MAKALAH REKAYASA PERANGKAT LUNAK
SIKLUS HIDUP PERANGKAT LUNAK

NAMA : RANI JUITA
NIM : 41813120165
DOSEN : WACHYU HARI HAJI. S.Kom.MM

JURUSAN SISTEM INFORMASI
FAKULTAS ILMU KOMPUTER
UNIVERSITAS MERCU BUANA
JAKARTA
2015

PENGEMBANGAN PERANGKAT LUNAK

Pengembangan perangkat lunak (*Software development*) merupakan salah satu dari tahap rancangan system rinci/detail dari Siklus Hidup Pengembangan Sistem (*Software Development Life Cycle* atau *SDLC).*

Tim proyek system mungkin mulai mencari paket perangkat lunak komersial yang sesuai atau mendukung spesifikasi rancangan system dan berjalan pada rancangan arsitektur komputernya. Paket perangkat lunak komersial secara luas tersedia untuk aplikasi fungsi spesifik dan aplikasi bisnis yang telah ditetapkan secara baku.

Tetapi untuk rancangan sistem yang terkait dengan kebutuhan khusus atau unik (memenuhi keperluan pemakai dan spesifikasi rancangan sistem) maka paket perangkat lunak komersial mungkin tidak sesuai atau mendukung kebutuhan pemakai secara langsung. Perangkat lunak yang diharapkan untuk mendukung rancangan sistem tersebut harus dibuat sendiri dari awal (*scratch*)

Sumber Perangkat Lunak Aplikasi

A. Perangkat Lunak Komersial dari Vendor
B. Perangkat Lunak Pesanan (*customized software*) dikembangkan secara in-house atau oleh kontraktor pemrograman independent

A. Perangkat Lunak Komersial dari Vendor

Paket (*off-the-self*) yang tersedia bisa diterapkan dalam berbagai kebutuhan bisnis. Beberapa paket bersifat generik dan multifungsional yang memungkinkan para pemakai memprogram sofware tersebut untuk kebutuhannya sendiri. Paket-paket tersebut mengotomisasi fungsi-fungsi bisnis dasar yang umumnya tidak terlalu bervariasi dari satu organisasi dengan organisasi lain. Contoh jenis paket adalah spreadsheet dan DBMS.

Keuntungan/kelebihan dari Perangkat Lunak Komersial :

1. Implementasi yang cepat

Software tersebut bersifat siap, teruji, dan terdokumentasi. Paket yang dibeli biasanya pengimplementasiannya jauh lebih cepat dari pada mengembangkan program yang sama secara in-house atau menyuruh kontraktor independen untuk mengembangkannya sehingga secara potensial membantu memecahkan *backlog* (penimbunan pekerjaan yang belum selesai).

2. Penghematan Biaya

Satu paket perangkat lunak komersial bisa dijual kepada banyak organisasi sehingga biaya pengembangan ditanggung oleh banyak pemakai, dan biaya total suatu paket akan lebih murah dari pada program pesanan yang sama.

3. Estimasi biaya dan waktu

Biaya atau harga paket komersial telah diketahui, dan tanggal pengimplementasian-nya mudah diestimasi. Sebaliknya program pesanan biasanya cenderung melampaui estimasi waktu dan biaya.

4. Reliabilitas

Sebelum diterbitkan di pasaran umum, paket perangkat lunak komersial pasti telah diuji secara teliti. Melalui penggunaan yang ekstensif oleh sejumlah organisasi, segala kesalahan yang dijumpai telah dideteksi dan dikoreksi sehingga peluang kesalahannya lebih sedikit.

Kerugian/kelemahan :

1. **Kesesuaian Rancangan sistem yang tidak baik**

Paket software komersial dibuat untuk berbagai organisasi, dan tidak untuk organisasi tertentu maka paket ini mungkin mempunyai beberapa fungsi yang tidak diperlukan atau mungkin tidak mempunyai fungsi yang diperlukan sehingga paket tersebut harus dimodifikasi. Jika vendor tidak membuat kode sumber (*source code*) yang bisa digunakan untuk penyesuaian dan tidak menyediakan layanan penyesuaian maka rancangan sistem mungkin harus diubah agar sesuai dengan paket tersebut. Jika hal ini terjadi sebaiknya mengembangkan program secara in-house agar programnya bisa memenuhi spesifikasi rancangan sistem yang tepat.

2. **Ketergantungan Vendor**

Jika organisasi memerlukan perubahan paketnya maka organisasi akan tergantung pada vendor dalam perolehan dukungannya, dan jika vendor telah tiada maka organisasi akan kesulitan mencari dukungannya.

3. **Biaya tidak langsung dari kerusakan SDLC**

Seringkali apa yang ingin dicapai, manajemen tidak melaksanakan SDLC menyeluruh atau mungkin melewati tahap SDLC, dan secara langsung menuju ke paket perangkat lunak komersial Strategi ini seringkali mengakibatkan paket perangkat lunak komersial tidak berjalan sesuai yang diharapkan dan masalah sistem serta organisasional yang terjadi sebelum implementasi paket tersebut tetap muncul sehingga menimbulkan kesulitan atau harus dibayar kemudian yaitu adanya peningkatan biaya implementasi, operasi, dan pemeliharaan.

Menyiapkan permohonan untuk proposal berorientasi kinerja

Terkait dengan pemrolehan (akuisisi) perangkat lunak komersial maka perlu membuat atau **menyiapkan Permohonan Proposal** (*Request For Proposal* atau *RFP*) berorientasi kinerja untuk menyeleksi vendor dan paket perangkat lunak komersial yang tepat. Faktor-faktor evaluasi **mencakup pemenuhan spesifikasi rancangan detail untuk output, input, proses, dan database** serta cocok dengan **batasan waktu dan biaya**nya, juga penggunaan **benchmark** yang mensimulasi kebutuhan sistem baru (bentuk ***prototyping***) harus diterapkan pada setiap paket dari vendor.

