Lambang Probo Sumirat, dkk

madza media

DASAR - DASAR
REKAYASA
PERANGKAT LUNAK

DASAR-DASAR

Rekayasa Perangkat Lunak

Lambang Probo Sumirat
Dwi Cahyono
Yudi Kristyawan
Slamet Kacung

madza media

DASAR-DASAR

Rekayasa Perangkat Lunak

Edisi Pertama

ISBN 978-623-130-241-0
15,5 x 23 cm
126 h.
cetakan ke-1, 2023

Penulis
Lambang Probo Sumirat, dkk

Penerbit
Madza Media
Anggota IKAPI: No.273/JTI/2021
Kantor 1: Jl. Pahlawan, Simbatan, Kanor, Bojonegoro
Kantor 2: Jl. Bantaran Indah Blok H Dalam 4a Kota Malang
redaksi@madzamedia.co.id
www.madzamedia.co.id

KATA PENGANTAR

Segala puji bagi Allah SWT yang senantiasa memberikan kenikmatan serta berkah, baik nikmat sehat maupun nikmat bahagia kepada penulis. Kini tiba saatnya penulis mengucap alhamdulillah atas selesainya buku "Dasar-dasar Rekayasa Perangkat Lunak". Buku ini membahas dasar-dasar pengetahuan tentang rekayasa perangkat lunak atau disebut juga dengan rekayasa *software* atau *Software Engineering*.

Dalam penyusunan buku ini, penulis melakukan beberapa studi terkait pelaksanaan pembangunan atau pembuatan perangkat lunak di beberapa instansi, baik instansi pemerintahan, akademik maupun instansi swasta. Selain itu penulis juga beberapa kali melakukan perbandingan antar perangkat lunak yang terdapat pada masing-masing instansi. Dari hasil tersebut dapat disimpulkan bahwa setiap perangkat lunak yang akan dibangun atau dibuat harus memenuhi kebutuhan dan dapat menyelesaikan permasalahan dari penggunanya sehingga tercapai tujuan dari perangkat lunak.

Penulis mempercayai, kesempurnaan hanyalah milik Allah SWT semata. Kekurangan yang ada pada buku ini, harap untuk dimaklumi. Penulis berusaha untuk memberikan yang terbaik guna membuat pembaca nyaman ketika membaca buku ini. Oleh karena itu, penulis memohon maaf atas segala kekurangan dan kesalahan baik disengaja maupun tidak disengaja. Semoga pembaca bisa mendapatkan manfaat dari adanya buku ini dan terima kasih, selamat membaca.

Mei 2023

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

1

PENGENALAN REKAYASA PERANGKAT LUNAK

A. Definisi Perangkat Lunak

- Satu set item atau objek yang membentuk "konfigurasi" yang mencakup Program, Dokumen, Data
- Program komputer, prosedur, dan dokumentasi terkait serta data yang berkaitan dengan pengoperasian suatu sistem komputer

(IEEE Standard Glossary of Software Engineering Terminology, 1990)

1. Definisi Klasik (1969)

"The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines."

Penerapan prinsip *engineering* untuk memperoleh *software* yang ekonomis, reliabel dan bekerja efisien pada komputer.

2. Definisi IEEE (1993)

"*Software Engineering: (1) The application of a systematic, disciplines, quantifiable approach to the development, operation, and maintenance of software ; that is the application of engineering to software . (2) The study of approaches as in (1).*"

Rekayasa Perangkat Lunak : (1) Penerapan secara sistematis, disiplin, pendekatan terstruktur pada pengembangan, pengoperasian dan pemeliharaan *software* . (2) Pendekatan studi seperti pada (1).

Pengertian Rekayasa Perangkat Lunak sendiri adalah sebagai berikut:

Suatu disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal yaitu analisa kebutuhan pengguna, menentukan spesifikasi dari kebutuhan pengguna, desain, pengkodean, pengujian sampai pemeliharaan sistem setelah digunakan.

Jelaslah bahwa Rekayasa Perangkat Lunak 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 Karakteristik Perangkat Lunak

Untuk dapat mengerti secara jelas mengenai perangkat lunak, kita perlu mengetahui beberapa karakteristik yang terdapat pada perangkat lunak tersebut. Karakteristik tersebut adalah:

- *Pembuatan perangkat lunak berdasarkan* ***logika***. Ini menyebabkan pembuatan perangkat lunak berbeda antara satu programmer dengan programmer lainnya.
- *Perangkat lunak* ***dikembangkan*** *bukan dibuat oleh pabrik-pabrik tertentu*. Hal ini berarti bahwa perangkat lunak tidak dibuat secara massal, karena dalam pembuatan perangkat lunak memerlukan perancangan yang baik.
- *Perangkat lunak tidak akan pernah usang* karena selalu **diperbaharui**

Contoh Perangkat Lunak (Diskusi)

1. Sistem Akademik Mahasiswa (Simawa)
2. Aplikasi e-commerce
3. Aplikasi Kalender
4. Aplikasi Pendeteksi Plagiasi

Klasifikasi Perangkat Lunak

- Perangkat Lunak
  a. Sistem Operasi

     Program yang ditulis untuk mengendalikan dan meng-koordinasikan kegiatan dari sistem komputer

     Contoh: Windows, Linux, Macintos, Novel Natware, Android

  b. Bahasa Pemrograman

     Program yang digunakan untuk menerjemahkan instruksi-instruksi yang ditulis dalam bahasa pemrograman ke dalam bahasa mesin agar dapat dimengerti oleh komputer.

     Contoh: C Family, Java Family, Visual Studio.Net

  c. Utility Contoh: NORTON Utility, Compiler, DBMS

  d. Perangkat lunak Aplikasi: Program komputer yang sering dijumpai dan digunakan oleh pengguna komputer

     Contoh: Ms Office, Aplikasi Bisnis, Aplikasi Keuangan, Chatting, Winamp, Games dan lain-lain.

B. Jenis-jenis Perangkat Lunak

- ***System Software***

  Kumpulan dari beberapa program yang dibuat untuk memberikan servis terhadap program lainnya pada setiap level.

  Contoh: *compiler*, sistem operasi

- ***Real-time Software***

  Program yang dapat memonitor/menganalisa/mengontrol kejadian nyata yang terjadi di dunia ini

- ***Business Software***

  Program yang dapat mengakses, menganalisa dan memproses informasi bisnis.

- ***Engineering and Scientific Software***

  Contoh: Sistem *simulation, computer-aided design*.

- ***Embedded Software***

  Perangkat lunak terletak pada *read only memory* dan digunakan untuk mengontrol produk dan sistem yang akan dikirimkan untuk konsumen dan *industrial markets*.
- ***Artificial Intelligence (AI) Software***

  Program yang digunakan untuk teknik AI dan metodenya digunakan untuk memecahkan masalah yang kompleks.

  Contoh: *expert* sistem, pengenalan pola, *games*.
- ***Internet Software***

  Program yang men-*support* pengaksesan internet.

  Contoh: *search engine, browser, e-commerce*
- ***Software Tools and CASE environment***

  Tools dan program yang dapat membantu pembuatan aplikasi *software* dan sistem.

  contoh: *test tools* dan *version control tools*.

C. Mitos Perangkat Lunak

- Masih diyakini oleh banyak manajer dan praktisi
- Membahayakan karena mereka memiliki unsur-unsur kebenaran
- Setiap praktisi dan manajer harus memahami realitas bisnis perangkat lunak

Mitos dari Sudut Pandang *Client*

Tabel 1. Mitos dari Sudut Pandang *Client*

MITOS	REALITA
Sebuah pernyataan umum dari tujuan cukup untuk mendapatkan selanjutnya. Isian secara rinci dilakukan kemudian.	Definisi yang tidak rinci di awal *requirement* atau kebutuhan adalah penyebab utama kegagalan dan keterlambatan perangkat lunak.
Kebutuhan Proyek terus berubah, tetapi perubahan	Biaya dari perubahan untuk perangkat lunak untuk

dapat dengan mudah diakomodasi karena *software* fleksibel.	memperbaiki suatu kesalahan meningkat secara dramatis pada tahap selanjutnya dari kehidupan *software* .

Dari Sudut Pandang *Developer*

Tabel 2. Mitos dari Sudut Pandang *Developer*

MITOS	REALITA
Setelah program ditulis dan bekerja, pekerjaan pengembang telah selesai.	50%-70% dari usaha pengeluaran program terjadi setelah dikirim ke pelanggan.
Sampai program berjalan, tidak ada cara untuk menilai kualitas.	*Review Software* dapat lebih efektif dalam menemukan kesalahan daripada pengujian untuk kelas-kelas tertentu yang memiliki kesalahan.
Hanya diserahkan untuk proyek yang sukses adalah program yang jalan.	Suatu konfigurasi perangkat lunak meliputi dokumentasi, regenerasi *file*, masukan (*input*) data uji, dan hasil data uji.

Dari Sudut Pandang Manajemen

Tabel 3. Mitos dari Sudut Pandang Manajemen

MITOS	REALITA
Ada Buku standar sehingga perangkat lunak yang dikembangkan akan memuaskan.	Buku mungkin ada, tetapi mereka biasanya tidak *up to date* dan tidak digunakan.
Komputer dan kakas (*tools*) perangkat lunak yang tersedia sudah cukup..	CASE *tools* diperlukan tetapi biasanya tidak diperoleh atau tidak digunakan.
Kita selalu dapat menambahkan jumlah	"Menambahkan orang untuk proyek yang terlambat akan membuat penyelesaian

programmer jika proyek terlambat penyelesaiannya	perangkat lunak lebih terlambat "-. Brooks

Mengapa Rekayasa Perangkat Lunak?

- Untuk membuat perangkat lunak dengan benar (proses) dan mendapatkan perangkat lunak yang benar (produk)
- Adanya Kompleksitas Perangkat Lunak
  a. *Domain problem: Business Rule*
  b. *Ukuran Data: Digital and Non Digital*
  c. *Solution: Algorithm*
  d. *Tempat atau Situs*
- Perangkat Lunak harus benar
- Kebenaran perangkat lunak harus bisa dipelihara

Bagaimana Seharusnya Rekayasa Perangkat Lunak diterapkan?

- Cakupan Rekayasa Perangkat Lunak (2 hal)
  a. Produk = *Software*
     1) *Programs*
     2) *Documents*
     3) *Data*
  b. Proses bagaimana membangun perangkat lunak
     1) *Management process*
     2) *Technical process*

Produk Rekayasa Perangkat Lunak

- Produk diperoleh melalui tahapan Pengembangan = *Software Development Life Cycle* (SDLC)
- Contoh siklus hidup (SDLC):
  a. *Waterfall model*
  b. *V model*
  c. *Spiral model*
  d. *Fountain model*
  e. *Prototyping*

Proses Rekayasa Perangkat Lunak

- *Management Process* meliputi:
  a. *Project management*
  b. *Configuration management*
  c. *Quality Assurance management*
- *Technical Process,* digambarkan sebagai metode yang akan diterapkan dalam tahap tertentu dari SDLC
  a. *Analysis methods*
  b. *Design methods*
  c. *Programming methods*
  d. *Testing methods*
- Metode teknis ini yang memunculkan paradigma seperti berorientasi terstruktur, objek, aspek, dll.

Kapan Rekayasa Perangkat Lunak diterapkan?

- *Pre-project*
- *Project Initiation*
- *Project Realisation*
- *Software Delivery & Maintenance*

Siapa yang Terlibat Rekayasa Perangkat Lunak?

- *Manager*
  a. *Project Manager*
  b. *Configuration Manager*
  c. *Quality Assurance Manager*
- *Software Developer:*
  a. *Analyst*
  b. *Designer*
  c. *Programmer*
- *Support*
  a. *Administration*
  b. *Technical Support for Customer (help desk, customer care)*
  c. *Welfare* (Kesejahteraan)

CASE (Computer Aided Software Engineering)

- Perangkat lunak sistem yang dimaksudkan untuk memberikan
  dukungan otomatis untuk kegiatan proses perangkat lunak.
- CASE sistem sering digunakan untuk mendukung metode.
  Upper-CASE
- *Tools* (Alat/Kakas) untuk mendukung proses kegiatan awal penggalian kebutuhan, analisis dan desain
  Lower-CASE
- Alat untuk mendukung kegiatan berikutnya seperti pemrograman, *debugging* dan pengujian.

Apa Atribut Perangkat Lunak yang Baik?

Perangkat lunak ini 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.

Apa saja Tantangan Utama yang dihadapi Rekayasa Perangkat Lunak?

- *Heterogeneity* (keberagaman)

  Mengembangkan teknik untuk membangun perangkat lunak yang dapat mengatasi perbedaan platform dan lingkungan eksekusi.

- *Delivery* (pengiriman)

  Mengembangkan teknik yang mengakibatkan pengiriman perangkat lunak lebih cepat.

- *Trust* (kepercayaan)

  Mengembangkan teknik yang menunjukkan bahwa perangkat lunak dapat dipercaya oleh para penggunanya.

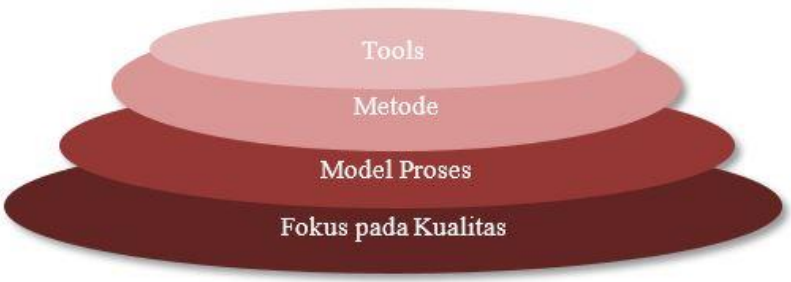
2

MODEL-MODEL PROSES DALAM REKAYASA PERANGKAT LUNAK

A. Definisi Rekayasa Perangkat Lunak

- Pembangunan dan Penggunaan prinsip-prinsip rekayasa dalam rangka mendapatkan perangkat lunak yang ekonomis yang handal dan bekerja efisien pada komputer yang nyata (*Fritz Bauer*)
- IEEE
  a. Aplikasi pendekatan sistematik, disiplin, terkuantifikasi pada pengembangan, operasi, perawatan perangkat lunak, yaitu aplikasi rekayasa pada perangkat lunak
  b. Studi pendekatan-pendekatan di atas

Teknologi Rekayasa Perangkat Lunak Berlapis



Gambar 1. Teknologi Rekayasa Perangkat Lunak Berlapis

Pandangan Umum Rekayasa Perangkat Lunak

- Rekayasa: analisis, desain, konstruksi, verifikasi, dan manajemen entitas teknis (dan sosial)
  a. Problem apa yang harus diselesaikan?
  b. Karakteristik entitas apa yang digunakan untuk

menyelesaikan masalah?

c. Bagaimana entitas (dan solusinya) direalisasikan?
d. Bagaimana entitas di konstruksi?
e. Pendekatan apa yang digunakan untuk menemukan kesalahan yang dibuat pada desain dan konstruksi entitas?
f. Bagaimana entitas didukung dalam jangka panjang, di mana koreksi, adaptasi, dan peningkatan selalu diminta pengguna pada entitas

Tiga Fase Umum Rekayasa Perangkat Lunak

- Fase definisi, fokus pada pertanyaan "apa"
- Fase pengembangan, fokus pada pertanyaan "bagaimana"
- Fase dukungan, fokus pada "perubahan":
  a. Koreksi
  b. Adaptasi
  c. Peningkatan
  d. Pencegahan

Definisi Proses

- Proses: kumpulan aktivitas-aktivitas (*activities*), tindakan-tindakan (*actions*) dan tugas-tugas (*tasks*) yang dilakukan ketika suatu produk (Perangkat Lunak) dibuat
- Tiap aktivitas terdiri dari beberapa tindakan, tiap tindakan terdiri dari beberapa tugas

B. Definisi Proses Perangkat Lunak

- *Software process* mendefinisikan pendekatan yang diambil untuk membuat perangkat lunak yang akan direkayasa.
- Rekayasa perangkat lunak juga mencakup teknologi yang mengisi proses – *Technical methods* dan *Automated tools*

Proses Perangkat Lunak

- *What it is?*

  langkah-langkah yang terprediksi sebagai *road map* agar penyelesaian produk perangkat lunak bisa tepat waktu dengan hasil yang berkualitas tinggi.
- *Who does it?*

  Software engineer & pihak manajemen
- *Why is it important?*

  Karena ia menyediakan stabilitas, kontrol, dan pengaturan pada setiap aktivitas yang dilakukan
- *What are the steps?*

  Tergantung pada produk yang dibuat
- *What is the work product?*

  Program, dokumen dan data yang dihasilkan
- *How do i ensure that i've done it right?*

  Mekanisme penilaian dan perbaikan untuk menilai kematangan/*maturity*-nya

Model Proses Umum

Setiap kegiatan, tindakan dan tugas berada dalam kerangka kerja (*framework*) atau model yang mendefinisikan hubungan antar mereka

Kerangka Kerja Proses

- Aktivitas Kerangka Kerja
  a. Komunikasi (*Communication*)
  b. Perencanaan (*Planning*)
  c. Pemodelan (*Modelling*)
     1) Analisis Kebutuhan
     2) Desain
  d. Konstruksi (*Construction*)
     1) Menyusun kode
     2) Pengujian
  e. Instalasi (*Deployment*)

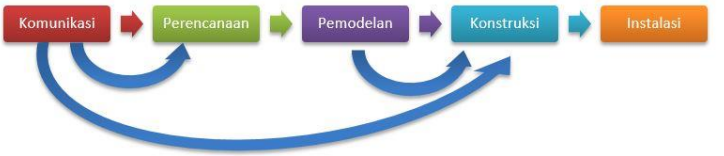
- Aktivitas Payung
  a. Manajemen Proyek Perangkat Lunak
  b. *Review* Teknik Formal
  c. Jaminan Mutu Perangkat Lunak
  d. Manajemen Konfigurasi Perangkat Lunak
  e. Persiapan dan Produksi Produk Pekerjaan
  f. Manajemen Penggunaan Kembali
  g. Pengukuran
  h. Manajemen Risiko

Aliran Proses (*Process Flow*)

- Salah satu aspek penting dalam proses Perangkat Lunak adalah aliran proses (*process flow)*
- Aliran proses menggambarkan bagaimana *framework* yang terdiri dari aktivitas (*activity*), tindakan (*action*) dan tugas (*task*) terjadi dalam setiap aktivitas kerangka kerja yang diselenggarakan sehubungan dengan urutan dan waktu
- Ada 4 jenis *process flow*:
  a. ***Linier***: Melaksanakan masing-masing dari lima *framework activity* secara urut dimulai dengan *communication* dan mencapai puncaknya dengan *deployment*.

Gambar 2. Alur Proses *Linier*

  b. ***Iterative***: Mengulang satu atau lebih kegiatan sebelum melanjutkan ke yang berikutnya



Gambar 3. Alur Proses *Iterative*

  c. ***Evolutionary***: Menjalankan aktivitas dengan cara melingkar. Setiap sirkuit yang melalui lima kegiatan mengarah ke versi yang lebih lengkap dari perangkat

lunak

Gambar 4. Alur Proses *Evolutionare*

d. ***Parallel***: menjalankan satu atau lebih kegiatan secara paralel dengan kegiatan lain. (Contoh *modelling* untuk satu aspek pada *software* akan dijalankan secara *parallel* dengan *construction* aspek lain pada *software*).

Gambar 5. Alur Proses *Parallel*

Mendefinisikan Kerangka Aktivitas (*Framework Activity*)

- Meskipun telah dijelaskan 5 kerangka aktivitas, tim perangkat lunak akan membutuhkan informasi lebih sebelum benar-benar bisa menjalankan salah satu dari aktivitas ini sebagai bagian dari proses perangkat lunak
- Untuk Perangkat Lunak kecil yang diminta oleh satu orang, maka kumpulan tugas (*task set*) adalah
  a. Menghubungi *stakeholder*
  b. Mendiskusikan kebutuhan (*requirement*) dan membuat catatan
  c. Mengelola catatan menjadi ringkasan yang ditulis dalam pernyataan kebutuhan.
  d. Mengirim hasilnya ke *stakeholder* untuk di-*review* dan disetujui

- Jika proyek ini jauh lebih kompleks dengan berbagai pihak, di mana masing-masing pihak memiliki sekumpulan kebutuhan yang berbeda (yang kadang bertentangan).

Sehingga untuk *requirement s*, maka aktivitas komunikasi mungkin memiliki enam tindakan yang berbeda (dijelaskan dalam bab 5) yaitu: *inception, elicitation, elaboration, negosiasi, specification, dan validation*. Masing-masing tindakan rekayasa Perangkat Lunak akan memiliki banyak tugas kerja dan sejumlah produk kerja yang berbeda

C. Kumpulan Tugas (*Task Set*)

- Adalah tugas pekerjaan yang sebenarnya harus dilakukan untuk mencapai tujuan dari tindakan rekayasa perangkat lunak
- Sebagai contoh, *elicitation* yang secara umum disebut "***requirement* gathering**" merupakan langkah penting dalam Rekayasa Perangkat Lunak. Tujuan dari kegiatan ini adalah untuk memahami apa yang diinginkan berbagai pemangku kepentingan (*stake holder*) dari perangkat lunak yang akan dibangun.

Contoh Task set (kumpulan tugas) untuk tindakan *requirement gathering* pada aktivitas komunikasi

Tabel 4. Contoh *Task Set*

Perangkat Lunak Kecil	Perangkat Lunak Besar
1. Buat daftar *stakeholder* untuk proyek 2. Undang seluruh *stakeholder* pada pertemuan informal 3. Mintalah setiap stokeholder membuat daftar	1. Buat daftar *stakeholder* untuk proyek 2. *Interview* setiap *stakeholder* secara terpisah untuk menetapkan secara keseluruhan yang diinginkan dan dibutuhkan. 3. Buat daftar pendahuluan yang berisi fungsi-fungsi dan fitur-fitur sesuai masukan dari *stakeholder*

Perangkat Lunak Kecil	Perangkat Lunak Besar
fitur dan fungsi yang diperlukan. 4. Diskusikan *requirement* dan buat daftar akhir 5. Membuat prioritas kebutuhan 6. Catat area yang tidak pasti	4. Lakukan penjadwalan untuk serangkaian pertemuan khusus 5. Melakukan pertemuan 6. Buatlah skenario pengguna sebagai bagian dari setiap pertemuan 7. Perhalus skenario pengguna berdasarkan *feedback* dari pengguna 8. Buatlah kebutuhan (*requirement*) *stakeholder* yang telah direvisi 9. Gunakan teknik *quality function deploy*ment untuk memprioritaskan kebutuhan 10. Kelompokkan *requirement* sehingga bisa dikirim secara *increment* 11. Catat batasan yang akan diletakkan pada sistem 12. Diskusikan metode untuk validasi sistem

Pola-pola Proses

- Pola-pola proses menentukan sekelompok aktivitas, aksi, tugas-tugas pekerjaan, produk-produk pekerjaan dan/atau perilaku yang berkaitan
- Sebuah *template* digunakan untuk menentukan *pattern*/pola
- Contoh-contoh umum:
  a. Komunikasi pelanggan (sebuah aktivitas proses)
  b. Analisis (sebuah aksi)
  c. Pengumpulan Kebutuhan (sebuah tugas proses)
  d. *Review* sebuah produk kerja (sebuah tugas proses)
  e. Model Desain (sebuah produk kerja)

Penilaian dan Perbaikan Proses

- Adanya proses Perangkat Lunak ada jaminan bahwa perangkat lunak akan dikirim tepat waktu, bahwa itu akan memenuhi kebutuhan pelanggan, atau bahwa ia akan menunjukkan teknis karakteristik yang akan mengakibatkan karakteristik kualitas jangka panjang (Bab 14 dan 15).
- *Process patterns* harus dibarengi dengan praktik rekayasa perangkat lunak yang solid
- Di samping itu, proses itu sendiri dapat dinilai untuk memastikan bahwa telah memenuhi seperangkat kriteria proses dasar yang telah terbukti penting bagi sukses rekayasa perangkat lunak.

