PERANCANGAN DAN PENGEMBANGAN SISTEM INFORMASI

Oleh :

AHMAD HIDAYAT

Artikel
Universitas Negeri Medan – Fakultas Ekonomi – Bisnis Digital
Surel : ahmadcontech2020@gmail.com

A. Pengertian Sistem dan Sistem Informasi

Sistem dapat diartikan bermacam-macam, tergantung dari latar belakang dan sudut pandang yang mengemukakannya. Ada yang mengartikan bahwa sistem itu hanya sekedar ‘metode’ atau ‘cara’, ada pula yang mengartikan sistem itu sebagai ‘komponen’, dan ada juga yang mengartikan sebagai ‘prosedur’. Faktanya, istilah sistem memiliki penerapan yang lebih luas. Sistem atom, sistem elektron, proton, dan neutron, hingga alam semesta; serta semua bentuk kehidupan tumbuhan dan hewan, adalah contoh sistem alam. Sistem buatan manusia bisa mencakup segala hal, mulai dari jam tangan, hingga peralatan canggih lainnya, dan sistem sosial, sampai pada sistem informasi.

Terlepas dari berbagai sudut pandangnya, semua sistem memiliki beberapa elemen atau unsur yang sama. Hall (2011: 5) mengartikan bahwa “Suatu sistem adalah sekelompok dari dua atau lebih komponen atau subsistem yang saling terkait dalam melayani pekerjaan untuk mencapai tujuan yang sama .” Beberapa literatur terkait dengan pengelolaan bisnis, untuk istilah ‘komponen’ lebih memberikan tekanan dengan isitilah *processing procedures* atau ‘prosedur-prosedur yang digunakan untuk pemrosesan’ dapat dalam bentuk aliran data, barang, dan sumber daya lainnya.

Penekanan bahwa sistem adalah prosedur yang saling berkaitan satu dengan yang lainnya dan digunakan untuk memproses data atau sumber daya untuk mencapai tujuan bersama, diantaranya diungkapkan oleh Simkin, Rose, dan Norman (2012 : 8); Jerry Fitz Gerald, Arda F. Fitz Gerald, dan Warren D. Stallings dalam Jugianto (2005 : 1). Pendapat para ahli ini juga disampaikan oleh Hall (2011 : 7) dan menyebutnya sebagai sistem informasi yaitu “Sistem informasi adalah sekumpulan prosedur formal dimana data dikumpulkan, diproses menjadi informasi, dan didistribusikan kepada pengguna.” Selanjutnya Hall (2011: 9) menjelaskan bahwa :

Nilai informasi bagi pengguna ditentukan oleh keandalannya. Tujuan informasi adalah mengarahkan pengguna ke tindakan yang diinginkan. Agar ini terjadi, informasi harus memiliki atribut yang pasti dalam hal relevansi, akurasi, kelengkapan, peringkasan, dan ketepatan waktu. Saat atribut ini ada hadir secara konsisten, informasi memiliki keandalan dan memberikan nilai kepada pengguna. Penyajian informasi yang tidak memperhatikan atribut tersebut, tidak akan bernilai dan hanya akan membuang-buang sumber daya, dan pada akhirnya dapat menyebabkan pengambilan keputusan yang tidak berfungsi.

Richard F. Neusche dalam Jugianto (2005 : 1) mendefinisikan bahwa "Suatu prosedur adalah urutan-urutan operasi klerikal (tulis menulis), biasanya melibatkan orang di dalam satu atau lebih departemen, yang diterapkan untuk menjamin penanganan yang seragam dari transaksi-transaksi bisnis yang terjadi." Prosedur dalam sebuah sistem dapat dikatakan sebagai subsistem dari sistem itu sendiri. Hall (2011 : 9) memberikan contoh bahwa dalam pengelolaan sistem informasi (*Management Information System*) dalam entitas bisnis bisa ditemukan fungsi-fungsi yang meliputi area keuangan, pemasaran, distribusi, dan personalia. Fungsi keuangan dapat meliputi aplikasi fortofolio, penganggaran, dan pencatatan keuangan; fungsi distribusi meliputi jadwal penghantaran, model pemuatan dan alokasi kendaraan, serta pengelolaan gudang; fungsi pemasaran meliputi analisis pasar, pengembangan produk baru, dan analisis produk; fungsi personalia atau *human resources manajemen* meliputi perekrutan dan penempatan pegawai, pelatihan dan pengembangan, gaji-upah-tunjangan pegawai, dan mutasi atau pemberhentian pegawai.

Sesuai dengan topik dan masalah yang dikaji dalam penelitian ini, selanjutnya akan dibahan mengenai sistem yang berhubungan dengan pengelolaan keuangan dan pengelolaan pegawai.

B. Model Pengembangan Manajemen Proyek

Bagian ini mengarah pada bagaimana manajemen proyek perangkat lunak dapat menciptakan produk akhir berupa software secara efektif. Analisis terperinci mengenai tentang persyaratan pembangunan proyek ini menjadi informasi yang akan memandu berjalannya proyek dan terciptanya produk perangkat lunak.

Aktivitas manajemen proyek perangkat lunak yang pertama adalah penentuan ruang lingkup perangkat lunak. Cakupan pengembangan proyek perangkat lunak menurut Pressman (2001 : 67) harus menjawab pertanyaan-pertanyaan berikut:

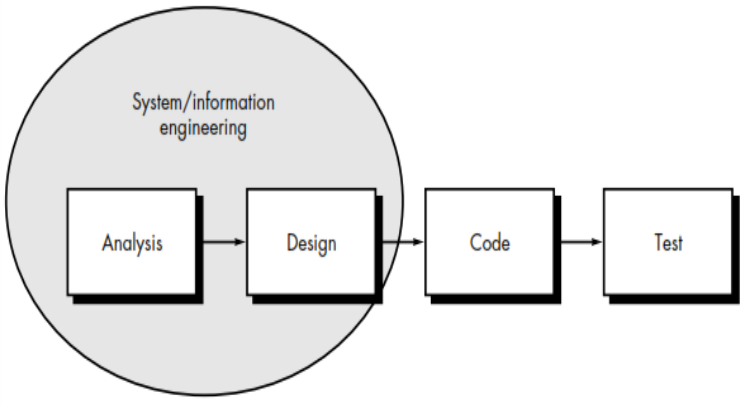
1. Konteks. Bagaimana perangkat lunak yang akan dibangun cocok menjadi sistem yang lebih besar, produk, atau konteks bisnis dan kendala apa yang dikenakan sebagai akibat dari konteks tersebut?
2. Tujuan informasi. Apakah objek data yang terlihat oleh pelanggan diproduksi sebagai keluaran dari perangkat lunak? Objek data apa yang diperlukan untuk input?
3. Fungsi dan kinerja. Fungsi apa yang dilakukan perangkat lunak tersebut mengubah data masukan menjadi keluaran? Apakah ada karakteristik kinerja khusus untuk ditangani?

Selanjutnya Pressman (2001 : 68) mengemukakan bahwa fase umum yang menjadi ciri proses pembuatan perangkat lunak dapat mengikuti berbagai pola atau model sesuai dengan rekayasa yang akan dikembangkan, yaitu :

1. *The linear sequential model* (model sekuensial linier)
2. *The prototyping model* (model pembuatan prototipe)
3. *The RAD model* (model RAD)
4. *The incremental model* (model inkremental)
5. *The spiral model* (model spiral)
6. *The WINWIN spiral model* (model spiral WINWIN)
7. *The concurrent development model* (model pengembangan konkuren)
8. *The component-based development model* (model pembangunan berbasis komponen)
9. *The formal methods model* (model metode formal)
10. *The fourth generation techniques model* (model teknik generasi keempat)

B.1 *The Linear Sequential Model* (Model Sekuensial Linier)

The linear sequential model (model sekuensial linier), terkadang disebut *classic life cycle* (siklus hidup klasik) atau *the waterfall model* (model air terjun), menyarankan pendekatan yang sistematis dan berurutan pengembangan perangkat lunak yang dimulai pada tingkat sistem dengan tahapan analisis, desain, pengkodean, dan pengujian. Untuk selanjutnya istilah model sekuensial linier ini akan disebut dengan *waterfall model* atau model air terjun.



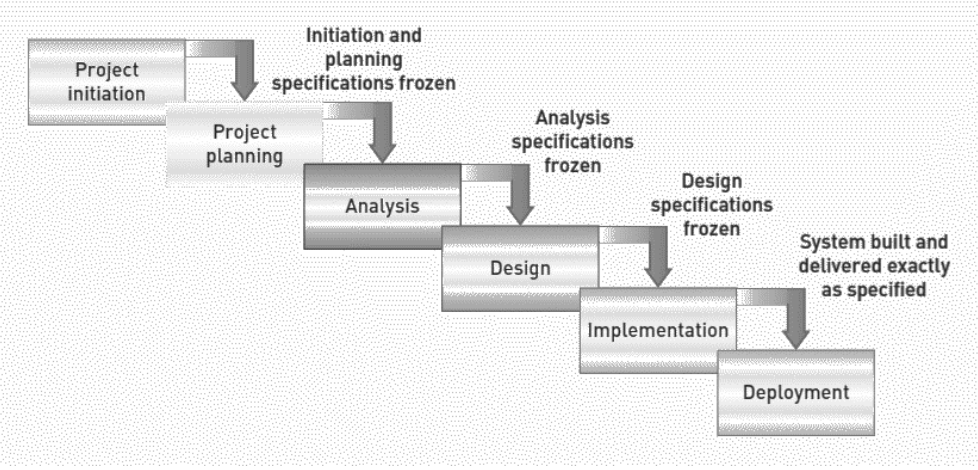
Sumber : Pressman (2001 : 29)

Gambar 1. Waterfall Model-1

Gambar *waterfall model* kemudian dikembangkan oleh Satzinger, Jackson, dan Burd (2016 : 299) menjelaskan bahwa :

> Model ini mengasumsikan bahwa tahapan dapat dilakukan dan diselesaikan secara berurutan. Pertama, rencana terperinci dikembangkan, kemudian persyaratan ditentukan secara menyeluruh, kemudian sistem dirancang hingga algoritme terakhir, dan kemudian diprogram, diuji, dan dipasang. Setelah sebuah proyek turun di atas air terjun ke fase berikutnya, tidak ada jalan untuk kembali. Dalam praktiknya, model air terjun mengasumsikan perencanaan yang kaku dan pengambilan keputusan akhir di setiap langkah proyek pengembangan. Seperti yang Anda duga, model air terjun tidak selalu berfungsi dengan baik. Sebagai manusia, pengembang jarang dapat menyelesaikan suatu fase tanpa membuat kesalahan atau mengabaikan komponen penting yang harus ditambahkan nanti. Namun, meskipun model air terjun tidak lagi digunakan dalam bentuknya yang paling murni, model ini tetap memberikan dasar yang berharga untuk pemahaman pengembangan. Tidak peduli sistem apa yang sedang dikembangkan, Anda perlu memasukkan inisiasi, perencanaan, analisis, desain, implementasi, dan penerapan kegiatan.

Selanjutnya Satzinger, Jackson, dan Burd memperbaiki *waterfall model* menjadi :



Gamber 2. Waterfall Model-2

Berikut ini penjelasan dari *waterfall model* :

1. *System/information engineering and modeling* (rekayasa dan pemodelan sistem/informasi). Karena perangkat lunak selalu merupakan bagian dari sistem atau bisnis yang lebih besar, pekerjaan dimulai dengan menetapkan persyaratan untuk semua elemen sistem dan kemudian mengalokasikan beberapa bagian dari persyaratan ini ke perangkat lunak. Tampilan sistem ini penting ketika perangkat lunak harus berinteraksi dengan elemen lain seperti perangkat keras, orang, dan database. Rekayasa dan analisis sistem mencakup pengumpulan persyaratan di tingkat sistem

dengan sedikit desain dan analisis tingkat atas. Rekayasa informasi mencakup pengumpulan persyaratan di tingkat bisnis strategis dan di tingkat area bisnis.