Penilaian paket

Setiap paket dari vendor harus dinilai. Penilaian tersebut meliputi :

a. Sebagian penilaian dari *benchmark* (tanda untuk menentukan tingginya suatu nama), dan penilaian lain dari sejumlah publikasi yang didasarkan pada survei dari sejumlah besar pengguna paket tersebut.
b. Kinerja pengoperasian (*operating performance*)
   Penilaian dari benchmark yang digunakan untuk mengukur hal-hal seperti transaksi perdetik (*transaction per second*) dan waktu respon (*response time*).
c. Dokumentasi
   Penilaian ini mencerminkan kuantitas dan kualitas prosedur tertulis, prosedur *online*, pedoman *quick start, online tutorial*, dan fasilitas *help*.
d. Mudah dipelajari

Penilain ini tergantung pada interface pemakai dan rancangan intuitif dari paket tersebut. Paket harus bisa dipelajari oleh rata-rata pemakai.

e. Mudah digunakan
   Menu yang mudah diikuti dan perintah yang jelas membantu kemudahan penggunaan.
f. Pengendalian dan penanganan kesalahan.
   Untuk menjaga kesalahan input, paket software harus menyediakan pencegahan kesalahan, pendeteksian kesalahan dan perbaikan kesalahan, serta menuliskan kesalahan ke file kesalahan.
g. Dukungan (*support)*.
   Menyediakan dukungan kebijakan dan teknis. Dukungan kebijakan mencakup jalur toll-free, garansi, dan pelatihan. Dukungan teknis menyediakan teknisi dan yang berpengalaman.

Menyeleksi paket

Menentukan paket software dari vendor yang menawarkan manfaat terbesar dengan biaya/harga termurah. Metode untuk menentukan angka penilaian total terlihat padaTabel 1. Bobot relatif ditentukan ke setiap faktor kinerja umum yang didasarkan pada kepentingan relatifnya.Base atau nilai dasarnya adalah 100. Penilaian setiap faktor kinerja 1 s/d 10 (1=jelek dan 10 = sangat bagus).Skor adalah bobot dikalikan penilaian. Setiap skor yang dihasilkan dijumlahkan yang merupakan angka penilaian total untuk setiap vendor.

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

Tabel 1. Penilaian Kinerja Umum

Misalkan biaya atau harga paket vendor A adalah $22.700 dan paket vendor B adalah $27.690. Paket mana yang harus dipilih ? Jawabannya ditentukan dengan membagi angka biaya total dengan skor total untuk memperoleh biaya perangka penilaian. Vendor A mempunyai penilaian= $22.700/620= $37, sedangkan Vendor B= $27.690/710= $39, tampak terlihat pada gambar 1.2

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

Tabel 2. Biaya per angka penilaian

Vendor A mempunyai penilaian lebih rendah, namun biaya per angka penilaiannya sebesar $37 menjadi pilihan biaya atau manfaat yang lebih baik dari pada vendor B.

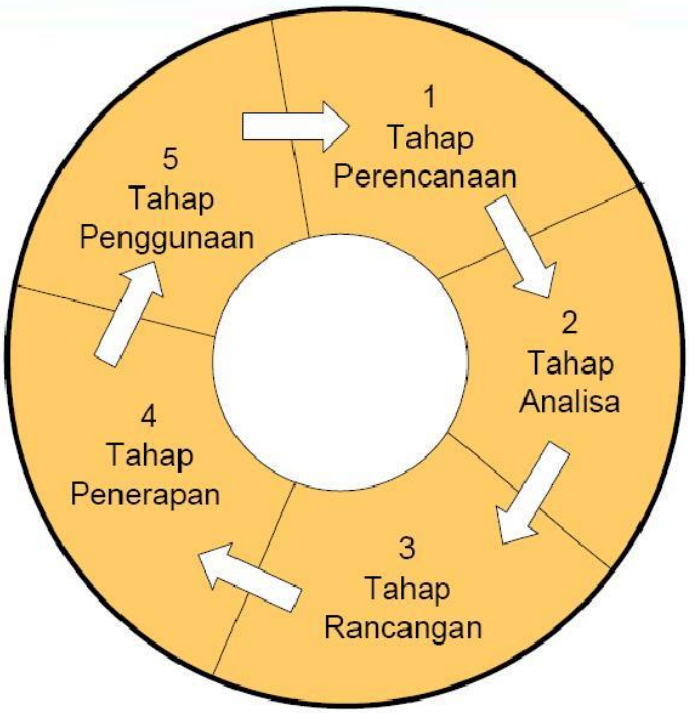
B. Perangkat Lunak Pesanan (*customized software*)

Jika system yang sedang dikembangkan tidak bisa didukung oleh paket software maka harus memesan dari perusahaan jasa/kontraktor independen atau membangun sendiri perangkat lunak (in-house) agar sesuai dengan rancangan sistemnya.

Siklus Hidup Pengembangan Perangkat Lunak (Software Development Life Cycle)

Pengembangan sistem (sistem development) dapat berarti menyusun sistem yang baru untuk menggantikan sistem yang lama secara keseluruhan atau memperbaiki sistem yang sudah ada. Sistem yang lama perlu diperbaiki atau diganti disebabkan karena beberapa hal, yaitu sebagai berikut :

1. Adanya permasalahan-permasalahan (problems) yang timbul di sistem yang lama. Permasalahan yang timbul dapat berupa :
   - Ketidakberesan dalam sistem yang lama yang menyebabkan sistem yang lama tidak dapat beroperasi sesuai dengan yang diharapkan.
   - Pertumbuhan organisasi, yang menyebabkan harus disusunnya sistem yang baru. Pertumbuhan organisasi diantaranya adalah kebutuhan informasi yang semakin luas, volume pengolahan data yang semakin meningkat, perubahan prinsip akuntansi/pengolahan data yang baru.
2. Untuk meraih kesempatan-kesempatan (opportunities), teknologi informasi telah berkembang dengan cepatnya. Perangkat keras komputer, perangkat lunak dan teknologi komunikasi telah begitu cepat berkembang. Organisasi mulai merasakan bahwa teknologi informasi ini perlu digunakan untuk meningkatkan pelayanan informasi sehingga dapat mendukung dalam proses pengambilan keputusan yang akan dilakukan oleh pihak manajemen.
3. Adanya instruksi-instruksi (directives), penyusunan sistem yang baru dapat juga terjadi karena adanya instruksi-instruksi dari atas pimpinan ataupun dari luar organisasi. Karena adanya permasalahan, kesempatan atau instruksi, maka sistem yang baru perlu dikembangkan untuk emecahkan permasalahan-permasalahan yang timbul, meraih kesempatan-kesempatan yang ada atau memenuhi instruksi yang diberikan.



Sistem yang baik adalah sistem yang selalu menyesuaikan dengan perubahan lingkungan yang terjadi disekitarnya atau sistem tersebut harus dinamis menuju pada keadaan yang lebih baik.

Tahap awal, yaitu tahap perencanaan, menyangkut studi kebutuhan user, studi kelayakan baik secara teknis maupun teknologi serta penjadwalan pengembangan suatu proyek sistem informasi.

