah ditentukan. Perhitungan ini selalu diupdate setiap dilakukan perubahan.

1. Dari menu **Project**, klik **Project Information**
2. Klik **Statistics**
3. Dari situ dapat dilihat berapa total biaya-biaya tsb.

MANUAL 6: BAGAIMANA ANDA MELIHAT JADWAL DAN DETAILNYA?

Setelah memasukkan semua data yang mendasar, sekarang waktunya me-review. Apakah anda akan memenuhi deadline? Jika tidak, perlu dilihat yang mana yang merupakan milestone dan pastikan lagi bahwa jadwal sudah disusun dengan efisien.

Pertama, lihat pada gambaran keseluruhan: dari tanggal start dan finish, kemudian pada critical path yaitu garis kegiatan di mana urutan tsb. yang menjadi penentu penyelesaian proyek. Kemudian periksa detailnya.

6.1 Lihat keseluruhan proyek pada screen monitor

Anda dapat mendapatkan gambaran keseluruhan proyek dari start ke finish dan melihat fase-fase utama yang akan muncul dengan melakukan zoom in atau zoom out dari Gantt Chart

1. Dari menu **View**, klik **Gantt Chart**
2. Dari menu **View**, klik **Zoom**, kemudian **Entire Project**, dan klik **OK**

6.2 Memeriksa tanggal start dan finish proyek

Anda dapat melihat informasi proyek yang penting seperti finish date, untuk melihat apakah proyek sudah sesuai dengan harapan.

1. Dari menu **Project**, klik **Project Information**, kemudian klik **Statistics**
2. Di situ terlihat tanggalnya

6.3 Identifikasi critical path

Critical path adalah suatu seri kegiatan yang harus diselesaikan pada waktunya supaya proyek dapat diselesaikan sesuai jadwal atau bisa dikatakan juga rantai kegiatan yang terpanjang.. Kebanyakan kegiatan dari proyek yang biasa memiliki beberapa **slack** (kesenggangan waktu) sehingga dapat ditunda sedikit tanpa mempengaruhi tanggal finish proyek. Namun, untuk *critical path*, hal ini tidak bisa dilakukan. Pada waktu modifikasi dilakukan, hati-hati perhatikan apakah kegiatan-kegiatan ini terpengaruh.

1. Dari menu **View**, pilih **Gantt Chart**
2. Klik kanan pada tanggal, kemudian pilih **Gantt Chart**

Wizzard

3. Ikuti instruksi
4. Selain itu juga bisa langsung dilihat pada **Network Diagram**, di mana critical path adalah yang diwarnai merah.

MANUAL 7: BAGAIMANA ANDA MEREKAM (SAVE) RENCANA ANDA?

Setelah Anda memasukkan kegiatan, sumberdaya, dan informasi biaya untuk proyek anda, anda dapat merekam suatu potret dari rencana asli/original anda, yang dinamakan baseline. Untuk merekam suatu *checkpoint* untuk kemajuan yang sesungguhnya dari proyek tsb., anda dapat merekam interim plan dan membandingkannya dengan baseline plan.

7.1 Merekam rencana baseline

Untuk merekam baseline plan, anda dapat melakukan hal berikut:

1. Dari menu Tools, point pada Tracking, dan kemudian klik Save Baseline
2. Klik Entire project untuk merekam project baseline. Klik Selected tasks untuk menambah kegiatan baru pada baseline yang ada
3. Klik OK

7.2 Save an interim plan

Setelah Anda merekam suatu baseline untuk informasi proyek anda, anda dapat merekam sampai 10 interim plan sebagai checkpoint selama berlangsungnya

proyek.

1. Dari menu **Tools**, point ke **Tracking**, kemudian klik **Save Baseline**
2. Klik **Save** interim plan
3. Di dalam **Copy** box, klik nama interim plan yang sekarang
4. Di dalam **Into** box, klik nama untuk interim plan berikutnya, atau tentukan nama yang baru
5. Klik **Entire project** untuk merekam suatu interim plan untuk keseluruhan proyek. Klik **Selected tasks** untuk merekam hanya sebagian saja dari jadwal
6. Klik OK

MANUAL 8: BAGAIMANA MEMONITOR PELAKSANAAN KEGIATAN YANG SESUNGGUHNYA?

Sekali Anda menetapkan proyek Anda dan pekerjaan telah dimulai, Anda dapat memonitor pelaksanaan sesungguhnya, yaitu *actual start*, *actual finish dates*, persentase penyelesaian, dan pekerjaan sesungguhnya. Dengan mengamati apa yang terjadi, akan terlihat bagaimana perubahan-perubahan mempengaruhi kegiatan-kegiatan lainnya, yang juga dapat mempengaruhi penyelesaian proyek.