  Sejumlah pendekatan yang berbeda untuk penilaian *software* proses dan perbaikan telah diusulkan selama beberapa dekade terakhir

- **SCAMPI**: *Standard CMMI Assessment Method for Process Improvement*

  Menyediakan model proses penilaian dalam lima langkah yang menggabungkan lima fase yaitu: *initiating, diagnosing, establishing, acting, and learning*. Metode SCAMPI mengadopsi SEI CMMl sebagai dasar untuk penilaian.

- **CMM**

  Berdasarkan penilaian untuk *lnternal Process Improvement* (CBA lPl), menyediakan teknik diagnostik untuk menilai kematangan relatif dari perangkat lunak organisasi; menggunakan SEI CMM sebagai dasar untuk penilaian.

- **SPICE (ISO/lEC15504)**

  Standar yang mendefinisikan seperangkat *requirement* untuk penilaian proses Perangkat Lunak. Tujuan dari standar ini adalah untuk membantu organisasi dalam mengembangkan evaluasi, tujuan keberhasilan dari setiap proses *software* yang didefinisikan [lSO08].

- **ISO 9001:2000 *for Software***

  Standar umum yang berlaku untuk organisasi apapun yang ingin meningkatkan kualitas produk, sistem, atau jasa yang diberikannya

D. Penentuan Model Proses

Jenis-jenis Model Proses

1. *Sequence/Linier Model*

   Waterfall Model

   V-Model
2. *Incremental Process Model*
3. *Evolutionary Process Model*

   Prototyping

   Spiral Model
4. *Concurent Model*

NB: Setiap model proses mengadopsi satu atau lebih aliran proses (*proces flow*)

Waterfall Model

- Model *waterfall* disebut juga model sekuensial linier atau siklus kehidupan klasik.
- Model ini mengusulkan pendekatan kepada perkembangan perangkat lunak yang sistematik dan sekuensial yang mulai pada tingkat dan kemajuan sistem pada setiap fasenya.

Tabel 5. Kelebihan dan Kekurangan Model *Waterfall*

Kekurangan	Kelebihan
• Tidak fleksibel karena untuk menuju ke fase selanjutnya harus menunggu fase sebelumnya untuk melengkapi tugas yang saling memiliki ketergantungan. • Sangat sulit untuk *customer* ketika akan melakukan	• Model ini baik untuk produk dengan kebutuhan yang jelas dipahami atau ketika bekerja dengan *technical tools,* arsitektur dan infrastruktur yang

Kekurangan	Kelebihan
perubahan kebutuhan yang eksplisit. • *Customer* harus memiliki kesabaran sampai dengan produk perangkat lunak selesai di implementasikan. • *Customer* hanya terlibat di awal proyek.	telah dipahami dengan baik • ***Sederhana*** • ***langkah-secara terurut, fokus, dan mudah diikuti.***

Gambar 6. *Waterfall Model*

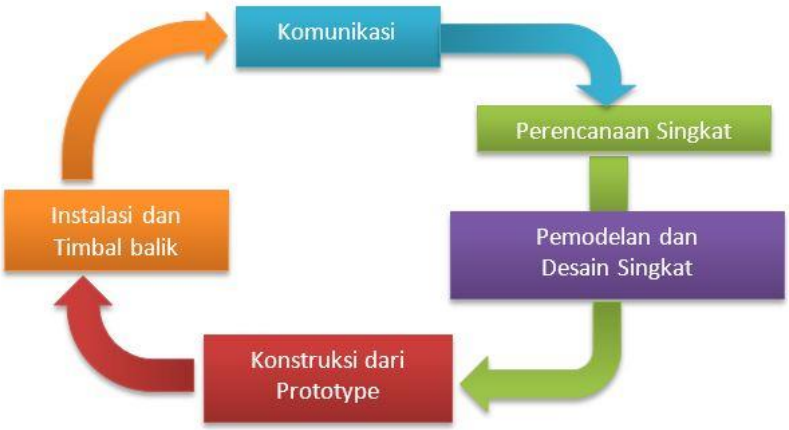
Prototyping Model

- Model ini akan digunakan jika perangkat lunak yang akan dibuat telah didefinisikan secara khusus oleh pelanggan dan pengembang.
- Pendefinisian tersebut antara lain mengidentifikasi kebutuhan *output*, pemrosesan, *input* detail, kepastian terhadap efisiensi algoritme, penyesuaian dari sebuah sistem operasi, dan bentuk-bentuk yang harus dilakukan oleh interaksi manusia dengan mesin.
- Intinya model ini dirancang untuk mendorong konsumen (atau pengembang) agar memahami kebutuhan.

Tabel 6. Kelebihan dan Kekurangan *Prototyping Model*

Kekurangan	Kelebihan
• Tidak ada cara untuk mengetahui banyaknya iterasi yang diperlukan. • Membuat prototipe/*prototyping* dapat mendorong ke arah	• Pelanggan dapat menjadi lebih mantap dalam melihat kemajuan. • Bermanfaat ketika kebutuhan sedang

Kekurangan	Kelebihan
perancangan sistem yang kurang baik. Sebab tujuan utama *protoyping* adalah perkembangan cepat, perancangan sistem kadang-kadang dapat rusak sebab sistem dibangun pada rangkaian "lapisan" tanpa integrasi pertimbangan global dari semua komponen lain.	berubah dengan cepat, ketika pelanggan adalah enggan untuk mengikat kepada satu set kebutuhan, atau ketika tak seorang pun secara penuh memahami area aplikasi itu.



Gambar 7. *Prototyping Model*

RAD *Model (Rapid Application Development)*

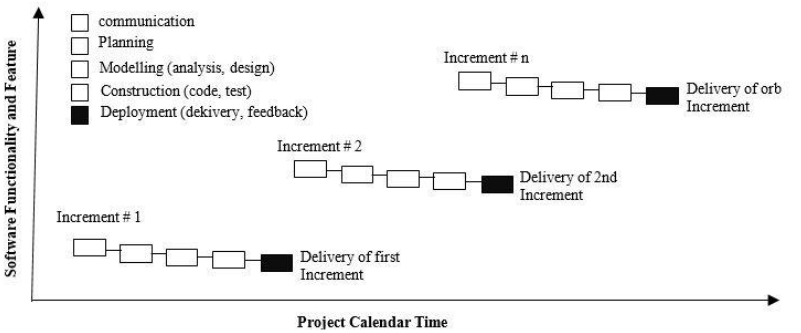
- Model ini akan digunakan jika perangkat lunak yang akan dibuat mempunyai siklus perkembangan yang sangat pendek.
- Jika kebutuhan dipahami dengan baik, proses RAD memungkinkan tim pengembangan menciptakan "sistem fungsionalitas yang utuh" dalam periode waktu yang sangat pendek.
- Model ini lebih menekankan pada perkembangan komponen program yang bisa dipakai kembali (*reusable*).

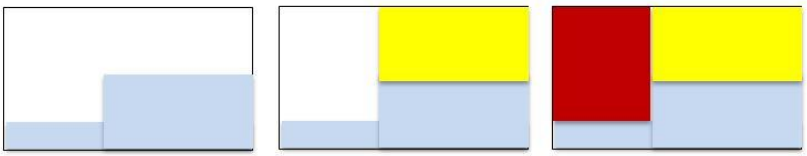
Incremental Model

- Model ini merupakan gabungan elemen-elemen model *waterfall* dan *prototype iterative*.
- Bersifat iteratif jika model tersebut ditandai dengan tingkah laku yang memungkinkan perekayasa perangkat lunak mengembangkan versi perangkat lunak yang lebih lengkap sedikit demi sedikit.

Tabel 7. Kelebihan dan Kekurangan *Incremental Model*

Kekurangan	Kelebihan
• Semangat melakukan proses membangun dan memperbaiki perangkat lunak bisa menurun • Kesalahan desain menjadi bagian dari sistem dan sulit untuk memperbaikinya • Klien melihat kemungkinan-kemungkinan yang ada sehingga ingin mengubah kebutuhan.	• Relatif sedikit jumlah programer atau developer yang digunakan. • Memberikan operasional mutu produk pada tiap *stage*, tetapi satu yang memuaskan hanya subset kebutuhan klien. • Mengurangi efek traumatis dari mengesankan suatu produk sepenuhnya baru pada organisasi klien dengan menyediakan suatu pengenalan berangsur-angsur. • Klien bisa melihat sistem dan memberikan *feedback*.





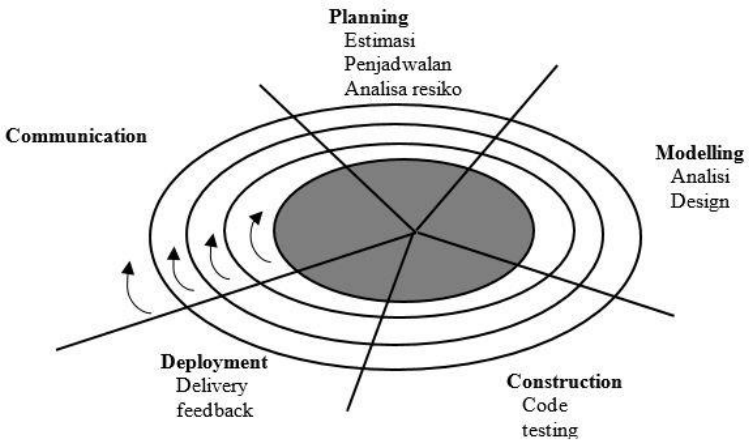
Gambar 8. *Incremental Model*

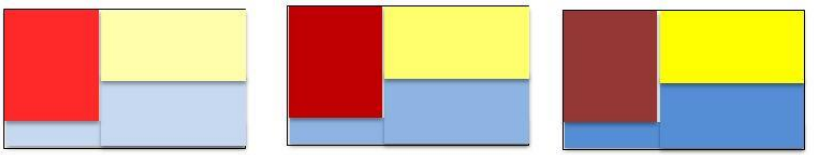
Spiral Model

- Spiral model merupakan suatu pendekatan realistis untuk pengembangan sistem skala besar.
- Model ini juga memelihara pendekatan sistematis seperti yang dianjurkan model *waterfall* namun memadukannya ke dalam kerangka kerja iteratif yang lebih realistis merefleksikan dunia nyata.

Tabel 8. Kelebihan dan Kekurangan *Spiral Model*

Kekurangan	Kelebihan
Rumit dan memerlukan perhatian dan pengetahuan manajemen untuk memulainya.	Masing-masing iterasi dari spiral dapat dikhususkan untuk kebutuhan proyek yang sesuai





Gambar 9. *Spiral Model*

Pemilihan Model Proses

- Aktivitas-aktivitas bingkai kerja akan selalu diaplikasikan pada setiap *project*,
- Tugas-tugas (dan derajat kekakuan) pada setiap aktivitas akan bervariasi bergantung pada :
  a. Tipe proyek
  b. Karakteristik proyek
  c. Penilaian umum; persetujuan tim proyek
- Untuk menyelesaikan sebuah proyek perangkat lunak bisa mengombinasikan lebih dari satu model proses

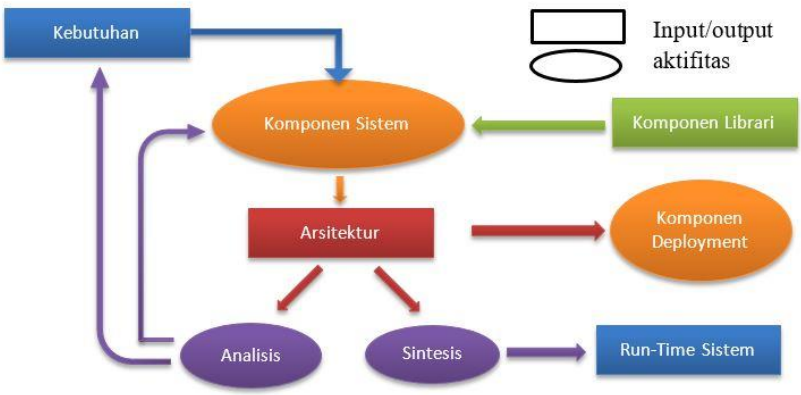
Model Proses Lainnya

- ***Component Based Development***

  Component-based software engineering (CBSE) muncul pada akhir tahun 1990an sebagai sebuah pendekatan untuk pengembangan sistem perangkat lunak yang berdasarkan pada penggunaan kembali komponen perangkat lunak. Kuncinya di sini adalah penggunaan kembali. Fokus metode ini adalah mencari komponen atau bagian sistem yang dapat digunakan, kemudian menggabungkannya dengan menggunakan rancangan arsitektur yang jelas. Pendekatan ini bertujuan untuk membangun sistem dengan cepat dan diikuti kualitas yang lebih baik. CBSE juga dapat dikenal sebagai *component-based development* (CBD).

  Komponen memiliki tingkat abstraksi yang lebih tinggi dibandingkan objek dan didefinisikan oleh *interface*-nya. Komponen juga biasanya lebih besar dibandingkan sebuah objek dan semua detail implementasinya disembunyikan dari komponen lainnya. CBSE berkembang menjadi sebuah

pendekatan pengembangan perangkat lunak yang penting bagi sistem perusahaan berskala-besar, ditambah dengan kebutuhan akan performa dan keamanan. Sehingga, cara untuk memenuhi kebutuhan tersebut adalah membangun perangkat lunak dengan menggunakan kembali komponen-komponen yang sudah ada.



Gambar 10. Contoh Gambaran Proses Pengembangan Berbasis Komponen yang digunakan dalam Sistem Robot

Hal-hal yang Penting dari CBSE adalah:

a. **Komponen *independent* yang dispesifikasikan melalui *interface*-nya**. Harus ada pemisahan yang jelas antara *interface* komponen dan implementasinya, yang berarti, implementasi dari sebuah komponen dapat digantikan yang lainnya tahap perlu mengganti bagian lainnya dari sistem.

b. **Standar komponen yang mendefinisikan *interface* dan memfasilitasi integrasi komponen**. Standar-standar ini terwujud di dalam sebuah model komponen. Standar tersebut mendefinisikan setidaknya mengenai bagaimana *interface* komponen seharusnya dispesifikasikan dan bagaimana komponen berkomunikasi. Beberapa model bahkan menjelaskan lebih lanjut mengenai *interface*s yang harus diimplementasikan oleh semua komponen yang *compatible*. Jika komponen sesuai dengan standar, maka operasinya bersifat independen dari bahasa

pemrogramannya. Komponen yang dikembangkan dalam bahasa pemrograman yang berbeda dapat diintegrasikan ke dalam sistem yang sama.

c. ***Middleware* yang menyediakan dukungan perangkat lunak untuk integrasi komponen**. Untuk membuat komponen-komponen independen dan terdistribusi dapat berjalan bersama, kita membutuhkan dukungan *middleware* yang menangani komunikasi komponen. *Middleware* untuk dukungan komponen menangani masalah-masalah level rendah dengan efisien, sehingga kita cukup hanya fokus pada masalah yang berkaitan dengan aplikasi. Sebagai tambahan, *middleware* untuk dukungan komponen dapat menyediakan dukungan untuk alokasi sumber daya, manajemen transaksi, keamanan, dan konkurensi.

d. **Proses pengembangan yang diarahkan pada rekayasa perangkat lunak berbasis komponen**. Kita membutuhkan proses pengembangan yang mendukung kebutuhan untuk berkembang, bergantung pada fungsionalitas dari komponen yang tersedia.

Pengembangan berbasis komponen mewujudkan bentuk praktik rekayasa perangkat lunak yang baik. Rancangan menggunakan sistem juga dapat dipahami, meskipun kita tetap harus membangun komponen daripada menggunakan kembali komponennya. Hal-hal yang mendasari CBSE sebagai prinsip desain kuat yang mendukung pembangunan perangkat lunak yang dapat dipahami dan dipelihara adalah:

a. Komponen bersifat mandiri, sehingga tidak menginterferensi operasi yang lain. Detail implementasinya juga tersembunyi dan implementasi komponen tersebut dapat diganti tanpa mempengaruhi kinerja keseluruhan sistem.

b. Komponen berkomunikasi melalui *interface* yang sudah didefinisikan dengan baik. Apabila *interface* dipelihara, komponen tersebut dapat digantikan oleh komponen lainnya yang menyediakan fungsionalitas tambahan atau

yang ditingkatkan.

c. Infrastruktur komponen menawarkan berbagai layanan standar yang dapat digunakan dalam sistem aplikasi. Hal ini mengurangi jumlah kode baru yang harus dibangun.

- ***Formal Method Model***
- ***Aspect Oriented Software Development***
- ***Unified Process***
- ***V Model***
- ***Concurent Model***

3

PLANNING
(PERENCANAAN)

A. Definisi Perencanaan (*Planning*)

Aktivitas perencanaan menekankan kepada manajemen dan teknik pelaksanaan yang memungkinkan tim pembuat perangkat lunak mendefinisikan *road map* guna mencapai tujuan.

Apa saja yang direncanakan? (Contoh *Road Map*)

Terdapat enam macam yang harus direncanakan dalam proyek perangkat lunak yaitu: Jadwal (*Timetable*), Kendala (*Constraints*), Sumber Daya (*Resources*), Tujuan (*Aims*), Ruang Lingkup (*Scopes*) dan Kebutuhan (*Requirement*).

Perencanaan Jadwal terdapat dua bagian utama yang dipertimbangkan yaitu waktu penyelesaian yang diinginkan dan ketersediaan sumber daya.

Sedangkan pada kendala, yang mempengaruhi perencanaan adalah ketergantungan, pengendalian internal, peraturan, persetujuan, dan keterbatasan sumber daya. Contoh keterbatasan sumber daya yang mempengaruhi perencanaan penjadwalan: tidak tersedia sumber daya, hari libur nasional, cuti tahunan atau komitmen eksternal.

Sumber daya yang mempengaruhi perencanaan dan juga perlu mendapat perhatian adalah: sumber daya manusia baik sumber daya internal maupun sumber daya eksternal, Anggaran biaya, peralatan dan kemungkinan.

Tujuan dari proyek juga mempengaruhi proses perencanaan yang meliputi: Apa yang kita inginkan dari proyek?, Tolok ukur

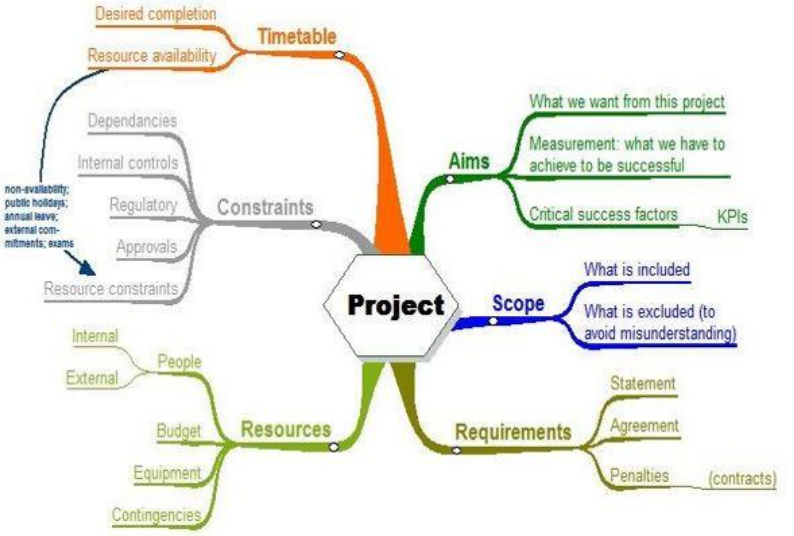
apa yang harus dicapai agar proyek sukses? Faktor penentu keberhasilan proyek.

Ruang lingkup proyek, dalam hal ini apa yang disertakan dan apa yang dikecualikan dalam ruang lingkup harus jelas, untuk menghindari kesalahpahaman.

Kebutuhan dalam perencanaan proyek haruslah melalui pernyataan (*statement*), persetujuan (*agreement*) dan sanksi atau hukuman (*Penalties*).

Mengapa Perencanaan Penting?

- Tidak ada cara yang mudah untuk:
  a. Mengetahui masalah-masalah yang akan terjadi.
  b. Mengetahui informasi penting apa yang masih belum didapat di akhir *project*.
  c. Mengetahui kesalahpahaman apa yang akan terjadi.
  d. Mengetahui isu bisnis apa yang akan berubah.
- Oleh karena itulah sebuah tim pembuat perangkat lunak yang bagus harus merencanakan pendekatan-pendekatan untuk menyelesaikan permasalahan tersebut.



Gambar 11. *Mind Map* Perencanaan Proyek

Kapan Perencanaan dilakukan?

Perencanaan dilakukan setelah *project* tim melakukan komunikasi dengan *customer* sehingga sudah mengetahui ruang lingkup dan *objective project* dan sebelum *project* tim melakukan modeling

Siapa yang Terlibat dalam Perencanaan

Orang-orang yang terlibat dalam proses perencanaan

- Semua *project* tim
- *Customer*

Sepuluh Prinsip Perencanaan

Sepuluh Prinsip di dalam perencanaan:

1. Memahami Ruang Lingkup *Project*
    a. Tidak mungkin kita menggunakan *road map* jika kita tidak tahu ke mana kita akan pergi.
    b. Ruang lingkup memberi petunjuk tim *software* tentang tujuan.
2. Libatkan *Stakeholder* dalam Aktivitas Perencanaan
    a. *Stakeholder* berwenang untuk menentukan prioritas dan membuat batasan *project*.
    b. Untuk mengakomodasi semua itu, *software engineer* harus sering bernegosiasi dengan *stakeholder* untuk memberikan *progress, time lines*, dan isu-isu yang berhubungan dengan *project*.
3. Memahami bahwa Perencanaan itu Bertahap
    a. Tidak ada *project* plan yang mulus.
    b. Seiring berjalannya waktu banyak kemungkinan terjadi perubahan. Sebagai konsekuensinya, perencanaan harus mampu mengakomodasi perubahan.
    c. Di dalam *iterative, incremental process models*, perencanaan ulang dilakukan setelah *progress* dan menerima *feedback* dari *user*.

4. Estimasikan sesuai dengan Pengetahuanmu
    a. Maksud dari estimasi adalah untuk mengetahui tenaga, biaya, waktu, berdasarkan pemahaman tim tentang *project* yang akan dibuat.
    b. Jika informasi tidak jelas dan tidak *reliable* maka estimasi juga tidak akan reliabel.
5. Perhitungkan Risiko juga ketika Membuat Perencanaan
    a. Jika kita telah mengetahui bahwa risiko yang akan terjadi mempunyai dampak yang besar dan kemungkinan terjadinya tinggi, maka *contingency planning* adalah memungkinkan.
    b. *Project* plan (termasuk jadwal) harus memperhatikan kemungkinan risiko-risiko yang ada akan terjadi.
6. Realistislah
    a. Orang tidak akan bekerja 100% setiap hari.
    b. Gangguan selalu ada di dalam komunikasi.
    c. Kelalaian dan ambiguitas adalah fakta kehidupan.
    d. Perubahan akan terjadi.
    e. *Software engineer* terbaik sekalipun dimungkinkan membuat kesalahan.
7. Menyesuaikan dengan *Granularity* ketika Mendefinisikan Perencanaan
    a. ***Granularity*** adalah tingkat kedetailan dari suatu *object* plan.
    b. Sebuah "*high-granularity*" plan mencakup detail pekerjaan yang akan direncanakan di setiap tahap (sehingga *tracking* dan *control* sering terjadi).
    c. Sebuah "*low-granularity*" plan mencakup tugas-tugas yang akan dilaksanakan dalam jangka waktu panjang.
    d. Secara *general granularity*, bergerak dari atas ke bawah seiring dengan waktu.
    e. Setelah beberapa minggu atau bulan, *project* dapat direncanakan secara detail.
    f. Aktivitas yang tidak berlangsung berbulan-bulan tidak

butuh "*high granularity*".