Tahap berikutnya adalah tahap analisis, yaitu tahap dimana kita berusaha mengenali segenap permasalahan yang muncul pada pengguna, mengenali komponen-komponen sistem, obyek-obyek, hubungan antar obyek dan sebagainya.

Kemudian tahap ketiga adalah tahap perancangan, yaitu tahap dimana kita mencoba mencari solusi permasalahan yang didapat dari tahap analisis.

Tahap keempat adalah tahap implementasi dimana kita mengimplementasikan perancangan sistem ke situasi yang nyata. Pada tahap ini. Kita mulai dengan pemilihan perangkat keras, penyusunan perangkat lunak aplikasi (pengkodean/coding) apakah sistem yang dibuat sudah sesuai dengan kebutuhan user atau belum.. Jika belum, proses selanjutnya adalah iteratif, yaitu kembali ke tahap-tahap sebelumnya. Disinilah keuntungan metodologi berorientasi obyek mulai terlihat, dimana mulai dari tahap analisis hingga tahap implementasi, kita bisa menggunakan tool yang sama sehingga proses iteratif itu dapat berjalan dengan lebih efektif serta efisien ditinjau dari segi uang dan waktu.

Kemudian tahap yang terakhir adalah tahap pemeliharaan / perawatan, dimana kita bisa mulai melakukan pengoperasian sistem dan jika diperlukan dapat melakukan perbaikan-perbaikan kecil. Kemudian jika waktu penggunaan sistem habis, maka kita akan masuk lagi pada tahap perencanaan.

Siklus Hidup Pengembangan Perangkat Lunak
(Software Development Life Cycle-SWDLC)

Pengembangan perangkat lunak berjangkauan antara dua sisi ekstrim, dari sindrom "spreadsheet untuk setiap aplikasi" sampai sindrom "reinventing the wheel". Sindrom pertama terjadi karena untuk setiap aplikasi terdapat spreadsheet yang siap pakai (ready-made) atau terdapat beberapa paket perangkat lunak komersial yang akan menjalankan aplikasi tersebut. Di sisi lain sistem mengembangkan program komputer baru dari pembuatan dari awal (scratch) untuk setiap aplikasi sistem tanpa mempedulikan apa yang telah dikembangkan secara in-house atau apa yang tersedia dari penjual (vendor) perangkat lunak.

Pembangunan program mengikuti tiga tahap Siklus Hidup Pengembangan Perangkat Lunak (Software Development Life Cycle-SWDLC), yaitu :
1. Rancangan *(Design*)
2. Kode (*Code*)
3. Uji (*Test*)

1. Rancangan (*Design*)
Bagian dari rancangan sistem terinci yang akan dikonversi ke program aplikasi yang dapat digunakan sebagai pedoman oleh programmer dalam menulis program. Alat (*tools*) rancangan program yang pokok adalah :
- Bagan Terstruktur (*Structure Chart*)
- Bahasa Inggris Terstruktur (*Structure English*)
- Tabel Keputusan (*Decision Tabel*)
- Pohon Keputusan (*Decision Tree*)
- Persamaan/mirip bahasa pemrograman (*Pseudocode*)
- Kamus Data (*Data Dictionary*)
- Diagram *Warnier/Orr* (W/O)
- Diagram *Jackson*

2. Kode (*Code*)
Menulis statemen dalam bahasa pemrograman yang diasumsikan dibuat dan dijalankan oleh programmer dan tidak secara otomatis seperti yang dibangkitkan oleh paket *CASE* (*Computer Aided Software Engineering*). Beberapa paket CASE akan membangkitkan kode dari beberapa rancangan terinci sehingga menghapus adanya kebutuhan pengkode manusia (*human coders*).

3. Uji (*Test*)
Pengujian terhadap semua modul kode untuk mendeteksi dan menghapus kesalahan.

Software Development Life Cycle-SWDLC menjadi komponen siklus hidup dari System Development Life Cylce - SDLC untuk beberapa alasan (D.Suryadi H.S & Bunawan,1995) :

• SDLC mencakup pengembangan sistem keseluruhan, yang memerlukan komponen-komponen lain disamping perangkat lunak.

• Dalam sistem yang memerlukan pengembangan perangkat lunak yang didasarkan pada rancangan sistem yang diciptakan oleh SDLC, SWDLC akan diinisiai.

- Apabila SWDLC menjadi berperan, maka SWDLC seperti halnya SDLC yang berbasis lebih luas, akan memberikan kumpulan acuan tahap-tahap yang diperlukan untuk mengembangkan perangkat lunak tersebut. SWDLC menjabarkan tugas-tugas dan prosedur-prosedur yang harus dijalankan dalam setiap tahap; hasil yang diciptakan oleh setiap tahap; dan metriks untuk menyusun jadwal, mengestimasi biaya, dan mengukur produktifitas.

• SWDLC memberikan atau menyediakan kerangka dan prosedur pengoperasian standar yang mendukung cara pengembangan perangkat lunak yang terstruktur dan terancang dengan baik.

Teknnologi Object Oriented

Teknologi object-oriented merupakan paradigma baru dalam rekayasa software yang didasarkan pada obyek dan kelas. Diakui para ahli bahwa object-oriented merupakan metodologi terbaik yang ada saat ini dalam rekayasa software. Object-oriented memandang software bagian per bagian dan menggambarkan satu bagian tersebut dalam satu obyek. Satu obyek dalam sebuah model merupakan suatu fokus selama dalam proses analisis, perancangan dan implementasi dengan menekankan pada state, perilaku (behavior) dan interaksi obyek dalam model tersebut.

Teknologi obyek menganalogikan sistem aplikasi seperti kehidupan nyata yang didominasi oleh obyek. Manusia adalah obyek, komputer adalah obyek. Obyek memiliki atribut : manusia memiliki nama, pekerjaan, rumah, dan lain-lain.
Mobil memiliki warna, merk, sejumlah roda, dan lain-lain. Komputer memiliki kecepatan, sistem operasi, dan lain-lain. Obyek dapat beraksi dan bereaksi. Manusia dapat berjalan, berbicara, makan, minum ; mobil dapat berjalan, mengerem ; komputer dapat mengolah data, menampilkan gambar, dan lain-lain.
Keunggulan teknologi obyek dengan demikian adalah bahwa model yang dibuat akan sangat mendekati dunia nyata yang masalahnya akan dipecahkan oleh sistem yang dibangun. Model obyek, atribut dan kelakuan bisa langsung diambil dari obyek yang ada di dunia nyata.
Sistem yang dibangun dengan teknologi obyek memiliki fleksibilitas yang tinggi terhadap perubahan karena menggunakan konsep komponen yang bisa digunakan kembali.

