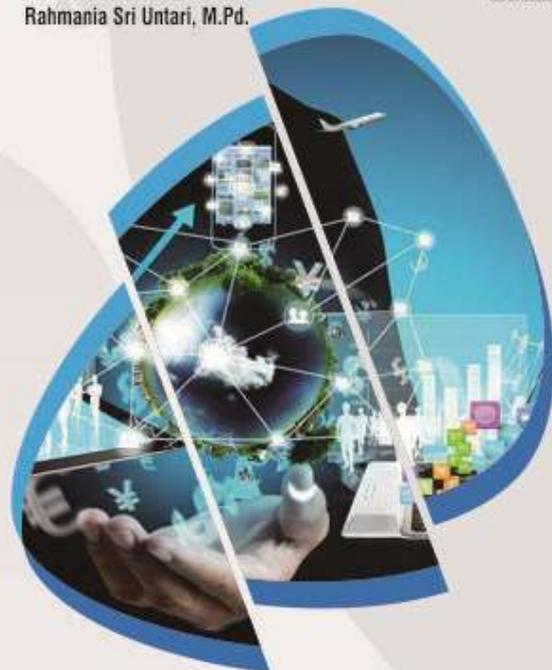


Fitria Nur Hasanah, M.Pd.
Rahmania Sri Untari, M.Pd.



ISBN 978-623-6833-89-6 (PDF)



UMSIDA Press
Universitas Muhammadiyah Sidoarjo
Jl. Mojopahit 666 B Sidoarjo
Sidoarjo, Jawa Timur

REKAYASA PERANGKAT LUNAK



BUKU AJAR REKAYASA PERANGKAT LUNAK

Oleh

**Fitria Nur Hasanah, M.Pd
Rahmania Sri Untari, M.Pd**



**Diterbitkan oleh
UMSIDA PRESS**

**UNIVERSITAS MUHAMMADIYAH SIDOARJO
2020**

BUKU AJAR
REKAYASA PERANGKAT LUNAK

Penulis:

Fitria Nur Hasanah, M.Pd
Rahmania Sri Untari, M.Pd

ISBN :

978-623-6833-89-6

Editor:

Mohammad Suryawinata, S.Pd., M.Kom

Design Sampul dan Tata Letak:

Mochammad Nashrullah, S.Pd.
Amy Yoga Prajati, S.Kom

Penerbit:

UMSIDA Press
Anggota IKAPI No. 218/Anggota Luar Biasa/JTI/2019
Anggota APPTI No. 002 018 1 09 2017

Redaksi

Universitas Muhammadiyah Sidoarjo
Jl. Mojopahit No 666B
Sidoarjo, Jawa Timur

Cetakan Pertama, September 2020

©Hak Cipta dilindungi undang undang

Dilarang memperbanyak karya tulis ini dengan sengaja, tanpa ijin tertulis dari penerbit.

KATA PENGANTAR

Alhamdulillah Puji Syukur Penulis sampaikan kehadiran Allah SWT, yang telah melimpahkan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan Buku Ajar Rekayasa Perangkat Lunak dengan baik. Buku ajar ini ditujukan kepada mahasiswa yang mengampuh mata kuliah Rekayasa Perangkat Lunak. Dengan dibuatnya Buku Ajar ini penulis berharap agar dapat bermanfaat dan membantu dalam memahami materi Rekayasa Perangkat Lunak yang semakin berkembang.

Ucapkan terima kasih penulis ucapkan kepada :

1. Dr. Hidayatulloh, M.Si., Rektor UMSIDA yang memberikan kesempatan luas kepada tim penulis untuk berkarya dan menyumbangkan pikiran sehingga buku ajar ini terselesaikan.
2. Dr. Akhtim Wahyuni, M.Ag Dekan Fakultas Psikologi dan Ilmu Pendidikan yang memberikan arahan dan motivasi kepada penulis dalam menyelesaikan buku ajar ini.
3. Rekan-rekan dosen Pendidikan Teknologi Informasi UMSIDA yang telah berbagi pengalaman dalam penulisan buku ajar, serta membantu dalam penyelesaian buku ajar Rekayasa Perangkat Lunak.

Penulis menyadari sekali bahwa Buku Ajar ini masih jauh dari kesempurnaan, maka dari itu penulis mengharapkan kritik dan saran pembaca demi kesempurnaan Buku Ajar ini kedepannya. Akhir kata penulis mengucapkan terima kasih, mudah-mudahan bermanfaat bagi para pembaca.

Sidoarjo, September 2020
Penulis

DAFTAR ISI

Kata Pengantar	i
Daftar Isi	ii
Bab I Pendahuluan	1
1. Deskripsi	1
2. Kemampuan Akhir	2
Bab II Konsep Rekayasa Perangkat Lunak.....	4
1. Definisi Rekayasa Perangkat Lunak	5
2. Tujuan Rekayasa Perangkat Lunak	7
3. Ruang Lingkup Rekayasa Perangkat Lunak.....	7
4. Jenis Perangkat Lunak	9
5. Tanggung Jawab Profesional Dan Etika	11
Bab III Perencanaan Perangkat Lunak	14
1. Tujuan Perencanaan Proyek.....	14
2. Teknik Pengumpulan Data.....	14
3. Ruang Lingkup Perangkat Lunak.....	17
4. Estimasi Proyek Perangkat Lunak.....	18
Bab IV Software Development Life Cycle (SDLC)	20
1. Definisi SDLC	20
2. Model SDLC.....	21
a. Model Waterfall	21
b. Model Prototipe	23
c. Model Rapid Application Development (RAD)....	26
d. Model Iteratif	29
e. Model Spiral	31

Bab V Basis Data.....	36
1. Definisi Basis Data.....	36
2. Entity Relationship Diagram (ERD).....	37
3. Study Kasus ERD.....	42
Bab VI Data Flow Diagram (DFD).....	49
1. Definisi DFD	49
2. Komponen DFD.....	50
3. Tingkatan atau Level DFD	54
4. Studi Kasus Pembuatan DFD	57
Bab VII Pemodelan UML	63
1. Kompleksitas Pengembangan Perangkat Lunak.....	63
2. Pemodelan Perangkat Lunak.....	63
3. <i>Unified Modelling Language</i> (UML)	64
4. Use Case Diagram.....	72
5. Activity Diagram	80
Bab VIII Manajemen Proyek Perangkat Lunak	91
1. Perencanaan Proyek RPL	91
2. Pengujian Perangkat Lunak.....	97
Bab IX Pemeliharaan Perangkat Lunak	104
1. Teknik Pemeliharaan Perangkat Lunak	104
2. Siklus Hidup Pemeliharaan Sistem	107
Daftar Pustaka	110
Biodata Penulis	

**BATANG TUBUH DAN
SUB-CAPAIAN PEMBELAJARAN MATA KULIAH**

BAB	Sub-Capaian Pembelajaran Mata Kuliah
BAB I PENDAHULUAN	Mahasiswa mampu memahami deskripsi mata kuliah Rekayasa Perangkat Lunak
BAB II KONSEP DASAR RPL	<ol style="list-style-type: none"> 1. Mahasiswa mampu menjelaskan definisi rekayasa perangkat lunak 2. Mahasiswa mampu menjelaskan tujuan rekayasa perangkat lunak 3. Mahasiswa mampu menjelaskan ruang lingkup rekayasa perangkat lunak 4. Mahasiswa mampu menjelaskan Jenis perangkat lunak 5. Mahasiswa mampu memahami tanggung Jawab profesional dan etika
BAB III PERENCANAAN PERANGKAT LUNAK	<ol style="list-style-type: none"> 1. Mahasiswa mampu menyusun tujuan perencanaan proyek 2. Mahasiswa mampu menerapkan teknik pengumpulan data 3. Mahasiswa mampu menguraikan ruang lingkup perangkat lunak 4. Mahasiswa mampu menyusun estimasi proyek perangkat lunak
BAB IV SOFTWARE DEVELOPMENT LIVE CYCLE (SDLC)	<ol style="list-style-type: none"> 1. Mahasiswa mampu menjelaskan definisi SDLC 2. Mahasiswa mampu menerapkan model SDLC

BAB V BASIS DATA	<ol style="list-style-type: none"> 1. Mahasiswa mampu menjelaskan definisi basis data 2. Mahasiswa mampu menerapkan ERD 3. Mahasiswa mampu menerapkan studi kasus ERD
BAB VI DATA FLOW DIAGRAM (DFD)	<ol style="list-style-type: none"> 1. Mahasiswa mampu menjelaskan definisi DFD 2. Mahasiswa mampu menjelaskan komponen DFD 3. Mahasiswa mampu menerapkan tingkatan level DFD 4. Mahasiswa mampu menerapkan studi kasus DFD
BAB VII PEMODELAN DAN UML	<ol style="list-style-type: none"> 1. Mahasiswa mampu menjelaskan kompleksitas pengembangan perangkat lunak 2. Mahasiswa mampu menjelaskan pemodelan perangkat lunak 3. Mahasiswa mampu menerapkan UML 4. Mahasiswa mampu menerapkan Use Case Diagram 5. Mahasiswa mampu menerapkan Activity Diagram
BAB VIII MANAJEMEN PROYEK PERANGKAT LUNAK	<ol style="list-style-type: none"> 1. Mahasiswa mampu menerapkan perencanaan proyek 2. Mahasiswa mampu menerapkan pengujian perangkat lunak
BAB IX PEMELIHARAAN PERANGKAT LUNAK	<ol style="list-style-type: none"> 1. Mahasiswa mampu menjelaskan teknik pemeliharaan perangkat lunak 2. Mahasiswa mampu menerapkan Siklus Hidup Pemeliharaan Sistem (SMLC)

BAB I

PENDAHULUAN

1. Deskripsi

Perangkat lunak (*software*) adalah program computer yang terasosiasi dengan dokumentasi perangkat lunak seperti dokumentasi terkait analisis kebutuhan, model desain, dan user manual Shalahuddin (2016). *IEEE Computer Society* mendefinisikan Rekayasa Perangkat Lunak (RPL) sebagai penerapan suatu pendekatan yang sistematis, disiplin dan terkuantifikasi atas pengembangan, penggunaan dan pemeliharaan perangkat lunak, serta studi atas pendekatan-pendekatan ini, yaitu penerapan pendekatan engineering atas perangkat lunak. RPL sendiri adalah suatu disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal yaitu analisa kebutuhan pengguna, menentukan spesifikasi dari kebutuhan pengguna, disain, pengkodean, pengujian sampai pemeliharaan sistem setelah digunakan. Dari pengertian ini jelaslah bahwa RPL tidak hanya berhubungan dengan cara pembuatan program komputer. Pernyataan "semua aspek produksi" pada pengertian di atas, mempunyai arti semua hal yang berhubungan dengan proses produksi seperti manajemen proyek, penentuan personil, anggaran biaya, metode, jadwal, kualitas sampai dengan pelatihan pengguna merupakan bagian dari RPL.

RPL merupakan pembangunan dengan menggunakan prinsip atau konsep rekayasa dengan tujuan menghasilkan perangkat lunak yang bernilai ekonomi yang dipercaya dan bekerja secara efisien menggunakan mesin. Perangkat lunak banyak dibuat dan pada akhirnya sering tidak digunakan karena tidak memenuhi kebutuhan pelanggan atau bahkan karena masalah non teknis seperti keengganan pemakai perangkat (*user*) untuk mengubah cara kerja dari manual ke

otomatis, atau ketidakmampuan *user* menggunakan komputer. Oleh karena itu, rekayasa perangkat lunak dibutuhkan agar perangkat lunak yang dibuat tidak menjadi perangkat lunak yang tidak terpakai.

RPL lebih fokus pada praktik pengembangan perangkat lunak dan mengirimkan perangkat lunak yang bermanfaat kepada pelanggan (*customer*). Adapun ilmu computer lebih fokus pada teori dan konsep dasar perangkat computer. Rekayasa perangkat lunak lebih fokus pada bagaimana membuat perangkat lunak yang memenuhi kriteria berikut:

- a. Dapat terus dipelihara setelah perangkat lunak selesai dibuat seiring berkembangnya teknologi dan lingkungan (*maintainability*)
 - b. Dapat diandalkan dengan proses bisnis yang dijalankan perubahan yang terjadi (*dependability robust*)
 - c. Efisien dari segi sumber daya dan penggunaan
 - d. Kemampuan untuk dipakai sesuai dengan kebutuhan (*usability*)
- Dari kriteria di atas maka perangkat lunak yang baik adalah perangkat lunak yang dapat memenuhi kebutuhan pelanggan (*customer*) atau *user* (pemakai perangkat lunak) atau berorientasi pada pelanggan atau pemakai perangkat lunak, bukan berorientasi pada pembuat atau pengembang perangkat lunak.

2. Kemampuan Akhir

Mahasiswa diharapkan akan dapat memiliki kompetensi sikap, pengetahuan dan ketrampilan yang berkaitan dengan materi yang ditunjukkan pada Gambar 1.1 setelah mempelajari materi pada buku ajar Rekayasa Perangkat Lunak.



Gambar 1.1 Peta Konsep Materi Rekayasa Perangkat Lunak

Bab II

Konsep Rekayasa Perangkat Lunak (RPL)

Setelah mempelajari bab konsep rekayasa perangkat lunak, maka diharapkan :

6. Mahasiswa mampu menjelaskan definisi rekayasa perangkat lunak
7. Mahasiswa mampu menjelaskan tujuan rekayasa perangkat lunak
8. Mahasiswa mampu menjelaskan ruang lingkup rekayasa perangkat lunak
9. Mahasiswa mampu menjelaskan Jenis perangkat lunak
10. Mahasiswa mampu memahami tanggung Jawab profesional dan etika

Bayangkan anda mempunyai sebidang tanah yang akan dibangun rumah. Lalu pertanyaan apa yang muncul pertama kali?

“Bagaimana proses pembangunan rumah anda ?”

- a. Jika anda memulai membangun dengan cepat ? (hanya dibantu oleh anak anda yang berumur 14 tahun)
- b. Jika anda pergi ke sembarang pengembang
- c. Jika Anda mempekerjakan seorang arsitek untuk mendesain dari awal

“Apakah yang akan dihasilkan ?”

Lalu, Bagaimana dengan membangun perangkat lunak ?

Software development biasanya akan melakukan hal yang sama etika mendapatkan persoalan sederhana yang membutuhkan solusi komputasi: *Berfikir sejenak, menghadap komputer dan kemudian mulai mengetikkan baris demi baris code. Tidak ada kertas-kertas yang memuat perancangan arsitektur dan algoritma secara rinci, karena semua rancangan itu ada di dalam kepala.*

Oleh karena itu kita memerlukan **Rekayasa Perangkat Lunak**

1. Definisi Rekayasa Perangkat Lunak

IEEE-Standar Glossary of Software Engineering Terminology, 1990: "Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system". Istilah Rekayasa Perangkat Lunak (RPL) secara umum disepakati sebagai terjemahan dari istilah Software engineering. Istilah *Software Engineering* mulai dipopulerkan pada tahun 1968 pada *software engineering Conference* yang diselenggarakan oleh NATO. Sebagian orang mengartikan RPL hanya sebatas pada bagaimana membuat program komputer. Padahal ada perbedaan yang mendasar antara perangkat lunak (*software*) dan program komputer. Perangkat lunak merupakan kumpulan dari program, prosedur, dan dokumen data lain yang saling berhubungan yang merepresentasikan masalah di dunia nyata yang dikonfigurasi dalam sebuah bentuk aplikasi yang harus dikerjakan komputer. Perangkat lunak adalah seluruh perintah yang digunakan untuk memproses informasi. Perangkat lunak dapat berupa program atau prosedur. Program adalah kumpulan perintah yang dimengerti oleh komputer sedangkan prosedur adalah perintah yang dibutuhkan oleh pengguna dalam memproses informasi (O'Brien, 1999).

Berdasarkan kajian di atas dapat disimpulkan, RPL merupakan disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal spesifikasi sistem sampai pemeliharaan sistem setelah digunakan. Perangkat Lunak yang dibuat harus mampu: tepat waktu, tepat anggaran, meningkatkan kinerja, mengoperasikan prosedur sistem dengan benar. Rekayasa perangkat lunak lebih fokus pada praktik pengembangan perangkat lunak dan mengirimkan perangkat lunak yang bermanfaat untuk pelanggan. Sehingga perangkat lunak yang baik adalah perangkat lunak yang

dapat memenuhi kebutuhan pelanggan atau user dengan kata lain berorientasi pada pelanggan atau pemakai perangkat lunak, **bukan** berorientasi pada pembuat atau pengembang.

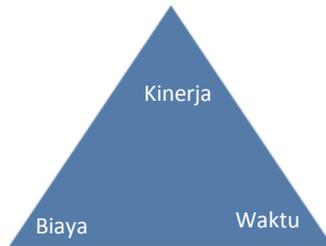
Pekerjaan yang terkait dengan rekayasa perangkat dapat dikategorikan menjadi tiga buah kategori umum tanpa melihat area dari aplikasi, ukuran proyek perangkat lunak, atau kompleksitas perangkat lunak yang akan dibuat. Setiap fase dialamatkan pada satu atau lebih pertanyaan yang diajukan sebelumnya.

Fase pendefinisian fokus pada “*what*” yang artinya harus mencari tahu atau mengidentifikasi informasi apa yang harus diproses, seperti apa fungsi dan performansi yang diinginkan, seperti apa perilaku yang diinginkan, apa kriteria validasi yang dibutuhkan untuk mendefinisikan sistem. Fase pengembangan yang fokus dengan “*how*” yang artinya selama tahap Pengembangan perangkat lunak seorang perekayasa perangkat lunak (*software engineer*) berusaha mendefinisikan bagaimana data distrukturkan dan bagaimana fungsi-fungsi yang dibutuhkan diimplementasikan di dalam arsitektur perangkat lunak, bagaimana detail procedural diimplementasikan, bagaimana karakter antarmuka tampilan, bagaimana desain ditranslasikan ke bahasa pemrograman, dan bagaimana pengujian akan dijalankan.

Fase pendukung (*support phase*) fokus pada perubahan terasosiasi pada perbaikan kesalahan (*error*), adaptasi yang dibutuhkan pada lingkungan perangkat lunak yang terlibat, dan perbaikan yang terjadi akibat perubahan kebutuhan pelanggan (*customer*). Fase pendukung terdiri dari empat tipe perubahan antara lain, koreksi (*correction*), adaptasi (*adaptation*), perbaikan (*enhancement*), dan pencegahan (*prevention*).

2. Tujuan Rekayasa Perangkat Lunak

Bidang rekayasa akan selalu berusaha menghasilkan output yang kinerjanya tinggi, biaya rendah dan waktu penyelesaian yang tepat. Tujuan RPL dapat ditunjukkan pada Gambar 1.



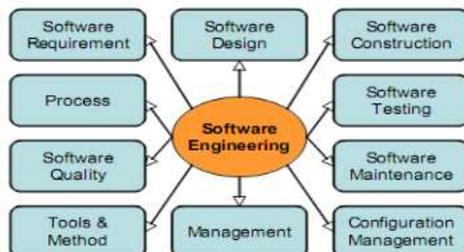
Gambar 1. Tujuan RPL

Secara lebih khusus kita dapat menyatakan tujuan RPL adalah:

- Memperoleh biaya produksi perangkat lunak yang rendah.
- Menghasilkan perangkat lunak yang kinerjanya tinggi, andal dan tepat waktu.
- Menghasilkan perangkat lunak yang dapat bekerja pada berbagai jenis platform.
- Menghasilkan perangkat lunak yang biaya perawatannya rendah.

3. Ruang Lingkup Rekayasa Perangkat Lunak

Berdasarkan definisi dari para ahli yang telah dibahas sebelumnya, maka ruang lingkup RPL dapat ditunjukkan di Gambar 2.



Gambar 2. Ruang Lingkup RPL

- a. Software requirements berhubungan dengan spesifikasi kebutuhan dan persyaratan perangkat lunak.
- b. Software design mencakup proses penentuan arsitektur, komponen, antarmuka, dan karakteristik lain dari perangkat lunak.
- c. Software construction berhubungan dengan detail pengembangan perangkat lunak, termasuk algoritma, pengkodean, pengujian, dan pencarian kesalahan.
- d. Software testing meliputi pengujian pada keseluruhan perilaku perangkat lunak.
- e. Software maintenance mencakup upaya-upaya perawatan ketika perangkat lunak telah dioperasikan.
- f. Software configuration management berhubungan dengan usaha perubahan konfigurasi perangkat lunak untuk memenuhi kebutuhan tertentu.
- g. Software engineering management berkaitan dengan pengelolaan dan pengukuran RPL, termasuk perencanaan proyek perangkat lunak.
- h. Software engineering tools and methods mencakup kajian teoritis tentang alat bantu dan metode RPL.
- i. Software engineering process berhubungan dengan definisi, implementasi, pengukuran, pengelolaan, perubahan dan perbaikan proses RPL.
- j. Software quality menitikberatkan pada kualitas dan daur hidup perangkat lunak.

4. Jenis Perangkat Lunak

Terdapat beberapa jenis perangkat lunak sesuai dengan ruang lingkungannya, diantaranya:

- a. *Software Requirements* berhubungan dengan spesifikasi kebutuhan dan persyaratan perangkat lunak
- b. *Software Desain* mencakup proses penampilan arsitektur, komponen, antar muka, dan karakteristik lain dari perangkat lunak
- c. *Software Construction* berhubungan dengan detail pengembangan perangkat lunak, termasuk algoritma, pengkodean, pengujian dan pencarian kesalahan
- d. *Software Testing* meliputi pengujian pada keseluruhan perilaku perangkat lunak
- e. *Software Maintenance* mencakup upaya-upaya perawatan ketika perangkat lunak telah dioperasikan
- f. *Software Configuration Management* berhubungan dengan usaha perubahan konfigurasi perangkat lunak untuk memenuhi kebutuhan tertentu
- g. *Software Engineering management* berkaitan dengan pengelolaan dan pengukuran RPL, termasuk perencanaan proyek perangkat lunak
- h. *Software Engineering Tools And Methods* mencakup kajian teoritis tentang alat bantu dan metode RPL
- i. *Software Engineering Process* berhubungan dengan definisi, implementasi pengukuran, pengelolaan, perubahan dan perbaikan proses RPL
- j. *Software Quality* menitik beratkan pada kualitas dan daur hidup perangkat lunak

Jenis software berdasar lisensi, terdapat 2 jenis yaitu ***open source dan proprietary***. Pertama yaitu ***open source software***, *Software* yang *source codenya* terbuka dan didistribusikan dalam suatu format lisensi yang memungkinkan pihak lain secara bebas memperbanyak dan memodifikasi *source code* (informasi)

didalamnya. Pada open source software ini, kepemilikan hak cipta tetap ada, akan tetapi lisensi memungkinkan dapat digunakan oleh orang lain serta dapat memodifikasi softwrenya. Jenis-jensi lisensi *open source software* diantaranya BSD license, Apache License, GNU General Public License (GPL), Mozilla Public License, dan MIT License.