8. Definisikan Bagaimana Kepedulian Kita terhadap Jaminan Kualitas
    a. Rencana harus mengidentifikasi bagaimana tim *software* perhatian untuk menjamin kualitas.
    b. Jika *technical reviews* ada maka harus dijadwalkan.
    c. Jika *pair programming* akan dilakukan pada saat konstruksi maka harus didefinisikan secara jelas pada saat perencanaan.
9. Deskripsikan Kepedulian Kita terhadap Perubahan
    a. Perencanaan yang bagus adalah perencanaan yang dapat terhindar dari perubahan yang tidak terkontrol.
    b. Kita harus mengidentifikasi bagaimana perubahan akan diakomodasi ketika pembuatan *software* berlangsung. Sebagai contoh, dapatkah *customer* meminta perubahan setiap waktu? Jika permintaan untuk mengubah terjadi, apakah tim harus mengimplementasikannya segera? Bagaimana akibat dan biaya akan ditangani?
10. Amati Perencanaan Sesering Mungkin dan Lakukan Penyesuaian Jika dibutuhkan.
    a. Ada saatnya *software project* mengalami problem di suatu hari.
    b. Oleh karena itu, mengamati progres harian sangat penting, guna menemukan masalah dan situasi di mana jadwal tidak sesuai dengan apa yang telah direncanakan.
    c. Ketika problem terjadi, maka Rencana disesuaikan.

B. Perencanaan Proyek Perangkat Lunak

- Proses manajemen proyek Perangkat Lunak dimulai dengan rangkaian aktivitas yang disebut Perencanaan Proyek Perangkat Lunak (*Software Project Planning*)
- Apa tujuan perencanaan proyek?

Adalah untuk memberikan batasan yang memungkinkan bagi manajer untuk mengestimasi sumber daya, biaya dan jadwal yang bisa dipertanggung jawabkan.

Tahapan-tahapan dalam Perencanaan Perangkat Lunak:

1. Memperkirakan (*estimation*)
2. Ruang Lingkup (*scoping*)
3. Risiko (*risk*)
4. Jadwal (*schedule*)
5. Strategi Pengendalian (*control strategy*)

Observasi terhadap Estimasi.

- Estimasi sumber daya, biaya dan jadwal pengembangan Perangkat Lunak memerlukan:
  a. Pengalaman
  b. Akses informasi historis yang baik
  c. Informasi historis.

     Dengan mengetahui data-data yang lalu kita dapat mengoptimalkan pekerjaan dan menghindari hal-hal yang bisa menimbulkan persoalan
  d. Keberanian untuk komitmen terhadap ketersediaan informasi
- Hal-hal yang mempengaruhi estimasi:
  a. "Project Complexity"
  b. "Project size"
  c. "Problem decomposition"
  d. Tingkatan "structural uncertainty". Struktur dalam hal ini adalah tingkatan kebutuhan, kemudahan fungsi yang akan dihasilkan dan informasi yang harus diproses.
- Risiko diukur berdasarkan tingkatan ketidakpastian estimasi terhadap sumber daya, biaya dan jadwal. Jika batasan proyek tidak jelas dan kebutuhan proyek senantiasa berubah maka hal ini bisa menimbulkan dampak yang membahayakan.

Perencanaan Ruang Lingkup Proyek (*Project Scope*)

- Apa yang dimaksud dengan ruang lingkup (*scopes*):
  a. Fungsi (*functions*): Estimasi biaya dan jadwal berorientasi secara fungsional.
  b. Kinerja (*performance*): berkaitan dengan proses dan waktu respon yang dispesifikasikan
  c. Batasan (*constraints*): mengidentifikasikan keterbatasan Perangkat Lunak terhadap perangkat keras, memori maupun terhadap sistem lainnya yang sudah ada.
  d. Antar-muka (*Interface s*)
  e. Reliabilitas (*reliability*)
- Untuk memahami ruang lingkup Perangkat Lunak
  a. Mengerti keinginan pelanggan
  b. Mengerti jenis bisnis yang dilakukan
  c. Mengerti ruang lingkup proyek
  d. Mengerti motivasi pelanggan
  e. Mengerti perubahan-perubahan yang mungkin terjadi
- Pertanyaan yang diajukan untuk memahami ruang lingkup Perangkat Lunak:
  a. Berkaitan dengan tujuan umum:
     1) Siapa yang menginginkan pekerjaan ini ?
     2) Siapa yang mempunyai solusi yang lain ?
     3) Apa keuntungan ekonominya jika solusi tersebut berhasil ?
  b. Berkaitan dengan pemahaman permasalahan:
     1) Bagaimana *output* yang diinginkan pelanggan?
     2) Masalah apa yang bisa di atasi oleh solusi tersebut ?
     3) Adakah batasan atau isu-isu kinerja khusus yang akan mempengaruhi cara pendekatan terhadap solusi?
  c. Berkaitan dengan efektivitas pertemuan:
     1) Apakah anda orang yang tepat utk. menjawab pertanyaan ini?
     2) Apakah pertanyaan saya relevan dengan problem

anda?

3) Apakah masih ada hal lain yang sebaiknya saya tanyakan?

Perencanaan Sumber Daya

Tugas kedua perencanaan Perangkat Lunak adalah meng-estimasi sumber daya yang dibutuhkan untuk menyelesaikan usaha pengembangan Perangkat Lunak tersebut.

1. Sumber Daya Manusia
   a. Mengevaluasi ruang lingkup dan keahlian yang dibutuhkan. Perencanaan harus menentukan posisi organisasi (seperti manajer, perekayasa Perangkat Lunak, dll.) dan spesialisasi (seperti telekomunikasi, *data base, client*/server).
   b. Jumlah orang yang dibutuhkan untuk sebuah proyek Perangkat Lunak bisa ditentukan setelah adanya estimasi usaha untuk pengembangan (seperti *person-months*).
2. Sumber Daya Perangkat Lunak *Reusable*

   Ada 4 katagori *software resource* yang bisa dipertimbangkan:
   a. Komponen *Off-the self*: perangkat lunak yang ada yang dapat diperoleh dari proyek sebelum yang telah divalidasi seluruhnya.
   b. Komponen *Full-Experience* : dikembangkan pada proyek yang lalu yang serupa dengan Perangkat Lunak yang akan dibangun
   c. Komponen *partial-experience*: proyek yang lalu dimodifikasi substansial untuk proyek saat ini.
   d. Komponen baru: komponen Perangkat Lunak yang harus dibangun oleh tim Perangkat Lunak sesuai dengan kebutuhan proyek sekarang.

Estimasi Proyek Perangkat Lunak

1. Pada masa-masa awal perhitungan, biaya perangkat lunak biasanya mendominasi proyek.

2. Katagori teknik estimasi:
    a. Mendasarkan estimasi pada proyek-proyek yang mirip yang sudah dilakukan sebelumnya
    b. Menggunakan "teknik dekomposisi" yang relatif sederhana untuk melakukan estimasi biaya dan usaha proyek.
    c. Menggunakan satu atau lebih model empiris untuk estimasi usaha dan biaya Perangkat Lunak.
3. Keputusan *Make-Buy*

    Dalam banyak area aplikasi Perangkat Lunak, biaya sering lebih efektif untuk mendapatkan dari pada mengembangkan Perangkat Lunak.
4. Akuisisi Perangkat Lunak
    a. Buat atau beli? Beli/beli lalu dimodifkasi/*Outsourcing*
    b. Petunjuk:
        1) Buat spesifikasi fungsi dan kinerja yang diharapkan
        2) Estimasi biaya internal pengembangan dan tgl. penyampaian
        3) Pilih 3 atau 4 perangkat lunak kandidat yang paling cocok
        4) Buat matriks perbandingan dari kandidat tersebut
        5) Evaluasi berdasarkan kualitas sebelumnya, dukungan vendor, reputasi dan dukungan purna jual, dll.
        6) Tanya komentar pemakai lain.
5. Analisis Akhir
    a. Apakah tanggal penyampaian akan lebih cepat dibandingkan mengembangkan sendiri?
    b. Apakah biaya pembelian + biaya pengubahan lebih kecil dari biaya pengembangan sendiri?
    c. Apakah biaya dukungan dari pihak luar lebih kecil dari biaya dukungan dari dalam?

Kerangka Dokumen Rencana Proyek Pengembangan Perangkat Lunak.

1. Pendahuluan
    a. Maksud dan tujuan proyek
    b. Fungsi utama perangkat lunak
    c. Sasaran yang akan dicapai
    d. Kendala proyek
2. Estimasi Proyek
    a. Metode estimasi
    b. Estimasi biaya & sumber daya manusia
3. Risiko Proyek
    a. Analisis risiko
    b. Manajemen risiko
4. Jadwal Proyek
    a. Kegiatan & waktu
    b. *Network planning*
    c. SD kegiatan
5. Sumber daya
    a. Manusia
    b. Perangkat keras
    c. Perangkat lunak
6. Organisasi
    a. Struktur organisasi
    b. Pelaporan
7. Lampiran.

C. Model estimasi Biaya dalam perencanaan proyek

- LOC
- *Function Point*
- Model COCOMO

LOC

Contoh Estimasi berbasis LOC :

Perangkat Lunak CAD akan menerima data geometri dua dan tiga dimensi dari seorang perekayasa yang akan berinteraksi dan mengontrol sistem CAD melalui suatu *interface* pengguna. Kajian spesifikasi sistem menunjukkan bahwa Perangkat Lunak akan mengeksekusi Workstation dan harus berinteraksi dengan berbagai periperal grafis komputer seperti mouse, digitizer dan printer laser.

Diketahui:

Perhitungan LOC untuk Fungsi Analisis geometri 3D (3DGA):

Optimis : 4600

Most likely : 6900

Pesimistik : 8600

Carilah EV untuk fungsi-fungsi lain

EV = (4600 + 4*6900 + 8600)/6

= 6800 LOC

Tabel 9. Perkiraan (estimasi) untuk metode LOC

Fungsi	LOC terestimasi
Interface pemakai dan fasilitas kontrol (UICF)	2.300
Analisis geometrik dua dimensi (2DGA)	5.300
Analisis geometrik tiga dimensi (3DGA)	6.800
Manajemen database (DBM)	3.350
Fasilitas display grafis komputer (CGDF)	4.950
Kontrol periperal (PC)	2.100
Modul analisis desain (DAM)	8.400
Baris kode terestimasi	**33.200**

Jika:

Produktivitas rata-rata organisasional = 620 LOC/person-month

Upah karyawan = $8.000 per bulan

Biaya per baris kode = $13

Maka : Tingkat produktivitas = jumlah titik fungsi / Jumlah orang-bulan

Jumlah Karyawan = 33.200 LOC / 620 LOC/bln = 53,5 = 54 orang

Estimasi biaya proyek berdasar LOC

= 33.200 LOC * $13

= $431.600

Estimasi biaya proyek berdasar upah

= 54 orang * $8.000

= $432.000

Estimasi berbasis FP (*Function Point*)

Dekomposisi untuk perhitungan berbasis FP berfokus pada harga domain info daripada fungsi Perangkat Lunak. Perencana proyek memperkirakan *input*, *output, inquiry, file* dan *interface* eksternal. Untuk tujuan perkiraan tersebut faktor pembobotan kompleksitas diasumsikan menjadi rata-rata.

Setiap faktor pembobotan kompleksitasnya diestimasi dan faktor penyesuaian kompleksitas dihitung seperti di bawah ini:

Faktor	**Harga**
Backup dan recovery	4
Komunikasi data	2
Pemrosesan terdistribusi	0
Kinerja kritis	4
Lingkungan operasi yang ada	3
Entri data online	4
Transaksi *input* pada layar ganda	5
File master yang diperbaharui secara 0nline	3
Nilai kompleks domain informasi	5
Pemrosesan internal yang kompleks	5
Kode yang didesain untuk dapat dipakai lagi	4
Konversi/instalasi dalam desain	3

Instalasi ganda	5
Aplikasi yang didesain bagi perubahan	5
Faktor penyesuaian kompleksitas	1.17
Total	**53.17**

Perkiraan harga domain informasi:

Nilai Domain Informasi	opt	likely	pess	Jumlah estimasi	Bobot	Jumlah FP
Jumlah	20	24	30	24	4	96
input	12	15	22	16	5	80
Jumlah	16	22	28	22	4	88
output	4	4	5	4	10	40
Jumlah inquiry	2	2	3	2	7	14
Jumlah *file*						
Jumlah *interface* eksternal						
Jumlah Total						**318**

Jumlah estimasi (lihat rumus EV)

Bobot (lihat kembali bab 4)

Jumlah FP = jumlah estimasi * bobot

Total faktor pembobotan = $\sum$Fi = 53,17

Total FP = 318

FP terestimasi = jumlah total * (0.65 + 0.01 * $\sum$Fi)

= 318 * (0,65 + 0.01 * 53,17) = 375

Diketahui :

Produktivitas = 6.5 LOC/pm (dari historis)

Upah = $ 8.000/m

Biaya FP = $\frac{\$\ 8.000}{65\ LOC}$ = $ 1.230

Estimasi biaya proyek

= Biaya FP * FP terestimasi

= $ 1.230 * 375

= $ 461.250

4

KEBUTUHAN PERANGKAT LUNAK

A. Kebutuhan (*Requirement*)

Sesuatu yang diminta, dibutuhkan

Menurut IEEE (*the institute of electrical and electronics engineers*)

- Kondisi atau kemampuan yang diperlukan pemakai untuk menyelesaikan persoalan untuk mencapai sebuah tujuan
- Kondisi atau kemampuan yang harus dimiliki atau dipunyai oleh sistem atau komponen sistem untuk memenuhi kontrak, standar, spesifikasi, atau dokumen formal lainnya.

Kebutuhan Perangkat Lunak

Adalah kondisi, kriteria, syarat atau kemampuan yang harus dimiliki oleh perangkat lunak untuk memenuhi apa yang disyaratkan atau diinginkan pemakai.

Secara kategoris, ada tiga buah jenis kebutuhan perangkat lunak [IEE93]:

1. Kebutuhan Fungsional (*Functional Requirement*)

   Disebut juga kebutuhan operasional, yaitu kebutuhan yang berkaitan dengan fungsi atau proses transformasi yang harus mampu dikerjakan oleh perangkat lunak.

   Contoh:

   a. Perangkat lunak harus dapat menyimpan semua rincian data pesanan pelanggan.
   b. Perangkat lunak harus dapat membuat laporan penjualan sesuai dengan periode waktu tertentu.

c. Perangkat lunak harus mampu menyajikan informasi jalur pengiriman barang terpendek.

2. Kebutuhan antarmuka (*Interface Requirement*)

Kebutuhan antarmuka yang menghubungkan perangkat lunak dengan elemen perangkat keras, perangkat lunak, atau basis data.

Contoh:

a. Perangkat untuk memasukkan data dapat berupa *keyboard*, *mouse* atau *scanner*.
b. Akses ke basis data menggunakan ODBC (*Open Database Connectivity*).

3. Kebutuhan Unjuk Kerja (*Performance Requirement*)

Kebutuhan yang menetapkan karakteristik unjuk kerja yang harus dimiliki oleh perangkat lunak, misalnya: kecepatan, ketepatan, frekuensi.

Contoh:

a. Perangkat lunak harus bisa mengolah data sampai 1 juta record untuk tiap transaksi.
b. Perangkat lunak harus dapat digunakan oleh multi*user* sesuai dengan otoritas yang diberikan pada *user* .
c. Waktu tanggap penyajian informasi maksimal selama satu menit.

Mengapa Kebutuhan Penting?

- Sangat mempengaruhi sukses atau gagalnya pelaksanaan pengembangan perangkat lunak.
- Menurut hasil survei DeMarco, 56% kegagalan proyek pengembangan perangkat lunak dikarenakan ketidak-lengkapan pendefinisian kebutuhan dari perangkat lunak tersebut.

Analisa Kebutuhan

Analisis Kebutuhan PL merupakan aktivitas awal dari siklus hidup pengembangan PL. Untuk proyek besar analisis kebutuhan

dilaksanakan setelah aktivitas *sistem information engineering* dan *software projek planning.*

- Proses mempelajari kebutuhan pemakai untuk mendapatkan definisi kebutuhan sistem atau perangkat lunak [IEE93].
- Proses untuk menetapkan fungsi dan unjuk kerja perangkat lunak, menyatakan antarmuka perangkat lunak dengan elemen-elemen sistem lain, dan menentukan kendala yang harus dihadapi perangkat lunak [PRE01].

Tujuan Pelaksanaan Analisis Kebutuhan adalah

1. Memahami masalah secara menyeluruh (komprehensif) yang ada pada perangkat lunak yang akan dikembang seperti ruang lingkup produk perangkat lunak (*product space*) dan pemakai yang akan menggunakannya.
2. Mendefinisikan apa yang harus dikerjakan oleh perangkat lunak untuk memenuhi keinginan pelanggan.

B. Prinsip-prinsip Pemodelan Kebutuhan

Banyak metode untuk merepresentasikan *requirement* modeling. Pada dasarnya metode-metode tersebut mempunyai prinsip-prinsip berikut:

1. Domain informasi dari problem harus direpresentasikan dan dimengerti. Domain informasi ini meliputi data yang mengalir ke dalam sistem (dari *end user*, sistem lain, atau alat eksternal), data yang dihasilkan sistem (bisa melalui *user interface*, *network interface*, *report*, grafik, dll.), data yang disimpan (data yang dipelihara secara permanen)
2. Fungsi-fungsi yang dimiliki oleh *software* harus didefinisikan. Fungsi-fungsi di dalam *software* menyediakan keuntungan langsung kepada pengguna dan juga memberi *support* internal untuk fitur yang lain. Fungsi dapat didefinisikan pada level abstraksi yang berbeda-beda, mulai dari yang paling umum sampai yang paling detail.
3. Perilaku (*behavior*) dari *software* harus direpresentasikan. Perilaku dari *software* biasanya dipengaruhi oleh interaksinya dengan lingkungan eksternal.

a. *Input* data disediakan oleh *end user* ,
b. *Control* data disediakan oleh sistem eksternal
c. *Monitoring* data dikumpulkan melalui *network* dan menyebabkan *software* bertindak secara khusus

4. Model yang menggambarkan informasi, fungsi, dan perilaku harus di partisi di dalam layer-layer.

   Problem yang kompleks dipecah ke dalam sub-problem – sub problem sampai semua sub problem mudah untuk dimengerti.

5. Proses analisis bergerak dari informasi yang sangat penting ke detail implementasi. *Requirement* modeling dimulai dari pendeskripsian problem dilihat dari perspektif **end-*user*.**

Tahapan Analisis Kebutuhan

Secara teknis pelaksanaan pekerjaan analisis kebutuhan perangkat lunak pada dasarnya terdiri dari urutan aktivitas:

- **Mempelajari dan Memahami Persoalan**
  a. siapa pemakai yang menggunakan perangkat lunak.
  b. di mana perangkat lunak akan digunakan.
  c. pekerjaan apa saja dari pemakai yang akan dibantu oleh perangkat lunak.
  d. apa saja cakupan dari pekerjaan tersebut, dan bagaimana mekanisme pelaksanaannya.
  e. apa yang menjadi kendala dilihat dari sisi teknologi yang digunakan atau dari sisi hukum dan standar.
- **Mengidentifikasi Kebutuhan Pemakai**
  a. Fungsi apa yang diinginkan pada perangkat lunak.
  b. Data atau informasi apa saja yang akan diproses.
  c. Kelakuan sistem apa yang diharapkan.
  d. Antarmuka apa yang tersedia (*software interfaces, hardware interfaces, user interfaces*, dan *communication interfaces*)
- **Mendefinisikan Kebutuhan Perangkat Lunak**
  a. Saya ingin data yang dimasukkan oleh bagian penjualan bisa langsung dijurnal.

b. Informasi neraca keuangan bisa saya lihat kapan saja.

Sebagai contoh, kebutuhan "data yang dimasukkan oleh bagian penjualan bisa langsung dijurnal" setelah dianalisis, diklasifikasikan dan diterjemahkan, pendefinisian kebutuhan:

1) Kebutuhan Fungsional
   a) Entri dan rekam data transaksi penjualan.
   b) *Retrieve* data transaksi penjualan untuk periode tertentu (periode sesuai dengan *input*-an periode yang di-*input* kan pada *keyboard*).
   c) Rekam data akumulasi transaksi penjualan periode tertentu ke jurnal umum berikut *account* pasangannya (kas).
2) Kebutuhan Antarmuka
   a) Antarmuka pemakai untuk memasukkan dan merekam data penjualan.
   b) Antarmuka pemakai untuk menyajikan dan menjurnal informasi transaksi penjualan pada periode tertentu.
   c) Antarmuka untuk jaringan lokal yang meng-hubungkan perangkat lunak aplikasi di bagian penjualan dengan perangkat lunak aplikasi di bagian akuntansi.
3) Kebutuhan Unjuk Kerja
   a) Proses jurnal hanya bisa dilakukan sekali setelah data transaksi penjualan direkam.
   b) Adanya otoritas pemakaian perangkat lunak dan akses data sesuai dengan bagian pekerjaan masing-masing.
   c) Kebutuhan dimodelkan/digambarkan dengan teknik analisis dan alat bantu tertentu

   Contoh kebutuhan fungsional dapat dimodelkan dengan menggunakan
   a) Data *flow* diagram, kamus data, dan spesifikasi

proses jika menggunakan analisis terstruktur.

b) *Use case* diagram dan skenario sistem jika menggunakan analisis berorientasi objek.

- **Membuat Dokumen Spesifikasi Kebutuhan Perangkat Lunak**

  Semua kebutuhan yang telah didefinisikan selanjutnya dibuat dokumentasinya yaitu Spesifikasi Kebutuhan Perangkat Lunak (SKPL) atau *Software Requirement Specification* (SRS).

- **Mengkaji Ulang (*Review*) Kebutuhan**

  Proses untuk mengkaji ulang (validasi) kebutuhan apakah SKPL sudah konsisten, lengkap, dan sesuai dengan yang diinginkan oleh pemakai.

  Sedangkan menurut Pressman [PRE01], analisis kebutuhan perangkat lunak dapat dibagi menjadi lima area pekerjaan, yaitu:

  a. Pengenalan masalah
  b. Evaluasi dan sistesis
  c. Pemodelan
  d. Spesifikasi
  e. Tinjau ulang (*review*)

Metode Analisa

Metode atau teknik untuk melakukan analisis kebutuhan perangkat lunak dapat dikelompokkan berdasarkan pendekatan yang diambil pada saat melakukan aktivitas tersebut

Salah satu metode yang paling populer untuk pendekatan ini adalah Analisis Terstruktur (*Structured Analysis*)

1. Berorientasi Aliran Data (*Data Flow Oriented* atau *Functional Oriented*)
2. Berorientasi Struktur Data (*Data Structured Oriented*)
3. Berorientasi Objek (*Object Oriented*)

Berorientasi Aliran Data (*Data Flow Oriented atau Functional Oriented*)

Pada metode ini, hasil analisis dan perancangan dimodelkan dengan menggunakan beberapa perangkat pemodelan seperti:

- *Data Flow Diagram* (DFD) dan Kamus Data (*data dictionary*) untuk menggambarkan fungsi-fungsi dari sistem (*system functions*).
- *Entity-Relationship Diagram* (ERD) untuk menggambarkan data yang disimpan (*data store d*).
- *State Transition Diagram* (STD) untuk menggambarkan perilaku sistem.
- *Structure Chart* untuk menggambarkan struktur program.