Didalam dunia perangkat lunak, penggunaan berulang merupakan hal yang biasa. Contohnya ide dan formula yang hampir sama digunakan berulang oleh programmer yang berbeda untuk mengembangkan aplikasi keuangan yang khusus diadaptasikan sesuai kebutuhan dan permintaan masing-masing klien. Oleh karena itu penggunaan berulang suatu obyek
merupakan hal yang seharusnya bisa dilakukan. Suatu obyek bisa diambil untuk dimodifikasi berupa penambahan atau pengulangan untuk memecahkan suatu masalah baru.

Ada empat prinsip dasar dari pemrograman berorientasi obyek, yaitu :abstraksi, enkapsulasi, modularitas dan hirarki.

Abstraksi :

Memfokuskan perhatian pada karakteristik obyek yang paling penting dan paling dominan yang bisa digunakan untuk membedakan obyek tersebut dari obyek lainnya. Dengan abstraksi ini developer bisa menerapkan konsep KIS (Keep It Simple) pada obyek didunia nyata yang memiliki tingkat kerumitan yang tinggi.

Enkapsulasi:

Menyembunyikan banyak hal yang terdapat dalam obyek yang tidak perludiketahui oleh obyek lain. Dalam praktek pemrograman enkapsulasi diwujudkan dengan membuat suatu kelas interface yang akan dipanggil oleh obyek lain, sementara didalam obyek yang dipanggil terdapat kelas lain yang mengimplementasikan apa yang terdapat dalam kelas interface. Obyek lain hanya tahu dia perlu memanggil kelas interface tanpa perlu tahu proses apa saja yang dilakukan didalam kelas implementasinya dan untuk menuntaskan proses tersebut perlu berhubungan dengan obyek mana saja. Dengan demikian bila terjadi proses perubahan pada proses implementasi maka obyek pemanggil tidak akan terpengaruhi secara langsung.

Modularitas :

Membagi sistem yang rumit menjadi bagian-bagian yang lebih kecil yang bisa mempermudah developer memahami dan mengelola obyek tersebut. Contohnya adalah sistem akademis yang bisa dibagi menjdi kemahasiswaan, daftar mata kuliah, pembayaran uang kuliah, dan lain-lain.

Hirarki :

Hirarki berhubungan dengan abstraksi dan modularitas, yaitu pembagian berdasarkan urutan dan pengelompokkan tertentu. Misalnya untuk menentukan obyek mana yang berada pada kelompok yang sama, obyek mana yang merupakan komponen dari obyek yang memiliki hirarki lebih tinggi. Semakin rendah hirarki obyek berarti semakin jauh abstraksi dilakukan terhadap suatu obyek. Ke empat prinsip dasar ini merupakan hal yang mendasari teknologi obyek yang perlu ditanamkan dalam cara berpikir developer berorientasi obyek.

Object Oriented Analysis dan Design (OOAD)

Object-oriented mencakup bidang aplikasi yang sangat luas. Para pengguna sistem komputer dan sistem lain yang didasarkan atas teknologi komputer merasakan efek object-oriented dalam bentuk meningkatnya aplikasi software yang mudah digunakan dan servis yang lebih fleksibel, yang muncul dalam berbagai bidang industri, seperti dalam perbankan, telekomunikasi, dan sebagainya. Sedangkan bagi software engineer, object-oriented berpengaruh dalam bahasa pemrograman, metodologi rekayasa, manajemen proyek, hardware dan sebagainya.

Analisis dan perancangan berorientasi obyek amat sangat perlu dilakukan dalam pengembangan sistem berorientasi obyek. Hanya dengan kemampuan menggunakan

bahasa pemrograman berorientasi obyek yang andal, kita dapat membangun suatu sistem berorientasi obyek, namun sistem aplikasi yang dibangun akan menjadi lebih baik lagi bila langkah awalnya didahului dengan proses analisis dan perancangan berorientasi obyek, terutama untuk membangun sistem yang mudah dipelihara.

Analisis berorientasi obyek adalah metode analisis yang memeriksa requirements (syarat/keperluan yang harus dipenuhi suatu sistem) dari sudut pandang kelas-kelas dan obyek-obyek yang ditemui dalam ruang lingkup permasalahan. Sedangkan perancangan berorientasi obyek adalah metode untuk mengarahkan arsitektur software yang didasarkan pada manipulasi obyek-obyek sistem atau subsistem.

Keunggulan Object Oriented

Sekarang ini terdapat beberapa paradigma yang digunakan dalam rekayasa software, diantaranya procedure-oriented, object-oriented, data structure-oriented, data flow-oriented dan constraint oriented. Tiap-tiap paradigma tersebut cocok untuk beberapa kasus dan bagian dari seluruh kemungkinan ruang lingkup aplikasi, tetapi menurut Booch berdasarkan pengalamannya, object-oriented dapat diaplikasikan dalam seluruh ruang lingkup.

Untuk memahami mengapa OOAD unggul dalam banyak kasus, harus memahami masalah yang dihadapi perusahaan rekayasa software. Pertama, software sulit untuk dimodifikasi bila memerlukan pengembangan. Kedua, proses pembuatan software memerlukan waktu yang cukup lama sehingga kadang kala melebihi anggaran dalam pembuatannya. Ketiga, para pogrammer selalu membuat software dari dasar karena tidak adanya kode yang bisa digunakan ulang (reuse).

Kebanyakan perusahaan tersebut sebelumnya telah menggunakan pemrogramam structural. Metode structural menggunakan pendekatan dengan pendekatan top-down, yang mana pendekatan ini memecahkan masalah dengan menbagi masalah kedalam komponen-komponen hingga didapatkan komponen yang tidak dapat dibagi-bagi lagi. Pendekatan dengan cara top-down ini telah membuat peningkatan dalam kualitas software, tetapi dalam sistem yang berskala besar, pendekatan ini sering menemui banyak masalah. Pemrograman structural seringkali tidak dapat mendesain apa yang akan terjadi dalam sistem yang telah selesai tanpa mengimplementasikan sistem terlebih dahulu. Jika ditemui kesalahan dalam sistem, maka desain harus disusun ulang dari awal sampai akhir.