Kedua yaitu ***proprietary software***, merupakan *Software* yang *source* codenya tertutup dan didistribusikan dengan suatu format lisensi yang membatasi pihak lain untuk menggunakan, memperbanyak dan memodifikasi. Lisensi proprietary software memungkinkan orang lain menggunakan software yang kita buat dengan diikuti penyerahan royalti (uang) ke pemilik hak ciptanya.

Perangkat lunak berdasar fungsionalnya, yaitu : *interfacing*, *operating sistem*, dan program aplikasi. ***Interfacing*** yaitu perangkat lunak yang menghubungkan suatu perangkat keras tertentu, seperti hardware driver, interfaces dengan perangkat keras lain. Contoh : Driver untuk Kamera, *Handphone* atau perangkat keras lainnya, Program interface seperti Sensor Suhu dengan LM 555, PPI 8255, Komunikasi Serial RS 232. ***Operating system***, Perangkat lunak yang menjalankan sistem komputer dan merupakan *interface* dari sistem komputer dan program aplikasi yang berjalan diatasnya. Beberapa OS yang dikenal secara luas: Microsoft Windows, Linux dan variansnya, seperti Redhat, SuSE, Mandrake, Debian, Unix, FreeBSD, *Macintosh (Apple)*. **Program Aplikasi**, yaitu perangkat lunak yang digunakan program ini digunakan untuk keperluan tertentu, yang tujuannya membantu pekerjaan manusia menjadi lebih mudah. Program ini yang banyak dibahas dalam pembuatan perangkat lunak. Program Aplikasi ini tergantung pada kebutuhan dari program itu sendiri, seperti: *Program Office*, *Program Graphics Design*, *Program Multimedia*, dan lain-lain.

Karakteristik perangkat lunak, diantaranya : Mempunyai daya guna yang tinggi (usability), Mempunyai kinerja sesuai fungsi yang

dibutuhkan pemakai, Mampu diandalkan (be reliable), Mudah dirawat/diperbaiki (*maintenability*), Lebih efisien, Mempunyai antarmuka yg menarik (*eye cathcing user interface*), Mempunyai siklus hidup yang cukup lama (*long life time*).

5. Tanggung Jawab Profesional Dan Etika

RPL melibatkan tanggung jawab yang lebih besar dari sekedar penerapan keahlian teknis. Rekayasawan perangkat lunak harus berlaku secara jujur dan etis jika ingin dihargai sebagai profesional. Perilaku etis lebih dari sekedar menjunjung tinggi hukum. Tanggung jawab profesional yang pertama adalah **kerahasiaan**, Rekayasawan harus menghargai kerahasiaan pegawai atau kliennya. Kedua, **Kompeten**, Rekayasawan tidak boleh memberi gambaran yang salah tentang tingkat kompetensinya, mereka tidak boleh secara sadar menerima pekerjaan yang diluar kompetensinya.

Perangkat lunak yang baik harus memberikan fungsi yang diperlukan dan kinerja bagi pengguna dan harus dipertahankan, diandalkan dan dapat diterima. **Maintainability** (mudah dipelihara/dirawat) *Software* harus berevolusi untuk memenuhi perubahan kebutuhan, **Dependability** (dapat diandalkan) *Software* harus dapat dipercaya, **Efficiency** (Daya Guna) Efisien dalam penggunaan sumber daya sistem (memori, hardware, listrik, dll), **Acceptability** (Dapat diterima) Perangkat Lunak harus diterima oleh pengguna seperti yang pernah dirancang. Ini berarti harus bisa dimengerti, digunakan dan kompatibel dengan sistem lainnya.

Tantangan utama yang dihadapi RPL, yaitu: (1) **Heterogeneity** (keberagaman), mengembangkan teknik untuk membangun perangkat lunak yang dapat mengatasi perbedaan platform dan lingkungan eksekusi; (2) **Delivery** (pengiriman), mengembangkan teknik yang mengakibatkan pengiriman perangkat lunak lebih cepat;

(3) *Trust* (kepercayaan), mengembangkan teknik yang menunjukkan bahwa perangkat lunak dapat dipercaya oleh para penggunanya.

Kesimpulan

RPL merupakan disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal spesifikasi sistem sampai pemeliharaan sistem setelah digunakan. Perangkat Lunak yang dibuat harus mampu: tepat waktu, tepat anggaran, meningkatkan kinerja, mengoperasikan prosedur sistem dengan benar. Rekayasawan perangkat lunak harus berlaku secara jujur dan etis jika ingin dihargai sebagai profesional.

Evaluasi

Setelah menyimak materi konsep rekayasa perangkat lunak, silahkan jawab pertanyaan berikut:

1. Apakah proses produksi perangkat lunak identik atau serupa dengan proses produksi pada pabrik/manufaktur pembuatan mobil? Jelaskan alasannya!
2. Bidang rekayasa perangkat lunak apakah sebagai bagian dari seni atau bagian dari teknik? Jelaskan alasannya!
3. Mengapa ada proses-proses atau tahapan-tahapan yang harus dilakukan dalam rekayasa perangkat lunak?
4. Mengapa rekayasa perangkat lunak sebaiknya fokus pada pelanggan atau pengguna?
5. Mengapa faktor sosial dari teknologi informasi sering sekali diabaikan oleh pengembang aplikasi?

Daftar Pustaka

Sukamto, Rosa A., & Shalahuddin, M. 2016. *Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Object*. Bandung: Informatika.

- Kadir dan Triwahyuni. 2013. *Pengantar Teknologi Informasi*. Yogyakarta: Andi.
- Pressman, R.S. 2012. *Rekayasa Perangkat Lunak: Pendekatan Praktisi*. Yogyakarta: Andi.
- Pressman, Roger S., 2001, *Software Engineering A Practitioner's Approach 7th ed*, McGraw-Hill, New York.

Bab III

Perencanaan Perangkat Lunak

Setelah mempelajari bab **Perencanaan Perangkat Lunak**, maka diharapkan :

5. Mahasiswa mampu menyusun tujuan perencanaan proyek
6. Mahasiswa mampu menerapkan teknik pengumpulan data
7. Mahasiswa mampu menguraikan ruang lingkup perangkat lunak
8. Mahasiswa mampu menyusun estimasi proyek perangkat lunak

Proses pengembangan perangkat lunak dimulai dengan beberapa aktivitas yang secara kolektif disebut dengan project planning (perencanaan proyek). Perencanaan proyek memberikan sebuah peta jalan bagi suksesnya rekayasa perangkat lunak.

1. Tujuan Perencanaan Proyek

Tujuan perencanaan proyek perangkat lunak adalah untuk menyediakan sebuah kerangka kerja yang memungkinkan manajer membuat estimasi yang dapat dipertanggungjawabkan mengenai sumber daya, biaya dan jadwal. Tujuan perencanaan dicapai melalui suatu proses penemuan informasi yang menunjuk ke estimasi yang dapat dipertanggungjawabkan.

2. Teknik Pengumpulan Data

Beberapa teknik pengumpulan data yang dilakukan pada saat merencanakan proyek perangkat lunak, yaitu teknik wawancara, teknik observasi, dan teknik kuisisioner.

- a. Wawancara

Dengan wawancara, lebih mudah dalam menggali bagian sistem mana yang lebih baik, dapat menggali kebutuhan user secara lebih luas, User dapat mengungkapkan kebutuhan lebih bebas. Panduan dalam melakukan wawancara yaitu: Buat jadwal wawancara dengan narasumber, Panduan wawancara, Pertanyaan yang jelas, Catat hasil wawancara.

Pengumpulan data dengan menggunakan wawancara mempunyai beberapa keuntungan sebagai berikut.

- 1) Lebih mudah dalam menggali bagian sistem mana yang dianggap baik dan bagian mana yang dianggap kurang baik
- 2) Jika da bagian tertentu yang menurut anda perlu untuk digali lebih dalam, anda dapat langsung menanyakan kepada narasumber
- 3) Dapat menggali kebutuhan *user* secara lebih bebas
- 4) *User* dapat mengungkapkan kebutuhannya secara lebih bebas

Selain mempunyai beberapa kelebihan tersebut, teknik wawancara juga memiliki kelemahan. Berikut ini adalah beberapa kelemahan dari teknik wawancara.

- 1) Wawancara akan sulit dilakukan jika narasumber kurang dapat mengungkapkan kebutuhannya
- 2) Pertanyaan dapat menjadi tidak terarah, terlalu fokus pada hal-hal tertentu dan mengabaikan bagian lainnya

b. Observasi

Observasi dilakukan dengan melakukan Analisis dapat melihat langsung bagaimana sistem lama berjalan. Dengan observasi mampu menghasilkan gambaran lebih baik jika dibandingkan dengan teknik lain. Panduan dalam melakukan observasi, yaitu Tentukan hal-hal apa saja yang akan diobservasi, minta ijin pada yang berwenang pada bagian yang akan diobservasi, jika ada yang anda tidak mengerti, coba bertanya jangan berasumsi sendiri.

Pengumpulan data dengan menggunakan observasi mempunyai keuntungan, yaitu:

- 1) Analisis dapat melihat langsung bagaimana sistem lama berjalan
- 2) Mampu menghasilkan gambaran lebih baik jika dibanding dengan teknik lainnya

Kelemahan dengan menggunakan teknik observasi, yaitu :

- 1) Membutuhkan waktu cukup lama karena jika observasi waktunya sangat terbatas, maka gambaran sistem secara keseluruhan akan sulit untuk diperoleh
- 2) Orang-orang yang sedang diamati biasanya perilakunya akan berbeda dengan perilaku sehari-hari (cenderung berusaha terlihat baik).
- 3) Dapat mengganggu pekerjaan orang-orang pada bagian yang sedang diamati

c. Kuisisioner

Kuisisioner dilakukan dengan harapan Hasil lebih objektif karena dapat dilakukan pada banyak orang sekaligus, waktu lebih singkat. Panduan ketika akan membagikan kuisisioner, yaitu : hindari pertanyaan isian, buat pertanyaan yang tidak terlalu banyak, buat pertanyaan yang singkat padat dan jelas.

Pengumpulan data dengan menggunakan kuisisioner mempunyai keuntungan, yaitu :

- 1) Hasilnya lebih obyektif, karena kuisisioner dapat dilakukan kepada banyak orang sekaligus
- 2) Waktunya lebih singkat

Kelemahan pengumpulan data dengan menggunakan kuisisioner adalah sebagai berikut:

- 1) Responden cenderung malas untuk mengisi kuisisioner
- 2) Sulit untuk membuat pertanyaan yang singkat, padat, jelas dan mudah dipahami

3. Ruang Lingkup Perangkat Lunak

Penentuan ruang lingkup perangkat lunak merupakan aktivitas pertama dalam perencanaan proyek perangkat lunak. Ruang lingkup perangkat lunak menggambarkan fungsi, kinerja, batasan, interface dan reliabilitas. Fungsi yang digambarkan dalam statmen ruang lingkup dievaluasi dan disaring untuk memberikan awalan yang lebih detail pada saat estimasi dimulai. Pertimbangan kinerja melingkupi pemrosesan dan kebutuhan waktu respon. Batasan ini mengidentifikasi dari batas yang ditempatkan pada perangkat lunak oleh perangkat keras eksternal, memori, atau sistem informasi yang ada.

Teknik yang banyak dipakai secara umum untuk menjembatani jurang komunikasi antara pelanggan dan pengembang serta untuk memulai proses komunikasi adalah dengan melakukan pertemuan atau wawancara pendahuluan. Gause & Weinberg mengusulkan bahwa analisis harus dimulai dengan mengajukan pertanyaan-pertanyaan bebas konteks, yaitu serangkaian pertanyaan yang akan membawa pada pemahaman mendasar terhadap masalah, orang yang menginginkan suatu solusi, sifat solusi yang diharapkan, dan efektivitas pertemuan itu. Beberapa pertanyaan bebas konteks pada pelanggan yang meliputi tujuan keseluruhan, serta keuntungan :

- a. Siapa di belakang permintaan kerja ini?
- b. Siapa yang akan memakai solusi ini?
- c. Apakah yang akan menjadi keuntungan ekonomi dari sebuah solusi yang sukses?
- d. Adakah sumberdaya lain bagi solusi ini?

4. Estimasi Proyek Perangkat Lunak

Estimasi tidak akan pernah menjadi ilmu pasti, disebabkan banyaknya variabel (manusia, teknik, lingkungan dan politik) yang

mempengaruhi biaya dan usaha akhir yang diaplikasikan untuk mengembangkannya. Beberapa pilihan untuk mencapai estimasi :

- a. Menunda estimasi sampai akhir proyek
- b. Mendasarkan estimasi pada proyek – proyek yang mirip yang sudah dilakukan
- c. Menggunakan teknik dekomposisi yang relatif sederhana
- d. Menggunakan satu atau lebih model empiris bagi estimasi usaha dan biaya perangkat

Kesimpulan

Perencanaan proyek memberikan sebuah peta jalan bagi suksesnya rekayasa perangkat lunak. Beberapa teknik pengumpulan data yang dilakukan pada saat merencanakan proyek perangkat lunak, yaitu teknik wawancara, teknik observasi, dan teknik kuisisioner. Ruang lingkup perangkat lunak menggambarkan fungsi, kinerja, batasan, interface dan reliabilitas.

Evaluasi

Setelah menyimak materi perencanaan perangkat lunak, silahkan jawab pertanyaan berikut:

1. Kegiatan apa saja yang dilakukan pada saat perencanaan proyek?
2. Teknik yang paling sesuai untuk menjembatani komunikasi dengan user adalah? Jelaskan alasannya!
3. Sebutkan jenis-jenis kebutuhan pengembangan sistem informasi!
4. Hal-hal apa saja yang dilakukan pada tahap desain sistem?

Daftar Pustaka

- Kendall, J.E. & Kendall, K.E. 2010. Analisis dan Perancangan Sistem. Jakarta: Indeks.
- Marakas, G.M. 2006. System Analysis Design: an Active Approach. New York: Mc.Graw-Hill.
- Sukamto, Rosa A., & Shalahuddin, M. 2016. *Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Object*. Bandung: Informatika
- Pressman, R.S. 2012. Rekayasa Perangkat Lunak: Pendekatan Praktisi. Yogyakarta: Penerbit Andi.
- Pressman, Roger S., 2001, *Software Engineering A Practitioner's Approach 7th ed*, McGraw-Hill, New York.

Bab IV

Software Development Life Cycle (SDLC)

Setelah mempelajari bab **Software Development Life Cycle (SDLC)**, maka diharapkan :

3. Mahasiswa mampu menjelaskan definisi SDLC
4. Mahasiswa mampu menerapkan model SDLC

1. Definisi SDLC

SDLC atau *Software Development Life Cycle* merupakan proses pengembangan atau mengubah suatu system perangkat lunak dengan menggunakan atau mengubah suatu system perangkat lunak dengan menggunakan model-model dan metodologi yang digunakan orang untuk mengembangkan system perangkat lunak. SDLC juga merupakan pola yang diambil untuk mengembangkan sistem perangkat lunak, yang terdiri dari tahap-tahap: rencana (*planning*), analisis (*analysis*), desain (*design*), implementasi (*implementation*), uji coba (*testing*) dan pengelolaan (*maintenance*).



Gambar 4.1. Model siklus pengembangan sistem

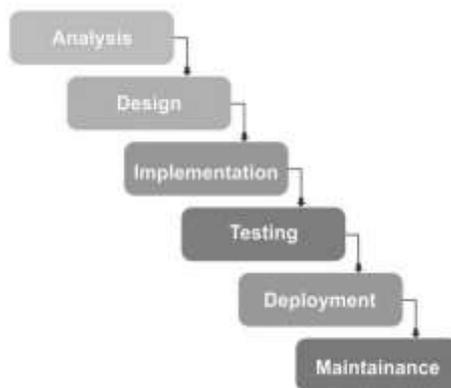
Dalam rekayasa perangkat lunak, konsep SDLC mendasari berbagai jenis metodologi pengembangan perangkat lunak. Metodologi-metodologi ini membentuk suatu kerangka kerja untuk perencanaan dan pengendalian pembuatan sistem informasi, yaitu

proses pengembangan perangkat lunak. Terdapat beberapa model SDLC yang dapat digunakan, semuanya mempunyai kelebihan dan kekurangan masing-masing pada tiap tahapannya. Hal yang paling penting yaitu mengenali type pelanggan/ customer dan memilih menggunakan model SDLC yang sesuai dengan karakter pelanggan dan sesuai dengan karakter pengembang.

2. Model SDLC

a. Model Waterfall

Model air terjun (Waterfall Model) adalah pendekatan klasik dalam pengembangan perangkat lunak yang menggambarkan metode pengembangan linier dan berurutan. Ini terdiri dari lima hingga tujuh fase, setiap fase didefinisikan oleh tugas dan tujuan yang berbeda, di mana keseluruhan fase menggambarkan siklus hidup perangkat lunak hingga pengirimannya. Setelah fase selesai, langkah pengembangan selanjutnya mengikuti dan hasil dari fase sebelumnya mengalir ke fase berikutnya.



Gambar 4.2 Model Waterfall

- 1) *Requirement Gathering and analysis* — Mengumpulkan kebutuhan secara lengkap kemudian dianalisis dan didefinisikan kebutuhan yang harus dipenuhi oleh program yang akan dibangun. Fase ini harus dikerjakan secara lengkap untuk bisa menghasilkan desain yang lengkap.
- 2) Desain ,dalam tahap ini pengembang akan menghasilkan sebuah sistem secara keseluruhan dan menentukan alur perangkat lunak hingga algoritma yang detail.
- 3) Implementasi adalah Tahapan dimana seluruh desain diubah menjadi kode kode program . Kode program yang dihasilkan masih berupa modul-modul yang akan diintegrasikan menjadi sistem yang lengkap.
- 4) *Integration & Testing*, Di tahap ini dilakukan penggabungan modul-modul yang sudah dibuat dan dilakukan pengujian ini dilakukan untuk mengetahui apakah software yang dibuat telah sesuai dengan desainnya dan fungsi pada software terdapat kesalahan atau tidak.
- 5) Verifikasi adalah klien atau pengguna menguji apakah sistem tersebut telah sesuai dengan yang disetujui.
- 6) *Operation & Maintenance* yaitu instalasi dan proses perbaikan sistem sesuai yang disetujui.

Keunggulan Model pendekatan pengembangan software metode waterfall adalah pencerminan kepraktisan rekayasa , yang bisa membuat kualitas software tetap terjaga. Jenis model yang bersifat lengkap sehingga proses pemeliharannya lebih mudah. Karena struktur logis dari model, kesalahan konseptual seringkali dapat dihindari. Model ini mengarah pada dokumentasi teknis yang luas, yang merupakan kelegaan bagi programmer dan pengembang baru dan juga berguna dalam tahap pengujian. Kemajuan proyek

dapat dipantau menggunakan tonggak sejarah. Total biaya dapat diperkirakan dengan akurasi relatif jika tidak ada konflik.

Kelemahan model waterfall ini adalah lambatnya proses pengembangan perangkat lunak. Dikarenakan proses yang satu tidak bisa diloncat-loncat maka dari itu model ini sangat memakan waktu dalam mengembangkannya. Kelemahan yang lain kinerja tidak optimal dan efisien. Konflik, bug, dan kesalahan program terkadang menyebabkan kenaikan biaya dan waktu yang cukup lama. Hal yang sama berlaku jika klien tidak puas. Spesifikasi yang awalnya dibuat seringkali sulit untuk dipahami oleh klien karena lebih abstrak daripada apa yang seharusnya dilakukan oleh perangkat lunak. Terutama dalam proyek-proyek outsourcing, ini bisa menjadi kerugian yang menentukan, karena tanggal rilis harus ditunda dan pasar mungkin telah berubah selama waktu ini.

Model waterfall adalah model SDLC yang paling sederhana, model ini hanya cocok untuk pengembangan perangkat lunak dengan spesifikasi yang tidak berubah-ubah.

b. Model Prototype

Prototyping adalah proses merancang sebuah prototype dimana prototype sendiri adalah sebuah model dari sebuah model produk yang mungkin belum memiliki semua fitur produk sesungguhnya namun sudah memiliki fitur – fitur utama dari produk sesungguhnya dan biasa digunakan untuk keperluan testing/uji coba untuk bahan uji coba sebelum berlanjut ke fase pembuatan produk sesungguhnya. Dengan metode prototyping ini pengembang dan pelanggan dapat saling berinteraksi selama proses pembuatan suatu produk.

Prototyping perangkat lunak adalah salah satu metode siklus hidup sistem yang didasarkan pada konsep model bekerja (working model). Tujuannya adalah mengembangkan model menjadi sistem final. Artinya sistem akan dikembangkan lebih cepat dari pada

metode tradisional dan biayanya menjadi lebih rendah. Ada banyak cara untuk melakukan prototyping, begitu pula dengan penggunaannya.



Gambar 4.3 Ilustrasi Model Prototype menurut Roger S. Pressman

Tahapan pengembangan model *prototype* yaitu:

- 1) Mendengarkan pelanggan, pada tahap ini dilakukan pengumpulan kebutuhan dari system dengan cara mendengar keluhan dari pelanggan. Untuk membuat suatu system yang sesuai kebutuhan, maka harus diketahui terlebih dahulu bagaimana system yang sedang berjalan untuk kemudian mengetahui masalah yang terjadi.
- 2) Merancang dan Membuat *Prototype*, pada tahap ini, dilakukan perancangan dan pembuatan prototype system. Prototype yang dibuat disesuaikan dengan kebutuhan system yang telah didefinisikan sebelumnya dari keluhan pelanggan atau pengguna.
- 3) Uji coba Pada tahap ini, *Prototype* dari system di uji coba oleh pelanggan atau pengguna. Kemudian dilakukan evaluasi kekurangan-kekurangan dari kebutuhan pelanggan. Pengembangan kemudian kembali mendengarkan keluhan dari pelanggan untuk memperbaiki Prototype yang ada.

Kelebihan model *prototype*, Adanya komunikasi yang baik antara pengembang dan pelanggan. Pengembangan dapat bekerja lebih baik dalam menentukan kebutuhan pelanggan. Lebih menghemat waktu dalam pengembangan system. Penerapan menjadi lebih mudah karena pemakai mengetahui apa yang diharapkannya. Pelanggan ikut dalam pengembangan sistem yang akan memudahkan pengembang mengetahui produk yang diharapkan pelanggan.