Berorientasi Objek (*Object Oriented*)

- Berbeda dengan pendekatan-pendekatan sebelumnya, pendekatan berorientasi objek memandang sistem yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata.
- Pada pendekatan ini, informasi dan proses yang dipunyai oleh suatu Objek "dienkapsulasi" (dibungkus) dalam satu kesatuan.
- Beberapa metode pengembangan sistem yang berorientasi objek ini diantaranya adalah:
  a. *Object Oriented Analysis* (OOA) dan *Object Oriented Design* (OOD) dari Peter Coad dan Edward Yourdon (1990).
  b. *Object Modeling Technique* (OMT) dari James Rumbaugh (1987).
  c. *Object Oriented Software Engineering* (OOSE).

C. Spesifikasi Kebutuhan Perangkat Lunak/*Software Requirement s Specification* (SRS)

Sebuah dokumen yang berisi pernyataan lengkap dari apa yang dapat dilakukan oleh perangkat lunak, tanpa menjelaskan bagaimana hal tersebut dikerjakan oleh perangkat lunak.

Tujuan Pembuatan SRS

- Pemakai potensial (pelanggan) dari sistem
- Pengembang sistem
  a. Mendefinisikan keinginan yang biasanya dinyatakan dalam bentuk penjelasan umum.
  b. Tujuan kedua:
     1) Sarana komunikasi antara pelanggan, pemakai, analis, dan perancang perangkat lunak.
     2) Dasar untuk merencanakan dan melaksanakan aktivitas pengujian sistem.
     3) Acuan untuk melakukan perbaikan dan perubahan perangkat lunak.

Syarat Pembentukan SRS

1. Mudah diidentifikasi
2. Diuraikan dengan jelas, simpel, sederhana, dan *concise* (jelas, tidak *ambiguous*)
3. Bisa divalidasi dan bisa dites (*test reliable, test accessable*)
4. Mampu untuk ditelusuri kembali (*tracebility*)

Hal-hal yang harus dihindari saat pembentukan:

- *Over specification* (penjelasan berlebih dan berulang-ulang sehingga menjadi tidak jelas)
- Tindakan *unconcistency* (seperti menggunakan istilah yang tidak konsisten)
- *Ambiguity* dalam kata atau kalimat seperti menyatakan keterukuran
- kebutuhan secara tidak jelas misalkan menggunakan kata-kata: minimal, maksimal, optimal, cepat, *user friendly*, efisien, fleksibel dan lainnya.
- Menuliskan "mimpi-mimpi", yaitu hal-hal yang tidak bisa dilakukan

Atribut Penulisan SRS yang Baik

Dokumen SRS yang baik (sempurna) akan ditulis secara:

- Benar (*correct*)
- Tepat (*precise*)
- *Unambiguouity*
- Lengkap (*complete*)
- Bisa diverifikasi (*verifiable*)
- Konsisten
- *Understandable*
- Bisa dimodifikasi (*modifiedable*)
- Dapat ditelusuri (*traceable*)
- Harus dapat dibedakan bagian *what* (bagian spesifikasi) dan *how* (bagian yang menjelaskan bagaimana menyelesaikan *what* tadi).
- Dapat mencakup dan melingkupi seluruh sistem
- Dapat melingkupi semua lingkungan operasional, misalnya interaksi fisik dan operasional.
- Bisa menggambarkan sistem seperti yang dilihat oleh pemakai.
- Harus toleran (bisa menerima) terhadap ketidaklengkapan, ketidakpastian (*ambiguous*) dan ketidakkonsistenan.
- Harus bisa dilokalisasi dengan sebuah *coupling*, yaitu hubungan ketergantungan antara dua model yang tidak terlalu erat.

Ada 9 macam orang yang terlibat dalam pembuatan SKPL:

1. Pemakai (*user*)
2. *Client*
3. *System analyst* (*system engineer*)
4. *Software engineer*
5. *Programmer*
6. *Test integration group*
7. *Maintenance group*

8. *Technical Support*
9. *Staff dan Clerical Work*

Keberhasilan pengembangan perangkat lunak bisa dilihat dari 10 aspek atau titik pandang:

- Ketelitian dari pembuatnya
- Kualitas dari spesifikasi perangkat lunak yang dihasilkan (baik, jika ada sedikit kesalahan)
- Integritas
- Ketelitian
- Proses pembuatan yang mantap
- Mudah dikembangkan
- Jumlah versi tidak banyak
- Ketelitian dari model pengembangan yang digunakan untuk meramal atribut perangkat lunak
- Efektivitas rencana tes dan integrasi
- Tingkat persiapan untuk sistem perawatan (mempersiapkan pencarian *bugs*)

5

PEMODELAN PERANGKAT LUNAK TERSTRUR

Pemodelan perangkat lunak terstruktur adalah perancangan sistem yang berorientasi pada alur data dengan konsep dekomposisi.

Pendekatan terstruktur dilengkapi dengan alat-alat (*tools*) dan teknik (*techniques*) yang dibutuhkan dalam pengembangan sistem, sehingga hasil akhir dari sistem yang dikembangkan akan diperoleh sistem yang strukturnya didefinisikan dengan baik dan jelas.

Melalui pendekatan terstruktur, permasalahan yang kompleks diorganisasi dapat dipecahkan dan hasil dari sistem akan mudah untuk dipelihara, fleksibel, lebih memuaskan pemakainya, mempunyai dokumentasi yang baik, tepat waktu, sesuai dengan anggaran biaya pengembangan, dapat meningkatkan produktivitas dan kualitasnya akan lebih baik (bebas kesalahan).

A. *Data Flow Diagram*

Data Flow Diagram (DFD) biasa disebut dengan Diagram Alir Dat (DAD) adalah alat pembuatan model yang memungkinkan profesional sistem untuk menggambarkan sistem sebagai suatu jaringan proses fungsional yang dihubungkan satu sama lain dengan alur data, baik secara manual maupun komputerisasi. DFD ini sering disebut juga dengan nama Bubble chart, Bubble diagram, model proses, diagram alur kerja, atau model fungsi.

DFD ini adalah salah satu alat pembuatan model yang sering digunakan, khususnya bila fungsi-fungsi sistem merupakan bagian yang lebih penting dan kompleks dari pada data yang dimanipulasi oleh sistem.

Dengan kata lain, DFD adalah alat pembuatan model yang memberikan penekanan hanya pada fungsi sistem. DFD ini merupakan alat perancangan sistem yang berorientasi pada alur data dengan konsep dekomposisi dapat digunakan untuk penggambaran analisa maupun rancangan sistem yang mudah dikomunikasikan oleh profesional sistem kepada pemakai maupun pembuat program.

DFD terdiri dari 2 bagian utama yaitu:

- *Context Diagram* (DAD Konteks Sistem) atau sering disebut juga dengan istilah Diagram Level 0.
- Diagram Level, yaitu diagram yang merupakan penjabaran dari masing-masing proses, hingga pada sub proses terkecil dari sebuah sistem. Diagram Level ini tergantung pada ruang lingkup atau kedalaman dari sebuah proses (level 1, 2, 3 ….)

DFD dapat digambarkan dengan menggunakan case tool seperti Power Design, Smart Draw dan lain-lain. DFD juga disertai dengan Kamus Data yang mendeskripsikan data yang mengalir pada aliran dara.

B. Komponen/Elemen *Data Flow Diagram*

Terdapat beberapa komponen/elemen yang digunakan dalam membuat atau menggambarkan DFD seperti berikut ini:

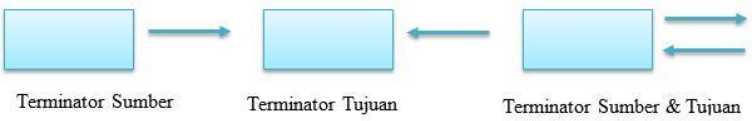
Tabel 10. Simbol Komponen/Elemen *Data Flow* Diagram

SIMBOL	NAMA KOMPONEN
	Terminator/Eksternal *Entity*/ Entitas Eksternal
	Process/Proses

atau	Data *Store*
	Data Flow/Alur Data

1. Komponen/Elemen Terminator/Eksternal *Entity*/Entitas Eksternal

Terminator mewakili entitas luar yang berkomunikasi dengan sistem yang sedang dikembangkan. Terdapat dua jenis terminator yaitu terminator sumber (*Source*): Terminator yang menjadi sumber informasi atau data. Dan Terminator Tujuan (Sink) merupakan terminator yang menjadi tujuan data /informasi dari Sistem.



Gambar 12. Jenis Terminator.

Terminator dapat berupa orang, sekelompok orang, organisasi, departemen di dalam organisasi, atau perusahaan yang sama tetapi di luar kendali sistem yang sedang dibuat modelnya Terminator dapat juga berupa departemen, divisi atau sistem di luar sistem yang berkomunikasi dengan sistem yang sedang dikembangkan.

Pemberian nama pada komponen/elemen terminator berupa Kata Benda sesuai dengan nama yang digunakan oleh pihak yang terlibat atau menggunakan sistem, seperti Fakultas, Mahasiswa, Bagian Penjualan.

Ada tiga hal penting yang harus diingat tentang terminator:

a. Terminator merupakan bagian/lingkungan luar sistem. Alur data yang menghubungkan terminator dengan

berbagai proses sistem, menunjukkan hubungan sistem dengan dunia luar.

b. Profesional Sistem Tidak berhak mengubah isi atau cara kerja organisasi atau prosedur yang berkaitan dengan terminator.

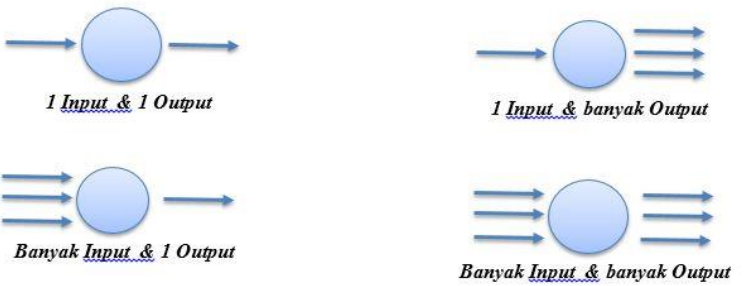
c. Hubungan yang ada antar terminator yang satu dengan yang lain tidak digambarkan pada DFD.

2. Komponen/Elemen Proses

Komponen proses menggambarkan bagian dari sistem yang mentransformasikan *input* menjadi *output*.

Proses diberi nama untuk menjelaskan proses/kegiatan apa yang sedang/akan dilaksanakan. Pemberian nama proses dilakukan dengan menggunakan kata kerja transitif (kata kerja yang membutuhkan obyek), seperti Membuat Jadwal, Menghitung Nilai, Mencetak KRS, Menghitung Jumlah Penjualan.

Ada empat kemungkinan yang dapat terjadi dalam proses sehubungan dengan *input* dan *output*:



Gambar 13. Jenis Alur *Input* dan *Output* pada Elemen Proses

Ada beberapa hal yang perlu diperhatikan tentang proses:

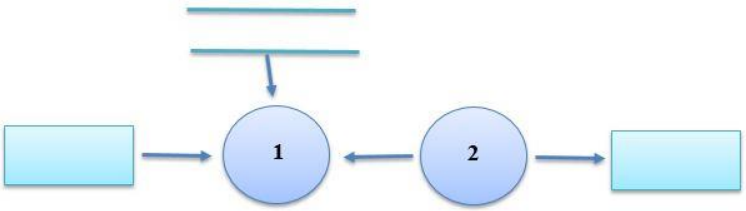
a. Proses harus memiliki *input* dan *output*.

b. Proses dapat dihubungkan dengan komponen terminator, data *store* atau proses melalui alur data.

c. Sistem/bagian/divisi/departemen yang sedang dianalisis oleh profesional sistem digambarkan dengan komponen

proses.

Berikut contoh menggambarkan elemen/komponen proses **salah:**

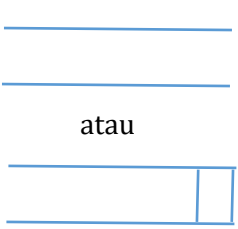


Gambar 14. Contoh Menggambarkan Elemen Proses yang Salah

Umumnya kesalahan proses di DFD adalah :

a. Proses mempunyai *input* tetapi tidak menghasilkan *output*. Kesalahan ini disebut dengan *black hole* (lubang hitam), karena data masuk ke dalam proses dan lenyap tidak berbekas seperti dimasukkan ke dalam lubang hitam (lihat proses 1).
b. Proses menghasilkan *output* tetapi tidak pernah menerima *input*. Kesalahan ini disebut dengan *miracle* (ajaib), karena ajaib dihasilkan *output* tanpa pernah menerima *input* (lihat proses 2).

3. Komponen/Elemen *Data Store*



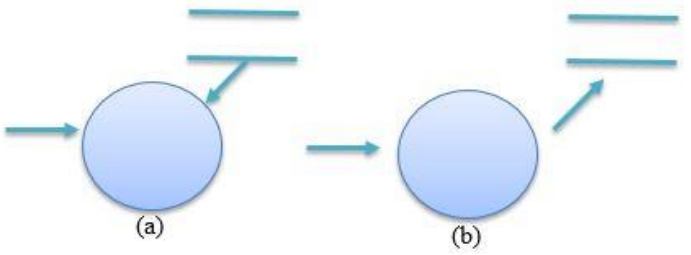
Komponen ini digunakan untuk membuat model sekumpulan paket data dan diberi nama dengan kata benda jamak, misalnya Mahasiswa.

Data *store* ini biasanya berkaitan dengan penyimpanan-penyimpanan, seperti *file* atau database yang berkaitan dengan penyimpanan secara komputerisasi, misalnya *file* disket, *file harddisk*, *file* pita magnetik. Data *store* juga berkaitan dengan penyimpanan secara manual seperti buku alamat, *file* folder, dan agenda. Suatu data *store* dihubungkan dengan alur data hanya pada komponen proses, tidak dengan komponen DFD

lainnya. Alur data yang menghubungkan data *store* dengan suatu proses mempunyai pengertian sebagai berikut:

a. Alur data dari data *store* yang berarti sebagai pembacaan atau pengaksesan satu paket tunggal data, lebih dari satu paket data, sebagian dari satu paket tunggal data, atau sebagian dari lebih dari satu paket data untuk suatu proses (lihat gambar 2 (a)).
b. Alur data ke data *store* yang berarti sebagai peng-*update*-an data, seperti menambah satu paket data baru atau lebih, menghapus satu paket atau lebih, atau mengubah/memodifikasi satu paket data atau lebih (lihat gambar 2 (b)).

Pada pengertian pertama jelaslah bahwa data *store* tidak berubah, jika suatu paket data/informasi berpindah dari data *store* ke suatu proses. Sebaliknya pada pengertian kedua data *store* berubah sebagai hasil alur yang memasuki data *store*. Dengan kata lain, proses alur data bertanggung jawab terhadap perubahan yang terjadi pada data *store* .



Gambar 15. Alur Elemen Data *Store*

4. Komponen/Elemen *Data Flow*/Alur Data.

Suatu *data flow*/alur data digambarkan dengan anak panah, yang menunjukkan arah menuju ke dan keluar dari suatu proses. Alur data ini digunakan untuk menerangkan perpindahan data atau paket data/informasi dari satu bagian sistem ke bagian lainnya.

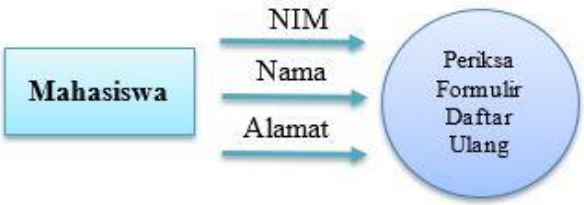
Selain menunjukkan arah, alur data pada model yang dibuat oleh profesional sistem dapat merepresentasikan bit, karakter, pesan, formulir, bilangan real, dan macam-macam informasi yang berkaitan dengan komputer. Alur data juga dapat merepresentasikan data/informasi yang tidak berkaitan dengan komputer.

Alur data perlu diberi nama sesuai dengan data/informasi yang dimaksud, biasanya pemberian nama pada alur data dilakukan dengan menggunakan kata benda, contohnya Laporan Penjualan.

Ada empat konsep yang perlu diperhatikan dalam penggambaran alur data, yaitu:

a. Konsep Paket Data (*Packets of Data*)

   Apabila dua data atau lebih mengalir dari suatu sumber yang sama menuju ke tujuan yang sama dan mempunyai hubungan, dan harus dianggap sebagai satu alur data tunggal, karena data itu mengalir bersama-sama sebagai satu paket.



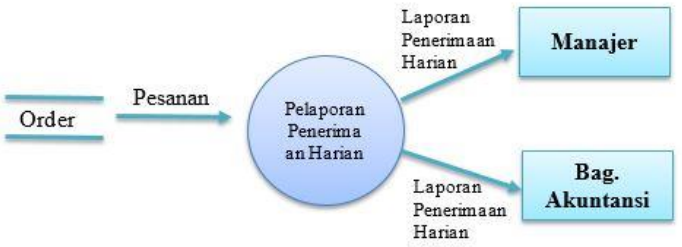
(a) Konsep Paket Data yang Salah

(b) Konsep Paket Data yang Benar

Gambar 16. Kosep Paket Data

b. Konsep Alur Data Menyebar

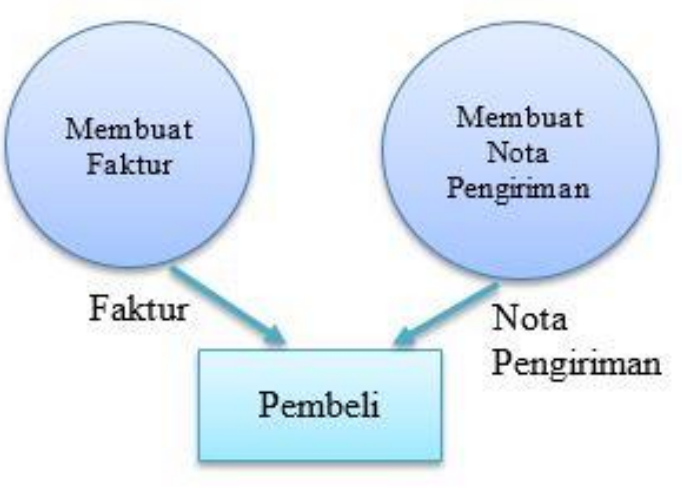
Alur data menyebar menunjukkan sejumlah tembusan paket data yang berasal dari sumber yang sama menuju ke tujuan yang berbeda, atau paket data yang kompleks dibagi menjadi beberapa elemen data yang dikirim ke tujuan yang berbeda, atau alur data ini membawa paket data yang memiliki nilai yang berbeda yang akan dikirim ke tujuan yang berbeda.



Gambar 17. Konsep Alur Data Menyebar

c. Konsep Alur Data Mengumpul

Beberapa alur data yang berbeda sumber bergabung bersama-sama menuju ke tujuan yang sama.

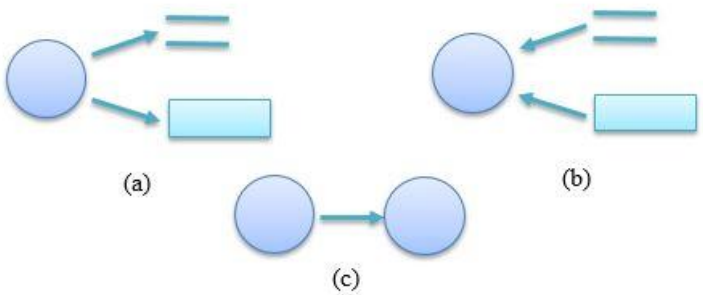


Gambar 18. Konsep Alur Data Mengumpul

d. Konsep Sumber atau Tujuan Alur Data

Semua alur data harus minimal mengandung satu proses. Maksud kalimat ini adalah:

1) Suatu alur data dihasilkan dari suatu proses dan menuju ke suatu data *store* dan/atau terminator (lihat gambar 6 (a)).
2) Sutu alur data dihasilkan dari suatu data *store* dan/atau terminator dan menuju ke suatu proses (lihat gambar 6 (b)).
3) Suatu alur data dihasilkan dari suatu proses dan menuju ke suatu proses (lihat gambar 6 (c)).



Gambar 19. Konsep Sumber dan Tujuan Alur Data

C. Bentuk Data *Flow Diagram*

Terdapat dua bentuk DFD, yaitu Diagram Alur Data Fisik, dan Diagram Alur data Logika. Diagram alur data fisik lebih menekankan pada bagaimana proses dari sistem diterapkan, sedangkan diagram alur data logika lebih menekankan proses-proses apa yang terdapat di sistem.

1. Diagram Alur Data Fisik (DADF)

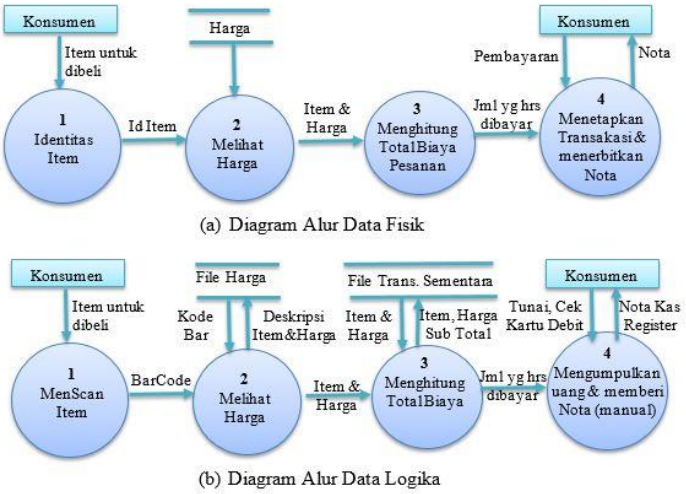
DADF lebih tepat digunakan untuk menggambarkan sistem yang ada (sistem yang lama). Penekanan dari DADF adalah bagaimana proses dari sistem diterapkan (dengan cara apa, oleh siapa dan di mana), termasuk proses-proses manual.

Untuk memperoleh gambaran bagaimana sistem yang ada diterapkan, DADF harus memuat:

a. Proses-proses manual juga digambarkan.
b. Nama dari alur data harus memuat keterangan yang cukup terinci untuk menunjukkan bagaimana pemakai sistem memahami kerja sistem.
c. Simpanan data dapat menunjukkan simpanan non komputer.
d. Nama dari simpanan data harus menunjukkan tipe penerapannya apakah secara manual atau komputerisasi. Secara manual misalnya dapat menunjukkan buku catatan, meja pekerja. Sedang cara komputerisasi misalnya menunjukkan *file* urut, *file database*.
e. Proses harus menunjukkan nama dari pemroses, yaitu orang, departemen, sistem komputer, atau nama program komputer yang mengakses proses tersebut.

2. Diagram Alur Data Logika (DADL)

DADL lebih tepat digunakan untuk menggambarkan sistem yang akan diusulkan (sistem yang baru). Untuk sistem komputerisasi, penggambaran DADL hanya menunjukkan kebutuhan proses dari sistem yang diusulkan secara logika, biasanya proses-proses yang digambarkan hanya merupakan proses-proses secara komputer saja.



Gambar 20. Konsep Alur DADF dan DADL

D. Syarat-syarat Membuat Data *Flow Diagram*

Syarat pembuatan DFD ini akan menolong profesional sistem untuk menghindari pembentukan DFD yang salah atau DFD yang tidak lengkap atau tidak konsisten secara logika. Beberapa syarat pembuatan DFD dapat menolong profesional sistem untuk membentuk DFD yang benar, menyenangkan untuk dilihat dan mudah dibaca oleh pemakai.

