

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/342914703>

Rekayasa Perangkat Lunak

Book · July 2020

CITATIONS

3

READS

4,518

1 author:



[Edwar Ali](#)

Universitas Sains dan Teknologi Indonesia

15 PUBLICATIONS 21 CITATIONS

SEE PROFILE

**PENERBIT
CV MFA**

REKAYASA PERANGKAT LUNAK



Edwar Ali

2019

REKAYASA PERANGKAT LUNAK



Penulis :
Edwar Ali

Editor :

Penerbit :
CV MFA

Hak Cipta :
Penulis

Alamat :
Jl. Tri Darma 866, Gendeng, Baciro, Gondokusuman, Yogyakarta 085725782088
E-mail : mfapublishing@gmail.com

Cetakan Pertama : Januari 2019

ISBN : 978-602-52449-0-2

Hak cipta @2018 pada penulis

Hak cipta dilindungi undang-undang.

Dilarang memperbanyak atau memindahkan sebagian atau seluruh isi buku ini ke dalam bentuk apapun, secara elektronik maupun mekanis, termasuk memfotocopy, memotret atau dengan teknik perekaman lainnya, tanpa ijin tertulis dari penerbit

**PENERBIT
CV MFA**

2019

Kata Pengantar

Segala puji sudah sepantasnya dihaturkan kepada Allah SWT, karena dengan ridho-NYA penulisan buku ini dapat diwujudkan. Mata kuliah Rekayasa Perangkat Lunak atau dalam istilah lain sering disebut *Software Engineering* merupakan salah satu mata kuliah kunci yang akan membentuk karakter, kemampuan/kompetensi dan kekuatan utama bagi mahasiswa pada program studi ilmu komputer. Lingkup materi untuk mata kuliah ini cukup luas dengan berbagai sasaran dan strategi dalam pembelajarannya. Namun dalam penulisan buku ajar ini telah diseleksi beberapa bagian penting sesuai dengan kebutuhan *stake holder* dan kondisi daerah secara khusus. Walaupun telah diambil beberapa bagian dari sumber-sumber yang digunakan, namun keterkaitan hal-hal yang ada dalam muatan proses rekayasa perangkat lunak tetap dijaga dalam buku ini.

Atas keberhasilan penyusunan dan penerbitan buku ini tidak terlepas dari dukungan berbagai pihak. Oleh karena itu, penulis mengucapkan banyak terima kasih kepada:

1. Direktorat Belmawa Kemenristekdikti di Jakarta yang telah memberikan dukungan material dan moril selama ini.
2. Ketua Yayasan Komputasi Riau yang menaungi institusi STMIK Amik Riau
3. Ketua STMIK Amik Riau
4. Para rekan-rekan dosen dan karyawan di lingkungan STMIK Amik Riau
5. Pihak lain yang terlibat secara langsung maupun tidak dalam proses penulisan buku ini.

Akhirnya penulis menyatakan bahwa buku ini masih jauh dari kesempurnaan. Penulis menerima masukan dan kritikan membangun demi pembaharuan pada terbitan berikutnya.

Pekanbaru, Des 2018

Penulis

Daftar Isi

Kata Pengantar.....	i
Daftar Isi.....	ii
Daftar Gambar.....	v
Pendahuluan.....	vi
BAB I REKAYASA PERANGKAT LUNAK.....	1
Defenisi Komputer.....	1
Sistem Komputer.....	1
Perangkat Lunak.....	4
Aplikasi Perangkat Lunak.....	6
Rekayasa Perangkat Lunak (RPL).....	9
Lapisan (layer) dalam Rekayasa Perangkat Lunak.....	14
Berbagai permasalahan dalam rekayasa perangkat lunak.....	15
BAB II MANAJEMEN PROYEK PERANGKAT LUNAK.....	18
Tujuan Manajemen Proyek.....	19
Proyek Perangkat Lunak.....	20
Pentingnya Manajemen Proyek Perangkat Lunak.....	20
Manajer Proyek Perangkat Lunak.....	22
Aktifitas Manajemen Proyek Perangkat Lunak.....	23
Tool Manajemen Proyek Perangkat Lunak.....	27
BAB III MODEL-MODEL PROSES PERANGKAT LUNAK.....	34
Pandangan umum tentang Rekayasa Perangkat Lunak.....	34
Model Proses Perangkat Lunak.....	35
A. MODEL SEKUENSIAL LINIER (WATERFALL)	36
B. MODEL PROTOTIPE.....	39
C. MODEL RAD.....	43
D. MODEL SPIRAL.....	45
BAB IV SOFTWARE DEVELOPMENT LIFE CYCLE.....	49
Sejarah Software Development Life Cycle (SDLC)	49
Aktifitas dalam SDLC.....	51
Communication.....	50
Requirement Gathering.....	50
Feasibility Study.....	54
System Analysis.....	60
Software Design.....	61
Coding.....	61
Testing.....	61
Integration.....	62
Implementation.....	62
Operation and Maintenance.....	62
BAB V PERSYARATAN PERANGKAT LUNAK.....	63
Rekayasa Kebutuhan.....	64
Proses Rekayasa Kebutuhan.....	64
Requirements Elicitation Process.....	66
Teknik Elisitasi Persyaratan.....	67
Karakteristik Persyaratan Software.....	70
Persyaratan Perangkat Lunak.....	71
Persyaratan Antarmuka Pengguna (user interface)	72

Tips dalam menyusun SKPL.....	73
Review Persyaratan Perangkat Lunak.....	77
BAB VI DESAIN PERANGKAT LUNAK.....	80
Prinsip dalam desain perangkat lunak.....	81
Tingkatan Dalam Desain Perangkat Lunak.....	83
Modularitas.....	84
Konkurensi.....	85
Kopling dan Kohesi.....	86
Verifikasi Desain.....	89
Konsep Desain Perangkat Lunak.....	90
BAB VII PERANGKAT LUNAK ANALISIS DAN TOOL DESAIN.....	101
A. Data Flow Diagram.....	102
B. Structure Chart.....	106
C. HIPO Diagram.....	109
D. Struktur English.....	110
E. Pseudo-Code.....	112
F. Entity Relationship Diagram Model.....	112
G. Data Dictionary (Kamus Data)	113
BAB VIII STRATEGI PERANCANGAN PERANGKAT LUNAK.....	116
A. Perancangan Terstruktur.....	117
B. Perancangan Berorientasi Fungsi.....	121
C. Perancangan Berbasis Objek (Object Oriented Design)	122
Proses Perancangan.....	126
Pendekatan-Pendekatan Perancangan Perangkat Lunak.....	127
BAB IX PERANCANGAN ANTAR MUKA PEMAKAI (USER INTERFACE).....	131
Prinsip Dalam Merancang User Interface.....	132
Memilih Elemen Antarmuka.....	137
Komponen GUI pada Aplikasi Khusus.....	142
Aktifitas Perancangan User Interface.....	142
BAB X IMPLEMENTASI PERANGKAT LUNAK.....	145
Pemrograman Terstruktur.....	145
Pemrograman Berbasis Fungsi.....	147
Gaya Pemrograman.....	148
Dokumentasi Perangkat Lunak.....	150
Tantangan Implementasi Perangkat Lunak.....	152
Berbagai permasalahan perangkat lunak.....	153
Fitur penting dalam pemrograman.....	155
BAB XI PENGUJIAN PERANGKAT LUNAK.....	158
Rencana Pengujian Perangkat Lunak.....	158
Validasi Perangkat Lunak.....	163
Verifikasi Perangkat Lunak.....	163
Pengujian Manual dan Otomatis.....	164
Pendekatan dalam Pengujian.....	165
Jenis-Jenis Metode Pengujian.....	165
Tingkatan Pengujian.....	168
Dokumentasi Pengujian.....	171
Perbedaan Pengujian dengan Kontrol dan Jaminan Kualitas & Audit.....	172
BAB XII PEMELIHARAAN PERANGKAT LUNAK.....	174
Mengapa Pemeliharaan Perlu di Lakukan ?	175
Bentuk-bentuk Pemeliharaan.....	177
Permasalahan Selama Pemeliharaan.....	181
Aktifitas Pemeliharaan.....	182
Tool untuk pemeliharaan perangkat lunak.....	184

BAB XIII REKAYASA PERANGKAT LUNAK MASA DEPAN.....	186
Pondasi yang lemah.....	187
Wujud Risiko.....	188
Pusaran Kompleksitas.....	188
Peningkatan Total Biaya Kepemilikan.....	188
Menguasai Inovasi Berbasis Perangkat Lunak.....	189
Bidang Rekayasa Perangkat Lunak tidak akan pernah mati.....	190
DAFTAR PUSTAKA	

Daftar Gambar

Gambar 1.1: Elemen perangkat keras komputer.....	2
Gambar 1.2: Ilustrasi perangkat lunak.....	3
Gambar 1.3: Elemen pembentuk sistem komputer.....	4
Gambar 1.4 : Wilayah cakupan RPL.....	11
Gambar 1.5: Lapisan dalam Rekayasa Perangkat Lunak.....	15
Gambar 2.1 : Kendala dalam proyek perangkat lunak.....	21
Gambar 2.2: Aktivitas perencanaan proyek.....	25
Gambar 2.3: Gantt Chart.....	30
Gambar 2.4: Contoh PERT.....	32
Gambar 3.1 : Model Sekuensial Linier (Waterfall Model)	36
Gambar 3.2: Model Prototipe.....	41
Gambar 3.3 : Model RAD (Rapid Application Development)	44
Gambar 3.4 : Model Spiral.....	47
Gambar 4.1 : Tahapan Software Development Life Cycle.....	50
Gambar 4.2: Bentuk-bentuk kelayakan.....	55
Gambar 5.1 : Proses elisitasi kebutuhan.....	66
Gambar 5.2: Kegiatan Wawancara.....	68
Gambar 6.1. Bentuk ilustrasi konsep modularitas.....	85
Gambar 6.2: Model kinerja modulitas.....	95
Gambar 6.3: Partisi Horizontal dan Vertikal.....	98
Gambar 7.1: Simbol-simbol DFD.....	104
Gambar 7.2 : Elemen module.....	107
Gambar 7.3 : Elemen Kondisi.....	107
Gambar 7.4: Kondisi Jump.....	108
Gambar 7.5 : Aturan loop.....	108
Gambar 7.6: Pola aliran data.....	109
Gambar 7.7 : Kontrol Flow.....	109
Gambar 7.8 : Bentuk HIPO Chart.....	110
Gambar 7.9: Entity Relationship Diagram.....	113
Gambar 7.10: Notasi dalam kamus data.....	115
Gambar 8.1: Konsep Cohesion dan Coupling.....	120
Gambar 8.2: Ilustrasi Top-Down Design.....	128
Gambar 8.3: Ilustrasi Bottom-Up Design.....	129
Gambar 9.1: Bentuk tampilan user interface berbasis CLI.....	139
Gambar 9.2: Elemen pada GUI.....	140
Gambar 9.3: Tahap Perancangan dan Pengembangan GUI.....	143
Gambar 10.1: Fitur penting dalam program.....	156
Gambar 11.1: Langkah-langkah yang terlibat dalam melakukan pengujian	161
Gambar 11.2: Ilustrasi proses blackbox testing.....	166
Gambar 11.3: Ilustrasi White-Box Testing.....	167
Gambar 11.4: Ilustrasi Pengujian Unit.....	168
Gambar 12.1: Komposisi berbagai jenis pemeliharaan (maintenance)	180
Gambar 12.2: Ilustrasi aktivitas pemeliharaan.....	182
Gambar 13.1: Gambaran umum teknologi ‘Cerdas’.....	187
Gambar 13.2: Peningkatan biaya kepemilikan.....	189
Gambar 13.3: Inovasi Berbasis Perangkat Lunak untuk Produk & Layanan Cerdas.....	190

REKAYASA PERANGKAT LUNAK



Pendahuluan

Selama tiga dekade pertama dari era komputerisasi, tantangan utama adalah mengembangkan hardware komputer yang dapat mengurangi biaya pengolahan dan penyimpanan data. Selama dekade tahun 1980 an, kemajuan pesat dari mikro elektronika menghasilkan kemampuan komputer yang lebih baik pada tingkat biaya yang lebih rendah. Namun masalah sekarang berbeda. Tantangan utama adalah mengurangi biaya dan memperbaiki kualitas solusi berbasis komputer (Solusi yang diimplementasikan dengan menggunakan software).

Software merupakan faktor kunci dalam keberhasilan suatu usaha, software dapat membedakan satu perusahaan dari perusahaan saingannya.

Perancangan perangkat lunak adalah disiplin manajerial dan teknis yang berkaitan dengan pembuatan dan pemeliharaan produk perangkat lunak secara sistematis, termasuk pengembangan dan modifikasinya, yang dilakukan pada waktu yang tepat dan dengan mempertimbangkan faktor biaya.

Tujuan perancangan perangkat lunak adalah untuk

- (1) memperbaiki kualitas produk perangkat lunak,
- (2) meningkatkan produktivitas, dan
- (3) memuaskan teknisi perangkat lunak.

Produk perangkat lunak adalah perangkat lunak yang digunakan oleh berbagai pengguna, bukan untuk pengguna pribadi. Dalam buku ajar ini akan dibahas tentang bagaimana suatu software tersebut dikembangkan melalui tahapan-tahapan yang sistematis beserta personil-personil yang terlibat dalam setiap aktivitas yang dilakukan.

BAB I

REKAYASA PERANGKAT LUNAK

Tujuan :

1. Mengenalkan sistem komputer dan perangkat lunak
2. Mempelajari klasifikasi perangkat lunak
3. Mengenalkan rekayasa perangkat lunak dan aplikasinya

Indikator keberhasilan :

1. Agar mahasiswa mampu menjelaskan komponen sistem komputer
2. Agar mahasiswa mampu menjelaskan pengertian perangkat lunak dan contohnya
3. Agar mahasiswa mampu menyebutkan dan menjelaskan contoh jenis dan aplikasi perangkat lunak
4. Agar mahasiswa mampu memahami tujuan dari rekayasa perangkat lunak.

Materi :

Definisi Komputer

Komputer merupakan suatu perangkat elektronika yang memiliki kemampuan untuk menerima dan mengolah data menjadi informasi, menjalankan program yang tersimpan dalam memori, serta dapat bekerja secara otomatis berdasarkan perangkat aturan tertentu. Berdasarkan uraian di atas, maka dapat diuraikan bahwa setidaknya suatu komputer harus memenuhi kaidah-kaidah berikut ini:

- a. Komputer dapat melakukan pengolahan data
- b. Komputer dapat memberikan/menghasilkan informasi
- c. Komputer merupakan alat elektornik

- d. Komputer dapat menerima input data (teks, angka, suara, signal, dll)
- e. Komputer menggunakan program yang tersimpan dalam memori komputer
- f. Komputer bekerja secara otomatis
- g. Komputer dapat menyimpan program dan data hasil olahannya.

Sistem Komputer

Sebuah sistem komputer tersusun atas tiga elemen yang saling terkait satu sama lainnya, yaitu :

1. *Hardware* (Perangkat Keras), merupakan kumpulan segala piranti atau komponen dari sebuah komputer yang sifatnya bisa dilihat secara kasat mata dan bisa diraba secara langsung. Dengan kata lain hardware merupakan komponen yang memiliki bentuk nyata secara fisik. Beberapa komponen perangkat keras mudah dikenali, seperti *casing* komputer, *keyboard*, dan monitor. Namun, ada banyak jenis komponen perangkat keras yang lainnya untuk membentuk sebuah komputer.



Gambar 1.1: Elemen perangkat keras komputer

2. *Software* (Perangkat Lunak), merupakan suatu data yang diprogram sedemikian rupa dan disimpan dalam bentuk digital yang tidak terlihat secara fisik tetapi tersimpan dalam media penyimpanan komputer. Software atau perangkat lunak dapat berupa program atau aktifitas menjalankan suatu perintah atau intruksi melalui fasilitas interaksi pada *software* (perangkat lunak) komputer sehingga sistem dapat beroperasi. Software juga dapat dikatakan sebagai penggerak dan pengendali hardware (perangkat keras).

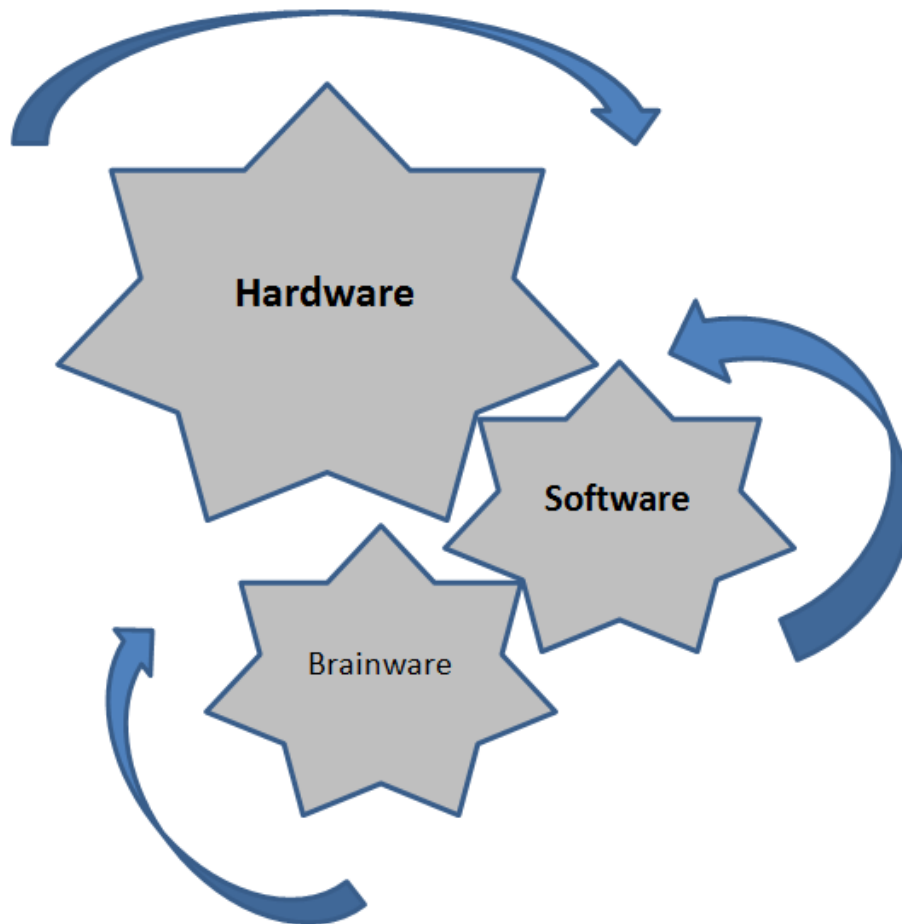


Gambar 1.2: Ilustrasi perangkat lunak

Software dibuat dengan menggunakan bahasa pemrograman yang ditulis atau dibangun oleh programmer yang selanjutnya dikompilasi dengan aplikasi kompilasi sehingga menjadi sebuah kode yang nantinya akan dikenali oleh mesin *hardware*.

3. *Brainware* (Perangkat Manusia/Otak), merupakan orang yang menggunakan (*user*), memakai ataupun mengoperasikan perangkat komputer. Seperti contoh dari *brainware* yaitu *programmer*, *operator* (sebutan untuk orang yang menggunakan suatu sistem komputer untuk fungsi tertentu), serta orang yang sedang menggunakan perangkat komputer. *Brainware* juga dapat

didefinisikan sebagai manusia yang terlibat dalam mengoperasikan atau memakai serta mengatur sistem di dalam perangkat komputer. Dapat diartikan juga sebagai perangkat intelektual yang mengoperasikan dan juga mengeksplorasi kemampuan dari perangkat keras (*hardware*) maupun perangkat lunak (*software*).



Gambar 1.3: Elemen pembentuk sistem komputer

Perangkat Lunak

Pada mata kuliah Rekayasa Perangkat Lunak (RPL) ini, sebelumnya kita harus mengerti dan memahami terlebih dahulu, apa yang dimaksud dengan perangkat lunak. Perangkat lunak saat ini telah menjadi kekuatan baru yang sangat menentukan dalam mendukung suatu aktifitas. Perangkat lunak menjadi mesin yang mengendalikan proses pengambilan

keputusan di dalam dunia bisnis, berfungsi sebagai basis dari berbagai bentuk pelayanan serta penelitian keilmuan modern. Saat ini perangkat lunak memiliki dua peran. Di satu sisi berfungsi sebagai sebuah produk, dan di sisi lain sebagai media yang mengantarkan sebuah produk.

Di masa lalu, perangkat lunak bersifat sederhana dan karenanya, pengembangan perangkat lunak merupakan kegiatan yang juga sederhana. Namun, seiring meningkatnya teknologi dan peradaban manusia, perangkat lunak menjadi lebih kompleks dan proyek perangkat lunak tumbuh menjadi lebih besar. Pengembangan perangkat lunak sekarang mengharuskan kehadiran tim yang dapat mempersiapkan rencana dan desain secara terperinci, melakukan pengujian, mengembangkan antar muka pengguna yang intuitif, dan mengintegrasikan semua aktifitas ini ke dalam sistem. Pendekatan baru ini menyebabkan munculnya sebuah disiplin ilmu yang dikenal sebagai rekayasa perangkat lunak.

Sebenarnya, apa yang dimaksud dengan perangkat lunak?

Perangkat lunak (*Software*) adalah: (1) Perintah/instruksi (program komputer) yang mana bila ia dieksekusi akan memberikan fungsi dan unjuk kerja seperti yang diinginkan. (2) Struktur data yang memungkinkan program memanipulasi data dan informasi secara proporsional. (3) Dokumen yang menggambarkan operasi dan kegunaan program.

Klasifikasi Perangkat Lunak :

1. Sistem Operasi (*operating system*), merupakan perangkat lunak yang berfungsi untuk mengoperasikan komputer serta menyediakan antarmuka dengan perangkat lunak lain atau dengan pengguna. Contoh perangkat lunak Sistem Operasi : MS DOS, MS Windows (dengan berbagai generasi), Macintosh, OS/2, UNIX (dengan

berbagai versi), LINUX (dengan berbagai distribusi), NetWare, dan sebagainya. Masing-masing sistem operasi memiliki perbedaan dalam lingkup/platform operasi, jumlah pemakai, metode interaksi pemakai, sifatnya (*opensource/closesource*) dan lisensi penggunaan.

2. Program Utilitas (*Utility*), merupakan program khusus yang berfungsi sebagai perangkat pemeliharaan suatu sistem komputer, seperti anti virus, partisi hardisk, manajemen hardisk, dan sebagainya. Contoh produk program utilitas : *Norton Utilities*, *PartitionMagic*, *McAfee*, dan sebagainya
3. Program Aplikasi, merupakan program yang dikembangkan untuk memenuhi kebutuhan yang spesifik. Contoh : aplikasi akuntansi, aplikasi perbankan, aplikasi manufaktur, dan sebagainya.
4. Paket Pemrograman, merupakan program yang dikembangkan untuk kebutuhan umum, seperti : pengolah kata/*text editor* (misalnya: *MS-Word*, *Notepad*, *Latex*, dll), pengolah angka / lembar kerja (misalnya: *MS Excel*, dll), presentasi (Misalnya: *MS PowerPoint*, dll), dan desain grafis (Misalnya: *3D*, *CorelDraw*, *PhotoShop*, dan sebagainya)
5. Bahasa Pemrograman, merupakan sebuah instruksi standar yang bertugas untuk memberikan instruksi kepada komputer. Sering disebut juga dengan bahasa komputer atau bahasa pemrograman komputer. Bahasa pemrograman juga dapat dikatakan sebagai alat untuk menampung suatu kumpulan dari aturan sintaks dan semantik yang khususnya dipakai untuk mendefinisikan sebuah program yang ada di komputer.

Contoh bahasa pemrograman yang paling umum digunakan, yaitu : *Java Script*, *PHP*, *HTML*, *C++*, *Python*, dan sebagainya.

Aplikasi Perangkat Lunak

Perangkat lunak dapat diaplikasikan ke berbagai situasi dan lingkungan di mana serangkaian langkah prosedural telah didefinisikan. Cakupan perangkat lunak berikut ini menunjukkan betapa luasnya potensi pengembangan aplikasi yang dapat dilakukan :

a. Perangkat Lunak Sistem.

Merupakan sekumpulan program yang ditulis untuk memberikan layanan terhadap program-program yang lain. Di dalam berbagai kasus, area operasi perangkat lunak sistem ditandai dengan eratnya interaksi dengan perangkat keras komputer, penggunaan oleh banyak pemakai (*multi user*), operasi konkuren yang membutuhkan penjadwalan, berbagi pakai sumber daya dan pengaturan proses yang canggih, struktur-struktur data yang kompleks, dan interface eksternal yang bersifat ganda (teks dan grafis).

b. Perangkat Lunak *Real-Time*.

Program-program yang memonitori, menganalisa, dan mengontrol kejadian di dunia nyata pada saat berlangsungnya suatu aktifitas disebut perangkat lunak *real-time*. Elemen-elemen perangkat lunak *real-time* mencakup komponen pengumpul data yang mengumpulkan dan memformat informasi dari lingkungan eksternal. Sebuah komponen yang mampu melakukan analisa diperlukan untuk mentransformasi informasi pada saat dibutuhkan oleh aplikasi. Konsep kerja perangkat lunak *Real-time* berbeda dengan fungsi interaksi atau *timesharing*. Sistem *real-time* harus merespon di dalam suatu rentang waktu secara konstan dan konsisten. Sedangkan waktu respon sebuah sistem yang bersifat interaktif (atau *timesharing*) secara normal dapat diperpanjang tanpa memberikan resiko kerusakan pada hasil yang diperoleh.

c. Perangkat Lunak Bisnis.

Pemrosesan informasi bisnis merupakan area aplikasi perangkat lunak yang paling luas saat ini. Sistem diskrit (contohnya *payroll*, *account payable*, *inventory*, dsb) telah mengembangkan perangkat lunak sistem informasi manajemen (SIM) yang mengakses satu atau lebih database besar yang berisi informasi bisnis.

d. Perangkat Lunak Teknik dan Ilmu Pengetahuan.

Perangkat lunak teknik dan ilmu pengetahuan ditandai dengan keberadaan *algoritma number crunching*. Perangkat lunak ini memiliki cakupan aplikasi mulai dari aplikasi untuk sistem astronomi sampai vulkanologi, dari analisa otomotif sampai dinamika orbit pesawat ruang angkasa, dan dari biologi molekuler sampai pabrik yang sudah diotomatisasi.

e. *Embedded Software*.

Produk pintar telah menjadi bagian yang umum bagi hampir semua level konsumen dan pasar industri. *Embedded software* dapat memberikan fungsi yang terbatas serta fungsi *esoteric* (misalnya *keypad control* untuk sebuah *microwave*) atau memberikan kemampuan kontrol dan fungsi yang penting (contohnya fungsi digital dalam sebuah mobil seperti kontrol bahan bakar, penampilan panel di *dashboard*, sistem pengereman, dan sebagainya).

f. Perangkat Lunak Komputer Personal.

Pasar perangkat lunak komputer personal telah berkembang selama dekade terakhir. Program pengolah kata, *spreadsheet*, multimedia, manajemen database, aplikasi keuangan bisnis dan personal, jaringan eksternal atau akses database hanya merupakan beberapa contoh saja dari ratusan aplikasi yang ada. Setiap waktu jumlah, variasi, versi, dan teknologinya terus mengalami kemajuan yang pesat.

g. Perangkat Lunak Kecerdasan Buatan.

Perangkat lunak kecerdasan buatan (*Artificial Intelligent/AI*) menggunakan algoritma non-numeris untuk memecahkan masalah kompleks yang tidak dapat dilakukan perhitungan atau analisa secara langsung. Area kecerdasan buatan yang aktif adalah sistem pakar, disebut juga sistem berbasis pengetahuan. Sistem yang lain adalah Jaringan Syaraf Tiruan, *Voice and Image Recognition*, *game playing*, dan sebagainya.

Rekayasa Perangkat Lunak (RPL)

Pandangan umum, menyatakan bahwa untuk dapat memproduksi suatu produk didukung oleh elemen-elemen yang dikenal sebagai faktor-faktor produksi. Elemen-elemen yang terdapat dalam faktor-faktor produksi tersebut adalah Sumber daya alam/ fisik (*Physical Resources*), Sumber daya manusia/ Tenaga kerja (*Labour*), Modal (*Capital*), Kewirausahaan (*Entrepreneurship*), dan Sumber daya informasi (*Information Resources*). Menghadapi era revolusi industri 4.0, elemen-elemen yang disebutkan di atas tidaklah cukup. Oleh karena itu perlu tambahan berupa dukungan teknologi. Dukungan teknologi dari berbagai bentuk disiplin akan berkolaborasi dalam menghasilkan suatu produk. Termasuk produk berupa perangkat lunak.

Untuk menghasilkan suatu perangkat lunak (*software*) tersebut butuh dukungan banyak aspek, proses, dan tindakan yang semuanya itu menyatu dalam suatu keilmuan yang disebut dengan Rekayasa Perangkat Lunak (*Software Engineering*). Secara sederhana target dari Rekayasa Perangkat Lunak itu adalah dalam rangka menghasilkan suatu *software* yang berkualitas.

Sedangkan yang dimaksud dengan Rekayasa Perangkat Lunak adalah suatu disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal yaitu *communication*, *requirements capturing* (analisa kebutuhan pengguna), *specification* (menentukan spesifikasi dari kebutuhan pengguna), desain, *coding*, *testing* sampai *maintenance* (pemeliharaan sistem) setelah digunakan.

Secara umum, tujuan Rekayasa Perangkat Lunak (RPL) memiliki banyak kesamaan dengan disiplin ilmu rekayasa yang lain. Ilmu rekayasa akan selalu berusaha menghasilkan keluaran (*output*) yang performanya tinggi, waktu penyelesaian yang singkat dan hemat. Secara lebih khusus dapat

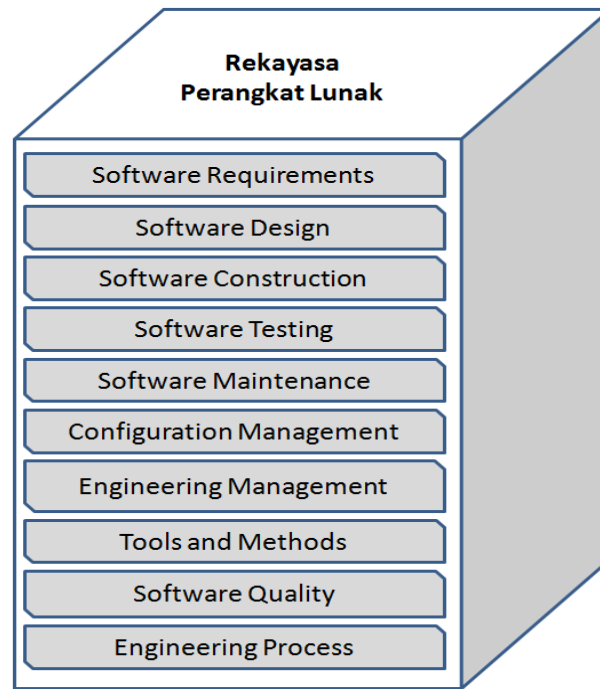
dinyatakan bahwa tujuan Rekayasa Perangkat Lunak adalah sebagai berikut:

1. Menghasilkan perangkat lunak yang berunjuk kerja tinggi, andal serta tepat waktu.
2. Menghasilkan perangkat lunak yang memiliki kekuatan dalam menghadapi berbagai ancaman dan gangguan, baik yang bersumber dari internal maupun eksternal.
3. Menghasilkan perangkat lunak dengan biaya produksi yang rendah.
4. Menghasilkan perangkat lunak yang biaya pemeliharaan yang rendah.
5. Menghasilkan perangkat lunak yang bisa bekerja di berbagai macam platform.

Rekayasa perangkat lunak menyediakan metode untuk menangani kompleksitas dalam sistem perangkat lunak dan memungkinkan untuk dilakukannya pengembangan sistem perangkat lunak yang handal. Muara dari aktifitas ini dapat memaksimalkan produktifitas. Selain aspek teknis pengembangan perangkat lunak, ia juga mencakup kegiatan manajemen yang meliputi mengarahkan tim kerja, menyusun anggaran, menyiapkan jadwal kegiatan, dan sebagainya. Gagasan tentang rekayasa perangkat lunak pertama kali diajukan pada tahun 1968. Semenjak itu, rekayasa perangkat lunak berkembang sebagai disiplin ilmu teknik secara penuh, yang diterima sebagai bidang yang melibatkan studi dan penelitian yang mendalam. Metode dan alat bantu rekayasa perangkat lunak telah berhasil diimplementasikan dalam berbagai skala aplikasi yang tersebar di berbagai lapisan masyarakat pengguna.

Jadi hal yang perlu digaris bawahi, bahwa Rekayasa Perangkat Lunak (RPL) merupakan serangkaian proses yang amat panjang untuk membuat atau menciptakan suatu perangkat lunak yang berkualitas, bukan merupakan cabang ilmu komputer yang mempelajari tentang *technical coding*.

Apabila dilihat dari persepsi wilayah cakupannya, RPL memiliki wilayah operasional yang dapat dilihat pada Gambar 1.4 berikut ini.



Gambar 1.4 : Wilayah cakupan RPL

1. ***Software Requirements.***

Dalam proses rekayasa perangkat lunak, fase ini adalah aktifitas pertama yang harus dilakukan. Fase ini didominasi oleh peran pengguna dalam menerjemahkan ide atau pandangannya ke dalam dokumen persyaratan. Hal yang harus diperhatikan bahwa mendefinisikan dan mendokumentasikan kebutuhan pengguna secara singkat dan tidak ambigu adalah langkah besar pertama untuk mencapai produk berkualitas tinggi.

Fase ini meliputi serangkaian tugas, yang membantu menentukan dampak perangkat lunak pada organisasi, kebutuhan pelanggan, dan bagaimana pengguna akan berinteraksi dengan perangkat lunak yang dikembangkan. Persyaratan yang telah ditentukan adalah dasar dari desain sistem. Jika persyaratan tidak benar, produk akhir juga akan mengandung kesalahan. Perlu dicatat bahwa aktifitas penetapan persyaratan adalah sama seperti semua aktifitas rekayasa perangkat lunak lainnya di mana harus disesuaikan dengan

kebutuhan proses, proyek, produk, dan orang-orang yang terlibat di dalamnya. Juga, persyaratan harus ditentukan pada tingkat detail yang berbeda. Hal ini karena persyaratan tersebut dimaksudkan untuk personil seperti pengguna, manajer bisnis, sistem *engineer*, dan sebagainya. Sebagai contoh, manajer bisnis tertarik untuk mengetahui fitur apa yang dapat diimplementasikan dalam anggaran yang dialokasikan sedangkan pengguna akhir tertarik untuk mengetahui betapa mudahnya menggunakan fitur perangkat lunak.

2. Software Design.

Secara umum yang meliputi proses penampilan arsitektur, komponen, antar muka (*interface*), dan karakteristik lain dari suatu perangkat lunak. *Software design* adalah salah satu fase dalam rekayasa perangkat lunak, di mana terdapat cetak biru yang dikembangkan untuk memberikan layanan sebagai dasar untuk membangun sistem perangkat lunak. IEEE mendefinisikan *software design* sebagai aktifitas mendefinisikan: arsitektur, komponen, antarmuka, dan karakteristik lain dari suatu sistem atau komponen dan hasil dari proses itu.

Pada tahap desain, banyak keputusan penting dan strategis yang dibuat untuk mencapai fungsionalitas dan kualitas sistem yang diinginkan. Keputusan ini harus diperhitungkan agar berhasil dalam mengembangkan perangkat lunak dan melaksanakan pemeliharaannya sehingga kualitas produk akhir ditingkatkan.

3. Software Construction.

Kegiatan ini meliputi aktifitas yang berhubungan dengan hal-hal detil dalam pengembangan perangkat lunak, termasuk algoritma, pengkodean, pencarian kesalahan (*debug*) dan pengujian (*testing*).

4. Software Testing.

Secara umum kegiatan ini meliputi pengujian pada kinerja perangkat lunak secara keseluruhan. Pengujian perangkat lunak ditujukan untuk menentukan kebenaran, kelengkapan dan

kualitas perangkat lunak yang sedang dikembangkan. IEEE mendefinisikan pengujian sebagai proses melaksanakan atau mengevaluasi sistem atau komponen sistem dengan cara manual atau otomatis untuk memverifikasi bahwa hal itu memenuhi persyaratan yang ditentukan atau untuk mengidentifikasi perbedaan antara hasil yang diharapkan dan fakta yang ditemukan.

Pengujian perangkat lunak terkait erat dengan verifikasi syarat dan validasi. Verifikasi mengacu pada proses memastikan bahwa perangkat lunak dikembangkan sesuai dengan spesifikasinya. Untuk verifikasi, teknik seperti ulasan, analisis, inspeksi, dan penelusuran umum digunakan. Sedangkan validasi mengacu pada proses pengecekan bahwa perangkat lunak yang dikembangkan memenuhi persyaratan yang ditentukan oleh pengguna.

5. **Software Maintenance.** Secara umum aktifitasnya mencakup upaya-upaya perawatan ketika perangkat lunak telah dioperasikan. Perangkat lunak tidak pernah usang. Namun, perlu adanya peningkatan untuk memenuhi persyaratan pengguna yang terus berubah secara dinamis. Untuk melakukan pembaharuan seperti itu maka sistem perangkat lunak harus dilakukan *maintenance* (pemeliharaan). IEEE mendefinisikan *maintenance* sebagai suatu proses memodifikasi sistem perangkat lunak atau komponen setelah di-*delivery* untuk memperbaiki kesalahan, untuk meningkatkan kinerja atau atribut lainnya atau untuk menyesuaikan produk ke lingkungan yang berubah secara dinamis. Tujuannya adalah untuk memastikan bahwa perangkat lunak dapat mengakomodasi perubahan setelah sistem telah di-*delivery* dan digunakan.

6. **Software Configuration Management (SCM).**

Software Configuration Management (SCM) adalah suatu disiplin yang secara sistematis mengendalikan perubahan yang terjadi

selama pengembangan. SCM adalah proses yang terpisah dari proses pengembangan karena sebagian besar model pengembangan tidak dapat mengakomodasi perubahan kapan saja selama pengembangan.

7. **Software Engineering Management** yang berkaitan dengan pengelolaan dan pengukuran RPL, termasuk perencanaan proyek perangkat lunak.
8. **Software Engineering Tools and Methods** yang mencakup kajian teoritis tentang alat bantu (*tools*) dan metode RPL.
9. **Software Quality** yang menitikberatkan pada kualitas dan daur hidup perangkat lunak.
10. **Software Engineering Process** berhubungan dengan implementasi, definisi, pengukuran, pengelolaan, perubahan dan perbaikan proses Rekayasa Perangkat Lunak.

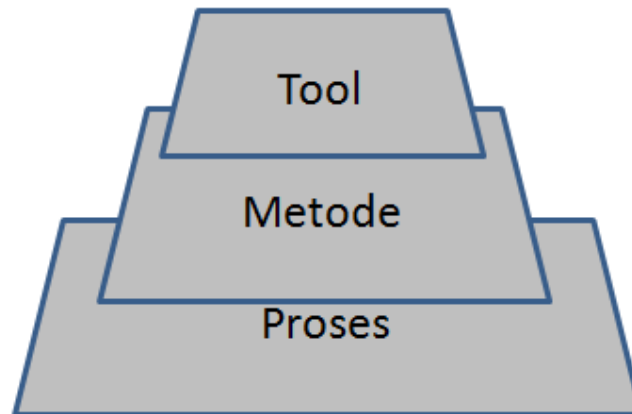
Lapisan (layer) dalam Rekayasa Perangkat Lunak

Rekayasa perangkat lunak dapat dipandang sebagai teknologi yang berlapis seperti yang tersusun sebagai berikut:

1. **Lapisan proses** memungkinkan pengembangan perangkat lunak dilakukan tepat waktu. Ini mendefinisikan serangkaian kegiatan secara garis besar yang harus disepakati untuk men-*delivery* teknologi rekayasa perangkat lunak secara efektif.
2. **Lapisan metode** memberikan pengetahuan teknis untuk mengembangkan perangkat lunak. Lapisan ini mencakup beragam tugas yang mencakup analisis kebutuhan, desain, pengkodean, pengujian, dan fase pemeliharaan pengembangan perangkat lunak.
3. **Lapisan alat (tool)** memberikan dukungan terkomputerisasi atau semi-terkomputerisasi untuk lapisan proses dan metode. Terkadang alat diintegrasikan sedemikian rupa sehingga tool lain dapat menggunakan informasi yang dibuat oleh satu tool.

Multi-penggunaan ini biasanya disebut sebagai *Computer-Aided Software Engineering* (CASE). CASE menggabungkan database

perangkat lunak, perangkat keras, dan rekayasa perangkat lunak untuk membuat rekayasa perangkat lunak yang analog dengan *Computer-Aided Design* (CAD) untuk perangkat keras. CASE membantu dalam pengembangan aplikasi termasuk analisis, desain, pembuatan kode, dan debugging dan pengujian. Ini dimungkinkan dengan menggunakan alat CASE, yang menyediakan metode otomatis untuk merancang dan mendokumentasikan teknik pemrograman struktur tradisional. Sebagai contoh, dua teknologi terkemuka yang menggunakan alat CASE adalah workstation berbasis PC dan generator aplikasi yang menyediakan antarmuka berbasis grafis untuk mengotomatisasi proses pengembangan.



Gambar 1.5: Lapisan dalam Rekayasa Perangkat Lunak

Berbagai permasalahan dalam rekayasa perangkat lunak

Rekayasa perangkat lunak adalah pendekatan sistematis untuk pengembangan, operasi, pemeliharaan, dan penghentian penggunaan suatu perangkat lunak. Ada beberapa masalah mendasar yang dihadapi rekayasa perangkat lunak, antara lain:

1. Masalah Ruang Lingkup (cakupan).

Masalah mendasar dari rekayasa perangkat lunak adalah masalah ruang lingkup. Pengembangan sistem yang sangat besar membutuhkan serangkaian metode yang sangat berbeda dibandingkan dengan mengembangkan sistem yang kecil. Dengan

kata lain, metode yang digunakan untuk mengembangkan sistem kecil umumnya tidak sama dengan sistem yang besar. Satu rangkaian metode yang berbeda harus digunakan untuk mengembangkan perangkat lunak besar. Setiap proyek besar melibatkan penggunaan teknologi dan manajemen proyek.

Untuk proyek perangkat lunak, dengan teknologi yang dimaksud adalah metode, prosedur dan alat yang digunakan. Dalam proyek kecil, metode informal untuk pengembangan dan manajemen dapat digunakan. Namun, untuk proyek besar, keduanya harus jauh lebih formal.

Saat berurusan dengan proyek perangkat lunak kecil, persyaratan teknologi rendah dan persyaratan manajemen proyek juga rendah. Namun, ketika skala berubah menjadi sistem besar, untuk menyelesaikan masalah seperti itu dengan benar, penting bagi kita untuk bergerak ke dua arah yaitu metode yang digunakan untuk pembangunan dan manajemen proyek untuk proyek pengembangan juga harus lebih formal.

2. Biaya, jadwal dan kualitas.

Biaya pengembangan sistem adalah biaya sumber daya yang digunakan untuk sistem, yang dalam sisi pandang perangkat lunak, adalah tenaga kerja, perangkat keras, perangkat lunak, dan sumber daya pendukung lainnya. Secara umum, komponen tenaga kerja bersifat dominan, karena pengembangan perangkat lunak sebagian besar padat karya dan biaya sistem komputasi sekarang cukup rendah.

Oleh karena itu, biaya proyek perangkat lunak diukur dalam hal orang/bulan, yaitu biaya dianggap sebagai jumlah total orang/bulan yang dihabiskan dalam proyek. Jadwal adalah faktor penting dalam banyak proyek. Tren bisnis menentukan bahwa waktu untuk memasarkan suatu produk harus dikurangi; artinya, waktu siklus dari konsep ke *delivery* harus kecil. Setiap bisnis dengan persyaratan seperti itu juga akan mensyaratkan bahwa waktu siklus untuk

membangun perangkat lunak yang dibutuhkan oleh bisnis menjadi lebih kecil.

Salah satu faktor utama yang mendorong aspek produksi adalah kualitas. Kualitas produk perangkat lunak memiliki tiga dimensi, yaitu:

- Operasi Produk
- Transisi Produk
- Revisi Produk

3. **Masalah konsistensi.**

Meskipun telah memiliki kualitas tinggi, biaya rendah dan waktu siklus kecil adalah tujuan utama dari setiap proyek, namun untuk organisasi ada tujuan lain yang ingin dicapai yaitu konsistensi. Sebuah organisasi yang terlibat dalam pengembangan perangkat lunak tidak hanya menginginkan biaya rendah dan kualitas tinggi untuk suatu proyek, tetapi menginginkannya secara konsisten.

Soal :

1. Sebutkan dan jelaskan elemen pembentuk suatu sistem komputer.
2. Jelaskan secara ringkas bentuk keterkaitan antara masing-masing elemen yang membentuk sistem komputer.
3. Uraikan dengan ringkas, makna dasar yang melekat pada definisi perangkat lunak (*software*)
4. Buatlah pengklasifikasian perangkat lunak dan jelaskan kegunaan masing-masingnya. Berikan contoh.
5. Uraikan dengan ringkas, pengertian Rekayasa Perangkat Lunak (*Software Engineering*).
6. Apakah tujuan mendasar dari kegiatan Rekayasa Perangkat Lunak dan apa indikasi bahwa kegiatan itu berhasil?

BAB II

MANAJEMEN PROYEK PERANGKAT LUNAK

Tujuan :

1. Mengenalkan tahapan proyek perangkat lunak
2. Mempelajari berbagai aktifitas dalam manajemen proyek perangkat lunak
3. Mempelajari tugas dan peran seorang manajer proyek perangkat lunak
4. Mengenalkan tool untuk proses perencanaan proyek perangkat lunak

Indikator Keberhasilan :

1. Mahasiswa mampu menjelaskan masing-masing tahapan dalam proyek perangkat lunak.
2. Mahasiswa memahami mekanisme dari kegiatan manajemen proyek perangkat lunak.
3. Mahasiswa mampu menjelaskan peran seorang manajer proyek dalam menentukan keberhasilan pencapaian suatu proyek perangkat lunak.
4. Mahasiswa mampu menerapkan tool-tool dalam sebuah contoh proyek perangkat lunak sederhana.

Materi :

Bentuk aktifitas dalam perusahaan yang bergerak dalam bidang IT, terkait dengan pengembangan perangkat lunak, dapat diklasifikasikan ke dalam dua bentuk, yaitu:

1. Pembuatan Perangkat Lunak
2. Manajemen Proyek Perangkat Lunak

Suatu proyek merupakan tugas yang terdefinisikan dengan baik, yang terdiri dari sekumpulan kegiatan yang dilakukan dengan kondisi terbatas untuk mencapai suatu tujuan yang spesifik. Suatu proyek memiliki karakteristik sebagai berikut:

- a. Setiap proyek memiliki tujuan yang **unik**/ spesifik.
- b. Proyek bukanlah merupakan kegiatan rutin, atau operasional harian
- c. Setiap proyek memiliki waktu mulai dan selesai.
- d. Proyek berakhir apabila tujuan telah tercapai.
- e. Proyek membutuhkan sumber daya selama ia berlangsung, seperti waktu, sumber daya manusia, keuangan, material, dan ilmu pengetahuan.