Kekurangan model *prototype* diantaranya Resiko tinggi yaitu untuk masalah-masalah yang tidak terstruktur dengan baik, ada perubahan yang besar dari waktu ke waktu, dan adanya persyaratan data yang tidak menentu. Interaksi pemakai penting. Sistem harus menyediakan dialog on-line antara pelanggan dan komputer. Hubungan pelanggan dengan komputer yang disediakan mungkin tidak mencerminkan teknik perancangan yang baik. Kurang fleksibel jika terjadi perubahan. Walaupun pemakai melihat berbagai perbaikan dari setiap versi *prototype*, tetapi pemakai mungkin tidak menyadari bahwa versi tersebut dibuat tanpa memperhatikan kualitas dan pemeliharaan jangka panjang.

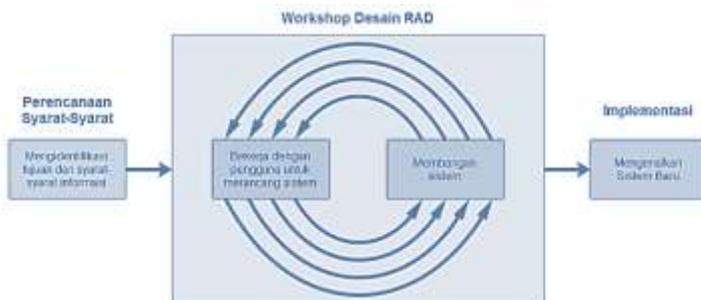
Model *prototype* sesuai jika diterapkan untuk menggali spesifikasi kebutuhan pelanggan secara lebih detail tetapi beresiko tinggi terhadap membengkaknya biaya dan waktu proyek.

c. Model Rapid Application Development (RAD)

Metode Rapid Application Development (RAD) adalah model pengembangan perangkat lunak yang pengembangannya tergolong dalam teknik incremental (bertingakat). *Rapid Application Development (RAD)* adalah strategi siklus hidup yang ditujukan untuk menyediakan pengembangan yang jauh lebih cepat dan mendapatkan hasil dengan kualitas yang lebih baik dibandingkan dengan hasil yang dicapai melalui siklus tradisional (McLeod, 2002).

Dalam proses pengembangannya model ini menggunakan metode iterative atau berulang dimana working model sistem dikonstruksikan di awal tahap pengembangan dengan tujuan menetapkan kebutuhan (requirement) user. Model ini membutuhkan waktu singkat dibanding metode lain yaitu sekitar 30-90 hari. Siklus kerjanya yang pendek juga membuat pengembangan sistem dengan model ini bisa diselesaikan dengan cepat. Tetapi terdapat kekurangan dari penggunaan model RAD sehingga kurang efektif penerapannya untuk proyek besar.

Pemaparan konsep yang lebih spesifik lagi dijelaskan oleh Pressman (2005) dalam bukunya, *“Software Engineering: A Practitioner’s Approach”*. Ia mengatakan bahwa RAD adalah proses model perangkat lunak inkremental yang menekankan siklus pengembangan yang singkat. Model RAD adalah sebuah adaptasi “kecepatan tinggi” dari model waterfall, di mana perkembangan pesat dicapai dengan menggunakan pendekatan konstruksi berbasis komponen. Jika tiap-tiap kebutuhan dan batasan ruang lingkup proyek telah diketahui dengan baik, proses RAD memungkinkan tim pengembang untuk menciptakan sebuah “sistem yang berfungsi penuh” dalam jangka waktu yang sangat singkat.



Gambar 4.4 Ilustrasi model RAD menurut Kendal 2010

RAD digunakan pada aplikasi sistem konstruksi, maka menekankan fase-fase. Ada tiga fase dalam RAD yaitu (kendall dan kendall, 2008) :

- 1) *Requirement planning*, dalam tahap ini diketahui apa saja yang menjadi kebutuhan sistem yaitu dengan mengidentifikasi kebutuhan informasi dan masalah yang dihadapi untuk menentukan tujuan, batasan-batasan sistem, kendala dan juga alternatif pemecahan masalah. Analisis digunakan untuk mengetahui perilaku sistem dan juga untuk mengetahui aktivitas apa saja yang ada dalam sistem tersebut.
- 2) *Design workshop*, yaitu mengidentifikasi solusi alternatif dan memilih solusi yang terbaik. Kemudian membuat desain proses bisnis dan desain pemrograman untuk data-data yang telah didapatkan dan dimodelkan dalam arsitektur sistem informasi. Tools yang digunakan dalam pemodelan sistem biasanya menggunakan unified modeling language (uml).
- 3) *Implentation*, setelah design workshop dilakukan, selanjutnya sistem diimplementasikan (coding) ke dalam bentuk yang dimengerti oleh mesin yang diwujudkan dalam bentuk program atau unit program. Tahap implementasi sistem merupakan tahap meletakkan sistem supaya siap untuk dioperasikan.

Kelebihan Metode *Rapid Application Development* (RAD), Sangat berguna dilakukan pada kondisi user tidak memahami kebutuhan apa saja yang digunakan pada proses pengembangan perangkat lunak. Metode RAD dapat dilakukan dengan waktu yang singkat yakni sekitar 30-90 hari. Biaya yang dikeluarkan menjadi lebih rendah karena waktu pengembangan yang singkat dan menggunakan komponen yang sudah ada. Proses pengiriman menjadi lebih mudah, hal ini dikarenakan proses pembuatan lebih banyak menggunakan potongan-potongan script.

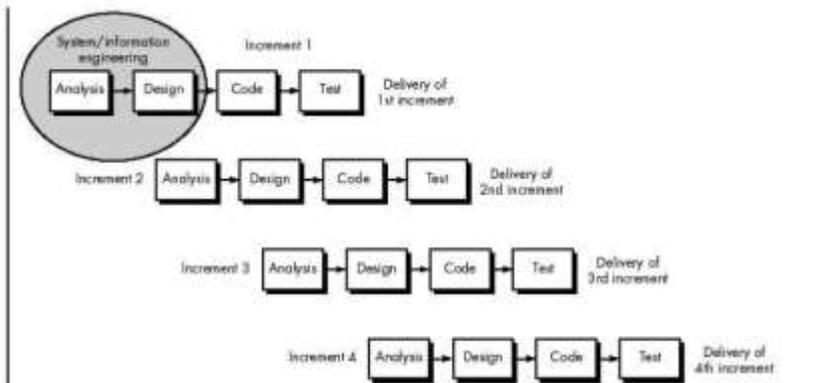
Mudah untuk diamati sehingga user lebih mengerti akan sistem yang dikembangkan. Lebih fleksibel karena pengembang dapat melakukan proses desain ulang pada saat yang bersamaan. Bisa mengurangi penulisan kode yang kompleks karena menggunakan wizard. Keterlibatan user semakin meningkat karena merupakan bagian dari tim secara keseluruhan. Mampu meminimalkan kesalahan-kesalahan dengan menggunakan alat-alat bantuan (*case tools*). Mempercepat waktu pengembangan sistem secara keseluruhan karena cenderung mengabaikan kualitas. Tampilan yang lebih standar dan nyaman dengan bantuan *software-software* pendukung.

Kekurangan *Rapid Application Development*, Beberapa hal yang perlu di perhatikan dalam implementasi pengembangan menggunakan model RAD : Membutuhkan SDM atau sumber daya manusia yang ahli untuk proyek berskala besar. RAD menuntut pengembang dan pelanggan memiliki komitmen dalam aktivitas *rapid fire* yang diperlukan untuk melengkapi sebuah sistem dalam waktu yang singkat. Jika komitmen tersebut tidak ada maka proyek RAD akan gagal. Dengan melakukan pembelian belum tentu bisa menghemat biaya dibandingkan dengan mengembangkan sendiri. Membutuhkan biaya tersendiri untuk membeli peralatan-peralatan penunjang seperti misalnya *software* dan *hardware*. Kesulitan melakukan pengukuran mengenai kemajuan proses. Kurang efisien karena apabila melakukan pengkodean dengan menggunakan tangan bisa lebih efisien. Ketelitian menjadi berkurang karena tidak menggunakan metode yang formal dalam melakukan pengkodean. Lebih banyak terjadi kesalahan apabila hanya mengutamakan kecepatan dibandingkan dengan biaya dan kualitas. Fasilitas-fasilitas banyak yang dikurangi karena terbatasnya waktu yang tersedia. Sistem sulit diaplikasikan di tempat yang lain. Fasilitas yang tidak perlu terkadang harus disertakan, karena menggunakan komponen yang sudah jadi, sehingga hal ini membuat biaya semakin meningkat.

d. Model Iteratif

Metode yang merupakan pengembangan dari *prototyping* model dan digunakan ketika requirement dari software akan terus berkembang dalam tahapan-tahapan pengembangan aplikasi tersebut. Sedikit pengertian tentang requirement software dari developer yang diterapkan pada tahap pertama iterasi, akan mendapatkan tanggapan dari user. Ketika requirement menjadi jelas, tahapan iterasi selanjutnya akan dilaksanakan.

Pada setiap iterasi, modifikasi desain yang dibuat dan kemampuan fungsional baru ditambahkan. Ide dasar di balik metode ini adalah untuk mengembangkan sistem melalui siklus berulang (iteratif) dan dalam porsi yang lebih kecil pada waktu (incremental).



Gambar 4.5 Ilustrasi model Iteratif

Pengembangan berulang dan Incremental adalah kombinasi dari kedua desain iteratif atau metode iteratif dan incremental membangun model untuk pembangunan. “Selama pengembangan perangkat lunak, lebih dari satu iterasi dari siklus pengembangan perangkat lunak mungkin berlangsung pada saat yang sama.” dan

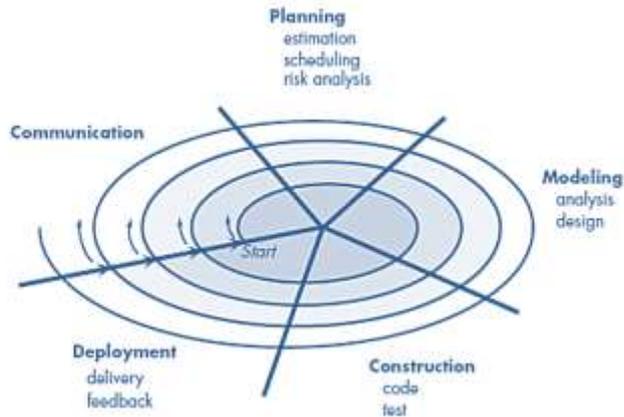
“Proses ini dapat digambarkan sebagai” akuisisi evolusi “atau” inkremental membangun “pendekatan.”

Keuntungan dari Iterative model, User dapat mencoba sistem yg sudah dikembangkan dan kemudian dapat memberikan masukan > keterlibatan user semakin intens dampak positif dalam pengembangan. Prototype relatif lebih mudah dibangun dan tidak memerlukan waktu yang lama. Dengan prototype, kesalahan & kelalaian dalam pengembangan dapat segera diketahui

Kelemahan dari Iterative model, Setiap iterasi bergantung prototype sebelumnya solusi final umumnya terjadi apabila ada perbedaan yg nyata pada prototype sebelumnya. Formal end-of-phase mungkin tidak terjadi, karena sangat sulit menentukan scope dari suatu prototype > proyek tidak pernah selesai. Dokumentasi seringkali tdk lengkap > fokus pada pembuatan prototype. Isu2 mengenai system backup & recovery, system performance dan system security, kurang/tidak diperhatikan dan sering terlupakan

e. Model Spiral

Model Spiral, merupakan model pengembangan system yang digambarkan berupa spiral. Model spiral ini tidak merepresentasikan rangkaian tahapan dengan penelusuran balik (back-tracking), tidak ada fase-fase tahapan yang tetap seperti spesifikasi atau perancangan. Setiap untaian pada spiral menunjukkan fase software process. Model spiral adalah model proses software yang evolusioner yang merangkai sifat iteratif dari prototipe dengan cara kontrol dan aspek sistematis dari model sekuensial linier. Model pengembangan perangkat lunak dengan metode spiral memiliki dua model yaitu prototyping dan waterfall. Model ini dikenal dengan sebutan Spiral Boehm.



Gambar 4.6 Ilustrasi model spiral

Fase-fase dari proses pengembangan perangkat lunak pada model spiral diwakili oleh tiap putaran. Dapat disimpulkan, putaran paling dalam berkaitan dengan kelayakan sistem, putaran berikutnya yaitu definisi kebutuhan, dilanjutkan dengan perancangan sistem, dan seterusnya. Putaran-putaran/loop pada model spiral dibagi menjadi empat sektor, yaitu:

1) *Objective Setting* (Penetapan Tujuan)

Hal-hal yang dilakukan pada tahap ini adalah mengidentifikasi kendala pada proses dan produk, tujuan spesifik untuk proyek, membuat rencana pengelolaan yang lebih rinci, merencanakan resiko dalam proyek, strategi alternative.

2) *Risk Assesment and Reduction* (Penilaian dan pengurangan resiko)

Tahapan pada risk manajemen reduction dilakukan melihat hasil identifikasi pada detail analisis pada resiko proyek di tahap sebelumnya. Sebagai contoh apabila ada

resiko bahwa requirement tidak pantas atau kurang, maka akan dibuat prototype sistem.

3) *Development and Validation* (Pengembangan dan Validasi)

Setelah tahap *risk assesment*, dapat dipilih model pengembangan sistem. Sebagai contoh, pembuatan prototype lembaran (throwaway) akan menjadi pendekatan pengembangan yang paling baik jika resiko user interface lebih besar atau dominan. Jika resiko utama yang diidentifikasi adalah integrasi subsistem, maka waterfall model mungkin adalah model pembangunan terbaik untuk digunakan.

4) *Planning* (Perencanaan)

Pada tahap planning, dilakukan review pada proyek dan pengambilan keputusan terkait kelanjutan tahapan pada putaran spiral selanjutnya. Apabilan telah diputuskan untuk dilanjutkan, rencana akan disusun untuk fase selanjutnya dari proyek.

Perbedaan utama model perangkat lunak lainnya dengan model spiral adalah penjelasan eksplisit mengenai resiko di tiap tahapan proses perangkat lunak yang dilalui (Ian Sommerville). Pada model spiral, Perangkat Lunak dikembangkan dalam seri dari versi evolusioner. Selama iterasi awal, versi sebelumnya akan menjadi protoype. Pada iterasi selanjutnya, terdapat kenaikan versi lebih komplit dari sistem yang diproduksi. Model spiral dibagi menjadi sebuah aktivitas *framework* yang didefinisikan oleh tim pengembang. Tiap aktivitas framework merepresentasikan 1 segmen dari jalur spiral. Sebagai permulaan proses evolusioner, tim pengembang melakukan aktivitas yang disiratkan oleh sirkuit spiral searah jarum jam, dimulai dari tengah. Resiko dipertimbangkan dari tiap evaluasi yang dibuat.

Spesifikasi produk dihasilkan di sirkuit pertama pada model spiral. Selanjutnya yaitu digunakan untuk mengembangkan sebuah prototype dan secara progresif menjadi versi Perangkat Lunak yang lebih canggih. Penyesuaian terhadap rencana proyek dihasilkan pada tiap lintasan yang melewati. Biaya dan jadwal pengembangan Perangkat Lunak disesuaikan berdasarkan feedback dari customer setelah diantarkan. Model spiral dapat diadaptasi untuk sepanjang hidup Perangkat Lunak, berbeda dengan model proses Perangkat Lunak yang lain, yang berhenti ketika Perangkat Lunak sampai kepada customer.

Sirkuit pertama dalam spiral model mempresentasikan konsep pengembangan proyek yang dimulai dari tengah spiral dan berlanjut untuk banyak iterasi sampai konsep pengembangan selesai. Spiral model akan tetap digunakan sampai Perangkat Lunak tidak digunakan. Jika dalam beberapa waktu proses terbengkalai, tetapi kapanpun perubahan diinisialisasi, proses dimulai pada entri poin yang sesuai.

Model spiral merupakan pendekatan yang realistis untuk mengembangkan sistem skala besar, karena dikembangkan sebagai proses yang berprogres, developer dan customer lebih baik mengerti dan memberi feedback pada tiap level evolusioner.

Kesimpulan

SDLC merupakan pola yang diambil untuk mengembangkan sistem perangkat lunak, yang terdiri dari tahap-tahap: rencana(planning), analisis (analysis), desain (design), implementasi (implementation), uji coba (testing) dan pengelolaan (maintenance). Model SDLC mempunyai kelebihan dan kekurangan masing-masing pada tiap tahapannya. Hal yang paling penting yaitu mengenali type pelanggan/ customer dan memilih menggunakan model SDLC yang sesuai dengan karakter pelanggan dan sesuai dengan karakter

pengembang. Beberapa model SDLC adalah model waterfall, model prototype, model RAD, model iterative, dan model spiral.

Evaluasi

Setelah menyimak materi SDLC, silahkan jawab pertanyaan berikut:

1. Apa risiko yang dihadapi jika pengembangan aplikasi tidak mengikuti tahapan-tahapan SDLC?
2. Jelaskan pendapat kalian alasan munculnya SDLC!
3. Mengapa model Waterfall dianggap sebagai model yang paling sederhana dan hanya cocok digunakan untuk aplikasi skala kecil?
4. Mengapa metode iterative cocok digunakan untuk pengembang dengan turnover staf yang tinggi?

Daftar Pustaka

- Kendall, Kenneth E. and Kendall, Julie E., 2011, *System Analysis and Design 8th ed*, Prentice Hall, New Jersey.
- Marakas, G.M. 2006. *System Analysis Design: an Active Approach*. New York: Mc.Graw-Hill.
- Pressman, R.S. 2012. *Rekayasa Perangkat Lunak: Pendekatan Praktisi*. Yogyakarta: Penerbit Andi.
- Pressman, Roger S., 2001, *Software Engineering A Practitioner's Approach 7th ed*, McGraw-Hill, New York.
- Simarmata, Janner. 2010. *Rekayasa Perangkat Lunak*. Yogyakarta: Andi
- Sommerville, Ian., 2011, *Software Engineering 9th ed*, Pearson Education, Boston.

Sukamto, Rosa A., & Shalahuddin, M. 2016. *Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Object*. Bandung: Informatika

Bab V

Basis Data

Setelah mempelajari bab **Basis Data**, maka diharapkan :

4. Mahasiswa mampu menjelaskan definisi basis data
5. Mahasiswa mampu menerapkan ERD
6. Mahasiswa mampu menerapkan studi kasus ERD

1. Definisi Basis Data

Basis data terdiri dari dua kata yaitu basis dan data, Basis dapat dikatakan gudang, markas, atau tempat berkumpul. Sedangkan data dapat diartikan representasi dari fakta dunia yang mewakili sebuah obyek (manusia, peristiwa, barang, keadaan dsb) yang direkam dalam bentuk angka, huruf, simbol, teks, gambar, bunyi atau kombinasinya. Elmasri menyampaikan bahwa istilah basis data lebih dibatasi pada arti implisit yang khusus mempunyai beberapa pengertian, yaitu :

- a. Basis data merupakan kumpulan data dari berbagai sumber yang secara logika mempunyai arti implicit. Sehingga apabila data terkumpul secara acak dan tanpa mempunyai arti, tidak dapat disebut basis data.
- b. Basis data dapat digunakan oleh beberapa pemakai dan beberapa aplikasi yang sesuai dengan kepentingan pemakai.
- c. Basis data merupakan penyajian suatu aspek dari dunia nyata (*real world* atau *miniworld*). Misalnya basis data perbankan, perpustakaan, pertanahan, perpajakan, dan lain-lain.
- d. Basis data perlu dirancang, dibangun dan data dikumpulkan untuk suatu tujuan tertentu.

Dari beberapa pengertian tersebut, dapat diambil kesimpulan bahwa sistem basis data merupakan sistem terkomputerisasi yang berujuan untuk memelihara data yang telah diolah dan

mempercepat proses saat dibutuhkan. Dengan kata lain basis data merupakan media untuk menyimpan data agar dapat diakses dengan mudah dan cepat. Kebutuhan basis data dalam sistem informasi meliputi memasukkan, menyimpan, mengambil kembali data dan membuat laporan berdasarkan data yang telah disimpan.

2. Entity Relationship Diagram

Pemodelan awal basis data yang paling banyak digunakan adalah *Entity Relationship Diagram* (ERD). ERD digunakan untuk pemodelan basis data relational. Diagram relasi entitas atau ERD merupakan suatu diagram dalam bentuk gambar atau simbol yang mengidentifikasi tipe dari entitas di dalam suatu sistem yang diuraikan dalam data dengan atributnya, dan menjelaskan hubungan atau relasi diantara entitas tersebut. Atau dapat dikatakan bahwa ERD adalah model jaringan yang menggunakan susunan data yang disimpan dalam sistem secara abstrak. ERD menekankan pada struktur dan relationship data.

Untuk dapat membuat entity relasional diagram, maka komponen yang harus terpenuhi adalah:

a. Obyek Data, Atribut dan Hubungan.

Obyek Data Adalah representasi dari hampir semua informasi gabungan yang harus dipahami oleh perangkat lunak. Objek data dapat berupa entitas eksternal (misalkan semua yang menghasilkan informasi), suatu benda (misal laporan atau tampilan), peristiwa (misalnya proses meminjam) atau event, peran (misalnya peminjam), unit organisasi atau suatu struktur. Sebagai contoh : orang atau mobil dapat dipandang sebagai objek data bila salah satu dari mereka dapat didefinisikan dalam bentuk atribut. Deskripsi objek data menghubungkan objek data dengan semua atributnya. Obyek data dihubungkan satu dengan yang lainnya,

misalkan seorang dapat memiliki mobil, dimana hubungan “memiliki” mengkonotasikan suatu hubungan khusus antara seorang dengan mobil.

Atribut Menentukan property suatu obyek data dan mengambil salah satu dari tiga karakteristik yang berbeda. Atribut dapat digunakan untuk:

- 1) Menamai sebuah contoh dari obyek data
- 2) Menggambarkan contoh
- 3) Membuat referensi ke contoh yang lain pada tabel yang lain.

Hubungan Obyek data disambungkan satu dengan lainnya dengan berbagai macam cara. Andaikan ada dua objek data, buku dan toko buku, obyek tersebut dapat diwakilkan dengan menggunakan dua notasi sederhana, dibangun suatu hubungan anatar buku dengan toko buku karena kedua obyek data tersebut berhubungan. Hubungan tersebut dapat berupa :

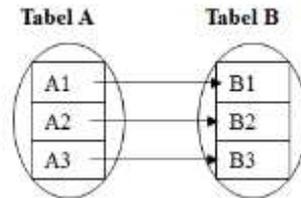
- 1) Toko buku memesan buku
- 2) Toko buku menampilkan buku
- 3) Toko buku menjual buku

Hubungan memesan, menampilkan, menjual mendefinisikan hubungan yang relevan antara buku dan toko buku. Penting untuk dicatat bahwa hubungan obyek mempunyai dua arah, dimana mereka dpaat dibaca dari dua arah, misalnya :toko buku memesan buku atau buku dipesan oleh toko buku.

b. Kardinalitas dan Modalitas Kardinalitas

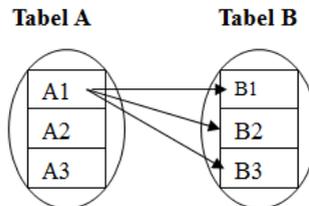
Model data harus dapat merepresentasikan jumlah peristiwa dari obyek di dalam hubungan yang diberikan.