2. *Project initiation* (inisiasi proyek) dan *project planning* (rencana proyek) yang dilanjutkan dengan kegiatan s*oftware requirements analysis* (analisis kebutuhan perangkat lunak). Untuk memahami sifat program yang akan dibangun, analis (pengembang) harus memahami domain informasi serta fungsi, perilaku, kinerja, dan antarmuka yang diperlukan. Persyaratan untuk sistem dan perangkat lunak didokumentasikan dan ditinjau bersama pelanggan.
3. *Design* (rancangan). Desain perangkat lunak sebenarnya adalah proses multistep yang berfokus pada empat atribut berbeda dari sebuah program: struktur data, arsitektur perangkat lunak, representasi antarmuka, dan detail prosedural (algoritmik). Proses desain menerjemahkan persyaratan menjadi representasi perangkat lunak yang dapat dinilai kualitasnya sebelum pengkodean dimulai. Seperti persyaratan, desain didokumentasikan dan menjadi bagian dari konfigurasi perangkat lunak.
4. Code generation (pembuatan kode). Desain harus diterjemahkan ke dalam bentuk yang dapat dibaca mesin. Langkah pembuatan kode melakukan tugas ini. Jika desain dilakukan secara rinci, pembuatan kode dapat dilakukan secara mekanis.
5. *Testing* (pengujian) dan *implementation* (implementasi). Setelah kode dibuat, pengujian program dimulai, ini dilakukan oleh tim pengembang dengan programmer. Proses pengujian berfokus pada internal logis dari perangkat lunak, memastikan bahwa semua pernyataan telah diuji, dan pada eksternal fungsional; yaitu, melakukan pengujian untuk mengungkap kesalahan dan memastikan bahwa input yang ditentukan akan menghasilkan hasil aktual yang sesuai dengan hasil yang diminta. setelah pengujian dilakukan berhasil, tahap selanjutnya adalah implementasi yaitu melakukan uji-coba melalui user sebagai sampel. Perubahan akan terjadi karena kesalahan telah ditemukan dalam tahap emplementasi , karena perangkat lunak harus disesuaikan untuk mengakomodasi perubahan di lingkungan eksternalnya (misalnya, perubahan yang diperlukan karena sistem operasi atau perangkat periferal baru), atau karena pelanggan memerlukan peningkatan fungsional atau kinerja. Dukungan atau pemeliharaan perangkat lunak menerapkan kembali setiap fase sebelumnya ke program yang sudah ada daripada yang baru.

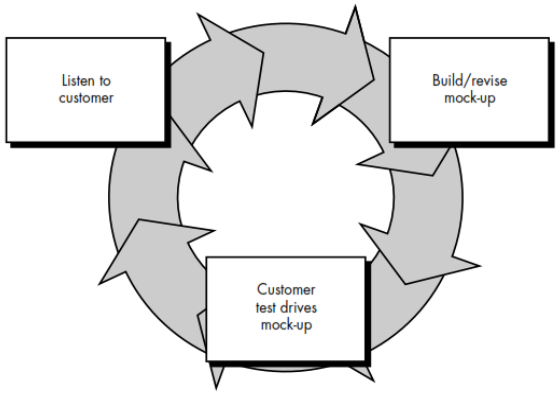
Waterfall model adalah paradigma tertua dan paling banyak digunakan untuk rekayasa perangkat lunak. Namun, kritik terhadap paradigma tersebut bahkan telah menyebabkan pendukung aktif mempertanyakan kemanjurannya yang tulis Hanna (1994 : 38-46) dengan artikel "*Farewell to Waterfalls*" pada dalam majalah *Sofware Magazine*. Masalah yang terkadang ditemui ketika w*aterfall model* diterapkan adalah:

1. Proyek nyata jarang mengikuti aliran sekuensial yang diusulkan dalam model. Meskipun *waterfall model* dapat mengakomodasi iterasi, ia melakukannya secara tidak langsung. Akibatnya, perubahan dapat menyebabkan kebingungan saat tim proyek melanjutkan.
2. Seringkali sulit bagi pelanggan untuk menyatakan semua persyaratan secara eksplisit. *Waterfall model* membutuhkan ini dan memiliki kesulitan untuk mengakomodasi ketidakpastian alami yang ada pada awal banyak proyek.
3. Pelanggan harus memiliki kesabaran. Versi kerja dari program tidak akan tersedia hingga akhir jangka waktu proyek. Kesalahan besar, jika tidak terdeteksi sampai program kerja ditinjau ulang, bisa menjadi bencana.

Dalam analisis menarik dari proyek aktual, Bradac (1994 : 774-784) menemukan bahwa sifat linier dari *waterfall model* mengarah ke "keadaan pemblokiran" di mana beberapa anggota tim proyek harus menunggu anggota lain dari tim untuk menyelesaikan tugas yang bergantung. Padahal, waktu yang dihabiskan untuk menunggu bisa melebihi waktu yang dihabiskan untuk pekerjaan produktif. Status pemblokiran cenderung lebih umum di awal dan akhir proses model air terjun. Walaupun demikian *waterfall model* tetap menjadi model prosedural yang banyak digunakan untuk rekayasa perangkat lunak. Meskipun memiliki kelemahan, ini secara signifikan lebih baik daripada pendekatan sembarangan untuk pengembangan perangkat lunak.

B.2 *The Prototyping Model / Mock-Up Model* (Model Pembuatan Prototipe)

Mock-Up Model dimulai dengan pengumpulan kebutuhan. Pengembang dan pelanggan bertemu dan merumuskan tujuan pembuatan perangkat lunak, mengidentifikasi persyaratan apa pun yang diketahui, dan garis besar pekerjaan yang harus dilakukan. Kemudian pelanggan membuat *mock-up* (visualisasi sebuah konsep desain web). *Mock-up* berfokus pada representasi aspek perangkat lunak yang akan dilihat oleh pelanggan/pengguna (misalnya, memasukkan pendekatan dan format keluaran) sehingga apabila diperlukan pelanggan/pengguna dapat memberikan masukkan kepada pengembang, sehingga pada saat yang sama pengembang lebih memahami apa yang diinginkan oleh pelanggan/pengguna.

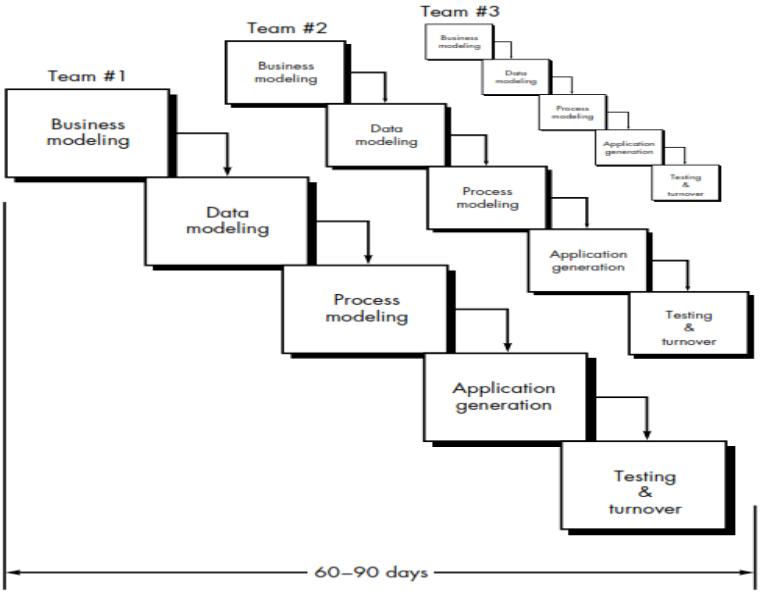


Sumber : Pressman (2001 : 31)

Gambar 3. Mock-Up Model

B.3 *The RAD Model (*Model Pengembangan Aplikasi Cepat)

Rapid Application Development (RAD) Model atau model pengembangan aplikasi cepat adalah model proses pengembangan perangkat lunak tambahan yang menekankan pada siklus pengembangan yang sangat singkat. *RAD model* adalah adaptasi "kecepatan tinggi" dari *waterfall model* di mana perkembangan cepat dicapai dengan menggunakan konstruksi berbasis komponen. Jika persyaratan dipahami dengan baik dan cakupan proyek dibatasi, proses *RAD model* memungkinkan tim pengembangan untuk membuat "sistem yang berfungsi penuh" dalam periode waktu yang sangat singkat (misalnya, 60 hingga 90 hari).



Sumber : Pressman (2001 : 33)

Gambar 4. RAD Model (Model Pengembangan Aplikasi Cepat)

Pressman menjelaskan bahwa *RAD model* digunakan untuk membangun aplikasi sistem informasi, mencakup tahapan sebagai berikut :

1. *Business modeling* (pemodelan bisnis). Aliran informasi antar fungsi bisnis dimodelkan sedemikian rupa sehingga menjawab pertanyaan-pertanyaan berikut: Informasi apa yang mendorong proses bisnis? Informasi apa yang dihasilkan? Siapa yang menghasilkannya? Kemana informasi dialirkan? Siapa yang memprosesnya?
2. *Data* Modeling (pemodelan data). Aliran informasi didefinisikan sebagai bagian dari fase pemodelan bisnis dan disempurnakan menjadi sekumpulan objek data yang diperlukan untuk mendukung bisnis. Karakteristik atau atribut dari setiap objek diidentifikasi dan hubungan antara objek-objek untuk didefinisikan.
3. *Process Modeling* (pemodelan proses). Objek data yang didefinisikan dalam fase pemodelan data diubah untuk mencapai arus informasi yang diperlukan untuk mengimplementasikan fungsi bisnis. Uraian promrosesan atau pengolahan data deskripsi untuk menambahkan, mengubah, menghapus, atau mengambil objek data.
4. *Application Generation* (pembuatan aplikasi). Model RAD menekankan penggunaan kembali komponen program yang ada bila memungkinkan atau membuat komponen yang dapat digunakan untuk mendukung komponen yang telah ada.
5. *Testing and* turnover (pengujian dan pergantian). Karena proses RAD lebih menekankan pada penggunaan komponen yang masih dapat digunakan dengan menambah komponen baru apabila diperlukan, ini akan mengurangi waktu pengujian secara keseluruhan. Namun, komponen baru harus diuji dan semua antarmuka harus dijalankan sepenuhnya.

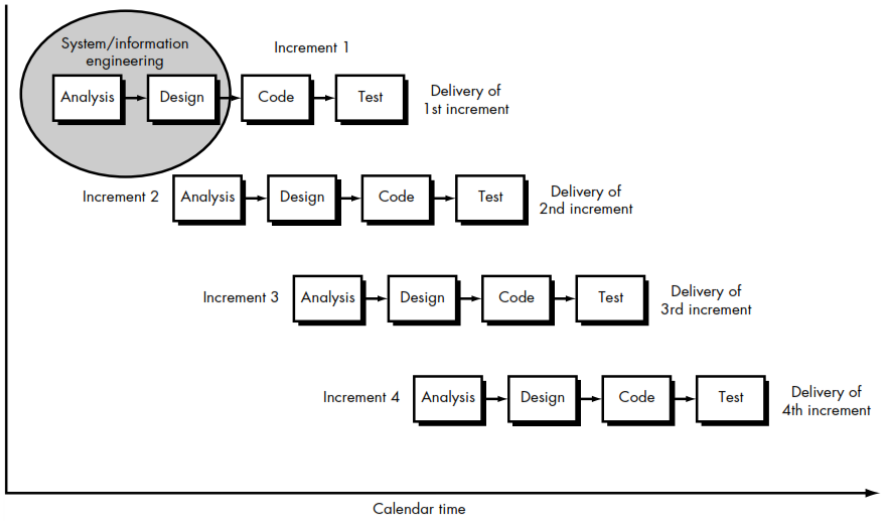
Sebagaimana model pemrosesan yang lainnya, pendekatan *RAD* memiliki kelemahan:

1. Untuk proyek besar namun dapat diskalakan, model pengembangan aplikasi cepat membutuhkan sumber daya manusia yang memadai untuk membuat jumlah tim model pengembangan aplikasi cepat yang tepat.
2. RAD membutuhkan pengembang dan pelanggan yang berkomitmen pada aktivitas cepat yang untuk menyelesaikan sistem dalam kerangka waktu yang sangat singkat.
3. Tidak semua jenis aplikasi sesuai untuk RAD. Jika sistem tidak dapat dimodulasi dengan benar, membangun komponen yang diperlukan untuk RAD akan bermasalah. Jika kinerja tinggi menjadi masalah dan kinerja ingin dicapai melalui penyetelan antarmuka ke komponen sistem, pendekatan RAD mungkin tidak berfungsi.