Syarat-syarat pembuatan DFD ini adalah:

- Pemberian nama untuk tiap komponen DFD
- Pemberian nomor pada komponen proses
- Penggambaran DFD sesering mungkin agar enak dilihat
- Penghindaran penggambaran DFD yang rumit
- Memastikan DFD yang dibentuk itu konsisten secara logika

1. Pemberian Nama untuk Tiap Komponen DFD

Seperti yang telah dijelaskan sebelumnya, komponen terminator mewakili lingkungan luar dari sistem, tetapi mempunyai pengaruh terhadap sistem yang sedang di-kembangkan ini. Maka agar pemakai mengetahui dengan lingkungan mana saja sistem mereka berhubungan, komponen terminator ini harus diberi nama sesuai dengan lingkungan luar yang mempengaruhi sistem ini. Biasanya komponen terminator diberi nama dengan kata benda.

Selanjutnya adalah komponen proses. Komponen proses ini mewakili fungsi sistem yang akan dilaksanakan atau menunjukkan bagaimana fungsi sistem dilaksanakan oleh seseorang, sekelompok orang atau mesin. Maka sangatlah jelas bahwa komponen ini perlu

diberi nama yang tepat, agar siapa yang membaca DFD khususnya pemakai akan merasa yakin bahwa DFD yang dibentuk ini adalah model yang akurat.

Pemberian nama pada komponen proses lebih baik menunjukkan aturan-aturan yang akan dilaksanakan oleh seseorang dibandingkan dengan memberikan nama atau identitas orang yang akan melaksanakannya. Ada dua alasan

mengapa bukan nama atau identitas orang (yang melaksanakan fungsi sistem) yang digunakan sebagai nama proses, yaitu:

a. Orang tersebut mungkin diganti oleh orang lain saat mendatang, sehingga bila tiap kali ada pergantian orang yang melaksanakan fungsi tersebut, maka sistem yang dibentuk harus diubah lagi.
b. Orang tersebut mungkin tidak melaksanakan satu fungsi sistem saja, melainkan beberapa fungsi sistem yang berbeda. Daripada menggambarkan beberapa proses dengan nama yang sama tetapi artinya berbeda, lebih baik tunjukkan dengan tugas/fungsi sistem yang sebenarnya akan dilaksanakan.

Karena nama untuk komponen proses lebih baik menunjukkan tugas/fungsi sistem yang akan dilaksanakan, maka lebih baik pemberian nama ini menggunakan kata kerja transitif.

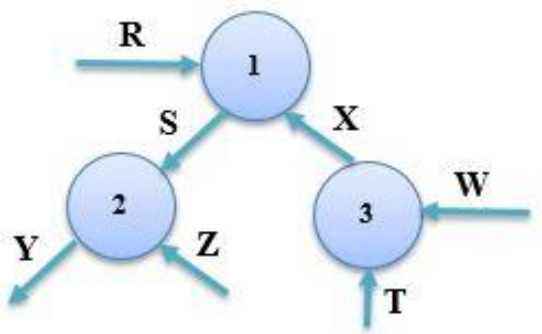
Pemberian nama untuk komponen data *store* menggunakan kata benda, karena data *store* menunjukkan data apa yang disimpan untuk kebutuhan sistem dalam melaksanakan tugasnya. Jika sistem sewaktu waktu membutuhkan data tersebut untuk melaksanakan tugasnya, maka data tersebut tetap ada, karena sistem menyimpannya.

Begitu pula untuk komponen alur data, namanya lebih baik diberikan dengan menggunakan kata benda. Karena alur data ini menunjukkan data dan informasi yang dibutuhkan dan yang dikeluarkan oleh sistem dalam pelaksanaan tugasnya.

2. Pemberian Nomor Pada Komponen Proses

Biasanya profesional sistem memberikan nomor dengan bilangan terurut pada komponen proses sebagai referensi. Tidak jadi masalah bagaimana nomor-nomor proses ini diberikan. Nomor proses dapat diberikan dari kiri ke kanan, atau dari atas ke bawah, atau dapat pula dilakukan dengan pola-pola tertentu selama pemberian nomor ini tetap konsisten pada nomor yang dipergunakan.

Nomor-nomor proses yang diberikan terhadap komponen proses ini tidak dimaksudkan bahwa proses tersebut dilaksanakan secara berurutan. Pemberian nomor ini dimaksudkan agar pembacaan suatu proses dalam suatu diskusi akan lebih mudah dengan hanya menyebutkan prosesnya saja jika dibandingkan dengan menyebutkan nama prosesnya, khususnya jika nama prosesnya panjang dan sulit.



Gambar 21. Contoh Pemberian Nomor Komponen Proses

Maksud pemberian nomor pada proses yang lebih penting lagi adalah untuk menunjukkan referensi terhadap skema penomoran secara hierarki pada levelisasi DFD. Dengan kata lain, nomor proses ini merupakan dasar pemberian nomor pada levelilasi DFD (lihat gambar Levelisasi DFD).

3. Penggambaran DFD Sesering Mungkin

Penggambaran DFD dapat dilakukan berkali-kali sampai secara teknik DFD itu benar, dapat diterima oleh pemakai, dan sudah cukup rapi sehingga profesional sistem tidak merasa malu untuk menunjukkan DFD itu kepada atasannya dan pemakai.

Dengan kata lain, penggambaran DFD ini dilakukan sampai terbentuk DFD yang enak dilihat, dan mudah dibaca oleh pemakai dan profesional sistem lainnya. Keindahan penggambaran DFD tergantung pada standar-standar yang diminta oleh organisasi tempat profesional sistem itu bekerja dan perangkat lunak yang dipakai oleh profesional sistem dalam membuat DFD.

Penggambaran yang enak untuk dilihat dapat dilakukan dengan memperhatikan hal-hal berikut ini:

a. Ukuran dan Bentuk Proses

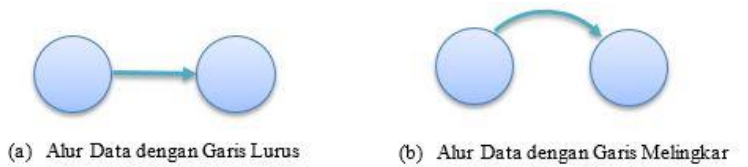
   Beberapa pemakai kadang-kadang merasa bingung bila ukuran proses satu berbeda dengan proses yang lain. Mereka akan mengira bahwa proses dengan ukuran yang lebih besar akan diduga lebih penting dari proses yang lebih kecil. Hal ini sebenarnya hanya karena nama proses itu lebih panjang dibandingkan dengan proses yang lain. Jadi, sebaiknya proses yang digambarkan memiliki ukuran dan bentuk yang sama.

b. Alur Data Melingkar dan Alur Data Lurus

   Alur data dapat digambarkan dengan melingkar atau hanya garis lurus. Mana yang lebih enak dipandang tergantung siapa yang akan melihat DFD tersebut.

c. DFD dengan Gambar Tangan dan Gambar Menggunakan Mesin

   DFD dapat digambarkan secara manual atau dengan menggunakan bantuan mesin, tergantung pilihan pemakai atau profesional sistem.



Gambar 22. Bentuk Alur Data

4. Hindari Penggambaran DFD yang Rumit

Tujuan DFD adalah untuk membuat model fungsi yang harus dilaksanakan oleh suatu sistem dan interaksi antar fungsi. Tujuan lainnya adalah agar model yang dibuat itu mudah dibaca dan dimengerti tidak hanya oleh profesional sistem yang membuat DFD, tetapi juga oleh pemakai yang berpengalaman dengan subyek yang terjadi. Hal ini berarti DFD harus mudah dimengerti, dibaca, dan menyenangkan untuk dilihat.

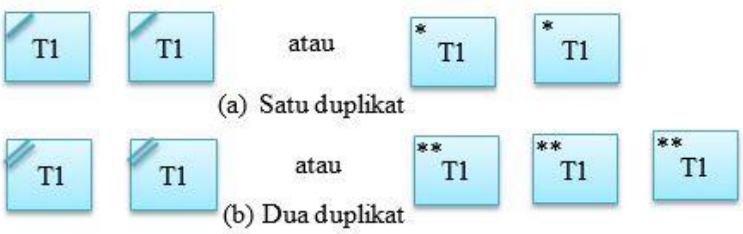
Pada banyak masalah, DFD yang dibuat tidak memiliki terlalu banyak proses (maksimal enam proses) dengan data *store*, alur data, dan terminator yang berkaitan dengan proses tersebut dalam satu diagram.

Bila terlalu banyak proses, terminator, data *store*, dan alur data digambarkan dalam satu DFD, maka ada kemungkinan terjadi banyak persilangan alur data dalam DFD tersebut. Persilangan alur data ini menyebabkan pemakai akan sulit membaca dan mengerti DFD yang terbentuk. Jadi semakin sedikit adanya persilangan data pada DFD, maka makin baik DFD yang dibentuk oleh profesional sistem.

Persilangan alur data ini dapat dihindari dengan meng-gambarkan DFD secara bertingkat-tingkat (levelisasi DFD), atau dengan menggunakan pemakaian duplikat ter-hadap komponen DFD.

Komponen DFD yang dapat menggunakan duplikat hanya komponen *store* dan terminator. Pemberian duplikat ini juga tidak dapat diberikan sesuka profesional sistem yang mem-buat DFD, tetapi makin sedikit pemakaian duplikat, makin baik DFD yang terbentuk.

Pemberian duplikat terhadap data *store* dilakukan dengan memberikan simbol garis lurus (|) atau asterik (*), sedangkan untuk terminator menggunakan simbol garis miring (/) atau asterik (*). Banyaknya pemberian simbol duplikat pada duplikat yang digunakan tergantung banyaknya duplikat yang digunakan.



Gambar 23. Contoh Pemakaian Simbol Duplikat pada Terminator

5. Penggambaran DFD secara Konsisten

Penggambaran DFD harus konsisten terhadap kelompok DFD lainnya. Profesional sistem menggambarkan DFD berdasarkan tingkatan DFD dengan tujuan agar DFD yang dibuatnya itu mudah dibaca dan dimengerti oleh pemakai sistem. Hal ini sesuai dengan salah satu tujuan atau syarat membuat DFD.

6. Penggambaran DFD

Tidak ada aturan baku untuk menggambarkan DFD. Tapi dari berbagai referensi yang ada, secara garis besar langkah untuk membuat DFD adalah:

a. Identifikasi terlebih dahulu semua entitas luar yang terlibat di sistem.

b. Identifikasi semua *input* dan *output* yang terlibat dengan entitas luar.

c. Buat Diagram Konteks (*Diagram Context*)

   Diagram ini adalah diagram level tertinggi dari DFD yang menggambarkan hubungan sistem dengan lingkungan luarnya.

   Caranya:

   1) Tentukan nama sistemnya.
   2) Tentukan batasan sistemnya.
   3) Tentukan terminator apa saja yang ada dalam sistem.
   4) Tentukan apa yang diterima/diberikan terminator dari/ke sistem.
   5) Gambarkan diagram konteks.

d. Buat Diagram *Level Zero*

   Diagram ini adalah dekomposisi dari diagram konteks. Caranya:

   1) Tentukan proses utama yang ada pada sistem.
   2) Tentukan apa yang diberikan/diterima masing-masing proses ke/dari sistem sambil memperhatikan konsep keseimbangan (alur data yang keluar/masuk

dari suatu level harus sama dengan alur data yang masuk/keluar pada level berikutnya).

3) Apabila diperlukan, munculkan data *store* (master) sebagai sumber maupun tujuan alur data.
4) Gambarkan diagram level zero.
    a) Hindari perpotongan arus data
    b) Beri nomor pada proses utama (nomor tidak menunjukkan urutan proses).

e. Buat Diagram Level Satu

Diagram ini merupakan dekomposisi dari diagram level zero.

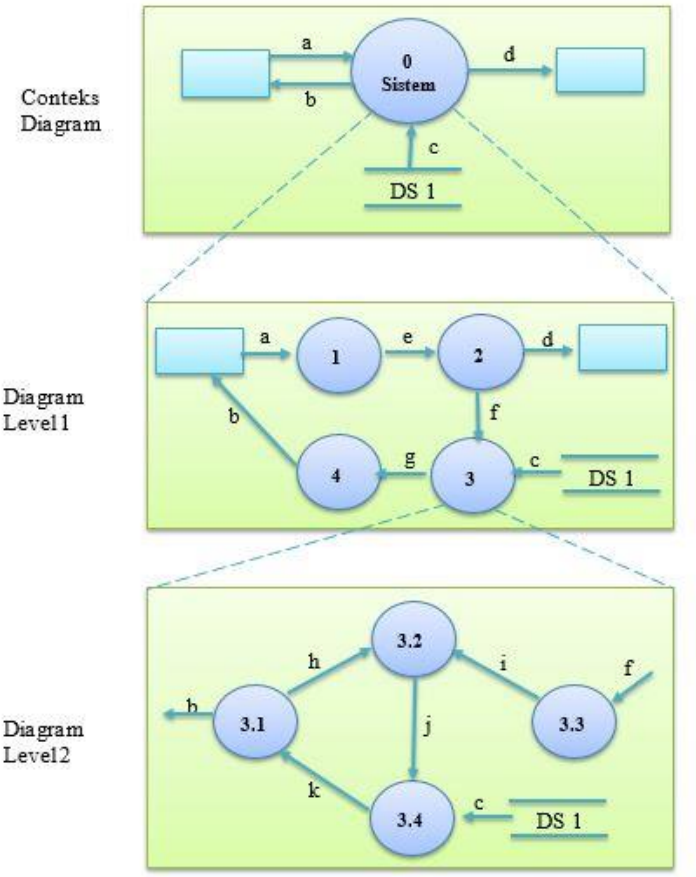
Caranya:

1) Tentukan proses yang lebih kecil (sub-proses) dari proses utama yang ada di *level zero*.
2) Tentukan apa yang diberikan/diterima masing-masing sub-proses ke/dari sistem dan perhatikan konsep keseimbangan.
3) Apabila diperlukan, munculkan data *store* (transaksi) sebagai sumber maupun tujuan alur data.
4) Gambarkan DFD level Satu
    a) Hindari perpotongan arus data.
    b) Beri nomor pada masing-masing sub-proses yang menunjukkan dekomposisi dari proses sebelumnya.

    Contoh: 1.1, 1.2, 2.1

f. DFD Level Dua, Tiga, …

Diagram ini merupakan dekomposisi dari level sebelumnya. Proses dekomposisi dilakukan sampai dengan proses siap dituangkan ke dalam program. Aturan yang digunakan sama dengan level satu.

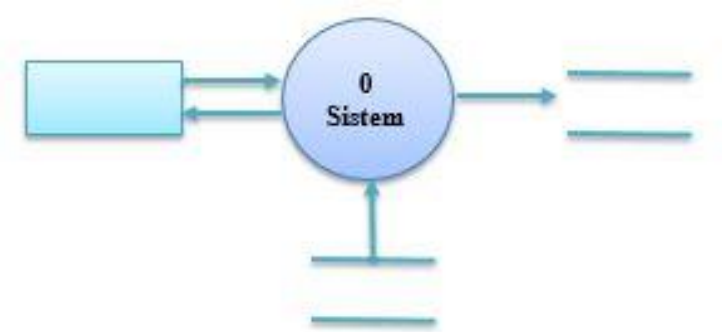


Gambar 24. Levelisasi DFD

Pada gambar di atas (Levelisasi DFD) terlihat bahwa Proses 0 diuraikan lagi ke dalam empat proses, penguraian ini digambarkan pada diagram Figure 0, sedangkan Proses 2 diuraikan kembali menjadi tiga proses yang digambarkan pada diagram Figure 2. Penguraian ini juga diikuti oleh alur data yang berkaitan dengan tiap proses yang diuraikan. Alur data yang berkaitan dengan tiap proses yang diuraikan dikenal dengan Alur data global. Jadi pada *balancing* DFD yang perlu diperhatikan adalah jumlah alur data global pada suatu level harus sama pada level berikutnya.

Ada beberapa hal yang perlu diperhatikan dalam penggambaran levelisasi DFD, yaitu:

1) Dalam diagram konteks, ada beberapa hal yang perlu diperhatikan seperti hubungan sistem dengan dunia luar yang mempengaruhinya, penggambaran sistem dalam satu proses, dan penggambaran data *store* (*optional*) yang dikenal dengan data *store* eksternal atau data *store* master. Data *store* eksternal ini maksudnya adalah data *store* itu dihasilkan oleh sistem yang sedang dianalisis, tetapi digunakan oleh sistem lain, atau data *store* itu dihasilkan oleh sistem lain tetapi digunakan oleh sistem yang sedang dianalisis.
2) *Balancing* (keseimbangan) dalam penggambaran levilisasi DFD perlu diperhatikan. *Balancing* DFD ini maksudnya keseimbangan antara alur data yang masuk/keluar dari suatu level harus sama dengan alur data yang masuk/keluar pada level berikutnya (lihat gambar Levelisasi DFD).



Gambar 25. Contoh Diagram Konteks

6

PEMODELAN PERANGKAT LUNAK BERORIENTASI *OBJECT*

Pemodelan perangkat lunak disusun berdasar pada spesifikasi kebutuhan perangkat lunak. Selanjutnya untuk keperluan pemodelan dalam buku ini menggunakan *Unified Modeling Language* (UML), khususnya pemodelan yang berorientasi pada obyek dengan menggunakan metode *Usecase Driven Object* proses ICONIX.

A. UML (*Unified Modeling Language*)

UMLadalah sebuah bahasa yang berdasar pada grafik/gambar untuk memvisualisasi, menypesifikasikan, membangun, dan pendokumentasian dari sebuah sistem pengembangan *software* berbasis OO (*Object-Oriented*).

UML tidak hanya merupakan sebuah 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.*

Salah satu dari tahapan SDLC atau *software development life* cycle, adalah desain. Desain bertujuan agar *software* yang akan dibuat dapat memenuhi kebutuhan *user* dan handal. Oleh karena itu, desain menjadi tahapan penting dalam proses pembuatan *software*. Dalam mendesain *software*, kita perlu mentrans-formasikan kebutuhan *user*, baik secara fungsional maupun non fungsional ke dalam model.

Model adalah sebuah abstraksi dari hal nyata. Model merupakan penyederhanaan dari sistem yang sebenarnya sehingga desain dari sebuah sistem dapat dimengerti oleh pihak lain. Untuk memodelkan sesuatu, tentu diperlukan bahasa pemodelan. Bahasa pemodelan dapat berupa pseudo-code, code, gambar, diagram, atau deskripsi yang menggambarkan sebuah sistem. Di sinilah UML berperan sebagai bahasa pemodelan.

Mengapa Harus Menggunakan UML?

Jawabannya adalah untuk menghindari ambiguitas, penggunaan kata yang terlalu banyak dan detail yang tidak penting (jika pemodelan menggunakan kode maupun *pseudocode*) sehingga menyebabkan kesalahpahaman.

Pemodelan menggunakan UML memiliki keuntungan, yaitu:

- UML adalah bahasa formal. Setiap elemen dari UML memiliki makna tersendiri sehingga tidak akan terjadi kesalah-pahaman.
- *UML singkat/ringkas. UML memiliki notasi yang jelas dan tidak berbelit-belit.
- UML komprehensif atau menyeluruh. UML dapat menggambarkan aspek-aspek penting dalam sistem.
- UML dapat meng-*handle* sistem yang besar maupun kecil.
- UML memiliki banyak pengguna. Sehingga tersedia banyak tutorial tentang cara penggunaannya.
- UML adalah bahasa standar. Sehingga UML tidak terikat dengan produk tertentu.

Berikut merupakan daftar diagram UML dan penjelasannya:

- ***Use Case***: Menggambarkan tentang interaksi antara sistem dan pengguna atau sistem lain. Sangat berguna untuk memetakan *requirement* atau kebutuhan dari suatu sistem.
- ***Activity***: Menggambarkan tentang aktivitas sekuensial maupun paralel dari sistem.
- ***Class***: Menggambarkan *class* , tipe, *interface* dan hubungan antar ketiganya.

- ***Object***: Menggambarkan *instance* objek dari *class* yang telah didefinisikan di *class* diagram.
- ***Sequence***: Menggambarkan interaksi antar objek di mana urutan dari interaksi tersebut merupakan hal penting.
- ***Communication***: Menggambarkan cara objek berinteraksi dan koneksi yang dibutuhkan untuk melakukan interaksi.
- ***Timing***: Menggambarkan interaksi antar objek di mana waktu merupakan hal yang penting.
- ***Component***: Menggambarkan komponen penting dalam sistem dan *interface* yang digunakan untuk saling berinteraksi.
- ***Package***: Menggambarkan hierarki dari sekelompok *class* dan component.
- ***State Machine***: Menggambarkan status dari objek selama masa aktifnya dan *event* yang dapat mengubah *state* dari objek tersebut.
- ***Deployment***: Menggambarkan bagaimana sistem dideploy di dunia nyata.

Secara umum, UML digunakan sebagai berikut:

- **Sketsa:** UML hanya digunakan untuk pembuatan sketsa yang menyampaikan poin-poin penting.
- ***Blueprint*:** UML digunakan untuk menyediakan spesifikasi dari sistem menggunakan UML diagram. Diagram ini kemudian dapat di-*generate* menggunakan UML *tool*. Pendekatan ini umumnya berkaitan dengan *software* dan biasanya menggunakan *forward* atau *reverse engineering* untuk menjaga model tetap sinkron dengan kode.
- **Bahasa pemrograman:** UML digunakan langsung dari model menjadi *executable code* (tidak hanya sebagian *code* seperti pada *forward engineering*). Hal ini berarti setiap aspek detail dari sistem akan dimodelkan. Keuntungan dari pendekatan ini adalah model dapat digunakan untuk meng-*generate code* ke berbagai *environtment*.

Pendekatan yang akan dipakai tergantung pada tipe aplikasi dan metode dalam proses pengembangan *software* yang digunakan (ex: *waterfall, interation*, dll.).

Sebagai contoh, pada industri tertentu seperti kesehatan, proyek *software* menggunakan pendekatan UML sebagai *blue-print*. Hal ini karena *software* harus berkualitas tinggi. Kualitas merupakan hal penting pada industri ini karena kesalahan kecil dapat menyebabkan efek yang fatal.

Seperti yang telah disebutkan, metode dalam proses pengembangan *software* juga berpengaruh pada pendekatan yang akan digunakan dalam UML.

Berikut merupakan metode yang sering digunakan dalam pengembangan *software*.

B. *Iconix Process*

ICONIX Process adalah metode pengembangan perangkat lunak yang berorientasi pada arsitektur. Metode ICONIX Process berada di antara *Rational Unified Process* (RUP) dan *Extreme Programing* (XP) Sebagai arsitektur sistem ICONIX Process konsentrasinya berada pada desain model.

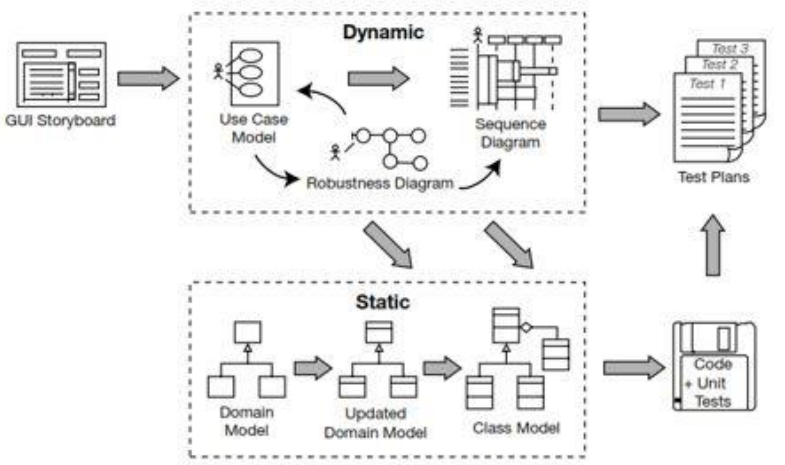
ICONIX Process digunakan sebagai panduan untuk membantu dalam pembangunan serta perancangan Web. Dalam penggunaan UML pada ICONIX Process pun tidak berlebihan, karena hanya membutuhkan beberapa diagram saja yang dianggap sudah cukup untuk melakukan analisa perancangan berbasis objek. Hingga pemodelan menjadi sebuah Web dengan analisa perancangan berbasis objek menggunakan ICONIX Process yang digambarkan pada gambar di bawah ini.