- 1) Satu ke satu (1:1) Misalnya: seorang suami hanya dapat memiliki satu istri, dan seorang istri hanya mempunyai satu suami. Contoh ilustrasi dapat ditunjukkan pada Gambar 5.1



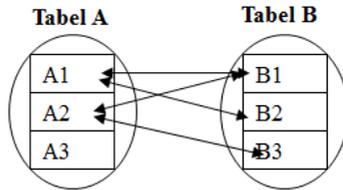
Gambar 5.1 Relasi Satu ke Satu

- 2) Satu ke banyak (1:N) Misalnya: seorang ibu kandung dapat memiliki banyak anak, tetapi seorang anak hanya dapat memiliki satu ibu kandung. Contoh ilustrasi dapat ditunjukkan pada Gambar 5.2



Gambar 5.2 Relasi Satu ke Banyak

- 3) Banyak ke banyak (M:N) Misalnya: seorang paman dapat memiliki banyak keponakan, sementara itu seorang keponakan dapat memiliki banyak paman. Contoh ilustrasi dapat ditunjukkan pada Gambar 5.3



Gambar 5.3 Relasi Banyak ke Banyak

Modalitas dari suatu hubungan adalah nol bila tidak ada kebutuhan eksplisit untuk hubungan yang terjadi atau hubungan itu bersifat opsional. Modalitas bernilai satu jika suatu kejadian dari hubungan merupakan perintah.

Untuk menggambarkan ERD setidaknya ada empat langkah yang harus dilakukan oleh perancang basis data yaitu:

- a. Menemukan atau mendefinisikan Entitas
- b. Menemukan atau mendefinisikan atribute
- c. Menemukan atau mendefinisikan Relasi
- d. Menggambarkan ERD menggunakan notasi-notasi standar

a. Entitas (*Entity*)

Entitas merupakan obyek yang mewakili sesuatu dalam dunia nyata dan dapat dibedakan antara satu dengan lainnya (unique). Setiap entitas memiliki beberapa atribut yang mendeskripsikan karakteristik dari objek tersebut. Atribut Dapat berupa:

- 1) Fisik (manusia, pegawai, dsb)
- 2) Abstrak/konsep (pekerjaan, mata kuliah, department, dsb)
- 3) Kejadian (penjualan, pembelian, peminjaman, dsb)

Entitas dapat dibedakan menjadi dua jenis yaitu **entitas kuat** dan **entitas lemah**. Entitas kuat yaitu keberadaannya tidak tergantung pada entitas yang lain, sementara **entitas lemah** adalah

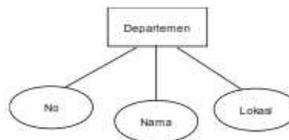
yang keberadaannya tergantung pada entitas lain. Notasi umum entitas kuat dengan nama entitas Anggota dan entitas lemah dengan nama entitas tanggungan ditunjukkan pada Gambar 5.4. Entitas tanggungan disebut sebagai **entitas lemah** karena jika data seorang pegawai dihapus maka data tanggungannya juga akan terhapus. Keberadaan data tanggungan tergantung pada data di pegawai. Lebih lanjut dapat dilihat pada Gambar 5.5.



Gambar 5.4 Contoh Notasi

Contoh :

Entitas	Atribut
Petugas	NIP, Nama, Alamat, Agama, Jenis kelamin
Departemen	No, Nama, Lokasi



Gambar 5.5 Contoh Penggunaan Simbol Entitas dan Atribut

Urutan langkah untuk mendefinisikan entitas dalam sistem adalah:

- Membuat gambaran cerita (ilustrasi) terkait sistem
- Menandai setiap objek yang diwakili oleh kata benda yang ada dalam ilustrasi
- Pada setiap objek, pastikan memiliki karakteristik yang akan dijadikan sebagai atribut
- Menentukan objek yang merupakan entitas (Jika memang ia memiliki karakteristik jadikan ia sebagai entitas)

- e. Menggambarkan entitas beserta atributnya menggunakan notasi simbol yang telah ditentukan.

3. Studi Kasus ERD

Contoh cara menentukan entitas:

Langkah 1. *Deskripsi tentang gambaran sistem (misalnya gambaran tentang system informasi perpustakaan):*

Departemen A mempunyai perpustakaan, perpustakaan ini dijaga oleh dua orang pegawai. Anggota perpustakaan ini adalah seluruh pegawai departemen yang sudah terdaftar menjadi anggota perpustakaan. Koleksi buku yang dimiliki ada sekitar 200 eksemplar, dengan berbagai jenis buku, pengarang dan penerbit. Untuk melakukan peminjaman buku, anggota melakukan proses peminjaman ke petugas dengan memberikan kartu anggota perpustakaan, lama peminjaman adalah dua minggu, proses pengembalian dilakukan dengan memberikan buku dan kartu anggota untuk pengecekan. Jika anggota terlambat mengembalikan maka akan dikenakan denda Rp.1000, 00 per hari.

Langkah 2. *Menandai setiap objek yang diwakili oleh kata benda yang ada dalam ilustrasi*

Departemen A mempunyai perpustakaan, perpustakaan ini dijaga oleh dua orang **pegawai**. **Anggota** perpustakaan ini adalah seluruh pegawai departemen yang sudah terdaftar menjadi anggota perpustakaan. Koleksi buku yang dimiliki ada sekitar 200 eksemplar, dengan berbagai **jenis buku, pengarang** dan **penerbit**. Untuk melakukan peminjaman buku, anggota melakukan proses **peminjaman** ke petugas dengan memberikan kartu anggota perpustakaan, lama peminjaman adalah dua minggu, proses

pengembalian dilakukan dengan memberikan buku dan kartu anggota untuk pengecekan. Jika anggota terlambat mengembalikan maka akan dikenakan **denda** Rp.1000, 00 per hari.

Langkah 3. Untuk setiap objek tersebut yakinkan bahwa ia memiliki karakteristik yang nanti disebut sebagai atribut

Pegawai : id, nama, Username, Password

Anggota : No_anggota, Nama, alamat, Nomor_telp, jenis_kelamin

Buku : Kode, kategori, Judul, Jumlah_halaman, ISBN, Pengarang, penerbit

Jenis buku : Kode, Jenis

Pengarang : Kode, Nama

Penerbit : Kode, Nama

Peminjaman : Kode_peminjaman, Id_Petugas, No_anggota, Kode_Buku, Tgl_pinjam, Tgl_kembali

Pengembalian : Kode_pengembalian, No_anggota, tgl_kembali, denda

b. Atribut

Atribut merupakan karakteristik atau sifat-sifat pada entitas. Nama atribut ini identik dengan nama kolom atau field pada suatu table dalam basis data. Kandidat Key adalah merupakan superkey yang jumlah atributnya paling sedikit.

Misalnya *candidat key* untuk entitas petugas antara lain:

- 1) Id_Pegawai
- 2) Nama (harus dipastikan bahwa tidak ada nama yang sama antara satu baris dengan baris yang lain)

Primary key merupakan suatu *candidat key* yang dipilih menjadi kunci utama karena sering dijadikan acuan untuk mencari

informasi, ringkas, menjadi keunikan suatu baris. Misalnya kodeBuku antara buku yang satu dengan buku yang lain pasti berbeda, dalam hal ini kodeBuku dapat digunakan sebagai suatu key. Gambar 5.6 menjelaskan simbol atau notasi primary key.



Gambar 5.6 Primary key

c. Relasi

Relasi menyatakan hubungan antara dua atau beberapa entitas. Setiap relasi mempunyai batasan (constraint) terhadap kemungkinan kombinasi entitas yang berpartisipasi. Batasan tersebut ditentukan dari situasi yang diwakili relasi tersebut. Ragam atau jenis relasi dibedakan menjadi beberapa macam antara lain adalah.

a) Relasi Binary

Relasi ini merupakan relasi antara 2 entitas. Relasi ini dibedakan menjadi :

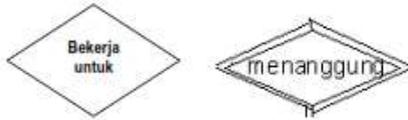
- 1) Relasi One-to-one (notasi 1:1)
- 2) Relasi One-to-many (notasi 1:N) atau many-to-one (notasi N:1)
- 3) Relasi Many-to-many (notasi M:N)

b) Relasi Ternary

Relasi ini merupakan relasi antara 3 entitas atau lebih. Dalam Relasi One-to-one (1:1) setiap atribut dari satu entitas berpasangan dengan satu attribute dari entitas yang direlasikan. Dalam relasi One-to-many (1:N) atau many-to-one (N:1) satu atribut berelasi dengan beberapa attribute dari entitas yang direlasikan. Dalam Many-to-many (M:N) satu atribut berelasi dengan beberapa attribute dari entitas yang direlasikan, begitu pula sebaliknya.

Notasi Relasi

Relasi dapat dikategorikan menjadi relasi kuat dan relasi lemah. gambar 5.7 menunjukkan notasi umum untuk relasi kuat dan relasi lemah.



Gambar 5.7 Relasi kuat dan relasi lemah

Langkah-langkah yang digunakan untuk mengidentifikasi relasi yaitu :

- 1) Tandai setiap hubungan dari gambaran sistem yang diwakili oleh kata kerja yang ada di dalam ilustrasi gambaran sistem beserta entitas yang berhubungan
- 2) Lakukan identifikasi rasio kardinalitas dari setiap hubungan
- 3) Lakukan identifikasi batasan partisipasi dari setiap hubungan/relasi yang ada beserta kemungkinan atribut yang muncul dari setiap hubungan

Gambarkan hubungan tersebut dalam bentuk notasi diagram dan gabungkan dengan notasi Entitas dan atribut yang dibuat sebelumnya. Misalnya tentukan relasi untuk sistem informasi perpustakaan dengan melihat deskripsi sistem di atas.

Langkah Penyelesaian ditunjukkan pada langkah-langkah berikut:

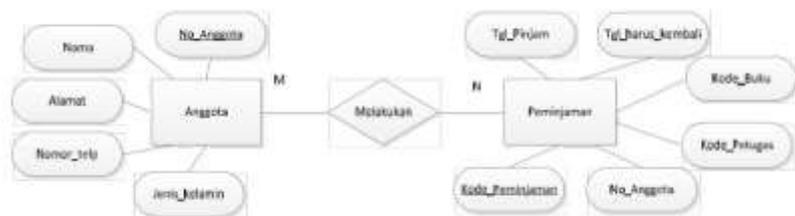
Langkah 1:

Tandai dan tentukan setiap hubungan yang ada pada gambaran sistem yang diwakili oleh kata kerja yang ada di dalam ilustrasi dan entitas yang berhubungan. Identifikasi hubungan antara entitas

Tabel Entitas dan Relasi

Entitas 1	Hubungan	Entitas 2
Anggota	Melakukan	Peminjaman
Pegawai	Mengelola	Peminjaman
Pengarang	Menulis	Buku
Penerbit	Menerbitkan	Buku
Anggota	Melakukan	Pengembalian
Pegawai	Mengelola	Pengembalian

Dari tabel di atas maka berikut relasinya :



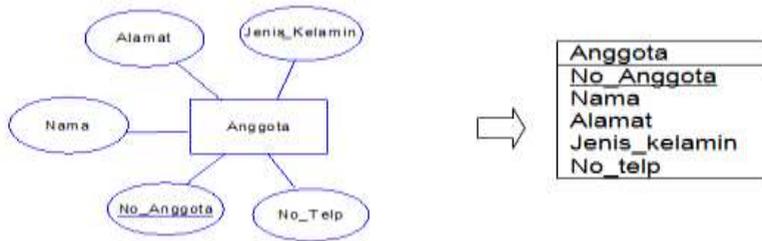
Gambar 5.8 Notasi hubungan entitas dan relasi

Langkah 2.

Mapping ER diagram ke tabel. Didalam data base yang menjadi pusat perhatian dan intisari dari sistem database itu adalah table dan relasinya. Tabel ini sama artinya dengan entitas, setiap orang bisa membuat table tetapi membuat table yang baik tidak semua orang dapat melakukannya. Kebutuhan akan membuat tabel yang baik ini maka muncul teori beberapa teori atau metode yaitu mapping ERD to table.

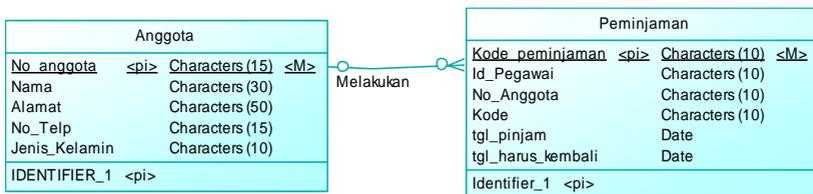
Contoh:

Dari relasi yang telah dibuat pada sistem informasi perpustakaan, pada setiap entitasnya pilih satu atribut kunci sebagai primary key (atribut harus unique).



Gambar 5.9. Mapping Notasi ke Diagram ER

Dengan cara yang sama dapat dilakukan mapping ERD to table pada semua entitas. Setelah semua entitas selesai dibuat, tentukan relasi antar entitas sesuai dengan tabel hubungan antar entitas di atas. Misalnya relasi antara entitas anggota dan Peminjamanya adalah melakukan seperti yang ditunjukkan pada Gambar 5.10



Gambar 5.10 Entity Relationship Diagram

Kesimpulan

Basis data merupakan kumpulan data dari berbagai sumber yang secara logika mempunyai arti implicit. Sehingga apabila data terkumpul secara acak dan tanpa mempunyai arti, tidak dapat disebut basis data. Pemodelan awal basis data yang paling banyak digunakan adalah Entity Relationship Diagram (ERD). ERD digunakan untuk pemodelan basis data relational. Diagram relasi entitas atau ERD adalah suatu diagram dalam bentuk gambar atau simbol yang mengidentifikasi tipe dari entitas di dalam suatu sistem yang

diuraikan dalam data dengan atributnya, dan menjelaskan hubungan atau relasi diantara entitas tersebut.

Evaluasi

Setelah menyimak materi basis data, silahkan jawab pertanyaan berikut:

1. Apa yang dimaksud dengan basis data?
2. Apakah fungsi basis data pada suatu sistem informasi?
3. Jelaskan yang kalian ketahui tentang ERD!
4. Jelaskan dengan singkat, langkah menyusun ERD!
5. Buatlah ERD untuk sistem informasi atau aplikasi yang berhubungan dengan dunia pendidikan!

Daftar Pustaka

- Elmasri dan Navathe. 2007. *Fundamentals of Database Systems, Fifth Edition*. Boston: Pearson Education, Inc. Addison Wesley
- Fatansyah. 2012. *Basis Data*. Bandung: Informatika
- Indrajani. 2015. *Database Design*. Jakarta: Elex Media Komputindo
- Kadir, Abdul. 2008. *Belajar Database menggunakan MySQL*. Yogyakarta : Andi Offset
- Sukamto, R. A., & Shalahuddin, M. 2016. *Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Object*. Bandung: Informatika

Bab VI

Data Flow Diagram (DFD)

Setelah mempelajari bab **Data Flow Diagram**, maka diharapkan :

5. Mahasiswa mampu menjelaskan definisi DFD
6. Mahasiswa mampu menjelaskan komponen DFD
7. Mahasiswa mampu menerapkan tingkatan level DFD
8. Mahasiswa mampu menerapkan tahapan pembuatan DFD
9. Mahasiswa mampu menerapkan studi kasus DFD

1. Definisi Data Flow Diagram

Data Flow Diagram (DFD) atau dalam bahasa Indonesia menjadi Diagram Alir Data adalah representasi grafik yang menggambarkan aliran informasi dan transformasi informasi yang diaplikasikan sebagai data yang mengalir dari input dan output. Penjelasan lain terkait DFD yaitu suatu cara atau metode untuk membuat rancangan sebuah sistem yang mana berorientasi pada alur data yang bergerak pada sebuah sistem nantinya. Dalam pembuatan Sistem Informasi, DFD sering digunakan. DFD dibuat oleh para analis untuk membuat sebuah sistem yang baik. Dimana DFD ini nantinya diberikan kepada para programmer dimana para programmer melakukan sebuah coding sesuai dengan DFD yang dibuat oleh para analis sebelumnya.

DFD merupakan sebuah teknik analisis yang digunakan untuk menggambarkan aliran input dalam sebuah sistem yang diolah oleh proses dan menghasilkan suatu keluaran (*output*). Diagram ini menggambarkan apa yang terjadi dalam sebuah sistem. DFD disajikan dalam bentuk gambar yang berisi simbol atau notasi yang digunakan untuk memahami mekanisme aliran data dalam suatu sistem. DFD merupakan alat perancangan sistem yang berorientasi pada alur data dengan konsep dekomposisi. DFD dapat pula digunakan untuk menggambarkan suatu analisa maupun rancangan

sistem yg mudah dikomunikasikan oleh profesional sistem (sistem analyst) kepada pemakai (*user*) maupun pembuat program (*programmer*).

Fungsi dan manfaat DFD antara lain adalah

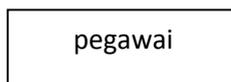
- a. DFD membantu para analis sitem meringkas informas tentang sistem, mengetahui hubungan antar sub-sub sistem, membantu perkembangan aplikasi secara efektif.
- b. DFD berfungsi sebagai alat komunikasi yang baik antara pemakai dan analis sistem.
- c. DFD dapat menggambarkan sejumlah batasan otomasi untuk pengembangan alternative sistem fisik.

2. Komponen DFD

Terdapat beberapa notasi atau simbol yang digunakan dalam DFD. Notasi tersebut merupakan karakteristik dari suatu system. Notasi dan simbol-simbol yang sering dipakai antara lain adalah : yaitu :

- a. *Terminator* atau *External Entity*.

Terminator disimbolkan dalam bentuk persegi panjang, yang mewakili entity luar dimana sistem berkomunikasi. Biasanya notasi ini melambangkan orang atau kelompok orang misalnya organisasi diluar sistem, grup, departemen, perusahaan pemerintah, dan berada di luar kontrol sistem yang dimodelkan. Pada sejumlah kasus dapat merupakan sistem lain, sebagai contoh adalah entity: pegawai, mahasiswa, peserta didik dll. Gambar dibawah ini menjelaskan notasi atau simbol Terminator (External Entity)



- b. Proses

Komponen proses menggambarkan suatu transformasi input menjadi output. Penamaan proses disesuaikan dgn proses atau kegiatan yang sedang dilakukan. Pemberian nama suatu proses menggunakan kata kerja transitif yaitu kata kerja yang membutuhkan objek. Komponen Proses disimbolkan dalam bentuk lingkaran dengan nama proses ditulis didalam lingkaran. Terdapat beberapa hal yang harus diperhatikan tentang komponen proses dalam diagram alur data antara lain adalah:

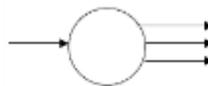
- 1) Setiap komponen proses harus memiliki input dan output.
- 2) proses dapat dihubungkan dgn komponen terminator, data store atau alur data.
- 3) Sistem, bagian, divisi atau departemen yang sedang dianalisis oleh profesional sistem dapat digambarkan dengan komponen proses.

Terdapat 4 kemungkinan yang dapat terjadi dalam suatu proses sehubungan dengan pengolahan komponen input menjadi output yaitu :

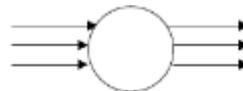
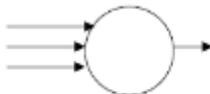
- 1) Satu input dan satu output
- 2) Satu input dan banyak output
- 3) Banyak input dan satu output
- 4) Banyak input dan banyak output



1) Satu input dan satu output



2) Satu input dan banyak output

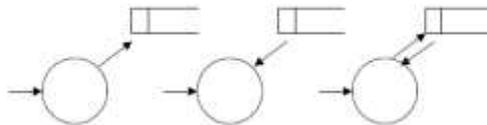


- 3) Banyak input dan satu output 4) banyak input dan banyak output

c. Penyimpanan Data (*Data Store*)

Data store digunakan untuk memodelkan kumpulan data atau paket data. Penyimpanan data kadangkala didefinisikan sebagai suatu mekanisme diantara dua proses yang dibatasi oleh jangka waktu tertentu. Data store dapat berupa file atau basis data yang tersimpan dalam unit penyimpanan seperti disket, harddisk. Alur data digunakan untuk menerangkan perpindahan data / paket data dari satu bagian ke bagian lainnya. Alur data dapat berupa kata, pesan, formulir atau informasi. Data store disimbolkan dengan garis sejajar. Terdapat beberapa hal yang harus diperhatikan dalam pendefinisian data store antara lain ialah :

- 1) Alur data dari proses menuju data store. Hal ini berarti bahwa data store berfungsi sebagai tujuan atau tempat penyimpanan dari suatu proses (proses write).
- 2) Alur data dari data store ke proses. Hal ini berarti data store berfungsi sebagai sumber atau suatu proses memerlukan data (proses read).
- 3) Alur data dari proses menuju data store dan sebaliknya. Hal ini berarti alur data berfungsi sebagai sumber dan tujuan (proses update)



d. Aliran data (*flow*)

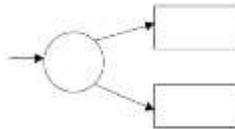
Alur data digunakan untuk menerangkan perpindahan data atau paket data dari satu bagian ke bagian lainnya. Alur data

menunjukkan aliran data yang dapat berupa masukan untuk sistem atau hasil proses system. Terdapat beberapa konsep tentang alur data antara lain adalah :

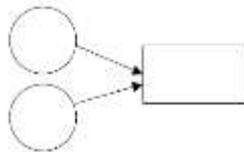
- 1) *Packets of data*. Apabila ada 2 data atau lebih yg mengalir dari 1 sumber yang sama menuju pada tujuan yang g sama pula dan mempunyai hubungan maka digambarkan dengan 1 alur data.



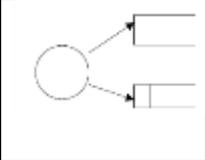
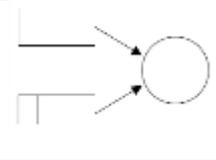
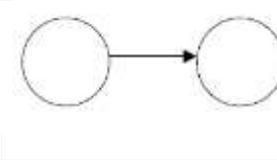
- 2) *Diverging data flow*. Apabila ada sejumlah paket data yg berasal dari sumber yg sama menuju pada tujuan yang berbeda atau paket data yg kompleks maka diagram dibagi menjadi beberapa elemen data yg dikirim ke tujuan yg berbeda.