Tujuan Manajemen Proyek

Proyek merupakan serangkaian rencana aktivitas yang saling berhubungan untuk mencapai tujuan yang spesifik. Proyek sistem informasi termasuk pembangunan sistem informasi baru, peningkatan sistem yang ada atau pergantian infrastruktur teknologi informasi perusahaan. Manajemen proyek mengacu pada penerapan pengetahuan, keterampilan, peralatan dan teknik untuk mencapai target tertentu dengan anggaran dan waktu yang telah ditentukan.

Kegiatan manajemen proyek mencakup aktivitas perencanaan pekerjaan, memperkirakan risiko, memperkirakan sumber daya yang diperlukan untuk menyelesaikan pekerjaan, pengorganisasian pekerjaan, mengelola sumber daya manusia dan material, menetapkan tugas, kegiatan mengarahkan dan mengendalikan proyek, melaporkan kemajuan dan menganalisis hasil.

Proyek perangkat lunak dilakukan untuk mencapai tujuan tertentu, yang diklasifikasikan ke dalam dua kategori, yaitu: tujuan proyek dan tujuan bisnis. Untuk tujuan proyek yang biasanya harus memenuhi hal-hal berikut ini:

1. Memenuhi persyaratan pengguna. Kembangkan proyek sesuai dengan kebutuhan pengguna setelah memahaminya.

2. Memenuhi tenggat waktu yang terjadwal. Selesaikan pilar utama proyek seperti yang dijelaskan dalam rencana proyek tepat waktu untuk menyelesaikan proyek sesuai dengan jadwal.
3. Kesesuaian anggaran. Kelola keseluruhan biaya proyek sehingga proyek berada dalam anggaran yang sudah dialokasikan.
4. Menghasilkan kualitas yang diinginkan. Pastikan kualitas dapat dicapai melalui proses yang akurat dan kinerja positif keseluruhan proyek.

Tujuan bisnis memastikan bahwa tujuan dan persyaratan organisasi dipenuhi dalam proyek. Secara umum, tujuan ini terkait dengan peningkatan proses bisnis, kepuasan pelanggan, dan peningkatan kualitas. Tujuan bisnis yang umum diikuti seperti hal-hal di bawah ini:

1. Mengevaluasi proses. Mengevaluasi proses bisnis dan membuat perubahan kapan dan di mana diperlukan saat proyek berlangsung.
2. Memperbaharui kebijakan dan proses. Memberikan fleksibilitas untuk memperbarui kebijakan dan proses organisasi untuk melakukan tugas secara efektif.
3. Pertahankan proyek sesuai jadwal. Mengurangi *downtime* (periode ketika tidak ada pekerjaan yang dilakukan) faktor-faktor seperti tidak tersedianya sumber daya selama pengembangan perangkat lunak.
4. Tingkatkan kualitas perangkat lunak. Gunakan proses yang sesuai untuk mengembangkan perangkat lunak yang memenuhi persyaratan organisasi dan memberikan keunggulan kompetitif bagi organisasi.

Proyek Perangkat Lunak

Proyek perangkat lunak merupakan suatu prosedur lengkap tentang pengembangan perangkat lunak, sejak dari pengumpulan data kebutuhan/persyaratan hingga pengujian dan maintenance.

Pentingnya Manajemen Proyek Perangkat Lunak

Perangkat lunak merupakan produk tidak berwujud (*intangible*). Pengembangan perangkat lunak adalah sejenis aliran baru di dalam dunia

bisnis dan masih sangat minim sumber daya manusia yang berpengalaman dalam membangun produk perangkat lunak. Hampir keseluruhan produk perangkat lunak yang dibuat harus menganut prinsip *'tailor made'* agar dapat memenuhi kebutuhan dan persyaratan dari pengguna. Hal yang paling penting adalah kondisi perkembangan teknologi yang berubah secara cepat dan setiap perangkat lunak bersifat unik sehingga tidak dapat diaplikasikan untuk sistem yang lain. Semua hambatan bisnis dan lingkungan seperti itu membawa risiko dalam pengembangan perangkat lunak sehingga penting dilakukan pengelolaan proyek perangkat lunak secara efisien.



Gambar 2.1 : Kendala dalam proyek perangkat lunak

Gambar 2.1 menunjukkan tiga kendala yang ada dalam proyek perangkat lunak. Hal ini adalah bagian penting dari organisasi perangkat lunak untuk menghasilkan produk berkualitas, menjaga biaya dalam batasan anggaran/kemampuan klien dan menyelesaikan proyek sesuai jadwal. Ada beberapa faktor, baik internal maupun eksternal, yang dapat berdampak pada segitiga tiga kendala ini. Salah satu dari ketiga faktor tersebut dapat berdampak buruk pada dua lainnya.

Oleh karena itu, manajemen proyek perangkat lunak sangat penting untuk memasukkan kebutuhan pengguna bersama dengan batasan anggaran dan waktu.

Manajer Proyek Perangkat Lunak

Seorang manajer proyek perangkat lunak adalah orang yang bertanggung jawab melaksanakan proyek perangkat lunak. Manajer proyek perangkat lunak sangat memahami semua fase SDLC (*Software Development Life Cycle*) yang akan dilalui dalam mengembangkan perangkat lunak. Manajer proyek mungkin tidak pernah terlibat langsung dalam menghasilkan produk akhir berupa perangkat lunak, tetapi ia mengendalikan dan mengelola kegiatan yang dilaksanakan dalam produksi secara keseluruhan.

Seorang manajer proyek memantau secara langsung proses pengembangan, menyiapkan dan melaksanakan berbagai rencana, mengatur sumber daya yang diperlukan dan memadai, mempertahankan komunikasi di antara semua anggota tim untuk mengatasi masalah biaya, anggaran, sumber daya, waktu, kualitas dan kepuasan pelanggan.

Berikut ini adalah tanggung jawab yang dimiliki oleh seorang manajer proyek perangkat lunak:

- a. Mengelola Manusia, meliputi:
 - Bertindak sebagai leader proyek
 - Mengkomunikasikan pekerjaan dengan pihak-pihak terkait
 - Mengelola sumber daya manusia
 - Menyusun hirarki pelaporan
- b. Mengelola Proyek, meliputi:
 - Mendefinisikan dan menetapkan ruang lingkup proyek
 - Mengelola kegiatan proyek
 - Memantau kemajuan dan kinerja proyek
 - Menganalisis resiko setiap tahapan
 - Mengambil langkah penting untuk menghindari resiko atau mengatasi masalah
 - Berlaku sebagai juru bicara proyek

Aktifitas Manajemen Proyek Perangkat Lunak

Manajemen proyek perangkat lunak terdiri dari sejumlah kegiatan, yang berisi perencanaan proyek, menentukan ruang lingkup produk perangkat lunak, menyusun perkiraan biaya, penjadwalan tugas dan kegiatan, dan manajemen sumber daya. Kegiatan manajemen proyek meliputi:

A. Perencanaan Proyek (*Project Planning*)

Proses perencanaan proyek melibatkan serangkaian kegiatan yang saling terkait diikuti secara teratur untuk mengimplementasikan persyaratan pengguna dalam perangkat lunak dan termasuk deskripsi serangkaian kegiatan perencanaan proyek dan individu yang bertanggung jawab untuk melakukan kegiatan ini. Selain itu, dalam proses perencanaan proyek terdapat hal-hal berikut ini:

- Tujuan dan ruang lingkup proyek
- Teknik yang digunakan untuk melakukan perencanaan proyek
- Upaya (dalam satuan waktu) individu yang terlibat dalam proyek
- Jadwal proyek dan *milestone*
- Sumber daya yang dibutuhkan untuk proyek
- Risiko yang terkait dengan proyek.

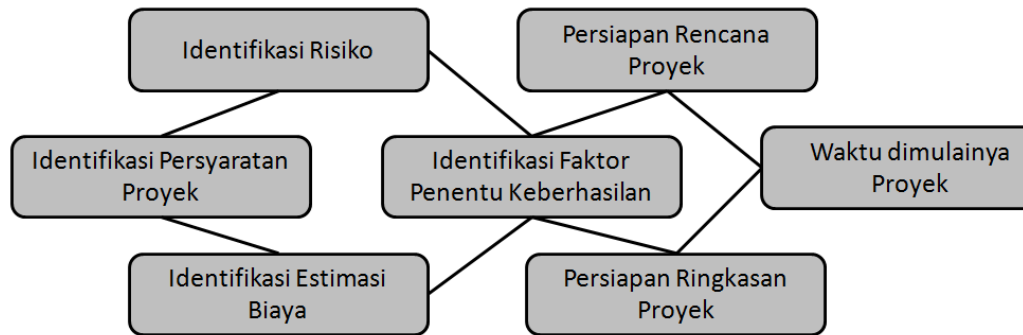
Proses perencanaan proyek terdiri dari beberapa kegiatan, yang penting untuk melaksanakan proyek secara sistematis. Kegiatan-kegiatan ini merujuk pada serangkaian tugas yang dilakukan selama periode waktu tertentu untuk mengembangkan perangkat lunak. Kegiatan-kegiatan ini meliputi estimasi waktu, upaya, dan sumber daya yang diperlukan dan risiko yang terkait dengan proyek.

Proses perencanaan proyek terdiri dari kegiatan-kegiatan berikut ini:

- A. Identifikasi persyaratan proyek. Sebelum memulai proyek, penting untuk melakukan identifikasi persyaratan proyek karena ia dapat membantu dalam melakukan kegiatan secara sistematis. Persyaratan ini terdiri dari informasi seperti ruang lingkup proyek, data dan

fungsi yang diperlukan dalam perangkat lunak, dan peran anggota tim manajemen proyek.

- B. Identifikasi estimasi biaya. Seiring dengan estimasi usaha dan waktu, perlu upaya untuk memperkirakan biaya yang harus dikeluarkan untuk suatu proyek. Estimasi biaya termasuk biaya perangkat keras, koneksi jaringan, dan biaya yang diperlukan untuk pemeliharaan komponen perangkat keras. Selain itu, biaya diperkirakan juga untuk individu yang terlibat dalam proyek.
- C. Identifikasi risiko. Risiko adalah peristiwa tak terduga yang memiliki efek buruk pada proyek. Proyek perangkat lunak melibatkan beberapa risiko (seperti risiko teknis dan risiko bisnis) yang mempengaruhi jadwal proyek dan meningkatkan biaya proyek. Identifikasi risiko sebelum proyek dimulai membantu dalam memahami kemungkinan dampaknya terhadap proyek.
- D. Identifikasi faktor penentu keberhasilan. Untuk membuat proyek berhasil, faktor penentu keberhasilan diikuti. Faktor-faktor ini merujuk pada kondisi yang memastikan peluang keberhasilan proyek yang lebih besar. Secara umum, faktor-faktor ini termasuk dukungan dari manajemen, anggaran yang sesuai, jadwal yang tepat, dan ahli perangkat lunak yang terampil.
- E. Persiapan ringkasan proyek. Rangkuman proyek memberikan gambaran singkat tentang ruang lingkup proyek, kualitas, waktu, biaya, dan kendala sumber daya seperti yang dijelaskan selama perencanaan proyek. Ini disiapkan oleh manajemen untuk mendapatkan persetujuan dari sponsor proyek.
- F. Persiapan rencana proyek. Rencana proyek memberikan informasi tentang sumber daya yang tersedia untuk proyek, individu yang terlibat dalam proyek, dan jadwal sesuai dengan mana proyek akan dilaksanakan.
- G. Waktu dimulainya proyek. Setelah perencanaan proyek selesai dan sumber daya ditugaskan untuk anggota tim, proyek perangkat lunak dimulai.



Gambar 2.2: Aktifitas perencanaan proyek

Setelah tujuan proyek dan tujuan bisnis ditentukan, tanggal akhir proyek pun ditetapkan. Tim manajemen proyek menyiapkan rencana dan jadwal proyek sesuai dengan tanggal berakhirnya proyek. Setelah menganalisis rencana proyek, manajer proyek mengomunikasikan rencana proyek dan tanggal berakhirnya kepada manajemen senior. Kemajuan proyek dilaporkan kepada manajemen dari waktu ke waktu. Demikian pula, ketika proyek selesai, manajemen senior diberitahu tentang itu. Dalam hal keterlambatan dalam menyelesaikan proyek, rencana proyek dianalisis ulang dan tindakan korektif diambil untuk menyelesaikan proyek. Proyek dilacak secara teratur dan ketika rencana proyek diubah, manajemen senior diberitahu.

B. Menentukan Ruang Lingkup

Ini adalah kegiatan mendefinisikan ruang lingkup proyek, termasuk semua kegiatan yang ada di dalamnya, dan proses yang perlu dilakukan untuk membuat produk perangkat lunak. Manajemen terhadap lingkup ini sangat penting karena menciptakan batasan-batasan proyek dengan mendefinisikan secara jelas apa yang akan dilakukan dalam proyek dan apa yang tidak akan dilakukan. Hal ini membuat proyek mengandung tugas-tugas terbatas dan terukur, yang dapat didokumentasikan dengan mudah dan pada gilirannya menghindari biaya dan waktu yang berlebihan.

Hal yang perlu dilakukan selama manajemen Ruang Lingkup Proyek adalah:

- Mendefenisikan ruang lingkup
- Menentukan verifikasi dan kontrol
- Membagi kegiatan menjadi kegiatan yang lebih kecil agar mudah mengelolanya
- Memverifikasi ruang lingkup
- Mengendalikan ruang lingkup apabila mengalami perubahan.

C. Estimasi Proyek

Untuk pengelolaan yang efektif, perkiraan akurat berbagai tindakan adalah suatu keharusan. Dengan estimasi yang benar, manajer dapat mengelola dan mengendalikan proyek dengan lebih efisien dan efektif.

Estimasi proyek meliputi hal-hal sebagai berikut:

a. Estimasi Ukuran (size) Perangkat Lunak

Ukuran perangkat lunak dapat diperkirakan baik dalam hal KLOC (*Kilo Line of Code*) atau dengan menghitung jumlah fungsi dalam perangkat lunak. Baris kode (*line of code*) bergantung pada aktifitas pengkodean. Jumlah fungsi bervariasi sesuai dengan kebutuhan pengguna atau perangkat lunak.

b. Estimasi usaha (*efforts*)

Manajer proyek memperkirakan usaha (*efforts*) dalam hal kebutuhan personil dan jam kerja yang diperlukan untuk menghasilkan perangkat lunak. Untuk suatu ukuran perangkat, estimasi usaha harus diketahui. Hal ini dapat diturunkan berdasarkan pengalaman manajer, data historis organisasi, atau ukuran perangkat lunak dapat diubah menjadi upaya dengan menggunakan beberapa rumus standar.

c. Estimasi Waktu

Setelah ukuran dan usaha diperkirakan, waktu yang diperlukan untuk menghasilkan perangkat lunak juga dapat diperkirakan. Usaha yang dibutuhkan dipisahkan ke dalam sub kategori sesuai

spesifikasi kebutuhan dan interdependensi dari berbagai komponen perangkat lunak. Tugas pengerjaan perangkat lunak dibagi menjadi tugas-tugas yang lebih kecil, kegiatan atau aktifitas oleh *Work Breakthrough Structure* (WBS). Tugas dijadwalkan dari hari ke hari atau dalam bulan kalender. Jumlah waktu yang diperlukan untuk menyelesaikan semua tugas dalam beberapa jam atau hari adalah total waktu yang diinvestasikan untuk menyelesaikan proyek.

d. Estimasi Biaya

Bagian ini mungkin dianggap sebagai yang paling sulit karena tergantung pada lebih banyak elemen daripada yang sebelumnya. Untuk memperkirakan biaya proyek, perlu dipertimbangkan hal-hal sebagai berikut:

- Ukuran dari perangkat lunak
- Kualitas perangkat lunak
- Perangkat keras
- Perangkat lunak tambahan (tool), lisensi dan sebagainya
- Tenaga ahli dengan keahlian yang spesifik
- Perjalanan yang dilakukan
- Komunikasi
- Pelatihan dan bantuan teknis.

Tool Manajemen Proyek Perangkat Lunak

Resiko dan ketidakpastian dapat mengalami peningkatan tergantung pada volume proyek, meskipun proyek dilaksanakan sesuai dengan metodologi yang ditetapkan.

Berikut ini adalah contoh tool-tool yang digunakan untuk dapat membantu mengelola proyek secara efektif.

1. Gantt Chart

Gantt Chart adalah sejenis grafik batang (*Bar Chart*) yang digunakan untuk menunjukkan aktivitas-aktivitas pada proyek serta jadwal dan waktu pelaksanaannya, seperti waktu dimulainya tugas tersebut dan

juga batas waktu yang digunakan untuk menyelesaikan tugas yang bersangkutan. Personil atau bagian yang ditugaskan untuk menyelesaikan tugas dalam proyek juga harus dituliskan dalam *Gantt Chart*.

Beberapa sebutan lain untuk *Gantt Chart* diantaranya adalah *Milestones Chart*, *Project Bar Chart* dan juga *Activity chart*. *Gantt Chart* yang dikembangkan oleh Henry Laurence Gantt pada tahun 1910 ini pada dasarnya adalah suatu gambaran atas perencanaan, penjadwalan dan pemantauan (monitoring) kemajuan setiap kegiatan atau aktivitas pada suatu proyek.

Gantt Chart merupakan salah satu alat yang sangat bermanfaat dalam merencanakan penjadwalan dan memantau kegiatan pada suatu proyek, mengkomunikasikan kegiatan-kegiatan yang harus dilaksanakan dan juga status pelaksanaannya. Dalam *Gantt Chart* juga dapat dilihat urutan kegiatan ataupun tugas yang harus dilakukan berdasarkan prioritas waktu yang ditentukan.

Cara membuat *Gantt Chart*:

1. Mengidentifikasi tugas

- Mengidentifikasi tugas-tugas yang perlu diselesaikan pada proyek tersebut
- Menentukan *milestone* (bagian pekerjaan dari suatu tugas) dengan menggunakan *brainstorming* ataupun *flowchart*.
- Mengidentifikasi durasi waktu yang diperlukan dalam menyelesaikan suatu tugas.
- Mengidentifikasi urutan pekerjaan ataupun tugas yang akan dikerjakan. Seperti tugas yang harus diselesaikan sebelum memulai suatu tugas yang baru ataupun tugas-tugas apa yang harus dilakukan secara bersamaan (paralel).

2. Menggambar sumbu horizontal

Gambarkan sumbu horizontal untuk waktu pelaksanaannya (dapat diletakan di atas atau di bawah halaman). Tandai dengan skala

waktu yang sesuai (dapat ditulis dalam satuan harian maupun mingguan).

3. Menuliskan tugas atau aktivitas

Tuliskan tugas atau bagian pekerjaan (*milestone*) yang akan dikerjakan berdasarkan urutan waktu pada bagian kiri. Gambarkan diagram batang (*Bar Graph*) untuk menunjukkan rentang waktu yang diperlukan untuk melakukan tugas yang bersangkutan. Gambarkan kotak dari kiri di mana waktu tugas tersebut dimulai sampai pada waktu tugas yang bersangkutan berakhir. Jika diperlukan presentasi kepada pihak klien, gambarkan bentuk intan (*Diamond*) pada tanggalnya. Gambarkan tepinya saja dan kotak tersebut jangan diisi.

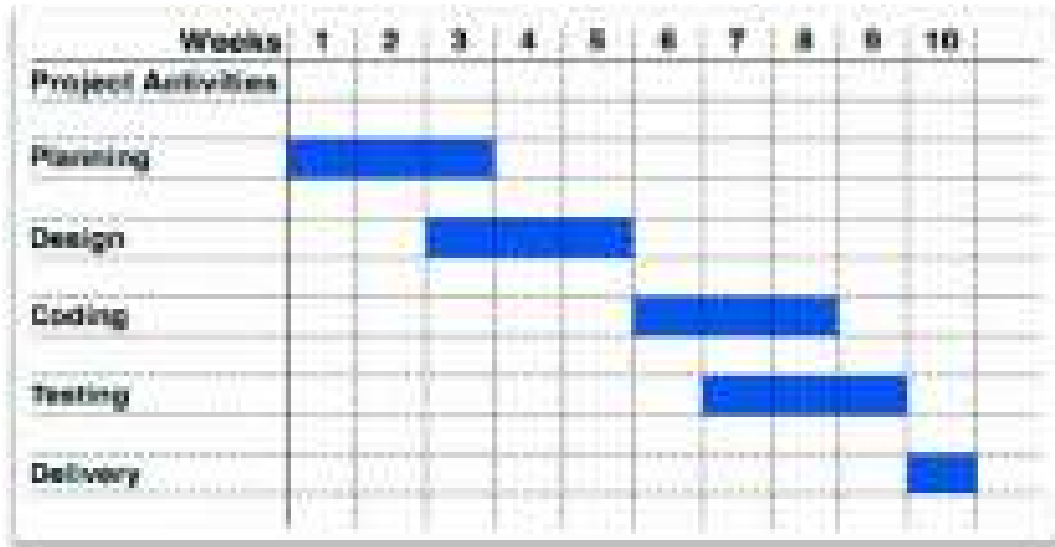
4. Melakukan pemeriksaan kembali

Lakukan pemeriksaan kembali, apakah semua tugas atau bagian pekerjaan untuk proyek tersebut sudah tertulis semuanya ke dalam *Gantt Chart*.

Cara Menggunakannya:

1. Saat Proyek sedang berlangsung, isikan gambar Intan (*diamond*) ataupun Grafik Batang pada *Gantt Chart* untuk menunjukkan bahwa tugas yang bersangkutan telah diselesaikan. Jika ada tugas masih berjalan (*in progress*), estimasikan kemajuan tugas yang bersangkutan dan isikan grafik batang sesuai dengan kemajuan tersebut.
2. Letakkan tanda vertikal untuk menunjukkan sejauh mana proyek tersebut sedang berlangsung.

Contoh:



Gambar 2.3: Gantt Chart

2. PERT (*Project Evaluation and Review Technique*)

Kompleksitas sebuah pengelolaan proyek, membutuhkan identifikasi dan pemetaan atas rangkaian kegiatan yang bisa saja harus dilakukan secara serial (berurutan) atau dapat dilakukan secara paralel. Pemetaan ini dapat disusun dalam bentuk model jaringan. Critical Path Method (CPM) dikembangkan pada tahun 1957 sebagai model jaringan untuk pemetaan alur sebuah proyek. CPM adalah metode perancangan alur proyek yang menggunakan perkiraan waktu tetap untuk setiap kegiatannya.

Walau mudah dimengerti dan digunakan, CPM tidak mempertimbangkan variasi waktu yang mungkin saja dapat terjadi dan dapat memiliki dampak yang besar terhadap target waktu penyelesaian sebuah proyek.

Program Evaluation and Review Technique (PERT) adalah suatu model jaringan yang mampu memetakan waktu penyelesaian kegiatan yang acak. Proses perencanaan dengan menggunakan PERT meliputi langkah-langkah seperti berikut:

- [1]. Mengidentifikasi kegiatan (activities) dan tonggak proyek (milestones) yang spesifik.

Dalam pengelolaan suatu proyek, sebuah aktivitas adalah kegiatan yang harus dikerjakan dan sebuah 'event' atau peristiwa merupakan tahapan penyelesaian dari satu atau lebih kegiatan. Keluaran dari tahapan ini adalah daftar tugas dalam tabel yang mencakup informasi tentang urutan dan durasi.

- [2]. Menentukan urutan yang tepat dari kegiatan-kegiatan.

Langkah ini membutuhkan analisa yang cukup mendalam mengenai relasi antara setiap kegiatan. Sebelum sebuah kegiatan dapat dimulai, semua kegiatan yang menjadi prasyarat bagi kegiatan tersebut harus sudah terlaksana (*terminated*).

- [3]. Menyusun model diagram jaringan.

Menggunakan informasi urutan aktivitas, diagram PERT dapat disusun dengan menunjukkan sifat urutan kegiatan (serial dan paralel). Beberapa draft dapat saja diperlukan untuk dapat secara benar menggambarkan hubungan antar aktivitas.

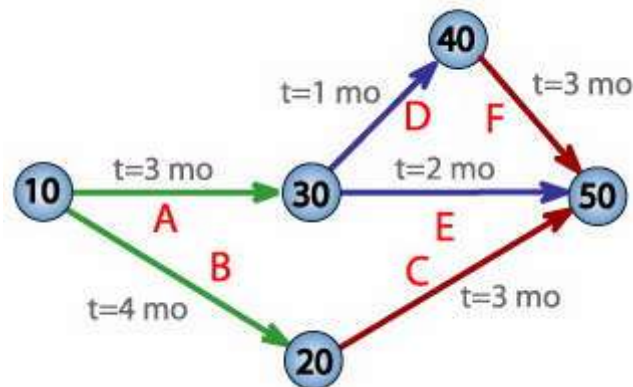
- [4]. Memperkirakan waktu yang diperlukan untuk masing-masing kegiatan.

Hari, minggu atau bulan adalah unit umum biasa digunakan waktu untuk penyelesaian kegiatan.

Sebuah fitur yang membedakan PERT adalah kemampuannya untuk menghadapi ketidakpastian di masa penyelesaian kegiatan. Untuk setiap aktivitas, model biasanya mencakup tiga perkiraan waktu: **Waktu Optimis**, yaitu perkiraan waktu yang paling singkat bagi penyelesaian aktivitas; **Waktu Perkiraan Paling Mungkin**, waktu penyelesaian yang memiliki probabilitas tertinggi (berbeda dengan : waktu yang diharapkan); dan **Waktu Pesimis**, yaitu waktu terpanjang yang mungkin diperlukan suatu kegiatan.

Waktu Rata-rata atau waktu yang diharapkan dan dapat ditampilkan dalam diagram dapat dihitung dari rumus =

$(\text{Waktu Optimis} + 4 \text{ Waktu Perkiraan Paling Mungkin} + \text{Waktu Pesimis}) / 6$



Gambar 2.4: Contoh PERT

- [5]. Menentukan tahapan dan jalur kritis.

Jalur kritis ditentukan dengan menjumlahkan waktu setiap kegiatan, mulai dari awal hingga akhir proyek. Jumlah terpanjang dari sebuah variasi urutan kegiatan merupakan jalur kritis. Dari contoh di atas maka alur A – D – F = 3 + 1 + 3 = 7 mo dan alur B – C = 4 + 3 = 7 mo, merupakan jalur kritis (*critical path*).

Sedangkan alur A – E = 3 + 2 = 5 mo merupakan jalur non-kritis. Dari analisa di atas, maka kegiatan E dapat ditunda tanpa maksimal 2 mo tanpa menunda penyelesaian keseluruhan proyek ini. Kegiatan E disebut memiliki waktu longgar (*slack time*).

- [6]. Melakukan pemantauan dan evaluasi serta koreksi pada diagram PERT selama proyek berlangsung.

Dalam dinamika pengelolaan proyek, secara berkala diagram PERT dapat dipantau, serta dikoreksi sesuai dengan perkembangan pelaksanaan proyek dengan memasukkan angka waktu yang telah terjadi pada setiap kegiatan yang sudah berlalu. Atau malah diagram dikoreksi untuk rencana kegiatan yang akan datang disebabkan perubahan asumsi selama proyek berlangsung.

Soal

1. Suatu proyek dilaksanakan dalam kondisi yang terbatas. Jelaskan elemen-elemen yang termasuk dalam kondisi tersebut.
2. Jelaskan aktifitas apa saja yang ada dalam manajemen proyek perangkat lunak.
3. Jelaskan tugas-tugas dari seorang manajer proyek perangkat lunak.
4. Buatlah contoh implementasi dari tool *Gantt Chart* dengan menggunakan MS Visio. Hasilnya dipresentasikan secara berkelompok.
5. Buatlah contoh implementasi dari tool *PERT*. Hasilnya dipresentasikan secara berkelompok.

BAB III

MODEL-MODEL PROSES PERANGKAT LUNAK

Tujuan :

1. Mempelajari ragam model proses perangkat lunak
2. Mengenalkan mekanisme dalam masing-masing model dalam proses perangkat lunak

Indikator Keberhasilan:

1. Agar mahasiswa memiliki kemampuan untuk menjelaskan fase-fase dalam merekayasa perangkat lunak dan kapan metode-metode tersebut harus digunakan.
2. Agar mahasiswa memiliki kemampuan membedakan masing-masing model proses perangkat lunak dan mengaplikasikannya pada suatu kasus aplikasi perangkat lunak yang sederhana.

Materi :

Pandangan umum tentang Rekayasa Perangkat Lunak

Segala aktifitas yang berkaitan dengan Rekayasa Perangkat Lunak dapat diklasifikasikan ke dalam tiga fase umum, tanpa memperhatikan lingkup operasional, skala proyek atau tingkat kompleksitasnya. Masing-masing fase akan memberi penekanan pada pencarian informasi-informasi yang sudah ditulis sedemikian rupa. Fase-fase tersebut antara lain :

- A. Fase Definisi (*Definition Phase*) berfokus pada “apa” (*what*), di mana pada fase ini pengembang perangkat lunak harus mengidentifikasi informasi apa yang akan diproses, fungsi dan unjuk kerja apa yang diperlukan, tingkah laku sistem seperti apa

yang diharapkan, interface apa yang akan dibangun, batasan desain apa yang ada dan kriteria validasi apa yang dibutuhkan untuk mendefinisikan sistem yang sukses. Kebutuhan (*requirement*) kunci dari sistem dan perangkat lunak yang didefinisikan. Metode yang diaplikasikan selama fase definisi berbeda, tergantung pada paradigma rekayasa perangkat lunak (atau kombinasi paradigma) yang diaplikasikan.

- B. Fase Pengembangan (*Development Phase*) berfokus pada *how* (bagaimana), yaitu di mana selama masa pengembangan perangkat lunak, teknisi harus mendefinisikan bagaimana data dikonstruksikan, bagaimana detail prosedur akan diimplementasikan, bagaimana interface dikembangkan, dan sebagainya.
- C. Fase Pemeliharaan (*Maintenance Phase*) berfokus pada perubahan yang dihubungkan dengan koreksi kesalahan, penyesuaian yang dibutuhkan ketika lingkungan perangkat lunak berkembang, serta perubahan sehubungan dengan perkembangan yang disebabkan oleh perubahan kebutuhan dari pengguna.

Model Proses Perangkat Lunak

Dalam rangka menyelesaikan permasalahan yang nyata dalam suatu kasus, tim pelaksana rekayasa perangkat lunak harus mengkombinasikan strategi pengembangan yang mencakup lapisan proses, metode dan alat-alat bantu (*tools*) serta fase-fase generik. Strategi yang dimaksud, sering diacukan sebagai model proses atau paradigma rekayasa perangkat lunak. Model proses untuk rekayasa perangkat lunak dipilih berdasarkan sifat aplikasi dan proyeknya, metode dan alat-alat bantu yang akan dipakai.

Pada bab ini selanjutnya akan dibahas bermacam-macam model proses yang berbeda pada proses pengembangan perangkat lunak. Penting untuk diingat bahwa masing-masing model sudah ditandai dengan cara tertentu sehingga diharapkan dapat membantu di dalam kontrol dan koordinasi dari

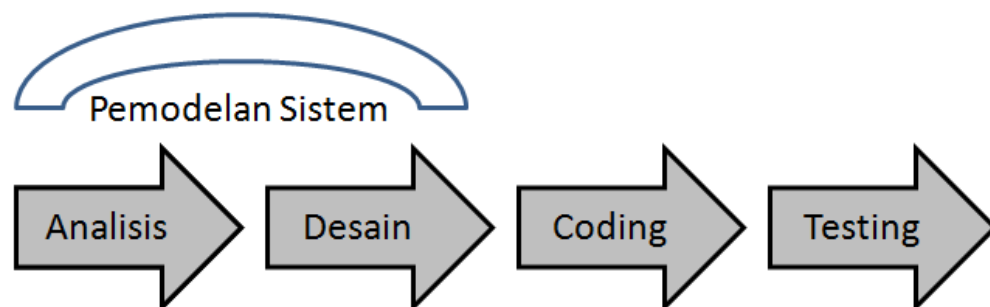
proyek perangkat lunak yang nyata. Dengan demikian, pada intinya semua model menunjukkan karakteristiknya yang secara umum adalah berbeda antara satu dengan yang lainnya.

A. MODEL SEKUENSIAL LINIER (*WATERFALL*)

Model *Waterfall* yang sering juga dikenal sebagai model air terjun adalah model proses pertama yang diperkenalkan. Ia sangat mudah dimengerti dan digunakan. Dalam model *Waterfall*, setiap fase harus diselesaikan sebelum fase berikutnya dapat dimulai dan tidak ada fase yang tumpang tindih. Model *Waterfall* adalah pendekatan SDLC paling awal yang digunakan untuk pengembangan perangkat lunak.

Pada pendekatan *Waterfall*, seluruh proses pengembangan perangkat lunak dibagi menjadi fase-fase terpisah. Hasil dari satu fase bertindak sebagai input untuk fase berikutnya secara berurutan. Ini berarti bahwa setiap fase dalam proses pengembangan dimulai hanya jika fase sebelumnya selesai. Model *Waterfall* adalah proses desain berurutan di mana kemajuan kegiatan dilihat sebagai bentuk aliran dari atas terus ke bawah (seperti air terjun) melalui beberapa fase.

Gambar berikut menggambarkan sekuensial linier untuk rekayasa perangkat lunak, yang sering disebut dengan siklus kehidupan klasik atau model air terjun.



Gambar 3.1 : Model Sekuensial Linier (*Waterfall Model*)

Sekuensial linier menggunakan sebuah pendekatan kepada pengembangan perangkat lunak yang sistematis dan sekuensial yang mulai pada tingkat dan kemajuan sistem pada seluruh analisis, desain, kode, pengujian (tes), dan pemeliharaan. Berikut ini penjelasan yang dapat diberikan untuk masing-masing tahap :

a) Analisis kebutuhan perangkat lunak

Analisis kebutuhan merupakan langkah awal untuk menentukan gambaran perangkat lunak yang akan dihasilkan ketika pengembang melaksanakan sebuah proyek pembuatan perangkat lunak. Perangkat lunak yang baik dan sesuai dengan kebutuhan pengguna sangat tergantung pada keberhasilan dalam melakukan analisis kebutuhan. Untuk proyek-proyek perangkat lunak yang besar, analisis kebutuhan dilaksanakan setelah aktivitas sistem information engineering dan software project planning.

Analisa kebutuhan yang baik belum tentu menghasilkan perangkat lunak yang baik, tetapi analisa kebutuhan yang tidak tepat menghasilkan perangkat yang tidak berguna. Keberhasilan mengetahui adanya kesalahan pada analisis kebutuhan pada tahap awal memang jauh lebih baik, tapi kesalahan analisis kebutuhan yang diketahui ketika sudah memasuki penulisan kode atau pengujian, bahkan hampir masuk dalam tahap penyelesaian merupakan kesalahan besar bagi pengembang perangkat lunak. Biaya dan waktu yang diperlukan akan menjadi sia sia.

Agar proses pengumpulan data kebutuhan berlangsung dengan baik, perekrayasa perangkat lunak (analisis) harus memahami persis domain permasalahan (*problem domain*), tingkah laku, unjuk kerja dan antarmuka (*interface*) yang diperlukan. Kebutuhan yang terkait dengan sistem maupun perangkat lunak didokumentasikan dan dilihat lagi oleh klien.

b) Desain

Desain perangkat lunak sebenarnya adalah proses multi langkah yang berfokus pada empat atribut sebuah program yang berbeda (struktur data, arsitektur perangkat lunak, representasi interface, dan detail (algoritma) prosedural. Proses desain menerjemahkan syarat/kebutuhan ke dalam sebuah representasi perangkat lunak yang dapat diperkirakan demi kualitas sebelum dimulai pemunculan kode (*coding*). Sebagaimana analisis, desain ini juga didokumentasikan.

c) Menghasilkan kode

Desain yang telah dihasilkan harus diterjemahkan ke dalam bentuk bahasa mesin yang dapat dibaca oleh perangkat keras. Langkah pembuatan kode meliputi pekerjaan dalam tahap ini, dan dapat dilakukan secara mekanis.

d) Pengujian (*Testing*)

Sekali kode dibuat, pengujian program sudah dapat dimulai. Proses pengujian berfokus pada logika internal perangkat lunak, 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.

e) Pemeliharaan (*Maintenance*)

Perangkat lunak akan mengalami perubahan dan penyesuaian setelah disampaikan kepada pelanggan. Perubahan akan terjadi karena kesalahan-kesalahan karena perangkat lunak harus disesuaikan untuk mengakomodasi perubahan-perubahan di dalam lingkungan eksternalnya atau karena pelanggan membutuhkan pengembangan aspek fungsional atau unjuk kerja. Pemeliharaan perangkat lunak

mengaplikasikan lagi setiap fase program sebelumnya dan tidak membuatnya dari awal lagi.

Tabel 3.1: Kelebihan dan Kekurangan Model Waterfall

Kelebihan	Kekurangan
• Menghindari masalah yang mengakibatkan pendekatan berbasis risiko dalam perangkat lunak • Menentukan mekanisme kegiatan jaminan kualitas perangkat lunak • Digunakan oleh proyek yang kompleks dan dinamis • Evaluasi ulang setelah setiap langkah memungkinkan perubahan dalam perspektif pengguna, kemajuan teknologi, atau perspektif keuangan. • Estimasi anggaran dan jadwal menjadi realistis saat pekerjaan berlangsung.	• Penilaian risiko proyek dan penyelesaiannya bukanlah tugas yang mudah. • Sulit memperkirakan anggaran dan jadwal di awal karena beberapa analisis tidak dilakukan sampai desain perangkat lunak dikembangkan.

B. MODEL PROTOTIPE

Prototipe adalah versi sistem atau bagian dari sistem yang dikembangkan dengan cepat untuk memeriksa persyaratan atau kelayakan dari beberapa keputusan desain yang diminta klien. Model prototipe diterapkan ketika informasi detail yang terkait dengan persyaratan input dan output dari sistem tidak tersedia. Dalam model ini, diasumsikan bahwa semua persyaratan mungkin tidak diketahui pada awal pengembangan sistem. Ini biasanya digunakan ketika sistem tidak ada atau dalam kasus sistem yang besar dan kompleks di mana tidak ada proses manual untuk menentukan persyaratan. Model ini memungkinkan pengguna untuk berinteraksi dan

bereksperimen dengan model kerja dari sistem yang dikenal sebagai prototipe. Prototipe memberi ruang kepada pengguna untuk merasakan kondisi sebenarnya dari sistem yang akan dikembangkan.

Jadi, prototipe berguna ketika klien atau pengembang tidak yakin dengan persyaratan, atau algoritma, efisiensi, aturan bisnis, waktu respons, dan sebagainya.

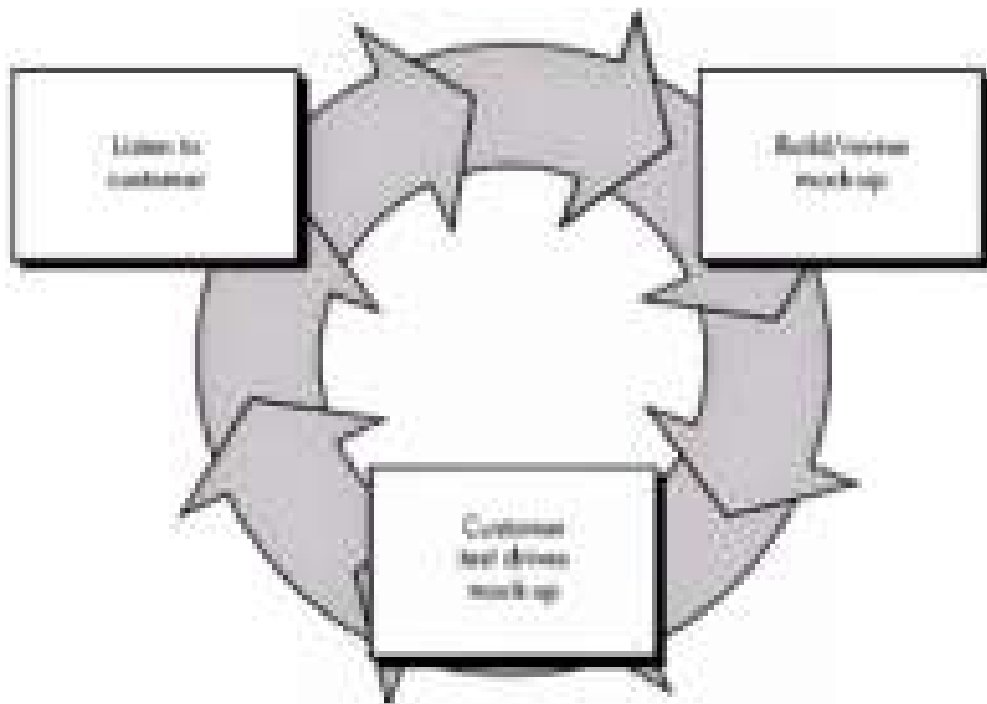
Dalam pembuatan prototipe, klien terlibat dalam proses pengembangan, yang meningkatkan kemungkinan penerimaan klien terhadap implementasi akhir perangkat lunak. Sementara beberapa prototipe dikembangkan dengan harapan bahwa ia akan dibuang, mungkin dalam beberapa kasus berevolusi dari prototipe ke sistem kerja.

Prototipe perangkat lunak dapat digunakan:

1. **Dalam rekayasa persyaratan**, prototipe dapat membantu dengan elisitasi dan validasi persyaratan sistem. Hal ini memungkinkan pengguna untuk bereksperimen dengan sistem, dan sebagainya, menyempurnakan persyaratan. Mereka mungkin mendapatkan ide-ide baru untuk persyaratan, dan menemukan area kekuatan dan kelemahan dalam perangkat lunak.
Lebih jauh lagi, ketika prototipe dikembangkan, ia mungkin mengungkapkan kesalahan dan dalam persyaratan. Spesifikasi mungkin kemudian dimodifikasi untuk mencerminkan perubahan.
2. **Dalam desain sistem**, prototipe dapat membantu untuk melakukan percobaan yang sesuai untuk memeriksa kelayakan desain yang diusulkan.

Model prototipe merupakan bentuk model sistem yang belum utuh menjadi sebuah hasil desain. Ia dibuat sebagai keperluan untuk berkomunikasi dengan calon pengguna, dan perancangan berfokus pada "*listen to customer*". Dengan demikian dalam proses pembuatan modelnya, antara pengembang dengan customer lebih banyak berkomunikasi (*feed back*)

terkait perancangannya. Fokusnya adalah agar pengembang lebih intensif berkomunikasi untuk memenuhi kebutuhan pengguna.



Gambar 3.2: Model Prototipe

Pada tahap pertama yaitu "*Listen to Customer*" yang merupakan proses komunikasi pengguna dengan pengembang yang dapat langsung diterapkan sesuai dengan keinginan pengguna. Selanjutnya masuk kepada tahap "*Build/Revise Mock-Up*" yaitu pembuatan pemodelan setengah jadi dan dilanjutkan ke tahap "*Customer Test Drives Mock-Up*" yang merupakan suatu kegiatan pengujian program yang dilakukan oleh customer. Apabila terdapat keinginan pengguna yang belum tercapai atau ada bagian yang ingin di tambahkan dari sistem program yang dikembangkan, maka aktifitas kembali dilanjutkan ke tahap semula "*Listen to Customer*".

Secara ideal prototipe berfungsi sebagai sebuah mekanisme untuk mengidentifikasi kebutuhan perangkat lunak. Prototipe dapat menjadi paradigma yang efektif bagi rekayasa perangkat lunak. Kuncinya adalah

mendefinisikan aturan-aturan main pada saat awal, yaitu pelanggan dan pengembang keduanya harus setuju bahwa prototype dibangun untuk berfungsi sebagai mekanisme pendefinisian kebutuhan. Prototype kemudian disingkirkan dan perangkat lunak actual direkayasa dengan tertuju kepada kualitas dan kemampuan pemeliharaan.

Tabel 3.2: Kelebihan dan Kekurangan Model Prototype

Kelebihan	Kekurangan
• Memberikan model yang dapat digunakan kepada pengguna di awal proses, memungkinkan penilaian awal dan meningkatkan kepercayaan pengguna. • Pengembang mendapatkan pengalaman dan wawasan dengan mengembangkan prototype di sana dengan menghasilkan implementasi persyaratan yang lebih baik. • Model prototyping berfungsi untuk memperjelas persyaratan, yang tidak jelas, sehingga mengurangi ambiguitas dan meningkatkan komunikasi antara pengembang dan pengguna. • Ada keterlibatan besar pengguna dalam pengembangan perangkat lunak. Oleh karena itu, persyaratan pengguna dipenuhi sejauh mungkin. • Membantu mengurangi risiko yang terkait dengan perangkat lunak.	• Jika pengguna tidak puas dengan prototype yang dikembangkan, maka prototype yang baru dapat dikembangkan. Proses ini berlangsung hingga prototype sempurna dikembangkan. Dengan demikian, model ini memakan waktu dan mahal. • Pengembang kehilangan fokus tujuan prototype yang sesungguhnya dan karenanya, dapat berkompromi dengan kualitas perangkat lunak. Misalnya, pengembang dapat menggunakan beberapa algoritma yang tidak efisien atau bahasa pemrograman yang tidak tepat saat mengembangkan prototype. • <i>Prototyping</i> dapat menyebabkan harapan yang salah bagi pengguna. Misalnya, sebuah situasi yang dibuat di mana pengguna percaya bahwa pengembangan sistem telah selesai padahal sesungguhnya belum. • Tujuan utama dari prototyping

	adalah pengembangan cepat, sehingga desain sistem dapat menyulitkan karena dikembangkan secara seri tanpa mempertimbangkan integrasi semua komponen lainnya.
--	--

C. MODEL RAD

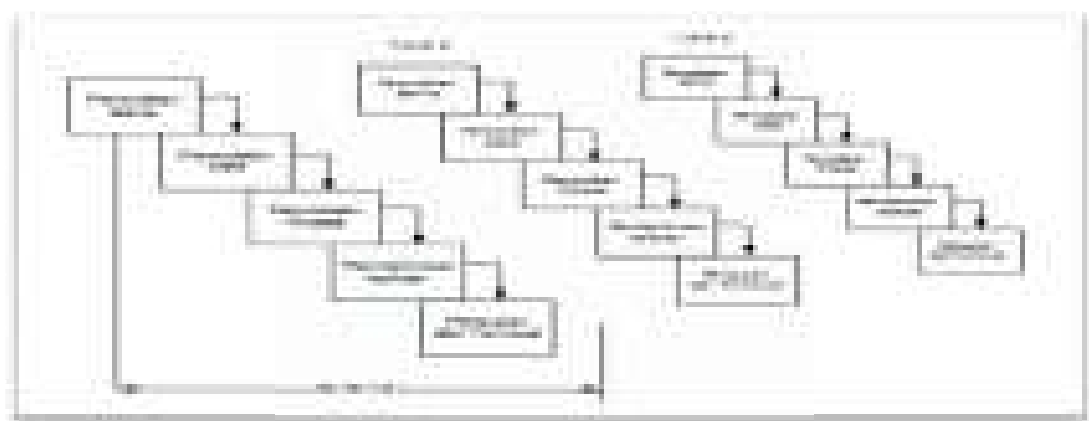
Rapid Application Development (RAD) adalah sebuah model proses perkembangan perangkat lunak sekuensial linier yang menekankan siklus perkembangan yang sangat pendek. Model RAD ini merupakan sebuah adaptasi “kecepatan tinggi” dari model sekuensial linier di mana perkembangan cepat dicapai dengan menggunakan 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 60-90 hari).