1. *Functional Requirement* Analysis. Tahap pertama dilakukan analisis kebutuhan fungsional secara detail untuk menentukan fungsi-fungsi yang akan tersedia pada perangkat lunak.
2. Domain Model Pada tahap kedua ini adalah membuat model dengan domain diagram di mana domain model ini menjadi awal dari permodelan secara statis yang menggambarkan

obyek-obyek yang saling berkaitan dengan aplikasi dan dunia nyata. Dari kebutuhan fungsional dan domain model yang telah terbentuk diharapkan pembangunan perangkat lunak tetap berada pada ruang lingkupnya, terwujud dengan baik dan jelas.

3. *Story Board The User Interface.* ICONIX Process menganjurkan untuk membuat desain antar muka pengguna atau *GUI Story Board* yang membantu memudahkan pemodelan *Use Case*.
4. *Use Case Model*. Diagram ini merupakan tahapan di mana membentuk sebuah diagram yang menggambarkan aktivitas yang dapat dilakukan dan siapa saja aktor yang terlibat.
5. *Robustness Analysis*. Diagram ini menjadi tahap yang menggambarkan detail setiap kalimat dari skenario *use case* yang sudah terbentuk, setiap *use case* memiliki skenario yang berbeda beda.

   Langkah *Robustness analysis* menjadi jembatan antara *use case* yang menggambarkan apa yang dapat dilakukan dengan perangkat lunak yang di bangun dan bagaimana desain yang dirancang untuk skenario setiap fungsi aplikasi.
6. *Sequence* Diagram. Diagram ini menggambarkan secara rinci proses yang terjadi sesuai objek dengan skenarionya.
7. *Updated Domain Model*. Pada tahap ini memastikan apakah desain *robustness* dan *sequence* diagram menampilkan objek baru atau terdapat objek yang hilang. Domain model akan mengalami perubahan jika terdapat objek yang seharusnya dihilangkan atau di tambahkan.
8. *Class model* diagram. Diagram ini adalah gambaran mengenai database yang saling berkaitan. *Class* diagram memiliki informasi yang lebih detail karena mempunyai atribut dan operasi-operasi yang di dapatkan setelah menggambarkan *sequence* diagram untuk masing-masing *use case*.



Gambar 26. *Block Diagram Iconix Process* pada *Use Case Driven Object*

Dari kedelapan tahap di atas, akan dijelaskan lebih rinci dengan menggunakan contoh kasus pembuatan aplikasi "Entry KRS" sebagai berikut:

Functional Requirement Analysis

Aplikasi Entry KRS adalah aplikasi yang digunakan oleh "Mahasiswa" dalam memilih dan memprogram Matakuliah yang akan ditempuh pada semester aktif.

Berikut tahapan proses pada aplikasi Entry KRS secara terurut:

Pihak Program Studi (Prodi):

- Menentukan Matakuliah yang ditawarkan
- Mengajukan permintaan kesediaan mengajar pada Dosen
- Melakukan ploting dosen matakuliah
- Membuat Jadwal Perkuliahan dan memperbaiki
- Memberikan informasi jadwal perkuliahan kepada Dosen
- Memberikan Surat Tugas Mengajar kepada Dosen

Dosen:

- Menyerahkan kesediaan mengajar matakuliah
- Menerima dan memverifikasi jadwal mengajar

Mahasiswa:

- Melihat dan memverifikasi Mata Kuliah yang ditawarkan
- Melihat Jadwal Perkuliahan
- Melakukan Entry Mata Kuliah yang di program (Entry KRS)
- Memverifikasi Kartu Studi Mahasiswa

C. Domain Model Diagram

Domain model adalah sebuah daftar proyek yang menunjukkan bagaimana semua berhubungan/relasi satu sama lain. Domain model menunjukkan hubungan agregasi dan generalisasi antara kelas domain.

Agregasi: relasi antara dua domain yang menyatakan bahwa domain asal memiliki domain tujuan

Simbol/Notasi relasi agregasi: 

Contoh:

Gambar 27. Relasi Agregasi Domain

Domain Mahasiswa mempunyai Domain KRS

Generalisasi: relasi antara dua domain yang menyatakan bahwa domain asal merupakan bagian dari domain tujuan

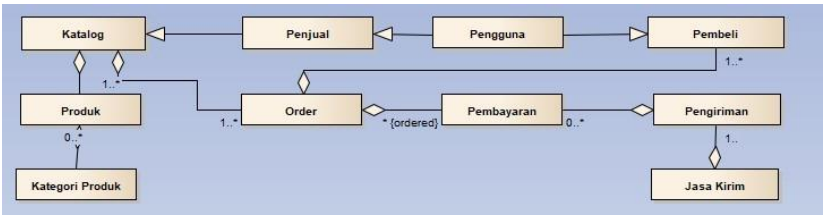
Simbol relasi Generalisasi: 

Contoh:

Gambar 28. Relasi Generalisasi Domain

Domain Matakuliah merupakan salah satu bagian dari Domain KRS

Contoh Domain Model E-Commerce Sederhana



Gambar 29. Contoh Domain Model E-Commerce Sederhana

D. GUI *Story Board*

GUI (*Graphical User Interface*) *Storyboard* adalah proses desain atau gambaran awal tampilan sistem yang dikembangkan. *Gui Sotryboard* dibuat berdasarkan fitur yang diperoleh dari kebutuhan fungsional yang telah dituliskan sebelumnya.

Contoh *GUI Story Board* Halaman *User Login*

GUI ini digunakan untuk semua pengguna melakukan login ke system. Login berhasil jika username dan password telah terdaftar serta user name dan password sesuai data yang tersimpan pada saat pendaftaran akun. Pada halaman ini pengguna juga dapat memilih link lupa password untuk merubah password karena lupa.

Gambar 30. Contoh GUI *Story Board* Halaman *User Login*

E. *Use Case Model Diagram*

Use Case diagram menggambarkan hubungan interaksi antara sistem dan aktor. *Use Case* dapat mendeskripsikan tipe interaksi antara si pengguna sistem dengan sistemnya.

Langkah awal untuk melakukan pemodelan, tentu perlunya suatu diagram yang mampu menjabarkan aksi aktor dengan aksi sistem itu sendiri, seperti yang terdapat pada *use case* diagram.

1. *Primary Use Case*

Primary Use Case digunakan untuk menggambarkan interaksi antar aktor dengan sistem secara keseluruhan dan relasi atau hubungan antar *use case*.

Adapun, fungsi dari *use case* diagram sebagai berikut:

a. Berguna memperlihatkan proses aktivitas secara urut dalam sistem.
b. Mampu menggambarkan proses bisnis, bahkan menampilkan urutan aktivitas pada sebuah proses.
c. Sebagai *bridge* atau jembatan antara pembuat dengan konsumen untuk mendeskripsikan sebuah sistem.

Manfaat dari *use case* di antaranya:

a. Menggunakannya sebagai kebutuhan verifikasi.
b. Menjadi gambaran *interface* dari sebuah sistem karena setiap sistem yang dibangun haruslah memiliki *interface* .
c. Mengidentifikasi siapa saja orang yang dapat berinteraksi dengan sistem, serta apa yang dapat dilakukan oleh sistem.
d. Memberikan kepastian mengenai kebutuhan sistem, sehingga tidak membingungkan.
e. Memudahkan proses komunikasi antara domain *expert* dan *end user.*

Komponen *Use Case* Diagram terdiri dari beberapa item yaitu:

a. Sistem

Sebuah sistem digambarkan ke dalam bentuk persegi. Fungsinya untuk membatasi *use case* dengan interaksi dari luar sistem. Sistem pada umumnya diberikan label yang sesuai. Namun, umumnya sistem ini tidaklah diberi gambar karena tidak terlalu memberikan makna pada sebuah diagram.

b. Aktor

Sebenarnya Aktor bukanlah bagian dari sistem, akan tetapi memiliki pengaruh besar terhadap sistem. Aktor memiliki peran yang sangat penting untuk menggambarkan siapa yang berinteraksi dengan sistem.

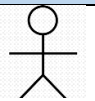
Aktor memberi dan menerima informasi dari sistem meskipun bukan sebagai kontrol.

c. *Use Case*

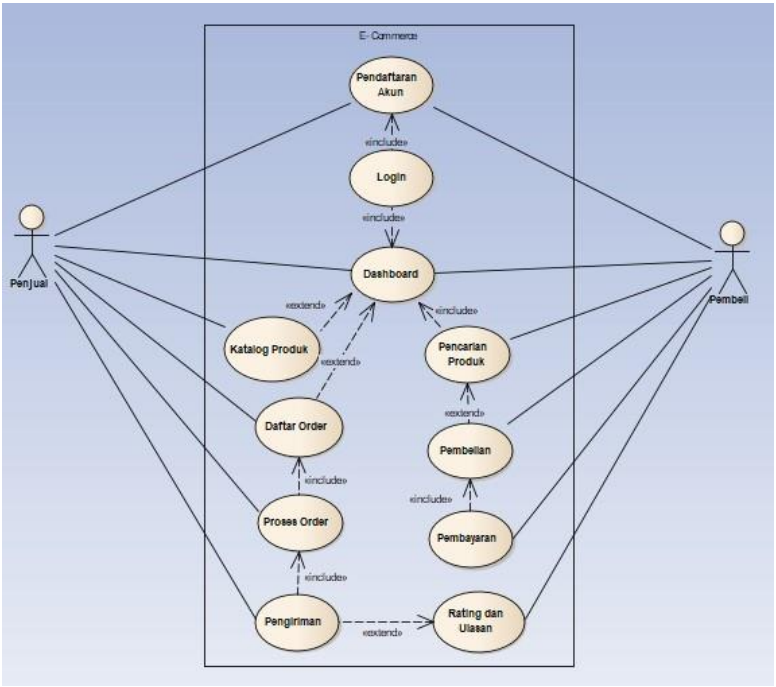
Use case adalah komponen yang menggambarkan fungsional dalam sebuah sistem. Sehingga konsumen maupun pembuat saling mengenal dan mengerti mengenai alur sistem yang akan dibuat.

Simbol-simbol *Use Case* Diagram

Tabel 11. Simbol-simbol Use Case Diagram

Simbol	Keterangan
	**Aktor**: mewakili peran orang, sistem yang lain, atau alat ketika berkomunikasi dengan *use case*
	***Use Case***: abstraksi atau interaksi antara system dengan aktor
	Association: abstraksi dari penghubung antara *actor* dengan *use case*
	Generalisasi: menunjukkan spesialisasi *actor* untuk dapat berpartisipasi dengan *use cae*
	Includes: menunjukkan bahwa suatu *use case* seluruhnya merupakan fungsionalitas dari *use case* lainnya
	Extends: menunjukkan bahwa suatu *use case* merupakan tambahan fungsional dari *use case* lainnya jika suatu kondisi terpenuhi

Berikut contoh *Use Case* Model *E-Commerce*:



Gambar 31. Contoh *Primary Use Case E-Commerce* Sederhana

Terdapat dua jenis *Use Case Diagram* yaitu *Use Case Primary* dan *Use Case* Narasi. *Use Case Primary* dibuat untuk menggambarkan secara keseluruhan dari Sistem yang dibuat, sehingga terlihat alur proses antara *actor* dengan masing-masing *use case* dan alur proses antar *use case*. Sedangkan *Use Case* Narasi dibuat untuk menggambarkan proses dari sebuah GUI.

2. *Use Case* Narasi

Use Case Narasi ini menggambarkan proses interaksi antara aktor (pengguna) dengan sebuah GUI/*Interface*/Form. Narasi yang terdapat pada *use case* terdiri dari tiga bagian:

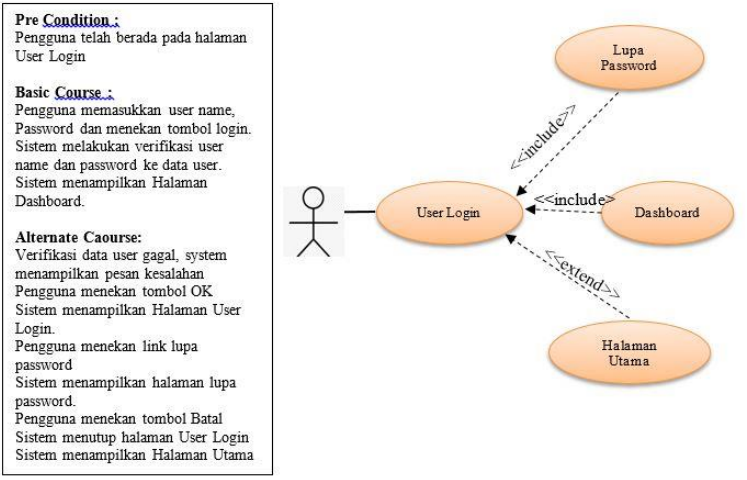
a. ***Pre Condition***, menjelaskan keadaan atau kondisi prasyarat GUI dijalankan

b. ***Basic Course/Basic Flow***, menjelaskan bagaimana proses/interaksi antara aktor dan GUI (Sistem) berjalan

seperti yang diharapkan.

c. ***Alternate Course/Alternate Flow***, menjelaskan bagaimana kemungkinan proses/interaksi alternatif antara *actor* dan GUI (Sistem)

Berikut contoh *use case* narasi dari GUI *User Login*

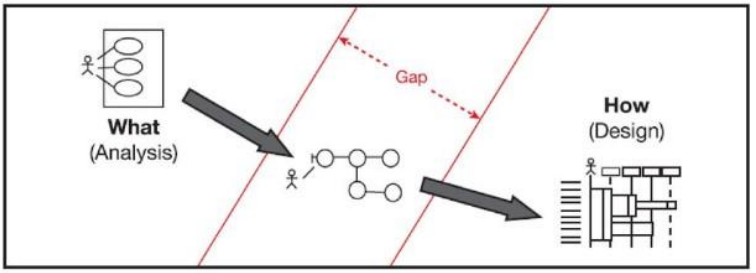


Gambar 32. Contoh *Use Case* Narasi Halaman *User Login*

F. *Robustness Analysis Diagram*

Robustness Diagram adalah diagram yang menggambarkan interaksi antara aktor dengan satu *use case* pada GUI. Perilaku yang di berikan *actor* sebagai *input* mendapatkan respon dari sistem sebagai *output*.

Robustness Diagram dapat dikatakan sebagai penghubung antara *Use Case* diagram dengan *Sequence* diagram sebagaimana terlihat pada gambar berikut:



Gambar 33. Posisi *Robustness* Diagram

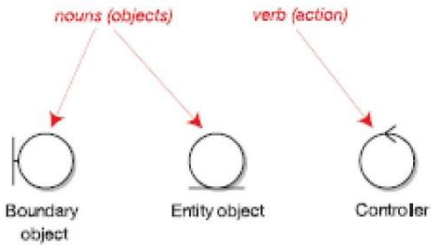
Struktur *Robustness* Diagram

Terdapat beberapa simbol yang ada di *Robustness* seperti terlihat pada tabel berikut:

Tabel 12. Simbol-simbol *Robustness* Diagram

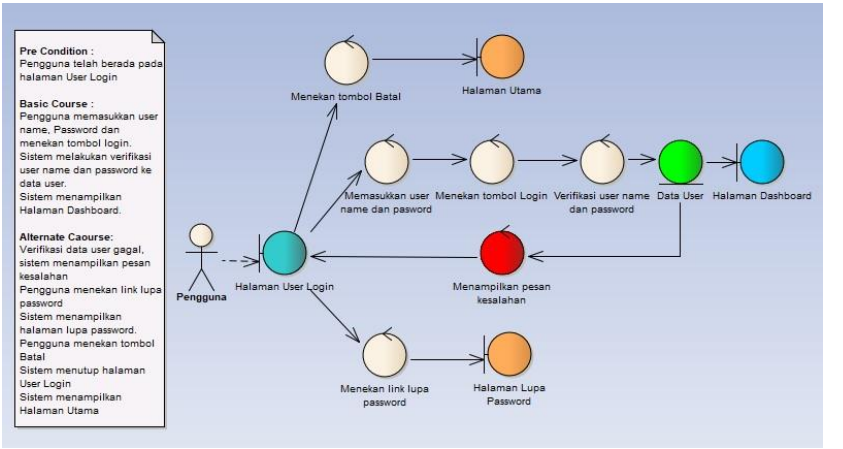
Simbol	Keterangan
Aktor 	Menggambarkan tokoh atau seseorang yang berinteraksi dengan sistem. Dan dapat menerima dan memberi informasi pada sistem.
Boundary 	Menggambarkan hubungan kegiatan yang akan dilakukan.
Entity 	Menggambarkan hubungan kegiatan yang akan dilakukan.
Controller 	Menggambarkan penghubung antara boundary dengan tabel
Message 	Mengindikasikan komunikasi antar objek.
Life Line	Mengindikasikan keberadaan sebuah objek dalam basis waktu.

Robustness Diagram juga memiliki aturan dalam pembuatannya di mana aktor hanya boleh berhubungan langsung dengan *boundary, boundary* dengan *controller, controller* dengan *controller, controller* dengan *entity*. Seperti yang terlihat gambar berikut ini:



Gambar 34. Struktur *Robustness* Diagram

Contoh Robustness Diagram Halaman *User* Login



Gambar 35. Contoh *Robustness* Diagram Halaman *User Login*

G. *Sequence Model Diagram*

Sequence diagram atau diagram urutan adalah sebuah diagram yang digunakan untuk menjelaskan dan menampilkan interaksi antar objek-objek dalam sebuah sistem secara terperinci. Selain itu *sequence digaram* juga akan menampilkan pesan atau perintah yang dikirim, beserta waktu pelaksanaannya. Objek-objek yang berhubungan dengan berjalannya proses operasi biasanya diurutkan dari kiri ke kanan.

Diagram *sequence* terdiri dari dua bagian, yaitu bagian vertikal yang menunjukkan waktu dan bagian horizontal yang menunjukkan objek-objek. Setiap objek, termasuk *actor*, memiliki waktu aktif yang digambarkan dengan kolom vertikal yang disebut dengan *lifeline*. Sedangkan pesan atau perintah digambarkan sebagai garis panah dari satu *lifeline* ke *lifeline* yang lain.

Diagram *sequence* dapat digunakan untuk menggambarkan serangkaian langkah yang dilakukan sebagai respon dari sebuah peristiwa untuk menghasilkan suatu *output* tertentu. *Sequence* diagram berhubungan dan berkaitan erat dengan *use case* diagram, di mana satu *use case* diagram akan menjadi satu diagram *sequence*.

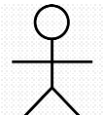
Tujuan *Sequence* Diagram

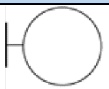
Tujuan utama pembuatan diagram *sequence* adalah untuk mengetahui urutan kejadian yang dapat menghasilkan *output* yang diinginkan. Selain itu, tujuan dari diagram *sequence* ini mirip dengan *activity* diagram, seperti menggambarkan alur kerja dari sebuah aktivitas, serta dapat menggambarkan aliran data dengan lebih detail, termasuk data atau perilaku yang diterima atau dikirimkan.

Komponen-komponen yang digunakan

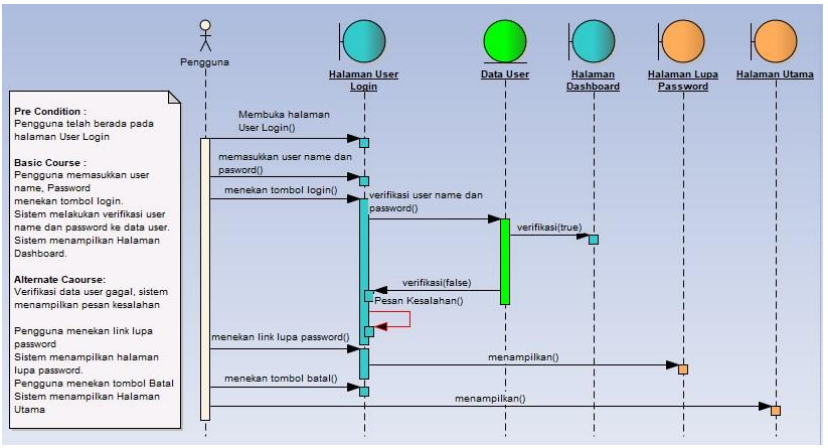
Berikut beberapa komponen utama yang sering digunakan:

Tabel 13. Simbol-simbol *Sequence* Diagram

Simbol	Keterangan
Aktor 	Komponen ini menggambarkan seorang pengguna (*user*) yang berada di luar sistem dan sedang berinteraksi dengan sistem. Dalam *sequence* diagram, aktor biasanya digambarkan dengan simbol *stick figure*.
Activation box	Komponen *activation box* ini merepresentasikan waktu yang dibutuhkan suatu objek untuk menyelesaikan tugasnya. Semakin lama waktu yang diperlukan, maka secara otomatis *activation box*-nya juga akan menjadi lebih panjang. Komponen ini digambarkan dengan bentuk persegi panjang.
Lifeline	Komponen ini digambarkan dengan bentuk garis putus-putus. *Lifeline* ini biasanya memiliki kotak yang berisi objek yang memiliki fungsi untuk menggambarkan aktivitas dari objek.
Objek 	Komponen objek ini digambarkan sesuai dengan *symbol* dan nama objek. Biasanya objek berfungsi untuk

Simbol	Keterangan
	mendokumentasikan perilaku sebuah objek pada sebuah sistem. (contoh: *Boundary, Entity,* Aktor)
Messages 	Komponen ini untuk menggambarkan komunikasi antar objek. *Messages* biasanya muncul secara berurutan pada *lifeline*. Komponen *messages* ini direpresentasikan dengan anak panah. Inti dari sebuah diagram urutan terdapat pada komponen *lifeline* dan *messages* ini.

Contoh *Sequence* Diagram Halaman *User* Login



Gambar 36. Contoh *Sequence* Diagram Halaman *User Login*

H. Updated Domain Model

Seperti halnya pada domain model, pada *update* domain model ini hanya digunakan jika terdapat perubahan pada *use case* narasi, *robustness* dan *sequence* diagram. Hal ini di perlukan karena akan mempengaruhi pada tahap selanjutnya yaitu pada *class* diagram.

I. *Class Model Diagram*

Class diagram atau diagram kelas adalah salah satu jenis diagram struktur pada UML yang menggambarkan dengan jelas

struktur serta deskripsi *class* , atribut, metode, dan hubungan dari setiap objek. Ia bersifat statis, dalam artian diagram kelas bukan menjelaskan apa yang terjadi jika kelas-kelasnya berhubungan, melainkan menjelaskan hubungan apa yang terjadi.

Diagram kelas ini sesuai jika diimplementasikan ke proyek yang menggunakan konsep *object-oriented* karena gambaran dari *class* diagram cukup mudah untuk digunakan.

Desain model dari diagram kelas ini sendiri dibagi menjadi dua bagian. Bagian pertama merupakan penjabaran dari database. Bagian kedua merupakan bagian dari modul MVC, yang memiliki *class interface, class control*, dan *class entity*.

Fungsi *Class Diagram*

Diagram kelas ini memiliki beberapa fungsi, fungsi utamanya yaitu menggambarkan struktur dari sebuah sistem. Berikut ini adalah fungsi-fungsi lainnya:

- Menunjukkan struktur dari suatu sistem dengan jelas.
- Meningkatkan pemahaman tentang gambaran umum atau skema dari suatu program.
- Dapat digunakan untuk analisis bisnis dan digunakan untuk membuat model sistem dari sisi bisnis.
- Dapat memberikan gambaran mengenai sistem atau perangkat lunak serta relasi-relasi yang terkandung di dalamnya.