- 3) *Converging data flow*. Alur data yang mengumpul menunjukkan beberapa alur data yang berbeda dari sumber data yang berbeda bergabung bersama-sama menuju tujuan yang sama.



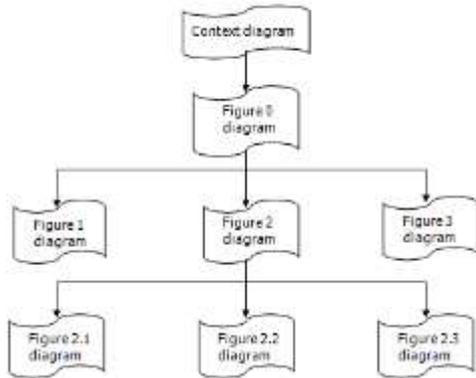
- 4) Sumber dan Tujuan. Arus data harus dihubungkan pada proses, baik dari maupun menuju proses.

		
Dari proses ke bukan proses	Dari bukan proses menuju proses	Dari proses ke proses

3. Tingkatan atau Level DFD

DFD dapat dibagi menjadi beberapa level yang lebih detail untuk merepresentasikan aliran informasi atau fungsi yang lebih detail. Dalam penerapannya tidak ada aturan yang baku tentang tingkatan atau level DFD. Secara umum terdapat beberapa tingkatan yang sering digunakan antara lain adalah sebagai berikut

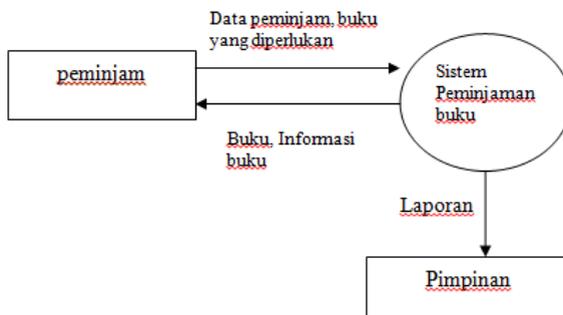
- a. DFD level 0 atau biasa disebut diagram konteks. Diagram konteks menggambarkan secara global aliran informasi dan data yang akan dilakukan oleh system. Diagram konteks ini merupakan level tertinggi (*top level*) yang menggambarkan hubungan antar system dengan entitas diluar system dan merupakan gambaran system secara keseluruhan. Komponen yang ada dalam diagram ini biasanya komponen proses, external entity dan alur data.
- b. DFD level 1, diagram ini menjelaskan lebih detail dari diagram konteks. Diagram ini merupakan dekomposisi diagram konteks. Beberapa proses dalam diagram konteks dapat dijelaskan lebih rinci.
- c. DFD level 2, diagram ini merupakan dekomposisi dari diagram level 1.
- d. Diagram level 3 dan seterusnya, diagram ini merupakan dekomposisi dari diagram level 2.



Adapun langkah-langkah yang dilakukan dalam pembuatan diagram alur data adalah sebagai berikut:

a. Membuat diagram konteks

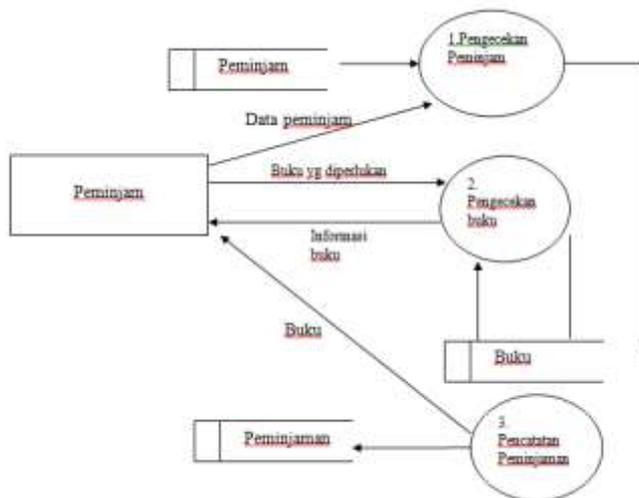
- 1) Tentukan nama sistemnya.
- 2) Tentukan batasan sistemnya.
- 3) Tentukan terminator apa saja yg ada dalam sistem.
- 4) Tentukan apa yg diterima dan diberikan terminator dari/pada sistem.
- 5) Gambarkan diagram *context*



b. Membuat diagram level 1

- 1) Tentukan proses utama yg ada pada sistem.

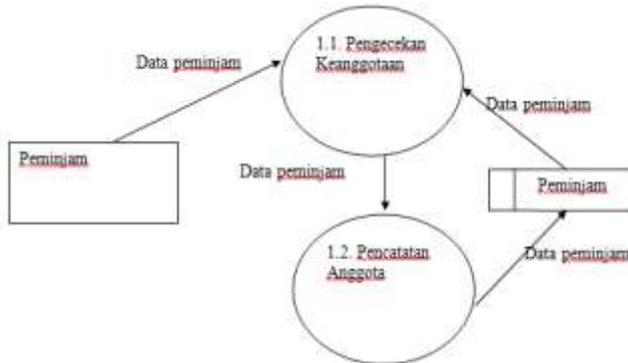
- 2) Tentukan apa yg diberikan/diterima masing-masing proses pada/dari sistem sambil memperhatikan konsep keseimbangan (alur data yg keluar/masuk dari suatu level harus sama dgn alur data yg masuk/keluar pada level berikutnya)
- 3) Apabila diperlukan, munculkan data store (master) sebagai sumber maupun tujuan alur data.
- 4) Gambarkan diagram level zero.
- 5) Hindari perpotongan arus data
- 6) Beri nomor pada proses utama (nomor tidak menunjukkan urutan proses).



c. Membuat diagram level 2

- 1) Tentukan proses yg lebih kecil (sub-proses) dari proses utama yg ada di level zero.
- 2) Tentukan apa yg diberikan/diterima masing-masing sub-proses pada/dari sistem dan perhatikan konsep keseimbangan.

- 3) Apabila diperlukan, munculkan data store (transaksi) sbg sumber maupun tujuan alur data.
- 4) Gambarkan DFD level Satu
- 5) Hindari perpotongan arus data.
- 6) Beri nomor pada masing-masing sub-proses yg menunjukkan dekomposisi dari proses sebelumnya. Misalnya 1.1, 1.2, 2.1



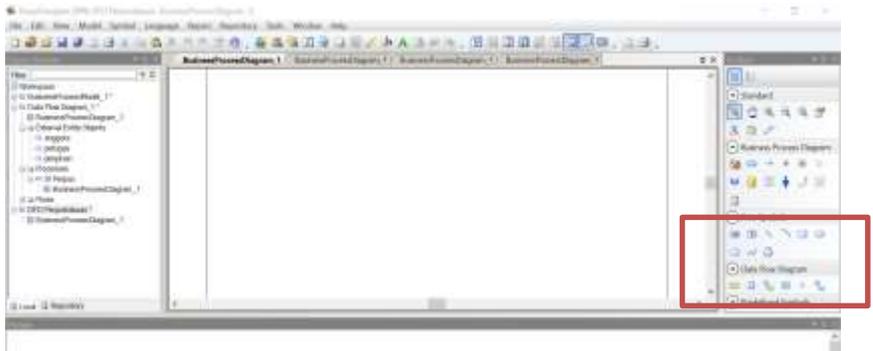
4. Studi Kasus Pembuatan DFD

Terdapat 3 pihak yang berkepentingan dalam sistem informasi ini yaitu anggota, pimpinan dan petugas. DFD Level 0 menyatakan tugas dan hak serang user dari sistem informasi ini, Setiap proses yang perlu dijelaskan diperlukan level yang lebih tinggi yang merupakan subproses dari proses yang ada.

Langkah 1 . Buka Power designer 16.5

Langkah 2. Pilih New Model – Information – Data Flow diagram

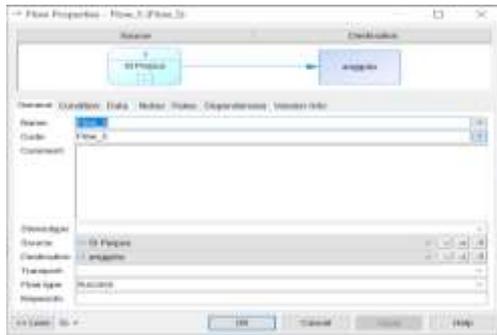




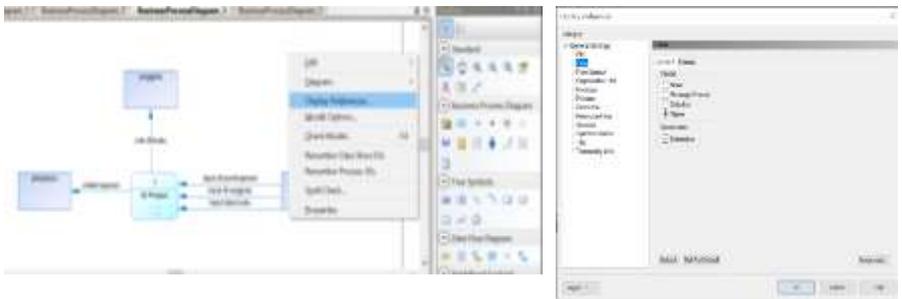
Langkah 4. Buat DFD Level Konteks (Diagram 0)



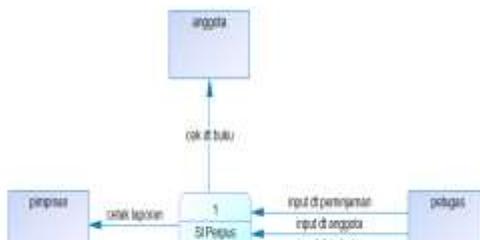
Langkah 5. Hubungkan Antar Eksternal Entity dengan Proses, dan beri nama pada masing-masing flow sesuai dengan hubungan antar entity ke proses



NB: Untuk menampilkan teks name pada data flow
Klik kanan pada area kosong di lembar kerja, pilih display preferences.

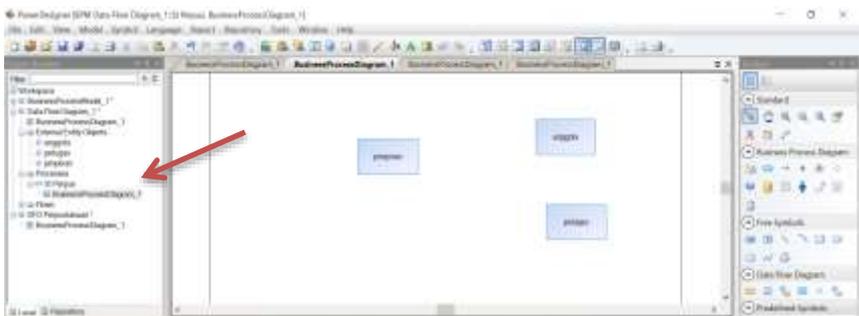


- ✓ Pilih flow dan centang pada name dan stereotype, klik set as default
- ✓ **Ulangi** pada *resource flow* dan centang name dan *stereotype*, klik *set as default*

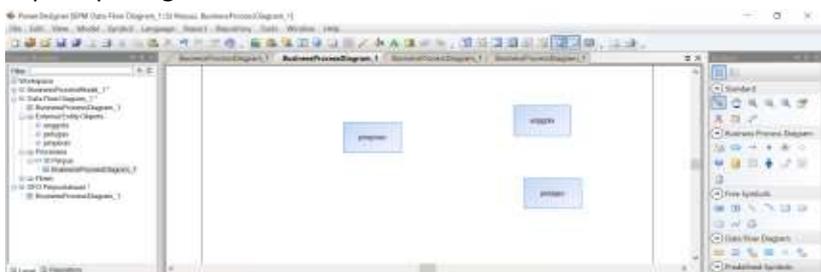


Langkah 6. Dekomposisi diagram level konteks untuk membuat Diagram level 1

- ✓ Klik kanan pada proses, pilih Decompose proses

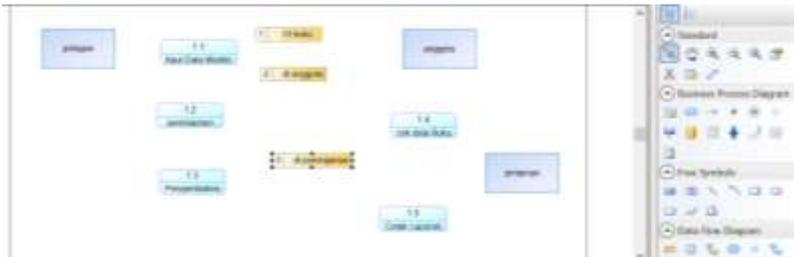


Klik pada menu kiri, pilih *Processes*, SI Perpus yang telah dibuat dan klik Business Process Diagram, sampai muncul eksternal entity seperti pada gambar di atas.

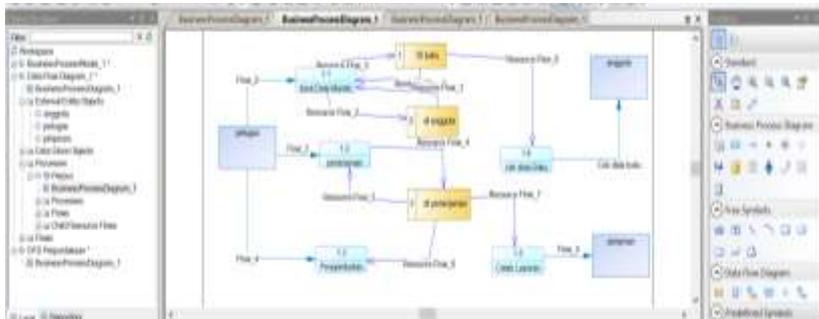


Langkah 7. Susun Diagram Level 1 pada DFD

Pada level 1, mulai disusun semua proses secara umum yang ada pada system (sesuai kebutuhan) yang ada pada diagram level konteks yang telah dibuat dan data store yang digunakan.



Hubungkan setiap *Entity*, proses, dan data store



✚ Kesimpulan

DFD merupakan sebuah teknik analisis yang digunakan untuk menggambarkan aliran input dalam sebuah sistem yang diolah oleh proses dan menghasilkan suatu keluaran (output). DFD disajikan dalam bentuk gambar yang berisi simbol atau notasi yang digunakan untuk memahami mekanisme aliran data dalam suatu sistem. DFD merupakan alat perancangan sistem yang berorientasi pada alur data dengan konsep dekomposisi. DFD dapat dibagi menjadi beberapa level yang lebih detail untuk merepresentasikan aliran informasi atau fungsi yang lebih detail. Tingkatan level pada DFD yaitu DFD level konteks atau nol, level 1, level 2, dan seterusnya.

Evaluasi

Setelah menyimak materi *data flow diagram*, silahkan jawab pertanyaan berikut:

1. Apa yang kalian ketahui tentang DFD?
2. Jelaskan yang kalian ketahui tentang diagram konteks!
3. Bagaimana tahapan melakukan perancangan sistem menggunakan DFD?
4. Buatlah perancangan pemrograman terstruktur menggunakan DFD dan kamus data untuk studi kasus sistem informasi apotek!

Daftar Pustaka

- Dennis, Alan., Wixom, Barbara H. and Roth, Roberta M., 2012, *System Analysis & Design 5th ed*, John Wiley & Sons, New Jersey.
- Kendall, J.E. & Kendall, K.E. 2010. Analisis dan Perancangan Sistem. Jakarta: Indeks.
- Marakas, G.M. 2006. System Analysis Design: an Active Approach. New York: Mc.Graw-Hill.
- Mc.,Leod, R. Jr. 2002. System Development: A Project Management Approach. New York: Leigh Publishing LLC.
- Simarmata, Janner. 2010. *Rekayasa Perangkat Lunak*. Yogyakarta: Andi
- Sommerville, Ian., 2011, *Software Engineering 9th ed*, Pearson Education, Boston.
- Sukamto, Rosa A., & Shalahuddin, M. 2016. *Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Object*. Bandung: Informatika

Bab VII **Pemodelan UML**

Setelah mempelajari bab **Pemodelan UML** , maka diharapkan :

6. Mahasiswa mampu menjelaskan kompleksitas pengembangan perangkat lunak
7. Mahasiswa mampu menjelaskan pemodelan perangkat lunak
8. Mahasiswa mampu menerapkan UML
9. Mahasiswa mampu menerapkan Use Case Diagram
10. Mahasiswa mampu menerapkan Activity Diagram

1. Kompleksitas Pengembangan Perangkat Lunak

Mengelola pengembangan perangkat lunak bukanlah hal yang mudah dikarenakan harus menyatukan ide dari beberapa orang yang mempunyai pemikiran berbeda. Kompleksitas sebuah perangkat lunak dapat dilihat dari hal-hal berikut:

- a. Kompleksitas domain atau permasalahan perangkat lunak.
- b. Kesulitan mengelola proses pengembangan perangkat lunak
- c. Kemungkinan fleksibilitas perubahan perangkat lunak
- d. Permasalahan karakteristik bagian-bagian perangkat lunak secara diskrit.

2. Pemodelan Perangkat Lunak

Pemodelan dalam suatu rekayasa perangkat lunak merupakan suatu hal yang dilakukan di tahapan awal. Di dalam suatu rekayasa dalam perangkat lunak sebenarnya masih memungkinkan tanpa melakukan suatu pemodelan. Namun hal itu tidak dapat lagi dilakukan dalam suatu industri perangkat lunak. Pemodelan dalam perangkat lunak merupakan suatu yang harus dikerjakan di bagian awal dari rekayasa, dan pemodelan ini akan mempengaruhi perkerjaan-pekerjaan dalam rekayasa perangkat lunak tersebut.

Di dalam suatu industri dikenal berbagai macam proses, demikian juga halnya dengan industri perangkat lunak. Perbedaan proses yang digunakan akan menguraikan aktivitas-aktivitas proses dalam cara-cara yang berlainan. Perusahaan yang berbeda

menggunakan proses yang berbeda untuk menghasilkan produk yang sama. Tipe produk yang berbeda mungkin dihasilkan oleh sebuah perusahaan dengan menggunakan proses yang berbeda. Namun beberapa proses lebih cocok dari lainnya untuk beberapa tipe aplikasi. Jika proses yang salah digunakan akan mengurangi kualitas kegunaan produk yang dikembangkan.

Pada rekayasa perangkat lunak, banyak model yang telah dikembangkan untuk membantu proses pengembangan perangkat lunak. Model-model ini pada umumnya mengacu pada model proses pengembangan sistem yang disebut *System Development Life Cycle* (SDLC) seperti yang telah dibahas di bab sebelumnya. Pemodelan perangkat lunak digunakan untuk mempermudah langkah berikutnya dari pengembangan sebuah sistem informasi sehingga lebih terencana. Salah satu perangkat pemodelan adalah *Unified Modelling Language* (UML).

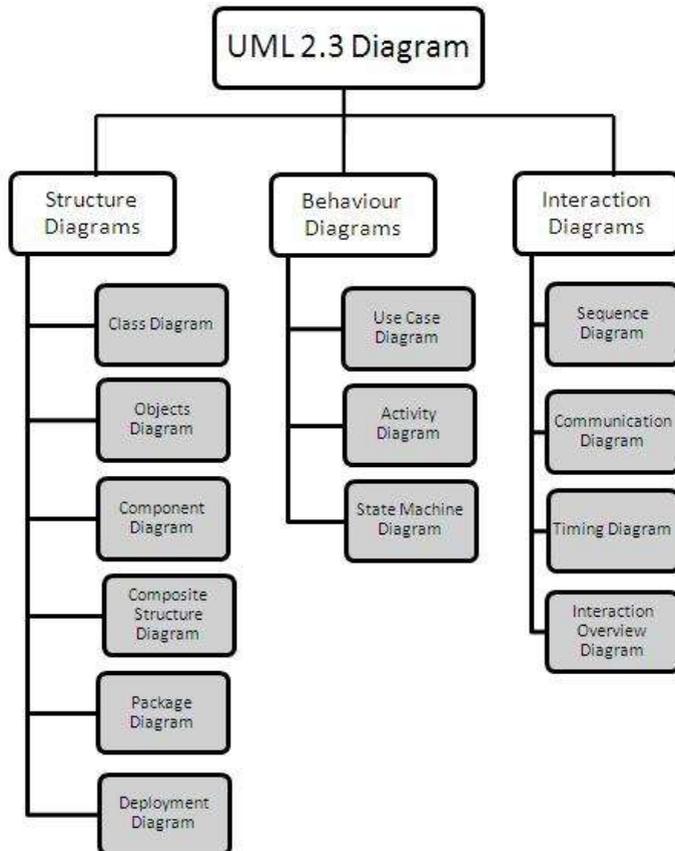
3. *Unified Modelling Language* (UML)

UML merupakan salah satu standart bahasa yang banyak digunakan di dunia industry untuk mendefinisikan *requirement*, membuat analisis & desain, serta menggambarkan arsitektur dalam pemrograman berorientasi objek. UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan mendokumentasikan sistem perangkat lunak. Pendekatan analisa & rancangan dengan menggunakan model *object oriented* mulai diperkenalkan sekitar pertengahan 1970 hingga akhir 1980 dikarenakan pada saat itu aplikasi *software* sudah meningkat dan mulai kompleks. Jumlah yang menggunakan metoda *object oriented* mulai diuji cobakan dan diaplikasikan antara 1989 hingga 1994, seperti halnya oleh Grady Booch dari Rational Software Co., dikenal dengan OOSE (*Object-Oriented Software Engineering*), serta James Rumbaugh dari

General Electric, dikenal dengan OMT (Object Modelling Technique). Kelemahan saat itu disadari oleh Booch maupun Rumbaugh adalah tidak adanya standar penggunaan model yang berbasis *object oriented*, ketika mereka bertemu ditemani rekan lainnya Ivar Jacobson dari *Objectory* mulai mendiskusikan untuk mengadopsi masing-masing pendekatan metoda *object oriented* untuk membuat suatu model bahasa yang uniform / seragam yang disebut UML (Unified Modeling Language) dan dapat digunakan oleh seluruh dunia.

Secara resmi bahasa UML dimulai pada bulan oktober 1994, ketika Rumbaugh bergabung *Booch* untuk membuat sebuah project pendekatan metoda yang uniform/seragam dari masing-masing metoda mereka. Saat itu baru dikembangkan draft metoda UML version 0.8 dan diselesaikan serta di release pada bulan oktober 1995. Bersamaan dengan saat itu, Jacobson bergabung dan UML tersebut diperkaya ruang lingkungannya dengan metoda OOSE sehingga muncul release version 0.9 pada bulan Juni 1996. Hingga saat ini sejak Juni 1998 UML version 1.3 telah diperkaya dan direspons oleh OMG (Object Management Group), Anderson Consulting, Ericsson, Platinum Technology, ObjectTime Limited, dll serta di pelihara oleh OMG yang dipimpin oleh Cris Kobryn. UML adalah standar dunia yang dibuat oleh Object Management Group (OMG), sebuah badan yang bertugas mengeluarkan standar-standar teknologi objectoriented dan software component.

