

REVIEW METODOLOGI PENGEMBANGAN PERANGKAT LUNAK

Asep Herman Suyanto
asep-hs@mail.ugm.ac.id
<http://www.asep-hs.web.ugm.ac.id>

BAB I PENDAHULUAN

Software adalah perintah (program komputer) yang bila dieksekusi memberikan fungsi dan unjuk kerja seperti yang diinginkan. Struktur data yang memungkinkan program memanipulasi informasi secara proporsional, dan dokumen yang menggambarkan operasi dan kegunaan program.

Software memiliki dua peran, di satu sisi berfungsi sebagai sebuah produk dan di sisi lain sebagai kendaraan yang mengantarkan sebuah produk. Sebagai produk, *software* mengantarkan potensi perhitungan yang dibangun oleh *software* komputer. Baik di dalam sebuah telepon seluler, atau beroperasi di sebuah mainframe komputer, *software* merupakan transformer informasi yang memproduksi, mengatur, memperoleh, memodifikasi, menampilkan, atau memancarkan informasi, dimana pekerjaan ini dapat sesederhana suatu bit tunggal atau sekomples sebuah simulasi multimedia. Sedangkan peran sebagai kendaraan yang dipakai untuk mengantarkan produk, *software* berlaku sebagai dasar untuk kontrol komputer (sistem operasi), komunikasi informasi (jaringan), dan penciptaan serta kontrol dari program – program lain (peranti dan lingkungan *software*).

Pada masa-masa awal, pemrograman masih dilihat sebagai “bentuk kesenian”. Hanya sedikit saja metode yang ada dan lebih sedikit lagi orang yang memahaminya. Para pemrogram kadang – kadang harus mempelajarinya dengan coba-coba. Tetapi sekarang *software* sudah menjadi lahan yang sangat kompetitif. *Software* yang dulu dibangun secara internal di dalam komputer sekarang sudah dapat diproduksi secara terpisah. Perusahaan yang pada awalnya harus membayar sepasukan pemrogram untuk menghasilkan aplikasi tertentu, sekarang dapat mengambil tenaga dari luar dan dilakukan pada partai ketiga. Biaya, jangka waktu yang tidak terbatas, dan kualitas, merupakan pengendali utama yang membuat persaingan usaha *software* tidak pernah berhenti selama beberapa dekade terakhir.

Software secara umum dapat di bagi dua yaitu *software* sistem dan *software* aplikasi. *Software* sistem dapat di bagi lagi menjadi tiga macam yaitu :

1. Bahasa pemrograman

Bertugas mengkonversikan arsitektur dan algoritma yang di rancang manusia ke dalam format yang dapat di jalankan komputer, contoh bahasa pemrograman di antaranya : BASIC, COBOL, Pascal, C++, FORTRAN

2. Sistem Operasi

Saat komputer pertama kali di hidupkan, sistem operasilah yang pertama kali di jalankan, sistem operasi yang mengatur seluruh proses, menterjemahkan masukan, mengatur proses internal, memanejemen penggunaan memori dan

memberikan keluaran ke peralatan yang bersesuaian, contoh sistem operasi : DOS, Unix, Windows 95, IBM OS/2, Apple's System 7

3. Utility

Software sistem dengan fungsi tertentu, misalnya pemeriksaan perangkat keras (*hardware troubleshooting*), memeriksa disket yang rusak (bukan rusak fisik), mengatur ulang isi *harddisk* (partisi, defrag), contoh Utility adalah Norton Utility

Software aplikasi merupakan bagian *software* yang sangat banyak di jumpai dan terus berkembang. Sebelum tahun 1990-an aplikasi yang di kenal yaitu pemroses kata (Word Star, Chi Write), pemroses tabel (Lotus 123, Quatro Pro), database (DBASE), dan hiburan (game). Pada perkembangan pemroses kata, tabel dan database saat ini telah di bundel menjadi aplikasi *office* dengan tambahan aplikasi untuk pembuatan presentasi. Yang berkembang sangat banyak saat ini adalah aplikasi multimedia dan internet. Contoh aplikasi multimedia adalah Winamp untuk memutar musik berformat MP3 atau CD Audio, kemudian RealPlayer yang dapat digunakan untuk menonton film atau VCD. Aplikasi internet yang umum di gunakan adalah untuk browsing, e-mail, chatting dan messenger. Aplikasi yang bersifat khusus di antaranya untuk membantu pekerjaan Engineer seperti AutoCAD (gambar struktur), Protel (gambar rangkaian elektronik), dan Matlab (pemroses dan visualisasi persamaan matematis).