Model RAD menekankan pada penyelesaian proyek dalam jumlah kecil. Jika proyek besar, itu dibagi menjadi serangkaian proyek yang lebih kecil. Masing-masing proyek yang lebih kecil ini direncanakan dan di-*delivery* secara individual. Dengan demikian, dengan serangkaian proyek yang lebih kecil, tugas akhir disampaikan dengan cepat dan dengan cara yang kurang terstruktur. Karakteristik utama dari model RAD adalah bahwa ia berfokus pada penggunaan kembali kode, proses, template, dan alat.

Keberadaan model pengembangan RAD, klien dapat melihat demo produk akhir jauh lebih cepat. Selama pembuatan prototipe untuk produk apa pun, untuk menghemat waktu dan uang, penting untuk membuat satu versi yang dapat digunakan kembali untuk perubahan cepat. Pendekatan RAD melingkupi fase-fase sebagai berikut :

1. Pemodelan Bisnis. Aliran informasi diantara fungsi-fungsi bisnis dimodelkan dengan suatu cara untuk menjawab pertanyaan-

- pertanyaan berikut : Informasi apa yang mengendalikan proses bisnis? Informasi apa yang dimunculkan? Siapa yang memunculkannya? Kemana informasi itu pergi? Siapa yang memprosesnya?
2. **Pemodelan Data.** Aliran informasi yang didefinisikan sebagai bagian dari fase business modelling disaring kedalam serangkaian objek data yang dibutuhkan untuk mendukung bisnis tersebut.
 3. **Pemodelan Proses.** Aliran informasi yang didefinisikan di dalam fase data modelling ditransfirmasikan 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. **Pembentukan Aplikasi.** RAD mengasumsikan pemakaian teknik generasi keempat. Selain menciptakan perangkat lunak dengan menggunakan bahasa pemrograman generasi ketiga yang konvensional, RAD lebih banyak memproses kerja untuk memakai lagi komponen program yang ada atau menciptakan komponen yang dapat dipakai lagi.
 5. **Pengujian dan Turnover.** Karena proses RAD menekankan pada pemakaian kembali, banyak komponen program telah diuji. Hal ini mengurangi keseluruhan waktu pengujian. Tetapi komponen baru harus diuji dan semua interface harus dilatih secara penuh.



Gambar 3.3 : Model RAD (*Rapid Application Development*)

Model RAD jauh lebih efektif karena memberikan model secara langsung kepada pelanggan. Pelanggan dapat dengan cepat meninjau prototipe dan perubahan dapat lebih mudah dilakukan selama pengembangan produk akhir.

Tabel 3.3: Kelebihan dan Kekurangan Model RAD

Kelebihan	Kekurangan
1. Hasil kerja lebih mudah untuk ditransfer karena abstraksi tingkat tinggi, skrip, dan kode perantara digunakan. 2. Memberikan fleksibilitas yang lebih besar karena perancangan ulang dilakukan menurut versi pengembang. 3. Terjadinya pengurangan aktifitas pengkodean manual karena adanya kode generator dan penggunaan kembali kode (<i>reuse</i>). 4. Mendorong keterlibatan user. 5. Kemungkinan cacat yang lebih rendah karena prototipe bersifat natural.	1. Berguna hanya untuk proyek yang lebih besar 2. Proyek RAD gagal jika tidak ada komitmen oleh pengembang atau pengguna untuk menyelesaikan perangkat lunak tepat waktu. 3. Tidak tepat ketika tingginya risiko teknis. Ini terjadi ketika aplikasi baru menggunakan teknologi baru atau ketika perangkat lunak baru membutuhkan tingkat interoperabilitas yang tinggi dengan sistem yang ada. 4. Karena minat pengguna dan pengembang dapat menyimpang dari iterasi tunggal ke yang berikutnya, persyaratan mungkin tidak bertemu dalam model RAD.

D. MODEL SPIRAL

Model spiral (*spiral model*) yang pada awalnya diusulkan oleh Boehm adalah model proses perangkat lunak yang evolusioner yang merangkai sifat iteratif dari prototipe dengan cara kontrol dan aspek sistematis dari

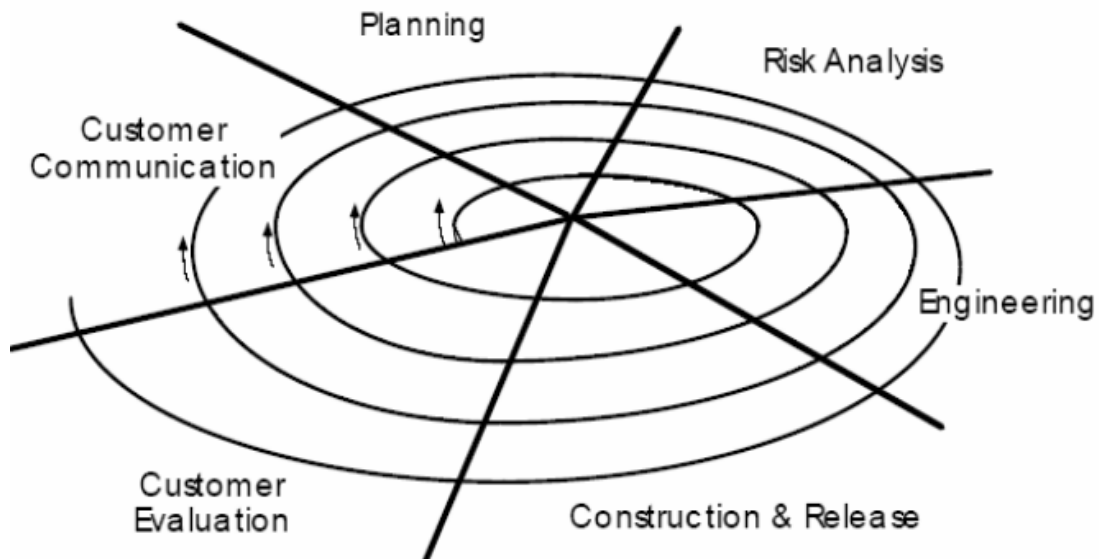
model sekuensial linier. Di dalam model spiral, perangkat lunak dikembangkan di dalam suatu deretan pertambahan.

Selama awal iterasi, hasil incremental dapat merupakan sebuah model atau prototipe pada lembaran 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 :

- Komunikasi pelanggan, tugas-tugas yang dibutuhkan untuk membangun komunikasi yang efektif diantara pengembang dan pelanggan.
- Perencanaan, tugas-tugas yang dibutuhkan untuk mendefinisikan sumber-sumber daya, ketepatan waktu, dan proyek informasi lain yang berhubungan.
- Analisis resiko, tugas-tugas yang dibutuhkan untuk menaksir resiko-resiko, baik manajemen maupun teknis.
- Perekayasaan, tugas-tugas yang dibutuhkan untuk membangun satu atau lebih representasi dari aplikasi tersebut.
- Konstruksi dan peluncuran, tugas-tugas yang dibutuhkan untuk mengkonstruksi, menguji, memasang (instal) dan memberikan pelayanan kepada pemakai (contohnya pelatihan dan dokumentasi).
- Evaluasi pelanggan, tugas-tugas yang dibutuhkan untuk memperoleh umpan balik dari pelanggan dengan didasarkan pada evaluasi representasi perangkat lunak, yang dibuat selama masa perekayasaan, dan diimplementasikan selama pemasangan.

Tujuan dari model spiral adalah menekankan kepada manajemen untuk mengevaluasi dan menyelesaikan risiko dalam proyek perangkat lunak. Berbagai aspek risiko dalam proyek perangkat lunak adalah penyimpangan proyek, perubahan persyaratan, kehilangan personil proyek utama, keterlambatan perangkat keras yang diperlukan,

persaingan dengan pengembang perangkat lunak lain dan terobosan teknologi yang membuat proyek menjadi usang.



Gambar 3.4 : Model Spiral

Model spiral menjadi sebuah pendekatan yang realistis bagi perkembangan sistem dan perangkat lunak skala besar. Karena perangkat lunak terus bekerja selama proses bergerak, pengembang dan pengguna memahami dan bereaksi lebih baik terhadap risiko dari setiap tingkat evolusi.

Model spiral menggunakan prototipe sebagai mekanisme pengurangan risiko. Tetapi yang lebih penting lagi, model spiral memungkinkan pengembang menggunakan pendekatan prototipe pada setiap keadaan di dalam evolusi produk. Model spiral menjaga pendekatan langkah demi langkah secara sistematis seperti yang diusulkan oleh siklus kehidupan klasik, tetapi memasukkannya ke dalam kerangka kerja iteratif yang secara realistis merefleksikan dunia nyata.

Soal

1. Uraikan dan jelaskan masing-masing model-model proses perangkat lunak.
2. Apakah tujuan penggunaan model spiral?
3. Jelas hal-hal apa saja yang menjadi kekuatan dari masing-masing model yang digunakan dalam mengembangkan perangkat lunak.
4. Jelaskan bagaimana mekanisme kerja dari model waterfall.
5. Jelaskan secara ringkas mekanisme kerja dari model prototype.
6. Jelaskan secara ringkas mekanisme kerja dari model RAD.

BAB IV

SOFTWARE DEVELOPMENT LIFE CYCLE

Tujuan :

1. Mengenalkan fase-fase yang ada dalam pengembangan perangkat lunak.
2. Memahami tahapan-tahapan SDLC.

Indikator Keberhasilan :

1. Agar mahasiswa mampu menjelaskan masing-masing fase yang ada dalam SDLC
2. Agar mahasiswa mampu membuat contoh-contoh penerapan SDLC dalam kasus-kasus sederhana.

Materi :

Sejarah Software Development Life Cycle (SDLC)

Praktik dan metode untuk mengembangkan perangkat lunak telah berevolusi selama beberapa dekade sejak penemuan komputer. Metode-metode tersebut telah beradaptasi dengan berbagai kondisi yang ada pada perangkat keras komputer, *tools* pengembangan, dan pemikiran modern tentang manajemen organisasi tim pengembangan perangkat lunak. Dengan kemajuan ini, metode baru pengembangan perangkat lunak telah tumbuh dari upaya pengembangan perangkat lunak pribadi dan publik menjadi menglobal di seluruh dunia.

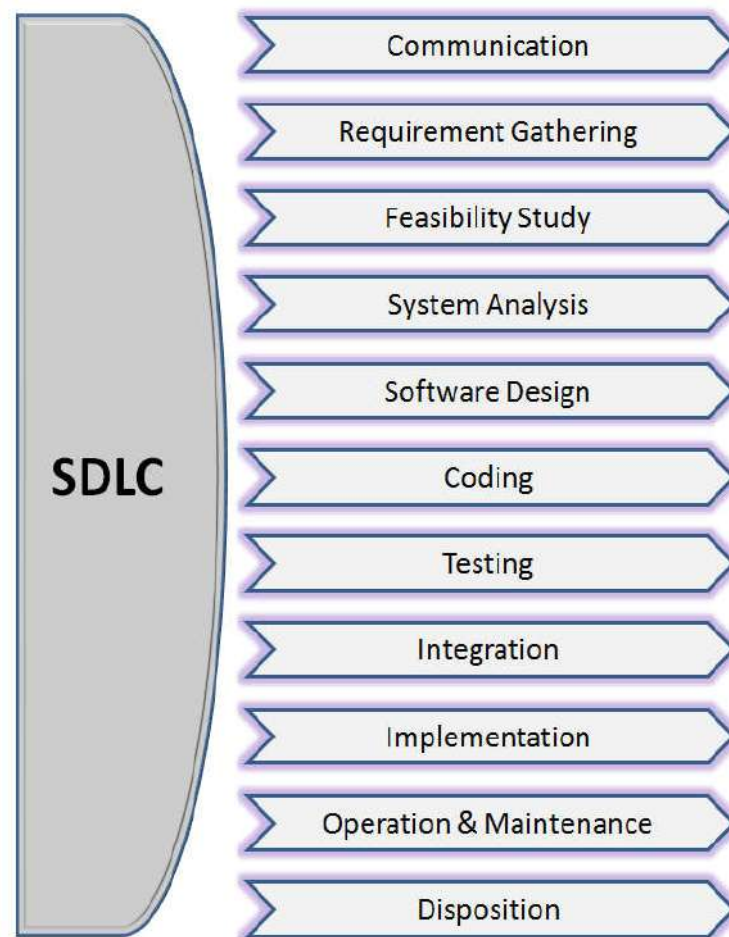
Metode-metode ini sangat bervariasi dalam pendekatan, namun mereka memiliki tujuan yang sama yaitu untuk mengembangkan perangkat lunak yang murah, efisien, dan efektif.

Perangkat lunak adalah produk kompleks yang dikembangkan dan disampaikan melalui serangkaian langkah yang terencana. Itulah satu-satunya kesamaan yang dimiliki berbagai metode yang ada dalam

pengembangan perangkat lunak, yaitu satu atau lain cara pengembangan perangkat lunak, seperti halnya semua produk, dimulai dari sebuah ide. Ide tersebut kemudian tersusun menjadi dokumen, atau mungkin prototipe, tergantung pada metode yang digunakan.

Dokumen, diagram, atau perangkat lunak yang berfungsi sebagai artefak yang dibuat dalam satu tahapan akan menjadi input untuk tahapan berikutnya. Akhirnya, perangkat lunak disampaikan (*delivery*) ke pelanggan. Urutan langkah yang digunakan oleh metode ini biasanya disebut sebagai *Software Development Life Cycle (SDLC)*.

SDLC merupakan tahapan-tahapan pekerjaan terdefinisi, terstruktur dengan baik yang dilakukan oleh analis sistem dan programmer dalam membangun suatu perangkat lunak yang berkualitas.



Gambar 4.1 : Tahapan *Software Development Life Cycle*

SDLC yang dilakukan dengan benar dapat memungkinkan tingkat kontrol dan dokumentasi manajemen tertinggi. Pengembang memahami apa yang harus mereka bangun dan mengapa. Semua pihak menyetujui tujuan di depan dan melihat rencana yang jelas untuk mencapai tujuan tersebut. Semua orang memahami biaya dan sumber daya yang dibutuhkan.

Beberapa jebakan dapat mengubah implementasi SDLC menjadi lebih banyak hambatan untuk pengembangan daripada alat yang membantu kami. Kegagalan untuk memperhitungkan kebutuhan pelanggan dan semua pengguna dan pemangku kepentingan dapat mengakibatkan pemahaman yang buruk tentang persyaratan sistem di awal. Manfaat SDLC hanya ada jika rencana tersebut diikuti dengan konsisten.

Aktifitas dalam SDLC:

Communication

Ini merupakan tahap pertama di mana pengguna mengungkapkan keinginan-keinginannya atas sebuah produk perangkat lunak. Pengguna menghubungi penyedia layanan dan melakukan negosiasi terkait beberapa hal, menyampaikan permintaan-permintaan kepada penyedia layanan secara tertulis.

Requirement Gathering

Setiap proyek perangkat lunak melewati fase yang disebut *Requirement Gathering*. Sebuah proyek yang sukses dimulai dengan serangkaian diskusi/pertemuan yang alot tentang apa yang harus dilakukan. Hal ini adalah tanggung jawab utama dari Tim Analisis untuk dapat mengumpulkan data persyaratan dari klien. Mendapatkan data persyaratan yang benar dari klien sering menjadi salah satu **rintangan terbesar** dalam proyek perangkat lunak apa pun. Jika tim analisis mengumpulkan persyaratan yang benar dan lengkap, proyek akan menghasilkan produk yang sangat berkualitas.

Bagi tim analis, perlu untuk memilih metode yang paling sesuai untuk mendapatkan data persyaratan. Juga, tim analis yang harus memutuskan siapa saja yang boleh terlibat dalam fase ini.

Jika tim analis menginvestasikan waktu dalam mengembangkan seperangkat persyaratan yang jelas, ringkas, benar dan terukur, biasanya memberi jaminan bahwa pengembangan perangkat lunak berkualitas sesuai kebutuhan klien. Bergantung pada situasi proyek, ada beberapa teknik yang dapat dipertimbangkan oleh tim analis untuk digunakan saat mengumpulkan data persyaratan dari klien.

Teknik Pengumpulan Data Kebutuhan

Ada banyak teknik yang tersedia untuk mengumpulkan data persyaratan. Setiap teknik memiliki kelebihan dalam skenario tertentu. Seringkali, tim analis perlu menggunakan beberapa teknik untuk mengumpulkan data persyaratan yang lengkap dan benar dari klien dan pemangku kepentingan. Berikut adalah beberapa teknik pengumpulan data persyaratan yang paling umum digunakan:

- 1) **Wawancara:** Ini adalah teknik yang paling umum digunakan dan bernilai. Dalam teknik ini, tim analis mengajukan pertanyaan tertentu kepada klien/pemangku kepentingan, mengenai persyaratan yang mereka inginkan atas perangkat lunak yang akan dikembangkan. Tim analis harus memastikan bahwa wawancara dilakukan pada bagian/unit kerja yang berbeda pada pihak pemangku kepentingan. Akan sangat baik untuk memulai sesi dengan wawancara tidak terstruktur untuk mendapatkan pemahaman tentang lingkungan kerja saat ini, tanyakan kepada orang yang diwawancara tentang pekerjaan/aktivitas dan masalah mereka. Setelah wawancara tidak terstruktur, langkah selanjutnya adalah wawancara terstruktur. Dalam wawancara terstruktur, tim analis menggunakan serangkaian pertanyaan yang disiapkan untuk mengumpulkan data persyaratan dari para pemangku kepentingan.

- 2) **Kuesioner:** Teknik ini merupakan pendekatan berbasis elektronik atau kertas untuk mengumpulkan data kebutuhan dari para pemangku kepentingan. Kuesioner dibagikan di antara para pemangku kepentingan. Kuesioner terdiri dari daftar semua pertanyaan yang relevan per proses tertentu. Kuisisioner adalah teknik yang baik untuk mengumpulkan data persyaratan dari lokasi terpisah. Kuesioner adalah metode yang tepat untuk mengumpulkan masukan dari sejumlah besar orang (populasi).
- 3) **Prototyping:** Beberapa pemangku kepentingan tidak memiliki pemahaman yang jelas tentang persyaratan yang mereka inginkan. Dalam skenario seperti itu, beberapa prototipe yang berbeda dibangun dengan kelompok persyaratan yang tersedia. Hal ini akan membantu klien untuk memahami sistem dan persyaratan lebih lanjut. Anda perlu mengulangi prosesnya sampai aplikasi memenuhi persyaratan utama.
- 4) **Analisis Dokumen:** Teknik ini dilakukan dengan melakukan analisis terhadap dokumen dari sistem yang ada saat ini. Tim analisis menggali informasi/persyaratan dari dokumen ini. Ini juga membantunya mempersiapkan pertanyaan untuk memvalidasi kebenaran dan kelengkapan data persyaratan.
- 5) **Observasi:** Dalam teknik ini, tim analisis mengumpulkan data persyaratan dengan cara mengamati proses kerja sistem saat ini. Ini membantunya untuk memahami seluruh sistem dan persyaratan yang terkait. Terkadang akan sangat baik dilakukan dengan cara berpartisipasi dalam proses kerja yang sebenarnya untuk memahami dan menangkap persyaratan.

Berikut adalah beberapa panduan yang dapat membantu tim analisis untuk menangkap data persyaratan lengkap yang benar:

- a. Pilih kelompok wawancara dengan bijak. Tim analisis harus mempertimbangkan berbagai faktor saat pemilihan grup: Kematangan Teknis, Pengetahuan Proses Bisnis, Spesialisasi, Minat, Departemen, bagian Organisasi, dan Ketersediaan Waktu.

- b. Sebelum memulai pengumpulan data persyaratan, ada baiknya meringkas/memperkenalkan proyek dan tujuan terkait, untuk menghindari kesalahpahaman tim yang terlibat dalam fase ini.
- c. Tim analis harus menindaklanjuti setiap pertanyaan dengan satu set pertanyaan klarifikasi untuk menggali lebih jauh informasinya.
- d. Selalu mengambil pendekatan kolaboratif sambil mengumpulkan data persyaratan.
- e. Kuesioner dalam bentuk deskriptif dan sederhana untuk dipahami.
- f. Jalankan Jajak Pendapat, sehingga pengguna member daftar peringkat kebutuhan mereka, dan memiliki kondisi untuk memberikan umpan balik.

Feasiblity Study

Setelah mengumpulkan data kebutuhan, tim melanjutkan dengan rencana umum proses perangkat lunak. Di tahap ini tim menganalisa jika sebuah perangkat lunak dapat didesain untuk memenuhi seluruh kebutuhan pengguna, dan jika terdapat kemungkinan perangkat lunak tidak lagi berguna. Selain itu, juga dianalisa jika proyek layak untuk diambil secara financial, praktis/operasional, dan teknologi. Ada beberapa algoritma tersedia, yang dapat membantu para pengembang untuk mengambil kesimpulan terkait kelayakan sebuah proyek perangkat lunak.

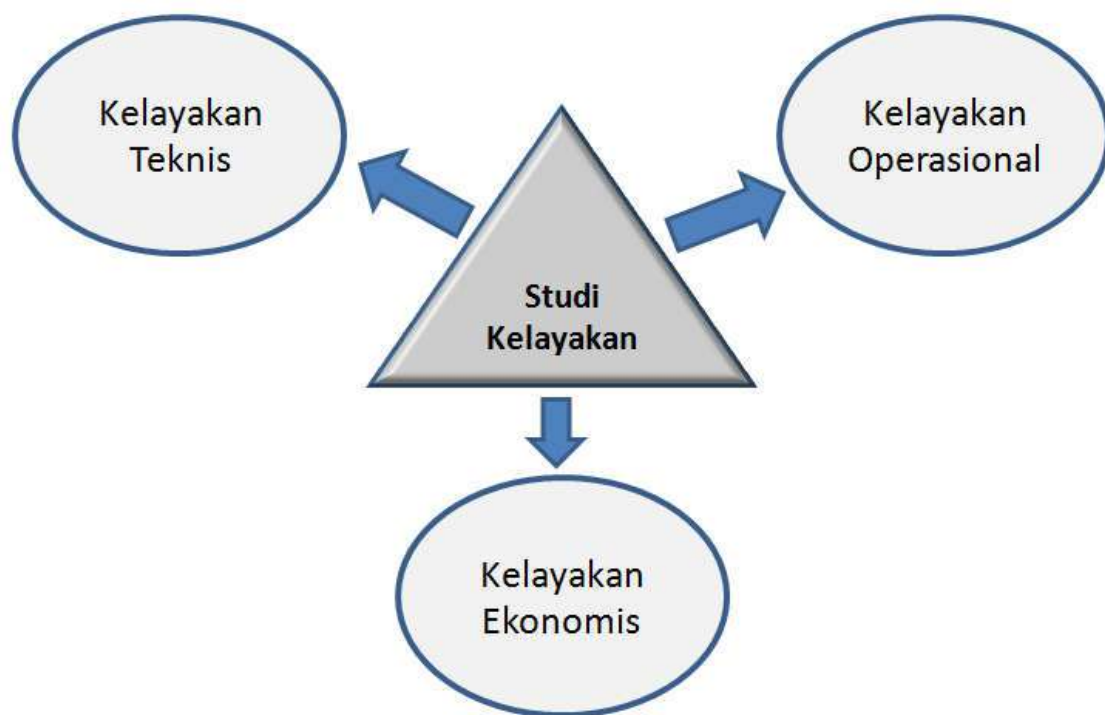
Kelayakan didefinisikan sebagai aktivitas untuk menilai secara praktis, sejauh mana proyek perangkat lunak dapat dilaksanakan dengan sukses. Untuk mengevaluasi kelayakan, studi dilakukan, untuk menentukan apakah solusi yang dipilih memenuhi persyaratan praktis dan dapat diterapkan dalam perangkat lunak. Informasi seperti ketersediaan sumber daya, perkiraan biaya untuk pengembangan perangkat lunak, manfaat dari perangkat lunak untuk organisasi setelah dikembangkan dan biaya yang harus dikeluarkan untuk pemeliharaannya dipertimbangkan selama studi kelayakan. Tujuan dari studi kelayakan adalah untuk menetapkan alasan untuk mengembangkan perangkat lunak yang dapat diterima pengguna,

dapat beradaptasi dengan perubahan dan sesuai dengan standar yang ditetapkan. Berbagai tujuan lain dari studi kelayakan tercantum di bawah ini:

- Untuk menganalisis apakah perangkat lunak akan memenuhi persyaratan organisasi,
- Untuk menentukan apakah perangkat lunak dapat diimplementasikan menggunakan teknologi saat ini dan dalam anggaran dan jadwal yang ditentukan,
- Untuk menentukan apakah perangkat lunak dapat diintegrasikan dengan perangkat lunak lain yang ada.

Bentuk-bentuk kelayakan:

Berbagai jenis kelayakan yang biasanya dipertimbangkan mencakup kelayakan teknis, kelayakan operasional, dan kelayakan ekonomi.



Gambar 4.2: Bentuk-bentuk kelayakan

Kelayakan teknis menilai sumber daya saat ini (seperti perangkat keras dan perangkat lunak) dan teknologi, yang diperlukan untuk memenuhi

persyaratan pengguna perangkat lunak dalam waktu dan anggaran yang akan dialokasikan. Dalam hal ini, tim pengembangan perangkat lunak memastikan apakah sumber daya dan teknologi yang ada saat ini dapat ditingkatkan atau ditambahkan ke dalam perangkat lunak untuk memenuhi persyaratan pengguna tertentu. Kelayakan teknis adalah melakukan tugas-tugas berikut:

- Menganalisa keterampilan dan kemampuan teknis anggota tim pengembangan perangkat lunak
- Menentukan apakah teknologi yang ada masih stabil dan mapan.
- Memastikan bahwa teknologi yang dipilih dalam pengembangan perangkat lunak yang memiliki sejumlah besar pengguna sehingga dapat dikonsultasikan ketika masalah muncul atau bilamana perbaikan diperlukan.

Kelayakan operasional menilai sejauh mana perangkat lunak yang diperlukan untuk melakukan serangkaian langkah untuk memecahkan masalah bisnis dan persyaratan pengguna. Kelayakan ini tergantung pada sumber daya manusia (tim pengembangan perangkat lunak) dan menunjukkan secara visual apakah perangkat lunak akan beroperasi setelah dikembangkan dan beroperasi setelah diinstal. Kelayakan operasional adalah kegiatan melakukan tugas-tugas berikut:

- Menentukan apakah masalah yang paling tinggi prioritasnya sehingga dapat diantisipasi dalam persyaratan pengguna.
- Menentukan apakah solusi yang disarankan oleh tim pengembangan perangkat lunak dapat diterima
- Analisis apakah pengguna akan mampu beradaptasi dengan perangkat lunak yang baru.
- Menentukan apakah organisasi puas dengan solusi alternatif yang diajukan oleh tim pengembangan perangkat lunak.

Kelayakan ekonomi menentukan apakah perangkat lunak yang diperlukan mampu menghasilkan keuntungan finansial bagi suatu organisasi. Ini

melibatkan biaya yang dikeluarkan oleh tim pengembangan perangkat lunak, perkiraan biaya pengadaan perangkat keras dan perangkat lunak, biaya melakukan studi kelayakan, dan seterusnya. Dengan demikian, penting untuk mempertimbangkan pengeluaran yang dilakukan pada kegiatan pembelian (seperti pembelian perangkat keras) dan kegiatan yang diperlukan untuk melakukan pengembangan perangkat lunak. Selain itu, perlu mempertimbangkan manfaat yang dapat dicapai dengan mengembangkan perangkat lunak. Perangkat lunak dikatakan layak secara ekonomi jika berfokus pada hal-hal yang tercantum di bawah ini:

- Biaya yang dikeluarkan dalam proses pengembangan perangkat lunak untuk menghasilkan keuntungan jangka panjang bagi suatu organisasi
- Biaya yang diperlukan untuk melakukan penelitian tentang perangkat lunak secara lengkap (seperti persyaratan elisitasi dan analisis persyaratan)
- Biaya pengadaan perangkat keras, perangkat lunak, tim pengembangan, dan pelatihan.

Proses Studi Kelayakan meliputi tahapan berikut ini:

1. *Information Assessment*: Mengidentifikasi informasi tentang apakah sistem dapat membantu dalam mencapai tujuan organisasi. Dalam hal ini juga dilakukan verifikasi bahwa sistem dapat diimplementasikan menggunakan teknologi baru sesuai anggaran dan apakah sistem dapat diintegrasikan dengan sistem yang ada.
2. Pengumpulan informasi: Menentukan sumber dari mana informasi tentang konten perangkat lunak dapat diperoleh. Umumnya, sumber-sumber ini termasuk pengguna (yang akan mengoperasikan perangkat lunak), organisasi (di mana perangkat lunak akan digunakan), dan tim pengembangan perangkat lunak (yang memahami persyaratan pengguna dan tahu bagaimana memenuhinya dalam perangkat lunak).

3. Penulisan laporan: Menggunakan laporan kelayakan, yang merupakan kesimpulan dari studi kelayakan oleh tim pengembangan perangkat lunak. Ini termasuk rekomendasi apakah pengembangan perangkat lunak harus dilanjutkan atau sebaliknya. Laporan ini juga dapat mencakup informasi tentang perubahan dalam lingkup perangkat lunak, anggaran, dan jadwal serta saran dari setiap persyaratan dalam sistem.
4. Informasi umum: Menjelaskan tujuan dan ruang lingkup studi kelayakan. Ini juga menggambarkan gambaran sistem, referensi proyek, akronim dan singkatan, dan kontak person yang dapat dihubungi. Gambaran sistem memberikan deskripsi tentang nama organisasi yang bertanggung jawab untuk pengembangan perangkat lunak, nama atau judul sistem, kategori sistem, status operasional, dan sebagainya. Referensi proyek menyediakan daftar referensi yang digunakan untuk menyiapkan dokumen ini seperti dokumen yang berkaitan dengan proyek atau dokumen yang dikembangkan sebelumnya yang terkait dengan proyek. Akronim dan singkatan menyediakan daftar istilah yang digunakan dalam dokumen ini beserta artinya. Kontak person terdiri dari personil dalam organisasi pengembang yang dapat dihubungi dengan tujuan komunikasi dengan pengguna untuk informasi dan koordinasi. Misalnya, pengguna memerlukan bantuan untuk memecahkan masalah dan mengumpulkan informasi seperti nomor kontak, alamat e-mail, dan sebagainya.
5. Ringkasan manajemen: Merupakan dokumen singkat atau bagian dari dokumen, yang diproduksi untuk tujuan tertentu. Laporan atau proposal yang lebih panjang atau sekelompok laporan. disusun sedemikian rupa sehingga pembaca dapat dengan cepat memahami banyak materi tanpa harus membaca secara menyeluruh.
6. Lingkungan: Mengidentifikasi individu yang bertanggung jawab untuk pengembangan perangkat lunak. Ini memberikan informasi tentang persyaratan input dan output, persyaratan pemrosesan

perangkat lunak dan interaksi perangkat lunak dengan perangkat lunak lain. Ini juga mengidentifikasi persyaratan keamanan sistem dan persyaratan pemrosesan sistem.

7. Prosedur fungsional saat ini: Menjelaskan prosedur fungsional saat ini dari sistem yang ada, apakah otomatis atau manual. Ini juga termasuk aliran data dari sistem saat ini dan jumlah anggota tim yang diperlukan untuk mengoperasikan dan memelihara perangkat lunak.
8. Tujuan Fungsional: Memberikan informasi tentang fungsi sistem seperti layanan baru, peningkatan kapasitas, dan sebagainya.
9. Tujuan kinerja: Memberikan informasi tentang tujuan kinerja seperti mengurangi staf dan biaya peralatan, meningkatkan kecepatan pemrosesan perangkat lunak, dan meningkatkan pengawasan.
10. Asumsi dan batasan: Memberikan informasi tentang asumsi dan kendala seperti kehidupan operasional perangkat lunak yang diusulkan, kendala keuangan, perubahan perangkat keras, perangkat lunak dan lingkungan operasi, dan ketersediaan informasi dan sumbernya.
11. Metodologi: Menjelaskan metode yang diterapkan untuk mengevaluasi perangkat lunak yang diusulkan untuk mencapai alternatif yang layak. Metode-metode ini termasuk survei, pemodelan, perbandingan, dan sebagainya.
12. Kriteria evaluasi: Mengidentifikasi kriteria seperti biaya, prioritas, waktu pengembangan, dan kemudahan penggunaan sistem, yang berlaku untuk proses pengembangan untuk menentukan opsi sistem yang paling sesuai.
13. Rekomendasi: Menjelaskan rekomendasi untuk sistem yang diusulkan. Ini termasuk penundaan dan risiko yang dapat diterima.
14. Perangkat lunak yang diusulkan: Menjelaskan konsep sistem secara keseluruhan serta prosedur yang akan digunakan untuk memenuhi persyaratan pengguna. Selain itu, ia memberikan informasi tentang peningkatan kinerja, waktu dan biaya sumber daya, dan dampak.

Perbaikan dilakukan untuk meningkatkan fungsionalitas dan kinerja perangkat lunak yang ada. Waktu dan biaya sumber daya termasuk biaya yang terkait dengan pengembangan perangkat lunak dari persyaratannya hingga pemeliharaan dan pelatihan stafnya. Dampak menggambarkan kemungkinan kejadian di masa depan dan mencakup berbagai jenis dampak seperti yang tercantum di bawah ini:

1. Dampak peralatan: Tentukan persyaratan peralatan baru dan perubahan yang harus dilakukan dalam persyaratan peralatan yang tersedia saat ini.
2. Dampak perangkat lunak: Tentukan penambahan atau modifikasi apa pun yang diperlukan dalam perangkat lunak yang ada dan perangkat lunak pendukung untuk beradaptasi dengan perangkat lunak yang diusulkan.
3. Dampak organisasi: Jelaskan perubahan apa pun dalam persyaratan organisasi, staf, dan skill.
4. Dampak operasional: Jelaskan efek pada operasi seperti prosedur operasi pengguna, pemrosesan data, prosedur pemasukan data, dan sebagainya.
5. Dampak perkembangan: Tentukan dampak perkembangan seperti sumber daya yang diperlukan untuk mengembangkan basis data, sumber daya yang diperlukan untuk mengembangkan dan menguji perangkat lunak, dan kegiatan spesifik yang akan dilakukan oleh pengguna selama pengembangan perangkat lunak.
6. Dampak keamanan: Jelaskan faktor keamanan yang dapat mempengaruhi pengembangan, desain, dan operasi lanjutan dari perangkat lunak yang diusulkan.

System Analysis

Pada tahapan ini pengembang menetapkan sebuah roadmap dari rencananya dan mencoba untuk menciptakan model perangkat lunak

terbaik yang cocok untuk proyek tersebut. Tahapan analisa sistem mencakup pemahaman terhadap keterbatasan produk perangkat lunak, mempelajari masalah yang terkait dengan sistem, mengidentifikasi dan mendapatkan dampak dari proyek bagi organisasi dan personil. Tim proyek melakukan analisa terhadap cakupan/lingkup proyek dan menyusun rencana jadwal dan sumber daya secara tepat.

Software Design

Tahapan berikutnya adalah menuangkan seluruh pengetahuan tentang kebutuhan-kebutuhan dan hasil analisa ke dalam bentuk rancangan produk perangkat lunak. Masukan-masukan dari pengguna dan informasi yang berhasil dikumpulkan pada saat fase *requirement gathering* merupakan bahan pokok untuk fase ini. Hasil dari fase ini ada dalam dua bentuk; rancangan secara logis (*logical design*), dan rancangan secara fisik (*physical design*). Para perancangan menghasilkan *meta-data* dan kamus data, diagram logis, data-flow diagrams (DFD), dan dalam beberapa kasus terdapat pseudo code.

Coding

Tahap ini juga dikenal sebagai fase pemrograman. Implementasi rancangan perangkat lunak dimulai dengan penulisan kode program dalam bahasa pemrograman yang sesuai dan mengembangkan program yang bebas dari kesalahan secara efisien.

Testing

Suatu hasil penelitian menyatakan bahwa rata-rata proses pengembangan perangkat lunak yang ada harus diuji coba. Pengujian perangkat lunak dilakukan selama kegiatan coding dilakukan oleh pengembang dan melalui pengujian yang dilakukan oleh ahlinya dengan berbagai level pemrograman seperti pengujian modul, pengujian program, pengujian produk, in-house testing, dan pengujian produk pada pengguna akhir. Penanganan yang

cepat atas *error* yang ditemukan dan memperbaikinya merupakan kunci keberhasilan suatu produk perangkat lunak.

Integration

Perangkat lunak dapat saja terintegrasi dengan *library* fungsi, database, dan program lainnya. Tahapan ini meliputi kegiatan untuk mengintegrasikan perangkat lunak dengan entitas ekseternal.

Implementation

Hal ini berarti melakukan instalasi perangkat lunak pada mesin milik pengguna. Pada saat itu, perangkat lunak dikonfigurasi akhir sesuai dengan mesin pemakai. Perangkat lunak diuji kembali kesesuaian dan adaptasi serta integrasinya.

Operation and Maintenance

Tahap ini mengkonfirmasikan operasi perangkat lunak. Jika diperlukan, pengguna diberikan training, atau didukung dengan dokumen tentang bagaimana mengoperasikan perangkat lunak dan bagaimana menjaga agar software dapat terus beroperasi secara berkelanjutan.

Soal

1. Uraikan secara ringkas output dari masing-masing tahap yang ada dalam SDLC
2. *Feasibility study* merupakan tinjauan terhadap kelayakan suatu proyek perangkat lunak. Jelaskan aspek-aspek kelayakan yang dimaksud dan berikan contohnya.
3. Mengapa SDLC dikatakan sebagai sebuah siklus dalam pengembangan perangkat lunak?

BAB V

PERSYARATAN PERANGKAT LUNAK

Tujuan :

1. Mengenalkan pentingnya *software requirements*
2. Menjelaskan konsep dasar *software requirements*
3. Mempelajari tentang instrument-instrumen yang digunakan dalam proses pengumpulan data persyaratan perangkat lunak

Indikator Keberhasilan :

1. Agar mahasiswa mampu memahami pengertian *software requierements* beserta kegunaannya.
2. Agar mahasiswa mampu menjelaskan konsep dasar *software requirements* dan menerapkannya dalam kasus-kasus sederhana.
3. Mahasiswa mampu menyusun dokumen persyaratan dalam bentuk kasus sederhana.

Materi :

Persyaratan perangkat lunak terkadang dikenal dengan istilah *software requirements* adalah deskripsi fitur dan fungsionalitas sistem yang dijadikan target pengembangan. Melalui dokumen persyaratan ini user menyampaikan harapannya terhadap produk perangkat lunak yang akan dihasilkan. Persyaratan dapat dibuat secara jelas atau tersembunyi (tersirat), diketahui atau tidak diketahui, diharapkan atau tidak diharapkan. Semuanya persepsi tersebut dilihat dari sudut pandang pengguna.

Output dari tahap pengumpulan data persyaratan dari proses pengembangan perangkat lunak adalah Spesifikasi Kebutuhan Perangkat

Lunak (SKPL) juga dikenal sebagai dokumen persyaratan. Dokumen ini meletakkan dasar untuk kegiatan rekayasa perangkat lunak dan dibuat ketika seluruh persyaratan telah diperoleh dan dianalisis. SKPL adalah dokumen formal, yang berfungsi sebagai representasi perangkat lunak yang memungkinkan pengguna untuk meninjau apakah SKPL sesuai dengan kebutuhan mereka. Selain itu, dokumen ini mencakup persyaratan pengguna untuk sistem serta spesifikasi detail persyaratan sistem.

Rekayasa Kebutuhan

Proses untuk mengumpulkan data terkait dengan persyaratan perangkat lunak dari klien, menganalisis, dan mendokumentasikannya dikenal dengan istilah rekayasa kebutuhan.

Tujuan kegiatan rekayasa kebutuhan adalah untuk mengembangkan dan menjamin keberadaan dokumen 'spesifikasi sistem' yang canggih dan deskriptif.

Proses Rekayasa Kebutuhan

Terdapat empat langkah proses dalam aktifitas rekayasa kebutuhan yang mencakup:

a. Studi Kelayakan

Di saat klien mendatangi pihak pengembang untuk menyampaikan gambaran produk yang diinginkan dikembangkan, maka sebelumnya telah muncul ide kasar tentang apa yang mampu dilakukan dan semua fitur yang diharapkan dari suatu perangkat lunak.

Merujuk kepada informasi ini, para analis melakukan studi secara mendalam tentang apakah sistem yang diinginkan dan fungsinya layak untuk dikembangkan.

Studi kelayakan ini difokuskan pada tujuan organisasi. Studi ini menganalisa apakah produk perangkat lunak dapat terwujud dalam bentuk yang dapat diimplementasikan, kontribusi proyek untuk organisasi, kendala pembiayaan, dan kesesuaian dengan nilai dan tujuan organisasi. Agar hal ini dapat diwujudkan maka akan

dilakukan kegiatan mengeksplorasi aspek teknis dari proyek dan produk seperti daya guna, pemeliharaan, produktivitas, dan kemampuan integrasi.

Hasil akhir dari fase ini diperoleh dalam bentuk laporan studi kelayakan yang harus mengandung komentar dan rekomendasi yang memadai untuk pihak manajemen tentang apakah proyek harus dilaksanakan atau tidak.

b. Pengumpulan data kebutuhan

Apabila studi laporan kelayakan menunjukkan hasil yang positif untuk melakukan proyek, maka tahap berikutnya dimulai dengan mengumpulkan data/informasi persyaratan dari pengguna. Tim analis berkomunikasi secara intensif dengan klien dan pengguna akhir untuk mengetahui keinginan mereka tentang apa yang harus disediakan pada perangkat lunak dan fitur apa yang mereka inginkan untuk disediakan pada perangkat lunak.

c. Spesifikasi Kebutuhan Perangkat Lunak (SKPL)

SKPL adalah dokumen yang dibuat oleh sistem analis setelah semua persyaratan perangkat lunak dikumpulkan dari berbagai pemangku kepentingan.

SKPL mendefinisikan bagaimana perangkat lunak yang dimaksud berinteraksi dengan perangkat keras, antarmuka eksternal, kecepatan operasi, waktu respons sistem, portabilitas perangkat lunak di berbagai platform, pemeliharaan, kecepatan pemulihan setelah adanya gangguan, keamanan, kualitas, batasan, dan sebagainya.

Persyaratan yang telah diterima dari klien ditulis dalam bahasa alami. Selanjutnya salah satu tanggung jawab sistem analis untuk mendokumentasikan persyaratan dalam bahasa teknis sehingga dapat dipahami dan digunakan oleh tim pengembangan perangkat lunak.

SKPL seharusnya memiliki fitur-fitur berikut:

- Persyaratan pengguna dinyatakan dalam bahasa natural

- Persyaratan teknis dinyatakan dalam bahasa terstruktur, yang digunakan di dalam organisasi tersebut
- Deskripsi rancangan harus ditulis dalam Pseudocode
- Format formulir dan cetakan layar GUI.
- Notasi yang bersifat kondisional dan matematis untuk DFD, dan sebagainya.

d. Validasi Kebutuhan Perangkat Lunak

Setelah spesifikasi persyaratan dikembangkan, persyaratan yang disebutkan dalam dokumen itu harus divalidasi. Pengguna mungkin meminta solusi yang tidak lazim atau tidak praktis yang mungkin menafsirkan persyaratan secara tidak akurat. Hal ini berpotensi menghasilkan peningkatan besar dalam hal biaya jika tidak disepakati sejak awal. Persyaratan dapat diperiksa terhadap kondisi berikut:

- Jika ia dapat diimplementasikan secara praktis
- Jika ia bersifat valid dan sesuai dengan fungsi dan domain perangkat lunak
- Jika ada ambiguitas
- Jika lengkap
- Jika dapat ditunjukkan

Requirements Elicitation Process

Proses elisitasi kebutuhan dapat digambarkan menggunakan diagram berikut ini.



Gambar 5.1 : Proses elisitasi kebutuhan

- a. **Requirements Gathering.** Pengembang mendiskusikan dengan klien dan pengguna akhir dan mengetahui harapan mereka dari perangkat lunak yang dikembangkan.

- b. **Requirements Organisation.** Para pengembang memprioritaskan dan mengatur persyaratan dalam urutan kepentingan, urgensi dan kenyamanan.
- c. **Negotiation & Discussion.** Apabila persyaratannya ambigu atau ada beberapa konflik dalam persyaratan berbagai pemangku kepentingan, hal ini kemudian dinegosiasikan dan didiskusikan dengan para pemangku kepentingan. Persyaratan kemudian dapat diprioritaskan dan dikompromikan dengan layak.
Persyaratannya berasal dari berbagai pemangku kepentingan. Untuk menghilangkan ambiguitas dan konflik, perlu dilakukan pembahasan untuk mendapatkan kejelasan dan kebenaran. Persyaratan yang tidak realistis dikompromikan secara wajar.
- d. **Requirements Specification.** Semua persyaratan formal dan informal, fungsional dan non-fungsional didokumentasikan menjadi suatu spesifikasi dan tersedia untuk pemrosesan tahap berikutnya.

Teknik Elisitasi Persyaratan

Elisitasi persyaratan adalah proses untuk mengetahui persyaratan untuk sistem perangkat lunak yang dimaksudkan dengan berkomunikasi dengan klien, pengguna akhir, pengguna sistem, dan orang lain yang memiliki andil dalam pengembangan sistem perangkat lunak.

Terdapat berbagai instrumen yang digunakan untuk memperoleh data-data persyaratan perangkat lunak. Beberapa di antaranya dijelaskan di bawah ini:

A. Wawancara

Wawancara adalah media yang sangat diandalkan untuk mengumpulkan persyaratan. Organisasi dapat melakukan beberapa jenis wawancara seperti:

- Wawancara terstruktur (tertutup), di mana setiap informasi yang akan dikumpulkan telah ditetapkan sebelumnya, sehingga pihak

yang terlibat harus mengikuti pola dan materi diskusi dengan tegas.

- Wawancara non-terstruktur (terbuka), di mana setiap informasi yang akan dikumpulkan tidak ditetapkan sebelumnya, lebih fleksibel dan sedikit bias.
- Wawancara oral
- Wawancara tertulis
- Wawancara satu-ke-satu yang diadakan antara dua orang pada sebuah meja
- Wawancara kelompok yang diadakan antara kelompok peserta. Mereka membantu untuk mengungkap setiap kebutuhan yang hilang karena banyak personil yang terlibat.



Gambar 5.2: Kegiatan Wawancara (Sumber: www.rebanas.com)

B. Survei

Pengembang dapat melakukan survei di sejumlah lokasi/unit kerja di pihak pemangku kepentingan dengan menanyakan tentang harapan dan persyaratan sistem yang diinginkan di masa yang akan datang.

C. Kuesioner

Sebuah dokumen dengan serangkaian pertanyaan obyektif yang ditetapkan sebelumnya dan opsi-opsi masing-masing diserahkan kepada semua pemangku kepentingan untuk dijawab, yang selanjutnya dikumpulkan dan dikompilasi.

Kelemahan dari teknik ini adalah, jika opsi untuk beberapa masalah tidak disebutkan dalam kuesioner, masalah ini mungkin dibiarkan tanpa pengawasan.

D. Analisa Pekerjaan

Tim pengembang dapat menganalisa operasi/unit yang membutuhkan sistem baru. Jika klien sudah memiliki beberapa perangkat lunak untuk melakukan operasi tertentu, itu dipelajari dan persyaratan sistem yang diusulkan dikumpulkan.