UML versi 2.3 terdiri dari 13 macam diagram yang dikelompokkan dalam 3 kategori. Pembagian kategori dan macam-macam diagram dapat dilihat pada Gambar 7.1



Gambar 7.1 Bagan UML

Berdasarkan Gambar 7.1 berikut penjelasan singkat dari pembagian kategori tersebut:

- a. Structure diagram, merupakan kumpulan diagram yang digunakan untuk menggambarkan struktur statis dari sistem yang dimodelkan.
- b. Behavior diagram, merupakan kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada suatu sistem.

- c. Interaction diagram, merupakan kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antar subsistem pada suatu sistem.

Pada buku ajar ini akan difokuskan pada pemodelan UML dengan *use case* dan *activity diagram*. UML tidak hanya merupakan bahasa pemrograman visual saja, namun juga dapat secara langsung dihubungkan ke berbagai bahasa pemrograman, seperti JAVA, C++, Visual Basic, atau bahkan dihubungkan secara langsung ke dalam sebuah object-oriented database. Begitu juga mengenai pendokumentasian dapat dilakukan seperti; requirements, arsitektur, design, source code, project plan, tests, dan prototypes. Untuk dapat memahami UML membutuhkan bentuk konsep dari sebuah bahasa model, dan mempelajari 3 (tiga) elemen utama dari UML seperti building block, aturan-aturan yang menyatakan bagaimana building block diletakkan secara bersamaan, dan beberapa mekanisme umum (common).

a. **Building blocks**

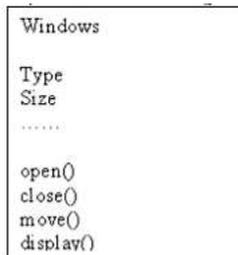
3 (tiga) macam yang terdapat dalam *building block* adalah kategori benda/Things, hubungan, dan diagram. Benda/things adalah abstraksi yang pertama dalam sebuah model, hubungan sebagai alat komunikasi dari bendabenda, dan diagram sebagai kumpulan / group dari benda-benda/things.

1) Benda/Things

Adalah hal yang sangat mendasar dalam model UML, juga merupakan bagian paling statik dari sebuah model, serta menjelaskan elemen-elemen lainnya dari sebuah konsep dan atau fisik. Bentuk dari beberapa benda/thing adalah sebagai berikut:

Pertama, sebuah kelas yang diuraikan sebagai sekelompok dari object yang mempunyai atribut, operasi, hubungan yang

semantik. Sebuah kelas mengimplementasikan 1 atau lebih interface. Sebuah kelas dapat digambarkan sebagai sebuah persegi panjang, yang mempunyai sebuah nama, attribute, dan metoda pengoperasiannya, seperti terlihat dalam Gambar 7.2.



Gambar 7.2 Sebuah kelas dari Model UML

Kedua, menggambarkan interface/antarmuka merupakan sebuah antarmuka yang menghubungkan dan melayani antar kelas dan atau elemen. Interface mendefinisikan sebuah set/kelompok dari spesifikasi pengoperasian, umumnya digambarkan dengan sebuah lingkaran yang disertai dengan namanya. Sebuah antar-muka berdiri sendiri dan umumnya merupakan pelengkap dari kelas atau komponen, seperti dalam Gambar 7.3.



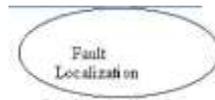
Gambar 7.3 sebuah interface / antar muka

Ketiga, collaboration yang didefinisikan dengan interaksi dan sebuah kumpulan / kelompok dari kelas-kelas/elemen-elemen yang bekerja secara bersama-sama. Collaborations mempunyai struktura dan dimensi. Pemberian sebuah kelas memungkinkan berpartisipasi didalam beberapa collaborations dan digambarkan dengan sebuah 'elips' dengan garis terpotong-potong.



Gambar 7.4 Collaborations

Keempat, sebuah 'use case' adalah rangkaian/uraian sekelompok yang saling terkait dan membentuk sistem secara teratur yang dilakukan atau diawasi oleh sebuah aktor. 'use case' digunakan untuk membentuk tingkah-laku benda/ things dalam sebuah model serta di realisasikan oleh sebuah collaboration. Umumnya 'use case' digambarkan dengan sebuah 'elips' dengan garis yang solid, biasanya mengandung nama, seperti terlihat dalam Gambar 7.5.



Gambar 7.5 Use case

Kelima, sebuah node merupakan fisik dari elemen-elemen yang ada pada saat dijalkannya sebuah sistem, contohnya adalah sebuah komputer, umumnya mempunyai sedikitnya memory dan processor. Sekelompok komponen mungkin terletak pada sebuah node dan juga mungkin akan berpindah dari node satu ke node lainnya. Umumnya node ini digambarkan seperti kubus serta hanya mengandung namanya, seperti terlihat dalam Gambar 7.6.



Gambar 7.6 Nodes

2) Hubungan/ *Relationship*

Ada 4 macam hubungan didalam penggunaan UML, yaitu; *dependency, association, generalization, dan realization.*

- a) *Dependency*, yaitu hubungan semantik antara dua benda/things yang mana sebuah benda berubah mengakibatkan benda satunya akan berubah pula. *Dependency* digambarkan sebuah panah dengan garis terputus-putus seperti terlihat dalam Gambar 7.7.



Gambar 7.7 *Dependency*

- b) *Association*, yaitu hubungan antar benda struktural yang terhubung diantara obyek. Kesatuan obyek yang terhubung merupakan hubungan khusus, yang menggambarkan sebuah hubungan struktural diantara seluruh atau sebagian. *Association* digambarkan dengan sebuah garis yang dilengkapi dengan sebuah label, nama, dan status hubungannya seperti terlihat dalam Gambar 7.8



Gambar 7.8 *Association*

- c) *Generalization*, yaitu menggambarkan hubungan khusus dalam obyek anak/child yang menggantikan obyek parent / induk . Dalam hal ini, obyek anak memberikan pengaruhnya dalam hal struktur dan tingkah lakunya kepada obyek induk. Digambarkan dengan garis panah seperti terlihat dalam Gambar 7.9.



Gambar 7.9 *Generalization*

- d) *Realization*, yaitu hubungan semantik antara pengelompokan yang menjamin adanya ikatan diantaranya. Hubungan ini dapat diwujudkan diantara interface dan kelas atau elements, serta antara use cases dan collaborations. Model dari sebuah hubungan realization seperti terlihat dalam Gambar 7.10.



Gambar 7.10 Realizations

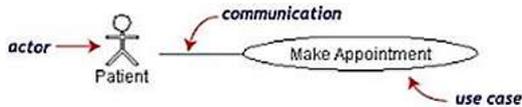
3) Diagram

UML sendiri terdiri atas pengelompokan diagram-diagram sistem menurut aspek atau sudut pandang tertentu. Diagram adalah yang menggambarkan permasalahan maupun solusi dari permasalahan suatu model. UML mempunyai 9 diagram, yaitu; *use-case, class, object, state, sequence, collaboration, activity, component, dan deployment diagram*. Diagram pertama adalah use case menggambarkan sekelompok use case dan aktor yang disertai dengan hubungan diantaranya.

4. Use Case Diagram

Use case diagram adalah teknik untuk merekam persyaratan fungsional sebuah system, menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Use case diagram menekankan kepada “apa” yang diperbuat oleh sistem, dan bukan “bagaimana”. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-create sebuah daftar belanja, dan sebagainya. Seorang atau sebuah aktor adalah sebuah entitas dapat

berupa manusia atau mesin yang berinteraksi dengan system untuk melakukan pekerjaan-pekerjaan tertentu.



Gambar 7.11 Use Case Diagram

a. Manfaat Use Diagram

Use case diagram akan sangat membantu kita dalam menyusun kebutuhan-kebutuhan (*requirement analisis*) sebuah sistem. Use case diagram akan di gunakan untuk mengkomunikasikan rancangan sistem dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.

b. Relationship dalam Use Case Diagram

Didalam use case diagram terdapat dua komponen penting yang saling berinteraksi dan berkomunikasi. Komponen tersebut adalah **aktor** dan **use case**. Ragam relasi yang terjadi diantara komponen-komponen tersebut adalah :

1) *Assosiations*

Relasi ini digunakan untuk menghubungkan para *actor* dan *case* di dalam suatu diagram. asosiasi ini dapat digambar dalam dua arah, tergantung dari kedudukan aktornya. jika aktornya adalah sebagai aktor utama (*primery actor*) maka arah relasi adalah dari *actor* menuju *case*. Jika aktornya adalah sebagai aktor tambahan (*secondary actor*) maka arah relasi dari *case* menuju *actor*.

2) *Dependency*

Relasi ini menyatakan suatu hubungan semantik antara dua unsur-unsur modeling yang mana perubahan satu elemen atau unsur model (unsur yang mengalir masuk) dapat mempengaruhi

unsur modeling yang lain (unsur yang dependent). Relasi ini menghubungkan antara use case satu dengan use case yang lain.

3) *Generalization/inheritance*

Relasi ini menyatakan hubungan hirarkis, turunan atau menyatakan suatu bagian dalam satu komponen. Sehingga aktor dapat diturunkan menjadi aktor-aktor yang lain atau use case dapat didekomposisi atau diturunkan menjadi use case yang lain. Relasi ini menunjukkan bahwa actor yang satu merupakan spesialisasi dari actor yang lain. Demikian juga use case satu merupakan spesialisasi dari usecase yang lain.

c. Karakteristik Use Case Diagram

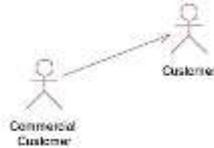
Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

d. Komponen Use Case Diagram

1) Aktor

Pada dasarnya *actor* bukanlah bagian dari *use case diagram*, namun untuk dapat terciptanya suatu *use case diagram* diperlukan beberapa *actor*. *Actor* tersebut mempresentasikan seseorang atau sesuatu (seperti perangkat, sistem lain) yang berinteraksi dengan sistem. Sebuah *actor* mungkin hanya memberikan informasi inputan pada sistem, hanya menerima informasi dari sistem atau keduanya

menerima, dan memberi informasi pada sistem. *Actor* hanya berinteraksi dengan *use case*, tetapi tidak memiliki kontrol atas *use case*. *Actor* digambarkan dengan *stick man*. *Actor* dapat digambarkan secara umum atau spesifik, dimana untuk membedakannya kita dapat menggunakan *relationship*.



Gambar 7.12 Contoh aktor pada Use Case

Ada beberapa kemungkinan yang menyebabkan *actor* tersebut terkait dengan sistem antara lain:

- Yang berkepentingan terhadap sistem dimana adanya arus informasi baik yang
- diterimanya maupun yang dia inputkan ke sistem.
- Orang ataupun pihak yang akan mengelola sistem tersebut.
- External resource* yang digunakan oleh sistem.
- Sistem lain yang berinteraksi dengan sistem yang akan dibuat.

2) Use Case

Use case adalah gambaran fungsionalitas dari suatu sistem, sehingga *customer* atau pengguna sistem paham dan mengerti mengenai kegunaan sistem yang akan dibangun.

Catatan : *Use case diagram* adalah penggambaran sistem dari sudut pandang pengguna

sistem tersebut (*user*), sehingga pembuatan *use case* lebih dititikberatkan pada fungsionalitas yang ada pada sistem, bukan berdasarkan alur atau urutan kejadian.

Cara menentukan *Use Case* dalam suatu sistem:

- Pola perilaku perangkat lunak aplikasi.
- Gambaran tugas dari sebuah *actor*.

- c) Sistem atau “benda” yang memberikan sesuatu yang bernilai kepada *actor*.
- d) Apa yang dikerjakan oleh suatu perangkat lunak (*bukan bagaimana cara mengerjakannya).



Gambar 7.13 Contoh Use Case

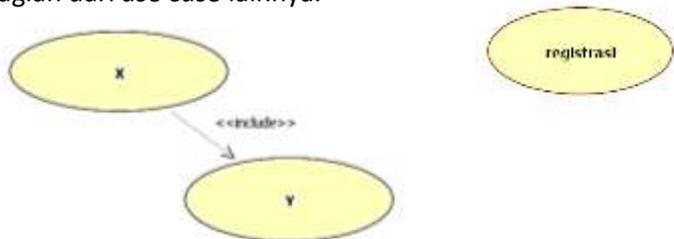
3) Relasi dalam Use Case

Ada beberapa relasi yang terdapat pada *use case diagram*:

- a) *Association*, menghubungkan link antar element.
- b) *Generalization*, disebut juga *inheritance* (pewarisan), sebuah elemen dapat merupakan spesialisasi dari elemen lainnya.
- c) *Dependency*, sebuah element bergantung dalam beberapa cara ke element lainnya.
- d) *Aggregation*, bentuk *association* dimana sebuah elemen berisi elemen lainnya.

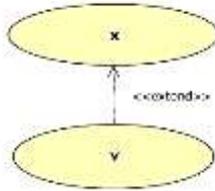
Sedangkan tipe relasi/ *stereotype* yang mungkin terjadi pada *use case diagram*:

- a) **<<include>>** , yaitu kelakuan yang harus terpenuhi agar sebuah *event* dapat terjadi, dimana pada kondisi ini sebuah *use case* adalah bagian dari *use case* lainnya.



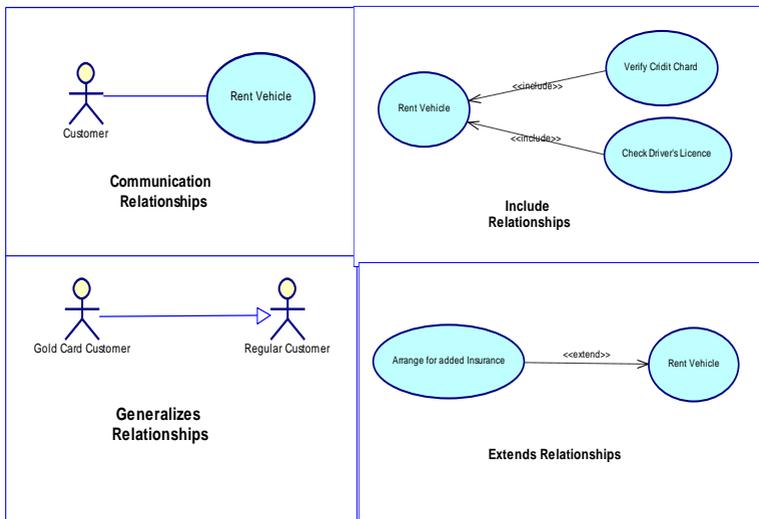
Gambar 7.14 Contoh relasi include

- b) **<<extends>>**, kelakuan yang hanya berjalan di bawah kondisi tertentu seperti menggerakkan alarm.



Gambar 7.15 Contoh relasi extends

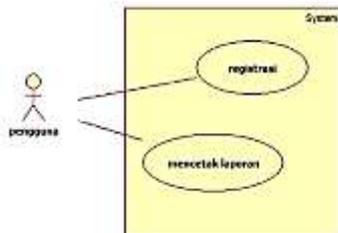
- c) **<<communicates>>**, mungkin ditambahkan untuk asosiasi yang menunjukkan asosiasinya adalah *communicates association*. Ini merupakan pilihan selama asosiasi hanya tipe *relationship* yang dibolehkan antara *actor* dan *use case*.



Gambar 7.16 Contoh seluruh relasi pada use case

4) Use Case Diagram

Use Case Diagram adalah gambaran graphical dari beberapa atau semua *actor*, *use case*, dan interaksi diantaranya yang memperkenalkan suatu sistem.

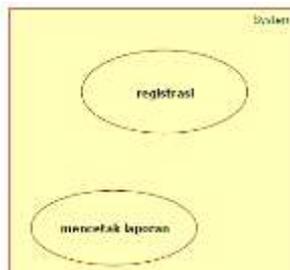


Gambar 7.17 Contoh Use Case Diagram

5) Batasan Sistem (*System Boundary*)

Dalam batasan sistem hal-hal yang penting yaitu:

- System boundary*.
- Mendefinisikan batas antara actor dan sistem.
- Dinotasikan bujur sangkar/persegi.
- Semua use case tercakup di dalamnya.



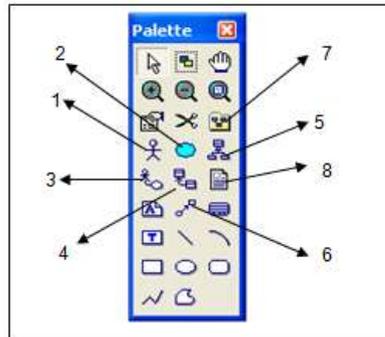
System Boundary

Gambar 7.18 Contoh System Boundary

e. Simbol-simbol (notasi) Use Case Diagram

Use case diagram menggunakan beberapa notasi atau simbol yang digunakan untuk menggambarkan fungsionalitas suatu sistem. Beberapa tools software menggunakan beberapa notasi yang agak berbeda akan tetapi fungsionalitasnya sama. Beberapa notasi use case diagram diperlihatkan dalam Gambar 19.

1. Actor.
2. Use case
3. asosiasi
4. dependency
5. Generalization/inheritance
6. Link/extended Dependency
7. package/kumpulan
8. File



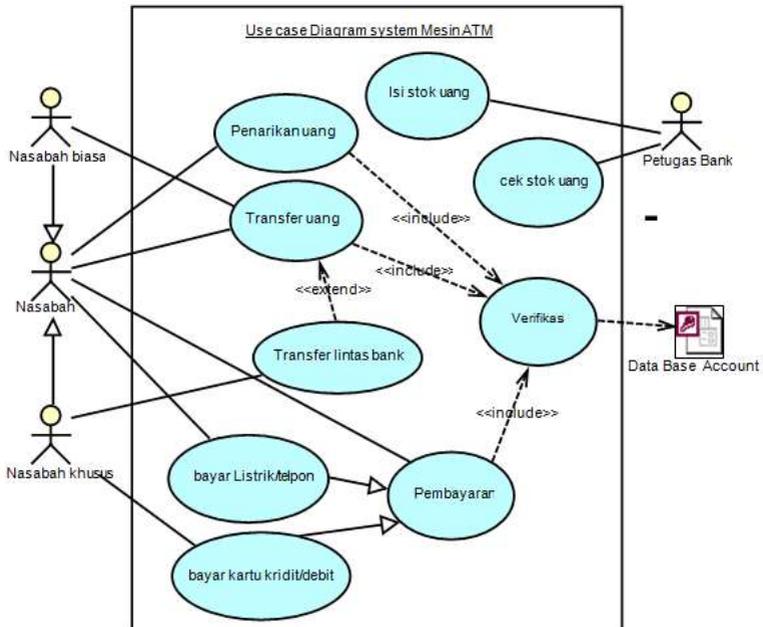
Gambar 7.19 Simbol pada use case Diagram

f. Contoh Use Case Diagram

Salah satu contoh sistem yang akan dijelaskan disini adalah "Sistem mesin ATM". Gambaran tentang proses business atau diskripsi persyaratan fungsionalitas dari sistem tersebut adalah sebagai berikut:

- 1) Nasabah dapat melakukan penarikan uang secara tunai
- 2) Nasabah dapat melakukan transaksi pembayaran seperti tagihan listrik, telepon, dll
- 3) Nasabah khusus dapat melakukan pembayaran kartu debit atau kredit
- 4) Nasabah biasa hanya dapat melakukan transfer uang ke rekening pada bank yang sama.
- 5) Nasabah khusus dapat melakukan transfer ke rekening pada bank yang berbeda

- 6) Untuk setiap transaksi, Sistem akan melakukan verifikasi ke database account
- 7) Petugas bank dapat mengecek persediaan uang di setiap ATM
- 8) Petugas bank mengisi ulang persediaan uang di mesin ATM



Gambar 7.20 Use case diagram sistem Mesin ATM

5. Activity Diagram

Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, decision yang mungkin terjadi, dan bagaimana mereka berakhir. Activity diagram juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. Activity diagram merupakan state diagram khusus, di mana sebagian besar state adalah action dan sebagian besar transisi di-trigger oleh selesainya

state sebelumnya (internal processing). Oleh karena itu activity diagram tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Activity diagram digunakan untuk menggambarkan langkah-langkah atau aktivitas pada suatu sistem. Pada setiap use case yang ada, maka terdapat paling sedikit satu activity diagram. Diagram ini menggambarkan proses bisnis dan urutan aktivitas dalam sebuah proses. Activity diagram dipakai pada business modeling untuk memperlihatkan urutan aktivitas proses bisnis. Struktur diagram ini mirip *flowchart* atau *Data Flow Diagram* (DFD) pada perancangan terstruktur. Sangat bermanfaat apabila kita membuat diagram ini terlebih dahulu dalam memodelkan sebuah proses untuk membantu memahami proses secara keseluruhan.

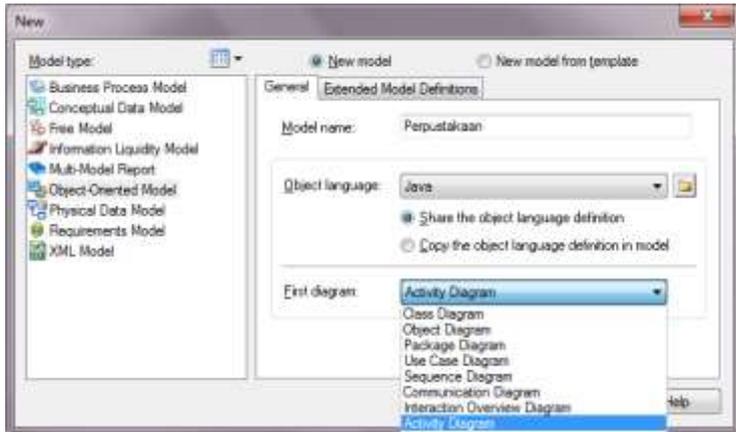
Activity diagram dibuat berdasarkan sebuah atau beberapa use case pada use case diagram. *Activity diagram* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, mulai dari mana berawal, kemungkinan yang bisa terjadi dan bagaimana berakhirnya serta menggambarkan proses yang mungkin terjadi pada beberapa eksekusi. *Activity diagram* tidak menggambarkan perilaku internal sebuah sistem tetapi lebih menggambarkan proses-proses dan jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu use case atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara use case menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

a. Langkah-langkah membuat Activity Diagram

Langkah-langkah yang dilakukan untuk membuat diagram activity adalah:

- 1) Buka aplikasi power designer
- 2) Dari menu pilih file → new atau menekan ctrl + N
- 3) Pilih *Object-Oriented Model* dan pada First diagram pilih *Activity diagram*



Gambar 7.21 Langkah awal Activity diagram

- 4) Berikut ini merupakan *Palette Power Designer* yang digunakan untuk membuat *activity diagram*.