Software lebih merupakan elemen logika dan bukan merupakan elemen sistem fisik. *Software* memiliki ciri yang berbeda dari *Hardware*, yaitu :

1. *Software* dibangun dan dikembangkan, tidak dibuat dalam bentuk yang klasik.
2. *Software* tidak pernah usang.
3. Sebagian besar *Software* dibuat secara *custom-built*, serta tidak dapat dirakit dari komponen yang sudah ada.

Karakteristik dari produk *software* :

1. *Maintainability*
2. *Dependability*
3. *Efficiency*
4. *Usability*

Proses (produksi) *software*, yaitu:

1. Spesifikasi *software*
2. Pengembangan *software*
3. Validasi (pengetesan dan pengujian)
4. Evolusi, pengembangan lanjutan

IEEE telah mengembangkan definisi yang lebih komprehensif, yaitu sebagai berikut :

software engineering : aplikasi dari sebuah pendekatan kuantitatif, disiplin, dan sistematis kepada pengembangan, operasi, dan pemeliharaan *software*.

Model – model proses untuk *software engineering* seperti model sekuensial linier, model prototipe, model RAD, model inkremental, model spiral, model asembli komponen, model pengembangan kongkuren, model metode formal, model teknik generasi keempat.

BAB II MODEL SEKUENSIAL LINIER

Model sekuensial linier untuk *software engineering*, sering disebut juga dengan **siklus kehidupan klasik** atau **model air terjun**. Model ini mengusulkan sebuah pendekatan kepada perkembangan software yang sistematis dan sekuensial yang mulai pada tingkat dan kemajuan sistem pada seluruh analisis, desain, kode, pengujian, dan pemeliharaan. Dimodelkan setelah siklus rekayasa konvensional, model sekuensial linier melingkupi aktivitas – aktivitas sebagai berikut :

1. Rekayasa dan pemodelan sistem/informasi

Karena sistem merupakan bagian dari sebuah sistem yang lebih besar, kerja dimulai dengan membangun syarat dari semua elemen sistem dan mengalokasikan beberapa subset dari kebutuhan ke software tersebut. Pandangan sistem ini penting ketika software harus berhubungan dengan elemen-elemen yang lain seperti software, manusia, dan database. Rekayasa dan analisis sistem menyangkut pengumpulan kebutuhan pada tingkat sistem dengan sejumlah kecil analisis serta disain tingkat puncak. Rekayasa informasi mencakup juga pengumpulan kebutuhan pada tingkat bisnis strategis dan tingkat area bisnis.

2. Analisis kebutuhan Software

Proses pengumpulan kebutuhan diintensifkan dan difokuskan, khususnya pada software. Untuk memahami sifat program yang dibangun, analisis harus memahami domain informasi, tingkah laku, unjuk kerja, dan interface yang diperlukan. Kebutuhan baik untuk sistem maupun software didokumentasikan dan dilihat lagi dengan pelanggan.

3. Desain

Desain software sebenarnya adalah proses multi langkah yang berfokus pada empat atribut sebuah program yang berbeda; struktur data, arsitektur software, representasi interface, dan detail (algoritma) prosedural. Proses desain menterjemahkan syarat/kebutuhan ke dalam sebuah representasi software yang dapat diperkirakan demi kualitas sebelum dimulai pemunculan kode. Sebagaimana persyaratan, desain didokumentasikan dan menjadi bagian dari konfigurasi software.