Hal ini akan terjadi pemborosan biaya dan waktu. Perbedaan antara metode structural dan OOAD terletak pada bagaimana data dan fungsi disimpan. Dalam metode structural, data dan fungsi disimpan terpisah. Biasanya semua data ditempatkan sebelum fungsi ditulis. Seringkali tidak terpikirkan sebelumnya untuk mengetahui data mana yang digunakan dalam suatu fungsi tertentu. Tetapi dalam OOAD, data dan fungsi yang berhubungan dalam suatu obyek disimpan bersama-sama dalam satu kesatuan. Orang akan lebih mudah memahami sistem sebagai obyek daripada prosedur, karena biasanya orang berpikir dalam bentuk obyek. Sebagai contoh, orang melihat mobil sebagai sebuah sistem yang memiliki mesin, roda , stir, tank bahan bakar dan sebagainya. Kebanyakan orang tidak melihat mobil sebagai sebuah urutan prosedur yang membuat mobil tersebut dapat

berjalan. Karena secara alamiah lebih mudah memikirkan suatu sistem dalam bentuk obyek-obyek, tidak heran kalau OOAD menjadi lebih terkenal.

Satu alasan mengapa object oriented menguntungkan bagi programmer adalah karena programmer dapat mendesain program dalam bentuk obyek-obyek dan hubungan antar obyek tersebut untuk kemudian dimodelkan dalam sistem nyata. Keuntungan yang lain adalah proses pembuatan software dapat dilakukan dengan lebih cepat karena software dibangun dari obyek-obyek standar, dapat menggunakan ulang model yang ada, dan dapat membuat model yang cepat melalui metodologi.

Kualitas yang tinggi dari software dapat dicapai karena adanya tested component. Lebih mudah dalam maintenance karena perbaikan kode hanya diperlukan pada satu tempat (bukan diurut dari awal). Mudah dalam membangun sistem yang besar karena subsistem dapat dibuat dan diuji secara terpisah. Mengubah sistem yang sudah ada tidak memerlukan membangun ulang keseluruhan sistem.

Mengorganisasi Proyek Pengembangan Perangkat Lumak

Perancang dan analis sistem terlibat dalam tim pengembangan perangkat lunak dan harus mengetahui bagaimana program ini dikode dan bagaimana hasil akhirnya. Untuk itu diperlukan keterampilan pengorganisasian dalam tim proyek. Pengorganisasian proyek pengembangan perangkat lunak memerlukan komunikasi, integrasi dan koordinasi yang baik. Pengorganisasian tim pemrograman menggunakan pendekatan organisasional.

Pendekatan Organisasional

Tiga cara untuk mengorganisasi tim pemrograman, yaitu :

1. Tim Pengembangan Program (*Program development team*)
2. Tim programmer kepala (*chief programmer team*)
3. Tim pemrograman bersama (*Egoless programming team*)

1. Tim Pengembangan Program (*Program development team*)

Tim pengembangan program dikelola oleh manajer tim atau seseorang yang terlibat dalam SDLC dari awal, dan didukung oleh perancang, pengkode, dan penguji (Gb.1.4 hal.14 Diktat kuliah) Jika perusahaan **menggunakan aturan 40-20-40** (lihat gb.1.5 hal.15 Buku Diktat Pengantar Implementasi) maka orang-orang yang memiliki keterampilan lebih tinggi harus ditugaskan untuk perancangan dan pengujian. Bila rancangan lengkap, jelas dan akurat maka tugas pengkodean akan menjadi proses yang sederhana yang dapat dijalankan oleh setiap orang yang telah kenal dengan sintaks bahasa pemrograman. Konsep ini mendukung terciptanya teknologi CASE.

2. Tim programmer kepala (*chief programmer team*)

Tim ini dibentuk dari programmer kepala atau senior yang banyak pengalaman dan pengetahuan **pemrograman**. Programmer kepala dapat berkomunikasi secara efektif dengan analis dan perancang sistem, pemakai, dan berbagai teknisi.

Programmer kepala didukung oleh asisten utama yang bertugas sebagai komunikator dengan orang lain pada tim atau penyampai informasi dari gagasan programmer kepala. Kedua orang tersebut didukung oleh Programmer pendukung/ yunior bertugas membantu programmer kepala dan asisten utama untuk proyek besar yang tidak dapat ditangani sendiri. Para programmer pendukung biasanya mengkode modul-modul tingkat rendah. Tim ini juga didukung oleh pustakawan, administrator, editor, dan klerk program.

3. Tim pemrograman bersama (*Egoless programming team*)

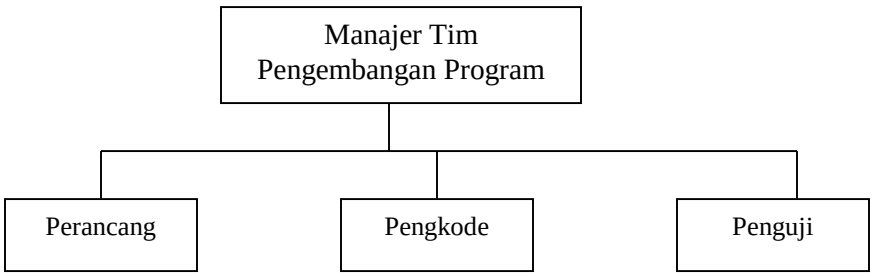
Tim ini terbentuk dari seluruh rekan yang bersama-sama bertanggung jawab atas pengembangan **perangkat lunak** tanpa supervisi langsung/pimpinan.

Perbedaan pendekatan-pendekatan tersebut :