E. Analisa Domain

Setiap perangkat lunak termasuk dalam beberapa kategori domain. Orang-orang ahli dalam domain dapat sangat membantu untuk menganalisis persyaratan umum dan khusus.

F. Brainstorming

Sebuah debat informal diadakan di antara berbagai pemangku kepentingan dan semua masukan mereka dicatat untuk analisis persyaratan lebih lanjut

G. Prototyping

Prototyping adalah membangun antarmuka pengguna tanpa menambahkan fungsionalitas detail bagi pengguna untuk menafsirkan fitur-fitur produk perangkat lunak yang dimaksudkan. Ini membantu memberikan ide persyaratan yang lebih baik. Jika tidak ada perangkat lunak yang dipasang di sisi klien untuk referensi pengembang dan klien tidak mengetahui persyaratannya sendiri, pengembang membuat prototipe berdasarkan persyaratan yang disebutkan sebelumnya. Prototype ditunjukkan kepada klien dan umpan balik dicatat. Umpan balik klien berfungsi sebagai masukan untuk pengumpulan kebutuhan.

H. Observasi

Istilah observasi berasal dari bahasa Latin yang berarti "melihat" dan "memperhatikan". Istilah observasi diarahkan pada kegiatan memperhatikan secara akurat, mencatat fenomena yang muncul, dan mempertimbangkan hubungan antar aspek dalam fenomena tersebut. Tim ahli mengunjungi organisasi atau tempat kerja klien. Mereka mengamati kerja sebenarnya dari sistem yang ada saat ini. Mereka mengamati alur kerja di sisi klien dan bagaimana masalah eksekusi. Tim itu sendiri menarik beberapa kesimpulan yang membantu untuk membentuk persyaratan yang diharapkan dari perangkat lunak yang akan datang.

Macam – macam teknik observasi:

a. Observasi Partisipan

Suatu observasi disebut observasi partisipan jika orang yang mengadakan observasi (observer) turut ambil bagian dalam kehidupan observer.

b. Observasi Sistematis

Observasi sistematis biasa disebut juga observasi berkerangka atau structured observation

c. Observasi Eksperimental

Observasi dapat dilakukan dalam lingkup alamiah/natural ataupun dalam lingkup experimental.

Karakteristik Persyaratan Software

Mengumpulkan kebutuhan perangkat lunak adalah dasar dari keseluruhan proyek pengembangan perangkat lunak. Oleh karena itu ia harus jelas, benar, dan terdefinisi dengan baik.

Spesifikasi Persyaratan Perangkat Lunak yang lengkap harus:

- Jelas
- Benar
- Konsisten
- Koheren

- Komprehensif
- Dapat dimodifikasi
- Dapat diverifikasi
- Memiliki prioritas
- Tidak ada unsur ambiguitas
- Dapat dilacak
- Sumbernya kredibel

Persyaratan Perangkat Lunak

Pengembang harus mencoba memahami persyaratan apa yang mungkin timbul dalam fase elisitasi persyaratan dan persyaratan apa yang diharapkan dari sistem perangkat lunak.

Persyaratan perangkat lunak secara luas harus dikategorikan dalam dua kategori:

A. Persyaratan Bersifat Fungsional

Persyaratan, yang terkait dengan aspek fungsional perangkat lunak termasuk dalam kategori ini.

Mereka mendefinisikan fungsi dan fungsi di dalam dan dari sistem perangkat lunak.

Contoh:

- Opsi pencarian yang diberikan kepada pengguna untuk mencari dari berbagai faktur.
- Pengguna harus dapat mengirim laporan apa pun ke manajemen.
- Pengguna dapat dibagi menjadi kelompok dan kelompok dapat diberikan hak terpisah.
- Harus mematuhi aturan bisnis dan fungsi administratif.
- Perangkat lunak dikembangkan agar kompatibilitas tetap utuh

B. Persyaratan Bersifat Non-Fungsional

Persyaratan, yang tidak terkait dengan aspek fungsional perangkat lunak, termasuk dalam kategori ini. Mereka adalah karakteristik implisit atau yang diharapkan dari perangkat lunak, yang mana pengguna membuat asumsi.

Persyaratan non-fungsional termasuk:

- Keamanan
- Logging
- Penyimpanan
- Konfigurasi
- Kinerja
- Biaya
- Interoperabilitas
- Fleksibilitas
- Pemulihan pasca bencana
- Aksesibilitas

Persyaratan dikategorikan secara logis sebagai:

- **Harus Memiliki:** Perangkat lunak tidak dapat dikatakan beroperasi tanpa keberadaannya.
- **Seharusnya:** Meningkatkan fungsionalitas perangkat lunak.
- **Bisa memiliki:** Perangkat lunak masih dapat berfungsi dengan baik dengan adanya persyaratan ini.
- **Daftar keinginan:** Persyaratan ini tidak dipetakan ke setiap tujuan perangkat lunak.

Saat mengembangkan perangkat lunak, 'Harus memiliki' harus diterapkan, 'Seharusnya' adalah masalah debat dengan pemangku kepentingan dan negasi, sedangkan 'Bisa memiliki' dan 'Daftar keinginan' dapat disimpan untuk pembaruan perangkat lunak.

Persyaratan Antarmuka Pengguna (user interface)

Antarmuka Pengguna adalah bagian penting dari perangkat lunak atau perangkat keras atau sistem hibrid. Perangkat lunak diterima secara luas apabila:

- mudah dioperasikan
- cepat memberikan respon
- efektif menangani kesalahan operasional

- menyediakan antarmuka pengguna yang sederhana namun konsisten.

Penerimaan pengguna sangat tergantung pada bagaimana pengguna dapat menggunakan perangkat lunak. Antarmuka pengguna adalah satu-satunya cara bagi pengguna untuk berinteraksi dengan sistem. Sistem perangkat lunak yang berfungsi baik juga harus dilengkapi dengan antarmuka pengguna yang menarik, jelas, konsisten, dan responsif. Jika tidak, fungsionalitas sistem perangkat lunak tidak dapat digunakan dengan cara yang nyaman. Suatu sistem dikatakan baik jika ia menyediakan sarana untuk menggunakannya secara efisien. Persyaratan antarmuka pengguna secara singkat disebutkan di berikut ini:

- Presentasi konten
- Navigasi Mudah
- Antarmuka yang sederhana
- Responsif
- Elemen antarmuka pengguna yang konsisten
- Mekanisme umpan balik
- Pengaturan standar
- Tata letak yang memiliki tujuan
- Penggunaan warna dan tekstur secara tepat.
- Berikan informasi bantuan
- Pendekatan sentris pengguna
- Pengaturan tampilan berbasis grup.

Tips dalam menyusun SKPL

Setiap SKPL harus memiliki pola tertentu. Dengan demikian, adalah penting untuk menstandarisasi struktur dokumen persyaratan untuk membuatnya lebih mudah dipahami. Pada standar IEEE yang digunakan untuk SKPL untuk mengatur persyaratan pada proyek yang berbeda, yang menyediakan cara berbeda untuk menyusun SKPL. Harap diperhatikan bahwa dalam semua dokumen persyaratan, dua bagian pertama adalah sama. Adapun

bagian-bagian yang harus ada dalam sebuah dokumen SKPL adalah sebagai berikut:

1. Pengantar

Bagian ini memberikan ringkasan dari seluruh informasi yang dijelaskan dalam SKPL. Di dalamnya terdapat tujuan dan ruang lingkup SKPL, yang menyatakan fungsi yang harus dilakukan oleh sistem. Selain itu, bagian ini menggambarkan definisi, singkatan, dan akronim yang digunakan. Referensi yang digunakan dalam SKPL sebaiknya menyediakan daftar dokumen yang direferensikan dalam dokumen.

2. Gambaran Umum

Bagian ini menjelaskan faktor-faktor yang mempengaruhi persyaratan sistem. Di dalamnya terdapat deskripsi singkat tentang persyaratan yang harus didefinisikan di bagian selanjutnya yang disebut 'persyaratan khusus'.

3. Gambaran Produk

Bagian ini menjelaskan apakah produk yang akan dikembangkan adalah produk independen atau bagian integral dari produk yang lebih besar. Di dalamnya juga terdapat bagian yang menentukan antarmuka dengan perangkat keras, perangkat lunak, sistem, dan model komunikasinya. Selain itu juga diinformasikan kendala memori dan operasi yang digunakan oleh pengguna.

4. Fungsi Produk

Bagian ini menyediakan ringkasan fungsi yang harus dilakukan oleh perangkat lunak. Fungsi disusun dalam daftar sehingga mudah dimengerti oleh pengguna.

5. Karakteristik Pemakai

Bagian ini memberikan gambaran tentang karakteristik umum dari pengguna

6. Batasan-batasan

Bagian ini memberikan gambaran umum tentang kendala seperti peraturan yang berlaku, fungsi audit, persyaratan keandalan, dan sebagainya.

7. Asumsi dan dependensi

Bagian ini memberikan daftar asumsi dan faktor yang mempengaruhi persyaratan sebagaimana tercantum dalam dokumen ini.

8. Pengelompokan Persyaratan

Bagian ini menentukan persyaratan yang dapat ditunda hingga rilis versi sistem yang akan datang atau mesti disegerakan.

9. Persyaratan khusus

Bagian ini menginformasikan semua persyaratan secara detail sehingga para perancang dapat merancang sistem sesuai dengan keinginan pengguna. Persyaratan termasuk deskripsi setiap input dan output dari sistem dan fungsi yang dilakukan sebagai tanggapan terhadap masukan yang diberikan.

10. Antar muka eksternal

Bagian ini menginformasikan tentang adanya antarmuka perangkat lunak dengan sistem lain, yang dapat mencakup antarmuka dengan sistem operasi dan sebagainya. Antarmuka eksternal juga menentukan interaksi perangkat lunak dengan pengguna, perangkat keras, atau perangkat lunak lainnya. Karakteristik setiap antarmuka pengguna dari produk perangkat lunak ditentukan dalam SKPL. Untuk antarmuka perangkat keras, SKPL menentukan karakteristik logis dari setiap antarmuka di antara perangkat lunak dan komponen perangkat keras. Jika perangkat lunak akan dieksekusi pada perangkat keras yang ada, maka karakteristik seperti pembatasan memori juga ditentukan.

11. Fungsi

Bagian ini menginformasikan kemampuan fungsional sistem. Untuk setiap kebutuhan fungsional, penerimaan dan pemrosesan input untuk menghasilkan output ditentukan. Ini termasuk pemeriksaan validitas pada input, urutan operasi yang tepat, hubungan input ke output, dan sebagainya.

12. Persyaratan unjuk kerja

Bagian ini menentukan batasan kinerja sistem perangkat lunak. Persyaratan kinerja terdiri dari dua jenis: persyaratan statis dan

persyaratan dinamis. Persyaratan statis (juga dikenal sebagai persyaratan kapasitas) tidak menghendaki adanya kendala pada sistem. Ini termasuk persyaratan seperti jumlah terminal dan pengguna yang dilayani. Persyaratan dinamis menentukan kendala pada pelaksanaan perilaku sistem, yang meliputi waktu respon (waktu antara awal dan akhir operasi dalam kondisi tertentu) dan *throughput* (jumlah total pekerjaan yang dilakukan dalam waktu tertentu).

13. Basis data logis persyaratan

Bagian ini menginformasikan persyaratan logis untuk disimpan dalam database. Dalam hal ini termasuk jenis informasi yang digunakan, frekuensi penggunaan, entitas data dan hubungan di antara mereka, dan seterusnya.

14. Batasan perancangan

Bagian ini menginformasikan semua kendala desain yang disebabkan oleh standar, keterbatasan perangkat keras, dan sebagainya. Pemenuhan standar menentukan persyaratan untuk sistem, yang sesuai dengan standar yang ditentukan. Standar-standar ini dapat mencakup prosedur pelaporan dan formatnya. Batasan perangkat keras menyiratkan bahwa perangkat lunak dapat beroperasi pada perangkat keras yang sudah ada atau beberapa perangkat keras yang ditentukan sebelumnya. Hal ini dapat menyebabkan keharusan adanya pembatasan saat mengembangkan desain perangkat lunak. Keterbatasan perangkat keras termasuk konfigurasi perangkat keras dari mesin dan sistem operasi yang akan digunakan.

15. Atribut sistem perangkat lunak

Bagian ini menyediakan informasi terkait atribut seperti keandalan, ketersediaan, pemeliharaan dan portabilitas. Sangat penting untuk menggambarkan semua atribut ini untuk memverifikasi bahwa hal tersebut dapat dicapai dalam sistem akhir.

16. Organisasi persyaratan yang spesifik

Bagian ini menginformasikan persyaratan sehingga mereka dapat disusun secara terorganisir dengan baik untuk pemahaman yang

optimal. Persyaratan dapat diatur berdasarkan mode operasi, kelas pengguna, objek, fitur, respons, dan hirarki fungsional.

17. Perubahan proses manajemen

Bagian ini menginformasikan perubahan proses manajemen untuk mengidentifikasi, mengevaluasi, dan memperbarui SKPL untuk mencerminkan perubahan dalam lingkup dan persyaratan proyek.

18. Persetujuan dokumen

Bagian ini memberikan informasi tentang pemberi persetujuan terhadap dokumen SKPL dengan rincian seperti nama pemberi izin, tanda tangan, tanggal, dan seterusnya.

19. Dukungan informasi

Bagian ini memberikan informasi seperti daftar isi, indeks, dan sebagainya. Ini diperlukan terutama ketika SKPL dipersiapkan untuk proyek besar dan kompleks.

Review Persyaratan Perangkat Lunak

Berhubung karena spesifikasi persyaratan secara formal ada di pikiran manusia, maka validasi persyaratan harus selalu melibatkan klien dan pengguna. Review terhadap kebutuhan, di mana SKPL secara hati-hati ditinjau oleh sekelompok orang termasuk perwakilan klien dan pengguna, adalah metode validasi yang paling umum dilakukan.

Ulasan dan komentar dapat digunakan sepanjang proses pengembangan perangkat lunak untuk jaminan kualitas dan pengumpulan data persyaratan. Tinjauan persyaratan adalah tinjauan oleh sekelompok orang untuk menemukan kesalahan dan menunjukkan hal-hal lain yang menjadi perhatian dalam spesifikasi persyaratan sistem. Grup peninjau harus mencakup penulis dokumen persyaratan, seseorang yang memahami kebutuhan klien, orang dari tim desain, dan orang yang bertanggung jawab untuk memelihara dokumen persyaratan. Ini juga merupakan praktik yang baik untuk melibatkan beberapa orang yang tidak terlibat langsung dengan pengembangan produk seperti ahli kualitas perangkat lunak.

Salah satu cara untuk mengatur pertemuan untuk melakukan peninjauan adalah meminta setiap peserta membahas persyaratan sebelum pertemuan dan menandai item yang mereka ragukan atau ada hal yang dirasa perlu diklarifikasi lebih lanjut. Daftar periksa (cek) dapat sangat berguna dalam mengidentifikasi item-item tersebut. Dalam pertemuan tersebut setiap peserta menelusuri daftar potensi kerusakan yang telah ia temukan.

Ketika anggota mengajukan pertanyaan, analis persyaratan (yang merupakan penulis dokumen spesifikasi persyaratan) memberikan klarifikasi jika tidak ada kesalahan atau menyetujui bilamana adanya kesalahan. Atau, rapat dapat dimulai dengan analis yang menjelaskan masing-masing persyaratan dalam dokumen. Para peserta mengajukan pertanyaan, berbagi keraguan, atau mencari klarifikasi. Daftar periksa sering digunakan dalam ulasan untuk memfokuskan upaya tinjauan dan untuk memastikan bahwa tidak ada sumber kesalahan utama yang diabaikan oleh pengulas. Daftar periksa yang baik biasanya akan tergantung pada proyek. Berikut contoh-contohnya:

- Apakah semua sumber daya perangkat keras ditentukan?
- Sudahkah waktu respons fungsi ditentukan?
- Sudahkah semua perangkat keras, perangkat lunak eksternal, dan antarmuka data telah ditentukan?
- Apakah semua fungsi yang diperlukan oleh klien telah ditentukan.
- Apakah setiap persyaratan dapat diuji?
- Apakah keadaan awal sistem didefinisikan?
- Apakah tanggapan terhadap kondisi luar biasa ditentukan?
- Apakah persyaratan mengandung batasan yang dapat dikontrol oleh perancang?
- Apakah kemungkinan modifikasi di masa mendatang ditentukan?

Soal

1. Apa yang dimaksudkan dengan rekayasa kebutuhan dalam kegiatan pengembangan perangkat lunak ?
2. Apakah tujuan dilakukannya kegiatan rekayasa kebutuhan perangkat lunak ?
3. Uraikan tahap yang dilakukan dalam proses rekayasa kebutuhan.
4. Apa sajian instrumen yang dapat digunakan dalam mendapatkan data-data/informasi yang terkait dengan persyaratan perangkat lunak?
5. Menurut anda, instrument apakah yang paling utama digunakan dalam mendapatkan data-data/informasi tentang persyaratan perangkat lunak ? Mengapa demikian ? Jelaskan!
6. Jelaskan maksud dari persyaratan yang bersifat fungsional.
7. Jelaskan maksud dari persyaratan yang bersifat non-fungsional.
8. Kasus: Anda secara berkelompok (maks. 5 orang) diminta untuk mendapatkan suatu organisasi/unit dalam organisasi sebagai contoh. Rencanakan jenis software yang akan dikembangkan. Selanjutnya lakukanlah kegiatan pengumpulan data persyaratan dengan menggunakan, setidaknya 3 instrumen yang ada. Pada akhir kegiatan, buatlah dokumen yang berisikan data-data/informasi terkait dengan kebutuhan perangkat lunak (SKPL) yang akan dikembangkan tersebut (*functional and non-functional requirements*).

BAB VI

DESAIN PERANGKAT LUNAK

Tujuan :

1. Mempelajari teknik dasar dalam melakukan perancangan perangkat lunak
2. Mempelajari spesifikasi masing-masing teknik dalam membangun perangkat lunak.

Indikator :

1. Mahasiswa memahami tujuan dari proses desain perangkat lunak
2. Mahasiswa memahami berbagai teknik yang digunakan dalam proses desain perangkat lunak
3. Mahasiswa mampu menjelaskan mekanisme yang terjadi dalam berbagai tingkatan dalam desain perangkat lunak.

Materi :

Setelah dokumen persyaratan untuk perangkat lunak yang akan dikembangkan tersedia, maka tahap desain perangkat lunak dimulai. Sementara aktifitas yang terkait dengan spesifikasi persyaratan sepenuhnya berhubungan dengan masalah domain, desain adalah tahap pertama mengubah masalah menjadi solusi. Dalam tahap desain, semua persyaratan pengguna dan bisnis serta pertimbangan teknis secara bersama-sama digunakan untuk merumuskan produk atau sistem perangkat lunak.

Desain perangkat lunak adalah proses untuk mengubah data/informasi yang terdapat dalam dokumen persyaratan pengguna menjadi beberapa pola yang relevan, yang membantu programmer dalam melakukan pengkodean

dan implementasi perangkat lunak. Selama proses desain, model persyaratan perangkat lunak (data, fungsi, perilaku) diubah menjadi model desain yang mendeskripsikan detail struktur data, arsitektur sistem, antarmuka, dan komponen yang diperlukan untuk mengimplementasikan sistem. Prinsip dasar yang tersirat dalam definisi di atas adalah adanya kegiatan memindahkan konsentrasi dari domain masalah ke domain solusi. Hal ini adalah tindakan yang dilakukan untuk mencoba menentukan bagaimana memenuhi persyaratan yang disebutkan dalam SKPL.

Untuk memenuhi persyaratan pengguna, maka dibuatlah dokumen SKPL (Spesifikasi Kebutuhan Perangkat Lunak) sedangkan untuk pengkodean dan implementasi, terdapat uraian kebutuhan persyaratan yang lebih spesifik dan terperinci dalam bentuk terminologi perangkat lunak. Output dari proses ini dapat langsung digunakan ke dalam implementasi sesuai bahasa pemrograman yang digunakan.

Prinsip dalam desain perangkat lunak

1. Desain perangkat lunak harus sesuai dengan model analisis.

Seringkali elemen desain sesuai dengan banyak persyaratan. Oleh karena itu, kita harus tahu bagaimana model desain memenuhi semua persyaratan yang diwakili oleh model analisis.

2. Pilih paradigma pemrograman yang tepat.

Paradigma pemrograman menggambarkan struktur sistem perangkat lunak. Tergantung pada sifat dan jenis aplikasi, paradigma pemrograman yang berbeda seperti paradigma berorientasi prosedur, berorientasi objek, dan prototyping dapat digunakan. Paradigma tersebut harus dipilih dengan mempertimbangkan kendala seperti waktu, ketersediaan sumber daya dan sifat persyaratan pengguna.

3. Desain perangkat lunak harus seragam dan terintegrasi.

Desain perangkat lunak dianggap seragam dan terintegrasi, jika antarmuka didefinisikan dengan benar di antara komponen desain. Untuk ini, aturan, format, dan gaya ditetapkan sebelum tim desain

mulai merancang perangkat lunak.

4. Desain perangkat lunak harus fleksibel.

Desain perangkat lunak harus cukup fleksibel untuk menyesuaikan perubahan yang terjadi dengan mudah. Untuk mencapai fleksibilitas, konsep desain dasar seperti abstraksi, penyempurnaan, dan modularitas harus diterapkan secara efektif.

5. Perancangan perangkat lunak harus memastikan seminimal mungkin adanya kesalahan konseptual (semantik).

Tim desain harus memastikan bahwa kesalahan konseptual utama dari desain seperti ambiguitas dan ketidakkonsistenan dibahas sebelumnya sebelum menangani kesalahan sintaksis yang ada dalam model desain.

6. Desain perangkat lunak harus terstruktur untuk menurunkan dekomposisi secara bertahap.

Perangkat lunak harus dirancang untuk menangani perubahan dan keadaan yang tidak biasa, dan jika diperlukan untuk penghentian, itu harus melakukannya dengan cara yang tepat sehingga fungsionalitas perangkat lunak tidak terpengaruh.

7. Perancangan perangkat lunak harus mewakili korespondensi antara perangkat lunak dan masalah dunia nyata.

Desain perangkat lunak harus disusun sedemikian rupa sehingga selalu berkaitan dengan masalah dunia nyata.

8. Penggunaan kembali perangkat lunak

Komponen perangkat lunak harus dirancang sedemikian rupa sehingga dapat digunakan kembali secara efektif untuk meningkatkan produktifitas.

9. Merancang untuk hasil yang teruji.

Praktik umum yang telah diikuti adalah menjaga fase pengujian terpisah dari fase desain dan implementasi. Yaitu, pertama perangkat lunak ini dikembangkan (dirancang dan diimplementasikan) dan kemudian diserahkan kepada penguji yang kemudian menentukan apakah perangkat lunak tersebut sesuai

untuk distribusi dan penggunaan selanjutnya oleh pengguna. Namun, telah menjadi jelas bahwa proses pemisahan pengujian adalah cacat yang serius, seolah-olah ada jenis kesalahan desain atau implementasi yang ditemukan setelah implementasi, maka seluruh atau sebagian besar perangkat lunak membutuhkan untuk diperbaiki. Dengan demikian, para ahli pengujian harus dilibatkan dari tahap awal. Misalnya, mereka harus dilibatkan dengan analisis untuk menyiapkan perangkat uji untuk menentukan apakah persyaratan pengguna terpenuhi.

10. Prototyping.

Prototyping harus digunakan ketika persyaratan tidak sepenuhnya didefinisikan diawal. Pengguna berinteraksi dengan pengembang untuk memperluas dan menyempurnakan persyaratan seiring dengan perkembangan yang terjadi. Dengan menggunakan prototipe, 'mock-up' sistem yang cepat dapat dikembangkan. *Mock-up* ini dapat digunakan sebagai cara yang efektif untuk memberikan nuansa apa yang akan terlihat oleh sistem dan menunjukkan fungsi yang akan dimasukkan dalam sistem yang dikembangkan. *Prototyping* juga membantu mengurangi risiko perancangan perangkat lunak yang tidak sesuai dengan kebutuhan pelanggan.

Perhatikan bahwa prinsip desain sering dibatasi oleh konfigurasi perangkat keras yang ada, bahasa yang digunakan untuk implementasi, file yang ada dan struktur data, dan praktik organisasi yang ada. Juga, evolusi setiap desain perangkat lunak harus dirancang secara cermat untuk evaluasi, referensi, dan pemeliharaan di masa mendatang.

Tingkatan Dalam Desain Perangkat Lunak

Apabila dilihat dari hasil yang akan diperoleh, desain perangkat lunak menghasilkan tiga tingkatan luaran, antara lain:

1. Desain Arsitektur

Desain arsitektur adalah versi abstrak tertinggi dari sistem. Ia mengidentifikasi perangkat lunak sebagai suatu sistem dengan banyak komponen yang saling berinteraksi. Pada tingkatan ini, para desainer mendapatkan ide dari domain solusi yang diusulkan.

2. Desain Tingkat Tinggi

Desain tingkat tinggi mengabaikan konsep 'satu entitas-banyak komponen' dari desain arsitektur ke dalam tampilan sub-sistem dan modul yang kurang terabstraksi dan menggambarkan interaksinya satu sama lain. Desain tingkat tinggi berfokus pada bagaimana sistem beserta semua komponennya dapat diimplementasikan dalam bentuk modul.

3. Rancangan Terperinci

Rancangan terperinci berkaitan dengan bagian implementasi dari apa yang dilihat sebagai sistem dan sub-sistemnya dalam dua tahap desain sebelumnya. Ini lebih rinci dalam hal modul dan implementasinya. Aktivitas pada tingkatan ini adalah mendefinisikan struktur logis dari setiap modul dan antarmukanya untuk berkomunikasi dengan modul lain.

Modularitas

Modularitas adalah teknik untuk membagi sistem perangkat lunak menjadi beberapa modul diskrit dan independen, yang diharapkan mampu menjalankan tugas secara mandiri. Modul-modul ini dapat berfungsi sebagai konstruksi dasar untuk seluruh perangkat lunak. Desainer cenderung merancang modul sedemikian rupa sehingga mereka dapat dieksekusi dan atau dikompilasi secara terpisah dan mandiri.

Desain modular secara tidak sengaja mengikuti aturan strategi pemecahan masalah 'membagi dan memecahkan', ini karena ada banyak manfaat lain yang terkait dengan desain modular perangkat lunak.

Keuntungan modularitas :

- Komponen yang lebih kecil, sehingga lebih mudah di-*maintenance*
- Program dapat dibagi berdasarkan aspek fungsional

- Tingkat abstraksi yang diinginkan dapat dimasukkan ke dalam program
- Komponen dengan kohesi yang tinggi dapat digunakan kembali lagi
- Eksekusi serentak dapat dimungkinkan untuk dilakukan
- Diinginkan dari aspek keamanan



Gambar 6.1. Bentuk ilustrasi konsep modularitas

Gambar 6.1 menunjukkan ilustrasi bahwa modul-modul diibaratkan sebuah *puzzle* yang dapat membentuk berbagai pola ketika mereka ditempatkan di tempat yang berbeda. Modul dapat dipindahkan dan ditambah dengan bebas tanpa mempengaruhi fungsi modul lain tetapi mengubah bentuk sistem (fungsionalitas).

Konkurensi

Konkurensi berarti kumpulan teknik dan mekanisme yang memungkinkan perangkat lunak untuk melakukan beberapa tugas yang berbeda secara bersamaan, atau tampilnya secara bersamaan. Pada pola eksekusi berurutan, berarti bahwa instruksi yang dituliskan akan dijalankan satu per satu yang menyiratkan bahwa hanya satu bagian dari program yang diaktifkan pada waktu tertentu. Katakanlah, perangkat lunak memiliki banyak modul, maka hanya satu dari semua modul yang dapat ditemukan

aktif dari dari proses eksekusi.

Sedangkan dalam desain perangkat lunak, konkurensi diimplementasikan dengan membagi perangkat lunak menjadi beberapa unit eksekusi independen, seperti modul dan mengeksekusinya secara paralel. Dengan kata lain, konkurensi memberikan kemampuan kepada perangkat lunak untuk mengeksekusi lebih dari satu bagian kode secara paralel satu sama lain.

Penting bagi programmer dan desainer untuk mengenali modul-modul tersebut, yang dapat dibuat eksekusi paralel.

Contoh: Fitur pemeriksaan ejaan dalam pengolah kata adalah modul perangkat lunak, yang berjalan sepanjang aplikasi tersebut beroperasi.

Kopling dan Kohesi

Ketika suatu program perangkat lunak dimodulasikan, tugas-tugasnya dibagi menjadi beberapa modul berdasarkan beberapa karakteristik. Seperti yang kita ketahui, modul adalah set instruksi yang disatukan sedemikian rupa untuk menjalankan beberapa tugas. Meskipun demikian, hal itu dianggap sebagai entitas tunggal, tetapi dapat merujuk satu sama lain untuk bekerja sama. Ada beberapa ukuran di mana kualitas desain modul dan interaksinya di antara mereka dapat diukur. Langkah-langkah ini disebut kopling dan kohesi.

A. Kohesi

Kohesi adalah ukuran yang menentukan tingkat keandalan dalam elemen modul. Semakin besar kohesi, semakin baik desain programnya.

Ada tujuh jenis kohesi, yaitu:

- *Co-incident cohesion*. Ini adalah kohesi yang tidak terencana dan acak, yang mungkin merupakan hasil dari memecah program menjadi modul yang lebih kecil untuk kepentingan modularisasi. Karena tidak terencana, itu bisa membingungkan para

programmer dan umumnya tidak diterima.

- Kohesi logis. Ketika elemen yang dikategorikan secara logis dimasukkan ke dalam sebuah modul, ini disebut kohesi logis.
- *Emporal Cohesion*. Ketika elemen-elemen modul diatur sedemikian rupa sehingga mereka diproses pada titik waktu yang sama, itu disebut kohesi temporal.
- Kohesi prosedural. Ketika elemen-elemen modul dikelompokkan bersama, yang dieksekusi secara berurutan untuk melakukan suatu tugas, hal itu disebut kohesi prosedural.
- Kohesi komunikasional. Ketika elemen-elemen modul dikelompokkan bersama-sama, yang dieksekusi secara berurutan dan bekerja pada data (informasi) yang sama, ini disebut kohesi komunikasi.
- Kohesi sekuensial. Ketika elemen-elemen modul dikelompokkan karena output dari satu elemen berfungsi sebagai input ke yang lain dan seterusnya, ini disebut kohesi sekuensial.
- Kohesi Fungsional. Ini dianggap sebagai tingkat kohesi tertinggi, dan sangat diharapkan. Elemen-elemen modul dalam kohesi fungsional dikelompokkan karena semuanya berkontribusi pada satu fungsi yang terdefinisi dengan baik. Ini juga dapat digunakan kembali.

B. Kopling

Dua modul dianggap independen jika satu dapat berfungsi sepenuhnya tanpa kehadiran yang lain. Jelas, jika dua modul independen, mereka dapat dipecahkan dan dimodifikasi secara terpisah. Namun, semua modul dalam suatu sistem tidak dapat independen satu sama lain, karena mereka harus berinteraksi sehingga mereka bersama-sama menghasilkan perilaku eksternal yang diinginkan dari sistem.

Semakin banyak koneksi antar modul, semakin tergantung mereka

dalam arti bahwa lebih banyak pengetahuan tentang satu modul diperlukan untuk memahami atau menyelesaikan modul lainnya. Oleh karena itu, semakin sedikit dan semakin sederhana koneksi antar modul, semakin mudah untuk memahami satu tanpa memahami yang lain. Kopling antar modul adalah kekuatan interkoneksi antar modul atau ukuran independensi antar modul.

Untuk mengatasi dan memodifikasi modul secara terpisah, kami ingin agar modul tersebut digabungkan secara longgar dengan modul lainnya. Pilihan modul menentukan sambungan antar modul. Kopling adalah konsep abstrak dan tidak mudah diukur. Jadi, tidak ada rumus yang dapat diberikan untuk menentukan sambungan antara dua modul. Namun, beberapa faktor utama dapat diidentifikasi sebagai mempengaruhi pemasangan antar modul.

Di antara mereka yang paling penting adalah jenis koneksi antar modul, kompleksitas antarmuka, dan jenis aliran informasi antar modul. Coupling meningkat dengan kompleksitas dan ketidakjelasan antarmuka antar modul. Agar tetap rendah, kami ingin meminimalkan jumlah antarmuka per modul dan kompleksitas setiap antarmuka. Antarmuka modul digunakan untuk meneruskan informasi ke dan dari modul lain. Kompleksitas antarmuka adalah faktor lain yang mempengaruhi kopling.

Semakin kompleks setiap antarmuka, semakin tinggi derajat kopling. Jenis aliran informasi di sepanjang antarmuka adalah faktor utama yang mempengaruhi kopling ketiga. Ada dua jenis informasi yang dapat mengalir di sepanjang antarmuka: data atau kontrol, Melewati atau menerima informasi kontrol berarti bahwa tindakan modul akan bergantung pada informasi kontrol ini, yang membuatnya lebih sulit untuk memahami modul dan memberikan abstraksi. Transfer informasi data berarti bahwa modul dilewatkan sebagai input

beberapa data ke modul lain dan mendapat balasan beberapa data sebagai output.

Berdasarkan uraian di atas maka dapat disimpulkan bahwa kopling merupakan ukuran yang menentukan tingkat keterkaitan antar modul program. Ini memberitahu pada tingkat apa modul mengganggu atau berinteraksi satu sama lain. Semakin rendah kopling, semakin baik programnya.

Ada lima tingkat kopling, yaitu:

- Penggabungan konten. Saat modul dapat mengakses atau memodifikasi secara langsung atau merujuk pada konten modul lain, disebut penggabungan level konten.
- *Common coupling*. Ketika beberapa modul memiliki akses baca dan tulis ke beberapa data global, ini disebut penggabungan umum atau global.
- Kontrol kopling. Dua modul disebut kontrol-coupled jika salah satu dari mereka memutuskan fungsi dari modul lain atau mengubah aliran pelaksanaannya.
- *Stamp coupling*. Ketika beberapa modul berbagi struktur data umum dan bekerja pada bagian yang berbeda, itu disebut *stamp coupling*.
- *Data coupling*. Data coupling adalah ketika dua modul berinteraksi satu sama lain dengan cara melewati data (sebagai parameter). Jika sebuah modul melewati struktur data sebagai parameter, maka modul penerima harus menggunakan semua komponennya.

Verifikasi Desain

Output dari proses desain perangkat lunak adalah dokumentasi desain, kode semu, diagram logika rinci, diagram proses, dan deskripsi rinci dari semua persyaratan fungsional ataupun non-fungsional.

Fase berikutnya, yang merupakan implementasi perangkat lunak, tergantung pada semua output yang disebutkan di atas.

Hal ini kemudian menjadi perlu untuk memverifikasi hasil sebelum melanjutkan ke tahap berikutnya. Pada awal setiap kesalahan terdeteksi, semakin baik atau tidak terdeteksi hingga pengujian produk. Jika output fase desain dalam bentuk notasi formal, maka alat terkait untuk verifikasi harus digunakan jika tinjauan desain menyeluruh dapat digunakan untuk verifikasi dan validasi.

Dengan pendekatan verifikasi terstruktur, peninjau dapat mendeteksi cacat yang mungkin disebabkan oleh menghadapi beberapa kondisi. Tinjauan desain yang baik penting untuk perancangan, akurasi, dan kualitas perangkat lunak yang baik.

Konsep Desain Perangkat Lunak

Setiap proses perangkat lunak ditandai oleh konsep dasar bersama dengan praktik atau metode tertentu. Metode mewakili cara di mana konsep diterapkan. Sebagai teknologi baru menggantikan teknologi yang lebih tua, banyak perubahan terjadi dalam metode yang digunakan untuk menerapkan konsep-konsep untuk pengembangan perangkat lunak. Namun, konsep dasar yang menggarisbawahi proses desain perangkat lunak tetap sama, beberapa di antaranya dijelaskan di sini.

A. Abstraksi

Abstraksi mengacu pada alat desain yang kuat, yang memungkinkan perancang perangkat lunak untuk mempertimbangkan komponen pada tingkat abstrak, sambil mengabaikan detail implementasi komponen. IEEE mendefinisikan abstraksi sebagai pandangan tentang masalah yang mengekstraksi informasi penting yang relevan dengan tujuan tertentu dan mengabaikan sisa informasi tersebut. Konsep abstraksi dapat digunakan dalam dua cara: (1) sebagai proses dan (2) sebagai entitas. Sebagai suatu **proses**, ini mengacu pada mekanisme menyembunyikan detail yang tidak relevan dan hanya mewakili fitur penting dari suatu item sehingga seseorang dapat fokus pada hal-hal

penting pada suatu waktu. Sebagai **entitas**, ini mengacu pada model atau tampilan item.

Setiap langkah dalam proses perangkat lunak dilakukan melalui berbagai tingkat abstraksi. Pada level tertinggi, garis besar solusi untuk masalah disajikan sedangkan pada level bawah, solusi untuk masalah disajikan secara terperinci. Misalnya, dalam fase analisis persyaratan, solusi untuk masalah disajikan dengan menggunakan bahasa lingkungan di mana adanya masalah dan saat aktifitas dilanjutkan melalui proses perangkat lunak, tingkat abstraksi berkurang dan pada tingkat terendah, kode sumber perangkat lunak diproduksi.

Ada tiga mekanisme abstraksi yang umum digunakan dalam desain perangkat lunak, yaitu, **abstraksi fungsional**, **abstraksi data** dan **abstraksi kontrol**. Semua mekanisme ini memungkinkan kita untuk mengontrol kerumitan proses desain dengan melanjutkan dari model desain abstrak ke model desain kongkrit secara sistematis.

- Abstraksi fungsional.

Ini melibatkan penggunaan subprogram yang diparameterisasi. Abstraksi fungsional dapat digeneralisasi sebagai koleksi subprogram yang disebut sebagai 'grup'. Di dalam kelompok-kelompok ini terdapat rutinitas yang dapat terlihat atau disembunyikan. Rutinitas yang terlihat dapat digunakan di dalam kelompok manapun yang mengandungnya serta dalam kelompok lain, sedangkan rutin yang tersembunyi hanya dapat digunakan dalam kelompok yang memilikinya.

- Abstraksi data.

Aktifitas ini melibatkan fungsi menentukan data yang menjelaskan objek data. Sebagai contoh, jendela objek data mencakup seperangkat atribut (tipe window, dimensi window) yang menggambarkan objek window dengan jelas. Dalam mekanisme abstraksi ini, detail representasi dan manipulasi diabaikan.

- Abstraksi kontrol.

Bagian ini menyatakan efek yang diinginkan, tanpa menyatakan mekanisme kontrol yang tepat. Misalnya, jika terdapat pernyataan dalam bahasa pemrograman (seperti C dan C++) adalah abstraksi implementasi kode mesin, yang melibatkan instruksi bersyarat. Di tingkat desain arsitektur, mekanisme abstraksi ini memungkinkan spesifikasi subprogram berurutan tanpa memperhatikan detail implementasi yang tepat.

B. Arsitektur

Arsitektur perangkat lunak mengacu pada struktur sistem, yang terdiri dari berbagai komponen program / sistem, atribut (properti) dari komponen-komponen tersebut dan hubungan di antara mereka. Arsitektur perangkat lunak memungkinkan para ahli perangkat lunak untuk menganalisis desain perangkat lunak secara efisien. Selain itu, ini juga membantu mereka dalam pengambilan keputusan dan penanganan risiko. Arsitektur perangkat lunak melakukan hal-hal sebagai berikut:

- Memberikan wawasan kepada semua pemangku kepentingan yang tertarik sehingga memungkinkan mereka untuk berkomunikasi satu sama lain
- Menyoroti keputusan desain awal, yang memiliki dampak besar pada kegiatan rekayasa perangkat lunak (seperti pengkodean dan pengujian) yang mengikuti fase desain
- Menciptakan model-model intelektual tentang bagaimana sistem diorganisasikan ke dalam komponen-komponen dan bagaimana komponen-komponen ini berinteraksi satu sama lain.

Saat ini, arsitektur perangkat lunak direpresentasikan secara informal dan tidak terencana. Meskipun konsep arsitektur sering diwakili dalam infrastruktur (untuk mendukung gaya arsitektur tertentu) dan tahap awal konfigurasi sistem, kurangnya karakterisasi arsitektur independen yang eksplisit membatasi keunggulan konsep desain ini dalam skenario

ini. Perhatikan bahwa arsitektur perangkat lunak terdiri dari dua elemen model desain, yaitu desain data dan desain arsitektur.

C. Pola

Pola memberikan deskripsi solusi untuk masalah desain berulang dari beberapa domain tertentu sedemikian rupa sehingga solusi dapat digunakan berulang kali. Tujuan dari masing-masing pola adalah untuk memberikan wawasan kepada desainer yang dapat menentukan hal-hal berikut:

- [1]. Apakah polanya dapat digunakan kembali
- [2]. Apakah polanya berlaku untuk proyek saat ini
- [3]. Apakah polanya dapat digunakan untuk mengembangkan pola desain yang serupa tetapi secara fungsional atau struktural berbeda.

Jenis Pola Desain

Ahli desain perangkat lunak dapat menggunakan pola desain selama seluruh proses desain perangkat lunak. Ketika model analisis dikembangkan, perancang dapat memeriksa deskripsi masalah pada berbagai tingkat abstraksi untuk menentukan apakah itu sesuai dengan satu atau lebih dari jenis pola desain berikut ini:

- [1]. Pola arsitektur.

Pola ini adalah strategi tingkat tinggi yang merujuk pada keseluruhan struktur dan organisasi sistem perangkat lunak. Artinya, mereka mendefinisikan unsur-unsur sistem perangkat lunak seperti subsistem, komponen, kelas, dll. Selain itu, mereka juga menunjukkan hubungan antara unsur-unsur bersama dengan aturan dan pedoman untuk menentukan hubungan ini. Perhatikan bahwa pola arsitektur sering dianggap setara dengan arsitektur perangkat lunak.

- [2]. Pola desain.

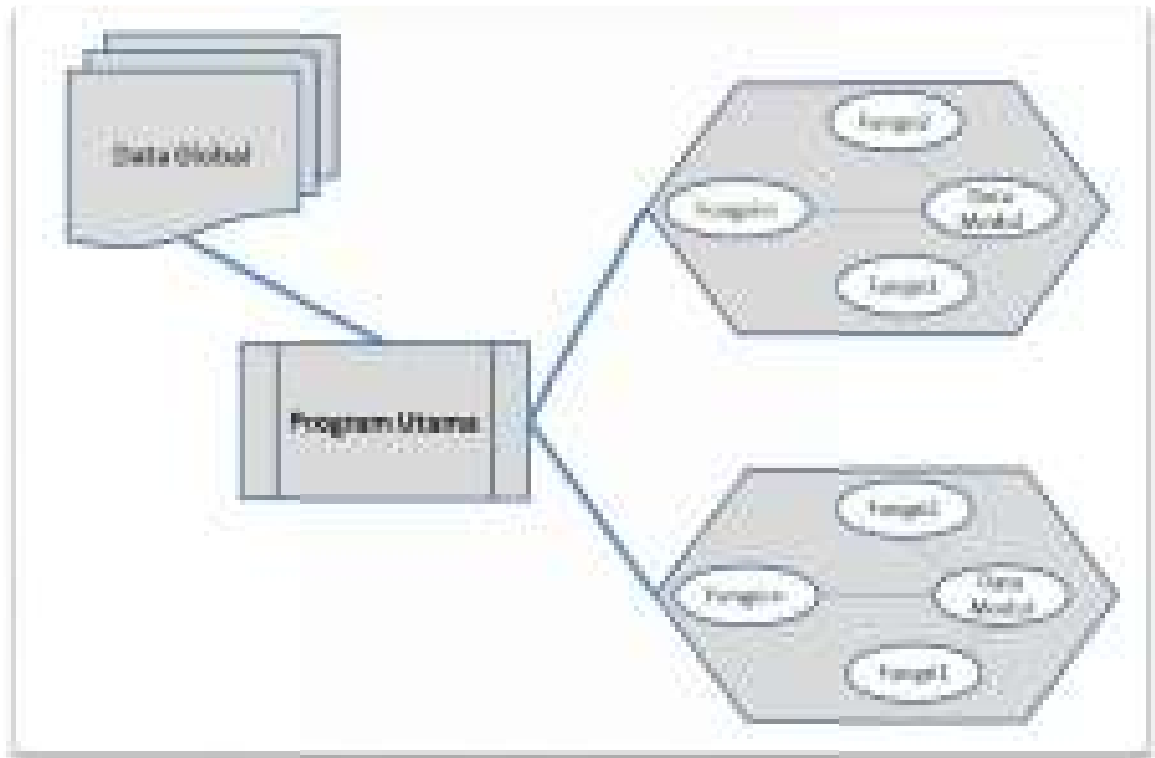
Pola ini adalah strategi tingkat menengah yang digunakan untuk menyelesaikan masalah desain. Mereka menyediakan sarana untuk penyempurnaan elemen (seperti yang didefinisikan oleh pola arsitektur) dari sistem perangkat lunak atau hubungan di antara mereka. Elemen desain spesifik seperti hubungan antara komponen atau mekanisme yang mempengaruhi interaksi komponen-ke-komponen ditangani oleh pola desain. Perhatikan bahwa pola desain sering dianggap setara dengan komponen perangkat lunak.

[3]. Idiom.

Pola-pola ini adalah pola tingkat rendah, yang spesifik untuk bahasa pemrograman. Mereka menggambarkan implementasi komponen perangkat lunak, metode yang digunakan untuk interaksi antar komponen perangkat lunak, dll., dalam bahasa pemrograman tertentu. Perhatikan bahwa idiom sering disebut sebagai pola pengkodean.

D. Modularitas

Modularitas dicapai dengan membagi perangkat lunak menjadi komponen-komponen yang unik bernama dan dialamatkan, yang juga dikenal sebagai modul. Sistem yang kompleks (program besar) dipartisi ke dalam satu set modul diskrit sedemikian rupa sehingga setiap modul dapat dikembangkan secara independen dari modul lain. Setelah mengembangkan modul, mereka terintegrasi bersama untuk memenuhi persyaratan perangkat lunak. Perhatikan bahwa semakin besar jumlah modul yang dibagi sistem, semakin besar upaya yang diperlukan untuk mengintegrasikan modul.



Gambar 6.2: Model kinerja modulatitas

Modularisasi desain membantu merencanakan pengembangan dengan cara yang lebih efektif, mengakomodasi perubahan dengan mudah, melakukan pengujian dan debugging secara efektif dan efisien, dan melakukan pekerjaan pemeliharaan tanpa mempengaruhi fungsi perangkat lunak.

E. Menyembunyikan Informasi (*Information Hiding*)

Modul seharusnya ditentukan dan dirancang sedemikian rupa sehingga struktur data dan detail pemrosesan dari satu modul tidak dapat diakses oleh modul lain. Mereka hanya menyampaikan informasi sebanyak itu satu sama lain, yang diperlukan untuk menyelesaikan fungsi perangkat lunak. Cara menyembunyikan detail yang tidak perlu disebut sebagai menyembunyikan informasi (*information hiding*). IEEE mendefinisikan informasi yang disembunyikan sebagai teknik enkapsulasi keputusan desain perangkat lunak dalam modul sedemikian rupa sehingga antarmuka modul mengungkapkan sesedikit

mungkin tentang cara kerja modul. Dengan demikian setiap modul adalah 'kotak hitam' untuk modul-modul lain dalam sistem.