4. Generasi Kode

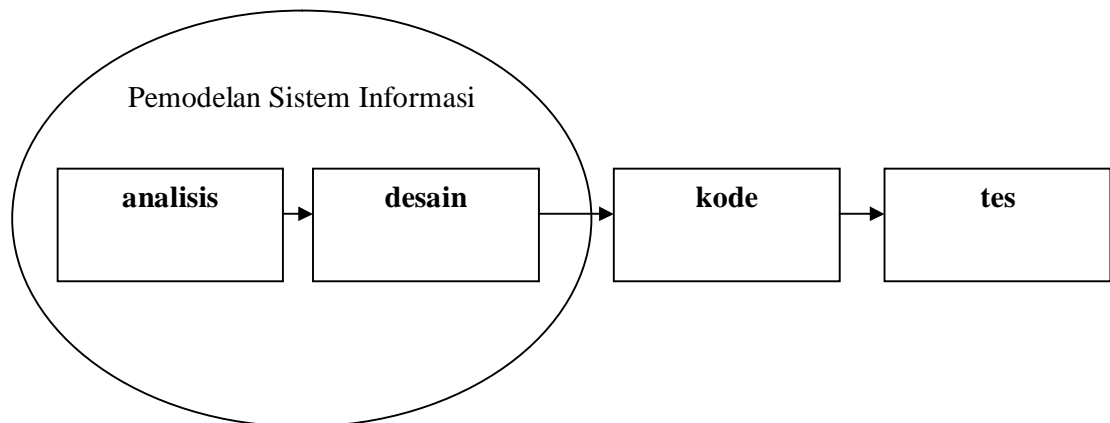
Desain harus diterjemahkan kedalam bentuk mesin yang bisa dibaca. Langkah pembuatan kode melakukan tugas ini. Jika desain dilakukan dengan cara yang lengkap, pembuatan kode dapat diselesaikan secara mekanis.

5. Pengujian

Sekali program dibuat, pengujian program dimulai. Proses pengujian berfokus pada logika internal software, memastikan bahwa semua pernyataan sudah diuji, dan pada eksternal fungsional, yaitu mengarahkan pengujian untuk menemukan kesalahan – kesalahan dan memastikan bahwa input yang dibatasi akan memberikan hasil aktual yang sesuai dengan hasil yang dibutuhkan.

6. **Pemeliharaan**

Software akan mengalami perubahan setelah disampaikan kepada pelanggan (perkecualian yang mungkin adalah software yang dilekatkan). Perubahan akan terjadi karena kesalahan – kesalahan ditentukan, karena software harus disesuaikan untuk mengakomodasi perubahan – perubahan di dalam lingkungan eksternalnya (contohnya perubahan yang dibutuhkan sebagai akibat dari perangkat peripheral atau sistem operasi yang baru), atau karena pelanggan membutuhkan perkembangan fungsional atau unjuk kerja. Pemeliharaan software mengaplikasikan lagi setiap fase program sebelumnya dan tidak membuat yang baru lagi.



Gambar 2.1 Model Sekuensial Linier

Masalah yang kadang terjadi ketika model sekuensial linier diaplikasikan adalah :

1. Jarang sekali proyek nyata mengikuti aliran sekuensial yang dianjurkan oleh model. Meskipun model linier bisa mengakomodasi iterasi, model ini melakukannya dengan cara tidak langsung. Sebagai hasilnya, perubahan – perubahan dapat menyebabkan keraguan pada saat tim proyek berjalan.
2. Kadang – kadang sulit bagi pelanggan untuk menyatakan semua kebutuhannya secara eksplisit. Model linier sekuensial memerlukan hal ini dan mengalami kesulitan untuk mengakomodasi ketidakpastian natural yang ada pada bagian awal beberapa proyek.

3. Pelanggan harus bersifat sabar. Sebuah versi kerja dari program – program kerja itu tidak akan diperoleh sampai akhir waktu proyek dilalui. Sebuah kesalahan besar, jika tidak terdeteksi sampai program yang bekerja tersebut dikaji ulang, bisa menjadi petaka.
4. Pengembang sering melakukan penundaan yang tidak perlu. Sifat alami dari siklus kehidupan klasik membawa kepada *blocking state* di mana banyak anggota tim proyek harus menunggu tim yang lain untuk melengkapinya tugas yang saling memiliki ketergantungan. Blocking state cenderung menjadi lebih lazim pada awal dan akhir sebuah proses sekuensial linier.