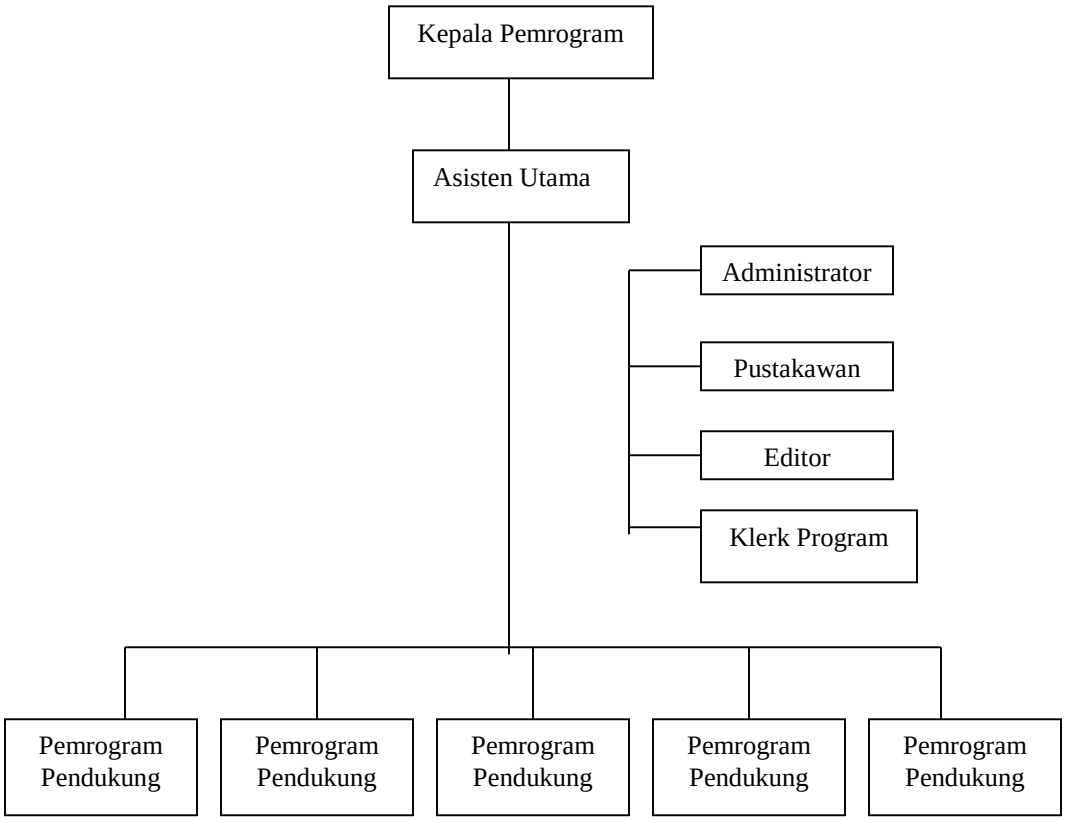
- ♠ Tim pengembangan program mengembangkan aturan 40-20-40 yaitu menekankan pada perancangan dan pengujian.
- ♠ Tim programmer kepala dan tim pemrograman bersama menekankan pada fungsi pengkodean.

Jumlah interface dan lintasan komunikasi dari pendekatan di atas:

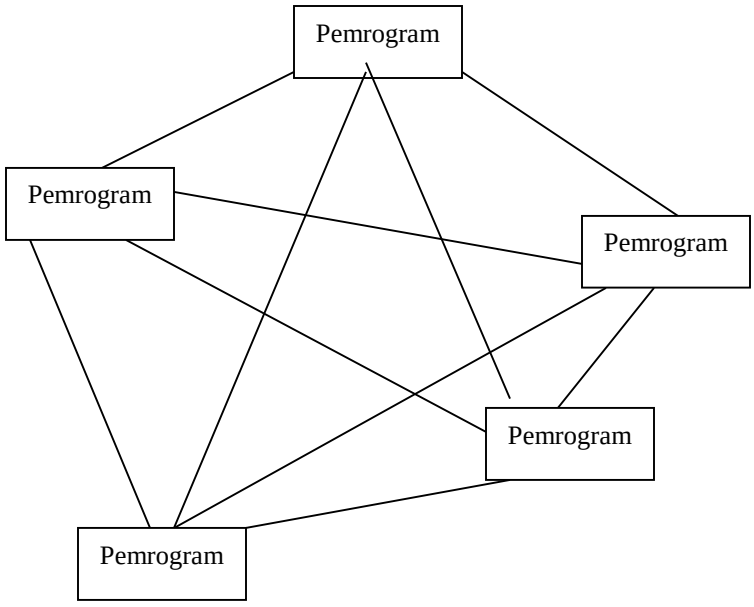
- ♣ **Tim pengembangan program** tersusun atas 2 perancang, 1 pengkode, 2 penguji. Interface dan lintasan komunikasi berada antara perancang dan pengkode, pengkode dan penguji, perancang dan penguji. Interface dan lintasan komunikasi ke manajer tim hanya memberikan rekapitulasi dan informasi kinerja karena manajer tidak terlibat langsung dalam pekerjaan yang sebenarnya. Jadi total interface dan lintasan komunikasi ada lima, dan satu interface manajemen.



- ♣ **Tim programmer kepala** terdiri dari lima programmer pendukung mempunyai lima interface dan lintasan komunikasi, dan lebih mungkin memenuhi deadline yang ketat.


Kepala Pemrogram
Asisten Utama
Administrator
Pustakawan
Editor
Klerk Program
Pemrogram Pendukung
Pemrogram Pendukung
Pemrogram Pendukung
Pemrogram Pendukung
Pemrogram Pendukung

- **Tim pemrograman bersama** terdiri dari lima programmer. Jumlah interface dan lintasan komunikasi = n(n-1)/2= 5(5-1)/2=10



Biasanya untuk komunikasi membutuhkan waktu dan mengurangi produktivitas. Segala jenis pekerjaan pengembangan biasanya waktu restart (mulai lagi) setelah setiap interupsi besarnya 30 menit sehingga peluang pemrograman yang bisa dilakukan waktunya sedikit. Oleh karena itu apabila terdapat lebih dari tiga programmer yang terlibat maka sebaiknya ditetapkan seorang supervisor atau pimpinan.

Konsep pabrik perangkat lunak

Pabrik-pabrik perangkat lunak yang menerapkan prinsip pengendalian kualitas dan manajemen proyek. Berbagai macam tujuan pabrik pengembangan perangkat lunak, yaitu :

- Penerapan cara termekanisasi (terekayasa) untuk pengembanan sistem dan perangkat lunak.
- Penggunaan perangkat pemodelan dan teknologi CASE
- Penginstalasian teknik manajemen proyek
- Penekanan pada kemungkinan kemampuan pemeliharaan (Maintainability), penggunaan (Usability), penggunaan ulang (reusability), kehandalan (reliability), perluasan faktor-faktor rancangan (extandability)→ MURRE
- Pencapaian produktivitas pengembangan perangkat lunak dan sistem yang optimal.

Mengukur Produktivitas Dalam Pengembangan Perangkat Lunak

Produktivitas dapat diukur dengan rumus :

$$\text{Produktivitas} = \frac{\text{Output yang dihasilkan}}{\text{Input yang dikonsumsi}}$$

Produktivitas pengembangan perangkat lunak dapat ditingkatkan dengan menaikkan output, menurunkan input, atau keduanya.

Input yang dikonsumsi relatif mudah diukur misalnya tenaga kerja, workstation, pasokan, sebaliknya output relatif tidak mudah diukur. Untuk **mengukur output** pengembangan perangkat lunak dapat menggunakan **teknik metrik**.