Keunggulan

Menggunakan diagram kelas memberikan banyak keunggulan bagi proses pengembangan perangkat lunak dan dalam bisnis. Berikut ini adalah keunggulan dari diagram kelas:

- Diagram kelas berfungsi untuk menjelaskan suatu model data untuk sebuah program, baik model data sederhana maupun kompleks.
- Memberikan gambaran umum tentang skema aplikasi dengan jelas dan lebih baik.
- Membantu kamu untuk menyampaikan kebutuhan dari suatu

sistem.

Komponen Penyusun *Class Diagram*

Diagram kelas memiliki tiga komponen penyusun. Berikut ini adalah komponen-komponennya:

Gambar 37. Komponen Penyusun Class Diagram

- **Komponen Atas**

  Komponen ini berisikan nama *class*. Setiap *class* pasti memiliki nama yang berbeda-beda, sebutan lain untuk nama ini adalah *simple name* (nama sederhana).

- **Komponen Tengah**

  Komponen ini berisikan atribut dari *class*, komponen ini digunakan untuk menjelaskan kualitas dari suatu kelas. Atribut ini dapat menjelaskan dapat ditulis lebih detail, dengan cara memasukkan tipe nilai.

- **Komponen Bawah**

  Komponen ini menyertakan operasi yang ditampilkan dalam bentuk daftar. Operasi ini dapat menggambarkan bagaimana suatu *class* dapat berinteraksi dengan data.

Hubungan antar Kelas

Setelah kita mengetahui penjelasan tentang diagram kelas, sekarang kita akan membahas hubungan antar kelasnya. Ada tiga hubungan dalam diagram kelas. Berikut ini adalah penjelasannya:

- **Asosiasi**

  Pertama ada asosiasi. Asosiasi dapat diartikan sebagai hubungan antara dua *class* yang bersifat statis. Biasanya asosiasi menjelaskan *class* yang memiliki atribut tambahan seperti *class* lain.

Gambar 38. Relasi Asosiasi *Class* Diagram

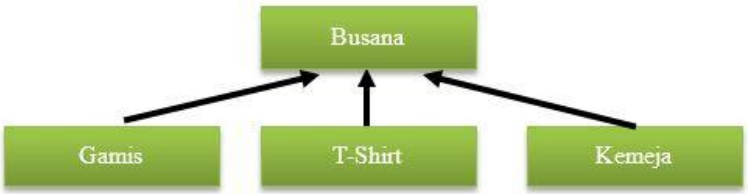
- **Agregasi**

  Agregasi adalah hubungan antara dua *class* di mana salah satu *class* merupakan bagian dari *class* lain, tetapi dua *class* ini dapat berdiri masing-masing.

Gambar 39. Relasi Agregasi *Class* Diagram

- **Generalisasi/Pewarisan**

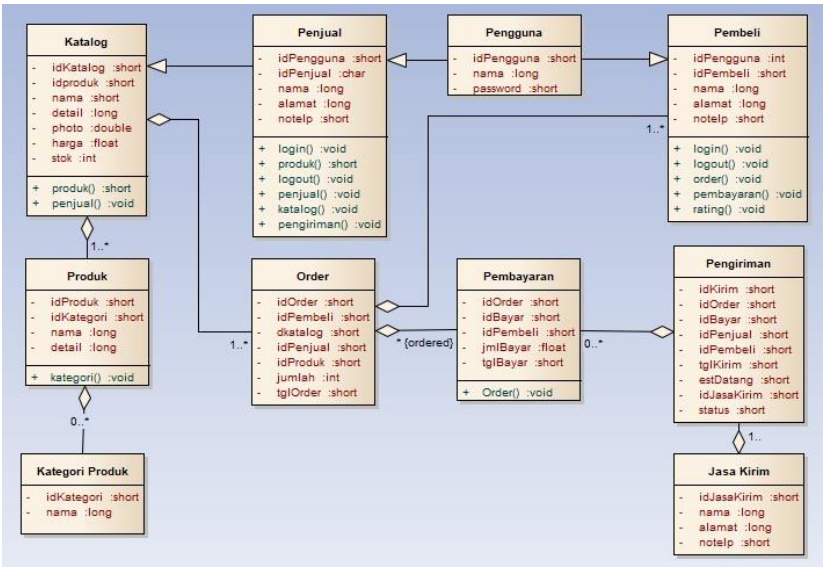
  Pewarisan atau *inheritance* dapat disebut juga *generalization* dalam *class* diagram adalah suatu kemampuan untuk mewarisi seluruh atribut dan metode dari *class* asalnya (*superclass*) ke *class* lain (*subclass*).



Gambar 40. Relasi Generalisasi *Class* Diagram

Contoh Penerapan

Berikut ini adalah contoh dari diagram *class* sistem *e-commerce* sederhana:



Gambar 41. Contoh *Class* Diagram *E-Commerce* Sederhana

7

USER INTERFACE DAN *USER EXPERIENCE*

A. Definisi *User Interface* (UI)

User Interface atau sering disebut UI, merupakan mekanisme komunikasi antara pengguna (*user*) dengan sistem pada sebuah perangkat lunak, baik berbasis *website, mobile, system* operasi ataupun perangkat lunak *interface hardware* (perangkat keras). Mekanisme itu disesuaikan dengan kebutuhan pengguna terhadap program yang dibangun atau dikembangkan. Cakupan UI meliputi tampilan fisik, penggunaan warna, tampilan animasi, hingga pola komunikasi suatu program dengan penggunanya.

Pada umumnya, seorang desainer UI akan membuat desain yang sekiranya memudahkan pengguna. Desain akan disesuaikan dengan tingkat kebutuhan dasar pengguna terhadap perangkat lunak tersebut. Hasil desain UI adalah perangkat lunak dengan segala fitur yang sesuai dengan kebutuhan pengguna dalam menggunakan perangkat lunak tersebut.

B. Definisi *User Experience* (UX)

Pengertian UX atau *User Experience* memang tidak terlalu jauh berbeda dengan UI. Perbedaannya terletak pada fokus utama hubungan komunikasi antara pengguna dengan programnya, yakni berfokus pada pengalaman penggunanya.

Seorang desainer UX akan merancang program aplikasi web atau *mobile* berdasarkan pengalaman dari pengguna atau *user* setelah menggunakan aplikasi web atau *mobile* tersebut. Dengan begitu *User experience* (UX) sesuai artinya dalam bahasa

Indonesia "pengalaman pengguna" adalah pengalaman yang diberikan *website* atau *software* kepada penggunanya agar interaksi yang dilakukan menarik dan menyenangkan. Kalau dulu aplikasi mempunyai *usability* yang bagus saja sudah cukup. Sekarang sebuah aplikasi juga harus memiliki *user experience* yang bagus.

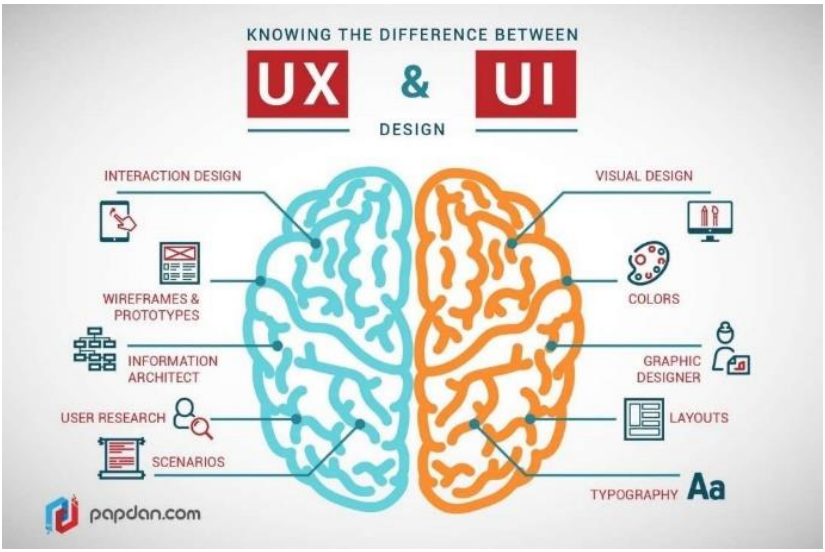
Seperti apa *user experience* itu? Saat membuka Instagram sampai berjam-jam tanpa bosan, saat *chatting* menggunakan WhatApp tanpa henti, saat berlama-lama mencari barang-barang jualan di toko *online*, berarti sudah menikmati *user experience* yang sudah diberikan oleh Instagram, WhatsApp dan juga toko *online*. Kenapa juga bisa berjam-jam sibuk dengan *smartphone*? Itu semua karena penerapan *user experience* dalam *smartphone* sudah sangat baik program yang dirancang menjadi lebih mudah digunakan oleh penggunanya.

C. Perbedaan Antara UI dan UX

Sebelumnya sudah disinggung bahwa perbedaan antara UI dan UX berapa pada fokus utamanya. Bila UI fokus pada interaksi pengguna dengan programnya, maka UX fokusnya pada pengalaman pengguna dalam menggunakan suatu aplikasi web atau *mobile*.

Seorang desainer UI akan mendesain program aplikasi web atau *mobile* sesuai dengan kebutuhan pengguna. Sehingga, ketika menggunakan program tersebut pengguna lebih mudah dan tidak kesulitan.

Sedangkan, desainer UX membuat program berdasarkan pengalaman dari penggunanya. Apa saja yang dirasakan dan kesulitan apa saja yang dihadapi ketika menggunakan program tersebut.



Gambar 42. Gambaran Perbedaan UI Dengan UX

Sebenarnya, keduanya memiliki tujuan yang sama dalam mendesain program aplikasi web ataupun *mobile*, yakni memudahkan penggunaannya. Oleh sebab itu, sering kali dalam proses perancangan sebuah program, desainer UI dan UX selalu berada dalam satu tim. Sebab, dengan perpaduan keduanya, sebuah program aplikasi web ataupun *mobile* menjadi sangat mudah digunakan oleh pengguna tanpa harus membaca panduan. Seringkali, desainer UI dan UX bertukar data analisis untuk menyempurnakan program yang tengah dibuatnya.

Jadi, UI dan UX sebenarnya berbeda. Perbedaannya pada fokus utama. UI fokus pada kebutuhan pengguna terhadap program aplikasi web atau *mobile*, sedangkan UX fokus pada pengalaman pengguna.

Pada dasarnya, *User Experience* adalah tentang "memahami pengguna". Tujuan UX adalah mencari tahu siapa mereka, apa yang mereka capai dan apa cara terbaik bagi mereka untuk melakukan "sesuatu".

UX berkonsentrasi pada bagaimana sebuah produk terasa dan apakah itu memecahkan masalah bagi pengguna.

Sedangkan *User Interface* adalah bagaimana suatu *website* atau aplikasi yang dibuat terlihat dan berbentuk seperti apa. Hal tersebut mencakup *Layout* (tata letak), Visual Design (desain visual) dan *Branding*.

Mengerti perbedaan antara UI Design dan UX Design, bukan sekedar untuk teori, tapi akan berpengaruh pada proses *design*. Beberapa orang (kalau bukan kebanyakan) menganggap *design* itu hanya terkait warna, pemilihan *font*, gambar/foto, dan *icon*. Padahal UX Design itu jauh melebihi warna dan sebagainya.

D. Desain UI

Terdapat dua hal yang harus diperhatikan oleh seorang *designer* saat membuat UI, yaitu *User* dan *Interface*.

User	***Interface***
• Mudah digunakan • Mudah dipahami • Tidak membingungkan • Jelas (antara tombol, link, bisa di klik atau tidak)	• Clean • Rapi • Modern • Keren • Kekinian • Colorfull • Tidak membosankan

Untuk menghasilkan UI (*User Interface*) yang baik, harus memperhatikan 2 hal di atas. Kata *User* ditempatkan di depan kata *Interface*, karena UI yang baik selalu memperhatikan dan mengutamakan *user*. UI yang baik akan membantu *user*. Dan, UI yang baik akan membuat *user* nyaman menggunakannya.

Untuk menjadi desainer UI, lebih diutamakan fokus pada bagian *Interface*. Dalam pembuatan UI, Anda harus memikirkan tentang bagaimana nanti *user* memakainya. Contohnya: Sebuah Tombol. Saat mendesain sebuah tombol, harus dipikirkan "Apakah *user* tahu kalau ini tombol yang bisa di-klik?". Tetapi, hal yang tidak kalah penting adalah harus diketahui bagaimana bentuk tombol itu. Maka dari itu, sebagai permulaan UI untuk fokus di

bagian *Interface* -nya terlebih dahulu agar familier dengan bentuk-bentuk UI.

Apakah familier dengan tool seperti Photoshop? GIMP? atau *software* sejenis lainnya? Jika belum, maka sebaiknya mulai belajar menggunakan *software* tersebut. Langkah pertama untuk menjadi UI Designer adalah mencoba untuk membuat UI. Tidak perlu bingung mau mulai dari mana atau mau mendesain apa. Cukup memilih desain-desain *interface* yang ada di internet. Pilih yang disukai lalu jiplaklah, buat semirip mungkin. Bisa *browsing* di situs seperti dribbble.com, behance.net atau kreavi.com. Banyak sekali desain UI di sana.

Kenapa harus menjiplak desain? Hal ini untuk membuat familier dengan UI. Bagaimana bentuk tombol, ukuran teks, jarak antar baris kalimat, jarak tiap elemennya.

Jadi, Semakin sering menjiplak sebuah desain, maka akan semakin familier dengan ukuran setiap elemen UI. Tapi ingat, jiplak menjiplak ini hanya untuk kepentingan **belajar**. **BUKAN** untuk di-*upload* di sosial media ataupun situs portfolio.

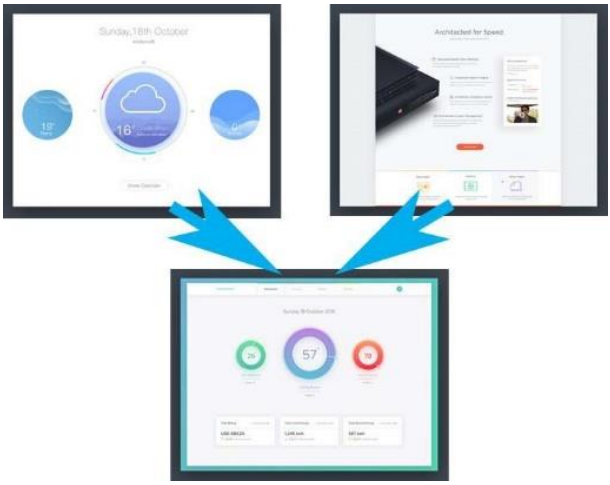
Jika telah menjiplak UI berkali-kali, maka sudah terbiasa membuat tombol, terbiasa membuat *dropdown, input text*, dan elemen-elemen UI lainnya. Kini, saatnya untuk membuat UI sendiri. Caranya gampang, Pertama, tentukan platform terlebih dahulu. Tentukan apa yang akan dibuat, *design* untuk *mobile app* (Android/iOS), *design* untuk web, atau bahkan *design* untuk *smartwatch app*?

Setelah itu, tentukan tema *design*-nya. Ingin membuat tampilan untuk aplikasi kesehatan? *website* artikel teknologi? atau tampilan untuk aplikasi pemesanan makanan? Setelah platform dan tema sudah dipilih, kini saatnya *browsing* lagi *design-design* yang disukai. Tapi kali ini, carilah *design* yang sesuai dengan platform dan tema yang dipilih.

Pilih dua *design* yang paling disukai, dan coba buat sesuatu yang baru dari hasil kombinasi dua *design* tersebut.

Saat mendesain sebuah UI, pikirkanlah bagaimana nanti *user* memakainya. Untuk melatih hal itu, biasakanlah mendesain

dengan sebuah alasan. Apa artinya? Design yang dibuat, komponen yang dibuat, sebaiknya memiliki alasan yang kuat kenapa diletakkan seperti itu atau berbentuk seperti itu.



Gambar 43. Contoh Desain UI Modifikasi dari Dua UI Jiplakan

Contoh 1:

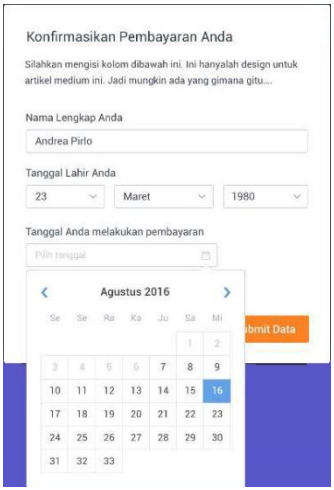
Mengapa menunya ada di bawah?

Mengapa aplikasi ini memiliki tiga menu utama. Dan dalam penggunaan aplikasi ini, *user* cenderung berpindah-pindah dari menu satu ke menu utama lain-nya dalam selang waktu relatif singkat. Maka dari itu, untuk mempermudah *user*, tiga menu utama ditaruh di bawah dan sisanya ditaruh di menu more yang terletak di paling kanan.

Mengapa ada tulisan yang tebal dan ada yang tidak tebal?

Ini adalah menu *Inbox*, tulisan yang tebal sebagai penanda belum dibaca. Dan yang tidak tebal sebagai penanda sudah pernah dibaca. Sehingga memudahkan *user* untuk membedakannya.

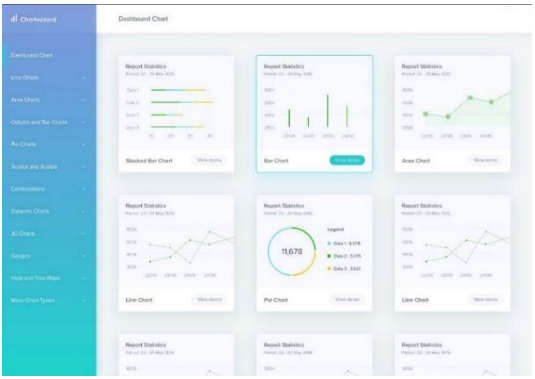
Contoh 2:



Terdapat dua input-an tanggal, tetapi mengapa beda? Padahal sama-sama memilih tanggal. Coba bayangkan jika menjadi *user*. Untuk mengisi tanggal lahir, lebih mudah memakai format pengisian yang atas atau yang bawah? Tentu lebih mudah memakai format yang atas. Karena, cukup mudah mengingat tanggal lahir. Dan dengan cepat akan mengisi hari, bulan dan tahunnya. Tapi, coba bayang-kan jika menggunakan format pengisian yang bawah, berapa kali harus mengklik tombol back untuk menuju bulan Maret 1980.

Kemudian untuk input-an tanggal pembayaran, mengapa format pengisiannya harus seperti itu? Ini adalah form konfirmasi pembayaran, setelah user melakukan pembayaran, biasanya user akan langsung mengkonfirmasikan pembayarannya. Coba bayangkan, jika user harus mengisi dengan format pengisian seperti tanggal lahir. Cukup merepotkan user, karena harus mengingat hari ini tanggal berapa. Tapi dengan format pengisian seperti di design, ketika user memilih tanggal, maka akan terbuka kalender bulan ini, dengan tanggal hari ini ditandai biru. Sehingga user dengan mudah memilih tanggal pembayarannya.

Contoh 3:



Mengapa menunya ada di kiri dan vertikal ke bawah? apakah karena ini *dashboard* yang tren-nya selalu meletakkan menu di sebelah kiri?

Meletakkan menu di kiri dan vertikal ke bawah, pertimbangannya adalah skalabilitas.

Sebelum merancang *design* ini, kami menganalisa bahwa ke depannya, menu di *dashboard* ini akan bertambah dan pertambahannya bisa cukup banyak. Jika membuat *design menu*-nya secara horizontal, maka kelak jika menunya sangat banyak, akan terjadi masalah pada *layout*-nya.

Tentu tidak diinginkan melihat dua baris menu terletak di bagian atas. Maka dari itu, diputuskan untuk membuat *design menu*-nya secara vertikal ke bawah. Jadi, jika ada pertambahan menu, tidak akan terjadi masalah pada *layout*-nya.

E. Pentingnya UX (*User Experience*)

- **Memudahkan Pengguna**

  Penerapan *user experience* akan memudahkan pengguna dalam menggunakan aplikasi. Karena di dalamnya sudah ada penilaian aspek *usability*. Setiap aplikasi pastilah dibuat agar para pengguna mudah untuk menggunakannya.

- **Menarik Minat Pengguna**

  Selain faktor kemudahan penggunaan, penerapan *user experience* juga untuk menarik minat pengguna. Tujuan aplikasi dibuat sudah pasti ingin penggunanya selalu menggunakan aplikasinya. Jika aplikasi tidak menarik untuk pengguna, sudah pasti bisa dengan mudah akan ditinggalkan.

- **Berdampak pada Faktor Kesuksesan**

  Seperti pada poin di atas, pengguna akan mudah meninggalkan aplikasi yang tidak memberikan pengalaman menarik. Contoh aplikasi *chatting* WhatsApp. Aplikasi ini makin hari makin banyak penggunanya karena memberikan pengalaman pengguna yang menarik. Kita bisa bandingan dengan aplikasi sejenis yang penggunanya semakin menurun setiap hari. Oleh karena itu, *user experience* penting

diterapkan pada sebuah aplikasi untuk meningkatkan kesuksesan atau minimal mempertahankan kesuksesan.

- **Menghasilkan UI yang Bagus**

  User interface itu merupakan keluaran dari penerapan *user experience*. Jika *user experience* pada sebuah aplikasi benar-benar diperhatikan penerapannya, maka akan menghasilkan desain UI yang bagus. Bagus di sini tidak harus warna-warni dan bling-bling, namun secara tampilan akan elegan dan menarik.

- **Untuk Memenangkan Persaingan**

  Apakah Anda mengetahui mengapa toko ritel seperti Indomaret dan Alfamart bisa memenangkan persaingan dari toko di sekelilingnya? Itu karena mereka menerapkan *user experience* yang bagus. Padahal barang yang dijual sama, secara harga bisa lebih mahal, tapi secara pengalaman yang mereka berikan itu tidak ada di toko biasa.

Contoh lain lagi yaitu antara Android dan IOS. Mengapa orang mau bayar mahal *smartphone* ini? Karena *user experience* yang mereka berikan memang terbaik. Berikut beberapa hasil penelitian mengenai pentingnya UX:

1. Menurut penelitian dari *Imaginovation*, sebuah lembaga penelitian berbasis di Amerika: Jika konten tidak dioptimalkan dengan baik, sebanyak 79% pengunjung akan keluar dari *website* tersebut dan mencari konten/produk lainnya.
2. Menurut penelitian dari lembaga riset *HubSpot*: pengguna ponsel 5X lebih punya kecenderungan untuk meninggalkan *website* jika *website* tidak dioptimalkan agar sesuai dengan perangkat yang mereka punya. (Berbahaya jika setidaknya ada 2/3 pelanggan yang akses *website* dari ponsel mereka sebenarnya ingin melakukan pembelian pada hari itu juga)
3. Menurut penelitian dari lembaga riset *MindTouch*: Ini kasus nyata, pendapatan dari *website* ESPN.com melonjak 35% setelah mereka mendengarkan keluhan pengguna mereka dan mendesain ulang *homepage* mereka.