Gambar 7.22 Palette Power Designer

Beberapa simbol yang digunakan untuk membuat *activity diagram*, yaitu:

Simbol	Keberangan
	Start Point
	End Point
	Activities
	Fork (Percabangan)
	Join (Penggabungan)
	Decision
Swimlane	Sebuah cara untuk mengelompokkan activity berdasarkan Actor (mengelompokkan activity dalam sebuah urutan yang sama)

Gambar 7.23 Simbol dalam activity diagram

Terdapat beberapa hal penting yang harus diketahui, yaitu:

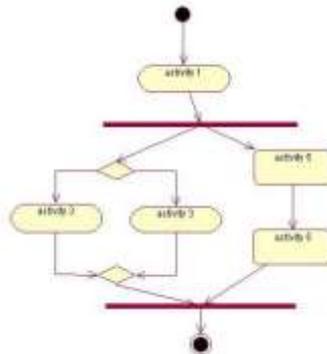
- 1) Activity menggambarkan sebuah pekerjaan atau tugas dalam workflow
- 2) Pada UML, activity digambarkan dengan simbol kotak 
- 3) Start state dengan tegas menunjukkan dimulainya suatu workflow pada sebuah activity diagram
- 4) Hanya ada satu start state dalam sebuah *workflow*
- 5) Pada UML, *start state* digambarkan dengan simbol lingkaran yang solid 

- 6) *End state* menggambarkan akhir atau terminal dari pada sebuah *activity diagram*
- 7) Bisa terdapat lebih dari satu *end state* pada sebuah *activity diagram*
- 8) Pada UML, *end state* digambarkan dengan simbol sebuah *bull's eye* 
- 9) *State transition* menunjukkan kegiatan apa berikutnya setelah suatu kegiatan sebelumnya.
- 10) Pada UML, *state transition* digambarkan oleh sebuah *solid line* dengan panah



- 11) *Decision* adalah suatu titik atau *point* pada *activity diagram* yang mengindikasikan suatu kondisi dimana ada kemungkinan perbedaan transisi
- 12) Pada UML, *decision* digambarkan dengan sebuah simbol *diamond/wajik*.

Contoh *activity diagram* sebagai berikut :



Gambar 7.24 *Activity diagram*

Setelah membuat *use case diagram* (DFD & ERD) guna menggambarkan aliran data dan sistem database yang digunakan, maka langkah selanjutnya adalah menggambarkan langkah-langkah atau aktivitas dari sistem tersebut yaitu dengan membuat *activity diagram*.

Berikut ini adalah contoh langkah-langkah atau prosedur untuk membuat *activity diagram* dari sebuah studi kasus.

b. Studi kasus : Membuat *activity diagram* perpustakaan

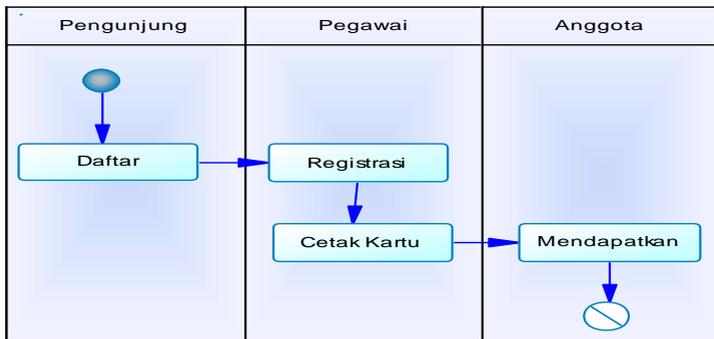
1) Proses pendaftaran anggota perpustakaan

Jika pengunjung perpustakaan ingin meminjam buku maka harus menjadi anggota, yang harus dilakukan pengunjung yaitu mendaftar, kemudian pustakawan meregristrasi lalu mencetak kartu anggota, setelah itu pustakawan memberikan kartu anggota, maka pengunjung sudah menjadi anggota dan dapat meminjam buku. Contoh scenario pendaftaran anggota perpustakaan dapat dilihat pada Tabel 7.1

Tabel 7.1 Skenario Pendaftaran Anggota

Name	Pendaftaran
Aktor	Pegawai, Pengunjung, Anggota
Purpose	Melakukan pendaftaran anggota perpustakaan
Typical Course of Events	
Actor Action	System Response
1. Pengunjung mendaftar dengan mengisi blanko biodata	4. sistem menampilkan form pendaftaran anggota perpustakaan
2. Pegawai membuka menu pendaftaran anggota perpustakaan	5. Sistem menerima input data anggota
3. Pegawai meregristrasi	
4. Pegawai mencetak kartu	

Pada aktifitas diagram diatas yaitu seorang mahasiswa yang ingin meminjam buku di perpustakaan kampus, tetapi mahasiswa tersebut belum mempunyai member atau belum pernah meminjam buku sama sekali dari perpus, oleh karena itu perlu adanya pendaftaran identitas si peminjam. Diagram aktifitas pendaftaran anggota dapat dilihat pada Gambar 7.24



Gambar 7.25. Aktifitas Diagram Pendaftaran Anggota

2) Proses peminjaman buku

Sebelum anggota meminjam buku diperpustakaan, anggota harus membawa kartu dan menunjukkannya kepada pustakawan. Pustakawan akan mengecek kartu, mengecek buku yang akan dipinjam jika tidak cocok maka selesai, jika cocok atau sesuai maka pustakawan memberikan buku kepada anggota untuk dipinjam lalu prosespun selesai. Contoh scenario peminjaman buku perpustakaan dapat dilihat pada Tabel 7.2

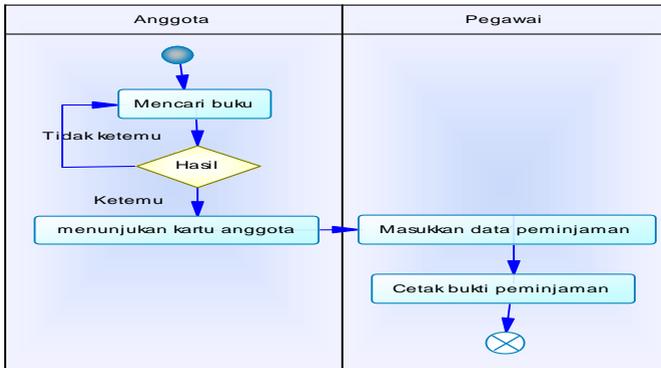
Tabel 7.2 Skenario Peminjaman Buku

Name	Peminjaman buku
Aktor	Pegawai, Anggota
Purpose	Melakukan peminjaman buku
Typical Course of Events	
Actor Action	System Response

- | | |
|--|--|
| 1. Anggota mencari data buku | 4. sistem menampilkan form peminjaman |
| 2. Menunjukkan kartu anggota | 5. Sistem menerima input data peminjaman |
| 3. Pegawai meinput data anggota dan buku dalam menu peminjaman | |

Diagram aktifitas peminjaman buku dapat dilihat pada Gambar

7.26



Gambar 7.26. Diagram aktifitas peminjaman buku

3) Diagram pengembalian buku

Contoh scenario pengembalian buku perpustakaan dapat dilihat pada Tabel 7.3

Tabel 7.3 Skenario Pengembalian Buku

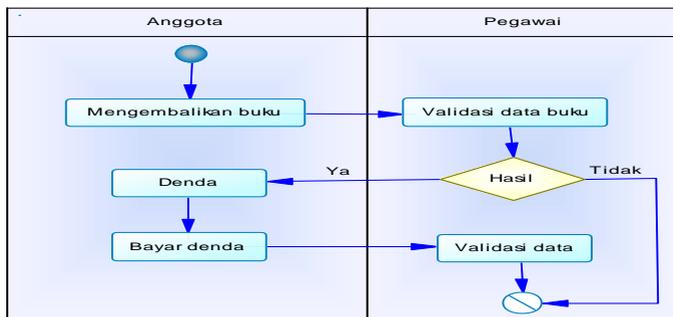
Name	Pengembalian
Aktor	Pegawai, Anggota
Purpose	Melakukan pengembalian buku
Typical Course of Events	
Actor Action	System Response

- | | |
|--|---|
| 1. Anggota membawa buku yang telah dipinjam kepada pegawai | 6. sistem menampilkan form pengembalian |
| 2. Anggota menyerahkan kartu anggota | 7. Sistem menerima input data pengembalian |
| 3. Pegawai memeriksa data-data anggota yang meminjam | 8. Sistem memvalidasi data meliputi data buku, masa waktu pengembalian. |
| 4. Pegawai mengecek buku yang dipinjam | |
| 5. Pegawai mengupdate data anggota | |

Alternative Course

9. jika melewati masa pengembalian maka anggota wajib membayar denda

Diagram aktifitas pemngembalian buku dapat dilihat pada Gambar 7.27



Gambar 7.27 Diagram aktifitas pemngembalian buku

4) Proses pembayaran denda

Jika anggota perpustakaan telat mengembalikan buku yang dipinjam maka anggota mendapat denda. prosesnya yaitu anggota menunjukkan kartu anggota kemudian pustakawan memvalidasi atau mengecek data, mengecek buku yang dipinjam anggota. Jika anggota meminjam buku sesuai waktu peminjaman maka proses selesai jika tidak atau telat maka anggota dikenai denda, pustakawan menentukan jumlah denda yang harus dibayar oleh anggota, pustakawan memvalidasi data setelah anggota membayar denda lalu proses selesai.

Kesimpulan

Use case diagram adalah teknik untuk merekam persyaratan fungsional sebuah system, menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Use case diagram menekankan kepada “apa” yang diperbuat oleh sistem, dan bukan “bagaimana”. Sebuah use case merepresentasikan sebuah interaksi antara aktor dengan sistem. Activity diagrams menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, decision yang mungkin terjadi, dan bagaimana mereka berakhir. Activity diagram digunakan untuk menggambarkan langkah-langkah atau aktivitas pada suatu sistem. Pada setiap use case yang ada, maka terdapat paling sedikit satu activity diagram.

Evaluasi

Setelah menyimak materi pemodelan UML, silahkan jawab pertanyaan berikut:

1. Apa fungsi penggambaran diagram *use case* dalam analisis berorientasi objek?
2. Kapan sebaiknya diagram sekuen, diagram kolaborasi, diagram komponen, dan diagram *deployment* digunakan dalam analisis dan desain berorientasi objek? Sebutkan alasannya!
3. Bandingkan studi kasus perancangan menggunakan DFD dan UML, sebutkan kelebihan dan kekurangan masing-masing metode tersebut!
4. Buat perancangan antarmuka sistem informasi manajemen perpustakaan!
5. Implementasikan hasil analisis dan desain studi kasus sistem informasi manajemen perpustakaan!

Daftar Pustaka

- Kendall, Kenneth E. and Kendall, Julie E., 2011, *System Analysis and Design 8th ed*, Prentice Hall, New Jersey.
- Marakas, G.M. 2006. *System Analysis Design: an Active Approach*. New York: Mc.Graw-Hill.
- Sprague, R.H. and McNurlin, B.C., ***Information Systems Management in Practice, 5th edition***, Prentice-Hall, 2002.
- Suendri. 2018. Implementasi Diagram UML pada Perancangan Sistem Informasi Remunerasi Dosen dengan Database Oracle. 3(1).
- Sukamto, Rosa A., & Shalahuddin, M. 2016. Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Object. Bandung: Informatika*
- Sulistyorini, Prastuti. 2009. Pemodelan Visual dengan Menggunakan UML dan Rational Rose. *Jurnal Teknologi Informasi Dinamik*. 14(1).

Bab VIII

Manajemen Proyek Perangkat Lunak

Setelah mempelajari bab **Manajemen Proyek Perangkat Lunak**, maka diharapkan :

3. Mahasiswa mampu menerapkan perencanaan proyek
4. Mahasiswa mampu menerapkan pengujian perangkat lunak

1. Perencanaan Proyek RPL

Proyek adalah urutan kegiatan yang unik, kompleks, dan saling terkait, memiliki satu tujuan, dan tujuan harus terselesaikan dalam waktu tertentu, sesuai anggaran, dan memenuhi spesifikasi. Manajemen/ pengelolaan proyek perangkat lunak bertujuan agar perangkat lunak dibuat sampai ke tangan pelanggan tepat waktu dan sesuai dengan harapan pelanggan. Proyek dilaksanakan dengan tujuan untuk optimasi penggunaan sumber daya guna mencapai tujuan yang telah ditetapkan. Untuk mencapai tujuan tersebut manajemen proyek harus dilaksanakan dengan cara sebagai berikut:

- a. Adanya koordinasi horisontal antar pelaksana yang tidak terlalu birokratis, sehingga pelaksanaan kegiatan dapat secara luwes dan cepat dilakukan antipasi bila terjadi penyimpangan.
- b. Adanya penanggung jawab tunggal, biasanya oleh pimpinan proyek yang berfungsi sebagai pusat informasi, integrator antar komponen yang terlibat dan sekaligus pelaksanaan koordinasi dengan pihak diluar proyek.

Proyek dapat diuraikan menjadi rincian kegiatan yang terstruktur, dimana setiap kegiatan dapat diuraikan menjadi elemen-elemen kegiatan yang mandiri dengan sifat-sifat, yaitu sebagai berikut: (1) Dapat dikelola sebagai suatu paket kerja, (2) Beban biaya dan waktu dapat diukur, (3) Prestasi, biaya dan kualitas dapat diukur,

(4) Dapat diintegrasikan menjadi suatu satuan kegiatan, dan (5) Dapat disusun secara hirarki berjenjang. Macam-macam proyek perangkat lunak yang dapat diperoleh adalah (a). pengadaan perangkat keras, (b) pengembangan perangkat lunak, (c) pemeliharaan perangkat lunak, dan (d) konsultasi IT.

Manajemen proyek perangkat lunak sangat dibutuhkan karena permasalahan yang sering timbul pada proyek perangkat lunak tanpa pengelolaan yang baik adalah pembengkakan biaya proyek dan waktu pengerjaan tidak sesuai rencana. Padahal waktu dan biaya biasanya sudah dianggarkan oleh pelanggan dan developer perangkat lunak harus mampu mengelola agar target perangkat lunak yang akan dibuat dapat tercapai.

Aktivitas manajemen perangkat lunak dijabarkan sebagai berikut:

a. Aktifitas Manajemen

Aktivitas manajemen perangkat lunak meliputi beberapa langkah yang terstruktur, langkah-langkah tersebut adalah sebagai berikut.

- 1) Proposal Writing (Pembuatan Proposal). Pimpinan proyek harus membuat rencana pekerjaan proyek yang akan dilakukan dari persiapan awal hingga selesainya proyek tersebut. Persiapannya meliputi, tujuan dan manfaat dijalankannya proyek, apa saja bentuk kegiatan yang dikerjakan, dan tahapan-tahapan pekerjaan.
- 2) Project Costing (Anggaran Proyek). Budget pengeluaran dan pemasukan proyek yang akan dikerjakan perlu dibuat yang serinci mungkin.
- 3) Project Planning and Scheduling (Penjadwalan dan Perencanaan Proyek). Perencanaan pelaksanaan proyek yang baik harusnya menggunakan jadwal yang tersusun rapi, dan penjadwalan tersebut dikonversi dengan seluruh kegiatan yang

akan dikerjakan dari study kelayakan, perencanaan, sampai implementasi dan maintenance proyek.

- 4) Project Monitoring and Review (Pemonitoran Proyek). Memonitor pelaksanaan proyek perlu dilakukan disetiap tahapan, sehingga kesalahan dan keterlambatan penyelesaian proyek dapat diketahui sedini mungkin.
- 5) Personal selection and evaluation (Evaluasi dan penyeleksi Personal). Sebelum dilaksanakannya sebuah proyek, maka personal yang terlibat dalam proyek, harus diseleksi sesuai dengan keterampilan dan pengalaman yang dimilikinya.
- 6) Report Writing and Presentation (Presentasi dan Laporan). Presentasi proposal proyek perlu dilakukan dengan menunjukkan prototype yang ada, sehingga pihak manajemen yakin akan proyek tersebut

b. Perencanaan Struktur Organisasi

Pembentukan tim diperlukan untuk melakukan pelaksanaan sebuah proyek yang akan melaksanakan pekerjaan proyek pada setiap tahapan, anggota tim ini dapat juga dirotasi bila ada anggota tim yang merasakan kesulitan pada tahapan pengerjaannya. Tugas-tugas yang terbagi dari penyusunan struktur organisasi, dapat dijabarkan sebagai berikut:

- a. *Project Manager* mempunyai tanggung jawab dan tugas yang bermacam-macam, tidak hanya terfokus pada hal-hal yg teknis sifatnya. Bagaimana layaknya seorang project manager harus mempunyai kemampuan membuat tim proyek agar tetap solid, mampu memonitor dan mengontrol budget serta mempunyai kemampuan analisis resiko yang baik.
- b. *System Analys* bertugas untuk mengembangkan definisi sistem dan planning project.

- c. *Designer* bertugas merancang produk yang sesuai dengan project yang sedang dikerjakan.
- d. *Programmer* bertugas untuk membuat program-program yang sesuai dengan project timnya, yang nantinya program ini akan berpengaruh terhadap pengerjaan project tersebut.
- e. *Tester* atau Penguji bertugas untuk melakukan uji per unit sistem apakah program yang dijalankan sesuai dengan apa yang diharapkan.

Tujuan perencanaan yaitu untuk menyediakan kerangka kerja yang memungkinkan manajer membuat estimasi yang dapat dipertanggungjawabkan mengenai sumber daya, biaya dan jadwal. Estimasi dibuat dengan sebuah kerangka waktu terbatas pada awal sebuah proyek perangkat lunak dan secara teratur diperbaharui secara teratur selagi proyek sedang berjalan.

c. Penjadwalan Proyek

Hal-hal yang perlu dilakukan pada penjadwalan proyek adalah : (a). Membagi kerja proyek menjadi lebih kecil dan memperkirakan waktu, sumber daya, dan personel untuk melakukan bagian kerja, (b) mengatur urutan pembagian kerja, (c) meminimalisir kebergantungan setiap bagian kerja agar tidak terjadi banyak waktu kosong (delay) karena saling menunggu bagian kerja lain selesai terlebih dahulu, (d) penjadwalan yang baik bergantung pada intuisi dan pengalaman pengelola proyek.

Permasalahan yang sering timbul pada saat penjadwalan proyek perangkat lunak, diantaranya : (a) memperkirakan waktu, sumber daya, dan biaya bukanlah hal yang mudah, (b) pembagian kerja tidak proporsional pada personel, (c) penambahan orang ke dalam proyek di tengah atau di belakang jalannya proyek dapat membuat proyek tidak tepat waktu karena memerlukan banyak komunikasi untuk

membuat orang yang baru masuk mengerti permasalahan yang terjadi, dan (d) masalah tidak terduga yang terjadi.

d. Manajemen Resiko

Manajemen resiko focus pada mengidentifikasi resiko dan membuat perencanaan yang dapat meminimalisir resiko yang mungkin terjadi pada proyek perangkat lunak. Resiko yang biasanya muncul pada proyek perangkat lunak adalah:

- a. Resiko proyek, yang mungkin terjadi adalah pergantian orang di dalam proyek, perubahan pengelolaan, dan tidak tersedianya perangkat keras.
- b. Resiko produk, yang mungkin terjadi adalah perubahan spesifikasi/kebutuhan produk, keterlambatan waktu pengumpulan kebutuhan, besarnya sistem tidak dapat diperkirakan, dan perangkat pendukung proyek tidak ada.
- c. Resiko bisnis, yang mungkin terjadi adalah perubahan teknologi dan persaingan produk.

Aktifitas yang dilakukan pada manajemen resiko adalah sebagai berikut:

- a. Identifikasi resiko, mengidentifikasi resiko proyek, resiko produk, dan resiko bisnis
 - b. Analisis resiko, memperkirakan konsekuensi dari resiko yang mungkin terjadi
 - c. Perencanaan resiko, membuat perencanaan untuk meminimalisir resiko
 - d. Pengawasan resiko, membuat mekanisme pengawasan resiko sepanjang proyek terjadi.
- #### **e. Spektrum Manajemen**

Manajemen perangkat lunak berfokus dengan tiga unsur yaitu manusia, masalah, dan proses.

1) Manusia

Faktor manusia sangat penting sehingga Software Engineering Institute telah mengembangkan sebuah model kematangan kemampuan manajemen manusia untuk mempertinggi kesiapan kesiapan organisasi perangkat lunak untuk mengerjakan aplikasi yang semakin kompleks dengan membantu menarik, menumbuhkan memotivasi, menyebarkan dan memelihara dan memelihara bakat yang dibutuhkan mengembangkan kemampuan perkembangan perangkat lunak mereka. Model kematangan manajemen manusia membatasi area praktis berikut kunci bagi masyarakat perangkat lunak: rekrutmen, seleksi, manajemen untuk kerja, pelatihan, kompensasi, perkembangan karir, disain, kerja dan organisasi, dan perkembangan tim atau kultur.

2) Masalah

Seorang manajer proyek perangkat lunak dihadapkan pada sebuah dilema pada awal proyek rekayasa perangkat lunak. Diperlukan perkiraan kuantitatif dan rencana organisasi, tetapi informasi yang solid tidak dapat diperoleh diperoleh. Analisis yang mendetail tentang kebutuhan perangkat lunak memberikan informasi memadai untuk suatu perhitungan, tetapi analisis sering memerlukan waktu berminggu-minggu atau bahkan berbulan-bulan. Lebih buruk lagi kebutuhan terkadang berubah-ubah, berubah secara reguler pada saat proyek berjalan. Seorang manajer harus bisa mengamati masalah pada awal dimulainya sebuah proyek. Pada skala minimum, ruang lingkup masalah harus dibangun dan ditentukan.

3) Proses

Proses perangkat lunak memberikan suatu kerangka kerja dimana rencana komprehensif bagi pengembangan perangkat lunak dapat dibangun. Didalam suatu industri dikenal berbagai macam proses, demikian juga halnya dengan industri perangkat lunak. Perbedaan proses yang digunakan akan menguraikan aktifitas-aktifitas proses dalam cara-cara yang berlainan.

2. Pengujian Perangkat Lunak

Pengujian sistem adalah pengujian program perangkat lunak yang lengkap dan terintegrasi. Perangkat lunak atau yang sering dikenal dengan sebutan software hanyalah satuan elemen dari sistem berbasis komputer yang lebih besar. Biasanya, perangkat lunak dihubungkan dengan perangkat lunak dan perangkat keras lainnya.