8.1 Memeriksa apakah kegiatan dilaksanakan sesuai rencana

Untuk menjaga pemenuhan jadual, yakinkan bahwa kegiatan mulai dilaksanakan dan diselesaikan pada waktunya. *Tracking Gantt view* dapat menolong untuk mencari titik permasalahan, kegiatan yang bervariasi

dibandingkan baseline plan. Dengan demikian anda dapat menyesuaikan sumberdaya, task dependencies, atau menghapus beberapa kegiatan untuk mengejar deadlines.

Tracking Gantt view memasangkan jadual yang sesungguhnya dengan rencana awal untuk setiap kegiatan. Setiap kali anda memasukkan pelaksanaan sesungguhnya, akan terlihat bahwa ada bar yang bergeser menunjukkan bagaimana pelaksanaan dibandingkan dengan baselinenya.

1. Dari menu **View**, klik **Tracking Gantt**
2. Untuk melihat field **Variance**, dari menu **View**, point ke **Table**, dan klik **Variance**
3. Jika diperlukan, klik **TAB** untuk melihat Variance fields
4. Dari menu **View**, point ke **Toolbars**, dan kemudian klik **Tracking**
5. Update perkembangan dari kegiatan-kegiatan di dalam proyek anda
    a. Jika suatu kegiatan sudah dimulai seperti jadwal, klik kegiatan tsb., kemudian klik Update as scheduled
    b. Jika suatu kegiatan tidak sesuai jadwal, di point berikutnya akan diterangkan bagaimana menanganinya.

8.2 Memasukkan tanggal mulai dan selesai yang sesungguhnya (actual) untuk suatu kegiatan

Kegiatan yang tidak sesuai jadwal perlu direkam dalam MS Project untuk perbandingan, dan bagaimana efek terhadap keseluruhan proyek yang telah direncanakan.

1. Dari menu **View**, klik **Gantt Chart**
2. Dari menu **View**, point ke **Toolbars**, kemudian

klik **Tracking** jika belum dipilih

3. Di dalam field **Task Name**, pilih kegiatan yang ingin diupdate.
4. Klik **Tools → Tracking → Update Tasks**. Dapat pula langsung double-click di field yang bersangkutan. Atau:
5. Di dalam **Actual**, ketik atau pilih tanggal di dalam **Start** atau **Finish** box. Jika anda memasukkan suatu tanggal finish, yakinkan bahwa kegiatan tsb. sudah diselesaikan 100%, karena MS Project mengasumsikan demikian.

Catatan: Untuk memasukkan actual duration, langkah yang sama dapat dilakukan.

8.3 Mengupdate perkembangan kegiatan dalam persentase

Untuk memudahkan, perkembangan kegiatan dapat juga dimasukkan dalam bentuk persentase.

1. Dari menu **View**, klik **Gantt Chart**
2. Di field **Task Name**, klik kegiatan yang ingin diupdate
3. Klik **Task Information**, kemudian klik **General** tab
4. Di dalam **Percent Complete** box, ketik nomor persentasenya
5. Klik OK.

Tip You can use the buttons on the Tracking toolbar to update progress on a task and to perform other tracking activities. To view the Tracking toolbar, point to Toolbars on the View menu, and then click Tracking.

8.4 Mengupdate pekerjaan yang sesungguhnya dengan periode waktu

Anda juga dapat melakukan tracking dari pekerjaan aktual dengan menggunakan timephased field di dalam MS Project. Tracking dengan cara ini menolong anda untuk mengupdate secara periodik karena anda dapat mengenter informasi dari hari tertentu di dalam jadwal Anda.

1. Dari menu **View**, klik **Task Usage**
2. Dari menu **Format**, point ke dalam **Details**, kemudian klik **Actual Work**
3. Di dalam bagian timephased dari view, di dalam Actual Work field, ketik actual work untuk setiap sumberdaya yang telah ditempatkan.

8.5 Lihat apakah kegiatan menggunakan lebih atau kurang sumberdayanya

Varians di dalam jadwal anda dapat bagus atau buruk, tergantung dari tipe dan keseriusan varians. Suatu kegiatan dengan pekerjaan yang lebih sedikit dari rencananya, misalnya biasanya adalah berita bagus, tapi ini juga bisa berarti bahwa sumberdaya anda tidak dialokasikan dengan efisien.