Manfaat menggunakan metrik yaitu :

a. Bisa mengelola proses pengembangan perangkat lunak
b. Bisa mengukur dampak perubahan misalnya perubahan ke teknologi CASE atau dari satu generasi bahasa komputer ke generasi lain
c. Bisa terjadi persepsi bahwa pengembangan perangkat lunak lebih bersifat ilmiah (produk yang dimekanisasi)

Dua metrik yang paling berperan adalah :

1. Jalur Kode yang bisa dieksekusi (***Lines Of Executable Code*** atau ***LOEC***)
2. Titik Fungsi (***Function Point***)

1. Mencacah Jalur Kode yang bisa Dieksekusi (*LOEC*)

Metrik LOEC mengukur cakupan pada pengkodean. Beberapa Profesional Sistem menggunakan Jalur Kode Sumber sebagai metriknya. Jalur Kode Sumber (*Source Line Of Code* atau *SLOC*) adalah segala jalur program yang bukan penjelasan maupun jalur kosong tanpa mempedulikan jumlah statemen (statemen yang bisa dan tidak bisa dieksekusi). *SLOC* bisa untuk mengukur *LOEC*.

Keuntungan menggunakan metric LOEC sebagai ukuran produktivitas output perangkat lunak adalah :

a. Mudah ditetapkan dan dibahas secara jelas.
   End user, manajer, dan profesional sistem biasanya memahami apa yang dimaksud dengan jalur kode yang bisa dieksekusi.
b. Diakui secara luas.
   Metrik ini seringkali digunakan oleh vendor sebagai alat pengembangan perangkat lunak.
c. Mudah diukur.
   LOEC dapat dihitung untuk menentukan ukuran program.
d. Mudah digunakan untuk estimasi.
   Ukuran perkiraan suatu program ditentukan berdasarkan dokumentasi rancangan sistem detail. Angka ini digunakan untuk mengestimasi waktu dan biaya proyek pengembangan perangkat lunak.

Contoh : program yang diusulkan **berisi 100K LOEC**. Jika **2K LOEC d**apat di-hasilkan oleh **satu orang per bulan**, maka untuk menyelesaikan proyek tersebut diperlukan diperlukan **50 orang** perbulan. Jika input yang diperlukan untuk mendukung satu orang per bulan sebesar **$9000** maka proyek tersebut membutuhkan biaya **$450.000.**

Apa kelemahan dari satu orang per bulan (*person-month*) dan pencacahan LOEC?

Frederick P. Brooks, Jr. Dalam bukunya berjudul The Mythical Man-Month bahwa the person-month sebagai unit pengukuran suatu pekerjaan adalah mitos. Ia menyatakan bahwa orang dan bulan bisa saling ditukar atau diganti. Misal memerlukan 50 orang untuk waktu 1 bulan atau memerlukan 10 orang dengan per orang memerlukan waktu 5 bulan Apabila tugas dibagikan kepada banyak pekerja tanpa komunikasi hal ini mustahil, tetapi apabila menggandakan tugas ukuran tim maka tidak akan menggandakan produktivitas.

Penggunaan person-month sebagai benchmark harus digunakan secara wajar dan konsisten.

Kelemahan lain metrik LOEC adalah mengukur dengan dasar jumlah LOEC yang lebih besar adalah yang produktif bukan jalur kode yang diperlukan. Misal untuk menulis suatu program Programmer assembly membutuhkan waktu 4 minggu untuk 1500 LOEC, programmer COBOL butuh waktu 2 minggu untuk 500 LOEC, dan programmer C butuh waktu 1 minggu untuk 300 LOEC. Dengan **metric LOEC programmer assembly yang paling produktif**. **Secara riil programmer yang paling produktif adalah programmer C. Yang lebih efektif dan efisien dijalankan adalah bahasa pemrograman C dan COBOL.**

2. Metrik Titik Fungsi (*Function Point*)

Metrik titik fungsi dirancang untuk mengatasi beberapa kelemahan metrik LOEC. Ada lima fungsi yang dianalisis untuk diukur oleh profesional sistem, yaitu :

1. Jumlah input, seperti form dan layar
2. Jumlah output, seperti laporan dan layar
3. Jumlah query yang diminta oleh end user
4. Jumlah file logic yang diakses dan digunakan
5. Jumlah interface ke aplikasi lain.

Metrik titik fungsi mengukur apa yang akan diberikan oleh tim pengembangan perangkat lunak kepada end user. Metrik ini mencakup perancangan, pengkodean, dan pengujian, dan metrik ini juga mengukur efisiensi dan efektifitas.

Kelebihan/keuntungan metrik titik fungsi :

♠ Mengukur produktivitas perangkat lunak menggunakan cara yang seragam tanpa memandang bahasa pemrograman yang digunakan.

- ♠ Mengukur efisiensi dan efektivitas. Efisiensi berkaitan dengan sumber-sumber yang dikonsumsi dalam pengembangan suatu aplikasi tertentu secara tepat waktu. Efektivitas berhubungan dengan kualitas program dan kemampuannya dalam memenuhi kebutuhan pemakai.

Di bawah ini adalah contoh menghitung titik fungsi.

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

Tabel.3 Analisis Titik Fungsi Proyek Pengembangan Software

Derajajat kompleksitas 1 s/d 10 (1= derajat rendah dan 10 derajat paling kompleks). Sebagai contoh 12 input mempunyai kompleksitas cukup rendah 2, dan 3 input kompleksitas rata-rata 5, serta 12 input mempunyai kompleksitas cukup tinggi 8. Semua titik fungsi dicacah atau dikalkulasi dengan cara yang sama. Jumlah total titik fungsi adalah 1272.

$$\text{Tingkat Produktivitas Pengembangan} = \frac{\text{Jumlah titik fungsi yang dihadirkan}}{\text{Jumlah person-month}}$$

Tingkat produktivitas pengembangan rata-rata untuk perangkat lunak aplikasi, umumnya berada antara 5 dan 10 artinya satu orang bisa menyerahkan atau menghasilkan sekitar 5 sampai 10 titik fungsi per bulan.

Jika proyek pengembangan perangkat lunak memerlukan 1272 titik fungsi dan tingkat penyerahan rata-rata dari tim yang akan mengembangkannya adalah 8 titik fungsi per person-month, maka proyek tersebut memerlukan sekitar 159 person-month, (1272/8=159). Jika satu person-month mengkonsumsi $10.000, maka proyek tersebut akan membutuhkan biaya ± $1.590.000

Jika organisasi menginstal teknologi CASE maka bisa kemungkinan menghasilkan sekitar 15 atau 20 titik fungsi per person-month.
Jika organisasi yang menggunakan pendekatan rancangan dan diadopsi dengan teknologi dan dikombinasikan dengan menerapkan penggunaan ulang kode sekitar 50% , maka tingkat produktivitas pengembangan bisa mencapai 70 titik fungsi per person-month.

Misalkan, diasumsikan biaya per person-month meningkat menjadi $12.000 dengan tingkat penyerahan 62 titik fungsi per person-month, maka jumlah person-month yang akan menghasilkan 1272 titik fungsi dikurangi menjadi 20,5 person-month (1272/62=20,5), dan biaya sebesar $246.000 → ($12.000 x 20,5= $246.000).
Penghematan biaya sebesar $1.344.000→($1.590.000 - $246.000 = $1.344.000).

Pengaruh Manajemen terhadap Produktivitas

1. Metrik Produktivitas yang tidak baik atau tidak tersedia

   Manajemen yang tidak baik dapat menurunkan produktivitas perangkat lunak sehingga manajer tidak akan bisa mengestimasi waktu dan biaya pengembangan perangkat lunak atau tidak bisa mengukur produktivitas/tingkat penyerahan.
2. Perencanaan dan pengontrolan yang tidak baik

   Manajer proyek membiarkan proyek berkembang semaunya tanpa mempunyai target penyelesaian dan penyerahannya.Untuk membantu mengurangi perencanaan dan pengontrolan yang tidak baik bisa menggunakan pengaplikasian teknik manajemen proyek seperti PERT.
3. Pencampuran keterampilan yang tidak baik.

   Tim pengembangan cenderung kurang memberi tekanan pada perancangan dan pengujian, dan lebih menekankan pada pengkodean. Idealnya campuran keterampilan adalah 40% perancangan, 20% pengkodean, dan 40% pengujian.
4. Pengkodean Prematur.

   Manajer proyek biasanya ingin menyelesaiakan proyek pengembangan secara cepat dan segera melakukan pengkodean dengan mengabaikan tahap rancangan sehingga

waktu dan biaya yang dikeluarkan tidak sebanding dengan manfaat atau keuntungan yang diperoleh.

5. Imbalan yang tidak adil.

   Upah atau gaji yang tidak adil kepada pelaksana tim yang berpotensi (berketrampilan dan motivasi tinggi) mengakibatkan pelaksana tersebut frustasi dan keluar. Cara yang bisa dilakukan untuk memberikan imbalan dengan pemberian bonus kinerja khusus, bonus perjalanan, dan pemberian kesempatan mengikuti seminar khusus.

Memproduksi Perangkat Lunak Berkualitas Tinggi

Sasaran akhir pengembangan perangkat lunak adalah untuk menghasilkan perangkat lunak berkualitas tinggi pada tingkat produktivitas tinggi yang memberi nilai tambah kepada perusahaan.

Apa yang dimaksud dengan kualitas perangkat lunak ?

Ada tiga dimensi untuk mengukur kualitas :

1. Faktor kinerja, dari sudut pandang end user seperti :
   a. Kinerja pengoperasian keseluruhan
   b. Kemudahan pembelajaran
   c. Pengontrolan dan penanganan kesalahan
   d. Dukungan dari pembuat dan pemelihara perangkat lunak.
2. Faktor rancangan MURRE mencakup :
   a. Kemungkinan pemeliharaan (Maintainability)
   b. Kemungkinan penggunaan (Usability)
   c. Kemungkinan penggunaan ulang (Reusability)
   d. Kehandalan (Reliability)
   e. Kemungkinan perluasan (Extendability)
3. Faktor Strategik PDM
   a. Meningkatkan produktivitas (Productivity)
   b. Menambah keragaman produk dan pelayanan (Development)
   c. Meningkatkan fungsi manajemen (Management)

Apa yang dimaksud dengan Jaminan Kualitas (*Quality Assurance atau QA*) ?

Adalah memastikan bahwa SWDLC yang digunakan dalam pengembangan perangkat lunak berkualitas sesuai dengan standart yang telah ditetapkan bagi produk tersebut.

Pada skala yang lebih luas, jaminan kualitas mencakup pemonitoran terus menerus terhadap semua tahap-tahap pengembangan sistem dan perangkat lunak dari perencanaan sampai implementasi, dan pengoreksian terhadap **proses pengembangan sistem dan perangkat lunak.**

Apa yang dimaksud dengan pengendalian kualitas ?

Pengendalian Kualitas (Quality Control) memfokuskan pada produk, yaitu apa yang dihasilkan. Pengendalian kualitas mengevaluasi sistem dan perangkat lunak setelah dikembangkan.

Kualitas harus dirancang ke dalam sistem dan perangkat lunak ketika mereka sedang dibuat, bukan setelah pembuatannya selesai. Jadi jaminan kualitas adalah teknik pencegahan kesalahan, sedangkan pengendalian kualitas merupakan teknik penghapusan kesalahan.

Menciptakan kelompok Jaminan Kualitas (*QA*)

Kelompok QA yang independen terbentuk atas wakil-wakil end user, analis system, perancang system, programmer terampil, yang semuanya tidak tergantung pada developer (pembuat).

Tugas-tugas QA mencakup :

1. Menetapkan standar untuk pengembangan sistem dan perangkat lunak (SDLC dan SWDLC)
2. Mengevaluasi laporan terdokumentasi yang siap diserahkan.
3. Menjalankan tahapan pemeriksaan rancangan sistem dan perangkat lunak.
4. Melakukan tahapan pemeriksaan pengkodean.
5. Menjalankan pengujian.

Merencanakan Proyek Siklus Hidup Pengembangan Perangkat Lunak (SWDLC)

Manajer Proyek menjadual dan memonitor semua tugas yang diperlukan untuk menyelesaikan SWDLC, dan Teknik yang digunakan adalah Teknik Tinjauan dan Evaluasi Program (Program Evaluation Review and Technic atau PERT).

Sasaran PERT adalah untuk menentukan rangkaian atau urutan pelaksanaan tugas pengembangan perangkat lunak dan untuk mengestimasi lamanya waktu yang diperlukan dari awal sampai selesainya pelaksanaan tugas.

Lamanya proyek yang terdiri atas serangkaian tugas yang harus dijalankan secara urut merupakan jalur kritis (critical path) darai proyek tersebut.

Empat langkah menyusun jaringan PERT pengembangan perangkat lunak :

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