Pengujian perangkat lunak adalah proses atau rangkaian proses yang dirancang untuk memastikan bahwa program computer menjalankan apa yang seharusnya dilakukan dan sebaliknya, memastikan program agar tidak melakukan hal yang tidak diharapkan. Sebuah perangkat lunak seharusnya dapat diprediksi dan konsisten. Tugas utama dari seorang penguji adalah untuk menemukan *bug* atau *error* sebanyak mungkin serta mengetahui bagaimana error atau bug itu dihasilkan. Sehingga dapat dikatakan bahwa pengujian adalah proses mengeksekusi program dengan tujuan untuk menemukan *error*. Pengujian merupakan proses penting dalam siklus pengembangan software untuk memastikan kualitas dari sebuah *software*.

Proses pengujian terdiri dari beberapa aktifitas yang biasa disebut sebagai siklus hidup pengujian. Pada beberapa praktek, masing-masing aktifitas dapat dilakukan secara formal atau informal. Gambar di bawah ini menunjukkan aktifitas pengujian berdasarkan Fewster. Sebuah perangkat lunak perlu dijaga kualitasnya bahwa

kualitas bergantung kepada kepuasan pelanggan (*customer*). Kualitas perangkat lunak perlu dijaga keperluan sebagai berikut:

- a. Agar dapat “*survive*” bertahan hidup di dunia bisnis perangkat lunak
- b. Dapat bersaing dengan perangkat lunak yang lain
- c. Penting untuk pemasaran global (*global marketing*)
- d. Mengefektifkan biaya agar tidak banyak membuang perangkat lunak karena kegagalan pemasaran atau kegagalan produksi
- e. Mempertahankan pelanggan (*customer*) dan mengingatkan keuntungan

Pengujian perangkat lunak adalah sebuah elemen topic yang sering dikaitkan dengan verifikasi dan validasi. Verifikasi mengacu pada sekumpulan aktifitas yang menjamin bahwa perangkat lunak mengimplementasikan dengan benar sebuah fungsi yang spesifik. Validasi mengacu pada sekumpulan aktifitas yang berbeda dan menjamin bahwa perangkat lunak yang dibangun dapat ditelusuri sesuai dengan kebutuhan pelanggan. Tahapan pengujian secara keseluruhan adalah sebagai berikut:



Pengujian perangkat lunak dapat dibedakan menjadi dua yaitu Black Box Testing dan White Box Testing.

a. **Black Box Testing**

Black Box Testing atau yang sering dikenal dengan sebutan pengujian spesifikasi fungsional merupakan metode pengujian untuk mengetahui apakah fungsi-fungsi masukan dan keluaran dari perangkat lunak sesuai dengan spesifikasi yang dibutuhkan. Dalam pengujian ini, tester menyadari apa yang harus dilakukan oleh program tetapi tidak memiliki pengetahuan tentang bagaimana

melakukannya. Misalnya menguji proses login maka kasus uji yang dibuat adalah :

- a. Jika user memasukkan nama pemakai (username) dan kata sandi (password) yang benar.
- b. Jika user memasukkan nama pemakai dan kata sandi yang salah, misalnya nama pemakai benar tetapi kata sandi salah atau sebaliknya, atau keduanya salah.



Gambar 8.1 Ilustrasi Black box testing

Kelebihan *Black Box Testing* yaitu:

- a. Efisien untuk segmen kode besar
- b. Akses kode tidak diperlukan
- c. Pemisahan antara perspektif pengguna dan pengembang

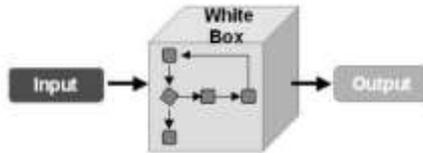
Kelemahan *Black Box Testing* yaitu:

- a. Cakupan terbatas karena hanya sebagian kecil dari skenario pengujian yang dilakukan
- b. Pengujian tidak efisien karena keberuntungan tester dari pengetahuan tentang perangkat lunak internal

b. *White Box Testing*

White Box Testing merupakan metode pengujian perangkat lunak di mana struktur internal diketahui untuk menguji siapa yang akan menguji perangkat lunak. Pengujian ini membutuhkan pengetahuan internal tentang kemampuan sistem dan pemrograman. Contoh dari pengujian white box adalah ketika

menguji alur dengan menelusuri pengulangan pada logika pemrograman.



Gambar 8.2 Ilustrasi *White Box testing*

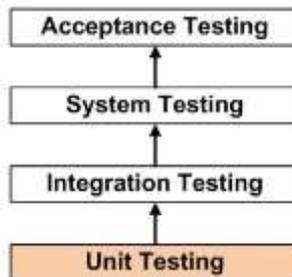
Kelebihan *White Box Testing* yaitu:

- Efisien dalam menemukan kesalahan dan masalah
- Diperlukan pengetahuan tentang internal perangkat lunak yang sedang diuji bermanfaat untuk pengujian menyeluruh
- Memungkinkan menemukan kesalahan tersembunyi
- Membantu mengoptimalkan kode

Kelemahan *White Box Testing* yaitu:

- Membutuhkan pengetahuan tingkat tinggi dari perangkat lunak internal yang sedang diuji
- Membutuhkan akses kode

Pengujian perangkat lunak memiliki urutan-urutan mengenai beberapa hal yang perlu dilakukan. Berikut adalah kategori pengujian perangkat lunak yang disusun secara kronologis:



- a. *Unit Testing*: Pengujian dilakukan pada setiap modul atau blok kode selama pengembangan. Pengujian ini biasanya dilakukan oleh developer yang menulis kode.
- b. *Integration Testing*: Pengujian yang dilakukan Sebelum, selama, dan setelah integrasi modul baru ke dalam paket perangkat lunak utama. Pengujian ini melibatkan pengujian setiap modul kode dari masing-masing individu. Satu perangkat lunak dapat berisi beberapa modul yang sering dibuat oleh beberapa developer yang berbeda.
- c. *System Testing*: Pengujian yang dilakukan oleh agen pengujian profesional pada produk perangkat lunak yang telah selesai sebelum perangkat lunak tersebut diperkenalkan secara umum.
- d. *Acceptance Testing*: Pengujian beta dari produk yang dilakukan oleh pengguna akhir yang sebenarnya.

Kesimpulan

Manajemen/pengelolaan proyek perangkat lunak bertujuan agar perangkat lunak dibuat sampai ke tangan pelanggan tepat waktu dan sesuai dengan harapan pelanggan. Aktivitas manajemen perangkat lunak meliputi beberapa langkah yang terstruktur, diantaranya pembuatan proposal, anggaran proyek, penjadwalan dan perencanaan proyek, monitor proyek, evaluasi dan penyeleksi personal, presentasi dan laporan. Untuk melakukan pelaksanaan sebuah proyek dibentuk tim yang akan melaksanakan pekerjaan proyek pada setiap tahapannya, anggota tim ini dapat juga dirotasi bila ada anggota tim yang merasakan kesulitan pada tahapan pengerjaannya.

Sebuah perangkat lunak perlu dijaga kualitasnya bahwa kualitas bergantung kepada kepuasan pelanggan (customer). Penting dilakukan pengujian perangkat lunak dalam siklus pengembangan software untuk memastikan kualitas dari sebuah software. Pengujian

perangkat lunak dapat dibedakan menjadi dua yaitu Black Box Testing dan White Box Testing.

Evaluasi

Setelah menyimak materi manajemen perangkat lunak, silahkan jawab pertanyaan berikut:

1. Apa pengertian manajemen proyek perangkat lunak?
2. Hal-hal apa saja yang harus dikelola dalam manajemen perangkat lunak? Jelaskan!
3. Mengapa harus dilakukan pengujian pada perangkat lunak?
4. Hal-hal apa saja yang harus diuji dalam perangkat lunak?
5. Apa perbedaan pengujian *black box* dengan pengujian *white box*? Jelaskan!

Daftar Pustaka

- Browman, Kevin. 2004. *System Analysis: A Beginner's Guide*. Palgrave Macmillan
- Dennis, Alan., Wixom, Barbara H. and Roth, Roberta M., 2012, *System Analysis & Design 5th ed*, John Wiley & Sons, New Jersey.
- Marakas, G.M. 2006. *System Analysis Design: an Active Approach*. New York: Mc.Graw-Hill.
- Mc.,Leod, R. Jr. 2002. *System Development: A Project Management Approach*. New York: Leigh Publishing LLC.
- Naik, K. and Tripathy, P., 2008, *Software Testing and Quality Assurance Theory and Practice*, John Wiley & Sons, New Jersey
- Pressman, R.S. 2012. *Rekayasa Perangkat Lunak: Pendekatan Praktisi*. Yogyakarta: Penerbit Andi.
- Pressman, Roger S., 2001, *Software Engineering A Practitioner's Approach 7th ed*, McGraw-Hill, New York.

- Sommerville, Ian., 2011, *Software Engineering 9th ed*, Pearson Education, Boston.
- Sukamto, Rosa A., & Shalahuddin, M. 2016. *Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Object. Bandung: Informatika*
- Whitten, J.L. & Bentley, L.D. 2004. *System Analysis & Design Methods: Sixth Edition*. New York: Mc.Graw-Hill.

Bab IX

Pemeliharaan Perangkat Lunak

Setelah mempelajari bab **Pemeliharaan Perangkat Lunak**, maka diharapkan :

3. Mahasiswa mampu menjelaskan teknik pemeliharaan perangkat lunak
4. Mahasiswa mampu menerapkan Siklus Hidup Pemeliharaan Sistem (SMLC)

Pemeliharaan perangkat lunak adalah proses umum perubahan/pengembangan perangkat lunak setelah diserahkan ke konsumen. Perubahan dapat berupa perubahan sederhana untuk membetulkan error koding atau perubahan yang lebih ekstensif untuk membetulkan error perancangan/perbaikan signifikan untuk membetulkan error spesifikasi/akomodasi persyaratan baru. Atau dapat didefinisikan sebagai suatu kombinasi dari berbagai tindakan yang dilakukan untuk menjaga suatu sistem dalam, atau memperbaikinya sampai, suatu kondisi yang bisa diterima.

Pada bulan April 1970 didefinisikan sebuah istilah untuk Teknologi Pemeliharaan yang mencakup pengertian yang lebih luas dari pada pengertian Pemeliharaan diatas. Istilah ini adalah Teroteknologi merupakan siklus terakhir dari SDLC yaitu dengan pemeriksaan periodik, audit dan permintaan pengguna akan menjadi source untuk melakukan perawatan system diseluruh masa hidup system. Istilah pemeliharaan perangkat lunak digunakan untuk menjabarkan aktivitas dari analisis sistem (*software engineering*) yang terjadi pada saat hasil produk perangkat lunak sudah dipergunakan oleh pemakai (user).

Biasanya pengembangan produk perangkat lunak memerlukan waktu antara 1 sampai dengan 2 tahun, tetapi pada fase

pemeliharaan perangkat lunak menghabiskan 5 sampai dengan 10 tahun. Aktivitas yang terjadi pada fase pemeliharaan antara lain: (1) Penambahan atau peningkatan atau juga perbaikan untuk produk perangkat lunak; (2) adaptasi produk dengan lingkungan mesin yang baru; (3) Pembetulan permasalahan yang timbul.

“ Pemeliharaan sistem berawal begitu sistem baru menjadi operasional dan berakhir masa hidupnya ” .

Tujuan dari pemeliharaan system adalah :

- a. Untuk memperpanjang usia kegunaan asset dari system tersebut. Hal ini penting dilakukan terutama dinegara berkembang karena kurangnya sumber daya modal untuk penggantian. Dinegara-negara maju kadang-kadang lebih menguntungkan untuk ‘mengganti’ daripada ‘memelihara’.
- b. Untuk menjamin ketersediaan optimum peralatan
- c. Untuk menjamin kesiapan operasional dari seluruh peralatan yang diperlukan dalam keadaan darurat setiap waktu.
- d. Untuk menjamin keselamatan orang yang menggunakan sarana tersebut.

1. Teknik Pemeliharaan Perangkat lunak

Pemeliharaan perangkat lunak dibedakan menjadi *corrective maintenance*, *adaptive maintenance*, *perfective maintenance*, dan *perfentive maintenance*.

a. Corrective Maintenance

Pemeliharaan ini untuk merespon terjadinya kesalahan-kesalahan saat produk dioperasikan baik berupa bug atupun berupa output yang tidak sesuai dengan kebutuhan pengguna.

b. Adaptive Maintenance

Pemeliharaan ini untuk merespon perubahan yang terjadi pada lingkungan yang mempengaruhi perangkat lunak tersebut

(seperti perangkat keras, sistem operasi, prosedur bisnis, kebijakan, dll).

c. *Perfective maintenance*

Pemeliharaan ini untuk merespon permintaan tambahan berupa fungsi-fungsi baru yang muncul setelah pengguna melakukan uji coba perangkat lunak tersebut.

d. *Preventif maintenance*

Pemeliharaan ini dilakukan untuk melakukan reengineering terhadap perangkat lunak agar lebih mudah diperbaiki, memiliki tingkat adaptasi yang tinggi dan mudah mengakomodasi munculnya kebutuhan baru.

Karakteristik perangkat lunak yang mudah dalam pemeliharaan adalah Perangkat lunak dikerjakan per modul, perangkat lunak mempunyai kejelasan, dokumentasi internal yang baik dan jelas, dan dilengkapi dokumen-dokumen pendukung lainnya.

Pemeliharaan juga mempengaruhi dokumen pendukung seperti :

- a. Dokumen spesifikasi kebutuhan perangkat lunak
- b. Dokumen rancangan
- c. Dokumen rencana pengujian
- d. Prinsip pengoperasian
- e. Petunjuk pemakaian

Manfaat pemeliharaan perangkat lunak yaitu : (1) memastikan kesesuaian dengan kebutuhan fungsionalitas teknis software, (2) memastikan kesesuaian kebutuhan pihak manajerial mengenai jadwal dan budget, (3) dapat meningkatkan efisiensi software berikut juga aktifitas pemeliharaannya.

2. Siklus Hidup Pemeliharaan Sistem (SMLC)

Tahapan-tahapan yang dilakukan pada SMLC terdiri dari :

a. Memahami Permintaan Pemeliharaan

- 1) Mentransformasi permintaan pemeliharaan menjadi perubahan
- 2) Menspesifikasi perubahan
- 3) Mengembangkan perubahan
- 4) Menguji perubahan
- 5) Melatih pengguna dan melakukan test penerimaan
- 6) Pengkonversian dan meluncurkan operasi
- 7) Mengupdate Dokumen
- 8) Melakukan pemeriksaan Pasca implementasi

b. *Maintainability* (Kemampuan pemeliharaan sistem)

Prosedur untuk peningkatan maintainability :

- 1) Menerapkan SDLC dan SWDLC
- 2) Menspesifikasi definisi data standar
- 3) Menggunakan bahasa pemrograman standart
- 4) Merancang modul-modul yang terstruktur dengan baik
- 5) Mempekerjakan modul yang dapat digunakan kembali
- 6) Mempersiapkan dokumentasi yang jelas, terbaru dan komprehensif
- 7) Menginstall perangkat lunak, dokumentasi dan soal-soal test di dalam sentral repositor sistem CASE atau CMS (change management system)

Untuk teknik pemeliharaan yang biasa dilakukan pada perangkat lunak, pada dasarnya tidak ada teknik atau metode yang 100 persen pasti. Hal itu karena pemeliharaan biasanya dilakukan sebagai solusi dari masalah yang muncul. Dengan kata lain, pemeliharaan mungkin tidak diperlukan andaikata tidak ada masalah yang muncul. Akan

tetapi, ada beberapa teknik yang biasa dilakukan dalam pemeliharaan perangkat lunak. Berikut beberapa diantaranya.

Kesimpulan

Pemeliharaan perangkat lunak adalah proses umum pengubahan/pengembangan perangkat lunak setelah diserahkan ke konsumen. Aktivitas yang terjadi pada fase pemeliharaan antara lain: (1) Penambahan atau peningkatan atau juga perbaikan untuk produk perangkat lunak; (2) adaptasi produk dengan lingkungan mesin yang baru; (3) Pembetulan permasalahan yang timbul. Jenis pemeliharaan perangkat lunak dibedakan menjadi *corrective maintenance*, *adaptive maintenance*, *perfective maintenance*, dan *preventive maintenance*.

Evaluasi

Setelah menyimak materi pemeliharaan perangkat lunak, silahkan jawab pertanyaan berikut:

1. Bagaimana menurut pendapat anda tentang pemeliharaan perangkat lunak?
2. Jelaskan perbedaan spesifik dari jenis-jenis pemeliharaan perangkat lunak!
3. Apakah yang dimaksud dengan *Maintainability* (Kemampuan pemeliharaan sistem)?
4. Menurut anda apakah pengaruh terbesar dari pemeliharaan perangkat lunak?
5. Apakah fungsi dari *Preventif maintenance*?

Daftar Pustaka

Kendall, J.E. & Kendall, K.E. 2010. Analisis dan Perancangan Sistem. Jakarta: Indeks.

- Kendall, Kenneth E. and Kendall, Julie E., 2011, *System Analysis and Design 8th ed*, Prentice Hall, New Jersey.
- Marakas, G.M. 2006. *System Analysis Design: an Active Approach*. New York: Mc.Graw-Hill.
- Mc.,Leod, R. Jr. 2002. *System Development: A Project Management Approach*. New York: Leigh Publishing LLC.
- Naik, K. and Tripathy, P., 2008, *Software Testing and Quality Assurance Theory and Practice*, John Wiley & Sons, New Jersey

DAFTAR PUSTAKA

- Browman, Kevin. 2004. *System Analysis: A Beginner's Guide*. Palgrave Macmillan
- Dennis, Alan., Wixom, Barbara H. and Roth, Roberta M., 2012, *System Analysis & Design 5th ed*, John Wiley & Sons, New Jersey.
- Elmasri dan Navathe. 2007. *Fundamentals of Database Systems, Fifth Edition*. Boston: Pearson Education, Inc. Addison Wesley
- Fatansyah. 2012. *Basis Data*. Bandung: Informatika
- Indrajani. 2015. *Database Design*. Jakarta: Elex Media Komputindo
- Kadir, Abdul. 2008. *Belajar Database menggunakan MySQL*. Yogyakarta : Andi Offset
- Kendall, J.E. & Kendall, K.E. 2010. *Analisis dan Perancangan Sistem*. Jakarta: Indeks.
- Kendall, Kenneth E. and Kendall, Julie E., 2011, *System Analysis and Design 8th ed*, Prentice Hall, New Jersey.
- Marakas, G.M. 2006. *System Analysis Design: an Active Approach*. New York: Mc.Graw-Hill.
- Mc.,Leod, R. Jr. 2002. *System Development: A Project Management Approach*. New York: Leigh Publishing LLC.
- Naik, K. and Tripathy, P., 2008, *Software Testing and Quality Assurance Theory and Practice*, John Wiley & Sons, New Jersey
- Pressman, R.S. 2012. *Rekayasa Perangkat Lunak: Pendekatan Praktisi*. Yogyakarta: Penerbit Andi.
- Pressman, Roger S., 2001, *Software Engineering A Practitioner's Approach 7th ed*, McGraw-Hill, New York.
- Simarmata, Janner. 2010. *Rekayasa Perangkat Lunak*. Yogyakarta: Andi
- Sommerville, Ian., 2011, *Software Engineering 9th ed*, Pearson Education, Boston.
- Sprague, R.H. and McNurlin, B.C., *Information Systems Management*

in Practice, 5th edition, Prentice-Hall, 2002.

- Suendri. 2018. Implementasi Diagram UML pada Perancangan Sistem Informasi Remunerasi Dosen dengan Database Oracle. 3(1).
- Sukamto, Rosa A., & Shalahuddin, M. 2016. *Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Object*. Bandung: Informatika
- Sulistiyorini, Prastuti. 2009. Pemodelan Visual dengan Menggunakan UML dan Rational Rose. Jurnal Teknologi Informasi Dinamik. 14(1).
- Whitten, J.L. & Bentley, L.D. 2004. *System Analysis & Design Methods: Sixth Edition*. New York: Mc.Graw-Hill.
- er S., 2001, *Software Engineering A Practitioner's Approach 7th ed*, McGraw-Hill, New York.
- Simarmata, Janner. 2010. *Rekayasa Perangkat Lunak*. Yogyakarta: Andi
- Sommerville, Ian., 2011, *Software Engineering 9th ed*, Pearson Education, Boston.
- Suendri. 2018. Implementasi Diagram UML pada Perancangan Sistem Informasi Remunerasi Dosen dengan Database Oracle. 3(1).
- Sukamto, Rosa A., & Shalahuddin, M. 2016. *Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Object*. Bandung: Informatika
- Sulistiyorini, Prastuti. 2009. Pemodelan Visual dengan Menggunakan UML dan Rational Rose. Jurnal Teknologi Informasi Dinamik. 14(1).

BIODATA PENULIS



Fitria Nur Hasanah, M.Pd. dilahirkan di Lamongan, 23 September 1987. Pada tahun 2008, penulis mendapatkan gelar Diploma Manajemen Informatika di Universitas Brawijaya, lulus Sarjana Pendidikan Teknik Informatika di Universitas Negeri Malang tahun 2011, kemudian melanjutkan gelar magister Pendidikan Kejuruan dengan konsentrasi Teknik Informatika di Universitas Negeri Malang lulus tahun

2015. Tahun 2011 penulis mengawali karirnya sebagai Guru di SMK Nasional Malang bidang Rekayasa Perangkat Lunak dan Teknik Komputer Jaringan. Selanjutnya tahun 2016 menjadi Dosen tetap di Prodi Pendidikan Teknologi Informasi Universitas Muhammadiyah Sidoarjo. Tahun 2018 menjabat sebagai Ketua Program Studi Pendidikan Teknologi Informasi sampai dengan sekarang, sekaligus sebagai Sekretaris Asosiasi Pendidikan Tinggi Informatika dan Komputer (APTIKOM) Provinsi Jawa Timur periode tahun 2020-2024. Beberapa penelitian yang pernah dilakukan oleh penulis adalah tentang model pembelajaran dan pengembangan media pembelajaran.



Rahmania Sri Untari, M.Pd lahir di Malang, 19 April 1989. Pendidikan Sarjana diselesaikan di Program Studi Pendidikan Teknik Informatika, Universitas Negeri Malang (UM) pada tahun 2011. Pendidikan S2 di Program Pascasarjana Pendidikan Kejuruan UM selesai pada tahun 2015. Pada tahun 2016, penulis melanjutkan S3 Pendidikan Kejuruan UM sampai sekarang. Pada tahun 2011 penulis memulai karirnya di SMAN 6 Surabaya sebagai guru

Teknik Informatika. Selanjutnya, pada tahun 2015 penulis melanjutkan karirnya untuk menjadi Dosen Tetap di Program Studi Pendidikan Teknologi Informasi (PTI) di Universitas Muhammadiyah Sidoarjo (UMSIDA) sampai sekarang. Minat penelitian penulis adalah pada bidang pembelajaran berbasis proyek, pengembangan media pembelajaran, pendidikan kejuruan, dan bidang pendidikan lainnya.