1. Dari menu **View**, klik **Gantt Chart**
2. Dari menu **View**, point ke **Table**, dan klik **Work**
3. Bandingkan nilai di dalam **Work**, **Baseline** dan **Actual** fields

MANUAL 9: BAGAIMANA MEMBANDINGKAN BIAYA SESUNGGUHNYA DENGAN ANGGARAN?

Anda mungkin ingin mengetahui berapa besar biaya yang telah dikeluarkan dan sebagaimana besar penyimpangan yang terjadi. Dengan mengetahui hal ini, anda akan dapat melakukan perbaikan dan penyesuaian selama masih bisa di dalam pelaksanaan proyek.

9.1 Masukkan biaya yang sesungguhnya secara manual

MS Project secara otomatis mengupdate biaya aktual dari perkembangan kegiatan berdasarkan metode accrual ayng dipilih dan juga tarif dari sumberdaya. Untuk memasukkan keadaan yang sesungguhnya, Anda dapat menginput secara manual. Untuk mengupdate biaya secara manual, Anda harus mematikan dulu *automatic updating* dari *actual cost*.

1. Dari menu **Tools**, pilih **Options**, kemudian pilih **Calculation** tab.
2. Hapus **Actual costs are always calculated by Microsoft Project** check box
3. Klik OK
4. Dari menu **View**, klik **Task Usage**
5. Dari menu **View**, point ke **Table**, kemudian klik **Tracking**
6. Jika diperlukan, press TAB untuk melihat Act. Cost field
7. Di dalam Act. Cost field, ketik actual cost untuk kegiatan yang sedang diupdate

9.2 Periksa apakah biaya yang terjadi lebih besar dari anggaran

MS Project menghitung biaya dari setiap kerja yang dilakukan oleh sumberdaya, total biaya untuk setiap kegiatan dan sumberdaya, dan juga total biaya proyek. Semua biaya-biaya ini dipertimbangkan sebagai biaya terjadwal atau projected cost, yang direfleksikan di dalam gambaran yang paling akhir dari perkembangan proyek.

1. Dari menu **View**, klik **Gantt Chart**
2. Dari menu **View**, point ke **Table**, kemudian **Cost**
3. Bandingkan nilai dari **Total Cost** dan **Baseline** fields
4. Untuk varians, lihat dari nilai di dalam **Variance** field

9.3 Melihat biaya total proyek

Anda dapat melihat biaya yang paling terakhir, baseline, atau aktual dan yang tersisa untuk melihat apakah anda masih di dalam budget yang ditetapkan sebelumnya. Biaya-biaya ini diupdate setiap kali MS Project menghitung proyek Anda.

1. Dari Project menu, klik Project Information
2. Klik Statistics – semuanya terlihat: current, baseline, dan remaining cost

MANUAL 10:
MENGKOORDINASIKAN BEBERAPA PROYEK SEKALIGUS

MS Project juga memiliki kemampuan untuk menggabungkan beberapa proyek menjadi satu proyek, untuk mengkoordinasikan jadwal dan juga sumberdaya yang ada.

1. Buka **New** file baru untuk dijadikan master project
2. Dari menu **View**, klik **Gantt Chart**
3. Di dalam field **Task Name**, klik baris di mana satu proyek ingin di-insert.
4. Dari menu **Insert**, pilih **Project**
5. Cari file project yang sudah dibuat oleh anda
6. Klik file yang diinginkan, dan klik **Insert**
7. Setiap kali anda save master project, file-file project yang telah diinsert akan diupdate. Jika anda tidak menginginkan file aslinya diupdate dengan perubahan-perubahan tsb., hapus box **Link to Project.** Dari Task Information pilih Advanced tab. Kemudian di Inserted Project Information, un-checked Link to project.

Catatan

- *Subprojects* diperlakukan seperti rangkuman kegiatan-kegiatan di dalam master project. Sama seperti kegiatan-kegiatan yang dapat disusun dalam suatu outline, demikian juga dengan proyek.
- Ketika mengkonsolidasikan proyek-proyek menjadi satu master project, sumberdaya-sumberdaya tetap ditinggalkan di dalam file proyek individual. Anda tidak dapat mengatur satu sumberdaya di satu proyek ke yang lainnya. Tapi sumberdaya-sumberdaya tsb., dapat digabungkan ke dalam suatu **shared resource pool** (Tools → resources Sharing → Share Resources). Ini dapat dilihat dari Help Index.