BAB III MODEL PROTOTIPE

Prototyping paradigma dimulai dengan pengumpulan kebutuhan. Pengembang dan pelanggan bertemu dan mendefinisikan obyektif keseluruhan dari software, mengidentifikasi segala kebutuhan yang diketahui, dan area garis besar dimana definisi lebih jauh merupakan keharusan kemudian dilakukan “perancangan kilat”. Perancangan kilat berfokus pada penyajian dari aspek – aspek software tersebut yang akan nampak bagi pelanggan atau pemakai (contohnya pendekatan input dan format output). Perancangan kilat membawa kepada konstruksi sebuah prototipe. Prototipe tersebut dievaluasi oleh pelanggan/pemakai dan dipakai untuk menyaring kebutuhan pengembangan software. Iterasi terjadi pada saat prototipe disetel untuk memenuhi kebutuhan pelanggan, dan pada saat yang sama memungkinkan pengembang untuk secara lebih baik memahami apa yang harus dilakukannya.

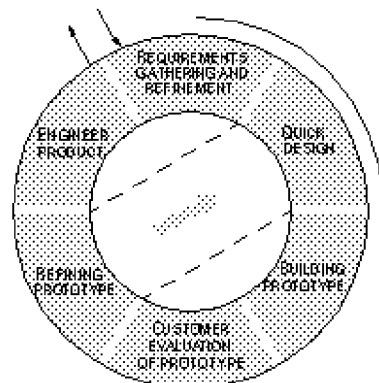


figure 3. The prototyping paradigm of software development

Sumber : http://www.ludd.luth.se/users/no/os_meth.8.html

Gambar 3.1 Prototipe Paradigma

Secara ideal prototipe berfungsi sebagai sebuah mekanisme untuk mengidentifikasi kebutuhan software. Bila prototipe yang sedang bekerja dibangun, pengembang harus mempergunakan fragmen – fragmen program yang ada atau mengaplikasikan alat – alat bantu (contohnya report generator, window

manager, dll) yang memungkinkan program yang bekerja untuk dimunculkan secara cepat.

Prototipe bisa juga menjadi masalah karena alasan sebagai berikut:

1. Pelanggan melihat apa yang tampak sebagai versi software yang bekerja tanpa melihat bahwa prototipe itu dijalin bersama – sama “dengan permen karet dan *baling wire*”, tanpa melihat bahwa di dalam untuk membuatnya bekerja, kita belum menyantumkan kualitas software secara keseluruhan atau kemampuan pemeliharaan untuk jangka waktu yang panjang. Ketika diberi informasi bahwa produk harus dibangun lagi agar tingkat kualitas yang tinggi bisa dijaga, pelanggan akan meneriakkan kecurangan dan permintaan agar dipakai “beberapa campuran” untuk membuat prototipe menjadi sebuah produk yang bekerja yang lebih sering terjadi, sehingga manajemen pengembangan software menjadi penuh dengan belas kasihan.
2. Pengembang sering membuat kompromi – kompromi implementasi untuk membuat prototipe bekerja dengan cepat. Sistem operasi atau bahasa pemrograman yang tidak sesuai bisa dipakai secara sederhana karena mungkin diperoleh dan dikenal; algoritma yang tidak efisien secara sederhana bisa diimplementasikan untuk mendemonstrasikan kemampuan. Setelah selang waktu tertentu, pengembang mungkin mengenali pilihan – pilihan tersebut dan melupakan semua alasan mengapa mereka tidak cocok. Pilihan yang kurang ideal telah menjadi bagian integral dari sebuah sistem.

Meskipun berbagai masalah bisa terjadi, prototipe bisa menjadi paradigma yang efektif bagi *Software Engineering*. Kuncinya adalah mendefinisikan aturan main pada saat awal; yaitu pelanggan dan pengembang keduanya harus setuju bahwa prototipe dibangun untuk berfungsi sebagai mekanisme pendefinisian kebutuhan. Prototipe kemudian disingkirkan (paling tidak sebagian), dan software aktual direkayasa dengan tertuju kepada kualitas dan kemampuan pemeliharaan.

BAB IV MODEL RAPIN APLICATION DEVELOPMENT

Rapin Application Development (RAD) adalah sebuah model proses perkembangan software sekuensial linier yang menekankan siklus perkembangan yang sangat pendek. Model RAD ini merupakan sebuah adaptasi “kecepatan tinggi” dari model sekuensial linier di mana perkembangan cepat dicapai dengan menggunakan pendekatan konstruksi berbasis komponen. Jika kebutuhan dipahami dengan baik, proses RAD memungkinkan tim pengembangan menciptakan “sistem fungsional yang utuh” dalam periode waktu yang sangat pendek (kira-kira 60 sampai 90 hari). Karena dipakai terutama pada aplikasi sistem konstruksi, pendekatan RAD melingkupi fase – fase sebagai berikut :

1. Bussiness modeling

Aliran informasi di antara fungsi – fungsi bisnis dimodelkan dengan suatu cara untuk menjawab pertanyaan – pertanyaan berikut : informasi apa yang mengendalikan proses bisnis? Informasi apa yang di munculkan?

Siapa yang memunculkannya? Ke mana informasi itu pergi? Siapa yang memprosesnya?

2. Data modeling

Aliran informasi yang didefinisikan sebagai bagian dari fase bussiness modelling disaring ke dalam serangkaian objek data yang dibutuhkan untuk menopang bisnis tersebut. Karakteristik (disebut atribut) masing – masing objek diidentifikasi dan hubungan antara objek – objek tersebut didefinisikan.

3. Proses modelling

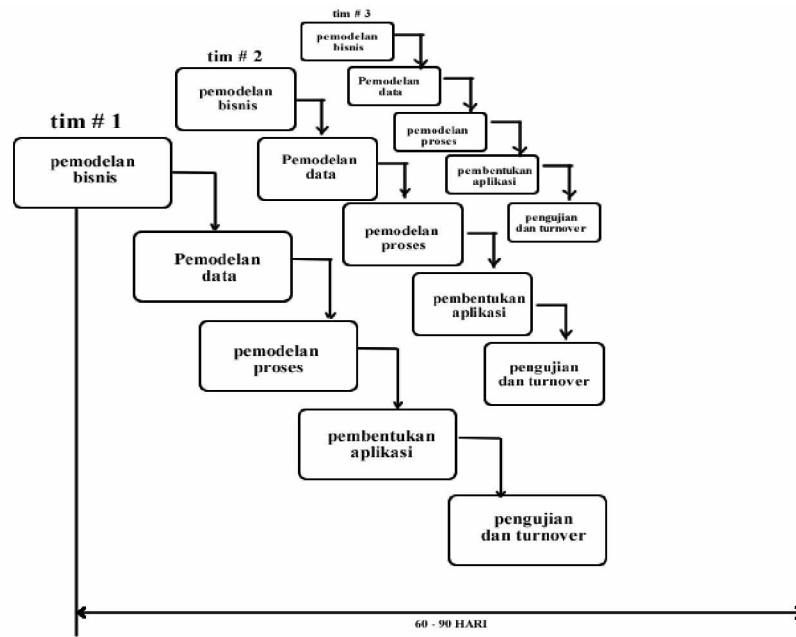
Aliran informasi yang didefinisikan di dalam fase data modeling ditransformasikan untuk mencapai aliran informasi yang perlu bagi implementasi sebuah fungsi bisnis. Gambaran pemrosesan diciptakan untuk menambah, memodifikasi, menghapus, atau mendapatkan kembali sebuah objek data.