Penyembunyian informasi sangat bermanfaat ketika modifikasi diperlukan selama tahap pengujian dan pemeliharaan. Beberapa keuntungan yang terkait dengan penyembunyian informasi tercantum di bawah ini:

- Menghasilkan kopling yang rendah
- Menekankan komunikasi melalui antar muka yang terkendali
- Mengurangi kemungkinan efek samping
- Membatasi efek perubahan pada satu komponen pada komponen lainnya
- Menghasilkan perangkat lunak berkualitas lebih tinggi.

F. Penyempurnaan bertahap

Penyempurnaan bertahap adalah strategi desain top-down yang digunakan untuk mendekomposisi sistem dari abstraksi tingkat tinggi ke tingkat abstraksi yang lebih rinci (level bawah). Pada tingkat abstraksi tertinggi, fungsi atau informasi didefinisikan secara konseptual tanpa memberikan informasi apa pun tentang cara kerja fungsi atau struktur internal data. Ketika kita melanjutkan ke tingkat abstraksi yang lebih rendah, semakin banyak detail tersedia.

Perancang perangkat lunak memulai proses penyempurnaan bertahap dengan membuat urutan komposisi untuk sistem yang dirancang. Setiap komposisi lebih rinci daripada yang sebelumnya dan mengandung lebih banyak komponen dan interaksi. Komposisi sebelumnya mewakili interaksi yang signifikan dalam sistem, sedangkan komposisi selanjutnya menunjukkan secara rinci bagaimana interaksi ini dicapai.

Untuk memiliki pemahaman yang jelas tentang konsep ini, mari kita perhatikan contoh penyempurnaan bertahap. Setiap program komputer terdiri dari input, proses, dan output.

1. *INPUT*

- *Dapatkan nama pengguna (string) melalui prompt.*
- *Dapatkan nilai pengguna (bilangan bulat dari 0 hingga 100) melalui prompt dan validasi.*

2. *PROSES*

3. *OUTPUT*

Ini adalah langkah pertama dalam penyempurnaan. Fase input dapat disempurnakan lebih lanjut seperti berikut ini.

1. *INPUT*

Dapatkan nama pengguna melalui prompt.

Dapatkan nilai pengguna melalui prompt.

While (grade tidak valid)

Tanya lagi:

2. *PROSES*

3. *OUTPUT*

Catatan: Penyempurnaan bertahap dapat juga dilakukan untuk fase PROSES dan OUTPUT.

G. Refactoring

Refactoring adalah kegiatan desain penting yang mengurangi kompleksitas desain modul untuk menjaga perilaku atau fungsinya agar tidak berubah. *Refactoring* dapat didefinisikan sebagai proses memodifikasi sistem perangkat lunak untuk meningkatkan struktur internal desain tanpa mengubah perilaku eksternalnya. Selama proses *refactoring*, desain yang ada diperiksa untuk semua jenis cacat seperti redundansi, algoritma yang dibangun dengan buruk dan struktur data, dll. Tujuannya adalah untuk meningkatkan kualitas desain. Misalnya, model desain mungkin menghasilkan komponen yang menunjukkan kohesi rendah (seperti komponen melakukan empat fungsi yang memiliki hubungan terbatas satu sama lain).

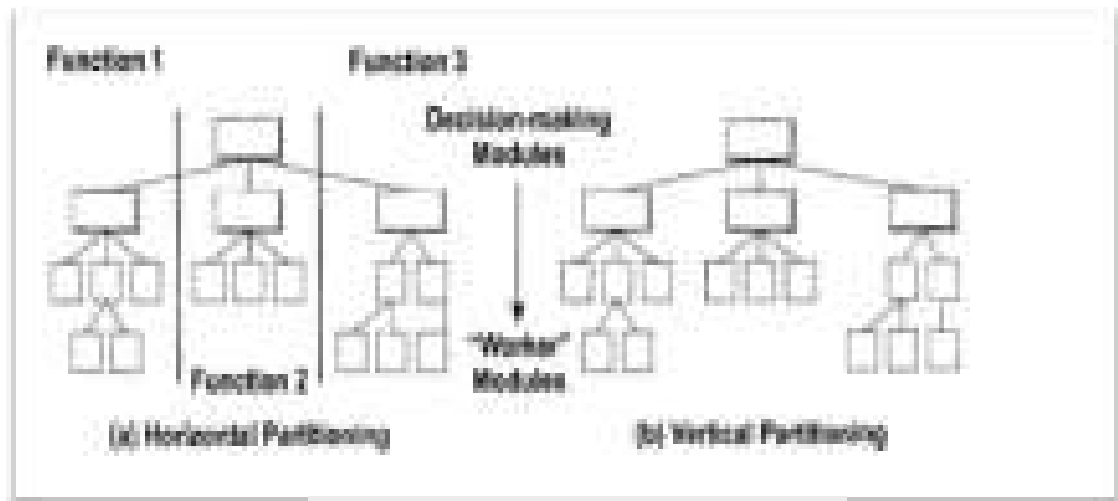
Perancang perangkat lunak dapat memutuskan untuk mengubah komponen menjadi empat komponen yang berbeda, masing-masing menunjukkan kohesi yang tinggi. Ini mengarah pada integrasi yang lebih mudah, pengujian, dan pemeliharaan komponen perangkat lunak.

H. Partisi Struktural

Ketika gaya arsitektur suatu desain mengikuti sifat hirarkis, struktur program dapat dipartisi baik secara **horizontal** maupun **vertikal**. Dalam partisi horisontal, modul kontrol digunakan untuk berkomunikasi antar fungsi dan menjalankan fungsi. Partisi struktural memberikan manfaat berikut:

- Pengujian dan pemeliharaan perangkat lunak menjadi lebih mudah.
- Dampak negatifnya menyebar secara perlahan.
- Perangkat lunak ini dapat diperluas dengan mudah.

Selain kelebihan ini, partisi horizontal juga memiliki beberapa kelemahan. Ia membutuhkan untuk melewati lebih banyak data di antarmuka modul, yang membuat aliran kontrol masalah lebih kompleks. Ini biasanya terjadi dalam kasus di mana data bergerak cepat dari satu fungsi ke fungsi lainnya.



Gambar 6.3: Partisi Horizontal dan Vertikal

Dalam partisi vertikal, fungsionalitas didistribusikan di antara modul dengan cara *top-down*. Modul di tingkat atas yang disebut modul kontrol melakukan pengambilan keputusan dan melakukan sedikit pemrosesan sedangkan modul di tingkat rendah yang disebut modul pekerja melakukan semua tugas input, komputasi dan keluaran.

I. Konkurensi

Komputer memiliki sumber daya yang terbatas dan harus digunakan seefisien mungkin. Untuk memanfaatkan sumber daya ini secara efisien, banyak tugas harus dijalankan bersamaan. Persyaratan ini menjadikan konkurensi sebagai salah satu konsep utama desain perangkat lunak. Setiap sistem harus dirancang untuk memungkinkan beberapa proses dijalankan secara bersamaan, kapan pun memungkinkan. Misalnya, jika proses saat ini menunggu beberapa peristiwa terjadi, sistem harus menjalankan beberapa proses lain dalam waktu yang bersamaan.

Namun, eksekusi bersamaan dari beberapa proses terkadang dapat menghasilkan situasi yang tidak diinginkan seperti keadaan tidak konsisten, kebuntuan, dll. Misalnya, diketahui terdapat dua proses A dan B dan item data Q1 dengan nilai '200'. Selanjutnya, anggaplah A dan B dieksekusi secara bersamaan dan pertama A membaca nilai Q1 (yaitu '200') untuk menambahkan '100' ke dalamnya. Namun, sebelum pembaruan A nilai Q1, B membaca nilai Q1 (yang masih '200') untuk menambahkan '50' untuk itu. Dalam situasi ini, apakah A atau B pertama memperbarui nilai Q1, nilai pasti akan salah dan menghasilkan kondisi sistem yang tidak konsisten. Ini karena tindakan A dan B tidak disinkronkan satu sama lain. Dengan demikian, sistem harus mengontrol eksekusi konkuren dan menyinkronkan tindakan proses konkuren.

Salah satu cara untuk mencapai sinkronisasi adalah saling adanya pengecualian, yang memastikan bahwa dua proses bersamaan tidak

mengganggu tindakan satu sama lain. Untuk memastikan ini, pengecualian bersama dapat menggunakan teknik penguncian (*lock*). Dalam teknik ini, proses perlu mengunci item data untuk dibaca atau diperbarui. Item data dikunci oleh beberapa proses tidak dapat diakses oleh proses lain sampai tidak terkunci. Ini menyiratkan bahwa proses, yang perlu mengakses item data yang dikunci oleh beberapa proses lain, harus menunggu.

Soal

1. Jelaskan, apa yang dimaksud dengan desain perangkat lunak?
2. Untuk melaksanakan aktifitas perancangan perangkat lunak, hal-hal apa saja yang harus telah terpenuhi?
3. Berdasarkan hasil yang akan diperoleh, terdapat 3 (tiga) tingkatan aktifitas dalam desain perangkat lunak. Uraikan masing-masingnya dan berikan contoh.
4. Jelaskan pola kerja dari pendesainan dengan konsep modularitas.
5. Jelaskan hal-hal apa saja yang dihasilkan dari kegiatan desain perangkat lunak.

BAB VII

PERANGKAT LUNAK ANALISIS DAN TOOL DESAIN

Tujuan :

1. Mempelajari berbagai tool yang dapat digunakan sebagai alat bantu analisis dan desain perangkat lunak.
2. Mempelajari bagaimana memilih tool yang relevan untuk melaksanakan suatu aktifitas dalam kegiatan analisis dan perancangan perangkat lunak

Indikator keberhasilan :

1. Mahasiswa memahami aturan-aturan penggunaan simbol-simbol dalam tool-tool analisis dan desain perangkat lunak.
2. Mahasiswa mampu menerapkan tool-tool dengan benar pada kasus sederhana yang diberikan pengajar.
3. Mahasiswa mampu menyampaikan argumentasi dari hasil rancangan yang mereka buat dengan menggunakan salah satu tool tersebut.

Materi :

Analisis dan perancangan perangkat lunak mencakup semua kegiatan, yang membantu transformasi spesifikasi kebutuhan menjadi implementasi ke dalam suatu perangkat lunak. Spesifikasi persyaratan menentukan semua persyaratan (kebutuhan) fungsional dan non-fungsional dari perangkat lunak. Spesifikasi persyaratan ini terdapat dalam bentuk dokumen yang dapat dibaca dan dipahami oleh manusia.

Analisis dan perancangan perangkat lunak adalah tahap transisi, yang membantu persyaratan yang dapat dibaca manusia diubah menjadi kode

yang sesungguhnya. Berikut ini adalah tool-tool analisis dan desain yang dapat digunakan oleh para software desainer:

A. Data Flow Diagram

Data Flow Diagram (DFD) adalah representasi grafis dari aliran data dalam suatu sistem informasi. Ini mampu menggambarkan aliran data yang masuk, aliran data keluar, data yang disimpan, dan berbagai subproses data bergerak. DFD dibangun menggunakan simbol dan notasi standar untuk menggambarkan berbagai entitas dan hubungannya. DFD tidak menyebutkan apa pun tentang bagaimana data mengalir melalui sistem.

Ada perbedaan mencolok antara DFD dan Flowchart Program. Flowchart Program menggambarkan aliran kontrol dalam modul-modul program. DFD menggambarkan aliran data dalam sistem pada berbagai level. Itu tidak mengandung kontrol atau elemen percabangan.

Manfaat DFD

Data Flow Diagram (DFD) adalah alat pembuatan model yang memungkinkan profesional sistem untuk menggambarkan sistem sebagai suatu jaringan proses fungsional yang dihubungkan satu sama lain dengan alur data, baik secara manual maupun komputerisasi.

DFD ini adalah salah satu alat pembuatan model yang sering digunakan, khususnya bila fungsi-fungsi sistem merupakan bagian yang lebih penting dan kompleks dari pada data yang dimanipulasi oleh sistem. Dengan kata lain, bahwa DFD merupakan alat pembuatan model yang memberikan penekanan hanya pada aspek fungsionalitas dari sistem.

DFD ini merupakan alat perancangan sistem yang berorientasi pada alur data dengan konsep dekomposisi dapat digunakan untuk penggambaran analisa maupun rancangan sistem yang mudah dikomunikasikan oleh profesional sistem kepada pemakai maupun pembuat program.

Tipe-tipe DFD

Data Flow Diagram ada dalam dua bentuk, yaitu logical dan physical, di mana:

A. **Logical DFD.** Tipe DFD ini berkonsentrasi pada proses sistem, dan aliran data dalam sistem. Misalnya dalam sistem perangkat lunak perbankan, bagaimana data dipindahkan antar entitas yang berbeda. Berikut ini adalah hal-hal penting yang berkaitan dengan *Logical DFD*:

1. Adalah representasi grafik dari sebuah sistem yang menunjukkan proses-proses dalam sistem tersebut dan aliran-aliran data ke dalam dan ke luar dari proses-proses tersebut.
2. Kita menggunakan DFD logis untuk membuat dokumentasi sebuah sistem informasi karena DFD logis dapat mewakili logika tersebut, yaitu apa yang dilakukan oleh sistem tersebut, tanpa perlu menspesifikasi dimana, bagaimana, dan oleh siapa proses-proses dalam sistem tersebut dilakukan.
3. Keuntungan dari DFD logis dibandingkan dengan DFD fisik adalah dapat memusatkan perhatian pada fungsi-fungsi yang dilakukan sistem.

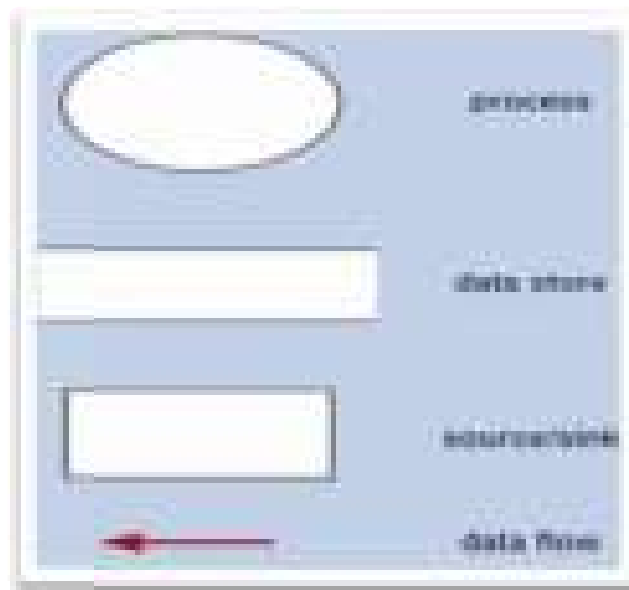
B. **Physical DFD.** Jenis DFD ini menunjukkan bagaimana aliran data benar-benar diimplementasikan dalam sistem. Ini lebih spesifik dan dekat dengan implementasi. Berikut ini adalah hal-hal penting yang berkaitan dengan *Physical DFD*:

1. Adalah representasi grafik dari sebuah sistem yang menunjukan entitas-entitas internal dan eksternal dari sistem tersebut, dan aliran-aliran data ke dalam dan keluar dari entitas-entitas tersebut.
2. Entitas-entitas internal adalah personel, tempat (sebuah bagian), atau mesin (misalnya, sebuah komputer) dalam sistem tersebut yang mentransformasikan data.

3. Maka DFD fisik tidak menunjukkan apa yang dilakukan, tetapi menunjukkan dimana, bagaimana, dan oleh siapa proses-proses dalam sebuah sistem dilakukan.

Komponen-komponen DFD

DFD terdiri atas sumber (*source/sink*), proses (*process*), penyimpanan (*data store*), dan aliran data (*data flow*) menggunakan rangkaian komponen berikut:



Gambar 7.1: Simbol-simbol DFD

- 1) Sumber (*Source/Sink*) adalah sumber dan tujuan dari data informasi. *Source* diwakili oleh persegi panjang dengan nama masing-masing.
- 2) Proses (*process*). Kegiatan dan tindakan yang dilakukan pada data diwakili oleh lingkaran atau segi empat persegi panjang.
- 3) Penyimpanan Data (*data store*). Ada dua varian penyimpanan data - ia dapat direpresentasikan sebagai persegi panjang dengan tidak adanya kedua sisi yang lebih kecil atau sebagai persegi panjang terbuka dengan hanya satu sisi yang hilang.

- 4) Aliran Data (*data flow*). Pergerakan data ditunjukkan oleh panah runcing. Pergerakan data ditunjukkan dari dasar panah sebagai sumbernya terhadap kepala panah sebagai tujuan.

Level-level DFD

- 1) Level 0. Level abstraksi tertinggi DFD dikenal sebagai Level 0 DFD, yang menggambarkan seluruh sistem informasi sebagai satu diagram yang menyembunyikan semua rincian yang mendasari. Level 0 DFD juga dikenal sebagai DFD level konteks.
- 2) Level 1. Level 0 DFD dipecah menjadi DFD Level 1 yang lebih spesifik. Level 1 DFD menggambarkan modul-modul dasar dalam sistem dan aliran data di antara berbagai modul. Level 1 DFD juga menyebutkan proses dasar dan sumber informasi.
- 3) Level 2. Pada level ini, DFD menunjukkan bagaimana data mengalir di dalam modul yang disebutkan di Level 1.

DFD tingkat yang lebih tinggi dapat diubah menjadi DFD tingkat yang lebih spesifik dengan tingkat pemahaman yang lebih dalam kecuali tingkat spesifikasi yang diinginkan telah tercapai.

Tips dalam menggambarkan DFD

1. Pilihlah kalimat yang tepat sehingga jelas dengan mudah mana proses yang didekomposisi atau tidak didekomposisi.
2. Nama proses harus terdiri dari kata kerja (*verb*) dan kata benda (*noun*).
3. Nama yang dipakai untuk proses, data store, dataflow harus konsisten.
4. Setiap level dekomposisi, aliran datanya harus konsisten dengan level sebelumnya.
5. Usahakan agar *external entity* pada setiap level konsisten peletakkannya (posisi penggambarannya).

6. Banyaknya proses yang disarankan pada setiap level tidak melebihi 5 (lima) proses. Melebihi itu sebaiknya diturunkan pada level berikutnya.
7. Dekomposisi berdasarkan kelompok data lebih disarankan (memudahkan aliran data ke storage yang sama)
8. Nama Proses yang bersifat umum hanya untuk proses yang masih akan didekomposisi
9. Pada Proses yang sudah tidak didekomposisi, nama proses dan nama datanya harus sudah spesifik (misalnya: mencetak laporan).
10. Aliran ke data storage harus melewati setidaknya satu proses, tidak boleh langsung dari *external entity*.
11. Aliran data untuk proses *report* harus berupa aliran keluar data keluar dari sebuah simbol proses. Akan ada aliran masuk jika perlu parameter untuk menghasilkan *report*.
12. Aliran data yang tidak ada data store-nya harus dicermati, apakah memang tidak mencerminkan *persisten entity* (perlu disimpan dalam file/tabel), yaitu kelak hanya akan menjadi variabel dalam program.

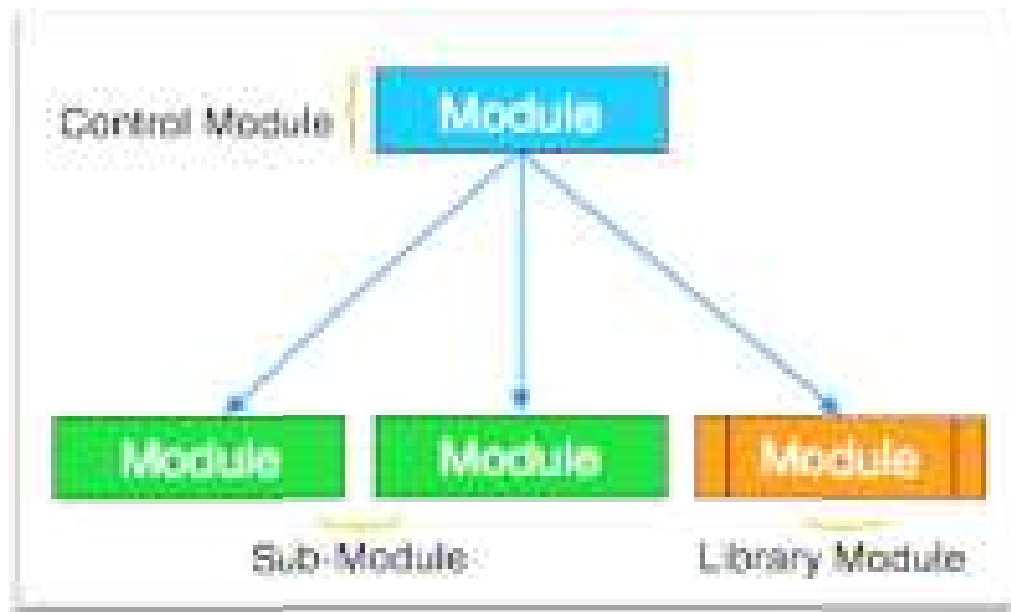
B. Structure Chart

Structure chart adalah bagan yang berasal dari Data Flow Diagram. Ia merepresentasikan sistem secara lebih detil daripada DFD. Di dalamnya terdapat aktifitas memecah seluruh sistem menjadi modul fungsional terendah, menjelaskan fungsi dan sub-fungsi dari setiap modul sistem ke detail yang lebih lengkap daripada DFD.

Structure Chart merupakan struktur hirarkis modul. Pada setiap lapisan, terdapat informasi tentang tugas khusus dilakukan.

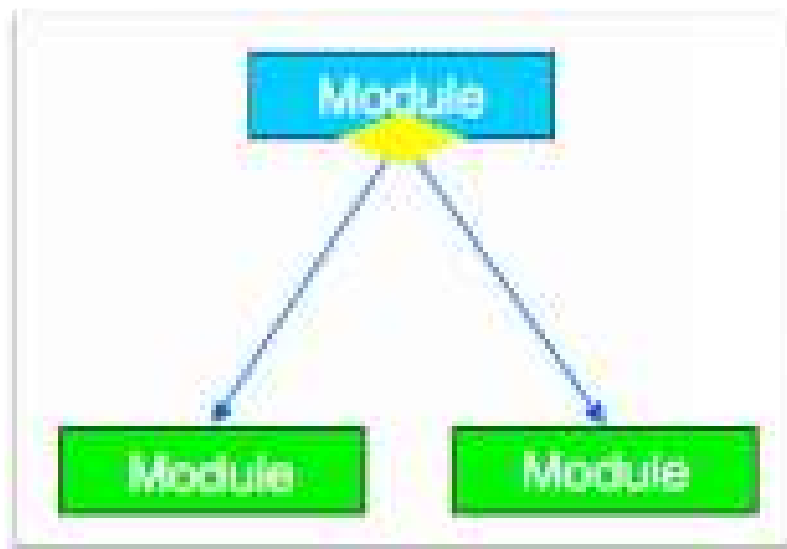
Berikut ini adalah symbol-simbol yang digunakan dalam *Structure chart*:

- 1) Module. Ia mewakili proses atau subrutin atau tugas. Modul kontrol akan dikirimkan ke lebih dari satu sub-modul. Modul *library* dapat digunakan kembali dan dapat dipanggil dari modul apa pun.



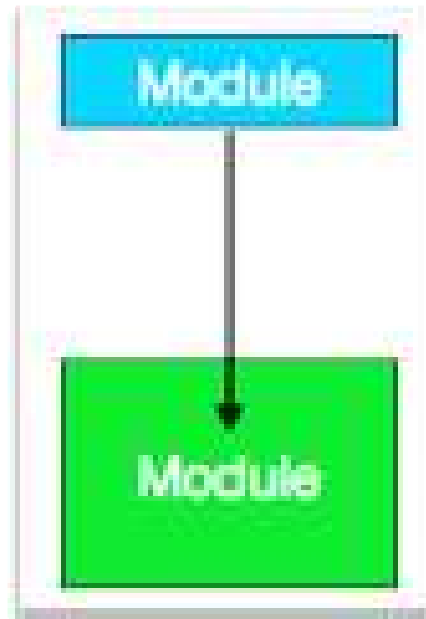
Gambar 7.2 : Elemen module

- 2) Kondisi. Ia diwakili oleh symbol berlian kecil di bagian bawah modul. Ini menggambarkan bahwa modul kontrol dapat memilih salah satu dari sub-rutin berdasarkan pada beberapa kondisi.



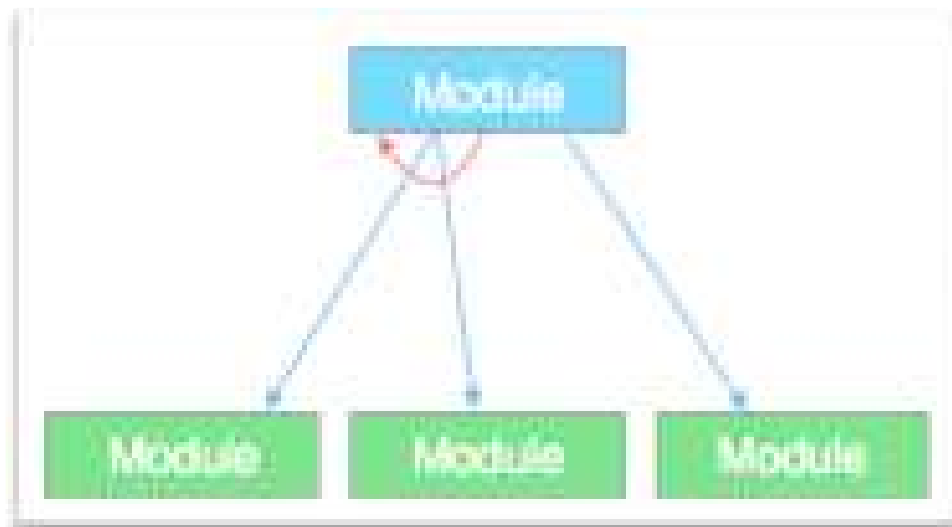
Gambar 7.3 : Elemen Kondisi

- 3) *Jump*. Simbol yang diperlihatkan di dalam modul untuk menggambarkan bahwa kontrol akan melompat di tengah sub-modul.



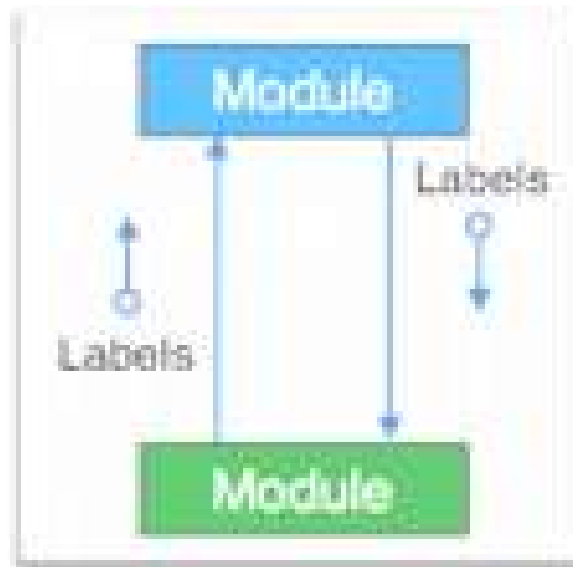
Gambar 7.4: Kondisi Jump

- 4) Loop. Disimbolkan dengan panah melengkung yang melambungkan lingkaran di modul. Semua sub-modul ditutupi oleh pelaksanaan pengulangan terus menerus pada modul.



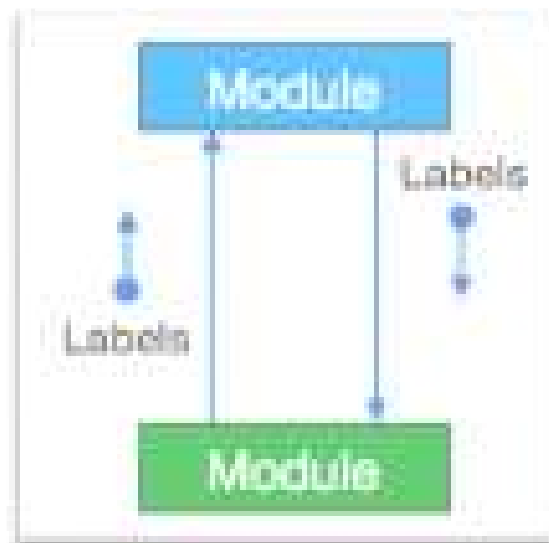
Gambar 7.5 : Aturan loop

- 5) Aliran data. Disimbolkan dengan anak panah yang diarahkan dengan lingkaran kosong di bagian akhir yang merepresentasikan aliran data.



Gambar 7.6: Pola aliran data

- 6) Kontrol Flow. Disimbolkan dengan anak panah yang diarahkan dengan lingkaran penuh di bagian akhir menunjukkan kontrol flow.



Gambar 7.7 : Kontrol Flow

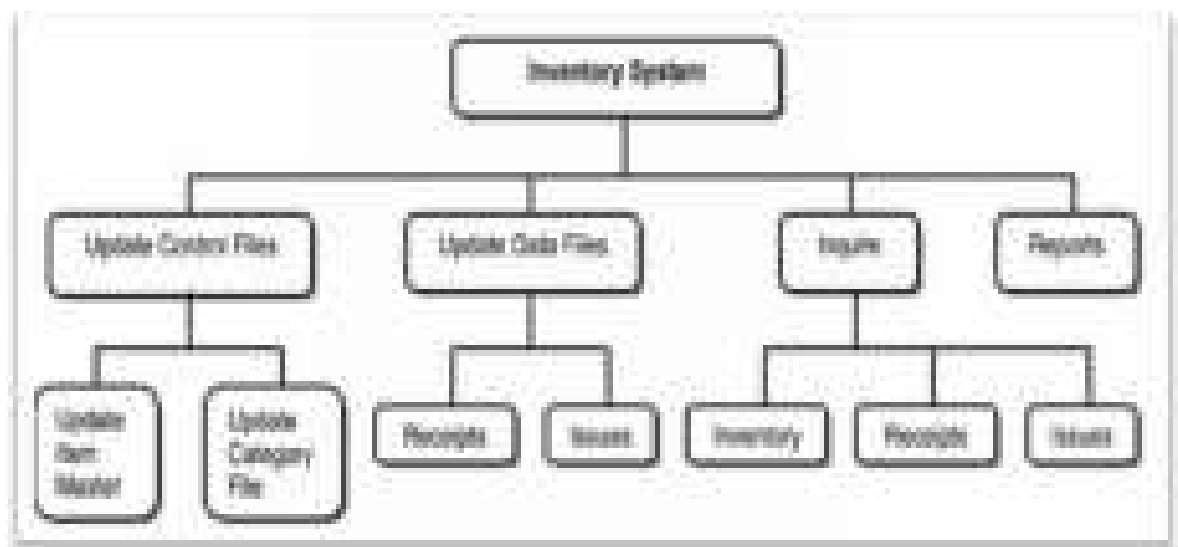
C. HIPO Diagram

Hierarchical Input Process Output (HIPO) diagram adalah kombinasi dari dua metode terorganisir untuk menganalisa sistem dan menyediakan

sarana dokumentasi. Model HIPO dikembangkan oleh IBM pada tahun 1970.

Diagram HIPO mewakili hirarki modul dalam sistem perangkat lunak. Analis menggunakan diagram HIPO untuk mendapatkan tampilan tingkat tinggi dari fungsi sistem. Ia mengurai fungsi menjadi sub-fungsi dengan cara hierarkis. HIPO menggambarkan fungsi-fungsi yang dilakukan oleh sistem.

Diagram HIPO baik untuk tujuan dokumentasi. Representasi grafisnya memudahkan desainer dan manajer untuk mendapatkan ide bergambar dari struktur sistem.



Gambar 7.8 : Bentuk HIPO Chart

Berbeda dengan diagram Input Process Output (IPO), yang menggambarkan aliran kontrol dan data dalam modul, HIPO tidak memberikan informasi apapun tentang aliran data atau aliran kontrol.

D. Struktur English

Kebanyakan programmer tidak menyadari gambaran menyeluruh sebuah perangkat lunak sehingga mereka hanya mengandalkan apa yang dikatakan oleh para manajernya. Hal ini adalah tanggung jawab pihak manajemen perangkat lunak yang lebih tinggi untuk memberikan informasi yang akurat kepada para programmer untuk mengembangkan kode yang akurat namun cepat.

Metode yang berbeda, yang menggunakan grafik atau diagram, kadang-kadang dapat diterjemahkan dengan cara yang berbeda oleh orang yang berbeda.

Oleh karena itu, analisis sistem dan perancang perangkat lunak menggunakan tool seperti Structured English. Hal ini tidak lain adalah deskripsi dari apa yang diperlukan untuk dikoding dan bagaimana cara mengkodennya. *Structured English* membantu programmer untuk menulis kode dengan bebas kesalahan. Di sini, baik *Structured English* dan *Pseudo-Code* mencoba untuk mengurangi kesenjangan pemahaman itu. *Structured English* menggunakan kata-kata bahasa Inggris sederhana dalam paradigma pemrograman terstruktur. Ini bukan kode pamungkas tetapi semacam deskripsi apa yang diperlukan untuk kode dan bagaimana cara mengkodennya. Berikut ini adalah beberapa kunci dalam pemrograman terstruktur:

If-then-else

Repeat-until

Sistem analisis menggunakan variabel dan nama data yang sama, yang disimpan dalam *Data Dictionary*, sehingga membuatnya lebih mudah untuk menulis dan memahami kode.

Contoh:

```

IF Customer has a Bank Account THEN
  IF Customer has no fact from previous account THEN
    Allow loan fully
  ELSE IF Management Approval is obtained THEN
    Allow loan fully
  ELSE
    Reject
  ENDIF
ENDIF
ELSE
  Reject
ENDIF

```

Kode yang ditulis dalam *Structured English* lebih seperti bahasa Inggris sehari-hari. Dengan demikian hal itu tidak dapat diimplementasikan secara langsung sebagai kode perangkat lunak. Bahasa Inggris terstruktur tidak bergantung pada bahasa pemrograman.

E. Pseudo-Code

Pseudo code ditulis lebih dekat dengan bahasa pemrograman. Ini dapat dianggap sebagai bahasa pemrograman tambahan, penuh komentar, dan deskripsi.

Pseudo code menghindari deklarasi variabel tetapi ditulis menggunakan beberapa konstruksi bahasa pemrograman yang sebenarnya, seperti C, Fortran, Pascal, dll.

Contoh:

```

Range r = [1, n]
i = first direction in p
while((i is a valid element) and (r contains more than one element))
{
    currdegree = number of directions in level
    if (currdegree = 1) then return r
    if (i commutes with another) {
        return Lastsection{r, num of commuting directions, currdegree}
    }
    else subdivide:-    r = Subsection(r, i, currdegree)
    i -> next direction in p
}
return r

```

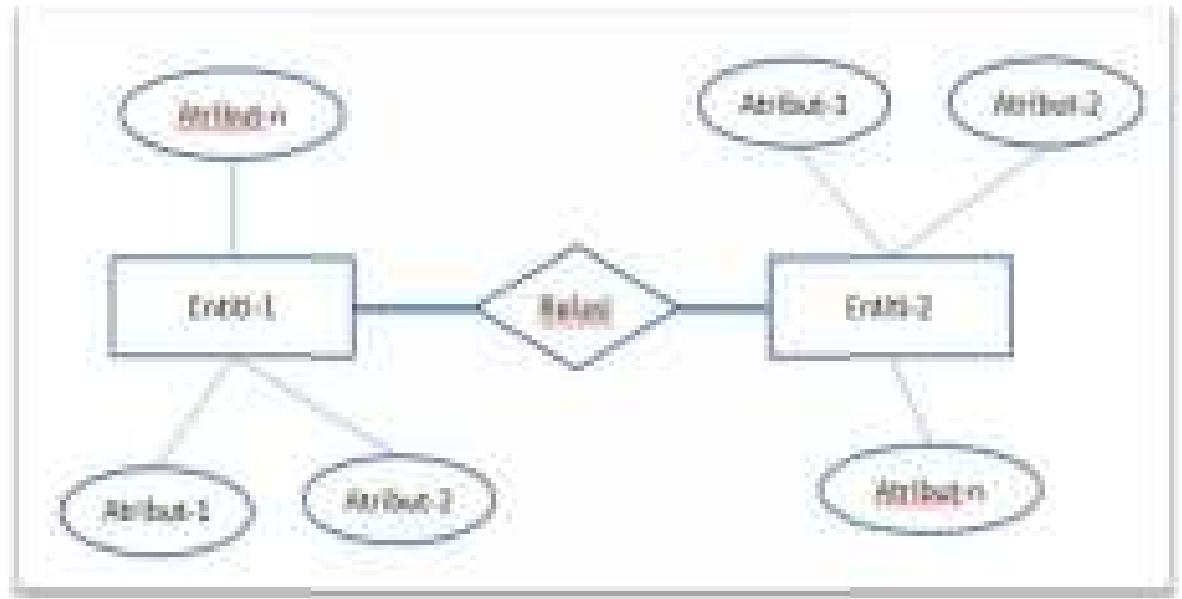
Pseudo code berisi lebih banyak detail pemrograman daripada Structured English. Ini menyediakan metode untuk melakukan tugas, seolah-olah komputer sedang mengeksekusi kode.

F. Entity Relationship Diagram Model

Model *Entity Relationship (ER)* adalah jenis pemodelan basis data berdasarkan fakta pada entitas dunia nyata dan hubungan di antara mereka. Kita dapat memetakan fakta dunia nyata ke dalam model basis data ER. Model ER menciptakan seperangkat entitas dengan atribut-atributnya, serangkaian kendala dan hubungan di antaranya.

Model ER paling baik digunakan untuk desain konseptual dari database.

Model ER dapat direpresentasikan sebagai berikut:



Gambar 7.9: Entity Relationship Diagram

- 1) Entiti - Entiti dalam Model ER adalah dunia nyata, yang memiliki beberapa properti yang disebut atribut. Setiap atribut ditentukan oleh set nilai yang sesuai, yang disebut domain.
Misalnya, Terdapat sebuah database mahasiswa. Di sini, seorang mahasiswa adalah sebuah entiti. Mahasiswa memiliki berbagai atribut seperti NIM, nama, umur dan tempat lahir, dll.
- 2) Relasi. Merupakan hubungan logis antara entiti disebut hubungan. Hubungan dipetakan dengan entiti dalam berbagai cara. Memetakan kardinalitas menentukan jumlah asosiasi antara dua entiti.

Bentuk pemetaan cardinalitas yang ada dalam ER Diagram adalah:

- a. *one to one*
- b. *one to many*
- c. *many to one*
- d. *many to many*

G. Data Dictionary (Kamus Data)

Kamus data adalah kumpulan informasi terpusat terkait data. Ia menyimpan arti dan asal data, hubungannya dengan data lain, format

data untuk penggunaan, dan sebagainya. Kamus data memiliki definisi yang ketat dari semua nama untuk memfasilitasi pengguna dan perancang perangkat lunak.

Kamus data sering dirujuk sebagai meta data repositori. Ini dibuat bersama dengan DFD (*Data Flow Diagram*) model program perangkat lunak dan diharapkan akan diperbarui setiap kali DFD diubah atau diperbarui.

Persyaratan Kamus Data

Data direferensikan melalui kamus data saat merancang dan mengimplementasikan perangkat lunak. Kamus data menghilangkan kemungkinan ambiguitas apa pun. Ini membantu menjaga pekerjaan programmer dan desainer agar selalu sinkron saat menggunakan referensi objek yang sama di mana-mana dalam program tersebut.

Kamus data menyediakan cara dokumentasi untuk sistem database lengkap di satu tempat. Validasi DFD dilakukan menggunakan kamus data.

Konten Kamus Data

Kamus data harus memuat informasi tentang hal-hal berikut:

- Data Flow
- Data Structure
- Data Elements
- Data Stores
- Data Processing

Aliran data yang digambarkan dengan menggunakan DFD seperti yang dipelajari sebelumnya dan diwakili dalam bentuk aljabar seperti yang simbol berikut ini.

Notasi	Keterangan
∞	Terdiri dari
∞	Dari atau And
[]	Pilihan opsional
[]	Anda? (Perulangan proses)
[]	Pilih salah satu pilihan yang ada
[]	Pemilihan pilihan didalam tanda []
∞	Konvensional atau notasi
∞	Field Kurat

Gambar 7.10: Notasi dalam kamus data

Sebagian besar pengembang perangkat lunak setuju bahwa perancangan basis data adalah langkah pertama untuk merancang perangkat lunak. Cara desainer menentukan tabel-tabel data sangat menentukan dalam keberhasilan pengembangan perangkat lunak. Setelah kegiatan mendesain tabel, kemudian dilanjutkan dengan membuat kamus data. Kamus data berfungsi sebagai dokumen yang menguraikan desain tabel-tabel yang dibuat, tipe data untuk setiap kolom, dan penjelasan singkat dari setiap bagian.

Soal

1. Uraikan setidaknya 5 alasan, mengapa diperlukan tool berupa perangkat lunak dalam aktifitas analisis dan desain?
2. Jelaskan aturan ringkas dalam menggambarkan DFD.
3. Jelaskan hasil apa yang diinginkan dari sebuah gambar diagram dengan menggunakan tool DFD?
4. Diagram HIPO mewakili hirarki modul dalam sistem perangkat lunak. Jelaskan maksudnya. Berikan contoh.
5. Jelaskan perbedaan antara *Structured English* dan *Pseudocode*.

BAB VIII

STRATEGI PERANCANGAN PERANGKAT LUNAK

Tujuan :

- e. Mempelajari beraneka strategi yang dapat digunakan dalam merancang suatu perangkat lunak.
- f. Mempelajari pendekatan-pendekatan yang dilakukan dalam merancang suatu perangkat lunak

Indikator keberhasilan :

- 1. Mahasiswa mampu menjelaskan keunggulan masing-masing strategi yang digunakan dalam merancang perangkat lunak.
- 2. Mahasiswa mampu menjelaskan masing-masing pendekatan yang digunakan dalam merancang suatu perangkat lunak.
- 3. Mahasiswa mampu menunjukkan contoh-contoh aplikasi dengan pendekatan perancangan yang digunakan.

Materi :

Desain perangkat lunak adalah proses untuk membuat konsep kebutuhan perangkat lunak menjadi implementasi perangkat lunak. Desain perangkat lunak mengambil persyaratan pengguna sebagai tantangan dan mencoba untuk menemukan solusi secara optimal. Sementara perangkat lunak sedang dikonseptualisasikan, sebuah rencana dicoret untuk menemukan desain terbaik yang mungkin untuk mengimplementasikan solusi yang dimaksudkan.

Strategi perancangan perangkat lunak adalah disiplin yang membantu perusahaan dalam memutuskan apa yang harus dibuat, mengapa membuatnya dan bagaimana berinovasi secara kontekstual, baik dengan

sifat segera maupun dalam jangka panjang. Proses ini melibatkan strategi desain dan interaksi antar hasil desain dan strategi bisnis. Strategi desain perangkat lunak lebih banyak focus pada pengelolaan kegiatan selama proses desain berlangsung. Organisasi kegiatan desain, baik itu direncanakan atau *ad-hoc* mencerminkan pendekatan desainer untuk menciptakan sebuah hasil desain. Selama proses pengembangan perangkat lunak adalah lumrah bahwa perancang perangkat lunak berpikir tentang daftar masalah desain dan tugas yang perlu dibahas dan ditangani serta hanya harus dilakukan dengan cara *ad-hoc*.

Ada beberapa cara/pendekatan sebagai bentuk strategi yang dilakukan dalam perancangan perangkat lunak:

A. Perancangan Terstruktur

Desain terstruktur adalah konseptualisasi masalah menjadi beberapa elemen solusi yang terorganisir dengan baik. Ini pada dasarnya berkaitan dengan desain solusi. Manfaat dari desain terstruktur adalah, memberikan pemahaman yang lebih baik tentang bagaimana masalah ini diselesaikan. Desain terstruktur juga membuat desainer lebih mudah berkonsentrasi pada masalah dengan lebih akurat.

Desain terstruktur sebagian besar didasarkan pada strategi 'membagi dan menaklukkan' di mana masalah dipecah menjadi beberapa masalah lebih kecil dan setiap masalah kecil diselesaikan secara terpisah sampai seluruh masalah diselesaikan.

Potongan-potongan kecil masalah diselesaikan dengan menggunakan modul solusi. Penekanan desain terstruktur bahwa modul-modul ini diatur dengan baik untuk mencapai solusi yang tepat.

Modul-modul ini disusun dalam hierarki. Mereka berkomunikasi satu sama lain. Desain terstruktur yang baik selalu mengikuti beberapa aturan untuk komunikasi di antara banyak modul, yaitu :

[1]. Kohesi.

Keberadaan kohesi diperlukan untuk menentukan seberapa dekat elemen-elemen suatu modul saling terkait. Kohesi suatu modul

menunjukkan betapa terikat eratnya elemen-elemen internal modul satu sama lain. Kohesi suatu modul memberi perancang gagasan tentang apakah elemen-elemen yang berbeda dari suatu modul menjadi satu dalam modul yang sama. Kohesi dan kopling jelas terkait. Biasanya semakin besar kohesi masing-masing modul dalam sistem, semakin rendah kopling antar modul. Ada beberapa tingkatan kohesi:

1. *Coincidental*
2. *Logical*
3. *Temporal*
4. *Procedural*
5. *Communicational*
6. *Sequential*
7. *Functional*

Coincidental adalah level terendah, dan fungsional adalah level tertinggi. Kohesi *Coincidental* terjadi ketika tidak ada hubungan yang bermakna antara unsur-unsur modul. Kohesi *Coincidental* dapat terjadi jika program yang ada dimodulasi dengan memotongnya menjadi beberapa bagian dan membuat modul bagian yang berbeda.

Modul memiliki kohesi *logical* jika ada beberapa hubungan logis antara elemen-elemen modul, dan elemen-elemen melakukan fungsi yang mengisi kelas logis yang sama. Contoh khas dari kohesi semacam ini adalah modul yang melakukan semua input atau semua output. Kohesi *temporal* sama dengan kohesi logis, kecuali bahwa unsur-unsurnya juga berkaitan dalam waktu dan dieksekusi bersama. Modul yang melakukan kegiatan seperti "inisialisasi", "pembersihan" dan "penghentian" biasanya terikat sementara.

Modul kohesi *prosedural* berisi elemen-elemen yang dimiliki oleh unit prosedural umum. Misalnya, *loop* atau urutan pernyataan keputusan dalam suatu modul dapat digabungkan untuk membentuk modul terpisah. Modul dengan kohesi *communicational* memiliki elemen yang

terkait dengan referensi ke input atau output data yang sama. Yaitu, dalam modul yang terikat secara komunikasi, elemen-elemennya bersama karena mereka beroperasi pada input atau output data yang sama.

Ketika elemen-elemen tersebut bersama-sama dalam modul karena output dari satu membentuk input ke yang lain, kita mendapatkan kohesi *sequential*. Kohesi *functional* adalah kohesi terkuat. Dalam modul yang terikat secara fungsional, semua elemen modul terkait dengan melakukan fungsi tunggal.

[2]. Coupling.

Dua modul dianggap independen jika satu dapat berfungsi sepenuhnya tanpa kehadiran yang lain. Jelas, jika dua modul independen, mereka dapat dipecahkan dan dimodifikasi secara terpisah. Namun, semua modul dalam suatu sistem tidak dapat independen satu sama lain, karena mereka harus berinteraksi sehingga mereka bersama-sama menghasilkan perilaku eksternal yang diinginkan dari sistem.