TAMBAHAN 1:
WBS (WORK BREAKDOWN STRUCTURE)

Ms Project memudahkan Anda untuk memberikan kode bagi setiap kegiatan sesuai dengan struktur WBSnya. Caranya adalah sbb:

1. Dari menu **Project**, pilih **WBS**, kemudian **Define Code**
2. Untuk membedakan proyek yang satu dari yang lain jika ada penggabungan beberapa proyek, anda dapat menggunakan project code prefix, ketikkan sebuah prefiks (misalnya huruf tertentu sebagai pembeda) di dalam **Project Code Prefix** box.
3. Untuk menentukan kode untuk kegiatan-kegiatan di level pertama, klik baris pertama di dalam kolom **Sequence**, kemudian klik panahnya, dan klik tipe karakter yang anda inginkan.
    a. Klik Numbers (ordered) untuk menunjukkan kode WBS dengan nomor.
    b. Klik Uppercase Letters (ordered) untuk menggunakan kode WBS dengan huruf besar misalnya A, B, C, dst.
    c. Klik Lowercase Letters (ordered) untuk menggunakan kode WBS dengan huruf kecil misalnya a, b,c, dst.
    d. Klik Characters (unordered) untuk menunjukkan kombinasi nomor dan huruf besar/kecil, contohnya Arch1, Const1, Insp1, dst. Ini yang paling fleksibel karena kodenya akan muncul setelah anda pertama mengetikkan.
4. Di kolom **Length**, ketikkan atau pilih nomor untuk menentukan maksimum panjang karakter untuk kode. Maksimum yang bisa digunakan adalah 255.

5. Di kolom **Separator**, klik baris pertama dan ketik atau pilih karakter untuk pemisah.

TAMBAHAN 2:
TASK DEPENDENCIES

MS Project selalu mempergunakan hubungan finish-to-start pada awalnya, jika anda tidak mengubah hubungan ini, berarti diasumsikan bahwa setelah satu kegiatan 100% baru kegiatan yang selanjutnya bisa dilakukan. Jika tidak demikian halnya, Ms Project juga dapat memfasilitasikan hubungan Start-to-start (SS), finish-to-finish (FF), dan start-to-finish (SF). Caranya adalah sbb:

1. Setelah satu kegiatan dibuat dan didefinisikan lamanya, di kolom predecessor dapat dibuat hubungan dengan kegiatan yang lainnya.
2. ***Tanpa*** tambahan perintah, jika dituliskan kode aktivitas yang mendahuluinya, berarti hubungannya adalah ***finish-to-start.***
3. Jika ingin diubah, maka harus dibuat dengan format berikut: kode aktivitas (hubungan) + satuan waktu/ persen. Contohnya: jika kegiatan nomor 2 dilakukan setelah kegiatan nomor satu dikerjakan sebanyak 50% (berarti hubungan start-to-start), maka di kolom predecessor kegiatan nomor 2 perlu dituliskan: 1(SS)+50%
4. 50% di contoh no. 3 juga dapat diganti dengan jumlah hari, minggu atau bulan, dst. Misalnya 1(SS)+2months
5. Hal ini dapat diadaptasikan baik untuk FF maupun SF.

Lampiran B

FEATURE ARTICLE
Managing Web Projects: Recipes for Success

by Geoff Hewson, Software Productivity Center Inc.

Developing web applications shares many similarities with conventional software development. However, the business realities of web development present the following significant challenges that conventional methods don't readily solve:

- Delivery timelines are slashed due to "time-to-market" pressures.
- Requirements are often vague initially, and only become clearer over time.
- Internet technology is evolving rapidly.
- The user interface is critical - your competitor is only one click away.
- Web development teams meld together a wide range of skills and cultures: marketing, artists, software developers, content authors, and managers. These people all have different mindsets and rely on different styles of communication.
- Web development teams are often geographically dispersed and include subcontractors.

Conventional software development methods (like the Software Engineering Institute's Capability Maturity Model for Software) are primarily focused on the needs of large- scale development projects that typically involving large teams and long development cycles. Controlling these types of projects is usually very document and

process intensive, which simply doesn't work in the rapidly moving, dynamic world of web development. Instead, the web project manager needs to organize and manage a project so that it emphasizes delivery and is flexible to change, yet at the same time keeps the team focused on the immediate work at hand. This is a challenge. Here are four recipes for success:

Identify Critical Functionality

While being quick to market in a particular e-business space is usually the critical success factor for an organization, understand that you don't have to implement 100% of the desired functionality in the first release. Entering the market quickly allows you to remain competitive, build brand awareness and obtain a market presence. Market research shows clearly that your web application doesn't have to have all its bells and whistles when it goes live, as long as it provides the important business functionality users are looking for.