4. Application generation

RAD mengasumsikan pemakaian teknik generasi ke empat. Selain menciptakan perangkat lunak dengan menggunakan bahasa pemrograman generasi ketiga yang konvensional, RAD lebih banyak memproses kerja untuk memkai lagi komponen program yang ada (pada saat memungkinkan) atau menciptakan komponen yang bisa dipakai lagi (bila perlu). Pada semua kasus, alat – alat bantu otomatis dipakai untuk memfasilitasi konstruksi perangkat lunak.

5. Testing and turnover

Karena proses RAD menekankan pada pemakaian kembali, banyak komponen program telah diuji. Hal ini mengurangi keseluruhan waktu pengujian. Tetapi komponen baru harus di uji dan semua interface harus dilatih secara penuh.



Gambar 4.1 Model RAD

Kekurangan model RAD adalah :

1. Bagi proyek yang besar tetapi berskala, RAD memerlukan sumber daya manusia yang memadai untuk menciptakan jumlah tim RAD yang baik.
2. RAD menuntut pengembangan dan pelanggan memiliki komitmen di dalam aktivitas *rapid-fire* yang diperlukan untuk melengkapi sebuah sistem, di dalam kerangka waktu yang sangat diperpendek. Jika komitmen tersebut tidak ada, proyek RAD akan gagal.

BAB V MODEL SPIRAL

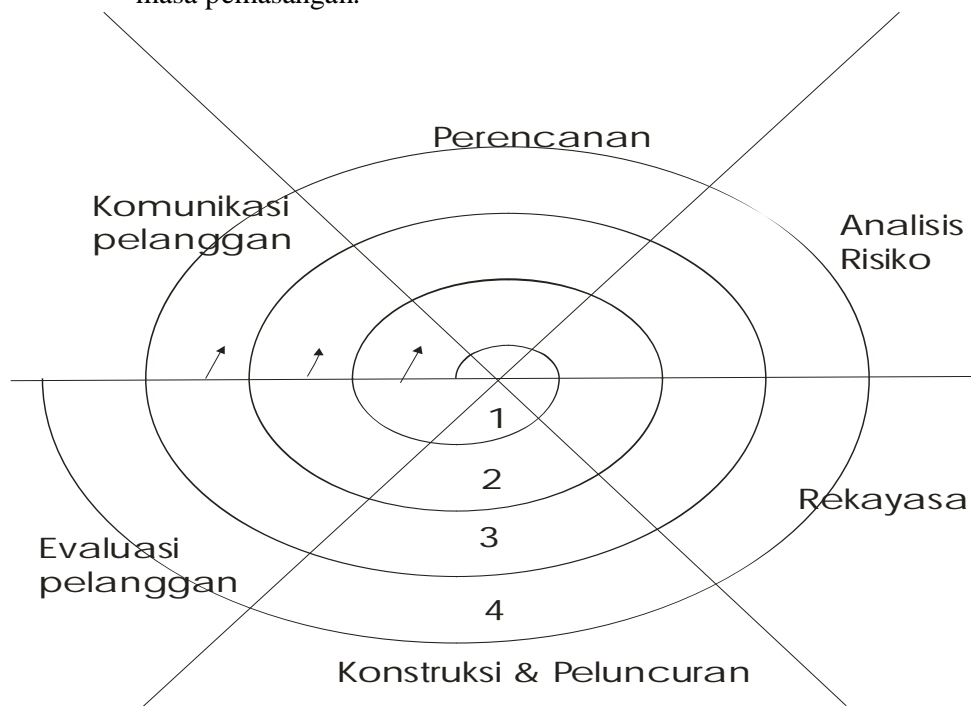
Model spiral (*spiral model*) adalah model proses software yang evolusioner yang merangkai sifat iteratif dari prototipe dengan cara kontrol dan aspek sistematis dari model sekuensial linier. Model ini berpotensi untuk pengembangan versi pertambahan software secara cepat. Di dalam model spiral, software dikembangkan di dalam suatu deretan pertambahan. Selama awal iterasi, rilis inkremental bisa merupakan sebuah model atau prototipe kertas. Selama iterasi berikutnya, sedikit demi sedikit dihasilkan versi sistem rekayasa yang lebih lengkap.

Model spiral dibagi menjadi sejumlah aktifitas kerangka kerja, disebut juga wilayah tugas, di antara tiga sampai enam wilayah tugas, yaitu :

1. Komunikasi Pelanggan
Tugas – tugas yang dibutuhkan untuk membangun komunikasi yang efektif di antara pengembangan dan pelanggan.
2. Perencanaan

Tugas – tugas yang dibutuhkan untuk mendefinisikan sumber – sumber daya, ketepatan waktu, dan proyek informasi lain yang berhubungan.

3. Analisis Risiko
Tugas – tugas yang dibutuhkan untuk menaksir risiko – risiko, baik manajemen maupun teknis.
4. Perencanaan
Tugas – tugas yang dibutuhkan untuk membangun satu atau lebih representasi dari aplikasi tersebut.
5. Konstruksi dan peluncuran
Tugas – tugas yang dibutuhkan untuk mengkonstruksi, menguji, memasang (instal) dan memberikan pelayanan kepada pemakai (contohnya pelatihan dan dokumentasi).
6. Evaluasi pelanggan
Tugas – tugas yang dibutuhkan untuk memperoleh umpan balik dari pelanggan dengan didasarkan pada evaluasi representasi software, yang dibuat selama masa perancangan, dan diimplementasikan selama masa pemasangan.



- 1 = proyek pengembangan konsep
- 2 = proyek pengembangan produk baru
- 3 = proyek perbaikan produk
- 4 = proyek pemeliharaan produk

Gambar 5.1 Model spiral yang disesuaikan untuk siklus hidup bagian dalam.

Kekurangan model spiral adalah sulitnya untuk meyakinkan konsumen (khususnya dalam situasi kontrak) bahwa pendekatan evolusioner bisa dikontrol. Model spiral memerlukan keahlian penaksiran risiko yang masuk akal, dan sangat bertumpu pada keahlian ini untuk mencapai keberhasilan. Jika risiko mayor tidak ditemukan dan diatur, pasti akan terjadi masalah. Akhirnya model itu sendiri masih baru dan belum dipergunakan secara luas seperti paradigma sekuensial dan prototipe.

BAB VI KESIMPULAN

Software dapat menjadi elemen kunci bagi evolusi sistem dan produk yang berbasis komputer. Selama empat dekade terakhir, *software* telah berkembang dari sebuah alat analisis dan pemecahan masalah yang terspesialisasi di dalam industri itu sendiri. Tetapi budaya dan sejarah pemrograman sebelumnya telah menciptakan serangkaian masalah yang sekarang muncul. *Software* telah menjadi faktor pembatas dalam evolusi sistem berbasis komputer. *Software* dirancang dari program-program, data, dan dokumen. Masing – masing dari item tersebut terdiri dari sebuah konfigurasi yang diciptakan sebagai bagian dari proses pengembangan *software*. Tujuan *software engineering* adalah menyediakan sebuah kerangka kerja guna membangun *software* dengan kualitas yang lebih tinggi.

Software engineering adalah sebuah disiplin ilmu yang mengintegrasikan proses, metode, dan alat – alat bantu bagi perkembangan proses perangkat lunak komputer. Sejumlah model proses yang berbeda untuk *software engineering* telah diusulkan, dan masing – masing mengungkapkan kelemahan dan kekuatan mereka, yang semuanya memiliki sederetan fase generik secara umum. Model – model proses untuk *software engineering* seperti model sekuensial linier, model prototipe, model RAD, model inkremental, model spiral, model assembli komponen, model pengembangan kongkuren, model metode formal, model teknik generasi keempat.

DAFTAR PUSTAKA

- Ivan Sudirman, *Perkembangan Software Komputer*, 2003, <http://www.ilmukomputer.com> (12 juli 2003).
- Pressman, Roger S, *Software Engineering : A Practitioner's Approach*, McGraw-Hill Companies, Inc. 1997.
- Software engineering methodology , http://www.ludd.luth.se/users/no/os_meth.8.html ,(20 Februari 2004).
- Software Development Methodology, http://www.hyperhot.com/pm_sdm.htm, (20 Februari 2004).