4. RAD tidak sesuai jika risiko teknis tinggi. Ini terjadi ketika aplikasi baru menggunakan banyak teknologi baru atau ketika perangkat lunak baru membutuhkan interoperabilitas tingkat tinggi dengan program komputer yang ada.

B.4 *The Incremental Model* (Model Inkremental)

Incremental model menggabungkan elemen *waterfall model* yang diterapkan berulang kali. Setiap persamaan linier menghasilkan peningkatan dalam hal metode atau teknik untuk diterapkan sebagai penyempurnaan perangkat lunak yang dibangun. Sebagai contoh, perangkat lunak pengolah kata yang dikembangkan menggunakan paradigma inkremental dapat memberikan fungsi dasar manajemen file, pengeditan, dan produksi dokumen dalam peningkatan pertama; kemampuan pengeditan dan produksi dokumen yang lebih canggih dalam kelipatan kedua; pemeriksaan ejaan dan tata bahasa dalam kelipatan ketiga; dan kemampuan tata letak halaman lanjutan dalam kelipatan keempat. Perlu dicatat bahwa aliran proses untuk setiap kenaikan dapat menggabungkan paradigma pembuatan prototipe.



Sumber : Pressman (2001 : 35)

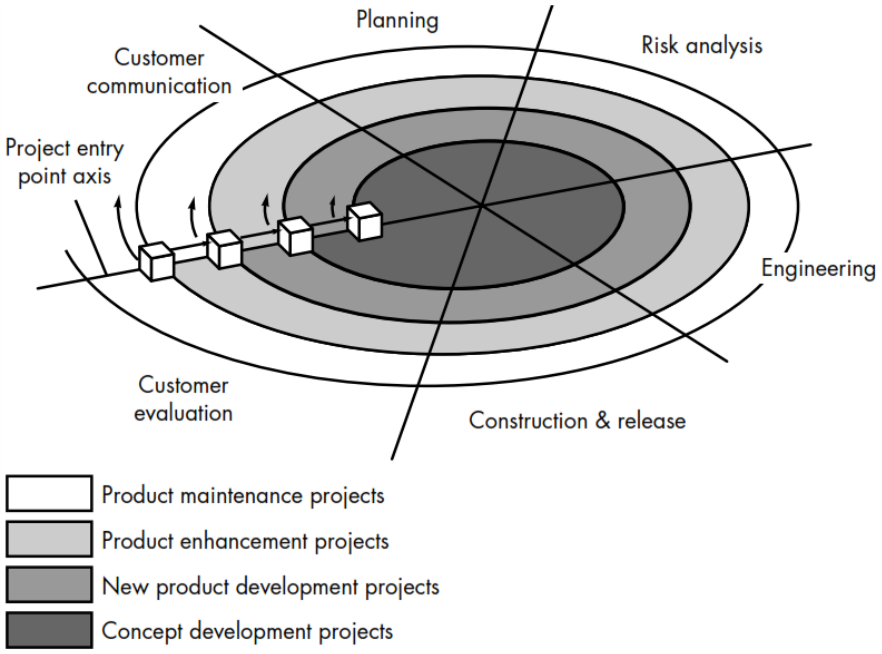
Gambar 5. *The Incremental Model* (Model Inkremental)

Ketika *incremental model* digunakan, kenaikan pertama seringkali merupakan produk inti. Artinya, persyaratan dasar ditangani, tetapi banyak fitur tambahan (beberapa diketahui, lainnya tidak diketahui) tetap tidak terkirim. Produk inti digunakan oleh pelanggan (atau menjalani tinjauan mendetail). Sebagai hasil dari penggunaan dan / atau evaluasi, rencana dikembangkan untuk kenaikan berikutnya. Rencana tersebut membahas modifikasi produk

inti untuk lebih memenuhi kebutuhan pelanggan dan penyampaian fitur dan fungsi tambahan. Proses ini diulangi setelah pengiriman setiap kenaikan, hingga produk lengkap diproduksi.

B.5 *The Spiral Model* (Model Spiral)

Spiral model awalnya diusulkan oleh Boehm (1988 : 61-67), adalah model proses perangkat lunak evolusioner yang memasangkan sifat iteratif dari prototipe dengan aspek terkontrol dan sistematis dari *waterfall model*. Ini memberikan potensi untuk pengembangan cepat versi tambahan perangkat lunak. Menggunakan *spiral model*, perangkat lunak dikembangkan dalam serangkaian rilis tambahan. Selama iterasi awal, rilis inkremental mungkin berupa model kertas atau prototipe. Selama iterasi selanjutnya, versi yang semakin lengkap dari sistem rekayasa diproduksi.



Sumber : Pressman (2001 : 37)

Gambar 6. Spiral Model

Spiral model dibagi menjadi beberapa aktivitas kerangka kerja, juga disebut wilayah tugas. Biasanya, ada antara tiga dan enam wilayah tugas. Gambar 2.6 menggambarkan *spiral model* yang berisi enam wilayah tugas.

1. *Customer communication* (komunikasi pelanggan), tugas yang diperlukan untuk membangun komunikasi yang efektif antara pengembang dan pelanggan.
2. *Planning* (perencanaan), tugas yang diperlukan untuk menentukan sumber daya, garis waktu, dan informasi terkait proyek lainnya.

3. *Risk analysis* (analisis risiko), tugas-tugas yang diperlukan untuk menilai risiko teknis dan manajemen.
4. *Engineering* (rekayasa), tugas yang diperlukan untuk membuat satu atau lebih representasi aplikasi.
5. *Construction and release* (konstruksi dan rilis), tugas yang diperlukan untuk membuat, menguji, menginstal, dan memberikan dukungan pengguna (misalnya, dokumentasi dan pelatihan).
6. *Customer evaluation* (evaluasi pelanggan), tugas yang diperlukan untuk mendapatkan umpan balik pelanggan berdasarkan evaluasi representasi perangkat lunak yang dibuat selama tahap rekayasa dan diterapkan selama tahap instalasi.

Masing-masing wilayah dihuni oleh seperangkat tugas kerja, yang disebut sekumpulan tugas, yang disesuaikan dengan karakteristik proyek yang akan dikerjakan. Untuk proyek kecil, jumlah tugas kerja dan formalitasnya rendah. Untuk proyek yang lebih besar dan lebih penting, setiap wilayah tugas berisi lebih banyak tugas kerja yang ditentukan untuk mencapai tingkat formalitas yang lebih tinggi. Dalam semua kasus, aktivitas yang disepakati, misalnya manajemen konfigurasi perangkat lunak dan jaminan kualitas perangkat lunak dapat diterapkan.

Saat proses evolusi ini dimulai, tim rekayasa perangkat lunak bergerak mengelilingi spiral searah jarum jam, dimulai dari tengah. Sirkuit pertama di sekitar spiral dapat menghasilkan pengembangan spesifikasi produk; lintasan berikutnya di sekitar spiral dapat digunakan untuk mengembangkan prototipe dan kemudian versi perangkat lunak yang semakin canggih. Setiap melewati daerah perencanaan menghasilkan penyesuaian dengan rencana proyek. Biaya dan jadwal disesuaikan berdasarkan umpan balik yang diperoleh dari evaluasi pelanggan. Selain itu, manajer proyek menyesuaikan jumlah iterasi yang direncanakan yang diperlukan untuk menyelesaikan perangkat lunak.

Tidak seperti model proses klasik yang berakhir saat perangkat lunak dikirimkan, *spiral model* dapat diadaptasi untuk diterapkan disepanjang masa pakai perangkat lunak komputer. Pandangan alternatif dari *spiral model* dapat dipertimbangkan dengan memeriksa sumbu titik masuk proyek, juga ditunjukkan pada Gambar 2.5. Setiap kubus yang ditempatkan di sepanjang sumbu dapat digunakan untuk mewakili titik awal untuk berbagai jenis proyek. Sebuah proyek pengembangan konsep dimulai pada inti spiral dan akan berlanjut (beberapa iterasi terjadi di sepanjang jalur spiral yang membatasi wilayah yang diarsir mulai dari pusat)

sampai pengembangan konsep selesai. Jika konsep akan dikembangkan menjadi produk yang sebenarnya, proses dilanjutkan melalui kubus berikutnya (titik masuk proyek pengembangan produk baru) dan proyek pengembangan baru dimulai. Produk baru ini akan berkembang melalui sejumlah iterasi di sekitar spiral, mengikuti jalur yang membatasi wilayah yang memiliki bayangan lebih terang daripada intinya. Intinya, spiral, bila dikarakterisasi dengan cara ini, tetap beroperasi sampai perangkat lunak dihentikan. Ada kalanya proses tidak aktif, tetapi setiap kali perubahan dimulai, proses dimulai pada titik masuk yang sesuai (misalnya, peningkatan produk).

Spiral model adalah pendekatan realistis untuk pengembangan sistem dan perangkat lunak skala besar. Karena perangkat lunak berkembang seiring dengan kemajuan proses, pengembang dan pelanggan lebih memahami dan bereaksi terhadap risiko di setiap tingkat evolusi. *Spiral model* menggunakan prototipe sebagai mekanisme pengurangan risiko, tetapi yang lebih penting, memungkinkan pengembang untuk menerapkan pendekatan prototipe pada setiap tahap dalam evolusi produk. Ia mempertahankan pendekatan bertahap sistematis yang disarankan oleh siklus hidup klasik tetapi menggabungkannya ke dalam kerangka kerja berulang yang lebih realistis mencerminkan dunia nyata. *Spiral model* menuntut pertimbangan langsung dari risiko teknis di semua tahapan proyek dan jika diterapkan dengan benar akan mengurangi risiko sebelum menjadi bermasalah.

Namun seperti paradigma lainnya, *spiral model* bukanlah obat mujarab. Mungkin sulit untuk meyakinkan pelanggan (terutama dalam situasi kontrak) bahwa pendekatan evolusioner dapat dikendalikan. Ini menuntut keahlian penilaian risiko yang cukup dan mengandalkan keahlian ini untuk sukses. Jika risiko besar tidak ditemukan dan dikelola, niscaya masalah akan muncul. Akhirnya, model tersebut belum digunakan tidak seperti halnya pada paradigma *waterfall model* atau *mock-up model*. Dalam *spiral model* diperlukan beberapa tahun sebelum efektivitas paradigma penting ini dapat ditentukan dengan kepastian mutlak.

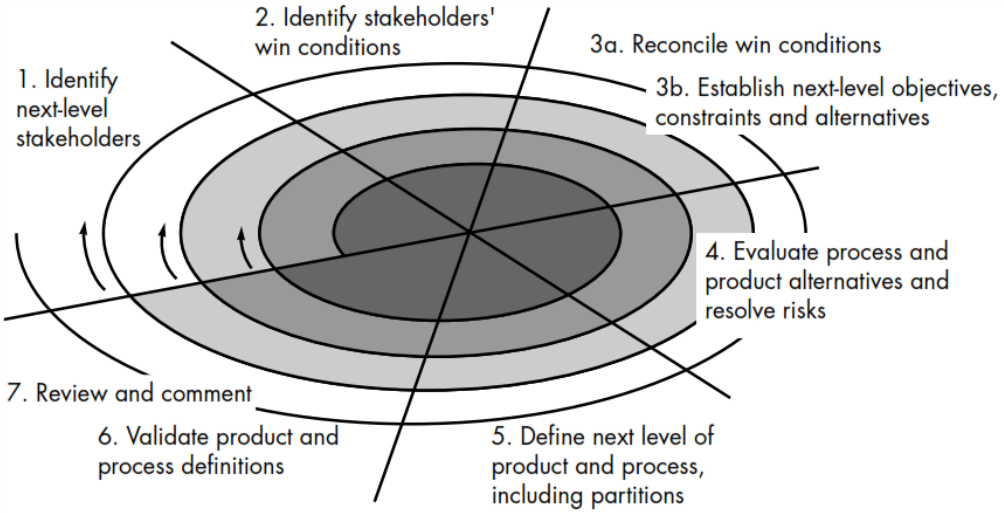
B.6 *The WINWIN Spiral Model* (Model Spiral WINWIN)

Spiral Model yang dibahas dalam Bagian 2.1.2.5 menyarankan aktivitas kerangka kerja yang membahas komunikasi pelanggan. Tujuan dari aktivitas ini adalah untuk mendapatkan persyaratan proyek dari pelanggan. Dalam konteks yang ideal, pengembang hanya menanyakan pelanggan apa yang dibutuhkan dan pelanggan memberikan detail yang cukup untuk melanjutkan. Sayangnya, hal ini jarang terjadi. Pada kenyataannya, pelanggan

dan pengembang masuk ke dalam proses negosiasi, di mana pelanggan mungkin diminta untuk menyeimbangkan fungsionalitas, kinerja, dan karakteristik produk atau sistem lainnya terhadap biaya dan waktu ke pasar. Negosiasi terbaik berjuang untuk hasil yang *WINWIN* "sama-sama menguntungkan". Artinya, pelanggan menang dengan mendapatkan sistem atau produk yang memenuhi sebagian besar kebutuhannya dan pengembang menang dengan bekerja sesuai anggaran dan tenggat waktu yang realistis yang dapat dicapai.

WINWIN spiral model diartikan oleh Boehm (1998:22-44) sebagai serangkaian aktivitas negosiasi di awal setiap lintasan di sekitar spiral, pengembang dapat melakukan :

1. *Identification* (identifikasi) identifikasi kebutuhan pelanggan
2. *Determination* (penentuan) kondisi dan permasalahan pelanggan
3. *Negotiation* (negosiasi) pembicaraan dengan mengenai kebutuan, kondisi, dan pemasalahan menjadi kondisi yang 'WIN-WIN' antara pelanggan dengan pengembang



Sumber : Pressman (2001 : 39)

Gambar 7. Model Spiral WINWIN

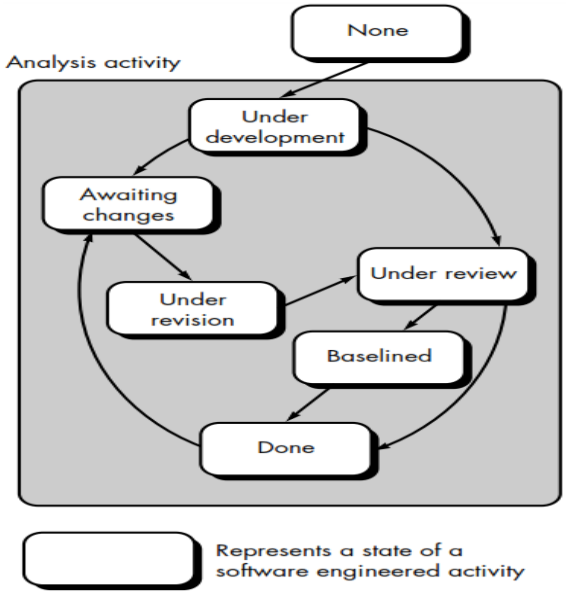
Berhasil menyelesaikan langkah-langkah awal ini mencapai hasil yang sama-sama menguntungkan, yang menjadi kriteria utama untuk melanjutkan ke definisi perangkat lunak dan sistem. Model spiral WINWIN diilustrasikan pada Gambar 2.7. Selain penekanan pada negosiasi awal, model spiral WINWIN, Boehm (1996:73-82) memperkenalkan tiga tonggak proses, yang disebut titik jangkar, yang membantu menetapkan penyelesaian satu siklus di sekitar spiral dan memberikan tonggak keputusan sebelum proyek perangkat lunak dilanjutkan.

Intinya, titik jangkar mewakili tiga pandangan berbeda tentang kemajuan saat proyek melintasi spiral. Titik jangkar pertama, *life cycle objectives (LCO)* atau tujuan siklus hidup, mendefinisikan serangkaian tujuan untuk setiap aktivitas rekayasa perangkat lunak utama. Misalnya, sebagai bagian dari LCO, serangkaian tujuan menetapkan definisi persyaratan sistem / produk tingkat atas. Titik jangkar kedua, *life cycle architecture (LCA) atau* arsitektur siklus hidup, menetapkan tujuan

tives yang harus dipenuhi ketika arsitektur sistem dan perangkat lunak didefinisikan. Misalnya, sebagai bagian dari LCA, tim proyek perangkat lunak harus menunjukkan bahwa mereka telah mengevaluasi penerapan komponen perangkat lunak siap pakai dan dapat digunakan kembali serta mempertimbangkan dampaknya terhadap keputusan arsitektur. *Initial operational capability (IOC)* atau kemampuan operasional awal adalah titik jangkar ketiga dan mewakili sekumpulan tujuan yang terkait dengan persiapan software untuk instalasi/pendistribusian, persiapan lokasi sebelum instalasi, dan bantuan yang dibutuhkan oleh semua pihak yang akan menggunakan atau mendukung software tersebut.

B.7 *The Concurrent Development – CD Model* (Model Pengembangan Konkuren)

CD Model (Model pengembangan konkuren) disebut juga rekayasa bersamaan, dapat direpresentasikan secara skematis sebagai rangkaian aktivitas teknis utama, tugas, dan status terkaitnya.



Sumber : Pressman (2001 : 41)

Gambar 8. *The Concurrent Development – CD Model*

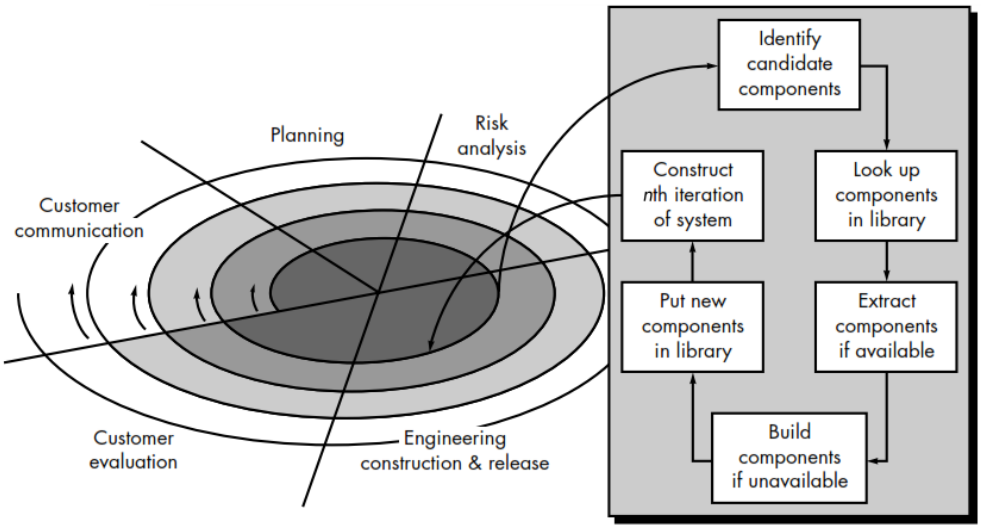
Davis dan Sitaram (1994:38-51) memberikan paradigma seperti pada Gambar 2.7, di sini penulis memaparkan lebih sederhana sebagai berikut :

1. Pada kondisi *None* atau pengembangan tidak menemukan permasalahan, dapat langsung melakukan transisi ke dalam kondisi pengembangan. Namun, jika pelanggan menunjukkan bahwa perubahan dalam persyaratan harus dilakukan artinya pengembang menemukan permasalahan, aktivitas analisis berpindah dari *under development* (status dalam pengembangan) ke status *a waiting changes* (menunggu perubahan) untuk selanjutnya masuk ke *under revision* (status perubahan pengembangan) .
2. Apabila pengembang tidak menemukan permasalahan dalam pengembangan, maka akan masuk pada satus *under review* (evaluasi dengan pelanggan) untuk mencapai kesepakatan dari *baselined* (kebutuhan yang diinginkan pelanggan).
3. Pekerjaan pengembangan dianggap *done* (selesai) apabila *baselined* yang ditetapkan sudah tercapai.
4. Dalam perjalanan pengembangan telah dianggap selesai, terkadang muncul ide dari pelanggan (bisa terjadi setelah emplementasi) untuk menambahkan bagian yang tidak terpikirkan sebelumnya, dalam hal ini, pengembang akan kembali ke tahap *a waiting changes* (menunggu perubahan).

Dari paradigma tersebut, dapat disimpulkan bahwa *CD model* lebih cocoh diterapkan pada pelanggan yang homogen (institusi pendidikan, institusi pemerintahan, perpustakaan, atau perusahaan-perusahaan yang sejenis), dimana pengembangan dapat mengembangkan atau sebelumnya terlah memiliki aplikasi untuk itu.

B.8 *The Component-Based Development Model*

(Model Pembangunan Berbasis Komponen)



Sumber : Pressman (2001 : 42)

Gambar 9. The Component-Based Development Model

Model pembangunan berbasis komponen berorientasi pada objek yang menekankan pembuatan kelas untuk merangkum data dan algoritma yang digunakan dalam memanipulasi data. Jika dirancang dan diterapkan dengan benar, kelas berorientasi objek dapat digunakan kembali di berbagai aplikasi dan sistem berbasis komputer. *CBD model* sebagai mana terlihat pada Gambar 2.8 menggabungkan banyak karakteristik *spiral model*. Menurut Niestrasz (1992: 160-165), ini bersifat evolusioner, menuntut pendekatan berulang untuk pembuatan perangkat lunak. Namun, model pengembangan berbasis komponen menyusun aplikasi dari komponen perangkat lunak yang telah dikemas (disebut kelas). Kegiatan teknik dimulai dengan identifikasi kelas kandidat. Ini dilakukan dengan memeriksa data yang akan dimanipulasi oleh aplikasi dan algoritma yang akan diterapkan untuk menyelesaikan manipulasi. Data dan algoritme yang sesuai dikemas ke dalam kelas.

Setelah kelas kandidat diidentifikasi, perpustakaan kelas dicari untuk menentukan apakah kelas-kelas ini sudah ada. Jika ya, mereka akan diekstrak dari perpustakaan dan digunakan kembali. Jika kelas kandidat tidak berada di perpustakaan, itu direkayasa menggunakan metode berorientasi objek.

Proses pengembangan perangkat lunak terpadu [JAC99] merupakan perwakilan dari sejumlah *CBD model* yang telah diusulkan dibanyak industri. Dengan menggunakan *Unified Modeling Language* (UML), proses terpadu tersebut mendefinisikan komponen yang akan digunakan untuk membangun sistem dan antarmuka yang akan menghubungkan komponen tersebut. Menggunakan kombinasi pengembangan iteratif dan inkremental, proses terpadu mendefinisikan fungsi sistem dengan menerapkan pendekatan berbasis skenario (dari sudut pandang pengguna). Ini kemudian memasangkan fungsi dengan kerangka arsitektur yang mengidentifikasi bentuk perangkat lunak yang akan diambil.

B.9 *The Formal Methods - FM Model* (Model Metode Formal)

FM model mencakup serangkaian aktivitas yang mengarah pada spesifikasi matematis formal perangkat lunak komputer. *FM model* memungkinkan seorang pengembang perangkat lunak untuk menentukan, mengembangkan, dan memverifikasi sistem berbasis komputer dengan menerapkan notasi matematika yang ketat. Menurut Mills, Dyer, dan Linger (1987 : 19-25), variasi dari pendekatan ini, disebut rekayasa perangkat lunak ruang bersih, saat ini diterapkan oleh beberapa organisasi pengembangan perangkat lunak. Ketika *FM Model* digunakan selama pengembangan, metode tersebut menyediakan mekanisme untuk

menghilangkan banyak masalah yang sulit diatasi dengan menggunakan paradigma rekayasa perangkat lunak lainnya. Ketidakjelasan, ketidaklengkapan, dan ketidakkonsistenan dapat ditemukan dan diperbaiki dengan lebih mudah, tidak melalui tinjauan adhoc tetapi melalui penerapan analisis matematis. Ketika *FM Model* digunakan selama desain, mereka berfungsi sebagai dasar untuk verifikasi program dan oleh karena itu memungkinkan pengembang perangkat lunak untuk menemukan dan memperbaiki kesalahan yang mungkin tidak terdeteksi. Pressman (2001 : 44), menuturkan kekhawatiran tentang penerapann *FM Model* dalam lingkungan bisnis bahwa :

1. Pengembangan *FM Model* saat ini cukup memakan waktu dan mahal.
2. Karena hanya sedikit pengembang perangkat lunak yang memiliki latar belakang yang diperlukan untuk menerapkan *FM Model*, diperlukan pelatihan ekstensif.
3. Sulit untuk menggunakan *FM Model* sebagai mekanisme komunikasi untuk pelanggan yang secara teknis tidak canggih.

Meskipun ada kekhawatiran ini, ada kemungkinan bahwa pendekatan *FM Model* akan mendapatkan pengikut di antara pengembang perangkat lunak yang harus membangun perangkat lunak yang kritis terhadap keselamatan (misalnya, pengembang avionik pesawat dan perangkat medis) dan di antara pengembang yang akan mengalami kesulitan ekonomi yang parah jika kesalahan perangkat lunak terjadi.

B.10 *The fourth generation techniques model* (model teknik generasi keempat)

Istilah teknik generasi keempat (4GT) mencakup beragam alat perangkat lunak yang memiliki satu kesamaan: masing-masing memungkinkan pengembang perangkat lunak untuk menentukan beberapa karakteristik perangkat lunak pada tingkat tinggi. Alat tersebut kemudian secara otomatis menghasilkan kode sumber berdasarkan spesifikasi pengembang. Ada sedikit perdebatan bahwa semakin tinggi level di mana perangkat lunak dapat dispesifikasikan ke sebuah mesin, semakin cepat sebuah program dapat dibangun. Paradigma 4GT untuk rekayasa perangkat lunak berfokus pada kemampuan untuk menentukan perangkat lunak menggunakan bentuk bahasa khusus atau notasi grafik yang menggambarkan masalah yang harus diselesaikan dalam istilah yang dapat dipahami pelanggan.

Saat ini, lingkungan pengembangan perangkat lunak yang mendukung paradigma 4GT mencakup beberapa atau semua alat berikut: bahasa nonprocedural untuk kueri basis data, pembuatan laporan, manipulasi data, interaksi layar dan definisi, pembuatan kode;

kemampuan grafis tingkat tinggi; kemampuan spreadsheet, dan pembuatan otomatis HTML dan bahasa serupa yang digunakan untuk pembuatan situs Web menggunakan alat perangkat lunak canggih. Awalnya, banyak alat yang disebutkan sebelumnya hanya tersedia untuk domain aplikasi yang sangat spesifik, tetapi saat ini lingkungan 4GT telah diperluas untuk menangani sebagian besar kategori aplikasi perangkat lunak.

Seperti paradigma lain, 4GT dimulai dengan langkah pengumpulan persyaratan. Idealnya, pelanggan akan menjelaskan persyaratan dan ini akan langsung diterjemahkan ke dalam prototipe operasional. Tapi ini tidak bisa diterapkan. Pelanggan mungkin tidak yakin tentang apa yang diperlukan, mungkin ambigu dalam menentukan fakta yang diketahui, dan mungkin tidak dapat atau tidak mau untuk menentukan informasi dengan cara yang dapat digunakan oleh alat 4GT.

Untuk alasan ini, dialog pelanggan dengan pengembang untuk mendapatkan model proses tetap menjadi bagian penting dari pendekatan 4GT.

Untuk aplikasi kecil, dimungkinkan untuk beralih langsung dari langkah pengumpulan persyaratan ke implementasi menggunakan bahasa generasi keempat nonprocedural (4GL) atau model yang terdiri dari jaringan ikon grafis. Namun, untuk upaya yang lebih besar, perlu mengembangkan strategi desain untuk sistem, bahkan jika 4GL akan digunakan. Penggunaan 4GT tanpa desain (untuk proyek besar) akan menyebabkan kesulitan dalam pemeliharaan, hal yang sama juga telah ditemui ketika mengembangkan perangkat lunak menggunakan pendekatan konvensional.

Implementasi menggunakan 4GL memungkinkan pengembang perangkat lunak untuk mewakili hasil yang diinginkan dengan cara yang mengarah ke pembuatan kode otomatis untuk membuat hasil tersebut. Tentunya, struktur data dengan informasi yang relevan harus ada dan mudah diakses oleh 4GL. Untuk mengubah implementasi 4GT menjadi produk, pengembang harus melakukan pengujian menyeluruh, mengembangkan dokumentasi yang berarti, dan melakukan semua solusi lainnya.

Seperti semua paradigma rekayasa perangkat lunak, model 4GT memiliki kelebihan dan kekurangan. Para pendukung mengklaim pengurangan dramatis dalam waktu pengembangan perangkat lunak dan produktivitas sangat meningkat bagi orang-orang yang membuat perangkat lunak. Para penentang mengklaim bahwa alat 4GT saat ini tidak jauh lebih mudah digunakan daripada bahasa pemrograman, bahwa kode sumber yang dihasilkan oleh

alat tersebut tidak efisien, dan bahwa pemeliharaan sistem perangkat lunak besar yang dikembangkan menggunakan 4GT masih dipertanyakan.

Pressman (2001 : 45), menuturkan ada beberapa manfaat dalam klaim kedua belah pihak dan mungkin untuk meringkas keadaan saat ini dari pendekatan 4GT :

1. Penggunaan 4GT adalah pendekatan yang layak untuk banyak area aplikasi yang berbeda. Ditambah dengan alat rekayasa perangkat lunak berbantuan komputer dan generator kode, 4GT menawarkan solusi yang kredibel untuk banyak masalah perangkat lunak.
2. Data yang dikumpulkan dari perusahaan yang menggunakan 4GT menunjukkan bahwa waktu yang dibutuhkan untuk memproduksi perangkat lunak sangat berkurang untuk aplikasi kecil dan menengah dan bahwa jumlah desain dan analisis untuk aplikasi kecil juga berkurang.
3. Namun, penggunaan 4GT untuk upaya pengembangan perangkat lunak yang besar menuntut analisis, desain, dan pengujian (kegiatan rekayasa perangkat lunak) sebanyak atau lebih untuk mencapai penghematan waktu yang besar sebagai hasil dari penghapusan pengkodean.

C. Pengembangan Sistem Informasi

Pengembangan sistem informasi yang direalisasikan dengan bantuan komputer (*Computerized Information System*) melalui suatu tahapan tertentu. Di muka telah dipaparkan, mengenai model-model pengembangan sistem informasi. Dalam tulisan ini, akan mengkaji lebih mendalam mengenai *waterfall model* yang dipandang sebagai model yang banyak digunakan oleh para pengembang. Dari uraian sebelumnya diketahui bahwa aktivitas yang biasa dilakukan oleh pengembang dalam pembuatan perangkat lunak adalah (1) investigasi Sistem dan Analisis Sistem, (2) Studi Kelayakan, (3) Desain Sistem, (4) Coding, (5) testing, dan (6) emplementasi.

C.1 Investigasi Sistem dan Analisis Sistem

Bagian ini merupakan tahap penyelidikan awal, dimana pengembang menyelami keadaan institusi pelanggan. Hal yang perlu diperhatikan menurut Kristanto (2018 : 41) adalah:

1. Memahami dan memperjelas apa yang diharapkan oleh pelanggan (sistem informasi bagaimana yang mereka perlukan)
2. Menentukan ruang lingkup dari studi sistem informasi
3. Menentukan kelayakan dari masing-masing alternatif dengan memikirkan keuntungan/kerugian yang mungkin timbul.

Apabila satu alternatif sudah terpilih untuk dilaksanakan melalui suatu sistem, maka selanjutnya mengadakan studi kelayakan.

C.2 Studi Kelayakan

Studi kelayakan merupakan tahap yang paling penting, karena di dalamnya menyangkut aspek sistem yang akan dibuat atau diusulkan. Laporan studi kelayakan yang dibuat oleh pengembang, selanjutnya dibaca oleh pelanggan, dan pada tahap ini pelanggan akan memberikan masukan-masukan yang lebih mendalam sesuai dengan harapannya. Kemudian pengembang melakukan perubahan laporan studi kelayakan tersebut untuk disepakati bersama.

Laporan studi kelayakan menyangkut tentang latar belakang, masalah dan hambatan yang dihadapi pelanggan; tujuan sistem yang akan dibuat; pernyataan kelayakan ekonomi, teknis, dan operasional; kebutuhan fungsional dan rencana penyusunan sistem yang akan dibuat, meliputi : batasan sistem, tugas-tugas yang akan dilakukan, jadual untuk melakukan tugas tersebut; tim penyusunan sistem; rekomendasi dan persetujuan sistem yang akan dibuat.

C.3 Desain Sistem

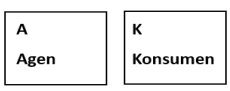
Berpedoman pada laporan studi kelayakan yang sudah disetujui, pengembang selanjutnya membuat desain sistem yang akan dibuat meliputi :

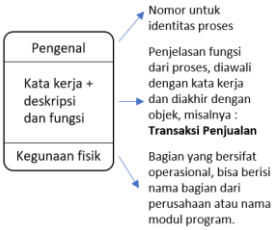
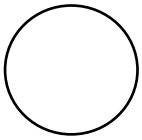
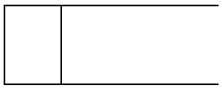
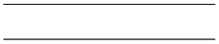
1. **DFD (Data Flow Diagram)**

   DFD adalah suatu model logika data atau proses yang dibuat untuk menggambarkan darimana asal data dan kemana tujuan data dikeluarkan dari sistem, dimana disimpan, proses apa yang menghasilkan data tersebut, dan interkasi antara data yang tersimpan dan proses yang dikenakan pada data tersebut…Ada dua teknik dasar DFD yang umum yaitu Gane and Sarson dan Yourdon and De Marco (Kristanto, 2018 : 61).

Berikut ini perbedaan lambang yang digunakan oleh kedua teknik tersebut dapat dicontohkan sebagaimana terlihat pada tabel beriktu ini.

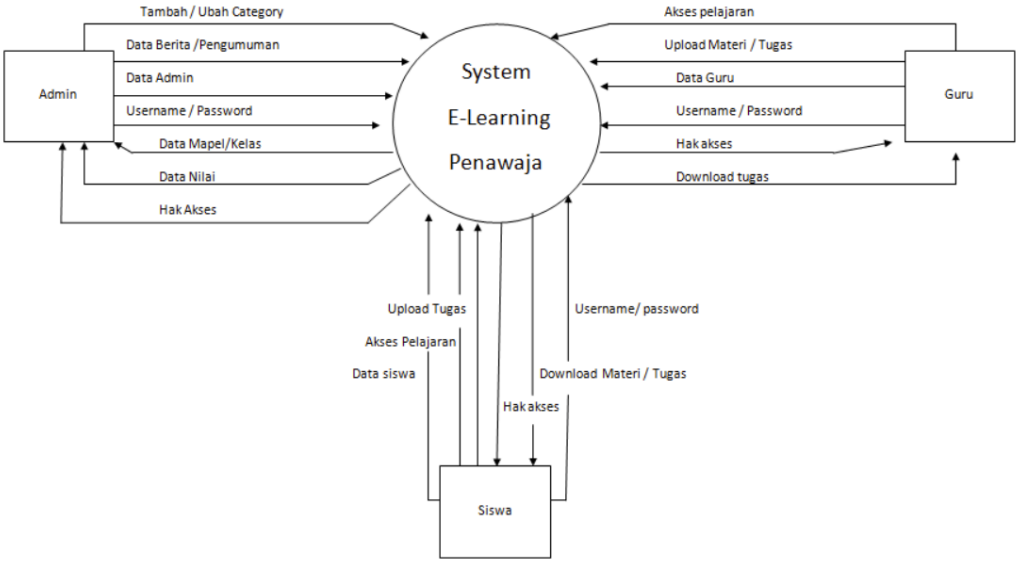
Tabel 1. Simbol Teknik Gane and Sarsor dan Yourdaon and Marco

Untuk lambang	Gane and Sarson	Yourdon and De Marco
Entiti luar Merupakan lingkungan luar sistem yang menjadi sumber atau tujuan dari aliran data.	 Digambarkan dengan simbul bujur sangkar. Apabila entiti lebih dari satu, dapat dilakukan dengan memberikan garis diagonal disebelah kanan bawan dengan menyertakan jumlah entitinya.	 Digambarkan dengan simbul persegi biasa atau persegi. Aturannya sama dengan teknik Gane and Sarson

Untuk lambang	Gane and Sarson	Yourdon and De Marco
Aliran Data Menggambarkan aliran/arus data dari satu bagian ke bagian lainnya.		
Proses Disebut juga fungsi yang mentransformasikan data secara umum.		
Berkas atau Tempat Penyimpanan Merupakan komponen yang berfungsi untuk menyimpan data atau file.		 Digambarkan dengan garis paralel

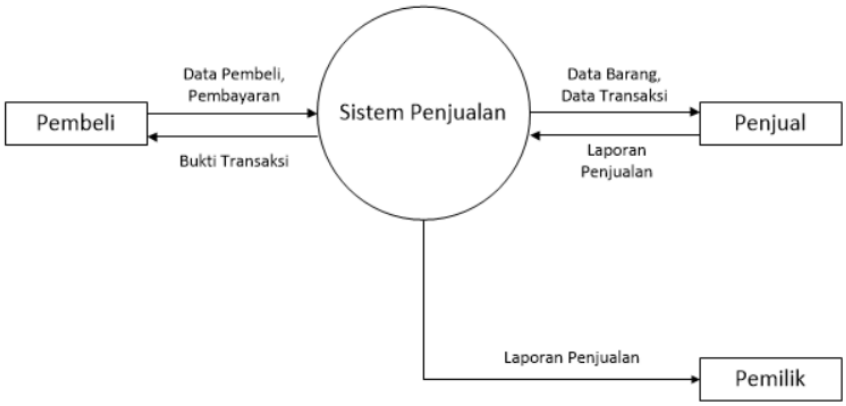
2. Context Diagram

Diagram konteks adalah sebuah diagram sederhana yang menggambarkan hubungan antara entiti luar dengan masukan dan keluaran dari sistem. Diagram konteks direpresentasikan dengan lingkaran tunggal yang mewakili keseluruhan dari sistem. Berikut ini contoh diagram konteks :



Sumber : http://www.waskhas.com

Gambar 10. Diagram Konteks sistem E-Learning



Sumber : https://astizardiaz.wordpress.com

Gambar 11. Diagram Konteks Sistem Penjualan

3. Algorithma kejadian

Daftar kejadian digambarkan dalam kalimat sederhana dan berfungsi untuk memodelkan kejadian dalam lingkungan sehari-hari dan membutuhkan tanggapan atau respon dari sistem. Misalnya, konsumen memesan barang (ke sistem), konsumen membatalkan pesanan barang (ke sistem), dan manajemen meminta laporan penjualan barang (dari sistem).

Kristanto (2018 : 71), mengingatkan bahwa dalam mendeskripsikan daftar kejadian, pelaku adalah entiti luar, jadi bukan dari sistem. Contohnya adalah sebagai berikut :

Pernyataan Keliru	Pernyataan Benar
PEMESANAN KONSUMEN DITERIMA OLEH SISTEM	KONSUMEN MEMESAN
SISTEM MENGELUARKAN LAPORAN UNTUK MANAJER	MANAJER MENERIMA LAPORAN

4. *Data dictionary* (kamus data)

Dikutip dari https://eskagrey.wordpress.com, Kamus data adalah suatu daftar data elemen yang terorganisir dengan definisi yang tetap dan sesuai dengan sistem, sehingga pengguna dan analis sistem mempunyau pengertian yang sama tentang input, output dan komponen data store.

Kamus data adalah referensi terkait dengan data (metadata). Salah satu tujuan penting dari kamus data adalah untuk menjaga konsistensi data, misalnya untuk jenis kelamin

L untuk pria dan W untuk wanita, sehingga aturan ini berlaku untuk semua aplikasi yang berinteraksi dengan sistem.

Kamus data juga dapat berfungsi sebagai berikut:

a. Menjelaskan arti aliran data dan penyimpanan data dalam DFD.
b. Mendeskripsikan komposisi paket data yang bergerak melalui aliran. (misal: nama dapat diuraikan menjadi nama depan dan nama belakang).
c. Mendeskripsikan komposisi penyimpanan data.
d. Mendeskripsikan hubungan detail antar media penyimpanan.
e. Menyajikan spesifikasi nilai dan satuan yang relevan bagi media penyimpanan dan aliran.

Elemen data dapat didefinisikan sebagai berikut ini:

a. Menguraikan arti dari alur data dan data store dalam DFD.
b. Menguraikan komposisi paket data pada alur data ke dalam alur yang lebih kecil. (contoh: alamat yang terdiri dari nama jalan, kota dan kode pos).
c. Menguraikan komposisi paket data dalam data store.
d. Menspesifikasikan nilai dan unit informasi dalam alur data dan data store.
e. Menguraikan hubungan yang terinci antara data store dalam suatu ERD.

Tabe 2. Notasi Kamus Data

Notas	Arti
=	Terdiri dari , terbentuk dari, sama dengan
+	Dan
( )	Opsional
{ }	Iterasi / pengulangan
[ ]	Pilih satu dari beberapa alternatif (pilihan)
**	Komentar
@	Identifier suatu data store
|	Pemisah dalam bentuk [ ]
Alias	Nama lain untuk suatu data

Tabel 3. Contoh dari Notasi Kamus Data

Notasi	Contoh
=	Nama=Nama_depan + Nama_belakang
( )	Nama_langganan = (title) + nama_depan + (nama_tengah) + nama_belakang
{ }	Pesanan = nama_pelanggan + alamat + 1 {item} 10
[ ]	Jenis kelamin = [pria][wanita]
**	Penjualan = *jumlah penjualan tiap tahun*
Alias	Client alias customer

5. Desain formulir masukan

Kristanto (2018 : 97) menjelaskan bahwa dalam merancang atau mendesain form masukan ada beberapa hal yang harus diperhatikan yaitu:

1. Sasaran
   - Form masukan yang dibuat harus mudah diisi oleh penerima atau pemakai
   - Dapat menghindari atau memperkecil kemungkinan kesalahan pengisian data
2. Langkah yang diambil
   - Panjang setiap item dalam form masukan harus jelas
   - Untuk setiap item pilihan cantumkan semua pilihan yang ada
   - Beri keterangan untuk item yang akan diisi oleh system

6. Desain formulir keluaran

Kristanto (2018 : 101) menjelaskan bahwa dalam merancang atau mendesain form keluaran ada beberapa hal yang harus diperhatikan yaitu:

1. Sasaran
   - Form keluaran yang dihasilkan harus mudah dipahami oleh penerima atau pemakai dan bersifat informatif
   - Dapat diarsip secara mudah
2. Langkah yang diambil
   - Tentukan item apa saja yang akan ditampilkan
   - Item yang bersifat kode diberikan deskripsinya
   - Berikan penomoran halaman, juga tanggal dan jam bila dirasa perlu
   - Sesuaikan dengan ukuran kertas dan jenis printer

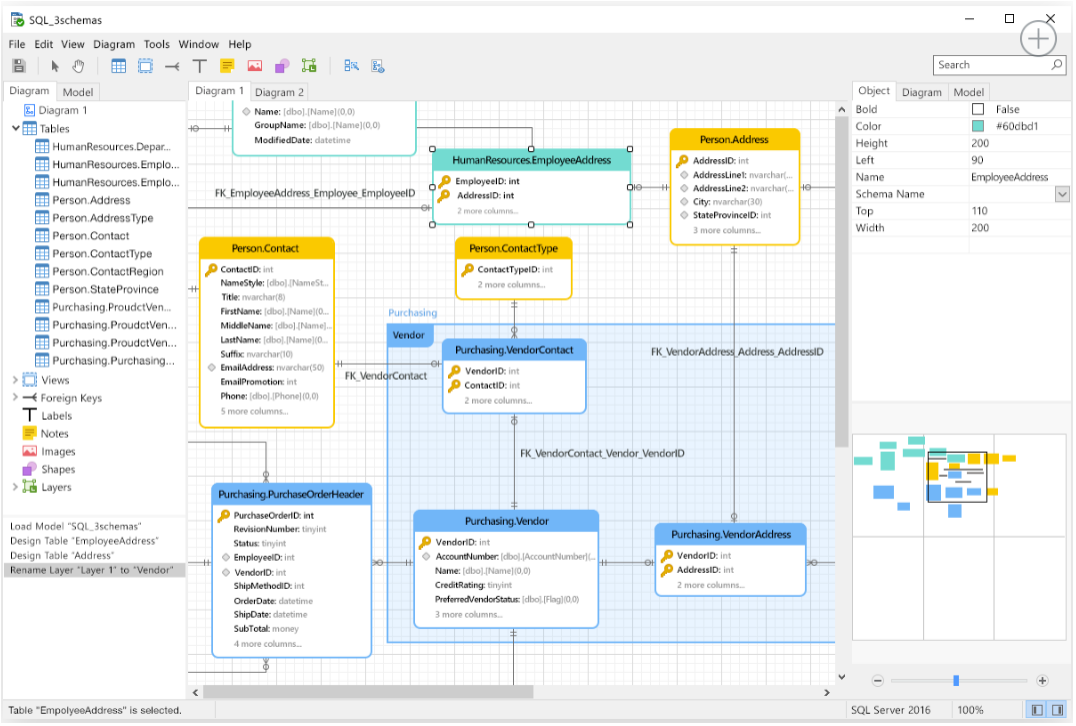
C.4 Desain database

Database, pada tingkat yang paling sederhana adalah kumpulan data terkait. Tidak harus elektronik, kartu katalog yang dulu dimiliki perpustakaan tentunya adalah database. Tentunya juga buku telepon atau alamat, itu dapat dikatakan database. Seiring dengan perkembangan teknologi, kini database sudah dalam bentuk digital atau elektronek. Kristanto (2018 : 79-81), menjelaskan :

> Basis data adalah kumpulan data, yang dapat digambarkan sebagai aktivitas dari satu atau lebih organisasi yang berelasi. Sebagai contoh, basis data universitas berisi informasi mengenai entiti mahasiswa, dosen, fakultas, mata kuliah, dan ruang kelas. Relasi diantara entitas, seperti pengambilan mata kuliah yang dilakukan oleh mahaiswa, penugasan dosen, dan penggunaan ruang kelas.
> Manajemen Sistem Basis Data (Database Management System – DBMS) adalah perangkat lunak yang didesain untuk membantu dalam hal pemeliharaan dan utilitas kumpulan data dalam jumlah besar…Penggunaan DBMS untuk mengelola data mempunyai beberapa keuntungan, yaitu :
> - Kebebasan data dan akses yang efisien
> - Mereduksi waktu pengembangan aplikasi
> - Integritas dan keamanan data
> - Administrasi keseragaman data
> - Akses bersamaan dan perbaikan dari terjadinya *crashes* (tabrakan dari proses serentak)

Salah satu perangkat lunak yang memiliki banyak fasilitas untuk perancangan basis data adalah Navicat. Dari situs https://www.navicat.com dijelaskan bahwa :

> Navicat Premium adalah alat pengembangan database yang memungkinkan Anda terhubung secara bersamaan ke database MySQL, MariaDB, MongoDB, SQL Server, Oracle, PostgreSQL, dan SQLite dari satu aplikasi. Kompatibel dengan database cloud seperti Amazon RDS, Amazon Aurora, Amazon Redshift, Microsoft Azure, Oracle Cloud, Google Cloud, dan MongoDB Atlas. Anda dapat dengan cepat dan mudah membangun, mengelola, dan memelihara database Anda…Kemudahan yang diberikan Navicat diantaranya adalah :
> - Transfer Data, Sinkronisasi Data, dan Sinkronisasi Struktur membantu memigrasi data dengan lebih mudah dan lebih cepat untuk overhead yang lebih sedikit. Memberikan panduan langkah demi langkah yang terperinci untuk mentransfer data ke berbagai DBMS.
> - Visual SQL/Query Builder membantu membuat, mengedit, dan menjalankan pernyataan/kueri SQL tanpa harus khawatir tentang sintaks dan penggunaan perintah yang tepat.
> - Membuat, modifikasi, dan mengelola semua objek database menggunakan desainer objek profesional. Merubah database menjadi representasi grafis menggunakan desain database yang canggih dan alat pemodelan sehingga dapat membuat model, membuat, dan memahami database yang kompleks dengan mudah.



Gambar 12. Model Desain Database dengan Navicat

C.5 Coding

Untuk memerintahkan komputer agar dapat bekerja sesuai dengan yang diinginkan, diperlukan bahasa pemrograman untuk berkomunikasi dengan komputer. Coding atau penulisan program yang dilakukan oleh programmer setelah menerima dokumen sistem informasi yang sudah dibuat oleh pengembang (perancang sistem), adalah cara agar dapat berkomunikasi dengan komputer. Menurut https://www.codeconquest.com,

> Ada ribuan bahasa pengkodean yang ada saat ini. Bahasa pengkodean tidak seperti bahasa sehari-hari – tidak ada kosakata atau abjad. Mereka lebih seperti kode – perintah khusus, singkatan dan cara menyusun teks. Semua perangkat lunak ditulis dalam beberapa jenis bahasa pengkodean. Dan setiap bahasa pengkodean itu unik, dirancang dengan sistem operasi tertentu, platform, gaya pengkodean, dan tujuan penggunaan.

Bahasa pemrograman yang populer digunakan diantaranya adalah Vusual Basic, Delphi, C, Java, PHP, HTML, Phython, dan ASP.

Pressman (2018 : 432) menyampaikan bahwa dalam proses coding atau penulisan program harus diperhatikan hal sebagai berikut :

1. *Modularity* (modularitas). Notasi desain harus mendukung pengembangan perangkat lunak modular dan menyediakan sarana untuk spesifikasi antarmuka.

2. *Overall simplicity* (kesederhanaan secara keseluruhan). Notasi desain harus relatif sederhana untuk dipelajari, relatif mudah digunakan, dan umumnya mudah dibaca.
3. *Ease of editing* (kemudahan pengeditan). Desain prosedural mungkin memerlukan modifikasi seiring dengan proses perangkat lunak. Kemudahan representasi desain dapat diedit dapat membantu memfasilitasi setiap tugas rekayasa perangkat lunak.
4. *Machine readability* (keterbacaan mesin). Notasi yang dapat dimasukkan langsung ke dalam sistem pengembangan berbasis komputer menawarkan keuntungan yang signifikan.
5. *Maintainability* (pemeliharaan). Pemeliharaan perangkat lunak adalah fase paling mahal dari siklus hidup perangkat lunak. Pemeliharaan konfigurasi perangkat lunak hampir selalu berarti pemeliharaan representasi desain prosedural.
6. *Structure enforcement* (terstruktur). Manfaat pendekatan desain yang menggunakan konsep pemrograman terstruktur telah didiskusikan. Notasi desain yang memaksakan penggunaan hanya pada konstruksi terstruktur mempromosikan praktik desain yang baik.
7. *Automatic processing* (pemrosesan otomatis). Desain prosedural berisi informasi yang dapat diproses untuk memberikan wawasan baru atau lebih baik kepada desainer tentang kebenaran dan kualitas desain. Wawasan tersebut dapat ditingkatkan dengan laporan yang disediakan melalui alat desain perangkat lunak.
8. Data representation (representasi data). Kemampuan untuk merepresentasikan data lokal dan global merupakan elemen penting dari desain tingkat komponen. Idealnya, notasi desain harus mewakili data tersebut secara langsung.
9. Logic verification (verifikasi logika). Verifikasi otomatis dari logika desain adalah tujuan yang terpenting selama pengujian perangkat lunak. Notasi yang meningkatkan kemampuan untuk memverifikasi logika sangat meningkatkan kecukupan pengujian.

2.6 Testing (pengujian)

Pengujian perangkat lunak adalah elemen penting dari jaminan kualitas perangkat lunak dan mewakili tinjauan akhir dari spesifikasi, desain, dan pembuatan kode. Meningkatnya visibilitas perangkat lunak sebagai elemen sistem dan biaya yang terkait dengan kegagalan perangkat lunak memotivasi kekuatan untuk pengujian menyeluruh yang

terencana dengan baik. Menurut Glen Myers dalam Pressman (2001 : 439), menyatakan sejumlah aturan yang dapat berfungsi dengan baik sebagai tujuan pengujian:

1. Pengujian adalah proses mengeksekusi program dengan maksud menemukan kesalahan.
2. Kasus uji yang baik adalah kasus yang memiliki probabilitas tinggi untuk menemukan kesalahan yang belum ditemukan.
3. Tes yang berhasil adalah tes yang mengungkap kesalahan yang belum ditemukan.

Sebelum menerapkan metode untuk merancang kasus uji yang efektif, Davis (1994) dalam Pressman (2001 : 440) menyarankan serangkaian prinsip pengujian yang telah diadaptasi untuk digunakan, yaitu :

1. Semua pengujian harus dapat dilacak ke persyaratan pelanggan. Tujuan pengujian perangkat lunak adalah untuk mengungkap kesalahan. Oleh karena itu, cacat yang paling parah (dari sudut pandang pelanggan) adalah yang menyebabkan program gagal memenuhi persyaratannya.
2. Pengujian harus direncanakan jauh sebelum pengujian dimulai. Perencanaan uji dapat dimulai segera setelah model persyaratan selesai. Definisi rinci kasus uji dapat dimulai segera setelah model desain dipantapkan. Oleh karena itu, semua pengujian dapat direncanakan dan dirancang sebelum kode apa pun dibuat.
3. Prinsip Pareto berlaku untuk pengujian perangkat lunak. Prinsip Pareto menyiratkan bahwa 80 persen dari semua kesalahan kemungkinan besar dapat dilacak hingga 20 persen dari semua komponen program.
4. Pengujian harus dimulai dalam skala kecil dan berlanjut ke pengujian dalam. Tes pertama yang direncanakan dan dilaksanakan biasanya berfokus pada individu komponen. Saat pengujian berlangsung, fokus bergeser dalam upaya untuk menemukan kesalahan dalam kelompok komponen yang terintegrasi dan akhirnya di seluruh sistem.
5. Pengujian menyeluruh tidak dimungkinkan. Jumlah permutasi jalur bahkan untuk program berukuran sedang sangatlah besar. Karena alasan ini, tidak mungkin untuk mengeksekusi setiap kombinasi jalur selama pengujian. Itu mungkin, namun logika program dan untuk memastikan bahwa semua kondisi dalam desain tingkat komponen telah dilaksanakan.

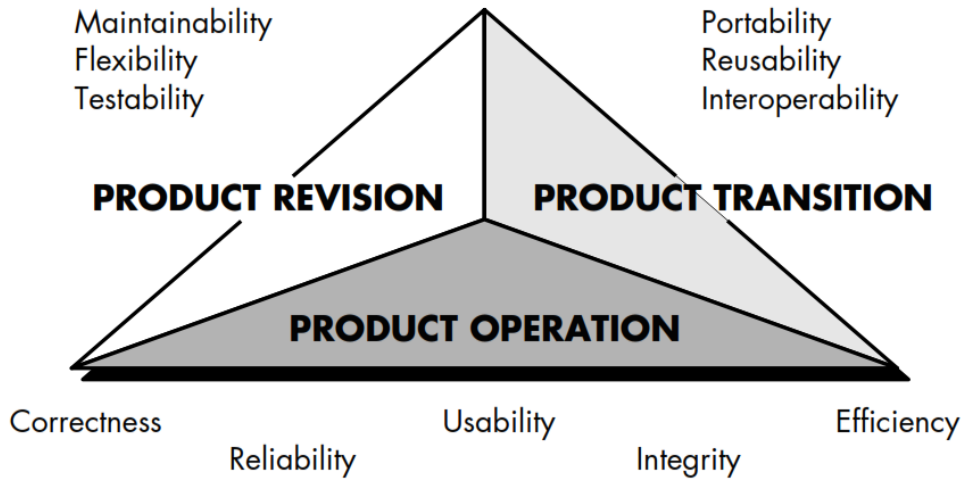
6. Agar efektif, pengujian harus dilakukan oleh seorang independen pihak ketiga. Yang paling efektif, yang kami maksud adalah pengujian yang memiliki probabilitas tertinggi untuk menemukan kesalahan (tujuan utama pengujian).

Pengujian perangkat lunak dapat dilakukan dengan berbagai cara. Salah satunya adalah dengan menggunakan metode *black-box testing* (pengujian kotak hitam). *Black-box testing*, juga disebut pengujian perilaku, berfokus pada persyaratan fungsional perangkat lunak yang memungkinkan perangkat lunak mendapatkan rangkaian kondisi input yang akan sepenuhnya menjalankan semua persyaratan fungsional untuk suatu program. Menurut Pressman (2001 : 460), penggunaan *black-box testing* adalah untuk menemukan :

(1) fungsi yang salah atau hilang, (2) kesalahan antarmuka, (3) kesalahan dalam struktur data atau akses basis data eksternal, (4) kesalahan perilaku atau kinerja, dan (5) inisialisasi dan penghentian kesalahan." Untuk itu pengujian yang dirancang untuk menjawab pertanyaan-pertanyaan berikut:

- Bagaimana validitas fungsional diuji?
- Bagaimana perilaku dan kinerja sistem diuji?
- Kelas masukan apa yang akan membuat kasus uji bagus?
- Apakah sistem sangat sensitif terhadap nilai masukan tertentu?
- Bagaimana batasan kelas data diisolasi?
- Berapa kecepatan data dan volume data yang dapat ditoleransi oleh sistem?
- Apa pengaruh kombinasi data tertentu terhadap operasi sistem?