What you need to do to implement this strategy is to inventory the business functions you want to provide through your web application, sort them in order of business importance, and determine the minimum set of functions you can afford to go live with. This allows you to plan the evolution of your web application over time, delivering sub-sets of this functionality in a series of development projects and enhancements, starting with the highest level of priority and moving downward.

This is a surprisingly powerful strategy, as it makes it much easier to stomach the de-scoping of your development project because you already have a road map that shows you where and when new functionality is to be

introduced.

Building a Web Site Is an Evolutionary Process

Building a new web site/application, or making a major update to an existing site/application, is best approached as an evolution of ideas, rather than the more straightforward "analyze -> build -> test -> implement" approach of conventional software development. There are two keys for succeeding at this evolutionary strategy.

The first is to get your project into "integration mode" as quickly as possible. This is the point where most of the difficult technical issues and strategic uncertainties have been resolved, and the project team is focused on delivering the "right stuff" in increments. Each increment builds and stabilizes new business functionality on top of an operational prototype of your web application.

The second key is the approach to getting your project into integration mode. You need a clear enough definition of the web application you're going to develop in order to move forward to construction. In many respects this is a miniature version of planning the overall evolution of your web application. In this case, you aim to get as complete an architectural picture of the web application/site as possible. This tells you the software components that need to be developed, the content pages that need to be written, and the User Interface (UI) look and feel. Typically you would develop an exploratory prototype to demonstrate the appearance of the UI and the connectivity between key systems. Tying these elements back to the prioritized list of business functions should allow you to get customer and management approval.

At this point, integration mode kicks in. Functionality

and content are assigned to each increment of development in order of decreasing business priority and technical risk. If the business climate changes, this prioritized, iterative approach allows you to change tack mid-stream by introducing new or changed requirements in additional iterations, at the appropriate level of priority. Of course, adding new work to your project is usually done at the expense of the lower priority work you had planned to do in later stages of your project, which leads us to...

Aggressive Scope Control

Aggressive Scope Control is the "slash and burn" approach to project management. Given the tight time constraints of most web projects, there always comes a time when you have to trim scope or deliver late. Delivering late is rarely an option, and the short development timelines limit the effectiveness of adding resources to your project, as it takes too long to bring new team members up to speed. Your only realistic option is to be aggressive about cutting back on the lower priority functionality and content of your web application to bring the project back on target.

Human nature makes us try to avoid cutting scope for fear that the functionality will be lost forever. This is where the previous two strategies help. Having a long-term plan for the evolution of your web application allows you to find a place where de-scoped functionality can be scheduled for later delivery.

Be sure, however, to have a clear understanding of the absolute minimum set of business functions the web application must provide for it to be viable. You don't want to cut so much functionality that your web application becomes inoperable. If you're facing this, all you may be

able to do is admit that your delivery target is unrealistic and "grin and bear it."

Ruthless Execution

The last strategy I want to discuss helps you maintain project team focus. Although web applications are often developed in a highly changeable environment, it's important to avoid spinning your wheels unnecessarily by dealing with issues that are of lower priority than the immediate work at hand.

The strategy of Ruthless Execution allows your project to remain open to change, while at the same time keeps the team's work efforts focused on the highest priority work at all times. As changes to the project are proposed (new or changing requirements, new deadlines etc), you capture them immediately in a change request database (using a software tool or a simple list in a spreadsheet).

All potential changes are prioritized relative to one another, and to the current development plan. Changes are introduced into the project only if they are of higher priority than currently planned work. If this occurs, re-evaluate the revised schedule and immediately apply aggressive scope control to re-align the project with your delivery timeline.

While change requests are being captured and reviewed, the project team continues to focus on the work of the current iteration of development. Wherever possible, only apply changes to later project iterations, leaving your current iteration intact. If you've done a good job of business prioritization, this will be feasible more often than not.

Conclusion

There you have it. Four straightforward strategies to help you cope with some of the uncertainties and fluidity of work hinges on being able to keep an objective view of the true business priorities for your web application, and aligning your initial development plan, scope control and change decisions to support those priorities. You won't be able to eliminate the dynamic, high-speed, high-change nature of a web project, but you should be able to bring a little order to the chaos.

Resources

For Geoff Hewson's biography, see the *E-ssentials!* Hall of Fame.

For more resources on Project Management visit SPC's online Resource Center.

Lampiran C

Prosedur *crashing* (lihat bab 6 untuk keterangan mengenai crashing) selengkapnya adalah sebagai berikut: