

Bahan Ajar Rekaya Perangkat Lunak

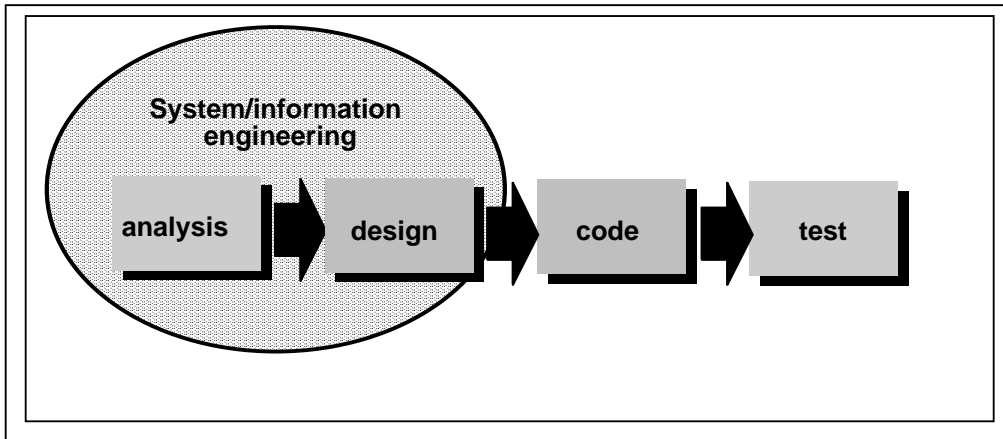
SOFTWARE PROCESS MODEL I

Disiapkan oleh: Umi Proboyekti, S.Kom, MLIS

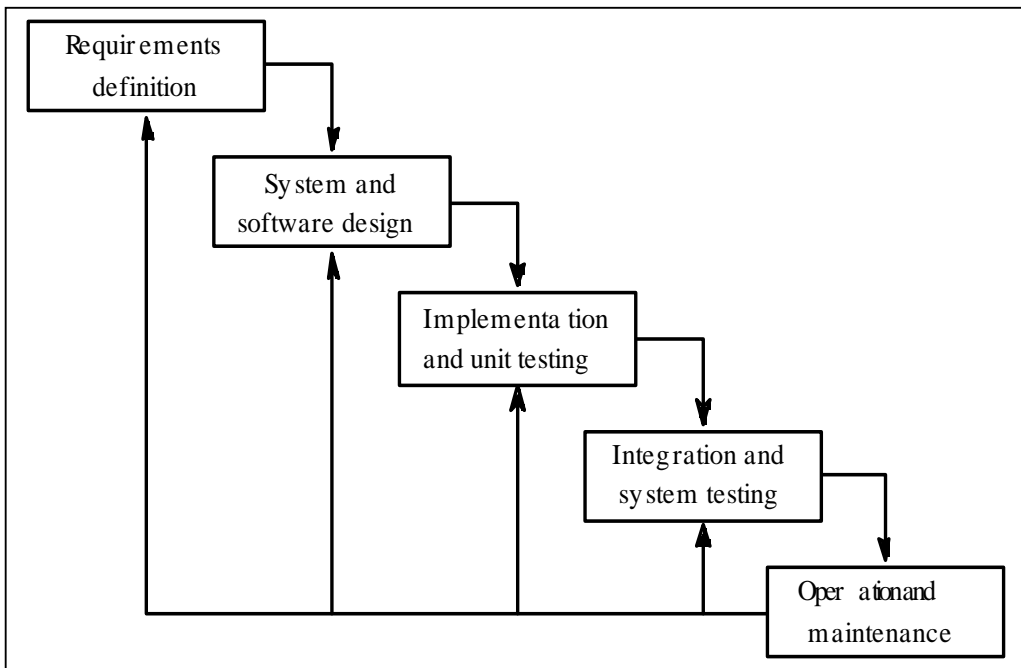
Linear Sequential Model/ Waterfall Model

Model ini adalah model klasik yang bersifat sistematis, berurutan dalam membangun software. Berikut ini ada dua gambaran dari waterfall model. Sekalipun keduanya menggunakan nama-nama fase yang berbeda, namun sama dalam intinya.

Fase-fase dalam Waterfall Model menurut referensi Pressman:



Fase-fase dalam Waterfall Model menurut referensi Sommerville :



1. **Requirements analysis and definition:** Mengumpulkan kebutuhan secara lengkap kemudian dianalisis dan didefinisikan kebutuhan yang harus dipenuhi oleh program yang akan dibangun. Fase ini harus dikerjakan secara lengkap untuk bisa menghasilkan desain yang lengkap.
2. **System and software design:** Desain dikerjakan setelah kebutuhan selesai dikumpulkan secara lengkap.
3. **Implementation and unit testing:** desain program diterjemahkan ke dalam kode-kode dengan menggunakan bahasa pemrograman yang sudah ditentukan. Program yang dibangun langsung diuji baik secara unit.
4. **Integration and system testing:** Penyatuan unit-unit program kemudian diuji secara keseluruhan (system testing).
5. **Operation and maintenance:** mengoperasikan program dilingkungannya dan melakukan pemeliharaan, seperti penyesuaian atau perubahan karena adaptasi dengan situasi sebenarnya.

Kekurangan yang utama dari model ini adalah kesulitan dalam mengakomodasi perubahan setelah proses dijalani. Fase sebelumnya harus lengkap dan selesai sebelum mengerjakan fase berikutnya.

Masalah dengan waterfall :

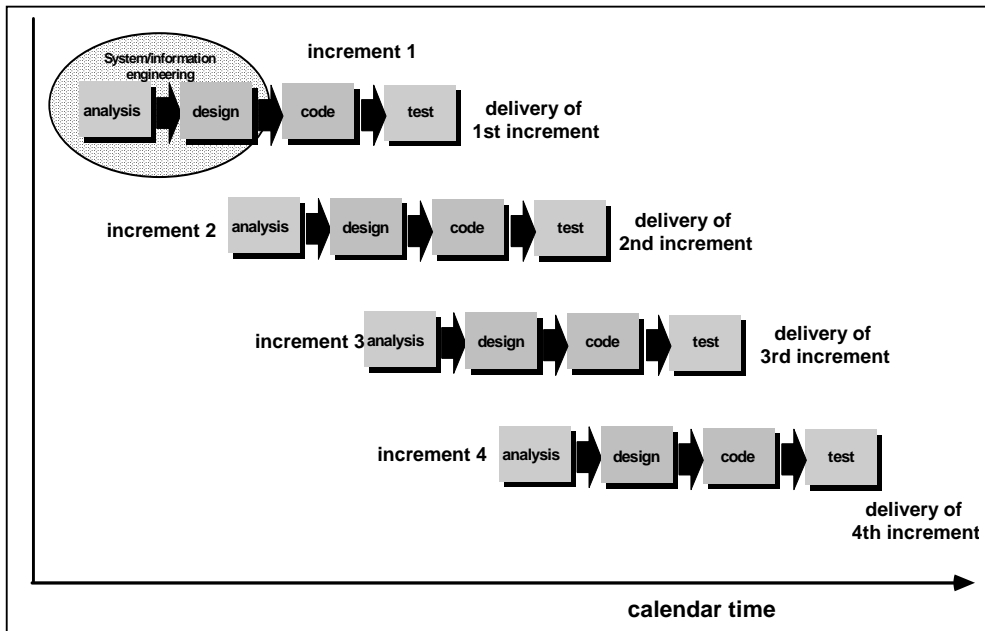
1. Perubahan sulit dilakukan karena sifatnya yang kaku.
2. Karena sifat kakunya, model ini cocok ketika kebutuhan dikumpulkan secara lengkap sehingga perubahan bisa ditekan sekecil mungkin. Tapi pada kenyataannya jarang sekali konsumen/pengguna yang bisa memberikan kebutuhan secara lengkap, perubahan kebutuhan adalah sesuatu yang wajar terjadi.
3. Waterfall pada umumnya digunakan untuk rekayasa sistem yang besar dimana proyek dikerjakan di beberapa tempat berbeda, dan dibagi menjadi beberapa bagian sub-proyek.

Evolutionary Software Process Models

Bersifat iteratif/ mengandung perulangan. Hasil proses berupa produk yang makin lama makin lengkap sampai versi terlengkap dihasilkan sebagai produk akhir dari proses.

Dua model dalam evolutionary software process model adalah:

1. Incremental Model (Original: Mills)

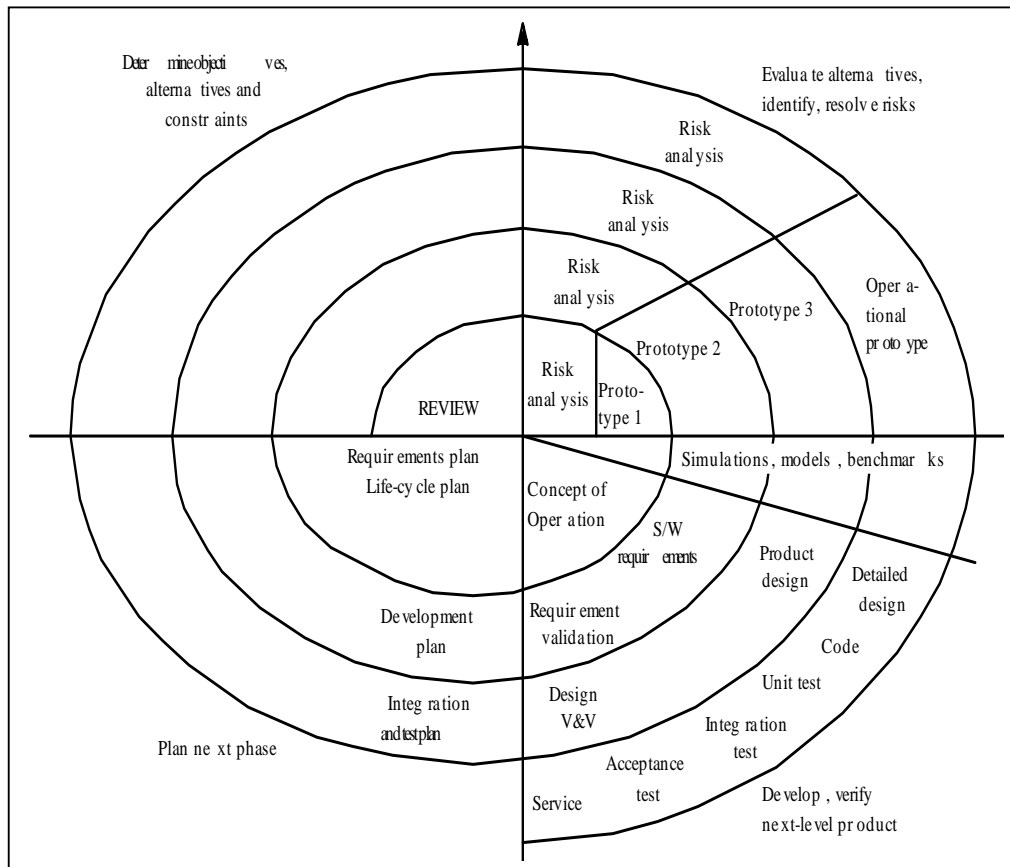


1. kombinasikan element-element dari waterfall dengan sifat iterasi/perulangan.
2. element-element dalam waterfall dikerjakan dengan hasil berupa produk dengan spesifikasi tertentu, kemudian proses dimulai dari fase pertama hingga akhir dan menghasilkan produk dengan spesifikasi yang lebih lengkap dari yang sebelumnya. Demikian seterusnya hingga semua spesifikasi memenuhi kebutuhan yang ditetapkan oleh pengguna.
3. produk hasil increment pertama biasanya produk inti (core product), yaitu produk yang memenuhi kebutuhan dasar. Produk tersebut digunakan oleh pengguna atau menjalani review/pengecekan detail. Hasil review tersebut menjadi bekal untuk pembangunan pada increment berikutnya. Hal ini terus dikerjakan sampai produk yang komplit dihasilkan.
4. model ini cocok jika jumlah anggota tim pengembang/pembangun PL tidak cukup.
5. Mampu mengakomodasi perubahan secara fleksibel.
6. Produk yang dihasilkan pada increment pertama bukanlah prototype, tapi produk yang sudah bisa berfungsi dengan spesifikasi dasar.

Masalah dengan Incremental model:

1. cocok untuk proyek berukuran kecil (tidak lebih dari 200.000 baris coding)
2. mungkin terjadi kesulitan untuk memetakan kebutuhan pengguna ke dalam rencana spesifikasi masing-masing hasil increment

2. Spiral Model (Original: Boehm)



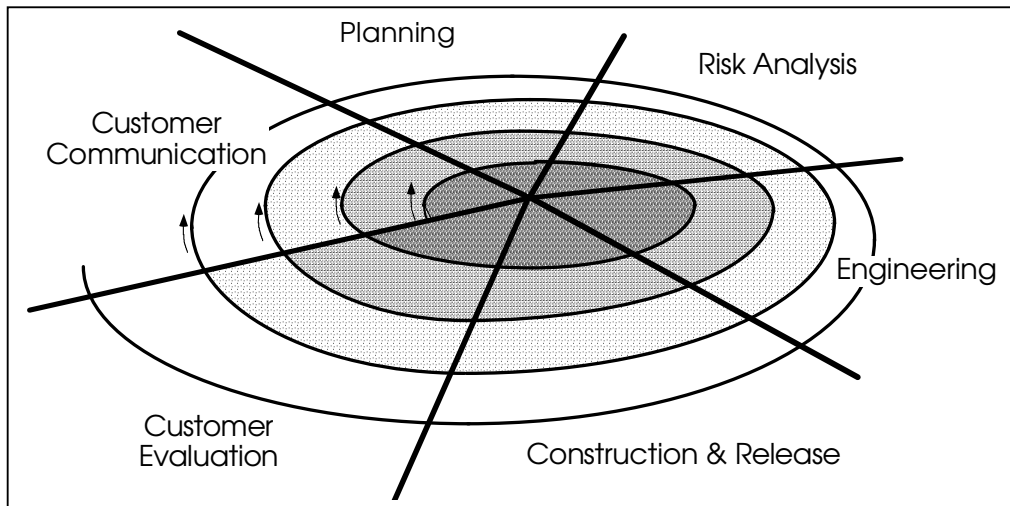
Proses digambarkan sebagai spiral. Setiap loop mewakili satu fase dari software process. Loop paling dalam berfokus pada kelayakan dari sistem, loop selanjutnya tentang definisi dari kebutuhan, loop berikutnya berkaitan dengan desain sistem dan seterusnya. Setiap Loop dibagi menjadi beberapa sektor :

1. **Objective settings (menentukan tujuan):** menentukan tujuan dari fase yang ditentukan. Batasan-batasan pada proses dan produk sudah diketahui. Perencanaan sudah disiapkan. Resiko dari proyek sudah diketahui. Alternatif strategi sudah disiapkan berdasarkan resiko-resiko yang diketahui, dan sudah direncanakan.
2. **Risk assessment and reduction (Penanganan dan pengurangan resiko):** setiap resiko dianalisis secara detil pada sektor ini. Langkah-langkah penanganan dilakukan, misalnya membuat prototype untuk mengetahui ketidakcocokan kebutuhan.
3. **Development and Validation (Pembangunan dan pengujian):** Setelah evaluasi resiko, maka model pengembangan sistem dipilih. Misalnya jika resiko user interface dominan, maka membuat prototype User Interface. Jika bagian keamanan yang bermasalah, maka

menggunakan model formal dengan perhitungan matematis, dan jika masalahnya adalah integrasi sistem model waterfall lebih cocok.

4. **Planning**: Proyek dievaluasi atau ditinjau-ulang dan diputuskan untuk terus ke fase loop selanjutnya atau tidak. Jika melanjutkan ke fase berikutnya rencana untuk loop selanjutnya.

Pembagian sektor tidak bisa saja dikembangkan seperti pada pembagian sektor berikut pada model variasi spiral di bawah ini:



1. Customer communication: membangun komunikasi yang baik dengan pengguna/customer.
2. Planning: mendefinisikan sumber, batas waktu, informasi-informasi lain seputar proyek
3. Risk analysis: identifikasi resiko manajemen dan teknis
4. Engineering: pembangunan contoh-contoh aplikasi, misalnya prototype
5. Construction and release : pembangunan, test, install dan support.
6. Customer evaluation: mendapatkan feedback dari pengguna berdasarkan evaluasi PL pada fase engineering dan fase instalasi.

Pada model spiral, resiko sangat dipertimbangkan. Resiko adalah sesuatu yang mungkin mengakibatkan kesalahan.

Model spiral merupakan pendekatan yang realistis untuk PL berskala besar.

Pengguna dan pembangun bisa memahami dengan baik software yang dibangun karena setiap kemajuan yang dicapai selama proses dapat diamati dengan baik.

Namun demikian, waktu yang cukup panjang mungkin bukan pilihan bagi pengguna, karena waktu yang lama sama dengan biaya yang lebih besar.

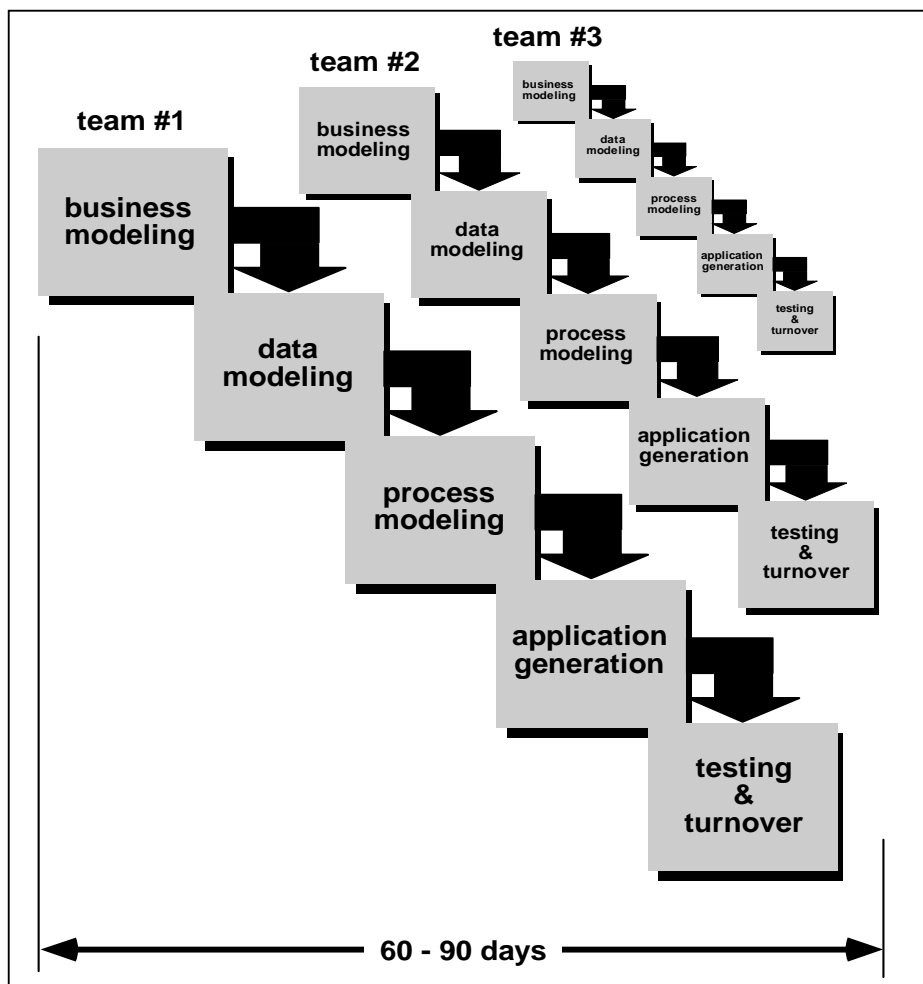
RAD (Rapid Application Development)

RAD adalah model proses pembangunan PL yang incremental. RAD menekankan pada siklus pembangunan yang pendek/singkat. RAD mengadopsi model waterfall dan pembangunan dalam waktu singkat dicapai dengan menerapkan component based construction. Waktu yang singkat adalah batasan yang penting untuk model ini.

Jika kebutuhan lengkap dan jelas maka waktu yang dibutuhkan untuk menyelesaikan secara komplit software yang dibuat adalah misalnya 60 sampai 90 hari.

Kelemahan dalam model ini:

1. tidak cocok untuk proyek skala besar
2. proyek bisa gagal karena waktu yang disepakati tidak dipenuhi
3. sistem yang tidak bisa dimodularisasi tidak cocok untuk model ini
4. resiko teknis yang tinggi juga kurang cocok untuk model ini



Fase-fase di atas menggambarkan proses dalam model RAD. Sistem dibagi-bagi menjadi beberapa modul dan dikerjakan dalam waktu yang hampir bersamaan dalam batasan waktu yang sudah ditentukan.

1. **Business modelling** : menjawab pertanyaan-pertanyaan: informasi apa yang mengendalikan proses bisnis? Informasi apa yang dihasilkan? Siapa yang menghasilkan informasi? Kemana informasi itu diberikan? Siapa yang mengolah informasi? → kebutuhan dari sistem
2. **Data modelling**: aliran informasi yang sudah didefinisikan, disusun menjadi sekumpulan objek data. Ditentukan karakteristik/atribut dan hubungan antar objek-objek tersebut → analisis kebutuhan dan data
3. **Process Modelling** : objek data yang sudah didefinisikan diubah menjadi aliran informasi yang diperlukan untuk menjalankan fungsi-fungsi bisnis.
4. **Application Generation**: RAD menggunakan component program yang sudah ada atau membuat component yang bisa digunakan lagi, selama diperlukan.
5. **Testing and Turnover**: karena menggunakan component yang sudah ada, maka kebanyakan component sudah melalui uji atau testing. Namun component baru dan interface harus tetap diuji.

Prototyping Model

Kadang-kadang klien hanya memberikan beberapa kebutuhan umum software tanpa detail input, proses atau detail output. Di lain waktu mungkin dimana tim pembangun (developer) tidak yakin terhadap efisiensi dari algoritma yang digunakan, tingkat adaptasi terhadap sistem operasi atau rancangan form user interface. Ketika situasi seperti ini terjadi model prototyping sangat membantu proses pembangunan software.

Proses pada model prototyping yang digambarkan pada gambar 1, bisa dijelaskan sebagai berikut:

- pengumpulan kebutuhan: developer dan klien bertemu dan menentukan tujuan umum, kebutuhan yang diketahui dan gambaran bagian-bagian yang akan dibutuhkan berikutnya. Detail kebutuhan mungkin tidak dibicarakan disini, pada awal pengumpulan kebutuhan
- perancangan : perancangan dilakukan cepat dan rancangan mewakili semua aspek software yang diketahui, dan rancangan ini menjadi dasar pembuatan prototype.
- Evaluasi prototype: klien mengevaluasi prototype yang dibuat dan digunakan untuk memperjelas kebutuhan software.

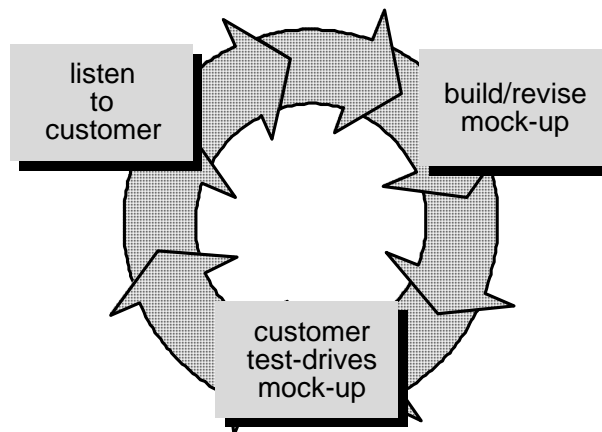
Perulangan ketiga proses ini terus berlangsung hingga semua kebutuhan terpenuhi. Prototype-prototype dibuat untuk memuaskan kebutuhan klien dan untuk memahami kebutuhan klien lebih baik.

Prototype yang dibuat dapat dimanfaatkan kembali untuk membangun software lebih cepat, namun tidak semua prototype bisa dimanfaatkan.

Sekalipun prototype memudahkan komunikasi antar developer dan klien, membuat klien mendapat gambaran awal dari prototype, membantu mendapatkan kebutuhan detail lebih baik namun demikian prototype juga menimbulkan masalah:

1. dalam membuat prototype banyak hal yang diabaikan seperti efisiensi, kualitas, kemudahan dipelihara/dikembangkan, dan kecocokan dengan lingkungan yang sebenarnya. Jika klien merasa cocok dengan prototype yang disajikan dan berkeras terhadap produk tersebut, maka developer harus kerja keras untuk mewujudkan produk tersebut menjadi lebih baik, sesuai kualitas yang seharusnya.
2. developer biasanya melakukan kompromi dalam beberapa hal karena harus membuat prototype dalam waktu singkat. Mungkin sistem operasi yang tidak sesuai, bahasa pemrograman yang berbeda, atau algoritma yang lebih sederhana.

Agar model ini bisa berjalan dengan baik, perlu disepakati bersama oleh klien dan developer bahwa prototype yang dibangun merupakan alat untuk mendefinisikan kebutuhan software.



Component-based Development Model

Component-based development sangat berkaitan dengan teknologi berorientasi objek. Pada pemrograman berorientasi objek, banyak class yang dibangun dan menjadi komponen dalam suatu software. Class-class tersebut bersifat reusable

artinya bisa digunakan kembali. Model ini bersifat iteratif atau berulang-ulang prosesnya.

Secara umum proses yang terjadi dalam model ini adalah:

1. identifikasi class-class yang akan digunakan kembali dengan menguji class tersebut dengan data yang akan dimanipulasi dengan aplikasi/software dan algoritma yang baru
2. Class yang dibuat pada proyek sebelumnya disimpan dalam class library, sehingga bisa langsung diambil dari library yang sudah ada. Jika ternyata ada kebutuhan class baru, maka class baru dibuat dengan metode berorientasi objek.
3. bangun software dengan class-class yang sudah ditentukan atau class baru yang dibuat, integrasikan.

Penggunaan kembali komponen software yang sudah ada menguntungkan dari segi:

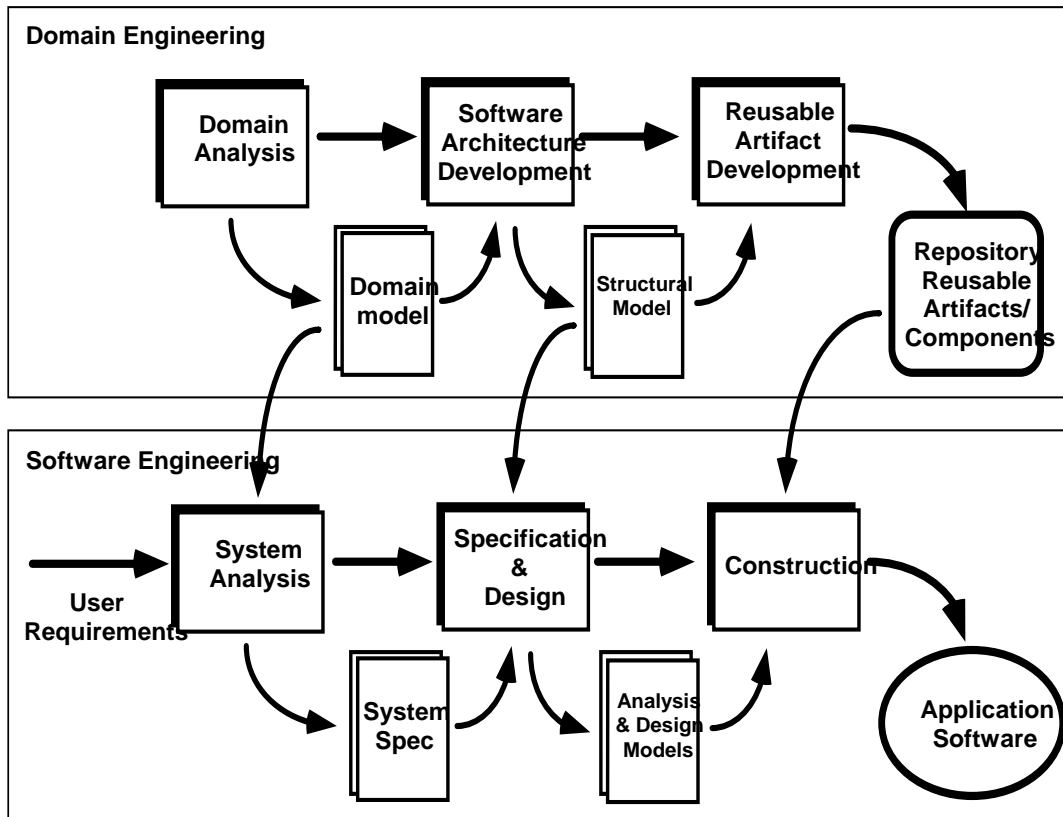
1. siklus waktu pengembangan software, karena mampu mengurangi waktu 70%
2. biaya produksi berkurang sampai 84% arena pembangunan komponen berkurang

Pembangunan software dengan menggunakan komponen yang sudah tersedia dapat menggunakan komponen COTS (Commercial off-the-shelf) – yang bisa didapatkan dengan membeli atau komponen yang sudah dibangun sebelumnya secara internal.

Component-Based Software Engineering (CBSE) adalah proses yang menekankan perancangan dan pembangunan software dengan menggunakan komponen software yang sudah ada. CBSE terdiri dari dua bagian yang terjadi secara paralel yaitu software engineering (component-based development) dan domain engineering seperti yang digambarkan pada Gambar 2:

1. domain engineering menciptakan model domain bagi aplikasi yang akan digunakan untuk menganalisis kebutuhan pengguna. Identifikasi, pembangunan, pengelompokan dan pengalokasikan komponen-komponen software supaya bisa digunakan pada sistem yang ada dan yang akan datang.
2. software engineering (component-based development) melakukan analisis terhadap domain model yang sudah ditetapkan kemudian menentukan spesifikasi dan merancang berdasarkan model struktur dan spesifikasi sistem, kemudian melakukan pembangunan software dengan menggunakan komponen-komponen yang sudah ditetapkan berdasarkan

analisis dan rancangan yang dihasilkan sebelumnya hingga akhirnya menghasilkan software.



Extreme Programming (XP) Model

Model proses ini diciptakan dan dikembangkan oleh Kent Beck. Model ini adalah model proses yang terbaru dalam dunia rekayasa perangkat lunak dan mencoba menjawab kesulitan dalam pengembangan software yang rumit dan sulit dalam implementasi.

Menurut Kent Beck XP adalah : "A lightweight, efficient, low-risk, flexible, predictable, scientific and fun way to develop software". Suatu model yang menekankan pada:

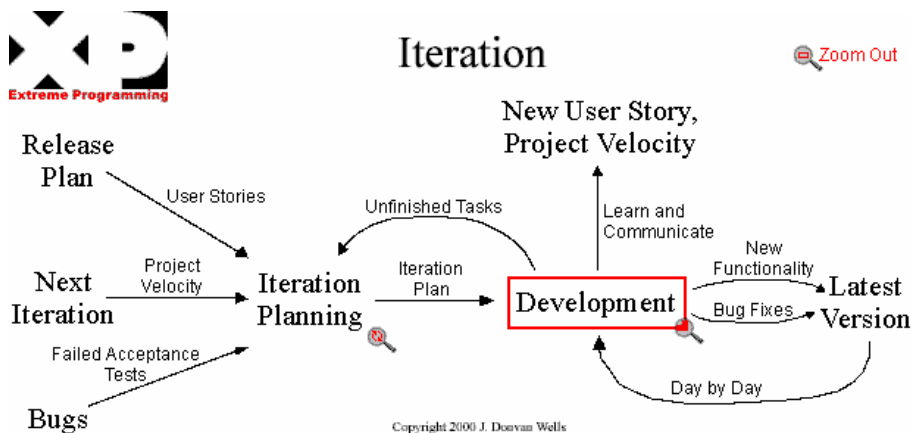
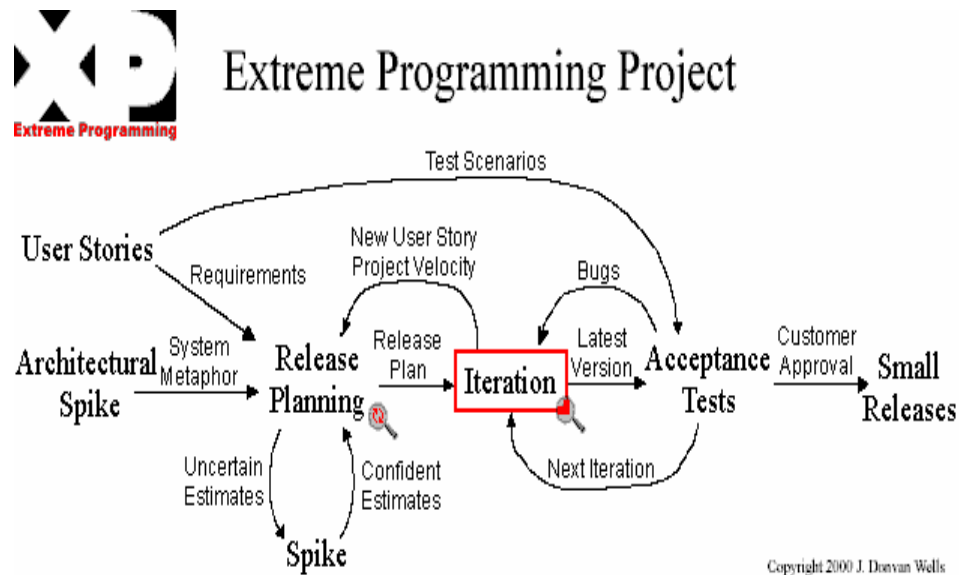
- keterlibatan user secara langsung
- pengujian
- pay-as-you-go design

Adapun empat nilai penting dari XP

1. Communication/Komunikasi : komunikasi antara developer dan klien sering menjadi masalah. Karena itu komunikasi dalam XP dibangun dengan melakukan pemrograman berpasangan (pair programming). Developer didampingi oleh pihak klien dalam melakukan coding dan unit testing sehingga klien bisa terlibat langsung dalam pemrograman sambil

berkomunikasi dengan developer. Selain itu perkiraan beban tugas juga diperhitungkan.

2. Simplicity/ sederhana: Menekankan pada kesederhanaan dalam pengkodean: "What is the simplest thing that could possibly work?" Lebih baik melakukan hal yang sederhana dan mengembangkannya besok jika diperlukan. Komunikasi yang lebih banyak mempermudah, dan rancangan yang sederhana mengurangi penjelasan.
3. Feedback / Masukan/Tanggapan: Setiap feed back ditanggapi dengan melakukan tes, unit test atau system integration dan jangan menunda karena biaya akan membengkak (uang, tenaga, waktu).
4. Courage / Berani: Banyak ide baru dan berani mencobanya, berani mengerjakan kembali dan setiap kali kesalahan ditemukan, langsung diperbaiki.



Penutup

Dari model-model di atas dan model-model yang akan dibahas kemudian, tidak ada satupun model yang cocok untuk semua jenis proyek pembuatan software. Penggunaan lebih dari satu model sangatlah dimungkinkan. Misalnya dalam model spiral dan model incremental dalam fase tertentu menggunakan model lain untuk mendapatkan hasil yang baik.

Diadaptasi dari:

1. Pressman, Roger.S. "Software Engineering : A Practioner's Approach." 4th . McGrawHill. 1997
2. Sommerville, Ian. "Software Engineering". 6th. Addison Wesley. 2001.
3. Beck, Kent. "Extreme Programming". www.extremeprogramming.org