Hasil akhir dari pengujian perangkat lunak, menurut McCall, Richards, dan Walters (1977) dalam Pressman (2001 : 509) digambarkan sebagai berikut :



Gambar 13. Faktor-Faktor Kualitas Perangkat Lunak

1. *Product revision* (mudah untuk dilakukan perubahan)
    a. *Maintainability* (pemeliharaan). Diperlukan upaya untuk menemukan dan memperbaiki kesalahan dalam program.
    b. *Flexibility* (fleksibilitas). Upaya yang diperlukan untuk mengubah program operasional.
    c. *Testability* (testabilitas). Upaya yang diperlukan untuk menguji program untuk memastikan bahwa program tersebut menjalankan fungsi yang diinginkan.
2. *Product transition* (adaptasi terhadap perubahan lingkungan)
    a. *Portability* (portabilitas). Upaya yang diperlukan untuk mentransfer program dari satu perangkat keras dan / atau lingkungan sistem perangkat lunak ke yang lain.
    b. *Reusability* (dapat digunakan kembali). Sejauh mana program atau bagian dari program dapat digunakan kembali dalam aplikasi lain, terkait dengan pengemasan dan ruang lingkup fungsi yang dijalankan oleh program.
    c. *Interoperability* (interoperabilitas). Diperlukan upaya untuk menggabungkan satu sistem ke sistem lainnya.
3. *Product operation* (memenuhi kebutuhan informasi dan mudah digunakan)
    a. *Correctness* (ketepatan). Sejauh mana suatu program memenuhi spesifikasinya dan memenuhi tujuan misi pelanggan.
    b. *Reliability* (keandalan). Sejauh mana program dapat diharapkan untuk menjalankan fungsi yang diinginkan dengan presisi yang dibutuhkan.
    c. *Efficiency* (efisiensi). Jumlah sumber daya komputasi dan kode yang dibutuhkan oleh program untuk menjalankan fungsinya.
    d. *Integrity* (integritas). Sejauh mana akses ke perangkat lunak atau data oleh orang yang tidak berwenang dapat dikontrol.
    e. *Usability* (kegunaan). Diperlukan upaya untuk mempelajari, mengoperasikan, menyiapkan masukan, dan menginterpretasikan keluaran suatu program.

D. Analisis Biaya Pengembangan Perangkat Lunak

Untuk menghitung berapa biaya pembuatan sistem perangkat lunak, perlu diketahui rate gaji orang yang bekerja di bidang tersebut. Berikut rata-rata gaji perbulan berdasarkan sumber https://www.qerja.com/journal/view/83-standar-gaji-berbagai-posisi-di-bidang-it dengan menggunakan harga terendah dan nilai tengah.

1. Programmer (Rp 4.000.000,00 – Rp 10.000.000)

   Tugas utama seorang programmer adalah mengaktualisasikan konsep yang datang dari system analyst agar dapat dijalankan dengan menggunakan bahasa pemrograman tertentu.

2. Software Engineer (Rp 5.000.000,00 – Rp 20.000.000)

   Tugas dari seorang software engineer mirip dengan pekerjaan seorang programmer dan system analyst, maka dari itu software engineer diwajibkan paham mengenai keduanya. Yang membedakan software engineer dengan kedua profesi tersebut adalah seorang software engineer harus mendalami ilmu mengenai Software Development Life Cycle (SDLC) sebagai modal untuk mengembangkan software, mulai dari requirement hingga maintenance.

3. IT Consultant (Rp 6.000.000,00 – Rp 15.000.000)

   IT consultant bekerja memberi saran dan ide kepada client tentang bagaimana mengoptimalkan penggunaan teknologi informasi untuk memenuhi target bisnis atau menyelesaikan suatu masalah. IT consultant juga bekerja untuk memperbaiki struktur dan efisiensi dari sistem IT organisasi client.

4. System Analyst (Rp 7.000.000,00 – Rp 25.000.000)

   Pekerjaan utama system analyst adalah merancang solusi IT untuk meningkatkan efisiensi bisnis dan produktifitas organisasi client dengan melakukan pendekatan logis terhadap masalah teknis. System analyst biasanya datang dari seorang programmer yang sudah mahir dan memiliki pengalaman sebagai pengembang software.

5. Database engineer (Rp 5.000.000,00 – Rp 25.000.000)

   Database engineer adalah individu yang ahli dalam SQL dengan pengalaman langsung dalam mengoptimalkan atau men-debug berbagai platform. Database engineer umumnya bekerja untuk mendesain dan memonitor database kompleks. Mereka bertanggung jawab di bidang pemeliharaan yang memastikan database tersebut responsif dan dapat menyediakan data yang valid.

Untuk menghitung biaya pengembangan perangkat lunak ini dilihat dari aktivitas pekerjaan pembuatan sistem yaitu :

1. Analisis kebutuhan sistem informasi; aktivitas ini terdiri dari aktivitas mengumpulkan informasi dari pengguna mengenai perangkat lunak yang dibuat dan aktivitas penentuan sumber daya (teknologi, sumber daya manusia, perangkat keras dan sumber daya lainnya) yang diperlukan untuk membuat perangkat lunak.
2. Perancangan sistem informasi; Aktivitas ini terdiri dari aktivitas merancang *mock-up* perangkat lunak, merancang antarmuka pengguna (*user interface)*, rancangan keamanan, rancangan *output*, dan rancangan lainnya.
3. Perancangan database; dalam banyak hal, data adalah aset terpenting yang dimiliki perusahaan mana pun, dan kemampuan untuk menggunakannya dengan benar dapat mengubah bisnis. Data sering kali terkunci di beberapa sistem yang tidak dapat berbicara satu sama lain secara efektif dan disimpan dalam format berbeda yang menyulitkan untuk mendapatkan wawasan yang berarti. Aktivitas database engineer adalah mendesain atau merancang database susai dengan kebutuhkan saat ini dan saat mendatang, dapat diakseses dengan mudah oleh pemangku kepentingan dengan keamanan data yang handal.
4. Pengembangan sistem informasi; aktivitas in adalah aktivitas menulis kode-kode program aplikasi (*coding*) yang dilakukan untuk mengembangkan perangkat lunak.
5. *User Acceptance Test* (UAT); aktivitas ini adalah aktivitas pengujian yang dilakukan oleh pengguna untuk memastikan semua fitur perangkat lunak yang dikembangkan telah berjalan dengan baik seperti yang direncanakan sebelumnya.
6. Uji kelayakan sistem informasi; aktivitas ini adalah aktivitas pengujian security dan performance dari perangkat lunak yang dikembangkan sebelum diimplementasikan pada server di data center Pusintek.
7. Implementasi sistem informasi; aktivitas ini adalah pengimplementasian perangkat lunak yang telah dikembangkan ke server yang berada di data center.

Setelah mengetahui aktivitas-aktivitas yang terkait dengan pengembangan perangkat lunak, kemudian dihitung biaya aktivitas-aktivitas tersebut. Untuk mengetahui biaya aktivitas, maka kami membentuk dua *resource pools*, yaitu biaya langsung dan biaya tidak langsung. Biaya ini dibagi ke masing-masing aktivitas pengembangan perangkat lunak berdasarkan waktu yang *Developer* habiskan di masing- masing aktivitas. SDM yang

digunakan terdiri dari Software Engineer, IT Consultant, System Analyst, Database engineer, dan Stakeholder.

Tabel 4. Contoh Perhitungan Biaya Pengembangan Sistem

Kegiatan	SDM	Waktu	Biaya		Output
			Alokasi (%)	Jumlah (Rp)	
Analisis kebutuhan sistem informasi	IT Consultant, System Analyst, dan Klien	2 bulan	16%	30.000.000	Laporan Studi Kelayakan
Perancangan sistem informasi	System Analyst	3 bulan	27%	50.000.000	Dokumen Rancangan Sistem
Perancangan Database	Database engineer	2 bulan	8%	15.000.000	Database
Coding	Programmer/Software Engineer, System Analyst, Database engineer	5 bulan	38%	70.000.000	Aplikasi
Testing	Software Engineer, IT Consultant, System Analyst, Database engineer, dan Klien	0,25 bulan	5%	10.000.000	Apliksi Teruji
Emplementasi	System Analyst, Database engineer, Software Engineer, User	0,5 bulan	3%	5.000.000	User Terlatih
Biaya Tidak Langsung dan lainnya	Listrik + Telepon + ATK + Konsumsi	-	3%	5.000.000	
			TOTAL BIAYA	185.000.000	

DAFTAR PUSTAKA

Hall, James A. (2011). Accounting Information Systems. 7ed. Cengage Learning Academic Resource Center, USA.

Simkin, Mark G., Jacob M. Rose, & Carolyn Strand Norman. (2012). Core Concepts of Accounting Information Systems. John Wiley & Sons, Inc. USA.

Jurgianto, HM. (2005). Analisis dan Desain (Sistem Informasi: Pendekatan Terstruktur Teori dan Praktik Aplikasi Bisnis), Penerbit Andi, Yogyakarta.

Pressman, Roger S. (2001). Software Engineering (A Practitioner's Approach), 5ed, Mcgraw-Hill Higher Education, USA.

Satzinger, John W., Robert B. Jackson, & Stephen D. Burd. (2016). Systems Analysis and Design in a Changing World, 7ed, Cengage Learning, USA.

Hanna, M. (1995). Farewell to Waterfalls. Software Magazine, May 1995, hlm. 38-46.

Bradac, M., D. Perry, & L. Votta. (1994). Prototyping a Process Monitoring Experiment. Software Engineering, Vol. SE-20, No. 10, Oktober 1994, hlm. 774-784.

Boehm, B. (1998). Using the WINWIN Spiral Model: A Case Study. IEEE Software, Vol. 31, No. 7, Juli 1998, hlm. 33-44.

Boehm, B. (1996). Anchoring the Software Process. IEEE Software, Vol. 13, No. 4, Juli 1996, hlm. 73-82.

Davis, A. and P. Sitaram. (1994). A Concurrent Process Model for Software

Development. Software Engineering Notes, ACM Press, Vol. 19, No. 2, April 1994, hlm. 38-51.

Nierstrasz, O., S. Gibbs, and D. Tsichritzis. (1992). Component-Oriented Software Development. CACM, Vol. 35, No. 9, September 1992, hlm. 160-165.

Mills, H.D., M. Dyer, and R. Linger. (1987). Cleanroom Software Engineering. IEEE Software, September 1987, hlm. 19-25.

http://www.waskhas.com/2016/07/contoh-gambar-diagram-konteks-dan-dfd.html, diakses 27 Desember 2020.

https://astizardiaz.wordpress.com/2017/01/17/dfd-erd-sistem-informasi-penjualan-barang/diagram-konteks, diakses 27 Desember 2020.

https://eskagrey.wordpress.com/2017/07/24/analisis-dan-perancangan-sistem-kamus-data, diakses 27 Desember 2020.

Novikov, Alexander M., Dmitry A. Novikov, (2013), Research Methodology From Philosophy of Science to Research Design. CRC Press is an imprint of Taylor & Francis Group, an Informa business. Boca Raton.

Kristanto, Andri. (2018). Perancangan Sistem Informasi dan Aplikasinya. Penerbit Gava Media. Yogyakarta.

https://www.codeconquest.com/what-is-coding/common-programming-languages. Common Coding Languages. Diakses Tanggal 28 Desember 2020.

https://www.qerja.com/journal/view/83-standar-gaji-berbagai-posisi-di-bidang-it, Standar Gaji Berbagai Posisi di Bidang IT. Diakses Tanggal 6 Februari 2021.