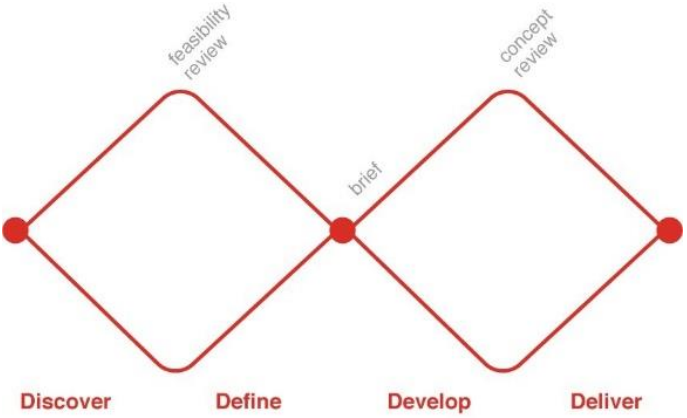
4. Menurut Adobe: 39% orang akan berhenti mengakses *website* jika gambar tidak dimuat-muat atau terlalu lama *loading*-nya.

Jadi intinya, *website* sekarang ini fungsinya mirip seperti toko. Bayangkan *user* sebagai seorang calon pembeli yang masuk ke toko Anda. Jika mereka mengalami pengalaman yang buruk, contohnya calon pembeli tidak bisa menemukan apa yang mereka butuhkan. Tak hanya itu, bisa juga pihak toko tidak berhasil menjangkau calon pembelinya, maka si calon pembeli pasti akan pergi dan tidak akan kembali lagi.

F. Dasar-dasar *User Experience* (UX)

- ***Double Diamond***

  Alur kerja ini didasarkan pada apa yang kita sebut dalam lingkaran UX sebagai double diamond, dipopulerkan oleh British Design Council, dengan tim Anda terbagi untuk memahami ide melalui penelitian dan kemudian berkumpul untuk mendefinisikan tantangan, membaginya untuk membuat sketsa secara individual, berbagi ide, memutuskan apa yang terbaik ke Model proses desain ''double diamond' yang dipelopori oleh British Design Council, langkah-langkahnya melibatkan tahapan proyek berikut; Memahami, Mendefinisikan, Membagi, Memutuskan, Prototipe dan Validasi. depannya, pengujian dan validasi.



Gambar 44. *Double Diamond*

G. Menyiapkan Langkah

Hal pertama adalah memulai dengan tantangan mendasar dan menulisnya seperti proposal, tanyakan pada diri sendiri, "apa masalah yang sesungguhnya yang akan dipecahkan?". Pernyataan tantangan adalah keterangan singkat yang ditetapkan ke proyek yang berisi tujuan.

Tantangan ini bisa fitur produk saat ini yang perlu disaring atau produk yang sama sekali baru. Apapun tugas Anda, cukup sesuaikan bahasa agar sesuai dengan tujuan yang ingin Anda capai. Pernyataan harus dikaitkan dengan tujuan tim Anda, berfokus pada pengguna, memberikan inspirasi dan ringkas.

Berikut adalah beberapa contoh produk nyata yang telah dikerjakan:

- Merancang sebuah sistem untuk mengelola pengobatan dan perawatan lanjutan pasien penderita *clubfoot*.
- Membuat sebuah aplikasi yang menyederhanakan sistem keuangan kompleks dan menguranginya ke hal-hal penting saja.
- Merancang aplikasi seluler yang konsisten di seluruh platform yang berbeda tanpa mengorbankan merek.

H. Memperbarui Pernyataan Tantangan

Setelah menuliskan beberapa variasi tujuan, presentasikan kepada tim untuk mendapatkan sebuah konsensus. Mungkin perlu memasukkan batas waktu karena ini akan membantu tim berfokus pada masalah. Jadi dengan penambahan tersebut, penyesuaian untuk daftar di atas bisa menjadi:

- Merancang sebuah sistem untuk mengelola pengobatan dan perawatan lanjutan anak-anak di bawah usia 2 tahun penderita *clubfoot* diluncurkan pada Q1 tahun ini.
- Membuat aplikasi keuangan sederhana yang memungkinkan pembelian dan menjual saham cukup dengan mengetuk tombol tanpa membutuhkan pengetahuan dasar dunia keuangan, dengan peluncuran awal Juli 2017.
- Menghasilkan panduan desain yang fleksibel di beberapa

platform dan memosisikan merek perusahaan secara efektif pada setiap platform hingga akhir tahun ini.

Ketika pernyataan tantangan selesai, tampilkan dalam tempat yang menonjol sehingga bisa dilihat saat bekerja. Harus memeriksanya kembali secara konstan, bahkan mungkin memperbarui atau memodifikasinya selama proyek berjalan.

I. Memvalidasi Masalah

Langkah berikutnya adalah meneliti tantangan dan mempelajari masalah tersebut. Apa yang perlu diketahui adalah apakah pemahaman tim tentang masalah adalah valid. Cukup sering kita melihat masalah dari sudut pandang kita sendiri, yang berbahaya karena kebanyakan dari kita di dunia teknologi sebenarnya adalah *power user* dan pada kenyataannya merupakan pengguna minoritas. Kita adalah minoritas vokal dan bisa tertipu saat berpikir sesuatu dapat menjadi masalah padahal tidak.

Ada berbagai metode pengumpulan data untuk memvalidasi tantangan. Masing-masing bergantung pada tim dan jika memiliki akses ke pengguna. Tujuannya adalah untuk mendapatkan pemahaman yang lebih baik dari masalah yang dihadapi.

J. Wawancara Internal

Wawancara dengan para pemangku kepentingan bisa informatif untuk menemukan wawasan dalam sebuah perusahaan atau tim. Proses wawancara termasuk melakukan wawancara kepada setiap anggota tim dan pemangku kepentingan di perusahaan, dari pemasaran hingga keuangan. Ini akan membantu untuk menemukan apa yang dipikir sebagai tantangan nyata dan apa solusi potensial yang bisa dipikirkan. Ketika memberikan solusi, maka tidak akan berbicara tentang solusi teknis, melainkan apa yang bisa menjadi skenario terbaik dan tujuan akhir bagi perusahaan atau produk. Misalnya menggunakan tantangan di atas "*memiliki software clubfoot di 80% fasilitas medis hingga akhir tahun ini*" dapat menjadi tujuan besar yang menjadi target.

Ada sebuah pengingat, metode validasi adalah yang paling tidak disukai karena menghambat diskusi dan kolaborasi tim,

berpotensi menciptakan suasana tertutup dalam sebuah organisasi. Meskipun demikian, ini bisa menghasilkan beberapa informasi bagus tentang klien dan tantangan desain yang bisa saja dilewatkan dengan para pemangku kepentingan

K. Presentasi Kilat

Presentasi kilat adalah presentasi sangat singkat yang hanya berlangsung beberapa menit. Mirip dengan wawancara internal, namun kali ini menghadirkan setiap pemangku kepentingan dalam satu ruangan. Kemudian Anda Memilih lima atau enam orang pemangku kepentingan (pemasaran, penjualan, desain, keuangan, penelitian dll.) untuk berbicara, masing-masing berfokus pada tantangan dari perspektif mereka selama maksimal 10 menit. Topiknya harus mencakup presentasi mereka:

- Tujuan bisnis
- Tantangan proyek dari sudut pandang pengguna/*customer* (ini bisa faktor teknis, pengumpulan penelitian, pembuatan desain dll.)
- Penelitian pengguna yang dimiliki saat ini

Berikan waktu 5 menit di akhir untuk sesi pertanyaan, dengan orang yang dipilih untuk mencatat semuanya. Setelah selesai, mungkin ingin memperbarui tantangan untuk merefleksikan pembelajaran yang baru. Tujuannya adalah untuk mengumpulkan daftar poin-poin utama yang bisa mendorong fitur atau alur yang membantu mencapai tujuan produk.

L. Wawancara Pengguna

Wawancara pengguna adalah cara yang bagus untuk mempelajari tentang titik derita orang di setiap tugas yang diberikan.

Ini mungkin adalah cara terbaik untuk belajar tentang pengalaman pengguna, titik derita, dan alur. Aturlah setidaknya lima wawancara pengguna, lebih banyak lagi jika Anda memiliki akses kepada mereka. Jenis pertanyaan yang Anda tanyakan kepada mereka harus mencakup:

- Bagaimana mereka menyelesaikan tugas yang ada? Misalnya, ingin menyelesaikan tantangan untuk aplikasi keuangan di atas, bisa ditanyakan kepada mereka "bagaimana untuk membeli saham dan efek saat ini?"
- Apa yang mereka suka tentang alur ini?
- Apa yang tidak mereka suka tentang alur ini?
- Apa produk sejenis yang saat ini digunakan pengguna?
- Apa yang mereka suka?
- Apa yang tidak mereka suka?
- Jika mereka memiliki tongkat ajaib dan bisa mengubah satu hal tentang proses ini hal apakah itu?

Ide melakukan wawancara adalah agar pengguna berbicara tentang tantangan yang mereka alami, bukan pada poin diskusi. Ketika pengguna berhenti berbicara, berikan waktu sebentar karena mereka bisa saja sedang mengumpulkan pemikirannya. Kemungkinan banyak orang yang akan terus berbicara setelah berhenti sejenak selama beberapa detik.

Catat seluruhnya dan jika mungkin rekam percakapan tersebut untuk membantu merekam apa pun yang mungkin terlewatkan. Tujuannya adalah membandingkan tantangan terhadap wawasan pengguna yang dikumpulkan. Apakah mereka selaras? Apakah dapat mempelajari sesuatu yang membantu memperbarui pernyataan tantangan?

M. Penelitian Bidang Etnografi

Melihat pengguna dalam lingkungan alami mereka adalah cara yang bagus untuk memahami bagaimana pengguna mengatasi tantangan mereka sendiri.

Ini adalah bidang tempat untuk mengamati pengguna, dalam konteks saat melakukan sesuatu seperti bagaimana mereka berbelanja, bagaimana mereka melakukan perjalanan ke tempat kerja, bagaimana mereka mengirim pesan SMS dll. Alasannya adalah karena dalam beberapa kasus orang akan memberi tahu apa yang mereka pikir ingin didengarkan. Namun jika melihat sendiri pengguna melakukan tindakan dan tugasnya, ini bisa

menjadi penuh wawasan. Pada dasarnya pengamatan tanpa mengganggu, mencatat hal-hal yang mereka rasa mudah atau sulit dan hal-hal yang mungkin mereka lewatkan. Tujuannya adalah untuk melibatkan diri dalam lingkungan pengguna agar lebih berempati dengan titik kelemahan mereka.

Teknik ini biasanya melibatkan beberapa pekerjaan yang dilakukan selama periode waktu yang lebih lama dan membutuhkan peneliti untuk memimpin bagian proyek ini. Namun inilah yang mungkin paling berwawasan karena bisa melihat sekelompok orang yang dipelajari di lingkungan alami mereka.

N. Mengumpulkan Semuanya

Setelah menyelesaikan tahap pembelajaran proyek, maka harus mengambil satu pemeriksaan terakhir pada tantangan. Apakah berada di jalur yang benar? Apakah ada sesuatu yang perlu disesuaikan? Tuliskan semua hal yang telah dipelajari dan kelompokkan mereka ke dalam kategori. Ini bisa menjadi dasar dari fitur atau alur, bergantung pada masalah yang akan diselesaikan. Juga bisa digunakan untuk memperbarui dan merevisi tantangan.

Setelah Anda memiliki masukan dan wawasan yang cukup, saatnya untuk menerapkan pengetahuan itu untuk membuat pemetaan proyek.

O. Pemetaan Proyek

Masalah yang coba diselesaikan biasanya terdiri dari berbagai tipe orang (atau pengguna), masing-masing dengan andil di alur proyek. Berdasarkan pembelajaran, perlu mendaftar para pengguna. Ini bisa jadi tipe pengguna atau pemangku kepentingan, misalnya, "*dokter yang merawat clubfoot*", "*pasien yang menderita clubfoot*", "*perawat yang merawat pasien*", dll. Tuliskan masing-masing pengguna di sisi kiri selembar kertas atau tulis pada papan tulis. Di sisi sebelah kanan, tuliskan tujuan masing-masing pengguna.

Gambar 45. Contoh Pemetaan Proyek Perangkat Lunak

Yang terakhir untuk setiap pengguna, tuliskan jumlah langkah yang diperlukan untuk mencapai tujuan mereka. Misalnya untuk "*dokter yang merawat clubfoot*" tujuannya adalah "*menyembuhkan pasien yang menderita clubfoot*", sehingga langkah-langkahnya adalah "*mendaftarkan pasien dalam sistem*", "*memulai rencana kesehatan*", "*membuat siklus ulasan kesehatan medis*" dan "*melakukan prosedur medis*".

Hasilnya adalah pemetaan proyek dengan langkah-langkah utama dalam prosesnya. Anggap saja itu sebagai ringkasan proyek tanpa terlalu banyak detail. Ini juga memungkinkan anggota tim menilai apakah pemetaan cocok dengan pernyataan tantangan. Kemudian, ketika membagi setiap langkahnya, akan ada detail lebih lanjut. Namun untuk saat ini, pemetaan proyek memberikan rincian tingkat tinggi dari langkah yang perlu diambil pengguna untuk menyelesaikan tujuan akhir mereka.

P. *Wireframing* dan *Storyboarding*

1. *Crazy* 8s

Crazy Eight adalah adalah latihan pembuatan ide desain yang menarik dalam waktu singkat, cara membuatnya yaitu menggunakan kertas yang dilipat menjadi 8 dan membuat sketsa menggunakan spidol atau pulpen dari ide yang mengalir selama 8 menit. Singkatnya, kita membuat ide untuk *design* kita total 8 ide dalam kurun waktu 8 menit jadi 1 ide 1 menit DI KERTAS

Jika bekerja dengan tim, disarankan semua anggota melakukan metode Crazy Eight. Latihan ini akan menyentak

otak dan membuat berpikir tentang tantangan. Biasanya sketsa akan menjadi *wireframe* desain antarmuka.

Setelah semua anggota tim menyajikan ide-idenya, Setiap orang harus menjelaskan masing-masing delapan ide mereka secara rinci dan mengapa mereka memilih untuk mengambil jalur tersebut. Ingatkan setiap anggota tim untuk mengguna-kan pembelajaran untuk pembenaran ide-ide mereka. Setelah semua orang mengemukakan idenya, saatnya memilih ide-ide tersebut. Setiap orang mendapat dua titik tempelan dan bisa memberikan suara pada ide mana pun. Mereka bisa memberikan kedua suaranya untuk sebuah ide jika mereka benar-benar menyukainya.

2. Menyaring Desain

Setelah pemungutan suara mengambil ide dengan suara terbanyak dan membuat sketsa ide akhir. Juga dapat memilih ide lain dari anggota tim. Berikan waktu sepuluh menit untuk menyelesaikan tugas ini. Setelah selesai, presentasikan kembali ide ini ke tim dan lakukan pemungutan suara seperti sebelumnya.

3. Membuat Ide *Storyboard*

Storyboard melibatkan perpaduan sketsa dan ide ke dalam alur komprehensif. Dengan desain di tangan, saatnya untuk membuat *storyboard* interaksi dengan pengguna. Di titik ini sebaiknya berpikir tentang langkah yang berbeda yang diambil pengguna. Cukup biasa untuk menggabungkan salah satu dari desain anggota tim ke dalam alur. Diperlukan proses langkah demi langkah yang jelas dengan beberapa titik di mana pengguna mungkin berbeda. Lihat kembali pemetaan proyek untuk memvalidasi desain terhadap tujuan.

4. Membuat Prototipe

Membuat prototipe bukan tentang menciptakan potongan kode yang sempurna, namun untuk membuat sesuatu yang bisa dipercaya bila digunakan oleh seseorang. Alat yang digunakan untuk membuat prototipe berbeda dari orang ke

orang. Beberapa alat seperti *Keynote* atau *Powerpoint* karena memaksa untuk memikirkan alur dan tidak merancang detail. Mungkin ingin meluangkan waktu untuk mempelajari alat seperti Balsamiq, Marvel atau Framer yang bisa memberikan kontrol perilaku yang lebih banyak. Apapun alat (bantu) yang digunakan pastikan itu adalah alat yang membuat fokus pada alur dan terlihat nyata.

Pengujian prototipe pada pengguna yang nyata sehingga sebisa mungkin dapat dipercaya tapi pada saat yang bersamaan tidak memerlukan berminggu-minggu jam kerja untuk membuatnya.

Prototipe harus cukup nyata untuk bisa dipercaya. Membuat prototipe adalah keseimbangan antara waktu dan realitas, jadi berhati-hatilah agar tidak melenceng ke salah satu sisi secara ekstrem. Bila tidak, akan membuang waktu percuma.

5. Pengujian-Kegunaan Desain

Akan bagus sekali jika memiliki lab pengujian. Bila tidak, membuat lab tidak sulit asalkan memperhatikan pembuatan lingkungan yang nyaman bagi pengguna serta tidak mengganggu mereka. Pengujian biasanya melibatkan pengguna dan dua orang dari tim, satu mencatat dan lainnya mengajukan pertanyaan. Persiapan yang baik adalah dengan menggunakan aplikasi seperti *Hangouts* dan merekam tindakannya, ini juga berguna jika menginginkan seluruh tim untuk mengamati dari ruangan yang berbeda. Hal ini cukup menakutkan bagi kami sebagai pembuat aplikasi untuk melakukannya saat melihat desain kami keluar di alam liar. Ini bisa menjadi pengalaman yang menyegarkan dan menenangkan.

Storyboard termasuk menempatkan semua sketsa dan ide bersama-sama ke dalam alur yang komprehensif.

6. Yang Perlu ditanyakan

Saat menguji desain, minta pengguna untuk melakukan tugas di aplikasi dan minta mereka agar berbicara dengan

suara keras serta mengungkapkan apa yang mereka lakukan dan mengapa. Ini mungkin terdengar aneh dilakukan, namun hal ini membantu untuk mengetahui apa yang mereka pikirkan. Cobalah untuk tidak mengganggu atau memberi tahu mereka apa yang harus dilakukan saat mereka terhenti. Cukup tanyakan kepada pengguna mengapa mereka mengambil alur tertentu setelah mereka menyelesaikan (atau TIDAK menyelesaikan).

Apa yang perlu Anda ketahui:

a. Apa yang mereka suka dari prototipe?
b. Apa yang mereka tidak suka dari prototipe?
c. Apa saja titik deritanya?
d. Mengapa alur bekerja
e. Mengapa alur tidak bekerja
f. Apa yang ingin mereka tingkatkan?
g. Apakah keseluruhan desain/alur memenuhi kebutuhan mereka?

7. Mengunjungi Kembali Desain dan Rentetan Pengujian lagi

Prototipe yang akan digunakan berdasarkan masukkan dari pengguna untuk melakukan revisi desain, dan meng-analisis apa yang berhasil dan apa yang tidak. Jangan takut untuk membuat *storyboard wireframe* yang benar-benar baru dan membuat prototipe baru. Memulai lagi dari awal bisa membuat alur yang lebih baik dibandingkan mencoba untuk memindahkan sesuatu pada prototipe sebelumnya.

Setelah puas dengan desain, bisa dilakukan pengujian lagi dan menyempurnakannya. Dalam kasus di mana prototipe sama sekali tidak mencapai target, mungkin berpikir proyek itu gagal. Kenyataannya, tidak, mungkin akan menghabiskan waktu *development* lebih singkat dibandingkan jika telah membuat desain dan mengetahui lebih banyak tentang apa yang benar-benar disukai pengguna. Dengan *design sprints*, kami memiliki filosofi yaitu menang atau belajar, jadi jangan

terlalu menyalahkan jika ide tersebut tidak bekerja seperti yang direncanakan.

Buatlah! setelah melakukan pengujian ide, sehingga pengguna menyetujui. Pemangku kepentingan berinvestasi karena mereka telah terlibat sejak awal. Sekarang saat yang tepat untuk membuatnya. Sekarang, harus memiliki gagasan yang jelas tentang apa yang perlu dilakukan dan apa prioritas dari pengalaman ini. Pada setiap aktivitas proyek, perlu dilakukan pengujian yang berguna untuk membantu memvalidasi pekerjaan dan menjaga tetap pada jalur.

Bukan menjadi penekanan betapa pentingnya mencari informasi sebanyak mungkin sebelum berkomitmen untuk melakukan banyak pekerjaan, waktu dan energi pada sesuatu yang mungkin saja tidak menjadi solusi yang tepat.

DAFTAR PUSTAKA

Anne Kaldewaij, 1990, Programming: The Derivation of Algorithms. Prentice Hall.

BEA Systems, IBM, Microsoft, 2002, Business Process Execution Language for Web Services, Version 1.0. IBM developerWorks, Available from http://www.106.ibm.com/developerworks/webservices/library/ws-bpel1/

Doug Rosenberg Kendall Scott, 2001, Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example, Addison Wesley First Edition.

Doug Rosenberg and Matt Stephens, 2007, Use Case Driven Object Modeling with UML: Theory and Practice, Apress.

Emerging Technologies Toolkit. IBM alphaWorks (2003). Available from http://www.alphaworks.ibm.com/tech/ettk/

Gardner, T et al., 2003, Draft UML 1.4 Pro*file* for Automated Business Processes with a mapping to the BPEL 1.0. IBM alphaWorks, Available from http://dwdemos.alphaworks.ibm.com/wstk/common/wstkdoc/services/demos/uml2bpel/README.htm.

Hidayattullah Himawan & Mangaras Yanu F., 2020, *Interface User* Experience, Lembaga Penelitian dan Pengabdian kepada Masyarakat UPN Veteran Yogyakarta

Howard Podeswa, 2003, UML for the IT Business Analyst, Second Edition: A Practical Guide to *Requirement* s Gathering Using the Unified Modeling Language, Course Technology, a part of Cengage Learning.

Ian Sommerville, 2016, *Software Engineering* Tenth Edition, Pearson Education Limited

Jean-François Monin and Michael G. Hinchey, 2003, Understanding Formal Methods. London: Springer Verlaag.

Karl Wiegers, Joy Beatty, 2013, *Software Requirement* s, 3rd Edition, Microsoft Press.

Kenneth H. Rosen, 2012, Discrete Mathematics and Its Applications, 7th Edition. McGraw-Hill.

Michael Fischer, 2011, Practical Formal Methods Using Temporal Logics. John Wiley and Sons, Ltd.

Michael Huth and Mark Ryan, 2004, Logic in Computer Science: Modeling and Reasoning about System, 2nd Edtion. Cambridge University Press.

Mordechai Ben-Ari, 2001, Mathematical Logic for Computer Science, 2nd Edition. Springer Verlag.

Selic, B, 2003, The Pragmatics of Model-Driven Development. IEEE *Software* special issue on Model-Driven Architecture.

T. H. Cormen, et al, 2009, Introduction to Algorithms, 3rd Edition. MIT Press.

PROFIL PENULIS

Lambang Probo Sumirat, aktif menjadi dosen teknik informatika fakultas teknik Universitas Dr. Soetomo sejak tahun 2009. Mengampu matakuliah Rekayasa Perangakat Lunak sejak tahun 2011 hingga sekarang. Dan pernah menjadi praktisi pada bagian software solution pada PT. Greatsoft Solusi Indonesia dari tahun 2017 hingga 2022.

Buku ini membahas dasar-dasar pengetahuan tentang rekayasa perangkat lunak atau disebut juga dengan rekayasa software atau Software Engineering.

Dalam penyusunan buku ini, penulis melakukan beberapa studi terkait pelaksanaan pembangunan atau pembuatan perangkat lunak di beberapa instansi, baik instansi pemerintahan, akademik maupun instansi swasta. Selain itu penulis juga beberapa kali melakukan perbandingan antar perangkat lunak yang terdapat pada masing-masing instansi. Dari hasil tersebut dapat disimpulkan bahwa setiap perangkat lunak yang akan dibangun atau dibuat harus memenuhi kebutuhan dan dapat menyelesaikan permasalahan dari penggunanya sehingga tercapai tujuan dari perangkat lunak.

Madza Media
redaksi@madzamedia.co.id
www.madzamedia.co.id
@madzamedia

ISBN 978-623-130-241-0