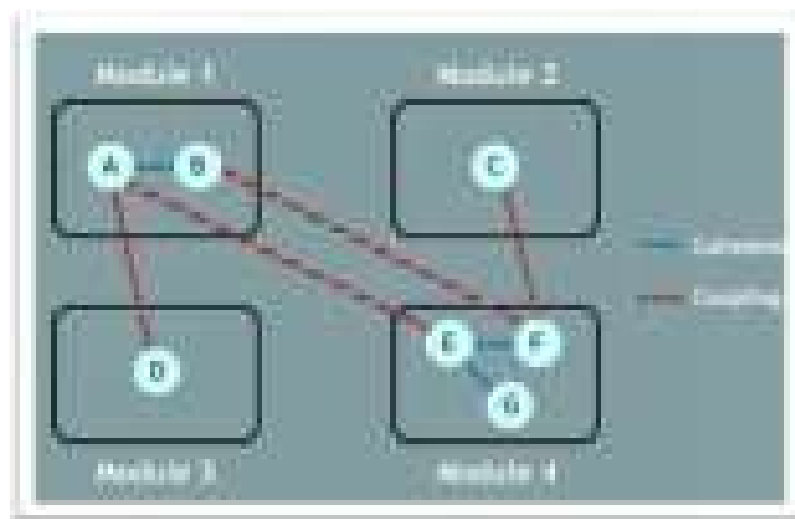
Semakin banyak koneksi antar modul, semakin tergantung mereka dalam arti bahwa lebih banyak pengetahuan tentang satu modul diperlukan untuk memahami atau menyelesaikan modul lainnya. Oleh karena itu, semakin sedikit dan semakin sederhana koneksi antar modul, semakin mudah untuk memahami satu modul tanpa harus memahami yang lain. Kopling antar modul adalah kekuatan interkoneksi antar modul atau ukuran independensi antar modul.

Untuk mengatasi dan memodifikasi modul secara terpisah, diinginkan agar modul tersebut digabungkan secara fleksibel dengan modul lainnya. Pilihan modul menentukan sambungan antar modul. Kopling adalah konsep abstrak dan tidak mudah diukur. Jadi, tidak ada rumus yang dapat diberikan untuk menentukan sambungan antara

dua modul. Namun, beberapa faktor utama dapat diidentifikasi sebagai mempengaruhi pemasangan antar modul.

Di antara mereka yang paling penting adalah jenis koneksi antar modul, kompleksitas antar muka, dan jenis aliran informasi antar modul. Coupling meningkat dengan kompleksitas dan ketidakjelasan antar muka antar modul. Agar tetap rendah, diinginkan untuk meminimalkan jumlah antar muka per modul dan kompleksitas masing-masing antar muka. Antar muka modul digunakan untuk meneruskan informasi ke dan dari modul lain. Kompleksitas antar muka adalah faktor lain yang mempengaruhi kopling.

Semakin kompleks setiap antar muka, semakin tinggi derajat koplingnya. Jenis aliran informasi di sepanjang antar muka adalah faktor utama yang mempengaruhi kopling. Ada dua jenis informasi yang dapat mengalir di sepanjang antarmuka yaitu data atau kontrol, Melewati atau menerima informasi kontrol berarti bahwa tindakan modul akan bergantung pada informasi kontrol ini, yang membuatnya lebih sulit untuk memahami modul dan memberikan abstraksi. Transfer informasi data berarti bahwa modul dilewatkan sebagai input beberapa data ke modul lain dan mendapat balasan beberapa data sebagai output.



Desain terstruktur yang baik memiliki tingkat kohesi yang tinggi dan pengaturan *coupling* yang rendah.

B. Perancangan Berorientasi Fungsi

Dalam desain berorientasi fungsi, sistem ini terdiri dari banyak sub-sistem yang lebih kecil yang dikenal sebagai fungsi. Fungsi-fungsi ini mampu melakukan tugas penting dalam sistem. Sistem dianggap sebagai kombinasi dari semua fungsi.

Desain berorientasi fungsi mewarisi beberapa bagian dari desain terstruktur di mana adanya aktifitas membagi dan menyelesaikan metodologi yang digunakan.

Mekanisme desain ini membagi keseluruhan sistem menjadi fungsi yang lebih kecil, yang menyediakan sarana abstraksi dengan menyembunyikan informasi dan operasinya. Modul-modul fungsional ini dapat berbagi informasi di antara mereka sendiri dengan menggunakan informasi yang lewat dan menggunakan informasi yang tersedia secara global.

Karakteristik lain dari fungsi adalah apabila sebuah program memanggil fungsi, maka fungsi mengubah keadaan program, yang terkadang tidak dapat diterima oleh modul lain. Desain berorientasi fungsi, berfungsi dengan baik di mana status sistem tidak penting dan program/fungsi bekerja pada input pada suatu bagian.

Proses Perancangan

1. Keseluruhan sistem dilihat sebagai bentuk bagaimana data mengalir dalam sistem melalui DFD.
2. DFD menggambarkan bagaimana fungsi mengubah data dan keadaan sistem secara keseluruhan.
3. Seluruh sistem secara logis dipecah menjadi unit-unit yang lebih kecil yang dikenal sebagai fungsi atas dasar operasi mereka dalam sistem.
4. Setiap fungsi kemudian dijelaskan secara detil.

C. Perancangan Berbasis Objek (*Object Oriented Design*)

Dalam metode perancangan ini kita mengambil pandangan yang sedikit berbeda dari pengembangan sistem dari apa yang telah kita bahas sebelumnya. Pada bagian sebelumnya, kita cenderung melihat sistem dari sudut pandang fungsional atau berbasis proses, dan menghubungkannya dengan struktur data. Sementara pendekatan ini menghasilkan sistem kerja yang dirancang dengan baik. Pandangan di antara praktisi saat ini adalah bahwa sistem yang dihasilkan saat ini umumnya cenderung kaku dan membuatnya sulit untuk merespon dengan cepat terhadap perubahan dalam kebutuhan pengguna.

Tidak seperti dua pendahulunya, pendekatan berorientasi objek menggabungkan data dan proses (disebut metode) menjadi entitas tunggal yang disebut objek. Objek biasanya sesuai dengan hal-hal nyata yang ditangani sistem, seperti pelanggan, pemasok, kontrak, dan faktur. Model berorientasi objek mampu secara menyeluruh merepresentasikan hubungan yang kompleks dan untuk merepresentasikan data dan pemrosesan data dengan notasi yang andal. Hal ini memungkinkan campuran analisis dan desain yang lebih mudah dalam proses pertumbuhan aplikasi. Tujuan dari pendekatan Berorientasi Objek adalah untuk membuat elemen sistem lebih modular, sehingga meningkatkan kualitas sistem dan efisiensi analisis dan desain sistem.

Dalam pendekatan Berorientasi Objek, kita cenderung lebih fokus pada perilaku sistem. Fitur utama yang kami dokumentasikan adalah Object atau Class. *Object Oriented Design* (OOD) bekerja antar entitas dan karakteristiknya, bukan fungsi yang terlibat dalam sistem perangkat lunak. Strategi desain ini berfokus pada entitas dan karakteristiknya. Seluruh konsep solusi perangkat lunak bersiklus di areal entitas yang terlibat.

Konsep-konsep pokok dalam konsep *Object Oriented Design*, adalah sebagai berikut:

1. Objek. Semua entitas yang terlibat dalam perancangan solusi dikenal sebagai objek. Misalnya, orang, rumah sakit, badan usaha, dan klien diperlakukan sebagai objek. Setiap entitas memiliki beberapa atribut yang terkait dengannya dan memiliki beberapa metode untuk dilakukan pada atribut.
2. Kelas. Merupakan deskripsi umum dari suatu objek. Objek adalah turunan kelas. Class mendefinisikan semua atribut, yang bisa dimiliki objek dan metode, yang mendefinisikan fungsionalitas objek. Dalam desain solusi, atribut disimpan sebagai variabel dan fungsionalitas didefinisikan sebagai metode atau prosedur.
3. Enkapsulasi. Dalam OOD, atribut (variabel data) dan metode (operasi pada data) digabungkan bersama disebut enkapsulasi. Enkapsulasi tidak hanya memadukan informasi penting dari suatu objek secara bersamaan, tetapi juga membatasi akses data dan metode dari dunia luar. Ini disebut menyembunyikan informasi.
4. Warisan. OOD memungkinkan kelas serupa untuk menumpuk secara hierarkis di mana kelas bawah atau sub-kelas dapat mengimpor, menerapkan, dan menggunakan kembali variabel dan metode yang diperbolehkan dari kelas super langsung mereka. Properti OOD ini dikenal sebagai warisan. Ini membuatnya lebih mudah untuk menentukan kelas secara spesifik dan untuk membuat kelas umum dari kelas spesifik.
5. Polimorfisme. Bahasa OOD menyediakan mekanisme di mana metode yang melakukan tugas serupa tetapi bervariasi dalam argumen, dapat diberi nama yang sama. Ini disebut polimorfisme, yang memungkinkan satu antarmuka melakukan tugas untuk berbagai jenis. Tergantung pada bagaimana fungsi dipanggil, masing-masing bagian dari kode dijalankan.

Mekanisme OOD

Cobalah untuk mengidentifikasi beberapa objek umum dan kemudian lihat apakah anda dapat mengidentifikasi kelas yang ada. Misalnya,

dalam aplikasi perbankan otomatis, biasanya diharapkan sistem multi-level karena mengandung hal-hal berikut:

- Antar muka pengguna
- Perangkat lunak aplikasi untuk melakukan pemrosesan
- Database.

Di bagian **antar muka**, contoh objek bisa berupa menu, tombol, atau kotak dialog. Di tingkat **perangkat lunak** aplikasi, objek dapat berupa transaksi atau proses bisnis. Di tingkat model data objek akan menjadi tabel **database**.

Setiap objek memiliki *Identity* (nama), *State* (data yang terkait dengannya) dan perilaku (hal-hal yang dapat kita lakukan untuk itu). Seperti metodologi terstruktur lainnya, dalam pengembangan berorientasi objek ada beberapa metodologi dan tool yang tersedia untuk digunakan. Salah satu yang dapat digunakan adalah tool UML (*Unified Modeling Language*). Ini masih merupakan metode yang relatif baru dan mulai diminati oleh para pengembang perangkat lunak.

Sekilas tentang UML (*Unified Modeling Language*)

Unified Modeling Language (UML) adalah bahasa Berorientasi Objek untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan artefak sistem perangkat lunak, serta untuk pemodelan bisnis (UML Document Set, 2001). Perangkat Lunak Rasional dan mitranya mengembangkan UML. Ini adalah penerus bahasa pemodelan yang ditemukan di Booch (Booch 1994), OOSE / Jacobson, OMT dan metode lainnya.

Dengan menawarkan bahasa pengembangan umum, UML membebaskan pengembang dari ikatan kepemilikan yang begitu umum dalam rekayasa perangkat lunak dan dapat membuat sistem pengembangan menjadi sulit dan mahal. Perusahaan-perusahaan besar seperti IBM, Microsoft, dan Oracle disatukan di bawah payung UML. Karena kenyataan bahwa UML menggunakan notasi bawaan

yang sederhana, bahkan pengguna dengan keterampilan rekayasa perangkat lunak yang terbatas pun dapat memahami model UML. Faktanya, banyak pendukung bahasa mengklaim bahwa kesederhanaan UML adalah manfaat utamanya. Jika pengembang, pelanggan, dan pelaksana semuanya dapat memahami diagram UML, mereka lebih cenderung menyetujui fungsionalitas yang dimaksud, sehingga meningkatkan peluang mereka untuk membuat aplikasi yang benar-benar memecahkan masalah.

UML, bahasa pemodelan visual, tidak dimaksudkan sebagai bahasa pemrograman visual. Notasi UML berguna untuk menggambarkan secara grafis analisis berorientasi objek dan model desain. Ini tidak hanya memungkinkan kita untuk menentukan persyaratan sistem dan menangkap suatu keputusan desain, tetapi juga meningkatkan komunikasi di antara semua orang yang relevan yang terlibat dalam tugas pengembangan. Penekanan dalam pemodelan harus pada analisis dan desain.

UML memiliki beberapa diagram yang dapat kita gunakan untuk memodelkan suatu sistem, tetapi minimum yang diperlukan adalah:

- Diagram kelas (class) untuk menunjukkan objek dalam sistem dan tautan apa pun di antara mereka. Ini adalah alat yang berguna untuk tahap analisis atau desain.
- Diagram *use case* untuk membantu menangkap apa yang dilakukan sistem dan siapa yang berinteraksi dengannya. Ini paling berguna untuk menunjukkan tujuan sistem.
- *Sequence Diagram* untuk menangkap peristiwa yang terjadi dalam sistem dan memeriksa bagaimana sistem berperilaku.
- Model data untuk menangkap data. Ini biasanya lebih bermanfaat bagi tahap implementasi model siklus hidup.

Seiring dengan meningkatnya nilai strategis perangkat lunak bagi banyak perusahaan, kalangan industri mencari teknik untuk mengotomatisasi produksi perangkat lunak dan meningkatkan kualitas serta mengurangi biaya dan waktu. Teknik-teknik ini termasuk teknologi komponen, pemrograman visual, pola dan kerangka kerja. Kalangan bisnis juga mencari teknik untuk mengelola kompleksitas sistem karena mereka meningkatkan ruang lingkup dan skala. Secara khusus, mereka mengakui perlunya menyelesaikan masalah arsitektur berulang, seperti distribusi fisik, konkurensi, replikasi, keamanan, keseimbangan beban, dan toleransi kesalahan. Selain itu pengembangan untuk *World Wide Web*, sementara membuat beberapa hal lebih sederhana, telah memperburuk masalah arsitektur ini. Unified Modeling Language (UML) dirancang untuk menanggapi kebutuhan ini.

Jawabannya bermuara pada satu kata: **komunikasi**. Ini kata yang penting. Alasan utama mengapa perangkat lunak sangat sulit untuk dikembangkan adalah komunikasi. Kesulitan muncul karena kita harus berkomunikasi dengan banyak pengembang. UML penting karena dapat membantu pengembang perangkat lunak berkomunikasi. Kita harus menggunakannya dengan cara yang membantu komunikasi dan tidak menghambatnya.

Proses Perancangan

Proses desain perangkat lunak dapat dianggap sebagai serangkaian langkah yang terdefinisi dengan baik. Meskipun bervariasi sesuai dengan pendekatan desain (berorientasi fungsi atau berorientasi objek), namun ia memiliki langkah-langkah berikut:

1. Desain solusi dibuat berdasarkan kebutuhan atau sistem yang digunakan sebelumnya dan atau diagram urutan sistem.

2. Objek diidentifikasi dan dikelompokkan ke dalam kelas atas kesamaan nama dalam satu karakteristik atribut.
3. Hirarki kelas dan hubungan di antaranya didefinisikan.
4. Kerangka kerja aplikasi didefinisikan.

Pendekatan-Pendekatan Perancangan Perangkat Lunak

Berikut adalah dua pendekatan umum untuk perancangan perangkat lunak, yaitu:

1. Pendekatan *Top Down*

Suatu sistem terdiri dari lebih dari satu sub-sistem dan mengandung sejumlah komponen. Selanjutnya, sub-sistem dan komponen ini mungkin memiliki sub-sistem dan komponennya sendiri, dan menciptakan struktur hierarkis dalam sistem.

Desain top-down mengambil seluruh sistem perangkat lunak sebagai satu kesatuan dan kemudian menguraikannya untuk mencapai lebih dari satu sub-sistem atau komponen berdasarkan beberapa karakteristik. Setiap sub-sistem atau komponen kemudian diperlakukan sebagai suatu sistem dan didekomposisi lebih lanjut. Proses ini terus berjalan hingga tingkat sistem terendah dalam hirarki top-down tercapai.

Desain top-down dimulai dengan model umum sistem dan terus mendefinisikan bagian yang lebih spesifik dari itu. Ketika semua komponen tersusun, seluruh sistem menjadi ada. Dalam bahasa yang sederhana pendekatan *top down* adalah membuat modul kompleks dibagi menjadi beberapa sub modul.

Desain top-down lebih cocok ketika solusi perangkat lunak perlu dirancang dari awal dan detail spesifik tidak diketahui. Berikut ini adalah gambar ilustrasi mekanisme pendekatan *top down design*.



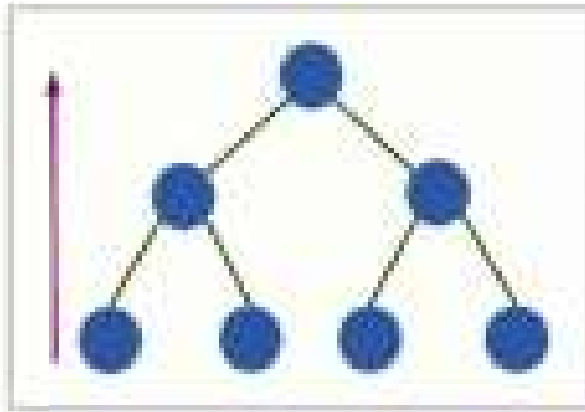
Gambar 8.2: Ilustrasi *Top-Down Design*

2. Pendekatan *Bottom Up*

Model desain bottom up dimulai dengan komponen yang paling spesifik dan mendasar. Ini hasil dengan menyusun tingkat komponen yang lebih tinggi dengan menggunakan komponen tingkat dasar atau lebih rendah. Itu terus menciptakan komponen tingkat yang lebih tinggi sampai sistem yang diinginkan tidak berevolusi sebagai satu komponen tunggal. Dengan masing-masing tingkat yang lebih tinggi, jumlah abstraksi meningkat.

Strategi bottom-up lebih cocok ketika sistem perlu dibuat dari beberapa sistem yang ada, di mana aplikasi dasar dapat digunakan dalam sistem yang lebih baru.

Kedua pendekatan *top-down* dan *bottom-up* tidak praktis secara individual. Sebagai gantinya, kombinasi yang baik dari keduanya digunakan. Berikut ini adalah gambar ilustrasi mekanisme pendekatan *Bottom Up*.



Gambar 8.3: Ilustrasi *Bottom-Up Design*

Metode Pendekatan Mana yang Akan di Pilih?

Untuk memutuskan metode desain mana yang paling cocok untuk suatu proyek perangkat lunak, pertimbangkan hal berikut:

1. Akankah siklus konsep desain produk akan menjadi sangat eksperimental? Apakah Anda mencoba membuat sesuatu yang sama sekali baru? Jika demikian, pendekatan iteratif Bottom-Up mungkin yang terbaik untuk proyek tersebut.
2. Apakah proyek Anda dibatasi oleh anggaran yang ketat? Jika demikian, pendekatan Top-Down dapat akan membantu dalam memaksimalkan penghematan dengan merencanakan anggaran secara menyeluruh pada awal siklus desain konsep produk Anda.
3. Apakah Anda membangun sistem yang besar dan rumit? Sistem dan mesin yang kompleks mendapat manfaat dari pendekatan Top-Down karena memecah tujuan proyek menjadi masalah yang lebih kecil yang lebih mudah dipecahkan.
4. Agar proyek Anda berhasil, apakah Anda membutuhkan pendapat semua orang untuk didengar? Jika masalah yang Anda coba selesaikan membutuhkan banyak kreatifitas, pendekatan Bottom-Up dapat membantu meningkatkan semua kreativitas dalam grup Anda dengan membiarkan mereka bereksperimen dan menyuarakan pendapat mereka.

Soal :

1. Jelaskan tujuan dari perancangan perangkat lunak.
2. Jelaskan mekanisme perancangan secara terstruktur.
3. Jelaskan mekanisme perancangan berbasis fungsi.
4. Jelaskan mekanisme perancangan berbasis objek.
5. Jelaskan maksud dari perancangan perangkat lunak dengan pendekatan *top-down*.
6. Jelaskan maksud dari perancangan perangkat lunak dengan pendekatan *bottom-up*.
7. Uraikan secara umum mekanisme perancangan perangkat lunak
8. Jelaskan maksud dari istilah berikut:
 - a. *Class*
 - b. *Object*
 - c. Polimorfisme
 - d. *Encapsulation*
 - e. Pewarisan (*inheritance*)

BAB IX

PERANCANGAN ANTAR MUKA PEMAKAI (USER INTERFACE)

Tujuan :

1. Mempelajari prinsip-prinsip yang harus dikuasai dalam melakukan perancangan *user interface* sebagai bagian penting dalam membangun perangkat lunak.
2. Mempelajari bentuk-bentuk elemen *user interface*.
3. Mempelajari aktifitas yang dilakukan desainer dalam merancang *user interface*.

Indikator keberhasilan :

1. Mahasiswa mampu menjelaskan hal-hal apa saja yang harus dipenuhi dalam membuat suatu *user interface*.
2. Mahasiswa mampu menyusun aturan-aturan dalam bentuk dokumen yang terkait dengan organisasi sebuah *user interface* pada sebuah aplikasi sederhana.
3. Mahasiswa mampu menggunakan salah satu perangkat lunak yang digunakan dalam membuat *user interface*.

Materi :

Desain *User Interface* (UI) berfokus untuk mengantisipasi apa yang mungkin perlu dilakukan pengguna dan memastikan bahwa antar muka memiliki elemen yang mudah diakses, dipahami, dan digunakan untuk memfasilitasi tindakan tersebut. UI menyatukan konsep-konsep dari desain interaksi, desain visual, dan arsitektur informasi.

User Interface adalah tampilan aplikasi *front-end* di mana pengguna berinteraksi untuk menggunakan perangkat lunak. Pengguna dapat

memanipulasi dan mengontrol perangkat lunak serta perangkat keras dengan menggunakan antarmuka pengguna. Saat ini, *user interface* ditemukan di hampir setiap tempat di mana teknologi digital ada, mulai dari komputer, ponsel, mobil, pemutar musik, pesawat terbang, kapal, dll.

User Interface merupakan bagian dari perangkat lunak dan dirancang sedemikian rupa sehingga diharapkan dapat memberikan wawasan pengguna perangkat lunak. UI menyediakan platform dasar untuk interaksi manusia-komputer.

UI dapat berupa grafis, berbasis teks, berbasis audio-video, tergantung pada kombinasi perangkat keras dan perangkat lunak yang mendasarinya. UI dapat berupa perangkat keras atau perangkat lunak atau kombinasi keduanya.

Prinsip Dalam Merancang *User Interface*

Dalam aktifitas merancang antar muka pemakai perlu diperhatikan hal-hal sebagai berikut:

1. Kejelasan

Kejelasan adalah pekerjaan pertama dan terpenting dari setiap perancangan antar muka. Agar efektif menggunakan antar muka yang dirancang, orang-orang harus dapat mengenali apa itu antarmuka, peduli mengapa mereka menggunakannya, memahami apa antarmuka yang membantu mereka berinteraksi, memprediksi apa yang akan terjadi ketika mereka menggunakannya, dan kemudian berhasil berinteraksi dengannya. Meskipun ada ruang untuk misteri dan kepuasan yang tertunda dalam antarmuka, tidak ada ruang untuk kebingungan. Kejelasan menginspirasi kepercayaan diri dan mengarah ke penggunaan lebih lanjut. Seratus layar yang jelas lebih baik daripada satu yang berantakan.

2. Keberadaan antar muka memungkinkan terjadinya interaksi

Antar muka dibuat untuk memungkinkan interaksi antara manusia dan aneka fungsi yang sangat luas. Ia dapat membantu memperjelas, menerangi, memungkinkan, menunjukkan hubungan, menyatukan

kita, memisahkan kita, mengelola harapan kita, dan memberi kita akses ke layanan. Tindakan mendesain antarmuka bukanlah seni. Antar muka bukanlah monumen bagi dirinya sendiri. Antar muka melakukan pekerjaan dan efektivitasnya dapat diukur. Antar muka terbaik dapat menginspirasi, membangkitkan, membingungkan, dan mengintensifkan hubungan manusia dengan dunia luar.

3. Fokus pada pengguna

Saat ini kita hidup di dunia yang penuh interupsi. Sulit untuk membaca dengan tenang tanpa ada yang mencoba mengalihkan perhatian dan mengarahkan perhatian ke tempat lain. Perhatian itu sangat berharga. Jangan mengotori sisi aplikasi yang dibuat dengan materi yang dapat digeser-geser. Jika seseorang membaca, biarkan mereka selesai membaca sebelum menunjukkan berbagai iklan ini dan itu (jika harus). Hormati perhatian dan bukan hanya pembaca yang akan lebih senang, hasil yang diperoleh akan menjadi lebih baik. Ketika digunakannya aplikasi adalah tujuan utama, perhatian menjadi prasyarat.

4. Pengguna tetap memegang kendali

Manusia merasa paling nyaman ketika mereka merasa mengendalikan diri dan lingkungan mereka. Perangkat lunak tanpa pamrih menyingkirkan kenyamanan itu dengan memaksa orang melakukan interaksi yang tidak direncanakan, jalur yang membingungkan, dan hasil yang mengejutkan. Buat pengguna tetap memegang kendali dengan secara teratur menampilkan status sistem, dengan menjelaskan penyebab (jika user melakukan ini, maka itu yang akan terjadi) dan dengan memberikan wawasan tentang apa yang diharapkan di setiap pilihan.

5. Dapat dimanipulasi secara langsung

Idealnya, antarmuka pada suatu aplikasi dibuat sedikit sehingga pengguna memiliki perasaan manipulasi langsung objek yang menjadi fokus mereka.

6. Satu tindakan utama pada tiap layar

Setiap layar yang didesain harus mendukung satu tindakan nilai nyata kepada orang yang menggunakannya. Ini membuatnya lebih mudah untuk dipelajari, lebih mudah digunakan, dan lebih mudah untuk ditambahkan atau dibangun ketika diperlukan. Layar yang mendukung dua atau lebih tindakan utama menjadi membingungkan dengan cepat.

7. Persiapkan fungsi-fungsi sekunder

Layar dengan satu tindakan utama dapat memiliki beberapa tindakan sekunder tetapi harus tetap sekunder. Misalnya anda membuat sebuah aplikasi yang menampilkan artikel. Fokus utamanya adalah agar user membaca artikel tersebut. Aksi selanjutnya (sekunder) akan ada pilihan untuk tindakan berikutnya setelah anda membaca (*share*, *forward*, dan lainnya).

8. Berikan langkah natural berikutnya

Sangat sedikit interaksi yang dimaksudkan untuk menjadi yang terakhir. Jadi pikirkanlah dengan saksama langkah berikutnya untuk setiap interaksi yang dimiliki seseorang dengan antarmuka. Antisipasi apa interaksi berikutnya dan desain untuk mendukungnya. Sama seperti yang kita suka dalam percakapan manusia, berikan pembukaan untuk interaksi lebih lanjut. Jangan biarkan seseorang menggantung karena mereka telah melakukan apa yang Anda ingin mereka lakukan. Memberi mereka langkah alami berikutnya yang membantu mereka mencapai tujuan mereka lebih jauh.

9. Penampilan mengikuti perilaku

Manusia paling nyaman dengan hal-hal yang berperilaku seperti yang diharapkannya. Ketika seseorang atau sesuatu berperilaku secara konsisten dengan harapan kita, maka kita merasa seperti memiliki hubungan yang baik dengannya. Untuk itu elemen yang dirancang harus terlihat seperti bagaimana mereka berperilaku. Formulir mengikuti fungsi. Dalam prakteknya ini berarti bahwa

seseorang harus dapat memprediksi bagaimana elemen antar muka akan berperilaku cukup hanya dengan melihatnya. Jika terlihat seperti tombol, tombol itu harus bertindak sebagai tombol.

10. Masalah konsistensi

Mengikuti prinsip sebelumnya, elemen layar tidak boleh tampil konsisten satu sama lain kecuali mereka saling konsisten satu sama lain. Elemen yang berperilaku sama harus terlihat sama. Dalam upaya untuk menjadi konsisten, perancang pemula sering mengaburkan perbedaan penting dengan menggunakan perlakuan visual yang sama (sering digunakannya kembali kode yang sama) ketika perlakuan visual yang berbeda sesuai.

11. Adanya hirarki visual yang bekerja secara ketat

Hirarki visual yang kuat dicapai ketika ada urutan tampilan yang jelas ke elemen visual pada layar. Yaitu, ketika pengguna melihat item yang sama dalam urutan yang sama setiap waktu. Lemahnya hierarki visual memberikan sedikit petunjuk tentang di mana harus mengistirahatkan pandangan seseorang dan akhirnya merasa berantakan dan membingungkan. Jika elemen visual tunggal ditambahkan ke layar, perancang mungkin perlu mengatur ulang visual semua elemen untuk sekali lagi mencapai hierarki yang kuat. Kebanyakan orang tidak memperhatikan hirarki visual tetapi ini adalah salah satu cara termudah untuk memperkuat (atau melemahkan) suatu desain.

12. Mengurangi beban kognitif

Lakukan penyederhanaan sehingga jumlah fitur yang banyak terlihat seperti segelintir saja. Hal ini dapat membantu orang memahami antarmuka yang didesain lebih mudah dan cepat, karena desainer telah menggambarkan hubungan konten yang melekat dalam desainnya. Kelompokkan secara bersama-sama seperti sebuah elemen, tunjukkan hubungan alami dengan posisi penempatan dan orientasi. Dengan mengatur konten secara cerdas, akan mengurangi beban kognitif pengguna. Pengguna tidak perlu

memikirkan bagaimana elemen terkait karena telah dilakukan untuk mereka. Jangan memaksa pengguna untuk mencari tahu hal tersebut. Tunjukkan kepada mereka dengan merancang hubungan tersebut ke layar peraga.

13. Buat *highlight*

Warna benda-benda fisik akan mengalami perubahan ketika cahaya berubah. Dalam pencahayaan penuh di siang hari kita dapat melihat pohon yang sangat berbeda dari yang terlihat saat matahari terbenam. Seperti di dunia fisik, di mana warna adalah banyak hal yang diarsir, warna tidak harus banyak menentukan dalam antarmuka. Ini dapat membantu, digunakan untuk menyoroti, digunakan untuk memandu perhatian, tetapi seharusnya tidak menjadi satu-satunya pembeda suatu hal. Untuk membaca lebih lama atau menggunakan suatu tampilan lebih lama, gunakan cahaya atau warna latar yang diredam.

14. Penyajian yang progresif

Buat dan tampilkan apa yang hanya diperlukan di setiap layar. Jika orang membuat pilihan, tunjukkan informasi yang cukup untuk memungkinkan mereka memilih, lalu masuki detail pada layar berikutnya. Hindari kecenderungan untuk terlalu menjelaskan atau menunjukkan semuanya sekaligus. Jika memungkinkan, tangguhkan keputusan untuk masuk ke layar berikutnya secara progresif mengungkapkan informasi yang diperlukan. Ini akan membuat interaksi akan lebih jelas.

15. Strategi *inline*

Dalam antar muka yang ideal, bantuan tidak diperlukan karena antarmuka dapat dipelajari dan digunakan. Langkah di bawah ini, faktanya adalah satu di mana bantuan bersifat inline dan kontekstual, hanya tersedia kapan dan di mana diperlukan. Selebihnya ia tersembunyi dari pandangan.

16. Selalu dimulai dari awal

Pengalaman pertama kali dengan antar muka sangat penting, namun sering diabaikan oleh para desainer. Untuk membantu pengguna memahami dengan cepat terkait hal yang didesain, hal yang terbaik adalah merancang untuk keadaan nol, keadaan di mana belum ada apa-apa yang terjadi. Apabila banyak friksi interaksi dalam kondisi awal interaksi maka akan membuat pemahaman lebih lama. Sebaliknya, begitu orang memahami aturan, mereka memiliki kemungkinan sukses yang jauh lebih tinggi.

17. Desain hebat yang tidak terlihat

Properti aneh dari suatu rancangan yang hebat adalah bahwa biasanya tidak diketahui oleh orang-orang yang menggunakannya. Salah satu alasannya adalah jika desain berhasil, pengguna dapat fokus pada tujuan mereka sendiri dan bukan antarmuka. Ketika mereka menyelesaikan tujuan mereka, mereka puas dan tidak perlu memikirkan situasi tersebut bisa terjadi.

18. Kolaborasi dengan disiplin ilmu lain

Disiplin ilmu desain visual dan grafis, tipografi, *copywriting*, arsitektur informasi dan visualisasi adalah bagian dari desain antarmuka. Semua itu dapat dipelajari dan dijadikan spesialisasi.

19. Keberadaan antar muka harus digunakan

Seperti dalam kebanyakan disiplin desain, desain antarmuka dipandang berhasil ketika orang menggunakan apa yang telah dirancang. Desain dipandang gagal ketika orang memilih untuk tidak menggunakannya. Tidaklah cukup untuk sebuah antar muka yang hanya untuk memuaskan ego dari perancangannya. Intinya antar muka itu harus digunakan.

Memilih Elemen Antarmuka

Pengguna yang telah terbiasa dengan elemen antarmuka tertentu untuk berinteraksi dengan sistem, maka sebaiknya dibuat konsisten dan dapat diprediksi dalam pilihan antar muka dan tata letaknya. Apabial hal ini

dilakukan maka akan membantu memudahkan dalam penyelesaian tugas, efisiensi, dan kepuasan.

Elemen antarmuka yang baik mencakup hal-hal berikut ini :

- Kontrol Input: *buttons, text fields, checkboxes, radio buttons, dropdown lists, list boxes, toggles, date field*
- Komponen Navigasi: *breadcrumb, slider, search field, pagination, slider, tags, icons*
- Komponen Informasi: *tooltips, icons, progress bar, notifications, message boxes, modal windows*
- *Containers*: isi.

Perangkat lunak menjadi lebih populer jika antarmuka penggunaannya memenuhi persyaratan sebagai berikut:

- Menarik
- Mudah digunakan
- Responsif dalam waktu singkat
- Jelas untuk dimengerti
- Konsisten pada semua layar antarmuka

Secara umum *user interface* dikelompokkan atas dua bagian:

A. *Command Line Interface (CLI)*

CLI telah menjadi alat interaksi yang hebat dengan komputer sampai monitor tampilan video muncul. CLI adalah pilihan pertama dari banyak pengguna dan pemrogram teknis. Ini adalah antarmuka minimum yang dapat diberikan oleh perangkat lunak kepada pengguna.

CLI menyediakan prompt perintah, tempat di mana pengguna mengetikkan perintah dan atau memberi umpan balik (*feed back*) ke sistem. Pengguna perlu mengingat sintaks perintah dan penggunaannya. CLI sebelumnya tidak diprogram untuk menangani kesalahan pengguna secara efektif.

Perintah adalah referensi berbasis teks untuk mengatur instruksi, yang diharapkan akan dieksekusi oleh sistem. Ada beberapa metode seperti makro, skrip yang memudahkan pengguna untuk beroperasi.

CLI menggunakan lebih sedikit sumber daya komputer dibandingkan dengan GUI.

Bentuk contoh tampilan user interface berbasis CLI dapat dilihat pada contoh berikut.

```

vivek@vivek:~/scripts$ pwd
/home/vivek/scripts
vivek@vivek:~/scripts$ ls -l
total 12
-rw-r--r-- 1 vivek vivek  545 Sep 28  2009 conditions.sh
-rw-r--r-- 1 vivek vivek 1728 Sep 28  2009 gmail_parser.py
-rw-r--r-- 1 vivek vivek   56 Dec  1  2007 hddmonit.sh
-rw-r--r-- 2 vivek vivek  4096 Mar 18 01:34 php
-rw-r--r-- 1 vivek vivek 10283 Jul 29  2012 popodynka.sh
-rw-r--r-- 2 vivek vivek  4096 Mar 18 01:34 python
vivek@vivek:~/scripts$ cd python
vivek@vivek:~/scripts/python$ pwd
/home/vivek/scripts/python
vivek@vivek:~/scripts/python$
  
```

Gambar 9.1: Bentuk tampilan user interface berbasis CLI

Elemen-elemen yang terdapat pada CLI:

- [1]. *Command Prompt*. Merupakan notifikasi berbasis teks yang sebagian besar menunjukkan konteks di mana pengguna bekerja. Ini dihasilkan oleh sistem perangkat lunak.
- [2]. *Kursor (cursor)*. Merupakan garis horizontal kecil atau garis vertikal dari tinggi garis, untuk mewakili posisi karakter saat mengetik. Kursor sebagian besar ditemukan dalam keadaan berkedip. Bergerak saat pengguna menulis atau menghapus sesuatu.
- [3]. *Perintah (command)*. Merupakan instruksi yang dapat dieksekusi oleh sistem. Ini mungkin memiliki satu atau lebih parameter. Output pada eksekusi perintah ditampilkan sebaris pada layar.

Ketika output dihasilkan, *command prompt* ditampilkan pada baris berikutnya.

B. Graphical User Interface (GUI)

Graphical User Interface (GUI) menyediakan sarana grafis pengguna untuk berinteraksi dengan sistem. GUI dapat menjadi kombinasi antara perangkat keras dan perangkat lunak. Menggunakan GUI, pengguna menafsirkan perangkat lunak.

Biasanya, GUI lebih banyak mengkonsumsi sumber daya daripada CLI. Dengan kemajuan teknologi, para programmer dan desainer menciptakan desain GUI yang kompleks yang bekerja dengan lebih efisien, akurat, dan cepat.

Elemen yang ada pada GUI

GUI menyediakan satu set komponen untuk berinteraksi dengan perangkat lunak atau perangkat keras.

Setiap komponen grafis menyediakan cara untuk bekerja dengan sistem. Sistem GUI memiliki elemen-elemen berikut seperti:



Gambar 9.2: Elemen pada GUI

- *Window*. Merupakan area di mana konten aplikasi ditampilkan. Isi di jendela dapat ditampilkan dalam bentuk ikon atau daftar, jika jendela mewakili struktur file. Lebih mudah bagi pengguna untuk menavigasi dalam sistem file di jendela eksplorasi. Windows dapat diperkecil, diubah ukurannya atau dimaksimalkan ke ukuran layar. Mereka dapat dipindahkan ke mana saja di layar. Jendela mungkin berisi jendela lain dari aplikasi yang sama, yang disebut jendela anak.
- *Tab*. Jika suatu aplikasi memungkinkan untuk menjalankan beberapa kejadian itu sendiri, mereka muncul di layar sebagai jendela terpisah. Tabbed Document Interface telah muncul untuk membuka banyak dokumen di jendela yang sama. Antarmuka ini juga membantu dalam melihat panel preferensi dalam aplikasi. Semua peramban web modern menggunakan fitur ini.
- *Menu*. Merupakan larik perintah standar, dikelompokkan bersama dan ditempatkan di tempat yang terlihat (biasanya atas) di dalam jendela aplikasi. Menu dapat diprogram untuk muncul atau disembunyikan pada klik mouse.
- *Icon*. Merupakan gambar kecil yang mewakili aplikasi terkait. Ketika icon-icon ini diklik atau diklik dua kali, maka jendela aplikasi terbuka. Icon menampilkan aplikasi dan program yang diinstal pada sistem dalam bentuk gambar kecil.
- *Kursor*. Berinteraksi perangkat seperti mouse, touch pad, pena digital diwakili dalam GUI sebagai kursor. Pada layar kursor mengikuti instruksi dari perangkat keras hampir secara real-time. Kursor juga disebut pointer dalam sistem GUI. Mereka digunakan untuk memilih menu, jendela dan fitur aplikasi lainnya.

Komponen GUI pada Aplikasi Khusus

Suatu GUI yang terdapat pada aplikasi yang memuat satu atau lebih elemen GUI, seperti:

1. Jendela Aplikasi - Sebagian besar jendela aplikasi menggunakan konstruksi yang disediakan oleh sistem operasi tetapi banyak yang menggunakan jendela buatan pelanggan mereka sendiri untuk memuat konten aplikasi.
2. Kotak Dialog (*dialog box*) - Ini adalah jendela anak yang berisi pesan untuk pengguna dan permintaan untuk beberapa tindakan yang harus diambil. Sebagai contoh: Aplikasi menghasilkan dialog untuk mendapatkan konfirmasi dari pengguna untuk menghapus file.
3. Text-Box - Menyediakan area bagi pengguna untuk mengetik dan memasukkan data berbasis teks.
4. Tombol (*button*) - Tombol ini meniru tombol kehidupan nyata dan digunakan untuk mengirimkan input ke perangkat lunak.
5. *Radio button* - Menampilkan opsi yang tersedia untuk dipilih. Hanya satu yang dapat dipilih di antara semua yang ditawarkan.
6. *Check Box* - Fungsi-fungsi yang mirip dengan daftar-kotak. Ketika opsi dipilih, kotak ditandai sebagai dicentang. Beberapa opsi diwakili oleh kotak centang dapat dipilih.
7. *List Box* - Memberikan daftar item yang tersedia untuk dipilih. Lebih dari satu item dapat dipilih.

Selain dari elemen di atas, masih terdapat elemen yang tidak kalah penting seperti *Sliders*, *Combo-box*, *Data-grid*, dan *Drop-down list*.

Aktifitas Perancangan User Interface

Ada sejumlah aktivitas yang dilakukan untuk merancang antarmuka pengguna. Proses desain dan implementasi GUI mirip SDLC. Model apa pun dapat digunakan untuk implementasi GUI di antara *Waterfall*, *Iterative* atau *Spiral Model*.

Sebuah model yang digunakan untuk desain dan pengembangan GUI harus memenuhi langkah-langkah spesifik GUI yang terlihat pada gambar berikut.



Gambar 9.3: Tahap Perancangan dan Pengembangan GUI

1. *GUI Requirement Gathering* - Para desainer mungkin ingin memiliki daftar semua persyaratan fungsional dan non-fungsional GUI. Ini dapat diambil dari pengguna dan solusi perangkat lunak yang ada.
2. *User Analysis* - Studi perancang yang akan menggunakan perangkat lunak GUI. Target audiensi penting karena detail desain berubah sesuai dengan pengetahuan dan tingkat kompetensi pengguna. Jika pengguna paham teknis, canggih dan kompleks GUI dapat dimasukkan. Untuk pengguna pemula, informasi lebih lanjut disertakan pada bagaimana-untuk perangkat lunak.
3. *Task Analysis* - Desainer harus menganalisis tugas apa yang harus dilakukan oleh solusi perangkat lunak. Di sini, di GUI, tidak masalah bagaimana itu akan dilakukan. Tugas dapat diwakili dengan cara hierarkis mengambil satu tugas utama dan membaginya lebih jauh ke

dalam sub-tugas yang lebih kecil. Tugas memberikan tujuan untuk presentasi GUI. Arus informasi di antara sub-tugas menentukan aliran isi GUI dalam perangkat lunak.

4. *GUI Desain and implementation* - Desainer setelah memiliki informasi tentang persyaratan, tugas dan lingkungan pengguna, desain GUI dan mengimplementasikan ke dalam kode dan menanamkan GUI dengan perangkat lunak bekerja atau dummy di latar belakang. Itu kemudian diuji sendiri oleh para pengembang.
5. *Testing* - Pengujian GUI dapat dilakukan dengan berbagai cara. Organisasi dapat memiliki pemeriksaan internal, keterlibatan langsung pengguna dan rilis versi beta hanya sedikit. Pengujian dapat mencakup kegunaan, kompatibilitas, penerimaan pengguna, dll.

Soal

1. Jelaskan, mengapa user interface berbasis CLI lebih sedikit menggunakan sumber daya komputer bila dibandingkan dengan GUI?
2. Jelaskan faktor-faktor penentu keberhasilan dalam melakukan perancangan user interface.
3. Jelaskan aktifitas-aktifitas apa saja yang dilakukan dalam perancangan user interface.
4. Jelaskan elemen-elemen apa saja yang terdapat pada GUI.
5. Uraikan dan jelaskan setidaknya 5 (lima) prinsip dasar yang harus dipenuhi dalam mengembangkan suatu *user interface*.
6. Tuliskan nama-nama perangkat lunak (tool) yang dapat digunakan untuk membantu dalam perancangan *user interface*.

BAB X

IMPLEMENTASI PERANGKAT LUNAK

Tujuan :

1. Mempelajari bentuk-bentuk strategi implementasi perangkat lunak.
2. Mempelajari bermacam permasalahan yang terjadi sepanjang proses implementasi dilakukan.
3. Mempelajari tentang model-model implementasi perangkat lunak.

Indikator keberhasilan :

1. Mahasiswa mampu menjelaskan metode-metode yang digunakan dalam kegiatan pemrograman.
2. Mahasiswa mengetahui fase-fase apa saja yang dilalui sepanjang aktifitas implementasi perangkat lunak.
3. Mahasiswa memahami tipe-tipe implementasi dan penerapannya.
4. Mahasiswa memahami peran dokumentasi terhadap keberlanjutan suatu perangkat lunak.

Materi :

Tahap implementasi perangkat lunak merupakan suatu proses perubahan spesifikasi sistem menjadi sistem yang dapat dijalankan oleh mesin

Pemrograman Terstruktur

Sepanjang proses pemrograman, baris kode terus mengalami penambahan, dengan demikian ukuran perangkat lunak meningkat. Secara bertahap, hal itu menjadi tidak mungkin bagi programmer untuk mengingat aliran program secara utuh. Jika seseorang lupa bagaimana perangkat lunak dan

program yang mendasarinya, file, prosedur dibangun, maka menjadi sangat sulit untuk berbagi pekerjaan, men-*debug*, dan memodifikasi program. Solusi untuk hal ini adalah pemrograman terstruktur. Metode ini mendorong pengembang untuk menggunakan subrutin dan loop dari pada menggunakan lompatan sederhana (pintas) dalam kode, sehingga menciptakan kejelasan dalam susunan kode dan meningkatkan efisiensinya. Pemrograman terstruktur juga membantu programmer untuk mengurangi waktu pengkodean dan mengatur kode dengan benar.

Pemrograman terstruktur menyatakan bagaimana program harus dikodekan. Ini menggunakan tiga konsep utama:

1. Analisis *top-down*. Perangkat lunak selalu dibuat untuk melakukan beberapa pekerjaan yang rasional. Karya rasional ini dikenal sebagai masalah dalam bahasa perangkat lunak. Dengan demikian sangat penting bahwa kita memahami bagaimana memecahkan masalah. Dengan konsep analisis *top-down*, masalah dipecah menjadi bagian-bagian lebih kecil di mana masing-masingnya memiliki beberapa kegunaan. Setiap masalah diselesaikan secara individual dan langkah-langkah pemecahan masalahnya dinyatakan dengan jelas.
2. Pemrograman Modular. Saat pemrograman, kode dipecah menjadi grup instruksi/perintah yang lebih kecil. Kelompok-kelompok ini dikenal sebagai modul, subprogram, atau subrutin. Pemrograman modular dibuat berdasarkan pemahaman analisis *top-down*. Hal ini akan mencegah adanya lompatan menggunakan pernyataan 'goto' dalam program, yang sering membuat aliran program tidak dapat/sulit dilacak.
3. Pemrograman Terstruktur. Dalam referensi dengan analisis *top-down*, pemrograman terstruktur membagi sub-modul ke dalam unit kode yang lebih kecil berdasarkan urutan eksekusinya. Pemrograman terstruktur menggunakan struktur kontrol (*Sequential*, *Loop*, *Decission*), yang mengontrol aliran program, sedangkan pemrograman terstruktur menggunakan struktur kontrol untuk mengatur instruksi dalam pola yang dapat ditentukan.

Pemrograman Berbasis Fungsi

Pemrograman berbasis fungsi adalah gaya bahasa pemrograman yang menggunakan konsep fungsi matematika. Suatu fungsi dalam matematika harus selalu menghasilkan hasil yang sama dalam menerima argumen yang sama. Dalam bahasa prosedural, aliran program berjalan melalui prosedur, yaitu kontrol program ditransfer ke prosedur yang disebutkan. Sementara aliran kontrol berpindah dari satu prosedur ke prosedur lainnya, program mengubah statusnya.

Dalam pemrograman prosedural, adalah mungkin untuk prosedur untuk menghasilkan hasil yang berbeda ketika dipanggil dengan argumen yang sama, karena program itu sendiri dapat berada dalam keadaan yang berbeda ketika memanggilnya. Ini adalah properti serta kelemahan pemrograman prosedural, di mana urutan atau waktu pelaksanaan prosedur menjadi penting.

Pemrograman berbasis fungsi menyediakan sarana komputasi sebagai fungsi matematika, yang menghasilkan suatu hasil yang terlepas dari status program. Hal ini memungkinkan untuk dilakukan prediksi terhadap perilaku program.

Pemrograman berbasis fungsi menggunakan konsep-konsep berikut:

1. Fungsi utama dan tingkat tinggi.

Fungsi-fungsi ini memiliki kemampuan untuk menerima fungsi lain sebagai argumen atau mengembalikan fungsi lain sebagai hasil (*return value*).

2. Fungsi-fungsi murni.

Fungsi-fungsi ini tidak termasuk pembaruan destruktif, yaitu ia tidak mempengaruhi I/O atau memori dan jika mereka tidak digunakan, mereka dapat dengan mudah dihapus tanpa menghambat fungsi-fungsi lain pada program.

3. Rekursi.

Rekursi adalah teknik pemrograman di mana fungsi memanggil dirinya sendiri dan mengulangi kode program di dalamnya kecuali

beberapa kondisi yang ditentukan sebelumnya cocok. Rekursi adalah cara membuat loop dalam pemrograman fungsional.

4. Evaluasi yang ketat.

Ini adalah metode untuk mengevaluasi ekspresi yang dilewatkan ke fungsi sebagai argumen. Pemrograman fungsional memiliki dua jenis metode evaluasi, yaitu ketat atau tidak ketat. Evaluasi yang ketat selalu mengevaluasi ekspresi sebelum memanggil fungsi yang dirujuk. Evaluasi yang tidak ketat tidak mengevaluasi ekspresi kecuali diperlukan.

5. *λ -calculus*.

Sebagian besar bahasa pemrograman fungsional menggunakan *λ -calculus* di dalamnya. *λ -ekspresi* dieksekusi dengan cara mengevaluasinya saat terjadi proses.

Gaya Pemrograman

Gaya pemrograman adalah seperangkat pedoman yang digunakan untuk memformat instruksi pemrograman. Sangat berguna untuk mengikuti gaya karena memudahkan programmer untuk memahami kode, memeliharanya, dan membantu mengurangi kemungkinan terjadinya kesalahan. Panduan dapat dikembangkan dari penetapan pengkodean yang digunakan dalam organisasi dengan variasi gaya yang terjadi untuk bahasa pemrograman yang berbeda. Elemen kunci dari panduan gaya pemrograman termasuk konvensi penamaan, penggunaan komentar dan pemformatan (indentasi, dan spasi kosong). Dalam beberapa bahasa (misalnya *Python*), indentasi digunakan untuk menunjukkan struktur kontrol (maka indentasi yang benar diperlukan), sementara dalam bahasa lain indentasi digunakan untuk meningkatkan tampilan agar mudah terlihat dan keterbacaan kode program (misalnya *Java*).

Penetapan Penamaan

Ketika menamai variabel, fungsi/metode, kelas, file, dan sebagainya, adalah penting untuk mengikuti aturan penamaan, dan menggunakan ejaan

bahasa Indonesia/Inggris yang benar (ini membantu dengan operasi pencarian/temukan/pengantian). Aturan penamaan digunakan untuk meningkatkan tampilan visual dan mengurangi upaya yang diperlukan untuk membaca dan memahami kode. Hal itu dapat bervariasi dalam bahasa pemrograman yang berbeda. Berikut ini adalah panduan penetapan penamaan yang baik:

1. Semua penamaan harus bersifat deskriptif (apabila diharuskan).
2. Sedapat mungkin hindari singkatan. Cobalah untuk tetap memberi nama campuran yang tepat agar mudah dipahami tetapi tidak terlalu lama. Misalnya, hindari **"jnsklm"**. Sebaiknya ditulis **"jeniskelamin"**.
3. Spasi tidak boleh digunakan. Beberapa bahasa akan menggunakan tanda pisah untuk nama (mis. **Berat bersih**), sementara bahasa lain akan menggunakan garis bawah (mis. **Berat_bersih** dalam Python, SQL). Java menggunakan kata campuran untuk variabel yang dimulai dengan huruf kecil (mis. **beratBersih**).
4. Konstanta biasanya didefinisikan sebagai semua huruf besar menggunakan garis bawah untuk memisahkan kata-kata (misalnya MAX_HEIGHT).

Baris Komentar

Komentar dapat meningkatkan keterbacaan program. Seperti halnya aturan penamaan, penting untuk menggunakan ejaan bahasa Indonesia/Inggris yang benar. Harap dicatat bahwa bahasa pemrograman yang berbeda memungkinkan gaya berkomentar pun berbeda. Berikut ini adalah saran untuk membuat baris komentar:

1. Bersikap konsisten dengan penggunaan Anda untuk mengomentari sintaks, misalnya:

//Ini adalah sebaris komentar

*/*Ini adalah blok komentar*

*yang lebih dari dua baris */*

2. Mulailah setiap file/modul dengan komentar di bagian atas yang menggambarkan isinya.
3. Mulai setiap kelas dengan komentar yang menyertainya yang menjelaskan untuk apa dan bagaimana seharusnya digunakan.
4. Mulai setiap fungsi dengan komentar yang menjelaskan penggunaan fungsi.
5. Nama-nama variabel harus cukup deskriptif untuk tidak perlu dikomentari. Jika ini sulit, berikan komentar singkat pada deklarasi.
6. Hanya beri komentar pada bagian yang rumit, tidak jelas, bagian yang menarik atau penting dari kode selama implementasi.
7. Ikuti konvensi tata bahasa yang normal untuk memastikan keterbacaan komentar.

Dokumentasi Perangkat Lunak

Dokumentasi perangkat lunak adalah bagian penting dari proses perangkat lunak. Dokumen yang ditulis dengan baik menyediakan tool yang hebat dan sarana penyimpanan informasi yang diperlukan untuk mengetahui tentang proses perangkat lunak. Dokumentasi perangkat lunak juga menyediakan informasi tentang cara menggunakan produk.

Dokumentasi yang terpelihara dengan baik harus melibatkan dokumen-dokumen berikut:

1. Dokumentasi persyaratan.

Dokumentasi ini berfungsi sebagai alat utama untuk perancang perangkat lunak, pengembang, dan tim penguji untuk melaksanakan tugasnya masing-masing. Dokumen ini berisi semua deskripsi fungsional, non-fungsional dan perilaku dari perangkat lunak yang dimaksud.

Sumber dokumen ini dapat menyimpan data sebelumnya tentang perangkat lunak, sudah menjalankan perangkat lunak di bagian akhir klien, wawancara klien, kuesioner, dan penelitian. Umumnya disimpan dalam bentuk *spreadsheet* atau dokumen pengolah kata dengan tim manajemen perangkat lunak *high-end*.

Dokumentasi ini berfungsi sebagai dasar untuk perangkat lunak yang akan dikembangkan dan terutama digunakan dalam tahap verifikasi dan validasi. Kebanyakan *test case* dibangun langsung dari dokumentasi persyaratan.

2. Dokumentasi Hasil Desain Perangkat Lunak.

Dokumentasi ini berisi semua informasi yang diperlukan, yang diperlukan untuk membangun perangkat lunak. Isi dari dokumen tersebut adalah: (a) arsitektur perangkat lunak tingkat tinggi, (b) detail desain perangkat lunak, (c) diagram alir data, (d) perancangan basis data

Dokumen-dokumen ini berfungsi sebagai repositori bagi pengembang untuk mengimplementasikan perangkat lunak. Meskipun dokumen-dokumen ini tidak memberikan rincian tentang bagaimana kode program, mereka memberikan semua informasi yang diperlukan yang diperlukan untuk pengkodean dan implementasi.

3. Dokumentasi teknis.

Dokumentasi ini dikelola oleh pengembang dan pemrogram yang sebenarnya. Dokumen-dokumen ini, secara keseluruhan, mewakili informasi tentang kode. Saat menulis kode, para programmer juga menyebutkan tujuan kode, siapa yang menulisnya, di mana dibutuhkan, apa yang dilakukannya dan bagaimana ia melakukannya, apa sumber daya lain yang digunakan kode, dll.

Dokumentasi teknis meningkatkan pemahaman antara berbagai programmer bekerja pada kode yang sama. Ini meningkatkan kemampuan penggunaan ulang kode. Itu membuat debugging mudah dan dapat dilacak.

Ada berbagai alat otomatis tersedia dan beberapa dilengkapi dengan bahasa pemrograman itu sendiri. Misalnya *Java* yang tersedia tool bernama *JavaDoc* untuk menghasilkan dokumentasi teknis kode.

4. Dokumentasi pengguna.

Dokumentasi ini berbeda dari semua penjelasan di atas. Semua dokumentasi sebelumnya dipertahankan untuk memberikan

informasi tentang perangkat lunak dan proses pengembangannya. Tetapi dokumentasi pengguna menjelaskan bagaimana produk perangkat lunak seharusnya bekerja dan bagaimana seharusnya digunakan untuk mendapatkan hasil yang diinginkan.

Dokumentasi ini mungkin termasuk, prosedur instalasi perangkat lunak, panduan cara, panduan pengguna, metode penghapusan instalasi dan referensi khusus untuk mendapatkan informasi lebih lanjut seperti pembaruan lisensi dan sebagainya.

Tantangan Implementasi Perangkat Lunak

Ada beberapa tantangan yang dihadapi oleh tim pengembangan saat mengimplementasikan perangkat lunak. Beberapa dari mereka disebutkan di bawah ini:

1. Penggunaan kembali kode (*code reuse*).

Antarmuka pemrograman bahasa masa kini sangat canggih dan dilengkapi fungsi perpustakaan yang sangat besar. Namun, untuk menurunkan biaya produk akhir, manajemen organisasi lebih memilih untuk menggunakan kembali kode, yang dibuat sebelumnya untuk beberapa perangkat lunak lain. Ada masalah besar yang dihadapi oleh programmer untuk pemeriksaan kompatibilitas dan memutuskan berapa banyak kode untuk digunakan kembali.

2. Manajemen Versi.

Setiap kali perangkat lunak baru diberikan kepada pelanggan, pengembang harus mempertahankan versi dan konfigurasi dokumentasi terkait. Dokumentasi ini harus sangat akurat dan tersedia tepat waktu.

3. Target-Host.

Program perangkat lunak, yang sedang dikembangkan di organisasi, perlu dirancang untuk mesin host di akhir pelanggan.

Tetapi kadang-kadang, tidak mungkin untuk merancang perangkat lunak yang bekerja pada mesin target.

Berbagai permasalahan perangkat lunak

Perangkat lunak itu mahal. Selama beberapa dekade terakhir, dengan kemajuan teknologi, biaya perangkat keras mengalami penurunan secara konsisten. Di sisi lain, biaya perangkat lunak terus mengalami peningkatan. Akibatnya, rasio perangkat keras dan lunak untuk sistem komputer telah menunjukkan kondisi yang berlawanan sejak tahun-tahun awal.

Alasan utama tingginya biaya perangkat lunak adalah pengembangan perangkat lunak masih bersifat padat karya. Biaya utama pembuatan perangkat lunak adalah tenaga kerja yang dipekerjakan. Biaya pengembangan perangkat lunak umumnya diukur dalam hitungan orang per bulan dari upaya yang dihabiskan dalam pengembangan. Dan produktifitas sering diukur dalam industri dalam hal DLOC (*Delivered Line of Code*) per orang-bulan.

Untuk mengatasi dan memodifikasi modul secara terpisah, diinginkan agar modul tersebut dapat digabungkan secara fleksibel dengan modul lainnya. Pilihan modul menentukan keterkaitan antar modul. Kopling adalah konsep abstrak dan tidak mudah diukur. Jadi, tidak ada rumus yang dapat diberikan untuk menentukan keterkaitan antara dua modul. Namun, beberapa faktor utama dapat diidentifikasi sebagai hal yang mempengaruhi pemasangan antar modul.

Di antara mereka yang paling penting adalah jenis koneksi antar modul, kompleksitas antarmuka, dan jenis aliran informasi antar modul. Coupling meningkat dengan kompleksitas dan ketidakjelasan antar muka antar modul. Agar tetap rendah, diinginkan agar meminimalkan jumlah antarmuka per modul dan kompleksitas setiap antar muka. Antar muka modul digunakan untuk meneruskan informasi ke dan dari modul lain. Kompleksitas antarmuka adalah faktor lain yang mempengaruhi kopling.

Lambat, mahal, dan tidak dapat handal adalah isu yang mengemuka dalam banyak kasus. Ada banyak contoh yang dikutip tentang proyek

perangkat lunak yang terlambat dan memiliki kelebihan biaya. Industri perangkat lunak telah mendapatkan reputasi yang kurang baik karena tidak dapat memenuhi target tepat waktu dan sesuai anggaran. Kegagalan perangkat lunak berbeda dari kegagalan, katakanlah, sistem mekanik atau listrik.

Produk dari disiplin teknik lain ini gagal karena perubahan sifat fisik atau listrik dari sistem yang disebabkan oleh penuaan. Dalam perangkat lunak, kegagalan terjadi karena adanya *bug* atau kesalahan yang diperkenalkan selama proses desain dan pengembangan. Oleh karena itu, meskipun perangkat lunak mungkin gagal setelah beroperasi dengan benar selama beberapa waktu, *bug* yang menyebabkan kegagalan itu ada sejak awal. Itu hanya dieksekusi pada saat kegagalan terjadi.

Ini sangat berbeda dari sistem lain, di mana jika suatu sistem gagal, umumnya berarti bahwa kadang-kadang sebelum kegagalan pengembangan sistem beberapa masalah yang tidak dijumpai sebelumnya.

Masalah perubahan dan pengerjaan ulang. Setelah perangkat lunak di-*delivery* dan digunakan, ia memasuki fase pemeliharaan. Semua sistem membutuhkan perawatan, tetapi untuk sistem lain sebagian besar disebabkan oleh masalah yang disebabkan oleh penuaan. Perangkat lunak perlu dipelihara bukan karena beberapa komponennya aus dan perlu diganti, tetapi karena sering ada beberapa kesalahan sisa yang tersisa dalam sistem yang harus dihapus ketika mereka ditemukan.

Pemeliharaan melibatkan memahami perangkat lunak yang ada, memahami efek perubahan, membuat perubahan, baik kode dan dokumen, menguji bagian-bagian baru (perubahan), dan menguji ulang bagian-bagian lama yang tidak diubah. Pemeliharaan bisa korektif dan adaptif.

Fitur penting dalam pemrograman

Kode yang ditulis untuk perangkat lunak harus sesuai dengan persyaratan pengguna. Suatu program dikatakan baik jika kode perangkat lunaknya sempurna atau mengandung kesalahan yang minim. Untuk kinerja perangkat lunak yang efektif, beberapa fitur tertentu diperlukan di hampir semua bahasa yang digunakan untuk menulis kode perangkat lunak. Fitur-fitur ini tercantum di bawah ini:

1. Sederhana.

Kode perangkat lunak harus ditulis dengan cara yang sederhana dan ringkas. Kesederhanaan harus dipertahankan dalam organisasi, implementasi, dan desain kode perangkat lunak.

2. Modularitas.

Memecah perangkat lunak menjadi beberapa modul tidak hanya membuatnya mudah dimengerti tetapi juga mudah untuk melakukan *debug*. Dengan fitur modularitas, segmen kode yang sama dapat digunakan kembali dalam satu atau lebih program perangkat lunak.

3. Desain.

Kode perangkat lunak dirancang dengan benar jika disajikan dengan cara yang tepat. Desain perangkat lunak harus diputuskan sebelum memulai menulis kode perangkat lunak. Menulis kode perangkat lunak dengan gaya yang spesifik dan konsisten membantu pengembang perangkat lunak lain dalam meninjaunya.

4. Efisiensi.

Suatu program dikatakan efisien jika menggunakan sumber daya yang tersedia secara optimal.

5. Kejelasan.

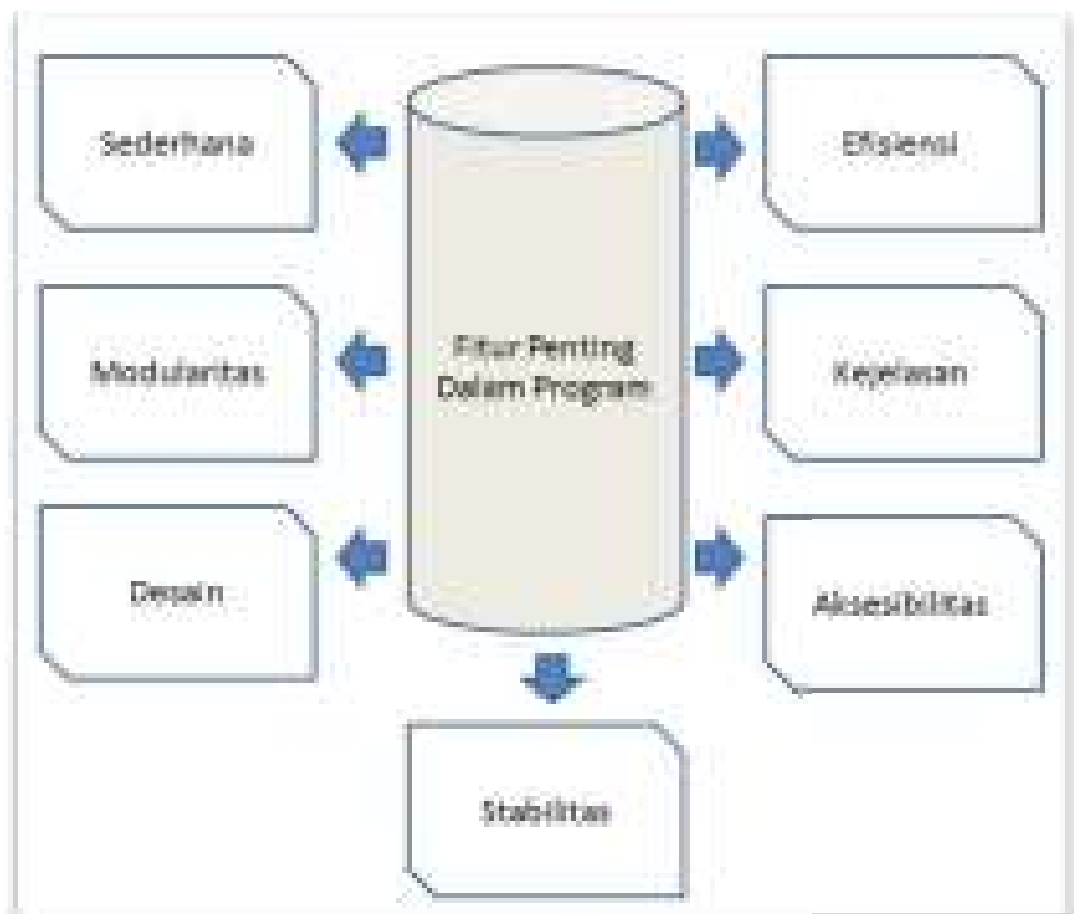
Kode perangkat lunak harus jelas sehingga pengembang dapat memahami program tanpa mengalami kerumitan. Kejelasan dapat dicapai dengan menggunakan fitur-fitur seperti kesederhanaan, keterbacaan dan modularitas. Perhatikan bahwa kejelasan terdiri dari kejelasan kode, kejelasan desain, dan kejelasan tujuan sehingga orang tahu apa yang terjadi di setiap level dalam program perangkat lunak.

6. Aksesibilitas.

Kode perangkat lunak harus ditulis sedemikian rupa sehingga komponen perangkat lunak (misalnya, file dan fungsi) mudah tersedia dan dapat diakses. Agar file dan fungsi dapat diakses, mereka harus memiliki nama yang bermakna serta adanya teks pendukung untuk setiap gambar. Demikian pula, harus ada *hyperlink* dan alat bantu navigasi untuk membantu pengguna dalam mencari informasi dari berbagai bagian kode perangkat lunak.

7. Stabilitas

Kode perangkat lunak dikatakan stabil jika mereka dapat bekerja dengan benar pada platform yang berbeda tanpa mempengaruhi tata letak dan konsistensi mereka. Untuk memeriksa stabilitas, kode perangkat lunak harus diuji untuk kesalahan dan ketidakkonsistenan.



Gambar 10.1. Fitur penting dalam program

Soal:

1. Uraikan secara ringkas tentang metode pemrograman terstruktur.
2. Jelaskan, mengapa tim programming harus memiliki kesepakatan terkait dengan penggunaan nama fungsi, variable, dan tipe data dalam membangun suatu aplikasi. Berikan contoh.
3. Jelaskan apa yang anda fahami tentang konsep rekursi. Berikan contoh.
4. Uraikan dan jelaskan klasifikasi dokumentasi perangkat lunak.

BAB XI

PENGUJIAN PERANGKAT LUNAK

Tujuan :

1. Mempelajari jenis-jenis pengujian yang dilakukan terhadap perangkat lunak
2. Mempelajari aspek-aspek dan strategi pengujian perangkat lunak
3. Mempelajari tool-tool yang dibutuhkan dan cara penggunaan tool tersebut dalam proses pengujian perangkat lunak.

Indikator keberhasilan :

1. Mahasiswa memahami pentingnya fase pengujian terhadap keberhasilan suatu produk perangkat lunak.
2. Mahasiswa memiliki kemampuan untuk membedakan beragam jenis pengujian dan target yang dicapainya.
3. Mahasiswa mampu membuat dokumen hasil review terhadap suatu perangkat lunak.

Materi :

Pengujian Perangkat Lunak adalah evaluasi perangkat lunak terhadap persyaratan yang dikumpulkan dari pengguna dan spesifikasi sistem yang diharapkan. Pengujian dilakukan pada tingkat fase dalam siklus hidup pengembangan perangkat lunak atau pada level modul dalam kode program. Pengujian perangkat lunak terdiri dari kegiatan validasi dan verifikasi.

Rencana Pengujian Perangkat Lunak

Rencana pengujian menjelaskan bagaimana pengujian akan dicapai. Ini adalah dokumen yang menentukan tujuan, ruang lingkup, dan metode

pengujian perangkat lunak. Ini menentukan tugas-tugas pengujian dan orang-orang yang terlibat dalam melaksanakan tugas-tugas itu, item pengujian, dan fitur yang akan diuji. Ini juga menggambarkan lingkungan untuk pengujian dan desain uji dan teknik pengukuran yang akan digunakan. Perhatikan bahwa rencana pengujian yang ditetapkan dengan benar adalah perjanjian antara penguji dan pengguna yang menguraikan peran pengujian dalam perangkat lunak.

Rencana pengujian lengkap membantu orang-orang yang tidak terlibat dalam kelompok uji untuk memahami mengapa validasi produk diperlukan dan bagaimana hal itu dilakukan. Namun, jika rencana pengujian tidak lengkap, mungkin tidak dapat memeriksa bagaimana perangkat lunak beroperasi saat diinstal pada sistem operasi yang berbeda atau ketika digunakan dengan perangkat lunak lain. Untuk menghindari masalah ini, IEEE menyatakan beberapa komponen yang harus dicakup dalam rencana pengujian. Komponen-komponen ini tercantum dalam Tabel 11.1 berikut ini.

Tabel 11.1 : Komponen-komponen rencana pengujian

Komponen	Tujuan
Tanggung jawab	Menugaskan tanggung jawab kepada orang yang berbeda dan membuat mereka tetap fokus.
Asumsi	Hindari salah tafsir terkait jadwal.
Pengujian	Memberikan abstrak dari seluruh proses dan menguraikan tes khusus. Lingkup, jadwal, dan durasi pengujian juga diuraikan.
Komunikasi	Rencana komunikasi (siapa, apa, kapan, bagaimana dengan orang-orang) dikerahkan
Analisis Risiko	Identifikasi area yang sangat kritis untuk mencapai kesuksesan.
Laporan kondisi cacat	Menentukan kondisi cacat yang harus didokumentasikan sehingga tidak terjadi pengujian dan perbaikan yang berulang.
Lingkungan	Menjelaskan data, antarmuka, area kerja, dan

	lingkungan teknis yang digunakan dalam pengujian. Semua ini ditentukan untuk mengurangi atau menghilangkan sumber kesalahpahaman dan potensi keterlambatan.
--	---

Rencana pengujian yang dikembangkan dengan cermat memfasilitasi pelaksanaan pengujian yang efektif, analisis kesalahan yang tepat, dan persiapan laporan kesalahan. Untuk mengembangkan rencana pengujian, sejumlah langkah diikuti, seperti yang tercantum di bawah ini:

1. **Tetapkan tujuan rencana pengujian (*Set objectives of test plan*).**

Sebelum mengembangkan rencana pengujian, perlu dipahami tujuannya. Tetapi, sebelum menentukan tujuan suatu rencana pengujian, perlu untuk menentukan tujuan perangkat lunak. Ini karena tujuan rencana pengujian sangat tergantung pada perangkat lunak. Misalnya, jika tujuan perangkat lunak adalah untuk memenuhi semua kebutuhan pengguna, maka rencana pengujian dihasilkan untuk memenuhi tujuan ini.

2. **Mengembangkan matriks uji (*Develop a test matrix*).**

Matriks uji menunjukkan komponen perangkat lunak yang akan diuji. Ini juga menentukan tes yang diperlukan untuk memeriksa komponen ini. Matriks uji juga digunakan sebagai bukti uji untuk menunjukkan bahwa ada tes untuk semua komponen perangkat lunak yang memerlukan pengujian. Selain itu, tes matriks digunakan untuk menunjukkan metode pengujian, yang digunakan untuk menguji seluruh perangkat lunak.

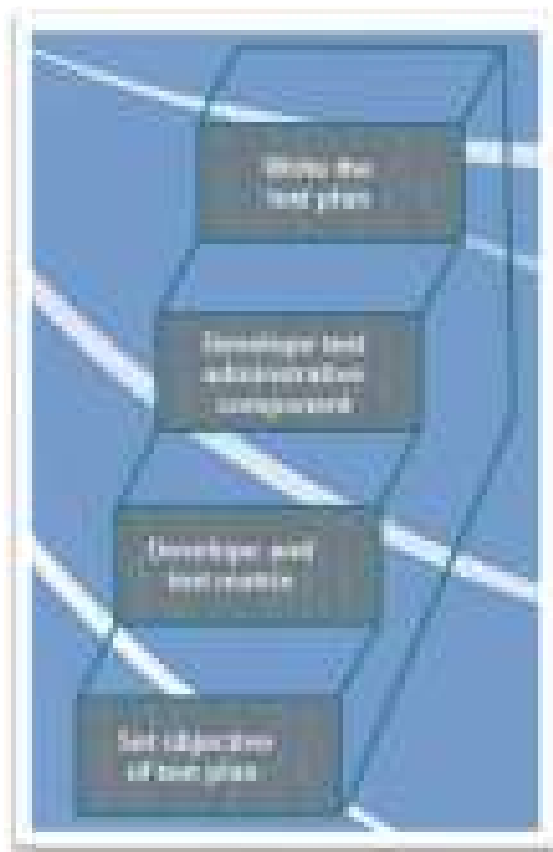
3. **Mengembangkan komponen administrasi pengujian (*Develop test administrative component*).**

Rencana pengujian harus disiapkan dalam waktu yang tetap sehingga pengujian perangkat lunak dapat dimulai sesegera mungkin. Tujuan komponen administratif dari rencana pengujian adalah untuk menentukan jadwal waktu dan sumber daya (orang-orang administrasi yang terlibat saat mengembangkan rencana pengujian) yang

diperlukan untuk melaksanakan rencana pengujian. Namun, jika rencana implementasi (rencana yang menggambarkan bagaimana proses dalam perangkat lunak dilakukan) dari perubahan perangkat lunak, rencana pengujian juga berubah. Dalam hal ini, jadwal untuk melaksanakan rencana pengujian juga akan terpengaruh.

4. **Tuliskan rencana pengujian (*Write the test plan*).**

Komponen rencana pengujian seperti tujuannya, matriks uji, dan komponen administratif didokumentasikan. Semua dokumen ini kemudian dikumpulkan bersama untuk membentuk rencana pengujian yang lengkap. Dokumen-dokumen ini disusun secara informal atau formal.



Gambar 11.1: Langkah-langkah yang terlibat dalam melakukan pengujian

Secara informal, semua dokumen dikumpulkan dan disimpan bersama. Penguji membaca semua dokumen untuk mengekstrak informasi yang

diperlukan untuk menguji perangkat lunak. Di sisi lain, secara formal, poin-poin penting diambil dari dokumen dan disimpan bersama. Ini memudahkan penguji untuk mengekstrak informasi penting, yang mereka perlukan selama pengujian perangkat lunak.

Rencana pengujian memiliki banyak bagian, seperti yang tercantum di bawah ini:

1. Ringkasan.

Menjelaskan tujuan dan fungsi perangkat lunak yang akan dilakukan. Ini juga menjelaskan tujuan rencana pengujian seperti mendefinisikan tanggung jawab, mengidentifikasi lingkungan pengujian dan memberikan detail lengkap dari sumber dari mana informasi dikumpulkan untuk mengembangkan rencana pengujian.

2. Lingkup pengujian.

Menentukan fitur dan kombinasi fitur, yang akan diuji. Fitur-fitur ini dapat mencakup manual pengguna atau dokumen sistem. Ini juga menentukan fitur dan kombinasinya yang tidak diuji.

3. Metodologi pengujian.

Menentukan jenis-jenis pengujian yang diperlukan untuk fitur pengujian dan kombinasi fitur-fitur ini seperti pengujian regresi dan penekanan pengujian. Ini juga menyediakan deskripsi sumber data uji bersama dengan bagaimana data uji berguna untuk memastikan bahwa pengujian memadai seperti pemilihan nilai batas atau nol. Selain itu, ini menjelaskan prosedur untuk mengidentifikasi dan merekam hasil pengujian.

4. Fase uji.

Mengidentifikasi berbagai jenis pengujian seperti pengujian unit, pengujian integrasi dan memberikan deskripsi singkat tentang proses yang digunakan untuk melakukan pengujian ini. Selain itu, mengidentifikasi penguji yang bertanggung jawab untuk melakukan pengujian dan memberikan deskripsi rinci tentang sumber dan jenis data yang akan digunakan. Ini juga menjelaskan prosedur

mengevaluasi hasil pengujian dan menjelaskan produk kerja, yang dimulai atau diselesaikan pada fase ini.

5. Lingkungan pengujian.

Mengidentifikasi perangkat keras, perangkat lunak, alat pengujian otomatis; sistem operasi, compilers dan situs yang dibutuhkan untuk melakukan pengujian, serta kebutuhan staf dan pelatihan.

6. Jadwal.

Memberikan jadwal rinci kegiatan pengujian dan menetapkan tanggung jawab kepada masing-masing orang. Selain itu, ini menunjukkan ketergantungan kegiatan pengujian dan kerangka waktu untuk mereka.

7. Persetujuan dan distribusi.

Mengidentifikasi individu yang menyetujui rencana pengujian dan hasilnya. Ini juga mengidentifikasi orang-orang kepada siapa dokumen rencana pengujian didistribusikan.

Validasi Perangkat Lunak

Validasi adalah proses memeriksa apakah perangkat lunak memenuhi persyaratan pengguna. Ini dilakukan pada akhir SDLC. Jika perangkat lunak sesuai dengan persyaratan yang dibuat, itu divalidasi.

- Validasi memastikan produk yang sedang dikembangkan sesuai dengan kebutuhan pengguna.
- Validasi menjawab pertanyaan. "Apakah kita mengembangkan produk yang mencoba semua kebutuhan pengguna dari perangkat lunak ini?".
- Validasi menekankan pada kebutuhan pengguna.

Verifikasi Perangkat Lunak

Verifikasi adalah proses konfirmasi jika perangkat lunak memenuhi persyaratan bisnis, dan dikembangkan sesuai dengan spesifikasi dan metodologi yang tepat.

- Verifikasi memastikan produk yang sedang dikembangkan sesuai dengan spesifikasi desain.
- Verifikasi menjawab pertanyaan - "Apakah kita mengembangkan produk ini dengan mengikuti semua spesifikasi desain dengan tegas?"
- Verifikasi berkonsentrasi pada desain dan spesifikasi sistem.

Sasaran pengujian adalah:

- Kesalahan (*error*). Ini adalah kesalahan pengkodean yang sebenarnya yang dibuat oleh pengembang. Selain itu, ada perbedaan dalam output perangkat lunak dan output yang diinginkan, dianggap sebagai kesalahan.
- *Fault*. Ketika kesalahan ada kesalahan terjadi. Kesalahan, juga dikenal sebagai bug, adalah hasil dari kesalahan yang dapat menyebabkan sistem gagal.
- Kegagalan (*failure*). Kegagalan dikatakan ketidakmampuan sistem untuk melakukan tugas yang diinginkan. Kegagalan terjadi ketika kesalahan ada dalam sistem.

Pengujian Manual dan Otomatis

Pengujian dapat dilakukan secara manual atau menggunakan alat pengujian otomatis:

- Pengujian Manual. Proses pengujian ini dilakukan tanpa bantuan alat pengujian otomatis. Penguji perangkat lunak menyiapkan kasus uji untuk berbagai bagian dan tingkat kode, menjalankan tes dan melaporkan hasilnya kepada manajer.

Pengujian manual adalah memakan waktu dan sumber daya. Penguji perlu mengkonfirmasi apakah atau tidak kasus uji yang benar digunakan. Sebagian besar pengujian melibatkan pengujian manual.

- Pengujian Otomatis. Prosedur pengujian ini yang dilakukan dengan bantuan alat pengujian otomatis. Keterbatasan dengan pengujian manual dapat diatasi menggunakan alat uji otomatis.

Sebuah aktifitas pengujian perlu memeriksa apakah halaman web dapat dibuka di Internet Explorer. Ini dapat dengan mudah dilakukan dengan pengujian manual. Tetapi untuk memeriksa apakah web-server dapat memuat 1 juta pengguna, sangat tidak mungkin untuk menguji secara manual.

Ada perangkat lunak dan alat perangkat keras yang membantu tester dalam melakukan pengujian beban, pengujian tegangan, pengujian regresi.

Pendekatan dalam Pengujian

Tes dapat dilakukan berdasarkan dua pendekatan, yaitu:

1. Pengujian fungsionalitas
2. Pengujian implementasi

Ketika fungsionalitas sedang diuji tanpa mengambil implementasi yang sebenarnya dalam keprihatinan itu dikenal sebagai pengujian black-box. Sisi lain dikenal sebagai pengujian kotak putih di mana tidak hanya fungsi yang diuji tetapi cara penerapannya juga dianalisis.

Tes yang lengkap adalah metode yang paling diinginkan untuk pengujian yang sempurna. Setiap nilai yang mungkin dalam kisaran nilai input dan output yang diuji.

Jenis-Jenis Metode Pengujian

1. Black-box Testing

Ini dilakukan untuk menguji fungsionalitas program dan juga disebut pengujian '*Behavioral*'. Penguji dalam hal ini, memiliki satu set nilai input dan hasil yang diinginkan masing-masing. Pada saat memberikan masukan, jika output sesuai dengan hasil yang diinginkan, program diuji 'oke', dan bermasalah sebaliknya.



Gambar 11.2: Ilustrasi proses *blackbox testing*

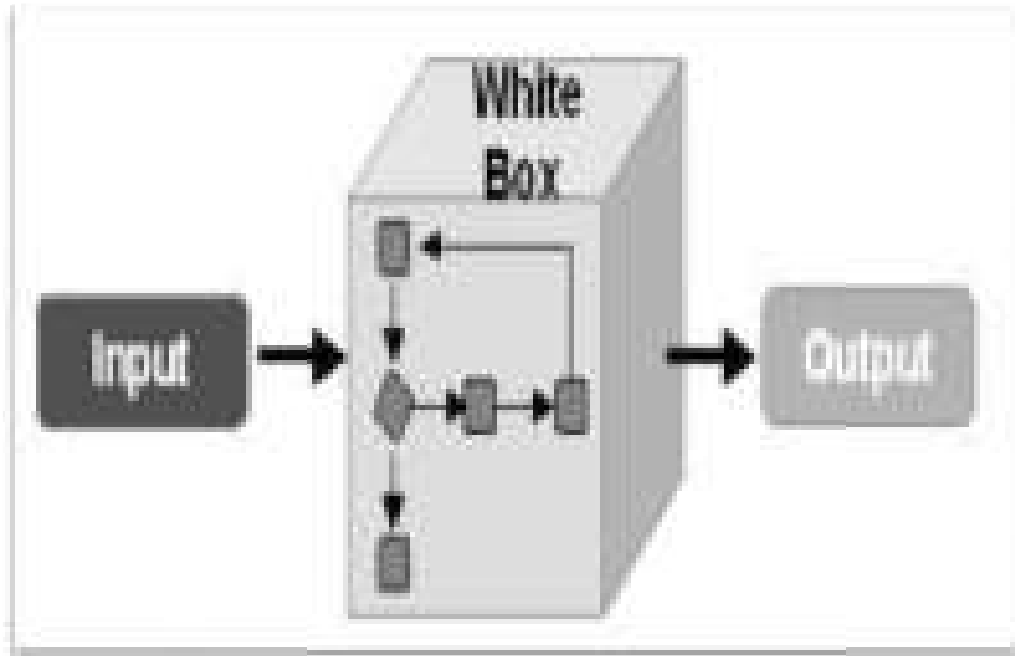
Dalam metode pengujian ini, desain dan struktur kode tidak diketahui oleh penguji, dan insinyur pengujian dan pengguna akhir melakukan tes ini pada perangkat lunak.

Teknik pengujian black-box:

- *Equivalence class*. Masukan dibagi menjadi kelas-kelas serupa. Jika satu elemen dari suatu kelas lulus tes, diasumsikan bahwa semua kelas dilewatkan.
- *Boundary values*. Masukan dibagi menjadi nilai akhir yang lebih tinggi dan lebih rendah. Jika nilai-nilai ini lulus tes, diasumsikan bahwa semua nilai di antara keduanya juga akan lulus.
- *Cause-effect graphing*. Dalam kedua metode sebelumnya, hanya satu nilai input pada satu waktu yang diuji. Penyebab (input) - Efek (output) adalah teknik pengujian di mana kombinasi nilai input diuji dengan cara yang sistematis.
- *Pair-wise Testing*. Perilaku perangkat lunak tergantung pada beberapa parameter. Dalam pengujian berpasangan, beberapa parameter diuji secara berpasangan untuk nilainya yang berbeda.
- *State-based testing*. Perubahan sistem menyatakan pada penyediaan input. Sistem ini diuji berdasarkan status dan masukan mereka.

2. White-box Testing

Hal ini dilakukan untuk menguji program dan implementasinya, untuk meningkatkan efisiensi atau struktur kode. Ini juga dikenal sebagai pengujian 'Struktural'.



Gambar 11.3: Ilustrasi White-Box Testing

Dalam metode pengujian ini, desain dan struktur kode diketahui oleh penguji. Programmer melakukan tes ini pada kode.

Di bawah ini adalah beberapa teknik pengujian *White-box*:

- *Control-flow testing*. Tujuan pengujian aliran-kontrol untuk menyiapkan kasus uji yang mencakup semua pernyataan dan kondisi cabang. Kondisi cabang diuji untuk keduanya benar dan salah, sehingga semua pernyataan dapat dicakup.
- *Data-flow testing*. Teknik pengujian ini menekankan untuk mencakup semua variabel data yang termasuk dalam program. Ini menguji di mana variabel dinyatakan dan didefinisikan dan di mana mereka digunakan atau diubah.

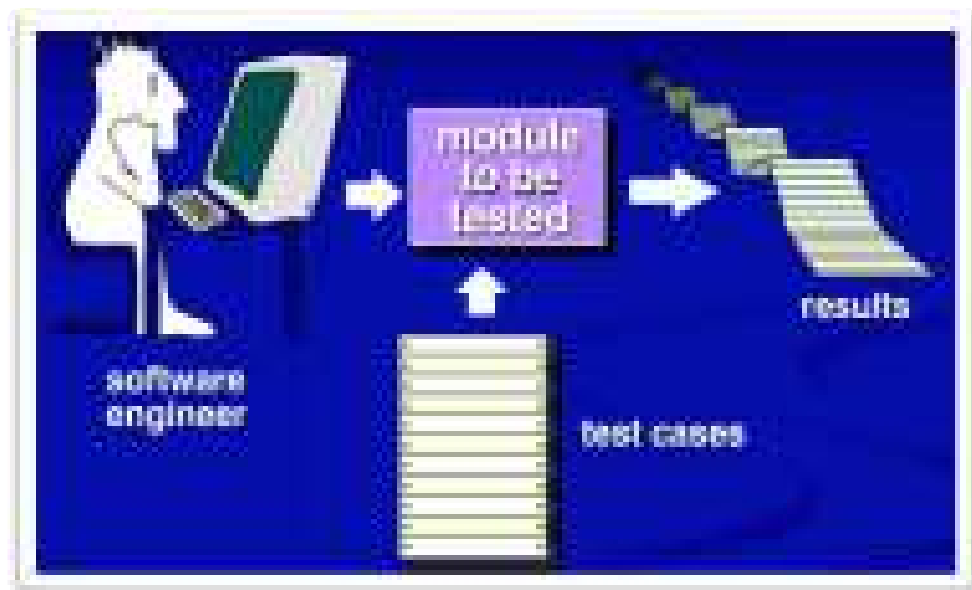
Tingkatan Pengujian

Pengujian itu sendiri dapat didefinisikan pada berbagai tingkat SDLC. Proses pengujian berjalan paralel dengan pengembangan perangkat lunak. Sebelum melompat ke tahap berikutnya, tahap diuji, divalidasi dan diverifikasi.

Pengujian secara terpisah dilakukan hanya untuk memastikan bahwa tidak ada bug tersembunyi atau masalah yang tersisa dalam perangkat lunak. Perangkat lunak diuji pada berbagai tingkatan:

1. Pengujian Unit (*Unit Testing*)

Saat coding, programmer melakukan beberapa tes pada unit program untuk mengetahui apakah itu bebas dari kesalahan. Pengujian dilakukan dengan pendekatan pengujian *white-box*. Pengujian unit membantu pengembang memutuskan bahwa masing-masing unit program berfungsi sesuai kebutuhan dan bebas dari kesalahan.



Gambar 11.4: Ilustrasi Pengujian Unit

Pengujian unit tidak hanya dilakukan satu kali selama pengembangan perangkat lunak, tetapi diulang setiap kali perangkat lunak dimodifikasi atau digunakan dalam lingkungan baru. Beberapa poin lain yang dicatat tentang pengujian unit tercantum di bawah ini:

- Setiap unit diuji secara terpisah terlepas dari unit perangkat lunak lain.
- Pengembang melakukan sendiri proses pengujian ini.
- Metode pengujian white-box digunakan dalam pengujian ini.

Pengujian unit digunakan untuk memverifikasi kode yang dihasilkan selama pengkodean perangkat lunak dan bertanggung jawab untuk menilai kebenaran unit kode sumber tertentu. Selain itu, pengujian unit melakukan fungsi-fungsi berikut:

- Menguji semua jalur kontrol untuk mengungkap kesalahan maksimum yang terjadi selama pelaksanaan kondisi yang ada di unit yang sedang diuji.
- Memastikan bahwa semua pernyataan dalam unit telah dieksekusi setidaknya satu kali.
- Menguji struktur data (seperti tumpukan, antrian) yang mewakili hubungan antara elemen data individual.
- Memeriksa berbagai input yang diberikan kepada unit. Ini karena setiap rentang input memiliki nilai maksimum dan minimum dan input yang diberikan harus berada dalam kisaran nilai-nilai ini.
- Memastikan bahwa data yang dimasukkan dalam variabel adalah tipe data yang sama seperti yang didefinisikan dalam unit.
- Memeriksa semua perhitungan aritmatika yang ada di unit dengan semua kemungkinan kombinasi nilai input.

2. Tes integrasi

Bahkan jika unit perangkat lunak bekerja dengan baik secara individual, ada kebutuhan untuk mencari tahu apakah unit-unit tersebut jika digabungkan bersama juga akan bekerja tanpa kesalahan. Misalnya, passing argumen dan pembaruan data, dll.

3. Pengujian Sistem

Perangkat lunak ini dikompilasi sebagai produk dan kemudian diuji secara keseluruhan. Ini dapat diselesaikan dengan menggunakan satu atau beberapa tes berikut:

- Pengujian Fungsionalitas - Menguji semua fungsi perangkat lunak terhadap persyaratan.
- Pengujian kinerja - Pengujian ini membuktikan seberapa efisien perangkat lunak. Ini menguji efektivitas dan waktu rata-rata yang diambil oleh perangkat lunak untuk melakukan tugas yang diinginkan. Pengujian kinerja dilakukan dengan cara pengujian beban dan pengujian stres di mana perangkat lunak diletakkan di bawah beban pengguna dan data yang tinggi di bawah berbagai kondisi lingkungan.
- Keamanan & Portabilitas - Pengujian ini dilakukan ketika perangkat lunak dimaksudkan untuk bekerja pada berbagai platform dan diakses oleh sejumlah orang.

4. Tes penerimaan (*acceptance*)

Ketika perangkat lunak siap untuk diserahkan kepada pelanggan, ia harus melalui tahap terakhir pengujian di mana ia diuji untuk interaksi dan tanggapan pengguna. Ini penting karena bahkan jika perangkat lunak cocok dengan semua persyaratan pengguna dan jika pengguna tidak suka cara tampilannya atau bekerja, itu mungkin ditolak.

- Pengujian alfa - Tim pengembang sendiri melakukan pengujian alfa dengan menggunakan sistem seolah-olah digunakan di lingkungan kerja. Mereka mencoba mencari tahu bagaimana reaksi pengguna terhadap beberapa tindakan dalam perangkat lunak dan bagaimana sistem harus menanggapi masukan.
- Pengujian beta - Setelah perangkat lunak diuji secara internal, itu diserahkan kepada pengguna untuk menggunakannya di bawah

lingkungan produksi mereka hanya untuk tujuan pengujian. Ini belum menjadi produk yang dikirim. Pengembang berharap bahwa pengguna pada tahap ini akan membawa masalah kecil, yang dilewati untuk hadir.

5. Pengujian Regresi

Setiap kali produk perangkat lunak diperbarui dengan kode baru, fitur atau fungsionalitas, itu diuji secara menyeluruh untuk mendeteksi apakah ada dampak negatif dari kode yang ditambahkan. Ini dikenal sebagai pengujian regresi.

Dokumentasi Pengujian

Dokumen pengujian disiapkan pada tahap yang berbeda, meliputi:

1. Sebelum pengujian

Pengujian dimulai dengan kasus uji hasil. Dokumen-dokumen berikut diperlukan untuk referensi:

- *SRS Document*. Dokumen Persyaratan Fungsional
- *Test-Policy-document*. Ini menjelaskan seberapa jauh pengujian harus dilakukan sebelum melepaskan produk.
- *Test-Strategy-document*. Ini menyebutkan aspek detail dari tim uji, matriks tanggung jawab dan hak / tanggung jawab manajer tes dan insinyur uji.
- *Traceability-Matrix-document*. Ini adalah dokumen SDLC, yang terkait dengan proses pengumpulan kebutuhan. Ketika persyaratan baru datang, mereka ditambahkan ke matriks ini. Matriks ini membantu penguji mengetahui sumber kebutuhan. Mereka dapat dilacak ke depan dan ke belakang.

2. Saat pengujian

Dokumen berikut mungkin diperlukan saat pengujian dimulai dan sedang dilakukan:

- Dokumen *Test Case*. Dokumen ini berisi daftar tes yang harus dilakukan. Ini termasuk rencana uji unit, rencana uji integrasi, rencana uji sistem dan rencana uji penerimaan.
- Tes Deskripsi. Dokumen ini adalah deskripsi mendetail tentang semua kasus dan prosedur pengujian untuk melaksanakannya.
- *Test case report*. Dokumen ini berisi laporan kasus uji sebagai hasil dari tes.
- *Test logs*. Dokumen ini berisi log pengujian untuk setiap laporan uji coba.

3. Sesudah pengujian

Dokumen yang diperlukan setelah pengujian adalah Ringkasan Pengujian. Dokumen ini adalah analisis kolektif semua laporan dan catatan pengujian. Ini merangkum dan menyimpulkan jika perangkat lunak siap diluncurkan. Perangkat lunak ini dirilis di bawah sistem kontrol versi jika siap diluncurkan.

Perbedaan Pengujian dengan Kontrol dan Jaminan Kualitas & Audit

Pengujian perangkat lunak adalah berbeda dengan jaminan kualitas perangkat lunak, kontrol kualitas perangkat lunak, dan pengauditan perangkat lunak.

- Jaminan kualitas perangkat lunak - Ini adalah sarana pemantauan proses pengembangan perangkat lunak, yang memastikan bahwa semua tindakan diambil sesuai standar organisasi. Pemantauan ini dilakukan untuk memastikan bahwa metode pengembangan perangkat lunak yang tepat diikuti.
- Kontrol kualitas perangkat lunak - Ini adalah sistem untuk menjaga kualitas produk perangkat lunak. Ini mungkin termasuk aspek fungsional dan non-fungsional dari produk perangkat lunak, yang meningkatkan niat baik organisasi. Sistem ini memastikan bahwa pelanggan menerima produk berkualitas untuk kebutuhan mereka dan produk disertifikasi sebagai *'fit for use'*.

- Audit perangkat lunak - Ini adalah tinjauan prosedur yang digunakan oleh organisasi untuk mengembangkan perangkat lunak. Sebuah tim auditor, independen dari tim pengembangan memeriksa proses perangkat lunak, prosedur, persyaratan dan aspek lain dari SDLC. Tujuan dari audit perangkat lunak adalah untuk memeriksa perangkat lunak dan proses pengembangannya, baik sesuai standar, aturan dan peraturan.

Soal

1. Jelaskan hal apa saja yang mendasari pentingnya dilakukan pengujian terhadap perangkat lunak.
2. Jelaskan hal apa yang dimaksud dengan validasi dan verifikasi dalam konteks pengujian perangkat lunak.
3. Jelaskan secara lengkap mekanisme white-box testing.
4. Jelaskan secara lengkap mekanisme black-box testing.
5. Tugas:

Anda diminta untuk mendapatkan suatu aplikasi bisnis yang bersifat on-line. Lakukanlah review terhadap perangkat lunak tersebut. Buatlah laporan lengkap hasil review sesuai dengan struktur penulisan yang diberikan oleh dosen. Selanjutnya sesuai jadwal akan dilakukan presentasi secara berkelompok.

BAB XII

PEMELIHARAAN PERANGKAT LUNAK

Tujuan :

1. Mempelajari konsep pemeliharaan perangkat lunak
2. Mempelajari bentuk-bentuk pemeliharaan perangkat lunak
3. Mempelajari beragam aktifitas yang dilakukan dalam kegiatan pemeliharaan

Indikator keberhasilan :

1. Mahasiswa memahami pentingnya fase pemeliharaan untuk menjamin keberlanjutan suatu perangkat lunak
2. Mahasiswa mampu menjelaskan klasifikasi pemeliharaan perangkat lunak
3. Mahasiswa mengenali permasalahan yang umum dialami dalam kegiatan pemeliharaan perangkat lunak
4. Mahasiswa mampu menjelaskan aktifitas yang ada dalam proses pemeliharaan perangkat lunak.

Materi :

Istilah pemeliharaan perangkat lunak digunakan untuk menggambarkan kegiatan yang terjadi setelah pengiriman (*delivery*) produk perangkat lunak kepada pelanggan. Fase pemeliharaan dari siklus hidup sistem adalah periode waktu di mana suatu produk perangkat lunak telah menjalankan tugasnya (beroperasi). Biasanya siklus pengembangan sistem membutuhkan waktu 1 hingga 2 tahun, sedangkan fase pemeliharaan mencapai 5 hingga 10 tahun.

Dalam istilah umum, pemeliharaan berarti memperbaiki hal-hal yang rusak atau tidak normal. Kegiatan pemeliharaan mencakup kegiatan membuat peningkatan pada sistem, mengadaptasi produk ke lingkungan baru, dan memperbaiki masalah. Karena perubahan tidak dapat dihindari, mekanisme harus dikembangkan untuk mengevaluasi, mengendalikan, dan membuat modifikasi. Tujuan pemeliharaan perangkat lunak adalah mempertahankan kualitasnya dari waktu ke waktu dengan meningkatkan basis pelanggan, memenuhi persyaratan tambahan, membuatnya lebih mudah digunakan dan lebih efisien serta menggunakan teknologi yang lebih baru.

Mengapa Pemeliharaan Perlu di Lakukan ?

Pemeliharaan mesti dilakukan untuk memastikan bahwa perangkat lunak terus memenuhi persyaratan yang diinginkan oleh pengguna. Pemeliharaan berlaku untuk semua perangkat lunak yang dikembangkan menggunakan model siklus hidup perangkat lunak apa pun. Produk perangkat lunak mengalami perubahan karena tindakan perangkat lunak korektif dan non-korektif termasuk adanya dinamika dalam teknologi dan organisasi. Pemeliharaan secara terus menerus dilakukan untuk hal-hal berikut ini:

1. Melakukan koreksi terhadap kesalahan yang ditemukan.
2. Meningkatkan kualitas rancangan perangkat lunak.
3. Mengimplementasikan perangkat tambahan/pendukung.
4. Sebagai keperluan untuk antar muka dengan perangkat lunak lain.
5. Menyesuaikan program sehingga perangkat keras, perangkat lunak, fitur sistem, dan fasilitas telekomunikasi yang berbeda dapat digunakan secara bersama (kolaborasi).
6. Melakukan migrasi dari perangkat lunak sebelumnya.

Selama berlangsungnya waktu pemakaian suatu perangkat lunak, mungkin diperlukan dimodifikasi agar dapat beroperasi di lingkungan yang berbeda. Untuk memigrasikannya ke lingkungan baru, pemelihara perlu menentukan tindakan yang diperlukan untuk menyelesaikan kegiatan migrasi, dan kemudian mengembangkan dan mendokumentasikan langkah-langkah yang diperlukan untuk

menjalani rencana migrasi yang mencakup persyaratan migrasi, tool migrasi, konversi produk dan data, eksekusi, verifikasi, dan dukungan (*support*). Perangkat lunak yang mengalami migrasi juga dapat melibatkan sejumlah aktivitas tambahan seperti:

- Penjelasan maksud kegiatan: pernyataan mengapa lingkungan lama tidak lagi didukung, diikuti oleh deskripsi lingkungan baru dan tanggal ketersediaan hasilnya.
- operasi paralel: menyediakan lingkungan lama dan baru secara bersamaan sehingga pengguna mengalami kelancaran dalam proses transisi ke lingkungan baru.
- pemberitahuan waktu penyelesaian: ketika migrasi terjadwal telah selesai, maka pemberitahuan disampaikan ke semua pihak yang terkait.
- review pasca tindakan: penilaian terhadap keberhasilan operasional secara paralel dan dampak perubahan ke lingkungan baru;
- pengarsipan data: menyimpan data dari perangkat lunak yang terdahulu.

7. Menghentikan pemakaian perangkat lunak yang digunakan saat ini.

Setelah perangkat lunak mencapai akhir masa pakainya, maka untuk selanjutnya harus dihentikan. Kegiatan analisa harus dilakukan untuk membantu dalam membuat keputusan penghentian tersebut. Hasil analisa ini harus dimasukkan dalam rencana penghentian pemakaian, yang mencakup persyaratan penghentian, dampak, alternative penggantian, jadwal penggantian, dan upaya yang mesti dilakukan. Menghentikan pemakaian perangkat lunak memerlukan sejumlah aktivitas yang mirip dengan proses migrasi.

Terdapat lima karakteristik utama dari aktivitas tenaga pemelihara sistem, antara lain:

- memiliki kendali atas fungsi sehari-hari yang bersifat regular dari perangkat lunak

- memiliki kendali atas terjadinya berbagai modifikasi terhadap perangkat lunak
- menyempurnakan fungsi dan fitur yang ada
- mengidentifikasi berbagai potensi ancaman keamanan dan memperbaiki bila ada kerentanan sistem
- mencegah terjadinya penurunan kinerja perangkat lunak ke tingkat yang tidak dapat ditoleransi oleh sistem.

Bentuk-bentuk Pemeliharaan

Sepanjang masa pakai perangkat lunak, jenis perawatan dapat bervariasi berdasarkan sifatnya. Hal ini mungkin hanya tugas pemeliharaan rutin karena beberapa *bug* ditemukan oleh beberapa pengguna atau mungkin merupakan peristiwa besar dalamnya sendiri berdasarkan ukuran atau sifat pemeliharaan. Berikut ini adalah beberapa jenis perawatan berdasarkan karakteristik mereka:

1. Pemeliharaan Korektif.

Ini termasuk modifikasi dan perbaruan yang dilakukan untuk memperbaiki atau memperbaiki masalah, yang ditemukan oleh pengguna atau disimpulkan oleh laporan kesalahan pengguna. Pemeliharaan korektif berkaitan dengan perbaikan kesalahan atau cacat yang ditemukan pada fungsi sistem sehari-hari. Kerusakan dapat terjadi karena kesalahan dalam desain perangkat lunak, logika dan pengkodean. Kesalahan desain terjadi ketika perubahan yang dilakukan pada perangkat lunak tidak benar, tidak lengkap, salah dalam mengkomunikasikan, atau permintaan perubahan disalah pahami. Kesalahan logis dihasilkan dari pengujian dan kesimpulan yang tidak valid, implementasi spesifikasi desain yang salah, aliran logika yang salah, atau uji data yang tidak lengkap. Semua kesalahan ini, disebut sebagai kesalahan residual, mencegah perangkat lunak untuk memenuhi spesifikasi yang disepakati. Perhatikan bahwa

kebutuhan pemeliharaan korektif biasanya dimulai oleh laporan *bug* yang dibuat oleh pengguna.

Dalam hal terjadi kegagalan sistem karena adanya kesalahan, tindakan diambil untuk mengembalikan operasional sistem perangkat lunak. Pendekatan dalam pemeliharaan korektif adalah dengan menemukan spesifikasi asli untuk menentukan apa yang semula dirancang untuk dilakukan oleh sistem. Namun, karena tekanan dari manajemen, tim pemeliharaan terkadang menggunakan perbaikan darurat yang dikenal sebagai **penambalan**. Pemeliharaan korektif menyumbang 20% dari semua kegiatan pemeliharaan.

2. Pemeliharaan Adaptif.

Ini termasuk modifikasi dan pembaruan yang diterapkan untuk menjaga agar produk perangkat lunak tetap mutakhir dan disesuaikan dengan dunia teknologi dan lingkungan bisnis yang terus berubah. Pemeliharaan adaptif adalah implementasi perubahan di bagian sistem, yang telah dipengaruhi oleh perubahan yang terjadi di bagian lain pada sistem. Pemeliharaan adaptif terdiri dari aktifitas mengadaptasi perangkat lunak untuk perubahan di lingkungan seperti perangkat keras atau sistem operasi. Istilah lingkungan dalam konteks ini mengacu pada kondisi dan pengaruh yang terjadi (dari luar) pada sistem. Misalnya, aturan bisnis, pola kerja, dan kebijakan pemerintah memiliki dampak signifikan pada sistem perangkat lunak. Misalnya, kebijakan pemerintah untuk transaksi hanya menggunakan 'mata uang Rupiah' akan memiliki efek signifikan pada sistem perangkat lunak. Penerimaan perubahan ini akan meminta bank dalam negeri untuk membuat perubahan signifikan dalam sistem perangkat lunak mereka untuk mengakomodasi mata uang ini. Akun pemeliharaan adaptif untuk 25% dari semua kegiatan pemeliharaan.

3. *Perfective Maintenance*.

Pemeliharaan *Perfective* terutama berkaitan dengan penerapan persyaratan pengguna yang baru atau diubah. Pemeliharaan *Perfective* mencakup tugas melakukan peningkatan fungsional pada

sistem di samping aktifitas untuk meningkatkan kinerja bahkan ketika perubahan belum disarankan oleh adanya kesalahan. Ini termasuk meningkatkan fungsi dan efisiensi kode serta mengubah fungsi sistem sesuai kebutuhan pengguna yang berubah-ubah.

Contoh pemeliharaan *Perfective* termasuk memodifikasi program penggajian untuk memasukkan item perubahan komponen pembayaran dan menambahkan laporan baru dalam sistem analisis penjualan. Pemeliharaan *Perfective* menyumbang sebesar 50%, yaitu yang terbesar dari semua kegiatan pemeliharaan.

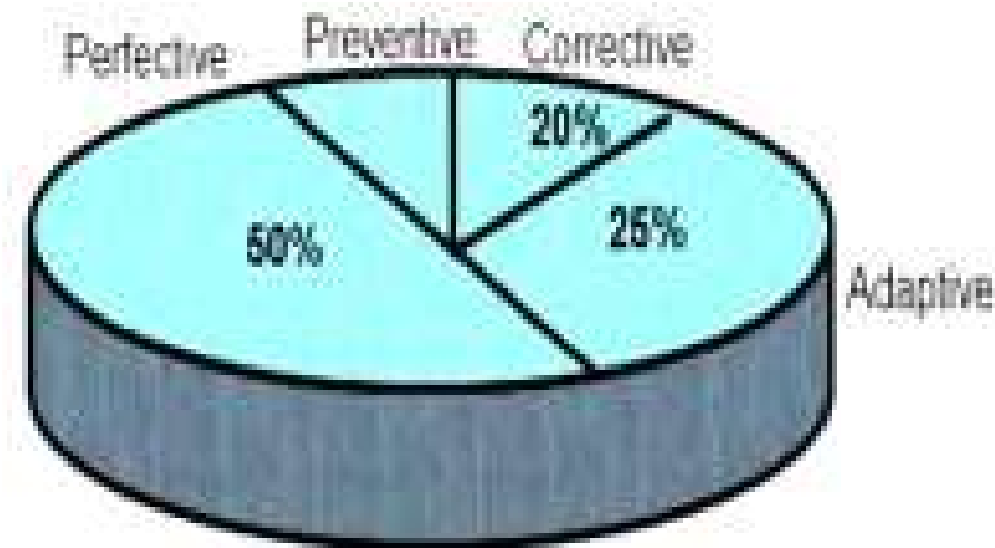
4. Perawatan Preventif.

Ini termasuk modifikasi dan perbaruan untuk mencegah masalah perangkat lunak di masa mendatang. Hal ini bertujuan untuk menghadapi masalah, yang tidak signifikan pada saat ini tetapi dapat menyebabkan masalah serius di masa depan. Pemeliharaan preventif melibatkan kegiatan untuk mencegah terjadinya kesalahan. Ini cenderung mengurangi kompleksitas perangkat lunak sehingga meningkatkan pemahaman terhadap program dan meningkatkan pemeliharaan perangkat lunak. Ini terdiri dari pembaruan dokumentasi, optimisasi kode, dan restrukturisasi kode. Pembaruan dokumentasi melibatkan modifikasi dokumen yang dipengaruhi oleh perubahan agar sesuai dengan kondisi sistem saat ini. Optimasi kode melibatkan memodifikasi program untuk eksekusi yang lebih cepat atau penggunaan ruang penyimpanan yang efisien. Restrukturisasi kode melibatkan transformasi struktur program untuk mengurangi kompleksitas dalam kode sumber dan membuatnya lebih mudah untuk dipahami.

Pemeliharaan preventif terbatas hanya pada organisasi pemeliharaan dan tidak ada permintaan eksternal yang diperoleh untuk jenis pemeliharaan ini. Perawatan preventif hanya memiliki bobot 5% dari semua aktivitas perawatan.

Sudah jelas bahwa kegiatan pemeliharaan mengkonsumsi porsi besar dari total anggaran siklus hidup. Tidak jarang pemeliharaan perangkat lunak untuk memperhitungkan 70 persen dari total biaya siklus hidup sistem (dengan pengembangan membutuhkan 30 persen). Sebagai aturan umum, distribusi upaya untuk pemeliharaan perangkat lunak mencakup 60 persen dari anggaran pemeliharaan untuk peningkatan, masing-masing 20 persen untuk penyesuaian dan koreksi.

Sulit menemukan angka terbaru untuk upaya relatif yang ditujukan untuk berbagai jenis pemeliharaan ini. Sebuah survei yang dilakukan oleh Lientz dan Swanson menemukan bahwa sekitar 50% dari perawatan adalah perfective, 25% adaptif dan 20% korektif, seperti yang ditunjukkan pada gambar berikut ini.



Gambar 12.1: Komposisi berbagai jenis pemeliharaan (*maintenance*)

Jika pemeliharaan menghabiskan 70 persen dari total upaya siklus hidup yang dikhususkan untuk produk perangkat lunak tertentu, dan jika 60 persen pemeliharaan digunakan untuk meningkatkan produk, maka 42 persen dari total upaya siklus hidup untuk produk tersebut didedikasikan untuk peningkatan produk. Mengingat perspektif ini, jelas bahwa produk yang dikirim ke pelanggan pada akhir siklus pengembangan hanyalah versi

awal dari sistem. Beberapa penulis menyarankan bahwa model siklus hidup yang sesuai untuk perangkat lunak adalah: pengembangan -> evolusi -> evolusi -> evolusi.

Permasalahan Selama Pemeliharaan

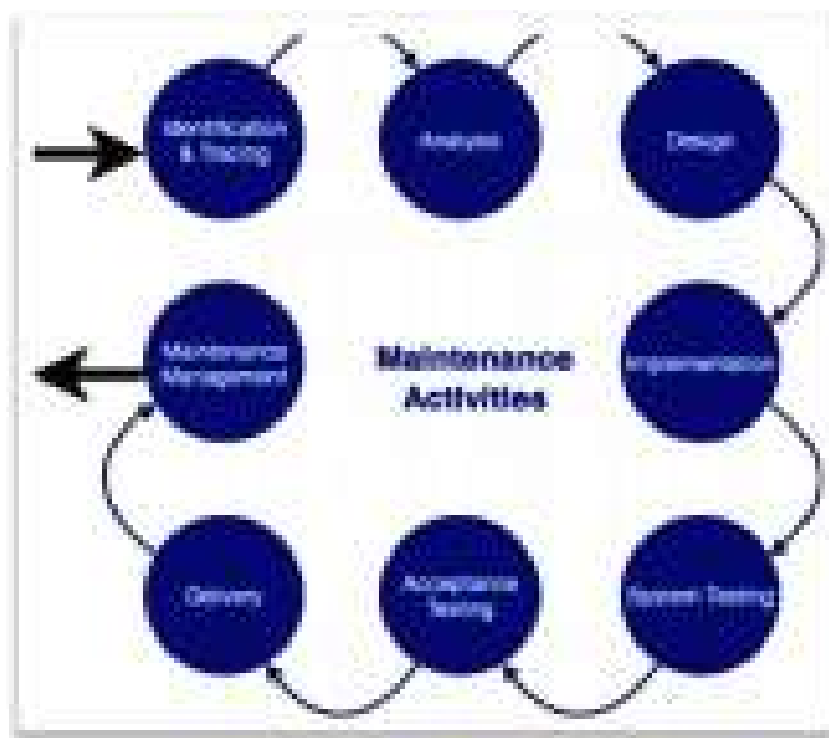
Perspektif ini membuat semakin jelas bahwa tujuan utama pengembangan adalah menghasilkan sistem yang dapat terus bertahan. Tingginya biaya pemeliharaan adalah karena sejumlah masalah yang tercantum di bawah ini:

1. Perawatan dipandang tidak sama berharganya dengan aktifitas pengembangan sistem. Hal ini dianggap selesai cukup dengan keterampilan atau pengalaman.
2. Pengguna tidak sepenuhnya sadar akan masalah pemeliharaan atau biayanya yang tinggi.
3. Beberapa alat dan teknik tersedia untuk maintenance.
4. Rencana pengujian yang baik masih belum memadai.
5. Standar, prosedur, dan pedoman tidak didefinisikan dengan baik dan ditegakkan. Program-program yang dipelihara mungkin telah dikembangkan bertahun-tahun yang lalu tanpa teknik modern.
6. Pemeliharaan dipandang sebagai kejahatan yang diperlukan, sering diturunkan ke programmer junior. Tidak ada klasifikasi pekerjaan manajer pemeliharaan di bidang MIS.
7. Program sering dipertahankan tanpa mempedulikan struktur dan dokumentasi. Ketika sebuah sistem berubah, strukturnya cenderung tidak mampu beradaptasi. Ini membuat sistem lebih sulit untuk memahami dan membuat perubahan karena program menjadi kurang kohesif.
8. Adanya standar minimal untuk pemeliharaan.
9. Programmer berharap tidak akan terlibat dalam komitmen yang mengikat bahwa mereka ikut serta ke siklus pemeliharaan.

10. Perubahan yang dilakukan pada suatu program dapat memicu terjadinya kesalahan baru, yang memicu permintaan perubahan lebih lanjut.
11. Keterkaitan antara program dan dokumentasi terkadang hilang selama proses pemeliharaan. Dokumentasi itu mungkin menjadi bantuan yang tidak dapat diandalkan untuk memahami program.

Aktifitas Pemeliharaan

IEEE menyediakan kerangka kerja untuk kegiatan proses pemeliharaan berurutan. Ini dapat digunakan secara berulang dan dapat diperpanjang sehingga barang dan proses yang disesuaikan dapat dimasukkan.



Gambar 12.2: Ilustrasi aktifitas pemeliharaan

Aktifitas ini berjalan seiring dengan masing-masing fase sebagai berikut:

1. *Identification & Tracing.*

Tahap pertama berkaitan dengan identifikasi dan manajemen permintaan modifikasi. Pemelihara memvalidasi permintaan

modifikasi, mengklasifikasikannya sesuai dengan kategori pemeliharaan seperti pemeliharaan korektif atau adaptif dan menetapkan prioritasnya. Berdasarkan hal itu mereka dapat memperkirakan sumber daya yang diperlukan untuk memenuhi permintaan tersebut. Permintaan tersebut kemudian ditelusuri melalui melalui suatu mekanisme dan akhirnya dikonfirmasi.

2. *Analysis.*

Modifikasi yang dilakukan selanjutnya perlu dianalisis untuk mengkaji dampaknya terhadap sistem termasuk implikasi keselamatan dan keamanan. Jika kemungkinan dampaknya berat, maka perlu dicarikan solusi alternative untuk mencegah dampak buruk yang berpotensi terjadi. Satu rangkaian modifikasi yang dibutuhkan kemudian diwujudkan menjadi spesifikasi kebutuhan. Biaya modifikasi/pemeliharaan juga dianalisis dan diestimasi.

3. *Design.*

Modul baru, yang perlu diganti atau dimodifikasi, dirancang berdasarkan spesifikasi persyaratan yang ditetapkan pada tahap sebelumnya. Selanjutnya dokumentasi perlu diperbarui.

4. *Implementation.*

Modul baru dikodekan dengan bantuan desain terstruktur yang dibuat dalam langkah desain. Setiap programmer diharapkan untuk melakukan pengujian unit secara paralel.

5. *System Testing.*

Pengujian integrasi dilakukan di antara modul yang baru dibuat. Pengujian integrasi juga dilakukan antara modul-modul baru dan sistem. Akhirnya sistem diuji secara keseluruhan, mengikuti prosedur pengujian regresif.

6. *Acceptance Testing .*

Setelah menguji sistem secara internal, diuji untuk penerimaan dengan bantuan pengguna. Jika pada keadaan ini, pengguna mengeluhkan beberapa masalah yang ditunjukkan atau dicatat untuk ditangani dalam iterasi berikutnya.

7. *Delivery.*

Setelah tes penerimaan, sistem ini digunakan di seluruh organisasi baik oleh paket pembaruan kecil atau instalasi baru dari sistem. Pengujian akhir berlangsung di sisi klien setelah perangkat lunak di-*delivery*.

Fasilitas pelatihan disediakan apabila diperlukan, di samping *hard copy manual* pengguna.

8. *Maintenance Management.*

Manajemen pemeliharaan adalah bagian penting dari pemeliharaan sistem yang melekat pada dokumen standar sebagai lampiran. Ini termasuk, antara lain, kegiatan manajemen yang perlu dilakukan untuk mengatur proses. Hal ini termasuk bagian informatif dari standar.

Tool untuk pemeliharaan perangkat lunak

Pemeliharaan perangkat lunak mencakup memodifikasi sistem perangkat lunak yang ada dan merekam/mencatat semua modifikasi yang dilakukan. Untuk ini, berbagai tool/alat pemeliharaan digunakan. Salah satu alat pemeliharaan yang umum digunakan adalah editor teks. Alat ini membuat salinan dokumentasi atau kode. Fitur utama alat ini adalah menyediakan media untuk memutar kembali (bila diperlukan) dari versi file saat ini ke yang sebelumnya.

Soal

1. Apakah esensi dasar dari kegiatan pemeliharaan perangkat lunak ?
2. Uraikan jenis-jenis perawatan perangkat lunak.
3. Salah satu jenis pemeliharaan adalah yang bersifat korektif. Jelaskan maksudnya dan berikan contoh.
4. Jelaskan target utama dari adaptif maintenance.

5. Jelaskan, mengapa Perfective Maintenance memiliki bobot terbesar dalam aktifitas pemeliharaan perangkat lunak?
6. Jelaskan secara ringkas, aktifitas yang terjadi dalam proses maintenance.

BAB XIII

REKAYASA PERANGKAT LUNAK MASA DEPAN

Tujuan :

- g. Mempelajari tren teknologi rekayasa perangkat lunak di masa mendatang.
- h. Mempelajari basis keilmuan agar dapat eksis di dunia rekayasa perangkat lunak pada masa depan.

Indikator keberhasilan :

- 1. Mahasiswa memahami arah dan pergerakan teknologi rekayasa perangkat lunak.
- 2. Mahasiswa mampu menjelaskan strategi yang baik untuk beradaptasi dengan teknologi rekayasa perangkat lunak di masa datang.

Materi :

Melebihi dari masa-masa sebelumnya, masyarakat memimpikan masa depan yang akan terjadi di mana teknologi cerdas seperti AI (*Artificial Intelligence*), IoT (*Internet of Thing*), *Big Data*, dan kendaraan otonom dapat meningkatkan kualitas hidup sebagian besar penduduk dunia. Semua kemajuan ini sepenuhnya tergantung pada inovasi berbasis perangkat lunak. Oleh karena itu, menguasai kemampuan untuk berinovasi dalam perangkat lunak adalah kunci mutlak untuk membangun posisi terdepan di pasar mana pun yang akan bergantung pada teknologi cerdas.



Gambar 13.1: Gambaran umum teknologi ‘cerdas’

Pondasi yang lemah

Strategi bisnis yang paling ampuh pada akhirnya adalah ilmu untuk memastikan kesuksesan masa depan. Hal ini melibatkan fungsi-fungsi: mengantisipasi masa depan, mengidentifikasi masalah kritis dan merancang rencana tindakan untuk sampai ke sana. Ada satu masalah kritis yang umum terjadi pada semua teknologi cerdas, di mana keseluruhan bisnis rekayasa perangkat lunak yang ada saat ini didasarkan hal yang secara fundamental adalah lemah.

Masalah-masalah ini (di bidang rekayasa perangkat lunak) tidak pernah ditangani dengan lebih baik. Pada sisi lain di mana disiplin ilmu teknik lainnya secara sistematis membangun di atas formalisme yang memberi mereka sebuah dasar yang kuat. Hal tersebut sangat jarang terjadi di bidang rekayasa perangkat lunak. Misalnya: Teknik Aeronautical

menggunakan Aerodinamika, Mekanika dan Termodinamika Penerbangan secara rutin. Semuanya itu didasarkan pada azas *scientific* yang terbukti kebenarannya dan memenuhi prinsip matematika. Sebaliknya, arus utama rekayasa perangkat lunak bergantung pada penggunaan metode informal yang dilakukan **atas dasar coba-coba**.

Wujud Risiko

Membangun teknologi cerdas di atas fondasi yang lemah adalah bisnis yang pada dasarnya berisiko. Saat ini, rekayasa perangkat lunak berjuang untuk membangun sistem yang terbukti kuat, andal, dan tangguh yang melakukan tugas yang tampaknya relatif sederhana dan terbatas. Fakta bahwa hingga saat ini relatif sedikit nyawa telah hilang karena kegagalan perangkat lunak secara langsung adalah karena konteks di mana sebagian besar sistem perangkat lunak saat ini beroperasi dengan kondisi yang dibatasi. Sebagai contoh, di dunia otomotif, perangkat lunak saat ini sebagian besar digunakan untuk aplikasi bantuan/kenyamanan pengemudi di mana efek kegagalan dibatasi pada sistem yang tidak kritis dalam kendaraan pribadi dan pengemudi tetap diharapkan berfungsi sebagai mekanisme keselamatan tertinggi.

Pusaran Kompleksitas

Sistem cerdas di masa depan akan menjadi pesanan yang jauh lebih kompleks daripada yang dibangun hari ini, tidak hanya dalam hal kemampuan mereka tetapi juga ketergantungan mereka pada sistem rumit dan canggih lainnya. Misalnya, dari perspektif perangkat lunak, kendaraan otonom level 5 akan jauh lebih canggih daripada kendaraan apa pun di jalan saat ini. Tetapi *Smart Mobility* juga membayangkan iringan kendaraan otonom bersama-sama dalam konvoi - sistem kompleks dari sistem yang kompleks. Konsekuensi dari kegagalan setiap bagian dari sistem seperti itu berpotensi menyebabkan kekacauan dan mempengaruhi banyak kehidupan. Fondasi yang lemah yang menjadi dasar rekayasa perangkat lunak saat ini

membuat tidak ada cara untuk mengurangi masalah ini dengan tingkat kepastian yang meyakinkan.

Peningkatan Total Biaya Kepemilikan

Fondasi yang lemah dari rekayasa perangkat lunak mengarah pada total biaya kepemilikan untuk produk perangkat lunak. Tetapi karena sebagian besar organisasi rekayasa perangkat lunak menganut paradigma dasar yang lemah, biaya ini telah menjadi norma yang dapat diterima. Jelas kemudian ada peluang besar untuk mengganggu status quo untuk setiap organisasi rekayasa perangkat lunak yang mampu memberikan biaya kepemilikan lebih rendah dengan menggantikan paradigma fondasi yang lemah dengan pendekatan yang lebih ketat.

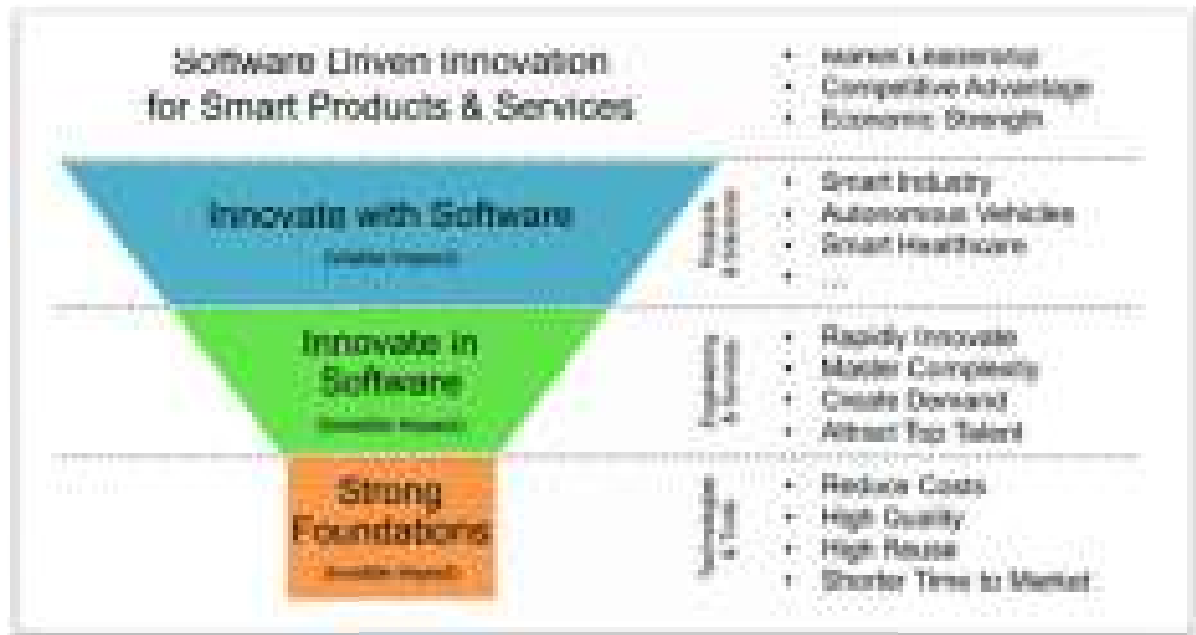


Gambar 13.2: Peningkatan biaya kepemilikan

Menguasai Inovasi Berbasis Perangkat Lunak

Menguasai inovasi berbasis perangkat lunak dimulai dengan membangun dasar yang kuat dalam rekayasa perangkat lunak. Ini berarti mengadopsi dan menggunakan alat dan teknik analitis yang mencegah, mendeteksi, dan menghilangkan cacat di awal siklus pengembangan, seperti perkakas *Verum Dezyne*. Adopsi yang kuat dan fundamental memungkinkan organisasi rekayasa perangkat lunak untuk mengurangi waktu & biaya pengembangan

fitur, mencapai hasil berkualitas tinggi, dan membangun basis kode yang dapat dipelihara dan digunakan kembali. Dengan manfaat dasar yang kuat, tim pengembangan dapat fokus pada inovasi cepat dalam perangkat lunak dan penguasaan yang meningkatkan kompleksitas. Bisnis yang dapat dengan cepat berinovasi dalam perangkat lunak juga dapat lebih cepat berinovasi dengan perangkat lunak, memungkinkan *delivery* produk dan layanan cerdas yang terdepan di pasar.



Gambar 13.3: Inovasi Berbasis Perangkat Lunak untuk Produk & Layanan Cerdas

Bidang Rekayasa Perangkat Lunak tidak akan pernah mati

Ditengah berbagai macam isu-isu yang berkaitan dengan kelemahan yang ada pada bidang rekayasa perangkat lunak, dapat diyakini bahwa bidang ini tidak akan pernah mati dan akan terus berdinamika. Alasan-alasannya adalah sebagai berikut:

1. Pemrograman membutuhkan talenta. Pengkodean membutuhkan waktu berjam-jam untuk kompilasi dan debugging. Untuk melaksanakan aktifitas itu masih banyak dibutuhkan talenta-talenta yang handal.

2. Kemajuan baru tidak terelakkan. Jika Anda berpikir bahwa kumpulan perangkat pengembang saat ini sangat kuat, tunggu sampai anda tahu tentang tool pemrograman yang akan datang. Rekayasa perangkat lunak akan selalu berada di jalur peningkatan dan evolusi yang konstan.
3. Peluang Baru Akan Bermunculan. Umumnya manajemen perusahaan berbasis perangkat lunak hanya dibatasi untuk perusahaan multinasional dan tingkat tinggi lainnya. Tetapi, saat ini Anda dapat menemukan bantuan perangkat lunak di bidang kesehatan, pendidikan, usaha kecil, dan bahkan hukum. Jadi, ruang lingkup selalu melebar, dan ahli perangkat lunak akan diperlukan untuk menguji dan mengembangkan kode baru setiap saat.
4. Peningkatan Posisi. Ketika pasar menjadi jenuh dengan para profesional yang berspesialisasi dalam bidang yang sama, proses seleksi akan disesuaikan untuk membuat proses lebih intensif pada kualifikasi. Jadi, itu akan sampai pada pertanyaan mendasar tentang bagaimana anda akan menonjol dari tengah populasi.
5. Selalu Ada Masalah yang Harus Dipecahkan. Perangkat lunak telah berkembang sejak aplikasi pertamanya dalam memecahkan masalah matematika. Sekarang, sistem bantu yang cerdas di ponsel anda bahkan dapat membuat lelucon untuk menghibur. Peluang selalu tersedia jika anda berani terlihat cukup keras. Di tahun-tahun mendatang, kita dapat mengharapkan perangkat menjadi lebih intuitif berkat para ahli perangkat lunak.
6. *Learning Machine and Deep Learning*. Google Pixel 2 membuat kagum semua orang ketika diluncurkan karena mampu mengambil foto dengan berbagai gaya dengan kamera tunggal sementara ponsel lain membutuhkan pengaturan lensa ganda. Jika anda berpikir Pixel 2 menggunakan sihir untuk menyelesaikan tugas seperti itu, sentral sebenarnya adalah algoritma *learning machine* yang mendasarinya di mana ia mengidentifikasi perbedaan antara subjek yang difokuskan dan latar belakang. Google sekarang mempekerjakan lebih dari

- 30.000 orang pengembang untuk bekerja pada platform AI dan *learning machine* mereka untuk meningkatkannya lebih lanjut.
7. Dominasi Komputer. Faktor komputer yang saat ini bervariasi dalam ukuran dan beratnya hampir tanpa batasan yang jelas. Bahkan smartphone anda adalah komputer yang mampu melakukan tugas luar biasa. Perangkat melakukan lebih dari apa yang biasa mereka lakukan.
 8. Komputer Akan Selalu Memiliki Pembatas. Meskipun seberapa canggih komputer itu, ia jauh lebih lambat daripada otak manusia. Masih ada banyak masalah di dunia yang tidak bisa diselesaikan oleh komputer. Programmer akan selalu berada di garis depan dalam mengatasi masalah seperti itu dan menemukan solusi melalui komputasi.
 9. Bukan Sekedar Tentang Pengkodean. Di saat datang untuk menangani masalah pekerjaan nyata dengan perangkat lunak, pengkodean menempati posisi kedua dalam proses “menemukan solusi”. Oleh karena itu, ahli perangkat lunak akan diperlukan untuk menemukan proses yang tepat bahkan jika ada alat yang dapat mengotomatisasi penulisan kode.
 10. Perangkat Keras Akan Berubah. Saat perangkat keras di sekitar kita berevolusi dan menjadi lebih cepat dan lebih efisien, mereka akan membutuhkan seperangkat kode yang sesuai untuk membuka potensi penuh mereka. Tidak ada penggunaan perangkat keras komputer jika Anda tidak dapat mengaksesnya dengan perangkat lunak yang tepat.
 11. Ini adalah *engineering*. Rekayasa perangkat lunak adalah cabang teknik. Apakah anda pikir seseorang yang mengikuti beberapa kelas online akan dapat mengalahkan ahli perangkat lunak yang terverifikasi? Kompleksitas rekayasa perangkat lunak membuatnya unik, dan beberapa orang mungkin mengatakan itu pekerjaan mudah, tetapi semakin anda mengenal elemen inti, semakin baik

anda memahami apa yang membedakan ahli perangkat lunak dengan insinyur dari bidang yang lain.

12. Tidak ada ketergantungan. Raksasa perangkat lunak saat ini dimulai dengan sebuah ide untuk membuat sesuatu yang unik dan berbeda. Mempelajari rekayasa perangkat lunak tidak berarti Anda harus bekerja dengan para pemimpin di industri Perangkat Lunak. Memulai sesuatu dari Anda sendiri juga akan membawa lebih banyak manfaat karena Anda akan memiliki semua kebebasan untuk membawa sesuatu yang baru ke meja kerja.
13. *Scalable*. Jika Anda berpikir bahwa rekayasa perangkat lunak adalah tentang duduk di ruangan dan melakukan pengkodean sepanjang hari, maka Anda bisa salah. Ada kebutuhan untuk ahli perangkat lunak di hampir setiap bidang sekarang. Baik itu manufaktur atau perawatan kesehatan.
14. *Open Arms of Automobiles*. Mobil-mobil *self-driving* adalah hal yang penting sekarang. Tapi, mobil otonom ini tidak mungkin terjadi tanpa upaya gabungan dari insinyur mesin dan perangkat lunak. Membuat mobil lebih pintar hanyalah salah satu dari banyak pintu yang dibuka untuk ahli perangkat lunak untuk ekspansi bidang terapannya.
15. *IoT* adalah teknologi baru. Rumah pintar dan peralatan pintar merupakan teknologi terbaru. *Internet of Things (IoT)* menggemparkan dunia ketika orang mulai menyadari kemampuan aplikasinya. Dengan sensasi IoT, permintaan untuk ahli perangkat lunak juga meningkat dan tren ini pasti akan tetap seperti ini selama manusia bergantung pada teknologi.

Tantangan Rekayasa Perangkat Lunak

Rekayasa perangkat lunak menggunakan pendekatan yang jelas dan sistematis untuk mengembangkan berbagai perangkat lunak. Pendekatan ini dianggap sebagai cara paling efektif untuk menghasilkan perangkat lunak berkualitas tinggi. Namun, terlepas dari pendekatan sistematis dalam

pengembangan perangkat lunak ini, masih ada beberapa tantangan serius yang dihadapi oleh rekayasa perangkat lunak. Beberapa dari tantangan ini tercantum di bawah ini:

1. Metode yang digunakan untuk mengembangkan proyek skala kecil atau menengah tidak cocok ketika datang ke pengembangan sistem skala besar atau kompleks.
2. Perubahan dalam pengembangan perangkat lunak tidak dapat dihindari. Di dunia saat ini, perubahan terjadi dengan cepat dan mengakomodasi perubahan ini untuk mengembangkan perangkat lunak yang lengkap adalah salah satu tantangan utama yang dihadapi oleh para ahli perangkat lunak.
3. Kemajuan dalam teknologi komputer dan perangkat lunak telah diperlukan untuk perubahan dalam sifat sistem perangkat lunak. Sistem perangkat lunak yang tidak dapat mengakomodasi perubahan tidak banyak digunakan. Dengan demikian, salah satu tantangan rekayasa perangkat lunak adalah untuk menghasilkan perangkat lunak berkualitas tinggi beradaptasi dengan perubahan kebutuhan dalam jadwal yang dapat diterima. Untuk memenuhi tantangan ini, pendekatan berorientasi objek lebih disukai, tetapi mengakomodasi perubahan pada perangkat lunak dan pemeliharannya dalam biaya yang dapat diterima masih merupakan tantangan.
4. Komunikasi informal menghabiskan sebagian besar waktu yang dihabiskan untuk proyek perangkat lunak. Pemborosan waktu seperti itu menunda penyelesaian proyek dalam waktu yang ditentukan.
5. Pengguna umumnya hanya memiliki gagasan yang samar tentang ruang lingkup dan persyaratan sistem perangkat lunak. Ini biasanya menghasilkan pengembangan perangkat lunak, yang tidak memenuhi persyaratan pengguna.
6. Perubahan biasanya dimasukkan dalam dokumen tanpa mengikuti prosedur standar apa pun. Dengan demikian, verifikasi semua perubahan seperti itu seringkali menjadi sulit.

7. Pengembangan perangkat lunak berkualitas tinggi dan andal membutuhkan perangkat lunak untuk diuji secara menyeluruh. Meskipun pengujian menyeluruh terhadap perangkat lunak menghabiskan sebagian besar sumber daya, meremehkannya karena alasan apa pun menurunkan kualitas perangkat lunak.

Selain tantangan utama yang disebutkan di atas, tanggung jawab analisis sistem, perancang, dan pemrogram biasanya tidak didefinisikan dengan baik. Juga, jika persyaratan pengguna tidak didefinisikan secara tepat, pengembang perangkat lunak dapat salah menafsirkan maksudnya. Semua tantangan ini perlu diatasi untuk memastikan bahwa perangkat lunak dikembangkan dalam waktu yang ditentukan dan perkiraan biaya dan juga memenuhi persyaratan yang ditentukan oleh pengguna.

Soal :

1. Jelaskan mengapa fondasi dalam pengembangan teknologi perangkat lunak masih lemah?
2. Uraikan dan jelaskan arah dan perkembangan teknologi rekayasa perangkat lunak di masa depan.
3. Jelaskan, apa yang anda ketahui tentang IoT (Internet of Thing).
4. Menurut anda, bagaimana cara agar dapat menjadi seorang sistem enginer yang baik di masa depan?

Daftar Pustaka

- [1]. A.S, Rosa, M. Shalahuddin. 2015. "Rekayasa Perangkat Lunak". Informatika. Bandung.
- [2]. Ali, Edwar. 2016. "Metode User Centered Design (UCD) dalam Membangun Aplikasi Layanan Manajerial di Perguruan Tinggi. Vol. 2. No. 2. SATIN-Sains dan Teknologi Informasi. pp. 1-6.
- [3]. Bell, Douglas. 2005. Software Engineering for Student, A Programming Approach. 4th edition. Addison Wesley.
- [4]. Heryanto, Imam, Totok Triwibowo. 2009. "Manajemen Proyek Berbasis Teknologi Informasi. Mengelola Proyek Secara Sistematis Menggunakan Microsost Project". Penerbit Informatika. Bandung.
- [5]. SWEBOK: Guide to the Software Engineering Body of Knowledge- A Straw Man Version. 1998. IEEE Computer Society.
- [6]. IEEE Standard for Developing Software Life Cycle Processes. 1995. IEEE Computer Society. New York. NY.
- [7]. Pressman, Roger. S.. 2010, "*Software Engineering : A Practioner's Approach*." 7th edition. McGrawHill.
- [8]. Siahaan, Daniel. 2012. "Analisa Kebutuhan Dalam Rekayasa Perangkat Lunak". Andi Publisher. Jogjakarta.
- [9]. Simarmata, Janner. 2010. "Rekayasa Perangkat Lunak". Andi Publisher. Jogjakarta.
- [10]. Sommerville, Ian. 2009. "*Software Engineering*". 9th edition. Addison Wesley. Boston. USA.
- [11]. www.tutorialspoint.com

Tentang Penulis



Edwar Ali

Lahir di Pariaman pada tanggal 22 September 1973. Menamatkan pendidikan sarjana dan magister pada Universitas Putera Indonesia YPTK Padang. Penulis saat ini adalah dosen tetap pada STMIK Amik Riau Pekanbaru. Sejak tahun 2006, penulis sering terlibat dalam berbagai program hibah pengembangan institusi dan penelitian yang diselenggarakan oleh Kemenristekdikti. Selain berkarir sebagai dosen, juga aktif terlibat dalam berbagai kegiatan sebagai konsultan IT dan membantu berbagai program pemerintahan daerah Riau khususnya di bidang IT. Untuk koresponden penulis dapat dihubungi di mail: edwarali@stmik-amik-riau.ac.id, FB: Edwar ali, HP/WA: 08127633349.